Final Report

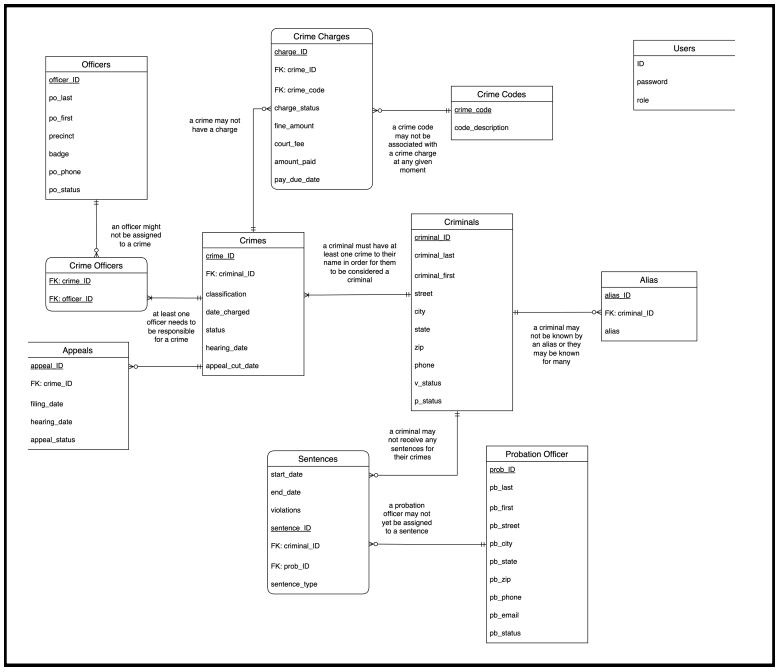
Introduction to Databases - Final Project

Google Drive Link: https://drive.google.com/drive/folders/1J6Qj81uohedWaKX7OXbDyvK7oRc4iFaD?usp=sharing

Authors: Jenesis Blancaflor, Nabira Ahmad, Levith Andrade Cuellar

Database Design

E-R Diagram



The only modification we made from our previous E-R diagram was the addition of a table named “user”. This keeps some information about our users in addition to the built-in user table that the database itself provides.

Schema Statements

Alias(alias_ID, criminal_ID(FK), alias)

Appeals(appeal_ID, crime_ID(FK), filing_date, hearing_date, appeal_status)

Crime Charges(charge_id, crime_ID(FK), crime_code(FK), charge_status, fine_amount, court_fee, amount_paid, pay_due_date)

Crime Codes(crime_code, code_description)

Crime Officers(crime_id(FK), officer_ID(FK))

Criminals(criminal_ID, criminal_last, criminal_first, street, city, state, zip, phone, v_status, p_status)

Crimes(crime_ID, criminal_ID(FK), classification, date_charged, status, hearing_date, appeal_cut_date)

Officers(officer_ID, po_last, po_first, precinct, badge, po_phone, po_status)

Probation Officers(prob_ID, pb_last, pb_first, pb_street, pb_city, pb_state, pb_zip, pb_phone, pb_email, pb_status)

Sentences(sentence_ID, criminal_ID(FK), prob_ID(FK), start_date, end_date, violations, sentence_type)

Users(ID, password, role)

Database Programming

Where the Database is Hosted

The database is hosted on my MySQL and XAMPP.

Where the Application is Hosted

The application is hosted through XAAMP and the Apache Web Server.

Instructions to Deploy

1. Deploy XAMPP and start Apache Web Server and MySQL
2. Add application file into htdocs of XAMPP folder
3. Connect to the database through local IP address
4. Deploy application by typing into browser: localhost/crimes_database
5. Login or register to access data

Implementation of Advanced SQL Commands

We used our advanced SQL command procedures in our app to retrieve the information for each profile. For the criminal, officer and probation officer profiles, we called their respective procedures in PHP and queried their outputs, then used that data as the information for our profile pages.

Calling the procedure using the user's search input.

```php
$sql = "CALL criminal_profile($criminal_id)";
$result = $conn->query($sql);
```

Retrieving the data from the procedure into variables.

```php
if ($result) {
    while ($row = $result->fetch_assoc()) {
        // General Information
        $criminalID = $row["criminal_ID"];
        $criminalFirstName = $row["criminal_first"];
        $criminalLastName = $row["criminal_last"];
```

Database Security - Database Level

Settings for End Users

We defined three different kinds of end users:

- Officers
- Criminals
- Probation Officers

Officers and Probation Officers have the highest level of access to the platform.

They are able to SELECT, INSERT, UPDATE and DELETE from the database. In out platform we allow them to INSERT a new criminal, UPDATE information about a criminal and DELETE information about an appeal (which does not involve any foreign keys.

They are also able to execute PROCEDURES necessary to retrieve profile information for crimes, criminals, probation officers and other officers.

Criminals have the lowest level of access to the platform.

They are able to SELECT information from tables that may contain information about them and that are necessary to retrieve their profiles — criminals, crimes, sentences, crime charges and appeals.

They are also able to execute PROCEDURES necessary to retrieve profile information about themselves — criminals — and the crimes they’ve commited.

```
// Officers and Probation Officers

// If the registered user is an officer or probation officer,
if ($role == "officer" or $role == "probation_officer"){

    // Set privileges.
    $createUserStatement = "CREATE USER '$username'@'localhost' IDENTIFIED BY '$hashed_password';

                            GRANT SELECT, INSERT, UPDATE, DELETE ON crime.* TO '$username'@'localhost';

                            GRANT EXECUTE ON PROCEDURE crime.criminal_profile TO '$username'@'localhost';
                            GRANT EXECUTE ON PROCEDURE crime.crime_profile TO '$username'@'localhost';
                            GRANT EXECUTE ON PROCEDURE crime.probation_officer_info TO '$username'@'localhost';
                            GRANT EXECUTE ON PROCEDURE crime.officer_profile TO '$username'@'localhost';
                            GRANT EXECUTE ON PROCEDURE crime.criminal_crimes TO '$username'@'localhost';
                            FLUSH PRIVILEGES;
                            ";
}

// Criminals

// If the registered user is a criminal,
else if ($role == "criminal"){

    // Set privileges.
    $createUserStatement = "CREATE USER '$username'@'localhost' IDENTIFIED BY '$hashed_password';

                            GRANT SELECT ON crime.criminals TO '$username'@'localhost'; FLUSH PRIVILEGES;
                            GRANT SELECT ON crime.crimes TO '$username'@'localhost'; FLUSH PRIVILEGES;
                            GRANT SELECT ON crime.sentences TO '$username'@'localhost'; FLUSH PRIVILEGES;
                            GRANT SELECT ON crime.crime_charges TO '$username'@'localhost'; FLUSH PRIVILEGES;
                            GRANT SELECT ON crime.appeals TO '$username'@'localhost'; FLUSH PRIVILEGES;

                            GRANT EXECUTE ON PROCEDURE crime.criminal_profile TO '$username'@'localhost'; FLUSH PRIVILEGES;
                            GRANT EXECUTE ON PROCEDURE crime.crime_profile TO '$username'@'localhost'; FLUSH PRIVILEGES;
                            GRANT EXECUTE ON PROCEDURE crime.criminal_crimes TO '$username'@'localhost'; FLUSH PRIVILEGES;
                            ";
}
```

This is a screenshot of how we restricted privileges to certain users as we described in the previous page.

Access Control

The above screenshot also exemplifies how we managed access control. This screenshot belongs to our registration page. When a user registers on our platform an equivalent user is generated on our server. Depending on what the user registers as (an officer, probation officer or criminal), the platform will grant the user certain privileges.

When a user returns to login we handle their credentials securely through a PHP session. We have every subsequent page inherit their login credentials to re-establish connection with the database under the user's account on the server. This helps in enforcing the user's privileges and limitations even in the event that they're able access a part of our application that was not intended for their role.

```
// Start the session to retrieve session data
session_start();

// Set database credentials
$host = "localhost";
$username = $_SESSION['username'];
$password = $_SESSION['password'];
$database = "crime";
$conn = new mysqli($host, $username, $password, $database);
```

Database Security - Application Level

Description

Database security at the application level is incorporated in our project through the use of **hashing** and **prepared statements**.

Specifically, in the login page (index.php), the PHP function *password_verify(incoming_password, existing_hashed_password)* is used to encrypt user passwords and validate them against the securely hashed values in the database, which are hashed in the register page (register.php) through the PHP function *password_hash(incoming_password, algorithm_to_hash)*, as shown below.

```php
// Obtain user input from the HTML.
$username = $conn->real_escape_string($_POST['username']);
$password = $conn->real_escape_string($_POST['password']);
$role = $conn->real_escape_string($_POST['role']);

// Create a hashed password.
$hashed_password = password_hash($password, PASSWORD_DEFAULT);

$stmt = $conn->prepare("INSERT INTO users (username, password, role) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $username, $hashed_password, $role);
```

```php
// If the password is correct,
if (password_verify($password, $user['password'])) {

    // Store the user's information on the session.
    $_SESSION['user_id'] = $user['id'];
    $_SESSION['username'] = $username;
    $_SESSION['role'] = $user['role'];
    $_SESSION['password'] = $user['password'];

    // Redirect the user to the search landing page.
    header("Location: crime_search.php");
    exit();
```

In the screenshot below, we used the PHP function *htmlspecialchars(incoming_password)* for the update page (update.php) to **treat script tags as plain text** instead of executable code to protect against harmful scripts.

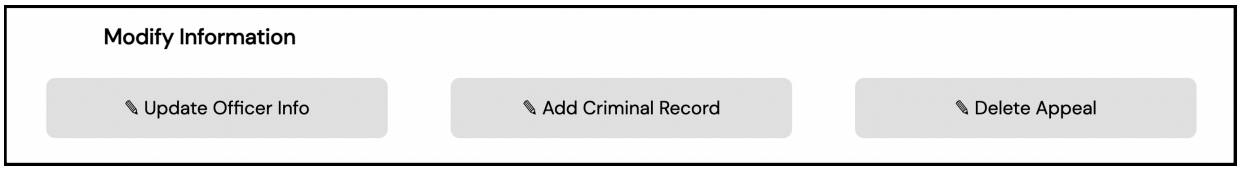
```
<!-- select table -->
<form action="<?php echo htmlspecialchars($_SERVER['PHP_SELF']); ?>" method="post" class="request">
    <label>Criminal Table:</label>
```

We also implemented other security measures in our app, including the use of **prepared statements** to protect against potential SQL injection attacks. In the update file (update.php), delete file (delete.php), and the add criminal file (add_criminal.php), prepared statements are used to separate SQL from user input to prevent malicious SQL attempts, increasing the app's security. An example of the prepared statement for the delete file is shown below.

```
<?php
if ($deleteButtonPressed) {
    // use prepared statement to delete the entry
    $sqlDelete = "DELETE FROM appeals WHERE appeal_id = ?";
    $stmtDelete = $conn->prepare($sqlDelete);
    $stmtDelete->bind_param("s", $selectedAppealId);

    if ($stmtDelete->execute()) {
        echo "<p style='text-align: center; color: green;'>Record deleted successfully</p>";
    } else {
        echo "<p style='text-align: center; color: red;'>Error deleting record: " . $stmtDelete->error . "</p>";
    }

    $stmtDelete->close();
}
```

```php
// Check if the form has been submitted
if ($_SERVER["REQUEST_METHOD"] == "GET" && isset($_GET["searchInput"]) && isset($_GET["searchCategory"])) {
    // Retrieve the search input and category from the form
    $searchInput = $_GET["searchInput"];
    $searchCategory = $_GET["searchCategory"];

    // Redirection based on category
    $profileLinks = [
        'Criminal' => "criminal.php?id=$searchInput",
        'Crime' => "crime.php?id=$searchInput",
        'Probation Officer' => "prob_officer.php?id=$searchInput",
        'Officer' => "officer.php?id=$searchInput"
    ];

    if (isset($profileLinks[$searchCategory])) {
        header("Location: " . $profileLinks[$searchCategory]);
        exit;
    } else {
        $errorMessage = "No results found for the selected category.";
    }
}
```

Once a user successfully logs into our application, they are automatically lead to the search page. Depending on if the user is a officer of some sort or a criminal, their **access to searching information is limited**. If the user logged in is an officer or probation officer, they are able to search for all the profiles in the database including: criminals, crimes, officers, and probation officers by their corresponding ID's. If the user logged in is a criminal, they are only able to access their criminal profile and the crime profiles of those they involved in. Criminals are not able to access the profiles of the officers or probation officers.



The **flow of our application also adds to its security**. We placed buttons for updating, adding and deleting information under the officer profile page. This means that criminals do not have access to these buttons as the platform restricts them from searching for officers.