TRIGGERS TO FACILITATE TRANSACTIONS

To automatically synchronize data between the admins and users tables in MySQL, you can create triggers that will update the corresponding rows in the users table whenever an update is made to the admins table. Here's how you can do it:

sql

Copy code

-- Create trigger to synchronize updates from admins table to users table

DELIMITER //

CREATE TRIGGER sync\_admins\_to\_users AFTER UPDATE ON admins

FOR EACH ROW

BEGIN

UPDATE users

SET username = NEW.username,

email = NEW.admin\_email,

phone = NEW.admin\_phone,

school = NEW.admin\_school,

user\_password = NEW.admin\_password,

role = NEW.admin\_role

WHERE user\_code = NEW.admin\_code;

END;

//

DELIMITER ;

This trigger will fire after an update operation is performed on the admins table. It will update the corresponding row in the users table based on the admin\_code.

Please note the following:

The AFTER UPDATE keyword specifies that the trigger should execute after an update operation is performed on the admins table.

FOR EACH ROW indicates that the trigger will be executed for each row that is updated in the admins table.

NEW is used to refer to the new values of the columns being updated in the admins table.

The WHERE clause ensures that the update operation is applied only to the row in the users table where the user\_code matches the admin\_code of the updated row in the admins table.

Make sure to set the DELIMITER to // before creating the trigger and resetting it back to ; afterward. This is necessary because the trigger definition contains semicolons, which would otherwise terminate the CREATE TRIGGER statement prematurely.

With this trigger in place, whenever you update a row in the admins table, the corresponding row in the users table will be automatically updated to reflect the changes.

SECURITY OF INPUT FIELDS

The registration form you provided incorporates several security measures to enhance the overall security of the user registration process. Here's an explanation of these security measures:

Password Policy Validation:

The validatePassword function enforces a password policy, ensuring that passwords meet specific criteria:

Minimum length of 8 characters.

At least one uppercase letter.

At least one lowercase letter.

At least one digit (number).

At least one special character.

Input Validation and Sanitization:

The htmlspecialchars function is used to sanitize user input, preventing potential Cross-Site Scripting (XSS) attacks. It converts special characters to their HTML entities, ensuring that user input is safely rendered in the HTML output.

Password Hashing:

The user's password is hashed using the password\_hash function before being stored in the database. Password hashing is a crucial security measure to protect user passwords. It ensures that even if the database is compromised, the actual passwords remain secure.

Prepared Statements:

Prepared statements are used in the database queries. This protects against SQL injection attacks by separating SQL code from user input. The use of placeholders and binding parameters helps prevent malicious SQL code execution.

Error Handling:

The code includes try-catch blocks to handle exceptions, particularly PDOException for database-related errors. This prevents sensitive information from being exposed in error messages. Different error messages are displayed based on the type of error, improving user experience and security.

Unique Constraint Violation Handling:

The code checks for unique constraint violations (e.g., duplicate email or code) and provides a user-friendly error message. This prevents attackers from exploiting such constraints to enumerate valid user accounts.

Limited Error Disclosure:

The code avoids providing detailed error messages to users. Instead, it offers generic error messages to prevent potential attackers from gaining insights into the system's internal workings.

Conditional Execution:

The registerUser function only proceeds with user registration if the password passes validation and there are no other error messages. This ensures that the registration process is halted upon detecting security issues.

Server-Side Validation:

While client-side validation is common for improving user experience, server-side validation (in PHP) is crucial for security. Client-side validation can be bypassed, and server-side validation acts as a safety net to catch any potential manipulation or bypass attempts.

By incorporating these security measures, the registration form aims to mitigate common security risks associated with user input, password storage, and database interactions.