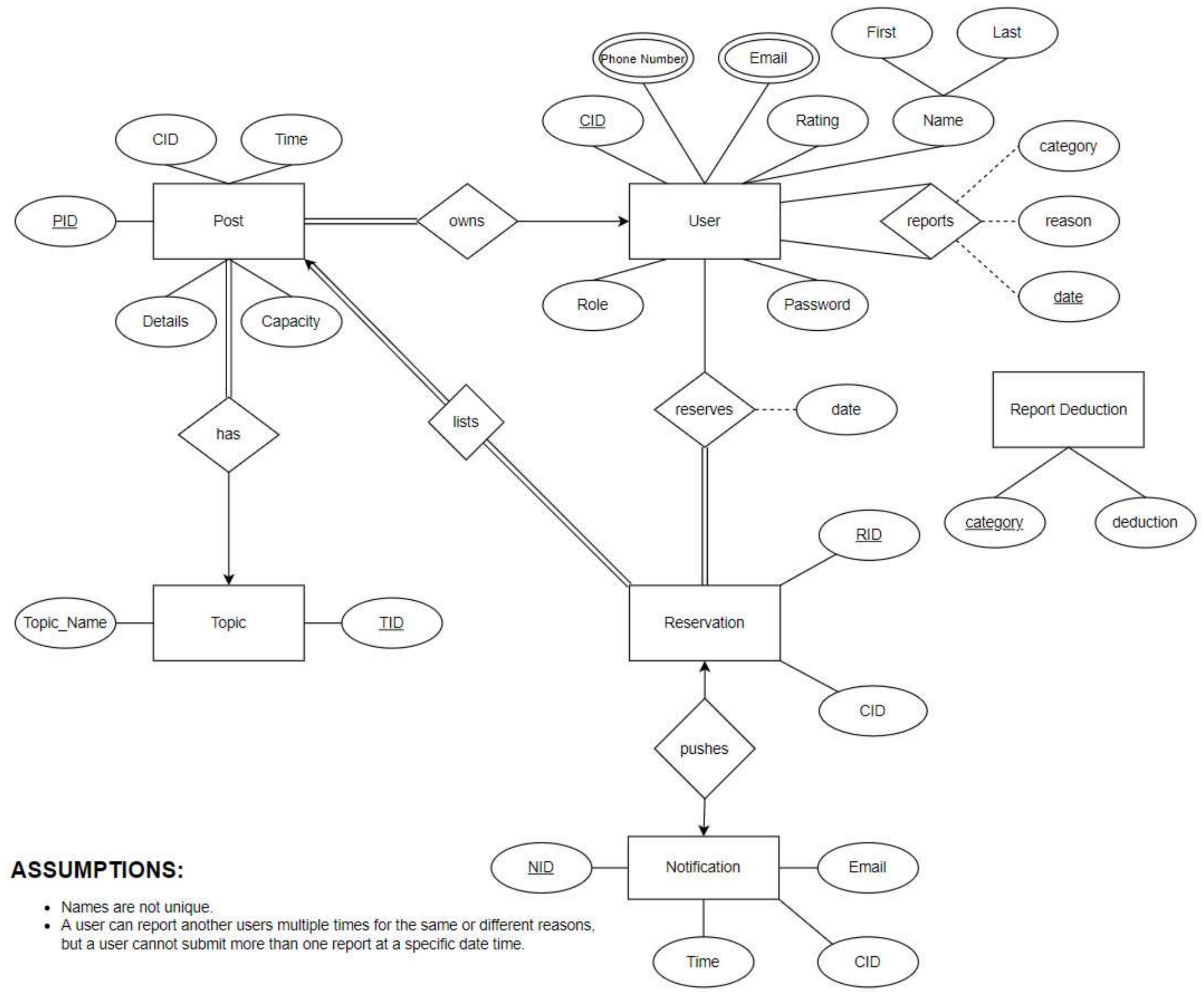
CS 4750 - PROJECT FINAL DELIVERABLE

Jack Liu (jml7ctd), Anh-Thu Nguyen (an6gzp), Ketian Tu (kt9sh)

Database Design

ER Diagram



ASSUMPTIONS:

- Names are not unique.
- A user can report another users multiple times for the same or different reasons, but a user cannot submit more than one report at a specific date time.

Schema Statements

Post(<u>PID</u>, CID, Time, Details, Capacity, TID)
User(<u>CID</u>, First_Name, Last_Name, Rating, Password, Role)
User_PhoneNumber(<u>CID</u>, <u>Number</u>)
User_Email(<u>CID</u>, <u>EmailAddress</u>)
Topic(<u>TID</u>, TopicName)
Reservation(<u>RID</u>, CID, PID)

Notification(NID, Time, Email, CID)
ReportDeduction(catergory, deduction)

reports(CID1, CID2, category, reason, date)
pushes(RID, NID)

Database Programming

Database Deployment Environment

We hosted our MySQL database in the local environment. This was due to time constraints as it helped with quick turnaround in development and testing. The .sql file containing our database has been included with the source code.

App Deployment Environment

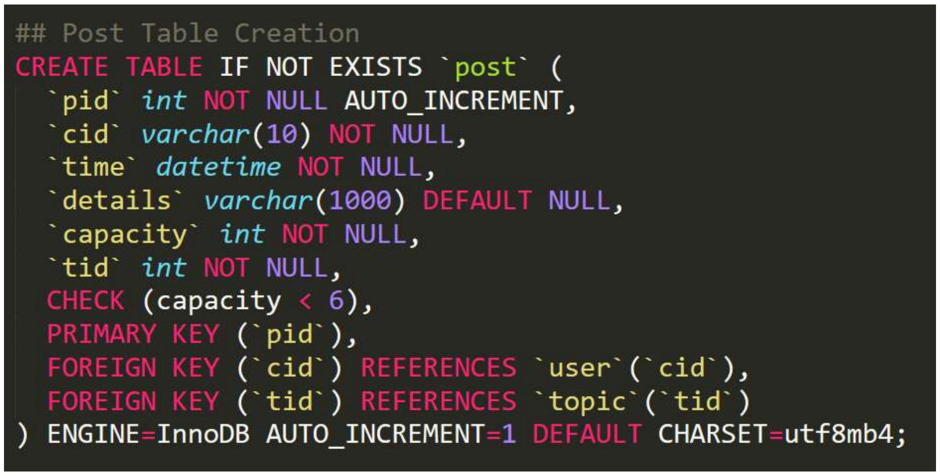
We host our application in the local environment using PHP. To successfully deploy and run the project, a user should install XAMPP in the machine and place the source code folder in the 'htdocs' folder. The detailed instructions to install XAMPPcan be found here: https://storage.googleapis.com/cs4750f21/supplement/XAMPP-setup.html

Deploy and run project steps

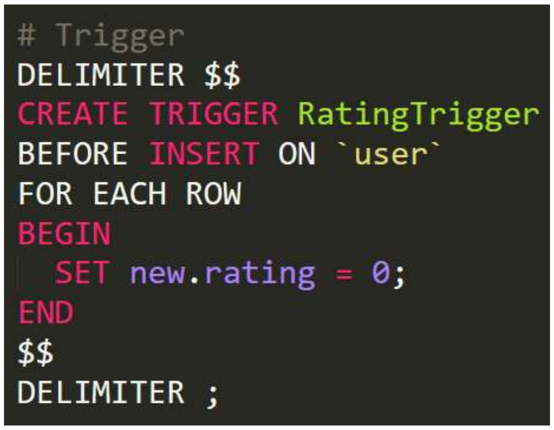
1. Obtain the source code from the submission
2. Install XAMPP (a local web server) by following the tutorial: https://storage.googleapis.com/cs4750f21/supplement/XAMPP-setup.html
3. After installation, open the XAMPP folder. Place the source code folder under folder 'htdocs'
4. Start the user interface for XAMPP, start the apache server
5. The project can be found in the local environment
6. For database, open the local MySQL environment(phpMyAdmin or MySQL workbench)
7. Create a database called "hooshangry" and import the SQL file in the source code
8. Create the appropriate user accounts for the DBMS
    a. Root - Username: "root", Password: "password"
    b. Login - Username: "login", Password: "login"
    c. Student - Username: "student", Password: "student"
    d. Grant appropriate privileges (shown in section 3) or just give all global privileges
9. Navigate to the login.php page as the main entry point to the site

Incorporation of advanced SQL commands in the app

We use two advanced SQL commands.
CHECK (Included in table schema)


```
## Post Table Creation
CREATE TABLE IF NOT EXISTS `post` (
  `pid` int NOT NULL AUTO_INCREMENT,
  `cid` varchar(10) NOT NULL,
  `time` datetime NOT NULL,
  `details` varchar(1000) DEFAULT NULL,
  `capacity` int NOT NULL,
  `tid` int NOT NULL,
  CHECK (capacity < 6),
  PRIMARY KEY (`pid`),
  FOREIGN KEY (`cid`) REFERENCES `user`(`cid`),
  FOREIGN KEY (`tid`) REFERENCES `topic`(`tid`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4;
```

Trigger


```
# Trigger
DELIMITER $$
CREATE TRIGGER RatingTrigger
BEFORE INSERT ON `user`
FOR EACH ROW
BEGIN
  SET new.rating = 0;
END
$$
DELIMITER ;
```

The first command makes sure that the seating capacity for one meeting is less than 6. This command is useful as it ensures that every meeting is following the University's social distancing rule. This check will be activated every time a new post is inserted into the database. If a potential post violates this rule, the table will not add the record.

The second command ensures that every member of the user table starts with a zero rating. This is useful as it ensures that nobody gets an unfair advantage when creating the account. This trigger will be activated before inserting into the user table.

Database Security at Database Level

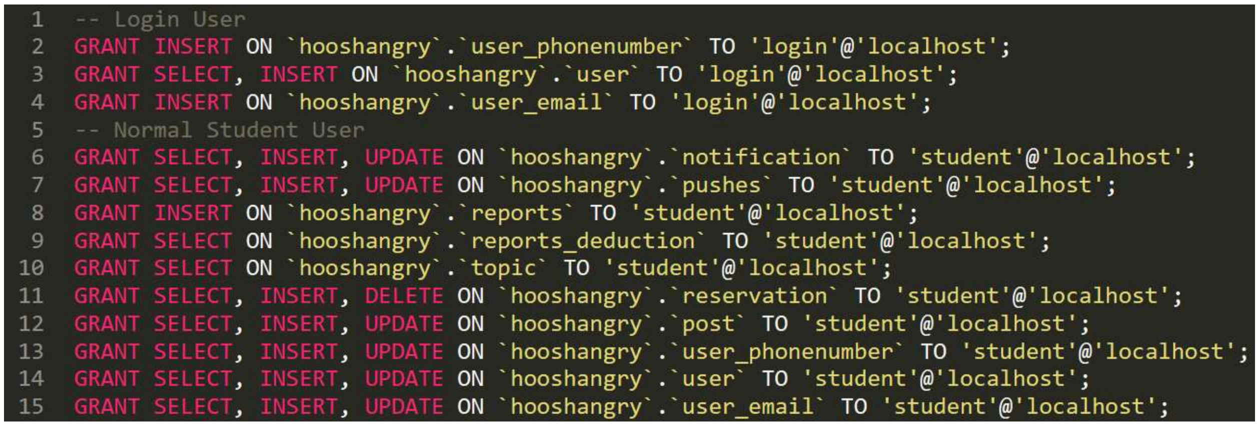
Specify whether the security is set for developers or end users

Our database security measures were mostly set for us developers to limit the user's access to certain tables. In this way, we ensure that some tables such as reports_deductions can never be modified by standard users (non-admin) through the application. However, there is some overlap with security for end users. For example, we limit who is able to read reports to maintain confidentiality for end users.

Discuss how you set up security at the database level (access control)

Our application features three different kinds of users that exchange control during the different sections of our website. Initially, there is a log-in user that only has access to the user table to look up usernames and passwords. It also has the ability to insert into user and user_email/user_phonenumber tables because it also handles account creation. However, it does not have any other permissions and thus is unable to read any other information. THis ensures that much of the database is gated behind an account wall. Once the user inputs a valid username/password, control is then switched to either a student user or admin user based on the account details that was used to log in. The student user can only do the appropriate actions for the core site functionality. This includes selecting, inserting, and updating posts etc. but does not include adding new topic categories or seeing other reports. On the other hand, the admin user has full privileges on all tables and can only be accessed by us as developers. The SQL commands used to set up these permissions are listed below.

SQL commands used to limit/set privileges:



```sql
-- Login User
GRANT INSERT ON `hooshangry`.`user_phonenumber` TO 'login'@'localhost';
GRANT SELECT, INSERT ON `hooshangry`.`user` TO 'login'@'localhost';
GRANT INSERT ON `hooshangry`.`user_email` TO 'login'@'localhost';
-- Normal Student User
GRANT SELECT, INSERT, UPDATE ON `hooshangry`.`notification` TO 'student'@'localhost';
GRANT SELECT, INSERT, UPDATE ON `hooshangry`.`pushes` TO 'student'@'localhost';
GRANT INSERT ON `hooshangry`.`reports` TO 'student'@'localhost';
GRANT SELECT ON `hooshangry`.`reports_deduction` TO 'student'@'localhost';
GRANT SELECT ON `hooshangry`.`topic` TO 'student'@'localhost';
GRANT SELECT, INSERT, DELETE ON `hooshangry`.`reservation` TO 'student'@'localhost';
GRANT SELECT, INSERT, UPDATE ON `hooshangry`.`post` TO 'student'@'localhost';
GRANT SELECT, INSERT, UPDATE ON `hooshangry`.`user_phonenumber` TO 'student'@'localhost';
GRANT SELECT, INSERT, UPDATE ON `hooshangry`.`user` TO 'student'@'localhost';
GRANT SELECT, INSERT, UPDATE ON `hooshangry`.`user_email` TO 'student'@'localhost';
```

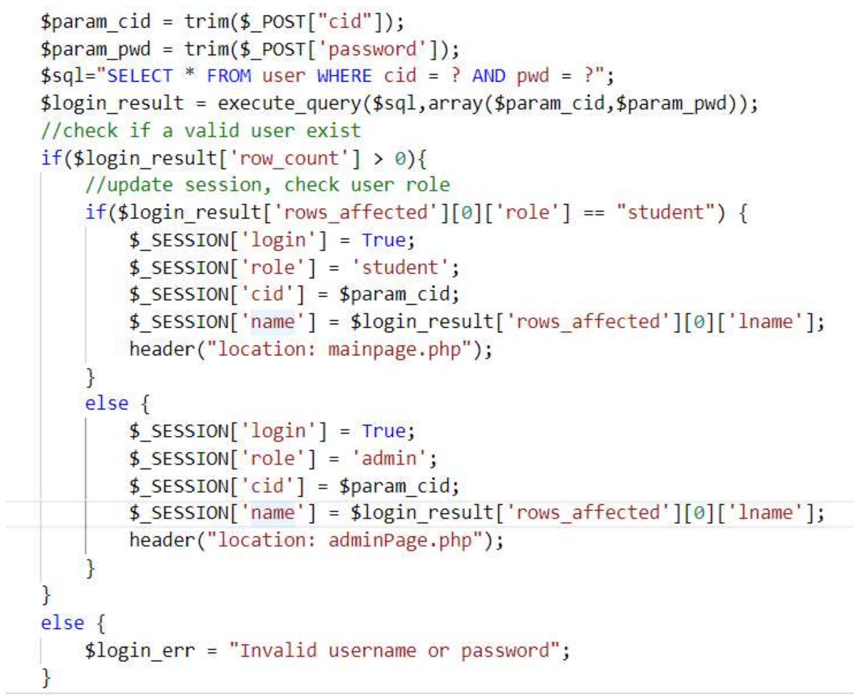
Database Security at Application Level

Discuss

We use several ways to incorporate security in the application level. For the first part, we implement an user authentication system that verifies the credentials of users. The process consists of matching the record of user name and password in the user table. If we are not able to find a pair that matches the cid/pwd pair from the database, the system will return a message in the interface. Secondly, we also implement a separate page for admin users. Once the user enters the right password, the application will determine if the user is a student or an admin user. An admin user will be routed to a different interface displaying all the posts in the database. Thirdly, we implement redirection to check the login status of a user. When a user is successfully logged in, the machine will store his/her information (name, cid, etc) using session. Everytime a user tries to use functionalities within the system, the system will check if the session storage is still valid. If the session expires, the user will be redirected to the login page. Finally, all SQL statements executed by the application first pass through a helper function which features a prepared statement. This helps to guard against SQL injection attacks and sanitizing of inputs.

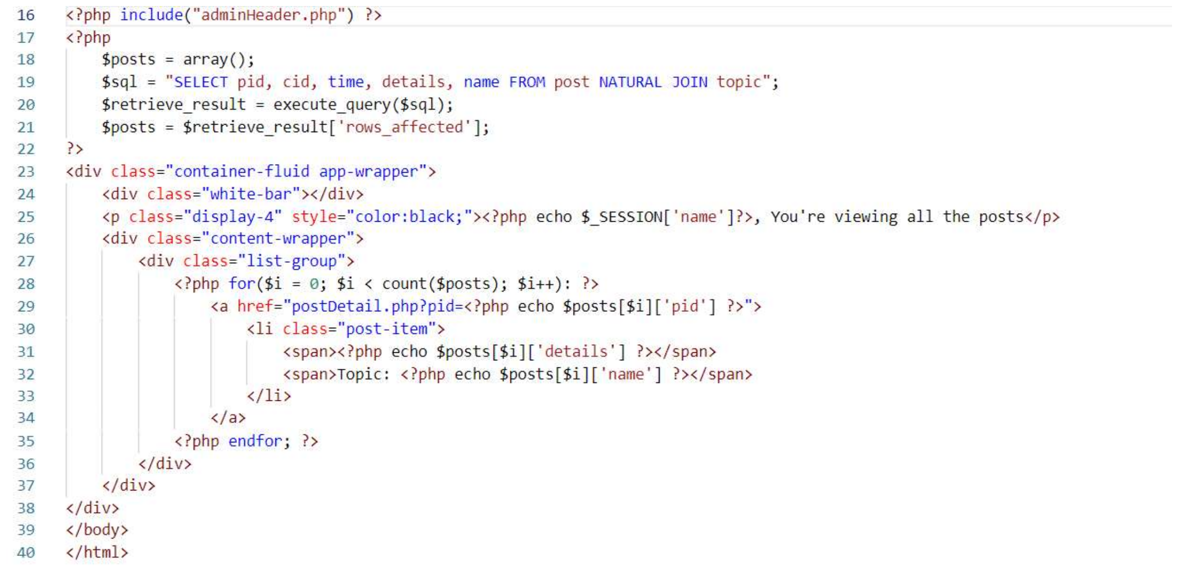
Code

- **Code snippet for #1 (Login code)**



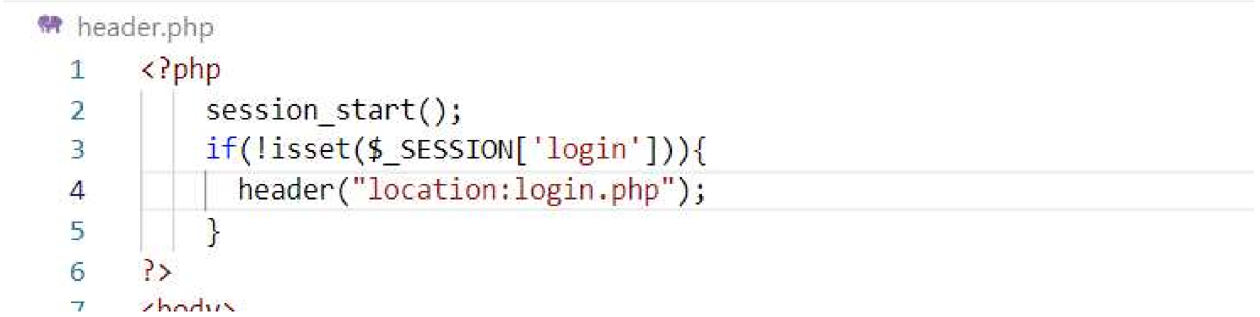
```php
$param_cid = trim($_POST["cid"]);
$param_pwd = trim($_POST['password']);
$sql="SELECT * FROM user WHERE cid = ? AND pwd = ?";
$login_result = execute_query($sql,array($param_cid,$param_pwd));
//check if a valid user exist
if($login_result['row_count'] > 0){
    //update session, check user role
    if($login_result['rows_affected'][0]['role'] == "student") {
        $_SESSION['login'] = True;
        $_SESSION['role'] = 'student';
        $_SESSION['cid'] = $param_cid;
        $_SESSION['name'] = $login_result['rows_affected'][0]['lname'];
        header("location: mainpage.php");
    }
    else {
        $_SESSION['login'] = True;
        $_SESSION['role'] = 'admin';
        $_SESSION['cid'] = $param_cid;
        $_SESSION['name'] = $login_result['rows_affected'][0]['lname'];
        header("location: adminPage.php");
    }
}
else {
    $login_err = "Invalid username or password";
}
```

- **Code snippet for #2 (Admin page)**



```php
<?php include("adminHeader.php") ?>
<?php
    $posts = array();
    $sql = "SELECT pid, cid, time, details, name FROM post NATURAL JOIN topic";
    $retrieve_result = execute_query($sql);
    $posts = $retrieve_result['rows_affected'];
?>
<div class="container-fluid app-wrapper">
    <div class="white-bar"></div>
    <p class="display-4" style="color:black;"><?php echo $_SESSION['name']?>, You're viewing all the posts</p>
    <div class="content-wrapper">
        <div class="list-group">
            <?php for($i = 0; $i < count($posts); $i++): ?>
                <a href="postDetail.php?pid=<?php echo $posts[$i]['pid'] ?>">
                    <li class="post-item">
                        <span><?php echo $posts[$i]['details'] ?></span>
                        <span>Topic: <?php echo $posts[$i]['name'] ?></span>
                    </li>
                </a>
            <?php endfor; ?>
        </div>
    </div>
</div>
</body>
</html>
```

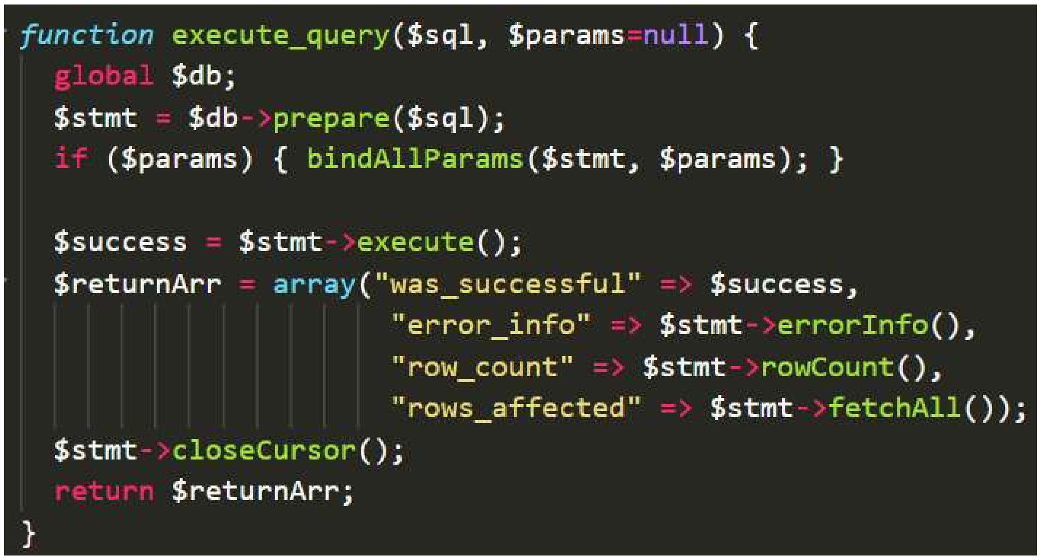
- **Code snippet for #3 (Redirect if not logged in)**



header.php

```php
<?php
    session_start();
    if(!isset($_SESSION['login'])){
      header("location:login.php");
    }
?>
<body>
```

- **Code snippet for #4 (Prepare statement)**



```php
function execute_query($sql, $params=null) {
  global $db;
  $stmt = $db->prepare($sql);
  if ($params) { bindAllParams($stmt, $params); }

  $success = $stmt->execute();
  $returnArr = array("was_successful" => $success,
                    "error_info" => $stmt->errorInfo(),
                    "row_count" => $stmt->rowCount(),
                    "rows_affected" => $stmt->fetchAll());
  $stmt->closeCursor();
  return $returnArr;
}
```