**CST-407 Activity 5 Guide**

Contents

[Activity 5: Database Security 2](#_Toc61937758)

[Part 1: Build a Jokes Application 2](#_Toc61937759)

[Jokes App Version 1: Add and Read Jokes 2](#_Toc61937760)

[Jokes App Version 2: User Accounts 13](#_Toc61937761)

[Part 2: SQL Injection 23](#_Toc61937762)

[SQL Injection Goal 1: Login Without A Password 25](#_Toc61937763)

[SQL Injection Goal 2: Get All Usernames and Passwords 29](#_Toc61937764)

[Conclusion 38](#_Toc61937765)

[Part 3: Security Improvements 38](#_Toc61937766)

[Security Goal 1: Use POST Methods to Process Forms Instead of GET 38](#_Toc61937767)

[Security Goal 2: Enforce Password Complexity Rules When Registering New Users 40](#_Toc61937768)

[Security Goal 3: Use Prepared Statements to Cancel the Effect of SQL Injection Attacks 41](#_Toc61937769)

[Security Goal 4: Use Add Slashes to Sanitize the Input of Every Form in the Application 42](#_Toc61937770)

[Security Goal 5: Use Hashed Passwords to Ensure That Clear Text Passwords Are Not Stored in the Database 43](#_Toc61937771)

# Activity 5: Database Security

This activity has multiple parts/assignments. All assignments must be completed prior to documentation submission.

**Special Note: The code for select activities can be found in the "CST-407 Activity 5 Jokes Basic" and the "CST-407 Activity 5 Database\_ddl\_files” zip files. Refer to these when directed in the guide below.**

## Part 1: Build a Jokes Application

**Overview**

In this activity, students will create an app that displays jokes and their answers, or punchline.

### Jokes App Version 1: Add and Read Jokes

*Version 1 Preview*

Version 1 of the application will have a search and data entry form. Figures 1 through 4 overview what the application will look like.

The index page will show two entry forms, **search** and **add**.

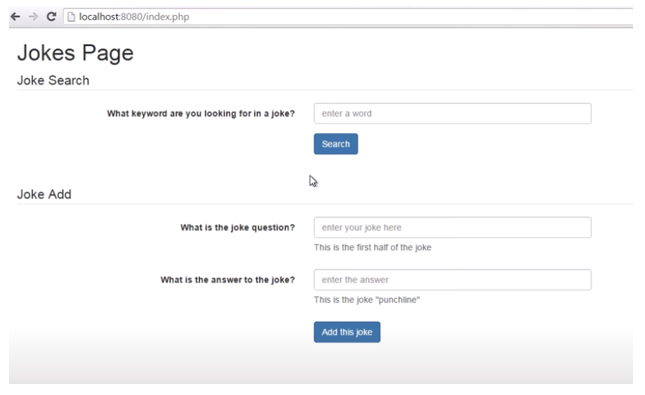


Figure 1. Search and Data Entry Form Shown on the **index.php** Page

Using a specific term, a person will be able to search for a joke.

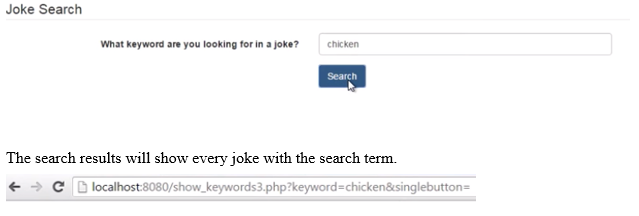


Figure 2. Joke Search Using a Search Term

The search results will show every joke with that search term.

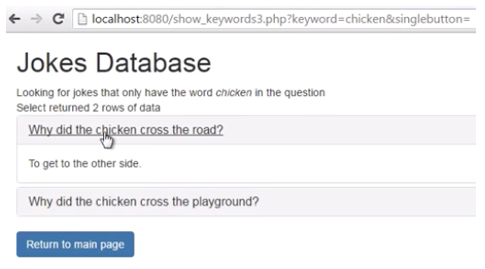


Figure 3. Jokes Database Search Results

The application will initially have a single table to store the jokes.

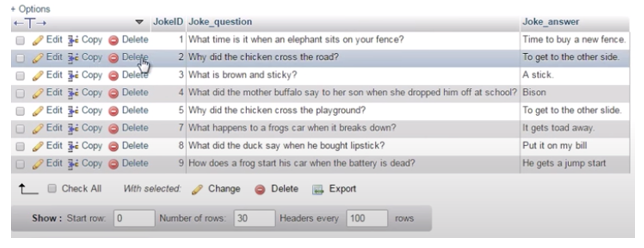


Figure 4. Jokes Table

**Execution**

Execute this activity according to the following guidelines:

1. Install MAMP on your computer. Refer to the "MAMP Installation Instructions," located in the Course Materials.
2. Research available tutorials for "Web and Database Security," to help with the following steps. Consider using a video tutorial found during your research and/or the following written instructions.

*Set up a MySQL database*

1. Either follow the steps below to manually set up the database or use the SQL DDL (SQL Data Definition Language) file below to create the database tables and add some sample jokes as seen in Figures 1–4.
2. The DDL file below is included as a separate file in the **"****CST-407 Activity 5 Database\_ddl\_files Zip File"** archive.

SET SQL\_MODE = "NO\_AUTO\_VALUE\_ON\_ZERO";

SET time\_zone = "+00:00";

--

-- Database: `jokesPart1`

--

-- --------------------------------------------------------

--

-- Table structure for table `jokes\_table`

--

CREATE TABLE `jokes\_table` (

`JokeID` int(11) NOT NULL,

`Joke\_question` varchar(200) NOT NULL,

`Joke\_answer` varchar(500) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--

-- Dumping data for table `jokes\_table`

--

INSERT INTO `jokes\_table` (`JokeID`, `Joke\_question`, `Joke\_answer`) VALUES

(1, 'What time is it when an elephant sits on your fence?', 'Time to buy a new fence.'),

(2, 'Why did the chicken cross the road?', 'To get to the other side'),

(3, 'What did the mother buffalo say to her son when she dropped him off at school?', 'Bison'),

(4, 'Why did the chicken cross the playground?', 'To get to the other slide.'),

(5, 'What happens to a frog when his car breaks down?', 'He gets toad.'),

(6, 'How does a frog start his car when the battery is dead?', 'He gets a jump start.'),

(7, 'Can a kangaroo jump higher than the empire state building?', 'Of course. The Empire State building cannot jump.\r\n'),

(13, ' What did the duck say when he purchased new lipstick?', 'Put that on my bill.'),

(14, 'Want to hear a joke?', 'Your life.'),

(15, ' How does NASA plan a party?', ' They planet.'),

(16, 'Cargo space?', 'No. Car no do that. Car no fly.'),

(17, 'What did the farmer say when he lost his tractor?', 'Where is my tractor?'),

(18, 'I\'ve been told I\'m condescending.', '(that means I talk down to people)'),

(19, 'It\'s hard to explain puns to kleptomaniacs...', 'because they always take things literally.'),

(20, 'When you look really closely, all mirrors...', 'look like eyeballs.'),

(21, 'Why did the old man fall in the well?', 'Because he couldn\'t see that well.'),

(22, 'What\'s ET short for?', 'He\'s only got little legs.'),

(23, 'How do you tell the gender of an ant?', 'Put it in a glass of water. If it sinks its a girl ant. If it floats its buoyant.'),

(24, 'Why do cows have hooves instead of feet?', 'They lactose.');

--

-- Indexes for dumped tables

--

--

-- Indexes for table `jokes\_table`

--

ALTER TABLE `jokes\_table`

ADD PRIMARY KEY (`JokeID`);

--

-- AUTO\_INCREMENT for dumped tables

--

--

-- AUTO\_INCREMENT for table `jokes\_table`

--

ALTER TABLE `jokes\_table`

MODIFY `JokeID` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=25;

1. If you elected to not use the SQL DDL file above, then perform the following steps to manually set up the database.
   1. Launch the MAMP application.
   2. Start the servers.
   3. Open the web start page.
   4. Launch the phpMyAdmin application.

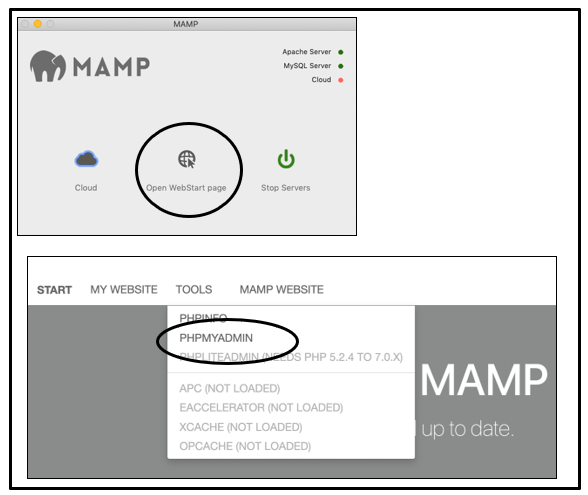


Figure 5. Startup Screens for MAMP and phpMyAdmin

1. Setup the Jokes Table in mySQLadmin.
2. Create three columns in the table: Joke\_ID, Joke\_question, and Joke\_answer.

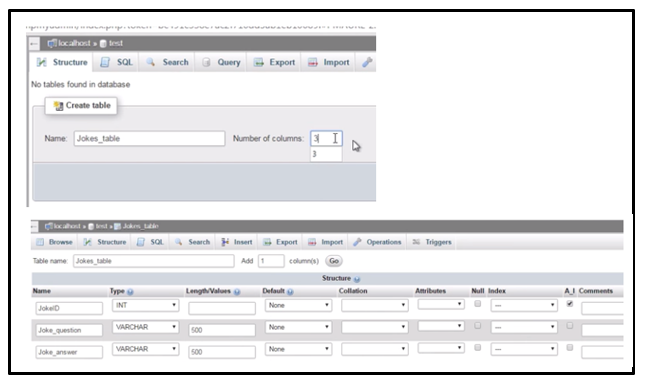


Figure 6. New Table in mySQLAdmin with JokeID, Joke\_question, and Joke\_answer Columns

1. Insert some new jokes into the table.

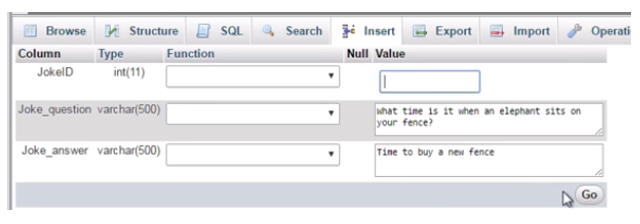


Figure 7. Inserting New Data Into the Jokes Table

*Write PHP Code for the Jokes Application*

1. Either (a) unzip the starting code zip file "**CST-407 Activity 5 Jokes Basic**" and place the contents in the **htdocs** folder of the MAMP application or (b) copy and paste the code examples in the following steps.
2. The starter code can be unzipped, and the **jokes-basic** folder can be placed in **htdocs**.

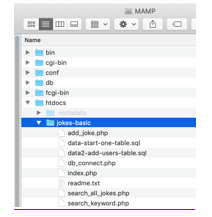


Figure 8. Starter Code Placed Into the **htdocs** Folder of MAMP

1. Create a file, **db\_connect.php**, that connects to the MySQL database.

<?php

ini\_set('display\_errors', '1');

ini\_set('display\_startup\_errors', '1');

error\_reporting(E\_ALL);

// modify these settings according to the account on your database server.

$host = "localhost";

$port = "3306";

$username = "root";

$user\_pass = "root";

$database\_in\_use = "JokesPart1";

$mysqli = new mysqli($host, $username, $user\_pass, $database\_in\_use);

if ($mysqli->connect\_error) {

echo "Failed to connect to MySQL: (" . $mysqli->connect\_errno . ") " . $mysqli->connect\_error;

}

echo $mysqli->host\_info . "<br>";

?>

1. Create a form that allows a user to search the database for a joke.

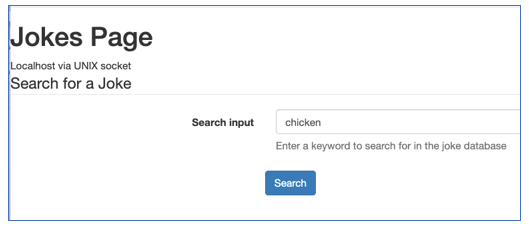


Figure 9. Application Screen That Allows a User to Search for Jokes

1. Use the following source code to create the **index.php** page as seen in Figure 9.

<html>

<head>

<!-- Compiled and minified CSS -->

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">

<!-- jQuery library -->

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>

<!-- Latest compiled JavaScript -->

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>

</head>

<body>

<h1>Jokes Page</h1>

<a href="search\_all\_jokes.php">Show all jokes</a>

<br>

<?php

ini\_set('display\_errors', '1');

ini\_set('display\_startup\_errors', '1');

error\_reporting(E\_ALL);

include "db\_connect.php";

?>

<form class="form-horizontal" action="search\_keyword.php">

<fieldset>

<legend>Search for a Joke</legend>

<div class="form-group">

<label class="col-md-4 control-label" for="keyword">Search input</label>

<div class="col-md-5">

<input id = "keyword" type="search" name="keyword" placeholder="e.g. chicken" class="form-control input-md" required="">

<p class="help-block">Enter a keyword to search for in the joke database</p>

</div>

</div>

<div class="form-group">

<label for="submit" class="col-md-4 control-label"></label>

<div class-"col-md-4">

<button id="submit" name="submit" class="btn btn-primary">Search</button>

</div>

</div>

</fieldset>

</form>

<hr>

</body>

</html>

1. The **search\_keyword.php** file is designed to display search results.

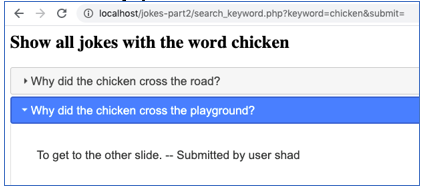


Figure 10. Search Results Formatted With an Accordion CSS Style

1. Use the following source code to create the page **search\_keyword.php** as seen in Figure 10.

<head>

<link rel="stylesheet" href="//code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">

<script src="https://code.jquery.com/jquery-1.12.4.js"></script>

<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>

<script>

$( function() {

$( "#accordion" ).accordion();

} );

</script>

</head><?php

include "db\_connect.php";

ini\_set('display\_errors', '1');

ini\_set('display\_startup\_errors', '1');

error\_reporting(E\_ALL);

$keywordfromform = $\_GET['keyword'];

echo $keywordfromform;

echo "<h2>Show all jokes with the word " . $keywordfromform . "</h2>";

$sql = "SELECT JokeID, Joke\_question, Joke\_answer FROM Jokes\_table WHERE Joke\_question LIKE '%$keywordfromform%'";

$result = $mysqli->query($sql);

if ($result->num\_rows > 0) {

// output data of each row

echo "<div id='accordion'>";

while($row = $result->fetch\_assoc()) {

echo "<h3>" . $row['Joke\_question'] . "</h3>";

echo "<div><p>" . $row["Joke\_answer"]. "</p></div>";

}

echo "</div>";

} else {

echo "0 results";

}

?>

1. Next, use the following source code to create a file, **search\_all\_jokes.php**, that will display all jokes.

<?php

include "db\_connect.php";

$sql = "SELECT JokeID, Joke\_question, Joke\_answer FROM Jokes\_table";

$result = $mysqli->query($sql);

if ($result->num\_rows > 0) {

// output data of each row

while($row = $result->fetch\_assoc()) {

echo "<h3>" . $row['Joke\_question'] . "</h3>";

echo "<div><p>" . $row["Joke\_answer"]. "</p></div>";

}

} else {

echo "0 results";

}

?>

1. Then, create a form that allows the user to add a joke and answer.

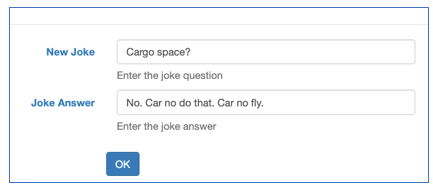


Figure 11. Input Form to Add a New Joke

1. Use the following source code for **index.php** to include a “New Joke” form, as seen in Figure 11.

<html>

<head>

<!-- Compiled and minified CSS -->

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">

<!-- jQuery library -->

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>

<!-- Latest compiled JavaScript -->

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>

</head>

<body>

<h1>Jokes Page</h1>

<a href="search\_all\_jokes.php">Show all jokes</a>

<br>

<?php

ini\_set('display\_errors', '1');

ini\_set('display\_startup\_errors', '1');

error\_reporting(E\_ALL);

include "db\_connect.php";

?>

<form class="form-horizontal" action="search\_keyword.php">

<fieldset>

<legend>Search for a Joke</legend>

<div class="form-group">

<label class="col-md-4 control-label" for="keyword">Search input</label>

<div class="col-md-5">

<input id = "keyword" type="search" name="keyword" placeholder="e.g. chicken" class="form-control input-md" required="">

<p class="help-block">Enter a keyword to search for in the joke database</p>

</div>

</div>

<div class="form-group">

<label for="submit" class="col-md-4 control-label"></label>

<div class-"col-md-4">

<button id="submit" name="submit" class="btn btn-primary">Search</button>

</div>

</div>

</fieldset>

</form>

<hr>

<form class="form-horizontal" action="add\_joke.php">

<fieldset>

<legend>Add a new Joke</legend>

<div class="form-group">

<label class="col-md-4 control-label" for="newjoke">New Joke</label>

<div class="col-md-5">

<input id = "newjoke" type="text" name="newjoke" placeholder="joke question" class="form-control input-md" required="">

<p class="help-block">Enter the joke question</p>

</div>

<label class="col-md-4 control-label" for="jokeanswer">Joke Answer</label>

<div class="col-md-5">

<input id = "jokeanswer" type="text" name="jokeanswer" placeholder="joke answer" class="form-control input-md" required="">

<p class="help-block">Enter the joke answer</p>

</div>

</div>

<div class="form-group">

<label for="submit" class="col-md-4 control-label"></label>

<div class-"col-md-4">

<button id="submit" name="submit" class="btn btn-primary">OK</button>

</div>

</div>

</fieldset>

</form>

<?php

$mysqli->close();

?>

</body>

</html>

1. Run the application. Add some new jokes. Search for jokes. Confirm that the user can add jokes, display all jokes, and search for jokes by keyword.

### Jokes App Version 2: User Accounts

1. Now, we will update the database and forms to include the ability to register, log in, and associate each joke with a user.
2. First, update the database schema to store jokes and associate each with a user account.

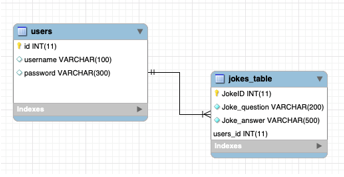


Figure 12. Database Schema for Two Tables That Store Jokes and Users

1. Either create the relationship between the **users** table and **jokes\_table** manually or use the following database script to create the schema pictured in Figure 12.

-- phpMyAdmin SQL Dump

-- version 4.8.5

-- https://www.phpmyadmin.net/

--

-- this is a two table solution. Jokes and Users tables. Used to track logins and ownership of jokes.

SET SQL\_MODE = "NO\_AUTO\_VALUE\_ON\_ZERO";

SET time\_zone = "+00:00";

--

-- Database: `JokesDB`

--

-- --------------------------------------------------------

--

-- Table structure for table `Jokes\_table`

--

CREATE TABLE `Jokes\_table` (

`JokeID` int(11) NOT NULL,

`Joke\_question` varchar(500) NOT NULL,

`Joke\_answer` varchar(500) NOT NULL,

`user\_id` char(100) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--

-- Dumping data for table `Jokes\_table`

--

INSERT INTO `Jokes\_table` (`JokeID`, `Joke\_question`, `Joke\_answer`, `user\_id`) VALUES

(1, 'What time is it when an elephant sits on your fence?', 'It\'s time to buy a new fence.', '13'),

(3, 'Why did the chicken cross the road?', 'To get to the other side.', '15'),

(4, 'What did the mother buffalo say to her son when she dropped him off at school?', 'Bison', '16'),

(5, 'Why did the chicken cross the playground?', 'To get to the other slide.', '13'),

(6, 'What happens to a frog\'s car when it breaks down?', 'It gets toad.', '15'),

(7, 'How does a frog start his car when the battery is dead?', 'He gets a jump start.', '16'),

(10, 'what to hear a joke?', 'Your life', '13'),

(17, 'Why can\'t you trust an atom?', 'They make up everything.', '15'),

(18, 'My sister bet me $100 that I couldn\'t make a car out of spagetti', 'You should have seen the look on her face when I drove pasta.', '16'),

(19, 'Where do animals go when their tail falls off?', 'The retail store.', '23');

-- --------------------------------------------------------

--

-- Table structure for table `users`

--

CREATE TABLE `users` (

`user\_id` int(11) NOT NULL,

`user\_name` text NOT NULL,

`password` text NOT NULL,

`email\_address` text,

`admin\_role` tinyint(1) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--

-- Dumping data for table `users`

--

INSERT INTO `users` (`user\_id`, `user\_name`, `password`, `email\_address`, `admin\_role`) VALUES

(13, 'bill', 'password', 'bill.gates@microsoft.com', 1),

(15, 'kim', 'password', ' kim.ill@north.ko', 0),

(16, ' don ', 'don', ' don.trump@whitehouse.gov ', 0),

(17, 'bob', ' bob ', ' bob@microsoft.com', 0),

(18, ' melinda ', 'm', ' melinda.gates@microsoft.com ', 0),

(19, 'jim', 'jim', 'jim@msn.com', 0),

(23, 'obama', 'password', 'barak@whitehouse.gov', 0);

--

-- Indexes for dumped tables

--

--

-- Indexes for table `Jokes\_table`

--

ALTER TABLE `Jokes\_table`

ADD PRIMARY KEY (`JokeID`);

--

-- Indexes for table `users`

--

ALTER TABLE `users`

ADD PRIMARY KEY (`user\_id`);

--

-- AUTO\_INCREMENT for dumped tables

--

--

-- AUTO\_INCREMENT for table `Jokes\_table`

--

ALTER TABLE `Jokes\_table`

MODIFY `JokeID` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=20;

--

-- AUTO\_INCREMENT for table `users`

--

ALTER TABLE `users`

MODIFY `user\_id` int(11) NOT NULL AUTO\_INCREMENT, AUTO\_INCREMENT=25;

1. Next, we will only allow logged in users to add a new joke. An attempt to add a joke without logging in will result in the following message.



Figure 13. Message Indicating That a New Joke Cannot be Added Unless First Logged In

1. Use the source code below to update the **add\_joke.php** file as seen in Figure 13.

<?php

session\_start();

if (! $\_SESSION['username']) {

echo "Only logged in users may access this page. Click <a href='login\_form.php'here </a> to login<br>";

exit;

}

include "db\_connect.php";

$new\_joke\_question = addslashes($\_GET['newjoke']);

$new\_joke\_answer = addslashes($\_GET['jokeanswer']);

$userid = $\_SESSION['userid'];

echo "<h2>Trying to add a new joke " . $new\_joke\_question . " and " . $new\_joke\_answer . "</h2>";

$stmt = $mysqli->prepare("INSERT INTO Jokes\_table (JokeID, Joke\_question, Joke\_answer, users\_id) VALUES (null, ?, ?, ?)");

$stmt->bind\_param("ssi", $new\_joke\_question, $new\_joke\_answer, $userid);

$stmt->execute();

$stmt->close();

include "search\_all\_jokes.php";

echo "<a href = 'index.php'>Return to main</a>";

?>

1. Now, we will create the following login form.

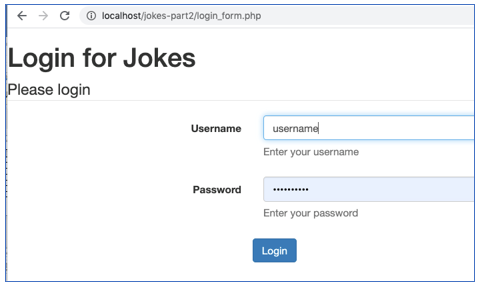


Figure 14. Login Form with Username and Password

1. Use the source code below in **login\_form.php** to create the form seen in Figure 14.

<html>

<head>

<!-- Latest compiled and minified CSS -->

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">

<!-- jQuery library -->

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>

<!-- Latest compiled JavaScript -->

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>

</head>

<body>

<h1>Login for Jokes</h1>

<?php

include "db\_connect.php";

//include "search\_all\_jokes.php";

?>

<form class="form-horizontal" action="process\_login.php" method="post">

<fieldset>

<legend>Please login</legend>

<div class="form-group">

<label class="col-md-4 control-label" for="username">Username</label>

<div class="col-md-5">

<input id = "username" type="text" name="username" placeholder="your name" class="form-control input-md" required="">

<p class="help-block">Enter your username</p>

</div>

</div>

<div class="form-group">

<label class="col-md-4 control-label" for="keyword">Password</label>

<div class="col-md-5">

<input id = "password" type="password" name="password" placeholder="password" class="form-control input-md" required="">

<p class="help-block">Enter your password</p>

</div>

</div>

<div class="form-group">

<label for="submit" class="col-md-4 control-label"></label>

<div class-"col-md-4">

<button id="submit" name="submit" class="btn btn-primary">Login</button>

</div>

</div>

</fieldset>

</form>

<?php

//include "search\_keyword.php";

$mysqli->close();

?>

</body>

</html>

1. Use the source code below to create the file **process\_login.php**.

<html>

<head>

</head>

<?php

session\_start();

error\_reporting(E\_ALL);

ini\_set('display\_errors', 1);

include "db\_connect.php";

$username = $\_POST['username'];

$password = $\_POST['password'];

echo "<h2>You attempted to login with " . $username . " and " . $password . "</h2>";

$stmt = $mysqli->prepare ("SELECT id, username, password FROM users WHERE username = ?");

$stmt->bind\_param("s", $username);

$stmt->execute();

$stmt->store\_result();

$stmt->bind\_result($userid, $fetched\_name, $fetched\_pass);

if ($stmt->num\_rows == 1 ) {

echo "Found 1 person with that username<br>";

$stmt->fetch();

if (password\_verify($password, $fetched\_pass)) {

echo " pw matches<br>";

echo "<p>Login success</p>";

$\_SESSION['username'] = $username;

$\_SESSION['userid'] = $userid;

}

else {

echo "Password does not match<br>";

}

} else {

echo "0 results. Not logged in<br>";

$\_SESSION = [];

session\_destroy();

}

echo "Session variable = ";

print\_r($\_SESSION);

echo "<br>";

echo "<a href='index.php'>Return to main page</a>";

?>

</html>

1. Next, create the following registration form.

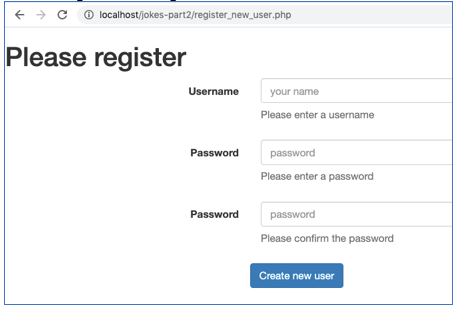


Figure 15. Form to Register a New User Account

1. Use the source code below to create the **register\_new\_user.php** file as seen in Figure 15.

<html>

<head>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>

</head>

<body>

<form class="form-horizontal" action="process\_new\_user.php" method = "get">

<fieldset>

<legend>Register new account</legend>

<div class="form-group">

<label class="col-md-4 control-label" for="keyword">New username</label>

<div class="col-md-5">

<input id = "username" type="text" name="username" placeholder="your name" class="form-control input-md" required="">

<p class="help-block">Create a new login name</p>

</div>

</div>

<div class="form-group">

<label class="col-md-4 control-label" for="keyword">New password</label>

<div class="col-md-5">

<input id = "password" type="password" name="password" placeholder="" class="form-control input-md" required="">

<p class="help-block">Create a new password</p>

</div>

</div>

<div class="form-group">

<label class="col-md-4 control-label" for="keyword">Confirm password</label>

<div class="col-md-5">

<input id = "password-confirm" type="password" name="password-confirm" placeholder="" class="form-control input-md" required="">

<p class="help-block">Retype the password</p>

</div>

</div>

<div class="form-group">

<label for="submit" class="col-md-4 control-label"></label>

<div class-"col-md-4">

<button id="submit" name="submit" class="btn btn-primary">OK</button>

</div>

</div>

</form>

</body>

</html>

1. Use the source code below to create the **process\_new\_user.php** file.

<?php

// add a new user to the database. requires input from register\_new\_user.php

error\_reporting(E\_ALL);

ini\_set('display\_errors', 1);

include "db\_connect.php";

$new\_username = $\_GET['username'];

$new\_password1 = $\_GET['password'];

$new\_password2 = $\_GET['password-confirm'];

echo "<h2>Trying to add a new user " . $new\_username . " pw = " . $new\_password1 . " and " . $new\_password2 . "</h2>";

// check to see if this username has already been registered.

$sql = "SELECT \* FROM users WHERE username = '$new\_username'";

$result = $mysqli->query($sql) or die (mysqli\_error($mysqli));

if ($result->num\_rows > 0) {

echo "The username " . $new\_username . " is already in use. Try another.";

exit;

}

// check to see if the password fields match

else if ($new\_password1 != $new\_password2) {

echo "The passwords do not match. Please try again.";

exit;

} else {

// add the new user

$sql = "INSERT INTO users (id, username, password) VALUES (null, '$new\_username', '$new\_password')";

$result = $mysqli->query($sql) or die (mysqli\_error($mysqli));

if ($result) {

echo "Registration success!";

}

else {

echo "Something went wrong. Not registered.";

}

}

echo "<a href = 'index.php'>Return to main</a>";

1. Next, we will give the user the option to log out.

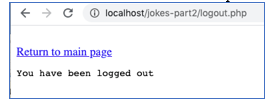


Figure 16. Message to User After Logging Out of the App

1. Use the source code below to create the **logout.php** file**.**

<?php

echo "You have been logged out<br>";

$\_SESSION = [];

session\_destroy();

?>

<a href="index.php">Return to main page</a>

1. Use the source code below to update the **add\_joke.php** file. This will add the user ID to the joke when adding a new joke.

<?php

session\_start();

if (! $\_SESSION['username']) {

echo "Only logged in users may access this page. Click <a href='login\_form.php'here </a> to login<br>";

exit;

}

include "db\_connect.php";

$new\_joke\_question = addslashes($\_GET['newjoke']);

$new\_joke\_answer = addslashes($\_GET['jokeanswer']);

$userid = $\_SESSION['userid'];

echo "<h2>Trying to add a new joke " . $new\_joke\_question . " and " . $new\_joke\_answer . "</h2>";

$stmt = $mysqli->prepare("INSERT INTO Jokes\_table (JokeID, Joke\_question, Joke\_answer, users\_id) VALUES (null, ?, ?, ?)");

$stmt->bind\_param("ssi", $new\_joke\_question, $new\_joke\_answer, $userid);

$stmt->execute();

$stmt->close();

include "search\_all\_jokes.php";

echo "<a href = 'index.php'>Return to main</a>";

?>

1. Use the source code below to update the file **search\_all\_jokes.php.** This will include the user who created each joke.

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>jQuery UI Accordion - Default functionality</title>

<link rel="stylesheet" href="//code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">

<link rel="stylesheet" href="/resources/demos/style.css">

<script src="https://code.jquery.com/jquery-1.12.4.js"></script>

<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>

<script>

$( function() {

$( "#accordion" ).accordion();

} );

</script>

</head>

<?php

include "db\_connect.php";

$sql = "SELECT JokeID, Joke\_question, Joke\_answer, users\_id FROM Jokes\_table";

$result = $mysqli->query($sql);

if ($result->num\_rows > 0) {

// output data of each row

while($row = $result->fetch\_assoc()) {

echo "<h3>" . $row['Joke\_question'] . "</h3>";

echo "<div><p>" . $row["Joke\_answer"] . " submitted by user #" . $row['users\_id'] . "</p></div>";

}

} else {

echo "0 results";

}

?>

1. Use the source code below to update the file **search\_keyword.php.** This will include the user information for each joke.

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>jQuery UI Accordion - Default functionality</title>

<link rel="stylesheet" href="//code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">

<script src="https://code.jquery.com/jquery-1.12.4.js"></script>

<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>

<script>

$( function() {

$( "#accordion" ).accordion();

} );

</script>

</head><?php

include "db\_connect.php";

error\_reporting(E\_ALL);

ini\_set('display\_errors', 1);

$keywordfromform = $\_GET['keyword'];

echo $keywordfromform;

echo "<h2>Show all jokes with the word " . $keywordfromform . "</h2>";

$keywordfromform = "%" . $keywordfromform . "%";

$stmt = $mysqli->prepare("SELECT JokeID, Joke\_question, Joke\_answer, users\_id, username FROM Jokes\_table JOIN users ON users.id = jokes\_table.users\_id WHERE Joke\_question LIKE ?");

$stmt->bind\_param("s", $keywordfromform);

$stmt->execute();

$stmt->store\_result();

$stmt->bind\_result($JokeID, $Joke\_question, $Joke\_answer, $userid, $username);

if ($stmt->num\_rows > 0) {

// output data of each row

echo "<div id='accordion'>";

while($stmt->fetch()) {

$safe\_joke\_question = htmlspecialchars($Joke\_question);

$safe\_joke\_answer = htmlspecialchars($Joke\_answer);

echo "<h3>" . $safe\_joke\_question . "</h3>";

echo "<div><p>" . $safe\_joke\_answer . " -- Submitted by user " . $username ."</p></div>";

}

echo "</div>";

} else {

echo "0 results";

}

?>

**Documentation**

All documentation will be submitted to the learning management system at the end of the activity. Ensure documentation of the following:

1. Create a Loom video that demonstrates the features of the application in working order. Demonstrate a successful registration, login, adding of a new joke, a display all jokes, and a search for a joke by keyword.

## Part 2: SQL Injection

**Overview**

In this activity, students will use the jokes application to demonstrate the ability to compromise the application with SQL injection attacks.

**Execution**

Execute this activity according to the following guidelines:

1. Use the code below to modify the **process\_login.php** page. This will show more details about the SQL results.

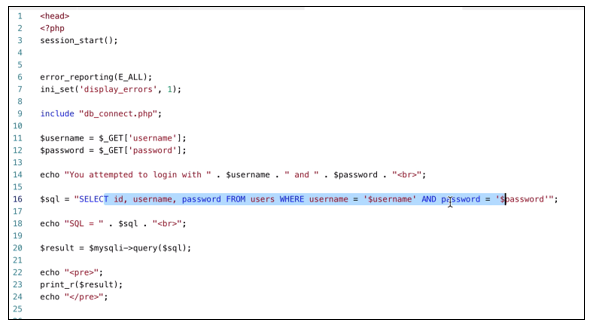


Figure 17. Code Update in **process\_login.php** File

1. The login results should now be shown on the screen when the app is run.

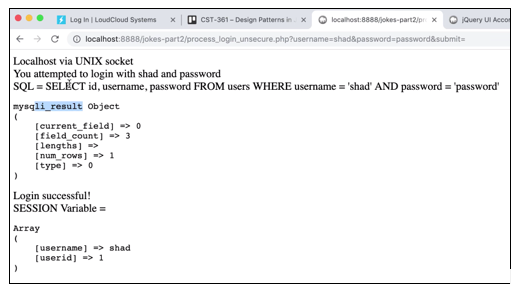


Figure 18. Successful Login Results and the SQL Statement Used in the Login Process

### SQL Injection Goal 1: Login Without A Password

In the following steps, you will use SQL injection techniques to retrieve unauthorized data from the Jokes application. Complete the following steps using the instructions. Repeat the SQL injection steps a second time while recording a Loom video to demonstrate and explain the process.

1. In the phpMyAdmin panel, run the following SQL statement.

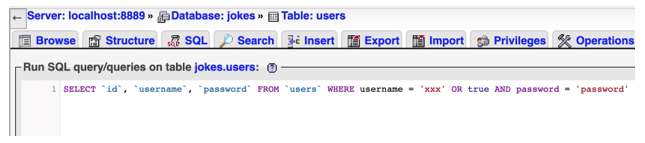


Figure 19. Selecting Items From the Users Table With an OR Statement

1. You should note that adding the OR true statement after the username will select any user, regardless of the string provided. The “xxx” string is not a valid username.
2. Change the “true” statement to 1 = 1. This is a logically equivalent statement.



Figure 20. Selecting Items From the Users Table With a 1=1 Statement

1. Run the following variation of the SQL statement. Note that the following statement will also yield a “true” result.



Figure 21. Selecting Items From the Users Table With a 1= '1' Statement

1. Modify the SQL select statement and run it again. The select statement can have empty strings for the username and password and still yield a result.



Figure 22. Selecting Items From the Users Table With an Empty Username and Password

1. Results should show all of the lines in the users table.

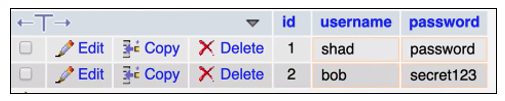


Figure 23. Selection Results Show all Users in the Table

1. Use this knowledge to input a similar statement in the login form. Use the same string for both the username and password in the login form.

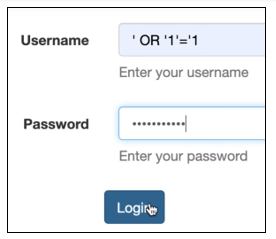


Figure 24. Performing a SQL Injection Attack in the Login Form

1. The resulting login should be successful even though the username and password credentials are invalid. Pay special attention to the resulting SQL statement that was executed to understand why the query selected all of the users.

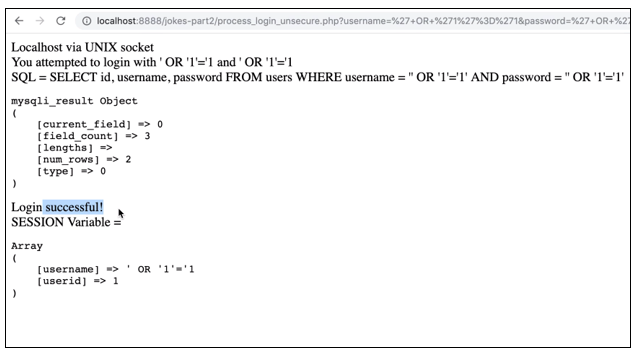


Figure 25. Successful Login With a SQL Injection Attack

1. Use the semicolon and comments marker in the SQL injection attack. Use the following string in the login form. Be sure to add a space following the username string. You can use any string you like for the password.

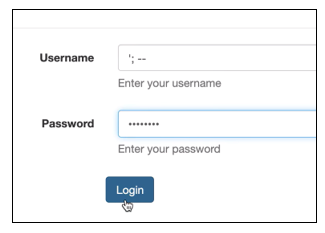


Figure 26. Using a SQL Statement Terminator and Comments in the Login Form

1. Notice the resulting SQL statement that is executed. The only item that could be selected from this query is an item with an empty username.

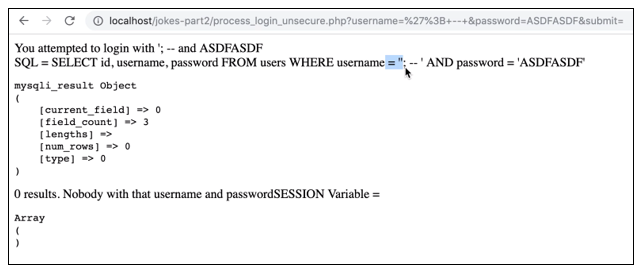


Figure 27. Unsuccessful Login Results That Show the User of a Semicolon and Double Hyphen Statement

1. The **; --** segment of the input is a semicolon (SQL statement terminator) and double hyphen (comment symbol). This has the effect of ending the SQL statement and ignoring the remainder of the line.
2. Input a login name with the format as seen in Figure 28. Any password should work for a successful login if the username is terminated correctly. Be sure to use a trailing space in the username field.

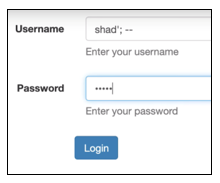


Figure 28. Login Attempt With the “Shad” Account and Invalid Password

1. The results should produce the following SQL statement and login results.

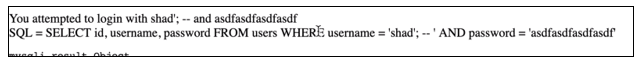


Figure 29. Resulting SQL Statement From the SQL Injection Attack

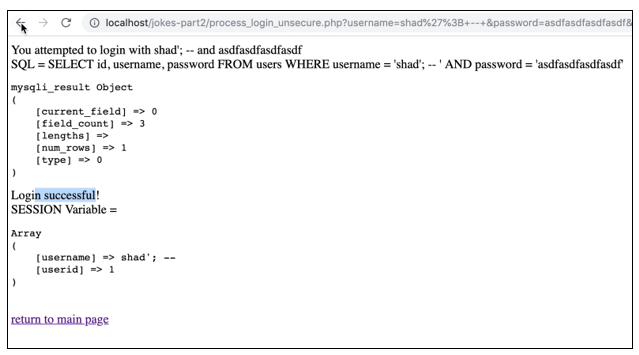


Figure 30. Approved Login

1. The result is an approved login even though the password is invalid. The password is not part of the query since it is after the comment hyphens.

### SQL Injection Goal 2: Get All Usernames and Passwords

1. Fetch all jokes from the database using the following SQL injection attack. Be sure to leave a trailing space in the query.



Figure 31. SQL Injection Attack on the “Search Jokes” Form

1. The search result should show all jokes in the database. This is not really a security breach, but it does demonstrate the power of a SQL injection attack.

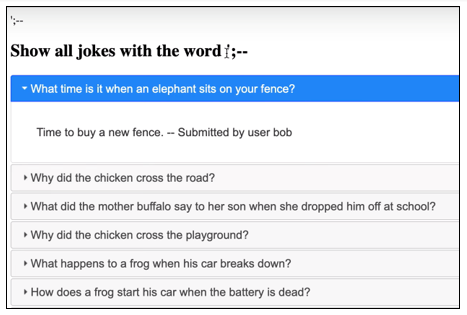


Figure 32. All Jokes are Displayed After the SQL Injection Attack

1. Add the following (line 33) in the **search\_keyword.php** file to display the SQL statement. This will help us see the results of SQL injection attacks.

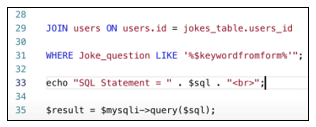


Figure 33. Updated search\_keyword.php File to show the SQL Statement

1. The search results should now show the SQL resulting statement on every search.

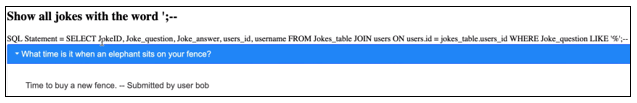


Figure 34. Search Results Now Show the SQL Statement That was Run

1. We will use SQL injection to find out how many columns were used in the application’s select statement. We can tell from Figure 34 that there are five columns (JokeID, Joke\_question, Joke\_answer, users\_id, and username), but an attacker normally would not be able to see this line.
2. Enter the following SQL statement in the search box.

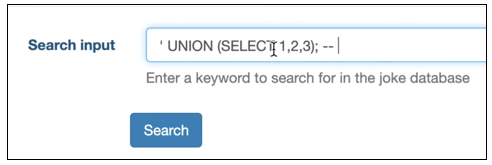


Figure 35. UNION Statement Attempts to Add Three Column Selection

1. This is an attempt to add a second search (UNION) to the normal search. If there happens to be three columns in the select statement, then there will be valid search results. However, there is an error in the SQL statement.

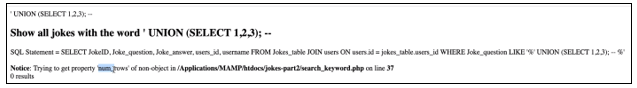


Figure 36. Error on Results Mean We Can Assume That There are Not Three Columns in the Select Statement

1. Use the following input to attempt a statement with four columns.

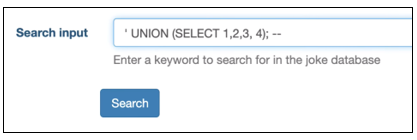


Figure 37. UNION Statement That Attempts to Select Four Columns

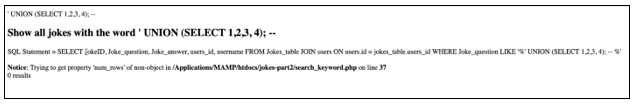


Figure 38. Selecting Four Columns Also Results in an Error

1. Use the following input to attempt a statement with 5 columns.

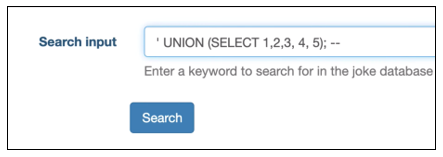


Figure 39. Adding a 5 Column UNION Statement



Figure 40. Success! The Search Results are Displayed

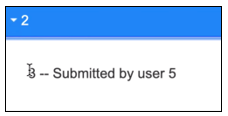


Figure 41. Notice That the Last Joke in the List Shows the Numbers That Were Used in the UNION Statement

1. This tells us that positions 2, 3 and 5 of the SQL query are being printed on the screen. The other two columns are apparently not being displayed. These are the positions in the select statement that we can utilize in the attack. **Joke\_question**, **Joke\_answer**, and **username** are the three columns being displayed.

*Background Information for the Information Schema Table*

1. MySQL has a system table called **information\_schema** that you can view if you have the appropriate “root” permissions.
2. Open the **information\_schema** in mySQLAdmin, as seen in Figure 42

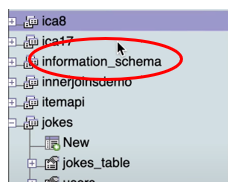


Figure 42. System Table Called **information\_schema**

1. The table’s job is to keep track of all databases, tables, and column names that are stored on the server. You can see a view of the schema here. Notice that there is a row in the schema called “tables” (circled in red).

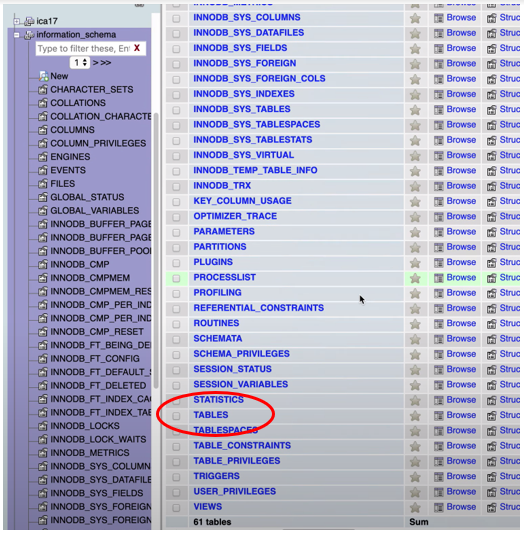


Figure 43. Partial Listing of the **information\_schema** Table

1. Browse the contents of “TABLES.” You should be able to find the name of the tables in the jokes application. Below you should see the **jokes\_table** and the **users** table.

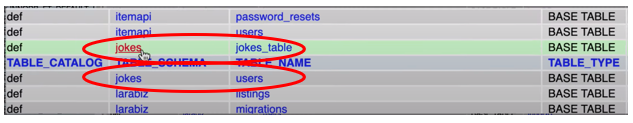


Figure 44. Two Entries for the Jokes Table are Displayed in the **information\_schema** Table

1. Of course, the **information\_schema** table should be kept private. Revealing these table names would enable an attacker to use SQL injection attacks to query these tables if he/she only knew their names and exact spelling.
2. Use the following attack in the search box to determine what tables are stored on the server.

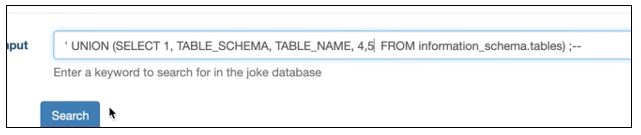


Figure 45. SQL Injection Attack in the Search Form

1. Displayed below the list of jokes is the list of **all tables** on the server. The tables we are interested in are found in the “jokes” schema. Although they are displayed in an odd position, as a joke would be displayed, we can still see the data.

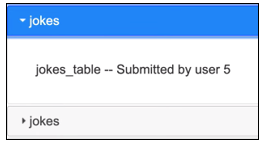


Figure 46. SQL Injection Attack Reveals That jokes\_table is Part of the Jokes Database



Figure 47. SQL Injection Attack Reveals that Users is Part of the Jokes Database

1. We learn from this query that the Jokes database contains two tables: **jokes\_table** and **users**. This allows us to perform the next attack.
2. Use the following query to show the columns found in the **users** table.



Figure 48. The Entire Query is Very Long so it is Shown in a Text Editor Program and Copied Into the Search Joke Form

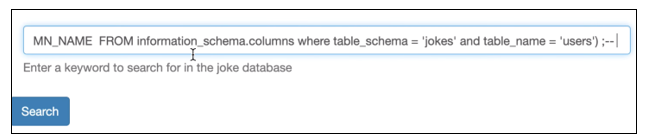


Figure 49. The Entire Query is Very Long so it is Shown in a Text Editor Program and Copied Into the Search Joke Form

1. Below the jokes, the search results show column names of the **users** table.



Figure 50. Resulting SQL Statement of the SQL Injection Attack

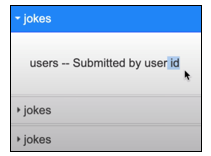


Figure 51. The First Column in the Users Table is *id*

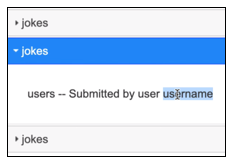


Figure 52. The Second Column in the Users Table is *username*

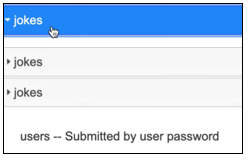


Figure 53. The Third Column in the Users Table is *password*

1. Use the next SQL attack to show the contents of the users table.



Figure 54. SQL Injection Attack to Dump the Users Table Shown in a Text Editor

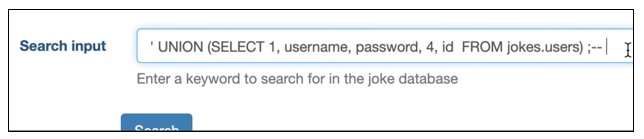


Figure 55. SQL Injection Attack to Dump the Users Table Shown in the Search Form



Figure 56. Resulting SQL Statement From the Attack

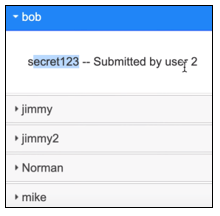


Figure 57. At the Bottom of the Results, We Can See the Users and Passwords!

1. Use Norman’s username and password to log in to the application.

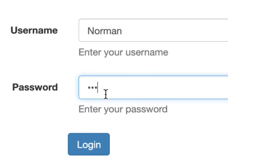


Figure 58. Login Attempt Using a Stolen Password

### Conclusion

Using SQL injection and knowledge of the MySQL system, it is possible to find the table names and query sensitive data from any application. Other database systems will have their version of an **information\_schema** table.

**Documentation**

All documentation will be submitted to the learning management system at the end of the activity. Ensure documentation of the following:

1. Create a Loom video that demonstrates the previous SQL injection attacks.

## Part 3: Security Improvements

**Overview**

In this activity, students will use the jokes application to add the following security improvements:

1. Use POST methods to process forms instead of GET.
2. Enforce password complexity rules when registering new users.
3. Use prepared statements to cancel the effect of SQL injection attacks.
4. Use add slashes to sanitize the input of every form in the application.
5. Use hashed passwords to ensure that clear text passwords are not stored in the database.

**Execution**

Execute this activity according to the following guidelines.

### Security Goal 1: Use POST Methods to Process Forms Instead of GET

1. The data input forms in the Jokes application all use a GET request to handle data. This is a problem because a GET request displays all of the form’s data in the browser’s URL.



Figure 59. URL Showing the Parameters of a GET Request. This is a Poor Security Practice for a Login Form.

1. Change the login form’s method to post (see line 23).



Figure 60. Updated Entry Form That Uses the POST Method Instead of GET

1. In the **process\_login.php** file, modify the code so that the **$username** and **$password** variables get their values from the $\_POST variables (see lines 11 and 12).

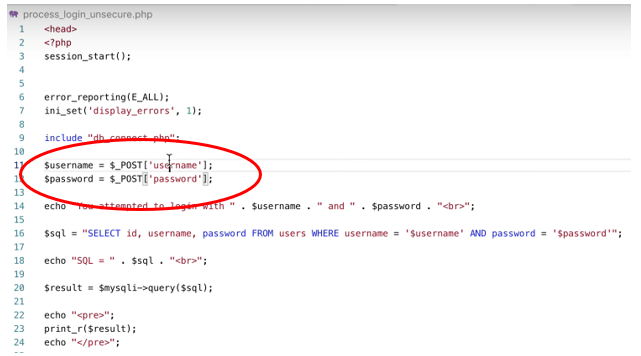


Figure 61. Login Processor is Now Looking for POST Data Instead of GET Data

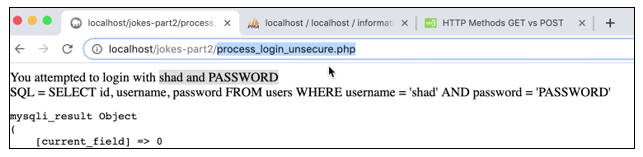


Figure 62. The Process Login Page Should Still Receive the Data From the Login Form, But the Data is Not Displayed in the URL Bar

### Security Goal 2: Enforce Password Complexity Rules When Registering New Users

1. Modify the Login form to have two password fields.



Figure 63. Registration Screen With Password and Password Confirmation Entry Fields

1. In the **process\_new\_user.php** file, add the following regex statements to enforce three password rules:
   1. Passwords must contain a number.
   2. Passwords must have at least one special character.
   3. Passwords must be at least 8 characters or longer.

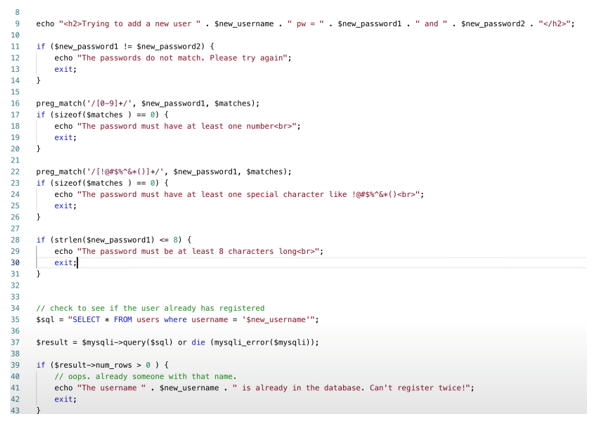


Figure 64. Updated Registration Processing to Enforce Password Complexity Rules

### Security Goal 3: Use Prepared Statements to Cancel the Effect of SQL Injection Attacks

SQL statements are vulnerable to attack because the parameters and commands are both mingled together in a single string. A prepared statement avoids SQL injection by separating the variables from the SQL command.

1. Update the **process\_login.php** file to include prepared statements.

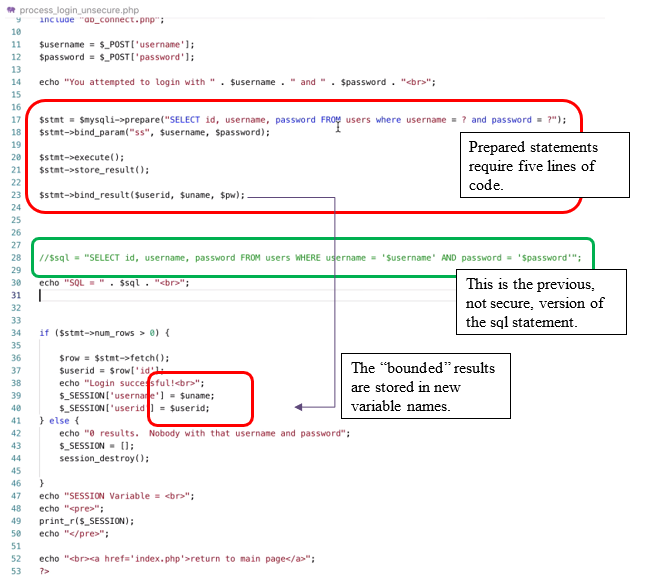


Figure 65. Updated Login Processing to Use Prepared Statements

1. Run the program and verify that previous SQL injection attacks on the login form are no longer effective.

### Security Goal 4: Use Add Slashes to Sanitize the Input of Every Form in the Application

1. For every form handler routine, there should be an **addslashes**( ) statement to sanitize inputs. This example comes from the **add\_joke.php** file.

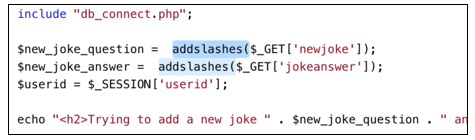


Figure 66. Updated Code That Includes **addshlashes**

1. Run the program and attempt another SQL injection attack into the input form. Verify the input is secure after applying **addslashes** to the program.

### Security Goal 5: Use Hashed Passwords to Ensure That Clear Text Passwords Are Not Stored in the Database

1. In PHP, there are several functions used to hash passwords. We will use statements **password\_hash** and **password\_verify** in the following steps:
   1. Password\_hash() – used to hash a password.
   2. Password\_verify() – used to compare a login attempt with a hashed password.
   3. Password\_get\_info() – returns the name of the hashing algorithm that was used on a password.
2. In the **process\_new\_user.php** file, add a hashed password method to the new password.

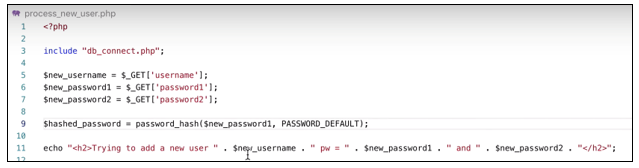


Figure 67. Password Hashing in the Registration Process

1. Add the hashed password to the database when inserting the new user.



Figure 68. Adding a Hashed Password to the Database

1. Register a new user.

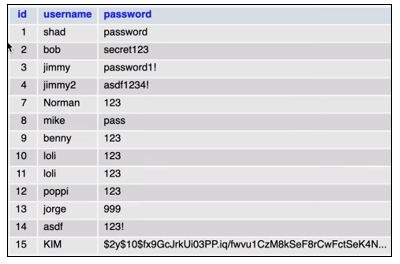


Figure 69. You Should See a Hashed Password in the Users Table

1. Make some changes to **process\_login.php** to handle the hashed passwords. See the script below.

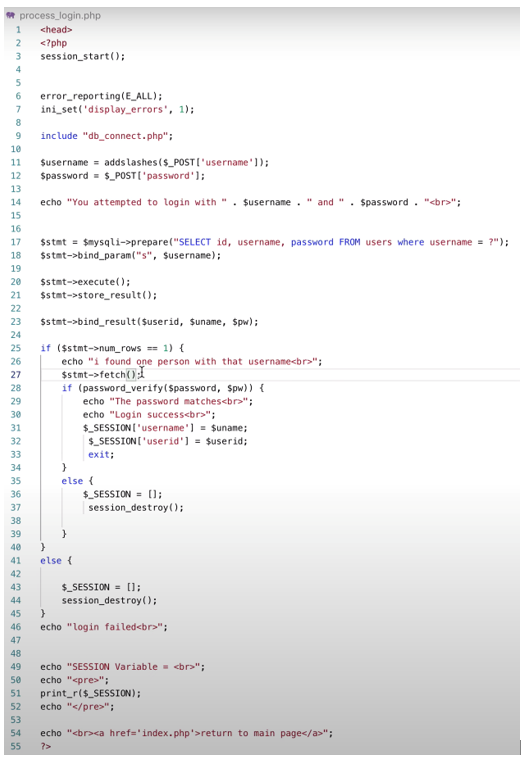


Figure 70. Verify a User Login Using Hashed Passwords

1. Perform a new registration and login to verify that the process can be done correctly.
2. Review the contents of the users table to confirm that the passwords being stored are hashed.

**Submission**

Submit the following to the learning management system:

**Part 1**

1. Create a Loom video that demonstrates the basic application: login, register, search for jokes, and add a new joke.

**Part 2**

1. Create a Loom video that demonstrates the following SQL injection attacks: login with no password, display of all user accounts and their passwords.

**Part 3**

1. Create a Loom video that demonstrates improved security: POST form handling, password complexity, prepared statements, and hashed passwords.