EECS 448

Software Engineering Lab

Lab #10: MySQL

Due Date and Deliverables

The deliverables for this lab will be due on **Wednesday, December 8, 2021 at 11:59 PM**. These include:

- Github repository link containing source code for the required files (see below)
- people.eecs.ku.edu link

Before Starting

Everyone in this course has been granted access to the EECS MySQL servers. You will be given your username and password by a TA. To access the MySQL server, connect to the EECS cycle servers and input the following command:

```
mysql -h mysql.eecs.ku.edu -u username -p
```

Do not enter anything after the -p. You'll be prompted to enter your password.

Please see the EECS MySQL reference for more details about the database server. Some important notes:

- Your DBs are not backed up beyond the scope of this course
- You are capped at 10MB
- You cannot store sensitive information on this server
- You all start with an empty database named the same as your user name

Once you enter the MySQL server, you'll then have to use valid MySQL shell commands. These are different from the commands you'll use to interact with the tables in the database. Unfortunately, things like tab completion aren't available, so you'll have to be extra careful when typing commands. If you're lost, type \h to see a list of options.

MySQL Shell

Selecting a Database

When working from the command line, you needed to make sure to specify which database you are going to be working with. The syntax is shown below:

```
USE <your login name>;
```

CREATE TABLE

Create is a command that you can use to create a new table in the database. You need to name the table, define the column names and what type of information each column represents, and what, if any, keys are in the table.

PRIMARY KEY serves as a unique identifier for each tuple or entry. For example, if you have a table of citizens, a column for SSN could serve as a primary key.

Here's an example of creating a table called persons that has an SSN as a primary key:

```
mysql> CREATE TABLE Persons
(
SSN int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
PRIMARY KEY (SSN)
);
```

NOTE: If you don't add the ';' at the end of the command, you will need to input the'\g' command after.

INSERT INTO

You can add entries, or tuples, to a table by using the command INSERT. You have to pick a table to insert into, the columns you want to insert data into, and the values for each column. When you insert, you are adding a new entry. If you need to update an existing entry, use UPDATE.

Example of add an entry to a table of customers that we assume to exist:

```
mysql> INSERT INTO Customers (CustomerName, ContactName, Address,
City, PostalCode, Country)

VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger',
'4006', 'Norway');
```

Example from W3Schools

SELECT FROM WHERE

One of the main uses of data bases, aside from having permanent data storage, is to query the database in order to generate reports of the existing data.

The format goes:

```
SELECT list-of-column-names
FROM table-name
WHERE criteria
```

For example, you may have a table of employees and their salaries, and you want to find out who makes more than $50,000. Select would be great for this.

```
SELECT emp_name,salary
FROM employees
WHERE salary>50000;
```

Note: The table might have 10 other columns, but the report we'll get will only keep the name and salary in view. If you want to see all columns, you can use the wildcard character, '*'. For example:

```
SELECT *
FROM employees
WHERE salary>50000;
```

Note that you can join multiple WHERE clauses using the AND and OR commands. For example:

```
WHERE salary>50000 AND title='manager';
```

UPDATE SET WHERE

If you need to change a value in an existing entry, you can do so with the UPDATE command. You need provide update the following information:

```
UPDATE table_name
SET col-name='value'
WHERE col-name='value';
```

Notes:

- the equal sign operator is used for both comparison in the WHERE clause and assignment in the SET clause
- You can set multiple values, just separate each assignment by a comma

Example from W3Schools:

```
UPDATE Customers
SET ContactName='Alfred Schmidt', City='Hamburg'
WHERE CustomerName='Alfreds Futterkiste';
```

JOIN

SQL allows you to join tables to gain insight into the information. There are several types of JOIN:

- INNER JOIN (also the default JOIN)
- OUTER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

In this lab, we will use INNER JOIN. INNER JOIN combines two tables based on some criteria, and the result is a combined listing of all entries from **both** tables that meet the criteria. In Venn diagram terms, it yields the intersection of two tables.

W3Schools has a great interactive example of an inner join. Here's the command from the example:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

To add some explanation to it, you have two tables, Customers and Orders. The goal is generate a report that show all customer names and order ids. In short, we are printing a list of all customers and the ids of the orders they've made. So we SELECT the CustomerName from the Customers table and the OrdersID from the Orders table, meaning only these columns will show up in the report. The Customers table **does not** contain any order ids, nor does the Orders table contain the customer names. To get a report that has all this data combined, we INNER JOIN Customers with Orders and the way we match an entry from the Customers table with an entry from the Orders tables is ON the criteria of the CustomerID from the Customers table matching the CustomerID from the Orders table.

The ORDER BY clause is entirely optional, but it does help group all the customers together by name.

Important things to note:

- The syntax of TableName.ColumnName instead of just the ColumnName since the two tables could have a column of the same name.
- Joining **does not** alter the tables in any way, it simple generates a report for your viewing and analysis

PRIMARY and FOREIGN keys

Most of the time, we need to have a unique identifier for each entry in a table. This is what we call a PRIMARY KEY. For example, a table of students could have a student_id as the PRIMARY KEY, or a table of orders could have an order_number. When a table needs to refer to another table's PRIMARY KEY, this is called a FOREIGN KEY. For example, if we have a table of Orders, each order would have its unique order_number, but we would also like to have a column for customer_id in our Orders table so we can know who ordered what.

Example Orders Table:

order_num	customer_id
1	franky97
2	jay321

3	angela31
4	angela31
5	jay321

Note how each order has a unique number, but the foreign key (again, which is a PRIMARY KEY from the Customers table) can be repeated. The FOREIGN KEY also has the benefit of preventing us from putting bad data in our Orders table by requiring the entry for customer_id to exist in the Customers table. In other, we can only make orders for the people listed in our Customers table.

Sometimes a MySQL database has different engines available for creating and managing tables, but not all of them support the use of foreign keys. The InnoDB engine does support foreign keys, however. To ensure that your table is using the InnoDB engine (and therefore that you're able to use foreign keys), use CREATE_TABLE in this way:

```
CREATE TABLE Customers (…) ENGINE=INNODB;
```

AUTO INCREMENT

A quick note: for some primary keys you don't really care what the key is as long as it's unique. Take our Orders table for example. The number doesn't have any meaning beyond just being a unique identifier for the order. Just like your SSN doesn't have any meaning beyond being a way to uniquely identify you.

When this occurs, having a PRIMARY that is set to AUTO INCREMENT is *very* handy. Example.

Other useful commands

SHOW TABLES

- Display a list of table names

SHOW COLUMNS FROM table-name

- Show list of columns from table-name

You can do a lot of other types of comparison. Here's a list of operators.

Commands to be VERY careful with

DELETE FROM WHERE

- Used to delete an entry or entries in a table

```
DELETE FROM employees
WHERE ssn='12345';
```

DROP

- Used to remove traits from tables or entire tables.
- Did you read that? A whole table! Gone. Just like that. Be very careful!

```
DROP TABLE table_name
```

TRUNCATE

- If you want to delete the entries in a table, but not the table itself, essentially emptying the table.

```
TRUNCATE TABLE table_name
```

Connecting PHP and MySQL

PHP and other server-side languages have the ability to connect to and query MySQL databases. The information gained from the database can then be combined with HTML in order to create the dynamic web pages you interact with everyday.

Example from php.net:

```
===mysqli===
PHP has a built in class that handles the interaction with a MySQL
database called mysqli. (Yep, it's a type but doesn't start with a
capital letter. I'll apologize for them.)

Here's an example of connecting, checking connection, querying, and
displaying the results from the database using mysqli:

<?php
$mysqli = new mysqli("database_URL", "my_user", "my_password",
"database_name");

/* check connection */
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_error);
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT
50,5";

if ($result = $mysqli->query($query)) {

    /* fetch associative array */
    while ($row = $result->fetch_assoc()) {
        printf ("%s (%s)\n", $row["Name"], $row["CountryCode"]);
    }

    /* free result set */
    $result->free();
```

```
}

/* close connection */
$mysqli->close();
?>
```

Explanation:

At construction time, you pass in the URL to the database (in our case, “mysql.eecs.ku.edu”), your user name, your password, and the name of the database. Recall that the database you've been given has the same name as your mysql user name. Next, we see if we were able to connect (e.g. maybe the network is down, or we misspelled something). We then build a query string; here you are just writing MySQL code. Then, you send the query to the database. The $result variable is an object that will contain all the rows of information. The while, goes through one row from the result at a time.

Where do those array keys come from? Well, those are the column names from your database.

Free dumps the information in $result. Finally, we close the connection.

Exercises

Exercise 1: Setup Database

We are going to create a *very* basic message board system. We will have two tables, Users and Posts. The Users will house the list of user’s ids. Each user_id will be a unique identifier for a user and can be a series of characters. Posts will contains the posts the users create. Each post will have the content, a post_id, and an author_id. The author_id will reference the user_id of whoever wrote the post. A post_id should be a simple numeric identifier that is automatically chosen by the database. The content will be a series of text, potentially longer than 255 characters.

The following tables:

- Users
    - user_id
- Posts
    - post_id
    - content
    - author_id

All you're doing is create tables. That's it.

Exercise 2: Create User Interface

Files needed:

- CreateUser.html
- CreateUser.php

Using HTML forms again, create a simple frontend that allows the user to create a new user to store in the database. A simple text field and submit button will do. When the user submits, the backend should tell the user if the new user was successfully stored in the database.

The new user should not be stored if:

- The user left the text field empty
- The user already exists

Note: The `msqli.query($query)` method returns, essentially, false if it does not work.

(Again, use “mysql.eecs.ku.edu” in place of “database_URL” when you’re connecting to your database.)

Exercise 3: Create Posts Interface
Files needed:

- CreatePosts.html
- CreatePosts.php

Create a simple form for the user to give their user name and to write a post. When the user submits the post, the backend should tell the user if the post was saved or not.

The post should not be saved if:

- The post has no text
- The post was not written by an existing user

Administrator Interaction

Exercise 4: Administrator Homepage
Files needed:

- AdminHome.html

This should be a series of links to the various administrative pages you will create in the following exercises. You can link directly to ".php" files if you need.

Exercise 5: View Users
Required files:

- ViewUsers.php

Display a table of all users.

Exercise 6: View User Posts
Required files:

- ViewUserPosts.html
- ViewUserPosts.php

Populate a drop down list with all the users. For this drop down box, you'll need to inject some PHP in your frontend. When had has chosen a user and clicks submit, the backend will display a table of all the posts that user has made.

Exercise 7: Delete Posts
Bring down the mod hammer!

Required Files:

- DeletePost.html
- DeletePost.php

The front end will show a table that has a column of posts, authors, and a column of checkboxes in with the heading "Delete?".

The admin should then be able to click the boxes of any posts he/she wants to delete. When the admin clicks submit, all the checked posts should be deleted and a confirmation displaying the post ids of all the deleted posts.

What to submit for grading

Just like in Labs 8 and 9, add links to your index.html page to create a menu. You should have links to exercises 1, 2, 3, and 4. Exercise 4 should contain links to the remaining exercises.

Create a GitHub repository containing the source code for the required HTML and PHP files. Submit the GitHub link and your people.eecs.ku.edu link to BlackBoard by **Wednesday, December 8 at 11:59 PM**.

Rubric

[15pts] Exercise 1
[15pts] Exercise 2
[15pts] Exercise 3
[5pts] Exercise 4
[10pts] Exercise 5
[20pts] Exercise 6
[20pts] Exercise 7