Lab: MySQL User Management, Access Control, and Encryption

Estimated time needed: 30 minutes

Objectives

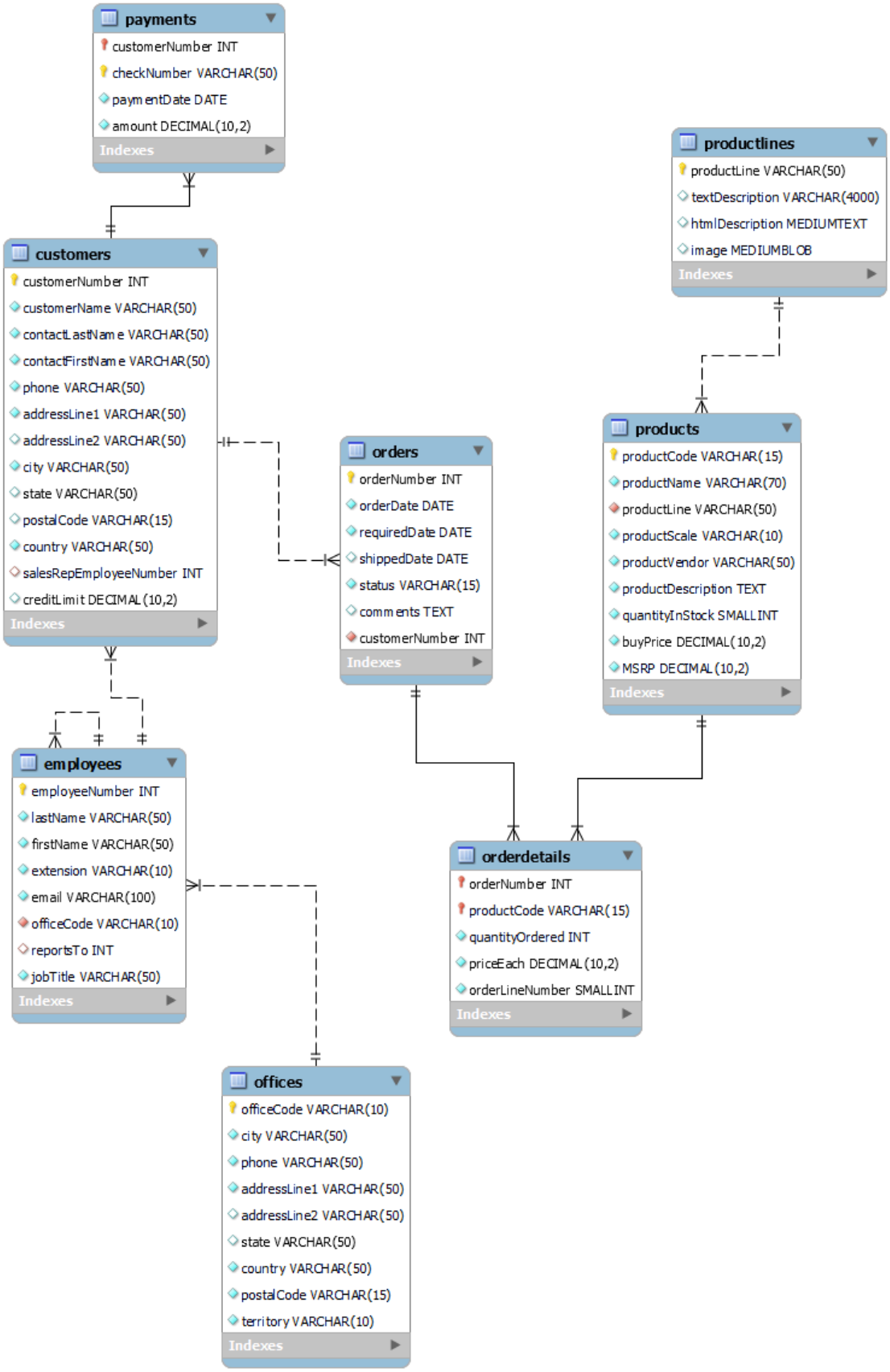
After completing this lab, you will be able to:

- Manage MySQL user accounts and roles using the phpMyAdmin graphical user interface (GUI) tool
- Control access to MySQL databases and their objects
- Secure your data by adding an extra layer of security using data encryption

Prerequisites

Database

In this lab, you will use a customer orders database, which is a modified version of the source database. It is recommended that you use the given database rather than the database from the original source to follow the lab instructions successfully.
The following ERD diagram shows the schema of the customer orders database.



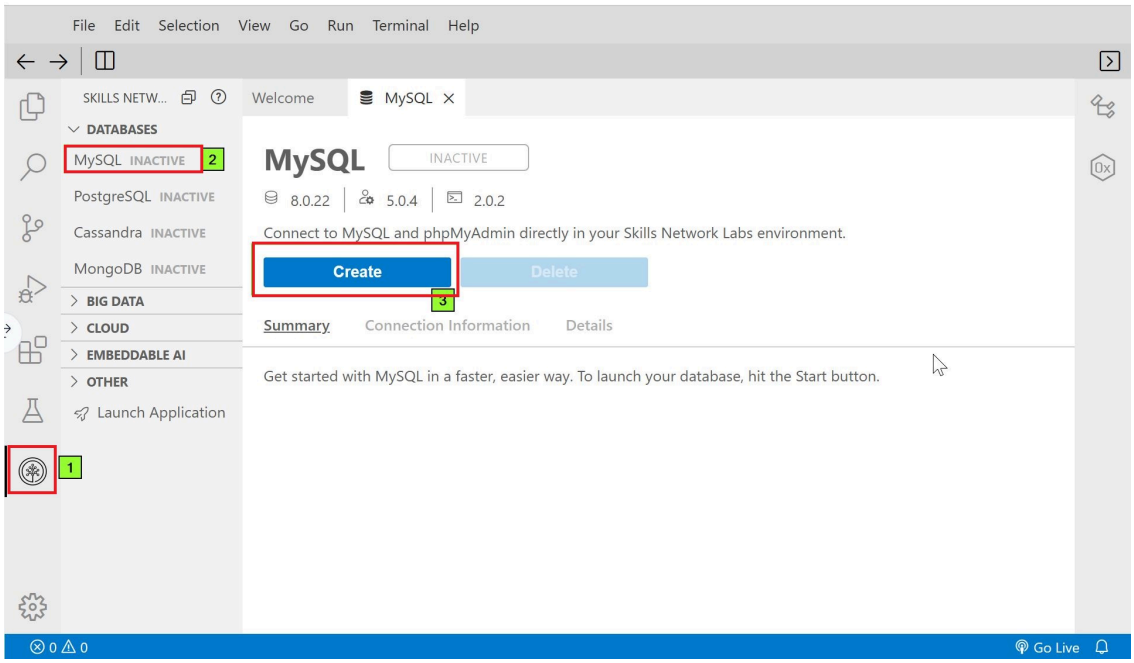
Exercise 1: Manage MySQL user accounts and roles

In this exercise, you will learn how to manage MySQL user accounts and roles using phpMyAdmin.
User management is the process of controlling which users are allowed to connect to the MySQL server and what permissions they have on each database. phpMyAdmin does not handle user management; rather, it

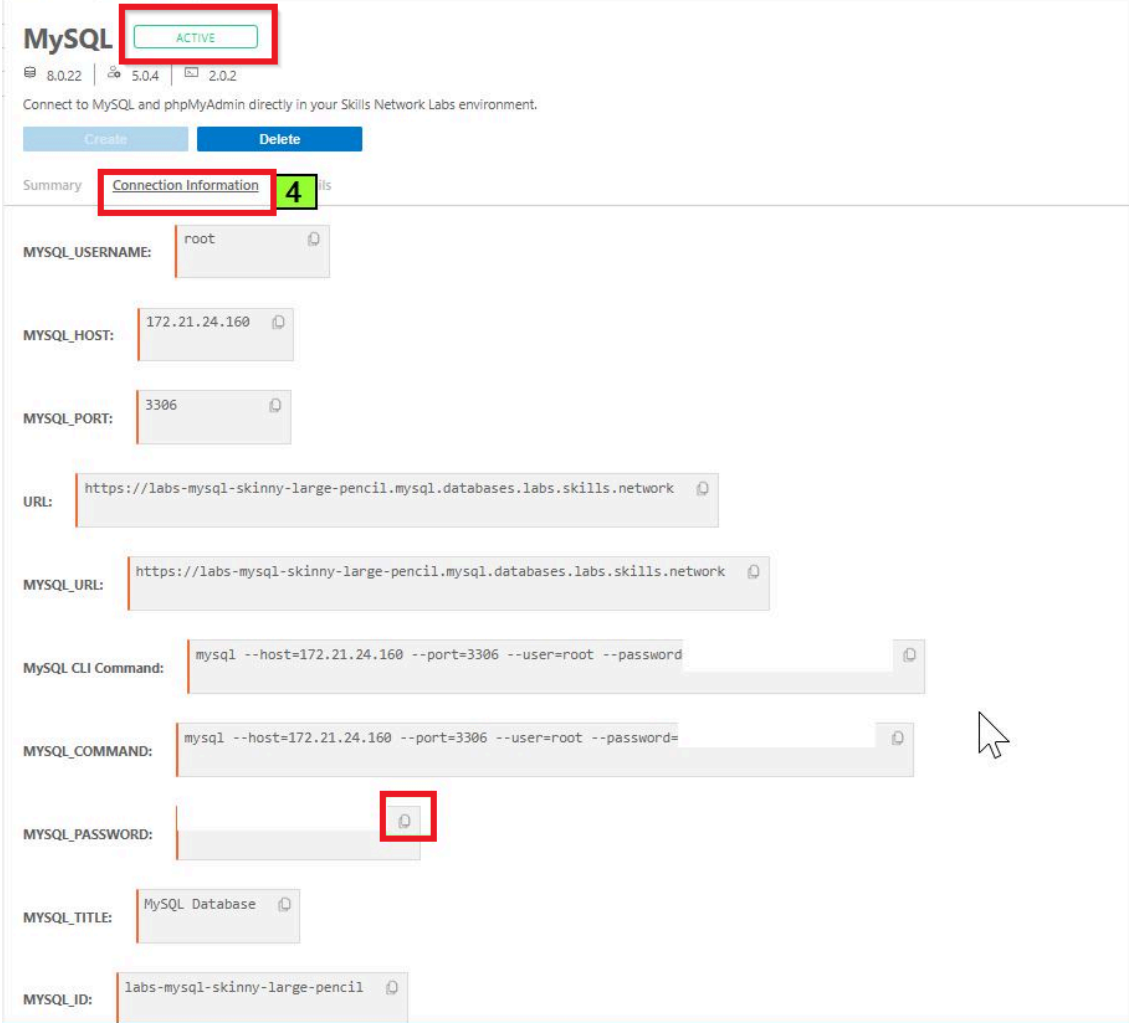
passes the username and password on to MySQL, which then determines whether a user is permitted to perform a particular action. Within phpMyAdmin, administrators have full control over creating users, viewing and editing privileges for existing users, and removing users.

Task 1

1. Go to **Skills Network Toolbox** by clicking the following icon from the side-by-side launched Cloud IDE.
2. From the **Databases** drop-down menu, click **MySQL** to open the MySQL service session tab.
3. Click the **Create** button and wait until the MySQL service session gets launched.



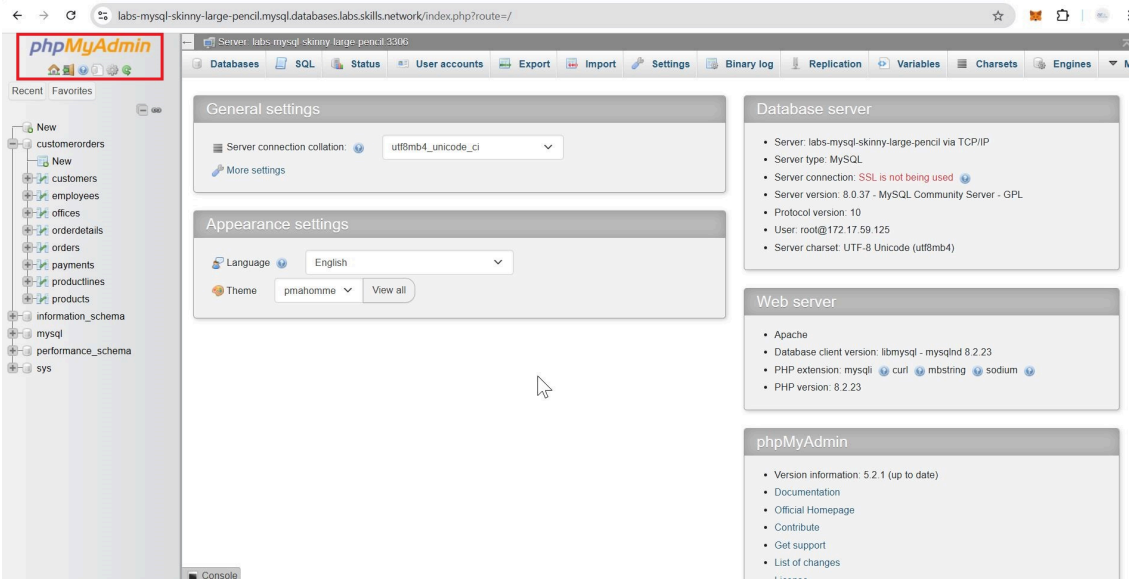
The MySQL server will take a few moments to start. Once it is ready, you will see the green "Active" label at the top of the window.



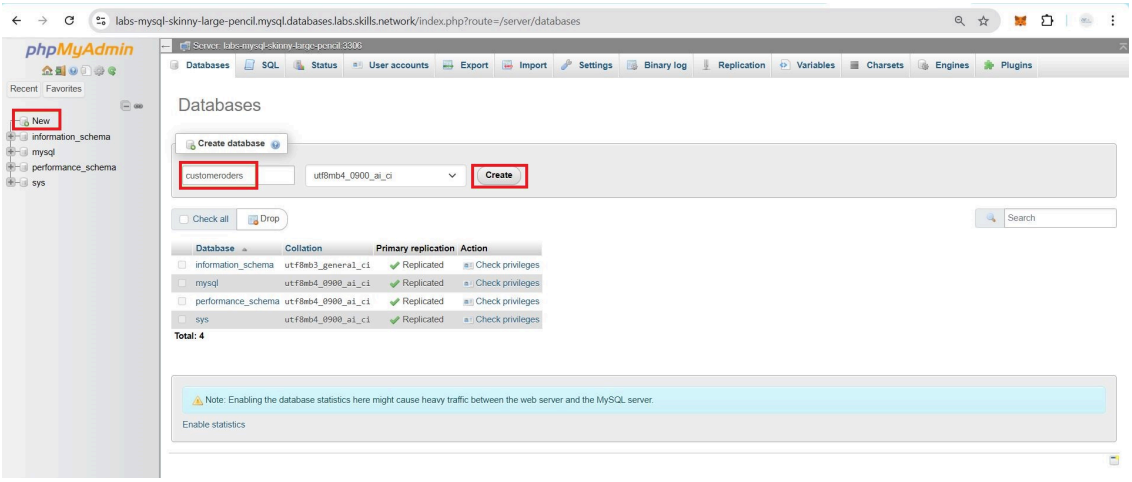
4. Whenever you are required to enter your MySQL service session password from the MySQL service session tab at any step of the lab, copy the password by clicking on the small copy button on the right of the password block. Paste the password into the terminal using **Ctrl + V** (Mac: ⌘ + V), and press **Enter** on the keyboard. For security reasons, you will not see the password as it is entered on the terminal.

Task 2

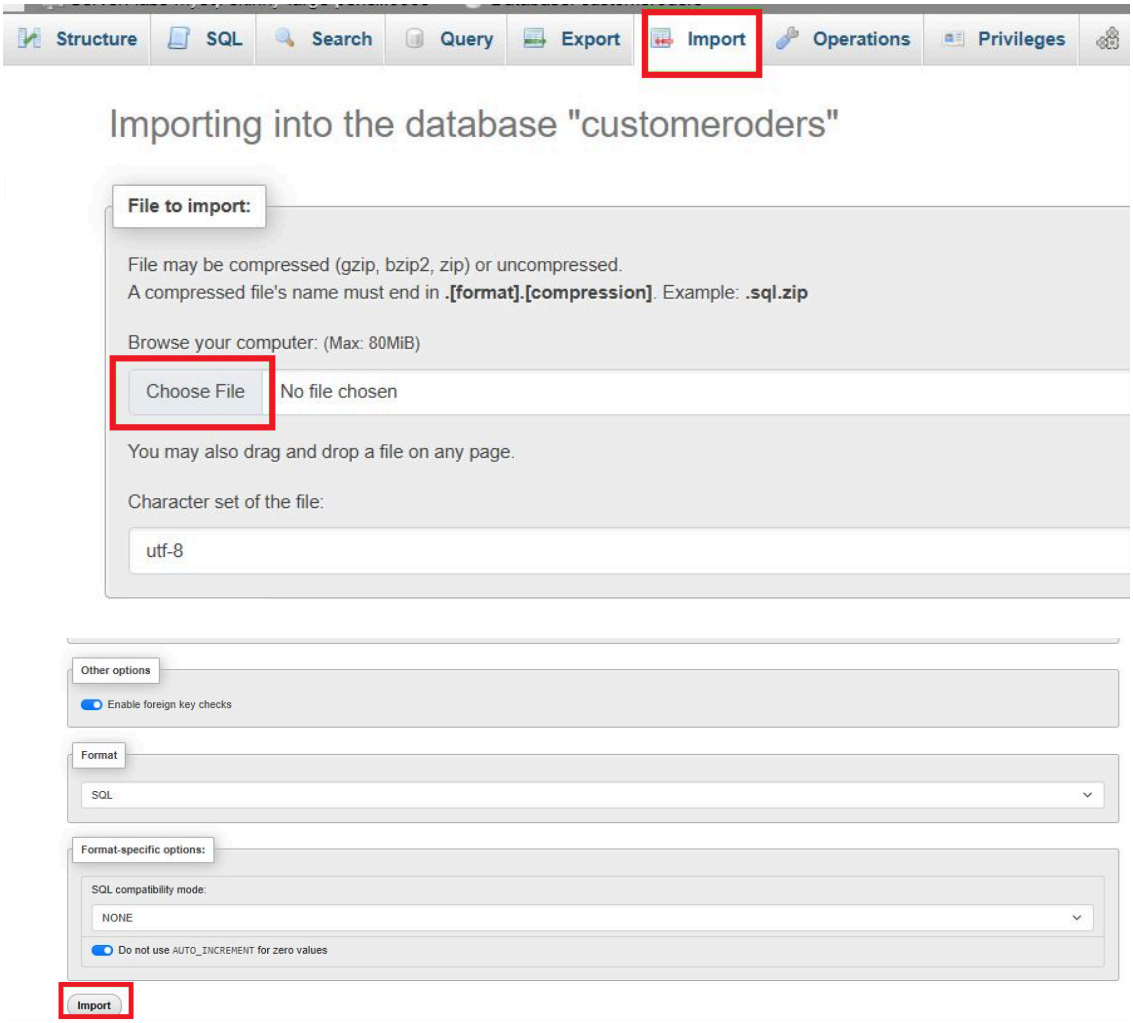
1. Click **phpMyAdmin** button from the mysql service session tab. You will see the phpMyAdmin GUI tool.

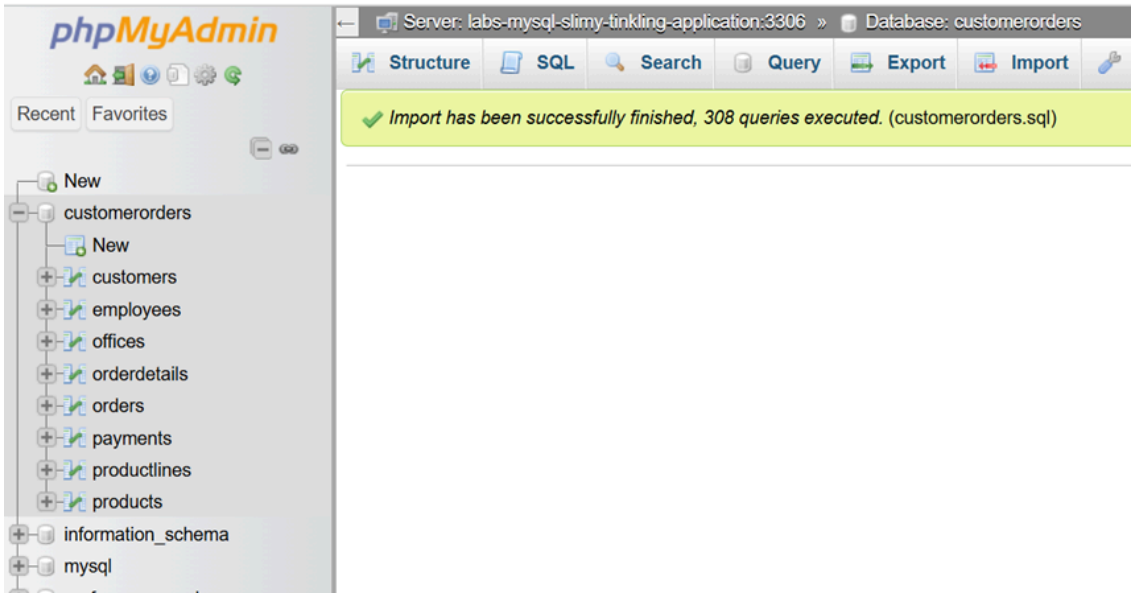


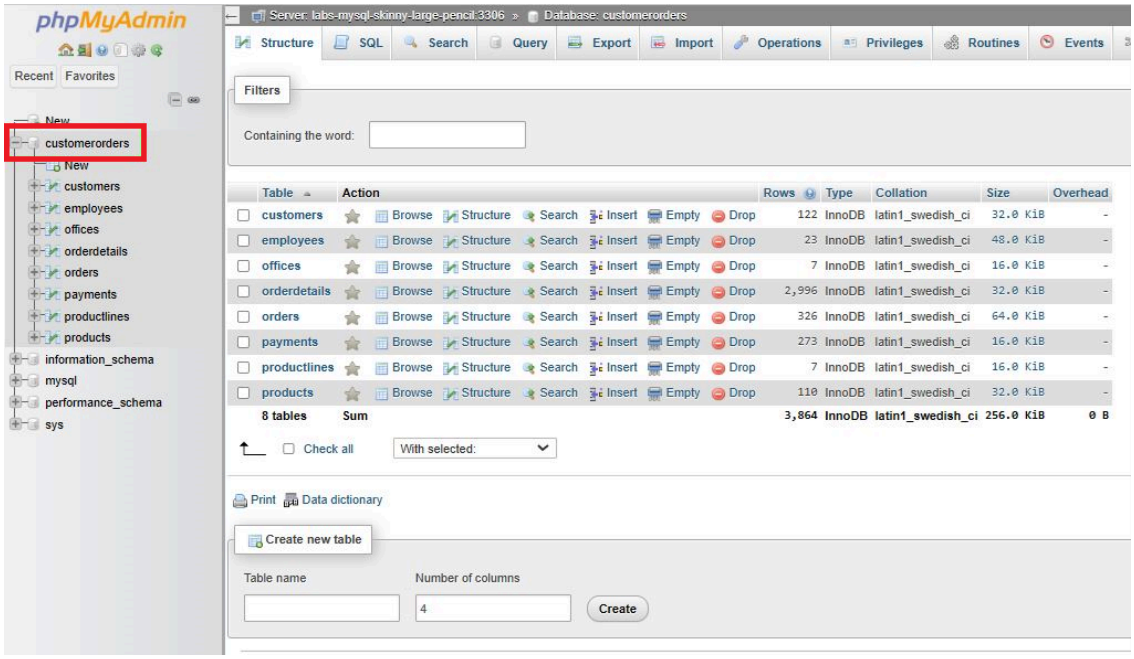
2. In the tree view, click **New** to create a new empty database. Then, enter **customerorders** as the name of the database and click **Create**.



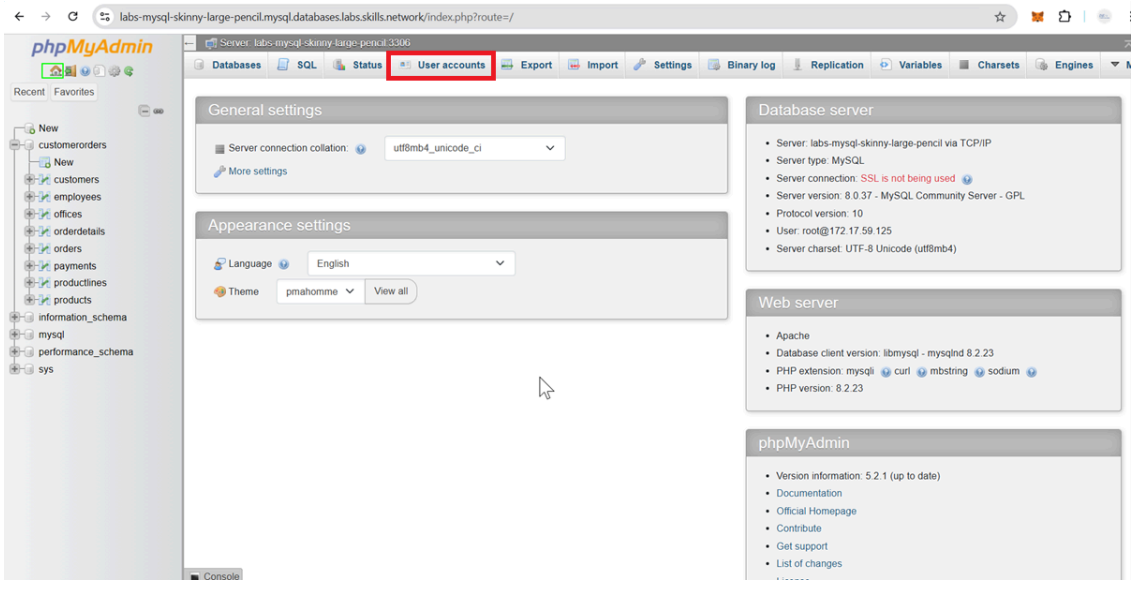
3. Go to the **Import** tab. Upload the following SQL script file using the **Choose File** button (first, right-click this SQL script file to download it to your local computer storage). Then click the **Import** button at the bottom. You will be notified when the import is successfully finished. Click the **Home** icon.

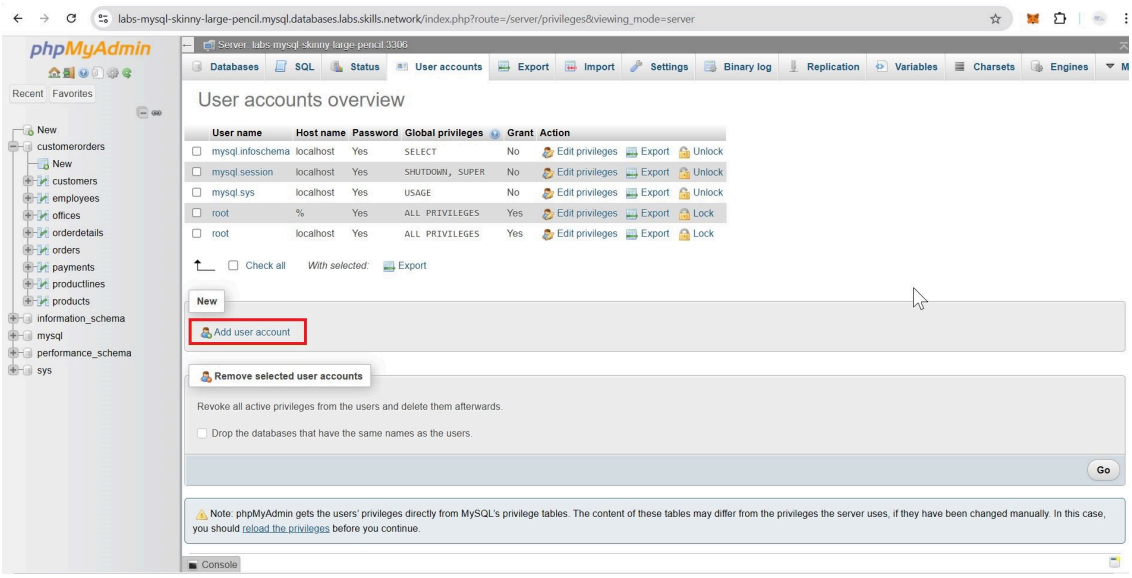




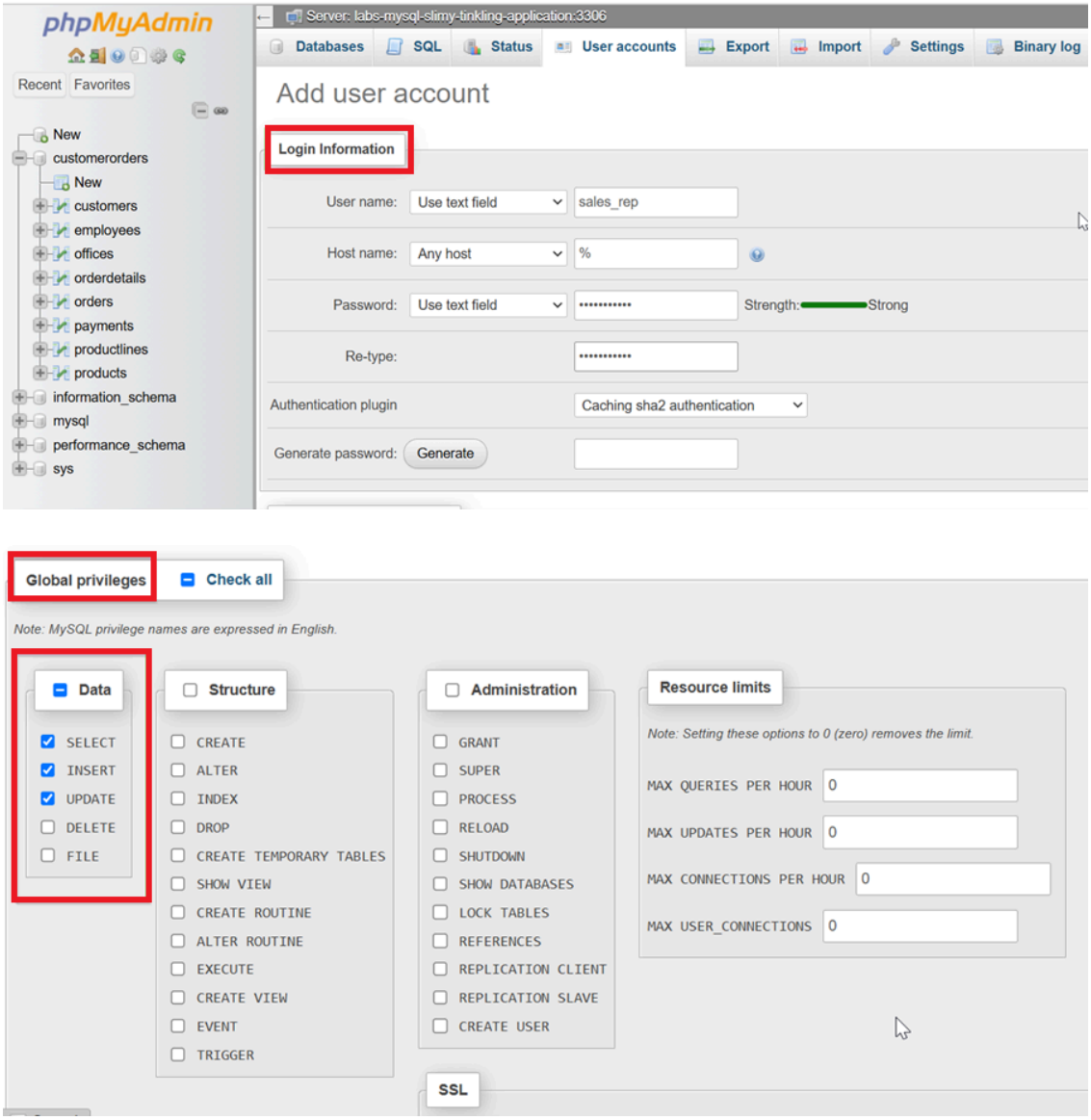


4. Now, you will create a user account with the custom role "sales_rep." sales_rep role will have access to limited tables. Go to the **User accounts** tab and click **Add user account**.

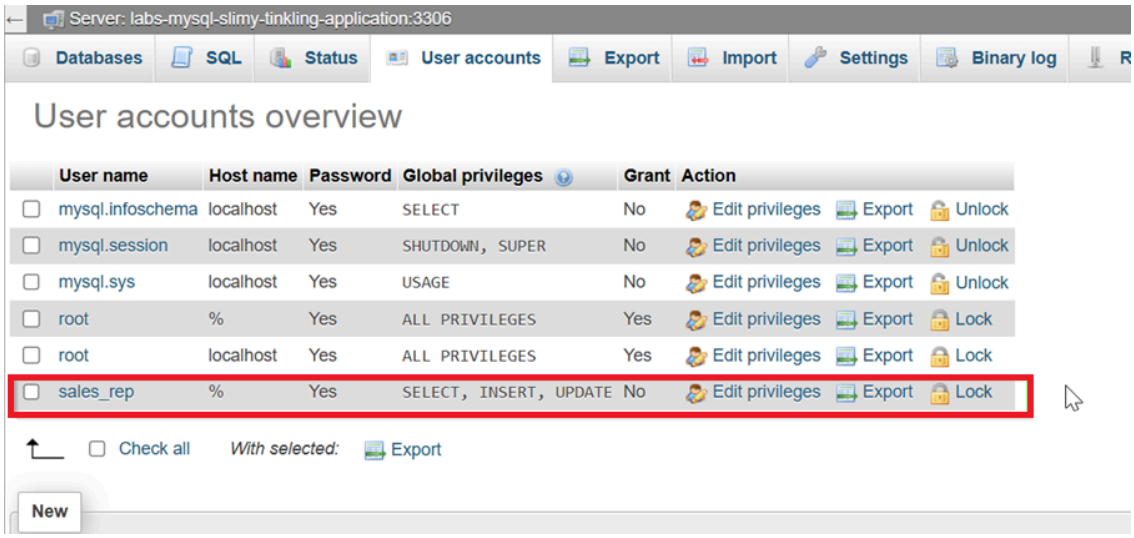




5. Fill in the Login Information as shown in the following image (enter your own password). Under Global privileges, click select option SELECT, INSERT, UPDATE under Data. Scroll down and click **Go**.



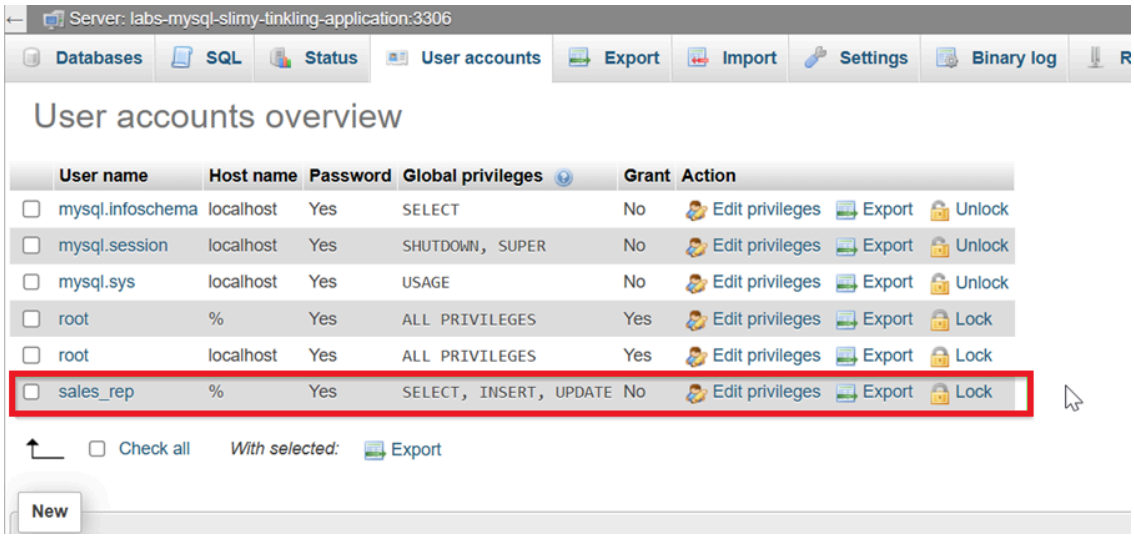
6. You have successfully created a user account with appropriate privileges.



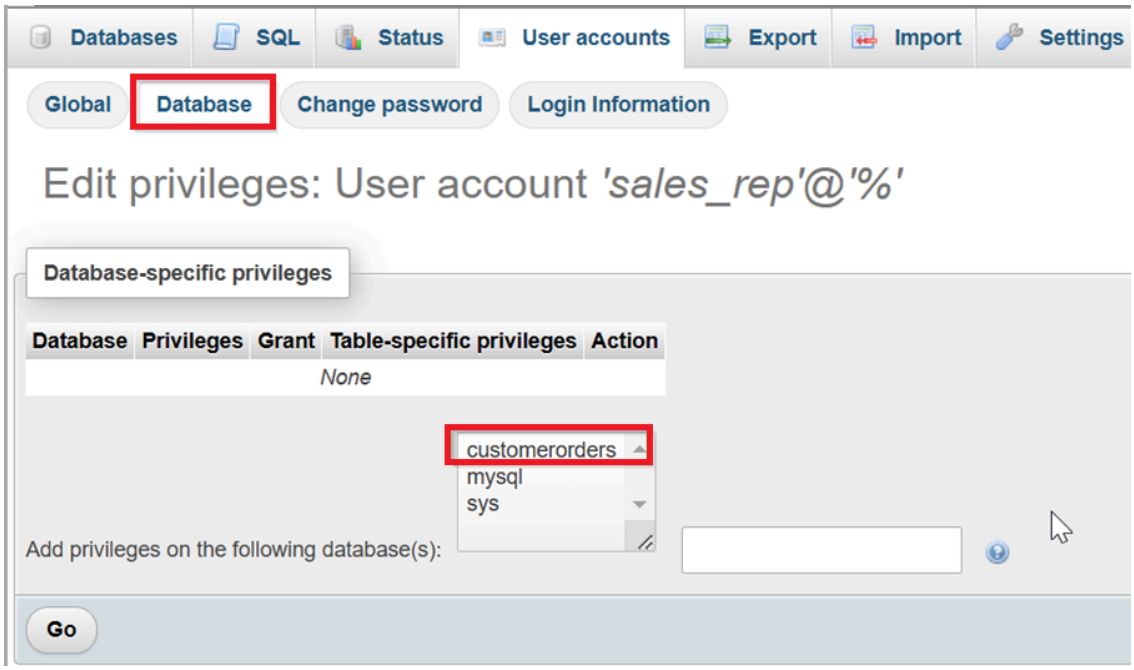
Exercise 2: Control access to MySQL databases and their objects

In this exercise, you will learn how to control access to MySQL databases and their objects.

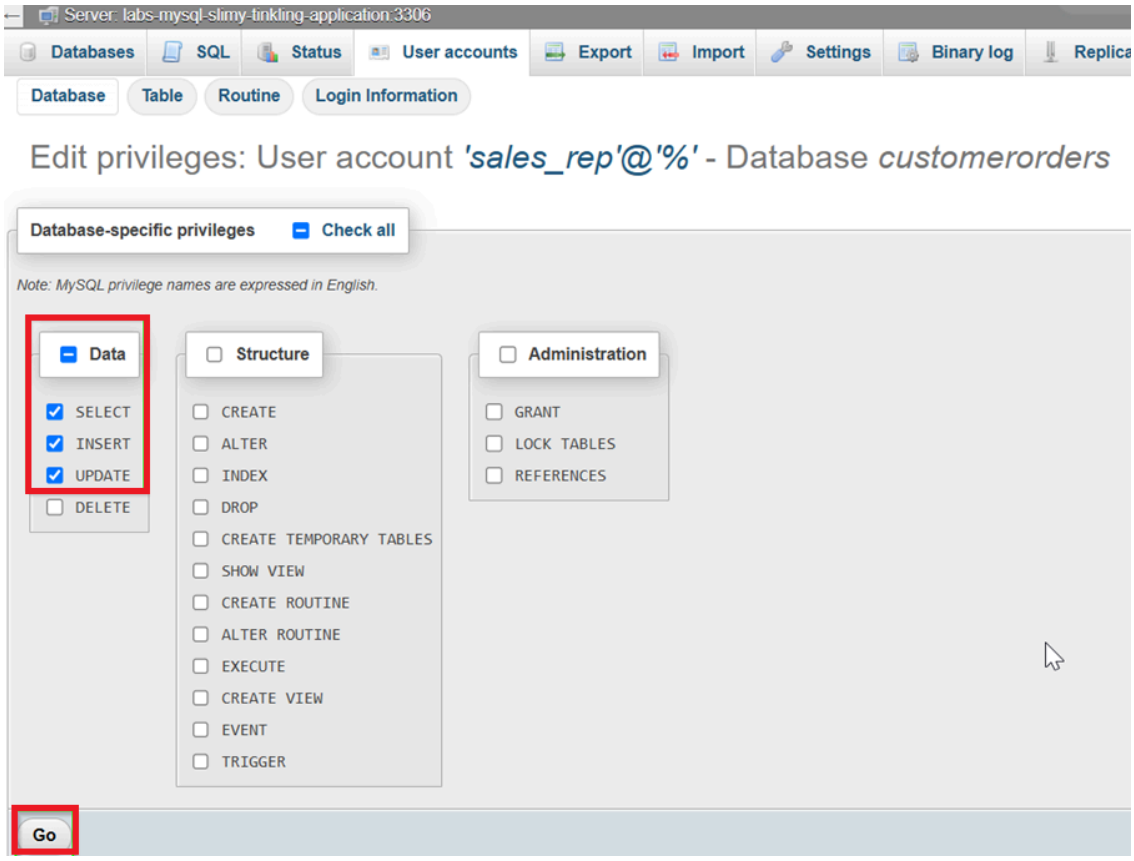
1. Making an exception to the user definition of the sales_rep role you created earlier, you will modify the privileges of this user. You will remove access to payments tables to sales_rep user and restrict sales_rep from updating all the other columns except the column creditLimit of the table customers of the database customerorders.
   Go to **Home > User accounts** tab. Click the **Edit privileges** option of the **sales_rep** user name.



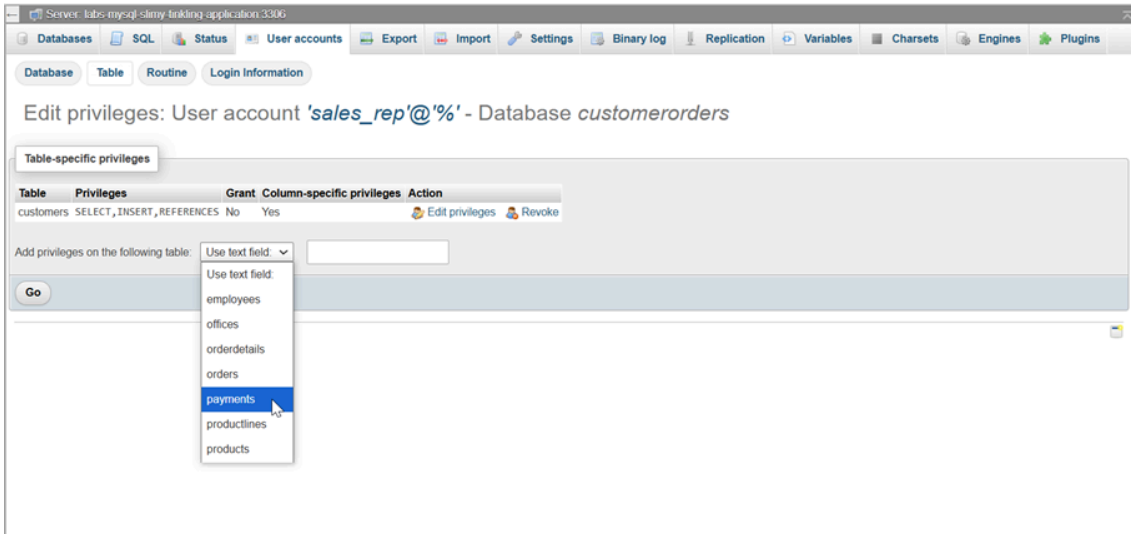
2. Under **Database** sub-tab, select **customerorders** database and click **Go**.



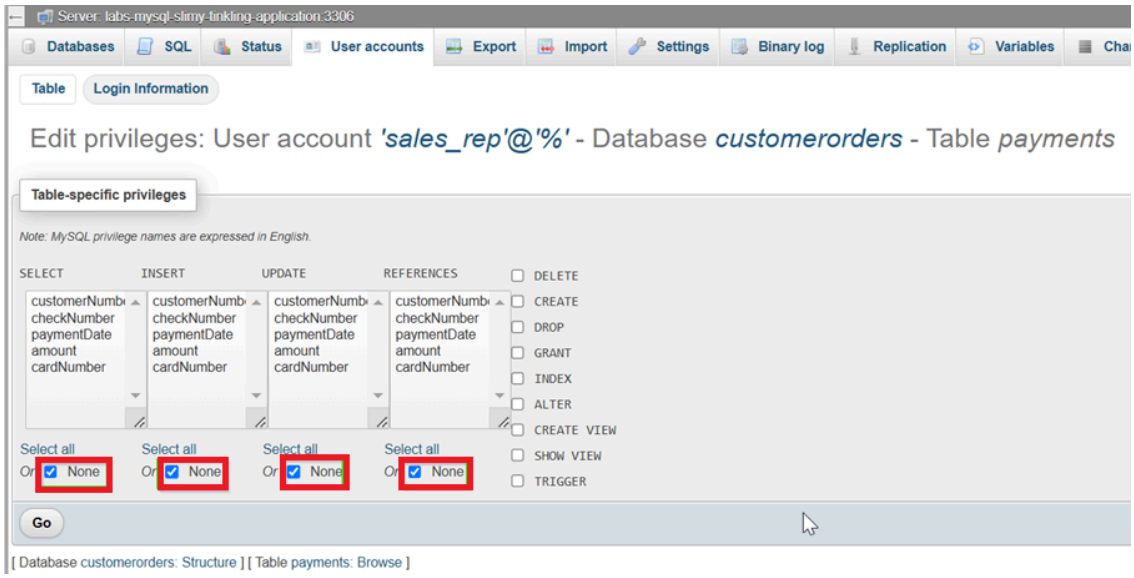
3. Under **Database-specific privileges**, select SELECT, INSERT, and UPDATE options and click **Go** at the bottom.



4. Switch to **Table** sub-tab. Select the **table** payments from the drop-down menu and click **Go**.

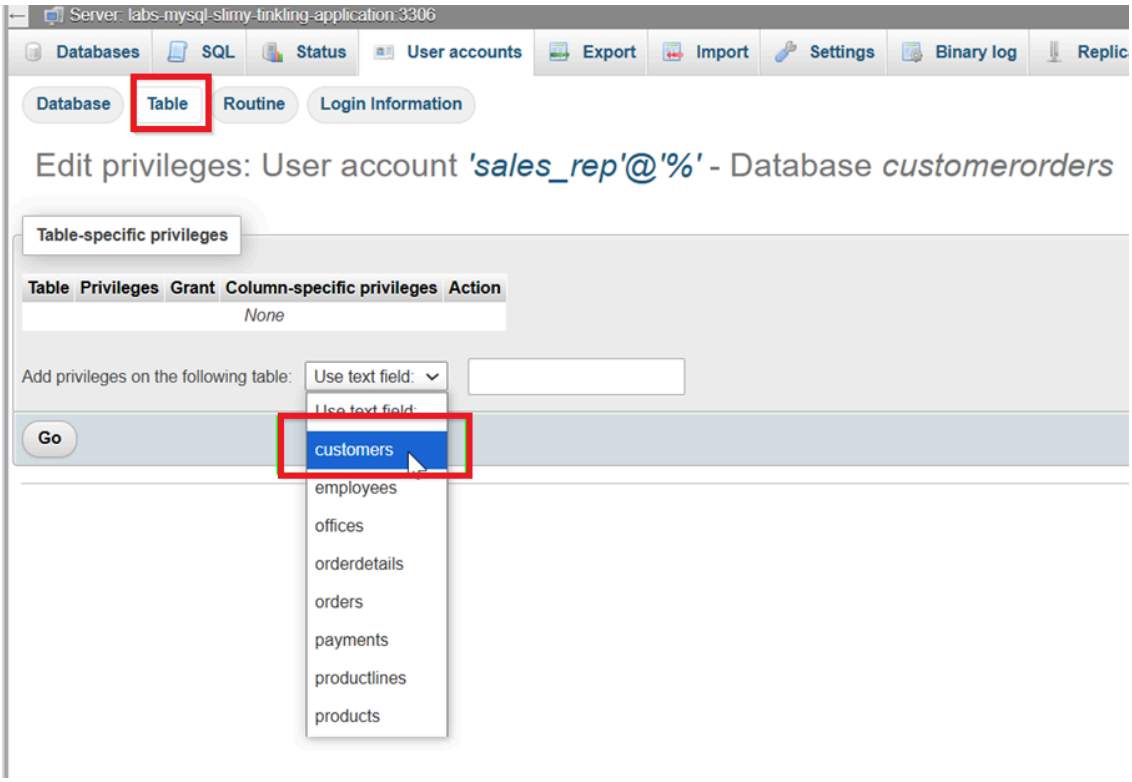


5. Click None option under all sections of SELECT, INSERT, UPDATE, REFERENCES and click **Go**.



6. As a practice exercise perform steps to remove access to employees, offices tables for **sales_rep** user.

7. Switch to **Table** sub-tab. Select the table **customers** from the drop-down menu and click **Go**.



8. Under **Table-specific privileges**, configure all the SQL commands and their custom access to the columns of the table customers. Then click **Go**. Such table-specific privilege configuration will restrict **sales_rep** from updating all the other columns except the column **creditLimit** of the table **customers** of the database **customerorders**.

9. As a practice exercise restrict access to update product table "buyPrice" column by sales_rep user.

Exercise 3: Secure data using encryption

In this exercise, you will learn how to secure your data by adding an extra layer of security using data encryption. Certain parts of your database may contain sensitive information that should not be stored in plain text. This is where encryption comes in.

You will implement encryption and decryption of a column in the customerorders database using the official AES (Advanced Encryption Standard) algorithm. AES is a symmetric encryption where the same key is used

to encrypt and decrypt the data. The AES standard permits various key lengths. By default, a key length of 128 bits is used. Key lengths of 196 or 256 bits can be used. The key length is a trade-off between performance and security.

1. Click the **MySQL CLI** button from the mysql service session tab.

2. First, you will need to hash your passphrase (consider your passphrase is **My secret passphrase**) with a specific hash length (consider your hash length is **512**) using a hash function (here you will use the hash function from **SHA-2** family). It is good practice to hash the passphrase you use since storing the passphrase in plaintext is a significant security vulnerability. Use the following command in the terminal to use the SHA2 algorithm to hash your passphrase and assign it to the variable key_str:

   ```
   SET @key_str = SHA2('My secret passphrase', 512);
   ```

3. Now, let's take a look at the customerorders database. First, you will connect to the database by entering the following command in the CLI:

   ```
   USE customerorders;
   ```

4. Next, let's take a quick look at the customers table in our database with the following command.

   ```
   SELECT * FROM customers LIMIT 5;
   ```

For demonstration purposes, suppose that the last column in the table, labelled addressLine1, contains sensitive data; storing such sensitive data in plain text is an enormous security concern, so let's go ahead and encrypt that column.

5. To encrypt the addressLine1 column, you will first convert the data in the column into binary byte strings of length 255 by entering the following command into the CLI.

   ```
   ALTER TABLE customers MODIFY COLUMN addressLine1 varbinary(255);
   ```

6. Now, to encrypt the addressLine1 column, execute the following command using the AES encryption standard and our hashed passphrase.

   ```
   UPDATE customers SET addressLine1  = AES_ENCRYPT(addressLine1 , @key_str);
   ```

7. Let's go ahead and see if the column was successfully encrypted by taking another look at the customers table. Run the same command as in step 4.

```
SELECT * FROM customers LIMIT 5;
```

8. The supposedly sensitive data is now encrypted and secured from prying eyes. However, we should still have a way to access the encrypted data when needed. To do this, we use the AES_DECRYPT command, and since AES is symmetric, we use the same key for both encryption and decryption. In our case, recall that the key was a passphrase, which was hashed and stored in the variable key_str. Suppose we need to access the sensitive data in that column. We can do so by entering the following command in the CLI:

```
SELECT cast(AES_DECRYPT(addressLine1 , @key_str) as char(255)) FROM customers;
```

9. As a practice exercise you need to encrypt/decrypt cardNumber column in the payments table.

Summary

Congratulations! In this lab, you learned how to manage MySQL user accounts and roles using the phpMyAdmin graphical user interface (GUI) tool. You also learned how to control access to MySQL

databases and their objects. Finally, you learned how to secure your data, adding an extra layer of security using data encryption.

Author(s)

Parikshit Jain

Other Contributor(s)

Rav Ahuja
Abhishek Gagneja