

Databases Demo

Now that we understand the high-level concepts of database operations, let's see how we can create and enter queries into a web application's database in order to carry out basic database administration tasks.

We'll be doing the following database administration tasks in this demo:

- Deploy a web application.
- Navigate to the web application's page to see what data can be viewed by a web client.
- Start a bash session in the database's container and log into the database.
- Use a SQL query to retrieve the contents of the database's primary table.
- Modify that SQL query to find specific data.
- Add new data to the database.
- Delete data from the database.

1. First, we'll navigate to our application's container set directory and stand it up.

- Run the following in a terminal:
 - `cd /home/instructor/Cybersecurity-Lesson-Plans/14-Web_Dev/deploying_databases`
- In this directory is a script called `reset_databases.sh` that will set up the database for this demo. Also, the script will reset the database back to its original state if, at any point, the wrong SQL query is used.
- Run the script with `./reset_databases.sh`
- Then bring up the app with `docker-compose up`.

2. We should be able to see our deployed web app now:

- Open Firefox and navigate to `127.0.0.1:10005` to show the GoodCorp Employee Directory page.
- Scroll down the page and explain that the employee entries here are being read from a database.

3. Open a new terminal window and enter an interactive bash session in the container:


- `docker exec -it activitydb bash` and the prompt should change to `root@<containerID>`.
- We should be familiar with how to enter an interactive bash session in a Docker container.

4. We're going to enter a MySQL session like earlier, but this time, we're also going to add the `-D` argument to specify a database to use.

- Run `mysql -u admin -p123456 -D goodcorpdb`
- The prompt will change to `MariaDB [goodcorpdb]`.
- You should never use such an obvious username and password combination.

5. **SELECT Queries:** In this database we have a table called `employees` that has the list of employees we saw on the webpage earlier. In order to find all of the entries in this table, we can use what is known as the `SELECT` statement to create a query that will return all of the entries from the `employees` table.

- While in the MySQL session, run `SELECT * FROM employees;`

 **Heads Up:** Always remember to end your queries with a semicolon `;` or your database will expect more SQL.

- Run `SELECT * FROM employees;` to show all of the employees we saw earlier:

```
MariaDB [goodcorpdb]> SELECT * FROM employees;
+----+-----+-----+-----+-----+-----+
| id | firstname | lastname | email | department | date_added |
+----+-----+-----+-----+-----+-----+
| 1 | Bob | Brew | bbrew@goodcorp.net | Sales and Marketing | 2020-09-16 11:00:44 |
| 2 | Andrew | Americano | aamericano@goodcorp.net | Research and Development | 2020-09-16 11:00:44 |
```

```

| 3 | Caroline | Cortado | ccortado@goodcorp.net | Human Resources | 2020-09-16 11:00:44 |
| 4 | Deborah | Doppio | ddoppio@goodcorp.net | Operations | 2020-09-16 11:00:44 |
| 5 | Emma | Espresso | eespresso@goodcorp.net | Research and Development | 2020-09-16 11:00:44 |
+---+-----+-----+-----+-----+-----+
5 rows in set (0.000 sec)

```

- We can see that we have five employees in this table and that they are organized by their employee **ID**, **first name**, **last name**, **department**, and the **date** and **time** that they were added to the database.

6. **WHERE Clause:** Sometimes we want to narrow down results when searching through a database, which we can do by using the **WHERE** clause. We can append the **WHERE** clause to queries to narrow down and modify what it's doing. **WHERE** acts as a conditional statement that will return database results when it is met.

- In this example, we'll use **WHERE** to find all employees that are in the Operations department. To do so, we're going to specify that when we use **SELECT * FROM employees**, we only want the rows where the **department** is equal to 'Operations':
- Run **SELECT * FROM employees WHERE department='Operations';**

```

MariaDB [goodcorpdb]> SELECT * FROM employees WHERE department='Operations'
-> ;
+---+-----+-----+-----+-----+-----+
| id | firstname | lastname | email | department | date_added |
+---+-----+-----+-----+-----+-----+
| 4 | Deborah | Doppio | ddoppio@goodcorp.net | Operations | 2020-09-16 11:00:44 |
+---+-----+-----+-----+-----+-----+
1 row in set (0.001 sec)

```

- Using **WHERE** to modify a SQL query is necessary when dealing with databases with huge amounts of data.

7. **INSERT INTO Queries:** If we want to add a new employee to our database, we can use the **INSERT INTO** query.

- This query is used to add new data to a database and takes the format of:

```

INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);

```

- Construct a query using this format. Enter the following:

```

INSERT INTO employees (firstname, lastname, email, department)
VALUES ('Luke', 'Latte', 'llatte@goodcorp.net', 'Logistics');

```

- We are not entering column values for **Id** or **date_added** because they be automatically incremented when we enter a new row.
- Run the query to show the Query OK, 1 row affected (0.001 sec) output, confirming a successful **INSERT** query.

- Now we'll check the entire table again with **SELECT * FROM employees;** to see our new employee, Luke, at the bottom of the table:

```


MariaDB [goodcorpdb]> SELECT * FROM employees;
+---+-----+-----+-----+-----+-----+
| id | firstname | lastname | email | department | date_added |
+---+-----+-----+-----+-----+-----+
| 1 | Bob | Brew | bbrew@goodcorp.net | Sales and Marketing | 2020-09-16 11:00:44 |
| 2 | Andrew | Americano | aamericano@goodcorp.net | Research and Development | 2020-09-16 11:00:44 |
| 3 | Caroline | Cortado | ccortado@goodcorp.net | Human Resources | 2020-09-16 11:00:44 |
| 4 | Deborah | Doppio | ddoppio@goodcorp.net | Operations | 2020-09-16 11:00:44 |
| 5 | Emma | Espresso | eespresso@goodcorp.net | Research and Development | 2020-09-16 11:00:44 |
| 6 | Luke | Latte | llatte@goodcorp.net | Logistics | 2020-09-16 11:29:00 |
+---+-----+-----+-----+-----+-----+
6 rows in set (0.000 sec)

```

- Return to Firefox and refresh the page to see how it shows up on the employee directory page.

- Go back to Firefox and refresh the page to see the new employee listed at bottom.

8. **DELETE Queries:** In order to delete the entry we just created, we can use a `DELETE` query. We'll want to use this together with the `WHERE` clause to specify deleting only a single entry.

- Run `DELETE FROM employees WHERE email='llatte@goodcorp.net';` and note the Query OK output.
-  **Heads Up:** You never want to enter a query like `DELETE FROM employees;` as it will delete the entire `employees` table!
 - If attackers gain access to a database, they may use `DELETE` to maliciously wipe out thousands or even millions of records, resulting in significant impact to a business's operations.
- Let's return to using `SELECT` to verify that our deleted user is no longer listed in the database.
- `SELECT * FROM employees;` to show that we're back down to five employees:

```
MariaDB [goodcorpdb]> SELECT * FROM employees;
```

id	firstname	lastname	email	department	date_added
1	Bob	Brew	bbrew@goodcorp.net	Sales and Marketing	2020-09-16 11:00:44
2	Andrew	Americano	aamericano@goodcorp.net	Research and Development	2020-09-16 11:00:44
3	Caroline	Cortado	ccortado@goodcorp.net	Human Resources	2020-09-16 11:00:44
4	Deborah	Doppio	ddoppio@goodcorp.net	Operations	2020-09-16 11:00:44
5	Emma	Espresso	eespresso@goodcorp.net	Research and Development	2020-09-16 11:00:44

5 rows in set (0.000 sec)

- For visual confirmation via the employee directory, reload the webpage to see that the entry is now gone.