HackerFrogs Afterschool
Beginner Cybersecurity Skills

Introduction to SQL with SQL Murder Mystery

Class:	**Web App Exploitation**
Workshop Number:	**AS-WEB-04**
Document Version:	**1.2**
Special Requirements:	**None**

Table of Contents

Introduction

Welcome to HackerFrogs Afterschool! HackerFrogs Afterschool is a a series of online workshops meant to teach cybersecurity skills and concepts to beginners who may not be familiar with the field of cybersecurity. This lesson is the third lesson for web app exploitation, which is a very prominent and in-demand specialization in cybersecurity. This lesson will cover the following learning objectives:

- Learn about the Structured Query Language (SQL) used in modern database software
- Complete the SQL Murder Mystery game using SQL knowledge

Workshop Guide Format Notes

When following instructions in this document, there will be instructions to type specific commands in the CLI window or in a search bar. These commands will be displayed in red, and will have a different `font` applied to them, as well as a light grey background color. For example:

```
this is a sample command text line
```

However, if the input is meant to be put into the web browser or other non-CLI program, it will be colored red, and **bolded** but will not be in a different font nor will it have a light-grey highlight. For example:

This is a sample user input text line

If there are code snippets in this document, they look similar to the format above, except that the text will be black. For example:

```
this is a sample code snippet line
```

SQL Murder Mystery Information

There is no account registration required to play the SQL Murder Mystery game. Simply open a web browser and navigate to the following URL:

https://mystery.knightlab.com/walkthrough.html

When that page has finished loading, we're ready to begin.

Part 1 – Learning the Objectives of the Game

Read through the webpage contents under the **Walkthrough for SQL Beginners** header. Within the text, we learn that the goal of the game is to solve a murder case by using SQL skills. We are given the following information about the case:

- The type of crime: **murder**
- The date of the crime: **Jan.15, 2018**
- The place where the crime took place: **SQL City**

Part 2 – Learning SQL Command and Syntax

Before we can attempt to solve this crime, we first have to learn about the system we'll be using to find our information:

NOTE: **Part 2** is meant to be used as a reference to learning the different SQL commands used for this exercise. Feel free to work through the webpage examples and descriptions and refer to this document if further clarification is needed. **Part 3** begins after the section on sort order (ASC and DESC) is completed.

What is SQL?

SQL (Structured Query Language), is a programming language used for managing data held in relational databases. Simply put, SQL allows users to access and manage data contained in relational databases, which are common components in most web applications.

Why Learn SQL?

As far as web exploitation is concerned, insecure implementation of SQL in web applications can lead to a very serious vulnerability called SQL injection. We'll learn about SQL injection in another workshop, but we should first learn how to interact with SQL as intended.

What is an ERD?

An **ERD** (Entity Relationship Diagram) is a visual representation of relationships among all tables in a database. They are very helpful tools for understanding the layout of a database and how all of its tables relate to each other.

Look at the ERD on the webpage, the grey arrows show relationships between the different tables, as indicated by the grey arrows connecting the tables.

The SELECT and FROM Commands

The **SELECT** command in SQL is the way through which search queries are returned, and the **FROM** command specifies which table in the database to query. A typical SELECT command uses the following syntax:

```
SELECT column_name FROM table_name;
```

The * Wildcard Character

The * wildcard character in SQL is used to select all columns in a table when used in place of a column name.

The LIMIT Clause

The LIMIT clause is used to reduce the number of results to the number specified. For example:

```
LIMIT 1
LIMIT 5
```

The above commands will limit the number of rows returned by a SQL query. In this example it is 1 row and 5 rows of data, respectively.

The Count Function

The **count** function returns the number of entries in a particular table. The syntax is as follows:

```
SELECT count() FROM table_name;
```

The DISTINCT Filter

The **distinct** filter removes all duplicate entries from query results. The syntax is as follows:

```
SELECT DISTINCT column_name FROM table_name;
```

The WHERE Filter

The WHERE filter is the most commonly used filter in SQL queries, because it allows for results to be filtered out in a very precise way. The simplest use of the WHERE filter occurs when it is used to only match results that fit a specific string, for example:

```
SELECT column_name FROM table_name WHERE column_name = 'example';
```

The above example returns the column_name values from the table_name table, but only the ones where the value is equal to the text **example**.

The AND / OR Clauses

The AND / OR clauses are usually mutually exclusively used to filter results that either match two criteria (the AND clause), or filter results that match either of two (the OR clause). Let's look at the following example query featuring the AND clause:

```
SELECT name, age, hair_color FROM person WHERE age = 29 AND
hair_color = 'black';
```

This query returns the columns **name**, **age** and **hair_color** from the **person** table, but only if the value in the **age** column is **29** and the value in the **hair_color** column is **black**.

Next, we'll look at an example query using the OR clause:

```
SELECT name, age FROM person WHERE age = 29 OR age = 39;
```

This query returns the columns **name** and **age** from the **person** table, but only if the value in the **age** column is **29** or **39**.

The LIKE Operator and the % Wildcard Character

The LIKE operator paired with the **%** wildcard character can be used in SQL queries to return partial string or number matches. When used by itself, the LIKE operator and **%** wildcard character matches any string or number, but it can also be used to match partial strings / numbers. For example:

```
SELECT name FROM person WHERE name LIKE '%smith';
```

The above command will return all names from the **person** table where the value of the name ends with the **smith** string.

The UPPER and LOWER Functions

The UPPER and LOWER functions are used to render string values of columns in all uppercase characters or lowercase characters, respectively. This is useful when writing queries for strings where we don't know the capitalization used in the database. For example:

```
SELECT name FROM person WHERE lower(name) LIKE '%smith%';
```

This query will return all names from the person table that contain the string **smith**, whether the name is capitalized or not, and whether or not it is part of a longer name, such as Naysmith or Smithson.

The Max and Min Functions

The **max** and **min** functions in SQL are used with column names, and they return the largest or smallest values in the column, respectively. For example:

```
SELECT max(age) FROM drivers_license;
SELECT min(age) FROM drivers_license;
```

The Sum and Avg Function

The **sum** and **avg** functions return the total or average values from all entries in a column, respectively. For example:

```
SELECT sum(age) FROM drivers_license;
SELECT avg(age) FROM drivers_license;
```

The ORDER BY clause and ASC, DESC keywords

The ORDER BY clause returns a query in either alphabetical or numerical order, and the ASC and DESC keywords specify whether the order is ascending order or descending order. For example:

```
SELECT * FROM drivers_license ORDER BY age DESC LIMIT 5;
```

Part 3 – Obtaining the Crime Report

Now that we're familiar with basic SQL queries, let's get the crime report relevant to our murder case. If we recall, we know the **type** of crime (murder), the **date** of the crime (20180115), and the **city** in which it took place (**SQL City**). Let's combine these into a query to get exactly the crime scene report we want.

In the webpage, navigate to any of the SQL input fields and enter the following query:

```
SELECT * FROM crime_scene_report
WHERE type = 'murder'
AND city = 'SQL City'
AND date = 20180115;
```

What is returned in the results gives us information about 2 witnesses:

- The first witness lives at the last house on **Northwestern Dr**
- The second witness' name is **Annabel**, and lives on **Franklin Ave**

Part 4 – Questioning the Witnesses

We'll need to run another query on the database to obtain the full names of the witnesses before we can question them.

Step 1

In the **Write a query that identifies the first witness** field, submit the following query:

```
SELECT * FROM person
WHERE address_street_name = 'Northwestern Dr'
ORDER BY address_number DESC LIMIT 1;
```

In the results, we find that the witness' name is **Morty Schapiro**.

Step 2

In the **Write a query that identifies the second witness** field, submit the following query:

```
SELECT * FROM person
WHERE name LIKE 'Annabel%'
AND address_street_name = 'Franklin Ave';
```

In the results, we find that the witness' name is **Annabel Miller**. Note that we include the % wildcard symbol in the query after the name **Annabel** so that it matches with any potential last names,

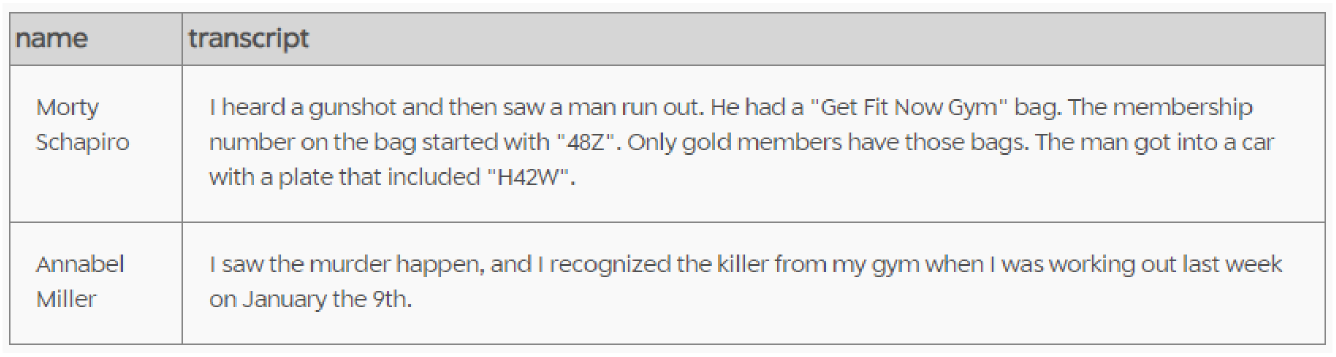
Step 3

Now that we have the names of the two witnesses, we will need to create a query with a **join** clause on both the **interview** and **person** tables. Note that when we use **join** clauses, we need to specify which columns we want data from, as well as which columns are used to join the tables according to the **<table name>.<column name>** syntax. For example, **person.name** specifies the **name** column from the **person** table, and if we join the **person** and **interview** tables, we would use the **on** clause with **interview.person_id = person.id**, since those are columns through which the two tables are connected.

In the **Write a query that shows the interview transcripts for our two subjects** box, use the following query:

```
SELECT person.name,interview.transcript
FROM person JOIN interview ON interview.person_id = person.id
WHERE name = 'Annabel Miller'
OR name = 'Morty Schapiro';
```

We should see the following in the output:



name	transcript
Morty Schapiro	I heard a gunshot and then saw a man run out. He had a "Get Fit Now Gym" bag. The membership number on the bag started with "48Z". Only gold members have those bags. The man got into a car with a plate that included "H42W".
Annabel Miller	I saw the murder happen, and I recognized the killer from my gym when I was working out last week on January the 9th.

From these two witness reports, we find the following clues:

- The suspect is **male**
- He had a **Get Fit Now Gym** bag
- The suspect's Get Fit Now Gym membership number starts with **48Z**.
- The suspect is a Get Fit Now Gym **gold** member.
- The suspect's license plate number contains the string **H42W**
- The suspect was at the gym on **January 9th**.

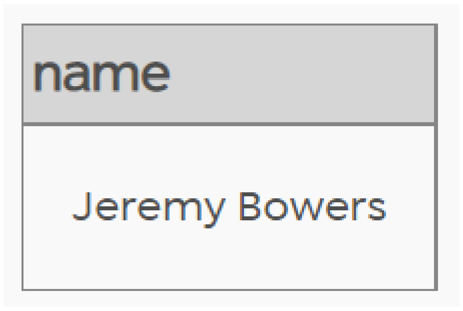
Part 5 – Deducing the Identity of the Murderer

Step 1

Now that we have a bunch of clues as to the murderer's identity, let's incorporate all those clues into a query and find out who they are. Seeing as we are incorporating data from the **get_fit_now_check_in** table (**check_in_date**, **id**), the **get_fit_now_member** table (**membership_status**), **person** table (**name**), and **drivers_license** tables (**plate_number**), we will have to use a lot of JOIN clauses, like the one shown below:

```
SELECT person.name FROM person
JOIN get_fit_now_member
ON person.id = get_fit_now_member.person_id
JOIN get_fit_now_check_in
ON get_fit_now_check_in.membership_id = get_fit_now_member.id
JOIN drivers_license
ON drivers_license.id = person.license_id
WHERE LOWER(get_fit_now_member.membership_status) = 'gold'
AND get_fit_now_check_in.check_in_date = 20180109
AND drivers_license.plate_number LIKE '%H42W%'
AND get_fit_now_member.id LIKE '48Z%';
```

This is what should return from that query:



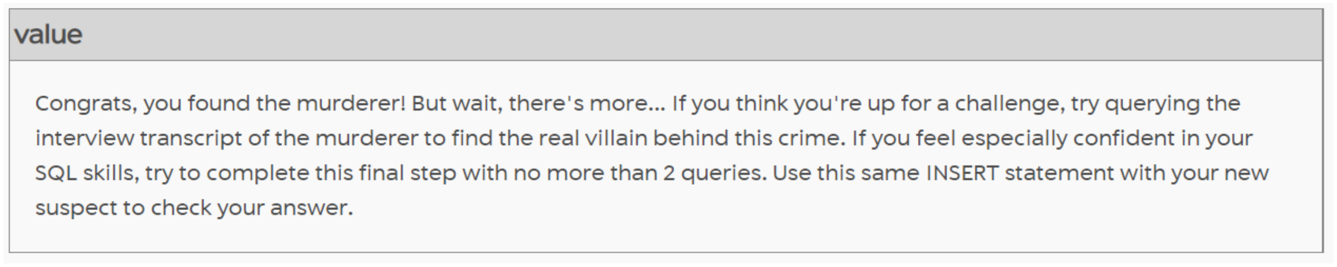
name
Jeremy Bowers

Step 2

Use the **Check your solution** box to verify we have the right person:

```
INSERT INTO solution VALUES (1, 'Jeremy Bowers'); SELECT value FROM
solution;
```

We should receive output that looks like this:



value
Congrats, you found the murderer! But wait, there's more... If you think you're up for a challenge, try querying the interview transcript of the murderer to find the real villain behind this crime. If you feel especially confident in your SQL skills, try to complete this final step with no more than 2 queries. Use this same INSERT statement with your new suspect to check your answer.

Looks like there's one final step to complete before we wrap this all up.

Part 6 - Identifying the Mastermind

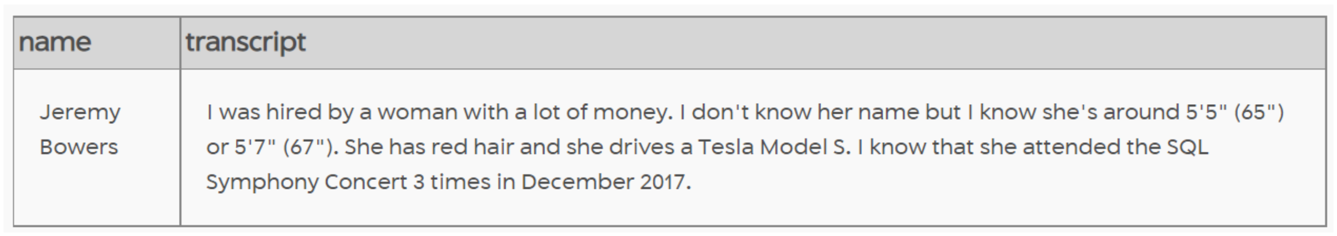
The murderer has been caught, but apparently there is one more person who needs to be brought to justice: the person who hired the murderer to do the job.

Step 1

We should use the following SQL query to obtain the interview statement from the murderer:

```
SELECT person.name,interview.transcript
FROM person JOIN interview
ON interview.person_id = person.id
WHERE name = 'Jeremy Bowers';
```

The data returned should look like this:



name	transcript
Jeremy Bowers	I was hired by a woman with a lot of money. I don't know her name but I know she's around 5'5" (65") or 5'7" (67"). She has red hair and she drives a Tesla Model S. I know that she attended the SQL Symphony Concert 3 times in December 2017.

The information here gives us a lot of clues as to what to search for, such as:

- A gender (**female**)
- A high **income**
- A height of **65** to **67** inches.
- **Red** hair
- A **Tesla Model S** car
- 3 instances of attending the **SQL Symphony Concert** in **December 2017**

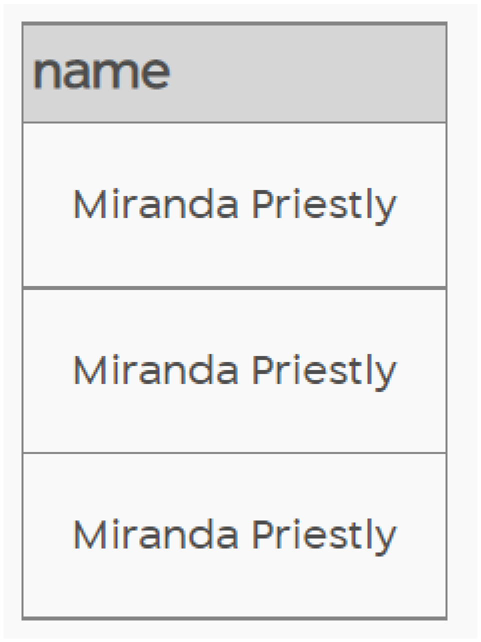
Step 2

For the following query, we'll need to use the JOIN clause across the following tables (columns): **facebook_event_checkin** (**event_name**, **date**, **person_id**), **person** (**id**, **license_id**, **ssn**), **drivers_license** (**id**, **car_make**, **car_model**, **gender**), and **income** (**ssn**, **annual_income**).

Using the following long-winded SQL query, we should identify who this mastermind is:

```
SELECT person.name FROM person
JOIN income
ON person.ssn = income.ssn
JOIN drivers_license
ON person.license_id = drivers_license.id
JOIN facebook_event_checkin
ON person.id = facebook_event_checkin.person_id
WHERE facebook_event_checkin.event_name = 'SQL Symphony Concert'
AND LOWER(drivers_license.hair_color) = 'red'
AND drivers_license.height BETWEEN 65 AND 67
AND LOWER(drivers_license.gender) = 'female'
AND drivers_license.car_make = 'Tesla'
AND drivers_license.car_model = 'Model S'
AND income.annual_income > 300000;
```

The results should look like this:



name
Miranda Priestly
Miranda Priestly
Miranda Priestly

NOTE: The name show up three times because the corresponding **person_id** appears three times in the **facebook_event_checkin** table appearing with the SQL Symphony Concert **event_name**.

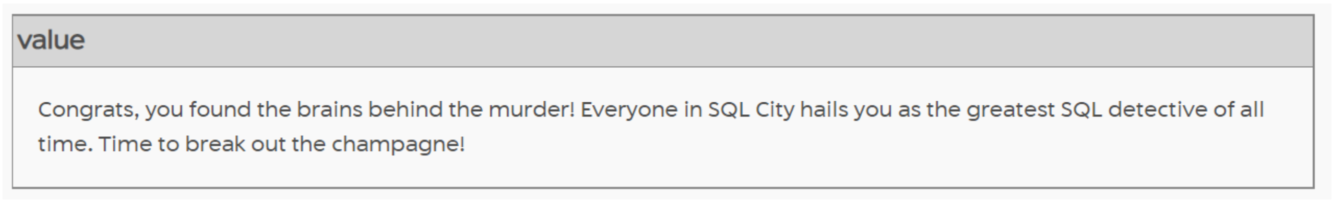
Now that we have the final name, we can input that into the **Check your solution** box:

Step 3

Now that we have the final name, we can input that into the **Check your solution** box:

```
INSERT INTO solution VALUES (1, 'Miranda Priestly'); SELECT value
FROM solution;
```

And with that, we see the final congratulations message:



value
Congrats, you found the brains behind the murder! Everyone in SQL City hails you as the greatest SQL detective of all time. Time to break out the champagne!

Summary

In today's workshop, we learned a lot about SQL and how to use SQL commands to return specific data entries from different tables in databases. We used these query skills in a game scenario to first identify witnesses to a crime, then used the clues the witnesses provided to identify the suspect of a crime. Finally, we used the suspect's testimony to identify the mastermind behind the crime.

In the next workshop, we'll be expanding on the SQL skills we learned today and learn about how SQL commands can be used in exploiting web apps using a technique called SQL injection.

Extra Credit

If you're looking for more SQL fun, there's another SQL learning game at the following URL:

SQL Island
https://sql-island.informatik.uni-kl.de/

NOTE: This game's default language is German. To access the English version of the game, click on the button beside the **SQL Island** text in the red banner, then select the **Sprache wechseln** option from the menu, then click on the **English** button.

Extra Research

The following articles will help us further understand some of the concepts covered in this workshop:

A Short Explanation of SQL:
https://www.sqlshack.com/sql-definition/

More articles on SQL (keep reading until the end of the Syntax page):
https://www.tutorialspoint.com/sql/index.htm

Until next time, HackerFrogs!