Query a database

Previously, you explored how SQL is an important tool in the world of cybersecurity and is essential when querying databases. You examined a few basic SQL queries and keywords used to extract needed information from a database. In this reading, you'll review those basic SQL queries and learn a new keyword that will help you organize your output. You'll also learn about the **Chinook** database, which this course uses for queries in readings and quizzes.

Basic SQL query

There are two essential keywords in any SQL query: **SELECT** and **FROM**. You will use these keywords every time you want to query a SQL database. Using them together helps SQL identify what data you need from a database and the table you are returning it from.

The video demonstrated this SQL query:

```
SELECT employee_id, device_id

FROM employees;
```

In readings and quizzes, this course uses a sample database called the **Chinook** database to run queries. The **Chinook** database includes data that might be created at a digital media company. A security analyst employed by this company might need to query this data. For example, the database contains eleven tables, including an **employees** table, a **customers** table, and an **invoices** table. These tables include data such as names and addresses.

As an example, you can run this query to return data from the **customers** table of the **Chinook** database:

```
SELECT customerid, city, country
FROM customers;
```

RunReset

```
+------------+---------------------+----------------+
| CustomerId | City                | Country        |
+------------+---------------------+----------------+
|          1 | São José dos Campos | Brazil         |
|          2 | Stuttgart           | Germany        |
|          3 | Montréal            | Canada         |
|          4 | Oslo                | Norway         |
|          5 | Prague              | Czech Republic |
|          6 | Prague              | Czech Republic |
|          7 | Vienne              | Austria        |
|          8 | Brussels            | Belgium        |
|          9 | Copenhagen          | Denmark        |
|         10 | São Paulo           | Brazil         |
|         11 | São Paulo           | Brazil         |
|         12 | Rio de Janeiro      | Brazil         |
|         13 | Brasília            | Brazil         |
|         14 | Edmonton            | Canada         |
```

```
|          15 | Vancouver           | Canada         |
|          16 | Mountain View       | USA            |
|          17 | Redmond             | USA            |
|          18 | New York            | USA            |
|          19 | Cupertino           | USA            |
|          20 | Mountain View       | USA            |
|          21 | Reno                | USA            |
|          22 | Orlando             | USA            |
|          23 | Boston              | USA            |
|          24 | Chicago             | USA            |
|          25 | Madison             | USA            |
+------------+---------------------+----------------+
(Output limit exceeded, 25 of 59 total rows shown)
```

SELECT

The **`SELECT`** keyword indicates which columns to return. For example, you can return the **`customerid`** column from the **`Chinook`** database with

```
SELECT customerid
```

You can also select multiple columns by separating them with a comma. For example, if you want to return both the **`customerid`** and **`city`** columns, you should write **`SELECT customerid, city`**.

If you want to return all columns in a table, you can follow the **`SELECT`** keyword with an asterisk (**`*`**). The first line in the query will be **`SELECT *`**.

Note: Although the tables you're querying in this course are relatively small, using **`SELECT *`** may not be advisable when working with large databases and tables; in those cases, the final output may be difficult to understand and might be slow to run.

FROM

The **`SELECT`** keyword always comes with the **`FROM`** keyword. **`FROM`** indicates which table to query. To use the **`FROM`** keyword, you should write it after the **`SELECT`** keyword, often on a new line, and follow it with the name of the table you're querying. If you want to return all columns from the **`customers`** table, you can write:

```
SELECT *

FROM customers;
```

When you want to end the query here, you put a semicolon (`;`) at the end to tell SQL that this is the entire query.

Note: Line breaks are not necessary in SQL queries, but are often used to make the query easier to understand. If you prefer, you can also write the previous query on one line as

```
SELECT * FROM customers;
```

ORDER BY

Database tables are often very complicated, and this is where other SQL keywords come in handy. **ORDER BY** is an important keyword for organizing the data you extract from a table.

ORDER BY sequences the records returned by a query based on a specified column or columns. This can be in either ascending or descending order.

Sorting in ascending order

To use the **ORDER BY** keyword, write it at the end of the query and specify a column to base the sort on. In this example, SQL will return the **customerid**, **city**, and **country** columns from the **customers** table, and the records will be sequenced by the **city** column:

```
SELECT customerid, city, country
FROM customers
ORDER BY city;
```

RunReset

```
+------------+--------------+----------------+
| CustomerId | City         | Country        |
+------------+--------------+----------------+
|         48 | Amsterdam    | Netherlands    |
|         59 | Bangalore    | India          |
|         36 | Berlin       | Germany        |
|         38 | Berlin       | Germany        |
|         42 | Bordeaux     | France         |
|         23 | Boston       | USA            |
|         13 | Brasília     | Brazil         |
|          8 | Brussels     | Belgium        |
|         45 | Budapest     | Hungary        |
|         56 | Buenos Aires | Argentina      |
|         24 | Chicago      | USA            |
|          9 | Copenhagen   | Denmark        |
|         19 | Cupertino    | USA            |
|         58 | Delhi        | India          |
|         43 | Dijon        | France         |
|         46 | Dublin       | Ireland        |
|         54 | Edinburgh    | United Kingdom |
|         14 | Edmonton     | Canada         |
|         26 | Fort Worth   | USA            |
|         37 | Frankfurt    | Germany        |
|         31 | Halifax      | Canada         |
|         44 | Helsinki     | Finland        |
|         34 | Lisbon       | Portugal       |
|         52 | London       | United Kingdom |
|         53 | London       | United Kingdom |
+------------+--------------+----------------+
(Output limit exceeded, 25 of 59 total rows shown)
```

The **ORDER BY** keyword sorts the records based on the column specified after this keyword. By default, as shown in this example, the sequence will be in ascending order. This means

- if you choose a column containing numeric data, it sorts the output from the smallest to largest. For example, if sorting on **`customerid`**, the ID numbers are sorted from smallest to largest.
- if the column contains alphabetic characters, such as in the example with the **`city`** column, it orders the records from the beginning of the alphabet to the end.

Sorting in descending order

You can also use the **`ORDER BY`** with the **`DESC`** keyword to sort in descending order. The **`DESC`** keyword is short for "descending" and tells SQL to sort numbers from largest to smallest, or alphabetically from Z to A. This can be done by following **`ORDER BY`** with the **`DESC`** keyword. For example, you can run this query to examine how the results differ when **`DESC`** is applied:

```
SELECT customerid, city, country
FROM customers
ORDER BY city DESC;
```

RunReset

```
+------------+---------------------+----------------+
| CustomerId | City                | Country        |
+------------+---------------------+----------------+
|         33 | Yellowknife         | Canada         |
|         32 | Winnipeg            | Canada         |
|         49 | Warsaw              | Poland         |
|          7 | Vienne              | Austria        |
|         15 | Vancouver           | Canada         |
|         27 | Tucson              | USA            |
|         29 | Toronto             | Canada         |
|         10 | São Paulo           | Brazil         |
|         11 | São Paulo           | Brazil         |
|          1 | São José dos Campos | Brazil         |
|          2 | Stuttgart           | Germany        |
|         51 | Stockholm           | Sweden         |
|         55 | Sidney              | Australia      |
|         57 | Santiago            | Chile          |
|         28 | Salt Lake City      | USA            |
|         47 | Rome                | Italy          |
|         12 | Rio de Janeiro      | Brazil         |
|         21 | Reno                | USA            |
|         17 | Redmond             | USA            |
|          5 | Prague              | Czech Republic |
|          6 | Prague              | Czech Republic |
|         35 | Porto               | Portugal       |
|         39 | Paris               | France         |
|         40 | Paris               | France         |
|         30 | Ottawa              | Canada         |
+------------+---------------------+----------------+
(Output limit exceeded, 25 of 59 total rows shown)
```

Now, cities at the end of the alphabet are listed first.

Sorting based on multiple columns

You can also choose multiple columns to order by. For example, you might first choose the **`country`** and then the **`city`** column. SQL then sorts the output by **`country`**, and for rows with the same **`country`**, it sorts them based on **`city`**. You can run this to explore how SQL displays this:

```
SELECT customerid, city, country
FROM customers
ORDER BY country, city;
```

RunReset

```
+------------+----------------------+----------------+
| CustomerId | City                 | Country        |
+------------+----------------------+----------------+
|         56 | Buenos Aires         | Argentina      |
|         55 | Sidney               | Australia      |
|          7 | Vienne               | Austria        |
|          8 | Brussels             | Belgium        |
|         13 | Brasília             | Brazil         |
|         12 | Rio de Janeiro       | Brazil         |
|          1 | São José dos Campos  | Brazil         |
|         10 | São Paulo            | Brazil         |
|         11 | São Paulo            | Brazil         |
|         14 | Edmonton             | Canada         |
|         31 | Halifax              | Canada         |
|          3 | Montréal             | Canada         |
|         30 | Ottawa               | Canada         |
|         29 | Toronto              | Canada         |
|         15 | Vancouver            | Canada         |
|         32 | Winnipeg             | Canada         |
|         33 | Yellowknife          | Canada         |
|         57 | Santiago             | Chile          |
|          5 | Prague               | Czech Republic |
|          6 | Prague               | Czech Republic |
|          9 | Copenhagen           | Denmark        |
|         44 | Helsinki             | Finland        |
|         42 | Bordeaux             | France         |
|         43 | Dijon                | France         |
|         41 | Lyon                 | France         |
+------------+----------------------+----------------+
(Output limit exceeded, 25 of 59 total rows shown)
```

Key takeaways

`SELECT` and **`FROM`** are important keywords in SQL queries. You use **`SELECT`** to indicate which columns to return and **`FROM`** to indicate which table to query. You can also include **`ORDER BY`** in your query to organize the output. These foundational SQL skills will support you as you move into more advanced queries.