Apply filters to SQL queries

Project description

The system needs to be more secure in my organization. In my job I must to confirm the system is safe, investigate all potential security issues, and provide all updates for employee computers as needed. These are the following steps of how I used SQL with filters to perform security-related tasks.

Retrieve after hours failed login attempts

A possible security event was detected outside of normal operating hours (after 6:00 PM). Because of this, it is necessary to examine all unsuccessful login attempts that took place during that timeframe, as they could indicate unauthorized access attempts.

To support this investigation, I developed a SQL query designed to isolate failed login events occurring specifically after business hours. The example below illustrates how the query was constructed to filter for those entries.

```
MariaDB [organization]> SELECT *
    -> FROM log_in_attempts
    -> WHERE login_time > '18:00' AND success = FALSE;
+----------+----------+------------+------------+---------+-----------------+---------+
| event_id | username | login_date | login_time | country | ip_address      | success |
+----------+----------+------------+------------+---------+-----------------+---------+
|        2 | apatel   | 2022-05-10 | 20:27:27   | CAN     | 192.168.205.12  |       0 |
|       18 | pwashing | 2022-05-11 | 19:28:50   | US      | 192.168.66.142  |       0 |
|       20 | tshah    | 2022-05-12 | 18:56:36   | MEXICO  | 192.168.109.50  |       0 |
```

The screenshot shows my SQL query and a sample of its output. The query identifies failed login attempts after 6:00 PM by selecting data from the **log_in_attempts** table and applying two conditions: `login_time > '18:00'` (after-hours attempts) and `success = FALSE` (failed logins).

Retrieve login attempts on specific dates

On **2022-05-09**, a suspicious event was detected. To investigate, all login activity from that date as well as the previous day must be reviewed.

The SQL query below shows how I filtered the dataset to return login attempts occurring on the specified dates.

```
MariaDB [organization]> SELECT *
    -> FROM log_in_attempts
    -> WHERE login_date = '2022-05-09' OR login_date = '2022-05-08';
+----------+----------+------------+------------+---------+----------------+---------+
| event_id | username | login_date | login_time | country | ip_address     | success |
+----------+----------+------------+------------+---------+----------------+---------+
|        1 | jrafael  | 2022-05-09 | 04:56:27   | CAN     | 192.168.243.140 |       0 |
|        3 | dkot     | 2022-05-09 | 06:47:41   | USA     | 192.168.151.162 |       0 |
|        4 | dkot     | 2022-05-08 | 02:00:39   | USA     | 192.168.178.71  |       0 |
```

The screenshot is divided into two sections: the top shows the SQL query I constructed, while the bottom displays a sample of the query results. This query retrieves all login attempts that occurred on **2022-05-09** and **2022-05-08**.

To build the query, I first selected all records from the **log_in_attempts** table. I then applied a `WHERE` clause using the `OR` operator to filter the results, returning only login attempts from either of the two dates. Specifically, the conditions `login_date = '2022-05-09'` and `login_date = '2022-05-08'` isolate logins from May 9th and May 8th, respectively.

Retrieve login attempts outside of Mexico

Upon reviewing the organization's login data, I identified potentially problematic attempts originating from locations outside of Mexico. These login attempts warrant further investigation.

The SQL query below illustrates how I filtered the data to isolate login attempts occurring outside of Mexico.

```
MariaDB [organization]> SELECT *
    -> FROM log_in_attempts
    -> WHERE NOT country LIKE 'MEX%';
+----------+----------+------------+------------+---------+----------------+---------+
| event_id | username | login_date | login_time | country | ip_address     | success |
+----------+----------+------------+------------+---------+----------------+---------+
|        1 | jrafael  | 2022-05-09 | 04:56:27   | CAN     | 192.168.243.140 |       0 |
|        2 | apatel   | 2022-05-10 | 20:27:27   | CAN     | 192.168.205.12  |       0 |
|        3 | dkot     | 2022-05-09 | 06:47:41   | USA     | 192.168.151.162 |       0 |
```

The screenshot is divided into two parts: the top displays my SQL query, and the bottom shows a sample of the resulting output. This query identifies all login attempts originating from countries other than Mexico.

To construct the query, I first selected all records from the **log_in_attempts** table. I then applied a `WHERE` clause with the `NOT` operator to exclude Mexico. Since the dataset represents Mexico as both `MEX` and `MEXICO`, I used the `LIKE 'MEX%'` pattern, where the `%` wildcard matches any number of additional characters, ensuring both representations are captured.

Retrieve employees in Marketing

My team needs to update computers for specific employees in the Marketing department. To do this, I first needed to identify which employee machines require updates.

The SQL query below demonstrates how I filtered the dataset to retrieve machines belonging to Marketing employees located in the East building.

```
MariaDB [organization]> SELECT *
    -> FROM employees
    -> WHERE department = 'Marketing' AND office LIKE 'East%';
+-------------+-------------+----------+------------+----------+
| employee_id | device_id   | username | department | office   |
+-------------+-------------+----------+------------+----------+
|        1000 | a320b137c219 | elarson  | Marketing  | East-170 |
|        1052 | a192b174c940 | jdarosa  | Marketing  | East-195 |
|        1075 | x573y883z772 | fbautist | Marketing  | East-267 |
```

The screenshot is divided into two sections: the top shows the SQL query I wrote, and the bottom presents a sample of the output. This query is designed to return all employees who belong to the Marketing department and are located in the East building.

To create it, I began by selecting all records from the **employees** table. I then applied a `WHERE` clause with the `AND` operator to narrow the results to only those meeting both criteria. Specifically:

- The condition `department = 'Marketing'` isolates employees in the Marketing department.

- The condition `office LIKE 'East%'` identifies employees assigned to the East building, since the office column lists “East” along with a specific office number.

By combining these conditions, the query successfully extracts the set of Marketing employees working in the East building.

Retrieve employees in Finance or Sales

The computers used by employees in the Finance and Sales departments also require updates. Because these groups need a different security patch, I first needed to identify which machines belong specifically to employees in those two departments.

The SQL query below shows how I filtered the data to return only employee machines from the Finance and Sales departments.

```
MariaDB [organization]> SELECT *
    -> FROM employees
    -> WHERE department = 'Finance' OR department = 'Sales';
+-------------+--------------+----------+------------+-------------+
| employee_id | device_id    | username | department | office      |
+-------------+--------------+----------+------------+-------------+
|        1003 | d394e816f943 | sgilmore | Finance    | South-153   |
|        1007 | h174i497j413 | wjaffrey | Finance    | North-406   |
|        1008 | i858j583k571 | abernard | Finance    | South-170   |
```

The screenshot consists of two sections: the top displays the SQL query I created, while the bottom shows part of the resulting output. This query is designed to retrieve all employees who belong to either the Finance or Sales departments.

To build it, I began by selecting all records from the **employees** table. I then applied a `WHERE` clause using the `OR` operator, which allowed me to include employees from both departments. The reason for using `OR` instead of `AND` is that an employee can only belong to one department, so I needed results from either group rather than requiring both conditions at the same time.

In this query, the condition `department = 'Finance'` filters for Finance employees, while the condition `department = 'Sales'` filters for Sales employees. When combined with the `OR` operator, the query successfully returns all employees from both departments.

Retrieve all employees not in IT

My team must perform one more security update, this time for employees outside of the Information Technology department. To prepare for the update, I first needed to gather details on those employees.

The SQL query below shows how I filtered the dataset to return machines belonging to employees who are not part of the Information Technology department.

```
MariaDB [organization]> SELECT *
    -> FROM employees
    -> WHERE NOT department = 'Information Technology';
+-------------+--------------+----------+-----------------+-------------+
| employee_id | device_id    | username | department      | office      |
+-------------+--------------+----------+-----------------+-------------+
|        1000 | a320b137c219 | elarson  | Marketing       | East-170    |
|        1001 | b239c825d303 | bmoreno  | Marketing       | Central-276 |
|        1002 | c116d593e558 | tshah    | Human Resources | North-434   |
```

The screenshot is split into two sections: the top shows my SQL query, and the bottom displays part of the output. This query retrieves all employees who are not part of the Information Technology department. To create it, I selected all records from the **employees** table and applied a `WHERE` clause with the `NOT` operator to exclude entries from that department.

Summary

Throughout these exercises, I used SQL queries to extract targeted information about login attempts and employee machines. The queries drew from two tables: **log_in_attempts** and **employees**. To refine the results, I applied logical operators such as `AND`, `OR`, and `NOT`, depending on the scenario. I also made use of the `LIKE` operator along with the `%` wildcard to capture pattern-based matches when needed.