Apply filters to SQL queries

Project description

In this project, I analyzed security-related data using SQL to better understand login activity and support security investigations. I used logical operators like **AND, OR, and NOT** to filter login and employee records based on time, date, location, and department.

Through this analysis, I identified failed login attempts after business hours, reviewed activity around a specific security event, flagged logins from unexpected locations, and retrieved employee information for targeted security actions. This project reflects how cybersecurity analysts use SQL in real-world scenarios to spot suspicious behavior, investigate incidents, and make informed security decisions.

Retrieve after hours failed login attempts

Scenario

My team was investigating **failed login attempts that occurred after business hours**. Since office hours end at **18:00**, any unsuccessful login attempts after this time could indicate **unauthorized access attempts** or malicious activity.

Approach

I queried the log_in_attempts table and applied multiple conditions to narrow down the results:

- Filtered login attempts that occurred **after 18:00**
- Retrieved only **unsuccessful login attempts**
- Combined both conditions using the AND operator

SQL Query Used

```
SELECT *
FROM log_in_attempts
WHERE login_time > '18:00' AND success = 0;
```

```
MariaDB [organization]> SELECT *
    -> FROM log_in_attempts
    -> WHERE login_time > '18:00' AND success = 0;
+----------+----------+------------+------------+---------+----------------+---------+
| 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 |
|       28 | aestrada | 2022-05-09 | 19:28:12   | MEXICO  | 192.168.27.57  |       0 |
|       34 | drosas   | 2022-05-11 | 21:02:04   | US      | 192.168.45.93  |       0 |
|       42 | cgriffin | 2022-05-09 | 23:04:05   | US      | 192.168.4.157  |       0 |
|       52 | cjackson | 2022-05-10 | 22:07:07   | CAN     | 192.168.58.57  |       0 |
```

Findings

The query returned **failed login attempts occurring outside normal business hours**. These results highlight authentication events that may require further investigation, such as brute-force attempts or unauthorized access from internal or external sources.

Security Relevance

Failed login attempts after business hours are often treated as **high-priority alerts** in security operations. Identifying these events allows security teams to:

- Detect suspicious behavior early
- Correlate activity with IP addresses or locations
- Take proactive response actions if needed

Retrieve login attempts on specific dates

Scenario

A **suspicious security event** was reported on **May 9, 2022**. To better understand the situation, I analyzed login activity from that day and the **previous day (May 8, 2022)** to identify patterns or anomalies.

Approach

I filtered login attempt records by date using the 0R operator to:

- Retrieve login attempts from **multiple dates in a single query**
- Compare authentication activity across consecutive days

SQL Query Used

```
SELECT *
FROM log_in_attempts
WHERE login_date = '2022-05-09' OR login_date = '2022-05-08';
```

```
200 rows in set, 5 warnings (0.002 sec)

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 |       1 |
|        3 | dkot     | 2022-05-09 | 06:47:41   | USA     | 192.168.151.162 |       1 |
|        4 | dkot     | 2022-05-08 | 02:00:39   | USA     | 192.168.178.71  |       0 |
|        8 | bisles   | 2022-05-08 | 01:30:17   | US      | 192.168.119.173 |       0 |
|       12 | dkot     | 2022-05-08 | 09:11:34   | USA     | 192.168.100.158 |       1 |
|       15 | lyamamot | 2022-05-09 | 17:17:26   | USA     | 192.168.183.51  |       0 |
|       24 | arusso   | 2022-05-09 | 06:49:39   | MEXICO  | 192.168.171.192 |       1 |
|       25 | sbaelish | 2022-05-09 | 07:04:02   | US      | 192.168.33.137  |       1 |
|       26 | apatel   | 2022-05-08 | 17:27:00   | CANADA  | 192.168.123.105 |       1 |
|       28 | aestrada | 2022-05-09 | 19:28:12   | MEXICO  | 192.168.27.57   |       0 |
```

Findings

The query returned **75 login attempts** across the two dates. Reviewing authentication activity across adjacent days provides valuable context when investigating security incidents.

Security Relevance

Analyzing login attempts over multiple related dates helps security teams:

- Identify repeated or escalating access attempts
- Detect abnormal login behavior
- Build an accurate incident timeline

Retrieve login attempts outside of Mexico

Scenario

As part of the investigation, I needed to identify **login attempts that did not originate in Mexico.** Since the dataset recorded country values in different formats (such as `MEX` and `MEXICO`), it was important to account for both while filtering the data.

Approach

I queried the `log_in_attempts` table and used the **NOT** and **LIKE** operators together. By applying the pattern `MEX%`, I was able to exclude all login attempts originating from Mexico, regardless of how the country name was stored.

SQL Query Used

```
SELECT *
FROM log_in_attempts
WHERE country NOT LIKE 'MEX%';
```

```
MariaDB [organization]> SELECT *
    -> FROM log_in_attempts
    -> WHERE country NOT 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 |       1 |
|        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 |       1 |
|        4 | dkot     | 2022-05-08 | 02:00:39   | USA     | 192.168.178.71  |       0 |
|        5 | jrafael  | 2022-05-11 | 03:05:59   | CANADA  | 192.168.86.232  |       0 |
|        7 | eraab    | 2022-05-11 | 01:45:14   | CAN     | 192.168.170.243 |       1 |
|        8 | bisles   | 2022-05-08 | 01:30:17   | US      | 192.168.119.173 |       0 |
|       10 | jrafael  | 2022-05-12 | 09:33:19   | CANADA  | 192.168.228.221 |       0 |
|       11 | sgilmore | 2022-05-11 | 10:16:29   | CANADA  | 192.168.140.81  |       0 |
```

Findings

This query returned **login attempts originating outside of Mexico**, helping narrow down potentially unusual or unexpected access activity.

Security Relevance

Login attempts from unexpected geographic locations can be a sign of **compromised credentials or unauthorized access**. Filtering out expected locations allows security teams to focus on higher-risk authentication events and prioritize further investigation.

Retrieve employees in Marketing

Scenario

As part of ongoing system updates, I needed to identify **employees in the Marketing department who are located in the East building**. This information helps ensure that updates and security actions are applied only to the relevant employee machines.

Approach

I queried the `employees` table and filtered the data using two conditions:

- Selected only employees from the **Marketing** department
- Used a **LIKE** pattern to match offices located in the **East** building

The **AND** operator was used to make sure both conditions were met at the same time.

SQL Query Used

```
SELECT *
FROM employees
WHERE department = 'Marketing' AND office LIKE 'East%';
```

```
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 |
|        1088 | k8651965m233 | rgosh    | Marketing  | East-157 |
|        1103 | NULL         | randerss | Marketing  | East-460 |
|        1156 | a184b775c707 | dellery  | Marketing  | East-417 |
|        1163 | h679i515j339 | cwilliam | Marketing  | East-216 |
+-------------+-------------+----------+------------+----------+
7 rows in set (0.001 sec)
```

Security Relevance

Filtering employee data by department and location allows security teams to:

- Apply updates in a **targeted and efficient** manner
- Reduce the risk of misconfigurations
- Minimize disruption to unrelated teams

Retrieve employees in Finance or Sales

Scenario

For the next update, the focus shifted to employees in the **Finance** and **Sales** departments. I needed to retrieve information for all employees in either of these departments so the appropriate systems could be updated.

Approach

I queried the `employees` table and used the **OR** operator to include employees from **both departments** in a single query. Since both conditions were based on the same column, I explicitly specified the department in each condition.

SQL Query Used

```
SELECT *
FROM employees
WHERE department = 'Finance' OR department = 'Sales';
```

```
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   |
|        1009 | NULL         | lrodriqu | Sales      | South-134   |
|        1010 | k242l212m542 | jlansky  | Finance    | South-109   |
|        1011 | l748m120n401 | drosas   | Sales      | South-292   |
|        1015 | p611q262r945 | jsoto    | Finance    | North-271   |
|        1017 | r550s824t230 | jclark   | Finance    | North-188   |
|        1018 | s310t540u653 | abellmas | Finance    | North-403   |
```

Findings

The query returned a list of employees working in the **Finance and Sales departments**, along with their associated device and office information. This made it easy to identify all systems affected by the update.

Security Relevance

Grouping multiple departments in one query helps security teams:

- Roll out updates **efficiently**
- Ensure no affected systems are missed
- Maintain consistent security controls across related teams

Retrieve all employees not in IT

Scenario

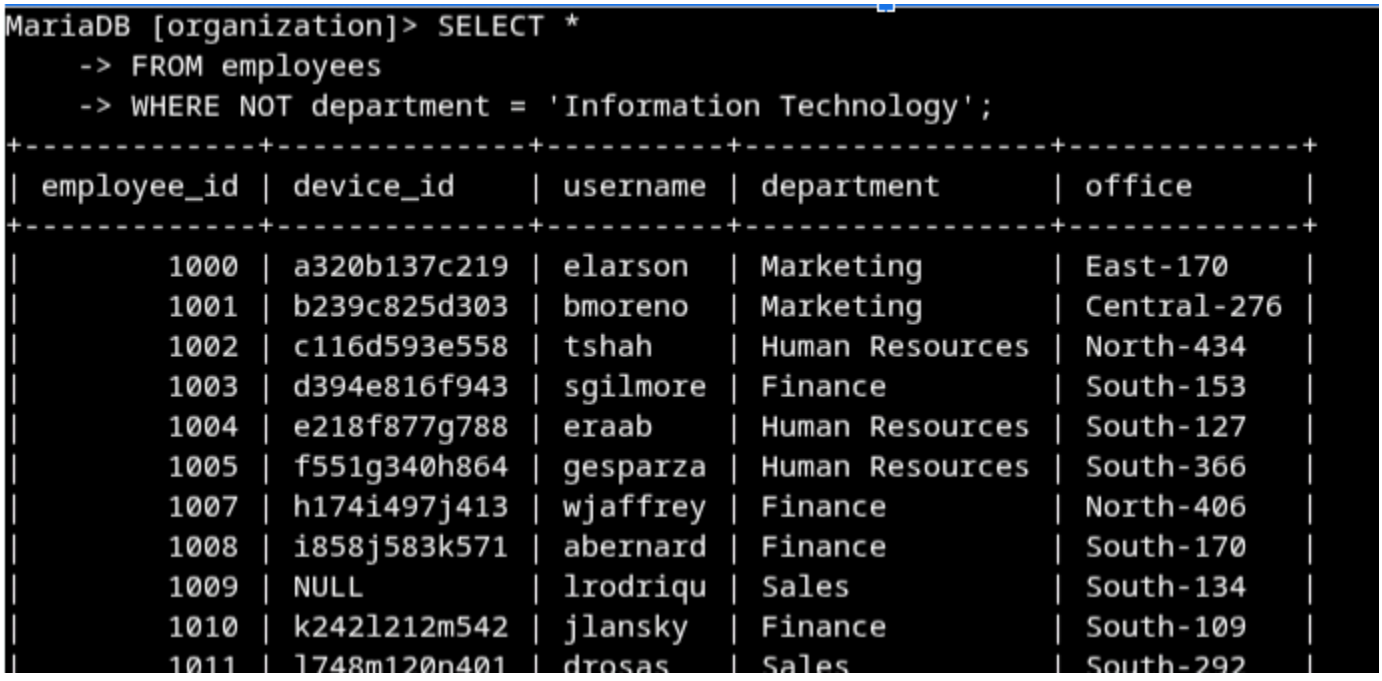
For the final update, employee computers in the **Information Technology** department had already been updated. I needed to identify **all remaining employees who were not part of IT** so the update could be applied to the rest of the organization.

Approach

I queried the `employees` table and used the **NOT** operator to exclude employees in the **Information Technology** department. This allowed me to focus only on users whose systems still required updates.

SQL Query Used

```
SELECT *
FROM employees
WHERE NOT department = 'Information Technology';
```



```
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   |
|        1003 | d394e816f943 | sgilmore | Finance         | South-153   |
|        1004 | e218f877g788 | eraab    | Human Resources | South-127   |
|        1005 | f551g340h864 | gesparza | Human Resources | South-366   |
|        1007 | h174i497j413 | wjaffrey | Finance         | North-406   |
|        1008 | i858j583k571 | abernard | Finance         | South-170   |
|        1009 | NULL         | lrodriqu | Sales           | South-134   |
|        1010 | k242l212m542 | jlansky  | Finance         | South-109   |
|        1011 | l748m120n401 | drosas   | Sales           | South-292   |
```

Findings

The query returned **161 employees** who are not part of the IT department. This confirmed which systems still needed attention and ensured no unnecessary updates were applied to IT-managed machines.

Security Relevance

Excluding specific departments helps security teams:

- Avoid **redundant or conflicting updates**
- Reduce operational risk
- Apply changes in a **controlled and efficient** manner

Summary

This project demonstrates my ability to use SQL to investigate security events and manage access-related data in a practical, real-world context. By applying logical operators to filter large datasets, I was able to isolate relevant login activity and employee information that required security attention.

The analysis focused on identifying abnormal login behavior, understanding activity around a reported security event, and supporting targeted system updates across different departments. Overall, this project highlights how structured SQL queries help cybersecurity teams quickly identify risks, prioritize responses, and maintain secure operations.