# Apply filters to SQL queries

## Project description

Project Title: Secure Network Infrastructure Enhancement

Objective: Strengthen the organization's network security by implementing comprehensive enhancements to the existing infrastructure, mitigating vulnerabilities, and fortifying defenses against potential threats.

Key Responsibilities and Achievements:

* Conducted a thorough security audit of the network infrastructure, identifying and prioritizing vulnerabilities using tools like Nessus and OpenVAS.
* Collaborated with cross-functional teams to design and implement a multi-layered defense strategy, including firewall configuration, intrusion detection systems (IDS), and endpoint security solutions.
* Led the deployment of a robust VPN (Virtual Private Network) solution to ensure secure remote access for employees, implementing strong encryption protocols and access controls.
* Established and maintained security best practices documentation, facilitating knowledge sharing and ensuring compliance with industry standards (e.g., NIST, ISO 27001).
* Conducted regular penetration testing and vulnerability assessments, providing timely reports and recommendations to senior management for risk mitigation.
* Orchestrated security awareness training sessions for staff members to promote a culture of cybersecurity vigilance and incident response readiness.

Technologies and Tools Used: Nessus, OpenVAS, Firewalls (e.g., Cisco ASA), IDS/IPS systems, VPN solutions, Security Information and Event Management (SIEM) tools, NIST and ISO security frameworks.

Outcome: Strengthened the organization's security posture by significantly reducing vulnerabilities, enhancing incident response capabilities, and fostering a culture of proactive cybersecurity awareness among employees.

## Retrieve after hours failed login attempts

To retrieve failed login attempts that occurred after hours from a log\_in\_attempts table (assuming it contains a timestamp column named login\_time and a column indicating success or failure, such as login\_status), you can use SQL to filter the data based on the time of login.

Let's assume "after hours" refers to login attempts after 6:00 PM:

SELECT \*

FROM log\_in\_attempts

WHERE login\_status = 'failed'

AND EXTRACT(HOUR FROM login\_time) >= 18;

This query retrieves all failed login attempts where the login\_status is 'failed' and the login\_time occurred after 6:00 PM. Adjust the time criteria (>= 18 for 6:00 PM onwards) according to your specific definition of "after hours."

## Retrieve login attempts on specific dates

To retrieve login attempts on specific dates from a `log\_in\_attempts` table (assuming it contains a `login\_time` timestamp column and possibly a `login\_status` column indicating success or failure), you can use SQL to filter the data based on the date.

Let's say you want to retrieve login attempts for a specific date, such as '2023-11-15':

```sql

SELECT \*

FROM log\_in\_attempts

WHERE DATE(login\_time) = '2023-11-15';

```

This query retrieves all login attempts where the `login\_time` falls on the specific date '2023-11-15'. Adjust the date in the query to fetch login attempts for different dates as needed.

Note: Some databases may handle date comparisons differently or have specific functions to extract dates, so the exact syntax might vary slightly depending on the database system you're using. This example uses the `DATE()` function, which extracts the date part from the `login\_time` timestamp.

## Retrieve login attempts outside of Mexico

To retrieve login attempts that originate from locations outside of Mexico from a `log\_in\_attempts` table (assuming it contains a `country` or `location` column indicating the country or location of login attempts), you can use SQL to filter the data based on the country.

Assuming 'Mexico' is the country you want to exclude:

```sql

SELECT \*

FROM log\_in\_attempts

WHERE country <> 'Mexico';

```

This query retrieves all login attempts where the `country` is not equal to 'Mexico'. Adjust the column name (`country`) and the condition (`<> 'Mexico'`) based on the column name and the specific value representing Mexico in your dataset.

If the `country` column is structured differently (e.g., using country codes instead of names), modify the query accordingly to match the actual data in your `log\_in\_attempts` table.

## Retrieve employees in Marketing

To retrieve employees who work in the Marketing department from an `employees` table (assuming it contains a `department` column indicating the department where employees work), you can use SQL to filter the data based on the department name.

Assuming 'Marketing' is the department name:

```sql

SELECT \*

FROM employees

WHERE department = 'Marketing';

```

This query retrieves all employees from the `employees` table where the `department` column is equal to 'Marketing'. Adjust the column name (`department`) and the condition (`= 'Marketing'`) based on the column name and the specific department name used in your dataset.

## Retrieve employees in Finance or Sales

To retrieve employees who work in either the Finance or Sales department from an `employees` table (assuming it contains a `department` column indicating the department where employees work), you can use SQL with the `IN` operator to filter the data based on multiple department names.

Here's an example query:

```sql

SELECT \*

FROM employees

WHERE department IN ('Finance', 'Sales');

```

This query retrieves all employees from the `employees` table where the `department` column matches either 'Finance' or 'Sales'. Adjust the column name (`department`) and the department names listed within the `IN` clause according to your specific dataset structure and department names.

## Retrieve all employees not in IT

To retrieve employees who do not work in the IT department from an `employees` table (assuming it contains a `department` column indicating the department where employees work), you can use SQL with the `NOT IN` operator to filter the data based on the department name.

Here's an example query:

```sql

SELECT \*

FROM employees

WHERE department NOT IN ('IT');

```

This query retrieves all employees from the `employees` table where the `department` column does not match 'IT'. Adjust the column name (`department`) and the department name listed within the `NOT IN` clause according to your specific dataset structure and department names.

## Summary

Certainly! Filters in SQL queries are used to narrow down the results returned from a database by specifying conditions that the data must meet. Here's a summary of some common filter types used in SQL queries:

1. \*\*WHERE Clause:\*\* The `WHERE` clause is fundamental for filtering data. It allows you to specify conditions to retrieve rows that meet specific criteria. For instance:

```sql

SELECT \* FROM table\_name WHERE column\_name = value;

```

2. \*\*Comparison Operators:\*\* SQL uses various comparison operators like `=`, `!=` (or `<>`), `<`, `>`, `<=`, `>=` to compare values in the `WHERE` clause.

```sql

SELECT \* FROM table\_name WHERE column\_name > value;

```

3. \*\*Logical Operators:\*\* SQL employs logical operators such as `AND`, `OR`, and `NOT` to combine conditions in the `WHERE` clause for more complex filtering.

```sql

SELECT \* FROM table\_name WHERE column1 = value1 AND column2 = value2;

```

4. \*\*IN Operator:\*\* The `IN` operator allows specifying multiple values for a column to filter rows that match any of the provided values.

```sql

SELECT \* FROM table\_name WHERE column\_name IN ('value1', 'value2', 'value3');

```

5. \*\*NOT Operator:\*\* The `NOT` operator negates a condition, filtering rows that do not meet the specified criteria.

```sql

SELECT \* FROM table\_name WHERE NOT column\_name = value;

```

6. \*\*BETWEEN Operator:\*\* The `BETWEEN` operator filters rows based on a range of values.

```sql

SELECT \* FROM table\_name WHERE column\_name BETWEEN value1 AND value2;

```

7. \*\*LIKE Operator:\*\* The `LIKE` operator is used for pattern matching using wildcard characters `%` (matches zero or more characters) and `\_` (matches a single character).

```sql

SELECT \* FROM table\_name WHERE column\_name LIKE 'pattern';

```

These filters allow SQL queries to extract precise subsets of data from large databases, making it easier to retrieve the necessary information based on specific criteria. Understanding and using these filters effectively are key skills for working with databases and extracting valuable insights from data.