Reference guide: SQL

Google Cybersecurity Certificate

Table of contents

Query a database

The `SELECT`, `FROM`, and `ORDER BY` keywords are used when retrieving information from a database.

`FROM`

Indicates which table to query; required to perform a query

> `FROM employees`
> Indicates to query the `employees` table

`ORDER BY`

Sequences the records returned by a query based on a specified column or columns

> `ORDER BY department`
> Sorts the records in ascending order by the `department` column; `ORDER BY department ASC` also sorts the records in ascending order by the `department` column
>
> `ORDER BY city DESC`
> Sorts the records in descending order by the `city` column

`ORDER BY country, city`
Sorts the records in ascending order by multiple columns; first sorts the output by `country`, and for records with the same `country`, sorts them based on `city`

SELECT

Indicates which columns to return; required to perform a query

`SELECT employee_id`
Returns the `employee_id` column

`SELECT *`
Returns all columns in a table

Apply filters to SQL queries

`WHERE` and the other SQL keywords and characters that follow are used when applying filters to SQL queries.

AND

Specifies that both conditions must be met simultaneously in a filter that contains two conditions

`WHERE region = 5 AND country = 'USA'`
Returns all records with a value in the `region` column of `5` and a value in the `country` column of `'USA'`

BETWEEN

Filters for numbers or dates within a range; `BETWEEN` is followed by the first value to include in the range, the `AND` operator, and the last value to include in the range

`WHERE hiredate BETWEEN '2002-01-01' AND '2003-01-01'`
Returns all records with a value in the `hiredate` column that is between `'2002-01-01'` and `'2003-01-01'`

= (equal to)

Used in filters to return only the records that contain a value in a specified column that is equal to a particular value

> `WHERE birthdate = '1980-05-15'`
> Returns all records with a value in the `birthdate` column that equals `'1980-05-15'`

> (greater than)

Used in filters to return only the records that contain a value in a specified column that is greater than a particular value

> `WHERE birthdate > '1970-01-01'`
> Returns all records with a value in the `birthdate` column that is greater than `'1970-01-01'`

>= (greater than or equal to)

Used in filters to return only the records that contain a value in a specified column that is greater than or equal to a particular value

> `WHERE birthdate >= '1965-06-30'`
> Returns all records with a value in the `birthdate` column that is greater than or equal to `'1965-06-30'`

< (less than)

Used in filters to return only the records that contain a value in a specified column that is less than a particular value

> `WHERE date < '2023-01-31'`
> Returns all records with a value in the `date` column that is less than `'2023-01-31'`

<= (less than or equal to)

Used in filters to return only the records that contain a value in a specified column that is less than or equal to a particular value

> `WHERE date <= '2020-12-31'`
> Returns all records with a value in the `date` column that is less than or equal to `'2020-12-31'`

`LIKE`

Used with `WHERE` to search for a pattern in a column

> `WHERE title LIKE 'IT%'`
> Returns all records with a value in the `title` column that matches the pattern of `'IT%'`
>
> `WHERE state LIKE 'N_'`
> Returns all records with a value in the `state` column that matches the pattern of `'N_'`

`NOT`

Negates a condition

> `WHERE NOT country = 'Mexico'`
> Returns all records with a value in the `country` column that is not `'Mexico'`

<> (not equal to)

Used in filters to return only the records that contain a value in a specified column that is not equal to a particular value; `!=` also used as an operator for not equal to

> `WHERE date <> '2023-02-28'`
> Returns all records with a value in the `date` column that is not equal to `'2023-02-28'`

`!=` (not equal to)

Used in filters to return only the records that contain a value in a specified column that is not equal to a particular value; `<>` also used as an operator for not equal to

> `WHERE date != '2023-05-14'`
> Returns all records with a value in the `date` column that is not equal to `'2023-05-14'`

OR

Specifies that either condition can be met in a filter that contains two conditions

> `WHERE country = 'Canada' OR country = 'USA'`
> Returns all records with a value in the `country` column of either `'Canada'` or `'USA'`

`%` (percentage sign)

Substitutes for any number of other characters; used as a wildcard in a pattern that follows `LIKE`

> `'a%'`
> Represents a pattern consisting of the letter `'a'` followed by zero or more characters
>
> `'%a'`
> Represents a pattern consisting of zero or more characters followed by the letter `'a'`
>
> `'%a%'`
> Represents a pattern consisting of the letter `'a'` surrounded by zero or more characters on each side

`_` (underscore)

Substitutes for one other character; used as a wildcard in a pattern that follows `LIKE`

`'a_'`
Represents a pattern consisting of the letter `'a'` followed by one character

`'a__'`
Represents a pattern consisting of the letter `'a'` followed by two characters

`'_a'`
Represents a pattern consisting of one character followed by the letter `'a'`

`'_a_'`
Represents a pattern consisting of the letter `'a'` surrounded by one character on each side

WHERE

Indicates the condition for a filter; must be used to begin a filter

```
WHERE title = 'IT Staff'
```
Returns all records that contain `'IT Staff'` in the `title` column; `WHERE` is placed before the condition of `title = 'IT Staff'` to create the filter

Join tables

The following SQL keywords are used to join tables.

FULL OUTER JOIN

Returns all records from both tables; the column used to join the tables is specified following `FULL OUTER JOIN` with syntax that includes `ON` and equal to (=)

```
SELECT *
FROM employees
FULL OUTER JOIN machines ON employees.device_id =
machines.device_id;
```
Returns all records from the `employees` table and `machines` table; uses the `device_id` column to join the two tables

`INNER JOIN`

Returns records matching on a specified column that exists in more than one table; the column used to join the tables is specified following `INNER JOIN` with syntax that includes `ON` and equal to (`=`)

```
SELECT *
FROM employees
INNER JOIN machines ON employees.device_id =
machines.device_id;
```

Returns all records that have a value in the `device_id` column in the `employees` table that matches a value in the `device_id` column in the `machines` table

`LEFT JOIN`

Returns all the records of the first table, but only returns records of the second table that match on a specified column; the first (or left) table appears directly after the keyword `FROM`; the column used to join the tables is specified following `LEFT JOIN` with syntax that includes `ON` and equal to (`=`)

```
SELECT *
FROM employees
LEFT JOIN machines ON employees.device_id =
machines.device_id;
```

Returns all records from the `employees` table but only the records from the `machines` table that have a value in the `device_id` column that matches a value in the `device_id` column in the `employees` table

`RIGHT JOIN`

Returns all of the records of the second table, but only returns records from the first table that match on a specified column; the second (or right) table appears directly after the `RIGHT JOIN` keyword; the column used to join the tables is specified following `RIGHT JOIN` with syntax that includes `ON` and equal to (`=`)

```
SELECT *
FROM employees
RIGHT JOIN machines ON employees.device_id =
machines.device_id;
```

Returns all records from the `machines` table but only the records from the `employees` table that have a value in the `device_id` column that matches a value in the `device_id` column in the `machines` table

Perform calculations

The following SQL keywords are aggregate functions and are helpful when performing calculations.

AVG

Returns a single number that represents the average of the numerical data in a column; placed after `SELECT`

```
SELECT AVG(height)
```

Returns the average height from all records that have a value in the `height` column

COUNT

Returns a single number that represents the number of records returned from a query; placed after `SELECT`

```
SELECT COUNT(firstname)
```

Returns the number of records that have a value in the `firstname` column

SUM

Returns a single number that represents the sum of the numerical data in a column; placed after `SELECT`

```
SELECT SUM(cost)
```

Returns the sum of costs from all records that have a value in the `cost` column

Logging System - Frontend Developer Guide

Available API Endpoints

All logging endpoints require **Admin role** authentication and use the `/logs` prefix.

1. Get API Request Logs

```
GET /logs/api
```

Query Parameters:

- `level` (optional): Filter by log level (DEBUG, INFO, WARNING, ERROR, CRITICAL)
- `path` (optional): Filter by request path (supports partial matching)
- `method` (optional): Filter by HTTP method (GET, POST, PUT, DELETE, etc.)
- `status_code` (optional): Filter by HTTP status code
- `user_id` (optional): Filter by user ID
- `user_email` (optional): Filter by user email
- `days` (default: 7): Number of days to look back
- `limit` (default: 100): Maximum number of logs to return
- `skip` (default: 0): Number of logs to skip (for pagination)

Response: Array of API log objects

2. Get System Logs

```
GET /logs/system
```

Query Parameters:

- `level` (optional): Filter by log level
- `source` (optional): Filter by source (module/function name)
- `days` (default: 7): Number of days to look back
- `limit` (default: 100): Maximum number of logs to return
- `skip` (default: 0): Number of logs to skip

Response: Array of system log objects

3. Get Error Logs Only

```
GET /logs/errors
```

Query Parameters:

- `days` (default: 7): Number of days to look back
- `limit` (default: 100): Maximum number of logs to return
- `skip` (default: 0): Number of logs to skip

Response: Array of error log objects

4. Get Specific Log by Request ID

```
GET /logs/request/{request_id}
```

Response: Single API log object or 404 if not found

5. Get Log Statistics

```
GET /logs/stats
```

Query Parameters:

- `days` (default: 7): Number of days for statistics

Response:

```json
{
  "total_logs": 1500,
  "error_logs": 45,
  "error_percentage": 3.0,
  "status_distribution": [
    {"_id": 200, "count": 1200},
    {"_id": 404, "count": 200},
    {"_id": 500, "count": 45}
  ],
  "top_error_paths": [
    {"_id": "/api/projects", "count": 15},
    {"_id": "/api/users", "count": 12}
  ],
  "slowest_endpoints": [
    {"_id": "/api/reports", "avg_duration": 2500.5},
    {"_id": "/api/exports", "avg_duration": 1800.2}
  ]
}
```

6. Cleanup Old Logs

```
DELETE /logs/cleanup
```

Query Parameters:

- `days` (default: 30): Delete logs older than this many days

Response:

```json
{
  "message": "Successfully deleted 150 old log entries"
}
```

Data Models

API Log Object

```json
{
  "id": "507f1f77bcf86cd799439011",
  "level": "ERROR",
  "method": "POST",
  "path": "/api/projects",
  "status_code": 500,
  "request_id": "123e4567-e89b-12d3-a456-426614174000",
  "user_id": "507f1f77bcf86cd799439012",
  "user_email": "user@example.com",
  "request_body": {"name": "Test Project"},
  "response_body": {"detail": "Internal server error"},
  "error_message": "Division by zero",
  "error_traceback": "Traceback...",
  "duration_ms": 152.34,
  "ip_address": "192.168.1.1",
  "user_agent": "Mozilla/5.0...",
  "created_at": "2023-01-01T00:00:00",
  "updated_at": "2023-01-01T00:00:00"
}
```

System Log Object

```json
{
  "id": "507f1f77bcf86cd799439013",
  "level": "INFO",
  "message": "Application started",
  "details": {"version": "1.0.0"},
  "source": "main.py",
  "request_id": null,
  "created_at": "2023-01-01T00:00:00",
  "updated_at": "2023-01-01T00:00:00"
}
```

Log Levels

- `DEBUG`: Detailed information for debugging
- `INFO`: General information about system operation
- `WARNING`: Something unexpected happened but system continues
- `ERROR`: Serious problem that prevented function execution
- `CRITICAL`: Very serious error that may abort the program

UI Implementation Tips

1. Log Viewer Component

- Use pagination with `limit` and `skip` parameters
- Implement filtering by level, method, status code, user
- Show logs in reverse chronological order (newest first)
- Color-code log levels (red for ERROR/CRITICAL, yellow for WARNING, etc.)

2. Error Dashboard

- Use `/logs/stats` to create charts and metrics
- Show error percentage and trends
- Display top error-prone endpoints
- Highlight slowest endpoints for performance monitoring

3. Real-time Updates

- Consider implementing periodic refresh for log views
- Show loading states during API calls
- Implement infinite scroll or traditional pagination

4. Log Details Modal

- Click on a log entry to show full details
- Format JSON request/response bodies nicely
- Show full error tracebacks with proper formatting
- Display all metadata (duration, IP, user agent, etc.)

5. Search and Filtering

- Provide date range picker for custom time periods
- Multi-select for log levels and HTTP methods
- Search by path, user email, or error message
- Clear all filters button

Security Notes

- All endpoints require Admin authentication
- Sensitive fields in request/response bodies are automatically redacted
- User passwords and tokens are not logged
- Access to logs should be restricted to authorized personnel only

Error Handling

- Handle 404 responses when logs are not found
- Show user-friendly messages for network errors
- Implement retry logic for failed requests
- Display appropriate loading and empty states