Mariadb Slow log

`/var/log/mysql/mariadb-slow.log` is the standard file path used to store the slow query log in MariaDB and MySQL databases on Linux systems. This log records any SQL queries that exceed a defined execution time threshold, helping database administrators diagnose and troubleshoot performance bottlenecks. [1, 2, 3, 4]
Here is a quick guide on how to verify, enable, and analyze this log file.

1. View Current Settings (MySQL Console)
Log into your MariaDB/MySQL console to see if the slow query log is active and where it is pointing: [1]
sql

```
SHOW VARIABLES LIKE 'slow_query_log%';
SHOW VARIABLES LIKE 'long_query_time';
```

Use code with caution.

- **`slow_query_log`**: Should be `ON` to record queries.
- **`slow_query_log_file`**: Path to the log file (e.g., `/var/log/mysql/mariadb-slow.log`).
- **`long_query_time`**: The threshold in seconds. Queries taking longer than this value are logged. [1, 2, 3, 4]

2. Enable it Temporarily (Runtime)
If the log is disabled, you can turn it on immediately without restarting the database service: [1, 2]
sql

```
SET GLOBAL slow_query_log = 'ON';
SET GLOBAL slow_query_log_file = '/var/log/mysql/mariadb-slow.log';
SET GLOBAL long_query_time = 2; -- Logs queries taking over 2 seconds
SET GLOBAL log_queries_not_using_indexes = 'ON'; -- Optional: logs unindexed queries
```

Use code with caution.

3. Enable it Permanently (Configuration File)
Runtime changes disappear if the database restarts. To make them permanent, edit your server's configuration file (usually `/etc/my.cnf`, `/etc/mysql/my.cnf`, or `/etc/mysql/mariadb.conf.d/50-`

`server.cnf`): [1, 2, 3]
ini

```
[mysqld]
slow_query_log = 1
slow_query_log_file = /var/log/mysql/mariadb-slow.log
long_query_time = 2
log_queries_not_using_indexes = 1
```

Use code with caution.

4. Set Linux Permissions
Because the `mysql` system user must write to this file, the directory and file permissions must be configured correctly in Linux: [1, 2]
bash

```
sudo mkdir -p /var/log/mysql
sudo touch /var/log/mysql/mariadb-slow.log
sudo chown -R mysql:mysql /var/log/mysql
sudo chmod 750 /var/log/mysql
```

Use code with caution.

After updating the configuration file and permissions, restart the service: [1, 2]
bash

```
sudo systemctl restart mariadb
# Or on some systems: sudo systemctl restart mysql
```

Use code with caution.

5. How to Analyze the Log
Slow query logs can get incredibly large and difficult to read manually. Instead of using `cat` or `vi`, use native parsing tools: [1, 2]

- **Using `mysqldumpslow`**: A built-in command-line tool that groups similar queries together and sorts them by execution time.
  bash

```
mysqldumpslow -s t /var/log/mysql/mariadb-slow.log
```

Use code with caution.

- **Using `pt-query-digest`**: Part of the advanced Percona Toolkit, this creates highly detailed and easily readable performance summaries.
  bash

```
pt-query-digest /var/log/mysql/mariadb-slow.log
```

Use code with caution.

[1, 2]

If you are experiencing issues with this log, please let me know:

- Is your database **currently experiencing slow downs** or high CPU usage?
- Do you need help **optimizing a specific slow query** found in this log?
- Are you getting a **permission denied error** when MariaDB tries to create the file?

AI responses may include mistakes. Learn more

- mariadb.com

  Slow Query Log Overview | Server | MariaDB Documentation
  Slow Query Log Overview. Complete Slow Query Log Overview gu...

- plesk

  Plesk

  How to enable the MySQL/MariaDB slow query log and analyze it on ...
  Answer * Connect to your Plesk server via SSH. * Enable the ...

- cPanel

  How to Enable the Slow Query Log in MySQL® or MariaDB
  To enable the Slow Query Log for MySQL or MariaDB: * Log in ...

Show all

what it could be useful?

The slow query log is one of the most powerful tools available for database optimization. It functions like a traffic camera for your database, catching the exact queries that are slowing down your applications.
Here is exactly how it is useful:
1. Finding Performance Bottlenecks

- **Identifies slow pages:** If a website page takes 5 seconds to load, this log reveals the exact SQL query causing the delay.
- **Pins down high CPU/Memory:** Queries that scan millions of rows without using indexes spike server resources. The log flags them immediately.

2. Discovering Missing Indexes

- **Spotting unindexed tables:** By enabling `log_queries_not_using_indexes`, the log captures queries performing full table scans.
- **Easy fixes:** Adding a single index to a column flagged in this log can often drop a query's execution time from 3 seconds to 3 milliseconds.

3. Catching Code Problems (Application Bugs)

- **Poorly written ORM queries:** Modern web frameworks (like Laravel, Hibernate, or Django) sometimes generate massive, inefficient SQL joins behind the scenes.
- **N+1 query problems:** It reveals loops where your application accidentally runs hundreds of tiny, redundant queries instead of one efficient query.

4. Database Capacity Planning

- **Tracking growth trends:** By comparing log sizes and query counts month-over-month, you can see if your database performance is degrading as your user base grows.
- **Postponing costly upgrades:** Optimizing the top 3 slow queries in this log can often slash server resource usage by 50%, saving you from paying for a larger, more expensive server.

To help you get the most out of this, let me know:

- Is your database **currently running slow**, or are you just setting this up proactively?

- What **application or framework** (e.g., WordPress, Node.js, Python) is connecting to this database?
- Would you like an example of **how to read a raw entry** from this log file?

mariadb-slow.log.1 is about 3.0G is it normal ?

No, a 3.0 GB `mariadb-slow.log.1` is not normal for a healthy, optimized production database.
The `.1` extension indicates this is a **rotated log file** (an older log preserved by Linux `logrotate`), meaning your database generated 3 GB of slow query data in a single log rotation period—typically just one week or one day.
A file this large means your database server is wasting massive amounts of CPU, RAM, and disk I/O processing inefficient queries.

Why is it so large?

1. **Missing Indexes (Most Common):** If a query searches a table with 1 million rows without an index, MariaDB must read all 1 million rows from the disk every single time. This is called a "Full Table Scan."
2. **`log_queries_not_using_indexes` is ON:** If this setting is enabled, MariaDB logs **every single query** that doesn't use an index, even if it executes in 0.001 seconds. On a busy site, this will inflate the log file to gigabytes in hours.
3. **`long_query_time` is set too low:** If this is set to `0` or a very low fraction (like `0.01` seconds) on a high-traffic application, almost every single query gets logged.
4. **An Application Loop Bug:** Your application code might be trapped in a loop, running the same slow query thousands of times per minute.

Step-by-Step Action Plan
1. Safely inspect it without crashing your terminal
Do **not** try to open a 3 GB file with `cat`, `nano`, or `vi`, as it can freeze your server's memory.
Instead, look at just the first few lines:
bash

```
head -n 50 /var/log/mysql/mariadb-slow.log.1
```

Use code with caution.

2. Check your configuration
Run this command inside the MariaDB console to check if it's logging non-indexed queries:
sql

```
SHOW VARIABLES LIKE 'log_queries_not_using_indexes';
SHOW VARIABLES LIKE 'long_query_time';
```

Use code with caution.

- If `log_queries_not_using_indexes` is `ON` and you have a busy database, **turn it off** to stop the log explosion:
  sql

  ```
  SET GLOBAL log_queries_not_using_indexes = 'OFF';
  ```

  Use code with caution.

3. Summarize the 3 GB file instantly
Do not read it manually. Use `mysqldumpslow` to extract the top 5 worst offending queries that are taking up all that space:
bash

```
mysqldumpslow -s c -t 5 /var/log/mysql/mariadb-slow.log.1
```

Use code with caution.

*(This sorts the file by **count** (`-s c`) and shows the top 5 (`-t 5`) query patterns causing the bulk of the log size).*
4. Reclaim your disk space safely
Once you have reviewed or backed up the file, you can empty it to reclaim 3 GB of disk space instantly without restarting MariaDB:
bash

```
sudo truncate -s 0 /var/log/mysql/mariadb-slow.log.1
```

Use code with caution.

If you'd like to fix this, let me know:

- What is the output of the `mysqldumpslow` **command** above? (You can paste a small snippet of the top query pattern).
- What are your current `long_query_time` and `log_queries_not_using_indexes` values?
- Is your server currently **running out of disk space** because of this?

meson@CiCADA: 17:56:36 /var/log/mysql$ head -n 50 /var/log/mysql/mariadb-slow.log.1 /usr/sbin/mariadbd, Version: 10.11.14-MariaDB-0ubuntu0.24.04.1-log (Ubuntu 24.04). started with: Tcp port: 0 Unix socket: /run/mysqld/mysqld.sock Time Id Command Argument # Time: 260725 22:46:02 # User@Host: cloud_user[cloud_user] @ localhost [] # Thread_id: 162 Schema: cloud QC_hit: No # Query_time: 0.089309 Lock_time: 0.000092 Rows_sent: 1 Rows_examined: 191623 # Rows_affected: 0 Bytes_sent: 68 # Full_scan: Yes Full_join: No Tmp_table: No Tmp_table_on_disk: No # Filesort: No Filesort_on_disk: No Merge_passes: 0 Priority_queue: No # # explain: id select_type table type possible_keys key key_len ref rows r_rows filtered r_filtered Extra # explain: 1 SIMPLE oc_filecache ALL NULL NULL NULL NULL 186499 191623.00 100.00 79.09 Using where # use `cloud`; SET timestamp=1784987162; SELECT COUNT(*) FROM `oc_filecache` WHERE `path` LIKE 'appdata_ocnu72fp1mua%'; # Time: 260726 1:46:03 # User@Host: cloud_user[cloud_user] @ localhost [] # Thread_id: 536 Schema: cloud QC_hit: No # Query_time: 0.067386 Lock_time: 0.000065 Rows_sent: 1 Rows_examined: 191623 # Rows_affected: 0 Bytes_sent: 68 # Full_scan: Yes Full_join: No Tmp_table: No Tmp_table_on_disk: No # Filesort: No Filesort_on_disk: No Merge_passes: 0 Priority_queue: No # # explain: id select_type table type possible_keys key key_len ref rows r_rows filtered r_filtered Extra # explain: 1 SIMPLE oc_filecache ALL NULL NULL NULL NULL 186499 191623.00 100.00 79.09 Using where # SET timestamp=1784997963; SELECT COUNT(*) FROM `oc_filecache` WHERE `path` LIKE 'appdata_ocnu72fp1mua%'; # Time: 260726 4:51:03 # User@Host: cloud_user[cloud_user] @ localhost [] # Thread_id: 975 Schema: cloud QC_hit: No # Query_time: 0.079616 Lock_time: 0.000082 Rows_sent: 1 Rows_examined: 191623 # Rows_affected: 0 Bytes_sent: 68 # Full_scan: Yes Full_join: No Tmp_table: No Tmp_table_on_disk: No # Filesort: No Filesort_on_disk: No Merge_passes: 0 Priority_queue: No # # explain: id select_type table type possible_keys key key_len ref rows r_rows filtered r_filtered Extra # explain: 1 SIMPLE oc_filecache ALL NULL NULL NULL NULL 186499 191623.00 100.00 79.09 Using where # SET timestamp=1785009063; SELECT COUNT(*) FROM `oc_filecache` WHERE `path` LIKE 'appdata_ocnu72fp1mua%'; # Time: 260726 7:56:02 # User@Host: cloud_user[cloud_user] @ localhost [] # Thread_id: 1316 Schema: cloud QC_hit: No # Query_time: 0.083674 Lock_time: 0.000084 Rows_sent: 1 Rows_examined: 191623 # Rows_affected: 0 Bytes_sent: 68 # Full_scan: Yes Full_join: No Tmp_table: No Tmp_table_on_disk: No # Filesort: No Filesort_on_disk: No Merge_passes: 0 Priority_queue: No MariaDB [(none)]> SHOW VARIABLES LIKE 'log_queries_not_using_indexes'; +-------------------------------+-------+ | Variable_name | Value | +-------------------------------+-------+ | log_queries_not_using_indexes | ON | +-------------------------------+-------+ 1 row in set (0.003 sec) MariaDB [(none)]> SHOW VARIABLES LIKE 'long_query_time'; +-----------------+----------+ | Variable_name | Value | +-----------------+----------+ | long_query_time | 2.000000 | +-----------------+----------+ 1 row in set (0.003 sec)

This output explains exactly why your log file reached 3 GB.

Your database belongs to a **Nextcloud or OwnCloud instance** (indicated by the `oc_filecache` table). The log is exploding because **`log_queries_not_using_indexes` is turned ON**, combined with Nextcloud constantly checking its app data folder.

The Root Cause
Look closely at this specific part of your log entry:

- **`Query_time: 0.089309`**: The query is actually very fast, taking only **0.08 seconds** (well below your 2.0-second `long_query_time` threshold).
- **`Full_scan: Yes`**: The query is doing a full table scan, checking all `191,623` rows in the `oc_filecache` table every single time it runs.
- **Why it's logged:** Because `log_queries_not_using_indexes = ON`, MariaDB is forced to write this fast query to your log file every few hours. On a busy server with multiple users or cron jobs, this fills up gigabytes of text very quickly.

How to Fix It
Step 1: Turn off the unindexed query logging (Instant Relief)
This stops MariaDB from writing fast, unindexed queries to your log, immediately freezing the growth of the file. Run this in your MariaDB console:
sql

```
SET GLOBAL log_queries_not_using_indexes = 'OFF';
```

Use code with caution.

To make this change permanent so it survives a server reboot, open your MariaDB configuration file (e.g., `/etc/mysql/mariadb.conf.d/50-server.cnf` or `/etc/my.cnf`) and change or add this line under the `[mysqld]` section:
ini

```
log_queries_not_using_indexes = 0
```

Use code with caution.

Step 2: Reclaim your 3 GB disk space
Now that the logging behavior is changed, you can safely wipe out the bloated backup log file to free up your hard drive:
bash

```
sudo truncate -s 0 /var/log/mysql/mariadb-slow.log.1
```

Use code with caution.

Step 3: Address the Nextcloud Index Issue (Optional but Recommended)
The query `SELECT COUNT(*) FROM oc_filecache WHERE path LIKE 'appdata_...%'` uses a wildcard (`%`) at the end of the string. Nextcloud *should* be using its built-in index on the `path` column, but sometimes these indexes get corrupted or missing during major Nextcloud upgrades.
You can tell Nextcloud to check and repair its database indexes automatically. Run this command in your terminal from your Nextcloud installation directory (usually `/var/www/nextcloud` or `/var/www/html`):
bash

```
sudo -u www-data php occ db:add-missing-indices
```

Use code with caution.

(Replace `www-data` with your web server user if you are using a different one like `nginx` or `apache`).

If you run into any issues, let me know:

- Did turning off `log_queries_not_using_indexes` successfully stop the log from growing?
- Do you know the **exact directory path** where your Nextcloud/OwnCloud instance is installed?
- Are you running this instance inside a **Docker container**, or directly on the host system?

Revision #2
Created 2026-08-27 09:18:56 UTC by meson
Updated 2026-08-27 09:20:42 UTC by meson