Checking Database Size and Related Issues

As your MySQL database grows, it’s crucial to track how space is being used across schemas, tables, and indexes. Uncontrolled growth can slow down queries, consume disk space, and complicate backups. While Elestio provides managed infrastructure, database storage optimization is still your responsibility. This guide explains how to inspect database size, find the largest tables and indexes, detect unused or bloated space, and optimize your data layout in MySQL.

Checking Database and Table Sizes

MySQL’s information_schema tables provide insights into how storage is distributed across your databases. This data helps prioritize cleanup, tuning, or archiving strategies. To calculate the total size used by each database:

```
SELECT table_schema AS db_name,
       ROUND(SUM(data_length + index_length) / 1024 / 1024, 2) AS size_mb
FROM information_schema.tables
GROUP BY table_schema
ORDER BY size_mb DESC;
```

This output includes both table data and indexes, giving you an overview of which databases are consuming the most space. To identify the largest tables across all schemas:

```
SELECT table_schema, table_name,
       ROUND((data_length + index_length) / 1024 / 1024, 2) AS total_size_mb
FROM information_schema.tables
ORDER BY total_size_mb DESC
LIMIT 10;
```

This helps pinpoint space-heavy tables so you can review their contents, indexes, or retention policy. To break down data size versus index size for the top tables in a specific schema:

```
SELECT table_name,
       ROUND(data_length / 1024 / 1024, 2) AS table_mb,
       ROUND(index_length / 1024 / 1024, 2) AS index_mb
FROM information_schema.tables
WHERE table_schema = 'your_database'
ORDER BY table_mb DESC
LIMIT 10;
```

Replace '`your_database`' with your actual schema name. A high index-to-data ratio could indicate overly aggressive indexing or opportunities for consolidation.

Detecting Bloat and Unused Space

MySQL tables especially those using the InnoDB storage engine can accumulate unused space over time due to updates, deletes, or internal fragmentation. This can inflate table size and degrade performance.

To list tables with free (unused) space that could be reclaimed:

```
SELECT table_name,
       ROUND(data_free / 1024 / 1024, 2) AS free_space_mb
FROM information_schema.tables
WHERE table_schema = 'your_database'
  AND data_free > 0
ORDER BY free_space_mb DESC
LIMIT 10;
```

Large data_free values may indicate internal fragmentation or deleted rows that haven’t been reclaimed yet. You can recover this space using table optimization. To view the number of rows deleted but not yet reclaimed (estimated):

```
SHOW TABLE STATUS FROM your_database;
```

Check the Rows and Data_free columns for each table. If many rows have been deleted but space hasn’t shrunk, the table may need to be optimized.

Optimizing and Reclaiming Storage

Once bloated or inefficient tables have been identified, MySQL provides several tools for optimization:

Reclaim free space and defragment tables

```
OPTIMIZE TABLE your_table;
```

This command rewrites the table and indexes, reclaiming space and improving performance. It's safe for InnoDB tables and especially useful after large `DELETE` or `UPDATE` operations.

Rebuild fragmented or oversized indexes

If indexes have grown large due to repeated updates or inserts, rebuilding them can reduce size and improve query speed. Use:

```
ALTER TABLE your_table ENGINE=InnoDB;
```

This effectively recreates the table and all associated indexes, helping reclaim space and improve internal ordering.

> Note: Both OPTIMIZE and ALTER ENGINE operations lock the table for a short period. Run these during maintenance windows if the table is actively queried.

Remove or archive old rows

For time-series data or logs, consider deleting or archiving old records:

```
DELETE FROM your_table
WHERE created_at < NOW() - INTERVAL 90 DAY;
```

Use EXPLAIN before executing large deletes to ensure they use indexes efficiently. You may also consider archiving to flat files or cold-storage tables.

Partition large tables for better control

If tables grow continuously (e.g., transaction logs or audit trails), use MySQL's **range partitioning** or **list partitioning**:

```
CREATE TABLE logs (
  id BIGINT,
  created_at DATE,
  ...
)
PARTITION BY RANGE (YEAR(created_at)) (
  PARTITION p2022 VALUES LESS THAN (2023),
  PARTITION p2023 VALUES LESS THAN (2024),
  PARTITION pmax VALUES LESS THAN MAXVALUE
);
```

Partitioning allows you to drop old data in chunks without full table scans or long DELETE operations.

Best Practices for Storage Management

- **Avoid storing large binary data in MySQL.** Store files like images and videos in external object storage and reference them by URL or metadata.
- **Monitor binary logs and purge them periodically.** If replication or point-in-time recovery isn’t needed beyond a certain timeframe, add to your config:

```
expire_logs_days = 7
```

- Or purge manually:

```
PURGE BINARY LOGS BEFORE NOW() - INTERVAL 7 DAY;
```

- **Track backup file size and location.** Ensure backups are stored on a separate volume or offsite to avoid filling the same disk as your live database.
- **Enable slow query logging** to detect inefficient queries that cause unnecessary data scans and table growth.
- **Use monitoring tools** (like Netdata, Prometheus exporters, or custom alert scripts) to track disk consumption trends over time.

Revision #1
Created 2025-04-30 09:06:20 UTC
Updated 2025-04-30 09:10:43 UTC