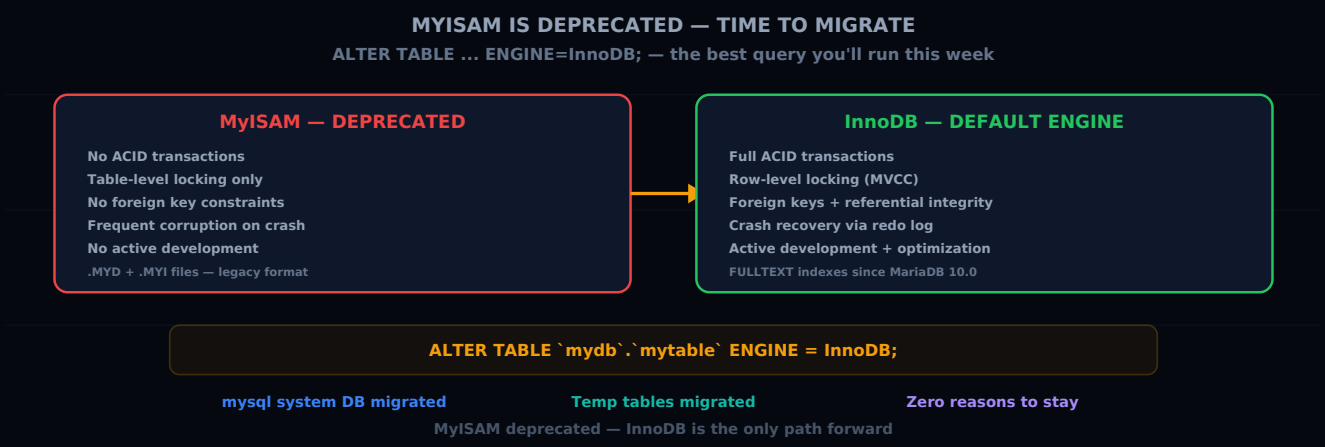
MyISAM Is Deprecated: Time to Migrate

Sylvain ARBAUDIE · August 4, 2025

MYSQL MYISAM INNODB MIGRATION



The End of an Era

MyISAM is officially deprecated. This is not a surprise — it has been obvious for years. But this time, it is written in the MariaDB / MySQL source code: the MyISAM engine is marked as deprecated, and the last bastions that used it internally have been migrated.

The `mysql` system database (the one storing users, privileges, grant tables) no longer uses MyISAM. Internal temporary tables do not either. The last two excuses for tolerating MyISAM in production have just disappeared.

Why MyISAM Survived So Long

To understand the current situation, we need to go back to MyISAM's history and its role in the ecosystem.

MyISAM was the default storage engine for MySQL until version 5.5 (2010). For over a decade, it was the default choice for millions of web applications. WordPress, Joomla, Drupal, phpBB — all these applications were developed and tested primarily with MyISAM.

MyISAM's advantages were real at the time:

- **Simplicity**: one `.MYD` file for data, one `.MYI` file for indexes. Easy to back up, easy to move.

- **Read performance**: for read-heavy workloads (blogs, showcase websites), MyISAM was fast.
- **Full-text search**: MyISAM supported full-text search long before InnoDB did.
- **Low memory footprint**: MyISAM used little RAM, which was crucial in the era of 512 MB servers.

But these advantages have become relics. InnoDB now offers all of this and much more.

Why You Must Migrate Now

No Transactions

MyISAM does not support ACID transactions. No `BEGIN`, no `COMMIT`, no `ROLLBACK`. Every statement is auto-committed. In case of a crash during a write, your data is in an indeterminate state.

No Row-Level Locking

MyISAM uses table-level locking. A single write locks the entire table, blocking all other reads and writes. With InnoDB, locking happens at the row level, enabling concurrency.

No Foreign Keys

MyISAM does not support foreign key constraints. No referential integrity at the database level. You depend entirely on the application to maintain data consistency.

Frequent Corruption

MyISAM tables are notoriously fragile. An abrupt server shutdown, a full disk, a `kill -9` of the mysqld process — and your tables are corrupted. `myisamchk` and `REPAIR TABLE` become your best friends, but they are not infallible.

No Active Development

This may be the most important argument. Nobody works on MyISAM anymore. No bug fixes, no performance optimizations, no new features. It is frozen code accumulating technical debt.

Migration: Simpler Than You Think

The good news is that migrating from MyISAM to InnoDB is generally straightforward.

Identify MyISAM Tables

```
SELECT table_schema, table_name, engine, table_rows,
       ROUND(data_length / 1024 / 1024, 2) AS data_mb
FROM information_schema.tables
WHERE engine = 'MyISAM'
  AND table_schema NOT IN ('mysql', 'information_schema',
                           'performance_schema', 'sys')
ORDER BY data_length DESC;
```

Convert a Table

```
ALTER TABLE mydb.mytable ENGINE = InnoDB;
```

That is it. MariaDB / MySQL rebuilds the table with the InnoDB engine. Indexes are recreated, data is copied. For small tables, it is instantaneous. For large tables, it may take a few minutes.

Points of Attention

A few special cases to watch during migration:

Full-text tables: If you use FULLTEXT indexes on MyISAM, good news — InnoDB has supported FULLTEXT indexes since MySQL 5.6 / MariaDB 10.0. The syntax is identical.

MERGE tables: If you use the MERGE engine (a union of MyISAM tables), you will need to rethink your architecture. InnoDB partitioning or views are alternatives.

COUNT()* without WHERE: MyISAM stores the exact row count, making `SELECT COUNT()* FROM table `instantaneous. InnoDB must scan an index. If your application frequently performs unconditional` COUNT(*)`, you will notice a difference (minor for tables under one million rows).

Disk space: InnoDB uses more disk space than MyISAM for the same data (on average 1.5 to 2x more), mainly due to MVCC and transaction management. Check your available space before migration.

Bulk Migration Script

For databases with many MyISAM tables, here is a systematic approach:

```
-- Generate ALTER TABLE commands
SELECT CONCAT('ALTER TABLE `', table_schema, '`.`', table_name,
              '` ENGINE=InnoDB;') AS migration_sql
FROM information_schema.tables
WHERE engine = 'MyISAM'
  AND table_schema NOT IN ('mysql', 'information_schema',
                           'performance_schema', 'sys')
ORDER BY data_length ASC;
```

Start with the smallest tables to validate the process, then move to the larger ones.

After Migration

Once all your tables are migrated to InnoDB, a few configuration adjustments are recommended:

```
# my.cnf
[mysqld]
default_storage_engine = InnoDB
innodb_buffer_pool_size = 70%  # of available RAM
innodb_log_file_size = 256M    # or more depending on load
innodb_flush_log_at_trx_commit = 1  # full durability
```

And disable MyISAM features you no longer need:

```
skip-external-locking
key_buffer_size = 8M  # minimum, for remaining system tables
```

Conclusion

MyISAM is deprecated. This is no longer an opinion; it is a technical fact. The system database has migrated, temporary tables have migrated, the code is in maintenance mode with no future.

If you still have MyISAM tables in production, the time to migrate is now. The conversion is simple, the benefits are immediate (transactions, row-level locking, crash recovery), and the risks of staying on MyISAM only increase.

`ALTER TABLE ... ENGINE=InnoDB;` — that is the best query you will run this week.

This article was originally published on Medium.