

BLOG

Understanding Lock Granularity in MySQL

Lukas Vileikis

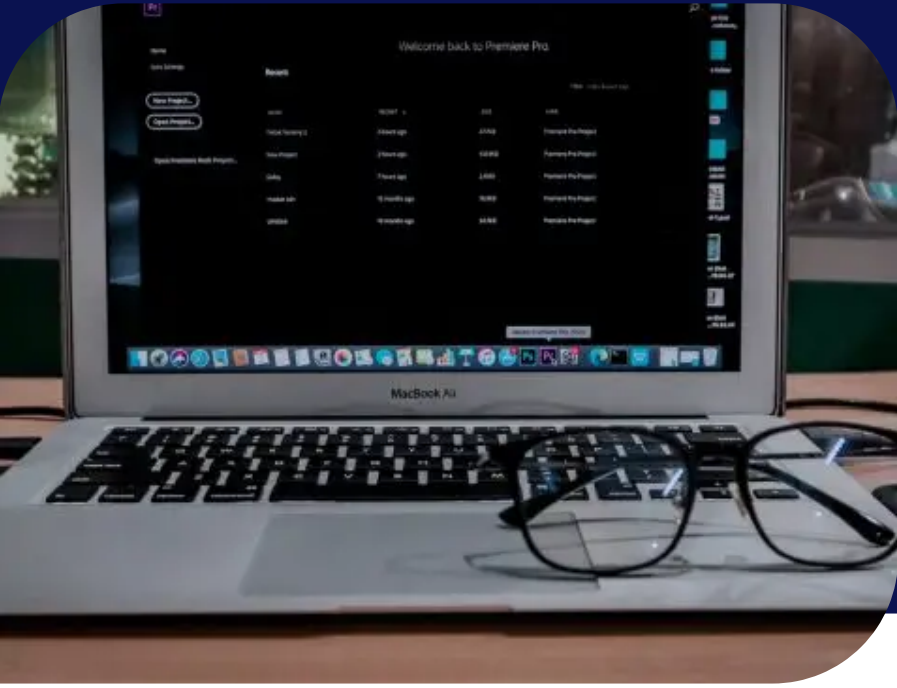
Published April 13, 2021

[Database - General](#)

[Performance Management](#)

[MariaDB](#)

[MySQL](#)



If you've been working with MySQL for some time, you have probably heard the terms "table-level locking" and "row-level locking". These terms refer to the lock granularity in MySQL – in this blog we will explain what they mean and what they can be used for.

What is Lock Granularity in MySQL?

Each MySQL storage engine supports different levels of granularity for their locks. MySQL has three lock levels: row-level locking, page-level locking and table-level locking. Each MySQL storage engine implements locking differently giving you some distinct advantages and disadvantages. We'll first look into what lock granularity is, then look into how everything works in different storage engines.

Broadly speaking, locks in MySQL fall into one of these categories. Locks can be:

Page-level – such types of lock granularities were available in older engines of MySQL, specifically BDB, [which is now obsolete as of MySQL 5.1](#). In short, BDB was a storage engine included in the older versions of MySQL and it was a transactional storage engine which performed page-level locks. Since these types of lock granularities are no longer used we will not go in-depth into them here, but in general, these locks are limited to the data and indexes that reside on a particular page. If you want to learn more about BDB, [the page on MariaDB](#) should provide some more information.

Table-level – MySQL uses table-level locking for all storage engines except InnoDB.

Row-level – row-level locking is used by InnoDB.

The Advantages and Disadvantages of Table-level Locking

MySQL uses table-level locking for all storage engines except InnoDB meaning that table-level locking is used for tables running the MyISAM, MEMORY and MERGE storage engines, permitting only one session to update tables at a time. Table-level locks have some distinct advantages over row-level locks (for example, table-level locking in general requires a little less memory than row-level locking because row-level locking requires some memory per row (or group) of the rows that are locked and it's usually fast because only one lock is involved. Table write locks are put on a table if there are no locks on it – if there are pre-existing locks on the table in question, the table lock requests are put in the read request queue. It's worth mentioning that table-level locking has some distinct disadvantages unique to itself too – for example, it might not be a very good fit for applications that require a lot of transactions that go "back and forth" (e.g., an online banking application) because only one session can write to a table at any one time and some of the tables that support table-level locking (such as MyISAM) do not support the ACID model.

Here's an example: imagine a banking application that uses two tables in a database – let's say those tables are called "checking" and "savings". You need to move \$100 from a person's checking account to his savings account. Logically, you would perform the following steps:

Make sure the account balance is greater than \$100.

Subtract \$100 from the checking account.

Add \$100 to the savings account.

To perform these actions, you would need a couple of queries, for example:

```
SELECT balance FROM checking WHERE account_id = 123;
UPDATE checking SET balance = balance - 100 WHERE account_id = 123;
UPDATE savings SET balance = balance + 100 WHERE account_id = 123;
```

These queries might look simple, but if you use MyISAM (we use MyISAM as an example as it's one of the primary storage engines that supports table-level locks), you should be familiar with the fact that the engine doesn't support ACID either which means that if the database server crashes while performing any of those queries, you're out of luck: people could end up with cash in both accounts or in neither one of them. The only engine that supports ACID-based transactions in MySQL is InnoDB, so if you need a lot of reliable transactions, it might be worth looking into it. InnoDB also supports row-level locking – this is what we will look into now.

The Advantages and Disadvantages of Row-level Locking

MySQL uses row-level locking for InnoDB tables to support simultaneous write access by multiple sessions. Some of the advantages of using row-level locking include the ability to lock a single row for long periods of time and fewer lock conflicts when many threads access different rows. However, row-level locking has disadvantages too: one of them is that row-level locking usually takes up more memory than page-level or table-level locking, it's also usually slower than page-level or table-level locks because the engine must acquire more locks. InnoDB is one of the engines that is supporting a row-level locking mechanism: it's also ACID compliant meaning that it is a good fit for transaction-based applications (refer to the example above). Now we will look into how lock granularity works in one of MySQL storage engines.

How does Lock Granularity Work in InnoDB?

InnoDB is widely known to be supporting row-level locking, but it's also worth noting that the engine supports multiple types of locking which means that you can use both row-level and table-level locks. InnoDB performs row-level locking by setting shared or exclusive locks on the index records it encounters when it searches

InnoDB also has other types of locks – some of them include shared and exclusive locks, intention locks, record locks, gap locks, next-key locks and next intention locks. Intention locks, for example, can also be shared or exclusive – such locks usually indicate that a transaction intends to set a certain type of a lock (a shared lock or an exclusive lock) on individual rows in a table, a record lock is a lock on an index record etc.

In general, InnoDB lock granularity differs from the lock granularity present in other MySQL storage engines (for example, MyISAM) because when table-level locking is in use, only one session to update certain tables at a time can run. When row-level locking is in use, MySQL supports simultaneous write access across multiple sessions making row-level locking storage engines (InnoDB) a suitable choice for mission-critical applications.

Lock Granularity and Deadlocks

Lock granularity and locking levels in MySQL can be a great thing, but they can also cause problems. One of the most frequent problems caused by lock granularity are deadlocks – a deadlock occurs when different MySQL transactions are unable to proceed because each of them holds a lock that the other needs. Thankfully, when using the InnoDB storage engine, deadlock detection is enabled by default – when a deadlock is detected, InnoDB automatically rolls back a transaction. If you encounter deadlocks when dealing with lock granularity in MySQL, don't fret – consider simply restarting your transaction. In order to proactively monitor your database, you should also consider utilizing the features provided by [ClusterControl](#).

How can ClusterControl Help You?

Here are some of the things [ClusterControl](#) developed by Severalnines can help you with:

- The protection of all of your business data
 - If your data is corrupted (that can be caused by not using an ACID-compliant storage engine or also by other factors as described above) the tool can run an automatic process that actually verifies that you can recover your data.
 - The tool can let you know which databases are not backed up or show you the status of your backups (whether they were successful or they failed)
- The automation of your database operations
 - ClusterControl can help you ensure that your sysadmins, developers and DBAs manage entire database clusters efficiently with minimal risks using industry best practices
- Effectively managing your database infrastructure in general
 - Today's shift in technologies combined with sophisticated infrastructure solutions requires advanced tools and knowledge to achieve high availability and optimal performance for your business-critical applications. [ClusterControl](#) can also help you with the deployment, monitoring, management and scaling of the most popular open source database technologies including MySQL, MariaDB, MongoDB, PostgreSQL, TimeScaleDB and the rest.

To learn more about how [ClusterControl](#) can help streamline your business operations, make sure to keep an eye on the [Severalnines database blog](#).

Summary


Different MySQL storage engines have different types of lock granularities available. Before deciding on the storage engine you should use, be sure to know as much information about the storage engine in question as possible (for example as already noted MyISAM should be avoided when dealing with mission-critical data because it's not ACID compliant), understand all of the related performance implications including lock granularities, deadlocks and the rest and choose wisely.

RELATED PRODUCTS


 **clustercontrol**

Worry-free database automation


Related content

- 

November 22, 2022 Sarah Morris

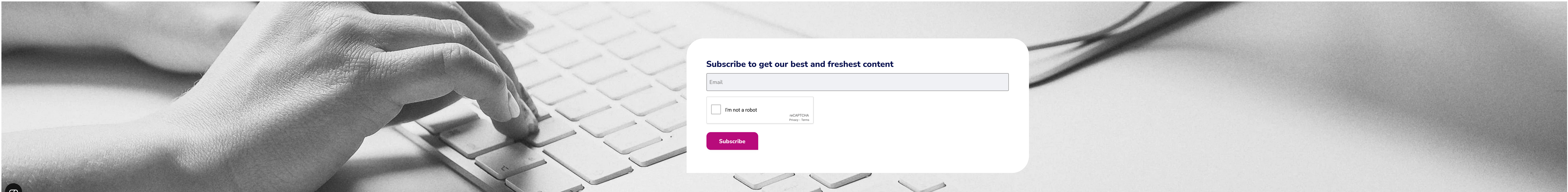
Severalnines named a 2022 IDC Innovator in PaaS
- 

November 1, 2022 Alex Yu

Introduction to the Severalnines ClusterControl API and SDK – Part 1
- 

September 28, 2022 Kyle Buzzell

Severalnines and Safespring partner to enable Sovereign DBaaS for European Cloud Market



Subscribe to get our best and freshest content

Email

☐ I'm not a robot

reCAPTCHA

Privacy - Terms

Subscribe

CCX

Backup Ninja

MariaDB

MongoDB

PostgreSQL

MySQL NDB

TimescaleDB

MySQL Galera

Hybrid cloud

Disaster recovery

Cloud Service Providers (CSPs)

Support

S9S slack

CC Training

CC Certification

Contact

Careers

Partners

this, that and more.

Email



I'm not a robot

reCAPTCHA

Privacy - Terms

Subscribe

