

## Why use InnoDB?

InnoDB is commonly viewed as anything but performant, especially when compared to MyISAM. Many actually call it slow. This view is mostly supported by old facts and mis-information. In reality, you would be very hard-pressed to find a current, production-quality MySQL Database Engine with the CPU efficiency of InnoDB. It has its performance "quirks" and there are definitely workloads for which it is not optimal, but for standard OLTP (Online Transaction Processing) loads, it is tough to find a better, safer fit.

### Performance

The performance claims for InnoDB are not idle, there are numbers to back it up and benchmarks run by some of the most well-respected MySQL consulting firms in existence. For example, the people behind [mysqlperformanceblog.com](http://www.mysqlperformanceblog.com), a leading MySQL tuning and optimization site run by Percona[1], ran a benchmark comparing Falcon, MyISAM and InnoDB. The benchmark was really supposed to be highlighting Falcon, except it was InnoDB that won the day, topping both Falcon and MyISAM in queries per second for almost every test: <http://www.mysqlperformanceblog.com/2007/01/08/innodb-vs-mysam-vs-falco...>

There are interesting trends in these graphs. InnoDB uses "clustered" indexes. This means that the data for the table is actually stored in the "leaves" of the primary key index, thus there is no need to fetch the row information separately. This is also one of the reasons for InnoDB's reputation of having a large memory footprint. When you are using MyISAM, the `key_buffer` is loaded with just the index of the table. In contrast, when InnoDB buffers its index it is implicitly buffering the data in the table as well, as the `innodb_buffer_pool` buffers on the memory page level.[8] As you might imagine, this can lead to excellent performance and very large memory requirements.

This technique of clustered primary keys is one reason for the fairly large margin in the benchmarks between MyISAM and InnoDB when the query in question uses the primary key for a range. What is somewhat surprising, though, is that InnoDB still wins in the tests that use a secondary index. This is comforting as it shows that the performance of the engine is not just based on key clustering, a performance boost that is very query dependent.

InnoDB uses some other tricks with indexes as well. It can build "adaptive hash indexes" for frequent queries and does so when an entire table comes close to fitting in memory.[5] These hash indexes are quite a bit faster than the standard BTree index (when the table is in memory). Again, this is another significant performance improvement at the expense of memory usage.

Returning to the benchmark above, we can also see that InnoDB still has some scaling issues when you get to high thread numbers (caused by many concurrent connections or requests). This is a known issue and while it improves with most major releases, it is good to keep in mind. Even with this issue, InnoDB is often used explicitly to enable concurrency.

## **Concurrency**

The irony is that while InnoDB has some mutex locking issues at high levels of concurrency, one tends to use it to enable concurrency at all. A major issue with MyISAM is the lack of row-level locking. This means that Drupal uses explicit table locks on some critical paths and that tables are implicitly locked on many non-critical paths, those that involve updates or inserts into the middle of a non-optimized table (which encompasses most inserts and updates). Row-level locking means these implicit locks usually don't happen. Drupal itself can be patched to remove many of the table locks from critical paths as they are no longer required. Locking sanity is now ensured at the engine level and this allows for many more concurrent operations, so many more in fact that you may have to limit the number of threads entering the InnoDB kernel to prevent the database server from thrashing. This is a much better problem to have than an artificial performance limit from table locks, but is a problem nonetheless.

A good example of run-away concurrency occurred with the drupal.org database server in early 2007. We had just converted to InnoDB and applied patches to Drupal to remove locking on the critical paths. At that time, we were using a version of MySQL that by default had unlimited InnoDB kernel threads. It was roughly an hour before we hit peak usage and the server ground to a halt. Looking at the MySQL process listing and vmstat output, it was easy to see what was going on. We had so many concurrent threads that switching between them (and the overhead inherent in that) was preventing any of them from moving forward at a reasonable pace. This is what you have to watch out for and why the InnoDB kernel thread limits are useful.

## **Reliability**

Any database administrator is assured to have the same nightmare, calling a client to tell them that their data is corrupted or just flat-out gone. MyISAM doesn't help these concerns as it ensures almost no data integrity. Hardware failures, unclean shutdowns and canceled operations are just a few of the events that can lead to MyISAM corruption. There are excellent tools to recover from this, but they are not guaranteed to work and their use requires sometimes extensive downtime for table checks/repairs.

On the other hand, InnoDB is a largely ACID (Atomicity, Consistency, Isolation, Durability) engine, built to guarantee consistency and durability. It does this through a transaction log (with the option of a two-phase commit if you have the binary log enabled), a double-write buffer and automatic checksumming and checksum validation of database pages. These safety measures not only prevent corruption on "hard" shutdowns, but can even detect hardware failure (such as memory failure/corruption) and prevent damage to your data.

Drupal.org has made use of this feature of InnoDB as well. The database in question contains a large amount of user contributed content, cvs messages, cvs history, forum messages, comments and, more critically, the issue queues for the entire Drupal project. This is not data where corruption is an option. In 2007, the master database server for the project went down. After examining the logs, it became clear that it hadn't crashed as such, but InnoDB had read a checksum from disk that didn't match the checksum it had in

memory. In this case, the checksum miss-match was a clear sign of memory corruption. Not only did it detect this, but it killed the MySQL daemon to prevent data corruption. In fact, it wouldn't let the MySQL daemon run for more than a half hour on that server without killing it after finding a checksum miss-match. When your data is of the utmost importance, this is very comforting behavior.

### **Data Security**

On a related topic, the transactional nature of InnoDB enables easy and online backups. A major issue with MyISAM is that any backup strategy designed to pull guaranteed, consistent backups will require table locks and any strategy that involves point-in-time recovery from binary logs will require a full database lock. This is totally unacceptable for a large, production-grade website. The only real way around this is to have a slave database server and pull backups from that machine.

InnoDB, on the other hand, can run a backup job in a single transaction and pull consistent, database-wide backups with only a short lock at the beginning of the job. The ease of pulling these backups quickly becomes addicting and makes it much easier to follow safe backup procedures.

## **Why use MyISAM?**

### **Simplicity**

So far, this has read somewhat like a paid advertisement for InnoDB. However, MyISAM has some very real advantages. One of these is the simplicity of the engine, it is very well understood and it's easy to write third-party tools to interact with it. There are very high-quality free tools, such as `mysqlhotcopy` available for MyISAM. It is much more difficult to write tools for an engine as complicated as InnoDB and this can be easily seen in the number of them available. Also, this simplicity allows for an ease of administration that is not there with InnoDB.

### **Optimization**

MyISAM's other main advantage is how long it has been around. There are many systems, Drupal for example, that are very much optimized for that particular engine. This is not to say that they perform poorly on InnoDB, but they are not optimized for it. For example, while many of Drupal's core queries are well indexed and use the primary key (thus benefiting from InnoDB's primary key clustering), some could be improved. The node table has a primary key on (nid,vid). Having this index is a good idea, but it is a two-integer index and there are eleven secondary indexes based on it. This doesn't mean much when you use MyISAM, but under InnoDB it means each of those secondary indexes has two-integer sized leaves identifying the primary key.

Another fact, is that there are some workloads MyISAM is better suited for. For example, Drupal's built-in search functionality performs horribly on InnoDB for very large data-sets, for example 100k+ rows. These tables are best left MyISAM. Fortunately, MySQL allows for mixing engines like this.

## Resource Usage

It is readily accepted in computer science that there is often a trade-off between speed and memory footprint. We have seen through the above benchmarks that InnoDB has some fast algorithms, however, this comes at a price. Not only does InnoDB use more memory than MyISAM, but the actual data files are often quite a bit larger. Add to this the fact that InnoDB has at least one quite large log file and you have a significant increase in resource use. This makes MyISAM a good fit for a resource-limited server. However, if you're concerned at all with high levels of concurrency, it is likely you have the funds to buy a server that can handle these increased resource demands.

## State Of MySQL Engines With Drupal

In summary, many of the historical concerns and rumors of InnoDB slowness are simply false. In most cases, InnoDB is the correct choice for a Drupal site. It provides increased concurrency, enhanced performance and much more data integrity than MyISAM ever can. However, the pluggable nature of MySQL engines allows the user to "mix and match" table engines inside a single database. This allows us to consider tables whose workloads fit MyISAM more so than InnoDB. The main candidates for MyISAM in a mostly InnoDB-centric database are the search tables. A database layout with most tables being InnoDB (for performance and data security) and the search tables being MyISAM (for performance on that particular workload and acknowledging the fact that, if that data was corrupted, it could be easily rebuilt) is an excellent fit.

## Determining InnoDB Resource Requirements

It is all well and good to wave one's hands and say "InnoDB clearly requires far more memory for these reasons," but it gets slightly difficult to pin down exactly how much more memory. This is true for several reasons:

1. How did you load your database?
  1. InnoDB table size is not a constant. If you took a straight SQL dump from a MyISAM table and inserted it into an InnoDB table, it is likely larger than it really needs to be. This is because the data was loaded out of primary key order and the index isn't tightly packed because of that. If you took the dump with the `--order-by-primary` argument to `mysqldump`, you likely have a much smaller table and will need less memory to buffer it.
2. What exactly is your table size?
  1. This is an easy question to answer with MyISAM: that information is directly in the output of "SHOW TABLE STATUS". However, the numbers from that same source for InnoDB are known to be estimates only [7]. The sizes shown are the physical sizes reserved for the tables and have nothing to do with the actual data size at that point. Even the row count is a best guess.
3. How large is your primary key?
  1. It was mentioned above that InnoDB clusters the data for a table around the primary key. This means that any secondary index leaves must contain the primary key of the data they "point to." Thus, if you have tables with a large primary key, you will need more memory to buffer a secondary index and more disk space to hold them. This is one of the reasons some people argue for short "artificial" primary keys for InnoDB tables when

there isn't one "natural" primary key.

In summary, there is no set method that will work for everyone to predict the needed resources. Worse than that, your needed resources will change with time as more inserts to your table increase its size and fragment the packing of the BTree. The best advice to be offered is use the mysqlreport tool available here (<http://hackmysql.com/mysqlreport>) to monitor your available innodb\_buffer\_pool and adjust accordingly, the most important InnoDB tunable. It is important to not run at 100% usage of the innodb buffer, as this likely means that you're not buffering as much as you could for reads, and that you're starving your write buffer which also lives in the same global innodb\_buffer.

## Resources

1. Percona  
<http://www.percona.com/team/peter-zaitsev.html>
2. General InnoDB Feature List  
<http://www.innodb.com/innodb/features/>
3. Graphed data from benchmarks  
<http://www.mysqlperformanceblog.com/2007/01/08/innodb-vs-myisam-vs-falco...>
4. Unformatted data from benchmarks  
[http://www.mysqlperformanceblog.com/files/benchmarks/innodb\\_scale.html](http://www.mysqlperformanceblog.com/files/benchmarks/innodb_scale.html)
5. InnoDB Adaptive Hash Index Support  
<http://dev.mysql.com/doc/refman/5.0/en/innodb-adaptive-hash.html>
6. InnoDB Insert Buffer  
<http://dev.mysql.com/doc/refman/5.0/en/innodb-insert-buffering.html>
7. InnoDB SHOW TABLE STATUS Restrictions (and other restrictions)  
<http://dev.mysql.com/doc/refman/5.0/en/innodb-restrictions.html>
8. InnoDB Internals - Source Files  
[http://forge.mysql.com/wiki/MySQL\\_Internals\\_Files\\_In\\_InnoDB\\_Sources](http://forge.mysql.com/wiki/MySQL_Internals_Files_In_InnoDB_Sources)

### About the author



*[Narayan Newton](#) is the Database Administrator for the Open Source Lab ([osuosl.org](http://osuosl.org)) and the Server Administrator for [Drupal.org](http://drupal.org). He has worked in IT for eight years with a focus on Open Source development and project hosting. He has specialized in database optimization, query optimization and infrastructure building.*

*With a massive, and growing, caffeine addiction he can usually be found working in a local coffee shop, catching indie bands/movies and riding bikes.*

Source: [Drupal Performance Agency](#)

License: Creative Commons Attribution-- ShareAlike 2.0