MySQL Storage Engines

Data in MySQL is stored in files (or memory) using a variety of different techniques. Each of these techniques employs different storage mechanisms, indexing facilities, locking levels and ultimately provides a range of different functions and capabilities. By choosing a different technique you can gain additional speed or functionality benefits that will improve the overall functionality of your application.

For example, if you work with a large amount of temporary data, you may want to make use of the MEMORY storage engine, which stores all of the table data in memory. Alternatively, you may want a database that supports transactions (to ensure data resilience).

Each of these different techniques and suites of functionality within the MySQL system is referred to as a storage engine (also known as a table type). By default, MySQL comes with a number of different storage engines pre-configured and enabled in the MySQL server. You can select the storage engine to use on a server, database and even table basis, providing you with the maximum amount of flexibility when it comes to choosing how your information is stored, how it is indexed and what combination of performance and functionality you want to use with your data.

This flexibility to choose how your data is stored and indexed is a major reason why MySQL is so popular; other database systems, including most of the commercial options, support only a single type of database storage. Unfortunately the 'one size fits all approach' in these other solutions means that either you sacrifice performance for functionality, or have to spend hours or even days finely tuning your database. With MySQL, we can just change the engine we are using.

In this article, we're not going to concentrate on the technical aspects of the different storage engines (although we will inevitably have to look at some of these elements); instead we will concentrate on how and where these different engines can be best employed. To achieve this, we'll have to look at some of the fundamental issues before moving on to the specifics of each engine type.

Determining Available Engines

You can determine a list of engines by using the show engines command within MySQL (assuming a MySQL server version later than 4.1.2).

Code:

```
mysql> show engines;
+------------+---------+----------------------------------------------------
-------+
| Engine     | Support | Comment
|
+------------+---------+----------------------------------------------------
-------+
| MyISAM     | DEFAULT | Default engine as of MySQL 3.23 with great
performance     |
| HEAP       | YES     | Alias for MEMORY
|
| MEMORY     | YES     | Hash based, stored in memory, useful for temporary
tables  |
| MERGE      | YES     | Collection of identical MyISAM tables
|
| MRG_MYISAM | YES     | Alias for MERGE
|
| ISAM       | NO      | Obsolete storage engine, now replaced by MyISAM
|
| MRG_ISAM   | NO      | Obsolete storage engine, now replaced by MERGE
|
| InnoDB     | YES     | Supports transactions, row-level locking, and
foreign keys |
| INNOBASE   | YES     | Alias for INNODB
|
| BDB        | NO      | Supports transactions and page-level locking
|
```

```
| BERKELEYDB | NO      | Alias for BDB
|
| NDBCLUSTER | NO      | Clustered, fault-tolerant, memory-based tables
|
| NDB        | NO      | Alias for NDBCLUSTER
|
| EXAMPLE    | NO      | Example storage engine
|
| ARCHIVE    | NO      | Archive storage engine
|
| CSV        | NO      | CSV storage engine
|
+------------+---------+-----------------------------------------------------
-------+
16 rows in set (0.01 sec)
```

The listing shows the full list of available database engines, and whether the support is available in the current database server.

For versions of MySQL earlier than 4.1.2, use

Code:

```
mysql> show variables like "have_%";
+------------------+----------+
| Variable_name    | Value    |
+------------------+----------+
| have_bdb         | YES      |
| have_crypt       | YES      |
| have_innodb      | DISABLED |
| have_isam        | YES      |
| have_raid        | YES      |
| have_symlink     | YES      |
| have_openssl     | YES      |
| have_query_cache | YES      |
```

```
+------------------+----------+
8 rows in set (0.01 sec)
```

You can configure the available engines within a MySQL installation by changing the options to the configure script. If you are using a pre-packaged MySQL binary distribution then most of the commonly used engines are included. Note, however, that if you want access to some of the more unusual types (particularly the CSV, ARCHIVE and BLACKHOLE engines) then you may want to build your MySQL database by hand.

Using an Engine

There are a number of ways you can specify the storage engine to use. The simplest method, if you have a preference for a engine type that fits most of your database needs to set the default engine type within the MySQL configuration file (using the option storage_engine or when starting the database server by supplying the --default-storage-engine or --default-table-type options on the command line).

More flexibility is offered by allowing you specify the storage engine to be used MySQL, the most obvious is to specify the engine type when creating the table:

Code: SQL

```
CREATE TABLE mytable (id int, title char(20)) ENGINE = INNODB
```

You can also alter the storage engine used for an existing table:

Code: SQL

```
ALTER TABLE mytable ENGINE = MyISAM
```

However, you should be careful when altering table types in this way as making a modification to a type that does not support the same indexes, field types or sizes may mean that you lose data. If you specify a storage engine that doesn't exist in the current database then a table of type MyISAM (the default) is created instead.

Differentiating the Engines

In order to make a decision about which engine to choose, we first need to think about the different core functionality provided in each engine that allows us to differentiate between them. We can generally divide up the core functionality into four areas; the supported field and data types, locking types, indexing and transactions. Some engines have unique functionality that can also drive your decision, we'll be taking a closer look at these specifics in a moment.

Field and Data Types

Although all of the engines support the common data types, i.e., integers, reals and character based storage, not all engines support other field types, particularly the BLOB (binary large object) or TEXT types. Other engines may support only limited character widths and data sizes.

These limitations, while directly affecting the information you store may also have a related effect to the types of searches you perform, or the indexes you create on that information. In turn, these differences can affect the performance and functionality of your application as you may have to make decisions about functionality based on the storage engine choice you make for the type of data you are storing.

Locking Types

Locking within database engines defines how access and updates to information are controlled. When an object in the database is locked for updating, other processes cannot modify (or in some cases read) the data until the update has completed.

Locking not only affects how many different applications can update the information in the database, it can also affect queries on that data. The reason for this is that the queries may be accessing data that may be being altered or updated. In general, such delays are minimal. The bulk of the locking mechanism is devoted to preventing multiple processes updating the same data. Since both additions (INSERT statements) and alterations (UPDATE statements) to the data require locking, you can imagine that multiple applications using the same database can

have a significant impact.

Locks are supported by different storage engines at different object levels, and these levels affect the concurrency of access to the information. Three different levels are supported, table locking, block locking and row locking. Table locking is most commonly supported and is the locking provided in MyISAM. It locks an entire table during an update. This will limit the number of applications that are updating a specific table to just one, and this can affect heavily used multi-user databases because it introduces delays into the update process.

Page level locking is used by the Berkeley DB storage engine and locks data according to the page (8Kb) of information that is being uploaded. When performing updates across a range of locations within the database, the locking is not a problem, but because adding rows involves locking the final 8Kb of the data structure, adding large numbers of rows, particularly of small data, can be a problem.

Row level locking provides the best concurrency; only individual rows within a table are locked, which means that many applications can be updating different rows of the same table without causing a lock situation. Only the InnoDB storage engine supports row level locking.

Indexing

Indexing can dramatically increase the performance when searching and recovering data from the database. Different storage engines provide different indexing techniques and some may be better suited for the type of data you are storing.

Some storage engines simply do not support indexing at all either because they use the indexing of the underlying tables (in the MERGE engine for example) or because the data storage method does not allow indexing (FEDERATED or BLACKHOLE engines).

Transactions

Transactions provide data reliability during the update or insert of information by enabling you to add data to the database, but only to commit that data when other conditions and stages in the application execution have completed successfully. For example, when transferring information from one account to another you would use

transactions to ensure that both the debit from one account and the credit to the other completed successfully. If either process failed, you could cancel the transaction and the changes would be lost. If the process completed,then we would confirm it by committing the changes.

Storage Engines

MyISAM

The MyISAM engine is the default engine in most MySQL installations and is a derivative of the original ISAM engine type supported in the early versions of the MySQL system. The engine provides the best combination of performance and functionality, although it lacks transaction capabilities (use the InnoDB or BDB engines) and uses table-level locking.

Unless you need transactions, there are few databases and applications that cannot effectively be stored using the MyISAM engine. However, very high-performance applications where there are large numbers of data inserts/updates compared to the number of reads can cause performance problems for the MyISAM engine. It was originally designed with the idea that more than 90% of the database access to a MyISAM table would be reads, rather than writes.

With table-level locking, a database with a high number of row inserts or updates becomes a performance bottleneck as the table is locked while data is added. Luckily this limitation also works well within the restrictions of a non-transaction database.

MyISAM Summary
Name MyISAM
Introduced v3.23
Default install Yes
Data limitations None
Index limitations 64 indexes per table (32 pre 4.1.2); Max 16 columns per index
Transaction No
Locking level Table

MERGE

The MERGE engine type allows you to combine a number of identical tables into a

single table. You can then execute queries that return the results from multiple tables as if they were just one table. Each table merged must have the same table definition.

The MERGE table is particularly effective if you are logging data directly or indirectly into a MySQL database and create an individual table per day, week or month and want to be able to produce aggregate queries from multiple tables. There are limitations to this however, you can only merge MyISAM tables and the identical table definition restriction is strictly enforced. Although this seems like a major issue, if you had used one of the other table types (for example InnoDB) then the merge probably wouldn't be required.

MERGE Summary
Name MERGE
Introduced v3.23.25
Default install Yes
Data limitations Underlying tables must be MyISAM
Index limitations N/A
Transaction No
Locking level Table

MEMORY

The MEMORY storage engine (previously known as the HEAP storage engine) stores all data in memory; once the MySQL server has been shut down any information stored in a MEMORY database will have been lost. However, the format of the individual tables is kept and this enables you to create temporary tables that can be used to store information for quick access without having to recreate the tables each time the database server is started.

Long term use of the MEMORY storage engine is not generally a good idea, because the data could so easily be lost. However, providing you have the RAM to support the databases you are working on, use of MEMORY based tables is an efficient way of running complex queries on large data sets and benefiting from the performance gains.

The best way to use MEMORY tables is to use a SELECT statement to select a larger data set from your original, disk-based, tables and then sub-analyse that information for the specific elements you want. I've used this technique in the past to

extract a month worth of web log data, actually from tables using the ARCHIVE storage engine, and then run the queries on specific URLs, sites and other focus points.

MEMORY Summary
Name MEMORY (HEAP, deprecated)
Introduced 1.0 (only known as MEMORY since 4.1)
Default install Yes
Data limitations BLOB and TEXT types not supported
Index limitations None
Transaction No
Locking level Table

EXAMPLE

The EXAMPLE engine is actually a programming example of a storage engine that can be used as the basis for other engines within the MySQL system. It does not support data inserts and isn't a practical engine for any form of database access. It is, however, a good guide to how to develop your own storage engine, and is therefore an effective guide for programmers.

EXAMPLE Summary
Name EXAMPLE
Introduced v4.1.3
Default install No
Data limitations N/A
Index limitations N/A
Transaction N/A
Locking level N/A

FEDERATED

The FEDERATED storage engine (added in MySQL 5.03) enables you to access data from remote MySQL database (other databases may be supported in the future) as if it were a local database. In effect, the MySQL server acts as a proxy to the remote server, using the MySQL client access library to connect to the remote host, execute queries and then reformat the data into the localized format.

In essence, it is a way for a server, rather than a client, to access a remote database

and can be an effective way of combining data from multiple hosts or of copying specific data from remote databases into local tables without the use of data exports and imports.

FEDERATED Summary
Name FEDERATED
Introduced v5.0
Default install No
Data limitations Limited by remote database
Index limitations N/A
Transaction No
Locking level No

ARCHIVE

The ARCHIVE storage engine supports only the INSERT and SELECT statements, but does support most of the MySQL field types. Information stored in an ARCHIVE storage engine table is compressed and cannot be modified and so ARCHIVE tables are perfect for storing log data (which you don't want to be able to change) or information that is no longer in active use (for example, old invoicing or sales data).

While the information is stored very efficient, care should be taken when accessing data stored in the ARCHIVE tables. Because the information is compressed, selects have to read the entire table, and that also means decompressing the information. This can obviously increase the time taken to perform complex searches and retrievals. If you are performing a large number of queries on the information in these tables it may be easier to temporarily copy your data to another, uncompressed, data type such as MyISAM.

ARCHIVE Summary
Name ARCHIVE
Introduced v4.1.3
Default install No
Data limitations Data can only be inserted (no updates)
Index limitations N/A
Transaction No
Locking level N/A

CSV

The CSV storage engine stores data not in a binary format, but in the form a CSV (Command Separated Values) file. Because of this, there are limitations to the data stored. It is not an efficient method for storing large volumes of data, or larger data types like BLOB, although such types are supported. There is also no indexing. However, because the data is stored in the CSV format it is exceedingly portable; these CSV files generated can easily be imported into many different software packages, including Excel, OpenOffice and database systems like Access or FileMaker.

In general, the CSV engine is impractical as a general database engine. It is, however, probably the most effective and easiest method for data exchange. What makes it so convenient is that we can use SELECT and INSERT statements to create the database, which in turn means that we can easily produce CSV files based on queries of other data.

With some careful work, the CSV storage engine can also be used as an effective way of getting information into MySQL. Here, you can create the tables first, shutdown the MySQL server, copy over CSV files that you have exported from Excel, Access or another database, and you can then import the data and copy it over to MyISAM or InnoDB tables.

CSV Summary
Name CSV
Introduced v4.1.4
Default install No
Data limitations None
Index limitations Indexing is not supported
Transaction No
Locking level Table

BLACKHOLE

Strange though it may seem, the BLACKHOLE engine does not actually store any data. Although you can create tables and indexes, all SQL statements that would add or update information to the database are executed without actually writing any data. The database structure is retained, however, and you can create any indexes on the (non-existent) information that you want.

Although this seems like a futile exercise, it does allow you to test out database structures and play with table definitions without actually creating any data. Even more useful, however, is that SQL statements on BLACKHOLE databases are written to the binary log, and therefore are replicated to slave databases.

You can use this functionality to update one or more slaves directly without writing any local data. There are a number of potential uses for this functionality. One such use I have employed in past is to write log data to a BLACKHOLE table, which is then echoed to two slaves. Because the write is instantaneous (there are no local disk files or indexes to update), I can maintain a high logging rate, and rely on the binary logging and slave replication to distribute the data. An extension of this, as suggested in the MySQL manual, is to use filtering to control the distribution of records to the slaves.

BLACKHOLE Summary
Name BLACKHOLE
Introduced 4.1.11
Default install No
Data limitations No data is stored, but statements are written to the binary log (and therefore distributed to slave databases)
Index limitations N/A
Transaction No
Locking level N/A

ISAM

The ISAM storage engine was the original engine type available with versions of MySQL up until MySQL 3.23, when the MyISAM storage engine was introduced. ISAM has a number of different limitations that make it impractical as a database engine. These include the storage format, which is native to the platform (and therefore not portable between systems), a maximum table size of just 4GB and limited text searching facilities. Indexes are also more limited. Since MyISAM is supported on the same platforms as ISAM, and provides better compatibility, portability and performance.

ISAM is included for backwards compatibility, you certainly shouldn't use ISAM for new databases, use MyISAM instead.

ISAM Summary

Name ISAM
Introduced v1.0
Default install Yes
Data limitations Limited maximum database size (4GB)
Index limitations Maximum 16 indexes per table, 16 parts per key
Transaction No
Locking level Table

Berkeley DB (BDB)

The Berkeley DB (or BDB) engine is based on the technology provided by the Berkeley DB storage system developed by SleepyCat software. BDB is a hash based storage mechanism, and the keys to the hash values are stored very efficiently. This makes the recovery of information--especially when accessed directly using a unique key incredibly quick, and by far the quickest of the available database types. Recovering full records is even quicker if you the data is short enough to be stored with the unique key (i.e., under 1024 bytes long). BDB is also one of only two types of storage engine that support transactions.

BDB is, however, limited in other ways. Although it uses page locking, locking only 8192 bytes of a table, rather than the entire table, during an update this can cause problems if you are performing a large number of updates in the same page (for example, inserting many rows). There is unfortunately no way round this. Sequential data access--for example a large quantity of rows matching non-indexed data--can be a lot slower because the data needs to be scanned row by row.

Recovery of information with BDB tables can also be a problem. Data in BDB is stored in a combination of the key index, the data file and binary data logs. A loss of data in any of these sections, even just one of the data logs, can make the data in the database totally unrecoverable.

Where BDB shines therefore is in locations where you can access specific blocks of data by a unique key that does not frequently change. I've successfully used BDB tables in the past to store look up information for data like categories or option lists where the small size and unique key structure make it quick and easy to recover information that is not often changed from its initial definition.

Berkeley DB (BDB) Summary
Name BDB

Introduced v3.23.34a
Default install No
Data limitations None
Index limitations Max 31 indexes per table, 16 columns per index;max key size 1024 bytes
Transaction Yes
Locking level Page (8192 bytes)

InnoDB

The InnoDB Engine is provided by Innobase Oy and supports all of the database functionality (and more) of MyISAM engine and also adds full transaction capabilities (with full ACID (Atomicity, Consistency, Isolation, and Durability) compliance) and row level locking of data.

The key to the InnoDB system is a database, caching and indexing structure where both indexes and data are cached in memory as well as being stored on disk. This enables very fast recovery, and works even on very large data sets. By supporting row level locking, you can add data to an InnoDB table without the engine locking the table with each insert and this speeds up both the recovery and storage of information in the database.

As with MyISAM, there are few data types that cannot effectively be stored in an InnoDB database. In fact, there are no significant reasons why you shouldn't always use an InnoDB database. The management overhead for InnoDB is slightly more onerous, and getting the optimization right for the sizes of in-memory and on disk caches and database files can be complex at first. However, it also means that you get more flexibility over these values and once set, the performance benefits can easily outweigh the initial time spent. Alternatively, you can let MySQL manage this automatically for you.

If you are willing (and able) to configure the InnoDB settings for your server, then I would recommend that you spend the time to optimize your server configuration and then use the InnoDB engine as the default.

InnoDB Summary
Name InnoDB
Introduced v3.23 (source only), v4.0 (source and binary)
Default install No

Data limitations None
Index limitations None
Transaction support Yes (ACID compliant)
Locking level Row

Summary

As you may have been able to conclude from the above summary of the different storage engines available, there are few reasons not to use either the MyISAM or InnoDB engine types. MyISAM will do in most situations, but if you have a high number of updates or inserts compared to your searches and selects then you will get better performance out of the InnoDB engine. To get the best performance out of InnoDB you need to tweak the parameters for your server, otherwise there is no reason not to use it.

But if both MyISAM and InnoDB are so great, why even consider using the other engine types! Simply because they provide specific functionality that is not otherwise available. The MERGE engine is an exceedingly effective way of querying data from multiple, identically defined, tables. The MEMORY engine is the best way to perform a large number of complex queries on data that would be inefficient to search on a disk based engine. The CSV engine is a great way to export data that could be used in other applications. BDB is excellent for data that has a unique key that is frequently accessed.

Some of these are possible to do with InnoDB (our separate MyISAM logs, for example, could be combined into a single InnoDB table), but the flexibility to choose an engine type that suits you and the data you are working with is what differentiates MySQL from other solutions.

Source: http://www.go4expert.com/articles/mysql-storage-engines-t2727/