
Inside MySQL 5.1 A DBA's Perspective

A MySQL® Technical White Paper

Table of Contents

1	Executive Summary	3
2	Overview of the MySQL Database Server	3
2.1	The MySQL Pluggable Storage Engine Architecture in 5.1	3
3	What's New in MySQL 5.1?	5
3.1	Data Warehousing and Business Intelligence	5
3.1.1	Table and Index Partitioning	5
3.1.2	Full Text Search Enhancements	7
3.1.3	XML/XPath Support	8
3.1.4	Archive Engine Enhancements	9
3.2	High Availability	9
3.2.1	Row-based and Hybrid Replication	9
3.3	Easier Manageability	10
3.3.1	Task/Event Scheduler	10
3.3.2	Dynamic General and Slow Query Logs	12
3.3.3	CSV Storage Engine	12
3.3.4	New Information Schema Objects	14
3.3.5	Return of libmysqld	14
3.4	Higher Performance	14
3.4.1	Parallel Data Import	14
3.4.2	Better Session and Problem SQL Identification	14
3.4.3	New Performance/Load Testing Utility	16
3.4.4	SQL Profiling Utility	16
4	Conclusion	17

1 Executive Summary

MySQL 5.1 continues the momentum of the ground-breaking 5.0 release that occurred in October 2005. Version 5.0 of MySQL greatly accelerated MySQL's push into modern enterprises and Web 2.0 companies because of the inclusion of important key features like stored procedures, triggers, views, the archive storage engine, and more. Now, with the release of 5.1, the MySQL database server provides more enterprise-caliber enhancements that greatly assist those wanting to use MySQL for data warehousing and business intelligence, applications having extreme high-availability requirements, and systems needing a powerful but autonomous-running database that requires little attention to continually service thousands of user requests per second.

This paper first provides a technical overview of MySQL and then takes a detailed look at each of the new features in version 5.1. Each enhancement demonstrates why the latest version of MySQL is the perfect choice for IT professionals who want to benefit from the flexibility and cost savings of open source software, but not sacrifice anything relating to database reliability, performance, and ease-of-use. Whether it's demanding enterprise transactional processing systems, large data warehouse applications, or high-volume traditional or Web 2.0 web sites, version 5.1 firmly keeps MySQL in its number-one spot as the most popular open source database in the world.

2 Overview of the MySQL Database Server

The MySQL database server is the most trusted and depended-on open source database platform in use today by modern enterprises who are creating applications that serve today's generation of customer who demands constant attention and access to their information. Whether it's a Web 2.0 application that grows through the contributions of its active user base, always-online telecommunications systems that cannot tolerate any downtime whatsoever, or hosted financial/CRM applications that manage critical data used for making key business decisions around-the-clock, demand for information is hotter than ever, with data being used in more creative ways than ever before.

Open source software is a key driver for many of these businesses, with the compelling combination of strong features and inexpensive financial commitment driving many to adopt full or partial open source stacks to pilot key systems. Of course, the database is the heart of any serious application, and all businesses rightfully treat the data that is in them as the crown jewels of their organization. Naturally, smart businesses don't just trust anyone to manage this critical asset, but instead they turn to proven leaders who offer technology that's reliable, high-performing, and easy-to-use.

These three attributes perfectly describe the MySQL database server. Reliability, high performance, and easy setup and maintenance. Backing up these priorities is an architecture that is unique in the database industry – the pluggable storage engine architecture.

2.1 The MySQL Pluggable Storage Engine Architecture in 5.1

Many large enterprises are choosing MySQL because it offers a new and different paradigm of database management. A key differentiator between MySQL and other database platforms – whether they are proprietary or open source – is the pluggable storage engine architecture of MySQL.

The MySQL pluggable storage engine architecture allows a database professional to select a specialized storage engine for a particular application need while being completely shielded from the need to manage any specific application coding requirements. Graphically depicted, the MySQL pluggable storage engine architecture looks like Figure 1:

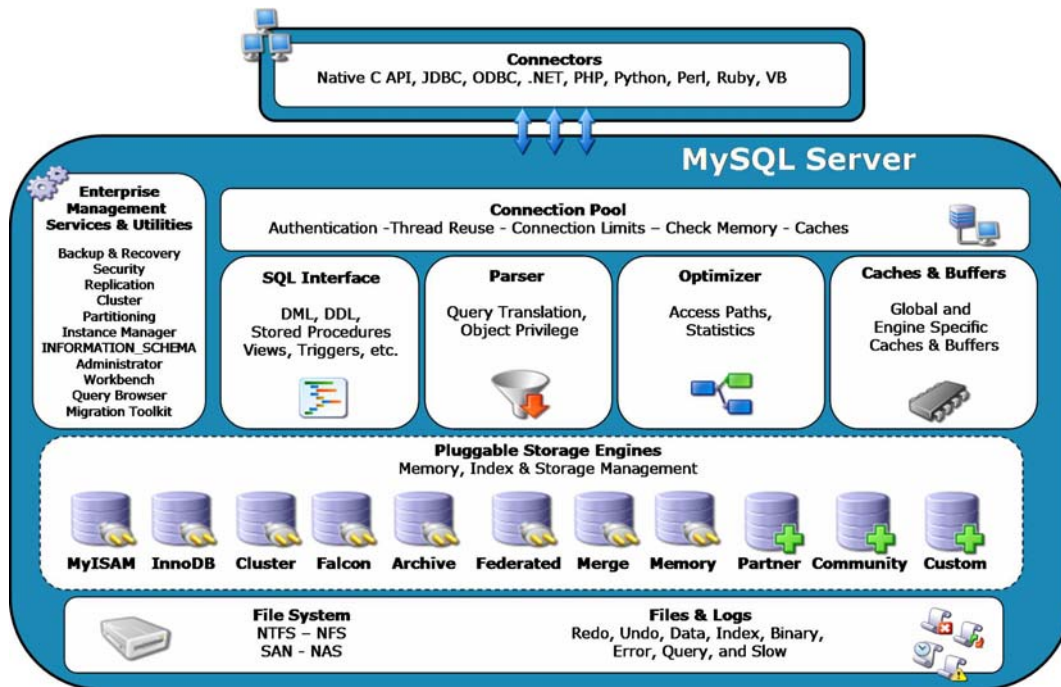


Figure 1 - The MySQL Pluggable Storage Architecture

The pluggable storage engine architecture provides a standard set of management and support services that are common among all underlying storage engines. The storage engines themselves are the components of the database server that actually perform actions on the underlying data that is maintained at the physical server level.

This efficient and modular architecture provides large performance and manageability benefits for those wishing to specifically target a particular application need – such as data warehousing, transaction processing, high availability situations, etc. – while enjoying the advantage of utilizing a set of interfaces and services that are independent of any one storage engine.

From a technical perspective some of the key differentiations in storage engines include:

- **Concurrency/Locking** – some applications have more granular lock requirements (such as row-level locks) than others. Choosing the right locking strategy can reduce overhead and therefore help with overall performance. This area also includes support for capabilities like multi-version concurrency control or “snapshot” read.
- **Transaction Support** – not every application needs transactions, and for those that don’t, the overhead of such support is avoided. But for those that do, there are very well defined requirements in a number of MySQL’s storage engines like ACID compliance and more.
- **Physical Storage** – this involves everything from the overall page size for tables and indexes as well as the format used for storing data to physical disk. Different storage strategies definitely have an impact on the overall performance of both read and write operations.
- **Index Support** – different application scenarios tend to benefit from different index strategies, and so each storage engine generally has its own indexing methods, although some indexing mechanisms (like B-tree indexes) are common to nearly all engines.
- **Memory Caches** – MySQL offers a couple of storage engines whose data only resides in RAM, which results in very fast performance. Others use different variations of memory caches for transactions, transaction logs, tables, and indexes.

- **Performance Aids** – includes things like multiple I/O threads for parallel operations, thread concurrency, database checkpointing, bulk insert handling, and more.

Each set of the pluggable storage engine infrastructure components are designed to offer a selective set of benefits for a particular application. Conversely, avoiding a set of component features helps steer clear of unnecessary overhead. So it stands to reason that understanding a particular application's set of requirements and selecting the proper MySQL storage engine can have a dramatic impact on overall system efficiency and performance. And a DBA can put multiple storage engines in play for the same application, which equates to the DBA having an extremely flexible and high-performance framework at their disposal for satisfying the exact requirements of an application.

Beginning in MySQL 5.1, a storage engine plug-in interface is provided to dynamically and easily install/uninstall storage engines from the MySQL Server. Before a storage engine can be used, the storage engine plugin shared library must be loaded into MySQL using the `INSTALL PLUGIN` statement. For example, if the `EXAMPLE` engine plugin is named `ha_example` and the shared library is named `ha_example.so`, a DBA would load the engine into MySQL with the following statement:

```
INSTALL PLUGIN ha_example SONAME 'ha_example.so';
```

To unplug a storage engine from the MySQL Server, the `UNINSTALL PLUGIN` statement is used:

```
UNINSTALL PLUGIN ha_example;
```

3 What's New in MySQL 5.1?

MySQL offers a number of new features in version 5.1 that directly target the needs of those building data warehouses, Web 2.0, and other heavy-duty Enterprise applications. Enhancements in the following areas will be covered in detail in the sections that follow:

- Data Warehousing and Business Intelligence
- High Availability
- Easier Manageability
- Higher Performance

3.1 Data Warehousing and Business Intelligence

The following new features can be used by any serious user of MySQL, however they will especially appeal to those using MySQL for data warehousing and/or business intelligence purposes.

3.1.1 Table and Index Partitioning

One of the great new features in MySQL 5.1 is support for horizontal data partitioning in both tables and indexes. The really good news about MySQL and the new 5.1 partitioning feature is all the major forms of partitioning are supported across all major MySQL storage engines:

1. **Range** – this partitioning mode allows a DBA to specify various ranges for which data is assigned. For example, a DBA may create a partitioned table that is segmented by three partitions that contain data for the 1980's, 1990's, and everything beyond and including the year 2000.
2. **Hash** – this partitioning mode allows a DBA to separate data based on a computed hash key that is defined on one or more table columns, with the end goal being an equal distribution of values

among partitions. For example, a DBA may create a partitioned table that has ten partitions that are based on the table's primary key.

3. **Key** – a special form of Hash where MySQL guarantees even distribution of data through a system-generated hash key.
4. **List** – this partitioning mode allows a DBA to segment data based on a pre-defined list of values that the DBA specifies. For example, a DBA may create a partitioned table that contains three partitions based on the years 2004, 2005, and 2006.
5. **Composite** – this final partitioning mode allows a DBA to perform sub-partitioning where a table is initially partitioned by, for example range partitioning, but then each partition is segmented even further by another method (for example, hash).

There are a number of benefits that come with partitioning, but the two main advantages are:

- **Increased performance** – during scan operations, the MySQL optimizer knows what partitions contain the data that will satisfy a particular query and will access only those necessary partitions during query execution. This ability is called “partition pruning”. For example, a million row table may be broken up into ten different partitions in range style so that each partition contains 100,000 rows. If a query is issued that only needs data from one of the partitions, and a table scan operation is necessary, only 100,000 rows will be accessed instead of a million. Obviously, it is much quicker for MySQL to sample 100,000 rows than one million so the query will complete much sooner. The same benefit is derived when indexed access is possible, because local partitioned indexes are created for partitioned tables. Finally, it is possible to stripe a partitioned table across different physical drives by specifying different file system/directory paths for specific partitions. This allows physical I/O contention to be reduced when multiple partitions are accessed at the same time.
- **Simplified data management** – partitioning allows a DBA to have more control over how data is managed inside of the database. By intelligently creating partitions, a DBA can simplify how certain data operations are performed. For example, a DBA can drop specific partitions in a partitioned table while the remaining partitions remain intact (as opposed to crafting a fragmentation-producing mass delete operation for the whole table). Further, partitions are automatically maintained by MySQL so the DBA doesn't have to manually separate and maintain a horizontal partitioning scheme for a table. For example, a DBA can create a history table that holds data for customers that are partitioned across various year ranges, and have those partitions automatically enforced by the database server with no DBA intervention being necessary.

An example of creating a range-based partitioned table would be:

```
mysql> CREATE TABLE part_tab
->   ( c1 int default NULL,
->   c2 varchar(30) default NULL,
->   c3 date default NULL
->
->   ) engine=mysam
->   PARTITION BY RANGE (year(c3)) (PARTITION p0 VALUES LESS THAN (1995),
->   PARTITION p1 VALUES LESS THAN (1996) , PARTITION p2 VALUES LESS THAN (1997) ,
->   PARTITION p3 VALUES LESS THAN (1998) , PARTITION p4 VALUES LESS THAN (1999) ,
->   PARTITION p5 VALUES LESS THAN (2000) , PARTITION p6 VALUES LESS THAN (2001) ,
->   PARTITION p7 VALUES LESS THAN (2002) , PARTITION p8 VALUES LESS THAN (2003) ,
->   PARTITION p9 VALUES LESS THAN (2004) , PARTITION p10 VALUES LESS THAN (2010),
->   PARTITION p11 VALUES LESS THAN MAXVALUE );
Query OK, 0 rows affected (0.00 sec)
```

Other items of interest regarding partitioning in MySQL 5.1 include:

- All major storage engines support partitioning (MyISAM, Archive, InnoDB, etc.)
- Indexing support for partitioned tables include local indexes, which mirror each partition in a one-to-one scheme. In other words, if a partitioned table has ten partitions, then a local index for that table would also contain ten partitions.

- Metadata regarding partitioned tables can be found in the INFORMATION_SCHEMA database, with a new PARTITIONS table being available.
- All SHOW commands support the return of partitioned table and index metadata.
- Maintenance functions and a number of other operations can be performed on partitions (rather than acting on a full table), including:
 - ADD PARTITION
 - DROP PARTITION
 - COALESCE PARTITION
 - REORGANIZE PARTITION
 - ANALYZE PARTITION
 - CHECK PARTITION
 - OPTIMIZE PARTITION
 - REBUILD PARTITION
 - REPAIR PARTITION

More information on MySQL partitioning can be found in the online reference manual at <http://dev.mysql.com/doc/refman/5.1/en/partitioning.html> as well as in the MySQL forums at <http://forums.mysql.com/list.php?106>.

3.1.2 Full Text Search Enhancements

As has already been mentioned, MySQL 5.1 adds support for a very flexible plugin API that enables loading and unloading of various components at runtime, without restarting the server. This includes not only storage engines, but full-text parsers as well. This new capability allows users to implement their own input filter on indexed text, enabling full-text search capability on arbitrary data such as PDF files or other document formats. A pre-parser full-text plugin performs the actual parsing and extraction of the text and hands it over to the built-in MySQL full-text search.

MySQL has a built-in parser that it uses by default for full-text operations (parsing text to be indexed, or parsing a query string to determine the terms to be used for a search). When parsing for indexing purposes, the parser passes each word to the server, which adds it to a full-text index. When parsing a query string, the parser passes each word to the server, which accumulates the words for use in a search.

The new plugin API enables a developer to provide a full-text parser of their own so that they have control over the basic duties of a parser. A parser plugin can operate in either of two roles:

1. The plugin can replace the built-in parser. In this role, the plugin reads the input to be parsed, splits it up into words, and passes the words to the server (either for indexing or for word accumulation). One reason to use a parser this way is that a developer needs to use different rules from those of the built-in parser for determining how to split up input into words. For example, the built-in parser considers the text “case-sensitive” to consist of two words “case” and “sensitive,” whereas an application might need to treat the text as a single word.
2. The plugin can act in conjunction with the built-in parser by serving as a front end for it. In this role, the plugin extracts text from the input and passes the text to the parser, which splits up the text into words using its normal parsing rules. In particular, this parsing will be affected by the ft_XXX system variables and the MySQL stopword list. One reason to use a parser this way is that a developer needs to index content such as PDF documents, XML documents, or MS .doc files. The built-in parser is not intended for those types of input but a plugin can pull out the text from these input sources and pass it to the built-in parser.

It is also possible for a parser plugin to operate in both roles. That is, it could extract text from non-plaintext input (the front end role), and also parse the text into words (thus replacing the built-in parser).

A full-text plugin is associated with full-text indexes on a per-index basis. That is, when you install a parser plugin initially, that does not cause it to be used for any full-text operations. It simply becomes available. For example, a full-text parser plugin becomes available to be named in a WITH PARSER clause when creating individual FULLTEXT indexes. The following could be used to create such an index at table-creation time:

```
CREATE TABLE mytable
(
  doc CHAR(255),
  FULLTEXT INDEX (doc) WITH PARSER my_parser
);
```

Or a developer can add the index after the table has been created:

```
ALTER TABLE mytable ADD FULLTEXT INDEX (doc) WITH PARSER my_parser;
```

The only SQL change for associating the parser with the index is the WITH PARSER clause.

Searches are specified as before, with no changes needed for queries. When a developer associates a parser plugin with a FULLTEXT index, the plugin is required for using the index. If the parser plugin is dropped, any index associated with it becomes unusable. Any attempt to use it a table for which a plugin is not available results in an error, although DROP TABLE is still possible.

Finally, full-text in 5.1 will see performance increases in many situations due to a new indexing algorithms that have been introduced.

3.1.3 XML/XPath Support

In version 5.1 of MySQL, basic Xpath (XML Path Language) support has been added to help better facilitate searching and managing XML data that is contained in a MySQL database. Two new functions are included for XPath support:

1. EXTRACTVALUE
2. UPDATEXML

The EXTRACTVALUE function assists in searching the contents of an XML document. An example of its use would be:

```
mysql> SELECT EXTRACTVALUE(doc, '/book/author/initial') FROM x;
+-----+
| EXTRACTVALUE(doc, '/book/author/initial') |
+-----+
| CJ                                         |
| J                                         |
+-----+
2 rows in set (0.01 sec)
```

The UPDATEXML function replaces a single portion of a given fragment of an XML markup target with a new XML fragment and then returns the changed XML. An example would be:

```
mysql> SELECT
-> UpdateXML('<a><b>ccc</b><d></d></a>', '/a', '<e>fff</e>') AS val1,
-> UpdateXML('<a><b>ccc</b><d></d></a>', '/b', '<e>fff</e>') AS val2,
-> UpdateXML('<a><b>ccc</b><d></d></a>', '//b', '<e>fff</e>') AS val3,
-> UpdateXML('<a><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val4,
-> UpdateXML('<a><d></d><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val5
```

```
-> \G
```

```
***** 1. row *****
val1: <e>fff</e>
val2: <a><b>ccc</b><d></d></a>
val3: <a><e>fff</e><d></d></a>
val4: <a><b>ccc</b><e>fff</e></a>
val5: <a><d></d><b>ccc</b><d></d></a>
```

More information on MySQL Xpath can be found in the online reference manual at <http://dev.mysql.com/doc/refman/5.1/en/xml-functions.html> as well as in the MySQL forums at <http://forums.mysql.com/list.php?44>.

3.1.4 Archive Engine Enhancements

The Archive storage engine, introduced in MySQL 5.0, is designed to store seldom-referenced or historical data in a very efficient storage footprint. In MySQL 5.1, the Archive engine has been enhanced to have faster I/O operations and lower memory requirements.

In addition, indexing support has been added for auto-increment columns. Finally, custom data directory assignments can now be specified so a DBA can choose exactly what drive or filesystem they want to put archive tables on.

More information on the Archive engine can be found in the online reference manual at <http://dev.mysql.com/doc/refman/5.1/en/archive-storage-engine.html> as well as in the MySQL forums at <http://forums.mysql.com/list.php?112>.

3.2 High Availability

Version 5.1 of MySQL offers a number of improvements for those needing extreme amounts of high availability for their databases, along with those desiring data redundancy, whether it be in the same data center or centers widely dispersed from a geographical standpoint.

3.2.1 Row-based and Hybrid Replication

MySQL is famous for its easy-to-setup and use replication capabilities. Until MySQL 5.1, statement-based replication has been the only mode of replication offered, and it has worked very well for the vast majority of replication needs that exist. There are, however, some replication scenarios where statement-based replication is not the best choice. For example, non-deterministic functions can be problematic in that end results from such functions that are written to a master may not be the same end results that are written to a slave server.

MySQL 5.1 offers two new replication modes in addition to statement-based mode that provide ways to ensure that all data replicated from a master server to one or more slave servers exactly match:

1. **Row-based replication.** Instead of sending the actual SQL statements from the master server to the slave (as statement-based replication does), the master writes events to its binary log that indicate how individual table rows are affected. These are instead sent to the slave server(s) for application. Row-based replication can get around the problem of non-deterministic functions and other like use cases.
2. **Mixed replication.** Both statement-based and row-based replication have their advantages and drawbacks, and because of this, a mixed replication mode is introduced in MySQL 5.1 that allows a best-of-both-worlds approach. MySQL will automatically detect when it is best to

use statement-based replication and when it is best to use row-based replication based on the user operation. Note that this is now the default setting in MySQL 5.1 and above.

Determining if row-based replication is enabled in a version of MySQL is easily done by issuing the following command:

```
mysql> show global variables like 'have_row%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_row_based_replication | YES |
+-----+-----+
1 row in set (0.00 sec)
```

Setting the replication mode is also easily accomplished by issuing commands such as the following:

```
mysql> SET GLOBAL binlog_format = 'STATEMENT';
mysql> SET GLOBAL binlog_format = 'ROW';
mysql> SET GLOBAL binlog_format = 'MIXED';
```

For a comparison of the advantages of statement-based and row-based replication, see the MySQL Manual at: <http://dev.mysql.com/doc/refman/5.1/en/replication-sbr-rbr.html>. For more information on MySQL replication, see the MySQL Manual at: <http://dev.mysql.com/doc/refman/5.1/en/replication.html> or visit the replication forms at: <http://forums.mysql.com/list.php?26>.

3.3 Easier Manageability

MySQL 5.1 introduces a number of new enhancements that make MySQL easier to use and increases the productivity of those working with the database server.

3.3.1 Task/Event Scheduler

One exciting new feature available in MySQL 5.1 is the task or event scheduler. The event scheduler is a built-in scheduling system that allows a DBA or developer to create and run jobs that perform various tasks in the database server. The jobs can be tasks that run only once or on a scheduled, recurring basis. The new event scheduler can many times alleviate the need to use the CRON scheduler on UNIX and Linux or the Task Scheduler on Windows.

A new parameter, `EVENT_SCHEDULER`, controls the enabling/disabling of the event scheduler subsystem in MySQL. Note that this parameter is dynamic and does not require a stop/start of the server. Once enabled, a user can create and manage events as long as they are authorized to do so. Any user with the `SUPER` privilege can manage events, as can anyone that has been granted the `EVENT` privilege. Note that the `EVENT` privilege's scope is at the schema level, so a user can have the ability to create and manage events that address one or more schemas and their objects.

Events are managed via the traditional `CREATE`, `ALTER`, and `DROP` DDL commands. An example of creating an event that optimizes two tables each week at 1AM might be:

```
mysql> CREATE EVENT OPTIMIZE_TABLES
-> ON SCHEDULE EVERY 1 WEEK
-> STARTS '2006-10-05 1:00:00'
-> ON COMPLETION PRESERVE
-> DO
```

```

-> BEGIN
-> OPTIMIZE TABLE test.table1;
-> OPTIMIZE TABLE test.table2;
-> END
-> //
Query OK, 1 row affected (0.01 sec)

```

To track the metadata and execution details of events, a new INFORMATION_SCHEMA object, EVENTS, has been created. The new object has the following structure:

Field	Type	Null
EVENT_CATALOG	varchar(64)	YES
EVENT_SCHEMA	varchar(64)	NO
EVENT_NAME	varchar(64)	NO
DEFINER	varchar(77)	NO
EVENT_BODY	longtext	NO
EVENT_TYPE	varchar(9)	NO
EXECUTE_AT	datetime	YES
INTERVAL_VALUE	varchar(256)	YES
INTERVAL_FIELD	varchar(18)	YES
SQL_MODE	longtext	NO
STARTS	datetime	YES
ENDS	datetime	YES
STATUS	varchar(8)	NO
ON_COMPLETION	varchar(12)	NO
CREATED	datetime	NO
LAST_ALTERED	datetime	NO
LAST_EXECUTED	datetime	YES
EVENT_COMMENT	varchar(64)	NO

You can also obtain metadata about events using two new SHOW commands:

```

mysql> show events like 'OPTIMIZE_TABLES'\G
***** 1. row *****
      Db: test
      Name: OPTIMIZE_TABLES
      Definer: root@%
      Type: RECURRING
      Execute at: NULL
      Interval value: 1
      Interval field: WEEK
      Starts: 2006-10-05 05:00:00
      Ends: NULL
      Status: ENABLED
1 row in set (0.00 sec)

```

```

mysql> show create event OPTIMIZE_TABLES \G
***** 1. row *****
      Event: OPTIMIZE_TABLES
      sql_mode:
      Create Event: CREATE EVENT `OPTIMIZE_TABLES` ON SCHEDULE EVERY 1 WEEK ON COMPLETION
      PRESERVE ENABLE DO BEGIN
      OPTIMIZE TABLE test.table1;
      OPTIMIZE TABLE test.table2;
      END
1 row in set (0.00 sec)

```

For more information about events, please refer to the MySQL 5.1 Reference Manual, <http://dev.mysql.com/doc/refman/5.1/en/events.html> and also the forum dedicated to events at <http://forums.mysql.com/list.php?119>.

3.3.2 Dynamic General and Slow Query Logs

MySQL has provided query diagnostic and tracing support for some time through the general and slow query logs. The general query log traces all SQL statements serviced by a MySQL server, while the slow query log tracks SQL statements that take longer than a specified amount of time or queries that do not use indexes. In earlier versions of MySQL, these logs were not dynamic in nature, in that enabling and disabling their use required the stopping/starting of the server.

In MySQL 5.1, the general and slow query logs can be dynamically enabled and disabled via simple SET commands. This enables a DBA to turn on/off SQL tracing in an ad-hoc fashion and only track the SQL they wish for a specific period of time. For example, to enable tracing of all SQL statements issued to a MySQL server, a DBA would enter:

```
mysql> set global general_log=1;
Query OK, 0 rows affected (0.01 sec)
```

Once a DBA has finished their trace session, they can easily disable SQL auditing by issuing:

```
mysql> set global general_log=0;
Query OK, 0 rows affected (0.01 sec)
```

3.3.3 CSV Storage Engine

In MySQL 5.1, a new engine has been provided that assists with the management and use of delimited data found in common flat files. The CSV (which stands for “Comma Separated Value”) engine was originally debuted in MySQL 5.0, although it wasn’t enabled for most installations and platforms with the release of version 5.0. But with version 5.1, the CSV engine is fully enabled and ready for action. In addition, CSV in 5.1 now supports both quoted and unquoted numeric strings.

The CSV engine provides four basic benefits to those using MySQL:

1. It allows flat files to be referenced via SQL and used alongside other data that has been loaded into MySQL.
2. Editing data stored via the CSV engine can be performed even when the MySQL server is down through standard file editors.
3. Data from any CSV engine file can easily be imported into any standard spreadsheet program (e.g. Microsoft Excel).
4. It allows for the instantaneous loading of massive amounts of data into the MySQL server.

The last point is particularly interesting for those needing to constantly load large amounts of flat file data into MySQL. For example, imagine having the need to load a file containing 5 million records into a MySQL table that uses the MyISAM storage engine:

```
mysql> desc client_detail;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+

```

```
| client_transaction_id | decimal(22,0) | YES | | NULL | | |
| transaction_timestamp | datetime      | YES | | NULL | | |
| transaction_comment   | varchar(30)   | YES | | NULL | | |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.02 sec)
```

```
mysql> load data infile '/usr/local/mysql519/data/gim/flatdata.dat'
-> into table client_detail
-> fields terminated by ',';
Query OK, 5023575 rows affected, 0 warnings (27.38 sec)
Records: 5023575 Deleted: 0 Skipped: 0 Warnings: 0
```

While MyISAM accepts the data pretty quickly, all the data in the flat file can instantly be made available to MySQL in the following manner. First, create a CSV table that mirrors the format of the incoming flat file:

```
mysql> create table client_detail_csv (client_transaction_id decimal(22,0),
-> transaction_timestamp datetime, transaction_comment varchar(30))
-> engine=csv;
Query OK, 0 rows affected (0.01 sec)
```

Then, go to the operating system and simply rename the operating system flat file to the file MySQL created for the new CSV table:

```
[mysql@linux1 ~]$ cd /usr/local/mysql519/data/gim

[mysql@linux1 gim]$ ls -l
total 785848
-rw-rw---- 1 mysql mysql      35 May  1 06:54 client_detail_csv.CSM
-rw-rw---- 1 mysql mysql         0 May  1 06:54 client_detail_csv.CSV
-rw-rw---- 1 mysql mysql    8718 May  1 06:54 client_detail_csv.frm
-rw-rw---- 1 mysql mysql    8718 May  1 05:56 client_detail.frm
-rw-rw---- 1 mysql mysql 221037300 May  1 06:00 client_detail.MYD
-rw-rw---- 1 mysql mysql    1024 May  1 06:00 client_detail.MYI
-rw-rw---- 1 mysql mysql 291367350 May  1 05:55 flatdata.dat

[mysql@linux1 gim]$ mv flatdata.dat client_detail_csv.CSV
```

The rename operation occurs instantly as nothing data-wise changes at the operating system level. At that point, MySQL has all the data available to it:

```
mysql> flush tables;
Query OK, 0 rows affected (0.00 sec)
mysql> select count(*) from client_detail_csv;
+-----+
| count(*) |
+-----+
| 5023575 |
+-----+
1 row in set (16.17 sec)
```

Thus, 5 million records have *instantaneously* been loaded into MySQL. The great thing is the same effect would be experienced if the file contained 10 million, 20 million, or 100 million records.

For more information on the CSV engine, see the MySQL Reference Manual at: <http://dev.mysql.com/doc/refman/5.1/en/csv-storage-engine.html> and the MySQL forums at: <http://forums.mysql.com/list.php?127>.

3.3.4 New Information Schema Objects

The INFORMATION_SCHEMA database in MySQL has been enhanced to include the following new objects in 5.1:

- **EVENTS** – contains information about tasks/jobs defined to the MySQL task scheduler.
- **ENGINES** – contains information about the various engines included with the MySQL server.
- **FILES** – contains information regarding the physical files used by MySQL Cluster (the NDB storage engine).
- **PARTITIONS** – contains information about partitioned tables and indexes.
- **PLUGINS** – contains information about pluggable storage engines currently in the server.
- **PROCESSLIST** – contains dynamic information about sessions currently logged into the server and their activity.
- **SESSION_STATUS and GLOBAL_STATUS** – displays information about MySQL diagnostics for a session or the MySQL server as a whole.
- **SESSION_VARIABLES and GLOBAL_VARIABLES** – displays information about MySQL configuration parms for a session or the MySQL server as a whole.

3.3.5 Return of libmysqld

MySQL 5.1 brings back the ability to deeply embed MySQL in applications such as small devices and public kiosks. In such situations, the embedded MySQL server has the advantage of running faster with MySQL applications since data isn't sent over the network wire; the server library is embedded in the compiled program.

The libmysqld server library is part of the MySQL 5.1 distribution and can be embedded with C, Python, Visual Basic, and other programming languages. For more information on how to use libmysqld, please see the MySQL Reference Manual at: <http://dev.mysql.com/doc/refman/5.1/en/libmysqld.html>.

3.4 Higher Performance

MySQL 5.1 brings to the table a number of performance enhancements that build on the current reputation of MySQL as being one of the fastest database servers available today.

3.4.1 Parallel Data Import

Users of MySQL who must constantly load large volumes of data from flat files into the database server will appreciate the new parallel load capabilities of the mysqlimport utility. In MySQL 5.1, a new --use-threads option allows multiple threads to act upon multiple load files at the same time, thus accomplishing a parallel load. This new option is particularly well suited for MySQL Cluster users needing faster load times.

3.4.2 Better Session and Problem SQL Identification

MySQL 5.1 adds new diagnostic capabilities for those needing to better understand the user activity on their systems, along with what SQL is being issued and could potentially be slowing down the server's overall performance.

User activity can now be reported better through the new PROCESSLIST object that exists in the INFORMATION_SCHEMA database. The structure of PROCESSLIST mirrors the output that is available in the SHOW FULL PROCESSLIST command:

```
mysql> select * from PROCESSLIST\G
***** 1. row *****
  ID: 3
  USER: root
  HOST: 192.168.0.4:2725
  DB: gim
  COMMAND: Sleep
  TIME: 4
  STATE:
  INFO: NULL
***** 2. row *****
  ID: 1
  USER: root
  HOST: 192.168.0.4:2723
  DB: information_schema
  COMMAND: Query
  TIME: 0
  STATE: preparing
  INFO: select * from PROCESSLIST
2 rows in set (0.00 sec)
```

```
mysql> SHOW FULL PROCESSLIST\G
***** 1. row *****
  Id: 1
  User: root
  Host: 192.168.0.4:2723
  db: information_schema
  Command: Query
  Time: 0
  State: NULL
  Info: SHOW FULL PROCESSLIST
***** 2. row *****
  Id: 3
  User: root
  Host: 192.168.0.4:2725
  db: gim
  Command: Sleep
  Time: 20
  State:
  Info: NULL
2 rows in set (0.00 sec)
```

The new PROCESSLIST object makes it easy to exclude/include via SQL only the information about currently connected processes that is desired.

In addition, two new objects in the MYSQL database make it easy to view SQL that has been captured by either the general query or slow query logs. Located in the mysql database, the GENERAL_LOG and SLOW_LOG objects (which use the CSV storage engine) mirror the general and slow query logs that have been available in MySQL for some time, but were previously only available in file format. This sometimes made searching for problem or resource-hungry SQL statements difficult, but in 5.1, it's very easy to identify SQL statements that are tuning candidates. For example, if you want to find the top five longest running SQL statements on a server, you could issue the following SQL:

```
mysql> select substr(sql_text,1,30), query_time, rows_examined
-> from slow_log
-> order by 2 desc limit 5;
```

```

+-----+-----+-----+
| substr(sql_text,1,30)          | query_time | rows_examined |
+-----+-----+-----+
| update t1 set c2=3 where c1=2  | 00:03:01   | 0              |
| insert into part_rms select *  | 00:01:42   | 8100000        |
| insert into part_date1 select  | 00:01:42   | 8100000        |
| INSERT INTO part_rmsj select * | 00:01:29   | 8100000        |
| insert into part_rmsm select * | 00:01:26   | 8100000        |
+-----+-----+-----+
5 rows in set (0.00 sec)

```

The general and slow query logs and accompanying SQL objects are dynamic in that you can turn the tracing of SQL statements on and off at will.

3.4.3 New Performance/Load Testing Utility

One very neglected area in database management is stress/performance/load testing. This discipline involves simulating load on a new or changed database system to ensure it can stand up against the pressure of an enterprise workload with many concurrent users and various types of activity. There are a few free load testing tools available for MySQL, as well as several expensive third-party tools.

In MySQL 5.1, a new utility is included that assists DBAs and developers with stress testing a MySQL server. The *mysqlslap* utility simulates a designated number of users submitting various SQL statements over a specified period of time. Once the load simulation is complete, a diagnostic report is produced with statistics detailing the outcome.

The *mysqlslap* utility is run from the command line and includes a variety of different options that control things such as how many concurrent users to simulate, whether submitted SQL statements should be generated automatically or supplied via a pre-defined SQL file, and much more.

For more information on *mysqlslap*, see the MySQL 5.1 reference manual at: <http://dev.mysql.com/doc/refman/5.1/en/mysqlslap.html>.

3.4.4 SQL Profiling Utility

Previously only included in Community versions of the MySQL Server, MySQL 5.1 now includes a SQL profiling utility in both the Community and Enterprise editions of the server. This interactive diagnostic aid allows a developer or DBA to profile a SQL query/command and gain insight into where the set of SQL code spends its time and the resources it is using. The SQL Profiler is built into the database server and can be dynamically enabled/disabled via the `mysql` client utility. To begin profiling one or more SQL queries, a user simply issues the following command:

```

mysql> set profiling=1;
Query OK, 0 rows affected (0.00 sec)

```

MySQL begins a new profiling session for the current user. At this point, any queries issued will be profiled:

```

mysql> select count(*) from client where broker_id=2;
+-----+
| count(*) |
+-----+
| 200      |
+-----+
1 row in set (0.00 sec)

```

Once the query completes, the following command can be issued to view the SQL profiles that have currently been stored for the session:

```
mysql> show profiles;
+-----+-----+-----+
| Query_ID | Duration | Query |
+-----+-----+-----+
|          0 | 0.00007300 | set profiling=1 |
|          1 | 0.00044700 | select count(*) from client where broker_id=2 |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

Other commands are available, the most basic of which is one that lists the steps a profiled query went through to satisfy a SQL request, along with each step's time:

```
mysql> show profile for query 1;
+-----+-----+
| Status | Duration |
+-----+-----+
| (initialization) | 0.00006300 |
| Opening tables | 0.00001400 |
| System lock | 0.00000600 |
| Table lock | 0.00001000 |
| init | 0.00002200 |
| optimizing | 0.00001100 |
| statistics | 0.00009300 |
| preparing | 0.00001700 |
| executing | 0.00000700 |
| Sending data | 0.00016800 |
| end | 0.00000700 |
| query end | 0.00000500 |
| freeing items | 0.00001200 |
| closing tables | 0.00000800 |
| logging slow query | 0.00000400 |
+-----+-----+
15 rows in set (0.00 sec)
```

For more information on the SQL profiler, please see the MySQL 5.1 reference manual at <http://dev.mysql.com/doc/refman/5.1/en/show-profiles.html>.

4 Conclusion

The MySQL database server continues to set the bar for open source database reliability, performance, and ease of use. Version 5.1 of MySQL provides more benefits for those wishing to utilize MySQL in large, enterprise-wide applications that have business intelligence, high availability, and high performance requirements. In addition, MySQL 5.1 makes the MySQL database server easier to use and administer than ever before.

The combination of a rich and robust feature set, open source flexibility, and a total cost of ownership that can't be beat make MySQL the natural choice for all modern enterprises needing a superior data management system on which to run their business.