MySQL 5.0's Pluggable Storage Engine Architecture

A MySQL® White Paper

October 16, 2005

Copyright © 2005, MySQL AB

Table of Contents

Introduction 3
Overview of the MySQL Pluggable Storage Engine Architecture.....3
The Common MySQL Database Server Layer.....5
Comparing Different Storage Engines.....6
A Quick Test Drive of the Pluggable Storage Engine Architecture.....8
The Impact of "Unplugging" a Storage Engine 12
Creating Your Own Storage Engine.....14
Conclusion.....14
About MySQL 14

Copyright © 2005, MySQL AB

Introduction

The reasons for the huge popularity and increasing adoption of MySQL as a serious database platform can many times be boiled down to two, but surprisingly contrasting, considerations:

1. How much MySQL is like other database platforms.
2. How much MySQL is different than other database platforms.

Of course, many companies are turning to MySQL because of the database server's high performance, rock-solid reliability, and uncomplicated mode of operation. But in addition, modern enterprises migrating from proprietary databases to MySQL find the transition very easy as MySQL sports ANSI standard SQL, a familiar stored procedure/function language, standard relational tables and indexes, and many other features that resemble most standard relational database characteristics. In this fashion, MySQL mirrors other proprietary databases, which means that the migration from other systems and the ramp-up time of training database personnel in MySQL is very quick and painless.

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

What exactly is the MySQL pluggable storage engine architecture and what benefits does it offer to today's modern enterprises? This paper addresses these questions and more by outlining what the MySQL pluggable storage engine architecture is, giving practical examples of how it works, and showcasing the many benefits that come from using it.

Overview of the MySQL Pluggable Storage Engine Architecture

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. The MySQL server architecture encapsulates the application programmer and DBA from all of the low-level implementation details at the storage level providing a consistent and easy application model and API. So while there are different capabilities across different storage engines, the application is shielded from these.

Graphically depicted, the MySQL pluggable storage engine architecture looks like Figure 1 on the next page.

Copyright © 2005, MySQL AB

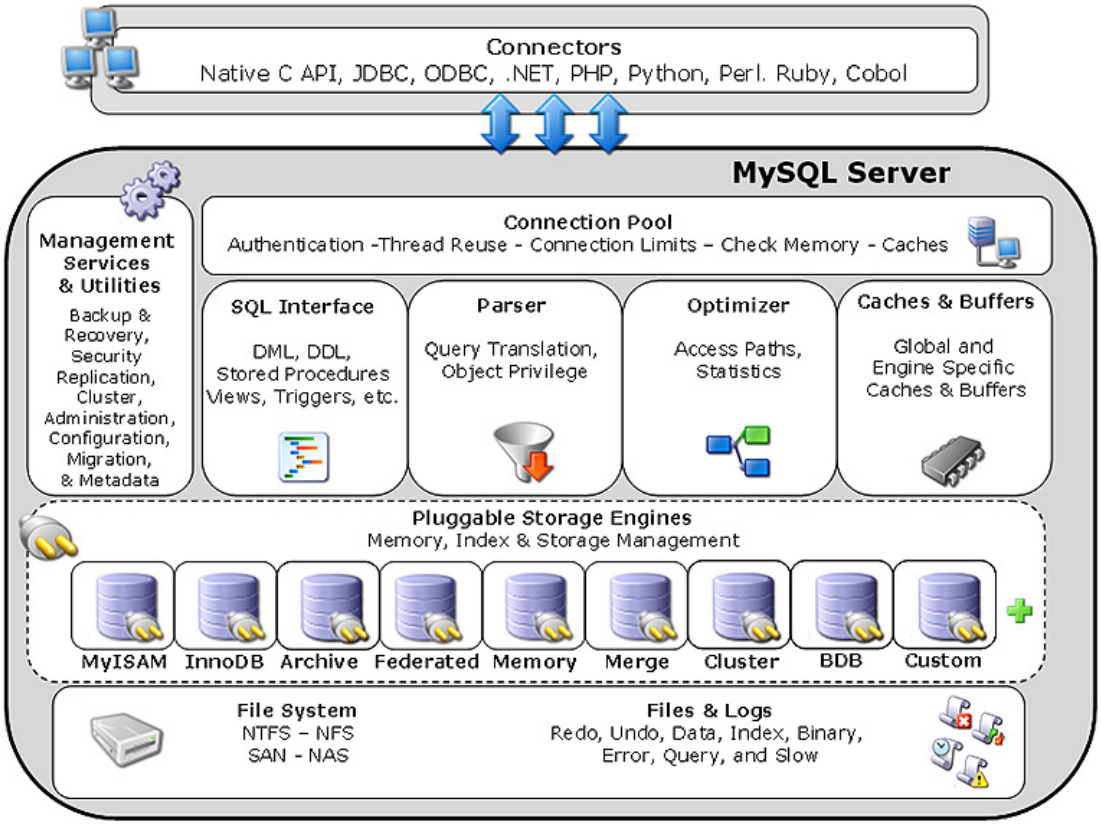


Figure 1 – MySQL Pluggable Storage Engine Architecture is both flexible and modular

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 huge 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.

The application programmer and DBA interact with the MySQL database through Connector APIs and service layers that are above the storage engines. If application changes bring about requirements that demand the underlying storage engine change, or that one or more additional storage engines be added to support new needs, no significant coding or process changes are required to make things work. The MySQL server architecture shields the application from the underlying complexity of the storage engine by presenting a consistent and easy to use API that applies across storage engines.

Let's first take a look at the common layer of the MySQL database server and then examine what a storage engine actually is and how they compare to one another in terms of functionality and use.

Copyright © 2005, MySQL AB

The Common MySQL Database Server Layer

A MySQL pluggable storage engine is the component in the MySQL database server that is responsible for performing the actual data I/O operations for a database as well as enabling and enforcing certain feature sets that target a specific application need. A major benefit of using specific storage engines is that you are only delivered the features needed for a particular application, and therefore you have less system overhead in the database, with the end result being more efficient and higher database performance. This is one of the reasons that MySQL has always been known to have such high performance, matching or beating proprietary monolithic databases in industry standard benchmarks.

From a technical perspective, what are some of the unique supporting infrastructure components that are in a storage engine? Some of the key differentiations include:

- **Concurrency** – 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, but for those that do, there are very well defined requirements like ACID compliance and more.
- **Referential Integrity** – the need to have the server enforce relational database referential integrity through DDL defined foreign keys.
- **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.
- **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 (like B-tree indexes) are common to nearly all engines.
- **Memory Caches** – different applications respond better to some memory caching strategies than others, so while some memory caches are common to all storage engines (like those used for user connections, MySQL's high-speed Query Cache, etc.), others are uniquely defined only when a particular storage engine is put in play.
- **Performance Aids** – includes things like multiple I/O threads for parallel operations, thread concurrency, database checkpointing, bulk insert handling, and more.
- **Miscellaneous Target Features** – this may include things like support for geospatial operations, security restrictions for certain data manipulation operations, and other like items.

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.

Let's now take a look at some of the more prominently used MySQL storage engines and how they compare to one another to understand the benefits each provides.

Copyright © 2005, MySQL AB

Comparing Different Storage Engines

As can be seen in Figure 1, there are a number of MySQL pluggable storage engines that can be used with the MySQL database server. Some of the more common engines include:

- **MyISAM** – the default MySQL pluggable storage engine and the one that is used the most in Web, data warehousing, and other application environments. Note that a MySQL server's default storage engine can easily be changed by altering the STORAGE_ENGINE configuration variable.
- **InnoDB** – used for transaction processing applications, and sports a number of features including ACID transaction support.
- **BDB** – an alternative transaction engine to InnoDB that supports COMMIT, ROLLBACK, and other transactional features.
- **Memory** – stores all data in RAM for extremely fast access in environments that require quick look ups of reference and other like data.
- **Merge** – allows a MySQL DBA or developer to logically group together a series of identical MyISAM tables and reference them as one object. Good for VLDB environments like data warehousing.
- **Archive** – provides the perfect solution for storing and retrieving large amounts of seldom-referenced historical, archived, or security audit information.
- **Federated** – offers the ability to link together separate MySQL servers to create one logical database from many physical servers. Very good for distributed or data mart environments.
- **Cluster/NDB** – the Clustered database engine of MySQL that is particularly suited for applications with high performance lookup needs that also require the highest possible degree of uptime and availability.
- **Other** – other storage engines include CSV (references comma-separated files as database tables), Blackhole (for temporarily disabling application input to the database) and an Example engine that helps jump start the process of creating custom pluggable storage engines.

While the above brief descriptions will give you a general idea of what type of application might benefit from a particular storage engine, a more detailed look at various common database tasks and needs across the various engines may help delineate the differences a little more. Keep in mind that the grid on the next page is not exhaustive by any means; for a more detailed analysis of each storage engine's feature set, please see the MySQL Reference Manual.

Copyright © 2005, MySQL AB

Feature	MyISAM	BDB	Memory	InnoDB	Archive	NDB
Storage Limits	No	No	Yes	64TB	No	Yes
Transactions (commit, rollback, etc.)		✔		✔		✔
Locking granularity	Table	Page	Table	Row	Row	Row
MVCC/Snapshot Read				✔	✔	✔
Geospatial support	✔					
B-Tree indexes	✔	✔	✔	✔		✔
Hash indexes			✔	✔		✔
Full text search index	✔					
Clustered index				✔		
Data Caches			✔	✔		✔
Index Caches	✔		✔	✔		✔
Compressed data	✔				✔	
Encrypted data (via function)	✔	✔	✔	✔	✔	✔
Storage cost (space used)	Low	Low	N/A	High	Very Low	Low
Memory cost	Low	Low	Medium	High	Low	High
Bulk Insert Speed	High	High	High	Low	Very High	High
Cluster database support						✔
Replication support	✔	✔	✔	✔	✔	✔
Foreign key support				✔		
Backup/Point-in-time recovery	✔	✔	✔	✔	✔	✔
Query cache support	✔	✔	✔	✔	✔	✔
Update Statistics for Data Dictionary	✔	✔	✔	✔	✔	✔

Of course, you can use multiple storage engines in a single application; you are not limited to using only one storage engine in a particular database. So, you can easily mix and match storage engines for the given application need. This is often the best way to achieve optimal performance for truly demanding applications: use the right storage engine for the right job.

Because you have such flexibility and choice with MySQL, you should carefully weigh your application's requirements before selecting a particular storage engine for use. For example, let's say you have a new world-wide Business Intelligence (BI) application with the following requirements:

- Heavy amounts of nightly data loads with a small time window for the loads to complete.
- Other than load activity, the only other operations are read-only in nature, which implies no need for high levels of data concurrency as only shared locks are used for reads.
- Application will be hosted from a web interface that requires full-text search capabilities.
- Transaction support is not necessary
- Data referential integrity is assured via the source transactional system.
- Data will be replicated to various geographical sites for performance benefits.

For this application, the natural choice would likely be MyISAM. Again, however, there may be more detailed parts of the application that are suited for a different storage engine. For example, maybe the BI application described above has both current and seldom-referenced historical data that must be kept online for government compliance reasons. In that case, a mixture of the MyISAM and Archive storage engines would be recommended.

Copyright © 2005, MySQL AB

Let's now examine the MySQL pluggable storage engine architecture in action to see how easy it is to make use of it and switch between various storage engines during an application design and testing phase.

A Quick Test Drive of the Pluggable Storage Engine Architecture

Let's take a quick test drive to show how simple it is to use MySQL's pluggable storage engine architecture and see the impact different storage engines can have on performance. All tests below were conducted on a Dell Red Hat Fedora Core 4 box with a Pentium 4 3.00 GHz processor (hyperthreading enabled) and 1GB of RAM.

For example, let's say you have a very insert-intensive/logging application that requires the absolute fastest response times possible when it comes to handling incoming insert activity. Reads of loaded data for analysis purposes is also required.

You want to test out different scenarios and find the best possible solution, so you settle on a simple iterative test of inserting three million sample logging records into a table to test insert load speed.

First, because you have some legacy systems on another database, you decide to first give it a try and see how well it performs for your simple test. You create the necessary test table and procedure, and then execute your test:

```
SQL> desc test_insert;
 Name                                      Null?    Type
 ----------------------------------------- -------- --------------------
 C1                                                 NUMBER(38)
 C2                                                 VARCHAR2(20)
 C3                                                 DATE

SQL> CREATE OR REPLACE PROCEDURE P_TEST_INSERT
  2  AS
  3      v_CTR NUMBER;
  4  BEGIN
  5
  6  --  start loop for insert
  7
  8      FOR CTR IN 1..3000000 LOOP
  9
 10          INSERT INTO TEST_INSERT VALUES (1, 'sample audit string',sysdate);
 11
 12      END LOOP;
 13
 14  END;
 15  /

Procedure created.

SQL> set timing on;
SQL> exec p_test_insert;

Elapsed: 00:03:03.01
SQL>
```

The legacy database doesn't do too bad and comes in at about one million records a minute. You now decide to try MySQL as you've heard MySQL is used on many high traffic web sites for its ability to handle heavy volumes of insert activity. You first decide to view all the available

Copyright © 2005, MySQL AB

storage engines for MySQL 5.0 by issuing a SHOW ENGINES from the MySQL command line utility:

```
mysql> show engines;
+------------+---------+----------------------------------------------------------------+
| Engine     | Support | Comment                                                        |
+------------+---------+----------------------------------------------------------------+
| MyISAM     | DEFAULT | Default engine as of MySQL 3.23 with great performance         |
| MEMORY     | YES     | Hash based, stored in memory, useful for temporary tables      |
| HEAP       | YES     | Alias for MEMORY                                               |
| 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        | YES     | Supports transactions and page-level locking                   |
| BERKELEYDB | YES     | Alias for BDB                                                  |
| NDBCLUSTER | NO      | Clustered, fault-tolerant, memory-based tables                 |
| NDB        | NO      | Alias for NDBCLUSTER                                           |
| EXAMPLE    | NO      | Example storage engine                                         |
| ARCHIVE    | YES     | Archive storage engine                                         |
| CSV        | NO      | CSV storage engine                                             |
| FEDERATED  | YES     | Federated MySQL storage engine                                 |
| BLACKHOLE  | YES     | /dev/null storage engine (anything you write to it disappears) |
+------------+---------+----------------------------------------------------------------+
```

You then duplicate your test and first target the InnoDB storage engine as you've heard it is very close to legacy databases in terms of functionality. First you validate your test object is using the InnoDB storage engine through the SHOW CREATE TABLE command:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5 to server version: 5.0.12-beta-max

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use test
Database changed

mysql> show create table insert_test\G
*************************** 1. row ***************************
       Table: insert_test
Create Table: CREATE TABLE `insert_test` (
  `c1` int(11) default NULL,
  `c2` varchar(20) default NULL,
  `c3` date default NULL
) ENGINE=INNODB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

Now you create a test stored procedure and run your test:

mysql> delimiter //
mysql> create procedure test_insert()
    -> begin
    -> declare v_ctr mediumint;
    -> set v_ctr = 0;
    -> while v_ctr < 3000000
    -> do
    ->     insert into insert_test values (1,'sample audit string',now());
    ->     set v_ctr = v_ctr + 1;
    -> end while;
    -> end
    -> //
Query OK, 0 rows affected (0.05 sec)

mysql> delimiter ;
mysql> call test_insert();
Query OK, 1 row affected (3 min 4.75 sec)
```

Performance of MySQL using the InnoDB storage engine is very close to the legacy database. You then decide to try the MyISAM storage engine, which is the default storage engine that comes with MySQL. Testing the MyISAM storage engine requires only one simple DDL

Copyright © 2005, MySQL AB

statement to change the current InnoDB table to MyISAM (the table was also truncated beforehand so it would be empty for the test):

```
mysql> truncate table insert_test;
Query OK, 0 rows affected (0.05 sec)

mysql> alter table insert_test engine=myisam;
Query OK, 0 rows affected, 0 warning (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> call test_insert();
Query OK, 1 row affected (1 min 49.88 sec)
```

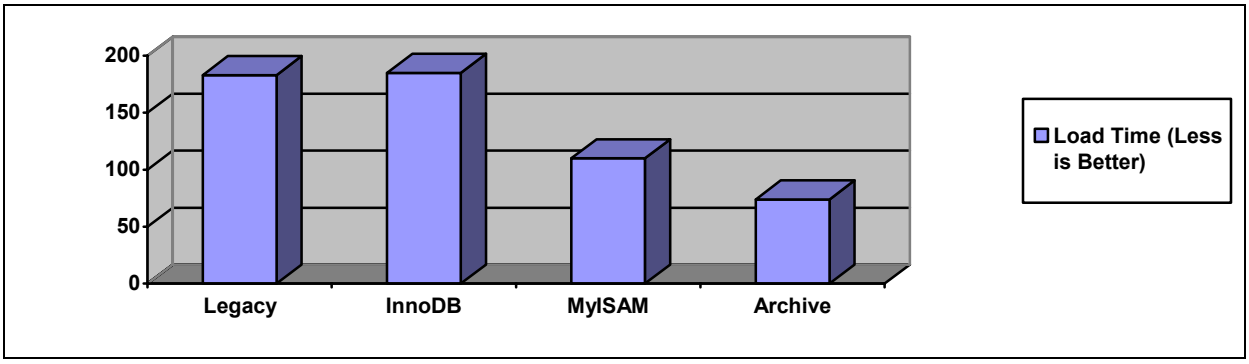
MyISAM does pretty well beating the legacy database and InnoDB by over 73 seconds, which equates to about a 40% improvement in overall performance. While you're pleased, you've heard about a new storage engine in MySQL 5.0 called Archive that's supposed to be even more efficient for insert activity, plus it offers the added benefit of transparent data compression (with reported storage savings being in the neighborhood of 80% in some cases), row-level locking, and a consistent/snapshot read. As these features help support your application, you decide to give it try. Testing the Archive engine again requires only one simple DDL statement to change the current MyISAM table to Archive:

```
mysql> truncate table insert_test;
Query OK, 0 rows affected (0.05 sec)

mysql> alter table insert_test engine=archive;
Query OK, 0 rows affected, 0 warning (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> call test_insert();
Query OK, 1 row affected (1 min 13.83 sec)
```

The MySQL 5.0 Archive engine beats the MyISAM insert test by 36 seconds, giving it a 33% improvement over MyISAM and a whopping 60% response time improvement over the legacy system and InnoDB.



Beyond the above insert speed test results, you can see how easy it is to switch from within MySQL to what amounts to a completely new database engine – one that can offer a dramatically positive impact for the properly targeted application.

Copyright © 2005, MySQL AB

The beautiful thing is, you don't have to worry about any dependencies that have been previously defined on the former storage engine object. Whether you have created procedures, triggers, views, functions, base indexes (keep in mind, different storage engines support different indexing strategies), etc., they all are retained after the ALTER TABLE statement and support the new storage engine object. Moreover, all the same SQL, SQL functions, and management functions such as replication transparently work without any changes or intervention necessary on part of the DBA or developer.

After your test, you want to check table scan times for your legacy system and MySQL. Once again, whether a table is loaded with data or not, switching between MySQL storage engines is a piece of cake. First, you perform a full table scan of the three million row legacy table with the following results:

```
SQL> select count(*) from test_insert where c1 = 1;

  COUNT(*)
----------
   3000000

Elapsed: 00:00:01.43
SQL>
```

Then you try scan times for the MySQL Archive, MyISAM, and InnoDB storage engines:

```
mysql> show create table insert_test\G
*************************** 1. row ***************************
       Table: insert_test
Create Table: CREATE TABLE `insert_test` (
  `c1` int(11) default NULL,
  `c2` varchar(20) default NULL,
  `c3` date default NULL
) ENGINE=ARCHIVE DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

mysql> select count(*) from insert_test where c1 = 1;
+----------+
| count(*) |
+----------+
|  3000000 |
+----------+
1 row in set (1.98 sec)

mysql> alter table insert_test engine=myisam;
Query OK, 3000000 rows affected, 0 warning (5.86 sec)
Records: 3000000  Duplicates: 0  Warnings: 0

mysql> select count(*) from insert_test where c1 = 1;
+----------+
| count(*) |
+----------+
|  3000000 |
+----------+
1 row in set (1.86 sec)

mysql> alter table insert_test engine=innodb;
Query OK, 3000000 rows affected (35.89 sec)
Records: 3000000  Duplicates: 0  Warnings: 0

mysql> select count(*) from insert_test where c1 = 1;
+----------+
| count(*) |
+----------+
|  3000000 |
+----------+
1 row in set (4.05 sec)
```

Copyright © 2005, MySQL AB

In these tests, the legacy system has a slight performance advantage in the full table scan with MyISAM coming in at less than half a second difference, followed closely by the Archive storage engine (with each scan being under two seconds), and then finally InnoDB. Notice how easy and quick it is to change MySQL storage engines, even when millions of rows of data exist in a table.

The Impact of "Unplugging" a Storage Engine

Because of MySQL's unique architecture, you can easily disable certain engines if you choose. This can actually have positive benefits as some engines carry various amounts of overhead when enabled even though they are not used. "Unplugging" them can ensure that only the necessary amount of resources are used by the MySQL server.

Of course, you don't want to disable engines that carry specific feature sets that pertain to your particular application, but once you determine your system's needs, you should then evaluate which MySQL storage engines you can "unplug" from the server.

Some have worried that certain large-scale features such as replication, stored procedures, etc., are embedded in the various storage engines, and disabling or not using a certain engine will have far reaching implications. As we've already seen in the above discussion on the various MySQL layers, this isn't true.

Let's say you determine you don't need the InnoDB transactional engine for your particular application. Disabling InnoDB in MySQL is quite easy and involves setting only one configuration parameter in the MySQL configuration file:

```
# The MySQL Server
[mysqld]
skip-innodb
.
.
.
```

We then stop and start the MySQL server and check if InnoDB is truly disabled.

```
mysql> show engines;
+------------+---------+----------------------------------------------------------------+
| Engine     | Support | Comment                                                        |
+------------+---------+----------------------------------------------------------------+
| MyISAM     | DEFAULT | Default engine as of MySQL 3.23 with great performance         |
| MEMORY     | YES     | Hash based, stored in memory, useful for temporary tables      |
| HEAP       | YES     | Alias for MEMORY                                               |
| 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     | DISABLED| Supports transactions, row-level locking, and foreign keys     |
| INNOBASE   | DISABLED| Alias for INNODB                                               |
| BDB        | YES     | Supports transactions and page-level locking                   |
| BERKELEYDB | YES     | Alias for BDB                                                  |
| NDBCLUSTER | NO      | Clustered, fault-tolerant, memory-based tables                 |
| NDB        | NO      | Alias for NDBCLUSTER                                           |
| EXAMPLE    | NO      | Example storage engine                                         |
| ARCHIVE    | YES     | Archive storage engine                                         |
| CSV        | NO      | CSV storage engine                                             |
| FEDERATED  | YES     | Federated MySQL storage engine                                 |
| BLACKHOLE  | YES     | /dev/null storage engine (anything you write to it disappears) |
+------------+---------+----------------------------------------------------------------+
```

Copyright © 2005, MySQL AB

Then we test a variety of MySQL support features to ensure all is well. Let's first test stored procedures:

```
mysql> show create table SH_Part\G
*************************** 1. row ***************************
       Table: SH_Part
Create Table: CREATE TABLE `SH_Part` (
  `Model` int(11) NOT NULL,
  `ProductID` int(11) NOT NULL,
  `Serial_Number` char(12) NOT NULL,
  `Sub_Category` char(5) NOT NULL,
  `Version` int(11) NOT NULL,
  `Part_Name` char(24) NOT NULL,
  `Comment1` char(30) NOT NULL,
  `Price` double NOT NULL,
  `VendorID` int(11) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

mysql> delimiter //
mysql> create procedure testing ()
    -> begin
    -> select count(*) from SH_Part;
    -> end
    -> //
Query OK, 0 rows affected (0.03 sec)

mysql> delimiter ;
mysql> call testing();
+----------+
| count(*) |
+----------+
|    40000 |
+----------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Now, how about a trigger that encrypts data on insert:

mysql> create table test (userid int, username varchar(20), userssn varchar(15));
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //
mysql> create trigger t_test_insert before insert on test
    -> for each row
    -> begin
    -> set NEW.userssn = aes_encrypt(NEW.userssn,'password');
    -> end
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter ;
mysql>  insert into test values (1, 'robin','432123454');
Query OK, 1 row affected, 0 warning (0.01 sec)

mysql> select * from test;
+--------+----------+-----------------+
| userid | username | userssn         |
+--------+----------+-----------------+
|      1 | robin    | O╕¿▒A"⌋╕Qi►ô╫▒v |
+--------+----------+-----------------+
1 row in set (0.00 sec)
```

We could continue on with all other MySQL 5.0 features, but the above tests should suffice to prove beyond any doubt that 5.0 features such as stored procedures, triggers, and views are not dependent on InnoDB or any other particular storage engine.

Copyright © 2005, MySQL AB

Creating Your Own Storage Engine

The marriage of open source freedom and the MySQL pluggable storage engine architecture means that you can extend the MySQL database server to include custom storage engines that meet specialized application needs that aren't 100% addressed by bundled MySQL storage engines. Many MySQL customers have done this and are experiencing great success in both small and large enterprises. To assist customers, MySQL ships with an Example storage engine that can be used to jump-start any custom storage engine project.

The subject matter for creating a custom storage engine is beyond the scope of this paper; however, a nice short tutorial on how to create a custom storage engine exists on the MySQL developer zone web site at http://dev.mysql.com/tech-resources/articles/creating-new-storage-engine.html. For anyone wishing to build a MySQL custom storage engine, the above referenced article offers a very nice start. MySQL and the open source community continue to develop new storage engines regularly both for specialized and mainstream purposes.

Conclusion

The growing popularity of MySQL 5.0 can be attributed to many factors, but one thing that is particularly attractive is MySQL's pluggable storage engine architecture, which affords great flexibility and choice to any IT professional that is charged with designing high-powered database applications. The benefits of working with a transparent data management support and services layer that is joined to interchangeable database storage engines is indeed unique in the database industry and is a combination that is hard to beat.

About MySQL

MySQL AB develops and supports a family of high performance, affordable database products — including MySQL Network, a comprehensive set of certified software and premium support services. The company's flagship product is MySQL, the world's most popular open source database, with more than 6 million active installations. Many of the world's largest organizations, including Yahoo!, Sabre Holdings, The Associated Press, Suzuki and NASA are realizing significant cost savings by using MySQL to power high-volume Web sites, business-critical enterprise applications and packaged software.

With headquarters in Sweden and the United States — and operations around the world — MySQL AB supports both open source values and corporate customers' needs in a profitable, sustainable business. For more information about MySQL, please visit www.mysql.com.

Copyright © 2005, MySQL AB