Adding to MySQL

A Senior Project

By

Joel Seligstein

Fall 2006

Adding to MySQL

Software applications and websites today rely heavily on large amounts of data. Whereas computers a decade ago had a few mega-bytes of memory, systems now use tera-bytes and peta-bytes of memory for storage and load balancing on databases. Accessing the data in a quick and efficient manner becomes the main challenge for software developers and database administrators. In recent history, many large-scale technology corporations are beginning to use open-source MySQL as their main database back-end. Despite qualms about open-source, MySQL has become an enterprise-ready solution and is overwhelmingly cheaper than similar commercial solutions such as Oracle and MSSQL. As an emerging leader, it is becoming important for community programmers to contribute to the project to support its growth. This semester, it has been my goal to learn and understand large amounts of the MySQL source code and create features that database administrators will use in their work.

The code base is a beast. While I have edited small software packages before, I was unexperienced and unprepared for the magnitude and depth of the code. Thus, I began work on smaller projects to deepen my understanding before undertaking an extremely large task. Before I explain what additions I have made, it may be necessary to explain how the database works. First and foremost, it is important to understand the server-client relationship. Most of MySQL's code base deals with the server. The program runs constantly on a server and accepts connections from other programs, the clients. A client connects to the server using MySQL's Application Programming Interface (API) and executes queries on the server. A query is written in a language called Structured Query Language (SQL), and describes actions that a client requests, including the ability INSERT, DELETE, UPDATE, and SELECT items in the database. The server parses the SQL, does the action requested, and returns a result, if any, to the client. Most of the time, queries execute at an incredible amount of speed, usually less than a second for almost all decently complex queries. It becomes a main responsibility of

a database administrator to determine the cause of slow-running queries. Thus, most of my development focused on helping administrators to make connections more manageable.

While logged into the official MySQL Command Line Interface (CLI), an administrator can run any queries necessary. A common one used is SHOW FULL PROCESSLIST. A sample output may look like:

```
mysql> show full processlist;
+------+------+-----------+---------+---------+------+----------+---------------------------------+
| Id   | User | Host      | db      | Command | Time | State    | Info                            |
+------+------+-----------+---------+---------+------+----------+---------------------------------+
| 34   | root | localhost | Static2 | Query   | 25   | Updating | UPDATE RegionTree SET lft= [...] |
| 920  | root | localhost | Static2 | Query   | 46   | Updating | UPDATE RegionTree SET rgt= [...] |
| 999  | root | localhost | Static2 | Query   | 10   | Updating | UPDATE RegionTree SET lft= [...] |
| 1786 | root | localhost | Static2 | Query   | 40   | Updating | UPDATE RegionTree SET lft= [...] |
| 1790 | root | localhost |         | Query   | 0    |          | show full processlist            |
+------+------+-----------+---------+---------+------+----------+---------------------------------+
5 rows in set (0.00 sec)
```

It is evident that the server is running 4 UPDATE queries and the current show query. Under the time column, it can be seen that these queries are long running, one taking over 45 seconds already. When a connection is running long queries, it may be necessary to understand what stage a current connection is in. For example, a website may be running MySQL as its database. When a user logs into the database, in one connection it may want to SELECT the user's data such as name or email address, INSERT a row logging what time the user logged in, and UPDATE the row that says when the user last logged in as well as UPDATE some of the user's information, such as IP address. Since this is a multiple-query process, seeing its status would be helpful. Thus, my first patch adds this capability. I first edited the parser to allow a new SET command called SET CONNECTION status. This can be invoked like so:

```
mysql> set connection status="Updating user's data.";
```

This involved adding a variable to the main connection class and verifying the setting of the status. Then, when a user runs SHOW FULL PROCESSLIST, a new column is appended to the end, showing the status:

```
mysql> show full processlist;
+----+------+------+------+---------+------+-------+----------------------+----------------------+
```

```
| Id | User | Host | db   | Command | Time | State | Info                  | Status                |
+----+------+------+------+---------+------+-------+-----------------------+-----------------------+
|  1 | root |      | NULL | Query   |    0 | NULL  | show full processlist | Updating user's data. |
+----+------+------+------+---------+------+-------+-----------------------+-----------------------+
1 row in set (0.00 sec)
```

As seen above, the status is now helpful for long-running query investigation as long as the application sets its status at regular intervals.

My second patch on the code follows a similar theme of handling out-of-control queries. When a query runs too long or holds a lock on a table (preventing other queries from accessing data), an administrator will want to log in and kill a query. Currently, the syntax allows for a user to terminate a query in a connection or terminate the entire connection for a certain query. For example, an administrator may run:

```
mysql> KILL CONNECTION 3;
```
or
```
mysql> KILL QUERY 3;
```

The first would kill the entire connection with Id of 3 (see the output from SHOW FULL PROCESSLIST above), and the second would only terminate the query currently running in connection 3 while allowing the connection to continue executing any other queries it needs. My patch allows the administrator to add a condition to these queries. This is necessary to solve a well-known race condition. When an administrator wants to kill a query, he runs SHOW FULL PROCESSLIST to see which connection to kill, and then runs the KILL query. However, the connection may have already finished that query and moved on, in which case the administrator would have rather it continued to run. My patch adds a column to SHOW FULL PROCESSLIST which displays a query id, a unique number assigned to each query a server ever runs, regardless of connection.

```
mysql> show full processlist;
+----+------+------+------+---------+------+-------+-----------------------------------------+----------+
| Id | User | Host | db   | Command | Time | State | Info                                    | Query_Id |
+----+------+------+------+---------+------+-------+-----------------------------------------+----------+
|  1 | joel |      | NULL | Query   |    5 | init  | select benchmark(1000,repeat('a',10)) |       15 |
|  2 | root |      | NULL | Query   |    0 | NULL  | show full processlist                   |       16 |
+----+------+------+------+---------+------+-------+-----------------------------------------+----------+
2 rows in set (0.00 sec)
```

After seeing the specific query id number, the administrator can kill a query based on whether or not that query is still running or not:

```
mysql> kill query 1 with query 15;
Query OK, 0 rows affected (0.00 sec)
```

or

```
mysql> kill connection 1 with query 15;
Query OK, 0 rows affected (0.00 sec)
```

There are two main checks to be done to ensure this is okay as well. First, the user must have sufficient privileges to do such a command. The following error message displays if the user does not have the SUPER privilege or does not own the query:

```
mysql> kill connection 4 with query 30;
ERROR 1095 (HY000): You are not owner of thread 4
```

Also, the thread must exist and be running the specified query:

```
mysql> kill connection 4 with query 31;
ERROR 1472 (HY000): Cannot kill thread 4, Query ID mismatch
```

Thus, this patch solves the race condition for killing queries by allowing an administrator with proper privileges to kill queries only when the query is still running.

The third and final small patch I wrote was to enable administrators to get insight into other connections that are running. Currently an administrator can view everything about a connection by running SHOW VARIABLES or SHOW STATUS:

```
mysql> show session variables;
+--------------------------+--------------+
| Variable_name            | Value        |
+--------------------------+--------------+
| auto_increment_increment | 1            |
| auto_increment_offset    | 1            |
| automatic_sp_privileges  | ON           |
| ...                      |              |
| version_compile_os       | pc-linux-gnu |
| wait_timeout             | 28800        |
+--------------------------+--------------+

mysql> show session status;
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| Aborted_clients  | 0     |
| Aborted_connects | 0     |
| ...              |       |
| Threads_running  | 1     |
| Uptime           | 147   |
+------------------+-------+
```

However, it is not currently possible to look at another connection's variables or status. My patch

allows that capability:

```
mysql> show session variables like 'wait_timeout';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wait_timeout  | 28800 |
+---------------+-------+
1 row in set (0.00 sec)

mysql> show session variables like 'wait_timeout' for connection 2;
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wait_timeout  | 14400 |
+---------------+-------+
1 row in set (0.00 sec)
```

Thus, now administrators have more capability to discover problems with other connections.

After adding functions, variables, and other capabilities to the database code, I wanted to attempt a larger feature. On websites and other systems, it may be necessary to store hierarchical data. A good example is categories for eBay or Amazon. For instance, part of their category data may visually look like:

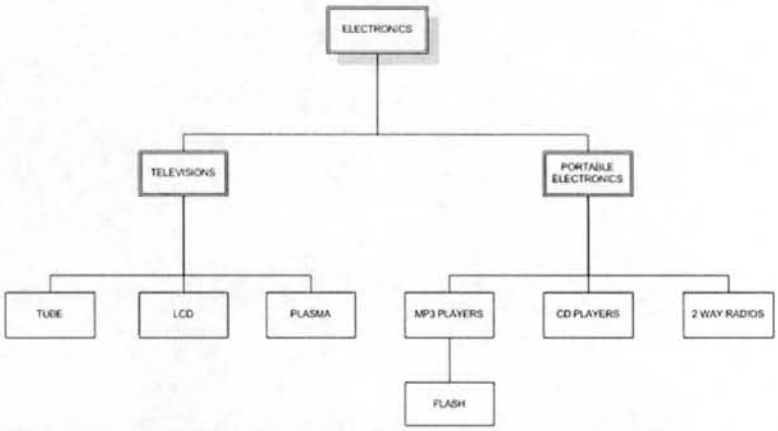

Illustration 1: From "Managing Hierarchical Data" on the MySQL Developer website: http://dev.mysql.com/tech-resources/articles/hierarchical-data.html

There are two traditional methods for storing this data in a database. First, an application may store each item and its parent. For instance, it may store "LCD" with parent "TELEVISIONS". This solution has some inherent positives as it is very easy to get any one item's children or many items' parent. Also, the insert or update is very easy as the parent can just be changed to move an item or set

to create a new item. However, it is cumbersome and inefficient to get an entire path to one item. For example, if we had the item "FLASH", we would have to make three requests: one to find the parent of "FLASH" which is "MP3 PLAYERS", another to find its parent "PORTABLE ELECTRONICS", and a third to find its parent "ELECTRONICS". This can make for lots of queries if the tree were much deeper.

Another method is to view the hierarchical data as sets and subsets. Then, a set diagram can be drawn. Once the sets are laid out horizontally, the sets can be numbered with a left number and a right number. The same tree above can be numbered like so:

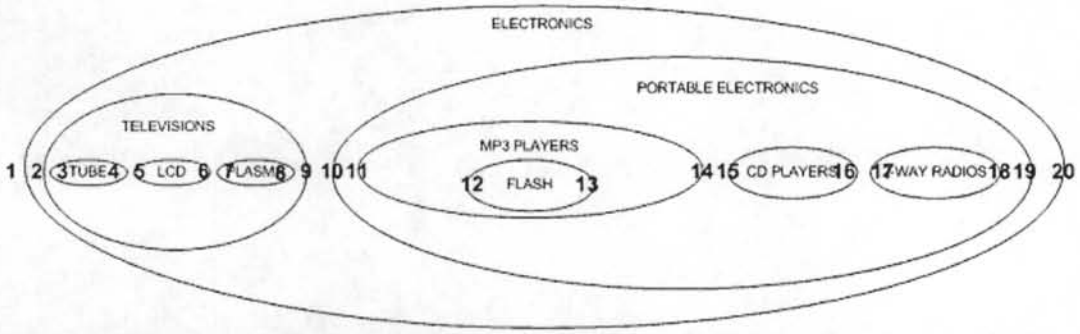

Illustration 2: From "Managing Hierarchical Data" on the MySQL Developer website: http://dev.mysql.com/tech-resources/articles/hierarchical-data.html

Once these numbers are in the database, a coder could request any item and its entire subtree by simply selecting between its numbers. For instance, a query could be ran asking for and left greater than 2 with a right less than or 9. This would return "TUBE", "LCD", and "PLASMA", noticeably "TELEVISION"'s children. It is also easy to select a path to the item by requesting any left less than its left and right greater than its right. For example, if a coder asked for any left less than 12 and right greater than 13, "MP3 PLAYERS", "PORTABLE ELECTRONICS", and "ELECTRONICS" would be returned. This is the same path from above in one query and lookup rather than three. However, this method is limiting as well as it is near impossible to select only one set of children without selecting the grandchildren. Also, inserts, deletes, and updates into this structure could quite easily require the

entire tree to be renumbered.

Other commercial databases have solved this solution with a feature called CONNECT BY PRIOR. The data is stored in the parent-child form above, and the user requests the data by specifying how to connect one result from the other. This allows the user to write one simple query and the database can handle it efficiently internally. A sample query and result would look like:

```
mysql> select name,parent from tree start with "PORTABLE ELECTRONICS" connect by prior name = parent;
+--------------+----------------------+
| name         | parent               |
+--------------+----------------------+
| MP3 PLAYERS  | PORTABLE ELECTRONICS |
| FLASH        | MP3 PLAYERS          |
| CD PLAYERS   | PORTABLE ELECTRONICS |
| 2-WAY RADIOS | PORTABLE ELECTRONICS |
+--------------+----------------------+
```

As seen above, it would return any item with that has a parent of "PORTABLE ELECTRONICS" (since that is what the start clause designated) as well as its children. Thus, this method is extremely convenient: inserts, deletes, and updates are quick and simple as well as selecting trees and paths can be simple.

Adding this feature is not nearly as trivial as the others. This requires intercepting result sets before they are sent to the client and adding new entries based on new queries. The queries have to be duplicated from the original queries and updated to include the new condition. This is not quite as easy as it sounds and requires enormous effort and understanding of internals of the code. As of the writing of this paper, this feature is not complete. I have duplicated the queries in a couple of different ways, but not inserted the new conditions yet, despite working on this project non-stop for the last few weeks of the semester. I intend to finish this feature in the near future to contribute a significant change to the community, adding a feature many Oracle users employ in their applications.

My project has turned out to be a wonderful success. I have learned an incredible amount of the MySQL code base. I finished three patches to add significant features to the application. I spoke briefly about these patches at a recent MySQL informal conference and received a great amount of

interest. The maintainers of the code were present and the new public versions of MySQL will contain the code for my three features. I have been told that my CONNECT BY implementation will be inserted into the code as well. In conclusion, I have made significant contributions to MySQL's code base and will benefit from the learned skills in the future by adding to other code bases or MySQL as needed. Thus, I have accomplished my goal of aiding database administrators and understanding such a large complex code base.

APPENDIX A

show_for_connection.patch

This is a patch file for the changes pertaining to allowing show commands for other threads. This is a specific file designating lines to change in the original source document. The + indicates added lines, the – indicates removed lines, and lines with no prefix are simply guiding lines.

```
diff -Nur -x bdb mysql-5.0.26/sql/sql_show.cc mysql-5.0.26.show_for_connection/sql/sql_show.cc
--- mysql-5.0.26/sql/sql_show.cc     2006-10-04 04:24:21.000000000 -0700
+++ mysql-5.0.26.show_for_connection/sql/sql_show.cc       2006-10-30 12:22:10.000000000 -0800
@@ -3489,14 +3489,79 @@
 int fill_variables(THD *thd, TABLE_LIST *tables, COND *cond)
 {
   DBUG_ENTER("fill_variables");
-  int res= 0;
+  int error= 0;
   LEX *lex= thd->lex;
   const char *wild= lex->wild ? lex->wild->ptr() : NullS;
+  Item *thd_item;
+  THD *show_thd= thd;
+  THD *find_thd= NULL;
+  ulong thd_id= 0;
+
+  thd_item= lex->value_list.head();
+  /*
+    If no thread specified, don't try to get the value, this will
+    be a null pointer.
+  */
+  if (thd_item)
+  {
+    if ((!thd_item->fixed && thd_item->fix_fields(thd, &thd_item))
+        || thd_item->check_cols(1))
+    {
+      my_message(ER_SET_CONSTANTS_ONLY, ER(ER_SET_CONSTANTS_ONLY), MYF(0));
+      DBUG_RETURN(ER_SET_CONSTANTS_ONLY);
+    }
+    thd_id= (ulong)thd_item->val_int();
+  }
+
+  /*
+    If the thd_id matches the current thread id, we can skip the search
+    as well as skip the lock.
+  */
+
+  if (thd_id == thd->thread_id)
+    thd_id= 0;
+
+  /*
+    If we need to search for a thread, we need to lock the list,
+    search for a matching thread id, and check permissions.  If
+    all checks pass, we'll have a new valid show_thd.  Otherwise
+    the function will go ahead and error out.
+   */
+  if (thd_id)
+  {
+    VOID(pthread_mutex_lock(&LOCK_thread_count)); /* no unlink from list */
+    I_List_iterator<THD> iter(threads);
+    while ((find_thd= iter++))
+    {
+      if (find_thd->thread_id == thd_id)
+        break;
+    }
+    if (!find_thd)
+    {
+      error= ER_NO_SUCH_THREAD;
+      my_error(error, MYF(0), thd_id);
+      goto unlock_and_return;
+    }
```

```
+    if (!((thd->security_ctx->master_access & SUPER_ACL)
+        || !strcmp(thd->security_ctx->user, find_thd->security_ctx->user)))
+    {
+      error= ER_SPECIFIC_ACCESS_DENIED_ERROR;
+      my_error(error, MYF(0), "SUPER");
+      goto unlock_and_return;
+    }
+
+    show_thd= find_thd;
+  }
+
   pthread_mutex_lock(&LOCK_global_system_variables);
-  res= show_status_array(thd, wild, init_vars,
-                         lex->option_type, 0, "", tables->table);
+  error= show_status_array(show_thd, wild, init_vars,
+                           lex->option_type, 0, "", tables->table);
   pthread_mutex_unlock(&LOCK_global_system_variables);
-  DBUG_RETURN(res);
+
+unlock_and_return:
+  if (thd_id)
+    VOID(pthread_mutex_unlock(&LOCK_thread_count));
+  DBUG_RETURN(error);
 }


@@ -3505,17 +3570,82 @@
   DBUG_ENTER("fill_status");
   LEX *lex= thd->lex;
   const char *wild= lex->wild ? lex->wild->ptr() : NullS;
-  int res= 0;
+  int error= 0;
   STATUS_VAR tmp;
+  Item *thd_item;
+  THD *show_thd= thd;
+  THD *find_thd= NULL;
+  ulong thd_id= 0;
+
+  thd_item= lex->value_list.head();
+  /*
+     If no thread specified, don't try to get the value, this will
+     be a null pointer.
+  */
+  if (thd_item)
+  {
+    if ((!thd_item->fixed && thd_item->fix_fields(thd, &thd_item))
+        || thd_item->check_cols(1))
+    {
+      my_message(ER_SET_CONSTANTS_ONLY, ER(ER_SET_CONSTANTS_ONLY), MYF(0));
+      DBUG_RETURN(ER_SET_CONSTANTS_ONLY);
+    }
+    thd_id= (ulong)thd_item->val_int();
+  }
+
+  /*
+     If the thd_id matches the current thread id, we can skip the search
+     as well as skip the lock.  Same situation if we're looking
+     for a global status.
+  */
+  if (thd_id == thd->thread_id || lex->option_type == OPT_GLOBAL)
+    thd_id= 0;
+
+  /*
+     If we need to search for a thread, we need to lock the list,
+     search for a matching thread id, and check permissions.  If
+     all checks pass, we'll have a new valid show_thd.  Otherwise
+     the function will go ahead and error out.
+   */
+  if (thd_id)
+  {
+    VOID(pthread_mutex_lock(&LOCK_thread_count)); /* no unlink from list */
```

```
+    I_List_iterator<THD> iter(threads);
+    while ((find_thd= iter++))
+    {
+      if (find_thd->thread_id == thd_id)
+        break;
+    }
+    if (!find_thd)
+    {
+      error= ER_NO_SUCH_THREAD;
+      my_error(error, MYF(0), thd_id);
+      goto unlock_and_return;
+    }
+    if (!((thd->security_ctx->master_access & SUPER_ACL)
+        || !strcmp(thd->security_ctx->user, find_thd->security_ctx->user)))
+    {
+      error= ER_SPECIFIC_ACCESS_DENIED_ERROR;
+      my_error(error, MYF(0), "SUPER");
+      goto unlock_and_return;
+    }
+
+    show_thd= find_thd;
+  }
+
   ha_update_statistics();                    /* Export engines statistics */
   pthread_mutex_lock(&LOCK_status);
   if (lex->option_type == OPT_GLOBAL)
     calc_sum_of_all_status(&tmp);
-  res= show_status_array(thd, wild, status_vars, OPT_GLOBAL,
+  error= show_status_array(show_thd, wild, status_vars, OPT_GLOBAL,
                          (lex->option_type == OPT_GLOBAL ?
-                          &tmp: &thd->status_var), "",tables->table);
+                          &tmp: &show_thd->status_var), "",tables->table);
   pthread_mutex_unlock(&LOCK_status);
-  DBUG_RETURN(res);
+
+unlock_and_return:
+  if (thd_id)
+    VOID(pthread_mutex_unlock(&LOCK_thread_count));
+  DBUG_RETURN(error);
 }


diff -Nur -x bdb mysql-5.0.26/sql/sql_yacc.yy mysql-5.0.26.show_for_connection/sql/sql_yacc.yy
--- mysql-5.0.26/sql/sql_yacc.yy     2006-10-04 04:24:23.000000000 -0700
+++ mysql-5.0.26.show_for_connection/sql/sql_yacc.yy       2006-10-30 12:25:41.000000000 -0800
@@ -758,6 +758,7 @@
        sp_opt_default
        simple_ident_nospvar simple_ident_q
        field_or_var limit_option
+       opt_for_conn

 %type <item_num>
        NUM_literal
@@ -6542,12 +6543,14 @@
           { Lex->sql_command = SQLCOM_SHOW_WARNS;}
         | ERRORS opt_limit_clause_init
           { Lex->sql_command = SQLCOM_SHOW_ERRORS;}
-        | opt_var_type STATUS_SYM wild_and_where
+        | opt_var_type STATUS_SYM wild_and_where opt_for_conn
           {
             LEX *lex= Lex;
             lex->sql_command= SQLCOM_SELECT;
             lex->orig_sql_command= SQLCOM_SHOW_STATUS;
             lex->option_type= $1;
+           lex->value_list.empty();
+           lex->value_list.push_back($4);
             if (prepare_schema_table(YYTHD, lex, 0, SCH_STATUS))
               YYABORT;
           }
@@ -6557,12 +6560,14 @@
           { Lex->sql_command = SQLCOM_SHOW_MUTEX_STATUS; }
```

```
        | opt_full PROCESSLIST_SYM
          { Lex->sql_command= SQLCOM_SHOW_PROCESSLIST;}
-         | opt_var_type  VARIABLES wild_and_where
+         | opt_var_type VARIABLES wild_and_where opt_for_conn
          {
              LEX *lex= Lex;
              lex->sql_command= SQLCOM_SELECT;
              lex->orig_sql_command= SQLCOM_SHOW_VARIABLES;
              lex->option_type= $1;
+           lex->value_list.empty();
+           lex->value_list.push_front($4);
              if (prepare_schema_table(YYTHD, lex, 0, SCH_VARIABLES))
                YYABORT;
          }
@@ -6757,6 +6762,11 @@
          }
        ;

+opt_for_conn:
+       /* empty */                    { $$= 0;  }
+       | FOR_SYM CONNECTION_SYM expr { $$= $3; }
+       ;
+
 /* A Oracle compatible synonym for show */
 describe:
```

APPENDIX B

connection_status.patch

This is a patch file for the connection status changes. This is a specific file designating lines to change in the original source document. The + indicates added lines, the – indicates removed lines, and lines with no prefix are simply guiding lines.

```
diff -Nur -x bdb mysql-5.0.26/sql/item_create.cc set_session/sql/item_create.cc
--- mysql-5.0.26/sql/item_create.cc  2006-10-04 04:24:09.000000000 -0700
+++ set_session/sql/item_create.cc   2006-10-17 20:22:26.000000000 -0700
@@ -75,6 +75,11 @@
   return new Item_func_connection_id();
 }

+Item *create_func_connection_status(void)
+{
+  return new Item_func_connection_status();
+}
+
 Item *create_func_conv(Item* a, Item *b, Item *c)
 {
   return new Item_func_conv(a,b,c);
diff -Nur -x bdb mysql-5.0.26/sql/item_create.h set_session/sql/item_create.h
--- mysql-5.0.26/sql/item_create.h   2006-10-04 04:24:01.000000000 -0700
+++ set_session/sql/item_create.h    2006-10-17 19:58:25.000000000 -0700
@@ -31,6 +31,7 @@
 Item *create_func_cast(Item *a, Cast_target cast_type, int len, int dec,
                        CHARSET_INFO *cs);
 Item *create_func_connection_id(void);
+Item *create_func_connection_status(void);
 Item *create_func_conv(Item* a, Item *b, Item *c);
 Item *create_func_cos(Item* a);
 Item *create_func_cot(Item* a);
diff -Nur -x bdb mysql-5.0.26/sql/item_strfunc.cc set_session/sql/item_strfunc.cc
--- mysql-5.0.26/sql/item_strfunc.cc 2006-10-04 04:24:23.000000000 -0700
+++ set_session/sql/item_strfunc.cc  2006-10-17 20:40:35.000000000 -0700
@@ -3175,3 +3175,15 @@
   return str;
 }

+String *Item_func_connection_status::val_str(String *str)
+{
+  if(!current_thd->connection_status)
+  {
+    null_value= 1;
+    return 0;
+  }
+
+  str->copy(current_thd->connection_status, strlen(current_thd->connection_status),
system_charset_info);
+  return str;
+}
+
diff -Nur -x bdb mysql-5.0.26/sql/item_strfunc.h set_session/sql/item_strfunc.h
--- mysql-5.0.26/sql/item_strfunc.h  2006-10-04 04:24:10.000000000 -0700
+++ set_session/sql/item_strfunc.h   2006-10-17 20:16:26.000000000 -0700
@@ -842,3 +842,13 @@
   String *val_str(String *);
 };

+class Item_func_connection_status: public Item_str_func
+{
+public:
+  Item_func_connection_status(): Item_str_func() {}
+  void fix_length_and_dec() {
+    max_length= MAX_BLOB_WIDTH;
+  }
+  const char *func_name() const{ return "connection_status"; }
```

```
+  String *val_str(String *);
+};
diff -Nur -x bdb mysql-5.0.26/sql/lex.h set_session/sql/lex.h
--- mysql-5.0.26/sql/lex.h    2006-10-04 04:24:43.000000000 -0700
+++ set_session/sql/lex.h     2006-10-17 20:13:15.000000000 -0700
@@ -577,6 +577,7 @@
   { "CONCAT",         SYM(CONCAT)},
   { "CONCAT_WS",      SYM(CONCAT_WS)},
   { "CONNECTION_ID",  F_SYM(FUNC_ARG0),0,CREATE_FUNC(create_func_connection_id)},
+  { "CONNECTION_STATUS",F_SYM(FUNC_ARG0),0,CREATE_FUNC(create_func_connection_status)},
   { "CONV",           F_SYM(FUNC_ARG3),0,CREATE_FUNC(create_func_conv)},
   { "CONVERT_TZ",     SYM(CONVERT_TZ_SYM)},
   { "COUNT",          SYM(COUNT_SYM)},
diff -Nur -x bdb mysql-5.0.26/sql/set_var.cc set_session/sql/set_var.cc
--- mysql-5.0.26/sql/set_var.cc       2006-10-04 04:24:23.000000000 -0700
+++ set_session/sql/set_var.cc        2006-10-17 19:52:21.000000000 -0700
@@ -3299,6 +3299,51 @@
   return 0;
 }

+/****************************************************************************
+  Functions to handle SET CONNECTION STATUS [=] expr
+****************************************************************************/
+
+/*
+  Set the connection status for a thread
+
+  SYNOPSIS
+    status_arg      current status string
+
+  RETURN
+    0
+*/
+set_var_connection_status::set_var_connection_status(String *status_arg)
+{
+  if(status_arg)
+  {
+    status.copy(*status_arg);
+    is_null= 0;
+  } else {
+    is_null= 1;
+  }
+}
+
+int set_var_connection_status::check(THD *thd)
+{
+  return 0;
+}
+
+int set_var_connection_status::update(THD *thd)
+{
+  if(thd->connection_status)
+  {
+    my_free((gptr)thd->connection_status, MYF(0));
+    thd->connection_status= 0;
+  }
+  if(!is_null)
+  {
+    thd->connection_status= my_strdup(status.c_ptr(), MYF(0));
+  } else {
+    thd->connection_status= 0;
+  }
+  return 0;
+}
+

 /****************************************************************************
   Functions to handle SET PASSWORD
diff -Nur -x bdb mysql-5.0.26/sql/set_var.h set_session/sql/set_var.h
--- mysql-5.0.26/sql/set_var.h        2006-10-04 04:24:22.000000000 -0700
+++ set_session/sql/set_var.h 2006-10-17 18:46:34.000000000 -0700
```

```
@@ -923,6 +923,19 @@
   int update(THD *thd);
 };

+/* For SET CONNECTION STATUS */
+
+class set_var_connection_status: public set_var_base
+{
+  String status;
+  bool is_null;
+public:
+  set_var_connection_status(String *status_arg);
+  int check(THD *thd);
+  int update(THD *thd);
+};
+
+
 /* For SET NAMES and SET CHARACTER SET */

diff -Nur -x bdb mysql-5.0.26/sql/sql_class.cc set_session/sql/sql_class.cc
--- mysql-5.0.26/sql/sql_class.cc    2006-10-04 04:24:43.000000000 -0700
+++ set_session/sql/sql_class.cc     2006-10-17 19:58:01.000000000 -0700
@@ -238,6 +238,7 @@

   /* Variables with default values */
   proc_info="login";
+  connection_status= 0;
   where= THD::DEFAULT_WHERE;
   server_id = ::server_id;
   slave_net = 0;
diff -Nur -x bdb mysql-5.0.26/sql/sql_class.h set_session/sql/sql_class.h
--- mysql-5.0.26/sql/sql_class.h     2006-10-04 04:24:43.000000000 -0700
+++ set_session/sql/sql_class.h      2006-10-17 14:48:26.000000000 -0700
@@ -1147,6 +1147,9 @@
   */
   const char *proc_info;

+  /* for connection status*/
+  const char *connection_status;
+
   ulong client_capabilities;        /* What the client supports */
   ulong max_client_packet_length;

diff -Nur -x bdb mysql-5.0.26/sql/sql_show.cc set_session/sql/sql_show.cc
--- mysql-5.0.26/sql/sql_show.cc     2006-10-04 04:24:21.000000000 -0700
+++ set_session/sql/sql_show.cc      2006-10-17 16:47:30.000000000 -0700
@@ -1266,6 +1266,7 @@
   uint   command;
   const char *user,*host,*db,*proc_info,*state_info;
   char *query;
+  char *connection_status;
 };

 #ifdef HAVE_EXPLICIT_TEMPLATE_INSTANTIATION
@@ -1293,6 +1294,11 @@
   field->maybe_null=1;
   field_list.push_back(field=new Item_empty_string("Info",max_query_length));
   field->maybe_null=1;
+  if(verbose)
+  {
+    field_list.push_back(field=new Item_empty_string("Status",max_query_length));
+    field->maybe_null=1;
+  }
   if (protocol->send_fields(&field_list,
                             Protocol::SEND_NUM_ROWS | Protocol::SEND_EOF))
     DBUG_VOID_RETURN;
@@ -1373,15 +1379,15 @@
           uint length= min(max_query_length, tmp->query_length);
           thd_info->query=(char*) thd->strmake(tmp->query,length);
         }
```

```
+
+       if( verbose )
+       {
+         if(tmp->connection_status)
+           thd_info->connection_status= thd->strdup(tmp->connection_status);
+         else
+           thd_info->connection_status= 0;
+       }
+
          thread_infos.append(thd_info);
        }
      }
@@ -1392,6 +1407,13 @@
        protocol->store_null();
      protocol->store(thd_info->state_info, system_charset_info);
      protocol->store(thd_info->query, system_charset_info);
+     if(verbose)
+     {
+       if(thd_info->connection_status)
+         protocol->store(thd_info->connection_status, system_charset_info);
+       else
+         protocol->store_null();
+     }
      if (protocol->write())
        break; /* purecov: inspected */
    }
diff -Nur -x bdb mysql-5.0.26/sql/sql_yacc.yy set_session/sql/sql_yacc.yy
--- mysql-5.0.26/sql/sql_yacc.yy      2006-10-04 04:24:23.000000000 -0700
+++ set_session/sql/sql_yacc.yy       2006-10-17 18:51:34.000000000 -0700
@@ -8111,6 +8111,14 @@
          {
            Lex->var_list.push_back(new set_var_password($3,$5));
          }
+       | CONNECTION_SYM STATUS_SYM opt_equal expr
+         {
+           THD *thd= YYTHD;
+           Item *e= $4;
+           e->fix_fields(thd, &e);
+           String str, *res= e->val_str(&str);
+           Lex->var_list.push_back(new set_var_connection_status(res));
+         }
        ;

 internal_variable_name:
```

APPENDIX C

kill_with_query_id.patch

This patch file describes the changes pertaining to the additions to kill query. This is a specific file designating lines to change in the original source document. The + indicates added lines, the – indicates removed lines, and lines with no prefix are simply guiding lines.

```
diff -Nur -x bdb mysql-5.0.26/client/mysql_priv.h mysql-5.0.26.kill_query/client/mysql_priv.h
--- mysql-5.0.26/client/mysql_priv.h 2006-10-04 04:24:43.000000000 -0700
+++ mysql-5.0.26.kill_query/client/mysql_priv.h     2006-10-23 18:38:33.000000000 -0700
@@ -83,7 +83,7 @@
                             CHARSET_INFO *from_cs,
                             uint32 max_res_length,
                             CHARSET_INFO *to_cs, uint32 *result_length);
-void kill_one_thread(THD *thd, ulong id, bool only_kill_query);
+void kill_one_thread(THD *thd, ulong id, query_id_t query_id, bool only_kill_query);
 bool net_request_file(NET* net, const char* fname);
 char* query_table_status(THD *thd,const char *db,const char *table_name);

diff -Nur -x bdb mysql-5.0.26/sql/mysql_priv.h mysql-5.0.26.kill_query/sql/mysql_priv.h
--- mysql-5.0.26/sql/mysql_priv.h    2006-10-04 04:24:43.000000000 -0700
+++ mysql-5.0.26.kill_query/sql/mysql_priv.h2006-10-23 18:38:33.000000000 -0700
@@ -83,7 +83,7 @@
                             CHARSET_INFO *from_cs,
                             uint32 max_res_length,
                             CHARSET_INFO *to_cs, uint32 *result_length);
-void kill_one_thread(THD *thd, ulong id, bool only_kill_query);
+void kill_one_thread(THD *thd, ulong id, query_id_t query_id, bool only_kill_query);
 bool net_request_file(NET* net, const char* fname);
 char* query_table_status(THD *thd,const char *db,const char *table_name);

diff -Nur -x bdb mysql-5.0.26/sql/share/errmsg.txt mysql-5.0.26.kill_query/sql/share/errmsg.txt
--- mysql-5.0.26/sql/share/errmsg.txt       2006-10-04 04:24:10.000000000 -0700
+++ mysql-5.0.26.kill_query/sql/share/errmsg.txt    2006-10-24 15:53:26.000000000 -0700
@@ -5633,4 +5633,6 @@
       eng "String '%-.70s' is too long for %s (should be no longer than %d)"
 ER_NON_INSERTABLE_TABLE
       eng "The target table %-.100s of the %s is not insertable-into"
+ER_KILL_QUERY_ID_MISMATCH
+       eng "Cannot kill thread %lu, Query ID mismatch"

diff -Nur -x bdb mysql-5.0.26/sql/sql_parse.cc mysql-5.0.26.kill_query/sql/sql_parse.cc
--- mysql-5.0.26/sql/sql_parse.cc    2006-10-04 04:24:42.000000000 -0700
+++ mysql-5.0.26.kill_query/sql/sql_parse.cc2006-10-24 16:02:43.000000000 -0700
@@ -2054,7 +2054,7 @@
   {
     statistic_increment(thd->status_var.com_stat[SQLCOM_KILL], &LOCK_status);
     ulong id=(ulong) uint4korr(packet);
-    kill_one_thread(thd,id,false);
+    kill_one_thread(thd,id,0,false);
     break;
   }
   case COM_SET_OPTION:
@@ -4054,7 +4054,18 @@
   }
   case SQLCOM_KILL:
   {
-    Item *it= (Item *)lex->value_list.head();
+    query_id_t query_id= 0;
+    Item *q_item, *it= (Item *)lex->value_list.pop();
+    if(!lex->value_list.is_empty())
+    {
+      q_item= lex->value_list.pop();
+      if((!q_item->fixed && q_item->fix_fields(lex->thd, &q_item)) || q_item->check_cols(1))
+      {
+        my_message(ER_SET_CONSTANTS_ONLY, ER(ER_SET_CONSTANTS_ONLY), MYF(0));
+       goto error;
+      }
```

```
+      query_id= (ulong)q_item->val_int();
+    }

     if ((!it->fixed && it->fix_fields(lex->thd, &it)) || it->check_cols(1))
     {
@@ -4062,7 +4073,7 @@
                MYF(0));
       goto error;
     }
-    kill_one_thread(thd, (ulong)it->val_int(), lex->type & ONLY_KILL_QUERY);
+    kill_one_thread(thd, (ulong)it->val_int(), query_id, lex->type & ONLY_KILL_QUERY);
     break;
   }
 #ifndef NO_EMBEDDED_ACCESS_CHECKS
@@ -6883,15 +6894,23 @@
     kill_one_thread()
     thd                     Thread class
     id                 Thread id
+    query_id_arg       Query id to match against
+    only_kill_query  Kill only query or thread and query

   NOTES
     This is written such that we have a short lock on LOCK_thread_count
+
+    If query_id_arg != 0, the thread will only be killed if the thread's
+       current query matches.
+
+    If only_kill_query == true, then only the current query will be killed,
+       and not the entire thread
 */

-void kill_one_thread(THD *thd, ulong id, bool only_kill_query)
+void kill_one_thread(THD *thd, ulong id, query_id_t query_id_arg, bool only_kill_query)
 {
   THD *tmp;
-  uint error=ER_NO_SUCH_THREAD;
+  uint error= ER_NO_SUCH_THREAD;
   VOID(pthread_mutex_lock(&LOCK_thread_count)); // For unlink from list
   I_List_iterator<THD> it(threads);
   while ((tmp=it++))
@@ -6902,24 +6921,40 @@
       break;
     }
   }
-  VOID(pthread_mutex_unlock(&LOCK_thread_count));
+  /* Need to hold the lock through THD::awake to be sure another query
+     (with a different query_id_arg) is not started. */
+  if (!query_id_arg)
+    VOID(pthread_mutex_unlock(&LOCK_thread_count));
   if (tmp)
   {
     if ((thd->security_ctx->master_access & SUPER_ACL) ||
       !strcmp(thd->security_ctx->user, tmp->security_ctx->user))
     {
-      tmp->awake(only_kill_query ? THD::KILL_QUERY : THD::KILL_CONNECTION);
-      error=0;
+      if (query_id_arg != 0 && query_id_arg != tmp->query_id)
+      {
+        error= ER_KILL_QUERY_ID_MISMATCH;
+        my_error(error, MYF(0), id);
+      }
+      else
+      {
+        tmp->awake(only_kill_query ? THD::KILL_QUERY : THD::KILL_CONNECTION);
+        error= 0;
+      }
     }
     else
-      error=ER_KILL_DENIED_ERROR;
+    {
+      error= ER_KILL_DENIED_ERROR;
```

```
+       my_error(error, MYF(0), id);
+     }
+
      pthread_mutex_unlock(&tmp->LOCK_delete);
+     /* release the lock now only if we were checking for specific query */
+     if (query_id_arg)
+       VOID(pthread_mutex_unlock(&LOCK_thread_count));
    }

    if (!error)
      send_ok(thd);
-   else
-     my_error(error, MYF(0), id);
  }


diff -Nur -x bdb mysql-5.0.26/sql/sql_show.cc mysql-5.0.26.kill_query/sql/sql_show.cc
--- mysql-5.0.26/sql/sql_show.cc      2006-10-04 04:24:21.000000000 -0700
+++ mysql-5.0.26.kill_query/sql/sql_show.cc 2006-10-24 16:41:22.000000000 -0700
@@ -1266,6 +1266,7 @@
    uint   command;
    const char *user,*host,*db,*proc_info,*state_info;
    char *query;
+   query_id_t query_id;
  };

  #ifdef HAVE_EXPLICIT_TEMPLATE_INSTANTIATION
@@ -1293,6 +1294,11 @@
    field->maybe_null=1;
    field_list.push_back(field=new Item_empty_string("Info",max_query_length));
    field->maybe_null=1;
+   if (verbose)
+   {
+     field_list.push_back(field=new Item_int("Query_Id",FIELD_TYPE_LONGLONG));
+     field->maybe_null= 1;
+   }
    if (protocol->send_fields(&field_list,
                              Protocol::SEND_NUM_ROWS | Protocol::SEND_EOF))
      DBUG_VOID_RETURN;
@@ -1366,7 +1372,8 @@
            */
           uint length= min(max_query_length, tmp->query_length);
           thd_info->query=(char*) thd->strmake(tmp->query,length);
-        }
+       }
+       thd_info->query_id= tmp->query_id;
         thread_infos.append(thd_info);
       }
     }
@@ -1392,6 +1399,13 @@
       protocol->store_null();
     protocol->store(thd_info->state_info, system_charset_info);
     protocol->store(thd_info->query, system_charset_info);
+    if (verbose)
+    {
+      if (thd_info->query)
+        protocol->store(thd_info->query_id);
+      else
+        protocol->store_null();
+    }
     if (protocol->write())
       break; /* purecov: inspected */
   }
diff -Nur -x bdb mysql-5.0.26/sql/sql_yacc.yy mysql-5.0.26.kill_query/sql/sql_yacc.yy
--- mysql-5.0.26/sql/sql_yacc.yy      2006-10-04 04:24:23.000000000 -0700
+++ mysql-5.0.26.kill_query/sql/sql_yacc.yy 2006-10-23 18:38:14.000000000 -0700
@@ -6883,19 +6883,31 @@
  /* kill threads */

  kill:
-       KILL_SYM { Lex->sql_command= SQLCOM_KILL; } kill_option expr
```

```
+       KILL_SYM
        {
-         LEX *lex=Lex;
+         LEX *lex= Lex;
+         lex->sql_command= SQLCOM_KILL;
          lex->value_list.empty();
-         lex->value_list.push_front($4);
+       }
+       kill_option expr kill_with_query_option
+       {
+         Lex->value_list.push_front($4);
        };

+
+
 kill_option:
        /* empty */     { Lex->type= 0; }
        | CONNECTION_SYM { Lex->type= 0; }
        | QUERY_SYM      { Lex->type= ONLY_KILL_QUERY; }
          ;

+kill_with_query_option:
+       /* empty */
+       | WITH QUERY_SYM expr { Lex->value_list.push_front($3); }
+       ;
+
+
 /* change database */

 use:   USE_SYM ident
```