For a long time to suffer with this issue. Literature on the Internet a lot. I had to ask around at different forums, deeper digging in the manual and explain to himself some weird moments. So, short of stored procedures in MySQL.

Stored procedures - what is it?

Stored procedures have appeared since version 5 MySQL. They allow you to automate complex processes at the level of MySQL, rather than using external scripts for this. This gives us the highest rate of execution, as we are not chasing a large number of requests, but only once call a particular procedure (or function).

What do I need? Install MySQL server version 5 or higher. Procedures can be created as requests, such as via the command line of MySQL, but for convenience recommend to download the MySQL GUI Tools (http://dev.mysql.com/downloads/gui-tools). This package includes three programs - MySQL Administrator, MySQL Query Browser and MySQL Migration Toolkit. We need the first two. (Although you can get one MySQL Query Browser, but all those $$ in stored procedures can sometimes be confusing.)

The first stored procedure

So, open the MySQL Administrator, connect to the MySQL server and create a new schema (database): click **Catalogs**, select **Create New Schema** for **Schemata** (Ctrl+N). Name it as something (like db). Open the newly created schema, select the tab
Stored procedures
and then click Create Stored Proc. Call your procedure procedure1. In the body of the procedure (between
BEGIN
and
END
) enter the following:

{code}

```
SELECT "This is my stored procedure";
```

{/code}

And click **Execute SQL** - procedure is created. Open the MySQL Query Browser, select your schema (db) and enter the following query:

{code}

```
CALL procedure1();
```

{/code}

Voila! Congratulations.

Variables in MySQL

In order to extract some benefit from stored procedures in MySQL, you have to work with variables. Since it is beyond the scope of this article, I'll show a few examples.

Simple variables:

{code}

```
DECLARE iVar INT DEFAULT 0;
```

SET iVar = 5;

SELECT * FROM `data` WHERE `id` = iVar;

DECLARE iVar INT DEFAULT 0;

SELECT count(*) INTO iVar FROM `data`;

{/code}

System Variables:

{code}

SET @iVar = 5;

SELECT @iVar;

{/code}

The difference between simple and system variables in that system variables are accessible from outside the stored procedure. That is, to remove any information necessary to use the system, and the variables that are needed only inside the procedure should be simple.

Parameters in Stored Procedures

Here too, everything is easy. Change the first line that declares the procedure itself:

{code}

CREATE PROCEDURE `procedure1`(IN iInput1 INT, IN iInput2 INT)

{/code}

Here, the keyword **IN** indicates that the parameter is input. Further work with this option as a normal variable inside a procedure:

{code}

SELECT * FROM `data` WHERE `id` = iInput1 AND `id2` = iInput2;

{/code}

Conditions, loops. IF THEN ELSE, WHILE.

Conditions and loops you will definitely need to write complex stored procedures, but to dwell on this subject I will not. I think at least some programming skills you have, so it will show just the syntax.

{code}

IF condition

THEN

action;

ELSE

action;

END IF;

WHILE condition

DO

action;

END WHILE;

{/code}

A simple example

One of the good applications stored procedures - if you need to combine multiple queries into one, such as adding a theme in the forum and increase the total number of subjects. Assume table threads:

{code}

CREATE TABLE `threads` (

`id` INT NOT NULL AUTO_INCREMENT ,

`title` VARCHAR(255) NOT NULL,

`tag` VARCHAR(255) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE = MYISAM;

{/code}

Here we have a title heading a new topic. Well, the table, for example with different statistical variables site, including the total number of those in uniform.

{code}CREATE TABLE `variables` (

`id` INT NOT NULL AUTO_INCREMENT,

`name` VARCHAR (255) NOT NULL,

`value` INT NOT NULL DEFAULT 0,

PRIMARY KEY (`id`)

) ENGINE = MYISAM;{/code}

Here everything seems clear, let us there is a record with name = threads and value = 0. Create a new stored procedure procedure2.

{code}CREATE PROCEDURE `procedure2` (IN sTitle VARCHAR (255))

BEGIN

INSERT INTO `threads` (`title`) VALUES (sTitle);

UPDATE `variables` SET `value` = `value` + 1 WHERE `name` = 'threads';

END {/code}

Explain nothing special, just the two queries combined into one. Now we can call this procedure as follows:

{code}CALL procedure2 ('My new thread');{/code}

Thus, instead of passing two or more requests (for example through php), we can send one - optimization, clean code and can change at any time without affecting other scripts.

MySQL Cursors

Cursor keys to go through all the query results. On the theory difficult to explain, and show in practice. Add another table to our database - hits:

{code}CREATE TABLE `tags` (

`Id` INT NOT NULL AUTO_INCREMENT,

`Tag` VARCHAR (255) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE = MYISAM{/code}

Here we will record all the tags of all topics. The stored procedure will look like this:

{code}CREATE PROCEDURE `procedure3` ()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE sTag VARCHAR (255);

DECLARE iCount INT DEFAULT 0;

```
DECLARE rCursor CURSOR FOR

SELECT `tag` FROM `threads` WHERE 1;

DECLARE CONTINUE HANDLER FOR SQLSTATE '02000 'SET done = 1;

OPEN rCursor;

FETCH rCursor INTO sTag;

WHILE done = 0 DO

SELECT COUNT (*) INTO iCount FROM `tags` WHERE `tag` = sTag;

IF iCount = 0 THEN

INSERT INTO `tags` (`tag`) VALUES (sTag);
```

END IF;

FETCH rCursor INTO sTag;

END WHILE;

CLOSE rCursor;

END{/code}

Detail. The procedure goes through each topic, each tag will check on the table tags, and if the tag is missing, it adds.

Cursor for the query SELECT, which selects from the tags of all those (WHERE 1). Once the cursor to announce something like an exception - what to do when the results come to an end (SQLSTATE '02000' means is the end). In this case we write a variable done to subsequently exit the loop.

Open the cursor and get the first record. Next in the cycle - Select the number of matches from the tag table for the current tag and put the result into the variable iCount. If no results, then the INSERT query insert a new tag.

In the end, close cursor and exit procedures. Well, that's all.

Retrieving data.

Remember the system variables and consider another manipulation on our tables - a total number of tags and topics. Proceed directly to the procedure:

{code}CREATE PROCEDURE `procedure4` ()

BEGIN

DECLARE iTags INT DEFAULT 0;

DECLARE iThreads INT DEFAULT 0;

SELECT COUNT (*) INTO iTags FROM `tags`;

SELECT COUNT (*) INTO iThreads FROM `threads`;

SET @ tags = iTags, @ threads = iThreads;

END{/code}

Declare two variables - iTags - number of tags and iThreads - the total number of subjects.

Next, two simple select query to fill our variables. And at the end of assigning values ??to system variables current simple variables. When you call this procedure does not return anything, but after the call, we can assume the desired values ??of system variables:

{code}CALL procedure4 ();

SELECT @ tags, @ threads;{/code}

Conclusion

A conclusion and not be there;) will be happy to answer your questions - write in comments.
{jcomments on}