Transaction-Safe Feature in MySQL Databases

Index

Understanding the MySQL 5.0 Storage Engines

MySQL 5 supports several storage engines (InnoDB, MyISAM, NDB, Merge, Blackhole, Falcon, ...). The common are InnoDB and MyISAM. And this fact has some reason and implications to use one or other feature. MyISAM doesn't support transaction-safe tables, while InnoDB does it. So this is a pre-requirement to go ahead: we'll use InnoDB storage engine.

For more details, please see chapter 14 in MySQL 5.1 manual.

The Tools

Here we'll describe the main SQL tools that will allow us to make our transaction-safe statements.

- **COMMIT** : save the changes made;

- **SAVEPOINT** : create a savepoint where we can return;

- **ROLLBACK** : undo all changes made;

- **ROLLBACK TO SAVEPOINT** : undo all changes made until the savepoint.

Some more stuff you should know

MySQL starts with AUTOCOMMIT mode on by default. This mean all changes are saved one-by-one, in order they are called. So, we must use:

(1) - **AUTOCOMMIT=0**;

or

(2) - **START TRANSACTION**;

BEFORE use the tools described above. The difference between (1) and (2) is using (2) the AUTOCOMMIT mode will be enabled after end the transaction (a COMMIT or a ROLLBACK).

There are some statements that cannot be rolled back. I'll talk about this later.

When a SAVEPOINT is created with the same name as an older one, the older one is deleted.

Let's work a little

Some examples of transaction-safe statements using our SQL tools.

```
START TRANSACTION;
INSERT INTO tb1 id, name, email VALUES (" , 'A Simple Name', 'xxx@zzz.com');
INSERT INTO tb2 id, email VALUES (" , 'xxx@zzz.com');
COMMIT;

SET AUTOCOMMIT = 0;
SAVEPOINT my_point1;
UPDATE tb_money SET amount = amount-300 WHERE client_id = '10';
SAVEPOINT my_point2;
UPDATE tb_money SET amount = amount+300 WHERE client_id = '15';
SELECT amount FROM tb_money WHERE client_id = '10'; // amount = 500
SELECT amount FROM tb_money WHERE client_id = '15'; // amount = 1100
ROLLBACK TO SAVEPOINT my_point2;
SELECT amount FROM tb_money WHERE client_id = '15'; // amount = 800
ROLLBACK TO SAVEPOINT my_point1;
SELECT amount FROM tb_money WHERE client_id = '10'; // amount = 800
COMMIT;
```

Line-by-line this doesn't seems useful, but consider using these tools inside SQL functions,

stored procedures, triggers, scheduled events, subjects of next articles about databases I'll publish.

Another interesting possibility is use transactions in scripts of other programming languages, like C, PHP, Java, Python,..., that allows you iterate upon queries and work easier. This article talk about this from now.

More Tools - Using PHP

PHP has a class that directly supports transactions, the mysqli class. All other restrictions as use InnoDB tables remains valid.

The main idea is work with flags. Let's suppose you have a "join now" form, and the process consists of:

- a INSERT query the clients table of your database;
- send a confirmation email to the client;
- write an entry in a log file kept in the server (containing a time stamp, the client machine ip, and a email hash).

It would be pretty good if none of these steps are made if one of they fail, mainly if the confirmation email wasn't sent. A simple XML log file is a good idea to detect tries of code injection, and it is as important as your database.

See the script below:

```
<?
    /* getting the form variables */
    $name = $_POST['name'];
    $email = $_POST['email'];

    /* making to insert in log */
    $date = date("r");
    $ip = $_SERVER['REMOTE_ADDR'];
    $hash = md5($email);

    /* instantiating the class mysqli and doing what needs to be done */
    $my = new mysqli("localhost", "mysql_user", "mysql_password", "clients");
```

```
	$my->autocommit(FALSE);
	$sql = "INSERT INTO clients id, name, email VALUES ('', '$name' , '$email')";
	$my->query($sql);

	/* setting the mail variables, send it, and store the result in a flag */
	...
	$x = mail($email, $subject, $message);
	/* use a function or a method to insert the data in the log file */
	/* later I'll post in my blog how do it */
	$y = addToLog("log.xml", $date, $ip, $hash, $email); //returns true case successfully do it or false case an error occur

	/* And now... we're are  expecting for */
	if ($x == TRUE && $y == TRUE) {
		$my->commit();
		$my->close();
		echo "Success !";
	} else {
		$my->rollback();
		$my->close();
		echo "Nothing has been done !";
	}
?>
```

Alright, it is a simplified example, but it ilustrates how to use transactions.

IMPORTANT: There aren't directly support to SAVEPOINT features in this class. But you can use it like an normal query.

Not all can be undone

There are some SQL statements that cannot be undone with ROLLBACK, or ROLLBACK TO SAVEPOINT statements, because they implicitly cause a COMMIT, and you must avoid use they inside your transactions.

Technically speaking, these statements are from DDL (Data Definition Language) group, i.

e. statements used for alter the structure of the database and its components (tables, functions, stored procedures, triggers, scheduled events, etc...).

These statements are listed below (similar or alias are excluded, but they have the same effect):

ALTER FUNCTION,

ALTER PROCEDURE,

ALTER TABLE,

BEGIN,

CREATE DATABASE,

CREATE FUNCTION,

CREATE INDEX,

CREATE PROCEDURE,

CREATE TABLE,

DROP DATABASE,

DROP FUNCTION,

DROP INDEX,

DROP PROCEDURE,

DROP TABLE,

LOAD MASTER DATA,

LOCK TABLES,

LOAD DATA INFILE,

RENAME TABLE,

SET AUTOCOMMIT=1,

START TRANSACTION,

TRUNCATE TABLE,

UNLOCK TABLES.

References

- PHP 5 Manual, chapter VI - LXXX.
- MySQL 5.1 Manual, chapters 13.4, 14.2, 14.5.10
- Wikipedia: http://en.wikipedia.org/wiki/SQL
- To this article was used a MySQL 5.0, Apache 2.0 servers and PHP 5.0.

About this article

This article was originally written by Davis L. P. Peixoto.

Date: May 04, 2007