MySQL\_for\_Python\_Albert\_c14

isaster Recovery

In late September 2009, users of T-Mobile's Sidekick smartphone began noticing

outages in the service. The Sidekick was designed to push the envelope of technology

by storing most of its data remotely and requiring network connectivity for almost

every function it had. Come October, the company that ran the data servers for the

service, Danger, had still not restored the information. Danger had been taken over

by Microsoft in February 2007, and most of the employees that started the service

had left the company, leaving Microsoft's own employees to field any problems.

T-Mobile consequently informed their customers on 10th October that data recovery

was not forthcoming, and that they believed the data on all 800,000 customers was

irrecoverably lost.

The saga was further hampered by what Reuters, citing an email from Microsoft,

called "a confluence of errors from a server failure that hurt its main and backup

databases supporting Sidekick users." The commercial damage was significant:

T-Mobile pulled the Sidekick device off the shelves and could not sell them. By 15th

October, Microsoft had found and was replacing "most, if not all, customer data", but

the damage was done. Microsoft's reputation for data management was seriously

damaged, and T-Mobile had sustained substantial losses in the world's largest

national economy.

One thing that is certain in computing is that hardware will fail even if the software

is written soundly. For this reason, a disaster recovery plan should be implemented

for every database server. Even if you are not the administrator of the server, this

chapter will show you how to back up the data you use. By the end of this chapter,

we will have covered:

•When to implement one of several kinds of database backup plans

•What methods of backup and disaster recovery MySQL supports

•How to use Python to back up databasesDisaster Recovery

The purpose of the archiving methods covered in this chapter is to allow you, as

the developer, to back up databases that you use for your work without having

to rely on the database administrator. As noted later in the chapter, there are

more sophisticated methods for backups than we cover here, but they involve

system-administrative tasks that are beyond the remit of any development post

and are thus beyond the scope of this book.

Every database needs a backup plan

When archiving a database, one of the critical questions that must be answered is

how to take a snapshot backup of the database without having users change the data

in the process. If data changes in the midst of the backup, it results in an inconsistent

backup and compromises the integrity of the archive. There are two strategic

determinants for backing up a database system:

•Offline backups

•Live backups

Which you use depends on the dynamics of the system in question and the import

of the data being stored. In this chapter, we will look at each in turn and the way to

implement them.

Offline backups

Offline backups are done by shutting down the server so the records can be archived

without the fear of them being changed by the user. It also helps to ensure the server

shut down gracefully and that errors were avoided. The problem with using this

method on most production systems is that it necessitates a temporary loss of access

to the service. For most service providers, such a consequence is anathema to the

business model.

The value of this method is that one can be certain that the database has not changed

at all while the backup is run. Further, in many cases, the backup is performed

faster because the processor is not simultaneously serving data. For this reason,

offline backups are usually performed in controlled environments or in situations

where disruption is not critical to the user. These include internal databases, where

administrators can inform all users about the disruption ahead of time, and small

business websites that do not receive a lot of traffic.

Offline backups also have the benefit that the backup is usually held in a single file.

This can then be used to copy a database across hosts with relative ease.

[ 394 ]Chapter 14

Shutting down a server obviously requires system administrator-like authority. So

creating an offline backup relies on the system administrator shutting down the

server. If your responsibilities include database administration, you will also have

sufficient permission to shut down the server.

Live backups

Live backups occur while the server continues to accept queries from users, while it's

still online. It functions by locking down the tables so no new data may be written to

them. Users usually do not lose access to the data and the integrity of the archive, for

a particular point in time is assured.

Live backups are used by large, data-intensive sites such as Nokia's Ovi

services and Google's web services. However, because they do not always

require administrator access of the server itself, these tend to suit the backup

needs of a development project.

Choosing a backup method

After having determined whether a database can be stopped for the backup, a

developer can choose from three methods of archiving:

•Copying the data files (including administrative files such as logs and

tablespaces)

•Exporting delimited text files

•Backing up with command-line programs

Which you choose depends on what permissions you have on the server and how

you are accessing the data.

MySQL also allows for two other forms of backup: using the binary log

and by setting up replication (using the master and slave servers).

To be sure, these are the best ways to back up a MySQL database. But,

both of these are administrative tasks and require system-administrator

authority; they are not typically available to a developer. However, you

can read more about them in the MySQL documentation. Use of the

binary log for incremental backups is documented at:

http://dev.mysql.com/doc/refman/5.5/en/point-in-time-

recovery.html

Setting up replication is further dealt with at:

http://dev.mysql.com/doc/refman/5.5/en/replication-

solutions-backups.html

[ 395 ]Disaster Recovery

Copying the table files

The most direct way to back up database files is to copy from where MySQL stores

the database itself. This will naturally vary based on platform. If you are unsure

about which directory holds the MySQL database files, you can query MySQL

itself to check:

mysql> SHOW VARIABLES LIKE 'datadir';

+---------------+-----------------+

| Variable\_name | Value

|

+---------------+-----------------+

| datadir

| /var/lib/mysql/ |

+---------------+-----------------+

Alternatively, the following shell command sequence will give you the

same information:

$ mysqladmin variables | grep datadir

| datadir

| /var/lib/mysql/

|

Note that the location of administrative files, such as binary logs

and InnoDB tablespaces are customizable and may not be in the

data directory.

If you do not have direct access to the MySQL server, you can also write a simple

Python program to get the information:

#!/usr/bin/env python

import MySQLdb

mydb = MySQLdb.connect('<hostname>',

'<user>',

'<password>')

cursor = mydb.cursor()

runit = cursor.execute("SHOW VARIABLES LIKE 'datadir'")

results = cursor.fetchall()

print "%s: %s" %(cursor.fetchone())

Slight alteration of this program will also allow you to query several servers

automatically. Simply change the login details and adapt the output to clarify

which data is associated with which results.

[ 396 ]Chapter 14

Locking and flushing

If you are backing up an offline MyISAM system, you can copy any of the files

once the server has been stopped. Before backing up a live system, however, you

must lock the tables and flush the log files in order to get a consistent backup at a

specific point. These tasks are handled by the LOCK TABLES and FLUSH commands

respectively. When you use MySQL and its ancillary programs (such as mysqldump)

to perform a backup, these tasks are performed automatically. When copying files

directly, you must ensure both are done. How you apply them depends on whether

you are backing up an entire database or a single table.

LOCK TABLES

The LOCK TABLES command secures a specified table in a designated way. Tables

can be referenced with aliases using AS and can be locked for reading or writing.

For our purposes, we need only a read lock to create a backup. The syntax looks

like this:

LOCK TABLES <tablename> READ;

This command requires two privileges: LOCK TABLES and SELECT.

It must be noted that LOCK TABLES does not lock all tables in a database but

only one. This is useful for performing smaller backups that will not interrupt

services or put too severe a strain on the server. However, unless you automate

the process, manually locking and unlocking tables as you back up data can be

ridiculously inefficient.

FLUSH

The FLUSH command is used to reset MySQL's caches. By re-initiating the cache at

the point of backup, we get a clear point of demarcation for the database backup

both in the database itself and in the logs. The basic syntax is straightforward,

as follows:

FLUSH <the object to be reset>;

Use of FLUSH presupposes the RELOAD privilege for all relevant databases. What we

reload depends on the process we are performing. For the purpose of backing up, we

will always be flushing tables:

FLUSH TABLES;

How we "flush" the tables will depend on whether we have already used the LOCK

TABLES command to lock the table. If we have already locked a given table, we can

call FLUSH for that specific table:

FLUSH TABLES <tablename>;

[ 397 ]Disaster Recovery

However, if we want to copy an entire database, we can bypass the LOCK TABLES

command by incorporating the same call into FLUSH:

FLUSH TABLES WITH READ LOCK;

This use of FLUSH applies across the database, and all tables will be subject to the

read lock. If the account accessing the database does not have sufficient privileges

for all databases, an error will be thrown.

Unlocking the tables

Once you have copied the files for a backup, you need to remove the read lock you

imposed earlier. This is done by releasing all locks for the current session:

UNLOCK TABLES;

Restoring the data

Restoring copies of the actual storage files is as simple as copying them back

into place. This is best done when MySQL has stopped, lest you risk corruption.

Similarly, if you have a separate MySQL server and want to transfer a database,

you simply need to copy the directory structure from the one server to another.

On restarting, MySQL will see the new database and treat it as if it had been

created natively. When restoring the original data files, it is critical to ensure the

permissions on the files and directories are appropriate and match those of the

other MySQL databases.

Delimited backups within MySQL

MySQL allows for exporting of data from the MySQL command line. To do so, we

simply direct the output from a SELECT statement to an output file.

Using SELECT INTO OUTFILE to export data

Using sakila, we can save the data from film to a file called film.data as follows:

SELECT \* INTO OUTFILE 'film.data' FROM film;

[ 398 ]Chapter 14

This results in the data being written in a tab-delimited format. The file will be

written to the directory in which MySQL stores the sakila data. Therefore, the

account under which the SELECT statement is executed must have the FILE privilege

for writing the file as well as login access on the server to view it or retrieve it. The

OUTFILE option on SELECT can be used to write to any place on the server that

MySQL has write permission to use. One simply needs to prepend that directory

location to the file name. For example, to write the same file to the /tmp directory

on a Unix system, use:

SELECT \* INTO OUTFILE '/tmp/film.data' FROM film;

Windows simply requires adjustment of the directory structure accordingly.

Using LOAD DATA INFILE to import data

If you have an output file or similar tab-delimited file and want to load it into

MySQL, use the LOAD DATA INFILE command. The basic syntax is:

LOAD DATA INFILE '<filename>' INTO TABLE <tablename>;

For example, to import the film.data file from the /tmp directory into another table

called film2, we would issue this command:

LOAD DATA INFILE '/tmp/film.data' INTO TABLE film2;

Note that LOAD DATA INFILE presupposes the creation of the table into which the

data is being loaded. In the preceding example, if film2 had not been created, we

would receive an error. If you are trying to mirror a table, remember to use the SHOW

CREATE TABLE query to save yourself time in formulating the CREATE statement.

This discussion only touches on how to use LOAD DATA INFILE for inputting data

created with the OUTFILE option of SELECT. But, the command handles text files with

just about any set of delimiters. To read more on how to use it for other file formats,

see the MySQL documentation at:

http://dev.mysql.com/doc/refman/5.5/en/load-data.html

[ 399 ]Disaster Recovery

Archiving from the command line

If you use a MySQL client with a graphical user interface, how you back up will

depend on that client. Depending on your platform, MySQL ships with one or both

of the following command-line programs used for archiving:

mysqldump

On every MySQL server, you will find the program mysqldump. On Windows, it

is usually located in the same directory as the MySQL server executable. On Unix

variants, it will be in /usr/bin/.

This program functions like an automated MySQL client. It accepts login credentials

from the command line and, based on the options you give it, it will output the script

necessary to recreate the database you specify. The basic syntax is:

mysqldump -u <username> -p --database <dbname>

After providing the username and database name, you are prompted for the

password. This is because you have not specified it even though you have indicated

with the -p flag that you will use a password to log in. This is the more secure way

of running mysqldump. Optionally, you can state the password explicitly after the

-p flag, but this is not advisable as it then enters into your shell command history

as plain text.

Viewing the backup file

Running this command and providing the appropriate password will cause a dump

of the specified database. This will almost certainly run off your screen. To save it to

a file, use either a greater than sign (>) or the option --result-file= followed by

the filename. A dump of the sakila database would read as follows:

mysqldump -u skipper -p --result-file=sql.dump --database sakila

After that process is finished, you could open sql.dump in your favorite text editor to

see the following:

-- MySQL dump 10.11

--

-- Host: localhost

Database: sakila

-- ------------------------------------------------------

-- Server version

5.0.51a-3ubuntu5.4

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

...

/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;

[ 400 ]Chapter 14

--

-- Current Database: `sakila`

--

CREATE DATABASE /\*!32312 IF NOT EXISTS\*/ `sakila` /\*!40100 DEFAULT

CHARACTER SET latin1 \*/;

USE `sakila`;

--

-- Table structure for table `actor`

--

DROP TABLE IF EXISTS `actor`;

SET @saved\_cs\_client

= @@character\_set\_client;

SET character\_set\_client = utf8;

CREATE TABLE `actor` (

`actor\_id` smallint(5) unsigned NOT NULL auto\_increment,

...

KEY `idx\_actor\_last\_name` (`last\_name`)

) ENGINE=InnoDB AUTO\_INCREMENT=201 DEFAULT CHARSET=utf8;

So, the dump file contains all of the MySQL commands necessary to create the

infrastructure of the database. If you read on, you will soon encounter the following:

--

-- Dumping data for table `actor`

--

LOCK TABLES `actor` WRITE;

/\*!40000 ALTER TABLE `actor` DISABLE KEYS \*/;

INSERT INTO `actor` VALUES (1,'PENELOPE','GUINESS','2006-02-15 04:34:3

3'),(2,'NICK','WAHLBERG','2006-02-15 04:34:33'),(3,'ED','CHASE','2006-

02-15 04:34:33'),(4,'JENNIFER','DAVIS','2006-02-15 04:34:33'),(5,'JOHN

NY','LOLLOBRIGIDA','2006-02-15 04:34:33')...

The necessary INSERT commands are also included. The dump file is a single file

backup of the database.

[ 401 ]Disaster Recovery

Other options

In addition to the options discussed previously, mysqldump supports several other

flags. The most commonly used include:

•--all-databases: Dump all tables in all databases

•--compact: Produce more compact output

•--databases: Dump several databases

•--dump-date: Include dump date with the "Dump completed on" comment if

--comments is given

•--flush-logs: Flush the MySQL server log files before starting the dump

•--flush-privileges: Emit a FLUSH PRIVILEGES statement after dumping

•--help: Display help message and exit

•--host: Host to connect to (IP address or hostname)

•--ignore-table=db\_name.tbl\_name: Do not dump the given table

•--lock-all-tables: Lock all tables across all databases

•--lock-tables: Lock all tables before dumping them

•--log-error=file\_name: Append warnings and errors to the named file

•--opt: Shorthand for --add-drop-table --add-locks --create-options

--disable-keys --extended-insert --lock-tables --quick --set-

charset.

•--quick: Retrieve rows for a table from the server a row at a time

•--result-file=file: Direct output to a given file

•--single-transaction: Includes a BEGIN SQL statement before the data

•--skip-triggers: Do not dump triggers

•--tab=path: Produce tab-separated data files

•--tables: Override the --databases or -B option

•--triggers: Dump triggers for each dumped table

•--verbose: Verbose mode

•--version: Display version information and exit

•--where='where\_condition': Dump only rows selected by the given

•--xml: Produce XML output

the MySQL database

from the server

WHERE condition

[ 402 ]Chapter 14

Complete documentation on mysqldump can be found at http://dev.mysql.

com/doc/refman/5.5/en/mysqldump.html

Restoring the data

If you use mysqldump to create a standard MySQL backup as discussed above,

you can restore the data using the SOURCE command we used in Chapter 12. For

example, backing up sakila would use the following command:

mysqldump -u skipper -p --opt sakila > sakila.sql

After the password is entered and the database backed up to sakila.sql, we can

login and restore the data:

SOURCE sakila.sql;

It is worth noting that the file created by mysqldump does not always create the

database. If you want to specify the database in question, use either of the flags --

database or --databases. If you are transferring the backup to another server, you

will need to create a database for it first. Note, however, that the database used for

the backup need not have the same name as the original. It is possible to import a

backup of sakila into a database named alikas. Within the database system, the

only time database names become critical is in the use of cross-database references

such as triggers. Naturally, if you change the name of the database, you will need to

change the references used in all of your calling applications, as well.

Triggers are MySQL procedures that initiate an action based on an event

in a table. They allow information in one table to be updated, inserted, or

deleted based upon data being inserted, updated, or deleted in another

table. For the MySQL documentation on triggers, see: http://dev.

mysql.com/doc/refman/5.5/en/triggers.html

Then you will need to ensure the greater server environment is mirrored for a

functioning (as opposed to simply stored) backup.

mysqlhotcopy

In addition to mysqldump, Unix, Linux, and NetWare servers also support

mysqlhotcopy. This program is a Perl script that backs up live databases. Where

mysqldump functions like a MySQL client, mysqlhotcopy is a server administration

program. It must be run on the same host as the one that runs the database; it cannot

be run remotely.

[ 403 ]Disaster Recovery

It is worth noting that mysqlhotcopy only works on MyISAM and

Archive tables. Therefore, you cannot use it to back up other types

such as InnoDB.

mysqlhotcopy works by copying the salient files for a database to the directory of

your choice. Consequently, it will only work if the user who invokes it has read

access to those files. This will always be the root administrator of the system but,

depending on your setup, it may include others. In the following examples, it is

assumed that the one has appropriate permissions to access the files.

The most basic syntax of mysqlhotcopy is:

mysqlhotcopy <database name> <path for backup>

While that is the most basic call of the program, it will almost never work. You must

know the username and password of a user that has access to the database you are

archiving. Therefore, it is more commonly called like this:

mysqlhotcopy -u skipper -p secret sakila /path/to/a/directory/

Assuming the credentials and access permissions are valid, mysqlhotcopy will

then create a directory in which it stores copies of the files used for the database

you specified.

In addition to the syntax shown above, the following options are available with

mysqlhotcopy:

•–addtodest: Do not rename target directory (if it exists); merely add files to it

•–allowold: Do not abort if a target exists; rename it by adding an \_old suffix

•–checkpoint=db\_name.tbl\_name: Insert checkpoint entries

•–chroot=path: Base directory of the chroot jail in which mysqld operates

•–debug: Write a debugging log

•–dryrun: Report actions without performing them

•–flushlogs: Flush logs after all tables are locked

•–help: Display help message and exit

•–host=host\_name: Connect to the MySQL server on the given host

•–keepold: Do not delete previous (renamed) target when done

•–noindices: Do not include full index files in the backup

•–password[=password]: The password to use when connecting to the server

•–port=port\_num: The TCP/IP port number to use for the connection

[ 404 ]Chapter 14

•–quiet: Be silent except for errors

•–regexp: Copy all databases with names that match the given regular

•–resetmaster: Reset the binary log after locking all the tables

•–resetslave: Reset the master.info file after locking all the tables

•–socket=path: For connections to localhost

•–tmpdir=path: The temporary directory

•–user=user\_name: The MySQL username to use when connecting to

•–version: Display version information and exit

expression

the server

It should be noted that mysqlhotcopy is still in beta. Therefore, new functionality

will be added, and this list is therefore not exhaustive. A convenient way of accessing

the options available through your copy of mysqlhotcopy is to use either man or

perldoc from a command-line:

man mysqlhotcopy

Otherwise, you can use:

perldoc mysqlhotcopy

Either of these commands will give you the manual page for your local version.

Restoring from mysqlhotcopy is a matter of copying the directory that holds the file

archive to its appropriate location on your server. This is usually /var/lib/mysql

followed by the database name. If in doubt, see how to access this information for

your installation of MySQL in the section Copying the Table Files above.

Backing up a database with Python

As we have seen, there are three methods of archiving a MySQL database that a

developer can use:

•Copying the MySQL table files directly

•Exporting data to a delimited text file

•Creating a dumpfile

The first and last of these require special permissions on the server. To use Python

to manage the backup merely automates the process but still requires you to have

access beyond SELECT.

[ 405 ]Disaster Recovery

Using MySQLdb, however, we can export the data with only basic privileges. Simply

store the results of the SELECT statement into a variable, format it appropriately, and

write it to a file.

Summary

It has been a long road from the beginning of this book. We have gone from merely

installing MySQL for Python to doing some pretty sophisticated things with it. We

have moved beyond merely using the database to using Python to do programmatic

administration.

In this chapter, we have seen several ways to back up and restore a MySQL database.

We have looked at:

•When it is advisable to backup a running system instead of shutting it down

•The procedure for taking a snapshot of a running MySQL server

•What backup methods are available to a python database developer

•Which privileges are required when using certain archiving methods

From here, the sky is the limit for you. You have the Python and MySQL foundation

necessary to design and develop even large, database-driven applications.

Naturally, there is more to learn about MySQL. That is for the coming chapters of

the book.

[ 406 ]Index

Symbols