Dump and Restore a MySQL Database: Observations on Performance

Introduction

This article is based on Steve Moore's experience with making a copy of the TCIA production database to use for testing. My goal was to have a duplicate of the production data on a test MySQL server to protect the production data during development (standard stuff). The database dump was 6 GB. My first attempt at a restore failed, so I performed some additional research. This article is based on google + trial/error; your mileage may vary.

Standard Backup Restore

The standard backup and restore procedure for MySQL would use these MySQL commands from the command line (Linux):

```
mysqldump -u root -p[root_password] [database_name] > dumpfilename.sql
```

```
mysql -u root -p[root_password] < dumpfile.sql
```

Restore Issue on Large Database

The issue I ran into was that the restore operation was taking a long time (seemed to be greater than 24 hours) and then just stopped. There were no error messages that I could find in the output from the MySQL interpreter; it was as if the import just stopped working.

Two Part Solution

Split the Dump File

I was concerned that the MySQL interpreter was not able to handle the entire dump file in one operation. I have no direct evidence of this, but I thought I would break up the dump into separate commands and restore the database in pieces. The split command is a standard linux command that is not related to MySQL:

```
split --suffix-length=3 --numeric-suffixes --verbose --lines=250 dumpfile.mysql prefix.
```

This splits the database dump into separate files:

- The file names begin with "prefix."; you can choose any prefix. I used "DB." to remind me this is a database dump. The period is not required; it helped me distinguish between the prefix and generated suffix.
- The suffix length is three characters; you can choose a different length.
- I chose numeric suffixes. When you combine this with the suffix length of three, the files are named:
  - DB.000
  - DB.001
  - DB.002
  - ...
- Each output file contains 250 lines of text. The last file will likely contain less than 250 lines.
  - This is not a magic number. I came to it by trial and error to get the individual files to be of reasonable length.
  - The output files are 250 lines, and many of the lines are very long. The MySQL dump produces long insert statements; the split command reproduces those lines and does not add extra formatting.

The split command does not know anything about SQL statements. This procedure can split statements across files. Because we are going to process each file individually with the MySQL interpreter, this is a minor issue. The last SQL statement in one of the DB.xxx files may be incomplete and need text from the next file. I was lucky in that this only happened a few times in the files that contained commands to create tables. The MySQL dump created a single CREATE TABLE command that extended across several lines. The long insert statements were always complete on a single line and were not disturbed by the split function,.

The simple fix for this was to examine the tail of the output files. For those few files where the SQL was truncated, I copied the appropriate text from the beginning of the next file and repaired the truncated SQL. In the next file that started in the middle of an SQL statement, I commented out those few lines.

Different Mechanism to Restore Database

I now have a collection of individual SQL files that can be interpreted by the MySQL interpreter. I could have fed them into MySQL one by one, but found this other trick through google. I wrote this SQL statement and ran this through the MySQL interpreter:

```
drop database nciaa;

create database ncia;

source DB.000;

source DB.001;

source DB.002;

...
```

```
source DB.033;
```

When I ran this through the MySQL interpreter, it took approximately 100 minutes to process the entire script.

Followup

It would be interesting to run some more experiments to see:

- How long it takes to run the full dump through the MySQL interpreter using the standard procedure (I gave up after a failed attempt).
- How long it takes to run the individual DB.xxx files through the MySQL interpreter using the standard "mysql .... < DB.000" procedure

It would also be interesting to do some more reading in the MySQL documentation on the "source" statement to understand why this seemed to perform better. As mentioned above, this is all trial and error.