Title

MariaDB Administration Part 3 - MariaDB Database Maintance

Author

Bernard Szlachta (NobleProg Ltd)

Subfooter

MariaDB Administration Part 3 - MariaDB Database Maintance

Slide Show

Contents

MODULE 6

- Backup and Recovery
- Point-in-Time Recovery
- Maintenance and Crash Recovery
- myisamchk Syntax and Options
- Getting Table Information
- MySQL Local Setting
- National Characters and Sorting
- MySQL Server Time Zone

Backup and Recovery

- Logical versus physical backups
- Online versus offline backups
- Local versus remote backups
- Snapshot backups
- Full versus incremental backups
- Point-in-time recovery
- Backup scheduling, compression, and encryption
- Table maintenance

Logical versus physical backups

Logical backups

- CREATE DATABASE, CREATE TABLE statements and content (INSERT statements, text files or XML).

Physical backups

- - raw copies of the directories and files

Logical vs. Physical

Logical

- data pass MySQL server
- is much slower than physical
- output is usually larger
- backup and restore granularity (all databases, database, specific tables)
- does not include log or configuration files
- machine independent and highly portable.
- the server is not taken offline

Physical

- exact copies of database directories and files
- faster than logical
- more compact output

- database level granularity (depend of the engine)
- InnoDB table shares file storage with other InnoDB tables.)
- can include any related files such as log or configuration files.
- portable only to other machines that have identical or similar hardware
- can be performed while the MySQL server is not running. If the server is running, you need to flush buffers and lock the tables

Logical Backup - Exercises

- Use MySQL Admin GUI tool
- Dump and restore with mysqldump and mysql
- Use SELECT ... INTO OUTFILE statement and LOAD DATA INFILE statement or the mysqlimport client

MySQL useful dump options

```
mysqldump [options] db_name [tables] > tabledump.sql
mysqldump [options] --databases db_name1 [db_name2 db_name3...]   > databasesdump.sql
mysqldump [options] --all-databases  > alldbsdump.sql
```

- --opt
- --compact

Restoring a backup

1. source backup.sql
2. cat backup.sql | mysql
3. mysql < backup.sql
4. copy/paste

Exercise

1. Dump and restore a database using both options (--opt and --compact)
2. (Linux only) Compare the time. Use time command to measure the results.
3. Using mysqldump create a full copy of NP database to NewNP.
4. Using mysqldump create a copy of the *emp* table and call it *old_emp*.

Exercises hint

```
mysqldump world Country | sed s/"Country"/"CopyOfCountry"/  | mysql world
```

Physical Backup

- cp, scp, tar, rsync
- mysqlhotcopy (cp for restore) for MyISAM (linux only)
- ibbackup (ibback for restore) for InnoDB (commercial version only)
- START BACKUP (ndb_restore) for NDB tables (outside the scope of this course)

Exercise

- Backup and restore the database with mysqlhotcopy and cp

Online versus offline backups

Online

- Less intrusive to other clients
- Care must be taken to impose appropriate locking so that data modifications do not take place that compromise backup integrity
-

Offline

- Affects clients adversely because the server is unavailable during backup
- Simpler backup procedure because there is no possibility of interference from client activity

Local versus remote backups

Local

- mysqldump can connect to local or remote servers.
- SQL output (CREATE and INSERT statements), local or remote dumps can be done
- delimited-text output (with the --taboption), data files are created on the server host
- mysqlhotcopy performs only local backups

Remote

- SELECT ... INTO OUTFILE can be initiated from a remote client host, but the output file is created on the server host
- Physical backup methods typically are initiated locally on the MySQL server host so that the server can be taken offline, although the destination for file copies might be remote.

Fast Remote Backup Example

- It is possible speed the process (from 6 hours to 42 seconds) up using compression and ssh tunneling

```
ssh production.nobleprog.net "mysqldump hitra | bzip2 -1" |  bzip2 -d > hitra_backup.tgz
```

Exercise (optional)

- Dump the database of the person sitting next to you using

```
mysqldump -h {ipoftheperson}  > file
```

- Dump the database using ssh an pbzip command as described in the previous slide
- Measure and compare the time in both cases using "time" command

Full versus incremental backups

- An incremental backup consists of the changes made to the data since the full backup
- Incremental backups are made possible by enabling the server's binary log, which the server uses to record data changes
- One-way replication is an example of almost real-time backup

MODULE 7 - MySQL Log Files

- Log Output Destinations
- Error Log
- General Query Log
- Update Log

- Binary Log
- Slow Query Log
- MySQL Log Files

Log File Maintenance and Rotation

Log Output Destination and other options

- Turn logging on/of
- general_log=[{ON,OFF}]
- slow_query_log =[{ON,OFF}]
- Specify the path and filename
- general_log_file and slow_query_log_file
- if path is relative is starts in data directory
- Choosing Loging Format
- log-output=[TABLE,FILE,NONE]
- logging to tables incurs significantly more server overhead
- The session sql_log_off variable can be set to ON or OFF to disable or enable general query logging for the current connection.
-

The General Query Log

- Logs when clients connect or disconnect and each SQL statement received from clients
- Statements are written in the order that mysqld receives them, which might differ from the order in which they are executed.
- It contrasts to the binary log, for which statements are written after they are executed but before any locks are released
- You can disable the general query log at runtime:
- SET GLOBAL general_log = 'OFF'
- SET GLOBAL general_log = 'ON'
- The session sql_log_off variable can be set to ON or OFF to disable or enable general query logging for the current connection

Exercise

- Enable general log
- disable logging without restarting server

The Error Log

- The error log contains
- information when mysqld was started and stopped
- critical errors that occur while the server is running.
- tables which need to be automatically checked or repaired
- You can specify the filename--log-error[=file_name]
- If no file_name value is given, mysqld uses the name host_name.err in the data directory
- The log_warnings system variable can be used to control warning logging to the error log (Enabled by default)
- If the value is greater than 1, aborted connections are written to the error log
- Mysqld_safe script can send the messages to syslog (Unix-like)

Slow Query Log

- Consists of all SQL statements:
- that took more than long_query_time seconds to execute and
- required at least min_examined_row_limit rows to be examined
- that do not use indexes are logged in the slow query log if the --log-queries-not-using-indexes option is specified

- the --log-slow-admin-statements server option enables you to request logging of slow administrative statements such as OPTIMIZE TABLE, ANALYZE TABLE, and ALTER TABLE to the slow query log

Slow Query Log

- For runtime control the global slow_query_log and slow_query_log_file system variables can be used
- You can process a slow query log file using the mysqldumpslow command to summarize the queries that appear in the log.

Test your setup using

```
select benchmark(100000000,1+1);
```

Exercise

- Enable Slow Query Log
- Log all queries with execution time longer than 1 second
- Analyze the queries using mysqldumpslow

Server Log Maintenance

- mysql-log-rotate (RedHat) automate log maintance
- In Debian/Ubuntu mysql-server package installs log rotate script (analyze /etc/logrotate.d/mysql-server script)
- For the binary log, you can set the expire_logs_days system variable to expire binary log files automatically after a given number of days
- You can force MySQL to start using new log files by flushing the logs. (FLUSH LOGS, mysqladmin flush-logs, mysqladmin refresh,mysqldump --flush-logs, mysqldump --master-data command)
- The binary log is flushed when its size reaches the value of the max_binlog_size

Server Log Maintenance

A log flushing operation does the following:

- General query and slow query - the server closes and reopens log files
- Binary logging - the server closes the current binary log file and opens a new log file with the next sequence number
- Error log - renames the error log with the suffix -old and creates a new empty error log file.

To cause new general query and slow query log files to be created on Unix, rename the current logs before flushing them

- mv mysql.log mysql.old
- mysqladmin flush-logs

You can disable the general query log or slow query log at runtime:

- SET GLOBAL general_log = 'OFF';
- SET GLOBAL slow_query_log = 'OFF';

What is Binary Log?

- contains all statements that update data (or possible could update the data)
- stored in the form of “events” that describe the modifications
- logs how long each statement took

Binary Log Purposes

Replication

- The binary log is used on master replication servers as a record of the statements to be sent to slave servers. The master server sends the events contained in its binary log to its slaves, which execute those events to make the same data changes that were made on the master.

Point-in-time recovery

- After a backup file has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup.

Point-in-time recovery

- Recovering first from the backup files to restore the server to its state when the backup was made,
- Re-executing changes in subsequently written binary log files to redo data modifications up to the desired point in time
- Because the output of mysqlbinlog includes SET TIMESTAMP statements before each SQL statement recorded, the recovered data and related MySQL logs will reflect the original times at which the transactions were executed

PiT Recovery Steps and Tools part 1

- Turn on binary log

```
[mysqld]
log-bin=on
```

- Dump the database
- mysqldump > backup`date +%Y%m%d_%H%M%S`.sql
- Modify rows in the emp (useful stuff)
- insert into emp "badrecrod";
- "Accidently" Drop the emp table
- View mysqlbinlog utility to view the binary log
- To find the binary logs

```
mysql> SHOW MASTER STATUS;
```

- To see a listing of all binary log files, use this statement:

```
mysql> SHOW MASTER LOGS;
```

PiT Recovery Steps and Tools part 2

- Drop all databases and restore the backup

```
mysql < backupfile
```

- Select the statements you want to execute

```
mysqlbinlog binlogfile --start-datetime="2014-04-20 9:09:59"
--stop-datetime="2014-04-20 9:59:59" > file.sql
```

- Analyse file.sql
- Execute the commands

```
mysql < file.sql
```

Restoring in a specific log position

- More precise method about which part of the log to recover
- especially if many transactions occurred around the same time as a damaging SQL statement
- To determine the position numbers run the mysqlbinlog command with approximate time
- mysqlbinlog --start-datetime="2014-04-20 9:55:00" --stop-datetime="2014-04-20 10:05:00" /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
- Analyse the mysql_restore.sql file and glean out the positions you are interested in
- Execute the queries

```
mysqlbinlog --start-position=368315 --stop-position=369312 /var/log/mysql/bin.123456 | mysql
```

Maintenance and Crash Recovery

- You can use myisamchk to check, repair, or optimize database tables
- Backup your database before repairing or optimizing
- With myisamchk, you must make sure that the server does not use the tables at the same time so that there is no unwanted interaction between myisamchk and the server
- Statements below can be used directly or by means of the mysqlcheck client program. One advantage of these statements over myisamchk is that the server does all the work.
- CHECK TABLE
- REPAIR TABLE.
- OPTIMIZE TABLE.
- ANALYZE TABLE

myisamchk with external locking disabled

- If you run mysqld with external locking disabled (which is the default), you cannot reliably use myisamchkto check a table when mysqld is using the same table
- If you can be certain that no one will access the tables through mysqld while you run myisamchk, you only have to execute mysqladmin flush-tables before you start checking the tables
- If you cannot guarantee this, you must stop mysqld while you check the tables.
- If you run myisamchk to check tables that mysqld is updating at the same time, you may get a warning that a table is corrupt even when it is not.

myisamchk with external locking enabled

- You can use myisamchk to check tables at any time.
- In this case, if the server tries to update a table that myisamchk is using, the server will wait for myisamchk to finish before it continues.
- If you use myisamchk to repair or optimize tables, you must always ensure that the mysqld server is not using the table (this also applies if external locking is disabled). If you do not stop mysqld, you should at least do a mysqladmin flush-tables before you run myisamchk.
- Your tables may become corrupted if the server and myisamchk access the tables simultaneously.

MyISAM files

- tbl_name.frm Definition (format) file
- tbl_name.MYD Data file
- tbl_name.MYI Index file
-

MyISAM files

- Each of these three file types is subject to corruption in various ways, but problems occur most often in data files and index files
- myisamchk works by creating a copy of the .MYD data file row by row.
- It ends the repair stage by removing the old .MYD file and renaming the new file to the original file name.
- --quick does not create a temporary .MYD file, but instead assumes that the .MYD file is correct and generates only a new index file without touching the .MYD file. This is safe, because myisamchk automatically detects whether the .MYD file is corrupt and aborts the repair if it is.
- --quick --quick option (twice --quick) does not abort on some errors (such as duplicate-key errors) but instead tries to resolve them by modifying the .MYD file.
- Normally the use of two --quick options is useful only if you have too little free disk space to perform a normal repair. In this case, you should at least make a backup of the table before running myisamchk.

Checking MyISAM Tables for Errors

myisamchk stops after the first error it finds the -v (verbose) option which keeps going, up through a maximum of 20 errors

myisamchk tbl_name

- Finds 99.99% of all errors
- cannot find corruption that involves only the data file (which is very unusual)

myisamchk -m tbl_name

- Finds 99.999% of all errors.
- First checks all index entries for errors and then reads through all rows.
- Calculates a checksum for all key values in the rows and verifies that the checksum matches the checksum for the keys in the index tree.

myisamchk -e tbl_name

- Complete and thorough check of all data (-e means “extended check”).
- Does a check-read of every key for each row to verify that they indeed point to the correct row.
- May take a long time for a large table that has many indexes.

myisamchk -e -i tbl_name

- The same as previous plus prints additional statistical information

CHECK TABLE

- CHECK TABLE works for MyISAM, InnoDB, ARCHIVE and CSV tables
- Check views for problems, such as tables that are referenced in the view definition that no longer exist
- The FOR UPGRADE option checks whether the named tables are compatible with the current version of MySQL

How to Repair Tables

- You can use myisamchk or REPAIR TABLE (for all engines)
- Myisamchk command deals only with MyISAM tables
- Shut down your server before repair

Symptoms of Corrupted Tables

- queries that abort unexpectedly
- tbl_name.frm is locked against change
- Can't find file tbl_name.MYI (Errcode: nnn)
- Unexpected end of file
- Record file is crashed
- Got error nnn from table handler

Find out more about Errors

```
shell> perror 126 127 132 134 135 136 141 144 145
MySQL error code 126 = Index file is crashed
MySQL error code 127 = Record-file is crashed
MySQL error code 132 = Old database file
MySQL error code 134 = Record was already deleted (or record file crashed)
MySQL error code 135 = No more room in record file
MySQL error code 136 = No more room in index file
MySQL error code 141 = Duplicate unique key or constraint on write or update
MySQL error code 144 = Table is crashed and last repair failed
MySQL error code 145 = Table was marked as crashed and should be repaired
```

Repairing MyISAM tables

Stage 1

Checking your tables

- myisamchk *.MYI or myisamchk -e *.MYI
- If the mysqld server is stopped, use the --update-state to mark the table as “checked.”

Stage 2

Easy safe repair

1. First, try myisamchk -r -q tbl_name (quick recovery mode, which attempts to repair the index file without touching the data file)
2. If it didn’t helped use the following procedure:
    1. Make a backup of the data file before continuing.
    2. Use myisamchk -r tbl_name (-r means “recovery mode”)
        1. This removes incorrect rows and deleted rows from the data file
        2. Reconstructs the index file
    3. If the preceding step fails, use myisamchk --safe-recover tbl_name. Safe recovery mode uses an old recovery method that handles a few cases that regular recovery mode does not (but is slower)

Optimizing Table

```
shell> myisamchk –r tbl_name
```

- coalesce fragmented rows
- eliminate wasted space that results from deleting or updating rows
- other options

```
--analyze,
--sort-index,
--sort-records=index_num
OPTIMIZE TABLE SQL
```

- table repair
- key analysis
- also sorts the index tree so that key lookups are faster
- there is also no possibility of unwanted interaction between a utility and the server

Exercise

- Check the file size of a .myd file
- Remove a couple of rows in the myisam table
- Check the size of the table
- Run OPTIMIZE TABLE statement

- Check the sie of the table again

MODULE 8 - Running Multiple MariaDB Servers

- Running Multiple Servers in Windows
- Running Multiple Servers in Windows as Services
- Running Multiple Servers in Unix and Linux
- Using Client Tools in a Multi-Server Environment

Reasons for running multiple mysqld servers on the same machine

- To test a new release while leaving your existing production setup undisturbed
- Or you might want to give different users access to different mysqld servers that they manage themselves (service providers, projects, etc...)
- Two application use resources in different times (because of time zones, etc...) and you want to make use of these resources.
- Have more power which isn't available through virtualization
- Use different time zone and language settings

What needs to be different

- Connection to the server

```
--port=port_num
--socket=path
--bind-address=address
```

- PID filename

```
--pid-file=file_name
```

- Log files

```
--general_log or --log[=file_name]
--log-bin[=file_name]
--slow_query_log or --log-slow-queries[=file_name]
--log-error[=file_name]
```

- Temporary Directory

```
--tmpdir=path
```

- Data Directory

```
--datadir=path
```

- Configuration File (optional)

```
--defaults-file
```

- MySQL Binaries (optional)

Running Multiple Instances in Linux

- To compile MySQL binaries with different TCP/IP ports and Unix socket files so that each one is listening on different network interfaces
- Use The MySQL Instance Manager (mysqlmanager) - DEPRICATED!
- Use mysql_multi script

The mysql_multi script

```
[mysqld_multi]
mysqld     = /usr/bin/mysqld_safe
mysqladmin = /usr/bin/mysqladmin
user       = multi_admin
password   = multipass
```

```
[mysqld2]
socket     = /tmp/mysql.sock2
port       = 3307
pid-file   = /var/run/mysqld/mysqld2.pid
datadir    = /var/lib/mysql2
user       = mysql
```

Installing Service on Windows

```
cd c:\mariadb1\bin
mysql_install_db.exe --datadir=c:\mariadb1\data --service=MariaDB1
cd c:\mariadb2\bin
mysql_install_db.exe --datadir=c:\mariadb2\data --service=MariaDB2
```

edit c:/mariadb2/data/my.ini and change port and tmp file

Step By Step

- Copy the data directory to /var/lib/mysql2
- Check the permission to the deamon user (usually mysql)
- Paste the configuration from the previous slide to /etc/mysql/my.cnf
- Run mysqld_multi start 2
- Connect using mysql -u root --port 3307 -p
- Grant shutdown privilege to each instance
- GRANT SHUTDOWN ON *.*
- TO 'multi_admin'@'localhost' IDENTIFIED BY 'multipass';

Controlling Instances

```
shell> mysqld_multi [options] {start|stop|report} [GNR[,GNR] ...]
shell> mysqld_multi stop 8,10-13
```

Managing Configuration Files

- With --no-defaults, no option files are read.
- With --defaults-file=file_name, only the named file is read.
- Otherwise, option files in the standard list of locations are read, including any file named by the --defaults-extra-file=file_name option, if one is given. (If the option is given multiple times, the last value is used.)

MODULE 9 - MySQL Query Cache

- The Concept of Query Cache
- Testing Query Cache with SELECT
- Configuring Query Cache
- Checking Query Cache Status and Maintenance

The Query Cache

- Stores the text of a SELECT statement together with the corresponding result that was sent to the client.
- If an identical statement is received later, the server retrieves the results from the query cache rather than parsing and executing the statement again.
- The query cache is shared among sessions, so a result set generated by one client can be sent in response to the same query issued by another client.
- Useful in an environment where you have tables that do not change very often and for which the server receives many identical queries.
- This is a typical situation for many Web servers that generate many dynamic pages based on database content.
- When tables are modified, any relevant entries in the query cache are flushed.

The Query Cache

- The overhead for having the query cache active is 13% (he worst case scenario).
- Searches for a single row in a single-row table are 238% faster with the query cache than without it (minimum speedup)
- To disable the query cache at server startup, set the query_cache_size system variable to 0
- With some query cache configurations or server workloads, you might actually see a performance decrease:
- A query mix consisting almost entirely of a fixed set of SELECT statements is much more likely to benefit from enabling the cache than a mix in which frequentINSERT statements cause continual invalidation of results in the cache.

The Query Cache

- In some cases, a workaround is to use the SQL_NO_CACHE option to prevent results from even entering the cache for SELECT statements that use frequently modified tables, for example:

```
SELECT * FROM EMP;
INSERT INTO EMP….
```

- Would work faster if mysql would insert the cache and invalidate it

```
SELECT SQL_NO_CACHE * FROM EMP;
INSERT INTO EMP….
```

How the Query Cache Operates

- Queries are compared before parsing, the queries below are regarded as different by the query cache:
- SELECT * FROM tbl_name
- Select * from tbl_name
- The cache is not used for queries of the following types:
- Queries that are a subquery of an outer query
- Queries executed within the body of a stored function, trigger, or event
- If a query result is returned from query cache, the server increments the Qcache_hits status variable, not Com_select.
- If a table changes, all cached queries that use the table become invalid and are removed from the cache
- INSERT, UPDATE, DELETE, TRUNCATE, ALTER TABLE, DROP TABLE, or DROP DATABASE changes the data,
- even if the modification doesn’t effect the rows stored in the cache

When a query is NOT cached

- Uses BENCHMARK(), CONNECTION_ID(), CONVERT_TZ(), CURDATE() , etc…
- Uses User Defined Function or Stored Funcion
- Refers to user variables or local stored program variables
- Refers to tables in the mysql or INFORMATION_SCHEMA system database
- SELECT ... LOCK IN SHARE MODE SELECT ... FOR UPDATE SELECT ... INTO OUTFILE ... SELECT ... INTO DUMPFILE ... SELECT * FROM ... WHERE autoincrement_col IS NULL The last form is not cached because it is used as the ODBC workaround for obtaining the last insert ID value
- Statements within transactions that use SERIALIZABLE isolation level also cannot be cached because they use LOCK IN SHARE MODE locking
- Uses TEMPORARY tables
- Does not use any tables
- Generates warnings
- The user has a column-level privilege for any of the involved tables

Query Cache Configuration

- SET SESSION query_cache_type = OFF
- The maximum size of individual query results that can be cached can be set in query_cache_limit system variable (default 1MB).
- Maximum size that can be specified for the query cache at run time with the SET statement can be limited with --maximum-query_cache_size=32M option (in configuration file or parameter).
- Query Cache Configuration
- The query_cache_size system variable
- If 0 (zero) the query cache is disabled
- Individual clients can control cache behavior for the SESSION query_cache_type variable
- SET SESSION query_cache_type = OFF;

Query Cache Modes

- If the query cache size is greater than 0, the
- query_cache_type=[{ON,OFF,DEMAND}]
- variable influences how it works
- OFF prevents caching or retrieval of cached results.
- ON allows caching except of those statements that begin with SELECT SQL_NO_CACHE.
- DEMAND causes caching of only those statements that begin with SELECT SQL_CACHE.
-

Query Cache Min Reserved Unit

- The query cache allocates blocks for storing data on demand, when one block is filled, a new block is allocated.
- memory allocation operation is costly (timewise),
- the query cache allocates blocks with a minimum size given by the query_cache_min_res_unit (default 4KB)
- The default value of query_cache_min_res_unit is 4KB.
- If you have a lot of queries with small results, the big block size may lead to memory fragmentation, which triggers pruning (delete) queries from the cache due to lack of memory
- The number of free blocks and queries removed due to pruning are given by the values of the Qcache_free_blocks and Qcache_lowmem_prunes status variables.
- If most of your queries have large results (Qcache_total_blocks and Qcache_queries_in_cache), you can increase performance by increasing query_cache_min_res_unit
-

Query Cache Maintenance

FLUSH QUERY CACHE

- defragments the query cache to better utilize its memory
- does NOT remove any queries from the cache

RESET QUERY CACHE

- removes all query results from the query cache
- FLUSH TABLES statement also does this

SHOW STATUS LIKE 'Qcache%';

Exercises

- Execute the query below with and without the Query Cache
- SELECT * FROM cities;
- Check the memory utilized by the query cache