B.3.3.3 What to Do If MySQL Keeps Crashing

Each MySQL version is tested on many platforms before it is released. This does not mean that there are no bugs in MySQL, but if there are bugs, they should be very few and can be hard to find. If you have a problem, it always helps if you try to find out exactly what crashes your system, because you have a much better chance of getting the problem fixed quickly.

First, you should try to find out whether the problem is that the **mysqld** server dies or whether your problem has to do with your client. You can check how long your **mysqld** server has been up by executing **mysqladmin version**. If **mysqld** has died and restarted, you may find the reason by looking in the server's error log. See Section 5.4.2, “The Error Log”.

On some systems, you can find in the error log a stack trace of where **mysqld** died that you can resolve with the `resolve_stack_dump` program. See Section 5.8, “Debugging MySQL”. Note that the variable values written in the error log may not always be 100% correct.

Many unexpected server exits are caused by corrupted data files or index files. MySQL updates the files on disk with the `write()` system call after every SQL statement and before the client is notified about the result. (This is not true if you are running with the `delay_key_write` system variable enabled, in which case data files are written but not index files.) This means that data file contents are safe even if **mysqld** crashes, because the operating system ensures that the unflushed data is written to disk. You can force MySQL to flush everything to disk after every SQL statement by starting **mysqld** with the `--flush` option.

The preceding means that normally you should not get corrupted tables unless one of the following happens:

- The MySQL server or the server host was killed in the middle of an update.
- You have found a bug in **mysqld** that caused it to die in the middle of an update.
- Some external program is manipulating data files or index files at the same time as **mysqld** without locking the table properly.
- You are running many **mysqld** servers using the same data directory on a system that does not support good file system locks (normally handled by the `lockd` lock manager), or you are running multiple servers with external locking disabled.
- You have a crashed data file or index file that contains very corrupt data that confused **mysqld**.

- You have found a bug in the data storage code. This isn't likely, but it is at least possible. In this case, you can try to change the storage engine to another engine by using `ALTER TABLE` on a repaired copy of the table.

Because it is very difficult to know why something is crashing, first try to check whether things that work for others result in an unexpected exit for you. Try the following things:

- Stop the **mysqld** server with **mysqladmin shutdown**, run **myisamchk --silent --force */*.MYI** from the data directory to check all `MyISAM` tables, and restart **mysqld**. This ensures that you are running from a clean state. See Chapter 5, *MySQL Server Administration*.
- Start **mysqld** with the general query log enabled (see Section 5.4.3, “The General Query Log”). Then try to determine from the information written to the log whether some specific query kills the server. About 95% of all bugs are related to a particular query. Normally, this is one of the last queries in the log file just before the server restarts. See Section 5.4.3, “The General Query Log”. If you can repeatedly kill MySQL with a specific query, even when you have checked all tables just before issuing it, then you have isolated the bug and should submit a bug report for it. See Section 1.5, “How to Report Bugs or Problems”.
- Try to make a test case that we can use to repeat the problem. See Section 5.8, “Debugging MySQL”.
- Try the `fork_big.pl` script. (It is located in the `tests` directory of source distributions.)
- Configuring MySQL for debugging makes it much easier to gather information about possible errors if something goes wrong. Reconfigure MySQL with the `-DWITH_DEBUG=1` option to **CMake** and then recompile. See Section 5.8, “Debugging MySQL”.
- Make sure that you have applied the latest patches for your operating system.
- Use the `--skip-external-locking` option to **mysqld**. On some systems, the `lockd` lock manager does not work properly; the `--skip-external-locking` option tells **mysqld** not to use external locking. (This means that you cannot run two **mysqld** servers on the same data directory and that you must be careful if you use **myisamchk**. Nevertheless, it may be instructive to try the option as a test.)
- If **mysqld** appears to be running but not responding, try **mysqladmin -u root processlist**. Sometimes **mysqld** is not hung even though it seems unresponsive. The problem may be that all connections are in use, or there may be some internal lock problem. **mysqladmin -u root processlist** usually is able to make a connection even in these cases, and can provide useful information about the current number of connections and their status.

- Run the command **mysqladmin -i 5 status** or **mysqladmin -i 5 -r status** in a separate window to produce statistics while running other queries.
- Try the following:

  a. Start **mysqld** from **gdb** (or another debugger). See Section 5.8, “Debugging MySQL”.

  b. Run your test scripts.

  c. Print the backtrace and the local variables at the three lowest levels. In **gdb**, you can do this with the following commands when **mysqld** has crashed inside **gdb**:

  ```
  backtrace
  info local
  up
  info local
  up
  info local
  ```

  With **gdb**, you can also examine which threads exist with `info threads` and switch to a specific thread with `thread N`, where *N* is the thread ID.
- Try to simulate your application with a Perl script to force MySQL to exit or misbehave.
- Send a normal bug report. See Section 1.5, “How to Report Bugs or Problems”. Be even more detailed than usual. Because MySQL works for many people, the crash might result from something that exists only on your computer (for example, an error that is related to your particular system libraries).
- If you have a problem with tables containing dynamic-length rows and you are using only `VARCHAR` columns (not `BLOB` or `TEXT` columns), you can try to change all `VARCHAR` to `CHAR` with `ALTER TABLE`. This forces MySQL to use fixed-size rows. Fixed-size rows take a little extra space, but are much more tolerant to corruption.

  The current dynamic row code has been in use for several years with very few problems, but dynamic-length rows are by nature more prone to errors, so it may be a good idea to try this strategy to see whether it helps.
- Consider the possibility of hardware faults when diagnosing problems. Defective hardware can be the cause of data corruption. Pay particular attention to your memory and disk subsystems when troubleshooting hardware.