MySQL Server Hardware and OS Tuning:

1. Have enough physical memory to load your entire InnoDB file into memory – InnoDB is much faster when the file can be accessed in memory rather than from disk.

2. Avoid Swap at all costs – swapping is reading from disk, its slow.

3. Use Battery-Backed RAM.

4. Use an advanced RAID – preferably RAID10 or higher.

5. Avoid RAID5 – the checksum needed to ensure integrity is costly.

6. Separate your OS and data partitions, not just logically, but physically – costly OS writes and reads will impact your database performance.

7. Put your mysql temp space and replication logs on a separate partition than your data – background writes will impact your database when it goes to write/read from disk.

8. More disks equals more speed.

9. Faster disks are better.

10. Use SAS over SATA.

11. Smaller disks are faster than larger disks, especially in RAID configs.

12. Use Battery-Backed Cache RAID controllers.

13. Avoid software raids.

14. Consider using Solid State IO Cards (not disk drives) for your data partition – these cards can sustain over 2GB/s writes for almost any amount of data.

15. On Linux set your swappiness value to 0 – no reason to cache files on a database server, this is more of a web server or desktop advantage.

16. Mount filesystem with noatime and nodirtime if available – no reason to update database file modification times for access.

17. Use XFS filesystem – a faster, smaller filesystem than ext3 and has more options for journaling, also ext3 has been shown to have double buffering issues with MySQL.

18. Tune your XFS filesystem log and buffer variables – for maximum performance benchmark.

19. On Linux systems, use NOOP or DEADLINE IO scheduler – the CFQ and ANTICIPATORY scheduler have been shown to be slow vs NOOP and DEADLINE scheduler.

20. Use a 64-bit OS – more memory addressable and usable to MySQL.

21. Remove unused packages and daemons from servers – less resource stealing.

22. Put your host that use MySQL and your MySQL host in a hosts file – no dns lookups.

23. Never force kill a MySQL process – you will corrupt your database and be running for the backups.

24. Dedicate your server to MySQL – background processes and other services can steal from the db cpu time.

MySQL Montoring

MySQL Configuration:

25. Use innodb\_flush\_method=O\_DIRECT to avoid a double buffer when writing.

26. Avoid O\_DIRECT and EXT3 filesystem – you will serialize all your writes.

27. Allocate enough innodb\_buffer\_pool\_size to load your entire InnoDB file into memory – less reads from disk.

28. Do not make innodb\_log\_file\_size too big, with faster and more disks – flushing more often is good and lowers the recovery time during crashes.

29. Do not mix innodb\_thread\_concurrency and thread\_concurrency variables – these two values are not compatible.

30. Allocate a minimal amount for max\_connections – too many connections can use up your RAM and lock up your MySQL server.

31. Keep thread\_cache at a relatively high number, about 16 – to prevent slowness when opening connections.

32. Use skip-name-resolve – to remove dns lookups.

33. Use query cache if your queries are repetitive and your data does not change often – however using query cache on data that changes often will give you a performance hit.

34. Increase temp\_table\_size – to prevent disk writes.

35. Increase max\_heap\_table\_size – to prevent disk writes.

36. Do not set your sort\_buffer\_size too high – this is per connection and can use up memory fast.

37. Monitor key\_read\_requests and key\_reads to determine your key\_buffer size – the key read requests should be higher than your key\_reads, otherwise you are not efficiently using your key\_buffer.

38. Set innodb\_flush\_log\_at\_trx\_commit = 0 will improve performance, but leaving it to default (1), you will ensure data integrity, you will also ensure replication is not lagging

39. Have a test environment where you can test your configs and restart often, without affecting production.

MySQL Schema Optimization:

40. Keep your database trim.

41. Archive old data – to remove excessive row returns or searches on queries.

42. Put indexes on your data.

43. Do not overuse indexes, compare with your queries.

44. Compress text and blob data types – to save space and reduce number of disk reads.

45. UTF 8 and UTF16 is slower than latin1.

46. Use Triggers sparingly.

47. Keep redundant data to a minimum – do not duplicate data unnecessarily.

48. Use linking tables rather than extending rows.

49. Pay attention to your data types, use the smallest one possible for your real data.

50. Separate blob/text data from other data if other data is often used for queries when blob/text are not.

51. Check and optimize tables often.

52. Rewrite InnoDB tables often to optimize.

53. Sometimes, it is faster to drop indexes when adding columns and then add indexes back.

54. Use different storage engines for different needs.

55. Use ARCHIVE storage engine for Logging tables or Auditing tables – this is much more efficient for writes.

56. Store session data in memcache rather than MySQL – memcache allows for auto-expiring values and prevents you from having to create costly reads and writes to MySQL for temporal data.

57. Use VARCHAR instead CHAR when storing variable length strings – to save space since CHAR is fixed length and VARCHAR is not (utf8 is not affected by this).

58. Make schema changes incrementally – a small change can have drastic effects.

59. Test all schema changes in a development environment that mirrors production.

60. Do NOT arbitrarily change values in your config file, it can have disastrous affects.

61. Sometimes less is more in MySQL configs.

62. When in doubt use a generic MySQL config file.

MySQL metrics widget

Query Optimization:

63. Use the slow query log to find slow queries.

64. Use EXPLAIN to determine queries are functioning appropriately.

65. Test your queries often to see if they are performing optimally – performance will change over time.

66. Avoid count(\*) on entire tables, it can lock the entire table.

67. Make queries uniform so subsequent similar queries will use query cache.

68. Use GROUP BY instead of DISTINCT when appropriate.

69. Use indexed columns in WHERE, GROUP BY, and ORDER BY clauses.

70. Keep indexes simple, do not reuse a column in multiple indexes.

71. Sometimes MySQL chooses the wrong index, use USE INDEX for this case

72. Check for issues using SQL\_MODE=STRICT.

73. Use a LIMIT on UNION instead of OR for less than 5 indexed fields.

74. Use INSERT ON DUPLICATE KEY or INSERT IGNORE instead of UPDATE to avoid the SELECT prior to update.

75. Use a indexed field and ORDER BY instead of MAX.

76. Avoid using ORDER BY RAND().

77. LIMIT M,N can actually slow down queries in certain circumstances, use sparingly.

78. Use UNION instead of sub-queries in WHERE clauses.

79. For UPDATES, use SHARE MODE to prevent exclusive locks.

80. On restarts of MySQL, remember to warm your database, to ensure that your data is in memory and queries are fast.

81. Use DROP TABLE then CREATE TABLE instead of DELETE FROM to remove all data from a table.

82. Minimize the data in your query to only the data you need, using \* is overkill most of the time.

83. Consider persistent connections instead of multiple connections to reduce overhead.

84. Benchmark queries, including using load on the server, sometimes a simple query can have affects on other queries.

85. When load increases on your server, use SHOW PROCESSLIST to view slow/problematic queries.

86. Test all suspect queries in a development environment where you have mirrored production data.

MySQL Backup Procedures:

87. Backup from secondary replicated server.

88. Stop replication during backups to prevent inconsistencies on data dependencies and foreign constraints.

89. Stop MySQL altogether and take a backup of the database files.

90. Backup binary logs at same time as dumpfile if MySQL dump used – to make sure replication does not break.

91. Do not trust an LVM snapshot for backups – this could create data inconsistencies that will give you issues in the future.

92. Make dumps per table for easier single table recovery – if data is isolated from other tables.

93. Use –opt when using mysqldump.

94. Check and Optimize tables before a backup.

95. When importing temporarily disable foreign constraints for a faster import.

96. When importing temporarily disable unique checks for a faster import.

97. Calculate size of database/tables data and indexes after each backup to monitor growth.

98. Monitor slave replication for errors and delay with a cron script.

99. Perform Backups regularly.

100. Test your backups regularly.

And finally 101: Perform MySQL monitoring: Monitis Unveils The World’s First Free On-demand MySQL Monitoring.