**Lab Manual- Performnace Tuning MySQL**

**Prepared for**: TechPledge

**Date:** 18th Nov 2018

**Prepared by:** Bipin Sinhaa

Document Name: Lab Manual

**Document Number** MySQL309

**Contributor:**

Shruti Sinhaa

Table of Contents

[1 OBJECTIVE 3](#_Toc22828400)

[2 PRE-REQUISISTE 3](#_Toc22828401)

[3 Global Command 3](#_Toc22828402)

[4 MySQL Monitioring 5](#_Toc22828403)

[4.1 **Chaning configuration online** 5](#_Toc22828404)

[4.2 **Most Common MySQL Configuration Mistakes** 5](#_Toc22828405)

[4.2.1 Changing configuration online 7](#_Toc22828406)

[4.2.2 Global -vs- Local scope 8](#_Toc22828407)

[4.2.3 **default\_storage\_engine – choose the right engine first** 8](#_Toc22828408)

[4.2.4 **innodb\_buffer\_pool\_size – get the best out of your memory** 10](#_Toc22828409)

[4.3 **Writing the MySQL transaction log** 12](#_Toc22828410)

[4.4 **Setting the MySQL query cache size** 12](#_Toc22828411)

[4.5 **Regenerating MySQL indexes** 12](#_Toc22828412)

[4.6 **MySQL Server Hardware and OS Tuning:** 13](#_Toc22828413)

[4.7 **MySQL Schema Optimization:** 14](#_Toc22828414)

# OBJECTIVE

Tens of hundreds of thousands of hours are wasted every day world-wide (rough estimates) doing something that’s dead simple if only one understands few basic principles of MySQL operation.

Yes, you can start MySQL with no con]iguration and you can start using it for development right away. However, you can’t just put MySQL with default con]iguration to production and expect that it will handle the increasing workload with ease - you have to prepare your server for that.

In this Lab we are going to work on MySql cluster

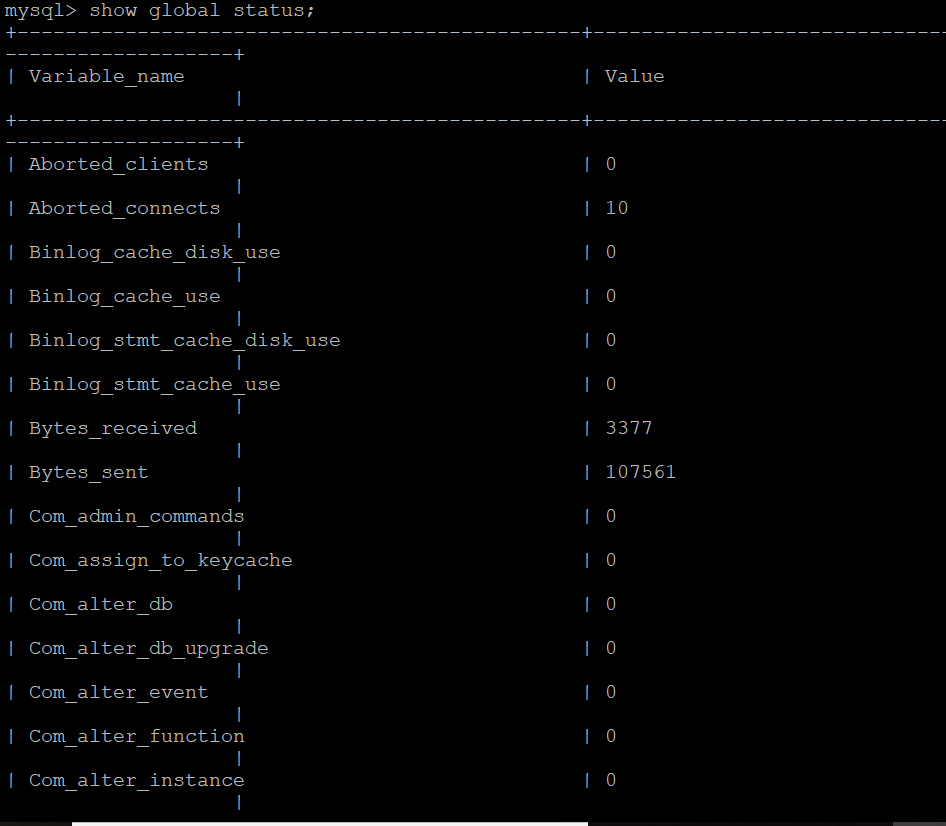
* Change only one thing at time
* Keep track record of everything
* Create a baseline
* Put all your server configuration under mysqld under my.cnf file

# PRE-REQUISISTE

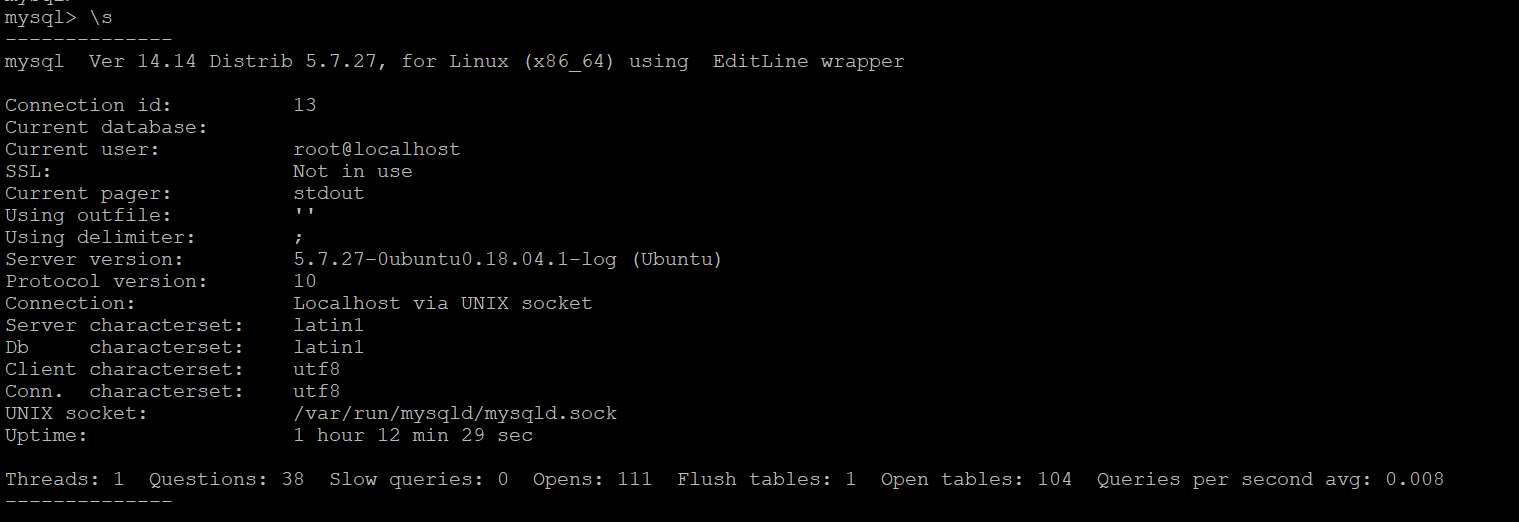
* Prior knowledge of Linux
* Accounts in AWS
* A local Computer with 4 CPU, 16 GB RAM, 200 GB disk space

# Global Command

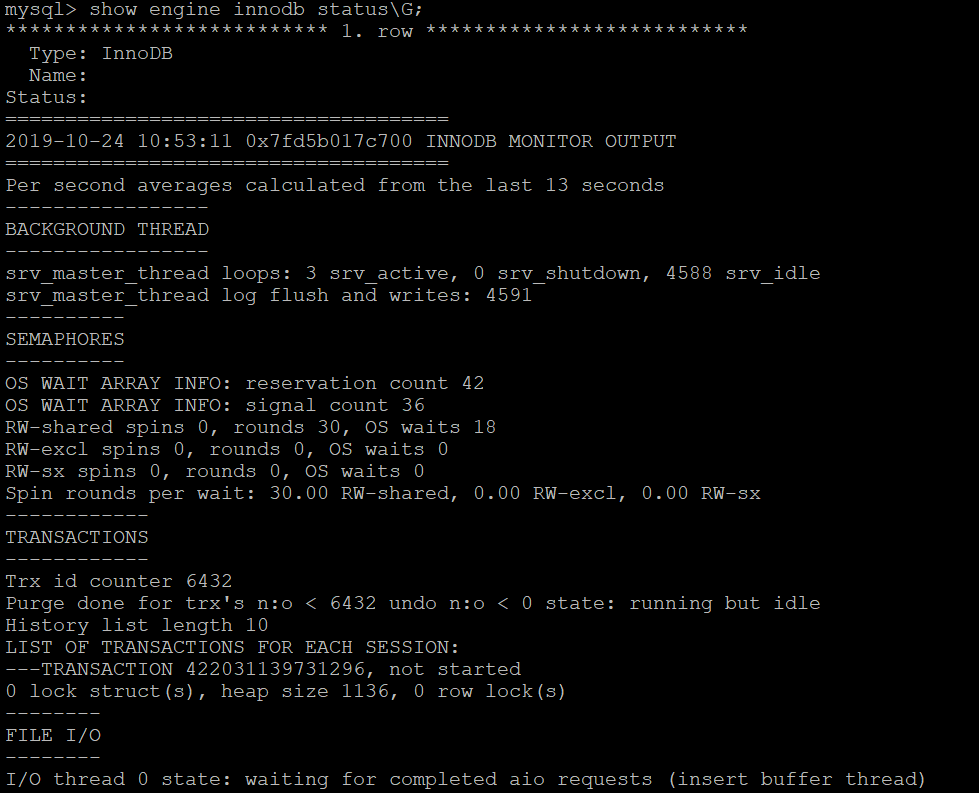
Mysql> show global status;



Mysql>\s



Mysql> show engine innodb status\G;



# MySQL Monitioring

## **Chaning configuration online**

MySQL is very easy to con]igure. It’s just one con]iguration ]ile you have to deal with, my.cnf, and it has one option per line, so the format is very convenient.

The location of MySQL con]iguration ]ile may be different across different operating systems and distributions, however on Linux it’s typically either /etc/my.cnf (Redhat style) or /etc/mysql/my.cnf (Debian style). On Windows, you can put it in a number of locations, however I suggest you use the data directory and create my.cnf ]ile there.

## **Most Common MySQL Configuration Mistakes**

* **Using Trial and Error approach**
* **Using Google for performance advice**

Never trust the ]irst response you ]ind on Google when searching for a performance advice, or value for your speci]ic variable. A lot of the advice on the internet is very generic and often lacks context.

* **Obsessing about fine-tuning the my.cnf**

Don’t get obsessed about ]ine-tuning the con]iguration – usually 10-15 variables will give you the most impact, and ]ine-tuning the variables is highly unlikely to have any additional bene]its. It can do harm though. If you still have a performance problem even after you have applied all the recommendations (and gotten rid of everything that you shouldn’t have touched in the ]irst place), the problem is probably somewhere else – bad queries, lack of resources, etc.

* **Changing many things at once**

When working with con]iguration, change only one thing at a time. Especially if you already have a solid con]iguration. Otherwise when things go bad, it maybe very hard or even impossible to ]igure out which setting could have caused the issue, so you will have to roll back all of the changes and then start one by one anyway.

* **Not keeping my.cnf in sync with the changes you make**

It’s no secret that many things can now be changed online without even touching my.cnf. Even the innodb buffer pool size can be changed online in MySQL 5.7. That’s very convenient, but. Make sure you update my.cnf after you are done with the changes or you will lose all these changes when MySQL is restarted and you’ll have to start over.

* **Redundant entries in my.cnf**

If you use the same variable twice, MySQL will not complain about it. In most cases, it will just use the last value found for the same variable, so be sure you don’t add the same variable twice, otherwise you may end up not seeing the impact. Also note that a dash “-” and an underscore “\_” can be used interchangeably, so innodb-log-file-size and innodb\_log\_file\_size are both referring to the same server setting

* **Multiplying buffer sizes**

When you add more memory to the server, don’t just multiply the size of all buffers in effect. Some buffers are local, some global. Some are storage engine related, some are server wide. In fact, there are very few variables that you need to increase in size as you add more memory. Yes, these are crucial to update, or you won’t have the desired effect, but only these and no other. I will talk about these variables as we progress

* **Using the wrong my.cnf section**

While MySQL con]iguration ]ile is simple, it’s important to mention that it does have sections and these are important. For example, there’s such sections as **[mysql], [client], [mysqld\_safe].** And there’s also a [mysqld] section, which is exactly the section you must use if you want server con]iguration to take effect. So all of the variables for server configuration should be placed after **[mysqld].**

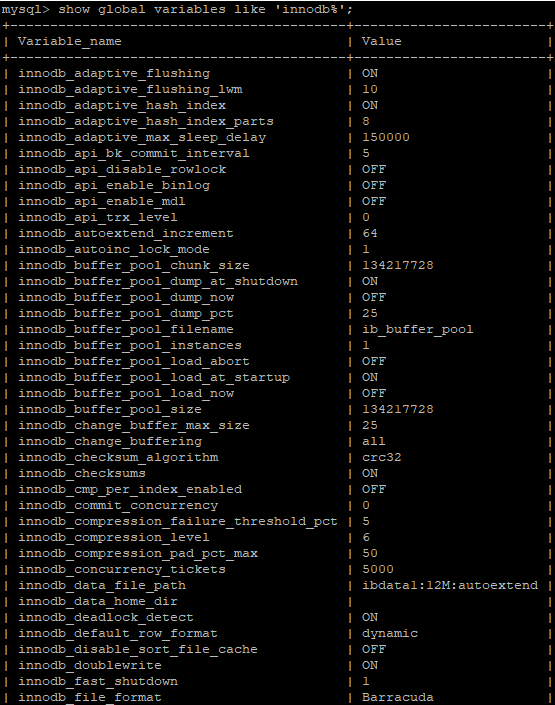
### Changing configuration online

there is a way to change some parameters online. And in fact it’s safe to try, even if you’re not sure if you can change it online - MySQL will just tell you that the variable is read-only, meaning that you should be changing the my.cnf ]ile instead and restart the server

$ set global innodb\_buffer\_pool\_size=128\*1024\*1024;

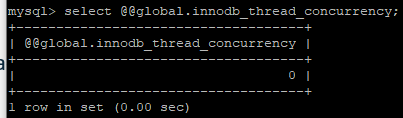


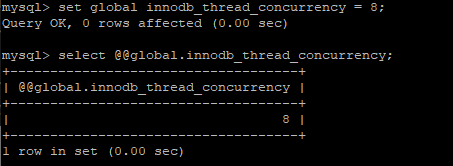
$show global variables like 'innodb%';



innodb thread concurrency. Let’s change that, but ]irst - let’s check the current value for it

mysql>select @@global.innodb\_thread\_concurrency;

  
mysql> set global innodb\_thread\_concurrency = 8;



### Global -vs- Local scope

We have just seen a way to alter global MySQL con]iguration, but here’s an interesting thing - quite often you don’t need to update the global con]iguration just to get that single query work properly. In fact, very often it’s better that you leave the global configuration untouched, because an optimization for one query may affect all of the other queries in the negative way

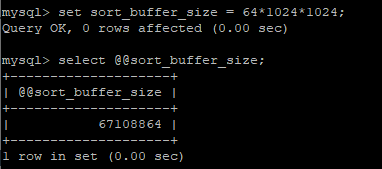
For example, one of the things I recommend is to keep the **sort\_buffer\_size** at its default value, because otherwise a full buffer is allocated for any session where sorting is done and that may end up wasting a lot of memory and time allocating it.

So :

For a query that needs to sort a lot of data , before you run it:

Mysql>set sort\_buffer\_size = 64\*1024\*1024;

Mysql> select @@sort\_buffer\_size;



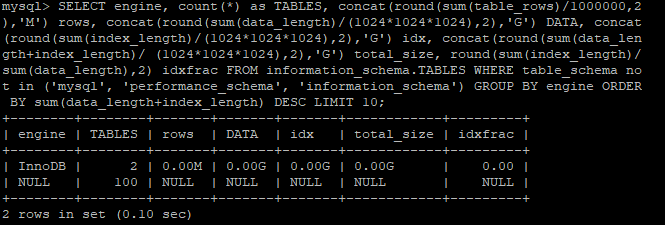
### **default\_storage\_engine – choose the right engine first**

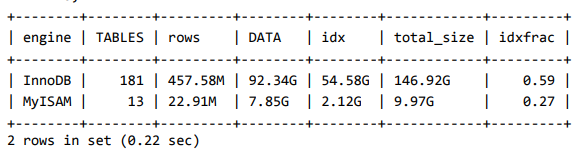
MySQL has supported pluggable storage engines since its inception over 20 years ago, but for a very long time MyISAM was the default storage engine and many people running MySQL didn’t even know anything about the underlying storage engines. After all, MySQL was initially designed to be a practical database for small websites and many applications got into habit of using MyISAM storage engine explicitly.

This seemed like a good idea ]irst, but here’s the problem: MyISAM was NOT designed with highly concurrent workload, number of CPU cores and RAID arrays in mind. And it was never meant to be resilient either. So as websites kept attracting more traf]ic, they could no longer scale, because MySQL queries would spend seconds waiting on table level locks (the only locking mechanism that MyISAM supports).

query that shows the storage engines you are using and a number of tables using each storage engine:

SELECT engine, count(\*) as TABLES, concat(round(sum(table\_rows)/1000000,2),'M') rows, concat(round(sum(data\_length)/(1024\*1024\*1024),2),'G') DATA, concat(round(sum(index\_length)/(1024\*1024\*1024),2),'G') idx, concat(round(sum(data\_length+index\_length)/ (1024\*1024\*1024),2),'G') total\_size, round(sum(index\_length)/sum(data\_length),2) idxfrac FROM information\_schema.TABLES WHERE table\_schema not in ('mysql', 'performance\_schema', 'information\_schema') GROUP BY engine ORDER BY sum(data\_length+index\_length) DESC LIMIT 10;





You can see that this particular customer had 13 MyISAM tables holding over 7GB worth of data combined.

### **innodb\_buffer\_pool\_size – get the best out of your memory**

What is InnoDB Buffer Pool? Computers use most of their memory to improve access to most commonly used data. This is known as caching and it is a very important part of computing, because accessing data on a disk can be 100 to 100,000 times slower, depending on the amount of data being accessed. Just think of it, a report that takes 1 second to generate when all data is in memory could take over a day to generate if all data had to be read from disk every single time (assuming also random I/O).

On a dedicated MySQL server running fully on InnoDB, as a rule of thumb, recommendation is to set the innodb-buffer-pool-size to 80% of the total available memory on the server.

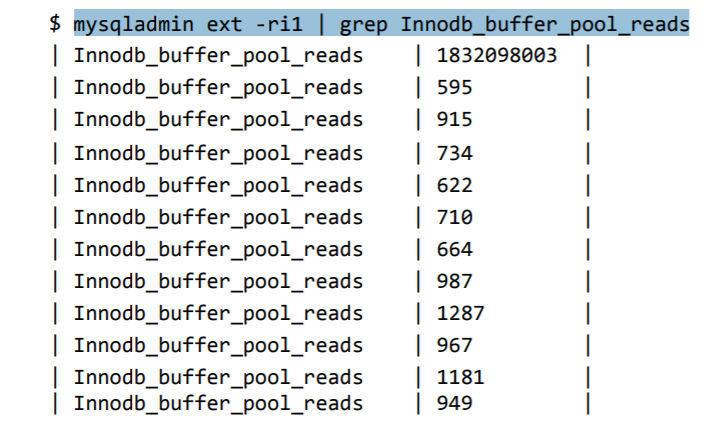
Why not 90% or 100%? Because other things need memory too:

* + Every query needs at least few kilobytes of memory (and sometimes – few megabytes!) • There’s various other internal MySQL structures and caches
  + InnoDB has a number of structures using memory beyond the buffer pool (Dictionary cache, File system, Lock system and Page hash tables, etc.)
  + There’s also some MySQL ]iles that must be in OS cache (binary logs, relay logs, innodb transaction logs).
  + Plus, you want to leave some room for the operating system memory structures.

To find out how much memory required by innodb buffer pool jisy run below command

# mysqladmin ext -ri1 | grep Innodb\_buffer\_pool\_reads

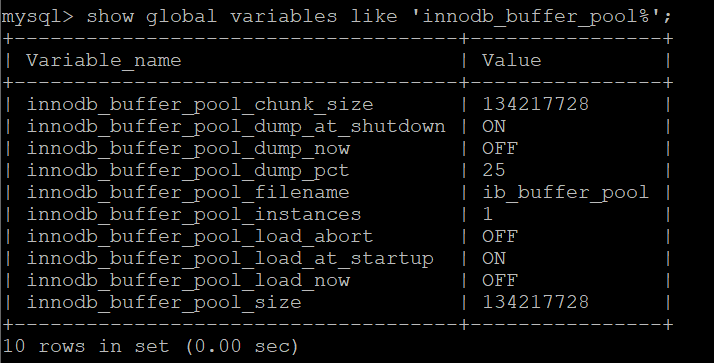
* + What you see here is the number of reads from disk into the buffer pool (per second). These numbers above are pretty darn high (luckily, this server has an IO device that can handle around 4000 random IO operations per second) and if this was an OLTP system, I would highly recommend to increase the innodb buffer pool size and add more memory to the server if needed.



To change the buffer pool size online

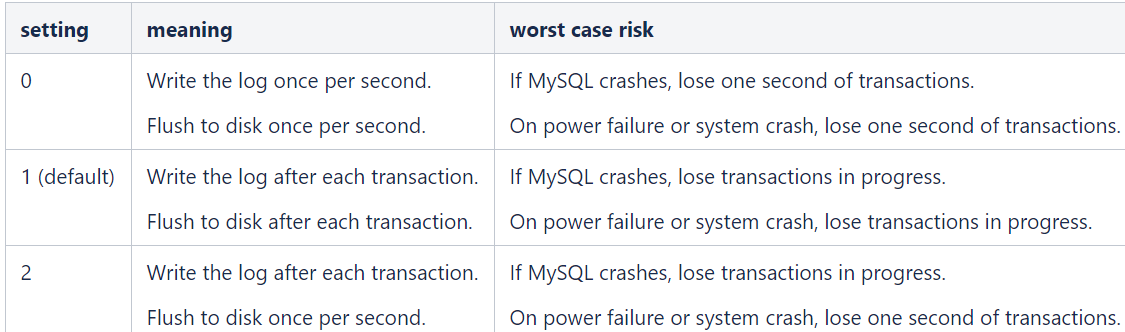
Mysql>set global innodb\_buffer\_pool\_size=128\*1024\*1024;

Mysql>show global variables like 'innodb\_buffer\_pool%';



## **Writing the MySQL transaction log**

Setting innodb\_flush\_log\_at\_trx\_commit to 0 or 2 will greatly improve throughput, while adding a minimal level of risk to the data. Under some circumstances, with some settings, up to one second of transactions can be lost



## **Setting the MySQL query cache size**

Increasing the MySQL query cache size will likely translate into improved VIVO performance in that once large pages have been fetched once, they're typically quite a bit faster to load on later fetches.

query\_cache\_size=0

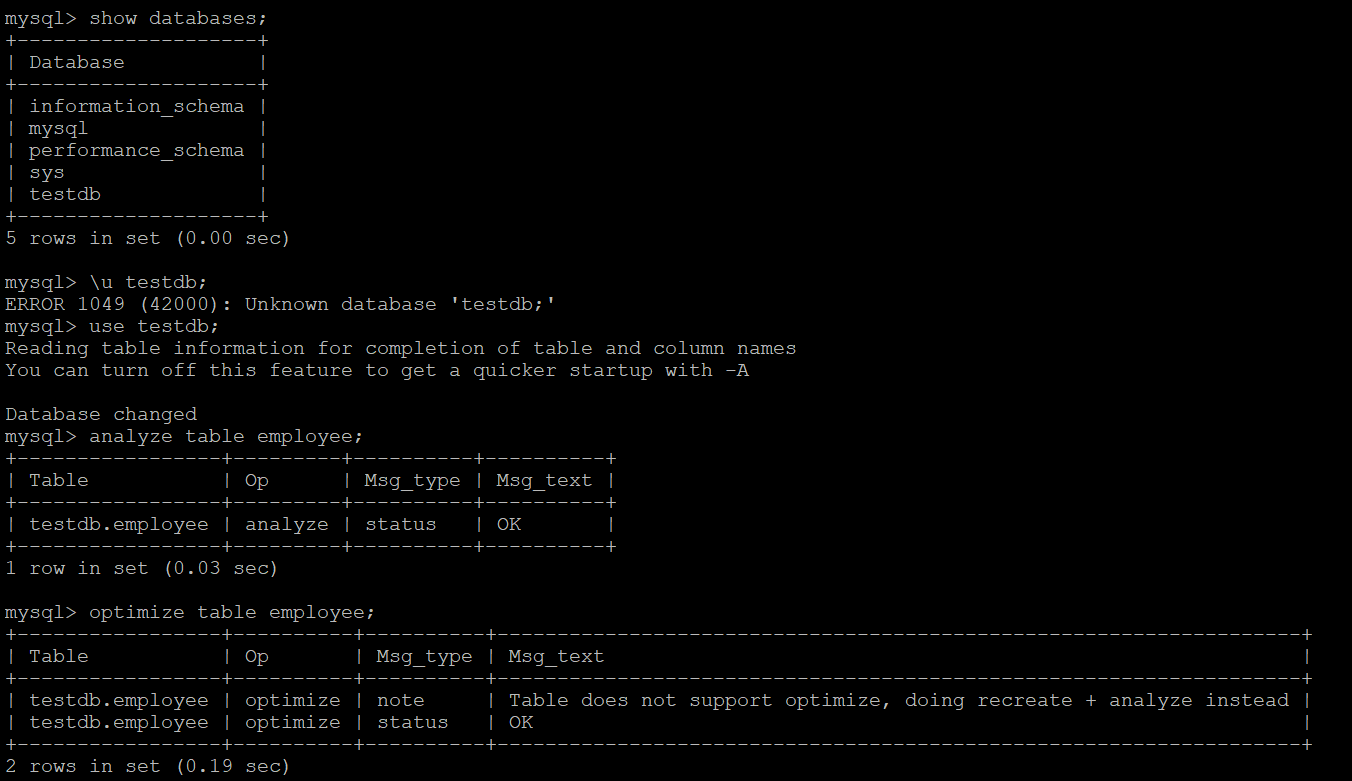
## **Regenerating MySQL indexes**

If performance is abysmal on a simple query, check for missing or corrupted MySQL indexes that may cause the query engine to do full table scans.

Mysql> use testdb;

Mysql> analyze table employee;

Mysql> optimize table employee;



## **MySQL Server Hardware and OS Tuning:**

* 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.
* Avoid Swap at all costs – swapping is reading from disk, its slow.
* Use an advanced RAID – preferably RAID10 or higher.
* Separate your OS and data partitions, not just logically, but physically – costly OS writes and reads will impact your database performance.
* 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.
* More disks equals more speed.
* Faster disks are better.
* Use SAS over SATA.
* Consider using Solid State IO Cards (not disk drives) for your data partition
* 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.
* 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.
* Use a 64-bit OS – more memory addressable and usable to MySQL.
* Remove unused packages and daemons from servers – less resource stealing.
* Never force kill a MySQL process – you will corrupt your database and be running for the backups.
* Dedicate your server to MySQL – background processes and other services can steal from the db cpu time.

## **MySQL Schema Optimization:**

* Keep your database trim.
* Archive old data – to remove excessive row returns or searches on queries.
* Keep redundant data to a minimum – do not duplicate data unnecessarily.
* Use linking tables rather than extending rows.
* Pay attention to your data types, use the smallest one possible for your real data.
* Check and optimize tables often.
* Rewrite InnoDB tables often to optimize.