

MYSQL-性能优化篇

1、为什么要进行数据库优化？

1、避免网站页面出现访问错误

由于数据库连接 timeout 产生页面 5xx 错误

由于慢查询造成页面无法加载

由于阻塞造成数据无法提交

2、增加数据库的稳定性

很多数据库问题都是由于低效的查询引起的

3、优化用户体验

流畅页面的访问速度

良好的网站功能体验

2、mysql 数据库优化

可以从哪几个方面进行数据库的优化？如下图所示：



A、SQL 及索引优化

根据需求写出良好的 SQL，并创建有效的索引，实现某一种需求可以多种写法，这时候我们就要选择一种效率最高的写法。这个时候就要了解 sql 优化

B、数据库表结构优化

根据数据库的范式，设计表结构，表结构设计的好直接关系到写 SQL 语句。

C、系统配置优化

大多数运行在 Linux 机器上，如 tcp 连接数的限制、打开文件数的限制、安全性的限制，因此我们要对这些配置进行相应的优化。

D、硬件配置优化

选择适合数据库服务的 cpu，更快的 IO，更高的内存；cpu 并不是越多越好，某些数据库版本有最大的限制，IO 操作并不是减少阻塞。

注：通过上图可以看出，该金字塔中，优化的成本从下而上逐渐增高，而优化的效果会逐渐降低。

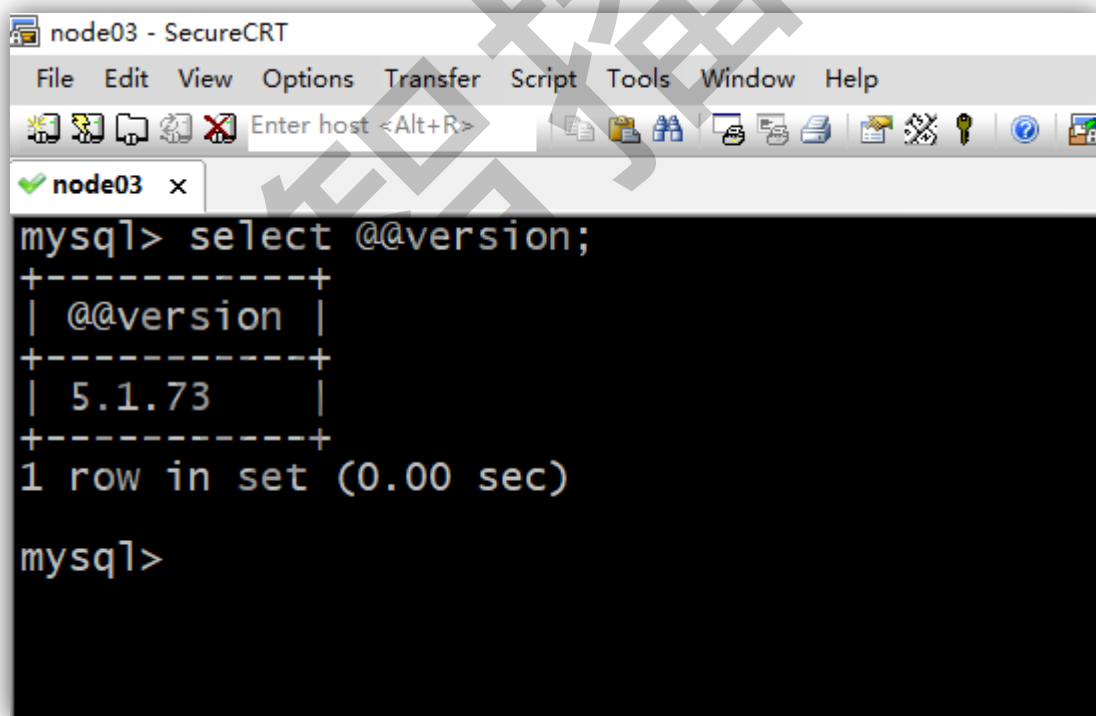
3、SQL 及索引优化

1、mysql 安装与卸载 (linux 在线安装与卸载)

2、数据库版本选择

1、查看数据库的版本

```
select @@version;
```



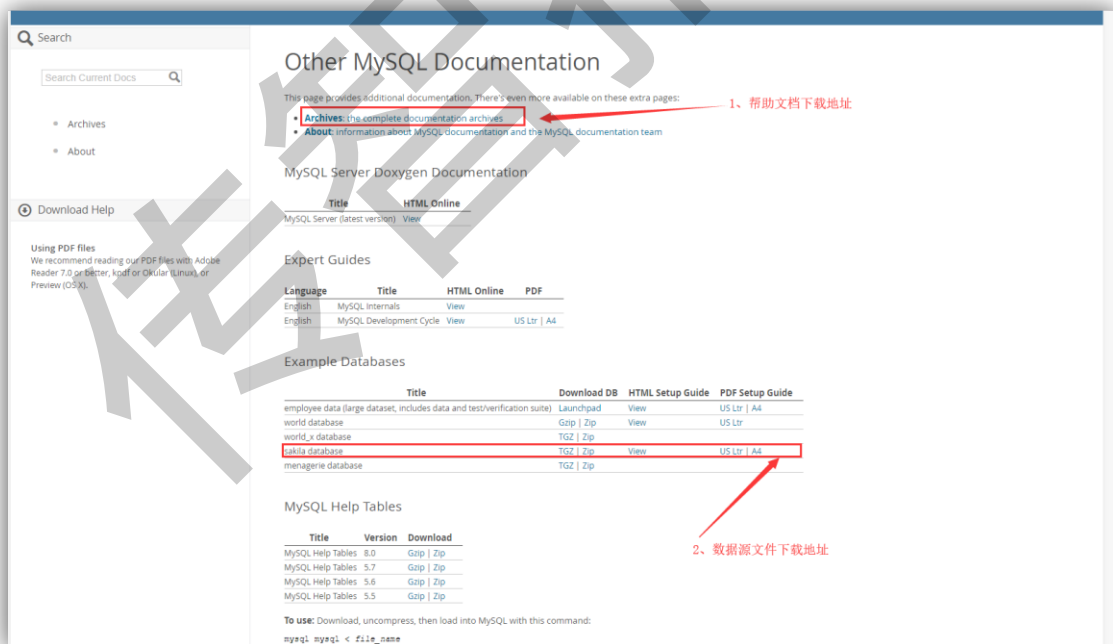
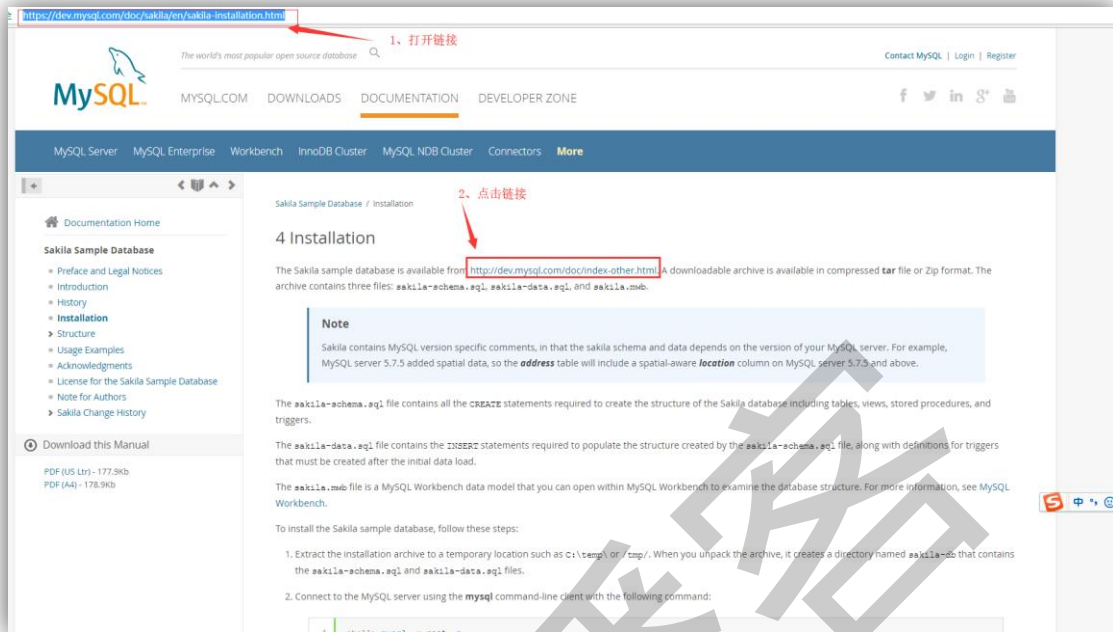
The screenshot shows a terminal window titled 'node03 - SecureCRT'. The terminal displays the command 'mysql> select @@version;' and its output. The output is formatted as a table with a single row showing the version '5.1.73'. Below the table, it indicates '1 row in set (0.00 sec)'. The prompt 'mysql>' is shown again at the bottom.

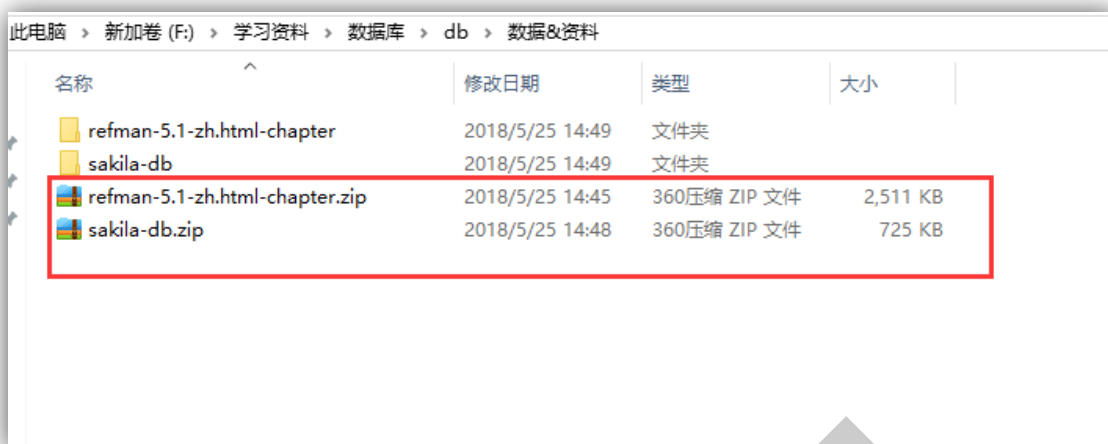
```
mysql> select @@version;
+-----+
| @@version |
+-----+
| 5.1.73    |
+-----+
1 row in set (0.00 sec)

mysql>
```

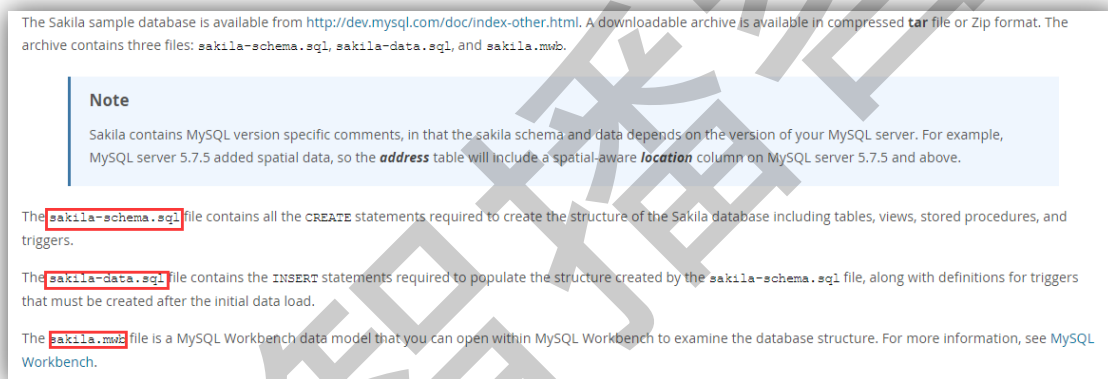
2、准备数据

网址：<https://dev.mysql.com/doc/sakila/en/sakila-installation.html>





sakila-db.zip 压缩包所包含的文件如下解释



加载数据

步骤如下图所示

To install the Sakila sample database, follow these steps:

1. Extract the installation archive to a temporary location such as `C:\temp\` or `/tmp/`. When you unpack the archive, it creates a directory named `sakila-db` that contains the `sakila-schema.sql` and `sakila-data.sql` files.

2. Connect to the MySQL server using the `mysql` command-line client with the following command:

```
1 shell> mysql -u root -p
```

Enter your password when prompted. A non-`root` account can be used as long as the account has privileges to create new databases.

3. Execute the `sakila-schema.sql` script to create the database structure by using the following command:

```
1 mysql> SOURCE C:/temp/sakila-db/sakila-schema.sql;
```

Replace `C:/temp/sakila-db` with the path to the `sakila-schema.sql` file on your system.

Note

On Windows, use slashes, rather than backslashes, when executing the `SOURCE` command.

4. Execute the `sakila-data.sql` script to populate the database structure with the following command:

```
1 mysql> SOURCE C:/temp/sakila-db/sakila-data.sql;
```

Replace `C:/temp/sakila-db` with the path to the `sakila-data.sql` file on your system.

5. Confirm that the sample database is installed correctly. Execute the following statements. You should see output similar to that shown here.

```
1 USE sakila;
```

```
1 Database changed
```

```
1 SHOW TABLES;
```

```
1 +-----+
2 | Tables_in_sakila
3 +-----+
4 | actor
5 | address
6 | category
7 | city
8 | country
9 | customer
10 | customer_list
11 | film
12 | film_actor
13 | film_category
14 | film_list
15 | film_text
16 | inventory
17 | language
18 | nicer_but_slower_film_list
19 | payment
20 | rental
21 | sales_by_film_category
22 | sales_by_store
23 | staff
24 | staff_list
25 | store
26 +-----+
27 22 rows in set (0.00 sec)
```



```
1 SELECT COUNT(*) FROM film;
```

COUNT(*)
1000

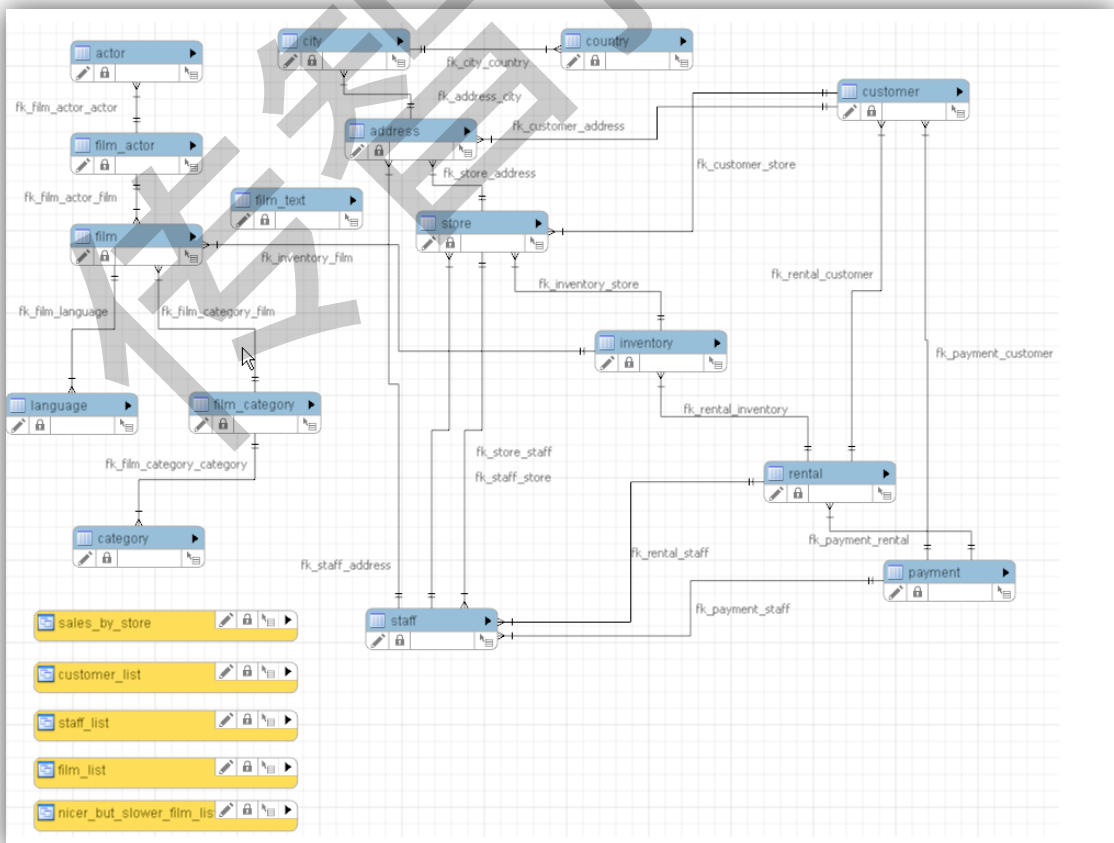
1 row in set (0.02 sec)

```
1 SELECT COUNT(*) FROM film_text;
```

COUNT(*)
1000

1 row in set (0.00 sec)

3、表结构关系



注：该表结构关系是用工具生成的。

4、 如何发现有问题的 SQL

MySQL 慢查日志的开启方式和存储格式

1、检查慢查日志是否开启：

```
show variables like 'slow_query_log'
```

```
mysql> show variables like 'slow_query_log';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| slow_query_log | OFF   |
+-----+-----+
1 row in set (0.00 sec)

mysql> █
```

```
show variables like 'slow_query_log'
```

```
//查看是否开启慢查询日志
```

```
set global slow_query_log_file='/usr/share/mysql/sql_log/mysql-slow.log'
```

```
//慢查询日志的位置
```

```
set global log_queries_not_using_indexes=on;
```

```
//开启慢查询日志
```



```
set global long_query_time=1;
```

//大于 1 秒钟的数据记录到慢日志中，如果设置为默认 0，则会有大量的信息存储在磁盘中，磁盘很容易满掉

2、查看所有日志的变量信息

```
show variables like '%log%'
```

```
mysql> show variables like '%log%';
```

+-----+-----+	
Variable_name	Value
+-----+-----+	
back_log	80
binlog_cache_size	32768
binlog_checksum	CRC32
binlog_direct_non_transactional_updates	OFF
binlog_error_action	IGNORE_ERROR
binlog_format	STATEMENT

binlog_gtid_simple_recovery	OFF	
binlog_max_flush_queue_time	0	
binlog_order_commits	ON	
binlog_row_image	FULL	
binlog_rows_query_log_events	OFF	
binlog_stmt_cache_size	32768	
binlogging_impossible_mode	IGNORE_ERROR	
expire_logs_days	0	
general_log	OFF	
general_log_file	/var/lib/mysql/mysql-host.log	
innodb_api_enable_binlog	OFF	
innodb_flush_log_at_timeout	1	
innodb_flush_log_at_trx_commit	1	
innodb_locks_unsafe_for_binlog	OFF	
innodb_log_buffer_size	8388608	
innodb_log_compressed_pages		ON
innodb_log_file_size	50331648	
innodb_log_files_in_group	2	
innodb_log_group_home_dir	./	
innodb_mirrored_log_groups	1	

innodb_online_alter_log_max_size	134217728	
innodb_undo_logs	128	
log_bin	OFF	
log_bin_basename		
log_bin_index		
log_bin_trust_function_creators	OFF	
log_bin_use_v1_row_events	OFF	
log_error	/var/log/mysqld.log	
log_output	FILE	
log_queries_not_using_indexes	ON	
log_slave_updates	OFF	
log_slow_admin_statements	OFF	
log_slow_slave_statements	OFF	
log_throttle_queries_not_using_indexes	0	
log_warnings	1	
max_binlog_cache_size	18446744073709547520	
max_binlog_size	1073741824	
max_binlog_stmt_cache_size	18446744073709547520	
max_relay_log_size	0	
relay_log		

relay_log_basename		
relay_log_index		
relay_log_info_file	relay-log.info	
relay_log_info_repository	FILE	
relay_log_purge	ON	
relay_log_recovery	OFF	
relay_log_space_limit	0	
simplified_binlog_gtid_recovery	OFF	
slow_query_log	OFF	
slow_query_log_file	/var/lib/mysql/mysql-host-slow.log	
sql_log_bin	ON	
sql_log_off	OFF	
sync_binlog	0	
sync_relay_log	10000	
sync_relay_log_info	10000	

+-----+-----+

61 rows in set (0.01 sec)

开启慢查日志：

```
show variables like 'slow_query_log'
```

//查看是否开启慢查询日志

```
set global slow_query_log_file=' /var/lib/mysql/mysql-host-slow.log '
```



//慢查询日志的位置

```
set global log_queries_not_using_indexes=on;
```

//开启慢查询日志

```
set global long_query_time=1;
```

//大于 1 秒钟的数据记录到慢日志中，如果设置为默认 0，则会有大量的信息存储在磁盘中，磁盘很容易满掉

验证慢查询日志是否开启：

在 mysql 操作中，

```
Show databases;
```

```
Use sakila;
```

```
select * from store;
```

```
select * from staff;
```

监听日志文件，看是否写入

```
tail -50f /var/lib/mysql/mysql-host-slow.log
```

```
[root@mysql-host data]# tail -50f /var/lib/mysql/mysql-host-slow.log
/usr/sbin/mysqld, Version: 5.6.40 (MySQL Community Server (GPL)). started with:
Tcp port: 3306 Unix socket: /var/lib/mysql/mysql.sock
Time          Id Command      Argument
/usr/sbin/mysqld, Version: 5.6.40 (MySQL Community Server (GPL)). started with:
Tcp port: 3306 Unix socket: /var/lib/mysql/mysql.sock
Time          Id Command      Argument
# Time: 180526 1:05:56
# User@Host: root[root] @ localhost [] Id: 4
# Query_time: 0.008039 Lock_time: 0.000052 Rows_sent: 2 Rows_examined: 2
use sakila;
SET timestamp=1527267956;
select * from store;
# Time: 180526 1:06:54
# User@Host: root[root] @ localhost [] Id: 4
# Query_time: 0.000401 Lock_time: 0.000105 Rows_sent: 2 Rows_examined: 2
SET timestamp=1527268014;
select * from staff;
```

3、MySQL 慢查日志的存储格式

如下图所示：

```
# Time: 180526 1:06:54
# User@Host: root[root] @ localhost [] Id: 4
# Query_time: 0.000401 Lock_time: 0.000105 Rows_sent: 2 Rows_examined: 2
SET timestamp=1527268014;
select * from staff;
```

说明：

- 1、# Time: 180526 1:06:54 ----->查询的执行时间
- 2、# User@Host: root[root] @ localhost [] Id: 4 ----->执行 sql 的主机信息
- 3、# Query_time: 0.000401 Lock_time: 0.000105 Rows_sent: 2 Rows_examined:

2----->SQL 的执行信息：

Query_time：SQL 的查询时间

Lock_time：锁定时间

Rows_sent：所发送的行数

Rows_examined：锁扫描的行数

- 4、SET timestamp=1527268014; ----->SQL 执行时间

5、select * from staff; -----→SQL 的执行内容

4 、 MySQL 慢 查 日 志 分 析 工 具 (mysqldumpslow)

1、介绍

如何进行查看慢查询日志，如果开启了慢查询日志，就会生成很多的数据，然后我们就可以通过对日志的分析，生成分析报表，然后通过报表进行优化。

2、用法

接下来我们查看一下这个工具的用法：

注意：在 mysql 数据库所在的服务器上，而不是在 mysql>命令行中

该工具如何使用：`mysqldumpslow -h`

```
192.168.216.200 (2) - SecureCRT
File Edit View Options Transfer Script Tools Window Help
Enter host <Alt+R>
node03 node03 (1) node03 (2) 192.168.216.200 192.168.216.200 (1) 192.168.216.200 (2) x
[root@mysql-host ~]# mysqldumpslow -h
Option h requires an argument
ERROR: bad option

Usage: mysqldumpslow [ OPTS... ] [ LOGS... ]

Parse and summarize the MySQL slow query log. Options are

--verbose      verbose
--debug        debug
--help         write this text to standard output

-v            verbose
-d            debug
-s ORDER      what to sort by (al, at, ar, c, l, r, t), 'at' is default
               al: average lock time
               ar: average rows sent
               at: average query time
               c: count
               l: lock time
               r: rows sent
               t: query time
-r            reverse the sort order (largest last instead of first)
-t NUM        just show the top n queries
-a            don't abstract all numbers to N and strings to 'S'
-n NUM        abstract numbers with at least n digits within names
-g PATTERN    grep: only consider stmts that include this string
-h HOSTNAME   hostname of db server for *-slow.log filename (can be wildcard),
               default is '*', i.e. match all
-i NAME       name of server instance (if using mysql.server startup script)
-l            don't subtract lock time from total time

[root@mysql-host ~]#
```

注意：不是在mysql>中，是在mysql服务所在的机器上

查看 verbose 信息

Mysqldumpslow -v

```
[root@mysql-host ~]# mysqldumpslow -v
Can't determine basedir from 'my_print_defaults mysqld' output: --datadir=/var/lib/mysql
--socket=/var/lib/mysql/mysql.sock
--symbolic-links=0
--sql_mode=NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES
--character-set-server=utf8
--collation-server=utf8_general_ci
--sql_mode=NO_ENGINE_SUBSTITUTION
[root@mysql-host ~]#
```

查看慢查询日志的前 10 个，mysqldumpslow 分析的结果如下

mysqldumpslow -t 10 /var/lib/mysql/mysql-slow.log


```
[root@mysql-host ~]# mysqldumpslow -t 10 /var/lib/mysql/mysql-host-slow.log
Reading mysql slow query log from /var/lib/mysql/mysql-host-slow.log
Count: 1 Time=0.01s (0s) Lock=0.00s (0s) Rows=2.0 (2), root[root]@localhost
select * from store

Count: 1 Time=0.00s (0s) Lock=0.00s (0s) Rows=2.0 (2), root[root]@localhost
select * from staff

Died at /usr/bin/mysqldumpslow line 161, <> chunk 2.
[root@mysql-host ~]#
```

如上图两条就是分析的结果，每条结果都显示是执行时间，锁定时间，发送的行数，扫描的行数

这个工具是最常用的工具，通过安装 mysql 进行附带安装，但是该工具统计的结果比较少，对我们的优化锁表现的数据还是比较少。

5、MySQL 慢查日志分析工具 (pt-query-digest)

1、介绍及作用

作为一名优秀的 mysql dba 也需要有掌握几个好用的 mysql 管理工具，所以我也一直在整理和查找一些能够便于管理 mysql 的利器。以后的一段时间内，将会花一大部分精力去搜索这些工具。

性能的管理一直都是摆在第一位的，dba 的很多工作管理层都看不到也没有办法衡量价值，但是如果一个系统慢的跟蜗牛一样，dba 通过监控调优把系统从崩溃边缘重新拉回到高铁时代。这种价值和触动应该是巨大的。（很多企业的领导认为系统跑不动了就需要换更快的 CPU、更大的内存、更快的存储，而且这还不是少数，所以 DBA 的价值也一直体现不出来，薪水自然也就不会很高）

mysql 的日志是跟踪 mysql 性能瓶颈的最快和最直接的方式了，系统性能出现瓶颈的时候，首先要打开慢查询日志，进行跟踪；这段时间关于慢查询日志的管理和查看 已经整理过两篇文章了，不经意间又发现了一个查看慢查询日志的工具：mk-query-digest，这个工具网上号称 mysql dba 必须掌握的十大工具之首。

2、安装 pt-query-digest 工具

1.1、快速安装 (注：必须先要安装 wget)

```
wget  
https://www.percona.com/downloads/percona-toolkit/2.2.16/RPM/percona-toolkit-2.2.16-1.noarch.rpm  
m && yum localinstall -y percona-toolkit-2.2.16-1.noarch.rpm
```

1.2、检查是否安装完成：

命令行中输入：pt-summary

显示如下图所示：说明安装成功！输入【[root@node03 mysql]# pt-query-digest

--help】

```
node03 node03 (1) x node03 (2) 192.168.216.200 192.168.216.200 (1) 192.168.216.200 (2)
[root@node03 mysql]# pt-query-digest --help
pt-query-digest analyzes mysql queries from slow, general, and binary log files.
It can also analyze queries from C<SHOW PROCESSLIST> and MySQL protocol data
from tcpdump. By default, queries are grouped by fingerprint and reported in
descending order of query time (i.e. the slowest queries first). If no C<FILES>
are given, the tool reads C<STDIN>. The optional C<DSN> is used for certain
options like L<"--since"> and L<"--until">. For more details, please use the
--help option, or try 'perldoc /usr/bin/pt-query-digest' for complete
documentation.

Usage: pt-query-digest [OPTIONS] [FILES] [DSN]

Options:
--ask-pass                Prompt for a password when connecting to MySQL
--attribute-aliases=a     List of attribute|alias,etc (default db|schema)
--attribute-value-limit=i A sanity limit for attribute values (default
                          4294967296)
--charset=s              -A Default character set
--config=A               Read this comma-separated list of config files;
                          if specified, this must be the first option on
                          the command line
--[no]continue-on-error  Continue parsing even if there is an error (
                          default yes)
--[no]create-history-table Create the --history table if it does not exist (
                          default yes)
--[no]create-review-table Create the --review table if it does not exist (
                          default yes)
--daemonize              Fork to the background and detach from the shell
--database=s            -D Connect to this database
--defaults-file=s        -F Only read mysql options from the given file
--embedded-attributes=a  Two Perl regex patterns to capture pseudo-
                          attributes embedded in queries
--expected-range=a       Explain items when there are more or fewer than
                          expected (default 5,10)
--explain=d              Run EXPLAIN for the sample query with this DSN
                          and print results
--filter=s               Discard events for which this Perl code doesn't
                          return true
```

```
[root@node03 mysql]# pt-summary
# Percona Toolkit System Summary Report #####
Date | 2018-05-25 10:07:51 UTC (local TZ: CST +0800)
Hostname | node03
Uptime | 2:56, 3 users, load average: 0.00, 0.00, 0.00
Platform | Linux
Release | CentOS release 6.7 (Final)
Kernel | 2.6.32-573.el6.x86_64
Architecture | CPU = 64-bit, OS = 64-bit
Threading | NPTL 2.12
SELinux | Enforcing
Virtualized | VMWare
# Processor #####
Processors | physical = 1, cores = 1, virtual = 1, hyperthreading = no
Speeds | 1x2712.003
Models | 1xIntel(R) Core(TM) i5-7200U CPU @ 2.50GHz
Caches | 1x3072 KB
# Memory #####
Total | 1.8G
Free | 617.9M
Used | physical = 1.2G, swap allocated = 1.9G, swap used = 0.0, virtual = 1.2G
Buffers | 19.8M
Caches | 637.3M
Dirty | 128 KB
UsedRSS | 498.2M
Swappiness | 60
DirtyPolicy | 20, 10
DirtyStatus | 0, 0
# Mounted Filesystems #####
Filesystem | Size Used Type Opts | Mountpoint
/dev/mapper/vg_node01-lv_root | 18G 50% ext4 rw | /
/dev/sda1 | 477M 7% ext4 rw | /boot
tmpfs | 932M 0% tmpfs rw,rootcontext="system_u:object_r:tmpfs_t:s0" | /dev/shm
# Disk Schedulers And Queue Size #####
```

1.3、工具使用简介：

pt-summary -help

wget http://percona.com/get/pt-summary

1、查看服务器信息

命令：pt-summary

2、查看磁盘开销使用信息

命令：pt-diskstats

3、查看 mysql 数据库信息

命令：pt-mysql-summary --user=root --password=123456

```
[root@node03 mysql]#
[root@node03 mysql]# pt-mysql-summary --user=root --password=admin
Warning: Using a password on the command line interface can be insecure.
# Percona Toolkit MySQL Summary Report #####
System time | 2018-05-25 10:19:52 UTC (local TZ: CST +0800)
# Instances #####
Port Data Directory      Nice OOM Socket
=====
/var/lib/mysql           0    0 /var/lib/mysql/mysql.sock
# MySQL Executable #####
Path to executable | /usr/sbin/mysqld
Has symbols | Yes
# Report On Port 3306 #####
User | root@%
Time | 2018-05-25 18:19:52 (CST)
Hostname | node03
Version | 5.6.40 MySQL Community Server (GPL)
Built On | Linux x86_64
Started | 2018-05-25 15:54 (up 0+02:25:24)
Databases | 4
Datadir | /var/lib/mysql/
Processes | 2 connected, 1 running
Replication | Is not a slave, has 0 slaves connected
Pidfile | /var/run/mysqld/mysqld.pid (exists)
# Processlist #####
Command          COUNT(*) Working SUM(Time) MAX(Time)
-----
Query            1          1      0      0
Sleep            1          0    8000    8000
User
Command          COUNT(*) Working SUM(Time) MAX(Time)
```

4、分析慢查询日志

```
命令：pt-query-digest /data/mysql/data/db-3-12-slow.log
```

5、查找 mysql 的从库和同步状态

```
命令：pt-slave-find --host=localhost --user=root --password=123456
```

6、查看 mysql 的死锁信息

```
pt-deadlock-logger --user=root --password=123456 localhost
```

7、从慢查询日志中分析索引使用情况

```
pt-index-usage slow_20131009.log
```

8、查找数据库表中重复的索引

```
pt-duplicate-key-checker --host=localhost --user=root --password=123456
```

9、查看 mysql 表和文件的当前活动 IO 开销

```
pt-ioprofile
```

10、查看不同 mysql 配置文件的差异

```
pt-config-diff /etc/my.cnf /etc/my_master.cnf
```

11、pt-find 查找 mysql 表和执行命令，示例如下

查找数据库里大于 2G 的表：

```
pt-find --user=root --password=123456 --tablesize +2G
```

查找 10 天前创建，MyISAM 引擎的表：

```
pt-find --user=root --password=123456 --ctime +10 --engine MyISAM
```

查看表和索引大小并排序

```
pt-find --user=root --password=123456 --printf "%T\t%D.%N\n" | sort -rn
```

12、pt-kill 杀掉符合标准的 mysql 进程

显示查询时间大于 60 秒的查询

```
pt-kill --user=root --password=123456 --busy-time 60 --print
```

kill 掉大于 60 秒的查询

```
pt-kill --user=root --password=123456 --busy-time 60 --kill
```

13、查看 mysql 授权

```
1、pt-show-grants --user=root --password=123456
```

```
2、pt-show-grants --user=root --password=123456 --separate --revoke
```

14、验证数据库复制的完整性

```
pt-table-checksum --user=root --password=123456
```

15、附录：

输出到文件

```
pt-query-digest slow-log > slow_log.report
```

输出到数据库表

```
pt-query-digest slow.log --review \  
h=127.0.0.1,D=test,p=root,P=3306,u=root,t=query_review \  
--create-reviewtable \  
--review-history t= hostname_slow
```

6、如何通过慢查日志发现有问题的 SQL

1、查询次数多且每次查询占用时间长的 sql

通常为 pt-query-digest 分析的前几个查询；该工具可以很清楚的看出每个 SQL 执

行的次数及百分比等信息，执行的次数多，占比比较大的 SQL

2、IO 大的 sql

注意 pt-query-digest 分析中的 Rows examine 项。扫描的行数越多，IO 越大。

3、未命中的索引的 SQL

注意 pt-query-digest 分析中的 Rows examine 和 Rows Send 的对比。说明该 SQL

的索引命中率不高，对于这种 SQL，我们要重点进行关注。

7、通过 explain 查询分析 SQL 的执行计划

1、使用 explain 查询 SQL 的执行计划

SQL 的执行计划侧面反映出了 SQL 的执行效率，具体执行方式如下所示：

在执行的 SQL 前面加上 explain 关键词即可；

```
mysql> explain select * from staff;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1  | SIMPLE     | staff | ALL  | NULL          | NULL | NULL    | NULL | 2    | NULL  |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

2、每个字段的说明：

1)、id 列数字越大越先执行，如果说数字一样大，那么就从上往下依次执行，id 列为 null 的就表示这是一个结果集，不需要使用它来进行查询。

2)、select_type 列常见的有：

A：simple：表示不需要 union 操作或者不包含子查询的简单 select 查询。有连接查询时，



外层的查询为 simple，且只有一个

B : primary : 一个需要 union 操作或者含有子查询的 select，位于最外层的单位查询的 select_type 即为 primary。且只有一个

C : union : union 连接的两个 select 查询，第一个查询是 dervied 派生表，除了第一个表外，第二个以后的表 select_type 都是 union

D : dependent union : 与 union 一样，出现在 union 或 union all 语句中，但是这个查询要受到外部查询的影响

E : union result : 包含 union 的结果集，在 union 和 union all 语句中,因为它不需要参与查询，所以 id 字段为 null

F : subquery : 除了 from 子句中包含的子查询外，其他地方出现的子查询都可能是 subquery

G : dependent subquery : 与 dependent union 类似，表示这个 subquery 的查询要受到外部表查询的影响

H : derived : from 字句中出现的子查询，也叫做派生表，其他数据库中可能叫做内联视图或嵌套 select

3) table

显示的查询表名，如果查询使用了别名，那么这里显示的是别名，如果不涉及对数据表的操作，那么这显示为 null，如果显示为尖括号括起来的<derived N>就表示这个是临时表，后边的 N 就是执行计划中的 id，表示结果来自于这个查询产生。如果是尖括号括起来的<union M,N>，与<derived N>类似，也是一个临时表，表示这个结果来自于 union 查询的 id 为 M,N 的结果集。

4) type

依次从好到差：system，const，eq_ref，ref，fulltext，ref_or_null，unique_subquery，index_subquery，range，index_merge，index，ALL，除了 all 之外，其他的 type 都可以使用到索引，除了 index_merge 之外，其他的 type 只可以用到一个索引

A：system：表中只有一行数据或者是空表，且只能用于 myisam 和 memory 表。如果是 Innodb 引擎表，type 列在这个情况通常都是 all 或者 index

B：const：使用唯一索引或者主键，返回记录一定是 1 行记录的等值 where 条件时，通常 type 是 const。其他数据库也叫做唯一索引扫描

C：eq_ref：出现在要连接过个表的查询计划中，驱动表只返回一行数据，且这行数据是第二个表的主键或者唯一索引，且必须为 not null，唯一索引和主键是多列时，只有所有的列都用作比较时才会出现 eq_ref

D : ref : 不像 eq_ref 那样要求连接顺序，也没有主键和唯一索引的要求，只要使用相等条件检索时就可能出现，常见与辅助索引的等值查找。或者多列主键、唯一索引中，使用第一个列之外的列作为等值查找也会出现，总之，返回数据不唯一的等值查找就可能出现。

E : fulltext : 全文索引检索，要注意，全文索引的优先级很高，若全文索引和普通索引同时存在时，mysql 不管代价，优先选择使用全文索引

F : ref_or_null : 与 ref 方法类似，只是增加了 null 值的比较。实际用的不多。

G : unique_subquery : 用于 where 中的 in 形式子查询，子查询返回不重复值唯一值

H : index_subquery : 用于 in 形式子查询使用到了辅助索引或者 in 常数列表，子查询可能返回重复值，可以使用索引将子查询去重。

I : range : 索引范围扫描，常见于使用 >, <, is null, between, in, like 等运算符的查询中。

J : index_merge : 表示查询使用了两个以上的索引，最后取交集或者并集，常见 and, or 的条件使用了不同的索引，官方排序这个在 ref_or_null 之后，但是实际上由于要读取所有索引，性能可能大部分时间都不如 range

K : index : 索引全表扫描，把索引从头到尾扫一遍，常见于使用索引列就可以处理不需要读取数据文件的查询、可以使用索引排序或者分组的查询。

L : all : 这个就是全表扫描数据文件，然后再在 server 层进行过滤返回符合要求的记录。

5). possible_keys

查询可能使用到的索引都会在这里列出来

6). key

查询真正使用到的索引，select_type 为 index_merge 时，这里可能出现两个以上的索引，其他的 select_type 这里只会出现一个。

7). key_len

用于处理查询的索引长度，如果是单列索引，那就整个索引长度算进去，如果是多列索引，那么查询不一定都能使用到所有的列，具体使用到了多少个列的索引，这里就会计算进去，没有使用到的列，这里不会计算进去。留意下这个列的值，算一下你的多列索引总长度就知道有没有使用到所有的列了。要注意，mysql 的 ICP 特性使用到的索引不会计入其中。另外，key_len 只计算 where 条件用到的索引长度，而排序和分组就算用到了索引，也不会计算到 key_len 中。

8). ref

如果是使用的常数等值查询，这里会显示 const，如果是连接查询，被驱动表的执行计划这里会显示驱动表的关联字段，如果是条件使用了表达式或者函数，或者条件列发生了内部隐式转换，这里可能显示为 func

9). rows

这里是执行计划中估算的扫描行数，不是精确值

10). extra

这个列可以显示的信息非常多，有几十种，常用的有

A : distinct : 在 select 部分使用了 distinct 关键字

B : no tables used : 不带 from 字句的查询或者 From dual 查询

C : 使用 not in()形式子查询或 not exists 运算符的连接查询，这种叫做反连接。即，一般连接查询是先查询内表，再查询外表，反连接就是先查询外表，再查询内表。

D : using filesort : 排序时无法使用到索引时，就会出现这个。常见于 order by 和 group by 语句中

E : using index : 查询时不需要回表查询，直接通过索引就可以获取查询的数据。

F : using join buffer (block nested loop), using join buffer (batched key access) : 5.6.x

之后的版本优化关联查询的 BNL , BKA 特性。主要是减少内表的循环数量以及比较顺序地扫描查询。

G : using sort_union , using_union , using intersect , using sort_intersection :

using intersect : 表示使用 and 的各个索引的条件时，该信息表示是从处理结果获取交集

using union : 表示使用 or 连接各个使用索引的条件时，该信息表示从处理结果获取并集

using sort_union 和 using sort_intersection : 与前面两个对应的类似，只是他们是出现在用 and 和 or 查询信息量大时，先查询主键，然后进行排序合并后，才能读取记录并返回。

H : using temporary : 表示使用了临时表存储中间结果。临时表可以是内存临时表和磁盘临时表，执行计划中看不出来，需要查看 status 变量，used_tmp_table，used_tmp_disk_table 才能看出来。

I : using where : 表示存储引擎返回的记录并不是所有的都满足查询条件，需要在 server 层进行过滤。查询条件中分为限制条件和检查条件，5.6 之前，存储引擎只能根据限制条件扫描数据并返回，然后 server 层根据检查条件进行过滤再返回真正符合查询的数据。5.6.x 之后支持 ICP 特性，可以把检查条件也下推到存储引擎层，不符合检查条件和限制条件的数据，直接不读取，这样就大大减少了存储引擎扫描的记录数量。extra 列显示 using index condition

J : firstmatch(tb_name) : 5.6.x 开始引入的优化子查询的新特性之一，常见于 where 字句含有 in()类型的子查询。如果内表的数据量比较大，就可能出现这个

K : loosescan(m..n) : 5.6.x 之后引入的优化子查询的新特性之一，在 in()类型的子查询中，

子查询返回的可能有重复记录时，就可能出现这个

除了这些之外，还有很多查询数据字典库，执行计划过程中就发现不可能存在结果的一些提示信息

11) filtered

使用 explain extended 时会出现这个列，5.7 之后的版本默认就有这个字段，不需要使用 explain extended 了。这个字段表示存储引擎返回的数据在 server 层过滤后，剩下多少满足查询的记录数量的比例，注意是百分比，不是具体记录数。

附图：

explain返回各列的含义
table：显示这一行的数据是关于哪张表的
type：这是重要的列，显示连接使用了何种类型。从最好到最差的连接类型为const、eq_reg、ref、range、index 和ALL
possible_keys：显示可能应用在这张表中的索引。如果为空，没有可能的索引。
key：实际使用的索引。如果为NULL，则没有使用索引。
key_len：使用的索引的长度。在不损失精确性的情况下，长度越短越好
ref：显示索引的哪一列被使用了，如果可能的话，是一个常数
rows：MYSQL认为必须检查的用来返回请求数据的行数

explain返回各列的含义
extra列需要注意的返回值
Using filesort: 看到这个的时候，查询就需要优化了。MYSQL需要进行额外的步骤来发现如何对返回的行排序。它根据连接类型以及存储排序键值和匹配条件的全部行的行指针来排序全部行
Using temporary 看到这个的时候，查询需要优化了。这里，MYSQL需要创建一个临时表来存储结果，这通常发生在对不同的列集进行ORDER BY上，而不是GROUP BY上

3、具体慢查询的优化案例

1、函数 Max()的优化

用途：查询最后支付时间-优化 max () 函数

语句：

```
select max(payment_date) from payment;
```

```
mysql> select max(payment_date) from payment;
+-----+
| max(payment_date) |
+-----+
| 2006-02-14 15:16:03 |
+-----+
1 row in set (0.00 sec)

mysql> 
```

执行计划：

```
explain select max(payment_date) from payment;
```

```
mysql> explain select max(payment_date) from payment;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | payment | ALL | NULL | NULL | NULL | NULL | 16086 | NULL |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> 
```




```
mysql> explain select max(payment_date) from payment \G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: payment
         type: ALL
possible_keys: NULL
         key: NULL
        key_len: NULL
         ref: NULL
         rows: 16086
      Extra: NULL
1 row in set (0.00 sec)

mysql> █
```

可以看到显示的执行计划，并不是很高效，可以拖慢服务器的效率，如何优化了？

创建索引

```
create index inx_paydate on payment(payment_date);
```

```
mysql> create index inx_paydate on payment(payment_date);
Query OK, 0 rows affected (0.06 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> █
```

```
mysql> explain select max(payment_date) from payment \G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: NULL
         type: NULL
possible_keys: NULL
         key: NULL
        key_len: NULL
         ref: NULL
         rows: NULL
      Extra: Select tables optimized away
1 row in set (0.00 sec)

mysql>
```

索引是顺序操作的，不需要扫描表，执行效率就会比较恒定，

2、函数 Count()的优化

需求：在一条 SQL 中同时查出 2006 年和 2007 年电影的数量

错误的方式：

语句：

```
select count(release_year='2006' or release_year='2007') from film;
```

```
mysql> select count(release_year='2006' or release_year='2007') from film;
+-----+
| count(release_year='2006' or release_year='2007') |
+-----+
| 1000 |
+-----+
1 row in set (0.00 sec)
mysql>
```

2006 和 2007 年分别是多少，判断不出来

```
select count(*) from film where release_year='2006' or release_year='2007';
```

```
mysql> select count(*) from film where release_year='2006' or release_year='2007';
+-----+
| count(*) |
+-----+
| 1000 |
+-----+
1 row in set (0.00 sec)
mysql>
```

正确的编写方式：

```
select count(release_year='2006' or null) as '06films',count(release_year='2007' or null)
```



as '07films' from film;

```
mysql> select count(release_year='2006' or null) as '06films',count(release_year='2007' or null) as '07films' from film;
+-----+-----+
| 06films | 07films |
+-----+-----+
|      1000 |          0 |
+-----+-----+
1 row in set (0.00 sec)

mysql>
```

区别：count (*) 和 count (id)

创建表并插入语句

create table t(id int);

insert into t values(1),(2),(null);

```
mysql> create table t(id int);
Query OK, 0 rows affected (0.02 sec)

mysql> insert into t values(1),(2),(null);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
```

Count (*) : select count(*)from t;

```
mysql> select count(*)from t;
+-----+
| count(*) |
+-----+
|          3 |
+-----+
1 row in set (0.00 sec)

mysql> █
```

Count (id) : select count(id)from t;

```
mysql> select count(id)from t;
+-----+
| count(id) |
+-----+
|          2 |
+-----+
1 row in set (0.00 sec)

mysql>
```

说明：

Count (id) 是不包含 null 的值

Count (*) 是包含 null 的值

3、子查询的优化

子查询是我们在开发过程中经常使用的一种方式，在通常情况下，需要把子查询优化为 join 查询但在优化是需要注意关联键是否有一对多的关系，要注意重复数据。

查看我们所创建的 t 表

```
show create table t;
```

```
mysql> show create table t;
+-----+-----+
| Table | Create Table |
+-----+-----+
| t     | CREATE TABLE `t` (
  `id` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
+-----+-----+
1 row in set (0.00 sec)

mysql>
```

接下来我们创建一个 t1 表

```
create table t1(tid int);
```

并插入一条数据

```
mysql> insert into t1 values(1);
Query OK, 1 row affected (0.01 sec)

mysql>
```

我们要进行一个子查询，需求：查询 t 表中 id 在 t1 表中 tid 的所有数据；

```
select * from t where t.id in (select t1.tid from t1);
```

```
mysql> select * from t where t.id in (select t1.tid from t1);
+----+
| id |
+----+
|  1 |
+----+
1 row in set (0.00 sec)

mysql>
```

接下来我们用 join 的操作来进行操作

```
select id from t join t1 on t.id =t1.tid;
```



```
mysql> select id from t join t1 on t.id =t1.tid;
+-----+
| id    |
+-----+
| 1     |
+-----+
1 row in set (0.00 sec)

mysql>
```

通过上面结果来看，查询的结果是一致的，我们就将子查询的方式优化为 join 操作。

接下来，我们在 t1 表中再插入一条数据

```
insert into t1 values (1);
```

```
select * from t1;
```

```
mysql> select * from t1;
+-----+
| tid   |
+-----+
| 1     |
| 1     |
+-----+
2 rows in set (0.00 sec)

mysql>
```

在这种情况下，如果我们使用子查询方式进行查询，返回的结果就是如下图所示：

```
mysql> select * from t where t.id in (select t1.tid from t1);
+-----+
| id    |
+-----+
| 1     |
+-----+
1 row in set (0.00 sec)

mysql>
```

如果使用 join 方式进行查找，如下图所示：

```
mysql> select id from t join t1 on t.id =t1.tid;
+-----+
| id    |
+-----+
|      1|
|      1|
+-----+
2 rows in set (0.00 sec)

mysql> 
```

在这种情况下出现了一对多的关系，会出现数据的重复，我们为了方式数据重复，不得不使用 distinct 关键词进行去重操作

```
select distinct id from t join t1 on t.id =t1.tid;
```

```
mysql> select distinct id from t join t1 on t.id =t1.tid;
+-----+
| id    |
+-----+
|      1|
+-----+
1 row in set (0.00 sec)

mysql> 
```

注意：这个一对多的关系是我们开发过程中遇到的一个坑，出现数据重复，需要大家注意一下。

例子：查询 sandra 出演的所有影片：

```
explain select title,release_year,length

from film

where film_id in (

select film_id from film_actor where actor_id in (

select actor_id from actor where first_name='sandra'));
```

4、group by 的优化

最好使用同一表中的列，

需求：每个演员所参演影片的数量-（影片表和演员表）

```
explain select actor.first_name,actor.last_name,count(*)  
  
from sakila.film_actor  
  
inner join sakila.actor using(actor_id)  
  
group by film_actor.actor_id;
```

```
mysql> explain select actor.first_name,actor.last_name,count(*) from sakila.film_actor inner join sakila.actor using(a  
ctor_id) group by film_actor.actor_id \G  
***** 1. row *****  
id: 1  
select_type: SIMPLE  
table: actor  
type: ALL  
possible_keys: PRIMARY  
key: NULL  
key_len: NULL  
ref: NULL  
rows: 200  
Extra: Using temporary; Using filesort  
***** 2. row *****  
id: 1  
select_type: SIMPLE  
table: film_actor  
type: ref  
possible_keys: PRIMARY,idx_fk_film_id  
key: PRIMARY  
key_len: 2  
ref: sakila.actor.actor_id  
rows: 13  
Extra: Using index  
2 rows in set (0.00 sec)  
mysql>
```

优化后的 SQL：

```
explain select actor.first_name,actor.last_name,c.cnt  
  
from sakila.actor inner join (  
  
select actor_id,count(*) as cnt from sakila.film_actor group by actor_id  
  
)as c using(actor_id);
```



```
mysql> explain select actor.first_name,actor.last_name,c.cnt from sakila.actor inner join ( select actor_id,count(*) a
s cnt from sakila.film_actor group by actor_id )as c using(actor_id)\G
***** 1. row *****
id: 1
select_type: PRIMARY
table: actor
type: ALL
possible_keys: PRIMARY
key: NULL
key_len: NULL
ref: NULL
rows: 200
Extra: NULL
***** 2. row *****
id: 1
select_type: PRIMARY
table: <derived2>
type: ref
possible_keys: <auto_key0>
key: <auto_key0>
key_len: 2
ref: sakila.actor.actor_id
rows: 27
Extra: NULL
***** 3. row *****
id: 2
select_type: DERIVED
table: film_actor
type: index
possible_keys: PRIMARY,idx_fk_film_id
key: PRIMARY
key_len: 4
ref: NULL
rows: 5462
Extra: Using index
3 rows in set (0.00 sec)
mysql>
```

说明：从上面的执行计划来看，这种优化后的方式没有使用临时文件和文件排序的方式了，取而代之的是使用了索引。查询效率老高了。

这个时候我们表中的数据比较大，会大量的占用 IO 操作，优化了 sql 执行的效率，节省了服务器的资源，因此我们就需要优化。

注意：

- 1、mysql 中 using 关键词的作用：也就是说要使用 using,那么表 a 和表 b 必须要有相同的列。
- 2、在用 Join 进行多表联合查询时，我们通常使用 On 来建立两个表的关系。其实还有一个更方便的关键字，那就是 Using。

3、如果两个表的关联字段名是一样的，就可以使用 Using 来建立关系，简洁明了。

5、Limit 查询的优化

Limit 常用于分页处理，时长会伴随 order by 从句使用，因此大多时候回使用 Filesorts 这样会造成大量的 IO 问题。

例子：

需求：查询影片 id 和描述信息，并根据主题进行排序，取出从序号 50 条开始的 5 条数据。

据。

```
select film_id,description from sakila.film order by title limit 50,5;
```

执行的结果：

film_id	description
51	A Insightful Panorama of a Forensic Psychologist And a Mad Cow who must Build a Mad Scientist in The First Manned Space Station
52	A Thrilling Documentary of a Composer And a Monkey who must Find a Feminist in California
53	A Epic Drama of a Madman And a Cat who must Face a A Shark in An Abandoned Amusement Park
54	A Awe-Inspiring Drama of a Car And a Pastry Chef who must Chase a Crocodile in The First Manned Space Station
55	A Awe-Inspiring Story of a Feminist And a Cat who must Conquer a Dog in A Monastery

在查看一下它的执行计划：

```
mysql> explain select film_id,description from sakila.film order by title limit 50,5;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1  | SIMPLE     | film  | ALL  | NULL         | NULL | NULL    | NULL | 1000 | Using filesort |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> explain select film_id,description from sakila.film order by title limit 50,5\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: film
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 1000
Extra: Using filesort
1 row in set (0.00 sec)
mysql>
```

对于这种操作，我们该用什么样的优化方式了？

优化步骤 1：

使用有索引的列或主键进行 order by 操作，因为大家知道，innodb 是按照主键的逻辑顺序进行排序的。可以避免很多的 IO 操作。

```
select film_id,description from sakila.film order by film_id limit 50,5;
```

film_id	description
51	A Insightful Panorama of a Forensic Psychologist And a Mad Cow who must Build a Mad Scientist in The First Manned Space Station
52	A Thrilling Documentary of a Composer And a Monkey who must Find a Feminist in California
53	A Epic Drama of a Madman And a Cat who must Face a A Shark in An Abandoned Amusement Park
54	A Awe-Inspiring Drama of a Car And a Pastry Chef who must Chase a Crocodile in The First Manned Space Station
55	A Awe-Inspiring Story of a Feminist And a Cat who must Conquer a Dog in A Monastery

查看一下执行计划

```
mysql> explain select film_id,description from sakila.film order by film_id limit 50,5\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: film
         type: index
possible_keys: NULL
      key: PRIMARY
     key_len: 2
        ref: NULL
       rows: 55
      Extra: NULL
1 row in set (0.00 sec)

mysql>
```

那如果我们获取从 500 行开始的 5 条记录，执行计划又是什么样的了？

```
explain select film_id,description from sakila.film order by film_id limit 500,5\G
```

```
mysql> explain select film_id,description from sakila.film order by film_id limit 500,5\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: film
         type: index
possible_keys: NULL
      key: PRIMARY
     key_len: 2
        ref: NULL
       rows: 505
      Extra: NULL
1 row in set (0.00 sec)

mysql>
```

```
mysql> explain select film_id,description from sakila.film order by film_id limit 1000,5\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: film
         type: index
possible_keys: NULL
      key: PRIMARY
     key_len: 2
        ref: NULL
       rows: 1000
      Extra: NULL
1 row in set (0.00 sec)

mysql>
```

随着我们翻页越往后，IO 操作会越来越大的，如果一个表有几千万行数据，翻页越后面，

会越来越慢，因此我们要进一步的来优化。

优化步骤 2、记录上次返回的主键，在下次查询时使用主键过滤。(说明：避免了数据量大时扫描过多的记录)

上次 limit 是 50,5 的操作，因此我们在这次优化过程需要使用上次的索引记录值，

```
select film_id,description from sakila.film where film_id >55 and film_id<=60 order by film_id limit 1,5;
```

查看执行计划：

```
mysql> explain select film_id,description from sakila.film where film_id >55 and film_id<=60 order by film_id limit 1,5\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: film
type: range
possible_keys: PRIMARY
key: PRIMARY
key_len: 2
ref: NULL
rows: 5
Extra: Using where
1 row in set (0.00 sec)
mysql>
```

```
mysql> explain select film_id,description from sakila.film where film_id >60 and film_id<=65 order by film_id limit 1,5\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: film
type: range
possible_keys: PRIMARY
key: PRIMARY
key_len: 2
ref: NULL
rows: 5
Extra: Using where
1 row in set (0.00 sec)
mysql>
```

```
mysql> explain select film_id,description from sakila.film where film_id >600 and film_id<=605 order by film_id limit 1,5\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: film
type: range
possible_keys: PRIMARY
key: PRIMARY
key_len: 2
ref: NULL
rows: 5
Extra: Using where
1 row in set (0.00 sec)
mysql>
```

结论：扫描行数不变，执行计划是很固定，效率也是很固定的

注意事项：

主键要顺序排序并连续的，如果主键中间空缺了某一列，或者某几列，会出现列出数据不足 5 行的数据；如果不连续的情况，建立一个附加的列 index_id 列，保证这一列数据要

自增的，并添加索引即可。

6、索引的优化

1、什么是索引？

索引的作用相当于图书的目录，可以根据目录中的页码快速找到所需的内容。

数据库使用索引以找到特定值，然后顺指针找到包含该值的行。在表中建立索引，然后在索引中找到符合查询条件的索引值，最后通过保存在索引中的 ROWID (相当于页码) 快速找到表中对应的记录。索引的建立是表中比较有指向性的字段，相当于目录，比如说行政区划代码，同一个地域的行政区划代码都是相同的，那么给这一列加上索引，避免让它重复扫描，从而达到优化的目的！

2、如何创建索引

在执行 CREATE TABLE 语句时可以创建索引，也可以单独用 CREATE INDEX 或 ALTER TABLE 来为表增加索引。

1、ALTER TABLE

ALTER TABLE 用来创建普通索引、UNIQUE 索引或 PRIMARY KEY 索引。

```
ALTER TABLE table_name ADD INDEX index_name (column_list)
```

```
ALTER TABLE table_name ADD UNIQUE (column_list)
```

```
ALTER TABLE table_name ADD PRIMARY KEY (column_list)
```

说明：其中 table_name 是要增加索引的表名，column_list 指出对哪些列进行索引，多

列时各列之间用逗号分隔。索引名 `index_name` 可选，缺省时，MySQL 将根据第一个索引列赋一个名称。另外，ALTER TABLE 允许在单个语句中更改多个表，因此可以在同时创建多个索引。

2、CREATE INDEX

CREATE INDEX 可对表增加普通索引或 UNIQUE 索引。

```
CREATE INDEX index_name ON table_name (column_list)
```

```
CREATE UNIQUE INDEX index_name ON table_name (column_list)
```

说明：`table_name`、`index_name` 和 `column_list` 具有与 ALTER TABLE 语句中相同的含义，索引名不可选。另外，不能用 CREATE INDEX 语句创建 PRIMARY KEY 索引。

3、索引类型

在创建索引时，可以规定索引能否包含重复值。如果不包含，则索引应该创建为 PRIMARY KEY 或 UNIQUE 索引。对于单列惟一性索引，这保证单列不包含重复的值。对于多列惟一性索引，保证多个值的组合不重复。

PRIMARY KEY 索引和 UNIQUE 索引非常类似。

事实上，PRIMARY KEY 索引仅是一个具有名称 PRIMARY 的 UNIQUE 索引。这表示一个表只能包含一个 PRIMARY KEY，因为一个表中不可能具有两个同名的索引。

下面的 SQL 语句对 students 表在 sid 上添加 PRIMARY KEY 索引。

```
ALTER TABLE students ADD PRIMARY KEY (sid)
```

4、删除索引

可利用 ALTER TABLE 或 DROP INDEX 语句来删除索引。类似于 CREATE INDEX

语句，DROP INDEX 可以在 ALTER TABLE 内部作为一条语句处理，语法如下。

```
DROP INDEX index_name ON talbe_name
```

```
ALTER TABLE table_name DROP INDEX index_name
```

```
ALTER TABLE table_name DROP PRIMARY KEY
```

其中，前两条语句是等价的，删除掉 table_name 中的索引 index_name。

第 3 条语句只在删除 PRIMARY KEY 索引时使用，因为一个表只可能有一个 PRIMARY KEY 索引，因此不需要指定索引名。如果没有创建 PRIMARY KEY 索引，但表具有一个或多个 UNIQUE 索引，则 MySQL 将删除第一个 UNIQUE 索引。

如果从表中删除了某列，则索引会受到影响。对于多列组合的索引，如果删除其中的某列，则该列也会从索引中删除。如果删除组成索引的所有列，则整个索引将被删除。

5、查看索引

```
mysql> show index from tblname;
```

```
mysql> show keys from tblname;
```

6、什么情况下，使用索引了？

1、表的主关键字

- 2、自动建立唯一索引
- 3、表的字段唯一约束
- 4、直接条件查询的字段（在 SQL 中用于条件约束的字段）
- 5、查询中与其它表关联的字段
- 6、查询中排序的字段（排序的字段如果通过索引去访问那将大大提高排序速度）
- 7、查询中统计或分组统计的字段
- 8、表记录太少（如果一个表只有 5 条记录，采用索引去访问记录的话，那首先需访问索引表，再通过索引表访问数据表，一般索引表与数据表不在同一个数据块）
- 9、经常插入、删除、修改的表（对一些经常处理的业务表应在查询允许的情况下尽量减少索引）
- 10、数据重复且分布平均的表字段（假如一个表有 10 万行记录，有一个字段 A 只有 T 和 F 两种值，且每个值的分布概率大约为 50%，那么对这种表 A 字段建索引一般不会提高数据库的查询速度。）
- 11、经常和主字段一块查询但主字段索引值比较多的表字段
- 12、对千万级 MySQL 数据库建立索引的事项及提高性能的手段

3、如何选择合适的列建立索引

- 1、在 where 从句，group by 从句，order by 从句，on 从句中的列添加索引
 - 2、索引字段越小越好（因为数据库数据存储单位是以“页”为单位的，数据存储的越多，IO 也会越大）
 - 3、离散度大的列放到联合索引的前面
- 例子：


```
select * from payment where staff_id =2 and customer_id =584;
```

注意:

是 index (staff_id , customer_id) 好，还是 index (customer_id , staff_id) 好

那我们怎么进行验证离散度好了？

A、我们先查看一下表结构

```
desc payment;
```

```
mysql> desc payment;
```

Field	Type	Null	Key	Default	Extra
payment_id	smallint(5) unsigned	NO	PRI	NULL	auto_increment
customer_id	smallint(5) unsigned	NO	MUL	NULL	
staff_id	tinyint(3) unsigned	NO	MUL	NULL	
rental_id	int(11)	YES	MUL	NULL	
amount	decimal(5,2)	NO		NULL	
payment_date	datetime	NO	MUL	NULL	
last_update	timestamp	NO		CURRENT_TIMESTAMP	on update CURRENT_TIMESTAMP

7 rows in set (0.00 sec)

B、分别查看这两个字段中不同的 id 的数量，数量越多，则表明离散程度越大：因此可

以通过下图看出：customer_id 离散程度大。

```
mysql> select count(distinct staff_id ),count(distinct customer_id) from payment;
```

count(distinct staff_id)	count(distinct customer_id)
2	599

1 row in set (0.03 sec)

```
mysql>
```

结论：由于 customer_id 离散程度大，使用 index (customer_id , staff_id) 好

C、mysql 联合索引

①命名规则：表名_字段名

1、需要加索引的字段，要在 where 条件中

2、数据量少的字段不需要加索引

3、如果 where 条件中是 OR 关系，加索引不起作用

4、符合最左原则

②什么是联合索引

1、两个或更多个列上的索引被称作联合索引，又被称为是复合索引。

2、利用索引中的附加列，您可以缩小搜索的范围，但使用一个具有两列的索引 不同于使用两个单独的索引。复合索引的结构与电话簿类似，人名由姓和名构成，电话簿首先按姓氏对进行排序，然后按名字对有相同姓氏的人进行排序。如果您知道姓，电话簿将非常有用；如果您知道姓和名，电话簿则更为有用，但如果您只知道名不姓，电话簿将没有用处。

所以说创建复合索引时，应该仔细考虑列的顺序。对索引中的所有列执行搜索或仅对前列执行搜索时，复合索引非常有用；仅对后面的任意列执行搜索时，复合索引则没有用处。

4、索引优化 SQL 的方法

1、索引的维护及优化（重复及冗余索引）

增加索引会有利于查询效率，但会降低 insert，update，delete 的效率，但实际上往往不是这样的，过多的索引会不但会影响使用效率，同时会影响查询效率，这是由于数据库进行查询分析时，首先要选择使用哪一个索引进行查询，如果索引过多，分析过程就会越慢，这样同样的减少查询的效率，因此我们要知道如何增加，有时候要知道维护和删除不需要的索引

2、如何找到重复和冗余的索引

重复索引：

重复索引是指相同的列以相同的顺序建立的同类型的索引，如下表中的 primary key 和

ID 列上的索引就是重复索引

```
create table test(  
id int not null primary key,  
name varchar(10) not null,  
title varchar(50) not null,  
unique(id)  
)engine=innodb;
```

冗余索引：

冗余索引是指多个索引的前缀列相同，或是在联合索引中包含了主键的索引，下面这个例子中 key (name , id) 就是一个冗余索引。

```
create table test(  
id int not null primary key,  
name varchar(10) not null,  
title varchar(50) not null,  
key(name,id)  
)engine=innodb;
```

说明：对于 innodb 来说，每一个索引后面，实际上都会包含主键，这时候我们建立的联合索引，又人为的把主键包含进去，那么这个时候就是一个冗余索引。

3、如何查找重复索引

工具：使用 pt-duplicate-key-checker 工具检查重复及冗余索引

```
pt-duplicate-key-checker -uroot -padmin -h 127.0.0.1
```

```
[root@mysql-host ~]# pt-duplicate-key-checker -uroot -padmin -h 127.0.0.1
# #####
# Summary of indexes
# #####
# Total Indexes 92
[root@mysql-host ~]#
```

4、索引维护的方法

由于业务变更，某些索引是后续不需要使用的，就要进行删除。

在 mysql 中，目前只能通过慢查询日志配合 pt-index-usage 工具来进行索引使用情况的分析；

```
pt-index-usage -uroot -padmin /var/lib/mysql/mysql-host-slow.log
```

```
[root@mysql-host ~]# pt-index-usage -uroot -padmin /var/lib/mysql/mysql-host-slow.log
ALTER TABLE `sakila`.`film` DROP KEY `idx_fk_language_id`, DROP KEY `idx_fk_original_language_id`, DROP KEY `idx_title`; -- type:non-unique
ALTER TABLE `sakila`.`staff` DROP KEY `idx_fk_address_id`, DROP KEY `idx_fk_store_id`; -- type:non-unique
ALTER TABLE `sakila`.`store` DROP KEY `idx_fk_address_id`; -- type:non-unique
[root@mysql-host ~]#
```

附：<https://www.percona.com/downloads/>

5、注意事项

设计好 MySQL 的索引可以让你的数据库飞起来，大大的提高数据库效率。设计 MySQL 索引的时候有以下几点注意：

1，创建索引

对于查询占主要的应用来说，索引显得尤为重要。很多时候性能问题很简单的就是因为忘记了添加索引而造成的，或者说没有添加更为有效的索引导致。如果不加索引的话，那么查找任何哪怕只是一条特定的数据都会进行一次全表扫描，如果一张表的数据量很大而符合条件的结果又很少，那么不加索引会引起致命的性能下降。

但是也不是什么情况都非得建索引不可，比如性别可能就只有两个值，建索引不仅没什么优势，还会影响到更新速度，这被称为过度索引。

2，复合索引

比如有一条语句是这样的：`select * from users where area='beijing' and age=22;`

如果我们是在 `area` 和 `age` 上分别创建单个索引的话，由于 `mysql` 查询每次只能使用一个索引，所以虽然这样已经相对不做索引时全表扫描提高了很多效率，但是如果在 `area`、`age` 两列上创建复合索引的话将带来更高的效率。如果我们创建了 `(area, age, salary)` 的复合索引，那么其实相当于创建了 `(area, age, salary)`、`(area, age)`、`(area)` 三个索引，这被称为最佳左前缀特性。

因此我们在创建复合索引时应该将最常用作限制条件的列放在最左边，依次递减。

3，索引不会包含有 NULL 值的列

只要列中包含有 NULL 值都将不会被包含在索引中,复合索引中只要有一列含有 NULL 值,那么这一列对于此复合索引就是无效的。所以我们在数据库设计时不要让字段的默认值为 NULL。

4, 使用短索引

对字符串列进行索引,如果可能应该指定一个前缀长度。例如,如果有一个 CHAR(255)的列,如果在前 10 个或 20 个字符内,多数值是惟一的,那么就不要对整个列进行索引。短索引不仅可以提高查询速度而且可以节省磁盘空间和 I/O 操作。

5, 排序的索引问题

mysql 查询只使用一个索引,因此如果 where 子句中已经使用了索引的话,那么 order by 中的列是不会使用索引的。因此数据库默认排序可以符合要求的情况下不要使用排序操作;尽量不要包含多个列的排序,如果需要最好给这些列创建复合索引。

6, like 语句操作

一般情况下不鼓励使用 like 操作,如果非使用不可,如何使用也是一个问题。like "%aaa%" 不会使用索引而 like "aaa%"可以使用索引。

7, 不要在列上进行运算

select * from users where

YEAR(adddate)

8，不使用 NOT IN 操作

NOT IN 操作都不会使用索引将进行全表扫描。NOT IN 可以 NOT EXISTS 代替