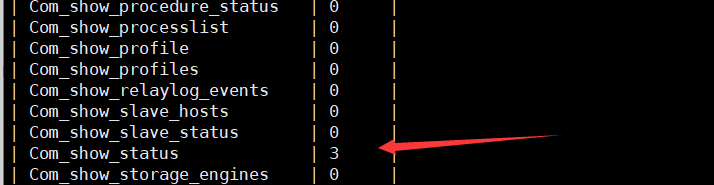
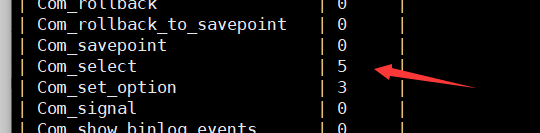
一、MySQL状态查询

mysql > show status; 命令,查看mysql所有运行状态；主要关心Com开头的这些数据

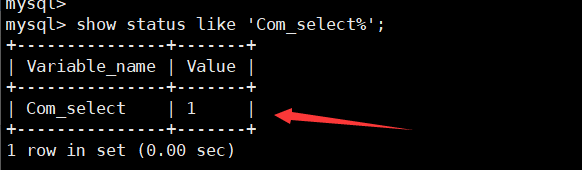
mysql > show status like 'Com%'; <=> show session status like 'Com%'; \\查询当前终端启动到执行命令时的次数



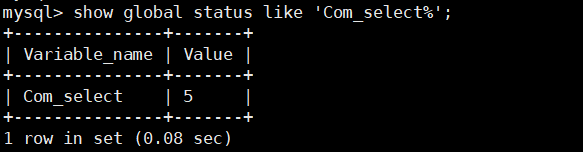
mysql > show global status like 'Com%'; \\查询数据库历史操作次数



mysql > show status like 'Com\_select%'; [\\查询当前终端启动到执行命令时，select命令执行过几次](\\\\查询当前终端启动到执行命令时，select命令执行过几次)

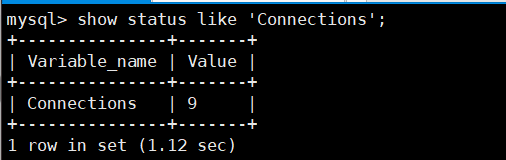


mysql > show global status like 'Com\_select%'; \\查询mysql最近一次启动到执行命令时，一共执行过几次select命令

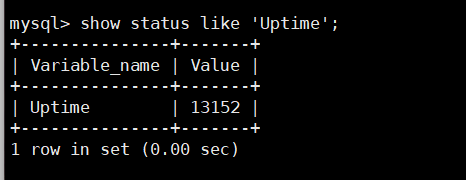


带\*\*\*\*\* 是经常用到的

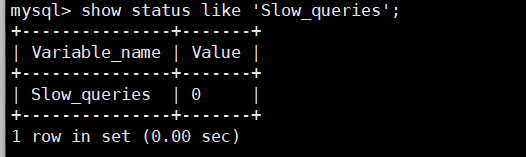
mysql > show status like 'Connections'; \*\*\* \\查看mysql最近一次启动到执行命令时，总共的连接次数



mysql > show status like 'Uptime'; \\查看mysql最近一次的启动时间；单位秒

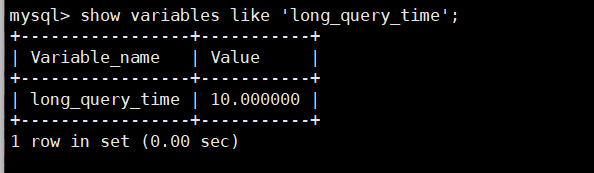


mysql > show status like 'Slow\_queries'; \*\*\*\*\* \\查看mysql最近一次启动到执行命令时，总共的慢查询次数（默认10s）



重点优化对象：慢查询 （默认为10s）

mysql > show variables like 'long\_query\_time'; \\查询当前mysql的慢查询时间设定情况



测试：

创建模拟生产环境中的特大数据表（过程请看大数据插入脚本）

1. 创建库

Create schema a;

1.创建tb\_dept\_bigdata（部门表）。

create table tb\_dept\_bigdata(

id int unsigned primary key auto\_increment,

deptno mediumint unsigned not null default **0**,

dname varchar(**20**) not null default '',

loc varchar(**13**) not null default ''

)engine=innodb default charset=utf8;

2.创建tb\_emp\_bigdata（员工表）。

create table tb\_emp\_bigdata(

id int unsigned primary key auto\_increment,

empno mediumint unsigned not null default **0**,/\*编号\*/

empname varchar(**20**) not null default '',/\*名字\*/

job varchar(**9**) not null default '',/\*工作\*/

mgr mediumint unsigned not null default **0**,/\*上级编号\*/

hiredate date not null,/\*入职时间\*/

sal decimal(**7**,**2**) not null,/\*薪水\*/

comm decimal(**7**,**2**) not null,/\*红利\*/

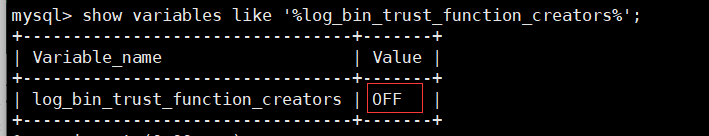
deptno mediumint unsigned not null default **0** /\*部门编号\*/

)engine=innodb default charset=utf8;

#3.开启log\_bin\_trust\_function\_creators参数。

由于在创建函数时，可能会报：This function has none of DETERMINISTIC.....因此我们需开启函数创建的信任功能。

show variables like '%log\_bin\_trust\_function\_creators%';



可通过set global log\_bin\_trust\_function\_creators=1的形式开启该功能，也可通过在my.cnf中永久配置的方式开启该功能，在[mysqld]下配置log\_bin\_trust\_function\_creators=1。

set global log\_bin\_trust\_function\_creators=1;

### **1.创建函数，保证每条数据都不同**

delimiter $$

drop function if exists rand\_string;

create function rand\_string(n int) returns varchar(255)

begin

declare chars\_str varchar(52) default 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';

declare return\_str varchar(255) default '';

declare i int default 0;

while i<n do

set return\_str=concat(return\_str,substring(chars\_str,floor(1+rand()\*52),1));

set i=i+1;

end while;

return return\_str;

end $$

#2.创建随机生成编号的函数。

delimiter $$

drop function if exists rand\_num;

create function rand\_num() returns int(5)

begin

declare i int default 0;

set i=floor(100+rand()\*100);

return i;

end $$

### **2.创建存储过程用于批量插入数据**

#1.创建往tb\_dept\_bigdata表中插入数据的存储过程

delimiter $$

drop procedure if exists insert\_dept;

create procedure insert\_dept(in start int(10),in max\_num int(10))

begin

declare i int default 0;

set autocommit=0;

repeat

set i=i+1;

insert into tb\_dept\_bigdata (deptno,dname,loc) values(rand\_num(),rand\_string(10),rand\_string(8));

until i=max\_num

end repeat;

commit;

end $$

#2.创建往tb\_emp\_bigdata表中插入数据的存储过程。

delimiter $$

drop procedure if exists insert\_emp;

create procedure insert\_emp(in start int(10),in max\_num int(10))

begin

declare i int default 0;

set autocommit=0;

repeat

set i=i+1;

insert into tb\_emp\_bigdata (empno,empname,job,mgr,hiredate,sal,comm,deptno) values((start+i),rand\_string(6),'developer',0001,curdate(),2000,400,rand\_num());

until i=max\_num

end repeat;

commit;

end $$

### **3.具体执行过程批量插入数据**

#1.首先执行随机生成字符串的函数。

#2.然后执行随机生成编号的函数。

#3.查看函数是否创建成功。

delimiter ; 结尾符号恢复默认

show function status\G

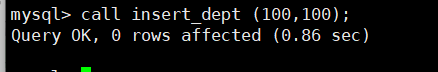
#4.执行插入数据的存储过程，并查看其创建情况。

show procedure status\G

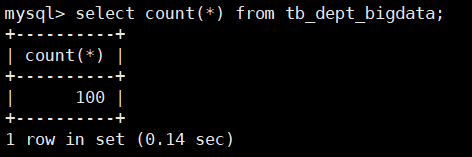
#5.执行存储过程，插入数据。

a.首先执行insert\_dept存储过程。

call insert\_dept (100,100);



select count(\*) from tb\_dept\_bigdata;

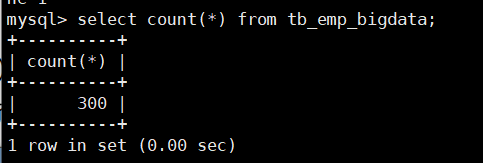


说明：deptno的范围[100,100)，因为deptno的值使用了rand\_num()函数。

b.然后执行insert\_emp存储过程。

call insert\_emp (100,300);

select count(\*) from tb\_emp\_bigdata;



说明：tb\_emp\_bigdata表中deptno编号的范围[100,300)，使用rand\_num()函数。

表示现在tb\_emp\_bigdata表插入了300条数据

注：对于部门表的deptno和员工表中deptno的数据都使用了rand\_num()函数进行赋值，确保两边的值能对应。deptno部门都是100个

#1.删除函数

drop function rand\_num;

drop function rand\_string;

#2.删除存储过程

drop procedure insert\_dept;

drop procedure insert\_emp;

①注意mysql中函数和存储过程的写法。

②注意存储过程的调用，call procedurename。

③注意开启对函数的信任，log\_bin\_trust\_function\_creators参数。

开启慢查询日志与配置慢查询时间

# vim /etc/my.cnf

slow\_query\_log\_file=/var/lib/mysql/slowquery.log 指定慢查询日志路径

slow\_query\_log=on 是否启动慢查询日志

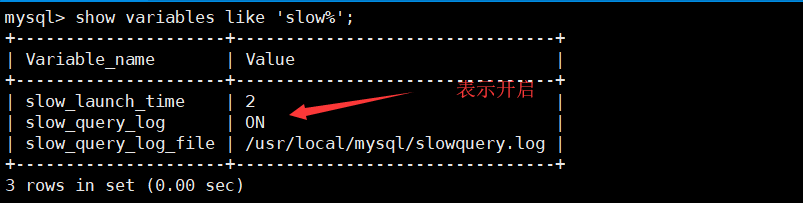
long\_query\_time=1 指定慢查询时间（超过该时间，则为慢查询，单位：s）

备注：慢查询时间可设置为0.00s这种格式，例如：long\_query\_time=0.2 将慢查询时间设置为0.2秒

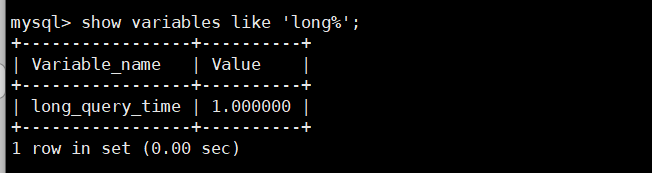
log-queries-not-using-indexes 该参数表示：同时记录下没有使用索引的查询

重启mysqld

mysql > show variables like 'slow%'; 查询当前mysql是否开启慢查询日志



mysql > show variables like 'long%'; 查询当前慢查询设定的时间



单位：s 秒

这里为了测试，我们进行一次慢查询操作

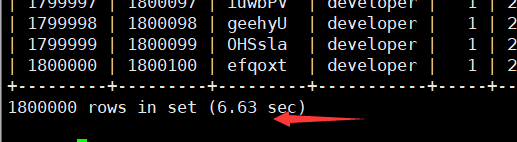
call insert\_emp(100,1800000);

select count(\*) from tb\_emp\_bigdata;

select \* from tb\_emp\_bigdata;

mysql > select \* from test.emp;

select \* from tb\_emp\_bigdata;

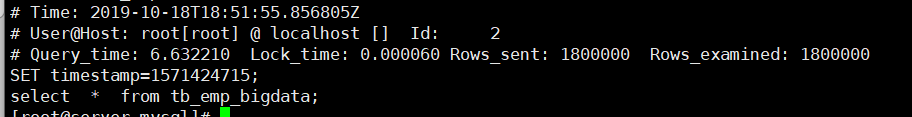
 这里可以看到这次查询共用了6.63s，超出了我们设定的慢查询时间

# cd /var/lib/mysql

# cat slowquery.log

......

SET timestamp=1537887627;



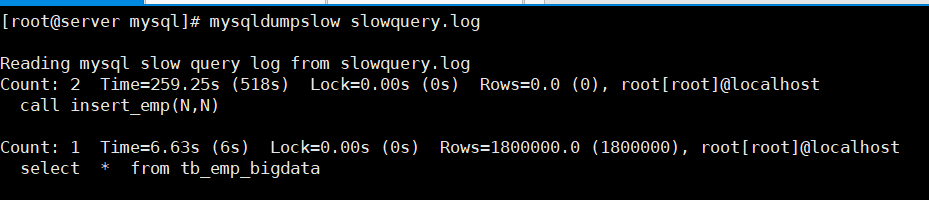
这时就可以看到我们之前操作过的命令已经出现在了慢查询日志里

mysql > select \*from tb\_emp\_bigdata where empno='123456';

这样操作就不会出现在日志中

可以通过mysqldumpslow命令分析

# mysqldumpslow /var/lib/mysql/slowquery.log



mysqldumpslow命令解析

mysqldumpslow -s c -t 10 /var/lib/mysql/slowquery.log

这会输出记录次数最多的10条SQL语句，其中：

-s, 是表示按照何种方式排序，c、t、l、r分别是按照记录次数、时间、查询时间、返回的记录数来排序，ac、at、al、ar，表示相应的倒叙；

-t, 是top n的意思，即为返回前面多少条的数据；

-g, 后边可以写一个正则匹配模式，大小写不敏感的；

例如：

mysqldumpslow -s r -t 10 /var/lib/mysql/slowquery.log

得到返回记录集最多的10个查询

mysqldumpslow -s t -t 10 -g “left join” /var/lib/mysql/slowquery.log

得到按照时间排序的前10条里面含有左连接的查询语句。

使用mysqldumpslow命令可以非常明确的得到各种我们需要的查询语句，对MySQL查询语句的监控、分析、优化是MySQL优化非常重要的一步。开启慢查询日志后，由于日志记录操作，在一定程度上会占用CPU资源影响mysql的性能，但是可以阶段性开启来定位性能瓶颈。

explain分析查询

使用 EXPLAIN 关键字可以模拟优化器执行SQL查询语句，从而知道MySQL是如何处理你的SQL语句的。这可以帮你分析你的查询语句或是表结构的性能瓶颈。通过explain命令可以得到:

1.表的读取顺序

2.数据读取操作的操作类型

3.哪些索引可以使用

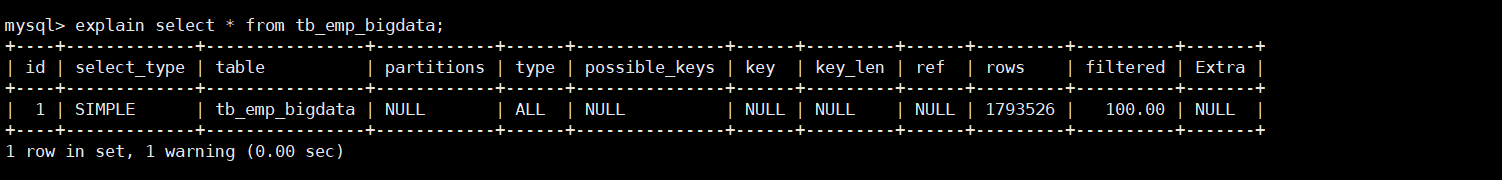
4.哪些索引被实际使用

5.表之间的引用

6.每张表有多少行被优化器查询

mysql> explain select \* from dept;

explain select \* from tb\_emp\_bigdata;



explain查询字段解析：

Table：显示这一行的数据是来自哪张表

possible\_keys：显示可应用在这张表中的索引。如果为空，没有可用的索引。

key：实际使用的索引。如果为NULL，即没有使用索引。

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

rows：MySQL认为必须检索的用来返回请求数据的行数

------------------------------------------------------------------------------------

type：这是最重要的字段之一，显示查询使用了何种类型的索引。从最好到最差的索引类型为system、const、eq\_reg、ref、range、index和ALL

system、const：可以将查询的变量转为常量.如id=1; id为 主键或唯一键.

eq\_ref：访问索引,返回某单一行的数据.(通常在联接时出现，查询使用的索引为主键或唯一键)

ref：访问索引,返回某个值的数据.(可以返回多行) 通常使用=时发生

range：这个连接类型使用索引返回一个范围中的行，比如使用>或<查找东西，并且该字段上建有索引时发生的情况(注:不一定好于index)

index：以索引的顺序进行全表扫描，优点是不用排序,缺点是还要全表扫描

ALL：全表扫描，应该尽量避免

------------------------------------------------------------------------------------

Extra：关于MYSQL如何解析查询的额外信息，主要有以下几种

using index：只用到索引,可以避免访问表.

using where：使用到where来过滤数据. 不是所有的where都会显示using where. 如以=方式访问索引.

using tmporary：用到临时表

using filesort：用到额外的排序. (当使用order by v1,而没用到索引时,就会使用额外的排序)

range checked for eache record(index map:N)：没有好的索引.

------------------------------------------------------------------------------------

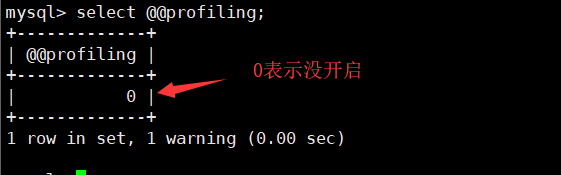
profiling分析查询

通过慢日志查询可以知道哪些SQL语句执行效率低下，通过explain我们可以得知SQL语句的具体执行情况，索引使用等，还可以结合show命令查看执行状态。

如果觉得explain的信息不够详细，可以通过profiling命令得到更准确的SQL执行消耗系统资源的信息。

profiling默认是关闭的。可以通过以下语句查看

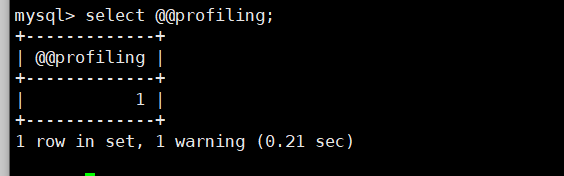
mysql > select @@profiling;



打开功能

mysql > set profiling=1;

mysql> select @@profiling;



打开后就可以开始对接下来的操作语句进行测试：

mysql > select \* from dept;

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

| deptno | dname | loc |

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

| 101 | DTJAINTdNB | uvScKrEV |

| 102 | PnWUJMJTDh | cNLppEfL |

| 103 | nicNHVJtbc | hDvrzuzp |

| 104 | YZdsERyOCu | ooBOpjay |

| 105 | uzpZewVqOZ | FbqAJEbw |

| 106 | ddgtBBbDLV | uneJjRHM |

| 107 | NEICNgqotd | kQzaBJOU |

| 108 | fQQIPaHjbE | RxMsAzYs |

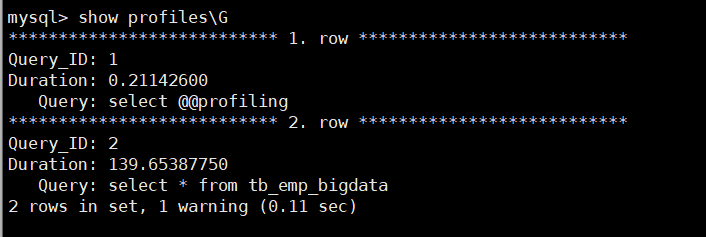
| 109 | UtjPwOJOaL | ENbRgfgs |

| 110 | tOQOtEOgqj | aystNIfy |

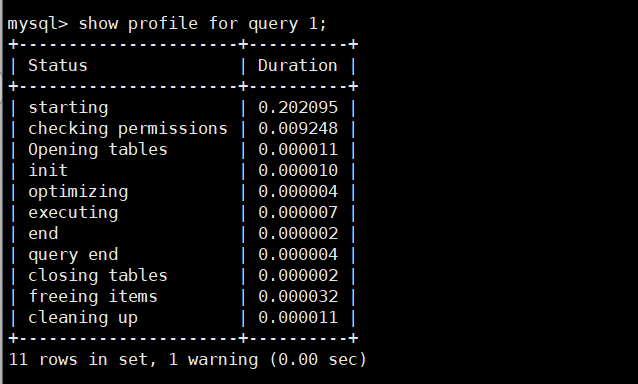
+--------+------------+----------+

查看简要信息：

mysql > show profiles\G

查看详细执行过程信息及每个步骤所耗费时间：

mysql> show profile for query 4;



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

| Status | Duration |

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

| starting | 0.000057 | 启动

| checking permissions | 0.000058 | 检查权限

| Opening tables | 0.000064 | 打开表

| init | 0.000020 | 初始化

| System lock | 0.000009 | 系统锁定

| optimizing | 0.000003 | 优化

| statistics | 0.000011 | 统计

| preparing | 0.000009 | 准备

| executing | 0.000002 | 执行

| Sending data | 2.433287 | 发送数据

| end | 0.000014 | 结束

| query end | 0.000010 | 查询结束

| closing tables | 0.000010 | 关闭表

| freeing items | 0.004363 | 释放项目内存

| logging slow query | 0.003992 | 写入慢查询日志

| cleaning up | 0.000022 | 清理项目

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

16 rows in set, 1 warning (0.00 sec)

----------------------------------------------------------------------------

二、MySQL 性能优化之索引的创建与删除 \*\*\*\*\*

介绍见world文档！

数据库优化概述：

1、数据库或表设计是否合理，一般在设计时需遵循三个范例示范（有时适当的使用逆范式效果会更好）

1.1 要求数据拥有原子性；指事务包含的所有操作要么全部成功，要么全部失败回滚（只要是关系型数据库，都符合）

1.2 满足1.1的前提下，数据需要保证拥有唯一性（其中某一列的内容不会重复，设置主键）

1.3 满足1.2的前提下，表之间数据可以通过关联查询（多表查询）

2、SQL语句的优化（通过时间的积累，会找到一些小窍门）

2.1 SQL语句种类

DQL 数据查询语言 select --查询数据

DDL 数据定义语言 create drop alter truncate rename --对表操作的命令

DML 数据操纵语言 insert update delete --对表内容操作的命令

DCL 数据控制语言 grant revoke --授权

select SQL语句优化重点

3、数据库的配置（配置文件的合理配置）

4、服务器硬件配置及操作系统类型

对数据库性能影响范围从上而下

索引的类型介绍

普通索引（INDEX）：这是最基本的索引类型，没唯一性之类的限制。

唯一性索引（UNIQUE INDEX）：和普通索引基本相同，但所有的索引列值保持唯一性。

主键（PRIMARY KEY）：主键是一种唯一索引，但必须指定为”PRIMARY KEY”。

全文索引（FULLTEXT）：在MYSQL中，全文索引的索引类型为FULLTEXT。全文索引可以在VARCHAR或者TEXT类型的列上创建。

单列索引和多列索引（复合索引）

索引可以是单列索引，也可以是多列索引。对相关的列使用索引是提高SELECT操作性能的最佳途径之一。

多列索引：

MySQL可以为多个列创建索引。一个索引可以包括15个列。对于某些列类型，可以索引列的左前缀，列的顺序非常重要。

多列索引可以视为包含通过连接索引列的值而创建的值的排序的数组。一般来说，即使是限制最严格的单列索引，它的限制能力也远远低于多列索引。

最左前缀

多列索引有一个特点，即最左前缀（Leftmost Prefixing）。假如有一个多列索引为key(firstname lastname age)，当搜索条件是以下各种列的组合和顺序时，MySQL将使用该多列索引：

firstname，lastname，age

firstname，lastname

firstname

也就是说，相当于还建立了key(firstname lastname)和key(firstname)。

多列索引原理解释：

a='id'

b='name'

c='sex'

多列索引：dl=a,b,c;dl=a,b;dl=c

索引主要用于下面的操作：

快速找出匹配一个WHERE子句的行。

删除行。当执行联接时，从其它表检索行。

有时MySQL不使用索引，即使有可用的索引。一种情形是当优化器估计到使用索引将需要MySQL访问表中的大部分行时。(在这种情况下，表扫描可能会更快些）。然而，如果此类查询使用LIMIT只搜索部分行，MySQL则使用索引，因为它可以更快地找到几行并在结果中返回。

索引的创建：

create命令创建

mysql> create index index\_name on table\_name (list\_name); 创建index索引

mysql> create unique index index\_name on table\_name (list\_name); 创建unique\_index索引

alter命令创建

mysql> alter table table\_name add index index\_name (list\_name); 创建index索引

mysql> alter table table\_name add unique (list\_name); 创建unique\_index索引

mysql> alter table table\_name add primary key (list\_name); 创建primary key索引

table\_name 要添加索引的表名称

list\_name 指定哪些列需要做索引，指定多列时使用逗号分隔（id,name,sex）

index\_name 指定索引名称，非必选，不指定则以指定列的第一列作为索引名

注意：create 命令无法创建primary key索引，如果需要使用到该索引，请使用alter命令创建

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

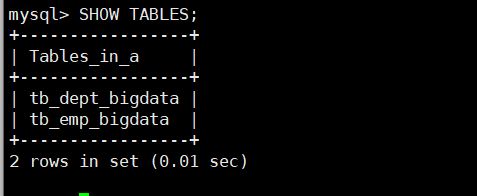
创建索引实例：

INDEX索引创建

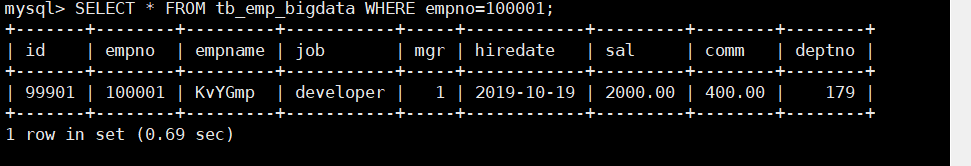
清除数据，然后插入180万数据

mysql> TRUNCATE TABLE tb\_emp\_bigdata;

mysql> CALL INSERT\_EMP (100,1800000);

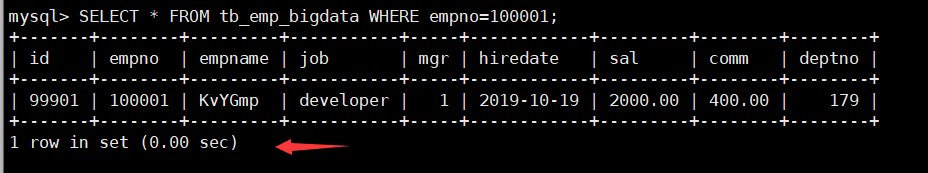


mysql> SELECT \* FROM tb\_emp\_bigdata WHERE empno=100001;

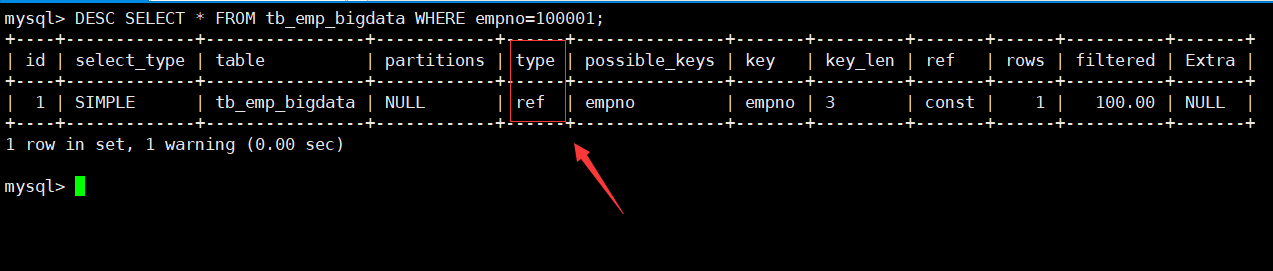
 未创建索引前，查询时间为0.69s

mysql> CREATE INDEX empno ON tb\_emp\_bigdata(empno ); 为tb\_emp\_bigdata表中empno字段创建一个名为empno的index索引(索引名称可不写)

mysql> SELECT \* FROM tb\_emp\_bigdata WHERE empno=100001;

 再次查询刚才的数据，时间明显降低

mysql> DESC SELECT \* FROM tb\_emp\_bigdata WHERE empno=100001;

当前的查询type为ref

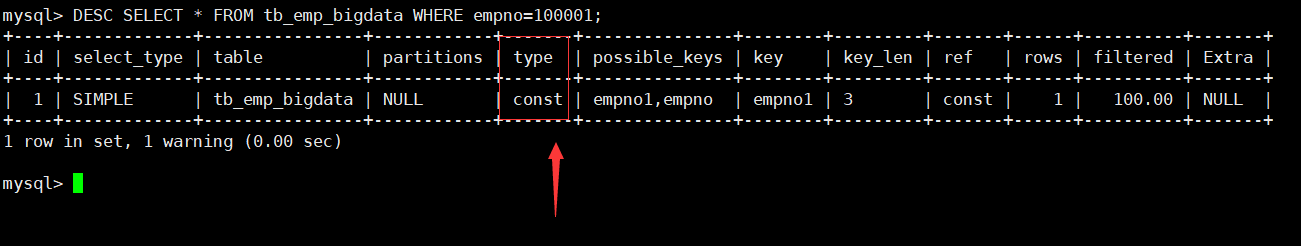
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

UNIQUE INDEX索引创建

mysql> CREATE UNIQUE INDEX empno1 ON tb\_emp\_bigdata(empno); 为tb\_emp\_bigdata表中empno字段创建一个名为empno1的unique index索引

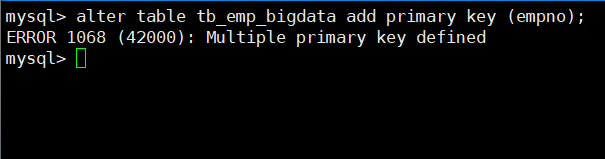
看一下当前查询使用的type是什么

mysql> DESC SELECT \* FROM tb\_emp\_bigdata WHERE empno=100001;

 当前查询的type为const

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

PRIMARY KEY索引创建 现在好像创建不了了；



mysql> alter table emp add primary key (empno); 因为create无法创建该类型的所以，所以只能用alter

mysql> select \* from emp where empno=1000000;

......

1 row in set (0.00 sec)

mysql> desc select \* from emp where empno=1000000;

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

| id | select\_type | table | partitions | type | possible\_keys | key | key\_len | ref | rows | filtered | Extra |

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

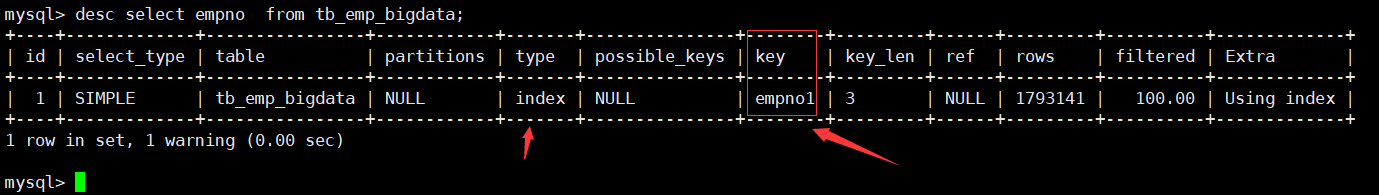
| 1 | SIMPLE | emp | NULL | const | PRIMARY | PRIMARY | 3 | const | 1 | 100.00 | NULL |

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

1 row in set, 1 warning (0.00 sec)

查看索引名称和类型

mysql> desc select empno from tb\_emp\_bigdata;



删除索引：

备注：删除INDEX或UNIQUE INDEX索引时必须加索引名称

mysql> drop index index\_name on table\_name;

mysql> alter table table\_name drop index index\_name;

mysql> alter table tables\_name drop primary key

实例：

mysql> DROP INDEX empno1 on tb\_emp\_bigdata; 删除INDEX或UNIQUE INDEX索引

Query OK, 1800000 rows affected (1.77 sec)

Records: 1800000 Duplicates: 0 Warnings: 0

DROP INDEX empno on tb\_emp\_bigdata;

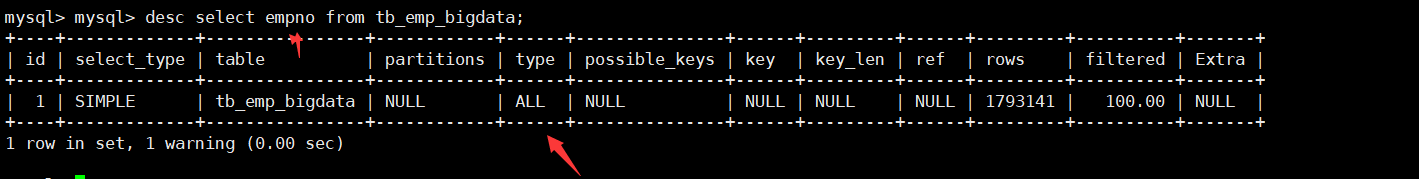
mysql> alter table emp drop index empno1; 删除INDEX或UNIQUE INDEX索引

mysql> alter table emp drop primary key; 删除PRIMARY KEY索引

Query OK, 1800000 rows affected (1.55 sec)

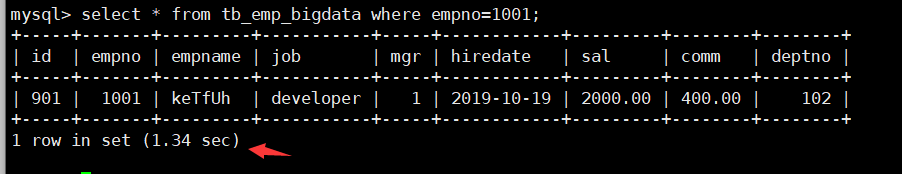
Records: 1800000 Duplicates: 0 Warnings: 0

现在索引类型为ALL了



mysql> select \* from tb\_emp\_bigdata where empno=1001;

删除索引后查询时间变长了；



--------------------------------------------------------------------------------------------------------------------