第 3 章 查询截取分析

第 3 章 查询截取分析

1、查询优化

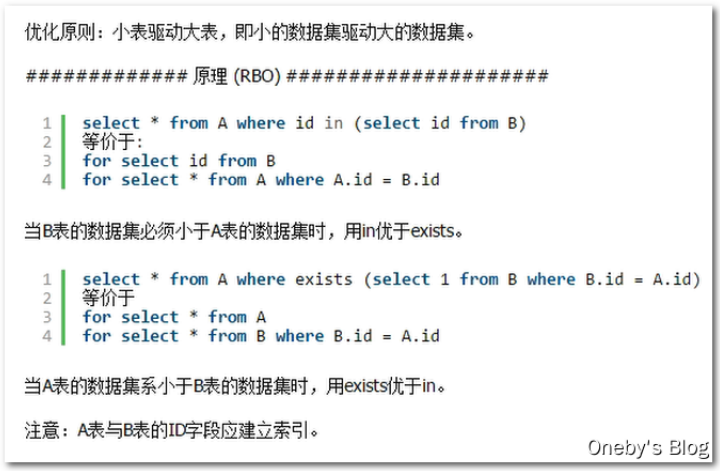
1.1、MySQL 优化原则

> **mysql 的调优大纲**

1. 慢查询的开启并捕获
2. explain+慢SQL分析
3. show profile查询SQL在Mysql服务器里面的执行细节和生命周期情况
4. SQL数据库服务器的参数调优

> **永远小表驱动大表，类似嵌套循环 Nested Loop**

1. EXISTS 语法：
    1. `SELECT ... FROM table WHERE EXISTS(subquery)`
    2. 该语法可以理解为：将查询的数据，放到子查询中做条件验证，根据验证结果（TRUE或FALSE）来决定主查询的数据结果是否得以保留。
2. EXISTS(subquery) 只返回TRUE或FALSE，因此子查询中的 `SELECT *` 也可以是 `SELECT 1` 或其他，官方说法是实际执行时会忽略SELECT清单，因此没有区别
3. EXISTS子查询的实际执行过程可能经过了优化而不是我们理解上的逐条对比，如果担忧效率问题，可进行实际检验以确定是否有效率问题。
4. EXISTS子查询往往也可以用条件表达式、其他子查询或者JOIN来替代，何种最优需要具体问题具体分析



结论：

1. 永远记住小表驱动大表
2. 当 B 表数据集小于 A 表数据集时，使用 in
3. 当 A 表数据集小于 B 表数据集时，使用 exist

---

in 和 exists 的用法

- tbl_emp 表和 tbl_dept 表

```
mysql> select * from tbl_emp;
+----+------+--------+
| id | NAME | deptId |
+----+------+--------+
|  1 | z3   |      1 |
|  2 | z4   |      1 |
|  3 | z5   |      1 |
|  4 | w5   |      2 |
|  5 | w6   |      2 |
|  6 | s7   |      3 |
|  7 | s8   |      4 |
|  8 | s9   |     51 |
+----+------+--------+
8 rows in set (0.00 sec)

```

```
mysql> select * from tbl_dept;
+----+----------+--------+
| id | deptName | locAdd |
+----+----------+--------+
|  1 | RD       | 11     |
|  2 | HR       | 12     |
|  3 | MK       | 13     |
|  4 | MIS      | 14     |
|  5 | FD       | 15     |
+----+----------+--------+
5 rows in set (0.00 sec)
```

- in 的写法

```
mysql> select * from tbl_emp e where e.deptId in (select id from tbl_dept);
+----+------+--------+
| id | NAME | deptId |
+----+------+--------+
|  1 | z3   |      1 |
|  2 | z4   |      1 |
|  3 | z5   |      1 |
|  4 | w5   |      2 |
|  5 | w6   |      2 |
|  6 | s7   |      3 |
|  7 | s8   |      4 |
+----+------+--------+
7 rows in set (0.00 sec)
```

- exists 的写法

```
mysql> select * from tbl_emp e where exists (select 1 from tbl_dept d where e.deptId = d.id);
+----+------+--------+
| id | NAME | deptId |
+----+------+--------+
|  1 | z3   |      1 |
|  2 | z4   |      1 |
|  3 | z5   |      1 |
|  4 | w5   |      2 |
|  5 | w6   |      2 |
|  6 | s7   |      3 |
|  7 | s8   |      4 |
+----+------+--------+
7 rows in set (0.00 sec)
```

1.2、ORDER BY 优化

> **ORDER BY子句，尽量使用Index方式排序，避免使用FileSort方式排序**

创建表

- 建表 SQL

```
create table tblA(
    #id int primary key not null auto_increment,
    age int,
    birth timestamp not null
);

insert into tblA(age, birth) values(22, now());
insert into tblA(age, birth) values(23, now());
insert into tblA(age, birth) values(24, now());

create index idx_A_ageBirth on tblA(age, birth);
```

- tblA 表中的测试数据

```
mysql> select * from tblA;
+------+---------------------+
| age  | birth               |
+------+---------------------+
|   22 | 2020-08-05 10:36:32 |
|   23 | 2020-08-05 10:36:32 |
|   24 | 2020-08-05 10:36:32 |
+------+---------------------+
3 rows in set (0.00 sec)
```

- tbl 中的索引

```
mysql> SHOW INDEX FROM tblA;
+-------+------------+---------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+-----
| Table | Non_unique | Key_name      | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Inde
+-------+------------+---------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+-----
| tblA  |          1 | idx_A_ageBirth |            1 | age         | A         |           3 |     NULL | NULL   | YES  | BTREE      |         |
```

```
| tblA  |          1 | idx_A_ageBirth |           2 | birth       | A         |           3 |     NULL | NULL   |      | BTREE      |         |
+-------+------------+----------------+-------------+-------------+-----------+-------------+----------+--------+------+------------+---------+-----
2 rows in set (0.00 sec)
```

CASE1：能使用索引进行排序的情况

- 只有带头大哥 age

```
mysql> EXPLAIN SELECT * FROM tblA where age>20 order by age;
+----+-------------+-------+-------+----------------+----------------+---------+------+------+--------------------------+
| id | select_type | table | type  | possible_keys  | key            | key_len | ref  | rows | Extra                    |
+----+-------------+-------+-------+----------------+----------------+---------+------+------+--------------------------+
|  1 | SIMPLE      | tblA  | index | idx_A_ageBirth | idx_A_ageBirth | 9       | NULL |    3 | Using where; Using index |
+----+-------------+-------+-------+----------------+----------------+---------+------+------+--------------------------+
1 row in set (0.01 sec)

mysql> EXPLAIN SELECT * FROM tblA where birth>'2016-01-28 00:00:00' order by age;
+----+-------------+-------+-------+---------------+----------------+---------+------+------+--------------------------+
| id | select_type | table | type  | possible_keys | key            | key_len | ref  | rows | Extra                    |
+----+-------------+-------+-------+---------------+----------------+---------+------+------+--------------------------+
|  1 | SIMPLE      | tblA  | index | NULL          | idx_A_ageBirth | 9       | NULL |    3 | Using where; Using index |
+----+-------------+-------+-------+---------------+----------------+---------+------+------+--------------------------+
1 row in set (0.00 sec)
```

- 带头大哥 age + 小弟 birth

```
mysql> EXPLAIN SELECT * FROM tblA where age>20 order by age,birth;
+----+-------------+-------+-------+----------------+----------------+---------+------+------+--------------------------+
| id | select_type | table | type  | possible_keys  | key            | key_len | ref  | rows | Extra                    |
+----+-------------+-------+-------+----------------+----------------+---------+------+------+--------------------------+
|  1 | SIMPLE      | tblA  | index | idx_A_ageBirth | idx_A_ageBirth | 9       | NULL |    3 | Using where; Using index |
+----+-------------+-------+-------+----------------+----------------+---------+------+------+--------------------------+
1 row in set (0.00 sec)
```

- mysql 默认升序排列，全升序或者全降序，都扛得住

```
mysql> EXPLAIN SELECT * FROM tblA ORDER BY age ASC, birth ASC;
+----+-------------+-------+-------+---------------+----------------+---------+------+------+-------------+
| id | select_type | table | type  | possible_keys | key            | key_len | ref  | rows | Extra       |
+----+-------------+-------+-------+---------------+----------------+---------+------+------+-------------+
|  1 | SIMPLE      | tblA  | index | NULL          | idx_A_ageBirth | 9       | NULL |    3 | Using index |
+----+-------------+-------+-------+---------------+----------------+---------+------+------+-------------+
1 row in set (0.00 sec)

mysql> EXPLAIN SELECT * FROM tblA ORDER BY age DESC, birth DESC;
+----+-------------+-------+-------+---------------+----------------+---------+------+------+-------------+
| id | select_type | table | type  | possible_keys | key            | key_len | ref  | rows | Extra       |
+----+-------------+-------+-------+---------------+----------------+---------+------+------+-------------+
|  1 | SIMPLE      | tblA  | index | NULL          | idx_A_ageBirth | 9       | NULL |    3 | Using index |
+----+-------------+-------+-------+---------------+----------------+---------+------+------+-------------+
1 row in set (0.01 sec)
```

CASE2：不能使用索引进行排序的情况

- 带头大哥 age 挂了

```
mysql> EXPLAIN SELECT * FROM tblA where age>20 order by birth;
+----+-------------+-------+-------+----------------+----------------+---------+------+------+----------------------------------------------+
| id | select_type | table | type  | possible_keys  | key            | key_len | ref  | rows | Extra                                        |
+----+-------------+-------+-------+----------------+----------------+---------+------+------+----------------------------------------------+
|  1 | SIMPLE      | tblA  | index | idx_A_ageBirth | idx_A_ageBirth | 9       | NULL |    3 | Using where; Using index; Using filesort |
+----+-------------+-------+-------+----------------+----------------+---------+------+------+----------------------------------------------+
1 row in set (0.01 sec)
```

- 小弟 birth 居然敢在带头大哥 age 前面

```
mysql> EXPLAIN SELECT * FROM tblA where age>20 order by birth,age;
+----+-------------+-------+-------+----------------+----------------+---------+------+------+----------------------------------------------+
| id | select_type | table | type  | possible_keys  | key            | key_len | ref  | rows | Extra                                        |
+----+-------------+-------+-------+----------------+----------------+---------+------+------+----------------------------------------------+
|  1 | SIMPLE      | tblA  | index | idx_A_ageBirth | idx_A_ageBirth | 9       | NULL |    3 | Using where; Using index; Using filesort |
+----+-------------+-------+-------+----------------+----------------+---------+------+------+----------------------------------------------+
1 row in set (0.00 sec)
```

- mysql 默认升序排列，如果全升序或者全降序，都 ok ， 但是一升一降 mysql 就扛不住了

```
mysql> EXPLAIN SELECT * FROM tblA ORDER BY age ASC, birth DESC;
+----+-------------+-------+-------+---------------+----------------+---------+------+------+-----------------------------+
| id | select_type | table | type  | possible_keys | key            | key_len | ref  | rows | Extra                       |
+----+-------------+-------+-------+---------------+----------------+---------+------+------+-----------------------------+
|  1 | SIMPLE      | tblA  | index | NULL          | idx_A_ageBirth | 9       | NULL |    3 | Using index; Using filesort |
```

```
+----+-------------+-------+-------+---------------+---------------+---------+------+------+--------------------------+
1 row in set (0.00 sec)
```

结论

1. MySQL支持二种方式的排序，FileSort和Index，**Index效率高，它指MySQL扫描索引本身完成排序**，FileSort方式效率较低。
2. ORDER BY满足两情况（最佳左前缀原则），会使用Index方式排序
    1. ORDER BY语句使用索引最左前列
    2. 使用where子句与OrderBy子句条件列组合满足索引最左前列
3. 尽可能在索引列上完成排序操作，遵照索引建的**最佳左前缀**

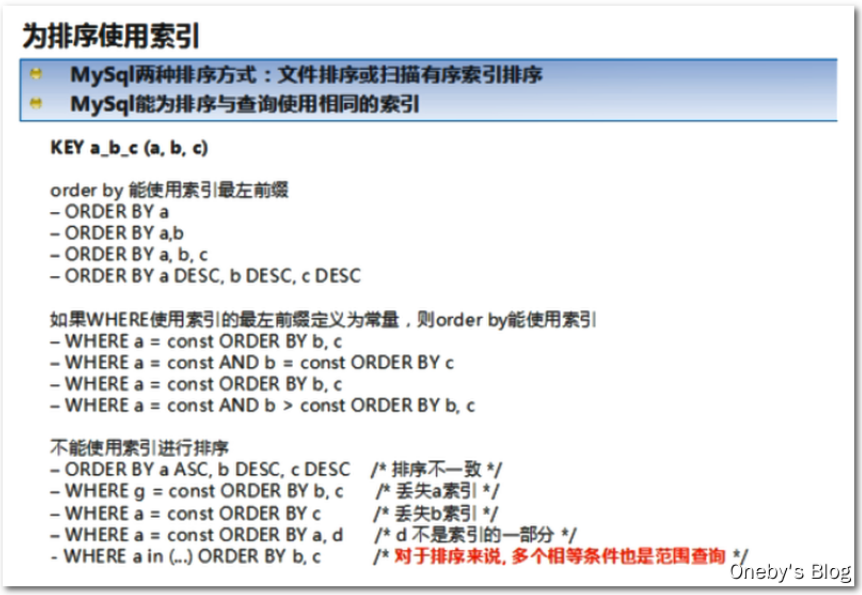
> **如果未在索引列上完成排序，mysql 会启动 filesort 的两种算法：双路排序和单路排序**

1. 双路排序
    1. MySQL4.1之前是使用双路排序，字面意思是两次扫描磁盘，最终得到数据。读取行指针和将要进行orderby操作的列，对他们进行排序，然后扫描已经排序好的列表，按照列表列表中读取对应的数据传输
    2. 从磁盘取排序字段，在buffer进行排序，再从磁盘取其他字段。
2. 单路排序
    1. 取一批数据，要对磁盘进行两次扫描，众所周知，I/O是很耗时的，所以在mysql4.1之后，出现了改进的算法，就是单路排序。
    2. 从磁盘读取查询需要的所有列，按照将要进行orderby的列，在sort buffer对它们进行排序，然后扫描排序后的列表进行输出，它的效率更快一些，避免了第二次读取数据，并且顺序IO，但是它会使用更多的空间，因为它把每一行都保存在内存中了。
3. 结论及引申出的问题：
    1. 由于单路是改进的算法，总体而言好过双路
    2. 在sort_buffer中，方法B比方法A要多占用很多空间，因为方法B是把所有字段都取出，所以有可能取出的数据的总大小超出了sort_buffer的容量，导致每次只能取sort_buffer容量进行排序（创建tmp文件，多路合并），排完再取取sort_buffer容量大小，再排…… 从而会导致多次I/O。
    3. 结论：本来想省一次I/O操作，反而导致了大量的/O操作，反而得不偿失。
4. 更深层次的优化策略：
    1. 增大sort_buffer_size参数的设置
    2. 增大max_length_for_sort_data参数的设置

遵循如下规则，可提高Order By的速度

1. Order by时select *是一个大忌，只Query需要的字段，这点非常重要。在这里的影响是：
    1. 当Query的字段大小总和小于max_length_for_sort_data，而且排序字段不是TEXT|BLOB类型时，会用改进后的算法——单路排序，否则用老算法——多路排序。
    2. 两种算法的数据都有可能超出sort_buffer的容量，超出之后，会创建tmp文件进行合并排序，导致多次I/O，但是用单路排序算法的风险会更大一些，所以要提高sort_buffer_size
2. 尝试提高 sort_buffer_size不管用哪种算法，提高这个参数都会提高效率，当然，要根据系统的能力去提高，因为这个参数是针对每个进程的
3. 尝试提高max_length_for_sort_data提高这个参数，会增加用改进算法的概率。但是如果设的太高，数据总容量超出sort_buffer_size的概率就增大，明显症状是高的磁盘I/O活动和低的率。

> **Order By 排序索引优化的总结**



1.3、GROUP BY 优化

> group by关键字优化

1. group by实质是**先排序后进行分组，遵照索引的最佳左前缀**
2. 当无法使用索引列，增大max_length_for_sort_data参数的设置+增大sort_buffer_size参数的设置
3. where高于having，能写在where限定的条件就不要去having限定了
4. 其余的规则均和 order by 一致

2、慢查询日志

2.1、慢查询日志介绍

> 慢查询日志是什么?

1. MySQL的慢查询日志是MySQL提供的一种日志记录，它用来**记录在MySQL中响应时间超过阈值的语句**，具体指运行时间超过**long_query_time**值的SQL，则会被记录到慢查询日志中
2. long_query_time的默认值为10，意思是运行10秒以上的SQL语句会被记录下来
3. 由他来查看哪些SQL超出了我们的最大忍耐时间值，比如一条sql执行超过5秒钟，我们就算慢SQL，希望能收集超过5秒的sql，结合之前explain进行全面分析。

2.2、慢查询日志开启

> 怎么玩?

说明:

1. 默认情况下，MySQL数据库没有开启慢查询日志，需要我们手动来设置这个参数。
2. 当然，如果不是调优需要的话，一般不建议启动该参数，因为开启慢查询日志会或多或少带来一定的性能影响。慢查询日志支持将日志记录写入文件

查看是否开启及如何开启

- 查看慢查询日志是否开启:
  - 默认情况下slow_query_log的值为OFF，表示慢查询日志是禁用的
  - 可以通过设置**slow_query_log**的值来开启
  - 通过 `SHOW VARIABLES LIKE '%slow_query_log%';` 查看 mysql 的慢查询日志是否开启

```
mysql> SHOW VARIABLES LIKE '%slow_query_log%';
+---------------------+-------------------------------+
| Variable_name       | Value                         |
+---------------------+-------------------------------+
| slow_query_log      | OFF                           |
| slow_query_log_file | /var/lib/mysql/Heygo-slow.log |
+---------------------+-------------------------------+
2 rows in set (0.00 sec)
```

- 如何开启开启慢查询日志:
  - `set global slow_query_log = 1;` 开启慢查询日志
  - 使用 `set global slow_query_log=1` 开启了慢查询日志只对当前数据库生效，如果MySQL重启后则会失效。

```
mysql> set global slow_query_log = 1;
Query OK, 0 rows affected (0.07 sec)

mysql> SHOW VARIABLES LIKE '%slow_query_log%';
+---------------------+-------------------------------+
| Variable_name       | Value                         |
+---------------------+-------------------------------+
| slow_query_log      | ON                            |
| slow_query_log_file | /var/lib/mysql/Heygo-slow.log |
+---------------------+-------------------------------+
2 rows in set (0.00 sec)
```

- 如果要永久生效，就必须修改配置文件my.cnf（其它系统变量也是如此）
  - 修改my.cnf文件，[mysqld]下增加或修改参数：slow_query_log和slow_query_log_file后，然后重启MySQL服务器。
  - 也即将如下两行配置进my.cnf文件

```
[mysqld]
slow_query_log =1
slow_query_log_file=/var/lib/mysql/Heygo-slow.log
```

  - 关于慢查询的参数slow_query_log_file，它指定慢查询日志文件的存放路径，系统默认会给一个缺省的文件host_name-slow.log（如果没有指定参数slow_query_log_file的话）

那么开启慢查询日志后，什么样的SQL参会记录到慢查询里面?

- 这个是由参数long_query_time控制，默认情况下long_query_time的值为10秒，命令：`SHOW VARIABLES LIKE 'long_query_time%';` 查看慢 SQL 的阈值

```
mysql> SHOW VARIABLES LIKE 'long_query_time%';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+
1 row in set (0.01 sec)
```

- 可以使用命令修改，也可以在my.cnf参数里面修改。
- 假如运行时间正好等于long_query_time的情况，并不会被记录下来。也就是说，在mysql源码里是判断大于long_query_time，而非大于等于。

2.3、慢查询日志示例

> **案例讲解**

- 查看慢 SQL 的阈值时间，默认阈值时间为 10s

```
mysql> SHOW VARIABLES LIKE 'long_query_time%';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+
1 row in set (0.00 sec)
```

- 设置慢 SQL 的阈值时间，我们将其设置为 3s

```
mysql> set global long_query_time=3;
Query OK, 0 rows affected (0.00 sec)
```

- 为什么设置后阈值时间没变?
  - 需要重新连接或者新开一个回话才能看到修改值。
  - 查看全局的 long_query_time 值：`show global variables like 'long_query_time';` 发现已经生效

```
mysql> set global long_query_time=3;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'long_query_time%';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+
1 row in set (0.00 sec)

mysql> show global variables like 'long_query_time';
+-----------------+----------+
| Variable_name   | Value    |
+-----------------+----------+
| long_query_time | 3.000000 |
+-----------------+----------+
1 row in set (0.00 sec)
```

- 记录慢 SQL 以供后续分析
  - 怼个 select sleep(4); 超过 3s ， 肯定会被记录到日志中

```
mysql> select sleep(4);
+----------+
| sleep(4) |
+----------+
|        0 |
+----------+
1 row in set (4.00 sec)

```

  - 慢查询日志文件在 /var/lib/mysql/ 下，后缀为 -slow.log

```
[root@Heygo mysql]# cd /var/lib/mysql/
[root@Heygo mysql]# ls -l
总用量 176156
-rw-rw----. 1 mysql mysql       56 8月   3 19:08 auto.cnf
drwx------. 2 mysql mysql     4096 8月   5 10:36 db01
-rw-rw----. 1 mysql mysql     7289 8月   3 22:38 Heygo.err
-rw-rw----. 1 mysql mysql      371 8月   5 12:58 Heygo-slow.log
-rw-rw----. 1 mysql mysql 79691776 8月   5 10:36 ibdata1
-rw-rw----. 1 mysql mysql 50331648 8月   5 10:36 ib_logfile0
-rw-rw----. 1 mysql mysql 50331648 8月   3 19:08 ib_logfile1
drwx------. 2 mysql mysql     4096 8月   3 19:08 mysql
srwxrwxrwx. 1 mysql mysql        0 8月   3 22:38 mysql.sock
```

```
drwx------. 2 mysql mysql     4096 8月   3 19:08 performance_schema

```

- 查看慢查询日志中的内容

```
[root@Heygo mysql]# cat Heygo-slow.log
/usr/sbin/mysqld, Version: 5.6.49 (MySQL Community Server (GPL)). started with:
Tcp port: 3306  Unix socket: /var/lib/mysql/mysql.sock
Time                 Id Command    Argument
# Time: 200805 12:58:01
# User@Host: root[root] @ localhost []  Id:    11
# Query_time: 4.000424  Lock_time: 0.000000 Rows_sent: 1  Rows_examined: 0
SET timestamp=1596603481;
select sleep(4);
```

- 查询当前系统中有多少条慢查询记录： `show global status like '%Slow_queries%';`

```
mysql> show global status like '%Slow_queries%';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Slow_queries  | 1     |
+---------------+-------+
1 row in set (0.00 sec)
```

配置版的慢查询日志

在 /etc/my.cnf 文件的 [mysqld] 节点下配置

```
slow_query_log=1;
slow_query_log_file=/var/lib/mysql/Heygo-slow.log
long_query_time=3;
log_output=FILE
```

> **日志分析命令 mysqldumpslow**

mysqldumpslow是什么?

在生产环境中，如果要手工分析日志，查找、分析SQL，显然是个体力活，MySQL提供了日志分析工具mysqldumpslow。

查看 mysqldumpslow的帮助信息

```
[root@Heygo mysql]# mysqldumpslow --help
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
```

---

mysqldumpshow 参数解释

1. s：是表示按何种方式排序
2. c：访问次数
3. l：锁定时间
4. r：返回记录
5. t：查询时间
6. al：平均锁定时间

7. ar：平均返回记录数
8. at：平均查询时间
9. t：即为返回前面多少条的数据
10. g：后边搭配一个正则匹配模式，大小写不敏感的

常用参数手册

1. 得到返回记录集最多的10个SQL

```
mysqldumpslow -s r -t 10 /var/lib/mysql/Heygo-slow.log
```

2. 得到访问次数最多的10个SQL

```
mysqldumpslow -s c- t 10/var/lib/mysql/Heygo-slow.log
```

3. 得到按照时间排序的前10条里面含有左连接的查询语句

```
mysqldumpslow -s t -t 10 -g "left join" /var/lib/mysql/Heygo-slow.log
```

4. 另外建议在使用这些命令时结合 | 和more使用，否则有可能出现爆屏情况

```
mysqldumpslow -s r -t 10 /var/lib/mysql/Heygo-slow.log | more
```

3、批量数据脚本

> 创建表

- 建表 SQL

```
CREATE TABLE dept
(
    deptno int unsigned primary key auto_increment,
    dname varchar(20) not null default "",
    loc varchar(8) not null default ""
)ENGINE=INNODB DEFAULT CHARSET=utf8;

CREATE TABLE emp
(
    id int unsigned primary key auto_increment,
    empno mediumint unsigned not null default 0,
    ename varchar(20) not null default "",
    job varchar(9) not null default "",
    mgr mediumint unsigned not null default 0,
    hiredate date not null,
    sal decimal(7,2) not null,
    comm decimal(7,2) not null,
    deptno mediumint unsigned not null default 0
)ENGINE=INNODB DEFAULT CHARSET=utf8;
```

> 设置参数

- 创建函数，假如报错：This function has none of DETERMINISTIC……
- 由于开启过慢查询日志，因为我们开启了bin-log，我们就必须为我们的function指定一个参数。
  - `log_bin_trust_function_creators = OFF` ，默认必须为 function 传递一个参数

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

  - 通过 `set global log_bin_trust_function_creators=1;` 我们可以不用为 function 传参

```
mysql> set global log_bin_trust_function_creators=1;
Query OK, 0 rows affected (0.00 sec)

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

- 这样添加了参数以后，如果mysqld重启，上述参数又会消失，永久方法在配置文件中修改'
  - windows下：my.ini --> [mysqld] 节点下加上 `log_bin_trust_function_creators=1`
  - linux下：/etc/my.cnf --> [mysqld] 节点下加上 `log_bin_trust_function_creators=1`

> 创建函数，保证每条数据都不同

- 随机产生字符串的函数

```
delimiter $$ # 两个 $$ 表示结束
create function rand_string(n int) returns varchar(255)
begin
    declare chars_str varchar(100) default 'abcdefghijklmnopqrstuvwxyz';
    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 $$
```

- 随机产生部门编号的函数

```
delimiter $$
create function rand_num() returns int(5)
begin
    declare i int default 0;
    set i=floor(100+rand()*10);
    return i;
end $$
```

> 创建存储过程

- 创建往emp表中插入数据的存储过程

```
delimiter $$
create procedure insert_emp(in start int(10),in max_num int(10))
begin
    declare i int default 0;
    set autocommit = 0;
    repeat
        set i = i+1;
        insert into emp(empno,ename,job,mgr,hiredate,sal,comm,deptno) values((start+i),rand_string(6),'salesman',0001,curdate(),2000,400,rand_num());
        until i=max_num
        end repeat;
    commit;
end $$
```

- 创建往dept表中插入数据的存储过程

```
delimiter $$
create procedure insert_dept(in start int(10),in max_num int(10))
begin
    declare i int default 0;
    set autocommit = 0;
    repeat
        set i = i+1;
        insert into dept(deptno,dname,loc) values((start+i),rand_string(10),rand_string(8));
        until i=max_num
        end repeat;
    commit;
end $$
```

> 调用存储过程

- 向 dept 表中插入 10 条记录

```
DELIMITER ;
CALL insert_dept(100, 10);
```

```
mysql> select * from dept;
+--------+---------+--------+
| deptno | dname   | loc    |
+--------+---------+--------+
|    101 | aowswej | syrlhb |
|    102 | uvneag  | pup    |
|    103 | lps     | iudgy  |
```

```
|    104 | jipvsk  | ihytx  |
|    105 | hrpzhiv | vjb    |
|    106 | phngy   | yf     |
|    107 | uhgd    | lgst   |
|    108 | ynyl    | iio    |
|    109 | daqbgsh | mp     |
|    110 | yfbrju  | vuhsf  |
+--------+---------+--------+
10 rows in set (0.00 sec)
```

- 向 emp 表中插入 50w 条记录

```
DELIMITER ;
CALL insert_emp(100001, 500000);
```

```
mysql> select * from emp limit 20;
+----+--------+-------+----------+-----+------------+---------+--------+--------+
| id | empno  | ename | job      | mgr | hiredate   | sal     | comm   | deptno |
+----+--------+-------+----------+-----+------------+---------+--------+--------+
|  1 | 100002 | ipbmd | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    101 |
|  2 | 100003 | bfvt  | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    107 |
|  3 | 100004 |       | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    109 |
|  4 | 100005 | cptas | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    101 |
|  5 | 100006 | ftn   | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    108 |
|  6 | 100007 | gzh   | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    102 |
|  7 | 100008 | rji   | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    100 |
|  8 | 100009 |       | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    106 |
|  9 | 100010 | tms   | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    100 |
| 10 | 100011 | utxe  | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    101 |
| 11 | 100012 | vbis  | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    104 |
| 12 | 100013 | qgfv  | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    104 |
| 13 | 100014 | wrvb  | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    105 |
| 14 | 100015 | dyks  | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    109 |
| 15 | 100016 | hpcs  | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    101 |
| 16 | 100017 | fxb   | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    108 |
| 17 | 100018 | vqxq  | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    102 |
| 18 | 100019 | rq    | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    102 |
| 19 | 100020 | l     | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    106 |
| 20 | 100021 | lk    | salesman |   1 | 2020-08-05 | 2000.00 | 400.00 |    100 |
+----+--------+-------+----------+-----+------------+---------+--------+--------+
20 rows in set (0.00 sec)
```

4、Show Profile

> **Show Profile 是什么?**

1. 是mysql提供可以用来分析当前会话中语句执行的资源消耗情况。可以用于SQL的调优测量
2. 官网：http://dev.mysql.com/doc/refman/5.5/en/show-profile.html
3. 默认情况下，参数处于关闭状态，并保存最近15次的运行结果

> 分析步骤

查看是当前的SQL版本是否支持Show Profile

- show variables like ‘profiling%’; 查看 Show Profile 是否开启

```
mysql> show variables like 'profiling%';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| profiling              | OFF   |
| profiling_history_size | 15    |
+------------------------+-------+
2 rows in set (0.01 sec)
```

开启功能 Show Profile ，默认是关闭，使用前需要开启

- `set profiling=on;` 开启 Show Profile

```
mysql> set profiling=on;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> show variables like 'profiling%';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| profiling              | ON    |
| profiling_history_size | 15    |
+------------------------+-------+
2 rows in set (0.00 sec)
```

运行SQL

- 正常 SQL

```
select * from tbl_emp;
select * from tbl_emp e inner join tbl_dept d on e.deptId = d.id;
select * from tbl_emp e left join tbl_dept d on e.deptId = d.id;
```

- 慢 SQL

```
select * from emp group by id%10 limit 150000;
select * from emp group by id%10 limit 150000;
select * from emp group by id%20 order by 5;
```

查看结果

- 通过 show profiles; 指令查看结果

```
mysql> show profiles;
+----------+------------+--------------------------------------------------------------------------+
| Query_ID | Duration   | Query                                                                    |
+----------+------------+--------------------------------------------------------------------------+
|        1 | 0.00052700 | show variables like 'profiling%'                                         |
|        2 | 0.00030300 | select * from tbl_emp                                                    |
|        3 | 0.00010650 | select * from tbl_emp e inner join tbl_dept d on e.'deptId' = d.'id'     |
|        4 | 0.00031625 | select * from tbl_emp e inner join tbl_dept d on e.deptId = d.id         |
|        5 | 0.00042100 | select * from tbl_emp e left join tbl_dept d on e.deptId = d.id          |
|        6 | 0.38621875 | select * from emp group by id%20 limit 150000                            |
|        7 | 0.00014900 | select * from emp group by id%20 order by 150000                         |
|        8 | 0.38649000 | select * from emp group by id%20 order by 5                              |
|        9 | 0.06782700 | select COUNT(*) from emp                                                 |
|       10 | 0.35434400 | select * from emp group by id%10 limit 150000                            |
+----------+------------+--------------------------------------------------------------------------+
10 rows in set, 1 warning (0.00 sec)
```

诊断SQL

- `show profile cpu, block io for query SQL编号;` 查看 SQL 语句执行的具体流程以及每个步骤花费的时间

```
mysql> show profile cpu, block io for query 2;
+----------------------+----------+----------+------------+--------------+---------------+
| Status               | Duration | CPU_user | CPU_system | Block_ops_in | Block_ops_out |
+----------------------+----------+----------+------------+--------------+---------------+
| starting             | 0.000055 | 0.000000 |   0.000000 |            0 |             0 |
| checking permissions | 0.000007 | 0.000000 |   0.000000 |            0 |             0 |
| Opening tables       | 0.000011 | 0.000000 |   0.000000 |            0 |             0 |
| init                 | 0.000024 | 0.000000 |   0.000000 |            0 |             0 |
| System lock          | 0.000046 | 0.000000 |   0.000000 |            0 |             0 |
| optimizing           | 0.000018 | 0.000000 |   0.000000 |            0 |             0 |
| statistics           | 0.000008 | 0.000000 |   0.000000 |            0 |             0 |
| preparing            | 0.000019 | 0.000000 |   0.000000 |            0 |             0 |
| executing            | 0.000003 | 0.000000 |   0.000000 |            0 |             0 |
| Sending data         | 0.000089 | 0.000000 |   0.000000 |            0 |             0 |
| end                  | 0.000004 | 0.000000 |   0.000000 |            0 |             0 |
| query end            | 0.000003 | 0.000000 |   0.000000 |            0 |             0 |
| closing tables       | 0.000005 | 0.000000 |   0.000000 |            0 |             0 |
| freeing items        | 0.000006 | 0.000000 |   0.000000 |            0 |             0 |
| cleaning up          | 0.000006 | 0.000000 |   0.000000 |            0 |             0 |
+----------------------+----------+----------+------------+--------------+---------------+
15 rows in set, 1 warning (0.00 sec)
```

- 参数备注：

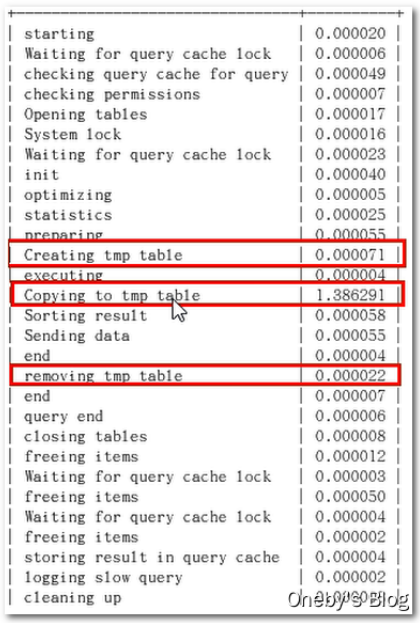
1. ALL：显示所有的开销信息
2. BLOCK IO：显示块IO相关开销
3. CONTEXT SWITCHES：上下文切换相关开销
4. CPU：显示CPU相关开销信息
5. IPC：显示发送和接收相关开销信息
6. MEMORY：显示内存相关开销信息
7. PAGE FAULTS：显示页面错误相关开销信息
8. SOURCE：显示和Source_function，Source_file，Source_line相关的开销信息
9. SWAPS：显示交换次数相关开销的信息

日常开发需要注意的结论

1. converting HEAP to MyISAM：查询结果太大，内存都不够用了往磁盘上搬了。
2. Creating tmp table：创建临时表，mysql 先将拷贝数据到临时表，然后用完再将临时表删除
3. Copying to tmp table on disk：把内存中临时表复制到磁盘，危险！！！
4. locked：锁表

举例

奇了怪了。。。老师的慢 SQL 我怎么复现不了，下面是老师的例子



starting	0.000020
Waiting for query cache lock	0.000006
checking query cache for query	0.000049
checking permissions	0.000007
Opening tables	0.000017
System lock	0.000016
Waiting for query cache lock	0.000023
init	0.000040
optimizing	0.000005
statistics	0.000025
preparing	0.000055
Creating tmp table	0.000071
executing	0.000004
Copying to tmp table	1.386291
Sorting result	0.000058
Sending data	0.000055
end	0.000004
removing tmp table	0.000022
end	0.000007
query end	0.000006
closing tables	0.000008
freeing items	0.000012
Waiting for query cache lock	0.000003
freeing items	0.000050
Waiting for query cache lock	0.000004
freeing items	0.000002
storing result in query cache	0.000004
logging slow query	0.000002
cleaning up	0.000001

5、全局查询日志

永远不要在生产环境开启这个功能。

> 配置启用全局查询日志

- 在mysql的my.cnf中，设置如下：

```
# 开启
general_log=1

# 记录日志文件的路径
general_log_file=/path/logfile

# 输出格式
log_output=FILE
```

> **编码启用全局查询日志**

- 执行如下指令开启全局查询日志

```
set global general_log=1;
set global log_output='TABLE';
```

- 此后，你所执行的sql语句，将会记录到mysql库里的general_log表，可以用下面的命令查看

```
select * from mysql.general_log;
```

```
mysql> select * from mysql.general_log;
+----------------------+----------------------------+-----------+-----------+--------------+-------------------------------------------+
| event_time           | user_host                  | thread_id | server_id | command_type | argument                                  |
+----------------------+----------------------------+-----------+-----------+--------------+-------------------------------------------+
| 2020-08-05 14:41:07 | root[root] @ localhost [] |        14 |         0 | Query        | select * from emp group by id%10 limit 150000 |
| 2020-08-05 14:41:12 | root[root] @ localhost [] |        14 |         0 | Query        | select COUNT(*) from emp                  |
| 2020-08-05 14:41:30 | root[root] @ localhost [] |        14 |         0 | Query        | select * from mysql.general_log           |
+----------------------+----------------------------+-----------+-----------+--------------+-------------------------------------------+
3 rows in set (0.00 sec)
```