



.....

# MySQL优化

.....

黄钧阳  
2016年11月21日

# 大纲

- ❖ 系统优化：硬件、架构
  - ❖ 数据库配置优化
  - ❖ 应用优化
-

# 系统优化

- ❖ 硬件：更快的硬盘、大内存、多核**CPU**、高带宽网络，专业的存储服务器（**NAS**、**SAN**）
  - ❖ 操作系统：**No Swap**、增加系统和**mysql**进程打开文件数量、使用支持大文件的文件系统
  - ❖ 架构：如果 **MySQL** 访问频繁，考虑 **Master/Slave** 读写分离；数据库分区、分表、数据库分片（分布式），也考虑使用相应缓存服务帮助 **MySQL** 缓解访问压力
-

# 数据库配置优化

- ❖ 采用源码自编译的方式安装**Mysql**
  - ❖ 针对 **MyISAM** 或 **InnoDB** 不同引擎进行不同定制性配置
  - ❖ 针对不同的应用情况进行合理配置（ **my.cnf** ）
-

# 数据库配置优化

## ❖ 公用选项

选项	缺省值	推荐值	说明
<code>max_connections</code>	151	1024	MySQL服务器同时处理的数据库连接的最大数量
<code>query_cache_size</code>	0 (不打开)	16M	查询缓存区的最大长度，按照当前需求，一倍一倍增加，本选项比较重要。如果写的操作很多，最好不打开，因为写会刷新此缓存。
<code>sort_buffer_size</code>	512K	16M	每个线程的排序缓存大小，一般按照内存可以设置为2M以上，推荐是16M，该选项对排序 <code>order by</code> ， <code>group by</code> 起作用
<code>record_buffer</code>	128K	16M	每个进行一个顺序扫描的线程为其扫描的每张表分配这个大小的一个缓冲区，可以设置为2M以上
<code>table_cache</code>	64	512	为所有线程打开表的数量。增加该值能增加mysqld要求的文件描述符的数量。

# 数据库配置优化

## ❖ MyISAM 选项

选项	缺省值	推荐值	说明
key_buffer_size	8M	256M	用来存放索引区块的缓存值, 建议128M以上, 不要大于内存的30%
read_buffer_size	128K	16M	用来做MyISAM表全表扫描的缓冲大小. 为从数据表顺序读取数据的读操作保留的缓存区的长度
myisam_sort_buffer_size	8M	128M	设置,恢复,修改表的时候使用的缓冲大小, 值不要设的太大

# 数据库配置优化

## ❖ InnoDB 选项

选项	缺省值	推荐值	说明
<code>innodb_buffer_pool_size</code>	128M	1G	InnoDB使用一个缓冲池来保存索引和原始数据, 这里你设置越大, 你在存取表里面数据时所需要的磁盘 I/O 越少, 推荐物理内存的60%左右, 这个参数非常重要
<code>innodb_additional_mem_pool_size</code>	8M	16M	InnoDB用来保存 metadata 信息, 表越多, 需要内存越多, 如果内存是4G, 最好不值超过200M
<code>innodb_flush_log_at_trx_commit</code>	1	0	0 代表日志只大约每秒写入日志文件并且日志文件刷新到磁盘; 1 为执行完没执行一条SQL马上commit; 2 代表日志写入日志文件在每次提交后, 但是日志文件只有大约每秒才会刷新到磁盘上. 对速度影响比较大, 同时也关系数据完整性
<code>innodb_log_file_size</code>	8M	256M	在日志组中每个日志文件的大小, 一般是 <code>innodb_buffer_pool_size</code> 的25%, 官方推荐是 <code>innodb_buffer_pool_size</code> 的 40-50%, 设置大一点来避免在日志文件覆写上不必要的缓冲池刷新行为
<code>innodb_log_buffer_size</code>	8M	16M	用来缓冲日志数据的缓冲区的大小. 推荐是8M, 最好是 8M-64M 之间

# 应用优化方式

## ❖ 使用查询缓存

查询缓存的目的很简单，将 **select** 查询的结果缓存在内存中，以供下次直接获取。在默认情况下，MySQL 是没有开启查询缓存的，我们可以进行以下配置：

```
query_cache_size = 268435456  
query_cache_type = 1  
query_cache_limit = 1048576
```

这样一来，MySQL 将拥有 256MB 的内存空间来缓存查询结果。对于以 **select** 查询为主的应用，查询缓存理所当然地起到性能提升的作用，不论是 **InnoDB** 还是 **MyISAM**，查询缓



# 应用优化方式

## ❖ 定位慢查询和不使用索引的查询

在 MySQL 中开启慢查询日志非常简单，在 `my.cnf` 中增加以下配置选项：

```
long_query_time = 1  
log-slow-queries = /data/var/mysql_slow.log
```

这意味着 MySQL 会自动将执行时间超过 1 秒的查询记录在指定路径的 `mysql_slow.log` 文件中。除此之外，你还可以将所有没有使用索引的查询也记录下来，只需增加以下选项即可：

```
log-queries-not-using-indexes
```

# 应用优化方式

- ❖ 设计合理的数据表结构
  - ❖ 建立合适有效的数据库索引
  - ❖ 查询**SQL**语句简洁高效
-

# 应用优化方式

## ❖ 表结构设计原则

- 选择合适的数据类型：如果能够定长尽量定长（例如varchar-->char）
- 为了避免联表查询，有时候可以适当的数据冗余（反范式），比如 邮箱、姓名这些不容易更改的数据
- 选择合适的表引擎，有时候 MyISAM 适合，有时候 InnoDB适合
- 为保证查询性能，最好每个表都建立有 auto\_increment 字段， 建立合适的数据库索引
- 避免nullable字段，最好给每个字段都设定 default 值
- 不要使用无法加索引的类型作为关键字段，比如 text类型
- 对于某些字段，例如“省份”、“性别”，可以定义为enum类型

# 应用优化方式

## ❖ 索引建立原则

- 一般针对数据分散的关键字建立索引，比如ID、QQ，像性别、状态值等等建立索引没有意义
- 索引要建立在那些用于where判断、join和order by的字段上
- 尽量使用短索引，一般对int、char/varchar、date/time 等类型的字段建立索引
- 尽量建立联合索引，但是要注意前缀性
- 一般建议每条记录最好有一个能快速定位的独一无二定位的唯一标识
- 不要过度索引，单表建立的索引不要超过5个，否则更新索引将很耗时

# 应用优化方式

## ❖ 高效SQL

- 能够快速缩小结果集的 **WHERE** 条件写在前面，如果有恒量条件，也尽量放在前面
- 尽量避免使用 **GROUP BY**、**DISTINCT**、**OR**、**IN** 等语句的使用，避免使用大表联合查询和子查询，因为将使执行效率大大下降
- 能够使用索引的字段尽量进行有效的合理排列，如果使用了联合索引，请注意提取字段的前后顺序
- 针对索引字段使用 **>**、**>=**、**=**、**<**、**<=**、**IF NULL**和**BETWEEN** 将会使用索引， 如果对某个索引字段进行 **LIKE** 查询，使用 **LIKE '%abc%'** 不能使用索引，使用 **LIKE 'abc%'** 将能够使用索引
- 如果在**SQL**里使用了**MySQL**部分自带函数，索引将失效，同时将无法使用 **MySQL** 的 **Query Cache**，比如 **LEFT()**、**SUBSTR()**、**TO\_DAYS()**、**DATE\_FORMAT()**，等，如果使用了 **OR** 索引也将失效

# 应用优化方式

## ❖ 使用 Explain 语句

- 如果在SELECT语句前放上关键词EXPLAIN，MySQL将解释它如何处理SELECT，提供有关表如何联接和联接的次序。
- EXPLAIN的每个输出行提供一个表的相关信息，并且每个行包括下面的列：

id	SELECT识别符。这是SELECT的查询序列号	select_type	SELECT类型
table	输出的行所引用的表	type	联接类型
possible_keys	possible_keys列指出MySQL能使用哪个索引在该表中找到行	key	key列显示MySQL实际决定使用的键（索引）
key_len	key_len列显示MySQL决定使用的键长度	ref	ref列显示使用哪个列或常数与key一起从表中选择行
rows	rows列显示MySQL认为它执行查询时必须检查的行数	Extra	该列包含MySQL解决查询的详细信息

# 应用优化方式

Explain效率检测使用

## ■ 语法: EXPLAIN SELECT *select\_options*

Type: 类型, 是否使用了索引还是全表扫描, const,ref,range,index,ALL

Key: 实际使用上的索引是哪个

Key\_len: 真正使用了索引的键长度

Ref: 显示了哪些字段或者常量被用来和 key 配合从表中查询记录出来

Rows: 显示了MySQL认为在查询中应该检索的记录数

Extra: 显示了查询中MySQL的附加信息, 关心Using filesort 和 Using temporary, 性能杀手

```
mysql> EXPLAIN EXTENDED SELECT u.subId,r.phoneNum,s.serviceId FROM subscribe as u,service as s,register as r WHERE r.regId=u.  
regId AND s.servId=u.servId AND (u.subscribeStatus=1 OR u.subscribeStatus>=3) AND u.expireTime <= now() LIMIT 0,20;  
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |  
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
| 1 | SIMPLE | s | index | PRIMARY | serviceId | 15 | NULL | 3 | Using index |  
| 1 | SIMPLE | u | ALL | FK_REGID,FK_SERVID | NULL | NULL | NULL | 3 | Using where |  
| 1 | SIMPLE | r | eq_ref | PRIMARY | PRIMARY | 4 | sp.u.regId | 1 | |  
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
3 rows in set, 1 warning (0.00 sec)
```

# 函数和索引

```
mysql> select count(1) from Tbl_ScoreDetail where FTime < left(now(), 10);
+-----+
| count(1) |
+-----+
|    174055 |
+-----+
1 row in set (0.29 sec)
```

```
mysql> select count(1) from Tbl_ScoreDetail where FTime < '2009-07-28';
+-----+
| count(1) |
+-----+
|    174055 |
+-----+
1 row in set (0.08 sec)
```

```
mysql> select count(1) from Tbl_ScoreDetail where FTime like '%2009-07-28%';
+-----+
| count(1) |
+-----+
|      1117 |
+-----+
1 row in set (0.22 sec)
```

```
mysql> select count(1) from Tbl_ScoreDetail where FTime like '2009-07-28%';
+-----+
| count(1) |
+-----+
|      1117 |
+-----+
1 row in set (0.10 sec)
```



# 应用优化方式



## ❖ 索引

- 在了解表的具体应用场景基础上建立索引；
- 为所有主键和外键列建立索引；
- 对出现在WHERE子句、JOIN子句、ORDER BY或GROUP BY子句中的列考虑建立索引；
- 对需要确保唯一性的列考虑建立索引；
- 对于WHERE子句中用AND连接并频繁使用的列使用组合索引，最频繁的列放在最左边；
- 数据更新频繁的列不宜建立索引；
- 数据量较小的表也不宜建立索引

# 应用优化方式

## ❖ 去除查询条件左端的任何标量函数

```
1 • explain
2 select * from att_attendance_record
3 where Year(sign_in_datetime)=2015;
```



Result Set Filter:  Export:  Wrap Cell Content: 

	id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
▶	1	SIMPLE	att_attendance_record	ALL	NULL	NULL	NULL	NULL	77896	Using where

# 应用优化方式

## ❖ 去除查询条件左端的任何数学运算

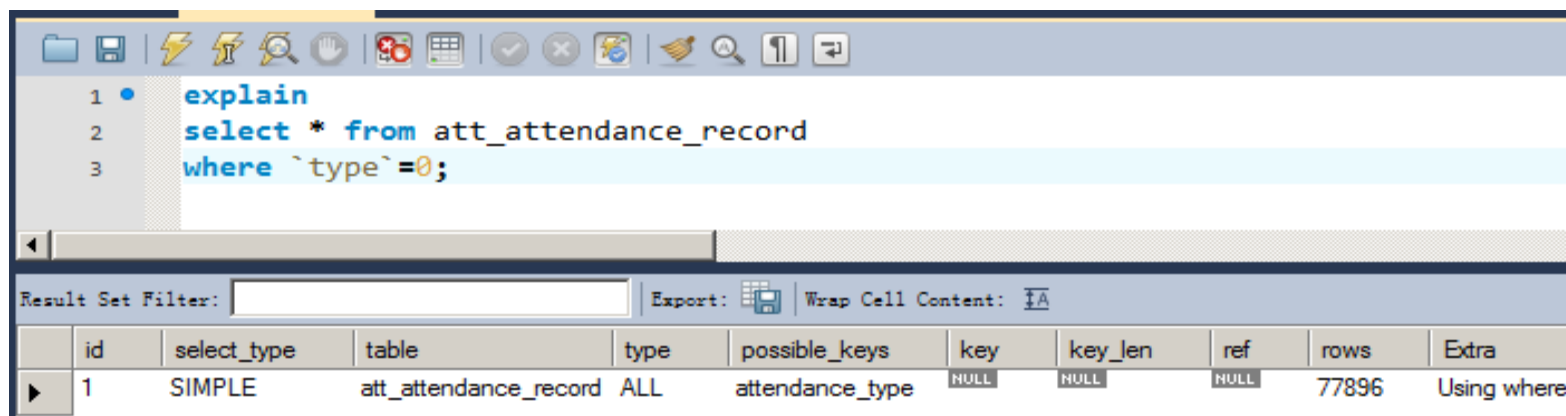
```
1 • explain
2 select * from att_attendance_record
3 where `type` + '1'='01';
```

Result Set Filter:  Export:  Wrap Cell Content: 

	id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
▶	1	SIMPLE	att_attendance_record	ALL	NULL	NULL	NULL	NULL	77896	Using where

# 应用优化方式

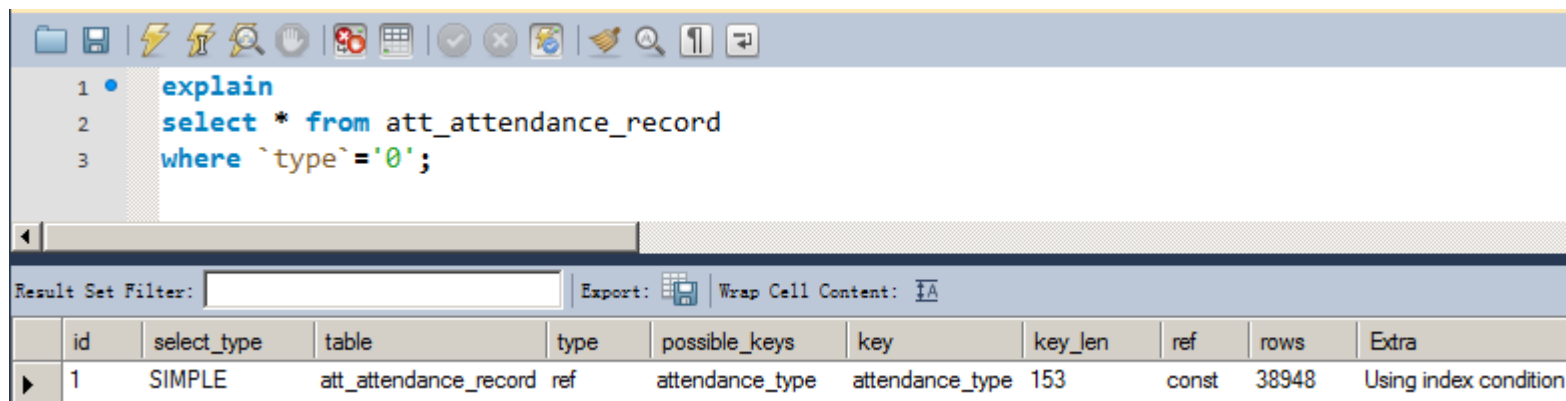
## ❖ 确保宿主变量定义与列数据类型匹配



```
1 • explain
2   select * from att_attendance_record
3   where `type`=0;
```

Result Set Filter:  Export: Wrap Cell Content:

	id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
▶	1	SIMPLE	att_attendance_record	ALL	attendance_type	NULL	NULL	NULL	77896	Using where



```
1 • explain
2   select * from att_attendance_record
3   where `type`='0';
```

Result Set Filter:  Export: Wrap Cell Content:

	id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
▶	1	SIMPLE	att_attendance_record	ref	attendance_type	attendance_type	153	const	38948	Using index condition

# 应用优化方式

## ❖ 尽可能用UNION ALL取代UNION

```
1 • explain
2 select * from att_attendance_record
3 where `type`='0'
4 union
5 select * from att_attendance_record
6 where `type`='a20f51eae4a246d3b3fbe02328f85da0'
```

	id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
▶	1	PRIMARY	att_attendance_record	ref	attendance_type	attendance_type	153	const	38948	Using index condition
	2	UNION	att_attendance_record	ref	attendance_type	attendance_type	153	const	1788	Using index condition
	N/A	UNION RESULT	<union1,2>	ALL	N/A	N/A	N/A	N/A	N/A	Using temporary

# 应用优化方式

## ❖ 使用组合索引时，注意“最左前缀”这个基本原则

- 最左前缀：就是最左优先，我们创建了lname、fname和age的多列索引,相当于创建了lname单列索引，(lname,fname)的组合索引以及(lname,fname,age)组合索引；
- 只有当复合索引的第一个字段出现在SQL语句的谓词条件中时，该索引才会被用到。
- 如复合索引为(lname,fname,age)，只要谓词条件中出现第一个字段lname，就可以用复合索引，否则不会用
- ```
SELECT `uid` FROM people WHERE `fname`='Zhiqun' AND `age`=26
```

上述查询语句因违法“最左前缀”原则，系统通常会扫描整表以匹配数据！



# Thank You !

[www.mysql.com/](http://www.mysql.com/)