mysql-optimization

数据库优化的目的

一、避免出现页面访问错误

- 由于数据库连接超时产生页面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

二、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

三、优化用户体验

- 流畅页面的访问速度
- 良好的网站功能体验

数据库优化的几个方面

SQL及索引优化

一、如何分析SQL查询

- 使用的是MySQL提供的sakila数据库。
- 数据库基于mysql Ver 14.14 Distrib 5.7.30, for Linux (x86_64) using EditLine wrapper版本

二、如何发现有问题的SQL

使用MySQL慢查询日志对有效率问题的SQL进行监控

```
#查看慢查询日志是否启用
show variables like 'slow_query_log';
#开启慢查询日志
set global slow_query_log=on;
#设置慢查询日志保存路径
set global slow_query_log_file='/opt/log/mysql/mysql-slow.log';
#设置记录未使用索引的查询
set global log_queries_not_using_indexes=on;
#设置慢查询时间，单位秒
set global long_query_time=1;
```

慢查询日志的存储格式

```
# Time: 140606 12:30:17
# User@Host: root[root] @ localhost []
# Query_time: 0.000031  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0
SET timestamp=1402029017;
show tables;
```

慢查询日志所包含的内容

- 执行SQL的主机信息
- SQL的执行信息
- SQL的执行时间
- SQL的内容

慢查询日志的分析工具

1. mysqldumpslow，MySQL官方提供，MySQL安装好后就有
2. pt-query-digest

如何通过慢查询日志发现有问题的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；
2. IO大的SQL，注意pt-query-digest分析中的Rows examine项
3. 未命中索引的SQL，注意pt-query-digest分析中Rows examine 和Rows Send 的对比

如何分析SQL查询

使用explain查询SQL的执行计划

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: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，表示没有可能的索引。
- 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
- 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队返回的行排序。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，或者是GROUP BY上

三、SQL优化示例

count()和max()的优化

- 查询最后支付时间，优化max()函数

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

可在payment表上建立payment_date的索引

```
create index idx_paymentdate on payment(payment_date)
```

- 在一条SQL中同时查出2006年和2007年电影的数量，优化count()函数

```
select count(release_year='2006' OR NULL) AS '2006',count(release_year='2007' OR NULL)
as '2007' from film;
```

count(*)与count(id)是有区别的，count(id)不包含为NULL的行。

子查询优化

- 通常情况下，需要把子查询优化为join查询，但在优化时要注意关联键是否有一对多的关系，要注意重复数据。

group by查询优化

- 优化前SQL：

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

- 优化后SQL：

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

limit查询优化

limit常用于分页处理，时常会伴随order by从句使用，因此大多时候会使用Filefilter、Filesort，这样会造成大量的IO问题。

- 优化前SQL：

```
explain select film_id,description from film order by title limit 50,5
```

- 使用有索引的列或主键进行order by操作：（越往后扫描的行越多）

```
explain select film_id,description from film order by film_id limit 50,5
```

- 记录上次返回的主键，在下次查询时使用主键过滤：（**数据被删除未考虑**）

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

四、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从句中出现的列
2. 索引字段约小越好
3. 离散度大的列放到联合索引的前面

   例：select * from payment where staff_id = 2 and customer_id = 584;是index(staff_id,customer_id)好，还是index(customer_id,staff_id)好?

   由于customer_id的离散度更大，所以应该使用index(customer_id,staff_id)。

   通过count(distinct customer_id),count(distinct staff_id)获取唯一值数量，数量越多离散度越高。

五、索引维护及优化示例

1. 重复及冗余索引

   查找重复及冗余索引SQL（在information_schema库中运行）：

```
select a.table_schema as '数据名',a.table_name as '表名',a.index_name as '索引',b.index_name as '索引2',a.column_name as '重复列名' from statistics a join statistics b on a.table_schema=b.table_schema and a.table_name=b.table_name and a.seq_in_index=b.seq_in_index and a.column_name=b.column_name where a.seq_in_index=1 and a.index_name<>b.index_name
```

   使用pt-duplicate-key-checker工具：

```
pt-duplicate-key-checker -uroot -p '123456' -h 127.0.0.1
```

2. 删除不用的索引

   目前MySQL中没有记录索引的使用情况，但在PerconMySQL和MariaDB中可以通过INDEX_STATISTICS表来查看哪些索引未使用，但在MySQL中目前只能通过查看慢查询日志配合pt-index-usage工具来进行索引使用情况的分析。

```
pt-index-usage -uroot -p '123456' mysql-slow.log
```

表结构优化

一、选择合适的数据类型

数据类型的选择，重点在与合适二字，如何确定选择的数据类型是否合适?

1. 使用可以存下你的数据的最小数据类型
2. 使用简单的数据类型。int要比varchar类型在MySQL处理上简单

3. 尽可能的使用not null定义字段
4. 尽量少用text类型，非用不可最好考虑分表

例：

- 使用int来存储日期时间，利用FROM_UNIXTIME(),UNIX_TIMESTAMP()两个函数来进行转换
- 使用bigint来存储IP地址，利用INET_ATON(),INET_NTOA()两个函数进行转换

二、表的范式优化

范式化是指数据库设计的规范，目前说到的范式化一般是指第三范式，也就是要求数据库表中不存在非关键字段对任意候选关键字段的传递函数依赖。

不符合第三范式要求的表存在下列问题：

1. 数据冗余
2. 数据的插入异常
3. 数据的更新异常
4. 数据的删除异常

三、表的反范式化优化

反范式化是指为了查询效率的考虑把原本符合第三范式的表适当的增加冗余，以达到优化查询效率的目的，反范式化是一种以空间来换时间的操作。

四、表的垂直拆分

所谓的垂直拆分，就是把原来有很多列的表拆分成多个表，这解决了表的宽度问题。通常垂直拆分可以按以下原则进行：

1. 把不常用的字段单独存放到一个表中
2. 把大字段独立存放到一个表中
3. 把经常一起使用的字段放到一起

五、表的水平拆分

表的水平拆分主要是为了解决单表的数据量过大的问题，水平拆分的表每一个表的结构都是完全一致的。

常用的水平拆分方法：

1. 对主键进行hash运算，如果要拆分成5个表则使用mod(主键,5)取出0-4个值
2. 针对不同的hashID把数据存到不同的表中

难点：

1. 跨分区表进行数据查询
2. 统计及后台报表操作

系统配置优化

一、操作系统配置优化

数据库是基于操作系统的，目前大多数MySQL都是安装在Linux系统之上，所以对于操作系统的一些参数配置也会影响到MySQL的性能，下面就列出一些常用到的系统配置。

网络方面的配置，要修改/etc/sysctl.conf文件

```
#增加tcp支持的队列数
net.ipv4.tcp_max_syn_backlog = 65535
#减少断开连接时，资源回收
net.ipv4.tcp_max_tw_buckets = 8000
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_fin_timeout = 10
```

打开文件数的限制，可以使用ulimit -a查看目录的各位限制，可以修改/etc/security/limits.conf文件，增加以下内容以修改打开文件数量的限制

```
* soft nofile 65535
* hard nofile 65535
```

除此之外最好在MySQL服务器上关闭iptables,selinux等防火墙软件。（使用硬件防火墙代替）

二、MySQL配置优化

MySQL可以通过启动时指定配置参数和使用配置文件两种方式进行配置，在大多数情况下配置文件位于/etc/my.cnf或是/etc/mysql/my.cnf，在Windows系统配置文件可以是位于C:/windows/my.ini文件，MySQL查找配置文件的顺序可以通过以下方法获得

```
mysql --verbose --help | grep -A 1 'Default options'
#命令执行结果打印
#Default options are read from the following files in the given order:
#/etc/mysql/my.cnf /etc/my.cnf ~/.my.cnf
```

注意：如果存在多个位置存在配置文件，则后面的会覆盖前面的。

MySQL配置文件常用参数说明

```
#1、非常重要的一个参数，用于配置Innodb的缓冲池，如果机器只运行数据库且只有Innodb表，则推荐配置量为总内存的75%
#innodb_buffer_pool_size
SELECT ENGINE,ROUND(SUM(data_length+index_length)/1024/1024,1) AS "Total MB" FROM information_schema.TABLES WHERE TABLE_SCHEMA NOT IN ("information_schema","performance_schema") GROUP BY ENGINE;
#innodb_buffer_pool_size >= Total MB
#2、MySQL5.5中新增参数，可以控制缓冲池的个数，默认情况下只有一个缓冲池
#innodb_buffer_pool_instances
#3、innodb log缓冲的大小，由于日志最长每秒钟就会刷新，所以一般不用太大
#innodb_log_buffer_size
#4、关键参数，对innodb的IO效率影响很大，默认值为1，可以去0，1，2三个值，一般建议设为2，但如果对数据安全性要求比较高则使用默认值1
#innodb_flush_log_at_trx_commit
#5、以下两个参数决定了Innodb读写的IO进程数，默认为4(可根据cpu个数设置)
#innodb_read_io_threads
#innodb_write_io_threads
#6、关键参数，控制Innodb每一个表使用独立的表空间，默认为OFF，也就是所有表都会建立在共享表空间中，建议设置为ON
#innodb_file_per_table
```

```
#7、决定MySQL在什么情况下刷新innodb表的统计信息，一般设置为OFF
#innodb_stats_on_metadata
```

使用第三方配置工具

Percon Configuration Wizard 生成推荐配置。

硬件优化

CPU选择

1. MySQL有一些工作只能使用到单核CPU，Replicate，SQL...
2. MySQL对CPU核数的支持并不是越多越好，MySQL5.5使用的服务器不要超过32核

Disk IOy优化

常用RAID级别简介

- RAID0：也称为条带，就是吧多个磁盘链接成一个硬盘使用，这个级别的IO最好。
- RAID1：也称为镜像，要求至少有两个磁盘，每组磁盘存储的数据相同。
- RAID5：也就是把多个（最少3个）硬盘合并成1个逻辑盘使用，数据读写时会建立奇偶校验信息，并且奇偶校验信息和相对应的数据分别存储于不同的磁盘上。当RAID5的一个磁盘数据发生损坏后，利用剩下的数据和相应的奇偶校验信息去恢复被损坏的数据。
- RAID1+0：就是RAID1和RAID0的结合，同时具备两个级别的优缺点。一般建议数据库使用这个级别。

思考：SNA和NAT是否适合数据库？（磁盘阵列）

1. 常用于高可用的解决方案。
2. 顺序读写效率很高，但是随机读写不如人意。
3. 数据库随机读写比率很高