MySql - 优化方案

1.数据库结构的设计

1.1 设计合理的数据库模型的原因

```
#1. 如果不能设计一个合理的数据库模型
    不仅会增加客户端和服务器段程序的编程和维护的难度
    而且将会影响系统实际运行的性能。所以，在一个系统开始实施之前，完备的数据库模型的设计是必须的
#2. 在一个系统分析、设计阶段，因为数据量较小，负荷较低
    我们往往只注意到功能的实现，而很难注意到性能的薄弱之处，
    等到系统投入实际运行一段时间后，才发现系统的性能在降低，这时再来考虑提高系统性能则要花费更多的人力物力，而整个系统也不可避免的形成了一个打补丁工程。
#3.在考虑整个系统的流程的时候，必须要考虑，在高并发大数据量的访问情况下，我们的系统会不会出现极端的情况。
    例如：对外统计系统在7月16日出现的数据异常的情况，并发大数据量的的访问造成，数据库的响应时间不能跟上数据刷新的速度造成。具体情况是：在日期临界时（00：00：00），判断数据库中是否有当前日期的记录，没有则插入一条当前日期的记录。在低并发访问的情况下，不会发生问题，但是当日期临界时的访问量相当大的时候，在做这一判断的时候，会出现多次条件成立，则数据库里会被插入多条当前日期的记录，从而造成数据错误。
#4.数据库的模型确定下来之后，我们有必要做一个系统内数据流向图，分析可能出现的瓶颈
```

1.2 针对大型的数据量提前进行分库和分表

```
#分库分表尽量在数据库设计初期敲定方案，否则后期会极大增加代码复杂性而且不易更改
#索引适合应对百万级别的数据量，千万级别数据量使用的好，勉强也能凑合，但如果是上亿级别的数据量，索引就无能为力了，因为单索引文件可能就已经上百兆或者更多了，那么，轮到我们的分表分区登场了
#分库分表的前提条件是在执行查询语句之前，已经知道需要查询的数据可能会落在哪一个分库和哪一个分表中。
```

```
# ---------------------- 分表 ----------------------

#1.垂直分表
    基于数据库中的"列"进行，某个表字段较多，可以新建一张扩展表，将不经常用或字段长度较大的字段拆分出去到扩展表中。
    在字段很多的情况下（例如一个大表有100多个字段），通过"大表拆小表"，更便于开发与维护，也能避免跨页问题，MySQL底层是通过数据页存储的，一条记录占用空间过大会导致跨页，造成额外的性能开销。另外数据库以行为单位将数据加载到内存中，这样表中字段长度较短且访问频率较高，内存能加载更多的数据，命中率更高，减少了磁盘IO，从而提升了数据库性能。

#水平分表
    水平分表也称为横向分表，比较容易理解，就是将表中不同的数据行按照一定规律分布到不同的数据库表中（这些表保存在同一个数据库中），这样来降低单表数据量，优化查询性能。最常见的方式就是通过主键或者时间等字段进行Hash和取模后拆分
```

```
# ------------------------分库 ----------------------------
垂直分库在“微服务”盛行的今天已经非常普及了。基本的思路就是按照业务模块来划分出不同的数据库，而不是像早期一样将所有的数据表都放到同一个数据库中：
例如：可以把查询库和系统库（增删改比较频繁的表）分开了，这样如果有大查询，不影响系统库
```

#分库分表带来的问题
1、事务一致性问题
2、跨节点关联查询 join 问题
　　切分之前，系统中很多列表和详情页所需的数据可以通过sql join来完成。而切分之后，数据可能分布在不同的节点上，此时join带来的问题就比较麻烦了，考虑到性能，尽量避免使用join查询

1.3表结构设计要注意的问题

1.能够用数字类型的字段尽量选择数字类型而不用字符串类型的（电话号码）
　　#这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接回逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。
2.对于不可变字符类型char和可变字符类型varchar 都是8000字节,在设计字段的时候可以灵活选择
　　char查询快，但是耗存储空间，例如用户名、密码等长度变化不大的字段可以选择CHAR
　　varchar查询相对慢一些但是节省存储空间,例如对于评论等长度变化大的字段可以选择VARCHAR。
3. 尽可能不要使用NULL值
　　因为建表的时候，如果不对创建的值设置默认值，MySQL都会设置默认为 NULL
　　NOT IN、!=等负向条件查询在有NULL值的情况下返回永远为空结果，查询容易出错
　　NULL列需要一个额外字节作为判断是否为NULL的标志位
　　MySQL难以优化对可为NULL的列的查询

4.最好不要用自增属性字段作为主键与子表关联。不便于系统的迁移和数据恢复。对外统计系统映射关系丢失
5.数据行的长度不要超过8020字节，如果超过这个长度在物理页中这条数据会占用两行从而造成存储碎片，降低查询效率
6.字段的长度在最大限度的满足可能的需要的前提下，应该尽可能的设得短一些，这样可以提高查询的效率，而且在建立索引的时候也可以减少资源的消耗

2.查询优化

2.1查询语句的注意事项

请尽量使用简单的查询，避免使用表链接
请尽量避免全表扫描，包括但不限于：
where子句条件横真或为空
使用LIKE
使用不等操作符（<>、!=）
查询含义is null的列
在非索引列上使用or
多条件查询时，请把简单查询条件或则索引列查询置于前面
请尽量指定需要查询的列，不要偷懒使用select *
如果不指定，一方面会返回多余的数据，占用宽带等
另一方面MySQL执行查询的时候，没有字段时会先去查询表结构有哪些字段
大些的查询关键字比小写快一点点
使用子查询会创建临时表，会比链接（JOIN）和联合（UNION）稍慢
在索引字段上查询尽量不要使用数据库函数，不便于缓存查询结果
当只要一行数据时，请使用LIMIT 1，如果数据过多，请适当设定LIMIT，分页查询
千万不要 ORDER BY RAND()，性能极低

2.2 应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：

```
select id from t where num is null
```
可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：
```
select id from t where num=0
```

2.3使用索引

```
# 创建时注意
      一般来说，每张表都需要有一个主键id字段
      常用于查询的字段应该设置索引
      varchar类型的字段，在建立索引的时候，最好指定长度
      查询有多个条件时，优先使用具有索引的条件
      像LIKE条件这样的模糊搜索对于字段索引是无效的，需要另外建立关键词索引来解决
      请尽量不要在数据库层面约束表和表之间的关系，这些表之间的依赖应该在代码层面去解决
#保证命中索引
```

```
#何时使用聚集索引或非聚集索引
下面的表总结了何时使用聚集索引或非聚集索引（很重要）。
动作描述                    使用聚集索引                使用非聚集索引
列经常被分组排序              应                        应
返回某范围内的数据            应                        不应
一个或极少不同值              不应                      不应
小数目的不同值                应                        不应
大数目的不同值                不应                      应
频繁更新的列                  不应                      应
外键列                        应                        应
主键列                        应                        应
频繁修改索引列                不应                      应
```

3.读写分离

3.1 什么是读写分离?

```
其实就是将数据库分为了主从库，一个主库用于写数据，多个从库完成读数据的操作，主从库之间通过某种机制进行数据的同步，是一种常见的数据库架构。
一个组从同步集群，通常被称为是一个“分组”。
```

3.2 数据库分组架构解决什么问题?

```
大多数互联网业务，往往读多写少，这时候，数据库的读会首先称为数据库的瓶颈，这时，如果我们希望能够线性的提升数据库的读性能，消除读写锁冲突从而提升数据库的写性能，那么就可以使用“分组架构”（读写分离架构）。
#读写分离是用来解决数据库的读性能瓶颈的
```

4.使用缓存

4.1为什么使用缓存?

缓存，也是互联网中常常使用到的一种架构方式，读写分离是通过多个读库，分摊了数据库读的压力，
而存储则是通过缓存的使用，减少了数据库读的压力。他们没有谁替代谁的说法，但是，如果在缓存的读写分离进行二选一时，还是应该首先考虑缓存。

#为什么呢?

缓存的使用成本要比从库少非常多；

缓存的开发比较容易，大部分的读操作都可以先去缓存，找不到的再渗透到数据库。

当然，如果我们已经运用了缓存，但是读依旧还是瓶颈时，就可以选择“读写分离”架构了。简单来说，我们可以将读写分离看做是缓存都解决不了时的一种解决方案。

4.2 具体使用

#使用redis等缓存，还有本地文件缓存等，可以极大地减少数据库查询次数。缓存这个东西，一定要分析自己系统的数据特点，适当选择。

对于一些常用的数据，比如配置信息等，可以放在缓存中

可以在本地缓存数据库的表结构

缓存的数据一定要注意及时更新，还有设置有效期

增加缓存务必会增加系统复杂性，一定要注意权衡