# MySQL优化

## 概述

MySQL数据库常见的两个瓶颈是：CPU和I/O的瓶颈。

CPU在饱和的时候一般发生在**数据装入内存或从磁盘上读取数据时候。**

磁盘I/O瓶颈发生在装入数据远大于内存容量的时候，如果应用分布在网络上，**那么查询量相当大的时候那么瓶颈就会出现在网络上。**

我们可以用mpstat, iostat, sar和vmstat来查看系统的性能状态。除了服务器硬件的性能瓶颈，对于MySQL系统本身，我们可以使用工具来优化数据库的性能。

## mysql优化的要点

1 表的设计合理化(符合 3NF，必要时允许数据冗余)

2.1 SQL语句优化(以查询为主)

2.2 适当添加索引(主键索引，唯一索引，普通索引(包括联合索引)，全文索引)

## 3NF(三大范式)

1. 第一范式

第一范式是最基本的范式。要求数据库表中的所有字段值都是不可分解的原子值，即要求列的原子性。

2. 第二范式

第二范式是建立在第一范式的基础之上的，要求数据库表中的记录(行)必须是唯一的，即要求行的唯一性。

通常通过设计一个主键来实现(建议主键不要有具体的业务含义)。

3. 第三范式

满足第三范式必须要满足第二范式。要求非主键列必须直接依赖于主键，不能存在传递依赖，及表中不能有冗余数据。

表中某字段的信息可以通过其他列推导出来，就不应该设计此列。

反3NF：没有冗余的数据库表设计未必是最优设计，有时为了提高效率，需要降低范式标准，适当增加冗余字段。

# 索引优化

## 1.索引

一般的应用系统，读写比例在10：1左右，而且插入操作和一般的更新操作很少出现性能问题，在生产环境中，我们遇到最多的也是最容易出现问题的，还是一些复杂的查询操作，因此对查询语句的优化是重中之重，**加速查询最好的方法就是索引。**

## 2.索引类型

- 普通索引

  是最基本的索引，它没有任何限制。

- 唯一索引

  与前面的普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一。

- 组合索引

  指多个字段上创建的索引，只有在查询条件中使用了创建索引时的第一个字段，索引才会被使用。

- 主键索引

  是一种特殊的唯一索引，一个表只能有一个主键，不允许有空值。一般是在建表的时候同时创建主键索引

- 全文索引

主要用来查找文本中的关键字，而不是直接与索引中的值相比较。fulltext索引跟其它索引大不相同，它更像是一个搜索引擎，而不是简单的where语句的参数匹配。fulltext索引配合match against操作使用，而不是一般的where语句加like。它可以在create table，alter table ，create index使用，不过目前只有char、varchar，text 列上可以创建全文索引。值得一提的是，在数据量较大时候，现将数据放入一个没有全局索引的表中，然后再用CREATE index创建fulltext索引，要比先为一张表建立fulltext然后再将数据写入的速度快很多。

**3.索引优化**

- 只要列中含有NULL值，就最好不要在此例设置索引，复合索引如果有NULL值，此列在使用时也不会使用索引
- 尽量使用短索引，如果可以，应该制定一个前缀长度
- 对于经常在where子句使用的列，最好设置索引，这样会加快查找速度
- 对于有多个列where或者order by子句的，应该建立复合索引
- 对于like语句，以%或者‘-’开头的不会使用索引，以%结尾会使用索引
- 尽量不要在列上进行运算（函数操作和表达式操作）

- **尽量不要使用not in和<>操作**
- 如果查询条件中带有 or ，则要求 or 中涉及到所有列都有索引，否则不会使用索引。

  建议：尽量避免使用 or 关键字。

# SQL慢查询的优化

**1.如何捕获低效sql**

1）slow_query_log

这个参数设置为ON，可以捕获执行时间超过一定数值的SQL语句。

2）ong_query_time

当SQL语句执行时间超过此数值时，就会被记录到日志中，建议设置为1或者更短。

3）slow_query_log_file

记录日志的文件名。

4）log_queries_not_using_indexes

这个参数设置为ON，可以捕获到所有未使用索引的SQL语句，尽管这个SQL语句有可能执行得挺快。

**2.慢查询优化的基本步骤**

1)先运行看看是否真的很慢，注意设置SQL_NO_CACHE

2）where条件单表查，锁定最小返回记录表。这句话的意思是把查询语句的where都应用到表中返回的记录数最小的表开始查起，单表每个字段分别查询，看哪个字段的区分度最高

3)explain查看执行计划，是否与1预期一致（从锁定记录较少的表开始查询）

4)order by limit 形式的sql语句让排序的表优先查

5)了解业务方使用场景

6)加索引时参照建索引的几大原则

7)观察结果，不符合预期继续从1开始分析

# 数据库表优化

- 表的字段尽可能用NOT NULL
- 字段长度固定的表查询会更快
- 把数据库的大表按时间或一些标志分成小表
- 将表拆分

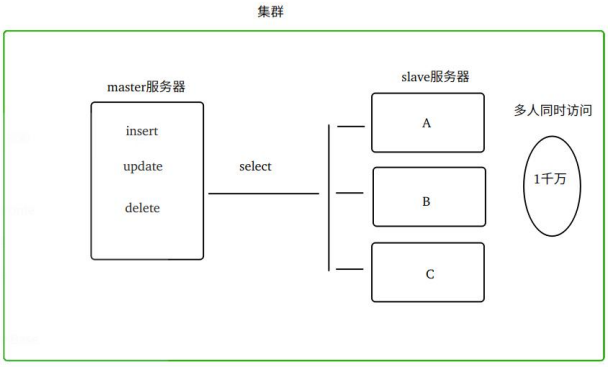
数据表拆分：主要就是垂直拆分和水平拆分。

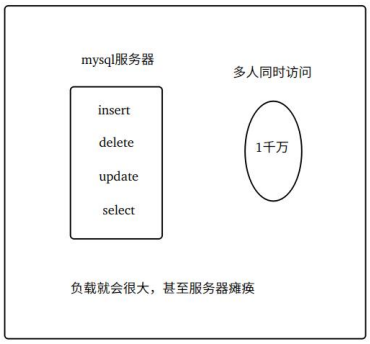
水平切分:将记录散列到不同的表中，各表的结构完全相同，每次从分表中查询, 提高效率。

垂直切分:将表中大字段单独拆分到另外一张表, 形成一对一的关系。

# 读写分离

当系统的并发访问量特别大的时候，单一的MySQL服务器的负载特别大，导致数据库性能下降，升值造成服务器崩溃。这个时候可以考虑搭建MySQL集群，使用读写分离技术来改善这种状况。集群中包含一台master服务器，多台slave服务器，master服务器负责执行DML(insert/delete/update)语句，slave服务器负载执行select语句，主服务器通过日志文件将操作同步到从服务器上。




# 分表技术

当业务数据越来越多的时候，会导致某些数据表的数据量非常巨大，导致系统的性能下降。可以通过分表的方式改善性能。

1. 水平分表

将一张大表中的数据，或者即将产生大量数据的表，按照业务无关的属性随机均匀的存入多张结构相同的分表中。

假设订单信息表，可以创建order_info_00，order_info_01...order_info_99，一共100张分表，订单编号是唯一的，每次存入的时候，用订单编号取hashcode后，再对100(分表数量)取模，得到的结果即为将要存储数据的分表的序号。

Java中可如下操作：

#取哈希值时有可能结果为Integer.MIN_VALUE,导致取序号出错int index = Math.abs(orderNo.hashcode() % 100);

String idxStr= String.format("%2d", index);

String tableName= "order_info_" + idxStr;

2. 垂直分表

在某些表中，可能会有占用空间比较大得字段，类型如text，varchar(3000)，用于存储文章内容，回帖内容等，这些字段会严重影响系统的检索速度，此类字段查询的次数也相对较少，这个时候可以将其提取出来，单独建表存储，与原来的表共用主键id。这样在保证了数据的关联一致的同时，加快了原来表的检索速度。

3. 数据库中文本视频类数据的存储

通常不直接将文本或视频内容存储在数据库中，而只是存储文本或视频所在的路径，查询时按照路径去检索文件的真正内容。(比如微信小程序项目，里边的音源，照片等)

# 优化原则

- 查询时，能不用 **就不用** ，尽量写全字段名

  mybatis中逆向工程里边，生成的tbMusicMapper.xml中。

  ```
  <sql id="Base_Column_List"> music_id, music_name, music_album_name, music_album_pic_url, music_mp3_url, music_artist_name, sheet_id </sql>
  ```

  整个文件里边，没有一个*号

  ```
  select * from user;

  select id,name from user;
  ```

- 大部分情况连接(内连接inner join 左连接 Left，右连接right)效率远大于子查询

  ```
  内连接
  SELECT
      s.sname,
  ```

```
    sc.cno,
    sc.degree
FROM
    student s
INNER JOIN score sc ON s.sno = sc.sno;

子查询
SELECT
    s.sname,
    sc.cno,
    sc.degree
FROM
    student s
where s.sno=(select cno,degree,sno from
score);
```

- 多使用explain和profile分析查询语句

  explain参数说明

  id:选择标识符
  select_type:表示查询的类型。
  table:输出结果集的表
  partitions:匹配的分区
  type:表示表的连接类型
  possible_keys:表示查询时，可能使用的索引
  key:表示实际使用的索引
  key_len:索引字段的长度
  ref:列与索引的比较
  rows:扫描出的行数(估算的行数)

filtered:按表条件过滤的行百分比
Extra:执行情况的描述和说明

```
mysql> explain select * from s1 where id =1;
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | s1    | NULL       | const | PRIMARY       | PRIMARY | 4       | const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.01 sec)

```

- 查看慢查询日志，找出执行时间长的sql语句优化
- 多表连接时，尽量小表驱动大表，即小表 join 大表
- 在千万级分页时使用limit
- 对于经常使用的查询，可以开启缓存