## MySQL设计优化总结（一）

MySQL优化分为以下几个方面

1. 数据表设计
2. SQL查询优化
3. 分库、分表、分区
4. 配置主从复制、读写分离
5. 其他配置

由于总体篇幅过长，本篇先对第1点方法进行总结

本篇内容详情参考《高性能MySQL（第3版）》第4章

**数据表设计**

一、遵循三范式，反三范式

1.第一范式

确保数据表中每列（字段）的原子性。

如果数据表中每个字段都是不可再分的最小数据单元，则满足第一范式。

例如：user用户表，包含字段id,username,password

2.第二范式

在第一范式的基础上更进一步，目标是确保表中的每列都和主键相关。

如果一个关系满足第一范式，并且除了主键之外的其他列，都依赖于该主键，则满足第二范式。

例如：一个用户只有一种角色，而一个角色对应多个用户。则可以按如下方式建立数据表关系，使其满足第二范式。

user用户表，字段id,username,password,role\_id

role角色表，字段id,name

用户表通过角色id（role\_id）来关联角色表

3.第三范式

在第二范式的基础上更进一步，目标是确保表中的列都和主键直接相关，而不是间接相关。

例如：一个用户可以对应多个角色，一个角色也可以对应多个用户。则可以按如下方式建立数据表关系，使其满足第三范式。

user用户表，字段id,username,password

role角色表，字段id,name

user\_role用户-角色中间表，id,user\_id,role\_id

像这样，通过第三张表（中间表）来建立用户表和角色表之间的关系，同时又符合范式化的原则，就可以称为第三范式。

4. 反范式化

反范式化指的是通过增加冗余或重复的数据来提高数据库的读性能。

例如：在上例中的user\_role用户-角色中间表增加字段role\_name。

反范式化可以减少关联查询时，join表的次数

1. 选择恰当的数据类型

原则：

1）数据类型更小的更好

2）简单就好（整型比字符操作代价低）

3）尽量避免NULL

具体类型选择：

1. DATETIME和TIMESTAMP列都可以存储相同类型的数据：时间和日期，精确到秒，TIMESTAMP只使用DATETIME一半的存储空间，但是TIMESTAMP允许的时间范围小。datetime类型取值范围：1000-01-0100:00:00 到9999-12-31 23:59:59,timestamp类型取值范围：1970-01-0100:00:00 到2037-12-31 23:59:59，DATETIME使用8个字节存储，与时区无关。

可以使用BIGINT存储微秒级别的时间戳，或者DOUBLE存储秒之后的小数部分

2）整数类型尽量使用存储空间小的，整数类型从小到大如下：tinyint、smallint、mediumint、int、bigint；分别使用8、16、24、32、64位存储空间。整数类型分为有符号的和无符号的（使用属性UNSIGNED），区别就在于存储数据的范围不同，其性能和存储空间一致。MYSQL可以为整数类型指定宽度，例如INT(11)，但是指定长度其实是没有太大意义的，其数据范围、存储内存和计算都没影响。

整型数据类型存储IP地址（IP转数字函数inet\_aton()，数字转IP函数inet\_ntoa()）

3）字符串类型的选优：

1：VARCHAR类型用来存储可变长字符串；适用于：字符串列的最大长度比平均长度大很多，列的更新很少，这时VARCHAR产生的碎片就不是影响性能的问题。需要注意的就是使用VARCHAR时定义长度很重要，不能过度的大方，更长会消费更多内存，所以长度合适最好。

2：CHAR类型用来存储定长字符串；适用于：很短的字符串，或者所有的值都很接近同一长度。例如MD5存储的加密信息。总结：对于经常变更的数据，CHAR比VARCHAR更好，因为定长的CHAR类型不容易产生碎片。对于非常短的列CHAR存储效率更高。

3：二进制字符串 BINARY/VARBINARY 存储的字节码，所以比较时比字符快。

4：BLOB和TEXT分别采用二进制和字符方式存储，对每个列最前max\_sort\_length字节排序。

4）使用枚举（ENUM）替代字符串类型；当使用到字段只有几个不变的选择（数据字典项）时可以使用枚举来存储这几个不变的字典项，可以减少存储，提高查询效率。

5）实数类型，不只为了存储小数部分也可以使用decimal存储比BIGINT还大的整数。如果存储的数据范围超过decimal的范围，建议将数据拆成整数和小数分开存储。精度高意味着需要额外的空间和计算开销，所以尽量在精确计算时才使用decimal。数据量比较大时可以使用BIGINT代替（将小数位数乘以对应倍数）如需要精确到万分之一，则将乘一百万后的结果存储在BIGINT中.

6）位数据类型，因为BIT类型检索场景不同得到的值不同，谨慎使用，一般需要用到是可以使用整型替代，如权限控制中应对多种权限组合时，通过代码定义，并通过位运算得出结果存储。（也可选择set存储，但不推荐）

位存储运算例子：

三、选择合适的存储引擎

（1）MyISAM

MyISAM 引擎是 MySQL 5.1 及之前版本的默认引擎，它的特点是：

1、MyISAM存储引擎的每一个表都被存放为三个以表名命名的物理文件。首先有任何存储引擎都不可缺少的存放表结构定义信息的.frm文件，.MYD(表的数据)和.MYI索引数据文件。每个表都有且仅有这样三个文件作为MyISAM存储类型的表的存储，也就是说不管这个表有多少个索引，都存放在同一个.MYI文件中。

2、不支持行锁，读取时对需要读到的所有表加锁，写入时则对表加排它锁

3、不支持事务

4、不支持外键

5、不支持崩溃后的安全恢复

6、在表有读取查询的同时，支持往表中插入新纪录

7、支持BLOB和TEXT的前500个字符索引，支持全文索引

8、支持延迟更新索引，极大提升写入性能

9、对于不会进行修改的表，支持压缩表，极大减少磁盘空间占用

在以下场合下，使用MyISAM是最理想的选择

1：选择密集型的表。MyISAM存储引擎在筛选大量数据时非常迅速，这是它最突出的优点。

2：插入密集型的表。MyISAM的并发插入特性允许同时选择和插入数据

（2）InnoDB

InnoDB 在 MySQL 5.5 及以后成为默认索引，它的特点是：

1、支持行锁，采用 MVCC 来支持高并发

2、支持事务

3、支持外键

4、支持崩溃后的安全恢复

5、在线热备份

在以下场合下，使用InnoDB是最理想的选择

1：更新密集的表：InnoDB存储引擎特别适合处理多重并发的更新请求。

2：事务：InnoDB 存储引擎是支持事务的标准MySQL存储引擎。

3：自动灾难恢复：与其它存储引擎不同，InnoDB表能够自动从灾难中恢复。

4：外键约束：MySQL支持外键的存储引擎只有InnoDB。