表结构与数据类型的优化

数据类型优化

▼ 更小的数据类型：能用tinyint 就不用int

因为在分配内存空间的时候是根据类型分配的

▼ 简单就好：简单数字类型的操作通常需要更少的CPU周期

比如数字比字符串操作代价更低，因为字符串相关的规则较为复杂，

比如日期格式用mysql 内建类型存储，不要用字符串存储

比如用整形存储IP地址 ——— 整形可以减少存储空间 但是需要转换（自己权衡）

▼ 尽量避免null：能设置非空就设置非空，能不适用null就不要使用null

因为包含NULL的列，对于mysql来说很难优化，因为允许null的列使得索引、索引统计和值的比较更加复杂。

▼ 整形类型：尽量使用满足需求的最小长度

▼ 字符和字符串的优化：

- varchar 根据实际内容长度保存数据

  1、使用最小符合需求的长度

  2、varchar(n) n 小于等于255 使用额外一个字节保存长度，大于255使用额外两个字节保存长度

  3、varchar(5)与varchar(255)保存同样的内容，硬盘存储空间相同，但是内存占用不同，是指定的大小，因为为了读取效率，操作系统对磁盘做了4k对齐的设置，如果5k的数据就需要2个4k的空间。在读取的时候虽然操作系统的页是4k，读取的内容可以是页的n次倍。提升磁盘和数据的交换效率。

  4、varchar在mysql5.6之前变更长度，或者从255一下变到255以上时，会导致锁表

  5、应用场景：

  a、存储长度波动较大的数据，如：文章，有的会很短，有的会很长

  b、如果长度经常变，更新场景比较少的场景常用varchar

c、适合保存多字节字符，如：汉子，特殊字符等

- char　固定长度字符串

  1、最大长度255

  2、存取/读取的时候会自动删除末尾的空格

  3、检索效率、写效率会比varchar高，空间换时间（因为是固定长度）

  4、应用场景

  a、存储长度波动不大 的数据，如：Md5摘要

  b、存储短字符串和经常更新字符串的场景

▼ BLOB 和TEXT 类型

一般不会用，入股数据太大一般用文件处理，从文件里面取数据

MySQL 把每个 BLOB 和 TEXT 值当作一个独立的对象处理。

两者都是为了存储很大数据而设计的字符串类型，分别采用二进制和字符方式存储。

▼ date、datetime和timestamp

- date

  占用的字节数比使用字符串、datetime、int存储要少，使用date类型只需要3个字节

  使用date类型还可以利用日期时间函数进行日期之间的计算

  date类型用于保存1000-01-01到9999-12-31之间的日期

- datetime

  占用8个字节

  与时区无关，数据库底层时区配置，对datetime无效

  可保存到毫秒

  可保存时间范围大

  不要使用字符串存储日期类型，占用空间大，损失日期类型函数的便捷性

- timestamp　目前用的多的　一般精确到秒

  占用4个字节

  时间范围：1970-01-01到2038-01-19

精确到秒

采用整形存储

依赖数据库设置的时区

自动更新timestamp列的值

▼ 使用枚举类型替代字符串

有时可以使用枚举类代替常用的字符串类型，mysql存储枚举类型会非常紧凑，会根据列表值的数据压缩到一个或两个字节中，mysql在内部会将每个值在列表中的位置保存为整数，并且在表的.frm文件中保存“数字-字符串”映射关系的查找表

create table enum_test(e enum('fish','apple','dog') not null);

insert into enum_test(e) values('fish'),('dog'),('apple');

select e+0 from enum_test;

select * from enum_test where e = 2 // apple

▼ 特殊类型数据

人们经常使用varchar(15)来存储ip地址，然而，它的本质是32位无符号整数不是字符串，可以使用INET_ATON()和INET_NTOA函数在这两种表示方法之间转换

案例：

select inet_aton('1.1.1.1')

select inet_ntoa(16843009)

合理使用范式和反范式

▼ 范式

- 优点

  范式化的更新通常比反范式要快

  数据冗余少

  范式化的数据比较小，可以放在内存中，操作比较快

- 缺点

  通常需要进行关联操作

▼ 反范式

- 缺点

  所有的数据都在同一张表中，可以避免关联

  可以设计有效的索引

- 缺点

  表格内的冗余较多，删除数据时候会造成表有些有用的信息丢失

▼ 注意的点

- 范式和反范式搭配使用

  a、合理搭配可以高效的获取数据

  b、另一个从父表冗余一些数据到子表的理由是排序的需要

  c、缓存衍生值也是有用的。如果需要显示每个用户发了多少消息（类似论坛的），可以每次执行一个昂贵的自查询来计算并显示它；也可以在user表中建一个num_messages列，每当用户发新消息时更新这个值。

- 案例

  范式设计：


```
SELECT b.用户名,b.电话,b.地址,a.订单ID
        ,SUM(c.商品价价*c.商品数量) as 订单价格
FROM `订单表` a
JOIN `用户表` b ON a.用户ID=b.用户ID
JOIN `订单商品表` c ON c.订单ID=b.订单ID
GROUP BY b.用户名,b.电话,b.地址,a.订单ID
```

  反范式设计：

反范式化查询订单信息：

```
SELECT a.用户名,a.电话,a.地址
,a.订单ID
,a.订单价格
FROM `订单表` a
```

主键的选择

▼ 自然主键

就是充当主键的字段本身具有一定的含义，是构成记录的组成部分，比如学生的学号，除了充当主键之外，同时也是学生记录的重要组成部分

▼ 代理主键

就是充当主键的字段本身不具有业务意义，只具有主键作用，比如自动增长的ID等

▼ 推荐使用代理主键

a、它们不与业务耦合，因此更容易维护

b、一个大多数表，最好是全部表，通用的键策略能够减少需要编写的源码数量，减少系统的总体拥有成本

字符集的选择

- 纯拉丁字符能表示的内容，没必要选择 latin1 之外的其他字符编码，因为这会节省大量的存储空间。
- 如果我们可以确定不需要存放多种语言，就没必要非得使用UTF8或者其他UNICODE字符类型，这回造成大量的存储空间浪费。

- MySQL的数据类型可以精确到字段，所以当我们需要大型数据库中存放多字节数据的时候，可以通过对不同表不同字段使用不同的数据类型来较大程度减小数据存储量，进而降低 IO 操作次数并提高缓存命中率。

存储引擎的选择

搜索引擎对比

	MyISAM	InnoDB
索引类型	非聚簇索引	聚簇索引
支持事务	否	是
支持表锁	是	是
支持行锁	否	是
支持外键	否	是
支持全文索引	是	是（5.6后支持）
适合操作类型	大量select	大量insert、delete、update

其它的搜索引擎都不支持数据持久化，如memory搜索引擎等

非聚簇索引——数据文件和索引文件不放一起

聚簇索引—— 两个文件放一起

存储引擎不同，对应的数据文件组织形式也不同

适当的数据冗余

- 被频繁引用且只能通过 Join 2张(或者更多)大表的方式才能得到的独立小字段。

  物化视图：oracle 有，mysql没有

- 这样的场景由于每次Join仅仅只是为了取得某个小字段的值，Join到的记录又大，会造成大量不必要的 IO，完全可以通过空间换取时间的方式来优化。不过，冗余的同时需要确保数据的一致性不会遭到破坏，确保更新的同时冗余字段也被更新。

适当拆分

- 当我们的表中存在类似于 TEXT 或者是很大的 VARCHAR类型的大字段的时候，如果我们大部分访问这张表的时候都不需要这个字段，我们就该义无反顾的将其拆分到另外的独立表中，以减少常用数据所占用的存储空间。这样做的一个明显好处就是每个数据块中可以存储的数据条数可以大大增加，既减少物理 IO 次数，也能大大提高内存中的缓存命中率。