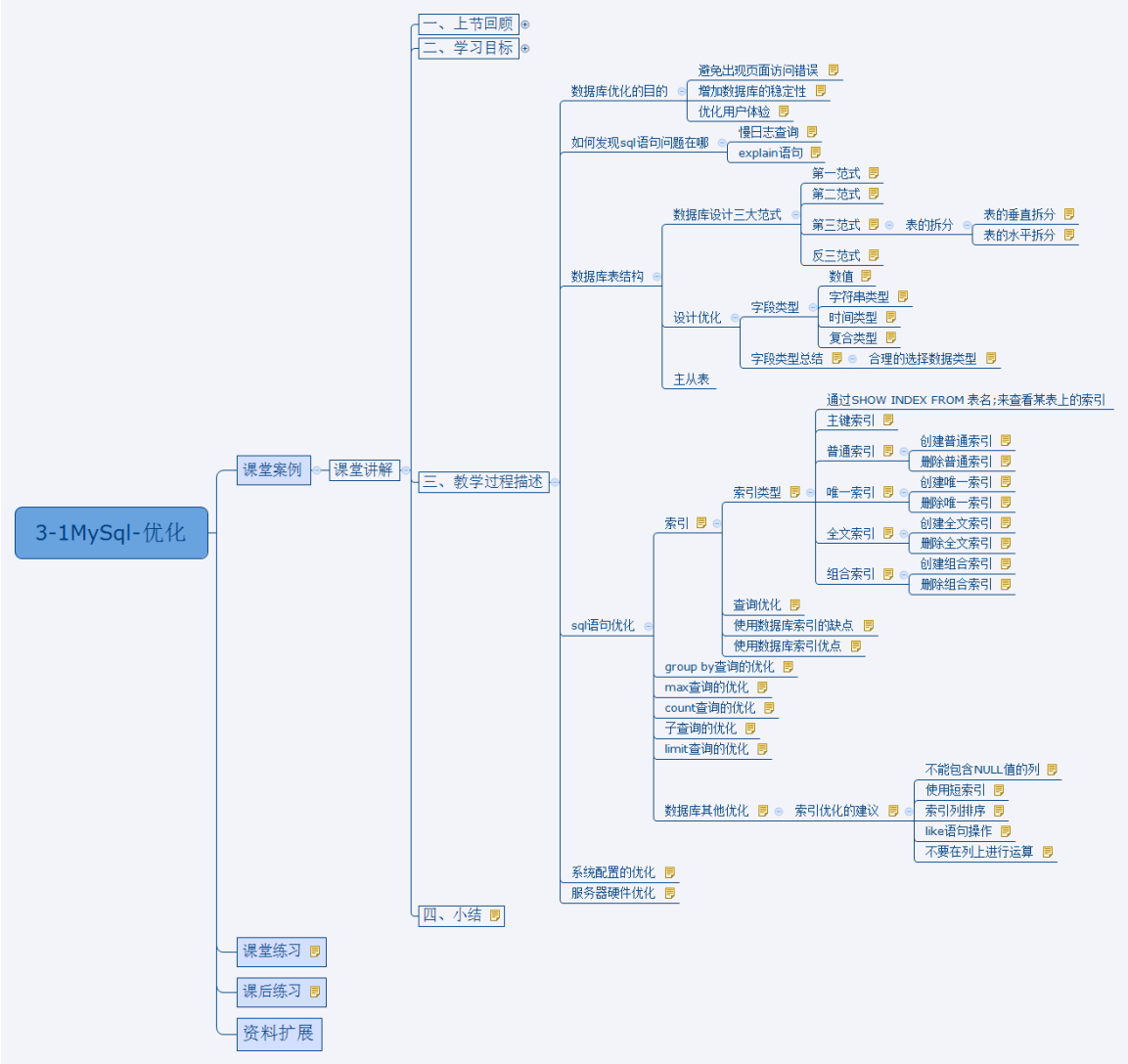
3-1MySql-优化



1. 课堂案例

课堂讲解

一、上节回顾

第二阶段

1、我们学习了如何做一个简单的cms系统

2、我们了解了ajax的使用

3、我们学习了数据库的增删改查

4、我们如何去考虑一个功能的开发

二、学习目标

学习如何更好的设计数据库

三、教学过程描述

数据库优化的目的

避免出现页面访问错误

由于数据库连接timeout产生页面5xx错误

由于慢查询造成数据无法提交

由于阻塞造成数据无法提交

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

优化用户体验

流畅页面的访问速度

良好的网站功能体验

如何发现sql语句问题在哪

慢日志查询

通过慢日志分析哪些sql语句是很占用时间的需要去优化的，操作步骤如下：

```
<!-- 查看是否开启了慢查询日志 -->
mysql> show variables like 'show_query_log';

<!-- 查看没有索引的查询是否也记录了 -->
mysql> show variables like '%log%';

<!-- 设置没有索引的查询也记录下来 -->
mysql> set global log_queries_not_using_indexes=on;

<!-- 超过设置的值，则认为是慢查询并被记录下来 -->
mysql> show variables like 'long_query_time';

<!-- 慢查询日志开启 -->
mysql> set global slow_query_log=on;

<!-- 查看慢查询日志存放的位置 -->
mysql> show variables like 'show%';
```

explain语句

1. explain:分析查询语句的效率

2. select_type:查询类型

SIMPLE:最简单的sql语句，最常见。大部分有效率的语句，类型都是SIMPLE

PRIMARY:查询的数据表是一个动态表，数据表是有一条sql语句生成的结果集构建出的临时表，该表无法使用索引，如下

select * from (select * from news)n，通过n得到结果集来查询，那n就是动态表

UNION:使用union关键字，用于合并两个或多个SELECT语句的结果集(一般少用)

SUBQUERY:子句查询，条件使用动态结果集，如下:

explain select * from t0 where id = (select id from t0 where id = 90)

DEPENDENT SUBQUERY:字句查询，关键字使用in

3、type:查询类型，包括语句使用索引的情况(以下排序速度从上到下)

system:理论上最快的，使用了主键，全表只有1行

const:在实际中最快的类型，使用了主键索引，而且返回的行数只有1行

eq_ref:使用了唯一索引，而且返回1行

ref:使用了普通索引，返回少数的行(where条件使用的是=and等关键字),但是最终还要看rows的数量

ref_or_null:同上，但是字段允许为空

4、key与possible_keys

key：sql语句使用的索引

possible_keys：sql语句可能或可以使用的索引->若存在可能使用的索引，而且其他的索引比现在所使用索引更有效率，可以通过FORCEINDEX(索引名)强制使用索引

key_len：索引涉及字段（组合）每行的长度（大小）（例如字段是int，长度就是4）

rows：sql语句查询的行数，该数字越少表示语句越有效率

extra：在查询中额外使用的工具或状态（有好有不好）

Usingindex：仅使用索引完成，没有使用实体表，效率极高。

注意下面语句select中字段的写法

explain select id from t0 where id < 1000

Usingwhere：使用条件（where，limit）限制了返回条数

Usingfilesort：必须优化，表示在查询中使用数据库的排序工具，效率极差。通过出现在排序的字段没有使用索引Usingtemporary：必须优化，语句运行使用了临时表，通常出现在关联查询的排序中，并且很多时候与Usingfilesort一起出现。有时会出现在groupby中

extra：在查询中额外使用的工具或状态（有好有不好）

Usingindex：仅使用索引完成，没有使用实体表，效率极高。

数据库表结构

数据库设计三大范式

第一范式

第一范式：即表的列的具有原子性,不可再分解，即列的信息，不能分解，要求表的每个字段必须是不可分割的独立单元

例如:顾客表(姓名、编号、地址、……)其中"地址"列还可以细分为国家、省、市、区等。

第二范式

在第一范式的基础之上，要求每张表只表达一个意思。表的每个字段都和主键有依赖关系

例如:订单表(订单编号、产品编号、定购日期、价格、……)，"订单编号"为主键，"产品编号"和主键列没有直接的关系，即"产品编号"列不依赖于主键列，应删除该列。

第三范式

在第二范式基础上，要求每张表除主键之外的其他字段都只能主键有直接决定依赖关系

如果一个关系满足第二范式,并且除了主键以外的其它列都不依赖于主键列,则满足第三范式.
例如:订单表(订单编号, 定购日期, 顾客编号, 顾客姓名, ……), 初看该表没有问题, 满足第二范式, 每列都和主键列"订单编号"相关, 再细看你会发现"顾客姓名"和"顾客编号"相关, "顾客编号"和"订单编号"又相关, 最后经过传递依赖, "顾客姓名"也和"订单编号"相关。为了满足第三范式, 应去掉"顾客姓名"列, 放入客户表中。

表的拆分

表的垂直拆分

所谓的垂直拆分, 就是把原来一个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

表的水平拆分

表的水平拆分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为:

1、对customer_id进行hash运算, 如果要拆分成5个表则使用mod(customer_id,5)取出0-4个值

2、针对不同的hashID把数据存到不同的表中

挑战:

1、跨分区表进行数据查询

2、统计及后台报表操作

前后台查询业务的分开, 前台用户查询拆分后的表, 后台用户查询用汇总后的表

反三范式

没有冗余的数据库未必是最好的数据库, 有时为了提高运行效率, 就必须降低范式标准, 适当保留冗余数据。具体做法是:

在概念数据模型设计时遵守第三范式, 降低范式标准的工作放到物理数据模型设计时考虑。降低范式就是增加字段, 减少了查询时的关联, 提高查询效率, 因为在数据库的操作中查询的比例要远远大于DML的比例。但是反范式化一定要适度, 并且在原本已满足三范式的基础上再做调整的。

设计优化

字段类型

数值

MySQL 的数值数据类型可以大致划分为两个类别，一个是整数，另一个是浮点数或小数。

许多不同的子类型对这些类别中的每一个都是可用的，每个子类型支持不同大小的数据，并且 MySQL

允许我们指定数值字段中的值是否有正负之分(UNSIGNED)或者用零填补(ZEROFILL)。

1bit 位　　1 字节=8bit　　1k=1024 字节　　1 兆=1024k　　1G=1023M　　1T=1024G

类型	大小	范围（有符号）	范围（UNSIGNED）	用途
TINYINT	1 字节	(-128，127)	(0，255)	小整数值
SMALLINT	2 字节	(-32 768，32 767)	(0，65 535)	大整数值
MEDIUMINT	3 字节	(-8 388 608，8 388 607)	(0，16 777 215)	大整数值
INT 或 INTEGER	4 字节	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8 字节	(-9 233 372 036 854 775 808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	极大整数值
FLOAT	4 字节	(-3.402 823 466 E+38，1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38)	0，(1.175 494 351 E-38，3.402 823 466 E+38)	单精度浮点数值
DOUBLE	8 字节	(1.797 693 134 862 315 7 E+308，2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度浮点数值
DECIMAL	对 DECIMAL(M,D) ，如果 M>D，为 M+2 否则为 D+2	依赖于 M 和 D 的值	依赖于 M 和 D 的值	小数值

INT

在 MySQL 中支持的 5 个主要整数类型是 TINYINT，SMALLINT，MEDIUMINT，INT 和 BIGINT。这些类型在很大程度上是相同的，只有它们存储的值的大小是不相同的。

MySQL 以一个可选的显示宽度指示器的形式对 SQL 标准进行扩展(如 INT(6),6即是其宽度指示器,该宽度指示器并不会影响int列存储字段的大小,也就是说,超过6位它不会自动截取,依然会存储,只有超过它本身的存储范围才会截取;此处宽度指示器的作用在于该字段是否有zerofill,如果有就未满足6位的部分就会用0来填充)，这样当从数据库检索一个值时，可以把这个值加长到指定的长度。例如，指定一个字段的类型为 INT(6)，就可以保证所包含数字少于 6 个的值从数据库中检索出来时能够自动地用空格填充。需要注意的是，使用一个宽度指示器不会影响字段的大小和它可以存储的值的范围。

万一我们需要对一个字段存储一个超出许可范围的数字，MySQL
会根据允许范围最接近它的一端截短后再进行存储。还有一个比较特别的地方是，MySQL
会在不合规定的值插入表前自动修改为 0。
unsigned 和 zerofil
UNSIGNED
修饰符规定字段只保存正值,即无符号,而mysql字段默认是有符号的。因为不需要保存数字的正、负符号，可以在储时节约一个"位"的空间(即翻一倍)。从而增大这个字段可以存储的值的范围。
注意这个修饰符要紧跟在数值类型后面;
ZEROFILL 修饰符规定 0（不是空格）可以用来真补输出的值。使用这个修饰符可以阻止 MySQL 数据库存储负值,如果某列设置为zerofill,那它自动就unsigned。这个值要配合int,tinyint,smallint,midiumint等字段的宽度指示器来用;XXint(M),如果没有zerofill,这个M的宽度指示器是没有意义的.(注意,测试前导0的时候,还是去黑窗口测试;)
为什么mysql存储的值要分有符号和无符号呢?因为一个字节,占8bit;也就1个bit有0和1两种可能,8个bit就是2^8 = 256种可能,也就是0~255;但如果是有符号的话,就得拿一个1bit来存储这个负号,本来8bit只剩7bit,2^7 = 128,也就是-128~127(正数部分包含一个0);
FLOAT、DOUBLE 和 DECIMAL 类型
MySQL 支持的三个浮点类型是 FLOAT、DOUBLE 和 DECIMAL 类型。FLOAT 数值类型用于表示单精度浮点数值，而 DOUBLE 数值类型用于表示双精度浮点数值。
与整数一样，这些类型也带有附加参数：一个显示宽度指示器和一个小数点指示器(必须要带有指示器,要不然会查不到结果,并且宽度指示器和XXint类型的宽度指示器不同,这里是有实际限制宽度的)。比如语句 FLOAT(7,3) 规定显示的值不会超过 7 位数字(包括小数位)，小数点后面带有 3 位数字。对于小数点后面的位数超过允许范围的值，MySQL 会自动将它四舍五入为最接近它的值，再插入它。
DECIMAL
数据类型用于精度要求非常高的计算中，这种类型允许指定数值的精度和计数方法作为选择参数。精度在这里指为这个值保存的有效数字的总个数，而计数方法表示小数点后数字的位数。比如语句 DECIMAL(7,3) 规定了存储的值不会超过 7 位数字，并且小数点后不超过 3 位。
FLOAT 类型在长度比较高比如 float(10,2)和 decimal(10,2)同时插入一个符合(10,2)宽度的数值,float 就会出现最后小数点出现一些出入;
UNSIGNED 和 ZEROFILL 修饰符也可以被 FLOAT、DOUBLE 和 DECIMAL 数据类型使用。并且效果与 INT 数据类型相同。
关于flaot和double
在这里我建议,干脆忘记mysql有double这个数据类型。至于why？就不要管它了

字符串类型

MySQL 提供了 8 个基本的字符串类型，可以存储的范围从简单的一个字符到巨大的文本块或二进制字符串数据。

1 英文字符 占用 1 字节

1 个中文字符 占用 2 个字节

类型	大小	用途
CHAR	0-255 字节	定长字符串
VARCHAR	0-255 字节	变长字符串
TINYBLOB	0-255 字节	不超过 255 个字符的二进制字符串
TINYTEXT	0-255 字节	短文本字符串
BLOB	0-65 535 字节=64k	二进制形式的长文本数据
TEXT	0-65 535 字节	长文本数据
MEDIUMBLOB	0-16 777 215 字节=16M	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215 字节	中等长度文本数据
LOGNGBLOB	0-4 294 967 295 字节=4G	二进制形式的极大文本数据
LONGTEXT	0-4 294 967 295 字节	极大文本数据

BINARY

BINARY不是函数，是类型转换运算符，它用来强制它后面的字符串为一个二进制字符串，可以理解为在字符串比较的时候区分大小写

CHAR 和 VARCHAR 类型

CHAR

类型用于定长字符串，并且必须在圆括号内用一个大小修饰符来定义。这个大小修饰符的范围从0-255。比指定长度大的值将被截短，而比指定长度小的值将会用空格作填补。

CHAR 类型可以使用 BINARY 修饰符。当用于比较运算时，这个修饰符使 CHAR 以二进制方式参于运算，而不是以传统的区分大小写的方式。

CHAR 类型的一个变体是 VARCHAR 类型。它是一种可变长度的字符串类型，并且也必须带有一个范围在 0-255 之间的指示器。

CHAR 和 VARCHGAR 不同之处在于 MYSQL 数据库处理这个指示器的方式：CHAR 把这个大小视为值的大小，不长度不足的情况下就用空格补足。而 VARCHAR 类型把它视为最大值并且只使用存储字符串实际需要的长度（增加一个额外字节来存储字符串本身的长度）来存储值。所以短于指示器长度的 VARCHAR 类型不会被空格填补，但长于指示器的值仍然会被截短。

因为 VARCHAR 类型可以根据实际内容动态改变存储值的长度，所以在不能确定字段需要多少字符时使用

VARCHAR

类型可以大大地节约磁盘空间、提高存储效率。但如果确切知道字符串长度,比如就在50~55之间,那就用 CHAR 因为 CHAR 类型由于本身定长的特性使其性能要高于 VARCHAR;

VARCHAR 类型在使用 BINARY 修饰符时与 CHAR 类型完全相同。

TEXT 和 BLOB 类型

对于字段长度要求超过 255 个的情况下, MySQL 提供了 TEXT 和 BLOB 两种类型。根据存储数据的大小, 它们都有不同的子类型。这些大型的数据用于存储文本块或图像、声音文件等二进制数据类型。

TEXT 和 BLOB 类型在分类和比较上存在区别。BLOB 类型区分大小写, 而 TEXT 不区分大小写。大小修饰符不用于各种 BLOB 和 TEXT 子类型。比指定类型支持的最大范围大的值将被自动截短。

时间类型

在处理日期和时间类型的值时, MySQL 带有 5 个不同的数据类型可供选择。

DATE、TIME 和 YEAR 类型

MySQL 用 DATE 和 YEAR 类型存储简单的日期值, 使用 TIME 类型存储时间值。这些类型可以描述为字符串或不带分隔符的整数序列。如果描述为字符串, DATE 类型的值应该使用连字号作为分隔符分开, 而 TIME 类型的值应该使用冒号作为分隔符分开。

需要注意的是, 没有冒号分隔符的 TIME 类型值, 将会被 MySQL 理解为持续的时间, 而不是时间戳。

MySQL 还对日期的年份中的两个数字的值, 或是 SQL 语句中为 YEAR 类型输入的两个数字进行最大限度的通译。因为所有 YEAR 类型的值必须用 4 个数字存储。MySQL 试图将 2 个数字的年份转换为 4 个数字的值。把在 00-69 范围内的值转换到 2000-2069 范围内。把 70-99 范围内的值转换到 1970-1979 之内。如果 MySQL 自动转换后的值并不符合我们的需要, 请输入 4 个数字表示的年份。

DATETIME 和 TIMESTAMP 类型

除了日期和时间数据类型, MySQL 还支持 DATETIME 和 TIMESTAMP 这两种混合类型。它们可以把日期和时间作为单个的值进行存储。这两种类型通常用于自动存储包含当前日期和时间的时间戳, 并可在需要执行大量数据库事务和需要建立一个调试和审查用途的审计跟踪的应用程序中发挥良好作用。

如果我们对 TIMESTAMP 类型的字段没有明确赋值, 或是被赋与了 null 值。MySQL 会自动使用系统当前的日期和时间来填充它。

在实际开发中 我们大多是把时间转换成时间戳存储

复合类型

MySQL 还支持两种复合数据类型 ENUM 和 SET，它们扩展了 SQL 规范。虽然这些类型在技术上是字符串类型，但是可以被视为不同的数据类型。一个 ENUM 类型只允许从一个集合中取得一个值；而 SET 类型允许从一个集合中取得任意多个值。

ENUM 类型

ENUM 类型因为只允许在集合中取得一个值，有点类似于单选项。在处理相互排拆的数据时容易让人理解，比如人类的性别。ENUM 类型字段可以从集合中取得一个值或使用 null 值，除此之外的输入将会使 MySQL 在这个字段中插入一个空字符串。另外如果插入值的大小写与集合中值的大小写不匹配，MySQL 会自动使用插入值的大小写转换成与集合中大小写一致的值。

ENUM 类型在系统内部可以存储为数字，并且从 1 开始用数字做索引。一个 ENUM 类型最多可以包含 65536 个元素，其中一个元素被 MySQL 保留，用来存储错误信息，这个错误值用索引 0 或者一个空字符串表示。

MySQL 认为 ENUM 类型集合中出现的值是合法输入，除此之外其它任何输入都将失败。这说明通过搜索包含空字符串或对应数字索引为 0 的行就可以很容易地找到错误记录的位置。

SET 类型

SET 类型与 ENUM 类型相似但不相同。SET 类型可以从预定义的集合中取得任意数量的值。并且与 ENUM 类型相同的是任何试图在 SET 类型字段中插入非预定义的值都会使 MySQL 插入一个空字符串。如果插入一个即有合法的元素又有非法的元素的记录，MySQL 将会保留合法的元素，除去非法的元素。

一个 SET 类型最多可以包含 64 项元素。还去除了重复的元素，所以 SET 类型中不可能包含两个相同的元素。

希望从 SET 类型字段中找出非法的记录只需查找包含空字符串或二进制值为 0 的行。

字段类型总结

1、虽然上面列出了很多字段类型,但最常用也就是 varchar(255),char(255),text,tinyint(4),smallint(6),mediumint,int(11)几种。

2、复合类型我们一般用tinyint,更快的时间更省的空间以及更容易扩展

3、关于手机号，推荐用char(11),char(11)在查询上更有效率，因为手机号是一个活跃字段参与逻辑会很多。

4、一些常用字段举例

姓名：char(20)

价格：DECIMAL(7, 3)

产品序列号：SMALLINT(5) unsigned

文章内容: TEXT

MD5: CHAR(32)

ip: char(15)

time: int(10)

email char(32)

合理的选择数据类型

1、选择合理范围内最小的。我们应该选择最小的数据范围，因为这样可以大大减少磁盘空间及磁盘I/0读写开销，减少内存占用，减少CPU的占用率。

2、选择相对简单的数据类型

数字类型相对字符串类型要简单的多，尤其是在比较运算时，所以我们应该选择最简单的数据类型，比如说在保存时间时，因为PHP可以良好的处理LINUX时间戳所以我们可以将日期存为int(10)要方便、合适、快速的多。

但是，工作中随着项目越做越多，业务逻辑的处理越来越难以后，我发现时间类型还是用时间类型本身的字段类型要好一些，因为mysql有着丰富的时间函数供我使用，方便我完成很多与时间相关的逻辑，比如月排行榜，周排行榜，当日热门，生日多少天等等逻辑

3、不要使用null

为什么这么说呢，因为MYSQL对NULL字段索引优化不佳，增加更多的计算难度，同时在保存与处理NULL类形时，也会做更多的工作，所以从效率上来说，不建议用过多的NULL。有些值他确实有可能没有值，怎么办呢？解决方法是数值弄用整数0，字符串用空来定义默认值即可。

4字符串类型的使用

字符串数据类型是一个万能是数据类型，可以储存数值、字符串、日期等。

保存数值类型最好不要用字符串数据类型，这样存储的空间显然是会更大，而且在排序时字符串的9是大于22的，其实如果进行运算时mysql会将字符串转换为数值类型，大大降低效果，而且这种转换是不会走原有的索引的。

如果明确数据在一个完整的集合中如男，女，那么可以使用set或enum数据类型，这种数据类型在运算及储存时以数值方式操作，所以效率要比字符串更好，同时空间占用更少。

5、VARCHAR与CHAR

VARCHAR是可变长度字符串类型，那么即然长度是可变的就会使用1，2个字节来保存字符的长度，如果长度在255内使用1个字节来保存字符长度，否则使用2个字符来保存长度。由于varchar是根据储存的值来保存数据，所以可以大大节约磁盘空间。

如果数据经常被执行更新操作，由于VARCHAR是根据内容来进行储存的，所以mysql将做更多的工作来完成更新操作，如果新数据长度大于老数据长度一些存储引擎会进行拆分操作处理。同时varchar会完全保留内部所有数据，最典型的说明就是尾部的空格。

CHAR固定长度的字符串保存类型，CHAR会去掉尾部的空格。在数据长度相近时使用char类型比较合适，比如md5加密的密码用户名等。

如果数据经常进行更新修改操作，那么CHAR更好些，因为char长度固定，性能上要快。

6、数值类型的选择

数值数据类型要比字符串执行更快，区间小的数据类型占用空间更少，处理速度更快，如tinyint可比bigint要快的多

选择数据类型时要考虑内容长度，比如是保存毫米单位还是米而选择不同的数值类型

7、整数

整数类型很多比如tinyint、int、smallint、bigint等，那么我们要根据自己需要存储的数据长度决定使用的类型，同时tinyint(10)与tinyint(100)在储存与计算上并无任何差别，区别只是显示层面上，但是我们也要选择适合合适的数据类型长度。可以通过指定zerofill属性查看显示时区别。

8、浮点数与精度数值

浮点数float在储存空间及运行效率上要优于精度数值类型decimal，但float与double会有舍入错误而decimal则可以提供更加准确的小数级精确运算不会有错误产生计算更精确，适用于金融类型数据的存储。

主从表

sql语句优化

索引

□

索引的作用相当于一本书的目录，索引能大大加快查询速度。索引优化和查询优化是相辅相成的。

□

索引优化与查询优化是多年经验积累的结晶，在此无法详述，但仍

然给出几条最基本的准则

□

索引是一个由某字段(组合)建立的排序好的文件

□ 查询优化的本质，其实就是减少查询涉及的行数

1.

只要搜索条件中，包含涉及到索引的字段(组合)，sql语句会自动

的选用索引

2. 语句使用了索引，是查找索引，而非查找数据表

3.

索引字段是字符的话，根据首字母排序，若首字母相同，则按照第

二字母排序，以此类推

4. 每条sql语句，最多执行一个索引

5.

mysql选用索引：首先寻找与搜素条件最匹配的索引，若没有，按

照搜索条件的顺序选用最匹配的索引

6.

建立索引组合时，根据组建索引是字段的顺序的先后来排序，建议

大家把筛选出更多数据的字段排在前面

7.

不要为每个字段都建立索引数量太多。理论上的组合，数据表的字

段数n，索引数2的n次方-

1，还没算上组合排序上的不同。索引能

大大的加快查询速度，同时会大大的降低写入的速度

8. 建立索引的准则：经常被用于搜索的字段（组合）

9. 索引涉及字段的值重复率越高，索引效率越低

索引类型

1. PRIMARY（pk）：主键，在所有类型的索引速度最快。每个数据表最多有一个，涉及到主键的字段（组合）值必须唯一。建议每个数据表都建立一个主键，该主键就是id

2. UNIQUE：唯一索引，速度仅次于主键，涉及到的字段（组合）值必须唯一。但一个数据表中可以存在多个

3. INDEX：普通索引，速度在三者中最慢，但是几乎没有任何限制

4. fulltext：全文索引，主要用于模糊搜索。表类型必须是myisam，涉及字段类型必须是字符型

5. spaital：地理信息索引，主要用于地理信息搜索。涉及字段类型必

须是地理信息型

通过SHOW INDEX FROM 表名;来查看某表上的索引

主键索引

主键索引

数据库表经常有一列或列组合，其值唯一标识表中的每一行。该列称为表的主键。
在数据库关系图中为表定义主键将自动创建主键索引，主键索引是唯一索引的特定类型。该索引要求主键中的每个值都唯一。当在查询中使用主键索引时，它还允许对数据的快速访问。

普通索引

这是最基本的索引，它没有任何限制。

创建普通索引

创建索引的语法：

```
CREATE INDEX 索引名 ON 表名 (列名[, 列名]...);
```

```
-- 测试数据
CREATE TABLE e_user(
  id int  NOT NULL PRIMARY KEY AUTO_INCREMENT,
  t_name varchar(10) DEFAULT NULL,
  t_job varchar(50) DEFAULT NULL
)
```

```
-- 创建索引
CREATE INDEX t_job_index ON e_user (t_job);
```

删除普通索引

```
--删除索引
DROP INDEX 索引名 ON 表名;
```

或者

```
ALTER TABLE 表名 DROP INDEX 索引名;
```

唯一索引

唯一索引

唯一索引是不允许其中任何两行具有相同索引值的索引。

当现有数据中存在重复的键值时，大多数数据库不允许将新创建的唯一索引与表一起保存。数据库还可能防止添加将在表中创建重复键值的新数据。例如，如果在employee表中职员的姓(lname)上创建了唯一索引，则任何两个员工都不能同姓。

创建唯一索引

添加唯一约束条件时，创建一个唯一索引。

```
CREATE UNIQUE INDEX 索引名 ON 表名(列名);
```

删除唯一索引

```
--删除索引
DROP INDEX 索引名 on 表名;
或者
ALTER TABLE 表名 DROP INDEX 索引名;
```

全文索引

1.FULLTEXT索引仅可用于 MyISAM 存储引擎；

2.对于较大的数据集，将你的资料输入一个没有FULLTEXT索引的表中，然后创建索引，其速度比把资料输入现有FULLTEXT索引的速度更为快；

切记：对于大容量的数据表，生成全文索引是一个非常消耗时间和硬盘空间的做法；

创建全文索引

```
 -- FULLTEXT 索引仅可用于 MyISAM表；


--建表时创建全文索引
CREATE TABLE e_user(
  id int  NOT NULL PRIMARY KEY AUTO_INCREMENT,
  t_name varchar(10) DEFAULT NULL
    FULLTEXT(t_name) # 创建全文索引
)engine = MyISAM;



--建表后创建全文索引
ALTER TABLE 表名 ADD FULLTEXT 索引名(列名);
或
```

CREATE FULLTEXT INDEX 索引名 ON 表名(列名)

删除全文索引

--删除全文索引

DROP INDEX 索引名 ON 表名;

或者

ALTER TABLE 表名 DROP INDEX 索引名;

组合索引

一般情况下，执行SQL查询语句都有比较多的限制条件，所以为了进一步提高MySQL的运行效率，就要考虑建立对多列建立索引。

创建组合索引

--创建组合索引

ALTER TABLE 表名 ADD INDEX 索引名(列名1(长度1),列名2(长度2),...);

删除组合索引

--删除组合索引

DROP INDEX 索引名 ON 表名;

或者

ALTER TABLE 表名 DROP INDEX 索引名;

查询优化

1. 任何sql语句的where条件，都必须涉及到索引。而且最好是主键，
其次是唯一，再次是普通（通常是先做好sql语句，最好才根据sql
建立索引）
2. 注意查询语句中的一些关键字的使用'=and' > '< > or likein‘ >
'!=notin not like'，因为越往后的涉及行数就会越多
3. 尽量避免使用leftjoin（关联查询）、groupby等的关键字
4. 在复杂的查询语句，查询语句的速度取决最慢的一条
5. 涉及到groupby，orderby的字段必须设置索引
6. (mysql)设置的查询条件，必须先排刷选条件多的字段，依次往后越
简单的语句通常效率就越高，不要为了完成某些功能而去把sql
7. 写得过于复杂，宁愿写多几条简单的

使用数据库索引的缺点

1）占用大量的磁盘空间

2)创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

3)当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

使用数据库索引优点

1)在数据库中用来加速对表的查询；

2)通过使用快速路径访问方法快速定位数据,减少了磁盘的I/O；

3)与表独立存放，但不能独立存在，必须属于某个表；

4)由数据库自动维护，表被删除时，该表上的索引自动被删除；

5)索引的作用类似于书的目录，几乎没有一本书没有目录，因此几乎没有一张表没有索引；

group by查询的优化

1. select category_id from news group by category_id

2. 在sql中，group by必须写在select的字段中，而且要对应好，不要加上多余字段

3. group by的字段可以有多个，把字段组合的所有全部输出

4. having与where，在一般情况下，where与having可以互动，但having还可以实现对group by的约束，如下语句 select category_id,count(category_id) from news group by category_id having count(category_id) >2

5. group by的很多功能都可以让php来实现

max查询的优化

优化就是建立一个索引

```
mysql> explain select max(payment_date) from payment \G;
```

优化就是建立一个索引

```
mysql> create index idx_paydate on payment(payment_date);
```

再次查看该条sql语句

```
mysql> explain select max(payment_date) from payment \G;
```

count查询的优化

```
select count(release_year='2006' or null) as '2006年电影数量',count(release_year='2007' or null) as '2007年电影数量' from film;
```

子查询的优化

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

mysql> select * from emp where deptno in(select deptno from dept);

优化为join 查询

mysql> select emp.* from emp join dept on emp.deptno=dept.deptno;

//如果查询的是一个字段，那么可以用distinct来去重

limit查询的优化

limit常用语分页处理，时常会伴随order_by从句使用，因此大多时候会使用filesorts，这样会造成大量的IO问题

mysql> select film_id,description from sakila.film order by title limit 50,5;

优化步骤1：使用有索引的列或主键进行order by 操作

select film_id,description from sakila.film order by film_id limit 50,5;

优化步骤2：记录上次返回的主键，在下次查询时使用主键过滤

select film_id,description from sakila.film where film_id>55 and film_id<=60 order by film_id limit 1,5;

**注意：要注意的id必须是连续的，如果不连续会造成查出来的数据有些没有那么多

数据库其他优化

1. 尽量不要使用rand(),curdata()等数据库自带函数，因为即使是得到值是一样，但是sql不会缓存相关语句的结果

2. 对于涉及行数过多的sql语句，可以使用limit关键限制行数，因为行数越少，查询越快，而且过多行数其实没有实际应用意义

3. 不要使用order by rand()，能在php实现的效果，不要放在mysql做，如使用array_rand让数据随机排

4. 避免使用select*，会出现搜索出多余字段的情况，影响效率。从效率上来讲，只写上需要的字段，但是会影响程序灵活性

5. 把IP地址保存为数字，使用方法ip2long及long2ip，该数字需要保存负数

6. 垂直分割：分表，防止数据冗余

7. 水平分割：数据表数据量越大，查询就越慢。因此可以把一些数据量非常大的数据，通过一些规则(例如是学号的开头)，变成N个表来保存

8. 拆分复杂的语句，或者批量删除与插入=>宁愿拆分几句来写，不要

写一句复杂

9. 不要使用永久连接

索引优化的建议

上面都在说使用索引的好处，但过多的使用索引将会造成滥用。因此索引也会有它的缺点：

1）虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行INSERT、UPDATE和DELETE。因为更新表时，MySQL不仅要保存数据，还要保存一下索引文件。

2）建立索引会占用磁盘空间的索引文件。一般情况这个问题不太严重，但如果你在一个大表上创建了多种组合索引，索引文件的会膨胀很快。索引只是提高效率的一个因素，如果你的MySQL有大数据量的表，就需要花时间研究建立最优秀的索引，或优化查询语句。

1、选择索引的数据类型

MySQL支持很多数据类型，选择合适的数据类型存储数据对性能有很大的影响。通常来说，可以遵循以下一些指导原则：

(1)越小的数据类型通常更好：越小的数据类型通常在磁盘、内存和CPU缓存中都需要更少的空间，处理起来更快。

(2)简单的数据类型更好：整型数据比起字符，处理开销更小，因为字符串的比较更复杂。在MySQL中，应该用内置的日期和时间数据类型，而不是用字符串来存储时间；以及用整型数据类型存储IP地址。

(3)尽量避免NULL：应该指定列为NOT NULL，除非你想存储NULL。在MySQL中，含有空值的列很难进行查询优化，因为它们使得索引、索引的统计信息以及比较运算更加复杂。你应该用0、一个特殊的值或者一个空串代替空值。

不能包含NULL值的列

1.包含NULL值的列都不会被包含在索引中

2.复合索引中只要有一列含有NULL值，那么这一列对于此复合索引就是无效的。

所以，我们在设计数据库时候，尽量不要设置字段的默认值为NULL。

使用短索引

1.对串列进行索引，如果可能，应该指定一个前缀长度。

例如，如果有一个CHAR(255)的列，如果在前10个或20个字符内，多数值是惟一的，那么就不要对整个列进行索引。

2.短索引的好处：不仅可以提高查询速度，而且可以节省磁盘空间和I/O操作。

索引列排序

1.MySQL查询只使用一个索引，如果where子句中已经使用了索引的话，那么order by中的列是不会使用索引的。因此数据库默认排序可以符合要求的情况下尽量不要使用排序操作。

2.尽量不要包含多个列的排序，如果需要最好给这些列创建复合索引。

like语句操作

一般情况下不鼓励使用like操作，如果非使用不可，如何使用也是有技巧的。

例如：

```
-- 不会使用索引
like '%aaa%'
like '_aaa%'

--可以使用索引
like “aaa%”
```

不要在列上进行运算

如果执行语句：

```
select * from e_user where YEAR(t_birthday) < 1980
```

，将在每个行上进行运算，这将导致索引失效。

因此，我们可以改成：select * from e_user where t_birthday< '1980-09-22';

系统配置的优化

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支出的队列数
   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

服务器硬件优化

如何选择CPU

思考：是选择单核更快的CPU还是选择核数更多的CPU？

1、MySQL有一些工作职能使用到单核CPU

Replicate,SQL……

2、MySQL对CPU核数的支持并不是越多越快

MySQL5.5使用的服务器不要超过32核

Disk IO优化

常用RAID级别简介

RAID0：也称为条带，就是把多个磁盘链接成一个硬盘使用，这个级别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

1、常用于高可用解决方案

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

四、小结

1、创建表合理的使用字段类型

2、严格按照数据库设计的三大范式

3、执行mysql的时候避免一些不好的语句使用，最好不要一条负责的语句处理

2. 课堂练习

1、掌握三大范式

2、掌握字段类型

3. 课后练习

优化第二阶段的项目 数据库

4. 资料扩展