🤔👀MySQL优化01

1. 优化你的MySQL查询缓存

在MySQL服务器上进行查询，可以启用高速查询缓存。让数据库引擎在后台悄悄的处理是提高性能的最有效方法之一。当同一个查询被执行多次时，如果结果是从缓存中提取，那是相当快的。
但主要的问题是，它是那么容易被隐藏起来以至于我们大多数程序员会忽略它。在有些处理任务中，我们实际上是可以阻止查询缓存工作的

```
// query cache does NOT work
$r = mysql_query("SELECT username FROM user WHERE signup_date >=
CURDATE()");
// query cache works!
$today = date("Y-m-d");
$r = mysql_query("SELECT username FROM user WHERE signup_date >=
'$today'");
```

2.用EXPLAIN使你的SELECT查询更加清晰

使用EXPLAIN关键字是另一个MySQL优化技巧，可以让你了解MySQL正在进行什么样的查询操作，这可以帮助你发现瓶颈的所在，并显示出查询或表结构在哪里出了问题。

EXPLAIN查询的结果，可以告诉你哪些索引正在被引用，表是如何被扫描和排序的等等。
实现一个SELECT查询（最好是比较复杂的一个，带joins方式的），在里面添加上你的关键词解释，在这里我们可以使用phpMyAdmin，他会告诉你表中的结果。举例来说，假如当我在执行joins时，正忘记往一个索引中添加列，EXPLAIN能帮助我找到问题的所在。

3.利用LIMIT 1取得唯一行 有时，当你要查询一张表时，你知道自己只需要看一行。你可能会去得一条十分独特的记录，或者只是刚好检查了任何存在的记录数，他们都满足了你的WHERE子句。在这种情况下，增加一个LIMIT 1会令你的查询更加有效。这样数据库引擎发现只有1后将停止扫描，而不是去扫描整个表或索引。

```
// do I have any users from Alabama?
// what NOT to do:
$r = mysql_query("SELECT * FROM user WHERE state = 'Alabama'");
if (mysql_num_rows($r) > 0) {
// ...
}
// much better:
$r = mysql_query("SELECT 1 FROM user WHERE state = 'Alabama' LIMIT 1");
if (mysql_num_rows($r) > 0) {
// ...
}
```

4.索引中的检索字段

索引不仅是主键或唯一键。如果你想搜索表中的任何列，你应该一直指向索引。

5.保证连接的索引是相同的类型

如果应用程序中包含多个连接查询，你需要确保你链接的列在两边的表上都被索引。这会影响MySQL如何优化内部联接操作。此外，加入的列，必须是同一类型。例如，你加入一个DECIMAL列，而同时加入另一个表中的int列，mysql将无法使用其中至少一个指标。即使字符编码必须同为字符串类型。

```
/ looking for companies in my state
$r = mysql_query("SELECT company_name FROM users
LEFT JOIN companies ON (users.state = companies.state)
WHERE users.id = $user_id");
// both state columns should be indexed
// and they both should be the same type and character encoding
// or MySQL might do full table scans
```

6.不要使用BY RAND()命令

这是一个令很多新手程序员会掉进去的陷阱。你可能不知不觉中制造了一个可怕的瓶颈。这个陷阱在你使用BY RAND（）命令时就开始创建了。如果您真的需要随机显示你的结果，有很多更好的途径去实现。诚然这需要写更多的代码，但是能避免性能瓶颈的出现。问题在于，MySQL可能会为表中每一个独立的行执行BY RAND()命令（这会消耗处理器的处理能力），然后给你仅仅返回一行。

```
// what NOT to do:
$r = mysql_query("SELECT username FROM user ORDER BY RAND() LIMIT 1");
// much better:
$r = mysql_query("SELECT count(*) FROM user");
$d = mysql_fetch_row($r);
$rand = mt_rand(0,$d[0] - 1);
$r = mysql_query("SELECT username FROM user LIMIT $rand, 1");
```

7.尽量避免SELECT *命令

从表中读取越多的数据，查询会变得更慢。他增加了磁盘需要操作的时间，还是在数据库服务器与WEB服务器是独立分开的情况下。你将会经历非常漫长的网络延迟，仅仅是因为数据不必要的在服务器之间传输。
始终指定你需要的列，这是一个非常良好的习惯。

```
// not preferred
$r = mysql_query("SELECT * FROM user WHERE user_id = 1");
$d = mysql_fetch_assoc($r);
echo "Welcome {$d['username']}";
// better:
$r = mysql_query("SELECT username FROM user WHERE user_id = 1");
$d = mysql_fetch_assoc($r);
echo "Welcome {$d['username']}";
// the differences are more significant with bigger result sets
```

8.从PROCEDURE ANALYSE()中获得建议

PROCEDURE ANALYSE()可让MySQL的柱结构分析和表中的实际数据来给你一些建议。如果你的表中已经存在实际数据了，能为你的重大决策服务。

9.准备好的语句

准备好的语句，可以从性能优化和安全两方面对大家有所帮助。
准备好的语句在过滤已经绑定的变量默认情况下，能给应用程序以有效的保护，防止SQL注入攻击。当然你也可以手动过滤，不过由于大多数程序员健忘的性格，很难达到效果。

```
// create a prepared statement
if ($stmt = $mysqli->prepare("SELECT username FROM user WHERE state=?"))
{
// bind parameters
$stmt->bind_param("s", $state);
// execute
$stmt->execute();
// bind result variables
$stmt->bind_result($username);
// fetch value
$stmt->fetch();
printf("%s is from %s\n", $username, $state);
$stmt->close();
}
```

10.将IP地址存储为无符号整型

许多程序员在创建一个VARCHAR（15）时并没有意识到他们可以将IP地址以整数形式来存储。当你用一个INT类型时，你只占用4个字节的空间，这是一个固定大小的领域。
你必须确定你所操作的列是一个UNSIGNED INT类型的,因为IP地址将使用32位unsigned integer。

```
$r = "UPDATE users SET ip = INET_ATON('{$_SERVER['REMOTE_ADDR']}') WHERE
user_id = $user_id";
```

11.永远为每张表设置一个ID

我们应该为数据库里的每张表都设置一个ID做为其主键，而且最好的是一个INT型的(推荐使用UNSIGNED)，并设置上自动增加的AUTO_INCREMENT标志。
就算是你users表有一个主键叫“email”的字段，你也别让它成为主键。使用VARCHAR类型来当主键会使得性能下降。另外，在你的程序中，你应该使用表的ID来构造你的数据结构。
而且，在MySQL数据引擎下，还有一些操作需要使用主键，在这些情况下，主键的性能和设置变得非常重要，比如，集群，分区……

在这里，只有一个情况是例外，那就是“关联表”的“外键”，也就是说，这个表的主键，通过若干个别的表的主键构成。我们把这个情况叫做“外键”。比如：有一个“学生表”有学生的ID，有一个“课程表”有课程ID，那么，“成绩表”就是“关联表”了，其关联了学生表和课程表，在成绩表中，学生ID和课程ID叫“外键”其共同组成主键。

12.使用ENUM而不是VARCHAR

ENUM类型是非常快和紧凑的。在实际上，其保存的是TINYINT，但其外表上显示为字符串。这样一来，用这个字段来做一些选项列表变得相当的完美。

如果你有一个字段，比如“性别”，“国家”，“民族”，“状态”或“部门”，你知道这些字段的取值是有限而且固定的，那么，你应该使用ENUM而不是VARCHAR。

MySQL也有一个“建议”(见第十条)告诉你怎么去重新组织你的表结构。当你有一个VARCHAR字段时，这个建议会告诉你把其改成ENUM类型。使用PROCEDURE ANALYSE()你可以得到相关的建议。

13.从PROCEDURE ANALYSE()取得建议

PROCEDURE ANALYSE() 会让MySQL帮你去分析你的字段和其实际的数据，并会给你一些有用的建议。只有表中有实际的数据，这些建议才会变得有用，因为要做一些大的决定是需要有数据作为基础的。

例如，如果你创建了一个INT字段作为你的主键，然而并没有太多的数据，那么，PROCEDURE ANALYSE()会建议你把这个字段的类型改成MEDIUMINT。或是你使用了一个VARCHAR字段，因为数据不多，你可能会得到一个让你把它改成ENUM的建议。这些建议，都是可能因为数据不够多，所以决策做得就不够准。

14.尽可能的使用NOT NULL

除非你有一个很特别的原因去使用NULL值，你应该总是让你的字段保持NOT NULL。这看起来好像有点争议，请往下看。

首先，问问你自己“Empty”和“NULL”有多大的区别(如果是INT，那就是0和NULL)?如果你觉得它们之间没有什么区别，那么你就不要使用NULL。(你知道吗?在Oracle里，NULL 和Empty的字符串是一样的!)
不要以为 NULL 不需要空间，其需要额外的空间，并且，在你进行比较的时候，你的程序会更复杂。当然，这里并不是说你就不能使用NULL了，现实情况是很复杂的，依然会有些情况下，你需要使用NULL值。

15.Prepared Statements

Prepared Statements很像存储过程，是一种运行在后台的SQL语句集合，我们可以从使用prepared statements获得很多好处，无论是性能问题还是安全问题。

Prepared Statements可以检查一些你绑定好的变量，这样可以保护你的程序不会受到“SQL注入式”攻击。当然，你也可以手动地检查你的这些变量，然而，手动的检查容易出问题，而且很容易被程序员忘了。当我们使用一些framework或是ORM的时候，这样的问题会好一些。

在性能方面，当一个相同的查询被使用多次的时候，这会为你带来可观的性能优势。你可以给这些Prepared Statements定义一些参数，而MySQL只会解析一次。

虽然最新版本的MySQL在传输Prepared Statements是使用二进制形式，所以这会使得网络传输非常有效率。
当然，也有一些情况下，我们需要避免使用Prepared Statements，因为其不支持查询缓存。但据说版本5.1后支持了。 在PHP中要使用prepared statements，你可以查看其使用手册：mysqli扩展或是使用数据库抽象层，如：PDO.

16.无缓冲的查询

正常的情况下，当你在你的脚本中执行一个SQL语句的时候，你的程序会停在那里直到没这个SQL语句返回，然后你的程序再往下继续执行。你可以使用无缓冲查询来改变这个行为。关于这个事情，在PHP的文档中有一个非常不错的说明：mysql_unbuffered_query()函数：

上面那句话翻译过来是说，mysql_unbuffered_query()发送一个SQL语句到MySQL而并不像mysql_query()一样去自动fethch和缓存结果。这会相当节约很多可观的内存，尤其是那些会产生大量结果的查询语句，并且，你不需要等到所有的结果都返回，只需要第一行数据返回的时候，你就可以开始马上开始工作于查询结果了。

然而，这会有一些限制。因为你要么把所有行都读走，或是你要在进行下一次的查询前调用mysql_free_result() 清除结果。而且， mysql_num_rows() 或 mysql_data_seek() 将无法使用。所以，是否使用无缓冲的查询你需要仔细考虑。

17.把IP地址存成UNSIGNED INT

很多程序员都会创建一个VARCHAR(15) 字段来存放字符串形式的IP而不是整形的IP。如果你用整形来存放，只需要4个字节，并且你可以有定长的字段。而且，这会为你带来查询上的优势，尤其是当你需要使用这样的WHERE条件：IP between ip1 and ip2。

我们必需要使用UNSIGNED INT，因为IP地址会使用整个32位的无符号整形。
而你的查询，你可以使用 INET_ATON()来把一个字符串IP转成一个整形，并使用INET_NTOA()把一个整形转成一个字符串IP。在PHP中，也有这样的函数 ip2long()和long2ip()。

18.固定长度的表会更快

如果表中的所有字段都是“固定长度”的，整个表会被认为是 “static” 或 “fixed-length”。例如，表中没有如下类型的字段： VARCHAR，TEXT，BLOB。只要你包括了其中一个这些字段，那么这个表就不是“固定长度静态表”了，这样，MySQL 引擎会用另一种方法来处理。

固定长度的表会提高性能，因为MySQL搜寻得会更快一些，因为这些固定的长度是很容易计算下一个数据的偏移量的，所以读取的自然也会很快。而如果字段不是定长的，那么，每一次要找下一条的话，需要程序找到主键。

并且，固定长度的表也更容易被缓存和重建。不过，唯一的副作用是，固定长度的字段会浪费一些空间，因为定长的字段无论你用不用，他都是要分配那么多的空间。
使用“垂直分割”技术(见下一条)，你可以分割你的表成为两个一个是定长的，一个则是不定长的。

19.垂直分割

“垂直分割”是一种把数据库中的表按列变成几张表的方法，这样可以降低表的复杂度和字段的数目，从而达到优化的目的。(以前，在银行做过项目，见过一张表有100多个字段，很恐怖)

在Users表中有一个字段是家庭地址，这个字段是可选字段，而且你在数据库操作的时候除了个人信息外，你并不需要经常读取或是改写这个字段。那么，为什么不把他放到另外一张表中呢?这样会让你的表有更好的扩展性。

20.拆分大的DELETE或INSERT语句

如果你需要在一个在线的网站上去执行一个大的DELETE或INSERT查询，你需要非常小心，要避免你的操作让你的整个网站停止响应。因为这两个操作是会锁表的，表一锁住了，别的操作都进不来了。

Apache会有很多的子进程或线程。所以，其工作起来相当有效率，而我们的服务器也不希望有太多的子进程，线程和数据库链接，这是极大的占服务器资源的事情，尤其是内存。

如果你把你的表锁上一段时间，比如30秒钟，那么对于一个有很高访问量的站点来说，这30秒所积累的访问进程/线程，数据库链接，打开的文件数，可能不仅仅会让你泊WEB服务Crash，还可能会让你的整台服务器马上掛了。

所以，如果你有一个大的处理，你定你一定把其拆分，使用LIMIT条件是一个好的方法。下面是一个示例：

21.越小的列会越快

对于大多数的数据库引擎来说，硬盘操作可能是最重大的瓶颈。所以，把你的数据变得紧凑会对这种情况非常有帮助，因为这减少了对硬盘的访问。

参看MySQL的文档Storage Requirements查看所有的数据类型。p程序员站
如果一个表只会有几列罢了（比如说字典表，配置表），那么，我们就没有理由使用INT来做主键，使用MEDIUMINT,SMALLINT或是更小的TINYINT会更经济一些。如果你不需要记录时间，使用DATE要比DATETIME好得多。

当然，你也需要留够足够的扩展空间，不然，你日后来干这个事，你会死的很难看，参看Slashdot的例子（2009年11月06日），一个简单的ALTER TABLE语句花了3个多小时，因为里面有一千六百万条数据。

22.选择正确的存储引擎

在MySQL中有两个存储引擎MyISAM和InnoDB，每个引擎都有利有弊。酷壳以前文章《MySQL: InnoDB 还是 MyISAM?》讨论和这个事情。hp程序员之家

MyISAM适合于一些需要大量查询的应用，但其对于有大量写操作并不是很好。甚至你只是需要update一个字段，整个表都会被锁起来，而别的进程，就算是读进程都无法操作直到读操作完成。另外，MyISAM对于 SELECT COUNT(*) 这类的计算是超快无比的。

InnoDB的趋势会是一个非常复杂的存储引擎，对于一些小的应用，它会比 MyISAM还慢。它支持“行锁”，于是在写操作比较多的时候，会更优秀。并且，它还支持更多的高级应用，比如：事务。

23.使用一个对象关系映射器（Object Relational Mapper）

使用 ORM (Object Relational Mapper)，你能够获得可靠的性能增涨。一个ORM可以做的所有事情，也能被手动的编写出来。但是，这需要一个高级专家。

ORM的最重要的是“Lazy Loading”，也就是说，只有在需要的去取值的时候才会去真正的去做。但你也需要小心这种机制的副作用，因为这很有可能会因为要去创建很多很多小的查询反而会降低性能。hperz.com
ORM还可以把你的SQL语句打包成一个事务，这会比单独执行他们快得多得多。
目前，个人最喜欢的PHP的ORM是：Doctrine。

24.小心“永久链接”

“永久链接”的目的是用来减少重新创建MySQL链接的次数。当一个链接被创建了，它会永远处在连接的状态，就算是数据库操作已经结束了。而且，自从我们的Apache开始重用它的子进程后——也就是说，下一次的HTTP请求会重用Apache的子进程，并重用相同的MySQL链接。
php手册：mysql_pconnect()

在理论上来说，这听起来非常的不错。但是从个人经验（也是大多数人的）上来说，这个功能制造出来的麻烦事更多。因为，你只有有限的链接数，内存问题，文件句柄数，等等。
而且，Apache运行在极端并行的环境中，会创建很多很多的子进程。这就是为什么这种“永久链接”的机制工作地不好的原因。在你决定要使用“永久链接”之前，你需要好好地考虑一下你的整个系统的架构。

🤔👀MySQL优化02

1.存储引擎的选择

存储引擎：MySQL中的数据、索引以及其他对象的存储方式

5.1之前默认存储引擎是MyISAM,5.1之后默认存储引擎是Innodb。

差异：

区别:

- MyISAM
- Innodb

文件格式 数据和索引是分别存储的，数据.MYD，索引.MYI 数据和索引是集中存储的.ibd
文件能否移动 能，一张表就对应.frm，MYD，MYI 3个文件 不能，关联的还有data下其他文件
记录存储顺序 按记录插入顺序保存 按主键大小有序插入空间碎片产生；

定时清理，使用命令optimize table表名实现不产生
事务 不支持 支持
外键 不支持 支持
锁颗粒 表级锁 行级锁

MyISAM引擎设计简单，数据以紧密格式存储，所以某些读取场景下性能很好。

如果没有特别的需求，使用默认的Innodb即可。

MyISAM：以读写插入为主的应用程序，比如博客系统、新闻门户网站。

Innodb：更新（删除）操作频率也高，或者要保证数据的完整性；并发量高，支持事务和外键保证数据完整性。比如OA自动化办公系统。

《高性能Mysql》一书中列举很多存储引擎，但是其强烈推荐使用Innodb即可

2.字段设计

- 数据库设计3大范式

第一范式（确保每列保持原子性）
第二范式（确保表中的每列都和主键相关）
第三范式（确保每列都和主键列直接相关，而不是间接相关）
通常建议使用范式化设计,因为范式化通常会使得执行操作更快。但这并不是绝对的,范式化也是有缺点的,通常需要关联查询，不仅代价昂贵,也可能使一些索引策略无效。

- 单表字段不宜过多

建议最多30个以内

字段越多,会导致性能下降,并且增加开发难度(一眼望不尽的字段,我们这些开发仔会顿时傻掉的)

- 使用小而简单的合适数据类型

a.字符串类型

固定长度使用char,非定长使用varchar,并分配合适且足够的空间

char在查询时,会把末尾的空格去掉;

b.小数类型

一般情况可以使用float或double,占用空间小,但存储可能会损失精度

decimal可存储精确小数,存储财务数据或经度要求高时使用decimal

c.时间日期

-- datetime:

范围：1001年~9999年

存储：8个字节存储,以YYYYMMDDHHMMSS的格式存储

时区：与时区无关

-- timestamp:

范围：1970年~2038年

存储：4个字节存储,存储以UTC格式保存,与UNIX时间戳相同

时区：存储时对当前的时区进行转换,检索时再转换回当前的时区

通常尽量使用timestamp,因为它占用空间小,并且会自动进行时区转换,无需关心地区时差

datetime和timestamp只能存储最小颗粒度是秒,可以使用BIGINT类型存储微秒级别的时间戳

d.大数据 blob和text

blob和text是为存储很大的数据的而设计的字符串数据类型,但通常建议避免使用

MySQL会把每个blob和text当做独立的对象处理,存储引擎存储时会做特殊处理,当值太大,innoDB使用专门的外部存储区域进行存储,行内存储指针,然后在外部存储实际的值。这些都会导致严重的性能开销

- 尽量将列设置为NOT NULL

a.可为NULL的列占用更多的存储空间

b.可为NULL的列,在使用索引和值比较时,mySQL需要做特殊的处理,损耗一定的性能

建议:通常最好指定列为NOT NULL,除非真的需要存储NULL值

- 尽量使用整型做主键

a.整数类型通常是标识列最好的选择,因为它们很快并且可以使用AUTO_INCREMENT

b.应该避免使用字符串类型作为标识列,因为它们很消耗空间,并且通常比数字类型慢

c.对于完全"随机"的字符串也需要多加注意。例如：MD5(),SHAI()或者UUID()产生的字符串。

这些函数生成的新值也任意分布在很大空间内，这会导致INSERT和一些SELECT语句很缓慢

3. 索引

- 使用索引为什么快

索引相对于数据本身,数据量小

索引是有序的，可以快速确定数据位置

Innodb的表示是索引组织表,表数据的分布按照主键排序

- 索引的存储结构

a.B+树

b.哈希(键值对的结构)

MySQL中的主键索引用的是B+树结构,非主键索引可以选择B+树或者哈希

通常建议使用B+树索引，因为哈希索引缺点比较多:无法用于排序、无法用于范围查询、数据量大时,可能会出现大量哈希碰撞,导致效率低下

- 索引的类型

按作用分类:

1.主键索引

2.普通索引:没有特殊限制,允许重复的值

3.唯一索引:不允许有重复的值,速度比普通索引略快

4.全文索引:用作全文搜索匹配,但基本用不上,只能索引英文单词,而且操作代价很大

按数据存储结构分类:

1.聚簇索引

定义：数据行的物理顺序与列值（一般是主键的那一列）的逻辑顺序相同，一个表中只能拥有一个聚集索引。

主键索引是聚簇索引,数据的存储顺序是和主键的顺序相同的

2.非聚簇索引

定义：该索引中索引的逻辑顺序与磁盘上行的物理存储顺序不同，一个表中可以拥有多个非聚集索引。

聚簇索引以外的索引都是非聚集索引,细分为普通索引、唯一索引、全文索引,它们也被称为二级索引。

主键索引的叶子节点存储的是"行指针",直接指向物理文件的数据行。

二级索引的叶子结点存储的是主键值

覆盖索引:可直接从非主键索引直接获取数据无需回表的索引

比如:

假设t表有一个(clo1,clo2)的多列索引

select clo1,clo2 from t where clo = 1;

那么,使用这条sql查询,可直接从(clo1,clo2)索引树中获取数据,无需回表查询,
因此我们需要尽可能的在select后只写必要的查询字段，以增加索引覆盖的几率。

多列索引:使用多个列作为索引,比如(clo1,clo2)

使用场景:当查询中经常使用clo1和clo2作为查询条件时,可以使用组合索引,这种索引会比单列索引更快

需要注意的是,多列索引的使用遵循最左索引原则

假设创建了多列索引index(A,B,C)，那么其实相当于创建了如下三个组合索引：

1.index(A,B,C)

2.index(A,B)

3.index(A)

这就是最左索引原则，就是从最左侧开始组合。

- 索引优化

1.索引不是越多越好,索引是需要维护成本的

2.在连接字段上应该建立索引

3.尽量选择区分度高的列作为索引,区分度count(distinct col)/count(*)表示字段不重复的比例,比例越大扫描的记录数越少，状态值、性别字段等区分度低的字段不适合建索引

4.几个字段经常同时以AND方式出现在Where子句中,可以建立复合索引,否则考虑单字段索引

5.把计算放到业务层而不是数据库层

6.如果有 order by、group by 的场景，请注意利用索引的有序性。

order by 最后的字段是组合索引的一部分，并且放在索引组合顺序的最后，避免出现 file_sort 的情况，影响查询性能

例如对于语句 where a=? and b=? order by c，可以建立联合索引(a,b,c)。

order by 最后的字段是组合索引的一部分，并且放在索引组合顺序的最后，避免出现 file_sort(外部排序) 的情况，影响查询性能对于语句 where a=? and b=? order by c，可以建立联合索引(a,b,c)。

如果索引中有范围查找，那么索引有序性无法利用，如 WHERE a>10 ORDER BY b;索引(a,b)无法排序。

- 可能导致无法使用索引的情况

1.is null 和 is not null

2.!= 和 <> (可用in代替)

3."非独立列":索引列为表达式的一部分或是函数的参数

例如:表达式的一部分:select id from t where id +1 = 5　函数参数:select id from t where to_days(date_clo) >= 10

4.like查询以%开头

5.or (or两边的列都建立了索引则可以使用索引)

6.类型不一致

如果列是字符串类型，传入条件是必须用引号引起来，不然无法使用索引

select * from tb1 where email = 999;

4. sql优化建议

1.首先了解一下sql的执行顺序,使我们更好的优化

(1)FROM:数据从硬盘加载到数据缓冲区，方便对接下来的数据进行操作

(2)ON:join on实现多表连接查询,先筛选on的条件,再连接表

(3)JOIN:将join两边的表根据on的条件连接

(4)WHERE:从基表或视图中选择满足条件的元组

(5)GROUP BY:分组，一般和聚合函数一起使用

(6)HAVING:在元组的基础上进行筛选，选出符合条件的元组（必须与GROUP BY连用）

(7)SELECT:查询到得所有元组需要罗列的哪些列

(8)DISTINCT:去重

(9)UNION:将多个查询结果合并

(10)ORDER BY：进行相应的排序

(11)LIMIT:显示输出一条数据记录

join on实现多表连接查询，推荐该种方式进行多表查询，不使用子查询(子查询会创建临时表,损耗性能)

避免使用HAVING筛选数据,而是使用where order by后面的字段建立索引,利用索引的有序性排序,避免外部排序

如果明确知道只有一条结果返回，limit

1.能够提高效率

2.超过三个表最好不要 join

3.避免 SELECT *，从数据库里读出越多的数据，那么查询就会变得越慢

4.尽可能的使用 NOT NULL列,可为NULL的列占用额外的空间,且在值比较和使用索引时需要特殊处理,影响性能

5.用exists、not exists和in、not in相互替代

原则是哪个的子查询产生的结果集小，就选哪个

select * from t1 where x in (select y from t2)

select * from t1 where exists (select null from t2 where y = x)

IN适合于外表大而内表小的情况；existx适合于外表小而内表大的情况

6.使用exists代替distinct

当提交一个包含一对多表信息（比如部门表和职员表）的查询时，避免在select子句中使用distinct，

一般可以考虑使用exists代替，使查询更为迅速，因为子查询的条件一旦满足，立马返回结果

低效写法：

select distinct dept_no,dept_name from dept d,emp e where d.dept_no=e.dept_no

高效写法：

select dept_no,dept_name from dept d where exists (select 'x' from emp e where e.dept_no=d.dept_no)　　备注：其中x的意思是：因为exists只是看子查询是否有结果返回，而不关心返回的什么内容，因此建议写一个常量，性能较高！

用exists的确可以替代distinct，不过以上方案仅适用dept_no为唯一主键的情况，如果要去掉重复记录，需要参照以下写法：

select * from emp where dept_no exists (select Max(dept_no)) from dept d, emp e where e.dept_no=d.dept_no
group by d.dept_no)

7. 避免隐式数据类型的转换
   隐式数据类型转换不能适用索引，导致全表扫描！t_tablename表的phonenumber字段为varchar类型
   以下代码不符合规范：
   select column1 into i_l_variable1 from t_tablename where phonenumber=18519722169;
   应编写如下：
   select column1 into i_lvariable1 from t_tablename where phonenumber='18519722169';

8. 分段查询
   在一些查询页面中，当用户选择的时间范围过大，造成查询缓慢。主要的原因是扫描行数过多。这个时候可以通过程序，

```
分段进行查询，循环遍历，将结果合并处理进行展示。
  5.explain分析执行计划
```

9. explain显示了mysql如何使用索引来处理select语句以及连接表。可以帮助选择更好的索引和写出更优化的查询语句。
   例：
   explain select user from mysql.user;

```
+----+-------------+-------+------------+------+---------------+---------+---------+------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key     | key_len | ref  | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+---------+---------+------+------+----------+-------+
|  1 | SIMPLE      | user  | NULL       | index | NULL          | PRIMARY | 276     | NULL |    6 |   100.00 | Using index |
+----+-------------+-------+------------+------+---------------+---------+---------+------+------+----------+-------+
```

标识符 含义
id
select标识符：这是select的查询序列号
select_type
select类型：

simple,简单select（不使用union和子查询）

primary,查询中包含任何复杂的子部分，最外层的select被标记为PRIMARY

union,union中的第二个或后面的select语句

DEPENDENT UNION：一般是子查询中的第二个select语句（取决于外查询，mysql内部也有些优化）

UNION RESULT：union的结果

SUBQUERY：子查询中的第一个select

DEPENDENT SUBQUERY：子查询中第一个select，取决于外查询（在mysql中会有些优化，有些dependent会直接优化成simple）

DERIVED：派生表的select（from子句的子查询）

table
显示数据来自于哪个表，有时不是真实的表的名字（虚拟表），虚拟表最后一位是数字，代表id为多少的查询

type
这个字段值较多，这里我只重点关注我们开发中经常用到的几个字段:system,const,eq_ref,ref,range,index,all;

性能由好到差依次为：
system>const>eq_ref>ref>range>index>all(一定要牢记)

system：
表只有一行记录，这个是const的特例，一般不会出现，可以忽略

const：
表示通过索引一次就找到了，const用于比较primary key或者unique索引。因为只匹配一行数据，所以很快

eq_ref：
唯一性索引扫描，表中只有一条记录与之匹配。一般是两表关联，关联条件中的字段是主键或唯一索引

ref：
非唯一行索引扫描，返回匹配某个单独值的所有行

range：
检索给定范围的行，一般条件查询中出现了>、<、in、between等查询

index：
遍历索引树。通常比ALL快，因为索引文件通常比数据文件小。all和index都是读全表，但index是从索引中检索的，而all是从硬盘中检索的。

all：遍历全表以找到匹配的行

possible_keys
显示可能应用在这张表中的索引，但不一定被查询实际使用

key
实际使用的索引，如果没有显示null

key_len
表示索引中使用的字节数，可通过该列计算查询中使用的索引的长度。
一般来说，索引长度越长表示精度越高，效率偏低；长度越短，效率高，但精度就偏低。并不是真正使用索引的长度，是个预估值。

ref
表示哪一列被使用了，常数表示这一列等于某个常数。

rows
大致找到所需记录需要读取的行数

filtered
表示选取的行和读取的行的百分比，100表示选取了100%，80表示读取了80%。

Extra
一些重要的额外信息

Using filesort：使用外部的索引排序，而不是按照表内的索引顺序进行读取。（一般需要优化）
Using temporary：使用了临时表保存中间结果。常见于排序order by和分组查询group by（最好优化）
Using index：表示select语句中使用了覆盖索引，直接冲索引中取值，而不需要回行（从磁盘中取数据）
Using where：使用了where过滤
Using index condition：5.6之后新增的，表示查询的列有非索引的列，先判断索引的条件，以减少磁盘的IO
Using join buffer：使用了连接缓存
impossible where：where子句的值总是false