## ****061 如何删除表？****

　　答案：运行命令 drop table table\_name;

## ****062 创建索引****

　　对于查询占主要的应用来说，索引显得尤为重要。很多时候性能问题很简单的就是因为我们忘了添加索引而造成的，或者说没有添加更为有效的索引导致。如果不加索引的话，那么查找任何哪怕只是一条特定的数据都会进行一次全表扫描，如果一张表的数据量很大而符合条件的结果又很少，那么不加索引会引起致命的性能下降。但是也不是什么情 况都非得建索引不可，比如性别可能就只有两个值，建索引不仅没什么优势，还会影响到更新速度，这被称为过度索引。

## ****063 复合索引****

　　比如有一条语句是这样的：select \* from users where area=’beijing’ and age=22;   
　　如果我们是在area和age上分别创建单个索引的话，由于**[MySQL](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/derrantcm/article/details/_blank)**查询每次只能使用一个索引，所以虽然这样已经相对不做索引时全表扫描提高了很多效率，但是如果在area、age两列上创建复合索引的话将带来更高的效率。如果我们创建了(area, age, salary)的复合索引，那么其实相当于创建了(area,age,salary)、 (area,age)、(area)三个索引，这被称为最佳左前缀特性。因此我们在创建复合索引时应该将最常用作限制条件的列放在最左边，依次递减。

## ****064 索引不会包含有NULL值的列****

　　只要列中包含有NULL值都将不会被包含在索引中，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**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/derrantcm/article/details/_blank)**设计时不要让字段的默认值为NULL。

## ****065 使用短索引****

　　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。短索引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。

## ****066 排序的索引问题****

**[mysql](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/derrantcm/article/details/_blank)**查询只使用一个索引，因此如果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不要使用排序操作；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。

## ****067 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，如何使用也是一个问题。like “%aaa%” 不会使用索引而like “aaa%”可以使用索引。

## ****068 MYSQL数据库设计数据类型选择需要注意哪些地方?****

VARCHAR和CHAR类型，varchar是变长的，需要额外的1-2个字节存储，能节约空间，可能会对性能有帮助。但由于是变长，可能发生碎片，如更新数据；   
　　使用ENUM（MySQL的枚举类）代替字符串类型，数据实际存储为整型。   
　　字符串类型   
　　要尽可能地避免使用字符串来做标识符，因为它们占用了很多空间并且通常比整数类型要慢。特别注意不要在MYISAM表上使用字符串标识符。MYISAM默认情况下为字符串使用了压缩索引（Packed Index），这使查找更为缓慢。据**[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库" \t "http://blog.csdn.net/derrantcm/article/details/_blank)**，使用了压缩索引的MYISAM表性能要慢6倍。   
　　还要特别注意完全‘随机’的字符串，例如由MD5（）、SHA1（）、UUID（）产生的。它们产生的每一个新值都会被任意地保存在很大的空间范围内，这会减慢INSERT及一些SELECT查询。

1. 它们会减慢INSERT查询，因为插入的值会被随机地放入索引中。这会导致分页、随机磁盘访问及聚集存储引擎上的聚集索引碎片。
2. 它们会减慢SELECT查询，因为逻辑上相邻的行会分布在磁盘和内存中的各个地方。

3）随机值导致缓存对所有类型的查询性能都很差，因为它们会使缓存赖以工作的访问局部性失效。如果整个数据集都变得同样“热”的时候，那么把特定部分的数据缓存到内存中就没有任何的优势了。并且如果工作集不能被装入内存中，缓存就会进行很多刷写的工作，并且会导致很多缓存未命中。   
　　如果保存UUID值，就应该移除其中的短横线，更好的办法是使用UHEX（）把UUID值转化为16字节的数字，并把它保存在BINARY（16）列中。

## ****069 不要在列上进行运算****

　　select \* from users where YEAR(adddate)<2007;   
　　将在每个行上进行运算，这将导致索引失效而进行全表扫描，因此我们可以改成   
　　select \* from users where adddate<‘2007-01-01’;   
　　不使用NOT IN和操作   
　　NOT IN和操作都不会使用索引将进行全表扫描。NOT IN可以NOT EXISTS代替，id != 3则可使用id>3 or id<3来代替。

## ****070 IS NULL 与 IS NOT NULL****

　　不能用null作索引，任何包含null值的列都将不会被包含在索引中。即使索引有多列这样的情况下，只要这些列中有一列含有null，该列就会从索引中排除。也就是说如果某列存在空值，即使对该列建索引也不会提高性能。   
　　任何在where子句中使用is null或is not null的语句优化器是不允许使用索引的。

## ****071 联接列****

　　对于有联接的列，即使最后的联接值为一个静态值，优化器是不会使用索引的。

## ****072 MySQL几种备份方式（重点）****

　　1、**逻辑备份**：使用mysql自带的mysqldump工具进行备份。备份成sql文件形式。   
　　　　**优点**：最大好处是能够与正在运行的mysql自动协同工作，在运行期间可以确保备份是当时的点，它会自动将对应操作的表锁定，不允许其他用户修改(只能访问)。可能会阻止修改操作。sql文件通用方便移植。   
　　　　**缺点**：备份的速度比较慢。如果是数据量很多的时候。就很耗时间。如果数据库服务器处在提供给用户服务状态，在这段长时间操作过程中，意味着要锁定表(一般是读锁定，只能读不能写入数据)。那么服务就会影响的。   
　　2、**物理备份**：直接拷贝mysql的数据目录。   
　　直接拷贝只适用于myisam类型的表。这种类型的表是与机器独立的。但实际情况是，你设计数据库的时候不可能全部使用myisam类型表。你也不可能因为myisam类型表与机器独立，方便移植，于是就选择这种表，这并不是选择它的理由。   
　　　　**缺点**：你不能去操作正在运行的mysql服务器(在拷贝的过程中有用户通过应用程序访问更新数据，这样就无法备份当时的数据)可能无法移植到其他机器上去。   
　　3、**双机热备份**。   
　　mysql数据库没有增量备份的机制。当数据量太大的时候备份是一个很大的问题。还好mysql数据库提供了一种主从备份的机制(也就是双机热备)   
　　　　**优点**：适合数据量大的时候。现在明白了。大的互联网公司对于mysql数据备份，都是采用热机备份。搭建多台数据库服务器，进行主从复制。

## ****073 想知道一个查询用到了哪个index,如何查看?****

　　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。使用方法，在select语句前加上explain就可以了。所以使用explain可以查看。

## ****074 数据库不能停机，请问如何备份? 如何进行全备份和增量备份?****

　　可以使用**逻辑备份**和**双机热备份**。   
　　完全备份：完整备份一般一段时间进行一次，且在网站访问量最小的时候，这样常借助批处理文件定时备份。主要是写一个批处理文件在里面写上处理程序的绝对路径然后把要处理的东西写在后面，即完全备份数据库。   
　　增量备份：对ddl和dml语句进行二进制备份。且5.0无法增量备份，5.1后可以。如果要实现增量备份需要在my.ini文件中配置备份路径即可，重启mysql服务器，增量备份就启动了。

## 

## ****075 MySQL添加索引****

　　普通索引 添加INDEX   
　　　　ALTER TABLE ‘table\_name’ ADD INDEX index\_name (‘column’);   
　　主键索引 添加PRIMARY KEY   
　　　　ALTER TABLE ‘table\_name’ ADD PRIMARY KEY (‘column’);   
　　唯一索引 添加UNIQUE   
　　　　ALTER TABLE ‘table\_name’ ADD UNIQUE (‘column’);   
　　全文索引 添加FULLTEXT   
　　　　ALTER TABLE ‘table\_name’ ADD FULLTEXT (‘column’);   
　　多列索引   
　　　　ALTER TABLE ‘table\_name’ ADD INDEX index\_name (‘column1’, ‘column2’, ‘column3’)

## ****076 什么情况下使用索引？****

**表的主关键字**   
　　自动建立唯一索引   
　　　　如zl\_yhjbqk（用户基本情况）中的hbs\_bh（户标识编号）   
　　**表的字段唯一约束**   
　　　　**[Oracle](http://lib.csdn.net/base/oracle" \o "Oracle知识库" \t "http://blog.csdn.net/derrantcm/article/details/_blank)**利用索引来保证数据的完整性   
　　　　如lc\_hj（流程环节）中的lc\_bh+hj\_sx（流程编号+环节顺序）   
　　**直接条件查询的字段**   
　　在SQL中用于条件约束的字段   
　　　　如zl\_yhjbqk（用户基本情况）中的qc\_bh（区册编号）   
　　　　select \* from zl\_yhjbqk where qc\_bh=’7001’   
　　**查询中与其它表关联的字段**   
　　字段常常建立了外键关系   
　　　　如zl\_ydcf（用电成份）中的jldb\_bh（计量点表编号）   
　　　　select \* from zl\_ydcf a,zl\_yhdb b where a.jldb\_bh=b.jldb\_bh and b.jldb\_bh=’540100214511’   
　　**查询中排序的字段**   
　　排序的字段如果通过索引去访问那将大大提高排序速度   
　　　　select \* from zl\_yhjbqk order by qc\_bh（建立qc\_bh索引）   
　　　　select \* from zl\_yhjbqk where qc\_bh=’7001’ order by cb\_sx（建立qc\_bh+cb\_sx索引，注：只是一个索引，其中包括qc\_bh和cb\_sx字段）   
　　**查询中统计或分组统计的字段**   
　　　　select max(hbs\_bh) from zl\_yhjbqk   
　　　　select qc\_bh,count(\*) from zl\_yhjbqk group by qc\_bh

## ****077 什么情况下应不建或少建索引****

**表记录太少**   
　　如果一个表只有5条记录，采用索引去访问记录的话，那首先需访问索引表，再通过索引表访问数据表，一般索引表与数据表不在同一个数据块，这种情况下**[oracle](http://lib.csdn.net/base/oracle" \o "Oracle知识库" \t "http://blog.csdn.net/derrantcm/article/details/_blank)**至少要往返读取数据块两次。而不用索引的情况下ORACLE会将所有的数据一次读出，处理速度显然会比用索引快。   
　　如表zl\_sybm（使用部门）一般只有几条记录，除了主关键字外对任何一个字段建索引都不会产生性能优化，实际上如果对这个表进行了统计分析后ORACLE也不会用你建的索引，而是自动执行全表访问。如：select \* from zl\_sybm where sydw\_bh=’5401’（对sydw\_bh建立索引不会产生性能优化）   
　　**经常插入、删除、修改的表**   
　　对一些经常处理的业务表应在查询允许的情况下尽量减少索引，如zl\_yhbm，gc\_dfss，gc\_dfys，gc\_fpdy等业务表。   
　　**数据重复且分布平均的表字段**   
　　假如一个表有10万行记录，有一个字段A只有T和F两种值，且每个值的分布概率大约为50%，那么对这种表A字段建索引一般不会提高数据库的查询速度。   
　　**经常和主字段一块查询但主字段索引值比较多的表字段**   
　　如gc\_dfss（电费实收）表经常按收费序号、户标识编号、抄表日期、电费发生年月、操作 标志来具体查询某一笔收款的情况，如果将所有的字段都建在一个索引里那将会增加数据的修改、插入、删除时间，从实际上分析一笔收款如果按收费序号索引就已 经将记录减少到只有几条，如果再按后面的几个字段索引查询将对性能不产生太大的影 响。

## ****078 千万级MySQL数据库建立索引的事项及提高性能的手段****

**1.对查询进行优化，应尽量避免全表扫描**，首先应考虑在 where 及 order by 涉及的列上建立索引。   
　　**2.应尽量避免在 where 子句中对字段进行 null 值判断**，否则将导致引擎放弃使用索引而进行全表扫描，如：select id from t where num is null可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：select id from t where num=0   
　　**3.应尽量避免在 where 子句中使用!=或<>操作符**，否则引擎将放弃使用索引而进行全表扫描。   
　　**4.应尽量避免在 where 子句中使用or 来连接条件**，否则将导致引擎放弃使用索引而进行全表扫描，如：select id from t where num=10 or num=20可以这样查询：select id from t where num=10 union all select id from t where num=20   
　　**5.in 和 not in 也要慎用**，否则会导致全表扫描，如：select id from t where num in(1,2,3) 对于连续的数值，能用 between 就不要用 in 了：select id from t where num between 1 and 3   
　　**6.避免使用通配符。**下面的查询也将导致全表扫描：select id from t where name like ‘李%’若要提高效率，可以考虑全文检索。   
　　**7.如果在 where 子句中使用参数，也会导致全表扫描。**因为SQL只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：select id from t where num=@num可以改为强制查询使用索引：select id from t with(index(索引名)) where num=@num   
　　**8.应尽量避免在 where 子句中对字段进行表达式操作**，这将导致引擎放弃使用索引而进行全表扫描。如：select id from t where num/2=100应改为:select id from t where num=100\*2   
　　**9.应尽量避免在where子句中对字段进行函数操作**，这将导致引擎放弃使用索引而进行全表扫描。如：select id from t where substring(name,1,3)=’abc’ ，name以abc开头的id应改为:select id from t where name like ‘abc%’   
　　**10.不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算**，否则系统将可能无法正确使用索引。   
　　**11.在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。**   
　　**12.不要写一些没有意义的查询**，如需要生成一个空表结构：select col1,col2 into #t from t where 1=0 这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：create table #t(…)   
　　**13.很多时候用 exists 代替 in 是一个好的选择**：select num from a where num in(select num from b)用下面的语句替换：select num from a where exists(select 1 from b where num=a.num)   
　　**14.并不是所有索引对查询都有效**，SQL是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL查询可能不会去利用索引，如一表中有字段sex，male、female几乎各一半，那么即使在sex上建了索引也对查询效率起不了作用。   
　　**15.索引并不是越多越好**，索引固然可以提高相应的 select 的效率，但同时也降低了insert 及 update 的 效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有 必要。   
　　**16.应尽可能的避免更新 clustered 索引数据列**，因为 clustered 索引数据列的顺序就是表记录的物理存储 顺序，一旦该列值改变将导致整个表记录的顺序的调整，会耗费相当大的资源。若应用系统需要频繁更新 clustered 索引数据列，那么需要考虑是否应将该索引建为 clustered 索引。   
　　**17.尽量使用数字型字段**，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。   
　　**18.尽可能的使用 varchar/nvarchar 代替 char/nchar** ，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。   
　　**19.任何地方都不要使用 select \* from t ，用具体的字段列表代替“\*”**，不要返回用不到的任何字段。   
　　**20.尽量使用表变量来代替临时表**。如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。   
　　**21.避免频繁创建和删除临时表**，以减少系统表资源的消耗。   
　　**22.临时表并不是不可使用，适当地使用它们可以使某些例程更有效**，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。   
　　**23.在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table**，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。   
　　**24.如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除**，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。   
　　**25.尽量避免使用游标**，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。   
　　**26.使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。**   
　　**27.与临时表一样，游标并不是不可使用。**对小型数据集使用 FAST\_FORWARD 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。   
　　**28.在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF。**无需在执行存储过程和触发器的每个语句后向客户端发送DONE\_IN\_PROC 消息。   
　　**29.尽量避免大事务操作，提高系统并发能力。**   
　　**30.尽量避免向客户端返回[大数据](http://lib.csdn.net/base/hadoop" \o "Hadoop知识库" \t "http://blog.csdn.net/derrantcm/article/details/_blank)量**，若数据量过大，应该考虑相应需求是否合理。

## ****079 MySql在建立索引优化时需要注意的问题****

**1，创建索引**   
　　对于查询占主要的应用来说，索引显得尤为重要。很多时候性能问题很简单的就是因为我们忘了添加索引而造成的，或者说没有添加更为有效的索引导致。如果不加索引的话，那么查找任何哪怕只是一条特定的数据都会进行一次全表扫描，如果一张表的数据量很大而符合条件的结果又很少，那么不加索引会引起致命的性能下降。但是也不是什么情况都非得建索引不可，比如性别可能就只有两个值，建索引不仅没什么优势，还会影响到更新速度，这被称为过度索引。   
　　**2，复合索引**   
　　比如有一条语句是这样的：select \* from users where area=’beijing’ and age=22;   
　　如果我们是在area和age上分别创建单个索引的话，由于mysql查询每次只能使用一个索引，所以虽然这样已经相对不做索引时全表扫描提高了很多效率，但是如果在area、age两列上创建复合索引的话将带来更高的效率。如果我们创建了(area, age,salary)的复合索引，那么其实相当于创建了(area,age,salary)、(area,age)、(area)三个索引，这被称为最佳左前缀特性。因此我们在创建复合索引时应该将最常用作限制条件的列放在最左边，依次递减。   
　　**3，索引不会包含有NULL值的列**   
　　只要列中包含有NULL值都将不会被包含在索引中，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。所以我们在数据库设计时不要让字段的默认值为NULL。   
　　**4，使用短索引**   
　　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。短索引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。   
　　**5，排序的索引问题**   
　　mysql查询只使用一个索引，因此如果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不要使用排序操作；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。   
　　**6，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，如何使用也是一个问题。like “%aaa%” 不会使用索引而like “aaa%”可以使用索引。   
　　**7，不要在列上进行运算**   
　　select \* from users where YEAR(adddate)   
　　**8，不使用NOT IN和操作**   
　　NOT IN和操作都不会使用索引将进行全表扫描。NOT IN可以NOT EXISTS代替，id != 3则可使用id>3 or id < 3

## ****080 数据库性能下降，想找到哪些sql耗时较长，应该如何操作? my.cnf里如何配置?****

　　1、show processlist;   
　　2、select \* from information\_schema.processlist ;   
　　3、可以在[mysqld]中添加如下：   
　　　　log =/var/log/mysql.log   
　　　　如果需要监控慢查询可以添加如下内容：   
　　　　log-slow-queries = /var/log/slowquery.log   
　　　　long\_query\_time = 1

## ****081 聚集索引****

　　术语“聚集”指实际的数据行和相关的键值都保存在一起。每个表只能有一个聚集索引。但是，覆盖索引可以模拟多个聚集索引。存储引擎负责实现索引，因此不是所有的存储索引都支持聚集索引。当前，SolidDB和InnoDB是唯一支持聚集索引的存储引擎。   
　　**优点：**   
　　**可以把相关数据保存在一起**。这样从磁盘上提取几个页面的数据就能把某个用户的数据全部抓取出来。如果没有使用聚集，读取每个数据都会访问磁盘。   
　　**数据访问快**。聚集索引把索引和数据都保存到了同一棵B-TREE中，因此从聚集索引中取得数据通常比在非聚集索引进行查找要快。   
　　**缺点：**   
　　**聚集能最大限度地提升I/O密集负载的性能**。如果数据能装入内存，那么其顺序也就无所谓了。这样聚集就没有什么用处。   
　　**插入速度严重依赖于插入顺序**。更新聚集索引列是昂贵的，因为强制InnoDB把每个更新的行移到新的位置。   
　　建立在聚集索引上的表在插入新行，或者在行的主键被更新，该行必须被移动的时候会进行分页。   
　　聚集表可会比全表扫描慢，尤其在表存储得比较稀疏或因为分页而没有顺序存储的时候。   
　　第二（非聚集）索引可能会比预想的大，因为它们的叶子节点包含了被引用行的主键列。第二索引访问需要两次索引查找，而不是一次。 InnoDB的第二索引叶子节点包含了主键值作为指向行的“指针”，而不是“行指针”。 这种策略减少了在移动行或数据分页的时候索引的维护工作。使用行的主键值作为指针使得索引变得更大，但是这意味着InnoDB可以移动行，而无须更新指针。

## ****082 索引类型****

**索引类型: B-TREE索引，哈希索引**   
　　**B-TREE索引**(默认的索引类型)加速了数据访问，因为存储引擎不会扫描整个表得到需要的数据。相反，它从根节点开始。根节点保存了指向子节点的指针，并且存储引擎会根据指针寻找数据。它通过查找节点页中的值找到正确的指针，节点页包含子节点的指针，并且存储引擎会根据指针寻找数据。它通过查找节点页中的值找到正确的指针，节点页包含子节点中值的上界和下界。最后，存储引擎可能无法找到需要的数据，也可能成功地找到包含数据的叶子页面。   
　　例：B-TREE索引 对于以下类型查询有用。匹配全名、匹配最左前缀、匹配列前缀、匹配范围值、精确匹配一部分并且匹配某个范围中的另一部分；   
　　**B-TREE索引的局限**：如果查找没有从索引列的最左边开始，它就没什么用处。不能跳过索引中的列，存储引擎不能优先访问任何在第一个范围条件右边的列。例：如果查询是where last\_name=’Smith’ AND first\_name LIKE ‘J%’ AND dob=’1976-12-23’;访问就只能使用索引的头两列，因为LIKE是范围条件。   
　　**哈希索**引建立在哈希表的基础上，它只对使用了索引中的每一列的精确查找有用。对于每一行，存储引擎计算出了被索引列的哈希码，它是一个较小的值，并且有可能和其他行的哈希码不同。它把哈希码保存在索引中，并且保存了一个指向哈希表中每一行的指针。   
　　因为索引只包含了哈希码和行指针，而不是值自身，MYSQL不能使用索引中的值来避免读取行。   
　　MYSQL不能使用哈希索引进行排序，因为它们不会按序保存行。   
　　哈希索引不支持部分键匹配，因为它们是由被索引的全部值计算出来的。也就是说，如果在（A，B）两列上有索引，并且WHERE子句中只使用了A，那么索引就不会起作用。   
　　哈希索引只支持使用了= IN（）和<=>的相等比较。它们不能加快范围查询。例如WHERE price > 100;   
　　访问哈希索引中的数据非常快，除非碰撞率很高。当发生碰撞的时候，存储引擎必须访问链表中的每一个行指针，然后逐行进行数据比较，以确定正确的数据。如果有很多碰撞，一些索引维护操作就有可能会变慢。

## ****083 FULLTEXT全文索引****

　　即为全文索引，目前只有MyISAM引擎支持。其可以在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。FULLTEXT索引也是按照分词原理建立索引的。西文中，大部分为字母文字，分词可以很方便的按照空格进行分割。中文不能按照这种方式进行分词。Mysql的中文分词插件Mysqlcft，有了它，就可以对中文进行分词。

## ****084 介绍一下如何优化MySql****

**一、在编译时优化MySQL**   
　　如果你从源代码分发安装MySQL，要注意，编译过程对以后的目标程序性能有重要的影响，不同的编译方式可能得到类似的目标文件，但性能可能相差很大，因此，在编译安装MySQL适应仔细根据你的应用类型选择最可能好的编译选项。这种定制的MySQL可以为你的应用提供最佳性能。   
　　技巧：选用较好的编译器和较好的编译器选项，这样应用可提高性能10-30%。（MySQL文档如是说）   
　　　　1.1、使用pgcc（Pentium GCC)编译器->（**使用合适的编译器**）   
　　　　该编译器（[http://www.goof.com/pcg/](http://www.goof.com/pcg/" \t "http://blog.csdn.net/derrantcm/article/details/_blank)）针对运行在奔腾处理器系统上的程序进行优化，用pgcc编译MySQL源代码，总体性能可提高10%。当然如果你的服务器不是用奔腾处理器，就不必用它了，因为它是专为奔腾系统设计的。   
　　　　**1.2、仅使用你想使用的字符集编译MySQL**   
　　　　MySQL目前提供多达24种不同的字符集，为全球用户以他们自己的语言插入或查看表中的数据。却省情况下，MySQL安装所有者这些字符集，热然而，最好的选择是指选择一种你需要的。如，禁止除Latin1字符集以外的所有其它字符集：   
　　　　**1.3、将mysqld编译成静态执行文件**   
　　　　将mysqld编译成静态执行文件而无需共享库也能获得更好的性能。通过在配置时指定下列选项，可静态编译mysqld。   
　　　　**1.4、配置样本**   
　　**二、调整服务器**   
　　**三、表类型（MySQL中表的类型）**   
　　很多MySQL用户可能很惊讶，**MySQL确实为用户提供5种不同的表类型，称为DBD、HEAP、ISAM、MERGE和MyIASM。DBD归为事务安全类，而其他为非事务安全类。**   
　　　　3.1、**事务安全**   
　　　　**DBD**   
　　　　Berkeley DB(DBD)表是支持事务处理的表，它提供MySQL用户期待已久的功能-事务控制。事务控制在任何数据库系统中都是一个极有价值的功能，因为它们确保一组命令能成功地执行。   
　　　　3.2、**非事务安全**   
　　　　**HEAP**   
　　　　HEAP表是MySQL中存取数据最快的表。这是因为他们使用存储在动态内存中的一个哈希索引。另一个要点是如果MySQL或服务器崩溃，数据将丢失。   
　　　　**ISAM**   
　　　　ISAM表是早期MySQL版本的缺省表类型，直到MyIASM开发出来。建议不要再使用它。   
　　　　**MERGE**   
　　　　MERGE是一个有趣的新类型，在3.23.25之后出现。一个MERGE表实际上是一个相同MyISAM表的集合，合并成一个表，主要是为了效率原因。这样可以提高速度、搜索效率、修复效率并节省磁盘空间。   
　　　　**MyIASM**   
　　　　这是MySQL的缺省表类型(5.5.5之前)。它基于IASM代码，但有很多有用的扩展。MyIASM比较好的原因：   
　　　　MyIASM表小于IASM表，所以使用较少资源。   
　　　　MyIASM表在不同的平台上二进制层可移植。   
　　　　更大的键码尺寸，更大的键码上限。   
　　　　3.3、**指定表类型**   
　　四、**优化工具**   
　　MySQL服务器本身提供了几条内置命令用于帮助优化。   
　　　　4.1、**SHOW**   
　　　　SHOW还能做更多的事情。它可以显示关于日志文件、特定数据库、表、索引、进程和权限表中有价值的信息。   
　　　　4.2、**EXPLAIN**   
　　　　当你面对SELECT语句时，EXPLAIN解释SELECT命令如何被处理。这不仅对决定是否应该增加一个索引，而且对决定一个复杂的Join如何被MySQL处理都是有帮助的。   
　　　　4.3、**OPTIMIZE**   
　　　　OPTIMIZE语句允许你恢复空间和合并数据文件碎片，对包含变长行的表进行了大量更新和删除后，这样做特别重要。OPTIMIZE目前只工作于MyIASM和BDB表。

## ****085 MySQL你都修改了那些配置文件来进行优化(问配置文件中具体修改的内容)?****

**innodb\_buffer\_pool\_size**:这是你安装完InnoDB后第一个应该设置的选项。缓冲池是数据和索引缓存的地方：这个值越大越好，这能保证你在大多数的读取操作时使用的是内存而不是硬盘。典型的值是5-6GB(8GB内存)，20-25GB(32GB内存)，100-120GB(128GB内存)。   
　　**innodb\_log\_file\_size**：这是redo日志的大小。redo日志被用于确保写操作快速而可靠并且在崩溃时恢复。一直到MySQL 5.1，它都难于调整，因为一方面你想让它更大来提高性能，另一方面你想让它更小来使得崩溃后更快恢复。幸运的是从MySQL 5.5之后，崩溃恢复的性能的到了很大提升，这样你就可以同时拥有较高的写入性能和崩溃恢复性能了。一直到MySQL 5.5，redo日志的总尺寸被限定在4GB(默认可以有2个log文件)。这在MySQL 5.6里被提高。   
　　一开始就把innodb\_log\_file\_size设置成512M(这样有1GB的redo日志)会使你有充裕的写操作空间。如果你知道你的应用程序需要频繁的写入数据并且你使用的时MySQL 5.6，你可以一开始就把它这是成4G。max\_connections:如果你经常看到‘Too many connections’错误，是因为max\_connections的值太低了。这非常常见因为应用程序没有正确的关闭数据库连接，你需要比默认的151连接数更大的值。max\_connection值被设高了(例如1000或更高)之后一个主要缺陷是当服务器运行1000个或更高的活动事务时会变的没有响应。在应用程序里使用连接池或者在MySQL里使用进程池有助于解决这一问题。   
　　**InnoDB配置**   
　　从MySQL 5.5版本开始，InnoDB就是默认的存储引擎并且它比任何其他存储引擎的使用都要多得多。那也是为什么它需要小心配置的原因。   
　　**innodb\_file\_per\_table**：这项设置告知InnoDB是否需要将所有表的数据和索引存放在共享表空间里 （innodb\_file\_per\_table = OFF）或者为每张表的数据单独放在一个.ibd文件（innodb\_file\_per\_table = ON）。每张表一个文件允许你在drop、truncate或者rebuild表时回收磁盘空间。这对于一些高级特性也是有必要的，比如数据压缩。但是它不会带来任何性能收益。你不想让每张表一个文件的主要场景是：有非常多的表（比如10k+）。   
　　MySQL 5.6中，这个属性默认值是ON，因此大部分情况下你什么都不需要做。对于之前的版本你必需在加载数据之前将这个属性设置为ON，因为它只对新创建的表有影响。   
　　**innodb\_flush\_log\_at\_trx\_commit**：默认值为1，表示InnoDB完全支持ACID特性。当你的主要关注点是数据安全的时候这个值是最合适的，比如在一个主节点上。但是对于磁盘（读写）速度较慢的系统，它会带来很巨大的开销，因为每次将改变flush到redo日志都需要额外的fsyncs。将它的值设置为2会导致不太可靠（reliable）因为提交的事务仅仅每秒才flush一次到redo日志，但对于一些场景是可以接受的，比如对于主节点的备份节点这个值是可以接受的。如果值为0速度就更快了，但在系统崩溃时可能丢失一些数据：只适用于备份节点。   
　　**innodb\_flush\_method**: 这项配置决定了数据和日志写入硬盘的方式。一般来说，如果你有硬件RAID控制器，并且其独立缓存采用write-back机制，并有着电池断电保护，那么应该设置配置为O\_DIRECT；否则，大多数情况下应将其设为fdatasync（默认值）。sysbench是一个可以帮助你决定这个选项的好工具。   
　　innodb\_log\_buffer\_size: 这项配置决定了为尚未执行的事务分配的缓存。其默认值（1MB）一般来说已经够用了，但是如果你的事务中包含有二进制大对象或者大文本字段的话，这点缓存很快就会被填满并触发额外的I/O操作。看看Innodb\_log\_waits状态变量，如果它不是0，增加innodb\_log\_buffer\_size。   
　　其他设置   
　　**query\_cache\_size**: query cache（查询缓存）是一个众所周知的瓶颈，甚至在并发并不多的时候也是如此。 最佳选项是将其从一开始就停用，设置query\_cache\_size = 0（现在MySQL 5.6的默认值）并利用其他方法加速查询：优化索引、增加拷贝分散负载或者启用额外的缓存（比如memcache或**[Redis](http://lib.csdn.net/base/redis" \o "Redis知识库" \t "http://blog.csdn.net/derrantcm/article/details/_blank)**）。如果你已经为你的应用启用了query cache并且还没有发现任何问题，query cache可能对你有用。这是如果你想停用它，那就得小心了。   
　　**log\_bin**：如果你想让数据库服务器充当主节点的备份节点，那么开启二进制日志是必须的。如果这么做了之后，还别忘了设置server\_id为一个唯一的值。就算只有一个服务器，如果你想做基于时间点的数据恢复，这（开启二进制日志）也是很有用的：从你最近的备份中恢复（全量备份），并应用二进制日志中的修改（增量备份）。二进制日志一旦创建就将永久保存。所以如果你不想让磁盘空间耗尽，你可以用 PURGE BINARY LOGS 来清除旧文件，或者设置 expire\_logs\_days 来指定过多少天日志将被自动清除。   
　　记录二进制日志不是没有开销的，所以如果你在一个非主节点的复制节点上不需要它的话，那么建议关闭这个选项。   
　　**skip\_name\_resolve**：当客户端连接数据库服务器时，服务器会进行主机名解析，并且当DNS很慢时，建立连接也会很慢。因此建议在启动服务器时关闭skip\_name\_resolve选项而不进行DNS查找。唯一的局限是之后GRANT语句中只能使用IP地址了，因此在添加这项设置到一个已有系统中必须格外小心。

## ****086 MySQL调优?****

**I 硬件配置优化**   
　　　　Ø CPU选择：多核的CPU，主频高的CPU   
　　　　Ø 内存：更大的内存   
　　　　Ø 磁盘选择：更快的转速、RAID、阵列卡，   
　　　　Ø 网络环境选择：尽量部署在局域网、SCI、光缆、千兆网、双网线提供冗余、0.0.0.0多端口绑定监听   
　　**II[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "http://blog.csdn.net/derrantcm/article/details/_blank)级优化**   
　　　　Ø 使用64位的操作系统，更好的使用大内存。   
　　　　Ø 设置noatime,nodiratime   
　　　　Ø 优化内核参数   
　　　　Ø 加大文件描述符限制   
　　　　Ø 文件系统选择 xfs   
　　**III Mysql设计优化**   
　　　　**III.1 存储引擎的选择**   
　　　　　　Ø Myisam：数据库并发不大，读多写少,而且都能很好的用到索引，sql语句比较简单的应用，TB数据仓库   
　　　　　　Ø Innodb：并发访问大，写操作比较多，有外键、事务等需求的应用，系统内存较大。   
　　　　**III.2 命名规则**   
　　　　　　Ø 多数开发语言命名规则：比如MyAdress   
　　　　　　Ø 多数开源思想命名规则：my\_address   
　　　　　　Ø 避免随便命名   
　　　　**III.3 字段类型选择**   
　　　　字段类型的选择的一般原则：   
　　　　　　Ø 根据需求选择合适的字段类型，在满足需求的情况下字段类型尽可能小。   
　　　　　　Ø 只分配满足需求的最小字符数，不要太慷慨。 原因：更小的字段类型更小的字符数占用更少的内存，占用更少的磁盘空间，占用更少的磁盘IO，以及占用更少的带宽。   
　　　　对于varchar和char的选择要根据引擎和具体情况的不同来选择，主要依据如下原则:   
　　　　　　1.如果列数据项的大小一致或者相差不大，则使用char。   
　　　　　　2.如果列数据项的大小差异相当大，则使用varchar。   
　　　　　　3.对于MyISAM表，尽量使用Char，对于那些经常需要修改而容易形成碎片的myisam和isam数据表就更是如此，它的缺点就是占用磁盘空间。   
　　　　　　4.对于InnoDB表，因为它的数据行内部存储格式对固定长度的数据行和可变长度的数据行不加区分（所有数据行共用一个表头部分，这个标头部分存放着指向各有关数据列的指针），所以使用char类型不见得会比使用varchar类型好。事实上，因为char类型通常要比varchar类型占用更多的空间，所以从减少空间占用量和减少磁盘i/o的角度，使用varchar类型反而更有利。   
　　　　　　5.表中只要存在一个varchar类型的字段，那么所有的char字段都会自动变成varchar类型，因此建议定长和变长的数据分开。   
　　　　**III.4 编码选择**   
　　　　　　单字节 latin1   
　　　　　　多字节 utf8(汉字占3个字节，英文字母占用一个字节)如果含有中文字符的话最好都统一采用utf8类型，避免乱码的情况发生。   
　　　　**III.5 主键选择原则**   
　　　　注：这里说的主键设计主要是针对INNODB引擎   
　　　　　　1.能唯一的表示行。   
　　　　　　2.显式的定义一个数值类型自增字段的主键，这个字段可以仅用于做主键，不做其他用途。   
　　　　　　3.MySQL主键应该是单列的，以便提高连接和筛选操作的效率。   
　　　　　　4.主键字段类型尽可能小，能用SMALLINT就不用INT，能用INT就不用BIGINT。   
　　　　　　5.尽量保证不对主键字段进行更新修改，防止主键字段发生变化，引发数据存储碎片，降低IO性能。   
　　　　　　6.MySQL主键不应包含动态变化的数据，如时间戳、创建时间列、修改时间列等。   
　　　　　　7.MySQL主键应当有计算机自动生成。   
　　　　　　8.主键字段放在数据表的第一顺序。   
　　　　推荐采用数值类型做主键并采用auto\_increment属性让其自动增长。   
　　　　**III.6 其他需要注意的地方**   
　　　　　　Ø NULL OR NOT NULL   
　　　　　　尽可能设置每个字段为NOT NULL，除非有特殊的需求，原因如下：   
　　　　　　　　1.使用含有NULL列做索引的话会占用更多的磁盘空间，因为索引NULL列需要而外的空间来保存。   
　　　　　　　　2.进行比较的时候，程序会更复杂。   
　　　　　　　　3.含有NULL的列比较特殊，SQL难优化，如果是一个组合索引，那么这个NULL 类型的字段会极大影响整个索引的效率。   
　　　　　　Ø 索引   
　　　　　　　　索引的优点：极大地加速了查询，减少扫描和锁定的数据行数。   
　　　　　　　　索引的缺点：占用磁盘空间，减慢了数据更新速度，增加了磁盘IO。   
　　　　　　添加索引有如下原则：   
　　　　　　　　1 选择唯一性索引。   
　　　　　　　　2.为经常需要排序、分组和联合操作的字段建立索引。   
　　　　　　　　3.为常作为查询条件的字段建立索引。   
　　　　　　　　4.限制索引的数据，索引不是越多越好。   
　　　　　　　　5.尽量使用数据量少的索引，对于大字段可以考虑前缀索引。   
　　　　　　　　6.删除不再使用或者很少使用的索引。   
　　　　　　　　7.结合核心SQL优先考虑覆盖索引。   
　　　　　　　　8.忌用字符串做主键。   
　　　　　　Ø 反范式设计   
　　　　　　适当的使用冗余的反范式设计，以空间换时间有的时候会很高效。   
　　**IV Mysql软件优化**   
　　　　Ø 开启mysql复制，实现读写分离、负载均衡，将读的负载分摊到多个从服务器上，提高服务器的处理能力。   
　　　　Ø 使用推荐的GA版本，提升性能   
　　　　Ø 利用分区新功能进行大数据的数据拆分   
　　**V Mysql配置优化**   
　　注意：全局参数一经设置，随服务器启动预占用资源。   
　　　　Ø key\_buffer\_size参数   
　　　　mysql索引缓冲，如果是采用myisam的话要重点设置这个参数，根据（key\_reads/key\_read\_requests）判断   
　　　　Ø innodb\_buffer\_pool\_size参数   
　　　　INNODB 数据、索引、日志缓冲最重要的引擎参数，根据（hit riatos和FILE I/O）判断   
　　　　Ø wait\_time\_out参数   
　　　　线程连接的超时时间，尽量不要设置很大，推荐10s   
　　　　Ø max\_connections参数   
　　　　服务器允许的最大连接数，尽量不要设置太大，因为设置太大的话容易导致内存溢出   
　　　　Ø thread\_concurrency参数   
　　　　线程并发利用数量，(cpu+disk)\*2,根据(os中显示的请求队列和tickets)判断   
　　　　Ø sort\_buffer\_size参数   
　　　　获得更快的–ORDER BY,GROUP BY,SELECT DISTINCT,UNION DISTINCT   
　　　　Ø read\_rnd\_buffer\_size参数   
　　　　当根据键进行分类操作时获得更快的–ORDER BY   
　　　　Ø join\_buffer\_size参数   
　　　　join连接使用全表扫描连接的缓冲大小，根据select\_full\_join判断   
　　　　Ø read\_buffer\_size参数   
　　　　全表扫描时为查询预留的缓冲大小，根据select\_scan判断   
　　　　Ø tmp\_table\_size参数   
　　　　临时内存表的设置，如果超过设置就会转化成磁盘表，根据参数(created\_tmp\_disk\_tables)判断   
　　　　Ø innodb\_log\_file\_size参数(默认5M)   
　　　　记录INNODB引擎的redo log文件，设置较大的值意味着较长的恢复时间。   
　　　　Ø innodb\_flush\_method参数(默认fdatasync)   
　　　　**[Linux](http://lib.csdn.net/base/linux" \o "Linux知识库" \t "http://blog.csdn.net/derrantcm/article/details/_blank)**系统可以使用O\_DIRECT处理数据文件，避免OS级别的cache，O\_DIRECT模式提高数据文件和日志文件的IO提交性能   
　　　　Ø innodb\_flush\_log\_at\_trx\_commit(默认1)   
　　　　　　1.0表示每秒进行一次log写入cache，并flush log到磁盘。   
　　　　　　2.1表示在每次事务提交后执行log写入cache，并flush log到磁盘。   
　　　　3.2表示在每次事务提交后，执行log数据写入到cache，每秒执行一次flush log到磁盘。   
　　**VI Mysql语句级优化**   
　　　　1.性能查的读语句，在innodb中统计行数,建议另外弄一张统计表，采用myisam，定期做统计.一般的对统计的数据不会要求太精准的情况下适用。   
　　　　2.尽量不要在数据库中做运算。   
　　　　3.避免负向查询和%前缀模糊查询。   
　　　　4.不在索引列做运算或者使用函数。   
　　　　5.不要在生产环境程序中使用select \* from 的形式查询数据。只查询需要使用的列。   
　　　　6.查询尽可能使用limit减少返回的行数，减少数据传输时间和带宽浪费。   
　　　　7.where子句尽可能对查询列使用函数，因为对查询列使用函数用不到索引。   
　　　　8.避免隐式类型转换，例如字符型一定要用’’，数字型一定不要使用’’。   
　　　　9.所有的SQL关键词用大写，养成良好的习惯，避免SQL语句重复编译造成系统资源的浪费。   
　　　　10.联表查询的时候，记得把小结果集放在前面，遵循小结果集驱动大结果集的原则。   
　　　　11.开启慢查询，定期用explain优化慢查询中的SQL语句。

## ****087 mysql是怎么备份的****

**一、备份的目的**   
　　　　**做灾难恢复**：对损坏的数据进行恢复和还原   
　　　　**需求改变**：因需求改变而需要把数据还原到改变以前   
　　　　**测试**：测试新功能是否可用   
　　**二、备份需要考虑的问题**   
　　　　可以容忍丢失多长时间的数据；   
　　　　恢复数据要在多长时间内完；   
　　　　恢复的时候是否需要持续提供服务；   
　　　　恢复的对象，是整个库，多个表，还是单个库，单个表。   
　　**三、备份的类型**   
　　　　**1、根据是否需要数据库离线**   
　　　　　　**冷备（cold backup）**：需要关mysql服务，读写请求均不允许状态下进行；   
　　　　　　**温备（warm backup）**： 服务在线，但仅支持读请求，不允许写请求；   
　　　　　　**热备（hot backup）**：备份的同时，业务不受影响。   
　　　　注：   
　　　　　　1、这种类型的备份，取决于业务的需求，而不是备份工具   
　　　　　　2、MyISAM不支持热备，InnoDB支持热备，但是需要专门的工具   
　　　　**2、根据要备份的数据集合的范围**   
　　　　　　**完全备份**：full backup，备份全部字符集。   
　　　　　　**增量备份**: incremental backup 上次完全备份或增量备份以来改变了的数据，不能单独使用，要借助完全备份，备份的频率取决于数据的更新频率。   
　　　　　　**差异备份**：differential backup 上次完全备份以来改变了的数据。   
　　　　　　建议的恢复策略：   
　　　　　　　　完全+增量+二进制日志   
　　　　　　　　完全+差异+二进制日志   
　　　　**3、根据备份数据或文件**   
　　　　　　**物理备份**：直接备份数据文件   
　　　　　　优点：备份和恢复操作都比较简单，能够跨mysql的版本，恢复速度快，属于文件系统级别的   
　　　　　　建议：不要假设备份一定可用，要测试mysql>check tables；检测表是否可用   
　　　　　　**逻辑备份**: 备份表中的数据和代码   
　　　　　　优点：恢复简单、备份的结果为ASCII文件，可以编辑与存储引擎无关可以通过网络备份和恢复   
　　　　　　缺点：备份或恢复都需要mysql服务器进程参与备份结果占据更多的空间，浮点数可能会丢失精度 还原之后，缩影需要重建   
　　**四：备份的对象**   
　　　　1、 数据；   
　　　　2、配置文件；   
　　　　3、代码：存储过程、存储函数、触发器   
　　　　4、os相关的配置文件   
　　　　5、复制相关的配置   
　　　　6、二进制日志   
　　**五、备份和恢复的实现**   
　　　　1、利用select into outfile实现数据的备份与还原。   
　　　　2、利用mysqldump工具对数据进行备份和还原   
　　　　3、利用lvm快照实现几乎热备的数据备份与恢复   
　　　　4、基于Xtrabackup做备份恢复。   
　　　　优势：   
　　　　　　1、快速可靠的进行完全备份   
　　　　　　2、在备份的过程中不会影响到事务   
　　　　　　3、支持数据流、网络传输、压缩，所以它可以有效的节约磁盘资源和网络带宽。   
　　　　　　4、可以自动备份校验数据的可用性。

## ****088 mysql 简单的 怎么登入 怎么创建数据库bbb创建 用户 密码 授权****

**一、创建用户**:   
　　CREATE USER用于创建新的MySQL账户。要使用CREATE USER，您必须拥有mysql数据库的全局CREATE USER权限，或拥有INSERT权限。对于每个账户，CREATE USER会在没有权限的mysql.user表中创建一个新记录。如果账户已经存在，则出现错误。   
　　使用自选的IDENTIFIED BY子句，可以为账户给定一个密码。user值和 密码的给定方法和GRANT语句一 样。特别是，要在纯文本中指定密码，需忽略PASSWORD关键词。要把 密码指定为由PASSWORD()函数返回的混编值，需包含关键字PASSWORD   
　　The create user command:mysql> CREATE USER yy IDENTIFIED BY ‘123’;   
　　面建立的用户可以在任何地方登陆。   
　　　　mysql> CREATE USER yy@localhost IDENTIFIED BY ‘123’;   
　　**二、授权**:   
　　命令:GRANT privileges ON databasename.tablename TO ‘username’@’host’   
　　说明: privileges - 用户的操作权限,如SELECT , INSERT , UPDATE 等.如果要授予所的权限则使   
　　用 ALL.;databasename - 数据库名,tablename-表名,如果要授予该用户对所有数据库和表的相应操   
　　作权限则可用表示, 如.\*.   
　　　　GRANT SELECT, INSERT ON test.user TO ‘pig’@’%’;   
　　　　GRANT ALL ON . TO ‘pig’@’%’;   
　　注意:用以上命令授权的用户不能给其它用户授权,如果想让该用户可以授权,用以下命令:   
　　GRANT privileges ON databasename.tablename TO ‘username’@’host’ WITH GRANT OPTION;   
　　刷新系统权限表   
　　flush privileges;   
　　**三、设置与更改用户密码**   
　　命令:SET PASSWORD FOR ‘username’@’host’ = PASSWORD(‘newpassword’);如果是当前登陆用户用SET PASSWORD = PASSWORD(“newpassword”);   
　　例子:SET PASSWORD FOR ‘pig’@’%’ = PASSWORD(“123456”);   
　　或：update mysql.user set password=password(‘新密码’) where User=”phplamp” and Host=”localhost”;   
　　**四、撤销用户权限**   
　　命令: REVOKE privilege ON databasename.tablename FROM ‘username’@’host’;   
　　说明: privilege, databasename, tablename - 同授权部分.   
　　例子: REVOKE SELECT ON . FROM ‘pig’@’%’;   
　　注意: 假如你在给用户’pig’@’%’授权的时候是这样的(或类似的):GRANT SELECT ON test.user TO ‘pig’@’%’, 则在使用 REVOKE SELECT ON . FROM ‘pig’@’%’;命令并不能撤销该用户对test数据库中user表的SELECT 操作.相反,如果授权使用的是GRANT SELECT ON . TO ‘pig’@’%’;则REVOKE SELECT ON test.user FROM ‘pig’@’%’;命令也不能撤销该用户对test数据库中user表的Select 权限.具体信息可以用命令SHOW GRANTS FOR ‘pig’@’%’; 查看.   
　　**五、删除用户**   
　　命令: DROP USER ‘username’@’host’;   
　　或：DELETE FROM user WHERE User=”phplamp” and Host=”localhost”;   
　　//删除用户的数据库   
　　mysql>drop database phplampDB;

## ****089 MySQL数据库同步怎样实现****

**1、安装配置**，两台服务器，分别安装好MySQL。采用单向同步的方式，就是Master的数据是主的数据，然后slave主动去Master哪儿同步数据回来。两台服务器的配置一样，把关键的配置文件拷贝一下，两台服务器做相同的拷贝配置文件操作。   
　　**2、配置Master服务器**，要考虑我们需要同步那个数据库，使用那个用户同步，我们这里为了简单起见，就使用root用户进行同步，并且只需要同步数据库abc。   
　　**3、配置Slave服务器**，我们的slave服务器主要是主动去Master服务器同步数据回来。   
　　**4、测试安装**，首先查看一下slave的主机日志：检查是否连接正常， 在Master查看信息，查看Master状态：查看Master下MySQL进程信息：在slave上查看信息：查看slave状态：查看slave下MySQL进程信息：再在Master的abc库里建立表结构并且插入数据，然后检查slave有没有同步这些数据，就能够检查出是否设置成功。

## ****090 查询mysql数据库中用户，密码，权限的命令****

　　查看MYSQL数据库中所有用户   
　　SELECT DISTINCT CONCAT(‘User: ”’,user,”’@”’,host,”’;’) AS query FROM mysql.user;   
　　查看数据库中具体某个用户的权限   
　　show grants for ‘cactiuser’@’%’;