1. **详细解释一下，主键，超键，候选键，外键。**

首先看看各种键的定义：

超键(super key):在关系中能唯一标识元组的属性集称为关系模式的超键

候选键(candidate key):不含有多余属性的超键称为候选键

主键(primary key):用户选作元组标识的一个候选键程序主键

外键(foreign key)如果关系模式R1中的某属性集不是R1的主键，而是另一个关系R2的主键则该属性集是关系模式R1的外键。

结合实例的具体解释：

假设有如下两个表：

学生（学号，姓名，性别，身份证号，教师编号）

教师（教师编号，姓名，工资）

超键：

由超键的定义可知，学生表中含有学号或者身份证号的任意组合都为此表的超键。如：（学号）、（学号，姓名）、（身份证号，性别）等。

候选键：

候选键属于超键，它是最小的超键，就是说如果再去掉候选键中的任何一个属性它就不再是超键了。学生表中的候选键为：（学号）、（身份证号）。

主键：

主键就是候选键里面的一个，是人为规定的，例如学生表中，我们通常会让“学号”做主键，教师表中让“教师编号”做主键。

外键：

外键比较简单，学生表中的外键就是“教师编号”。外键主要是用来描述两个表的关系。

其实这些键还是比较容易区分的，只是初学者一下子看到好几个名字相似的概念，容易自己把自己搞晕，自己给自己制造困难。多读两遍定义，然后结合实例很容易就能把它们区分开。希望这篇文章可以给一些初学者一点点的帮助。

1. **数据库事物的四个特效及含义**

数据库事务transanction正确执行的四个基本要素:ACID

原子性:整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。

一致性:在事务开始之前和事务结束以后，数据库的完整性约束没有被破坏。

隔离性:隔离状态执行事务，使它们好像是系统在给定时间内执行的唯一操作。如果有两个事务，运行在相同的时间内，执行 相同的功能，事务的隔离性将确保每一事务在系统中认为只有该事务在使用系统。这种属性有时称为串行化，为了防止事务操作间的混淆，必须串行化或序列化请 求，使得在同一时间仅有一个请求用于同一数据。

持久性:在事务完成以后，该事务所对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。

1. **详细解释一下视图的作用。请问，视图可以更改吗？**

测试表:user有id，name，age，sex字段

测试表:goods有id，name，price字段

测试表:ug有id，userid，goodsid字段

作用一：

提高了重用性，就像一个函数。如果要频繁获取user的name和goods的name。 就应该使用以下sql语言。示例：

select a.name as username, b.name as goodsname from user as a, goods as b, ug as c where a.id=c.userid and c.goodsid=b.id;

但有了视图就不一样了，创建视图other。示例

create view other as select a.name as username, b.name as goodsname from user as a, goods as b, ug as c where a.id=c.userid and c.goodsid=b.id;

创建好视图后，就可以这样获取user的name和goods的name。示例：

select \* from other;

以上sql语句，就能获取user的name和goods的name了。

作用二：

     对数据库重构，却不影响程序的运行。假如因为某种需求，需要将user拆房表usera 和表userb，该两张表的结构如下：

     测试表:usera有id，name，age字段

     测试表:userb有id，name，sex字段

    这时如果php端使用sql语句：select \* from user;那就会提示该表不存在，这时 该如何解决呢。解决方案：创建视图。以下sql语句创建视图：

create view user as select a.name,a.age,b.sex from usera as a, userb as b where a.name=b.name;

以上假设name都是唯一的。此时php端使用sql语句：select \* from user;就不会报错什么的。这就实现了更改数据库结构，不更改脚本程序的功能了。

作用三：

提高了安全性能。可以对不同的用户，设定不同的视图。例如：某用户只能获取user表的name和age数据，不能获取sex数据。则可以这样创建视图。示例如下：

create view other as select a.name, a.age from user as a;

    这样的话，使用sql语句：select \* from other; 最多就只能获取name和age的 数据，其他的数据就获取不了了。

作用四：

     让数据更加清晰。想要什么样的数据，就创建什么样的视图。经过以上三条作用的 解析，这条作用应该很容易理解了吧

对于某些视图比如未使用联结子查询分组聚集函数Distinct Union等，是可以对其更新的，对视图的更新将对基表进行更新；但是视图主要用于简化检索，保护数据，并不用于更新，而且大部分视图都不可以更新。

1. **详细解释一下drop,delete,truncate的区别**

不同点

1.truncate table 和 delete只删除记录不删除表的结构，drop语句将删除表的结构依赖的约束(constrain)，触发器(trigger),索引(index);

语句将删除表的结构被依赖的约束(constrain)，触发器(trigger)，索引(index);依赖于该表的存储过程/函数将保留，但是变为invalid状态

2.truncate之后的自增字段从头开始计数了，而delete的仍保留原来的最在数值。

总结

1.在速度上，一般来说，drop> truncate > delete。

2.在使用drop和truncate时一定要注意，虽然可以恢复，但为了减少麻烦，还是要慎重。

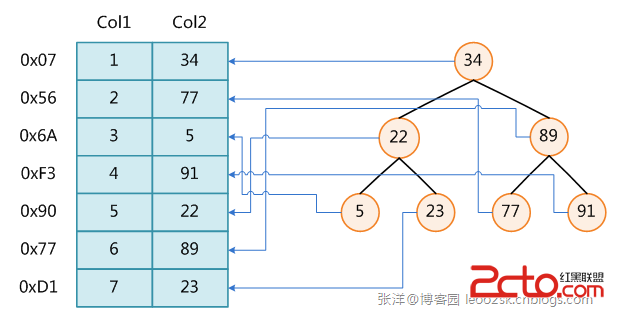
3.如果想删除部分数据用delete，注意带上where子句，回滚段要足够大；  
如果想删除表，当然用drop；  
如果想保留表而将所有数据删除，如果和事务无关，用truncate即可；  
如果和事务有关，或者想触发trigger，还是用delete  
如果是整理表内部的碎片，可以用truncate跟上reuse stroage，再重新导入/插入数据

1. **详细分析索引的工作原理及其种类**

数据库索引，是数据库管理系统中一个排序的数据结构，以协助快速查询、更[新数据](https://www.baidu.com/s?wd=%E6%96%B0%E6%95%B0%E6%8D%AE&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "/home/anwc/文档\\x/_blank)库表[中数据](https://www.baidu.com/s?wd=%E4%B8%AD%E6%95%B0%E6%8D%AE&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "/home/anwc/文档\\x/_blank)。索引的实现通常使用B树及其变种B+树。

在数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引。

为表设置索引要付出代价的：一是增加了数据库的存储空间，二是在插入和修改数据时要花费较多的时间(因为索引也要随之变动)。



图展示了一种可能的索引方式。左边是数据表，一共有两列七条记录，最左边的是数据记录的物理地址（注意逻辑上相邻的记录在磁盘上也并不是一定物理相邻的）。为了加快Col2的查找，可以维护一个右边所示的二叉查找树，每个节点分别包含索引键值和一个指向对应数据记录物理地址的指针，这样就可以运用二叉查找在O(log2n)的复杂度内获取到相应数据。

创建索引可以大大提高系统的性能。

第一，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

第二，可以大大加快数据的检索速度，这也是创建索引的最主要的原因。

第三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

第四，在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

第五，通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

也许会有人要问：增加索引有如此多的优点，为什么不对表中的每一个列创建一个索引呢？因为，增加索引也有许多不利的方面。

第一，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。

第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。

第三，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。

索引是建立在数据库表中的某些列的上面。在创建索引的时候，应该考虑在哪些列上可以创建索引，在哪些列上不能创建索引。一般来说，应该在这些列上创建索引：在经常需要搜索的列上，可以加快搜索的速度；在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；在经常使用在WHERE子句中的列上面创建索引，加快条件的判断速度。

同样，对于有些列不应该创建索引。一般来说，不应该创建索引的的这些列具有下列特点：

第一，对于那些在查询中很少使用或者参考的列不应该创建索引。这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。

第二，对于那些只有很少数据值的列也不应该增加索引。这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。

第三，对于那些定义为text, image和bit数据类型的列不应该增加索引。这是因为，这些列的数据量要么相当大，要么取值很少。

第四，当修改性能远远大于检索性能时，不应该创建索引。这是因为，修改性能和检索性能是互相矛盾的。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。

根据数据库的功能，可以在[数据库设计](http://www.2cto.com/database/" \t "/home/anwc/文档\\x/_blank)器中创建三种索引：唯一索引、主键索引和聚集索引。

唯一索引

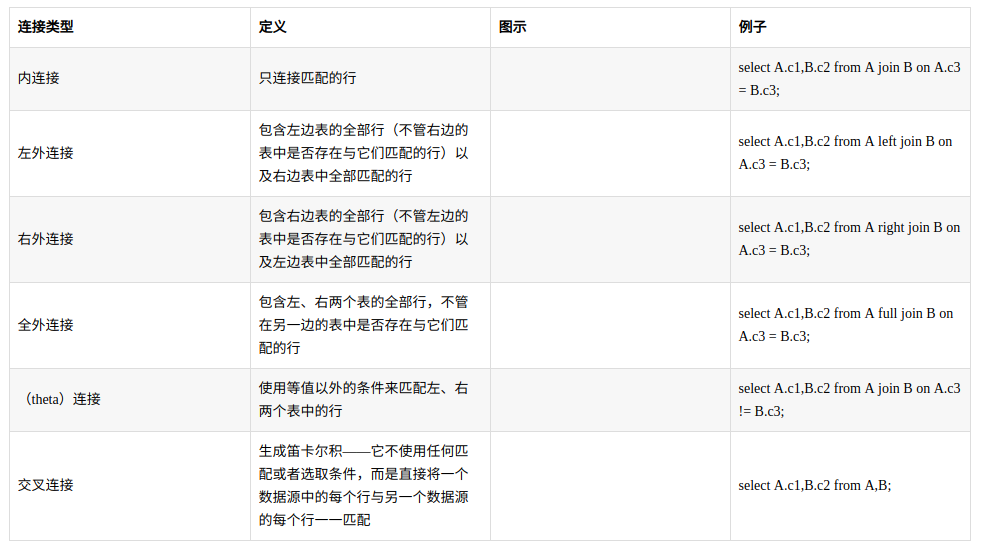
唯一索引是不允许其中任何两行具有相同索引值的索引。

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。 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。 聚集索引 在聚集索引中，表[中行](https://www.baidu.com/s?wd=%E4%B8%AD%E8%A1%8C&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "/home/anwc/文档\\x/_blank)的物理顺序与键值的逻辑（索引）顺序相同。一个表只能包含一个聚集索引。

如果某索引不是聚集索引，则表中行的物理顺序与键值的逻辑顺序不匹配。与非聚集索引相比，聚集索引通常提供更快的数据访问速度。

1. **详细说明数据库连接查询的种类**

在关系代数中，连接运算是由一个笛卡尔积运算和一个选取运算构成的。连接的全部意义在于在水平方向上合并两个数据集合（通常是表）。下面是几种常用的连接类型：

****

1. **你知道数据库范式吗？详细说明一下**

什么是范式：简言之就是，数据库设计对数据的存储性能，还有开发人员对数据的操作都有莫大的关系。所以建立科学的，规范的的数据库是需要满足一些

规范的来优化数据数据存储方式。在关系型数据库中这些规范就可以称为范式。

什么是三大范式：

第一范式：当关系模式R的所有属性都不能在分解为更基本的数据单位时，称R是满足第一范式的，简记为1NF。满足第一范式是关系模式规范化的最低要

求，否则，将有很多基本操作在这样的关系模式中实现不了。

第二范式：如果关系模式R满足第一范式，并且R得所有非主属性都完全依赖于R的每一个候选关键属性，称R满足第二范式，简记为2NF。

第三范式：设R是一个满足第一范式条件的关系模式，X是R的任意属性集，如果X非传递依赖于R的任意一个候选关键字，称R满足第三范式，简记为3NF.

注：关系实质上是一张二维表，其中每一行是一个元组，每一列是一个属性

1. **数据库优化方案简单说一下？**

1、选取最适用的字段属性

MySQL可以很好的支持大数据量的存取，但是一般说来，数据库中的表越小，在它上面执行的查询也就会越快。因此，在创建表的时候，为了获得更好的性能，我们可以将表中字段的宽度设得尽可能小。

例如，在定义邮政编码这个字段时，如果将其设置为CHAR(255),显然给数据库增加了不必要的空间，甚至使用VARCHAR这种类型也是多余的，因为CHAR(6)就可以很好的完成任务了。同样的，如果可以的话，我们应该使用MEDIUMINT而不是BIGIN来定义整型字段。

另外一个提高效率的方法是在可能的情况下，应该尽量把字段设置为NOTNULL，这样在将来执行查询的时候，数据库不用去比较NULL值。  
对于某些文本字段，例如“省份”或者“性别”，我们可以将它们定义为ENUM类型。因为在MySQL中，ENUM类型被当作数值型数据来处理，而数值型数据被处理起来的速度要比文本类型快得多。这样，我们又可以提高数据库的性能。

2、使用连接（JOIN）来代替子查询(Sub-Queries)

MySQL从4.1开始支持SQL的子查询。这个技术可以使用SELECT语句来创建一个单列的查询结果，然后把这个结果作为过滤条件用在另一个查询中。例如，我们要将客户基本信息表中没有任何订单的客户删除掉，就可以利用子查询先从销售信息表中将所有发出订单的客户ID取出来，然后将结果传递给主查询，如下所示：

DELETEFROMcustomerinfo

WHERECustomerIDNOTin(SELECTCustomerIDFROMsalesinfo)

使用子查询可以一次性的完成很多逻辑上需要多个步骤才能完成的SQL操作，同时也可以避免事务或者表锁死，并且写起来也很容易。但是，有些情况下，子查询可以被更有效率的连接（JOIN）..替代。例如，假设我们要将所有没有订单记录的用户取出来，可以用下面这个查询完成：

SELECT\*FROMcustomerinfo

WHERECustomerIDNOTin(SELECTCustomerIDFROMsalesinfo)

如果使用连接（JOIN）..来完成这个查询工作，速度将会快很多。尤其是当salesinfo表中对CustomerID建有索引的话，性能将会更好，查询如下：

SELECT\*FROMcustomerinfo

LEFTJOINsalesinfoONcustomerinfo.CustomerID=salesinfo.CustomerID

WHEREsalesinfo.CustomerIDISNULL

连接（JOIN）..之所以更有效率一些，是因为MySQL不需要在内存中创建临时表来完成这个逻辑上的需要两个步骤的查询工作。

3、使用联合(UNION)来代替手动创建的临时表

MySQL从4.0的版本开始支持union查询，它可以把需要使用临时表的两条或更多的select查询合并的一个查询中。在客户端的查询会话结束的时候，临时表会被自动删除，从而保证数据库整齐、高效。使用union来创建查询的时候，我们只需要用UNION作为关键字把多个select语句连接起来就可以了，要注意的是所有select语句中的字段数目要想同。下面的例子就演示了一个使用UNION的查询。

SELECTName,PhoneFROMclientUNION

SELECTName,BirthDateFROMauthorUNION

SELECTName,SupplierFROMproduct

4、事务

尽管我们可以使用子查询（Sub-Queries）、连接（JOIN）和联合（UNION）来创建[各种各样](https://www.baidu.com/s?wd=%E5%90%84%E7%A7%8D%E5%90%84%E6%A0%B7&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "/home/anwc/文档\\x/_blank)的查询，但不是所有的数据库操作都可以只用一条或少数几条SQL语句就可以完成的。更多的时候是需要用到一系列的语句来完成某种工作。但是在这种情况下，当这个语句块中的某一条语句运行出错的时候，整个语句块的操作就会变得不确定起来。设想一下，要把某个数据同时插入两个相关联的表中，可能会出现这样的情况：第一个表中成功更新后，数据库突然出现意外状况，造成第二个表中的操作没有完成，这样，就会造成数据的不完整，甚至会破坏数据库中的数据。要避免这种情况，就应该使用事务，它的作用是：要么语句块中每条语句都操作成功，要么都失败。换句话说，就是可以保持数据库中数据的一致性和完整性。事物以BEGIN关键字开始，COMMIT关键字结束。在这之间的一条SQL操作失败，那么，ROLLBACK命令就可以把数据库恢复到BEGIN开始之前的状态。

BEGIN; INSERTINTOsalesinfoSETCustomerID=14;UPDATEinventorySETQuantity=11WHEREitem='book';COMMIT;

事务的另一个重要作用是当多个用户同时使用相同的数据源时，它可以利用锁定数据库的方法来为用户提供一种安全的访问方式，这样可以保证用户的操作不被其它的用户所干扰。

5、锁定表

尽管事务是维护数据库完整性的一个非常好的方法，但却因为它的独占性，有时会影响数据库的性能，尤其是在很大的应用系统中。由于在事务执行的过程中，数据库将会被锁定，因此其它的用户请求只能暂时等待直到该事务结束。如果一个数据库系统只有少数几个用户来使用，事务造成的影响不会成为一个太大的问题；但假设有[成千上万](https://www.baidu.com/s?wd=%E6%88%90%E5%8D%83%E4%B8%8A%E4%B8%87&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "/home/anwc/文档\\x/_blank)的用户同时访问一个数据库系统，例如访问一个电子商务网站，就会产生比较严重的响应延迟。

其实，有些情况下我们可以通过锁定表的方法来获得更好的性能。下面的例子就用锁定表的方法来完成前面一个例子中事务的功能。

LOCKTABLEinventoryWRITESELECTQuantityFROMinventoryWHEREItem='book';

...

UPDATEinventorySETQuantity=11WHEREItem='book';UNLOCKTABLES

这里，我们用一个select语句取出初始数据，通过一些计算，用update语句将新值更新到表中。包含有WRITE关键字的LOCKTABLE语句可以保证在UNLOCKTABLES命令被执行之前，不会有其它的访问来对inventory进行插入、更新或者删除的操作。

6、使用外键

锁定表的方法可以维护数据的完整性，但是它却不能保证数据的关联性。这个时候我们就可以使用外键。

例如，外键可以保证每一条销售记录都指向某一个存在的客户。在这里，外键可以把customerinfo表中的CustomerID映射到salesinfo表中CustomerID，任何一条没有合法CustomerID的记录都不会被更新或插入到salesinfo中。

CREATETABLEcustomerinfo( CustomerIDINTNOTNULL,PRIMARYKEY(CustomerID))TYPE=INNODB;

CREATETABLEsalesinfo( SalesIDINTNOTNULL,CustomerIDINTNOTNULL,

PRIMARYKEY(CustomerID,SalesID),

FOREIGNKEY(CustomerID)REFERENCEScustomerinfo(CustomerID)ONDELETECASCADE)TYPE=INNODB;

注意例子中的参数“ONDELETECASCADE”。该参数保证当customerinfo表中的一条客户记录被删除的时候，salesinfo表中所有与该客户相关的记录也会被自动删除。如果要在MySQL中使用外键，一定要记住在创建表的时候将表的类型定义为事务安全表InnoDB类型。该类型不是MySQL表的默认类型。定义的方法是在CREATETABLE语句中加上TYPE=INNODB。如例中所示。

7、使用索引

索引是提高数据库性能的常用方法，它可以令数据库服务器以比没有索引快得多的速度检索特定的行，尤其是在查询语句当中包含有MAX(),MIN()和ORDERBY这些命令的时候，性能提高更为明显。

那该对哪些字段建立索引呢？

一般说来，索引应建立在那些将用于JOIN,WHERE判断和ORDERBY排序的字段上。尽量不要对数据库中某个含有大量重复的值的字段建立索引。对于一个ENUM类型的字段来说，出现大量重复值是很有可能的情况

例如customerinfo中的“province”..字段，在这样的字段上建立索引将不会有什么帮助；相反，还有可能降低数据库的性能。我们在创建表的时候可以同时创建合适的索引，也可以使用ALTERTABLE或CREATEINDEX在以后创建索引。此外，MySQL从版本3.23.23开始支持全文索引和搜索。全文索引在MySQL中是一个FULLTEXT类型索引，但仅能用于MyISAM类型的表。对于一个大的数据库，将数据装载到一个没有FULLTEXT索引的表中，然后再使用ALTERTABLE或CREATEINDEX创建索引，将是非常快的。但如果将数据装载到一个已经有FULLTEXT索引的表中，执行过程将会非常慢。

8、优化的查询语句

绝大多数情况下，使用索引可以提高查询的速度，但如果SQL语句使用不恰当的话，索引将无法发挥它应有的作用。

下面是应该注意的几个方面。

首先，最好是在相同类型的字段间进行比较的操作。

在MySQL3.23版之前，这甚至是一个必须的条件。例如不能将一个建有索引的INT字段和BIGINT字段进行比较；但是作为特殊的情况，在CHAR类型的字段和VARCHAR类型字段的字段大小相同的时候，可以将它们进行比较。

其次，在建有索引的字段上尽量不要使用函数进行操作。

例如，在一个DATE类型的字段上使用YEAE()函数时，将会使索引不能发挥应有的作用。所以，下面的两个查询虽然返回的结果一样，但后者要比前者快得多。

1. 在搜索字符型字段时，我们有时会使用LIKE关键字和通配符，这种做法虽然简单，但却也是以牺牲系统性能为代价的。

**优化Mysql数据库的8个方法**

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)<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';  
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来代替。

**数据库SQL优化大总结之 百万级数据库优化方案**

1.对查询进行优化，要尽量避免全表扫描，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

最好不要给数据库留NULL，尽可能的使用 NOT NULL填充数据库.

备注、描述、评论之类的可以设置为 NULL，其他的，最好不要使用NULL。

不要以为 NULL 不需要空间，比如：char(100) 型，在字段建立时，空间就固定了， 不管是否插入值（NULL也包含在内），都是占用 100个字符的空间的，如果是varchar这样的变长字段， null 不占用空间。  
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，否则将引擎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，如果一个字段有索引，一个字段没有索引，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 Name = 'admin'

可以这样查询：

select id from t where num = 10

union all

select id from t where Name = 'admin'  
 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

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)

6.下面的查询也将导致全表扫描：

select id from t where name like ‘%abc%’

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

.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

select id from t where datediff(day,createdate,’2005-11-30′) = 0 -–‘2005-11-30’ --生成的id

应改为:

select id from t where name like 'abc%'

select id from t where createdate >= '2005-11-30' and createdate < '2005-12-1'

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.Update 语句，如果只更改1、2个字段，不要Update全部字段，否则频繁调用会引起明显的性能消耗，同时带来大量日志。  
 14.对于多张大数据量（这里几百条就算大了）的表JOIN，要先分页再JOIN，否则逻辑读会很高，性能很差。  
 15.select count(\*) from table；这样不带任何条件的count会引起全表扫描，并且没有任何业务意义，是一定要杜绝的。  
 16.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。  
 17.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。  
 18.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。  
 19.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。  
 20.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。  
 21.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）。  
 22. 避免频繁创建和删除临时表，以减少系统表资源的消耗。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量，若数据量过大，应该考虑相应需求是否合理。

实际案例分析：拆分大的 DELETE 或INSERT 语句，批量提交SQL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相应。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，可能不仅仅会让你的WEB服务崩溃，还可能会让你的整台服务器马上挂了。  
　　所以，如果你有一个大的处理，你一定把其拆分，使用 LIMIT oracle(rownum),sqlserver(top)条件是一个好的方法。下面是一个mysql示例：

while(1){

　　//每次只做1000条

　　 mysql\_query(“delete from logs where log\_date <= ’2012-11-01’ limit 1000”);

　　if(mysql\_affected\_rows() == 0){

　　 　　//删除完成，退出！

　　 　　break；

　　}

//每次暂停一段时间，释放表让其他进程/线程访问。

usleep(50000)

}

1. **数据库引擎都有哪些？彼此区别在哪？**

ISAM：ISAM是一个定义明确且历经时间考验的数据表格管理方法，它在设计之时就考虑到数据库被查询的次数要远大于更新的次数。因此，ISAM执行读取操作的速度很快，而且不占用大量的内存和存储资源。ISAM的两个主要不足之处在于，它不支持事务处理，也不能够容错：如果你的硬盘崩溃了，那么数据文件就无法恢复了。如果你正在把ISAM用在关键任务应用程序里，那就必须经常备份你所有的实时数据，通过其复制特性，MYSQL能够支持这样的备份应用程序。  
 MyISAM：MyISAM是MySQL的ISAM扩展格式和缺省的数据库[引擎](https://www.baidu.com/s?wd=%E5%BC%95%E6%93%8E&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "/home/anwc/文档\\x/_blank)。除了提供ISAM里所没有的索引和字段管理的大量功能，MyISAM还使用一种表格锁定的机制，来优化多个并发的读写操作，其代价是你需要经常运行OPTIMIZE TABLE命令，来恢复被更新机制所浪费的空间。MyISAM还有一些有用的扩展，例如用来修复数据库文件的MyISAMCHK工具和用来恢复浪费空间的 MyISAMPACK工具。MYISAM强调了快速读取操作，这可能就是为什么MySQL受到了WEB开发如此青睐的主要原因：在WEB开发中你所进行的大量数据操作都是读取操作。所以，大多数虚拟主机提供商和INTERNET平台提供商只允许使用MYISAM格式。MyISAM格式的一个重要缺陷就是不能在表损坏后恢复数据。

InnoDB：InnoDB数据库引擎都是造就MySQL灵活性的技术的直接产品，这项技术就是MYSQL+API。在使用MYSQL的时候，你所面对的每一个挑战几乎都源于ISAM和MyISAM数据库引擎不支持事务处理（transaction process）也不支持外来键。尽管要比ISAM和 MyISAM引擎慢很多，但是InnoDB包括了对事务处理和外来键的支持，这两点都是前两个引擎所没有的。如前所述，如果你的设计需要这些特性中的一者或者两者，那你就要被迫使用后两个引擎中的一个了。

MEMORY: MEMORY是MySQL中一类特殊的存储引擎。它使用存储在内存中的内容来创建表，而且数据全部放在内存中。这些特性与前面的两个很不同。每个基于MEMORY存储引擎的表实际对应一个[磁盘文件](https://www.baidu.com/s?wd=%E7%A3%81%E7%9B%98%E6%96%87%E4%BB%B6&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "/home/anwc/文档\\x/_blank)。该文件的文件名与表名相同，类型为frm类型。该文件中只存储表的结构。而其数据文件，都是存储在内存中，这样有利于数据的快速处理，提高整个表的效率。值得注意的是，[服务器](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "/home/anwc/文档\\x/_blank)需要有足够的内存来维持MEMORY存储引擎的表的使用。如果不需要了，可以释放内存，甚至删除不需要的表。MEMORY默认使用哈希索引。速度比使用B型树索引快。当然如果你想用B型树索引，可以在创建索引时指定。注意，MEMORY用到的很少，因为它是把数据存到内存中，如果内存出现异常就会影响数据。如果重启或者关机，所有数据都会消失。因此，基于MEMORY的表的生命周期很短，一般是一次性的。

在实际工作中，选择一个合适的存储引擎是一个比较复杂的问题。每种存储引擎都有自己的优缺点，不能笼统地说谁比谁好。

InnoDB：支持事务处理，支持外键，支持崩溃修复能力和并发控制。如果需要对事务的完整性要求比较高（比如银行），要求实现并发控制（比如售票），那选择InnoDB有很大的优势。如果需要频繁的更新、删除操作的数据库，也可以选择InnoDB，因为支持事务的提交（commit）和回滚（rollback）。

MyISAM：插入数据快，空间和内存使用比较低。如果表主要是用于插入新记录和读出记录，那么选择MyISAM能实现处理高效率。如果应用的完整性、并发性要求比较低，也可以使用。

MEMORY：所有的数据都在内存中，数据的处理速度快，但是安全性不高。如果需要很快的读写速度，对数据的安全性要求较低，可以选择MEMOEY。它对表的大小有要求，不能建立太大的表。所以，这类数据库只使用在相对较小的数据库表。

注意，同一个数据库也可以使用多种存储引擎的表。如果一个表要求比较高的事务处理，可以选择InnoDB。这个数据库中可以将查询要求比较高的表选择MyISAM存储。如果该数据库需要一个用于查询的临时表，可以选择MEMORY存储引擎。

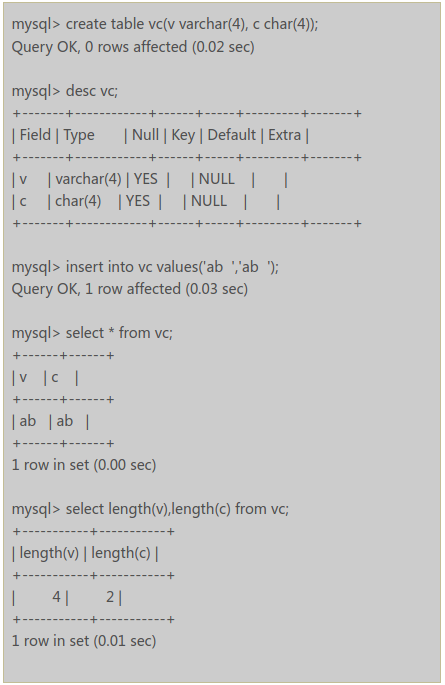
1. **MySQL中varchar与char的区别以及varchar(50)中的50代表的涵义**

CHAR是一种固定长度的类型，VARCHAR则是一种可变长度的类型。

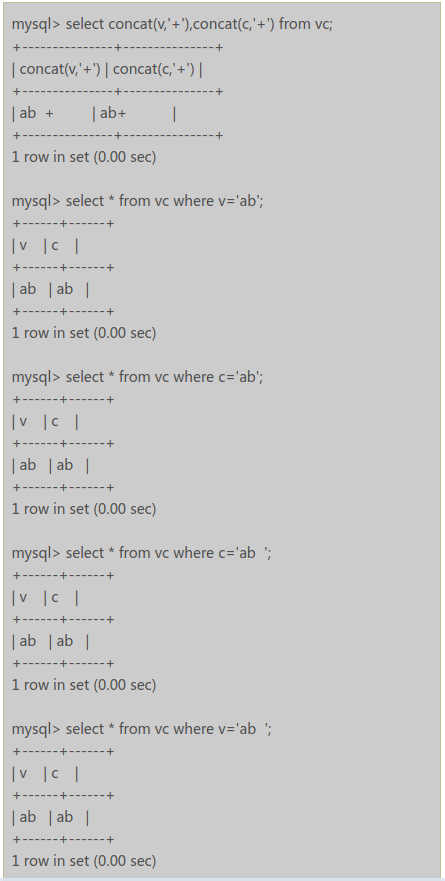
CHAR列的长度固定为创建表时声明的长度。长度可以为从0到255的任何值。当保存CHAR值时，在它们的右边填充空格以达到指定的长度。当检索到CHAR值时，尾部的空格被删除掉。在存储或检索过程中不进行大小写转换。

VARCHAR列中的值为可变长字符串。长度可以指定为0到65535之间的值。VARCHAR的最大有效长度由最大行大小和使用的字符集确定。在MySQL 4.1之前的版本，VARCHAR(50)的“50”指的是50字节（bytes）。如果存放UTF8汉字时，那么最多只能存放16个（每个汉字3字节）。从MySQL 4.1版本开始，VARCHAR(50)的“50”指的是50字符（character），无论存放的是数字、字母还是UTF8汉字（每个汉字3字节），都可以存放50个。

CHAR和VARCHAR类型声明的长度表示保存的最大字符数。例如，CHAR(30)可以占用30个字符。对于MyISAM表，推荐CHAR类型；对于InnoDB表，推荐VARCHAR类型。另外，在进行检索的时候，若列值的尾部含有空格，则CHAR列会删除其尾部的空格，而VARCHAR则会保留空格。如下所示：



可以看到，c列的length只有2，下面给字段加上“+”：



1. **数据库中有哪些种日志？**

错误日志

MySQL的错误日志（error log）记录MySQL服务进程mysqld在启动/关闭或运行过程中遇到的错误信息；

开启此日志功能需要在配置文件添加或者mysql中添加

# 配置文件中添加

[mysqld\_safe]

log-error=/data/3306/mysql\_oldboy3306.err

# mysq中添加

mysqld\_safe--default-file=/data/3306/my.cnf--log-error=/data/3306/mysql\_oldboy.err &

错误日志排查思路

1、先把日志文件备份并清空启动一下mysql服务，然后再查看日志文件报有什么错误

2、然后把mysql的3306目录下面所有文件都加上属主并且递归-R

3、然后查看一下有没有这个mysql ID

4、重新启动mysql服务

5、此时再查看一下服务有没有启动起来

慢查询日志

慢查询日志(show query log)记录执行时间超出指定值(long\_query\_time)的SQL语句；

慢查询日志（show query log）调整：

vim /data/3306/my.cnf

long\_query\_time = 1

log-slow-queries = /data/3306/slow.log

log\_queries\_not\_using\_indexes

慢查询的设置，对于数据库SQL的优化很重要

egrep "query" /data/3306/my.cnf|tail -3

二进制日志

二进制日志（binary log）：记录数据被修改的相关信息；

二进制日志（binary log）调整：

mysql> show variables like '%log\_bin%';

+---------------------------------+-------+

| Variable\_name | Value |

+---------------------------------+-------+

| log\_bin | ON | # 记录binlog开关

| log\_bin\_trust\_function\_creators | OFF |

| sql\_log\_bin | ON | # 临时不记录binlog开关（增量恢复）

+---------------------------------+-------+

1. **你了解主从一致性校验么？简单说说**

“MySQL主从复制”技术在互联网行业常见高可用架构中应用非常广泛，例如常见的一主一从复制架构、keepalived＋MySQL双主（主从）复制架构、MHA＋一主两从复制架构等等都应用了MySQL主从复制技术。但因主从复制是基于binlog的逻辑复制，难免出现复制数据不一致的风险，这个风险不但会引起用户数据访问前后不一致的风险，而且会导致后续复制出现1032、1062错误进而引起复制架构停滞的隐患，为了及时发现并解决这个问题，我们需要定期或不定期地开展主从复制数据一致性的校验和修复工作，那么如何实现这项工作呢？又如何实现这项工作的自动化呢？

数据一致性校验和修复方法

为了实现主从复制数据一致性校验和修复，我们首先推荐两个热门工具，分别是percona公司的 pt-table-checksum和pt-table-sync，前者用来实现主从复制数据一致性的校验，后者实现数据修复，将数据修复到一致。

1）在主库创建校验账号

GRANTUPDATE,INSERT,DELETE,SELECT, PROCESS, SUPER, REPLICATION SLAVE ON \*.\* TO 'hangxing'@'MasterIP'identified by 'PASSWORD';

GRANTALL ON test.\* TO 'hangxing'@'MasterIP' IDENTIFIED BY 'PASSWORD';

（2）在主库创建校验信息表

CREATETABLE IF NOT EXISTS checksums (

db char(64)NOT NULL,

tblchar(64) NOT NULL,

chunk intNOT NULL,

chunk\_timefloat NULL,

chunk\_indexvarchar(200) NULL,

lower\_boundarytext NULL,

upper\_boundarytext NULL,

this\_crcchar(40) NOT NULL,

this\_cntint NOT NULL,

master\_crcchar(40) NULL,

master\_cntint NULL,

tstimestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

PRIMARY KEY(db, tbl, chunk),

INDEXts\_db\_tbl (ts, db, tbl)

)ENGINE=InnoDB;

（3）判断主键

若无主键做校验和修复对性能影响非常重，数据校验和修复最重要的约束便是主健，无主键或唯一索引，将导致修复不成功。

主键判断语句：

SELECTDISTINCT CONCAT(t.table\_schema,'.',t.table\_name) astbl,t.engine,IF(ISNULL(c.constraint\_name),'NOPK','') AS nopk,IF(s.index\_type ='FULLTEXT','FULLTEXT','') as ftidx,IF(s.index\_type = 'SPATIAL','SPATIAL','') asgisidx FROM information\_schema.tables AS t LEFT JOINinformation\_schema.key\_column\_usage AS c ON (t.table\_schema =c.constraint\_schema AND t.table\_name = c.table\_name AND c.constraint\_name ='PRIMARY') LEFT JOIN information\_schema.statistics AS s ON (t.table\_schema =s.table\_schema AND t.table\_name = s.table\_name AND s.index\_type IN('FULLTEXT','SPATIAL')) WHERE t.table\_schema NOT IN('information\_schema','performance\_schema','mysql') AND t.table\_type = 'BASETABLE' AND (t.engine <> 'InnoDB' OR c.constraint\_name IS NULL ORs.index\_type IN ('FULLTEXT','SPATIAL')) ORDER BY t.table\_schema,t.table\_name;

（4）主从数据校验

 主从数据校验使用pt-table-checksum实现，要在主库上执行，执行校验通过参数控制校验全库全表还是只校验核心表。

校验指令举例：

./pt-table-checksum--nocheck-binlog-format --nocheck-plan --nocheck-replication-filters--replicate=test.checksums  --databases=db1--tables=tb1 -h 192.168.XXX.XX -P 3306-u'hangxing' -p'PASSOWRD' --recursion-method="processlist"

解析：

--no-check-binlog-format 不检查复制的binlog模式。

 --nocheck-replication-filters 不检查复制过滤器，建议启用。

--replicate=test.checksums 检查结果写入test库的checksums表里。

--databases=db1  --tables=tb1 校验db1库里的tb1表，若无参数则校验全库全表。

-h 192.168.XXX.XX -P 3306 主库IP地址和3306端口。

-u'hangxing' -p'PASSOWRD' 校验账号密码。

--recursion-method="processlist" 用processlist的方法来发现从库。

执行后的输出结果：

TS   ERRORS      DIFFS      ROWS  CHUNKS  SKIPPED   TIME  TABLE

03-23T15:29:17    0     1    30000       1       0  1.270 testhx1.testhx1

解析：

TS            ：完成检查的时间。

ERRORS     ：检查时候发生错误和警告的数量。

DIFFS        ：0表示一致，大于0表示不一致。主要看这一列有无不一致数据。

ROWS       ：表的行数。

CHUNKS    ：被划分到表中的块的数目。

SKIPPED    ：由于错误或警告或过大，则跳过块的数目。

TIME         ：执行的时间。

TABLE       ：被检查的表名。

1. **什么是乐观锁，什么是悲观锁？**

乐观锁不是数据库自带的，需要我们自己去实现。乐观锁是指操作数据库时(更新操作)，想法很乐观，认为这次的操作不会导致冲突，在操作数据时，并不进行任何其他的特殊处理（也就是不加锁），而在进行更新后，再去判断是否有冲突了。

与乐观锁相对应的就是悲观锁了。悲观锁就是在操作数据时，认为此操作会出现数据冲突，所以在进行每次操作时都要通过获取锁才能进行对相同数据的操作，这点跟java中的synchronized很相似，所以悲观锁需要耗费较多的时间。另外与乐观锁相对应的，悲观锁是由数据库自己实现了的，要用的时候，我们直接调用数据库的相关语句就可以了。

1. **创建GBK字符集的数据库NEWLHR，并查看已建库的完整语句**
2. **创建用户student,使之可以管理数据库student**
3. **查看创建的用户拥有哪些权限？**
4. **创建一innodb引擎字符集为GBK表test，字段为id和namevarchar(16)，查看建表结构及SQL语句**

以上SQL语句自主完成