**Sql优化**

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 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。

　　3.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

　　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.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。

　　6.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

　　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

　　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.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。

30.尽量避免大事务操作，提高系统并发能力。

**MySQL常见sql优化语句**

大批量插入数据

1. 对于Myisam 类型的表，可以通过以下方式快速的导入大量的数据。

ALTER TABLE tblname DISABLE KEYS;

loading the data

ALTER TABLE tblname ENABLE KEYS;

这两个命令用来打开或者关闭Myisam 表非唯一索引的更新。在导入大量的数据到一个非空的Myisam 表时，通过设置这两个命令，可以提高导入的效率。对于导入大量数据到一个空的Myisam 表，默认就是先导入数据然后才创建索引的，所以不用进行设置。

2. 而对于Innodb 类型的表，这种方式并不能提高导入数据的效率。对于Innodb 类型的表，我们有以下几种方式可以提高导入的效率：  www.2cto.com

a. 因为Innodb 类型的表是按照主键的顺序保存的，所以将导入的数据按照主键的顺序排列，可以有效的提高导入数据的效率。如果Innodb 表没有主键，那么[系统](http://www.2cto.com/os/" \t "http://www.2cto.com/database/201208/_blank)会默认创建一个内部列作为主键，所以如果可以给表创建一个主键，将可以利用这个优势提高导入数据的效率。

b. 在导入数据前执行SET UNIQUE\_CHECKS=0，关闭唯一性校验，在导入结束后执行SETUNIQUE\_CHECKS=1，恢复唯一性校验，可以提高导入的效率。

c. 如果应用使用自动提交的方式，建议在导入前执行SET AUTOCOMMIT=0，关闭自动提交，导入结束后再执行SET AUTOCOMMIT=1，打开自动提交，也可以提高导入的效率。

优化insert语句

如果你同时从同一客户插入很多行，使用多个值表的INSERT 语句。这比使用分开 INSERT 语句快(在一些情况中几倍)。Insert into test values(1,2),(1,3),(1,4)…

如果你从不同客户插入很多行，能通过使用INSERT DELAYED 语句得到更高的速度。Delayed 的含义是让insert 语句马上执行，其实数据都被放在内存的队列中，并没有真正的写入磁盘；这比每条语句都分别插入要快的多；LOW\_PRIORITY刚好相反，在所有其他用户对表的读写完成后才进行插入。

将索引文件和数据文件分在不同的磁盘上存放（利用建表中的选项）；

如果进行批量插入，可以增加bulk\_insert\_buffer\_size 变量值的方法来提高速度，但是，这只能对myisam表使用；  www.2cto.com

当从一个文本文件装载一个表时，使用LOAD DATA INFILE。这通常比使用很多INSERT语句快20倍；

根据应用情况使用replace 语句代替insert；

根据应用情况使用ignore 关键字忽略重复记录。

优化Group By语句

默认情况下，MySQL 排序所有GROUP BY col1，col2，....。查询的方法如同在查询中指定ORDER BY col1，col2，...。如果显式包括一个包含相同的列的ORDER BY子句，MySQL 可以毫不减速地对它进行优化，尽管仍然进行排序。如果查询包括GROUP BY 但你想要避免排序结果的消耗，你可以指定ORDER BY NULL禁止排序。

优化Order by语句

在某些情况中，MySQL 可以使用一个索引来满足ORDER BY 子句，而不需要额外的排序。where 条件和order by 使用相同的索引，并且order by 的顺序和索引顺序相同，并且order by 的字段都是升序或者都是降序。

mysql的性能优化包罗甚广：  
索引优化，查询优化，查询缓存，服务器设置优化，操作系统和硬件优化，应用层面优化（web服务器，缓存）等等。这里的记录的优化技巧更适用于开发人员，都是从网络上收集和自己整理的，主要是查询语句上面的优化，其它层面的优化技巧在此不做记录。

查询的开销指标：

执行时间  
检查的行数  
返回的行数

建立索引的几个准则：

1、合理的建立索引能够加速数据读取效率，不合理的建立索引反而会拖慢数据库的响应速度。  
2、索引越多，更新数据的速度越慢。  
3、尽量在采用MyIsam作为引擎的时候使用索引（因为MySQL以BTree存储索引），而不是InnoDB。但MyISAM不支持Transcation。  
4、当你的程序和数据库结构/SQL语句已经优化到无法优化的程度，而程序瓶颈并不能顺利解决，那就是应该考虑使用诸如memcached这样的分布式缓存系统的时候了。  
5、习惯和强迫自己用EXPLAIN来分析你SQL语句的性能。

1. count的优化

比如：计算id大于5的城市  
a. select count(\*) from world.city where id > 5;  
b. select (select count(\*) from world.city) – count(\*) from world.city where id <= 5;  
a语句当行数超过11行的时候需要扫描的行数比b语句要多， b语句扫描了6行，此种情况下，b语句比a语句更有效率。当没有where语句的时候直接select count(\*) from world.city这样会更快，因为mysql总是知道表的行数。

2. 避免使用不兼容的数据类型。

例如float和int、char和varchar、binary和varbinary是不兼容的。数据类型的不兼容可能使优化器无法执行一些本来可以进行的优化操作。  
在程序中，保证在实现功能的基础上，尽量减少对数据库的访问次数；通过搜索参数，尽量减少对表的访问行数,最小化结果集，从而减轻网络负担；能够分开的操作尽量分开处理，提高每次的响应速度；在数据窗口使用SQL时，尽量把使用的索引放在选择的首列；算法的结构尽量简单；在查询时，不要过多地使用通配符如 SELECT \* FROM T1语句，要用到几列就选择几列如：SELECT COL1,COL2 FROM T1；在可能的情况下尽量限制尽量结果集行数如：SELECT TOP 300 COL1,COL2,COL3 FROM T1,因为某些情况下用户是不需要那么多的数据的。不要在应用中使用数据库游标，游标是非常有用的工具，但比使用常规的、面向集的SQL语句需要更大的开销；按照特定顺序提取数据的查找。

3. 索引字段上进行运算会使索引失效。

尽量避免在WHERE子句中对字段进行函数或表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：  
SELECT \* FROM T1 WHERE F1/2=100 应改为: SELECT \* FROM T1 WHERE F1=100\*2

4. 避免使用!=或＜＞、IS NULL或IS NOT NULL、IN ，NOT IN等这样的操作符.

因为这会使系统无法使用索引,而只能直接搜索表中的数据。例如: SELECT id FROM employee WHERE id != “B%” 优化器将无法通过索引来确定将要命中的行数,因此需要搜索该表的所有行。在in语句中能用exists语句代替的就用exists.

5. 尽量使用数字型字段.

一部分开发人员和数据库管理人员喜欢把包含数值信息的字段  
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回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。

6. 合理使用EXISTS,NOT EXISTS子句。如下所示：

1.SELECT SUM(T1.C1) FROM T1 WHERE (SELECT COUNT(\*)FROM T2 WHERE T2.C2=T1.C2>0)  
2.SELECT SUM(T1.C1) FROM T1WHERE EXISTS(SELECT \* FROM T2 WHERE T2.C2=T1.C2)  
两者产生相同的结果，但是后者的效率显然要高于前者。因为后者不会产生大量锁定的表扫描或是索引扫描。如果你想校验表里是否存在某条纪录，不要用count(\*)那样效率很低，而且浪费服务器资源。可以用EXISTS代替。如：  
IF (SELECT COUNT(\*) FROM table\_name WHERE column\_name = ‘xxx’)可以写成：IF EXISTS (SELECT \* FROM table\_name WHERE column\_name = ‘xxx’)

7. 能够用BETWEEN的就不要用IN

8. 能够用DISTINCT的就不用GROUP BY

9. 尽量不要用SELECT INTO语句。SELECT INTO 语句会导致表锁定，阻止其他用户访问该表。

10. 必要时强制查询优化器使用某个索引

SELECT \* FROM T1 WHERE nextprocess = 1 AND processid IN (8,32,45) 改成：  
SELECT \* FROM T1 (INDEX = IX\_ProcessID) WHERE nextprocess = 1 AND processid IN (8,32,45)  
则查询优化器将会强行利用索引IX\_ProcessID 执行查询。

11. 消除对大型表行数据的顺序存取

尽管在所有的检查列上都有索引，但某些形式的WHERE子句强迫优化器使用顺序存取。如：  
SELECT \* FROM orders WHERE (customer\_num=104 AND order\_num>1001) OR order\_num=1008  
解决办法可以使用并集来避免顺序存取：  
SELECT \* FROM orders WHERE customer\_num=104 AND order\_num>1001 UNION SELECT \* FROM orders WHERE order\_num=1008  
这样就能利用索引路径处理查询。【jacking 数据结果集很多，但查询条件限定后结果集不大的情况下，后面的语句快】

12. 尽量避免在索引过的字符数据中，使用非打头字母搜索。这也使得引擎无法利用索引。

见如下例子：  
SELECT \* FROM T1 WHERE NAME LIKE ‘%L%’  
SELECT \* FROM T1 WHERE SUBSTING(NAME,2,1)=’L’  
SELECT \* FROM T1 WHERE NAME LIKE ‘L%’  
即使NAME字段建有索引，前两个查询依然无法利用索引完成加快操作，引擎不得不对全表所有数据逐条操作来完成任务。而第三个查询能够使用索引来加快操作，不要习惯性的使用 ‘%L%’这种方式(会导致全表扫描)，如果可以使用`L%’相对来说更好;

13. 虽然UPDATE、DELETE语句的写法基本固定，但是还是对UPDATE语句给点建议：

a) 尽量不要修改主键字段。  
b) 当修改VARCHAR型字段时，尽量使用相同长度内容的值代替。  
c) 尽量最小化对于含有UPDATE触发器的表的UPDATE操作。  
d) 避免UPDATE将要复制到其他数据库的列。  
e) 避免UPDATE建有很多索引的列。  
f) 避免UPDATE在WHERE子句条件中的列。

14. 能用UNION ALL就不要用UNION

UNION ALL不执行SELECT DISTINCT函数，这样就会减少很多不必要的资源  
在跨多个不同的数据库时使用UNION是一个有趣的优化方法，UNION从两个互不关联的表中返回数据，这就意味着不会出现重复的行，同时也必须对数据进行排序，我们知道排序是非常耗费资源的，特别是对大表的排序。  
UNION ALL可以大大加快速度，如果你已经知道你的数据不会包括重复行，或者你不在乎是否会出现重复的行，在这两种情况下使用UNION ALL更适合。此外，还可以在应用程序逻辑中采用某些方法避免出现重复的行，这样UNION ALL和UNION返回的结果都是一样的，但UNION ALL不会进行排序。

15. 字段数据类型优化：

a. 避免使用NULL类型：NULL对于大多数数据库都需要特殊处理，MySQL也不例外，它需要更多的代码，更多的检查和特殊的索引逻辑，有些开发人员完全没有意识到，创建表时NULL是默认值，但大多数时候应该使用NOT NULL，或者使用一个特殊的值，如0，-1作为默认值。  
b. 尽可能使用更小的字段，MySQL从磁盘读取数据后是存储到内存中的，然后使用cpu周期和磁盘I/O读取它，这意味着越小的数据类型占用的空间越小，从磁盘读或打包到内存的效率都更好，但也不要太过执着减小数据类型，要是以后应用程序发生什么变化就没有空间了。修改表将需要重构，间接地可能引起代码的改变，这是很头疼的问题，因此需要找到一个平衡点。  
c. 优先使用定长型

16. 关于大数据量limit分布的优化见下面链接（当偏移量特别大时，limit效率会非常低）：

<http://ariyue.iteye.com/blog/553541>  
附上一个提高limit效率的简单技巧，在覆盖索引(覆盖索引用通俗的话讲就是在select的时候只用去读取索引而取得数据，无需进行二次select相关表)上进行偏移，而不是对全行数据进行偏移。可以将从覆盖索引上提取出来的数据和全行数据进行联接，然后取得需要的列，会更有效率，看看下面的查询：  
mysql> select film\_id, description from sakila.film order by title limit 50, 5;  
如果表非常大，这个查询最好写成下面的样子：  
mysql> select film.film\_id, film.description from sakila.film  
inner join(select film\_id from sakila.film order by title liimit 50,5) as film usinig(film\_id);

17. 程序中如果一次性对同一个表插入多条数据，比如以下语句：

insert into person(name,age) values(‘xboy’, 14);  
insert into person(name,age) values(‘xgirl’, 15);  
insert into person(name,age) values(‘nia’, 19);  
把它拼成一条语句执行效率会更高.  
insert into person(name,age) values(‘xboy’, 14), (‘xgirl’, 15),(‘nia’, 19);

18. 不要在选择的栏位上放置索引，这是无意义的。应该在条件选择的语句上合理的放置索引，比如where，order by。

SELECT id,title,content,cat\_id FROM article WHERE cat\_id = 1;

上面这个语句，你在id/title/content上放置索引是毫无意义的，对这个语句没有任何优化作用。但是如果你在外键cat\_id上放置一个索引，那作用就相当大了。

19. ORDER BY语句的MySQL优化：  
a. ORDER BY + LIMIT组合的索引优化。如果一个SQL语句形如：

SELECT [column1],[column2],…. FROM [TABLE] ORDER BY [sort] LIMIT [offset],[LIMIT];

这个SQL语句优化比较简单，在[sort]这个栏位上建立索引即可。

b. WHERE + ORDER BY + LIMIT组合的索引优化，形如：

SELECT [column1],[column2],…. FROM [TABLE] WHERE [columnX] = [VALUE] ORDER BY [sort] LIMIT [offset],[LIMIT];

这个语句，如果你仍然采用第一个例子中建立索引的方法，虽然可以用到索引，但是效率不高。更高效的方法是建立一个联合索引(columnX,sort)

c. WHERE + IN + ORDER BY + LIMIT组合的索引优化，形如：

SELECT [column1],[column2],…. FROM [TABLE] WHERE [columnX] IN ([value1],[value2],…) ORDER BY [sort] LIMIT [offset],[LIMIT];

这个语句如果你采用第二个例子中建立索引的方法，会得不到预期的效果（仅在[sort]上是using index，WHERE那里是using where;using filesort），理由是这里对应columnX的值对应多个。  
目前哥还木有找到比较优秀的办法，等待高手指教。

d.WHERE+ORDER BY多个栏位+LIMIT，比如:

SELECT \* FROM [table] WHERE uid=1 ORDER x,y LIMIT 0,10;

对于这个语句，大家可能是加一个这样的索引:(x,y,uid)。但实际上更好的效果是(uid,x,y)。这是由MySQL处理排序的机制造成的