 优化SQL步骤

### ****2 .其他一些注意优化：****

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).最好不要使用"\*"返回所有： 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。

### ****3. 临时表的问题：****

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。

补充：

1.  select \* into destTbl from srcTbl  
2.  Insert into destTbl(fld1, fld2) select fld1, 5 from srcTbl

以上两句都是将 srcTbl 的数据插入到 destTbl，但两句又有区别的。   
  
第一句（select into from）要求目标表（destTbl）不存在，因为在插入时会自动创建。   
  
第二句（insert into select from）要求目标表（destTbl）存在，由于目标表已经存在，所以我们除了插入源表（srcTbl）的字段外，还可以插入常量，如例中的：5。

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。

### ****4. 游标的问题：****

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 （set nocount on），在结束时设置 SET NOCOUNT OFF（ 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（done\_in\_proc） 消息。

### ****5. 事务的问题：****

29).尽量避免大事务操作，提高系统并发能力。

### 6. 数据量的问题

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/hguisu/article/details/_blank)**量，若数据量过大，应该考虑相应需求是否合理。

### ****7. COUNT优化：****

31) count(\*) 优于count(1)和count(primary\_key)

　　很多人为了统计记录条数，就使用 count(1) 和 count(primary\_key) 而不是 count(\*) ，他们认为这样性能更好，其实这是一个误区。对于有些场景，这样做可能性能会更差，应为数据库对 count(\*) 计数操作做了一些特别的优化。

32）count(column) 和 count(\*) 是不一样的

这个误区甚至在很多的资深工程师或者是 DBA 中都普遍存在，很多人都会认为这是理所当然的。实际上，count(column) 和 count(\*) 是一个完全不一样的操作，所代表的意义也完全不一样。

　　count(column) 是表示结果集中有多少个column字段不为空的记录

　　count(\*) 是表示整个结果集有多少条记录

1)innodb引擎在统计方面和myisam是不同的，Myisam内置了一个计数器

 Count(\*)在没有查询条件的情况下使用 select count(\*) from table 的时候，Myisam直接可以从计数器中取出数据。而innodb必须全表扫描一次方能得到总的数量

2. 但是当有查询条件的时候，两者的查询效率一致。

4. 主键索引count(\*)的时候之所以慢

InnoDB引擎:

[1]     数据文件和索引文件存储在一个文件中，主键索引默认直接指向数据存储位置。

[2]     二级索引存储指定字段的索引，实际的指向位置是主键索引。当我们通过二级索引统计数据的时候，无需扫描数据文件；而通过主键索引统计数据时，由于主键索引与数据文件存放在一起，所以每次都会扫描数据文件，所以主键索引统计没有二级索引效率高。

[3]     由于主键索引直接指向实际数据，所以当我们通过主键id查询数据时要比通过二级索引查询数据要快。

l  MyAsm引擎

[1]     该引擎把每个表都分为几部分存储，比如用户表，包含user.frm，user.MYD和user.MYI。

[2]     User.frm负责存储表结构

[3]     User.MYD负责存储实际的数据记录，所有的用户记录都存储在这个文件中

[4]     User.MYI负责存储用户表的所有索引，这里也包括主键索引。

### ****8. 优化order by语句****

     基于索引的排序  
     MySQL的弱点之一是它的排序。虽然MySQL可以在1秒中查询大约15,000条记录，但由于MySQL在查询时最多只能使用一个索引。因此，如果WHERE条件已经占用了索引，那么在排序中就不使用索引了，这将大大降低查询的速度。我们可以看看如下的SQL语句:  
SELECT \* FROM SALES WHERE NAME = “name” ORDER BY SALE\_DATE DESC;  
     在以上的SQL的WHERE子句中已经使用了NAME字段上的索引，因此，在对SALE\_DATE进行排序时将不再使用索引。为了解决这个问题，我们可以对SALES表建立复合索引:

 ALTER TABLE SALES DROP INDEX NAME, ADD INDEX (NAME,SALE\_DATE)

     这样再使用上述的SELECT语句进行查询时速度就会大副提升。但要注意，在使用这个方法时，要确保WHERE子句中没有排序字段，在上例中就是不能用SALE\_DATE进行查询，否则虽然排序快了，但是SALE\_DATE字段上没有单独的索引，因此查询又会慢下来。

     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。例如：下列sql可以使用索引。

     SELECT \* FROM t1 ORDER BY key\_part1,key\_part2,... ;  
     SELECT \* FROM t1 WHERE key\_part1=1 ORDER BY key\_part1 DESC, key\_part2 DESC;  
     SELECT \* FROM t1 ORDER BY key\_part1 DESC, key\_part2 DESC;

    但是以下情况不使用索引：  
     SELECT \* FROM t1 ORDER BY key\_part1 DESC, key\_part2 ASC ； --order by 的字段混合 ASC 和 DESC

     SELECT \* FROM t1 WHERE key2=constant ORDER BY key1 ；-- 用于查询行的关键字与 ORDER BY 中所使用的不相同

     SELECT \* FROM t1 ORDER BY key1, key2 ；-- 对不同的关键字使用 ORDER BY ：

### 9. 优化GROUP BY

      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。  
例如 ：  
INSERT INTO foo  SELECT a, COUNT(\*) FROM bar GROUP BY a ORDER BY NULL;

### 10. 优化 OR

具体详解看：[mysql or条件可以使用索引而避免全表](http://blog.csdn.net/hguisu/article/details/7106159" \t "http://blog.csdn.net/hguisu/article/details/_blank)

4. Explain解释说明

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：   
如：

explain **select** surname,first\_name form a,b **where** a.id=b.id

分析结果形式如下：   
**table |  type | possible\_keys | key | key\_len  | ref | rows | Extra   
EXPLAIN列的解释**：

### 1 table:

显示这一行的数据是关于哪张表的

### 2 type:

这是重要的列，显示连接使用了何种类型。从最好到最差的连接类型为：system、const、eg\_reg、ref、ref\_or\_null、 range、indexhe、 ALL。  
**system:**表仅有一行(=系统表)。这是const联接类型的一个特例  
       **const:(PRIMARY KEY或UNIQUE)**  
            表最多有一个匹配行，它将在查询开始时被读取。因为仅有一行，在这行的列值可被优化器剩余部分认为是常数。  
            const表很快，因为它们只读取一次！  
            const用于用常数值比较PRIMARY KEY或UNIQUE索引的所有部分时。  
            在下面的查询中，tbl\_name可以用于const表：

1. **SELECT** \* **from** tbl\_name **WHERE** primary\_key=1；

**eq\_reg:key**  
  对于每个来自于前面的表的行组合，从该表中读取一行。这可能是最好的联接类型，除了const类型。  
           它用在一个索引的所有部分被联接使用并且索引是UNIQUE或PRIMARY KEY。  
           eq\_ref可以用于使用= 操作符比较的带索引的列。比较值可以为常量或一个使用在该表前面所读取的表的列的表达式。  
 

5. SQL核心语句(非常实用的几个技巧)

**1) 插入数据**

**批量插入:**

1. **INSERT** mytable (first\_column,second\_column,third\_column)
2. **VALUES** ('some data','some more data','yet more data') ,
3. **VALUES** ('some data','some more data','yet more data') ,
4. **VALUES** ('some data','some more data','yet more data')

**2）.清空数据表**

**TRUNCATE**  **TABLE**  `mytable`  (truncate table)

注意：删除表中的所有记录，应使用TRUNCATE TABLE语句。注意这里为什么要用TRUNCATE TABLE语句代替DELETE语句:当你使用TRUNCATE TABLE语句时，记录的删除是不作记录的。也就是说，这意味着TRUNCATE TABLE要比DELETE快得多。

**3）用SELECT创建记录和表**

　　INSERT语句与DELETE语句和UPDATE语句有一点不同，它一次只操作一个记录。然而，有一个方法可以使INSERT 语句一次添加多个记录。要作到这一点，你需要把INSERT语句与SELECT语句结合起来，象这样:

1. **INSERT** mytable(first\_column,second\_column)
2. **SELECT** another\_first,another\_second  **FROM** anothertable **WHERE** another\_first='Copy Me!';

　　这个语句从anothertable拷贝记录到mytable.只有表anothertable中字段another\_first的值为'Copy Me!'的记录才被拷贝。

　　当为一个表中的记录建立备份时，这种形式的INSERT语句是非常有用的。在删除一个表中的记录之前，你可以先用这种方法把它们拷贝到另一个表中。

如果你需要拷贝整个表，你可以使用SELECT INTO语句。例如，下面的语句创建了一个名为newtable的新表，该表包含表mytable的所有数据:

1. **SELECT** \* **INTO** newtable **FROM** mytable;

　　你也可以指定只有特定的字段被用来创建这个新表。要做到这一点，只需在字段列表中指定你想要拷贝的字段。另外，你可以使用WHERE子句来限制拷贝到新表中的记录。下面的例子只拷贝字段second\_columnd的值等于'Copy Me!'的记录的first\_column字段。

1. **SELECT**  first\_column  **INTO**  newtable
2. **FROM**  mytable
3. **WHERE**  second\_column='Copy Me!';

　　使用SQL修改已经建立的表是很困难的。例如，如果你向一个表中添加了一个字段，没有容易的办法来去除它。另外，如果你不小心把一个字段的数据类型给错了，你将没有办法改变它。但是，使用本节中讲述的SQL语句，你可以绕过这两个问题。

　　例如，假设你想从一个表中删除一个字段。使用SELECT INTO语句，你可以创建该表的一个拷贝，但不包含要删除的字段。这使你既删除了该字段，又保留了不想删除的数据。

　　如果你想改变一个字段的数据类型，你可以创建一个包含正确数据类型字段的新表。创建好该表后，你就可以结合使用UPDATE语句和SELECT语句，把原来表中的所有数据拷贝到新表中。通过这种方法，你既可以修改表的结构，又能保存原有的数据。