**互斥锁（Mutex）**

互斥锁是一个互斥的同步对象，意味着同一时间有且仅有一个线程可以获取它。

互斥锁可适用于一个共享资源每次只能被一个线程访问的情况

using (Mutex mutex = new Mutex(false, "Global\\Sunnysoft\_" + Application.StartupPath.GetHashCode()))

{

if (mutex.WaitOne(0, false))

{

Application.Run(new MdiForm());

}

else

{

MsgHelper.ShowMsg("应用程序已经在运行中...");

Application.ExitThread();

Application.Exit();

System.Environment.Exit(0);

}

}

**Task类**

https://www.cnblogs.com/luxiaoxun/p/3280146.html

使用Task的好处是在操作完成后获得一个返回值，构造一个Task<TResult>对象，并为泛型TResult参数传递一个操作的返回类型。

namespace Test

{

class Program

{

static void Main(string[] args)

{

Task<int> t = new Task< int >(n => Sum((int)n), 1000);

t.Start();

t.Wait();

Console.WriteLine(t.Result);

Console.ReadKey();

}

private static int Sum(int n)

{

int sum = 0;

for (; n > 0; --n)

checked{ sum += n;} //结果太大，抛出异常

return sum;

}

}

}

一个任务完成时，自动启动一个新任务。

namespace Test

{

class Program

{

static void Main(string[] args)

{

Task<Int32> t = new Task<Int32>(n => Sum((Int32)n), 1000);

t.Start();

//t.Wait();

Task cwt = t.ContinueWith(task => Console.WriteLine("The result is {0}",t.Result));

Console.ReadKey();

}

private static Int32 Sum(Int32 n)

{

Int32 sum = 0;

for (; n > 0; --n)

checked{ sum += n;} //结果溢出，抛出异常

return sum;

}

}

}

Socket

<http://www.cnblogs.com/weilengdeyu/archive/2013/03/08/2949101.html>

HTTP

https://www.cnblogs.com/jiang08/articles/3946015.html

WebRequest myRequest = WebRequest.Create("xxx");

WebResponse myResponse = myRequest.GetResponse();

myResponse.Close();

IDisposable

https://www.cnblogs.com/tiancai/p/6612444.html

# [sql优化的几种方法](https://www.cnblogs.com/tiancai/p/9635511.html)

在sql查询中为了提高查询效率，我们常常会采取一些措施对查询语句进行sql优化，下面总结的一些方法，有需要的可以参考参考。

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，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.下面的查询也将导致全表扫描：

select id from t where name like '%abc%'

7.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

8.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%'

9.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。

10.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。

11.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(...)

12.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)

13.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。

14.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。

15.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。

16.尽可能的使用 varchar 代替 char ，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。

17.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。

18.避免频繁创建和删除临时表，以减少系统表资源的消耗。

19.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。

20.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。

21.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。

22.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。

23.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。

24.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。

25.尽量避免大事务操作，提高系统并发能力。

26.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。

# IN和EXISTS区别及应用场景

　　　　in 和 exists的区别: 如果子查询得出的结果集记录较少，主查询中的表较大且又有索引时应该用in, 反之如果外层的主查询记录较少，子查询中的表大，又有索引时使用exists。其实我们区分in和exists主要是造成了驱动顺序的改变(这是性能变化的关键)，如果是exists，那么以外层表为驱动表，先被访问，如果是IN，那么先执行子查询，所以我们会以驱动表的快速返回为目标，那么就会考虑到索引及结果集的关系了 ，另外IN时不对NULL进行处理。

　　　　in 是把外表和内表作hash 连接，而exists是对外表作loop循环，每次loop循环再对内表进行查询。一直以来认为exists比in效率高的说法是不准确的。

# not in 和not exists

　　　　如果查询语句使用了not in 那么内外表都进行全表扫描，没有用到索引；而not extsts 的子查询依然能用到表上的索引。所以无论那个表大，用not exists都比not in要快。

## 关于事务和锁

https://www.cnblogs.com/knowledgesea/p/3714417.html