**[临时表](http://uule.iteye.com/blog/2037913)**

**博客分类：**

* [数据库](http://uule.iteye.com/category/132649)

[SQL知识整理一：触发器、存储过程、表变量、临时表](http://www.cnblogs.com/chengxingliang/p/3307458.html)

**什么时候使用临时表？**  
在SQL Server的性能调优中,如何在一段需要长时间的代码或被频繁调用的代码中处理临时数据集,表变量和临时表是两种选择。  
  
临时表是有它的作用的,但不能滥用,**当一个查询涉及很多表时,查询的笛卡尔积是非常大的,单纯的用索引可能解决不了查询时间的问题.这个时候可以用临时表,将一个查询分成多步完成,减少输出查询结果时的笛卡尔积**,会对提高查询效率有很大的帮助.比如  
查询的主表有上百W条数据,条件都是针对主表字段,然后1:M:M关联两个子表,如果直接关联查询,那么笛卡尔积可能为100W\*M\*100W\*n\*100W,可能就会造成耗时很长或查询超时.用临时表先筛选出主表的数据,比如筛选取主表后还有1000条数据满足条件要求,这时再用临时表来关联两个子表,笛卡尔积就会下降一个几何积,会明显提高查询速度.  
什么时候用临时表,取决于查询的复杂成度和数据量,简单的查询用临时表就得不偿失了.   
  
如利用临时表来组织数据，比普通表会更加的简洁、紧凑。这主要是在临时表中可以实现很多的特性。如可以进行预处理计算。如当发现基本标中的索引不怎么合适，也可以在数据库临时表中重新创建索引以优化原有的索引。特别是当需要多次访问某个表或者视图的时候，利用临时表来组织数据是一个提高效率的好方法。即使只是一个简单的查询，其效率的提升也是很明显的。为此，使用临时表最明显的一个好处就似乎可以提高数据库的性能，特别是查询的性能。  
  
　　另外使用临时表还可以减少中间表的产生。在进行某些操作时，本来往往需要一些中间表的帮助才可以完成。而现在数据库管理员可以让数据库在需要时自动生成中间表，并在用完后进行自动删除。如此的话，中间表的建立与删除就不需要数据库管理员人为的管理了。所以，使用临时表可以减少数据库系统中的垃圾表，也可以降低用户的工作量。  
  
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\  
**临时表和表变量的作用域范围：**

       表变量的作用域范围是所在的批处理，而临时表的作用域范围是整个会话。还有表变量不会产生统计信息，这有两个主要后果：第一是查询优化器对表变量行数的估计为固定值而不管表变量中的数据是什么，第二添加或删除数据并不会改变估计的值。  
不能为表变量创建索引但是可以为临时表创建索引。  
  
**临时表:**  
临时对象都以#或##为前缀，临时表是临时对象的一种。还有例如临时存储过程、临时函数之类的临时对象，临时对象都存储在tempdb中。  
以#前缀的临时表为本地的，因此只有在当前用户会话中才可以访问，而##前缀的临时表是全局的，因此所有用户会话都可以访问。  
临时表以会话为边界,只要创建临时表的会话没有结束临时表就会持续存在。当然用户在会话中可以通过DROP TABLE命令提前销毁临时表。  
  
除非使用 DROP TABLE 语句显式除去临时表，否则临时表将在退出其作用域时由系统自动除去：所有其它本地临时表在当前会话结束时自动除去。  
  
当存储过程完成时，将自动除去在存储过程中创建的本地临时表。由创建表的存储过程执行的所有嵌套存储过程都可以引用此表。但调用创建此表的存储过程的进程无法引用此表。  
  
全局临时表在创建此表的会话结束且其它任务停止对其引用时自动除去。任务与表之间的关联只在单个 Transact-SQL 语句的生存周期内保持。换言之，当创建全局临时表的会话结束时，最后一条引用此表的 Transact-SQL 语句完成后，将自动除去此表。  
  
当创建本地或全局临时表时，CREATE TABLE 语法支持除 FOREIGN KEY 约束以外的其它所有约束定义。如果在临时表中指定 FOREIGN KEY 约束，该语句将返回警告信息，指出此约束已被忽略，表仍会创建，但不具有 FOREIGN KEY 约束。在 FOREIGN KEY 约束中不能引用临时表。  
  
考虑使用表变量而不使用临时表。当需要在临时表上显式地创建索引时，或多个存储过程或函数需要使用表值时，临时表很有用。通常，表变量提供更有效的查询处理。  
  
**SQL Server创建和使用临时表:**  
创建临时表  
    方法一：  
        create table #临时表名(字段1 约束条件,  
                      字段2 约束条件,  
                  .....)  
        create table ##临时表名(字段1 约束条件,  
                          字段2 约束条件,  
                      .....)  
                        
    方法二：  
        select \* into #临时表名 from 你的表;  
        select \* into ##临时表名 from 你的表;  
注：以上的#代表局部临时表，##代表全局临时表  
  
查询临时表  
    select \* from #临时表名;  
    select \* from ##临时表名;  
  
删除临时表  
    drop table #临时表名;  
    drop table ##临时表名;   
  
      
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\  
**临时表的存储：**  
我们前面说过临时表存储在tempdb中因此临时表的访问是有可能造成物理IO的，当然在修改时也需要生成日志来确保一致性，同时锁机制也是不可缺少的。  
跟表变量另外一个显著区别就是，临时表可以创建索引也可以定义统计数据,因此SQL Server在处理访问临时表的语句时需要考虑执行计划优化的问题。简单地总结我们对于较小的临时计算用数据集推荐使用表变量。如果数据集比较大如果在代码中用于临时计算同时这种临时使用永远都是简单的全数据集扫描而不需要考虑什么优化，比如说没有分组或分组很少的聚合(比如说COUNT、SUM、AVERAGE、MAX等)也可以考虑使用表变量。使用表变量另外一个考虑因素是应用环境的内存压力，如果代码的运行实例很多就要特别注意内存变量对内存的消耗。一般对于大的数据集推荐使用临时表同时创建索引或者通过SQL Server的统计数据(Statisitcs)自动创建和维护功能来提供访问SQL语句的优化。如果需要在多个用户会话间交换数据当然临时表就是唯一的选择了。需要提及的是由于临时表存放在tempdb中因此要注意tempdb的调优。  
  
临时表存储在TempDB数据库中,所有的使用此SQL Server 实例的用户都共享这个TempDB,因为我们应该确保用来存储TempDB数据库的硬盘有足够的空间,以使之能够自己的增长.最好能够存储在一个拥有独立硬盘控制器上.因为这样不存在和其它的硬盘I/O进行争用.  
临时表主要会增加tempdb的IO，导致性能下降。  
  
**tempdb**  
tempdb全局存储内部对象，用户对象，临时表，临时对象，以及SQL Server操作创建的存储过程。每个数据库实例只有一个tempdb，所以可能存在性能以及磁盘空间瓶颈。各种形式的可用空间及过度的DDL/DML操作都会导致tempdb负载过重。这会导致运行在服务器上不相干程序运行缓慢或者运行失败。  
  
　　tempdb的一些常见通病如下：  
  
　　--耗完了tempdb的所有存储空间  
　　--读取tempdb时的I/O瓶颈造成的查询运行缓慢。  
　　--过度的DDL操作造成在系统表上的瓶颈。  
  
总的tempdb空间=用户对象+内部对象+存储的版本信息+空闲空间。  
这个空闲空间大小跟tempdb性能计数器上空闲空间是一样的。  
  
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\  
**局部临时表:**  
如果本地临时表由存储过程创建或由多个用户同时执行的应用程序创建，则 SQL Server 必须能够区分由不同用户创建的表。为此，**SQL Server 在内部为每个本地临时表的表名追加一个数字后缀。存储在 tempdb 数据库的 sysobjects 表中的临时表，**  
其全名由 CREATE TABLE 语句中指定的表名和系统生成的数字后缀组成。为了允许追加后缀，为本地临时表指定的表名 table\_name 不能超过 116 个字符。  
  
局部临时表不能够被其它连接所共享的原因就是因为局部临时表的表名后面加上了一个唯一字符来标识.如:  
  
create table #DimCustomer\_test   
(  
    ID   int IDENTITY (1,1)     not null,   
    WokNo                varchar(50),    
    primary key (ID)   
);  
  
现在我们来查看一下**TempDB中 sysobjects表**,我们会发现我们新创建的临时表#DimCustomer\_test已经被加上了后缀:  
USE TempDB  
GO  
SELECT name FROM sysobjects WHERE name LIKE ’%DimCustomer%’  
the Result is:  
#DimCustomer\_test\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_000000000005  
  
  
**全局临时表**  
create table ##DimCustomer\_test   
(  
    ID   int IDENTITY (1,1)     not null,   
    WokNo                varchar(50),    
    primary key (ID)      
);  
现在我们来查看一下TempDB中 sysobjects表,我们会发现我们新创建的临时表##DimCustomer\_test没有被加上了后缀:  
USE TempDB  
GO  
SELECT name FROM sysobjects WHERE name LIKE ’%DimCustomer%’  
The Result are:  
##DimCustomer\_test  
  
可以看到我们刚才创建的全局临时表名字并没有被加上标识.  
  
  
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\  
**相关例子：**

**Sql代码  [收藏代码](javascript:void())**

1. **drop** **table** #Tmp   --删除临时表#Tmp
2. **create** **table** #Tmp --创建临时表#Tmp
3. (
4. ID   **int** IDENTITY (1,1)     not null, --创建列ID,并且每次新增一条记录就会加1
5. WokNo                **varchar**(50),
6. **primary** **key** (ID)      --定义ID为临时表#Tmp的主键
7. );
8. **Select** \* **from** #Tmp    --查询临时表的数据
9. **truncate** **table** #Tmp --清空临时表的所有数据和约束

12. **Declare** @Wokno **Varchar**(500) --用来记录职工号
13. **Declare** @Str NVarchar(4000) --用来存放查询语句
14. **Declare** @Count **int** --求出总记录数
15. **Declare** @i **int**
16. **Set** @i = 0
17. **Select** @Count = Count(**Distinct**(Wokno)) **from** #Tmp
18. While @i < @Count
19. **Begin**
20. **Set** @Str = 'Select top 1 @Wokno = WokNo from #Tmp Where id not in (Select top ' + Str(@i) + 'id from #Tmp)'
21. **Exec** Sp\_ExecuteSql @Str,N'@WokNo Varchar(500) OutPut',@WokNo **Output**
22. **Select** @WokNo,@i --一行一行把职工号显示出来
23. **Set** @i = @i + 1
24. **End**

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\  
引用临时表的PRO的编译：  
  
例子

**Sql代码  [收藏代码](javascript:void())**

1. --SP1
2. **create** **procedure** Proc1
3. **as**
4. **begin**
5. **create** **table** #t1(a **int**, b **int**);
6. **insert** **into** #t1 **values**(1,2);
7. **select** \* **from** #t1;
8. **end**
10. **exec** Proc1
11. go
12. **exec** Proc1
13. GO

**Sql代码  [收藏代码](javascript:void())**

1. --SP2
2. **create** **procedure** Proc1
3. **as**
4. **begin**
5. **create** **table** #t1(a **int**, b **int**);
6. **insert** **into** #t1 **values**(1,2);
7. **exec** Proc2;
8. **end**
9. go
11. **create** **procedure** Proc2
12. **as**
13. **begin**
14. **select** \* **from** #t1;
15. **end**
16. go

19. **exec** Proc1
20. go
21. **exec** Proc1
22. go

我们可以看到，上面的2个例子，SP1跟SP2，  
**在SP1中第二次执行exec Proc1的时候，不会引起重编译，**  
但是**在SP2中第二次执行exec Proc1的时候，却会引起重编译**，为什么呢？  
  
  
参考答案：  
  
1.第一个case,我很久以前经常用表变量去inner join其它的表，当发现表变量数据稍多，整个语句就会变的很差，之后就一直按“数据很少就用表变量，数据多点就用临时表。”刚看到这个贴子，就想了想查了查，主要原因还是表变量的行数未知，SQL优化器必须估一个值，这个值往往是很小，在这种情况下去inner join其它表，基本上都会是Nested loop join，此时一旦表变量数据量大，性能将会变得极差。而这种情况下我们往往需要merge join 或 hash join才更有效率，recompile选项能让优化器去更新表变量的统计信息，而这样就能根据统计信息生成合理的执行计划。  
  
2.至于第二个关于存储过程的case，两个场景主要关键点还是**第二个proc2中引用非自己创建的临时表，这个引用是依靠临时表ID，当执行proc1生成临时表时，名字(执行计划依靠名字)不变但id会变，而proc2依靠这个id，这相当于proc2要查的临时表架构发生变化，所以每次都要重编译，如果在proc1里同时有两个exec proc2，那么第二个不会重编译，因为此时它可以重新第一个exec proc2的执行计划了**，因为两个exec proc2引用的临时表是同名同ID。  
  
  
相关文档参考：  
http://www.mssqltips.com/sqlservertip/2140/improving-sql-server-performance-when-using-table-variables/  
http://blogs.msdn.com/b/psssql/archive/2010/08/24/query-performance-and-table-variables.aspx  
http://blogs.msdn.com/b/sqlprogrammability/archive/2007/01/18/11-0-temporary-tables-table-variables-and-recompiles.aspx