文章编号：1005-8451（2010）04-0037-04

Sybase临时表技术的多层级应用研究

王奇成
（广州铁路集团 信息技术处，广州 510088）

摘 要：Sybase临时表用于存储过程或批命令中多个SQL语句间传递数据，可分解事务处理流程，减轻并发压力，改善查询性能。应用临时表，首先提倡多用，使编程思路清晰，优化处理性能；其次要巧用，大幅度提高存储过程执行速度；第3要注意临时表对存储过程预编译执行计划的影响，采用强迫执行计划的策略避免出现应用性能陡降的问题。对这3个层次的应用技术，本文结合实践，给出一些具体样例。

关键词：临时表；存储过程；SQL；性能优化；预编译；执行计划
中图分类号：TP39 文献标识码：A

Study on multi levels application of Sybase temporary table technology

WANG Qi-cheng
(Information Technology Institute, Guangzhou Railway Corporation, Guangzhou 510088, China)

Abstract: Sybase temporary table used between multiple SQL statements in the stored procedure or SQL batch to transfer data, could be decomposed processes, reduced concurrent pressure to improve query performance. First of all, it was promoted more use of temporary tables, cleared thinking programming , and optimized processing performance. Second, it must good use temporary tables, so significantly improve that stored procedures and the execution speed. Third, it should pay attention to the impact of pre-compiled execution plan of stored procedure with temporary table, use specific measures to avoid steep drop application performance problems. According to the three levels applying technology, the paper given some actual example on the practice.

Key words: temporary table; procedure; SQL; performance optimizing; pre-compile; implementing plan

临时表是大型数据库中普遍应用的一种技术，它用于多条SQL语句顺序执行时，语句间需传递数据的场合，功能虽相当于内存里面的缓存，但它由数据库引擎来管理，支持并发，当不同用户同时调用时，在临时表中看到的是不同拷贝的数据。临时表技术给SQL程序的开发带来便利，并可结合其他编程技术大幅度提高SQL性能，但使用时可能碰到一个特定问题，处理不当会导致性能低下。所以，应用临时表是个积累技巧、逐渐提高技能的过程，本文对这一过程进行探讨和展示。

1 临时表技术概述

1.1 Sybase数据库中临时表

Sybase数据库中临时表用#符号作为表名前缀，内部处理时自动添加一串唯一性后缀，以区分不同会话的调用。数据库会话一结束或存储过程执行完，临时表自动删除。

收稿日期：2010-04-02
作者简介：王奇成，高级工程师。

1.2 应用临时表的典型场景

临时表用在多条SQL语句组成的批命令和存储过程里面。存储过程也可以理解成经常调用的批命令，只是定义了名称，需要时按名称来调用，并预先编译生成执行计划存于数据库中，一旦被调用直接取出运行，不需临时编译，从而加快了响应速度。虽然构思得当，绝大部分的事务处理流程都可用单条语句实现，但那样不易编写，难以理解，修改维护较困难。实质上，对于单条SQL语句，数据库内部处理时也是要拆解成多个步骤进行的，各步之间传递数据同样要用到临时表。

1.3 应用临时表有个渐进过程

在批命令和存储过程中应用临时表，初始考虑主要是为了在多个SQL语句间传递数据，帮助拆解事务处理流程，优化SQL执行性能。

但是仅靠几个临时表，可能无法显著提高性能，SQL语句本身的执行速度，还得靠其他技术协同处理。用临时表结合其他技术，方能发挥威力。

应用临时表时，存储过程的预编译会带来一个意想不到的问题，如果忽视该问题没采取防范

措施，可能在某时刻应用性能陡降。所以存在一个慎用临时表的问题。

可见，在Sybase数据库中应用临时表技术，有一个逐渐加深理解、层层提高技能的过程。

2 基本层级

基本层级多用临时表处理复杂事务，减少查询负荷。打开一个较复杂系统的众多存储过程，可以看到临时表应用比较多。譬如：

```
create table #tb_station /* 查询的车站范围 */
( station_name char(10) )
insert into #tb_station select station_name from basic..on_net_station where bureau_code = 'Q' /* 广铁所属的联网站 */
```

这是铁路客票系统相关应用中的一段代码，把广州铁路集团所属的售票联网站取出放到一个临时表中，再把其他数据表跟这个临时表进行关联，这样减少了关联操作访问的数据量，而且减少了对基表on_net_station的争用，提高了并发度。

另外，在应用程序开发中用到的SQL语句，可以包含临时表。譬如用PowerBuilder环境创建Data Window时，针对复杂业务，可在数据窗设计中，切换到手工编写语法，使用包含临时表的SQL批命令。

应用临时表的初始常见用法，是用来处理复杂事务，把业务流程拆解成多个步骤进行，使得编程思路清晰，减少查询负荷，一定程度上提高性能。

3 第二层级

第二层级巧用临时表大幅度提高性能。存储过程中经常会用到游标，如果用等价的SQL语句来替换游标，可显著提高性能。Sybase中CASE语句可用来编写等价替换游标的高效率SQL语句，而要充分发挥CASE语句的威力，需要的最佳伴侣就是临时表。看下面的代码片段：

```
declare c_train_no cursor for select train_no, sum(a_yz) ……sum(b_yz) …… from #tb_total
open c_train_no
fetch c_train_no into @train_no, @a_yz …… @b_yz……
while @@sqlstatus != 2
begin
  if @a_yz = 0 select @rate_yz = 0 /* 硬座上座率 */
  else select @rate_yz = convert(decimal(8, 1),Round(convert(decimal(18,4),@b_yz) / convert(decimal(18,4),@a_yz) * 100, 1))
  ……
  /* 算出排序车次 */
  select @train_code_t = substring (@train_code,1,1)
  if @train_code_t in ('T','K')
  begin
    select @train_code_s = substring(@train_code,2,3)
    if @train_code_t = 'T' select @train_code_order = ' ' + right('000'+ltrim(rtrim(@train_code_s)),3)
    else select @train_code_order = '0' + right ('000'+ltrim(rtrim(@train_code_s)),3)
  end
  else select @train_code_order = @train_code
  /* 算出始发时间 */
  select @start_time = start_time from stop_time where train_no = @train_no and station_no = '01'
  select @start_time = substring(@start_time, 1,2) + ":" + substring(@start_time,3,2)

  insert into #tb_total_t values ( @start_time, @train_code_order, @train_no, @a_yz…@b_yz… @rate_yz… )
  fetch c_train_no into @train_no, @a_yz…… @b_yz……
end
```

以上代码在一个游标中算出上座率、排序车次和始发时间，可以用CASE语句结合临时表来替换，不仅速度快了许多，而且代码简洁易读，如下：

```
/* 首先构造一个临时表，始发时间预置成 xx:xx，排序车次预置成 xxxx，上座率预置成 0 */
insert into #tb_total_t
select 'xx:xx', 'xxxx', train_no,
sum(a_yz) ……sum(b_ yz) …… 0, 0, 0, 0
```

```
from #tb_total group by train_no
/* 更改排序车次和上座率 */
update #tb_total_t
set rate_yz  = case when a_yz = 0 then 0 else
convert( decimal(8,1), Round(b_ yz * 100.00 /
a_ yz, 1) ) end ……
      train_code_order = case substring
(train_code,1,1) when 'T' then ' ' when 'K' then
'0' else substring(train_code,1,1) end
       + right('000'+ltrim(rtrim(substring
(train_code,2,3))),3)
/* 更改始发时间 */
update #tb_total_t set a.start_time = substring
(b.start_time,1,2) + ":" + substring(b.start_time,
3,2)
from #tb_total_t a, stop_time b
where a.train_no = b.train_no and b.station_no
= '01'  /* 站号为01的站是始发站 */
```

这段代码用CASE语句结合临时表，替换判断和分支等流控语句，取消游标，把串行运算转换成可利用数据库并行运算机制的SQL语句，从而可大幅度提高性能。

4 第三层级

第三层级避开特定问题，在多用巧用的基础上慎用。

4.1 使用临时表可能带来的问题

临时表和CASE语句的紧密结合能切实提高性能，在业务复杂的场合，也确实不可避免地要用到临时表。但在使用临时表时，即使SQL语句做过优化，也还得注意可能会导致应用程序性能莫名其妙的低下，或者平时用得好好的，某一天突然性能陡降。

这种故障的缘由是，临时表在存储过程被保存并编译的时候，并不能确定表里存放多少数据，编译器就统统假设成存放了100行数据，并据此生成查询计划。而在实际应用场合，临时表里不是存放整整100行记录，如果存放几行、几十行、几百行，跟100行没有什么区别，但如果存放几千、几万行，区别就很大。在默认生成并保存的查询计划里，查询引擎采取的最佳策略，可能是对临时表做n遍全表扫描，而不是依据索引去取数据，这样，当临时表里存放几万行记录的时候，n遍全表扫描的性能比依据索引去取数就会明显低很多。或者随着业务的变化，临时表里面数据有所变化，引起查询引擎重新选择执行计划。

问题缘由还难以被发现。当调试存储过程，一条条拆解成SQL语句，逐个单独执行，是没有问题的。这是因为在SQL会话里创建的临时表，查询引擎就已经知道它里面存放多少条记录，从而能生成一个符合实际情况的执行策略。

4.2 解决方案

针对这个问题，可以使用Sybase一些特定的T-SQL语句，给予查询引擎优化提示，保证编译时生成并保存的查询计划，执行时也依照进行。这样，不管临时表数据变化跨度如何，查询计划都是稳定的，性能不会陡降。

譬如以下代码：

```
insert into #tb_temp_3 -- 临时表
select d.station_telecode,
 g.train_date,
 d.train_no,
 s.station_train_code,
 c.train_class_name,
 t.start_station_name
from  basic..stop_time s noholdlock,
      basic..train_dir t noholdlock,
      tic_train_direction d,
      basic..train_class c,
      #tb_station_date g -- 临时表
where d.train_no = s.train_no and ……
```

在实际应用中连续使用了3个月，而且经历了春运业务高峰的考验，没有问题，在某一天突然报告影响其他业务。如果用以下代码代换，就可以保证性能稳定：

```
set forceplan on  -- 强迫执行计划，按照连接
表出现的顺序进行
insert into #tb_temp_3  -- 临时表
select d.station_telecode,
 g.train_date,
 d.train_no,
 s.station_train_code,
 c.train_class_name,
```

```
    t.start_station_name
  from  #tb_station_date g,
    tic_train_direction d (index pk_tic_train_
direciton),
    basic..stop_time s (index idx_1) noholdlock,
    basic..train_dir t (index idx_1) noholdlock,
    basic..train_class c
  where d.train_no = s.train_no and ……
  set forceplan off  -- 关闭强迫执行计划
```

以上代码相对原代码最大的不同有两点：(1)由set forceplan on语句打开了强迫性执行计划。(2)多表关联查询中表的出现次序不一样，是按最优查询计划中表出现的顺序来排列。最优查询计划，可在业务经验、数据估算的基础上，借助isql等会话工具得到。

4.3 慎用临时表

所以，在存储过程或批命令中一定要慎用临时表，保证到预编译的查询计划最优，避免执行性能陡降。

5 具体实践

5.1 多用临时表处理复杂业务

客票系统处理业务复杂多变，存储数据量大，并发度高，本人围绕客票系统开发的系统，像广铁集团客票营销分析系统、外来工团体订票系统等等，也都具备类似特点，跟客票系统一样，用到了大量临时表。

当数据库系统中临时表使用很多，而又存在大量并发访问时，Sybase临时库会出现较多的争用现象。使用多个临时库，可以分担访问负荷，降低争用数据时相互等待的几率，提高并发度。

5.2 巧用临时表消除游标，显著改善存储过程的性能

本人在广铁客票营销分析等系统中大量使用临时表来消除游标，显著改善了存储过程的执行性能，提升十倍、百倍以上，而且涉及数据量越大，提升越多。

5.3 经历应用系统性能突降事件，开始慎用临时表

2010年春运，为实现及时准确发布余票信息的业务要求，在客票系统基础上开发了广州铁路集团余票信息发布系统，经过优化，达到每分钟可刷新一遍广州铁路集团全局余票数据。系统连续稳定运行3个月，经历了2010年春运大考。2010年3月，怀化地区突然上报联合站服务器上窗口售票被余票发布系统拖累。而这时，其他地区另外8台同样的站群服务器却运行正常。

经过反复研究分析，终于发现临时表对存储过程预编译计划的影响规律。后利用Sybase T-SQL里面的set forceplan on语句，改造原有SQL语句，给予数据库查询引擎以执行路径的提示，让存储过程预编译时就保存跟执行时一模一样的最优查询计划，保证了SQL语句执行性能的稳定，从而扫除了故障。

6 结束语

慎用临时表的提醒，主要是为了避免在存储过程实际执行时使用低效执行计划的问题。虽然在15版以上Sybase数据库中，这个问题在系统层面有所解决，但还是要注意临时表可能带来的负面影响。

全面掌握临时表的以上特性，可提高与性能相关的数据库开发能力。这种能力跟临时表的3个应用层级类似，大致分：

(1)初级：停留在数据的增、删、改、查阶段，完成业务即止，不考虑性能。

(2)中级：面对大量数据、高度并发的数据库环境，开始为SQL查询性能而努力，通常用调整参数、创建索引、清除数据等常见办法，但是遇到极高并发、大量数据、处理复杂业务、关联众多数据表查询的场合，效果可能不显著。

(3)高级：研究数据库引擎的内部工作机制，分析SQL每一种写法的优劣，用知根知底的语句去与数据库查询引擎相配合，从而大幅度提高SQL语句的执行效率。

参考资料：

[1] 王奇成. Sybase数据库的存储过程性能优化[J]. 铁路计算机应用，2005，14(3).

[2] Adam Machanic, Hugo Kornelis, Lara Rubbelke. SQL Server 2005编程艺术[M]. 颜 炯，薄建禄. 北京：人民邮电出版社，2008，7.