SYBASE
技术服务园地
连载（60）
SYBASE TECHNOLOGY SERVICE FIELD

解决方案

开发人员升级至ASE 15.0的10大理由（一）

升级软件包并非总是愉快的任务。无论是用户接口(UI)变化导致的生产力下降（例如不得不寻找以前熟知位置的功能），还是API的改变引起在重新编译代码时的大量调试和排错。升级Sybase ASE也一样。从开发人员的角度看，“不破不立”(老的还没坏，就不要修复)的成语往往导致对较早而熟悉版本的依附，即便是下一版本已经发布了很长时间，甚至是下一个版本的后续版本。

但是，我们也知道如果新版本提供了一些引人注目的升级理由，我们就更愿意去升级。尤其是针对应用程序今天可能会处理的更大数据量而带来的实质性性能提升，当然包括了即时响应时间。需求很高的新功能和特性，能极大减轻应用程序开发的挑战。通过更便捷的开发工具提高开发人员生产力，有助于简单任务的工具，例如输入/修改测试数据，还包括有助于鉴别问题根源的工具。

ASE 15针对开发人员在功能性、性能和生产力促进方面的重大改进包括以下特性。

（1）bigint、unsigned int和Identity改进。

（2）新哈希、统计和XML函数。

（3）基于函数的索引。

（4）计算列。

（5）SQL UDF与替代触发器（Instead of Triggers）。

（6）可配置的DSS查询优化。

（7）积极聚合/重定向连接和历史归档。

（8）新客户端工具（dbisql、计划查看器）。

（9）极端事务处理（语句缓存、延迟提交、tempdb）。

（10）语义分区。

虽然被组织成了“前10”列表，但其并非意味着列表的第5项比第8项更重要，第1项是其中最重要的。每项的重要性将取决于它与当前项目的相关度，顺序也可能随之改变。

可能在讨论中需要注意的一件事情是，一个功能当与另一功能结合时会扩大了该功能的力量，两个联合使用也许能达到超出预期的功能性。因此，常常能在本文中发现一些例程利用了不止一个新特性。

1 bigint、unsigned int和Identity改进

对应用程序开发人员来说，常见的缺憾是当使用ASE时缺少64位整型或无符号整型数据类型，而被强迫需要使用numeric数据类型。虽然这并非一个很大的问题，但它却一直是件麻烦事儿，因为numeric数据类型的结构类似于很久以前的二进制码的十进制表示法(Binary Coded Decimal Representation)。因此，连接较慢，且对于简单的算数或值比较运算中使用困难。因为OpenClient长期以来已经支持了bigint，这就变得更加遗憾了。从ASE 15.0开始，ASE就支持了64位整数“bigint”数据类型。除了bigint外，还新增了长期以来都高度需求的无符号整数类型。请注意，没有unsigned tinyint。Tinyint总是“无符号”，有效值范围是0~255。

表1 ASE15新的数据类型和范围

数据类型	范围
bigint	-9,223,372,036,854,775,808 ~ +9,223,372,036,854,775,807
unsigned bigint	0 ~ 18,446,744,073,709,551,615
unsigned int	0 ~ 4,294,967,295
unsigned smallint	0 ~ 65,535

以下是几个在应用程序开发时，新类型可能有用的常见场景。

（1）大identity值。随着在ASE 12.5中包括了integer identity之后，无符号类型和bigint数据类型将integer identity的范围扩展至20位数字，替代了对较难操作的numeric数据类型的需要。

（2）避免聚合算术溢出。对当今的大表多次执行sum()或count()时，聚合结果值会超出integer数据类型的范围。新支持的bigint聚合函数支持在该类聚合时无需先将数据类型转换成numeric。

（3）较大的位掩码。许多应用程序使用位掩码来存储状态或二进制属性信息，包括ASE本身的系统表。这将更适合binary()数据类型，因为无论平台的编码顺序特性，结果都将相同。

有可能有其它用例，或者一些相关的，例如应用程序创建的非identity的id值。

1.1 大Identity值：bigint与numeric()比较

拿identity的使用打个比方，我们都了解identity（也包括大部分id值）都是从0~#(某个较大正数)的范围。负数id值较为罕见，但一些应用程序将负id值用作失效的或培训用途。以往，使用符号整数的应用程序只能将id的值域限制在20亿个值的范围内，再加上id消失问题（例如使用identity时）和更大的数据集，就显得太小了。最普遍的解决方案是使用numeric数据类型。作为一种结构（与真正的数字比较），它在应用程序的算术运算如何执行方面有着自己特殊的性质。请考虑以下常见的应用程序场景：

（1）保留ID号。一个应用程序使用连续的键来记录交易编号、客户编号等。对于将添加多个实例的应用程序逻辑来说，应用程序需要“保留”一部分ID号。应用程序只需检索保留范围中的第一个数字并加1直至达到了保留范围的最后一个数字。

（2）带ID号的主/从处理。一个应用程序使用ID号来唯一标识数据元素，用法类似Sybase在sysobjects中使用的ID和其与syscolumns的主从关系。与其对每个主表数据行都采用一个游标迭代检索从表这种有害性能的方式，应用程序还不如将主表和对应从表的记录(根据主ID排序)的范围检索到数组中。当其处理每个主表行时，它将自上一次处理过的位置检索从表的数组，直至从表记录中的主ID值不再与当前的相符。

除此以外，使用NUMERIC或DECIMAL数据类型也不那么简单。因为它们是结构而非真正的数字，使用CT-Lib的C程序员在使用NUMERIC和DECIMAL数据类型时需要常规调用，例如cs_calc()来加1和使用cs_cmp()来比较值。需要对NUMERIC加1的Java程序员通常先需要将其转换为java.math.BigDecimal或java.math.BigInteger后再调用add()或equals()方法，而且对于DECIMAL来说需要从8个可用的round方法中指定一个。大量额外的工作仅仅是为了实现超过20亿的值。这些全都可通过使用bigint或unsigned bigint来轻松简化：

```
create table mytable (
customer_id    unsigned bigint identity not null,
cust _last _name      varchar(30)   not null,
cust _first _name     varchar(20)   not null,
)
```

1.2Reserve_identity()：小批量和伪序列对象

为了进一步帮助使用identity，ASE 15.0.2新增了reserve_identity函数和服务器配置参数。

表2　reserve_identity函数和服务器配置参数

改进	描述
reserve_identity()	调用会话将指定表的1个或多个identity值保留在池中。返回值是varchar与next_identity()类似。
identity reservation size	服务器配置参数，设置每个会话能通过reserve_identity()方法(缺省为1)对每个表保留的最大identity数值。

（1）小批量插入。这对于需要对后续子表插入identity值的应用程序相当有用。例如，考虑一个订单输入系统，订单id是identity值，订单项(order items)的键是订单id和订单项编号。在该类系统中，考虑每个门店都在晚上提交订单进行批量订单处理的情况。在ASE 15前，需要通过以下网络高度密集的应用程序逻辑：

```
For each order
SRV    Insert the order header
  Client    Retrieve the order id (@@identity value)
SRV    Insert the order items using the order id Next order
```

正如显示的，该逻辑对每个订单需要与服务器进行2次网络交互。假设检索@@identity值仅对整批在第一行插入时执行一次(本身也应如此)。如果处理100个订单，结果就是200个SQL批才能处理完100个订单。如果每个网络操作需要50ms，每个插入需要10ms，则至少需要10 s来处理—仅仅因为网络交互的原因。

通过保留identity函数，处理逻辑则变为：

```
Client    Reserve a micro-batch size of identities
SRV
{
Set identity insert on
Batch insert the order headers using reserved identities
Batch insert the order items using reserved identity values for order id
}
```

假设是同样的100个订单，整批仅需2次服务器交互。通过reserve_identity()函数，应用程序现在即能使用小批量并利用SQL批或甚至是块API来完成批处理情况，比原来的交互方法快数倍。

（2）伪序列对象。开发人员使用identity列的一个问题就是它的值是与一张表紧密相关的。如果使用面向对象到关系型的映射模式来将子类以单独的表实现，如果超类未通过物理表展现的话就会出现问题。例如，考虑一个简化的交易系统，其支持不

同类型的交易：资本、商品、衍生品、珍宝，等等。因为每种类型的数据元素不同，这4种交易类型可通过数据库中4张单独的表来实现。为了简单易用，父对象（Trade）通过包含连接的视图实现，它包含了4张表并返回一些通用的数据，例如trade_id、trade_date等。问题是trade_id值的分配必须是跨所有4张表唯一的。传统情况下，应用程序可有以下选择：

a.实现父“trade”表，它仅包含trade_id而几乎没有别的内容，因为4张不同的交易表之间分别实现了。它为每个交易造成了额外的插入损失，根据表的大小和需要的索引可能造成几次IO。

b.在每个表上使用单独的trade_id序列。结果就是每个表都和其它表有重合的trade_id。查询必须指定交易的类型。

其它DBMS通过实现驻留于表结构外的序列对象来解决该问题。无论何时对需要的表执行插入时，应用程序通常会执行与下面类似的语句：

```
assume the sequence object is called trade_seq
insert into equities_trades (trade_id, trade_date, ...)
    values (trade _seq.nextval(),getdate(),...) insert into commodity _trades (trade_id, trade_date, ...)
    values (trade _seq.nextval () ,getdate () ,...)
insert into equities_trades (trade_id, trade_date, ...)
values (trade _seq.nextval () ,getdate () ,...)
```

在上例中，从空数据库开始，trade_id应分别是1、2、3，资产交易(equities_trade)的trade_id为1和3，商品交易(commodity_trade)的trade_id=2。不幸的是，当前版本的ASE不支持序列对象。但是，可通过dummy表创建的伪序列对象和reserve_identity()函数实现类似的功能。Dummy表的创建类似如下语句：

```
create table trade_seq (trade_id unsigned bigint identity    not null)
with identity _gap=100
```

对表的插入与上面的语句很类似，只需一个小小的表达式改变：

```
assumes the trade_seq table has an identity column
insert into equities_trades (trade_id, trade_date, ...)
    values (convert(unsigned bigint, reserve_identity( 'trade _seq' ,1)) ,getdate() ,...) insert into commodity_trades (trade_id, trade_date, ...)
    values (convert(unsigned bigint, reserve_identity( 'trade _seq' ,1)) ,getdate() ,...) insert into equities_trades (trade_id, trade_date, ...)
values (convert(unsigned bigint,reserve_identity( 'trade _seq' ,1)) ,getdate() ,...)
```

因为reserve_identity返回varchar，需要调用函数将结果转换为列的合法值。注意并非真正需要对trade_seq表进行插入。可通过使用SQL UDF(后续将描述)来简化，例如：

```
create function next_seq (@numkeys unsigned bigint) returns unsigned bigint as
begin
return convert (unsigned bigint, reserve _identity('trade _seq' , @numkeys))
end
```

则以上SQL可简化为：

```
assumes the trade_seq table has an identity column
insert into equities_trades (trade_id, trade_date, ...)
values (dbo.next_seq(1)),getdate(),...)
insert into commodity _trades (trade_id, trade_date, ...)
values (dbo.next_seq(1)),getdate(),...)
insert into equities_trades (trade_id, trade_date, ...)
values (dbo.next_seq(1)),getdate(),...)
```

UDF的核心内容显示更新了将identity结构写入表头的方法(正如在ASE 15.0.3中测试的那样)。在任何情况下，根据因子的个数，以上结果都与序列对象相同。

a.连续枚举。如果并发插入在一个有identity列的表上发生，且服务器有多个引擎，则插入可能是非连续的。这是因为“identity grab size”配置参数支持SMP引擎占用“一块”identity值，以便减少对identity结构的竞争。实现了序列对象的DBMS也有类似的情况，SMP支持服务器用类似的方式“缓存”一块序列号码。

b.Identity间隔。在未预期的服务器瘫痪的情况下，在重新启动后，ASE会假定一定百分比的identity被“烧毁”，并从后续块的值开始。可通过配置“identity burning set factor”的值来进行控制，或许好的方法是使用“identity_gap”表属性。再次重申，根据DBMS，由于服务器的性能原因而事先缓存了序列块，相同的现象可能会发生。

由于这些行为的相似性，用reserve_identity()来实现伪序列对象的方法可能会对一些应用程序有用。

文/赛贝斯软件（中国）有限公司

（未完待续）