SYBASE®
An SAP® Company

性能和调优系列：查询处理和抽象计划

Adaptive Server® Enterprise

15.7

文档 ID：DC01080-01-1570-01

最后修订日期：2011 年 9 月

Sybase, Inc., One Sybase Drive, Dublin, CA 94568。

目录

第 1 章

了解查询处理

本章概述了 Adaptive Server® Enterprise 中的查询处理器。

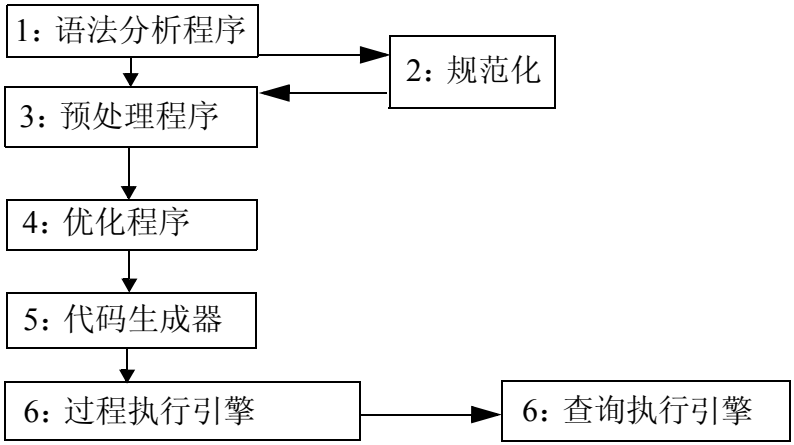
查询处理器旨在处理指定的查询。处理器可以生成使用极少资源即可执行的高效查询计划，并能够确保结果一致而正确。

为了有效地处理查询，查询处理器使用：

- 指定的查询
- 有关查询中指定的表、索引和列的统计信息
- 可配置变量

查询处理器必须使用若干模块执行若干步骤才能成功处理查询：

图 1-1：查询处理器模块



1 语法分析程序将 SQL 语句文本转换为称为 “查询树”的内部表示。

2 此查询树将进行规范化。这一过程涉及确定列和表名称，将查询树转换为共轭范式 (CNF)，并解析数据类型。此时，您可以确定该语句是否可从使用语句高速缓存中受益。

3 预处理器将某些类型的 SQL 语句 （例如包含子查询和视图的 SQL 语句）的查询树转换为更加高效的查询树。

4 优化程序分析执行 SQL 语句时可能存在的操作组合 （连接顺序、访问和连接方法、并行度），并基于开销估计从众多备选方案中选择一种有效组合。

5 代码生成器将优化程序生成的查询计划转换为更适合查询执行引擎的格式。

6 过程引擎直接执行诸如 create table、execute procedure 和 declare cursor 这样的命令语句。对于数据操作语言 (DML) 语句，例如 select、insert、 delete 和 update，该引擎会为所有查询计划设置执行环境，并调用查询执行引擎。

7 查询执行引擎执行由代码生成器提供的查询计划所指定的排序步骤。

查询优化程序

查询优化程序可提高联机事务处理 (OLTP) 和操作方面的决策支持系统 (DSS) 环境的速度和效率。您可以选择最适合所在查询环境的优化策略。

查询优化程序可进行自调优，它需要的干预比 Adaptive Server Enterprise 15.0 之前的版本少。查询优化程序很少依靠工作表在操作步骤间获得实现；但如果它确定散列和合并操作更有效，则可以使用更多工作表。

15.0 版的查询优化程序中的一些主要功能包括以下支持：

- 增强查询性能的新优化技术和查询执行运算符支持，例如：
  - 使用内存中排序和散列的即时分组和排序运算符支持，用于带 group by 和 order by 子句的查询
  - 对高效 join 操作的 hash 和 merge join 运算符支持
  - index union 和 index intersection 策略，用于对不同索引进行带谓词的查询

  第 4 页的表 1-1 是 Adaptive Server Enterprise 提供的优化技术和运算符支持的列表。其中许多技术直接映射为查询执行支持的运算符。请参见第 18 页的 “Lava 查询执行引擎”。
- 改进了索引选择，尤其是对使用 or 子句的连接，以及使用 and 搜索参数 (SARG) 的数据类型不匹配但兼容）的连接
- 改进了开销计算，利用**连接直方图**来防止因连接列中存在数据倾斜而可能导致的不准确性
- 连接排序和计划策略中基于开销的新清理和超时机制，针对大型、多向连接，以及星型和雪花模式连接
- 支持数据和索引分区 （为并行的构件块）的新优化技术，对超大型数据集尤其有用
- 改进了针对垂直和水平并行度的查询优化技术。请参见第 5 章 “并行查询处理”
- 通过以下途径改进了问题的诊断和解决：
  - 可搜索的 XML 格式跟踪输出
  - 来自新 set 命令的详细诊断输出。请参见第 3 章 “显示查询优化策略和估计值”

表 1-1：优化运算符支持

运算符	说明
hash join	确定查询优化程序是否可使用 hash join 算法。 hash join 可能会消耗更多运行时资源，但如果连接列中没有有用的索引，或与连接表中行数的乘积相比较而言，有大量的行符合 join 条件，则它非常有用。
hash union distinct	确定查询优化程序是否可使用 hash union distinct 算法，如果大部分行都不同，则该算法的效率较低。
merge join	确定查询优化程序是否可使用 merge join 算法，该算法依赖已排序的输入。例如，如果对合并键上的输入进行排序 （例如从索引扫描中），则 merge join 最为有用。如果对输入进行排序要求使用排序运算符，则会降低 merge join 的有用性。
merge union all	确定查询优化程序是否可对 union all 使用 merge 算法。merge union all 保留联合输入的结果行的顺序。如果输入已经过排序且父运算符 （如 merge join）受益于该排序，则 merge union all 尤其有用。否则， merge union all 可能需要排序运算符，而这会降低效率。
merge union distinct	确定查询优化程序是否可对 union 使用 merge 算法。 merge union distinct 与 merge union all 类似，但它不保留重复行。merge union distinct 需要已排序的输入并提供已排序的输出。
nested-loop-join	nested-loop-join 算法是 join 方法最常用的类型，在不需要排序的简单 OLTP 查询中最为有用。
append union all	确定查询优化程序是否可对 union all 使用 append 算法。
distinct hashing	确定查询优化程序是否可使用散列算法删除重复项，如果与行数相比不同值很少，则该运算符非常有效。
distinct sorted	确定查询优化程序是否可使用单传算法来消除重复项。 distinct sorted 依赖已排序的输入流，如果其输入未经过排序，则可能会增加 sort 运算符数。
group-sorted	确定查询优化程序是否可使用即时分组算法。 group-sorted 依赖分组列上排序的输入流，并在其输出中保持该顺序。
distinct sorting	确定查询优化程序是否可使用排序算法来消除重复项。如果输入未经过排序（例如，如果没有索引），并且排序算法生成的输出顺序可能有用，则 distinct sorting 会很有用；例如在合并连接中。
group hashing	确定查询优化程序是否可使用 group hashing 算法来处理集合。

方法	说明
multi table store ind	确定查询优化程序是否可对多个表 join 的结果进行重新格式化。使用 multi table store ind 可能会增加工作表的使用率。
opportunistic distinct view	确定查询优化程序能否在强制执行差别操作时使用更为灵活的算法。
index intersection	确定查询优化程序能否将多个索引扫描的交集用作搜索空间中查询计划的一部分。

优化查询时分析的因素

查询计划包含检索策略和一组有序的执行步骤，可用来检索查询所需的数据。制定查询计划时，查询优化程序将检查：

- 查询中每个表的大小 （按行和数据页计算），以及要读取的 OAM 页和分配页的数目。
- 查询中使用的表和列上存在的索引、索引类型以及每个索引的高度、叶页数量和聚簇比。
- 查询的索引范围；也就是说，能否通过检索索引叶页中的数据来满足查询而不用访问数据页。即使查询中不包括 where 子句，Adaptive Server 也可以使用将覆盖查询的索引。
- 索引中键的密度和分配。
- 可用数据高速缓存的大小、高速缓存支持的 I/O 大小和将使用的高速缓存策略。
- 物理读取和逻辑读取操作的开销；也就是说，从磁盘进行物理 I/O 页读取以及从主内存进行逻辑 I/O 读取的开销。
- join 子句 （具有最佳连接顺序和连接类型），考虑每个连接所需的开销和扫描次数，以及索引在限制 I/O 方面的效果。
- 如果 join 中没有用于内部表的有用索引，那么使用连接列的索引建立工作表 （内部临时表）的速度是否快于重复的表扫描。
- 查询是否包含可使用索引查找值而无需扫描表的 max 或 min 集合。
- 是必须重复使用数据或索引页以满足查询 （例如， join），还是可以因为只需对这些页进行一次扫描而使用读取和放弃策略。

对于每个计划，查询优化程序通过计算逻辑和物理 I/O 以及 CPU 处理的开销来确定总开销。如果存在代理表，还会估计额外的网络相关开销。然后，查询优化程序会选择开销最低的计划。

Adaptive Server 15.0.2 及更高版本的查询处理器将对存储过程中的语句的优化延迟到执行这些语句时进行。这有益于查询处理器，因为局部变量的值可用于优化其各自的语句。

早期版本的 Adaptive Server 使用缺省猜口令对使用局部变量的谓词进行选择性估计。

查询优化转换

在分析和预处理某个查询后，但在查询优化程序开始对其进行计划分析之前，要对该查询进行转换，以增加可优化的子句数。优化程序所做的这种转换更改是透明的，除非检查像 showplan、dbcc(200)、statistics io 或 set 命令这样的查询调优工具的输出。如果运行的查询通过添加经优化的搜索参数得到了改善，则会看到添加的子句。在 showplan 输出中，对于没有为其指定搜索参数或连接的表，该子句显示为 “Keys are” 消息。

转换为等效参数的搜索参数

优化程序查找查询子句，以转换为可供搜索参数使用的形式。这些子句在表 1-2 中列出。

表 1-2：等效的搜索参数

子句	转换
between	转换为 >= 和 <= 子句。例如， between 10 and 20 被转换为 >= 10 and <= 20。
like	如果模式中的第一个字符是常量，则 like 子句可能被转换为大于或小于查询。例如， like "sm%" 变为 >= "sm" and < "sn"。如果第一个字符是一个通配符，则诸如 like "%x" 这样的子句无法使用索引进行访问，但可以使用直方图值来估计匹配行的数目。
in(values_list)	转换为 or 查询列表，也就是说，int_col in (1, 2, 3) 变为 int_col = 1 or int_col = 2 or int_col = 3。 in-list 中元素的最大数量为 1025 个

适当应用搜索参数传递闭包

优化程序可将传递闭包应用于搜索参数。例如，以下查询在 title_id 上连接 titles 和 titleauthor，并在 titles.title_id 上包括一个搜索参数：

```
select au_lname, title
from titles t, titleauthor ta, authors a
where t.title_id = ta.title_id
        and a.au_id = ta.au_id
        and t.title_id = "T81002"
```

此查询已优化，就好像它也在 titleauthor.title_id 上包括该搜索参数：

```
select au_lname, title
from titles t, titleauthor ta, authors a
where t.title_id = ta.title_id
        and a.au_id = ta.au_id
        and t.title_id = "T81002"
        and ta.title_id = "T81002"
```

通过这一附加子句，查询优化程序可以使用 titles.title_id 上的索引统计信息来估计 titleauthor 表中匹配行的数目。更加精确的开销估计改善了索引和连接顺序的选择。

equi-join 谓词传递闭包

在适用的情况下，应用优化程序可将传递闭包应用于常规 equi-join 的连接列。以下查询指定 t1.c11 和 t2.c21 的 equi-join，以及 t2.c21 和 t3.c31 的 equi-join：

```
select *
from t1, t2, t3
where t1.c11 = t2.c21
      and t2.c21 = t3.c31
      and t3.c31 = 1
```

如果没有连接传递闭包，则只考虑以下连接顺序：(t1, t2, t3)、(t2, t1, t3)、(t2, t3, t1) 和 (t3, t2, t1)。通过在 t1.c11 = t3.31 上添加 join，查询处理器可扩展 join 顺序列表，使其包括以下顺序：(t1, t3, t2) 和 (t3, t1, t2)。搜索参数传递闭包能够将 t3.c31 = 1 指定的条件应用于 t1 和 t2 的 join 列。

类似地， equi-join 传递闭包还可应用于使用 or 谓词的 equi-joins，如下所示：

```
select *
from R,S
where R.a = S.a
and (R.a = 5 OR S.b = 6)
```

查询优化程序推断这等同于：

```
select *
from R,S
where R.a = S.a
and (S.a = 5 or S.b = 6)
```

or 谓词可以在 S 扫描中求值，并且有可能用于 or 优化，从而有效地使用 S 的索引。

再如，连接传递闭包可应用于复杂 SARG，因此，下面这样的查询：

```
select *
from R,S
where R.a = S.a and (R.a + S.b = 6)
```

被转换并推断为：

```
select *
from R,S
where R.a = S.a
and (S.a + S.b = 6)
```

复杂谓词可在 S 扫描中求值，从而因提前对结果集进行了过滤而使性能显著提高。

传递闭包仅用于上述常规 equi-join。 join 传递闭包不能用于：

- 非 equi-join ； 例如， t1.c1 > t2.c2
- 外连接；例如 t1.c11 *= t2.c2、 left join 或 right join
- 跨越子查询边界的连接
- 用于检查参照完整性或对视图执行 with check option 的连接

注释 自 Adaptive Server Enterprise 15.0 起，用来打开或关闭 join 传递闭包和 sort merge join 的 sp_configure 选项已停用。在适用的情况下，始终能够在 Adaptive Server Enterprise 15.0 及更高版本中应用 join 传递闭包。

为提供额外优化路径所进行的谓词转换和分解

谓词转换和分解增加了查询处理器可用的选择数。通过从用 or 链接的谓词块中将子句提取到由 and 链接的子句中，可添加可优化的查询子句。这些经过优化的附加子句表明有更多的访问路径可用于查询执行。最初的 or 谓词仍然保留以确保查询的正确性。

在谓词转换过程中：

1 是每个 or 子句中的精确匹配的简单谓词 （连接、搜索参数和 in 列表）被提取。在下面步骤 3 的查询中，该子句在每个块中精确匹配，因此，它被提取出来：

```
t.pub_id = p.pub_id
```

between 子句在谓词转换前被转换为大于等于和小于等于子句。以上查询示例在两个查询块 （尽管结束范围不同）中都使用了 between 15。以下等效子句由步骤 1 提取：

```
price >=15
```

2　同一表上的搜索参数被提取；在扩展过程中，引用同一个表的所有术语都被视为一个单一谓词。type 和 price 都是 titles 表中的列，因此，提取的子句是：

```
(type = "travel" and price >=15 and price <= 30)
或者
(type = "business" and price >= 15 and price <= 50)
```

3　in 列表和 or 子句被提取。如果块内的一个表有多个 in 列表，则只提取第一个列表。为查询示例提取的列表是：

```
p.pub_id in ("P220", "P583", "P780")
或者
p.pub_id in ("P651", "P066", "P629")
```

由于这些步骤可以重叠和提取同一子句，从而删除了重复项。

生成的每个术语都经过检查，以确定能否将其用作经过优化的搜索参数或 join 子句。只保留那些对于查询优化有用的术语。

附加的子句被添加到用户指定的查询子句中。

例如，在下面的查询中，优化的所有子句都用括号括起来放在 or 子句中：

```
        select p.pub_id, price
from publishers p, titles t
where (
     t.pub_id = p.pub_id
     and type = "travel"
     and price between 15 and 30
     and p.pub_id in ("P220", "P583", "P780")
     )
or  (
     t.pub_id = p.pub_id
     and type = "business"
     and price between 15 and 50
     and p.pub_id in ("P651", "P066", "P629")
     )
```

如上所示，谓词转换从用 or 链接的子句块中提取出用 and 链接的子句。它只提取所有加了括号的块中出现的子句。如果上例中有一个子句位于用 or 连接的某个块中，而它未出现在其它子句中，则不提取这个子句。

处理搜索参数和有用索引

必须区分可用于优化查询的 where 和 having 子句谓词，以及以后在查询处理期间用于过滤返回的行的子句谓词。

当 where 子句中的某列与索引键匹配时，可以使用搜索参数来确定数据行的访问路径。该索引可用于定位和检索匹配的数据行。一旦该行已定位于数据高速缓存，或该行已从磁盘读入数据高速缓存，则会应用任何剩余子句。

例如，如果 authors 表在 au_lname 上有一个索引，并且在 city 上有另一个索引，则每个索引都可用于为此查询确定匹配行：

```
select au_lname, city, state
from authors
where city = "Washington"
and au_lname = "Catmull"
```

查询优化程序使用统计信息（包括直方图、表中的行数、索引高度以及索引和数据页的集群比）来确定哪个索引可提供开销最低的访问。选择访问数据页的开销最低的索引并将其用于执行查询，并在访问数据行后将另一个子句应用到这些数据行。

不等式运算符

不等式运算符 < > 和 != 属于特殊情况。查询优化程序检查在为列建立了索引的情况下，是否应覆盖非聚簇索引，以及在索引覆盖查询时是否使用非匹配索引扫描。但是，如果索引不覆盖查询，则在索引扫描期间，将通过数据页的行 ID 查找来访问表。

搜索参数优化示例

以下为可完全优化的子句的示例。如果有这些列的统计信息，它们可用于帮助估计查询将返回的行数。如果有列的索引，则可以使用这些索引访问数据。

```
au_lname = "Bennett"
price >= $12.00
advance > $10000 and advance < $20000
au_lname like "Ben%" and price > $12.00
```

不能优化这些搜索参数，除非在其上建立功能索引：

```
advance * 2 = 5000  /*expression on column side
                      not permitted */
substring(au_lname,1,3) = "Ben" /* function on
                      column name */
```

如果使用以下格式编写，就可对这两个子句进行优化：

```
advance = 5000/2
au_lname like "Ben%"
```

考虑以下查询，在 au_lname 上只有一个索引：

```
select au_lname, au_fname, phone
   from authors
   where au_lname = "Gerland"
     and city = "San Francisco"
```

该子句限定为一个搜索参数：

```
au_lname = "Gerland"
```

- au_lname 上有一个索引。
- 对于列名没有函数或其它运算。
- 运算符是一个有效搜索参数运算符。

此子句符合除上述第一条外的所有条件； city 列上没有索引。在这种情况下， au_lname 上的索引用于查询。带有匹配姓氏的所有数据页都被调入高速缓存，并检查每个匹配行以查看 city 是否匹配搜索条件。

处理 *join*

查询优化程序处理连接谓词的方式与处理搜索参数的方式一样，它使用统计信息、表中的行数、索引高度、索引与数据页的集群比来确定哪个索引和连接方法可提供开销最低的访问。此外，查询优化程序还使用源自连接直方图的连接密度估计，而连接直方图可对限定连接行和要在外部和内部表中扫描的行进行准确估计。查询优化程序还必须决定最佳连接顺序，以便生成最有效的查询计划。下面几节将介绍处理连接时使用的主要技术。

连接密度和连接直方图

查询优化程序将开销模型用于使用连接属性的表规范化直方图的连接。该技术给出了一个确切的倾斜值 （即频率计数），并使用每个直方图中的域单元密度来估计相应域单元的单元计数。

连接密度是根据 “连接直方图”经过动态计算得出的，其中考虑了来自连接运算符两侧的直方图的连接。第一个直方图连接通常发生在两个基表之间以及两个属性都具有直方图时。每个直方图连接都在父连接投影的相应属性上创建一个新的直方图。

即便连接列的数据分配倾斜，也可以使用连接直方图技术进行准确的选择性估计，从而获得最佳连接顺序和性能。

表达式直方图选择性估计

Adaptive Server 15.0.2 之前的版本使用缺省 “猜口令”进行选择性估计。

Adaptive Server 15.0.2 及更高版本将直方图估计应用于单列谓词 （如果该列上存在直方图）。这样可使行估计更精确，并改进了查询计划的连接次序选择。

在此示列中，如果表达式的选择性非常大，最好还是将表 t1 放在连接顺序的开头：

```
select * from t1,t2 where substring(t1.charcol, 1, 3)
= "LMC" and t1.a1 = t2.b
```

具有混合数据类型的连接

一个基本要求是，尽可能为索引查找建立键，而不考虑任何连接谓词与索引键的混合数据类型。考虑以下查询：

```
create table T1 (c1 int, c2 int)
create table T2 (c1 int, c2 float)
create index i1 on T1(c2)
create index i1 on T2(c2)

select * from T1, T2 where T1.c2=T2.c2
```

假定 T1.c2 属于 int 类型并且它上面有一个索引， T2.c2 属于 float 类型并且具有一个索引。

只要数据类型可隐式转换，查询优化程序就可以使用索引扫描来处理连接。换句话说，即使外部表中查找值的数据类型不同于内部表的相应索引属性，查询优化程序也使用外部表中的列值在内部表上定位索引扫描。

使用表达式和 *or* 谓词的连接

请参见第 8 页的 “为提供额外优化路径所进行的谓词转换和分解”，其中介绍了查询优化程序如何处理使用表达式和 or 谓词的连接。

join 顺序

查询优化程序的主要任务之一是为 join 查询生成查询计划，以使查询执行期间所处理的连接中的关系顺序为最佳。这需要使用将消耗大量时间和内存的详细的计划搜索策略。查询优化程序通过几种有效的技术来获得最佳连接顺序。主要技术包括：

- 利用模糊策略获得一个较好的初始顺序，并以此作为排除其它后续连接顺序的上限。该模糊策略使用连接行估计和嵌套循环连接方法来获得初始顺序。
- 在模糊策略后采用详尽的顺序策略。在这种策略中，可能更好的连接顺序取代了在模糊策略中获得的连接顺序。这种顺序可使用任何 join 方法。
- 利用广泛采用的、基于开销和规则的清理技术排除不需要的连接顺序。清理技术的主要特点是，它始终可以将部分连接顺序 （可能连接顺序的前缀）与最佳完整连接顺序相比较，从而确定是否继续使用给定前缀。这将显著缩短确定最佳连接顺序所需的时间。
- 查询优化程序可识别和处理星型模式或雪花模式连接，并能够以最有效的方式处理其连接顺序。典型的星型模式 join 包含一个大型 Fact 表，该表具有 equi-join 谓词，通过这些谓词，该表与几个 Dimension 表连接。 Dimension 表不具有使它们彼此相连的 join 谓词；也就是说，Dimension 表本身之间没有连接，但在 Dimension 表和 Fact 表之间有 join 谓词。查询优化程序可使用特殊的 join 顺序技术，通过运用这种技术，大型 Fact 表被推至连接顺序的末尾，而 Dimension 表则被拉到前面，从而生成高效查询计划。但是，如果星型模式连接包含子查询、外连接或 or 谓词，则查询优化程序则不会使用该技术。

优化目标

优化的目标在于，通过一种便利方式使查询需求与最佳优化技术相符，从而确保最好地利用优化程序的时间和资源。查询优化程序允许您配置三种类型的优化目标，并且可在三个层次上指定这些目标：服务器级、会话级和查询级。

在所需的级别上设置优化目标。服务器级优化目标高于会话级，会话级又高于查询级。

利用这些优化目标可以选择一种最适合自己的查询环境的优化策略：

- allrows_mix — 缺省优化目标，也是混合查询环境中最有用的目标。allows_mix 可平衡 OLTP 和 DSS 查询环境的需要。
- allrows_dss — 对中等到高度复杂的可操作 DSS 查询最有用的目标。目前，在实验基础上提供该目标。
- allrows_oltp — 查询优化程序只考虑嵌套循环连接。

在服务器级别上，使用 sp_configure。例如：

```
sp_configure "optimization goal", 0, "allrows_mix"
```

在会话级别，使用 set plan optgoal。例如：

```
set plan optgoal allrows_dss
```

在查询级别，使用 select 或其它 DML 命令。例如：

```
select * from A order by A.a plan
   "(use optgoal allrows_dss)"
```

一般来讲，可以使用 select、update 和 delete 语句设置查询级别优化目标。不过，虽然可以在 insert...select 语句中设置优化目标，但不能在纯 insert 语句中设置查询级别优化目标。

限制优化查询所用时间

优化长期运行并且复杂的查询耗时又昂贵。超时机制在提供一个满意查询计划的同时有助于限制优化时间。查询优化程序提供一种机制，优化程序可通过该机制限制长期运行并且复杂的查询所用的时间；超时机制允许查询处理器合理停止优化过程。

优化期间，如果出现以下情况，则优化程序会触发 timeout：

- 已将至少一个完整计划保留为最佳计划，以及
- 已超出用户配置的 timeout 百分比限制。

您可以使用 optimization timeout limit 参数 （可设置为 0 和 1000 之间的任意值）来限制 Adaptive Server 在各个级别用于优化查询的时间量。optimization timeout limit 表示 Adaptive Server 优化查询时必须花费的估计查询执行时间的百分比。例如，指定值为 10，就是让 Adaptive Server 用 10% 的估计查询执行时间来优化查询。类似地，指定值为 1000，就是让 Adaptive Server 用 1000% 或 10 倍的估计查询执行时间来优化查询。

有关 *optimization timeout limit* 的详细信息，请参见 《系统管理指南》中的第 5 章 “设置配置参数”。

长超时值可用于具有复杂查询的存储过程的优化。通常，存储过程的优化时间越长，生成的计划越好；较长的优化时间可在存储过程的几次执行中分摊。

如果想缩短复杂即席查询 （这种查询通常需要很长的编译时间）的编译时间，可以使用较小的超时值。但是，就大部分查询而言，缺省超时值 10 应当足够。

在服务器级别上，使用 sp_configure 设置优化 timeout limit 配置参数。例如，要将优化时间限制为总查询处理时间的 10%，请输入：

```
sp_configure "optimization timeout limit", 10
```

在会话级别上，使用 set 设置超时。

```
set plan opttimeoutlimit <n>
```

其中， *n* 是 0 到 4000 之间的任何整数。

在查询级别上，使用 select 限制优化时间：

```
select * from <table> plan "(use opttimeoutlimit <n>)"
```

其中， n 是 0 到 1000 之间的任何整数。

并行度

Adaptive Server 支持在查询执行过程中应用水平和垂直并行度。垂直并行度是指利用不同系统资源 （如 CPU、磁盘等）同时运行多个运算符的能力。水平并行度是指对数据的指定部分运行同一运算符的多个实例的能力。

请参见第 5 章 “并行查询处理”，以了解对 Adaptive Server 中并行查询优化的详细论述。

优化问题

尽管查询优化程序可以有效优化大部分查询，但以下这些问题可能影响优化程序的效率：

- 如果统计信息近期没有更新，则实际的数据分配可能与用于优化查询的值不匹配。
- 指定的事务所引用的行可能不适合索引统计信息反映的模式。
- 一个索引可能访问表的大部分。
- 采用不可优化的格式编写 where 子句 (SARG)。
- 没有用于关键查询的适当索引。
- 编译存储过程后对基础表作了重大更改。
- 没有有关 SARG 或连接列的统计信息。

上述情况突显出需要遵循一些允许查询优化程序充分发挥其潜能的最佳做法：

创建搜索参数

为查询编写搜索参数时应遵循以下准则：

- 在搜索子句的列端避免使用函数、算术运算和其它表达式。在可能的情况下，将函数和其它运算移到子句的表达式端。
- 使用所有可用搜索参数，为查询处理器提供尽可能多的信息。
- 如果一个查询用于一个表的谓词多于 400 个，应将最可能有用的子句放在查询的靠前位置，因为优化期间只使用每个表上的前 102 个 SARG。（所有搜索条件都将用来限定行。）
- 对于使用 > （大于）的查询，如果可以将它们改写为使用 >= （大于等于），则执行效果可能更好。例如，下面的查询使用 int_col 上的索引查找 int_col 等于 3 的第一个值，然后继续扫描，查找第一个大于 3 的值。如果有很多 int_col 等于 3 的行，服务器就必须扫描大量页，以查找 int_col 大于 3 的第一行：

  ```
  select * from table1 where int_col > 3
  ```

  按如下方式编写该查询会更有效：

  ```
  select * from table1 where int_col >= 4
  ```

  如果有字符串和浮点数据，则这种优化将更为困难。
- 检查 showplan 的输出，查看使用了哪些键和索引。
- 如果预计会使用某个索引但未使用，应使用第 101 页的表 3-1 中 set 命令的输出来查看查询处理器是否考虑到该索引。

使用 SQL 派生表

与表示为两条或多条 SQL 语句的查询相比，表示为单条 SQL 语句的查询能更好地利用查询处理器。可以使用 SQL 派生表通过单个步骤表示查询，否则可能需要几个 SQL 语句和临时表，而在必须存储中间集合结果时尤其如此。例如：

```
select dt_1.* from
   (select sum(total_sales)
        from titles_west group by total_sales)
              dt_1(sales_sum),
   (select sum(total_sales)
         from titles_east group by total_sales)
              dt_2(sales_sum)
where dt_1.sales_sum = dt_2.sales_sum
```

下面的集合结果取自 SQL 派生表 dt_1 和 dt_2，并且在这两个 SQL 派生表之间计算连接。上述所有工作可以通过一条 SQL 语句实现。

有关详细信息，请参见 《Transact-SQL 用户指南》中的第 9 章 “SQL 派生表”。

根据对象大小调优

要了解查询和系统行为，需要知道表和索引的大小。在调优工作的几个阶段，需要用到描述大小的数据：

- 了解关于特定查询计划的 statistics i/o 报告。
- 了解查询处理器对查询计划的选择。 Adaptive Server 基于开销的查询处理器估计每种可能的访问方法所需的物理和逻辑 I/O，并选择开销最低的方法。
- 根据数据库对象的大小和对象上预期的 I/O 模式确定对象的放置。

  要提高性能，应将数据库对象分配到多台物理设备上，以便平均分配读取和写入磁盘操作。

  有关对象放置的描述，请参见 《性能和调优系列：物理数据库调优》。
- 理解性能更改。如果对象增长，它们的性能特性会发生更改。例如，如果某个表使用率很高，且通常被 100% 高速缓存。如果该表对于其高速缓存来讲增长得过大，访问该表的查询的性能就会下降。需要多次扫描的连接尤其会遇到这种情况。
- 制定容量计划。不论您是在设计新系统还是制订现有系统增长计划，都必须了解空间要求，以便规划物理磁盘和内存需求。
- 了解 sp_sysmon 报告中有关物理 I/O 的输出内容。

有关调整大小的详细信息，请参见 《系统管理指南，卷 2》。

Lava 查询执行引擎

在 Adaptive Server 中，所有查询计划都提交给过程执行引擎来执行。过程执行引擎通过以下方式推动查询计划的执行：

- 直接执行诸如 set、 while 和 goto 这类简单的 SQL 语句。
- 调用查询计划的实用程序模块，以执行 create table、 create index 和其它实用程序命令。
- 为存储过程和触发器设置上下文并推动其执行。
- 设置执行环境并调用查询执行引擎，以便为 select、 insert、 delete 和 update 语句执行查询计划。
- 为游标 open、 fetch 和 close 语句设置游标执行环境，并调用查询执行引擎来执行这些语句。
- 进行事务处理和执行后清除。

为支持当今的应用需求， Adaptive Server 15.0 及更高版本的查询执行引擎已被彻底重写。采用新的查询执行引擎和查询优化程序后， Adaptive Server 15.0 及更高版本中的过程执行引擎能够将新的查询优化程序生成的所有查询计划都传送给 Lava 查询执行引擎。

Lava 查询执行引擎执行 Lava 查询计划。优化程序选取的所有查询计划都编译到 Lava 查询计划中。但是，那些没有优化的 SQL 语句，例如 set 或 create 将编译到如同 Adaptive Server 15.0 之前的版本中那样的查询计划中，并且不由 Lava 查询执行引擎执行。非 Lava 查询计划由过程执行引擎执行，或者由过程引擎调用的实用程序模块执行。 Adaptive Server 15.0 及更高版本具有两种独特的查询计划，可以在 showplan 输出中清楚地看到这一点 （请参见第 3 章 “显示查询优化策略和估计值”）。

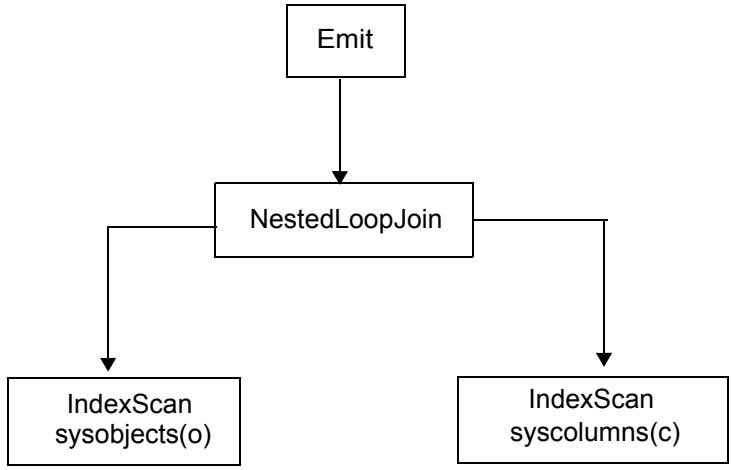
Lava 查询引擎

将 Lava 运算符树颠倒过来，便成为 Lava 查询计划：顶 Lava 运算符可具有一个或多个子运算符，子运算符又可具有一个或多个下一级子运算符，依此类推，最终形成一个 “自下而上”的运算符树。树的确切形状及其中的运算符由优化程序选择。

图 1-2 给出了以下查询的一个 Lava 查询计划示例：

```
Select o.id from sysobjects o, syscolumns c
where o.id <= 1 and o.id < 2
```

图 1-2：Lava 查询计划



该查询的 Lava 查询计划由四个 Lava 运算符组成。顶运算符为 Emit （也叫 Root）运算符，它通过将行发送到客户端或通过为局部变量赋值来分派查询执行结果。

Emit 的唯一子运算符是 NestedLoopJoin (NL Join)，它使用嵌套循环连接算法连接来自它的两个子运算符的行，这两个子运算符分别是：(1) sysobjects 扫描和 (2) syscolumns 扫描。

由于优化程序优化所有 select、 insert、 delete 和 update 语句，因此，它们始终会被编译到 Lava 查询计划中，并由 Lava 查询引擎执行。

有些 SQL 语句编译到混合查询计划中。这类计划具有多个步骤，其中有些步骤由 “实用程序”模块执行，而其最后一步是一个 Lava 查询计划。例如 select into 语句； select into 被编译到一个总共两步的查询计划中：

- create table 创建该语句的目标表。
- 一个 Lava 查询计划，用来将行插入目标表中。为执行该查询计划，过程执行引擎调用 create table 实用程序来执行第一步，以创建表。

然后，过程引擎调用 Lava 查询执行引擎来执行 Lava 查询计划，以选择行并将其插入目标表中。

另外两个生成混合查询计划的 SQL 语句是 alter table （仅在需要复制数据时使用）和 reorg rebuild。

同时还会生成和执行一个 Lava 查询计划以支持 bcp。 Adaptive Server 中对 bcp 的支持始终存在于 bcp 实用程序中。如今，在 15.0 及更高版本中， bcp 实用程序可生成一个 Lava 查询计划，并调用 Lava 查询执行引擎来执行该计划。

有关 Lava 查询计划的更多示例，请参见第 2 章 “使用 showplan”。

Lava 运算符

Lava 查询计划由 Lava 运算符构成。每个 Lava 运算符都是一个独立的软件对象，该对象可实现优化程序用来建立查询计划的某个基本物理操作。每个 Lava 运算符包含五种方法，这些方法可被其父运算符调用。这五种方法对应于查询执行的五个阶段：

- Acquire
- Open
- Next
- Close
- Release

因为 Lava 运算符全都提供相同的方法 （即 API 相同），因此它们可以互换，就像 Lava 查询计划中的构件块一样。例如，可以将图 1-2 中的 NLJoin 运算符替换为 MergeJoin 运算符或 HashJoin 运算符，而不会对查询计划中的任何其它三个运算符造成影响。

表 1-3 中列出了优化程序可选择用来建立 Lava 查询计划的 Lava 运算符：

表 1-3：Lava 运算符

运算符	说明
BulkOp	执行在 Lava 查询引擎中完成的 bcp 处理部分。只能在 bcp 实用程序而非优化程序创建的查询计划中找到。
CacheScanOp	读取内存中表中的行。
DelTextOp	作为 alter table drop 列处理的一部分，删除 text 页链。
DeleteOp	删除本地表中的行。 当整个 SQL 语句无法传送到远程服务器上时，删除代理表中的行。另请参见 RemoteScanOp。
EmitOp (RootOp)	路由查询执行结果行。可发送结果到客户端，或将结果值赋给局部变量或 fetch into 变量。 EmitOp 始终是 Lava 查询计划中的顶运算符。
EmitExchangeOp	将并行执行的子计划的结果行路由到父计划片段中的 ExchangeOp。EmitExchangeOp 总是直接出现在 ExchangeOp 下。请参见第 5 章 “并行查询处理”。

运算符	说明
GroupSortedOp (Aggregation)	当输入行已在 group-by 列上排序时，执行矢量集合 (group by)。另请参见 HashVectorAggOp。
GroupSorted (Distinct)	删除重复行。要求在所有列上排序输入行。另请参见 HashDistinctOp 和 SortOp (Distinct)。
HashVectorAggOp	执行矢量集合 (group by)。使用 Hash 算法对输入行分组，因此，对输入行顺序无要求。另请参见 GroupSortedOp (Aggregation)。
HashDistinctOp	使用可查找重复行的散列算法删除重复行。另请参见 GroupSortedOp (Distinct) 和 SortOp (Distinct)。
HashJoinOp	使用 HashJoin 算法执行两个输入行流的 join。
HashUnionOp	使用可查找和消除重复行的散列算法执行两个或多个输入行流的 union 操作。另请参见 MergeUnionOp 和 UnionAllOp。
InsScrollOp	实现所需的额外处理，以支持不敏感的可滚动游标。另请参见 SemiInsScrollOp。
InsertOp	将行插入本地表。 当整个 SQL 语句无法传送到远程服务器上时，在代理表中插入行。另请参见 RemoteScanOp。
MergeJoinOp	使用 mergejoin 算法，对在连接列上排序的两个行流执行 join。
MergeUnionOp	对两个或多个排序的输入流执行 union 或 union all 操作。确保输出流保持输入流的顺序。另请参见 HashUnionOp 和 UnionAllOp。
NestedLoopJoinOp	使用 NestedLoopJoin 算法执行两个输入流的连接。
NaryNestedLoopJoinOp	使用增强的 NestedLoopJoin 算法执行三个或更多输入流的连接。该运算符可取代 NestedLoopJoin 运算符的深度左树，如果可以跳过某些输入流的行，则可显著提高性能。
OrScanOp	将 in 或 or 值插入内存中的表，对这些值进行排序，并删除重复值。然后返回这些值，一次一个。只用于以下 SQL 语句：在同一列上具有 in 子句或多个 or 子句。
PtnScanOp	通过表扫描或索引扫描来访问行，从而读取本地表 （不考虑是否分区）中的行。
RIDJoinOp	从其左侧的子运算符接收一个或多个行标识符 (RID)，并调用其右侧的子运算符 (PtnScanOp) 来查找相应行。仅用于在同一表的不同列上具有 or 子句的 SQL 语句。
RIFilterOp (Direct)	推动子计划的执行，以强制实施可逐行检查的参照完整性约束。 只出现在对具有参照完整性约束的表的 insert、 delete 或 update 查询中。
RIFilterOp (Deferred)	推动子计划的执行，以强制实施只能在处理完受查询影响的所有行后才能检查的参照完整性约束。
RemoteScanOp	访问代理表。 RemoteScanOp 可以： • 从单个代理表读取行，以便通过本地主机上的 Lava 查询计划做进一步的处理。 • 将完整的 SQL 语句传递到远程主机以执行：insert、delete、update 和 select 语句。在这种情况下， Lava 查询计划由一个 EmitOp 组成，它将 RemoteScanOp 作为其唯一的子运算符。 • 将任意复杂查询计划片段传递到远程主机，以便执行并读取结果行 （函数传递）。
RestrictOp	对表达式求值。

运算符	说明
SQFilterOp	推动子计划的执行，以执行一个或多个子查询。
ScalarAggOp	执行标量集合，例如没有 group by 的集合。
SemiInsScrollOp	执行额外处理，以支持半不敏感的可滚动游标。另请参见 InsScrollOp。
SequencerOp	强制查询计划中不同的子计划有序执行。
SortOp	基于指定键排序其输入行。
SortOp (Distinct)	对其输入进行排序并删除重复行。另请参见 HashDisitnctOp 和 GroupSortedOp (Distinct)。
StoreOp	创建并协调工作表填充，如果需要，在工作表上创建聚簇索引。 StoreOp 只能有一个 InsertOp 作为子运算符； InsertOp 填充工作表。
UnionAllOp	对两个或更多输入流执行 union all 操作。另请参见 HashUnionOp 和 MergeUnionOp。
UpdateOp	当无法将整个 update 语句发送到远程服务器时，更改本地表或代理表的行中的列值。另请参见 RemoteScanOp。
ExchangeOp	启用并协调 Lava 查询计划的并行执行。可在查询计划的几乎任何两个 Lava 运算符之间插入 ExchangeOp，以便将计划分成可并行执行的多个子计划。请参见第 5 章 “并行查询处理”。

Lava 查询执行

可以分五个阶段执行 Lava 查询计划：

1 Acquire — 获得执行计划所需的资源，例如内存缓冲区，并创建工作表。

2 Open — 准备返回结果行。

3 Next — 生成下一个结果行。

4 Close — 进行清理；例如，通知访问层扫描已完成或截断工作表

5 Release — 释放在 “Acquire”阶段获得的资源，例如内存缓冲区，并删除工作表。

每个 Lava 运算符都具有一个与阶段同名的方法，将为每一个阶段调用该方法。

第 19 页的图 1-2 演示了查询计划执行过程：

- Acquire 阶段

  调用 Emit 运算符的 Acquire 方法。 Emit 运算符调用其 NLJoin 子运算符的 Acquire 方法，后者反过来在其左侧的子运算符 （sysobjects 的索引扫描）上调用 Acquire 方法，然后在其右侧的子运算符 （syscolumns 的索引扫描）上调用 Acquire 方法。

- Open 阶段

  调用 Emit 运算符的 Open 方法。 Emit 运算符调用 NLJoin 运算符的 Open 方法，后者只调用其左侧子运算符的 Open 方法。

- Next 阶段

  调用 Emit 运算符的 Next 方法。Emit 调用 NLJoin 运算符的 Next 方法，后者调用其左侧子运算符 （*sysobjects* 的索引扫描）的 Next 方法。索引扫描运算符从 *sysobjects* 中读取第一行，并将其返回给 NLJoin 运算符。然后， NLJoin 运算符调用其右侧的子运算符 （*syscolumns* 的索引扫描）的 Open 方法。接下来， NLJoin 运算符调用 *syscolumns* 的索引扫描的 Next 方法，以获得与 *sysobjects* 中行的连接键匹配的行。如果已找到匹配行，则将其返回给 Emit 运算符，后者再将其发送回客户端。重复调用 Emit 运算符的 Next 方法会生成更多结果行。

- Close 阶段

  返回所有行后，调用 Emit 运算符的 Close 方法，后者又会调用 NLJoin 运算符的 Close 方法，而这又会调用其两个子运算符的 Close 方法。

- Release 阶段

  调用 Emit 运算符的 Release 方法，对其它运算符的 Release 方法的调用沿查询计划传播。

成功执行 Release 阶段后，Lava 查询引擎将控制权交还给过程执行引擎，以便进行最后的语句处理。

更新操作如何运行

根据对用于定位行的索引和数据所做更改的不同， Adaptive Server 会采用不同的方法进行更新。两种主要的更新类型是**延迟更新**和**直接更新**。Adaptive Server 会尽可能执行直接更新。

直接更新

Adaptive Server 在单一的传递中执行直接更新：

- 定位受影响的索引和数据行。
- 将更改的日志记录写入事务日志中。
- 更改数据页和任何受影响的索引页。

有三种执行直接更新的技术：

- 现场更新
- 简易直接更新
- 昂贵的直接更新

和延迟更新相比，直接更新能限制日志扫描的数量、减少日志记录、减少 B 树索引的浏览 （减少锁争用），所以其所需开销更少，而且通常速度更快。另外，还因为 Adaptive Server 不必重新读取页以执行基于日志记录的修改，所以也能节省 I/O。

现场更新

Adaptive Server 会在任何可能的时候执行现场更新。

当 Adaptive Server 执行现场更新时，页上后续的行不会被移动；行 ID 保持不变并且行偏移表中的指针不会改变。

对于现场更新，必须满足下列要求：

- 不能改变被更改行的长度。
- 正在更新列不能是所有页锁定表上聚簇索引的键，或键的一部分。因为所有页锁定表上聚簇索引中的行按键顺序进行存储，所以键的改变也就总是意味着行位置的改变。
- 一个索引必须是唯一的，多个索引必须允许重复。
- update 语句满足第 29 页的 “通过连接对更新模式的约束”中列出的条件。
- 受影响的列不用于参照完整性。
- 在列上不能存在触发器。
- 表不能 （通过 Replication Server）被复制。

现场更新是最快的更新类型，因为它仅对数据页进行一次更改。它通过删除旧的索引行并插入新的索引行，来改变所有受影响的索引条目。现场更新只影响到那些键被更新所改变的索引，因为页和行的位置没有改变。

简易直接更新

如果 Adaptive Server 不能执行现场更新，它会尝试执行简易直接更新—更改行并在页上的相同偏移位置重写该行。将页上的后续行上下移动，以使页上的数据保持连续性，但行 ID 不变。行偏移表中的指针更改以反映新的位置。

简易直接更新必须满足下列要求：

- 行中的数据长度发生了变化，但行仍然在同一数据页上，或是行的长度未改变，但复制了表或表中一个触发器。
- 正在更新的列不能是聚簇索引的键，或键的一部分。因为 Adaptive Server 将按键顺序存储聚簇索引行，所以键的改变也就总是意味着行位置的改变。
- 一个索引必须是唯一的，多个索引必须允许重复。
- update 语句满足第 29 页的 “通过连接对更新模式的约束”中列出的条件。
- 受影响的列不用于参照完整性。

简易直接更新的速度几乎与现场更新同样快。它们需要相同数量的 I/O，但简易直接更新所需处理要稍微多一些。它对数据页进行两次更改 （行和偏移表）。任何被改动的索引键都是通过删除旧值和插入新值而更新。简易直接更新只影响到那些键被更新所改变的索引，因为页和行的 ID 没有改变。

昂贵的直接更新

如果数据不在同一页上， Adaptive Server 会执行昂贵的直接更新 （如有可能）。昂贵的直接更新将删除数据行 （包括所有索引条目），然后插入已修改的行和索引条目。

Adaptive Server 使用表扫描或索引来在其初始位置找到行，然后将其删除。如果表带有聚簇索引， Adaptive Server 则使用索引来确定行的新位置；否则， Adaptive Server 将新行插入到堆的末尾。

昂贵的直接更新必须满足下列要求：

- 行数据长度改变了，所以行不再位于同一数据页上，而是移动到了不同的页，或是该更新影响到了聚簇索引的键列。
- 用于查找行的索引未被更新更改。
- update 语句满足第 29 页的 “通过连接对更新模式的约束”中列出的条件。
- 受影响的列不用于参照完整性。

昂贵的直接更新是最慢的直接更新类型。它在某一数据页上执行 delete，却在另一数据页上执行 insert。所有的索引条目都必须更新，因为行位置发生了更改。

延迟更新

当不能满足直接更新条件时， Adaptive Server 使用延迟更新。延迟更新是最慢的更新类型。

在延迟更新中， Adaptive Server：

- 定位受影响的数据行，并在定位行时将数据页延迟的删除和插入写入日志记录。
- 读取事务的日志记录，并对数据页和任何受影响的索引行执行删除操作。
- 再次读取日志记录，对数据页上执行所有插入操作，插入所有受影响的索引行。

何时需要使用延迟更新

下列情况下总是需要使用延迟更新：

- 使用自连接的更新
- 对用于自参照完整性的列进行更新
- 对在相关子查询中被引用的表进行更新

下列情况下也需要使用延迟更新：

- 某个更新操作将某一行移动到新页上，并通过表扫描或聚簇索引对表进行访问。
- 表中不允许出现重复行，并且没有唯一索引来防止这种情况的出现。
- 用于查找数据行的索引不是唯一的，并且由于更新改变了聚簇索引键或是新行不在该页上，而导致了行的移动。

和直接更新相比，延迟更新需要更大的开销，因为后者要求 Adaptive Server 重新读取事务日志以对数据和索引进行最终更改。这涉及到了对索引树额外浏览。

例如，如果 title 上具有聚簇索引，则此查询执行延迟更新：

```
update titles set title = "Portable C Software" where
title = "Designing Portable Software"
```

延迟索引插入

当更新影响到用于访问表的索引或某个唯一索引中的列时， Adaptive Server 执行延迟索引更新。在此类更新中， Adaptive Server：

- 删除直接模式中的索引条目。
- 更新直接模式中的数据页，写入索引的延迟插入记录。
- 读取事务日志记录，并将新值插入延迟模式的索引中。

当更新更改了用于查找行的索引或影响到某个唯一索引时，必须使用延迟索引插入模式。查询只能对一个限定行更新一次 — 延迟索引更新模式可确保在索引扫描期间一行只找到一次，并且查询不会过早地与唯一约束发生冲突。

第 28 页的图 1-3 中的更新只更改姓氏，但索引行从一页移到了下一页。要执行更新， Adaptive Server：

1　读取索引页 1133，从该页删除对应于 “Greene” 的索引行，并将延迟索引扫描的记录记入日志。

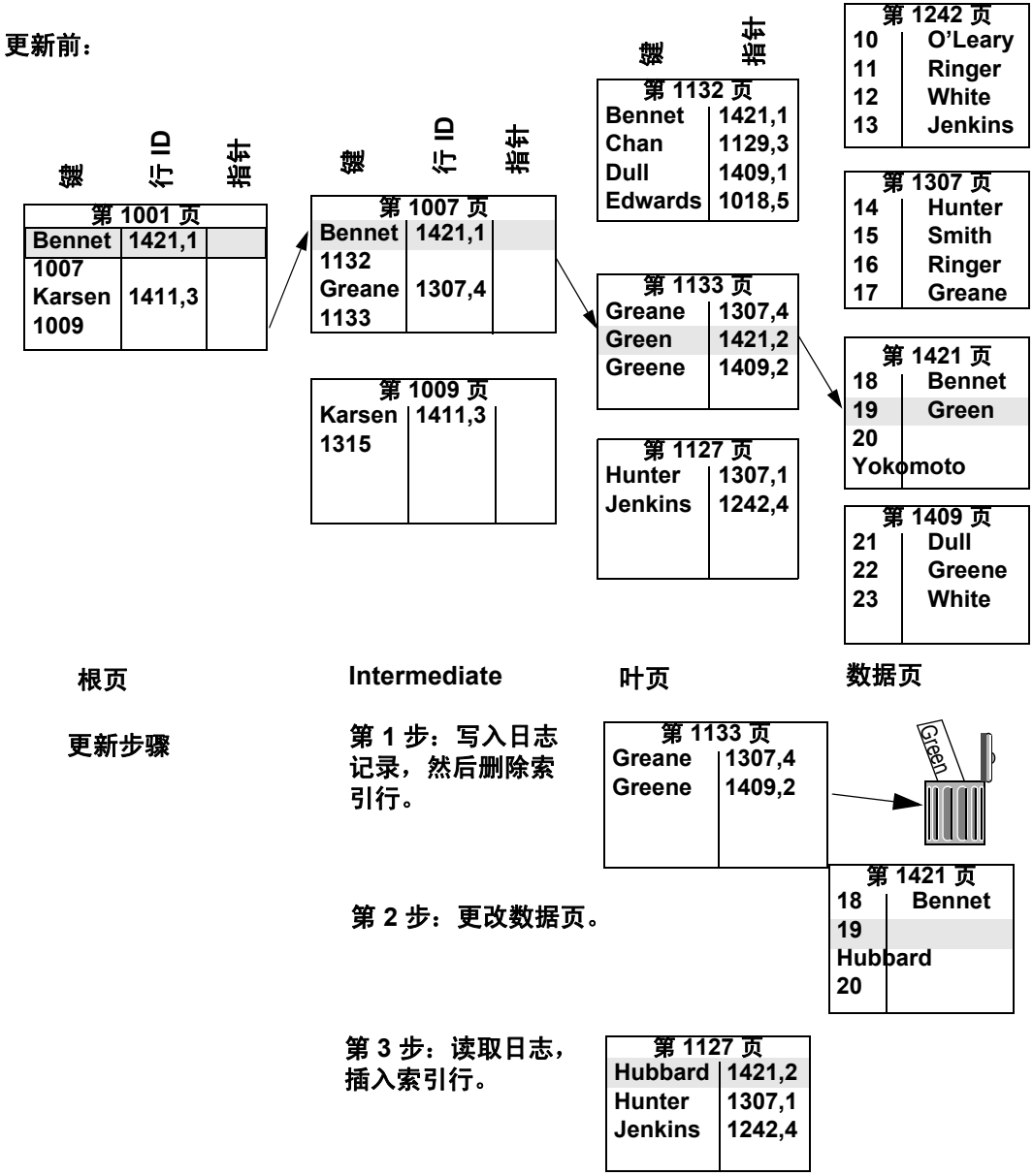
2　在直接模式中将数据页上的 “Green” 改为 “Hubbard”，并继续索引扫描，检查是否有更多的行需要更新。

3　在第 1127 页上插入对应 “Hubbard” 的新索引行。

图 1-3 显示了延迟更新操作之前的索引与数据页，以及延迟更新操作更改数据和索引页的顺序。

图 1-3：延迟索引更新

```
update employee
set lname = "Hubbard"
where lname = "Green"
```



对 titles 表进行类似的更新：

```
update titles
set title = "Computer Phobic's Manual",
    advance = advance * 2
where title like "Computer Phob%"
```

此查询暴露出一个潜在的问题。如果扫描 title 列上的非聚簇索引时找到“Computer Phobia Manual”，并更改了书名，将 advance 乘以 2；然后又找到新索引行 “Computer Phobic's Manual”，又将 advance 乘以 2，这会造成 advance 远远超出实际值。

延迟索引删除操作可能比昂贵的直接更新速度更快，但也可能会更慢，这取决于需要被扫描的日志记录数以及日志页是否仍在高速缓存中。

数据行延迟更新期间，在删除索引行和插入新索引行之间可能会有非常长的时间间隔。在此间隔期间，不存在和数据行相对应的索引行。如果此间隔期间，某个进程在隔离级别 0 上扫描索引，则它不会返回数据行的旧值或新值。

通过连接对更新模式的约束

当被更新的表是连接顺序中的最外层表，或是在连接顺序中该表仅位于只有一行符合条件的表之后，则涉及连接运算的更新和删除可使用直接模式、 deferred_varcol 模式或 deferred_index 模式。

update 和 delete 语句中的连接和子查询

在 update 和 delete 语句中使用 from 子句来执行连接是一种对 ANSI SQL 的 Transact-SQL 扩展。 ANSI SQL 格式的子查询可替代连接用于某些更新和删除任务。

下面的示例使用 from 语句执行一个连接：

```
update t1 set t1.c1 = t1.c1 + 50
from t1, t2
where t1.c1 = t2.c1
and t2.c2 = 1
```

以下示例显示了使用子查询的等效更新。

```
update t1 set c1 = c1 + 50
where t1.c1 in (select t2.c1
                from t2
                where t2.c2 = 1)
```

用于连接查询的更新模式取决于被更新的表是否是连接顺序中的最外层查询 — 如果不是最外层表，更新将以延迟模式执行。使用子查询的更新总是以直接更新、deferred_varcol 更新或 deferred_index 更新的方式执行。

如果查询使用 from 语法并因为连接顺序而执行延迟更新，则可使用 showplan 和 statistics io 确定使用子查询重写查询是否可提高性能。并不是所有使用 from 的查询都可使用子查询进行重写。

触发器与参照完整性中的删除和更新

使用 deleted 表或 inserted 表连接用户表的触发器以延迟模式运行。如果使用触发器只是为了实现参照完整性，而不是为了将更新和删除实现级联，那么假如使用声明的参照完整性代替触发器，就可能会避免在触发器中因延迟更新而引起的时间浪费。

优化更新

showplan 消息可提供有关更新是以直接模式还是延迟模式执行的信息。如果不能使用直接更新， Adaptive Server 以延迟模式更新数据行。因为有时优化程序不能确定将执行直接更新还是延迟更新，所以 showplan 消息将提供：

- “deferred_varcol”消息显示更新操作可能更改行的长度，因为某个可变长度列被更新了。如果被更新的行在页上，更新以直接模式执行；如果更新不在页上，则更新以延迟模式执行。
- “Deferred_index”消息表明对数据页的更改操作和对索引页的删除操作以直接模式执行，但对索引页的插入操作以延迟模式执行。

上述直接更新的类型取决于运行时的可用信息，因为实际上要读取页并进行检查以确定行是否在页上。

直接更新的设计

在设计和编写应用程序时，应注意可能引起延迟更新的不同之处。要帮助避免延迟更新，请执行以下操作：

- 至少创建一个表的唯一索引，以增加执行直接更新的可能性。
- 在更新其它键时，尽可能在 where 子句中使用非键列。
- 如果列中不使用空值，应在 create table 语句中将其定义为 not null。

更新类型和索引对更新模式的影响

第 31 页的表 1-4 显示了对于三种不同类型的更新，索引如何影响更新模式。任何情况下都不允许出现重复行。所有索引均建立在 title_id 上。这三种更新类型如下：

- 可变长度键列的更新：

```
update titles set title_id = value
    where title_id = "T1234"
```

- 固定长度非键列的更新：

```
update titles set pub_date = value
        where title_id = "T1234"
```

- 可变长度非键列的更新：

```
update titles set notes = value
    where title_id = "T1234"
```

表 1-4 显示了对于同一键，唯一索引可使用比非唯一索引效率更高的更新模式。应特别注意表阴影区中直接和延迟的差别。例如，使用唯一聚簇索引，所有更新都能以直接模式执行；但如果索引是非唯一的，则更新必须以延迟模式执行。

对于使用非唯一聚簇索引的表，表中任何其它列上的唯一索引都可提高更新执行性能。在某些情况下，可能需要在表中增加一个 IDENTITY 列，以便能将列作为键包括在索引中，否则该索引不具有唯一性。

表 1-4：索引对更新模式的影响

	更新为：		
索引	**可变长度键**	**固定长度列**	**可变长度列**
无索引	不适用	直接	deferred_varcol
聚簇，唯一	直接	直接	直接
聚簇，非唯一	延迟	延迟	延迟
聚簇，非唯一，其它列上有一个唯一索引	延迟	直接	deferred_varcol
非聚簇，唯一	deferred_varcol	直接	直接
非聚簇，非唯一	deferred_varcol	直接	deferred_varcol

如果索引的键长度是固定的，表中所示各种情况的更新模式唯一不同之处都在于非聚簇索引。对于非聚簇非唯一索引，键更新的模式是 deferred_index。对于非聚簇唯一索引，键更新的模式是直接更新。

如果 varchar 或 varbinary 的长度接近最大长度，则改用 char 或 binary。每个可变长度列都会将行加到顶部，增加可使用延迟更新的可能性。

如果使用 max_rows_per_page 减少每页允许的行数，则可增加直接更新数，因为增加可变长度列长度的更新可能仍在同一页上。

有关使用 max_rows_per_page 的详细信息，请参见 《性能和调优系列：物理数据库调优》。

调优更新时使用 *sp_sysmon*

可使用 showplan 确定某个更新是直接更新还是延迟更新，但 showplan 不能提供有关延迟或直接更新类型的详细信息。 sp_sysmon 或 Adaptive Server Monitor 的输出提供了有关采样间隔期间所执行的更新类型的详细统计信息。

如在调优更新时运行 sp_sysmon，可显示减少的延迟更新数、锁定数以及 I/O 数。

请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

第 2 章

使用 *showplan*

本章介绍由 showplan 实用程序输出的消息，该实用程序采用基于文本的格式显示批处理或存储过程中每个 SQL 语句的查询计划。

显示查询计划

若要查看查询计划，请使用：

```
set showplan on
```

若要停止显示查询计划，请使用：

```
set showplan off
```

您可以将 showplan 和其它 set 命令一起使用。

若要对某个存储过程显示查询计划，但不执行这些计划，请使用 set fmtonly 命令。

有关各选项如何交互的信息，请参见第 12 章 “创建和使用抽象计划”。

注释 不要对存储过程使用 set noexec，否则无法进行编译和执行，您也得不到所需输出。

Adaptive Server Enterprise 15.0 和更高版本中的查询计划

Adaptive Server 通常将 Transact-SQL 语句划分为两组：

- 可优化的。例如，以下查询是可优化的，因为它有很多关系 （表）：

  ```
  select * from t1, t2, t3, t4
  where t1.c1 = t2.c1 and . . .
  order by t3.c4
  ```

  查询处理器需要连接顺序、连接类型、搜索参数和排序可进行优化。

- 不可优化的。诸如 update statistics 和 dbcc 实用程序命令不可被优化。

在 Adaptive Server version 15.0 和更高版本中，优化程序和大多数执行引擎已重写。目前，实用程序命令会生成与早期版本几乎相同的 showplan 输出。但是， 15.0 和更高版本会为可优化的语句生成新的 showplan 输出。

showplan 必须显示的查询计划的一些新特征包括：

- 计划元素 — 查询计划可由 30 多种不同的运算符组成。
- 计划形状 — 查询计划是颠倒过来的运算符树。一般情况下，查询计划中的运算符越多，可能的树形状组合就越多。 15.0 和更高版本的查询计划比早期版本中的查询计划更复杂，其中提供了嵌套缩进，以帮助可视化这些查询计划的树形状。
- 并行执行的子计划。

相同的查询为什么会有不同的查询计划？

查询处理器可能会返回不同的查询计划，具体取决于您是否为 allrows_oltp、 allrows_mix 或 allrows_dss 配置了 set plan optgoal （除非您使用 forceplan 强制实施计划）：

- allrows_oltp — 查询处理器使用嵌套循环连接运算符。
- allrows_mix — 查询处理器允许使用嵌套循环连接和合并连接。查询处理器会计算它们相应的开销，以确定使用哪一种连接。
- allrows_dss — 查询处理器使用嵌套循环连接、合并连接或散列连接。查询处理器会计算它们相应的开销，以确定使用哪一种连接。

结合使用 set showplan 和 noexec

可以将 set noexec 与 set showplan 一起使用，以查看查询计划，但不执行查询。例如，以下查询不但输出查询计划，而且还执行查询，这可能会很费时间：

```
set showplan on
go
select * from really_big_table
select * from really_really_big_table
go
```

然而，如果包括 set noexec，则可以查看查询计划，而不运行查询。

首次使用存储过程时或如果生成的编译计划已被另一个会话占用，则会编译存储过程，因此 set noexec 可能产生意外结果，Sybase® 建议您改用 set fmtonly on。如果在另一个存储过程中包含存储过程，则启用 set noexec 时不会运行第二个存储过程。例如，如果创建两个存储过程：

```
create procedure sp_B
   as
   begin
       select * from authors
   end
```

和

```
create procedure sp_A
   as
   begin
       select * from titles
       execute sp_B
   end
```

个别情况下，它们的查询计划如下所示：

```
set showplan on
sp_B
QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
            The type of query is EXECUTE.
QUERY PLAN FOR STATEMENT 1 (at line 4).

     STEP 1

       The type of query is SELECT.
       1 operator(s) under root
       |ROOT:EMIT Operator (VA = 1)
```

```
        |
        |    |SCAN Operator (VA = 0)
        |    |  FROM TABLE
        |    |  titles
        |    |  Table Scan.
        |    |  Forward Scan.
        |    |  Positioning at start of table.
        |    |  Using I/O Size 2 Kbytes for data pages.
        |    |  With LRU Buffer Replacement Strategy for data pages.
```

如果启用 set noexec：

```
set noexec on
go
set showplan on
go
exec proc A
go
```

Adaptive Server 没有为过程 B 生成 showplan 输出，因为已启用 noexec，因此 Adaptive Server 实际上不执行或编译过程 B，且不会输出任何 showplan 输出。如果未启用 noexec，则 Adaptive Server 会为 A 和 B 存储过程编译和输出计划。

但如果使用 set fmtonly on：

```
use pubs2
go
create procedure sp_B
as
begin
     select * from authors
end
go
create procedure sp_A
as
begin
     select * from titles
     execute sp_B
end
go
set showplan on
go
set fmtonly on
go

QUERY PLAN FOR STATEMENT 1 (at line 1).
```

```
   STEP 1
     The type of query is SET OPTION ON.

sp_B
go

QUERY PLAN FOR STATEMENT 1 (at line 1).

   STEP 1
     The type of query is EXECUTE.

QUERY PLAN FOR STATEMENT 1 (at line 4).

     STEP 1

       The type of query is SELECT.
       1 operator(s) under root
       |ROOT:EMIT Operator (VA = 1)
       |
       |   |SCAN Operator (VA = 0)
       |   |  FROM TABLE
       |   |  authors
       |   |  Table Scan.
       |   |  Forward Scan.
       |   |  Positioning at start of table.
       |   |  Using I/O Size 2 Kbytes for data pages.
       |   |  With LRU Buffer Replacement Strategy for data pages.


au_id         au_lname                                  au_fname
         phone         address
         city                  state   country     postalcode
----------    ----------------------------------------   --------------------
         ------------  ----------------------------------------
         ------------------   -----   ----------  ----------

(0 rows affected)
(return status = 0)
sp_A
go

QUERY PLAN FOR STATEMENT 1 (at line 1).


   STEP 1
```

```
     The type of query is EXECUTE.


QUERY PLAN FOR STATEMENT 1 (at line 4).


     STEP 1

       The type of query is SELECT.
       1 operator(s) under root
       |ROOT:EMIT Operator (VA = 1)
       |
       |   |SCAN Operator (VA = 0)
       |   |  FROM TABLE
       |   |  titles
       |   |  Table Scan.
       |   |  Forward Scan.
       |   |  Positioning at start of table.
       |   |  Using I/O Size 2 Kbytes for data pages.
       |   |  With LRU Buffer Replacement Strategy for data pages.


QUERY PLAN FOR STATEMENT 2 (at line 5).

   STEP 1
     The type of query is EXECUTE.

title_id
     title

     type           pub_id     price                 advance
     total_sales
     notes

     pubdate                 contract
--------
     ------------------------------------------------------------------------
     -----------    --------   -------------------   ------------------------
     -----------
     ------------------------------------------------------------------------
-------------------------------------------------------------------------------
-----------------------------------------------
     --------------------    --------

(0 rows affected)
```

```
QUERY PLAN FOR STATEMENT 1 (at line 4).


     STEP 1

       The type of query is SELECT.
       1 operator(s) under root
       |ROOT:EMIT Operator (VA = 1)
       |
       |   |SCAN Operator (VA = 0)
       |   |  FROM TABLE
       |   |  authors
       |   |  Table Scan.
       |   |  Forward Scan.
       |   |  Positioning at start of table.
       |   |  Using I/O Size 2 Kbytes for data pages.
       |   |  With LRU Buffer Replacement Strategy for data pages.

au_id         au_lname                                  au_fname
         phone         address
         city                 state   country     postalcode
----------    ----------------------------------------   --------------------
         ------------  ----------------------------------------
         ------------------   -----   ----------   ----------
```

两个存储过程均运行，并且您将看到生成的 showplan 输出。

语句级别输出

每个查询计划的 showplan 输出的第一部分都提供语句级别信息，包括为其生成查询计划的查询的批处理或存储过程中的语句和行号：

```
QUERY PLAN FOR STATEMENT N (at line N).
```

此消息后面可能有一系列适用于语句的整个查询计划的消息。如果使用包含抽象计划的强制实施方式的抽象计划生成查询计划：

- 如果显式抽象计划是由 SQL 语句中的 plan 子句提供的，则消息为：

  ```
  Optimized using the Abstract Plan in the PLAN clause.
  ```

- 如果已在内部生成抽象计划 （即针对并行执行的 alter table 和 reorg 命令），则消息为：

  ```
  Optimized using the forced options (internally
  generated Abstract Plan).
  ```

- 如果高速缓存新语句，则输出包括：

  ```
  STEP 1

  The type of query is EXECUTE.
  Executing a newly cached statement.
  ```

- 如果重用高速缓存的语句，则输出包括：

  ```
  STEP 1

  The type of query is EXECUTE.
  Executing a previously cached statement.
  ```

- 如果查询重新编译语句，则输出包括：

  ```
  QUERY PLAN IS RECOMPILED DUE TO SCHEMACT.

  THE RECOMPILED QUERY PLAN IS:

  . . .

  QUERY PLAN FOR STATEMENT 1 (at line 1)

  . . .
  ```

- 如果抽象计划是从 sysqueryplans 检索的 （因为启用了自动抽象计划使用），则消息为：

  ```
  Optimized using an Abstract Plan (ID : N).
  ```

- 如果查询计划是并行查询计划，则以下消息显示执行该查询计划所需的进程 （事务协调器进程和工作进程）数：

  ```
  Executed in parallel by coordinating process and N
  worker processes.
  ```

- 如果查询计划是使用模拟统计信息优化的，则紧接着显示以下消息：

  ```
  Optimized using simulated statistics.
  ```

- 输出包括 VA=，指示运算符和每个运算符执行顺序的虚拟地址。查询处理器从 VA=0 启动。通常情况下，将首先执行扫描节点（叶节点）。

  注释　showplan 输出中的 VA= 适用于 Adaptive Server 15.0.2 ESD #2 和更高版本。在早期版本的 Adaptive Server 中，您将无法看到 VA=。

- Adaptive Server 对在查询执行过程中访问的每个数据库对象使用扫描描述符。缺省情况下，每个连接（或并行查询计划的每个工作进程）都有 28 个扫描描述符。如果查询计划需要访问的数据库对象超过 28 个，则从全局池中分配辅助扫描描述符。如果查询计划使用辅助扫描描述符，则输出以下消息，显示需要的总数：

  ```
  Auxiliary scan descriptors required: N
  ```

- 以下消息显示查询计划中出现的运算符的总数：

  ```
  N operator(s) under root
  ```

- 下一条消息显示查询计划的查询类型。对于查询计划，查询类型为 select、 insert、 delete 或 update：

  ```
  The type of query is SELECT.
  ```

- 如果 Adaptive Server 配置为启用资源限制，则在 showplan 输出的结尾输出最后一条语句级别消息。该消息显示优化程序的逻辑和物理 I/O 的估计的总开销：

  ```
  Total estimated I/O cost for statement N (at line M): X.
  ```

以下查询（带有 showplan 输出）显示其中一些消息：

```
use pubs2
go
set showplan on
go
select stores.stor_name, sales.ord_num
from stores, sales, salesdetail
where salesdetail.stor_id = sales.stor_id
and stores.stor_id = sales.stor_id
plan " ( m_join ( i_scan salesdetailind salesdetail)
( m_join ( i_scan salesind sales ) ( sort ( t_scan stores ) ) ) )"

QUERY PLAN FOR STATEMENT 1 (at line 1).
Optimized using the Abstract Plan in the PLAN clause.

STEP 1
The type of query is SELECT.
```

```
6 operator(s) under root

ROOT:EMIT Operator (VA = 6)

    |MERGE JOIN Operator (Join Type: Inner Join) (VA = 5)
    | Using Worktable3 for internal storage.
    |  Key Count:1
    |  Key Ordering:ASC
    |
    |   |SCAN Operator (VA = 0)
    |   |  FROM TABLE
    |   |  salesdetail
    |   |  Index : salesdetailind
    |   |  Forward Scan.
    |   |  Positioning at index start.
    |   |  Index contains all needed columns. Base table will not be read.
    |   |  Using I/O Size 2 Kbytes for index leaf pages.
    |   |  With LRU Buffer Replacement Strategy for index leaf pages.
    |
    |   |MERGE JOIN Operator (Join Type: Inner Join) (VA = 4)
    |   | Using Worktable2 for internal storage.
    |   |  Key Count: 1
    |   |  Key Ordering: ASC
    |   |
    |   |   |SCAN Operator (VA = 1)
    |   |   |  FROM TABLE
    |   |   |  sales
    |   |   |  Table Scan.
    |   |   |  Forward Scan.
    |   |   |  Positioning at start of table.
    |   |   |  Using I/O Size 2 Kbytes for data pages.
    |   |   |  With LRU Buffer Replacement Strategy for data pages.
    |   |
    |   |   |SORT Operator (VA = 3)
    |   |   | Using Worktable1 for internal storage.
    |   |   |
    |   |   |   |SCAN Operator (VA = 2)
    |   |   |   |  FROM TABLE
    |   |   |   |  stores
    |   |   |   |  Table Scan.
    |   |   |   |  Forward Scan.
    |   |   |   |  Positioning at start of table.
    |   |   |   |  Using I/O Size 2 Kbytes for data pages.
    |   |   |   |  With LRU Buffer Replacement Strategy for data pages.
```

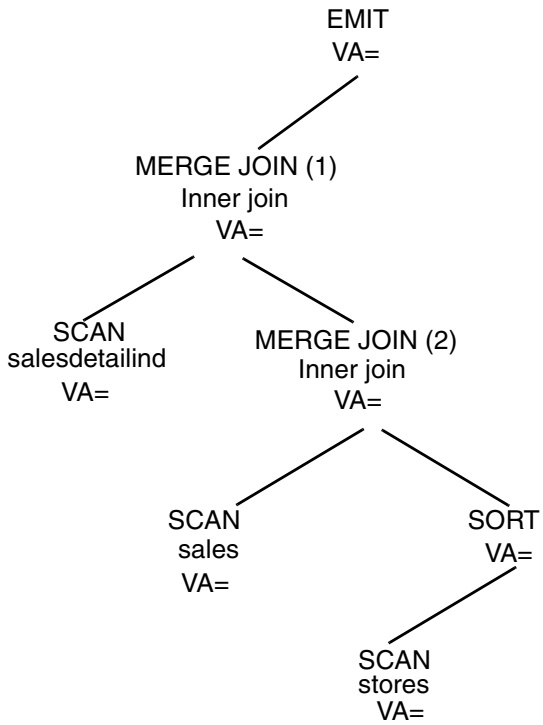
在语句级别输出后，将显示查询计划。查询计划的 showplan 输出由两部分构成：

- 运算符的名称 （某些还提供其它信息），显示在查询计划中执行哪些运算。
- 缩进的竖线 （“|”符号），显示查询计划运算符树的形状。

查询计划形状

每个运算符在树中的位置决定了它的执行顺序。执行沿着树最左边的分支开始，然后进行到右边。为了说明执行过程，本节分步介绍上一节示例中的查询计划的执行过程。图 2-1 显示该查询计划的图形表示。

图 2-1：查询计划



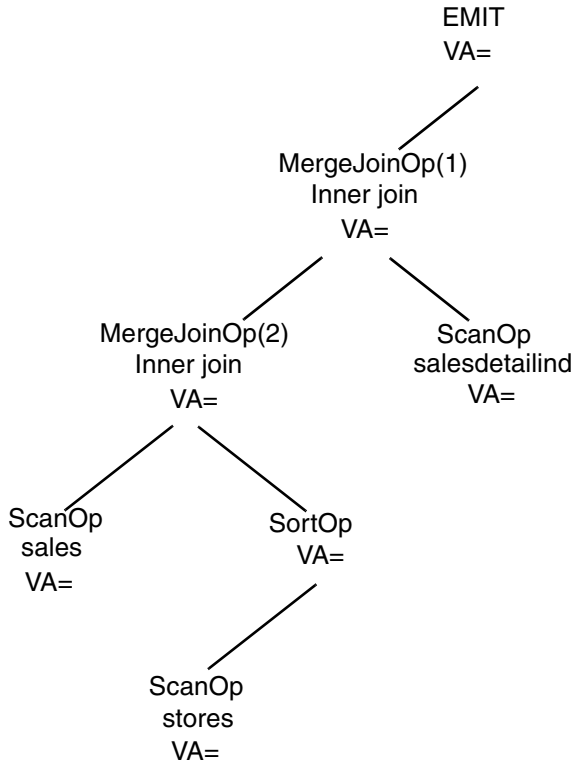
为了生成结果行，EMIT 运算符从其子运算符 MERGE JOIN (1) 中调用行，该子运算符从其左子运算符 SCAN 调用 salesdetailind 的行。当 EMIT 从其左子运算符收到行时，MERGE JOIN 运算符 (1) 会从其右子运算符 MERGE JOIN (2) 调用行。 MERGE JOIN 运算符 (2) 从其左子运算符 SCAN 调用 sales 的行。

当 MERGE JOIN 运算符 (2) 从其左子运算符收到行时，它会从其右子运算符 SCAN 调用行。 SCAN 运算符是数据阻塞运算符。也就是说，它需要其所有输入行，然后才能对输入行排序，因此 SORT 运算符始终从其子运算符 SCAN 调用 stores 的行，直到返回所有行。然后 SORT 运算符会对这些行进行排序，并将第一行传递给 MERGE JOIN 运算符 (2)。

MERGE JOIN 运算符 (2) 始终从其左子运算符或右子运算符调用行，直到它获得在连接键上相互匹配的两行。然后，匹配行向上传递到 MERGE JOIN 运算符 (1)。 MERGE JOIN 运算符 (1) 也始终从其子运算符调用行，直到找到匹配行，然后将匹配行向上传递给 EMIT 运算符以返回到客户端。实际上，已使用左深后缀递归策略对这些运算符进行了处理。

图 2-2 显示了同一个示例查询的替代查询计划的图形表示。该查询计划包含所有这些运算符，但树的形状不同。

图 2-2：替代查询计划



与图 2-2 中的查询计划相对应的 showplan 输出为：

```
QUERY PLAN FOR STATEMENT 1 (at line 1).

6 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

    |MERGE JOIN Operator (Join Type: Inner Join)
    | Using Worktable3 for internal storage.
    |  Key Count: 1
    |  Key Ordering: ASC
    |
    |   |MERGE JOIN Operator (Join Type:Inner Join)
    |   | Using Worktable2 for internal storage.
    |   |  Key Count:1
    |   |  Key Ordering:ASC
    |   |
    |   |   |SCAN Operator
    |   |   |  FROM TABLE
    |   |   |  sales
    |   |   |  Table Scan.
    |   |   |  Forward Scan.
    |   |   |  Positioning at start of table.
    |   |   |  Using I/O Size 2 Kbytes for data pages.
    |   |   |  With LRU Buffer Replacement Strategy for data pages.
    |   |
    |   |   |SORT Operator
    |   |   | Using Worktable1 for internal storage.
    |   |   |
    |   |   |   |SCAN Operator
    |   |   |   |  FROM TABLE
    |   |   |   |  stores
    |   |   |   |  Table Scan.
    |   |   |   |  Forward Scan.
    |   |   |   |  Positioning at start of table.
    |   |   |   |  Using I/O Size 2 Kbytes for data pages.
    |   |   |   |  With LRU Buffer Replacement Strategy for data pages.
    |
    |   |SCAN Operator
    |   |  FROM TABLE
    |   |  salesdetail
    |   |  Index : salesdetailind
    |   |  Forward Scan.
    |   |  Positioning at index start.
```

```
|   |  Index contains all needed columns. Base table will not be read.
|   |  Using I/O Size 2 Kbytes for index leaf pages.
|   |  With LRU Buffer Replacement Strategy for index leaf pages.
```

showplan 输出表达查询计划形状的方法是，使用缩进和竖线 （“|”）符号来表示哪些运算符在哪些运算符的下面，以及哪些运算符在树的同一分支或不同分支上。解释树形状的规则有两个：

- 第一个规则是，管状 “|” 符号构成的竖线从运算符的名称开始，向下继续，经过同一分支上在其下的所有运算符。
- 子运算符会缩进到每一嵌套层的左侧。

使用这些规则，可以从前面的 showplan 输出派生出图 2-2 中查询计划的形状，步骤如下：

1 ROOT 或 EMIT 运算符位于查询计划树的顶部。

2 MERGE JOIN 运算符 (1) 是 ROOT 的左子运算符。竖线从 MERGE JOIN 运算符 (1) 开始，向下经过整个输出的长度，因此所有其它运算符都在 MERGE JOIN 运算符 (1) 之下，并且在同一分支上。

3 MERGE JOIN 运算符 (1) 的左子运算符是 MERGE JOIN 运算符 (2)。

4 竖线从 MERGE JOIN 运算符 (2) 开始，向下经过 SCAN、SORT 和另一个 SCAN 运算符，然后结束。这些运算符均嵌套为 MERGE JOIN 运算符 (2) 下的子分支。

5 MERGE JOIN 运算符 (2) 下的第一个 SCAN 是其左子运算符，即 sales 表的 SCAN。

6 SORT 运算符是 MERGE JOIN 运算符 (2) 的右子运算符， stores 表的 SCAN 是 SORT 运算符的唯一子运算符。

7 在 stores 表的 SCAN 输出下面，有几个终止的竖线。这表明树的一个分支已结束。

8 下一个输出是针对 salesdetail 表的 SCAN。它与 MERGE JOIN 运算符 (2) 的缩进相同，表示它们在同一级别上。事实上，该 SCAN 是 MERGE JOIN 运算符 (1) 的右子运算符。

注释 大多数运算符不是一元运算符，就是二元运算符。也就是说，它们的正下方不是有一个子运算符，就是有两个子运算符。有两个以上子运算符的运算符称为 “n 元运算符”。没有子运算符的运算符是树中的叶运算符，称为 “零元运算符”。

获取查询计划的图形表示的另一种方法是使用命令 set statistics plancost on。有关详细信息，请参见 《Adaptive Server 参考手册：命令》。此命令用于比较查询计划的估计开销和实际开销。它将其输出显示为表示查询计划树的半图形化的树。这是一种诊断查询性能问题的非常有用的工具。

查询计划运算符

第 20 页的表 1-3 列出了查询计划运算符及每个运算符的说明。本节包含可提供有关各个运算符的详细信息的其它消息。

EMIT 运算符

EMIT 运算符显示在每个查询计划的顶部。 EMIT 是查询计划树的根，且始终只有一个子运算符。 EMIT 运算符可路由查询的结果行，方法是将结果行发送到客户端 （一个应用程序或其它 Adaptive Server 实例），或者将结果行的值赋给局部变量或 fetch into 变量。

SCAN 运算符

SCAN 运算符将行读入查询计划，并使它们可由查询计划中的其它运算符进行进一步的处理。 SCAN 运算符是叶运算符；也就是说，它没有任何子运算符。 SCAN 运算符可以从多个源读取行，因此标识它的 showplan 消息的后面始终是 FROM 消息，以标识正在执行哪种 SCAN。 FROM 消息为： FROM CACHE、 FROM OR、 FROM LIST 和 FROM TABLE。

FROM 高速缓存消息

该消息显示 CACHE SCAN 运算符正在读取单行的内存中的表。

FROM 或 LIST

OR 列表中的行多达 *N* 行；每一行对应查询中指定的不同的 OR 或 IN 值。

第一条消息显示 OR 扫描正在从包含 IN 列表或同一列中多个 or 子句的值的内存中的表读取行。 OR 列表只出现在对 in 列表使用特定 or 策略的查询计划中。第二条消息显示内存中的表可以具有的最大行数 (*N*)。因为 OR 列表在填充内存中的表时消除了重复的值，所以 *N* 可能小于 SQL 语句中出现的值的数量。例如，以下查询生成了使用特定 or 策略和 OR 列表的查询计划：

```
select s.id from sysobjects s where s.id in (1, 0, 1, 2, 3)
go

QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
    The type of query is SELECT.

    4 operator(s) under root

    ROOT:EMIT Operator (VA = 4)

    |NESTED LOOP JOIN Operator (VA = 3) (Join Type: Inner Join)
    |
    |   |SCAN Operator (VA = 2)
    |   | FROM OR List
    |   | OR List has up to 5 rows of OR/IN values.
    |
    |   |RESTRICT Operator (VA = 2)(0)(0)(0)(8)(0)
    |   |   |SCAN Operator (VA = 1)
    |   |   | FROM TABLE
    |   |   | sysobjects
    |   |   | s
    |   |   | Using Clustered Index.
    |   |   | Index : csysobjects
    |   |   | Forward Scan.
    |   |   | Positioning by key.
    |   |   | Index contains all needed columns. Base table will not be read.
    |   |   | Keys are:
    |   |   | id ASC
    |   |   | Using I/O Size 2 Kbytes for index leaf pages.
    |   |   | With LRU Buffer Replacement Strategy for index leaf pages.
```

该示例中的 IN 列表中有 5 个值，但只有 4 个值不相同，所以 OR 列表只将这 4 个不同的值放入其内存中的表。在查询计划示例中， OR 列表是 NESTED LOOP JOIN 运算符的左子运算符，SCAN 运算符是 NESTED LOOP JOIN 运算符的右子运算符。在执行此计划时，NESTED LOOP JOIN 运算符调用 or 命令以从其内存中的表返回行，然后 NESTED LOOP JOIN 运算符调用 SCAN 运算符以查找所有匹配的行 （一次一行，使用聚簇索引查找）。相对于读取 sysobjects 的所有行，再将各行中 sysobjects.id 的值与 IN 列表中的 5 个值作比较，该查询计划示例高效得多。

FROM TABLE

FROM TABLE 显示 PARTITION SCAN 运算符正在读取数据库表。第二条消息提供表名；如果有相关名，则在下一行输出。在前面的输出示例中的 FROM TABLE 消息下，sysobjects 是表名，s 是相关名。前面的示例在 FROM TABLE 消息下面还显示了其它几条消息。这些消息提供了有关 PARTITION SCAN 运算符如何指引 Adaptive Server 的访问层从正被扫描的表获取行的详细信息。

以下消息指出扫描是表扫描，还是索引扫描：

- Table Scan — 通过读取表中的页来获取行。
- Using Clustered Index — 使用聚簇索引来获取表的行。
- Index: *indexname* — 使用索引来获取表的行。如果该消息的前面没有“using clustered index”，则使用非聚簇索引。*indexname* 是将使用的索引的名称。

这些消息指出表或索引的扫描方向。扫描方向取决于创建索引时指定的顺序以及在 order by 子句中为列指定的顺序，或其它可由运算符在查询计划中进一步利用的有用的顺序 （例如，合并连接策略的排序顺序）。

如果 order by 子句包含对索引键的升序或降序限定符，则可以使用反向扫描，这与 create index 子句中的情况完全相反。

```
Forward scan

Backward scan
```

扫描方向消息的后面是定位消息，介绍如何访问表或叶级索引：

- Positioning at start of table — 从表的第一行开始向前扫描表。
- Positioning at end of table — 从表的最后一行开始向后扫描表。
- Positioning by key — 索引用于将扫描定位到第一个限定行。
- Positioning at index start/positioning at index end — 这些消息类似于对应的表扫描消息，只不过这是扫描索引，不是扫描表。

如果查询的性质允许限制扫描，则以下消息介绍：

- `Scanning only the last page of the table` — 在扫描使用索引并搜索标量集合的最大值时显示此消息。如果索引位于查找其最大值的列，并且索引值按升序排列，则最大值将在最后一页上。
- `Scanning only up to the first qualifying row` — 在扫描使用索引并搜索标量集合的最小值时显示此消息。

注释 如果索引键按降序排列，则上述针对最小集合和最大集合的消息将对调。

在某些情况下，要扫描的索引包含查询中所需的表的所有列。在这种情况下，将输出以下消息：

```
Index contains all needed columns. Base table will not
be read.
```

如果索引包含查询所需的所有列，则即使索引列上没有有用的键，优化程序也可能选择 `Index Scan`，而不选择 `Table Scan`。读取索引所需的 I/O 量可能远远小于读取基表所需的 I/O 量。不需要读取基表页的索引扫描称为*覆盖索引扫描*。

如果 index scan 使用键来定位扫描，则输出以下消息：

```
Keys are:
     Key <ASC/DESC>
```

该消息显示用作键的列的名称 （各键在各自的输出行上），并显示该键的索引顺序：`ASC` 表示升序， `DESC` 表示降序。

在描述 scan 运算符使用的访问类型的消息后，将输出有关 I/O 大小和缓冲区高速缓存策略的消息。

I/O 大小消息

I/O 消息为：

```
Using I/O size N Kbtyes for data pages.

Using I/O size N Kbtyes for index leaf pages.
```

这些消息报告在查询中使用的 I/O 大小。可能的 I/O 大小是 2、4、8 和 16 千字节。

如果查询中使用的表、索引或数据库使用具有大 I/O 池的数据高速缓存，则优化程序可以选择大 I/O。它可以选择使用一个 I/O 大小来读取索引叶页，使用另一个不同的大小来读取数据页。这一选择取决于高速缓存中缓冲池的可用大小、要读取的页数、对象的高速缓存绑定、表或索引页的集群比。

这两条消息或其中一条消息会出现在 SCAN 运算符的 showplan 输出中。对于表扫描，只输出第一条消息；对于覆盖索引扫描，只输出第二条消息。对于需要访问基表的 Index Scan，输出所有这两条消息。

在每个 I/O 大小消息后，将输出高速缓存策略消息：

```
With <LRU/MRU> Buffer Replacement Strategy for data
pages.

With <LRU/MRU> Buffer Replacement Strategy for index
leaf pages.
```

在 LRU 替换策略中，最近访问的页定位在高速缓存中，以便尽可能长时间地保留。在 MRU 替换策略中，最近访问的页定位在高速缓存中，以便快速替换。

在以下查询中显示了示例 I/O 和高速缓存消息：

```
use pubs2
go
set showplan on
go
select au_fname, au_lname, au_id from authors
where au_lname = "Williams"
go

QUERY PLAN FOR STATEMENT 1 (at line 1).

1 operator(s) under root


The type of query is SELECT.

ROOT:EMIT Operator (VA = 1)

    |SCAN Operator (VA = 0)
    |  FROM TABLE
    |  authors
    |  Index : aunmind
    |  Forward Scan.
    |  Positioning by key.
    |  Keys are:
    |    au_lname ASC
```

```
    |  Using I/O Size 2 Kbytes for index leaf pages.
    |  With LRU Buffer Replacement Strategy for index leaf pages.
    |  Using I/O Size 2 Kbytes for data pages.
    |  With LRU Buffer Replacement Strategy for data pages.
```

authors 表的 SCAN 运算符使用索引 aunmind，但还必须读取基表页来从 authors 中获取所有必需列。在该示例中，有两条 I/O 大小消息，每条消息的后面都跟有相应的缓冲区替换消息。

有两种表 SCAN 运算符，它们有其各自的消息 — RID SCAN 和 LOG SCAN。

RID scan

Positioning by Row IDentifier (RID) 扫描只位于使用优化程序可以选择的第二种 or 策略 （常规 or 策略）的查询计划中。当不同的列上存在多个 or 子句时，可以使用常规 or 策略。优化程序可为之选择常规 or 策略及其 showplan 输出的查询的示例为：

```
use pubs2
go
set showplan on
go
select id from sysobjects where id = 4 or name = 'foo'

QUERY PLAN FOR STATEMENT 1 (at line 1).

6 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator (VA = 6)

    |RID JOIN Operator (VA = 5)
    | Using Worktable2 for internal storage.
    |
    |   |HASH UNION Operator has 2 children.
    |   | Using Worktable1 for internal storage.
    |   |
    |   |   |SCAN Operator (VA = 0)
    |   |   |  FROM TABLE
    |   |   |  sysobjects
    |   |   |  Using Clustered Index.
    |   |   |  Index : csysobjects
    |   |   |  Forward Scan.
    |   |   |  Positioning by key.
    |   |   |  Index contains all needed columns. Base table will not be read.
```

```
|    |    |  Keys are:
|    |    |    id ASC
|    |    |  Using I/O Size 2 Kbytes for index leaf pages.
|    |    |  With LRU Buffer Replacement Strategy for index leaf pages.
|    |
|    |    |SCAN Operator (VA = 1)
|    |    |  FROM TABLE
|    |    |  sysobjects
|    |    |  Index : ncsysobjects
|    |    |  Forward Scan.
|    |    |  Positioning by key.
|    |    |  Index contains all needed columns. Base table will not be read.
|    |    |  Keys are:
|    |    |    name ASC
|    |    |  Using I/O Size 2 Kbytes for index leaf pages.
|    |    |  With LRU Buffer Replacement Strategy for index leaf pages.
|
|    |RESTRICT Operator (VA = 4)(0)(0)(0)(11)(0)
|    |
|    |    |SCAN Operator(VA = 3)
|    |    |  FROM TABLE
|    |    |  sysobjects
|    |    |  Using Dynamic Index.
|    |    |  Forward Scan.
|    |    |  Positioning by Row IDentifier (RID).
|    |    |  Using I/O Size 2 Kbytes for data pages.
|    |    |  With LRU Buffer Replacement Strategy for data pages.
```

在该示例中，where 子句包含两个分离，分别位于不同的列（id 和 name）上。其中每个列上都有索引（csysobjects 和 ncsysobjects），因此优化程序选择符合以下条件的查询计划：使用一个索引扫描来查找其 id 列为 4 的所有行，使用另一个索引扫描来查找其 name 为 “foo”的所有行。

因为单个行可能同时包含 ID 4 和名称 “foo”，所以该行将在结果集中出现两次。为了消除这些重复的行，索引扫描只返回限定行的行标识符 (RID)。两个 RID 流由 HASH UNION 运算符连接，这也会删除任何重复的 RID。

具有唯一 RID 的流被传递到 RID JOIN 运算符。 rid join 运算符创建一个工作表，并使用包含各个 RID 的单列行填充它。然后 RID JOIN 运算符将其 RID 工作表传递给 RID SCAN 运算符。 RID SCAN 运算符将该工作表传递到访问层，在那里，该工作表被视为无键的非聚簇索引，然后获取并返回与 RID 对应的行。

showplan 输出的最后一个 SCAN 是 RID SCAN。从输出示例中可以看到，RID SCAN 输出包含许多前面已讨论过的消息，但还包含两条只为 RID SCAN 输出的消息：

- Using Dynamic Index — 表示 SCAN 使用包含 RID 的工作表，该工作表是在执行过程中由 RID JOIN 运算符构建的，作为查找匹配行的索引。
- Positioning by Row Identifier (RID) — 表示直接通过 RID 查找行。

Log Scan

Log Scan 只出现在访问已插入或已删除的表的触发器中。这些表是在执行触发器时通过扫描事务日志动态建立的。仅在 insert、delete 或 update 查询修改了在其上为特定查询类型定义触发器的表之后，才执行触发器。以下示例是对 titles 表的 delete 查询，该表上定义了称为 deltitle 的删除触发器：

```
use pubs2
go
set showplan on
go
delete from titles where title_id = 'xxxx'

QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
   The type of query is DELETE.

   2 operator(s) under root

    |ROOT:EMIT Operator (VA = 2)

    |DELETE Operator (VA = 1)
    |  The update mode is direct.
    |
    |   |SCAN Operator (VA = 0)
    |   |  FROM TABLE
    |   |  titles
    |   |  Using Clustered Index.
    |   |  Index : titleidind
    |   |  Forward Scan.
    |   |  Positioning by key.
    |   |  Keys are:
    |   |    title_id ASC
```

```
    |   |  Using I/O Size 2 Kbytes for data pages.
    |   |  With LRU Buffer Replacement Strategy for data pages.
    |
    |  TO TABLE
    |  titles
    |  Using I/O Size 2 Kbytes for data pages.
```

此点以上的 showplan 输出针对实际的 delete 查询。以下的输出针对触发器 deltitle。

```
QUERY PLAN FOR STATEMENT 1 (at line 5).

STEP 1

The type of query is COND.

6 operator(s) under root

ROOT:EMIT Operator (VA = 6)

    |RESTRICT Operator (VA = 5)(0)(0)(0)(5)(0)
    |
    |   |SCALAR AGGREGATE Operator (VA = 4)
    |   |  Evaluate Ungrouped COUNT AGGREGATE.
    |   |
    |   |   |MERGE JOIN Operator (Join Type: Inner Join) (VA = 3)
    |   |   | Using Worktable2 for internal storage.
    |   |   |  Key Count: 1
    |   |   |  Key Ordering: ASC
    |   |   |
    |   |   |   |SORT Operator (VA = 1)
    |   |   |   | Using Worktable1 for internal storage.
    |   |   |   |
    |   |   |   |   |SCAN Operator (VA = 0)
    |   |   |   |   |  FROM TABLE
    |   |   |   |   |  titles
    |   |   |   |   |  Log Scan.
    |   |   |   |   |  Forward Scan.
    |   |   |   |   |  Positioning at start of table.
    |   |   |   |   |  Using I/O Size 2 Kbytes for data pages.
    |   |   |   |   |  With MRU Buffer Replacement Strategy for data pages.
    |   |   |
    |   |   |   |SCAN Operator (VA = 2)
    |   |   |   |  FROM TABLE
    |   |   |   |  salesdetail
    |   |   |   |  Index : titleidind
    |   |   |   |  Forward Scan.
```

```
    |   |   |   |  Positioning at index start.
    |   |   |   |  Index contains all needed columns. Base table will not be
                   read.
    |   |   |   |  Using I/O Size 2 Kbytes for index leaf pages.
    |   |   |   |  With LRU Buffer Replacement Strategy for index leaf pages.

QUERY PLAN FOR STATEMENT 2 (at line 8).

     STEP 1
          The type of query is ROLLBACK TRANSACTION.

QUERY PLAN FOR STATEMENT 3 (at line 9).

     STEP 1
          The type of query is PRINT.

QUERY PLAN FOR STATEMENT 4 (at line 0).
    STEP 1
         The type of query is GOTO.
```

定义触发器 deltitle 的过程包括 4 个 SQL 语句。使用 sp_helptext deltitle 可显示 deltitle 的文本。 deltitle 中的第一条语句已编译进查询计划中，其它三条语句编译进了遗留查询计划中并由过程执行引擎执行，而不是由查询执行引擎执行。

titles 表的 SCAN 运算符的 showplan 输出通过输出 Log Scan 指示它正在扫描日志。

DELETE、*INSERT* 和 *UPDATE* 运算符

DELETE、 INSERT 和 UPDATE 运算符通常只有一个子运算符。不过，它们可以使用多达两个的其它子运算符，来强制实施参照完整性约束，并在对文本列执行 alter table drop 时释放文本数据。

这些运算符通过插入、删除或更新属于目标表的行来修改数据。

DML 运算符的子运算符可以是 SCAN 运算符、 JOIN 运算符或任何数据流运算符。

可使用不同的更新模式修改数据，如以下消息所指定的：

```
The Update Mode is <Update Mode>.
```

表更新模式可以是 direct、 deferred、 deferred for an index 或 deferred for a variable column。工作表的更新模式始终是立即。

在以下消息中显示数据修改的目标表：

```
TO TABLE
<表名>
```

还显示用于数据修改的 I/O 大小：

```
Using I/O Size <N> Kbytes for data pages.
```

下一个示例使用 DELETE 运算符：

```
use pubs2
go
set showplan on
go
delete from authors where postalcode = '90210'

QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
The type of query is DELETE.

2 operator(s) under root

ROOT:EMIT Operator (VA = 2)

       |DELETE Operator (VA = 1)
       |  The update mode is direct.
       |
       |   |SCAN Operator (VA = 0)
       |   |  FROM TABLE
       |   |  authors
       |   |  Table Scan.
       |   |  Forward Scan.
       |   |  Positioning at start of table.
       |   |  Using I/O Size 4 Kbytes for data pages.
       |   |  With LRU Buffer Replacement Strategy for data pages.
       |
       |  TO TABLE
       |  authors
       |  Using I/O Size 4 Kbytes for data pages.
```

TEXT DELETE 运算符

在其中 DELETE、 INSERT 和 UPDATE 运算符可以拥有多个子运算符的另一种查询计划是针对 alter table drop textcol 命令的，其中 textcol 是其数据类型为 text、image 或 unitext 的列的名称。此版本的命令将 TEXT DELETE 运算符用于其查询计划。例如：

```
use tempdb
go
create table t1 (c1 int, c2 text, c3 text)
go
set showplan on
go
alter table t1 drop c2

QUERY PLAN FOR STATEMENT 1 (at line 1).
Optimized using the Abstract Plan in the PLAN clause.

STEP 1
The type of query is ALTER TABLE.

5 operator(s) under root

ROOT:EMIT Operator (VA = 5)

       |INSERT Operator (VA = 52)
       |  The update mode is direct.
       |
       |   |RESTRICT Operator (VA = 1)(0)(0)(3)(0)(0)
       |   |
       |   |   |SCAN Operator (VA = 0)
       |   |   |  FROM TABLE
       |   |   |  t1
       |   |   |  Table Scan.
       |   |   |  Forward Scan.
       |   |   |  Positioning at start of table.
       |   |   |  Using I/O Size 2 Kbytes for data pages.
       |   |   |  With LRU Buffer Replacement Strategy for data pages.
       |
       |   |TEXT DELETE Operator
       |   |  The update mode is direct.
       |   |
       |   |   |SCAN Operator (VA = 3)
       |   |   |  FROM TABLE
       |   |   |  t1
       |   |   |  Table Scan.
       |   |   |  Forward Scan.
```

```
|    |    |  Positioning at start of table.
|    |    |  Using I/O Size 2 Kbytes for data pages.
|    |    |  With LRU Buffer Replacement Strategy for data pages.
|
|  TO TABLE
|  #syb__altab
|  Using I/O Size 2 Kbytes for data pages.
```

使用 alter table 命令删除了 t1 中两个 text 列中的一个列。showplan 输出类似于 select into 查询计划，因为 alter table 在内部生成了 select into 查询计划。

INSERT 运算符调用其左子运算符 t1 的 SCAN 来读取 t1 的行，并构建新行（但只将 c1 和 c3 列插入 #syb_altab）。当所有新行都插入 #syb_altab 后，INSERT 运算符会调用其右子运算符 TEXT DELETE 来删除已从 t1 中删除的 c2 列的文本页链。

后处理将 t1 的原始页替换为 #syb_altab 的原始页，以完成 alter table 命令。

TEXT DELETE 运算符只出现在用于删除表的一部分而非全部文本列的 alter table 命令中，并且它总是作为 INSERT 运算符的右子运算符出现。

TEXT DELETE 运算符显示更新模式消息，与 INSERT、UPDATE 和 DELETE 运算符完全相同。

强制实施参照完整性的查询计划

当 INSERT、 UPDATE 和 DELETE 运算符用于具有一个或多个参照完整性约束的表时， showplan 输出还显示 DML 运算符的 DIRECT RI FILTER 和 DEFERRED RI FILTER 子运算符。参照完整性约束的类型决定是显示所有这两个运算符，还是只显示其中一个运算符。

以下示例表明将 insert 插入 pubs3 数据库的 titles 表中。该表有一个名为 pub_id 的列，该列引用 publishers 表的 pub_id 列。titles.pub_id 的参照完整性约束要求插入 titles.pub_id 的每个值必须在 publishers.pub_id 中具有相应的值。

查询及其查询计划为：

```
use pubs3
go
set showplan on
insert into titles values ("AB1234", "Abcdefg", "test", "9999", 9.95, 1000.00,
10, null, getdate(),1)

QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
The type of query is INSERT.
```

```
4 operator(s) under root


ROOT:EMIT Operator (VA = 3)

       |INSERT Operator (VA = 2)
       |  The update mode is direct.
       |
       |    |SCAN Operator (VA = 1)
       |    |  FROM CACHE
       |
       |    |DIRECT RI FILTER Operator has 1 children.
       |    |
       |    |    |SCAN Operator (VA = 0)
       |    |    |  FROM TABLE
       |    |    |  publishers
       |    |    |  Index : publishers_6240022232
       |    |    |  Forward Scan.
       |    |    |  Positioning by key.
       |    |    |  Index contains all needed columns. Base table will not be
                    read.
       |    |    |  Keys are:
       |    |    |    pub_id ASC
       |    |    |  Using I/O Size 2 Kbytes for index leaf pages.
       |    |    |  With LRU Buffer Replacement Strategy for index leaf pages.
       |
       |  TO TABLE
       |  titles
       |  Using I/O Size 2 Kbytes for data pages.
```

在该查询计划中，INSERT 运算符的左子运算符是 CACHE SCAN，它返回要插入 titles 的值所在的行。INSERT 运算符的右子运算符是 DIRECT RI FILTER 运算符。

DIRECT RI FILTER 运算符执行对 publishers 表的扫描，以查找其 pub_id 值与要插入到 titles 中的 pub_id 值相匹配的行。如果找到匹配行，则 DIRECT RI FILTER 运算符允许继续执行 insert，但如果在 publishers 中没有找到 pub_id 的匹配值，则 DIRECT RI FILTER 运算符会中止该命令。

在该示例中，DIRECT RI FILTER 可以在向 titles 插入行时，检查已插入的每一行的参照完整性约束并对其强制实施参照完整性约束。

下一个示例显示 DIRECT RI FILTER 与 DEFERRED RI FILTER 运算符一起以不同的模式运算：

```
use pubs3
go
set showplan on
go
update publishers set pub_id = '0001'

QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
The type of query is UPDATE.

13 operator(s) under root

ROOT:EMIT Operator (VA = 13)

       |UPDATE Operator (VA = 1)
       |  The update mode is deferred_index.
       |
       |   |SCAN Operator (VA = 0)
       |   |  FROM TABLE
       |   |  publishers
       |   |  Table Scan.
       |   |  Forward Scan.
       |   |  Positioning at start of table.
       |   |  Using I/O Size 2 Kbytes for data pages.
       |   |  With LRU Buffer Replacement Strategy for data pages.
       |
       |   |DIRECT RI FILTER Operator (VA = 7) has 1 children.
       |   |
       |   |   |INSERT Operator (VA = 6)
       |   |   |  The update mode is direct.
       |   |   |
       |   |   |   |SQFILTER Operator (VA = 5) has 2 children.
       |   |   |   |
       |   |   |   |   |SCAN Operator (VA = 2)
       |   |   |   |   |  FROM CACHE
       |   |   |   |
       |   |   |   |  Run subquery 1 (at nesting level 0).
       |   |   |   |
       |   |   |   |QUERY PLAN FOR SUBQUERY 1 (at nesting level 0 and at
                    line 0).
       |   |   |   |
       |   |   |   |   Non-correlated Subquery.
       |   |   |   |   Subquery under an EXISTS predicate.
```

```
|   |   |   |
|   |   |   |   |SCALAR AGGREGATE Operator (VA = 4)
|   |   |   |   | Evaluate Ungrouped ANY AGGREGATE.
|   |   |   |   | Scanning only up to the first qualifying row.
|   |   |   |   |
|   |   |   |   |   |SCAN Operator (VA = 3)
|   |   |   |   |   |  FROM TABLE
|   |   |   |   |   |  titles
|   |   |   |   |   |  Table Scan.
|   |   |   |   |   |  Forward Scan.
|   |   |   |   |   |  Positioning at start of table.
|   |   |   |   |   |  Using I/O Size 2 Kbytes for data pages.
|   |   |   |   |   |  With LRU Buffer Replacement strategy for data
                        pages.
|   |   |   |   |
|   |   |   |   |  END OF QUERY PLAN FOR SUBQUERY 1.
|   |   |   |
|   |   |   |  TO TABLE
|   |   |   |  Worktable1.
|   |
|   |DEFERRED RI FILTER Operator has (VA = 12) 1 children.
|   |   |
|   |   |   |SQFILTER Operator (VA = 11) has 2 children.
|   |   |   |
|   |   |   |   |SCAN Operator (VA = 8)
|   |   |   |   |  FROM TABLE
|   |   |   |   |  Worktable1.
|   |   |   |   |  Table Scan.
|   |   |   |   |  Forward Scan.
|   |   |   |   |  Positioning at start of table.
|   |   |   |   |  Using I/O Size 2 Kbytes for data pages.
|   |   |   |   |  With LRU Buffer Replacement Strategy for data pages.
|   |   |   |
|   |   |   |  Run subquery 1 (at nesting level 0).
|   |   |   |
|   |   |   |  QUERY PLAN FOR SUBQUERY 1 (at nesting level 0 and at line 0).
|   |   |   |
|   |   |   |   Non-correlated Subquery.
|   |   |   |   Subquery under an EXISTS predicate.
|   |   |   |
|   |   |   |   |SCALAR AGGREGATE Operator (VA = 10)
|   |   |   |   |  Evaluate Ungrouped ANY AGGREGATE.
|   |   |   |   |  Scanning only up to the first qualifying row.
|   |   |   |   |
|   |   |   |   |   |SCAN Operator (VA = 9)
|   |   |   |   |   |  FROM TABLE
```

```
|    |    |    |    |  publishers
|    |    |    |    |  Index : publishers_6240022232
|    |    |    |    |  Forward Scan.
|    |    |    |    |  Positioning by key.
|    |    |    |    |  Index contains all needed columns. Base table will
                        not be read.
|    |    |    |    |  Keys are:
|    |    |    |    |    pub_id ASC
|    |    |    |    |  Using I/O Size 2 Kbytes for index leaf pages.
|    |    |    |    |  With LRU Buffer Replacement Strategy for index leaf
                        pages.
|    |    |
|    |    |  END OF QUERY PLAN FOR SUBQUERY 1.|
|  TO TABLE
|  publishers
|  Using I/O Size 2 Kbytes for data pages.
```

titles 的参照完整性约束要求，对于 titles.pub_id 的每个值，都必须存在 publishers.pub_id 的一个值。不过，该示例查询更改了 publisher.pub_id 的值，因此必须执行检查以保持参照完整性约束。

该示例查询可以更改 publishers 中若干行的 publishers.pub_id 的值，因此在处理完 publishers 的所有行之前，无法执行确保 titles.pub_id 的所有值仍位于 publisher.pub_id 中的检查。

该示例调用延迟参照完整性检查：在读取 publishers 的每一行时，UPDATE 运算符会调用 DIRECT RI FILTER 运算符以在 titles 中搜索 pub_id 的值与要更改的值相同的行。如果找到一行，则表明 pub_id 的此值必须仍存在于 publishers 中以保持 titles 的参照完整性约束，因此 pub_id 的值被插入 WorkTable1 中。

在更新了 publishers 的所有行后，UPDATE 运算符会调用 DEFERRED RI FILTER 运算符来执行其子查询，以验证 Worktable1 中的所有值是否仍位于 publishers 中。DEFERRED RI FILTER 的左子运算符是一个从 Worktable1 读取行的 SCAN。右子运算符是一个执行存在子查询以在 publishers 中查找匹配值的 SQFILTER 运算符。如果没有找到匹配值，则该命令中止。

本节中的示例只在两个表之间使用简单的参照完整性约束。Adaptive Server 最多允许每个表有 192 种约束，因此可以生成更复杂的查询计划。当必须强制实施多个约束时，查询计划中仍只有一个 DIRECT RI FILTER 或 DEFERRED RI FILTER 运算符，但这些运算符可以有多个子计划，每个必须强制实施的约束一个子计划。

JOIN 运算符

Adaptive Server 提供了四种主要的 JOIN 运算符策略：NESTED LOOP JOIN、MERGE JOIN、HASH JOIN 和 NARY NESTED LOOP JOIN，后者是 NESTED LOOP JOIN 的变体。在 15.0 以前的版本中， NESTED LOOP JOIN 是主要 JOIN 策略。此外还提供 MERGE JOIN，但缺省情况下不启用该策略。

下面将对每个 JOIN 运算符做进一步的介绍，包括各种算法的一般说明。这些说明大致概括了各种 JOIN 策略所需的处理。

NESTED LOOP JOIN

NESTED LOOP JOIN（最简单的连接策略）是二元运算符，其左子运算符构成外部数据流，其右子运算符构成内部数据流。

对来自外部数据流的每一行，打开内部数据流。通常右子运算符是 scan 运算符。打开内部数据流能够有效地将扫描定位在限定所有可搜索参数的第一行。

限定行返回到 NESTED LOOP JOIN 的父运算符。继续调用连接运算符以从内部流返回限定行。

在为当前外部行返回内部流的最后一个限定行后，内部流关闭。执行调用以从外部流获取下一个限定行。该行中的值提供了用于打开扫描并将扫描定位到内部流的可搜索参数。此过程一直继续，直到 NESTED LOOP JOIN 的左子运算符返回 End Of Scan。

```
 -- Collect all of the title ids for books written by "Bloom".
select ta.title_id
from titleauthor ta, authors a
where a.au_id = ta.au_id
and au_lname = "Bloom"
go

QUERY PLAN FOR STATEMENT 1 (at line 2).

STEP 1
The type of query is SELECT.

3 operator(s) under root

ROOT:EMIT Operator (VA = 3)

       |NESTED LOOP JOIN Operator (Join Type: Inner Join)
       |
       |   |SCAN Operator (VA = 0)
```

```
|    |  FROM TABLE
|    |  authors
|    |  a
|    |  Index : aunmind
|    |  Forward Scan.
|    |  Positioning by key.
|    |  Keys are:
|    |    au_lname ASC
|    |  Using I/O Size 2 Kbytes for index leaf pages.
|    |  With LRU Buffer Replacement Strategy for index leaf pages.
|    |  Using I/O Size 2 Kbytes for data pages.
|    |  With LRU Buffer Replacement Strategy for data pages.
|
|    |SCAN Operator (VA = 1)
|    |  FROM TABLE
|    |  titleauthor
|    |  ta
|    |  Using Clustered Index.
|    |  Index : taind
|    |  Forward Scan.
|    |  Positioning by key.
|    |  Keys are:
|    |    au_id ASC
|    |  Using I/O Size 2 Kbytes for data pages.
|    |  With LRU Buffer Replacement Strategy for data pages.
```

authors 表与 titleauthor 表连接。已选择 NESTED LOOP JOIN 策略。NESTED LOOP JOIN 运算符的类型为“Inner Join”。首先，打开 authors 表，定位到 l_name 值为“Bloom”的第一行（使用 aunmind 索引）。然后，打开 titleauthor 表，使用聚簇索引“taind”定位到第一行，其 au_id 等于当前 authors 的行的 au_id 值。如果没有对查找内部数据流有用的索引，则优化程序可能生成重新格式化策略。

通常，如果有可用于限定内部数据流上连接谓词的有用索引，则 NESTED LOOP JOIN 策略有效。

MERGE JOIN

MERGE JOIN 运算符是二元运算符。左子运算符和右子运算符分别是外部数据流和内部数据流。这两个数据流均必须按 MERGE JOIN 的键值进行排序。

首先，从外部数据流获取行。这样可以初始化 MERGE JOIN 的连接键值。然后，从内部数据流获取行，直到遇到其键值等于（如果键列按降序排序，则为小于）或大于初始化键值的行。如果连接键相符，则传递该限定行以进行其它处理，然后继续调用 MERGE JOIN 运算符以从当前活动数据流中获取行。

如果新值大于当前的比较键，则这些值将在从其它数据流获取行时用作新的比较连接键。此过程一直继续，直到用尽了所有数据流。

通常，如果数据流扫描要求必须处理大多数行，并且任何输入数据流已按连接键排序（如果这些数据流很大），则 MERGE JOIN 策略有效。

```
select ta.title_id
from titleauthor ta, authors a
where a.au_id = ta.au_id
and au_lname = "Bloom"
go

QUERY PLAN FOR STATEMENT 1 (at line 2).

    STEP 1
        The type of query is EXECUTE.
        Executing a newly cached statement.

QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
The type of query is SELECT.

3 operator(s) under root

ROOT:EMIT Operator (VA = 3)

       |MERGE JOIN Operator (Join Type: Inner Join)
       | Using Worktable2 for internal storage.
       |  Key Count: 1
       |  Key Ordering: ASC
       |
       |   |SORT Operator
       |   | Using Worktable1 for internal storage.
       |   |
       |   |   |SCAN Operator
       |   |   |  FROM TABLE
       |   |   |  authors
       |   |   |  a
       |   |   |  Index : aunmind
       |   |   |  Forward Scan.
```

```
|    |    |  Positioning by key.
|    |    |  Keys are:
|    |    |    au_lname ASC
|    |    |  Using I/O Size 2 Kbytes for index leaf pages.
|    |    |  With LRU Buffer Replacement Strategy for index leaf pages.
|    |    |  Using I/O Size 2 Kbytes for data pages.
|    |    |  With LRU Buffer Replacement Strategy for data pages.
|
|    |SCAN Operator
|    |  FROM TABLE
|    |  titleauthor
|    |  ta
|    |  Index : auidind
|    |  Forward Scan.
|    |  Positioning at index start.
|    |  Using I/O Size 2 Kbytes for index leaf pages.
|    |  With LRU Buffer Replacement Strategy for index leaf pages.
|    |  Using I/O Size 2 Kbytes for data pages.
|    |  With LRU Buffer Replacement Strategy for data pages.
```

在上述示例中， sort 运算符是左子运算符或外部数据流。 sort 运算符的数据源是 authors 表。sort 运算符是必需的，因为 authors 表在 au_id 上没有索引，否则可以提供必需的排序顺序。 titleauthor 表的扫描是右子运算符 / 内部数据流。该扫描使用可为 MERGE JOIN 策略提供必需顺序的 auidind 索引。

从外部数据流 （authors 表是初始数据源）获取行以建立初始连接键比较值。然后从 titleauthor 表获取行，直到找到其连接键等于或大于比较键的行。

具有匹配键的内部数据流行存储在高速缓存中，以便需要重新获取它们。当外部数据流包含重复键时重新获取这些行。在获取了大于当前连接键比较值的 titleauthor.au_id 值后，MERGE JOIN 运算符将开始从外部数据流获取行，直至找到等于或大于当前 titleauthor.au_id 值的连接键值。此时，需重新开始扫描内部数据流。

MERGE JOIN 运算符的 showplan 输出中包含一条消息，指出将使用哪个工作表作为内部数据流的后备存储。如果高速缓存的内存中不再能够容纳具有重复连接键的内部行，则将其写入工作表中。高速缓存的行的宽度限于 64 千字节。

HASH JOIN

HASH JOIN 运算符是二元运算符。其左子运算符生成建立输入数据流。其右子运算符生成探查输入数据流。在从 HASH JOIN 运算符请求第一行时，通过完全读取建立输入数据流来生成建立集。从输入数据流读取每一行并使用散列键将其散列到适当的桶中。

如果没有足够的内存来容纳整个建立集，会将溢出的部分存放到磁盘上。这一部分称为*散列分区*，不应与表分区混淆。散列分区由散列桶集合构成。在读取整个左子运算符的数据流后，读取探查输入。

散列探查集的每一行。在相应的建立桶中进行查找以检查是否存在具有匹配散列键的行。如果建立集的桶位于内存中，则会发生这种情况。如果探查行溢出，则会将其写入相应的溢出探查分区。当探查行的键与建立行的键相匹配时，则会向上传递这两行的列的必要投影以进行其它处理。

在 HASH JOIN 算法的后续递归传递中处理溢出分区。在每次传递中都使用新的散列种子，以便在不同的散列桶间重新分配数据。此递归处理将一直继续，直到最后一个溢出分区完全位于内存之中。当建立集的散列分区包含许多重复项时， HASH JOIN 运算符会重新执行 NESTED LOOP JOIN 处理。

通常，如果必须处理源集中的大多数行，并且连接键上没有固有的有用排序，或者没有可升级为调用运算符的有用排序 （如连接键上的 order by 子句），则 HASH JOIN 策略非常适用。如果其中一个数据集非常小，可以完全位于内存中，则 HASH JOIN 的执行效果尤其好。在这种情况下，不会发生任何溢出，并且执行该 HASH JOIN 算法不需要任何 I/O。

```
select ta.title_id
from titleauthor ta, authors a
where a.au_id = ta.au_id
and au_lname = "Bloom"

QUERY PLAN FOR STATEMENT 1 (at line 2).

3 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

       |HASH JOIN Operator (Join Type: Inner Join)
       | Using Worktable1 for internal storage.
       |
       |   |SCAN Operator
       |   |  FROM TABLE
       |   |  authors
```

```
|    |  a
|    |  Index : aunmind
|    |  Forward Scan.
|    |  Positioning by key.
|    |  Keys are:
|    |    au_lname ASC
|    |  Using I/O Size 2 Kbytes for index leaf pages.
|    |  With LRU Buffer Replacement Strategy for index leaf pages.
|    |  Using I/O Size 2 Kbytes for data pages.
|    |  With LRU Buffer Replacement Strategy for data pages.
|
|    |SCAN Operator
|    |  FROM TABLE
|    |  titleauthor
|    |  ta
|    |  Index : auidind
|    |  Forward Scan.
|    |  Positioning at index start.
|    |  Using I/O Size 2 Kbytes for index leaf pages.
|    |  With LRU Buffer Replacement Strategy for index leaf pages.
|    |  Using I/O Size 2 Kbytes for data pages.
|    |  With LRU Buffer Replacement Strategy for data pages.
```

在上述示例中，建立输入数据流的源是 author.aunmind 的索引扫描。

该扫描只返回 au_lname 值为 “Bloom”的行。然后按这些行的 au_id 值散列这些行，并将其放入相应的散列桶中。在初始建立阶段完成后，打开并扫描探查流。按 au_id 列散列源索引 titleauthor.auidind 的每一行。产生的散列值用于确定应在建立集的哪个桶中搜索匹配的散列键。建立集的散列桶中的每一行与探查行的散列键比较是否相等。如果行匹配，则 titleauthor.au_id 列返回到 EMIT 运算符。

HASH JOIN 运算符的 showplan 输出包含一条消息，指出将使用哪个工作表作为溢出分区的后备存储。输入行的宽度限于 64 千字节。

NARY NESTED LOOP JOIN 运算符

优化程序从不评估或选择 NARY NESTED LOOP JOIN 策略。它是在代码生成过程中构造的运算符。如果编译器找到包含两个或更多个左深 NESTED LOOP JOIN 的系列，则尝试将它们转换为 NARY NESTED LOOP JOIN 运算符。如果满足其它两个要求，还可以转换扫描；每个 NESTED LOOP JOIN 运算符的类型为 “inner join”，每个 NESTED LOOP JOIN 的右子运算符为 SCAN 运算符。允许 RESTRICT 运算符位于 SCAN 运算符之上。

与执行一系列 NESTED LOOP JOIN 运算符相比，执行 NARY NESTED LOOP JOIN 在性能方面具有优势。以下示例阐明两种执行方法之间的根本区别。

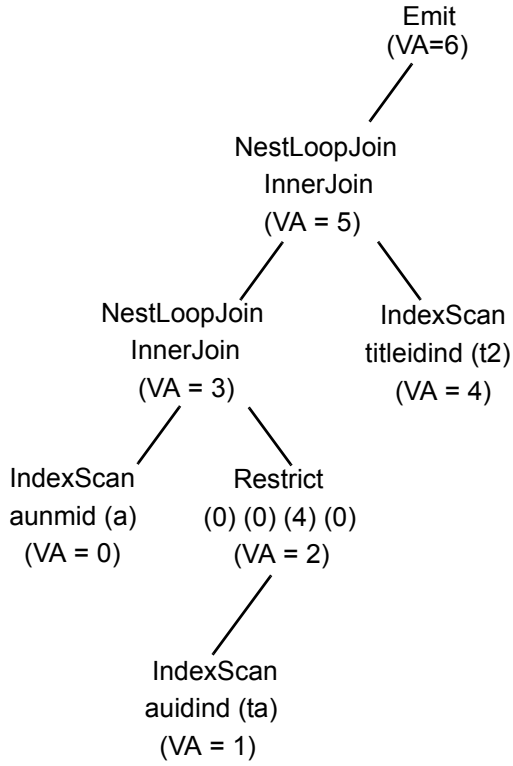
使用一系列 NESTED LOOP JOIN，扫描可以根据前面的扫描初始化的可搜索参数的值消除行。该扫描可能不是失败的扫描前面紧邻的扫描。使用一系列 NESTED LOOP JOIN，将完全读取前面的扫描，尽管它对失败的扫描没有任何影响。这可能导致产生大量不需要的 I/O。而使用 NARY NESTED LOOP JOIN，获取的下一行来自产生失败的可搜索参数值的扫描，这会大大提高效率。

```
select a.au_id, au_fname, au_lname
from titles t, titleauthor ta, authors a
where a.au_id = ta.au_id
and ta.title_id = t.title_id
and a.au_id = t.title_id
and au_lname = "Bloom"
QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
The type of query is SELECT.

4 operator(s) under root

   |ROOT:EMIT Operator (VA = 4)
   |
   |    |N-ARY NESTED LOOP JOIN Operator (VA = 3) has 3 children.
   |    |
   |    |   | SCAN Operator (VA = 0)
   |    |   | FROM TABLE
   |    |   | authors
   |    |   | a
   |    |   | Table Scan.
   |    |   | Forward Scan.
   |    |   | Positioning at start of table.
   |    |   | Using I/O Size 2 Kbytes for data pages.
   |    |   | With LRU Buffer Replacement Strategy for data pages.
   |
   |    |   |SCAN Operator (VA = 1)
   |    |   | FROM TABLE
   |    |   | titleauthor
   |    |   | ta
   |    |   | Table Scan.
   |    |   | Forward Scan.
   |    |   | Positioning at start of table.
   |    |   | Using I/O Size 2 Kbytes for data pages.
   |    |   | With LRU Buffer Replacement Strategy for data pages.
   |    |
   |    |   | SCAN Operator (VA = 2)
   |    |   | FROM TABLE
```

```
|     |    | titles
|     |    | t
|     |    | Index : titles_6720023942
|     |    | Forward Scan.
|     |    | Positioning by key.
|     |    | Index contains all needed columns. Base table will not be read.
|     |    | Keys are:
|     |    | title_id ASC
|     |    | Using I/O Size 2 Kbytes for index leaf pages.
|     |    | With LRU Buffer Replacement Strategy for index leaf pages.
```

图 2-3 描述了一系列 NESTED LOOP JOIN。

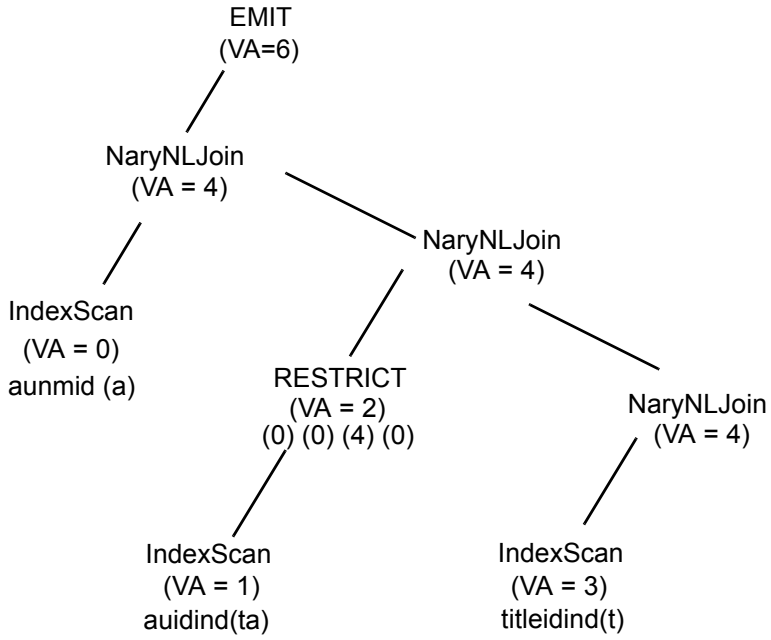
图 2-3：包含 Nested loop join 的 Emit 运算符树



所有查询处理器运算符都被赋予了虚拟地址。图 2-3 中包含 VA = 的行报告给定运算符的虚拟地址。

有效的连接顺序为 authors、 titleauthor、 titles。 RESTRICT 运算符是 titleauthors 的扫描的父运算符。下面将该计划转换为 NARY NESTED LOOP JOIN 计划：

图 2-4：NARY NESTED LOOP JOIN 运算符



转换保持 authors、 titleauthor 和 titles 的原始连接顺序。在上述示例中，titles 的扫描有两个可搜索的参数 — ta.title_id = t.title_id 和 a.au_id = t.title_id。因此，对 titles 的扫描可能因由 titleauthor 的扫描建立的可搜索参数值而失败，也可能因由 authors 的扫描建立的可搜索参数值而失败。如果因 authors 的扫描设置的可搜索参数值而没有从 titles 的扫描返回任何行，则继续扫描 titleauthor 没有任何意义。对于从 titleauthor 获取的每一行，titles 的扫描将失败。只有在从 authors 获取新行时，对 titles 的扫描才可能成功。这是已实现 NARY NESTED LOOP JOIN 的原因；对于对连续扫描返回的行没有作用的表，它们不再进行无用的读取。

在该示例中， NARY NESTED LOOP JOIN 运算符关闭 titleauthor 的扫描，从 authors 获取新行，根据从 authors 中获取的 au_id 重新定位 titleauthor 的扫描。这又是一个显著的性能改进，因为它不再对 titleauthor 表进行无用的读取，并清除了可能发生的相关 I/O。

半连接

半连接是 NESTED LOOP JOIN 运算符的变体，其结果集中包括 NESTED LOOP JOIN 运算符。当在两个表之间进行半连接时， Adaptive Server 会从包含第二个表中一个或多个匹配的第一个表返回行 （常规连接从第一个表只返回一次匹配行）。也就是说，半连接会在找到第一个匹配值时返回行，然后停止处理，而不是扫描一个表以返回所有匹配值。半连接也称为 “存在连接”。

例如，如果对 titles 和 titleauthor 表执行半连接：

```
select title
    from titles
where title_id in (select title_id from titleauthor)
and title like "A Tutorial%"
QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
    The type of query is SELECT.

    4 operator(s) under root

|ROOT:EMIT Operator (VA = 4)
   |
   |   |NESTED LOOP JOIN Operator (VA = 3) (Join Type: Left Semi Join)
   |   |
   |   |   |RESTRICT Operator (VA = 1)(0)(0)(0)(6)(0)
   |   |   |
   |   |   |   |SCAN Operator (VA = 0)
   |   |   |   | FROM TABLE
   |   |   |   | titles
   |   |   |   | Index : titleind
   |   |   |   | Forward Scan.
   |   |   |   | Positioning by key.
   |   |   |   | Keys are:
   |   |   |   | title ASC
   |   |   |   | Using I/O Size 2 Kbytes for index leaf pages.
   |   |   |   | With LRU Buffer Replacement Strategy for index leaf pages.
   |   |   |   | Using I/O Size 2 Kbytes for data pages.
   |   |   |   | With LRU Buffer Replacement Strategy for data pages.
   |   |
   |   |   |SCAN Operator (VA = 2)
   |   |   | FROM TABLE
   |   |   | titleauthor
   |   |   | Index : titleidind
   |   |   | Forward Scan.
   |   |   | Positioning by key.
```

```
|   |   | Index contains all needed columns. Base table will not be read.
|   |   | Keys are:
|   |   | title_id ASC
|   |   | Using I/O Size 2 Kbytes for index leaf pages.
|   |   | With LRU Buffer Replacement Strategy for index leaf pages.
```

Distinct 运算符

有三种可用于执行差别运算的一元运算符：GROUP SORTED Distinct、SORT Distinct 和 HASH Distinct。且各有优缺点。优化程序选择在整个查询计划的上下文中使用相对高效的 distinct 运算符。

有关所有查询处理器运算符的列表和说明，请参见第 20 页的表 1-3。

GROUP SORTED Distinct 运算符

您可以使用 GROUP SORTED Distinct 运算符来执行差别运算。GROUP SORTED Distinct 要求输入数据流按不同的列进行排序。它从其子运算符读取行，并初始化要过滤的当前不同列中的值。

该行返回到父运算符。当再次调用 GROUP SORTED 运算符以获取另一行时，它从其子运算符获取另一行并将值与当前高速缓存的值作比较。如果是重复值，则放弃该行，然后再次调用子运算符以获取新行。

这个过程将一直继续，直到找到新的不同行。高速缓存该行的不同列中的值，以便稍后用于消除相同的行。当前行返回到父运算符以做进一步处理。

GROUP SORTED Distinct 运算符返回已排序的数据流。优化程序可以使用它返回已排序的不同数据流这个属性来提高其它上游处理能力。GROUP SORTED Distinct 运算符是非阻塞运算符。它在获取了不同行后立即将该行返回给它的父运算符。它不需要处理完整个输入数据流即可开始返回行。下列查询收集不同的作者的姓和名：

```
select distinct au_lname, au_fname
from authors
where au_lname = "Bloom"

QUERY PLAN FOR STATEMENT 1 (at line 2).

STEP 1
The type of query is SELECT.

2 operator(s) under root

ROOT:EMIT Operator (VA = 2)
```

```
        |GROUP SORTED Operator (VA = 1)
        |Distinct
        |
        |    |SCAN Operator (VA = 0)
        |    |  FROM TABLE
        |    |  authors
        |    |  Index : aunmind
        |    |  Forward Scan.
        |    |  Positioning by key.
        |    |  Index contains all needed columns. Base table will not be read.
        |    |  Keys are:
        |    |    au_lname ASC
        |    |  Using I/O Size 2 Kbytes for index leaf pages.
        |    |  With LRU Buffer Replacement Strategy for index leaf pages.
```

该查询计划选择使用 GROUP SORTED Distinct 运算符来应用 distinct 属性，因为 scan 运算符按不同列 au_lname 和 au_fname 的排序顺序返回行。GROUP SORTED 不会产生任何 I/O，且 CPU 开销也最低。

您可以使用 GROUP SORTED Distinct 运算符实现矢量集合。请参见第 77 页的 “矢量集合运算符”。showplan 输出显示行 Distinct 以指出该 GROUP SORTED Distinct 运算符正在实现 distinct 属性。

SORT Distinct 运算符

SORT Distinct 运算符不要求其输入数据流按不同的键列进行排序。它是阻塞运算符，可读取其子运算符的所有数据流并在读取行时排序行。对所有行排序后，将不同的行返回给父运算符。返回按不同的键列排序的行。如果内存中无法容纳整个输入集，则使用内部工作表作为后备存储。

```
QUERY PLAN FOR STATEMENT 1 (at line 1)

STEP 1
The type of query is SELECT.

2 operator(s) under root

ROOT:EMIT Operator (VA = 2)

        |SORT Operator
        | Using Worktable1 for internal storage.
        |
        |    |SCAN Operator
        |    |  FROM TABLE
        |    |  authors
        |    |  Table Scan.
        |    |  Forward Scan.
```

```
    |   |  Positioning at start of table.
    |   |  Using I/O Size 2 Kbytes for data pages.
    |   |  With LRU Buffer Replacement Strategy for data pages.
```

authors 表的扫描不返回按不同的键列排序的行。这要求使用 SORT Distinct 运算符，而不是 GROUP SORTED Distinct 运算符。 SORT 运算符的不同键列是 au_lname 和 au_fname。 showplan 输出指明当内存中无法容纳整个输入集时，将使用 Worktable1 进行磁盘存储。

HASH Distinct 运算符

HASH Distinct 运算符不要求其输入集按不同的键列排序。它是非阻塞运算符。从子运算符读取行并按不同的键列散列。这决定了行所在的桶的位置。然后搜索相应的桶以查看是否已存在该键。如果行包含重复的键，则放弃该行，然后从子运算符获取另一行。如果没有存在任何重复的不同键，则将该行添加到桶中，并将该行向上传递给父运算符以做进一步的处理。返回行时没有按不同的键列对其排序。

当输入集尚未按不同的键列排序时，或者当优化程序无法利用来自计划中稍后的差别处理的排序时，通常使用 HASH Distinct 运算符。

```
select distinct au_lname, au_fname
from authors
where city = "Oakland"
go

QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
The type of query is SELECT.

2 operator(s) under root

ROOT:EMIT Operator (VA = 2)

    |HASH DISTINCT Operator (VA = 1)
    | Using Worktable1 for internal storage.
    |
    |     | SCAN Operator (VA = 0)
    |     | FROM TABLE
    |     | authors
    |     | Table Scan.
    |     | Forward Scan.
    |     | Positioning at start of table.
    |     | Using I/O Size 2 Kbytes for data pages.
    |     | With LRU Buffer Replacement Strategy for data pages.
```

在上述示例中， authors 表扫描的输出没有排序。优化程序可以选择使用 SORT Distinct 或 HASH Distinct 策略。SORT Distinct 策略提供的排序在计划中的其它任何部分都不可用，因此优化程序很可能选择使用 HASH Distinct 策略。优化程序最终将根据开销估计做出决定。HASH Distinct 的开销通常较小，因为未排序的输入数据流可随即消除驻留分区的行。 SORT Distinct 运算符在对整个数据集排序之前无法消除任何行。

HASH Distinct 运算符的 showplan 输出报告将使用 Worktable1。如果内存中无法容纳不同行结果集，则需要工作表。在这种情况下，会将部分处理的组写入磁盘。

矢量集合运算符

有三种用于矢量集合的一元运算符。它们是 GROUP SORTED COUNT AGGREGATE、 HASH VECTOR AGGREGATE 和 GROUP INSERTING 运算符。

有关所有查询处理器运算符的列表和说明，请参见第 20 页的表 1-3。

GROUP SORTED COUNT AGGREGATE 运算符

如第 74 页的 “GROUP SORTED Distinct 运算符”中所述， GROUP SORTED COUNT AGGREGATE 非阻塞运算符是 GROUP SORTED Distinct 运算符的变体。GROUP SORTED COUNT AGGREGATE 运算符要求输入集按 group by 列排序。其算法与 GROUP SORTED Distinct 的算法很类似。

从子运算符读取行。如果该行是新矢量的开头，则高速缓存其分组列，并初始化集合结果。

如果行属于当前正在被处理的组，则对集合结果应用集合函数。当子运算符返回新组开头的一行或 End Of Scan 时，当前矢量及其集合值返回到父运算符。

在处理完整个组后会返回 GROUP SORTED COUNT AGGREGATE 运算符中的第一行，而在开始处理新组时会返回 GROUP SORTED Distinct 运算符中的第一行。以下示例收集所有城市的列表，并指出在每个城市居住的作者的数目。

```
select city, total_authors = count(*)
from authors
group by city
plan
"(group_sorted
(sort (scan authors))
)"
```

```
QUERY PLAN FOR STATEMENT 1 (at line 1).
Optimized using the Abstract Plan in the PLAN clause.

STEP 1
The type of query is SELECT.

3 operator(s) under root

ROOT:EMIT Operator (VA = 3)

    |GROUP SORTED Operator (VA = 2)
    | Evaluate Grouped COUNT AGGREGATE.
    |
    |    |SORT Operator (VA = 1)
    |    | Using Worktable1 for internal storage.
    |    |
    |    |    | SCAN Operator (VA = 0)
    |    |    | FROM TABLE
    |    |    | authors
    |    |    | Table Scan.
    |    |    | Forward Scan.
    |    |    | Positioning at start of table.
    |    |    | Using I/O Size 2 Kbytes for data pages.
    |    |    | With LRU Buffer Replacement Strategy for data pages.
```

在上述查询计划中，authors 的扫描不按分组顺序返回行。SORT 运算符用于按分组列 city 对数据流排序。此时，GROUP SORTED COUNT AGGREGATE 运算符可用于计算 count 集合。

GROUP SORTED COUNT AGGREGATE 运算符的 showplan 输出报告正被使用的集合函数，如下所示：

```
|  Evaluate Grouped COUNT AGGREGATE.
```

HASH VECTOR AGGREGATE 运算符

HASH VECTOR AGGREGATE 运算符是阻塞运算符。必须处理完来自子运算符的所有行后，才能将来自 HASH VECTOR AGGREGATE 运算符的第一行返回给其父运算符。除此之外，其算法与 HASH Distinct 运算符的算法类似。

从子运算符获取行。在查询的分组列中散列各行。搜索散列桶以查看是否已存在矢量。

如果 group by 值不存在，则添加矢量，并使用第一行初始化集合值。如果 group by 值确实存在，则将当前行聚合到现有值。以下示例收集所有城市的列表，并指出在每个城市居住的作者的数目。

```
select city, total_authors = count(*)
from authors
group by city

QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
The type of query is SELECT.

2 operator(s) under root

ROOT:EMIT Operator (VA = 2)

    |HASH VECTOR AGGREGATE Operator (VA = 1)
    | GROUP BY
    | Evaluate Grouped COUNT AGGREGATE.
    | Using Worktable1 for internal storage.
    | Key Count: 1
    |
    |    |SCAN Operator (VA = 0)
    |    | FROM TABLE
    |    | authors
    |    | Table Scan.
    |    | Forward Scan.
    |    | Using I/O Size 2 Kbytes for data pages.
    |    | With LRU Buffer Replacement Strategy for data pages.
```

在上述查询计划中，HASH VECTOR AGGREGATE 运算符从其子运算符读取所有行，其子运算符扫描 authors 表。检查每一行以查看是否已存在当前 city 值的桶项。如果不存在，则添加具有新的 city 分组值的散列项行，并将计数结果初始化为 1。如果已有新行的 city 值的散列项，则应用集合函数。在这种情况下，将增加计数结果。

showplan 输出显示专用于 HASH VECTOR AGGREGATE 运算符的 group by 消息，然后输出分组集合消息：

```
| Evaluate Grouped COUNT AGGREGATE.
```

showplan 输出报告用于存储溢出的组和未处理的行的工作表：

```
| Using Worktable1 for internal storage.
```

GROUP INSERTING

GROUP INSERTING 是阻塞运算符。必须处理完来自子运算符的所有行后，才能从 GROUP INSERTING 返回第一行。

在 group by 子句中， GROUP INSERTING 限制为 31 或更少的列。该运算符以创建带有分组列的聚簇索引的工作表开始。因为每行是从子运算符获取的，因此将根据分组列完成在工作表中进行查找的操作。如果未找到行，则插入行。这样可有效创建新的组以及初始化其集合值。如果找到行，将根据计算的新值来更新新的集合值。 GROUP INSERTING 运算符返回按分组列排序的行。

```
select city, total_authors = count(*)
from authors
group by city
plan
'(group_inserting (i_scan auidind authors ))'
```

```
QUERY PLAN FOR STATEMENT 1 (at line 1).
Optimized using the Abstract Plan in the PLAN clause.

STEP 1
The type of query is SELECT.

2 operator(s) under root
   |ROOT:EMIT Operator (VA = 2)
|
|   |GROUP INSERTING Operator (VA = 1)
|   | GROUP BY
|   | Evaluate Grouped COUNT AGGREGATE
|   | Using Worktable1 for internal storage.
|   |
|   |   |SCAN Operator (VA = 0)
|   |   | FROM TABLE
|   |   | authors
|   |   | Table Scan.
|   |   | Forward Scan.
|   |   | Positioning at start of table.
|   |   | Using I/O Size 2 Kbytes for data pages.
|   |   | With LRU Buffer Replacement Strategy for data pages.
```

在上述示例中，group inserting 运算符是从构建带有聚簇索引键列 city 的工作表开始的。 group inserting 运算符继续清除 authors 表。对于每一行，均根据 city 值来完成查找。如果集合工作表中没有包含当前 city 值的行，将插入行。这将为包含初始化 count 值的当前 city 值创建新组。如果为当前 city 值找到行，将执行计算以增加 COUNT AGGREGATE 值。

compute by 消息

处理在 EMIT 运算符中已完成，并要求按查询中的任何 order by 要求对 EMIT 运算符的输入数据流进行排序。处理方式类似于在 GROUP SORTED AGGREGATE 运算符中的处理方式。

检查从子运算符读取的每一行以查看它是不是新组的开头。如果不是，则根据查询的请求组应用适当的集合函数。如果开始一个新组，则当前组及其集合值将返回到用户。然后，将开始新组，并根据新行的值初始化其集合值。以下示例收集所有城市的有序列表，并在城市列表之后报告每个城市的条目的数目。

```
select city
from authors
order by city
compute count(city) by city

QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
The type of query is SELECT.

2 operator(s) under root

Emit with Compute semantics

ROOT:EMIT Operator (VA = 2)

       |SORT Operator (VA = 1)
       | Using Worktable1 for internal storage.
       |
       |   |SCAN Operator (VA = 0)
       |   |  FROM TABLE
       |   |  authors
       |   |  Table Scan.
       |   |  Forward Scan.
       |   |  Positioning at start of table.
       |   |  Using I/O Size 2 Kbytes for data pages.
       |   |  With LRU Buffer Replacement Strategy for data pages.
```

在上述示例中， EMIT 运算符的输入数据流按 city 属性排序。针对每一行，增加 compute by 的计数值。当获取新的 city 值后，当前的 city 值和关联的计数值将返回给用户。新 city 值将成为新的 compute by 分组值，其计数将初始化为 1。

Union 运算符

UNION ALL 运算符

UNION ALL 运算符合并几个兼容的输入数据流，但不执行任何重复项消除操作。UNION ALL 运算符中的每个数据行将包括在该运算符的输出数据流中。

UNION ALL 运算符是 n 元运算符，显示以下消息：

```
UNION ALL OPERATOR  has N children.
```

N 是运算符中输入数据流的数量。

下列示例演示如何使用 UNION ALL：

```
select * from sysindexes where id < 100
union all
select * from sysindexes where id > 200

QUERY PLAN FOR STATEMENT 1 (at line 1).

  STEP 1
The type of query is SELECT.


3 operator(s) under root

  |ROOT:EMIT Operator (VA = 3)
  |
  |  |UNION ALL Operator (VA = 2) has 2 children.
  |  |

  |  |  |SCAN Operator (VA = 0)
  |  |  | FROM TABLE
  |  |  | sysindexes
  |  |  | Using Clustered Index.
  |  |  | Index : csysindexes
  |  |  | Forward Scan.
  |  |  | Positioning by key.
  |  |  | Keys are:
  |  |  |  id ASC
  |  |  | Using I/O Size 2 Kbytes for index leaf pages.
  |  |  | With LRU Buffer Replacement Strategy for index leaf pages.
  |  |  | Using I/O Size 2 Kbytes for data pages.
  |  |  | With LRU Buffer Replacement Strategy for data pages.
```

```
|   |
|   |   |SCAN Operator (VA = 1)
|   |   | FROM TABLE
|   |   | sysindexes
|   |   | Using Clustered Index
|   |   | Index : csysindexes
|   |   | Forward scan.
|   |   | Positioning by key.
|   |   | Keys are:
|   |   |  id ASC
|   |   | Using I/O Size 2 Kbytes for index leaf pages.
|   |   | With LRU Buffer Replacement Strategy for index leaf pages.
|   |   | Using I/O Size 2 Kbytes for data pages.
|   |   | With LRU Buffer Replacement Strategy for data pages.
```

UNION ALL 运算符以从其最左侧的子运算符获取所有行开始。在上述示例中，返回的所有 sysindexes 行的 ID 均小于 100。因为每个子运算符的数据流已清空，所以 UNION ALL 运算符会立即移动到其右侧的子运算符。将打开并清空此数据流。此操作会继续直到最后一个 （第 *N* 个）子运算符被清空。

MERGE UNION 运算符

MERGE UNION 运算符对几个已排序的兼容数据流执行 UNION ALL 操作，并消除这些数据流中的重复项。

MERGE UNION 运算符是 n 元运算符，显示以下消息：

```
MERGE UNION OPERATOR has <N> children.
```

<***N***> 是运算符中输入数据流的数量。

HASH UNION

HASH UNION 运算符使用 Adaptive Server 散列算法来对几个数据流同时执行 UNION ALL 操作，它还执行基于散列的重复项消除。

HASH UNION 运算符是 n 元运算符，显示以下消息：

```
HASH UNION OPERATOR has <N> children.
```

<***N***> 是运算符中输入数据流的数量。

HASH UNION 还显示它所使用的工作表的名称，其格式如下：

```
HASH UNION OPERATOR Using Worktable  <X>  for internal
storage.
```

HASH UNION 使用此工作表来临时存储无法在当前可用内存中处理的当前迭代的数据。

下列示例演示如何使用 HASH UNION：

```
select * from sysindexes
union
select * from sysindexes

QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
The type of query is SELECT.

3 operator(s) under root

    | ROOT:EMIT Operator (VA = 3)
    |
    |    |HASH UNION Operator has 2 children.
    |    | Using Worktable1 for internal storage.
    |    |
    |    |    |SCAN Operator
    |    |    |  FROM TABLE
    |    |    |  sysindexes
    |    |    |  Table Scan.
    |    |    |  Forward Scan.
    |    |    |  Positioning at start of table.
    |    |    |  Using I/O Size 2 Kbytes for data pages.
    |    |    |  With LRU Buffer Replacement Strategy for data pages.
    |    |
    |    |SCAN Operator (VA = 1)
    |    |    |  FROM TABLE
    |    |    |  sysindexes
    |    |    |  Table Scan.
    |    |    |  Forward Scan.
    |    |    |  Positioning at start of table.
    |    |    |  Using I/O size 2 Kbytes for data pages.
    |    |    |  With LRU Buffer Replacement Strategy for data pages.
```

SCALAR AGGREGATE 运算符

SCALAR AGGREGATE 运算符跟踪有关输入数据流的运行信息，如数据流中的行数，或数据流中给定列的最大值。

SCALAR AGGREGATE 运算符输出一个列表，其中最多包含 10 条描述它所执行的标量集合操作的消息。消息的格式如下：

```
Evaluate Ungrouped  <Type of Aggregate> Aggregate
```

<*Type of Aggregate*> 可以是以下任一种类型：count、 sum、 average、 min、 max、 any、 once-unique、 count-unique、 sum-unique、 average-unique 或 once。

以下查询对数据库 pubs2 中的 authors 表执行 SCALAR AGGREGATE （即解压缩）集合：

```
select count(*) from authors

QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
The type of query is SELECT.

2 operator(s) under root

ROOT:EMIT Operator (VA = 2)

    |SCALAR AGGREGATE Operator (VA = 1)
    | Evaluate Ungrouped COUNT AGGREGATE.
    |
    |    |SCAN Operator (VA =0)
    |    | FROM TABLE
    |    | authors
    |    | Index : aunmind
    |    | Forward Scan.
    |    | Positioning at index start.
    |    | Index contains all needed columns. Base table will not be read.
    |    | Using I/O Size 4 Kbytes for index leaf pages.
    |    | With LRU Buffer Replacement Strategy for index leaf pages.
```

SCALAR AGGREGATE 消息指出要执行的查询是拆组 count 集合。

RESTRICT 运算符

RESTRICT 运算符是根据列值计算表达式的一元运算符。 RESTRICT 运算符与多个列求值列表相关，可以在从子运算符获取行之前处理这些列表，也可以在从子运算符获取行之后处理，还可以在从子运算符获取行之后处理它们以计算虚拟列的值。

SORT 运算符

SORT 运算符在查询计划中只有一个子运算符。它的作用是使用指定的排序键从输入数据流生成输出数据流。

SORT 运算符可能执行数据流排序，但也有可能必须将结果临时存入工作表中。 SORT 运算符按下列格式显示工作表的名称：

```
Using Worktable<N>  for internal storage.
```

其中， <***N***> 是工作表在 showplan 输出中的数字标识符。

下面是使用 SORT 运算符和工作表的简单查询计划的示例：

```
select au_id from authors order by postalcode

QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
The type of query is SELECT.

2 operator(s) under root

ROOT:EMIT Operator (VA = 2)

    |SORT Operator (VA = 1)
    | Using Worktable1 for internal storage.
    |
    |    |SCAN Operator (VA = 0)
    |    | FROM TABLE
    |    | authors
    |    | Table Scan.
    |    | Forward Scan.
    |    | Positioning at start of table.
    |    | Using I/O Size 4 Kbytes for data pages.
    |    | With LRU Buffer Replacement Strategy for data pages.
```

SORT 运算符读取其子运算符并对行排序。在这种情况下，它将使用 postalcode 属性对从 authors 表获取的每一行进行排序。如果所有行均可以放入内存，则没有数据会溢出到磁盘上。但是，如果输入数据的大小超过可用缓冲区空间，则排序行会溢出到磁盘上。这些排序行会递归合并到较大的排序行中，直到排序行大小比用于读取和合并这些行的缓冲区小。

STORE 运算符

STORE 运算符用于创建和填充工作表，也可以在工作表上创建索引。作为查询计划执行的一部分，该工作表可由计划中的其它运算符使用。SEQUENCER 运算符可确保先执行与工作表和潜在索引创建相对应的计划片段，然后再执行使用该工作表的其它计划片段。由于执行过程是异步运行的，因此这在并行执行计划时很重要。

重新格式化策略使用 STORE 运算符创建包含聚簇索引的工作表。

如果 STORE 运算符用于重新格式化操作，它会输出以下消息：

```
Worktable <X> created, in <L> locking mode for
reformatting.
```

锁定模式 <***L***> 必须是 “allpages”、“datapages”、“datarows”中的一个。

STORE 运算符也会输出以下消息：

```
Creating clustered index.
```

如果 STORE 运算符不用于重新格式化操作，它会输出以下消息：

```
Worktable <X> created, in <L> locking mode.
```

以下示例适用于 STORE 运算符以及 SEQUENCER 运算符。

```
select * from bigun a, bigun b where a.c4 = b.c4 and a.c2 < 10

QUERY PLAN FOR STATEMENT 1 (at line 1).
Optimized using the Abstract Plan in the PLAN clause.

STEP 1
  The type of query is SELECT.

  7 operator(s) under root

  |ROOT:EMIT Operator (VA = 7)
  |
  |  |SEQUENCER Operator (VA = 6) has 2 children.
  |  |
```

```
|   |   |STORE Operator (VA = 5)
|   |   | Worktable1 created, in allpages locking mode, for REFORMATTING.
|   |   | Creating clustered index.
|   |   |
|   |   |   |INSERT Operator(VA = 4)
|   |   |   | The update mode is direct.
|   |   |   |
|   |   |   |   |SCAN Operator(VA = 0)
|   |   |   |   | FROM TABLE
|   |   |   |   | bigun
|   |   |   |   | b
|   |   |   |   | Table Scan.
|   |   |   |   | Forward Scan.
|   |   |   |   | Positioning at start of table.
|   |   |   |   | Using I/O Size 2 Kbytes for data pages.
|   |   |   |   | With LRU Buffer Replacement Strategy for data pages.
|   |   |   |
|   |   |   | TO TABLE (VA = 3)
|   |   |   | Worktable1.
|   |
|   |   |NESTED LOOP JOIN (Join Type:Inner Join)(VA = 7)
|   |   |
|   |   |   |SCAN Operator (VA = 2)
|   |   |   | FROM TABLE
|   |   |   | bigun
|   |   |   | a
|   |   |   | Table Scan.
|   |   |   | Forward Scan.
|   |   |   | Positioning at start of table.
|   |   |   | Using I/O Size 2 Kbytes for data pages.
|   |   |   | With LRU Buffer Replacement Strategy for data pages.
|   |   |
|   |   |   |SCAN Operator (VA = 1)
|   |   |   | FROM TABLE
|   |   |   | Worktable1.
|   |   |   | Using Clustered Index.
|   |   |   | Forward Scan.
|   |   |   | Positioning key.
|   |   |   | Using I/O Size 2 Kbytes for data pages.
|   |   |   | With LRU Buffer Replacement Strategy for data pages.
```

在上述计划示例中，在重新格式化策略中使用了 STORE 运算符。它位于 SEQUENCER 运算符的最左侧子运算符中的 SEQUENCER 运算符的正下方。

STORE 运算符将创建 Worktable1，它下面的 INSERT 运算符将填充该工作表。 STORE 运算符然后将在 Worktable1 上创建聚簇索引。该索引将建立在连接键 b.c4 上。

SEQUENCER 运算符

SEQUENCER 运算符是 n 元运算符，用来按顺序执行它下面的各个子计划。 SEQUENCER 运算符用于 reformatting 计划及某些集合处理计划。

除最右侧的子计划之外， SEQUENCER 运算符将执行其子运算符的每个子计划。在执行所有左子运算符的子计划后，它将执行最右侧的子计划。

SEQUENCER 运算符将显示以下消息：

```
SEQUENCER operator has N children.
```

```
select * from tab1 a, tab2 b where a.c4 = b.c4 and a.c2 < 10

QUERY PLAN FOR STATEMENT 1 (at line 1).
Optimized using the Abstract Plan in the PLAN clause.

  STEP 1
    The type of query is SELECT.

    7 operator(s) under root

    |ROOT:EMIT Operator (VA = 7)
    |
    |  |SEQUENCER Operator (VA = 6) has 2 children.
    |  |
    |  |  |STORE Operator (VA = 5)
    |  |  | Worktable1 created, in allpages locking mode, for REFORMATTING.
    |  |  | Creating clustered index.
    |  |  |
    |  |  |  |INSERT Operator (VA = 4)
    |  |  |  | The update mode is direct.
    |  |  |  |
    |  |  |  |  |SCAN Operator (VA = 0)
    |  |  |  |  | FROM TABLE
    |  |  |  |  | tab2
    |  |  |  |  | b
    |  |  |  |  | Table Scan.
    |  |  |  |  | Forward Scan.
    |  |  |  |  | Positioning at start of table.
    |  |  |  |  | Using I/O Size 2 Kbytes for data pages.
    |  |  |  |  | With LRU Buffer Replacement Strategy for data pages.
    |  |  |  |
    |  |  |  | TO TABLE
    |  |  |  | Worktable1.
    |  |
    |  |  |NESTED LOOP JOIN Operator (Join Type:Inner Join) (VA = 3)
    |  |  |
```

```
|   |   |   |SCAN Operator (VA = 2)
|   |   |   | FROM TABLE
|   |   |   | tab1
|   |   |   | a
|   |   |   | Table Scan.
|   |   |   | Forward Scan.
|   |   |   | Positioning at start of table.
|   |   |   | Using I/O Size 2 Kbytes for data pages.
|   |   |   | With LRU Buffer Replacement Strategy for data pages.
|   |   |   |
|   |   |   |SCAN Operator (VA = 1)
|   |   |   | FROM TABLE
|   |   |   | Worktable1.
|   |   |   | Using Clustered Index.
|   |   |   | Forward Scan.
|   |   |   | Positioning by key.
|   |   |   | Using I/O Size 2 Kbytes for data pages.
|   |   |   | With LRU Buffer Replacement Strategy for data pages.
```

在上述示例中， SEQUENCER 运算符实施重新格式化策略。 SEQUENCER 运算符的最左侧的分支将在 Worktable1 上创建聚簇索引。在执行并关闭此分支之后， SEQUENCER 运算符才能继续下一个子运算符。 SEQUENCER 运算符到达最右侧的子运算符后，即会打开并开始清除它，将行返回到其父运算符。 SEQUENCER 运算符的设计意图是，使最右侧分支中的运算符能够使用在 SEQUENCER 运算符的先前外部分支中创建的工作表。在此示例中， Worktable1 用于嵌套循环连接策略。 Worktable1 的扫描由来自 tab1 的外部扫描的每一行的聚簇索引上的键定位。

REMOTE SCAN 运算符

REMOTE SCAN 运算符将 SQL 查询发送给远程服务器进行执行。然后，它将处理由远程服务器返回的结果 （如果有）。 REMOTE SCAN 显示它所处理的 SQL 查询的格式化文本。

REMOTE SCAN 没有子运算符，或有 1 个子运算符。

SCROLL 运算符

SCROLL 运算符封装 Adaptive Server 中可滚动游标的功能。可滚动游标可能处于 insensitive 状态，这意味着它们将显示在打开游标时拍摄的与其相关的数据的快照；可滚动游标也可能处于 semi-sensitive 状态，这意味着将从活动数据检索要获取的下一行或几行。

SCROLL 运算符是一元运算符，显示以下消息：

```
SCROLL OPERATOR ( Sensitive Type:<T>)
```

类型可能是 insensitive 或 semi-sensitive。

以下是具有 insensitive 可滚动游标的计划示例：

```
declare CI insensitive scroll cursor for
select au_lname, au_id from authors
go
set showplan on
go
open CI

QUERY PLAN FOR STATEMENT 1 (at line 1).

STEP 1
The type of query is OPEN CURSOR CI.


QUERY PLAN FOR STATEMENT 1 (at line 2).

STEP 1
The type of query is DECLARE CURSOR.

2 operator(s) under root

ROOT:EMIT Operator (VA = 2)

   |SCROLL Operator (Sensitive Type: Insensitive) (VA = 1)
   | Using Worktable1 for internal storage.
   |
   |   |SCAN Operator (VA = 0)
   |   | FROM TABLE
   |   | authors
   |   | Table Scan.
   |   | Forward Scan.
   |   | Positioning at start of table.
   |   | Using I/O Size 4 Kbytes for data pages.
   |   | With LRU Buffer Replacement Strategy for data pages.
```

SCROLL 运算符是根 EMIT 运算符的子运算符，其唯一子运算符是 authors 表上的 SCAN 运算符。 SCROLL 消息指定 CI 游标处于 insensitive 状态。

可滚动游标行最初高速缓存在内存中。如果所处理的数据量超过高速缓存的物理内存限制，则 Worktable1 将用作此高速缓存的后备存储。

RID JOIN 运算符

RID JOIN 运算符是二元运算符，基于为相同源表生成的行 ID 来连接两个数据流。 SQL 表中的各数据行与唯一的行 ID (RID) 相关。可将 RID 连接看作是特殊情形下的自连接查询。左子运算符使用一组唯一限定 RID 来填充工作表。 RID 是将不同的过滤器应用到从相同源表的两个或多个不同的索引用例中返回的 RID 的结果。

RID JOIN 运算符用于执行常规 or 策略。该常规 -or 策略经常在查询的谓词包含可由相同表上的不同索引限定的分离集合时使用。在这种情况下，将基于可由该索引限定的谓词对每个索引进行扫描。针对限定的每个索引行，将返回 RID。

将对返回的 RID 进行唯一性处理，以便同一行不会返回两次 （如果两个或多个分离限定同一行，则可能发生此情况）。

RID JOIN 运算符将唯一 RID 插入工作表。唯一 RID 的工作表传递给 RID 连接右侧分支中的 scan 运算符。访问方法可通过迭代方式直接从工作表获取要处理的下一个 RID，并查找关联行。然后，该行返回到 RID JOIN 父运算符。

RID JOIN 运算符将显示以下消息：

```
Using Worktable <N> for internal storage.
```

此工作表用于存储从左子运算符生成的唯一 RID。

以下示例阐明 RID JOIN 运算符的 showplan 输出。

```
select * from tab1 a where a.c1 = 10 or a.c3 = 10

QUERY PLAN FOR STATEMENT 1 (at line 2).

  STEP 1
The type of query is SELECT.

6 operator(s) under root.

  |ROOT:EMIT Operator (VA = 6)
  |
  |  |RID JOIN Operator (VA = 5)
```

```
|   | Using Worktable2 for internal storage.
|   |
|   |  |HASH UNION Operator (VA = 6) has 2 children.
|   |  | Key Count: 1
|   |  |
|   |  |  |SCAN Operator (VA = 0)
|   |  |  | FROM TABLE
|   |  |  | tab1
|   |  |  | a
|   |  |  | Index: tab1idx
|   |  |  | Forward Scan.
|   |  |  | Positioning by key.
|   |  |  | Index contains all needed columns. Base table will not be read.
|   |  |  | Keys are:
|   |  |  | c1 ASC
|   |  |  | Using I/O Size 2 Kbytes for index leaf pages.
|   |  |  | With LRU Buffer Replacement Strategy for index leaf pages.
|   |  |  |
|   |  |  |SCAN Operator (VA = 4)
|   |  |  | FROM TABLE
|   |  |  | tab1
|   |  |  | a
|   |  |  | Index: tab1idx2
|   |  |  | Forward Scan.
|   |  |  | Positioning by key.
|   |  |  | Index contains all needed columns. Base table will not be read.
|   |  |  | Keys are:
|   |  |  |  c3 ASC
|   |  |  | Using I/O Size 2 Kbytes for index leaf pages.
|   |  |  | With LRU Buffer Replacement Strategy for index leaf pages.
|   |  |
|   |  |RESTRICT Operator (VA = 3)
|   |  |  |
|   |  |  |  |SCAN Operator (VA = 2)
|   |  |  |  | FROM TABLE
|   |  |  |  | tab1
|   |  |  |  | a
|   |  |  |  | Using Dynamic Index.
|   |  |  |  | Forward Scan.
|   |  |  |  | Positioning by Row IDentifier (RID).
|   |  |  |  | Using I/O Size 2 Kbytes for data pages.
|   |  |  |  | With LRU Buffer Replacement Strategy for data pages.
```

在上述示例中，扫描索引 tab1idx 以从 c1 值为 10 的 tab1 中获取所有 RID。Adaptive Server 将扫描 tab1idx2 以从 c3 值为 10 的 tab1 中获取所有 RID。

HASH UNION 运算符用于消除重复的 RID。其中 c1 和 c3 行的值均为 10 的任何 tab1 行都存在重复的 RID。

RID JOIN 运算符将所有返回的行均插入 Worktable2。在其完全填满之后，Worktable2 将传递到 tab1 的扫描。访问方法获取第一个 RID，查找关联行，并将其返回到 RID JOIN 运算符。在后续调用 tab1 的扫描运算符时，访问方法将获取要处理的下一个 RID 并返回其关联行。

SQLFILTER 运算符

SQLFILTER 运算符是执行子查询的 n 元运算符。它最左边的子运算符代表外部查询，其它子运算符代表与一个或多个子查询相关的查询计划片段。

最左侧的子运算符生成将代入其它子计划的相关值。

SQLFILTER 运算符将显示以下消息：

```
SQFILTER Operator has <N> children.
```

以下示例说明 SQLFILTER 的用法：

```
select pub_name from publishers
where pub_id =
(select distinct titles.pub_id from titles
   where publishers.pub_id = titles.pub_id
   and price > $1000)
QUERY PLAN FOR STATEMENT 1 (at line 1).
4 operator(s) under root

STEP 1
The type of query is SELECT.

4 operator(s) under root

ROOT:EMIT Operator (VA = 4)

  |SQFILTER Operator (VA = 3) has 2 children.
  |
  |  |SCAN Operator (VA = 0)
  |  |  FROM TABLE
  |  |  publishers
  |  |  Table Scan.
  |  |  Forward Scan.
  |  |  Positioning at start of table.
  |  |  Using I/O Size 8 Kbytes for data pages.
```

```
|   |  With LRU Buffer Replacement Strategy for data pages.
|
| Run subquery 1 (at nesting level 1)
|
|  QUERY PLAN FOR SUBQUERY 1 (at nesting level 1 and at line 3)
|
|  Correlated Subquery
|  Subquery under an EXPRESSION predicate.
|
|   |SCALAR AGGREGATE Operator (VA = 2)
|   |  Evaluate Ungrouped ONCE-UNIQUE AGGREGATE
|   |
|   |  |SCAN Operator (VA = 1)
|   |  |  FROM TABLE
|   |  |  titles
|   |  |  Table Scan.
|   |  |  Forward Scan.
|   |  |  Postitioning at start of table.
|   |  |  Using I/O Size 8 Kbytes for data pages.
|   |  |  With LRU Buffer Replacement Strategy for data pages.
|
| END OF QUERY PLAN FOR SUBQUERY 1
```

在上述示例中， SQLFILTER 运算符有两个子运算符。最左侧的子运算符是查询的外部块。它是 publishers 表的简单扫描。右子运算符用于对查询的子查询求值。 SQLFILTER 从外部块获取行。针对来自外部块的每一行， SQLFILTER 调用右子运算符来对子查询求值。如果子查询求值为 TRUE，将向 SQLFILTER 的父运算符返回一行。

EXCHANGE 运算符

EXCHANGE 运算符是一元运算符，可封装 Adaptive Server SQL 查询的并行处理。 EXCHANGE 几乎可以位于查询计划中的任何位置，并将查询计划分为计划片段。计划片段是一个查询计划树，其根为 EMIT 或 EXCHANGE:EMIT 运算符和其叶为 SCAN 或 EXCHANGE 运算符。串行计划是由单个进程执行的计划片段。

EXCHANGE 运算符的子运算符始终是 EXCHANGE:EMIT 运算符。 EXCHANGE:EMIT 是新计划片段的根。 EXCHANGE 运算符具有相关联的称为 Beta 进程的服务器进程，该 Beta 进程用作 EXCHANGE 运算符的工作进程的本地执行事务协调器。工作进程根据父 EXCHANGE 运算符及其 Beta 进程的指示执行计划片段。通常使用两个或多个进程以并行方式执行计划片段。 EXCHANGE 运算符和 Beta 进程可协调活动，包括片段边界之间的数据交换。

最顶层的计划片段（根为 EMIT 运算符，而不是 EXCHANGE:EMIT 运算符）由 Alpha 进程执行。 Alpha 进程是与用户连接相关的消耗程序进程。Alpha 进程是所有查询计划的工作进程的全局事务协调器。它负责初步设置所有计划片段的工作进程并最终将其冻结。在例外的情况下，它管理并协调所有片段的工作进程。

EXCHANGE 运算符显示以下消息：

```
Executed in parallel by N producer and P consumer processes.
```

生产者数量是指执行 EXCHANGE 运算符下面的计划片段的工作进程数。消耗程序数量是指执行包含 EXCHANGE 运算符的计划片段的工作进程数。消耗程序处理生产者传递给它们的数据。数据通过 EXCHANGE 运算符中设置的管道在生产者和消耗程序进程之间进行交换。当消耗程序从该管道读取行时，生产者的 EXCHANGE:EMIT 运算符会将这些行写入管道。管道机制可同步生产者写入和消耗程序读取，以便不丢失数据。

以下示例说明 master 数据库中对系统表 sysmessages 进行的并行查询：

```
use master
go
set showplan on
go
select count(*) from sysmessages t1 plan '(t_scan t1)  (prop t1 (parallel 4))

QUERY PLAN FOR STATEMENT 1 (at line 1).
Optimized using the forced options (internally generated Abstract Plan).
Executed in parallel by coordinating process and 4 worker processes.

4 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator
       |SCALAR AGGREGATE Operator
       |  Evaluate Ungrouped COUNT AGGREGATE.
       |
       |   |EXCHANGE Operator
       |   |Executed in parallel by 4 Producer and 1 Consumer processes.

       |   |
       |   |   |EXCHANGE:EMIT Operator
       |   |   |
       |   |   |   |SCAN Operator
       |   |   |   |  FROM TABLE
       |   |   |   |  sysmessages
       |   |   |   |  Table Scan.
```

```
|    |    |    |  Forward Scan.
|    |    |    |  Positioning at start of table.
|    |    |    |  Executed in parallel with a 4-way hash scan.
|    |    |    |  Using I/O Size 4 Kbytes for data pages.
|    |    |    |  With LRU Buffer Replacement Strategy for data pages.
```

上述示例中有两个计划片段。第一个片段在任何计划中不论是否并行，始终以 EMIT 运算符为根。上述示例中的第一个片段由 EMIT、 SCALAR AGGREGATE 和 EXCHANGE 运算符组成。此第一个片段始终由单个 Alpha 进程执行。在上述示例中，它也充当 Beta 进程，负责管理 EXCHANGE 运算符的工作进程。

第二个计划片段的根为 EXCHANGE:EMIT 运算符。其唯一的子运算符是 SCAN 运算符。 SCAN 运算符负责扫描 sysmessages 表。该扫描以并行方式执行：

```
Executed in parallel with a 4-way hash scan
```

这表明每个工作进程负责扫描表的大约四分之一。基于所拥有的数据页 ID 将页分配给工作进程。

EXCHANGE:EMIT 运算符通过写入由其父 EXCHANGE 运算符创建的管道来将数据行写入消耗程序。在上述示例中，管道是四比一式的多路复用器，包括多种可实现完全不同的行为的管道类型。

INSTEAD-OF TRIGGER 运算符

有两个运算符与 instead-of 触发器功能相关联：INSTEAD-OF TRIGGER 和 CURSOR SCAN。 Adaptive Server 15.0.2 及更高版本中提供 instead-of 触发器功能。当 inserts、 deletes 和 updates 操作原本不明确时， instead-of 触发器功能可使用伪表，以允许用户在视图上应用这些特定操作。

INSTEAD-OF TRIGGER 运算符

INSTEAD-OF TRIGGER 运算符只出现在其上创建了 instead-of 触发器的视图上 insert、 update 或 delete 语句的查询计划中。它的主要功能是创建和填充触发器中使用的已插入或已删除的伪表，以检查本应由原来的 insert、 update 或 delete 查询修改的行。包含 INSTEAD-OF TRIGGER 运算符的查询计划的唯一目的是，填充已插入和已删除的表 — 决不会尝试在语句中引用的视图上进行原 SQL 语句的实际操作。而是由触发器根据数据在已插入和已删除的伪表中的可用性来对视图的基础表执行更新。

以下是 INSTEAD-OF TRIGGER 运算符的 showplan 输出的一个示例：

```
create table t12 (c0 int primary key, c1 int null, c2 int null)
go
. . .
create view t12view as select c1,c2 from t12
go
create trigger v12updtrg on t12view
instead of update as
select * from deleted
go
update t12view set c1 = 3

QUERY PLAN FOR STATEMENT 1 (at line 1).

  STEP 1
    The type of query is SELECT.

2 operator(s) under root

   |ROOT:EMIT Operator (VA = 1)
   |
   |  |INSTEAD-OF TRIGGER Operator
   |  | Using Worktable1 for internal storage.
   |  | Using Worktable2 for internal storage.
   |  |
   |  |  |SCAN Operator (VA = 0)
   |  |  | FROM TABLE
   |  |  | t12
   |  |  | Table Scan.
   |  |  | Forward Scan.
   |  |  | Positioning at start of table.
   |  |  | Using I/O Size 2 Kbytes for data pages.
   |  |  | With LRU Buffer Replacement Strategy for data pages.
```

在上述示例中， v12updtrig instead-of 触发器是在 t12view 上定义的。对 t12view 的更新导致创建 INSTEAD-OF TRIGGER 运算符。 INSTEAD-OF TRIGGER 运算符会创建两个工作表。 Worktable1 和 Worktable2 分别用于容纳已插入的行和已删除的行。由于这些表要在语句间保持，因此它们是唯一的。触发器执行会导致输出以下 showplan 行。

```
QUERY PLAN FOR STATEMENT 1 (at line 3).

  STEP 1
    The type of query is SELECT.

1 operator(s) under root
```

```
  |ROOT:EMIT Operator (VA = 1)
  |
  |   |SCAN Operator (VA = 0)
  |   | FROM CACHE
```

上述 showplan 语句输出是针对触发器语句 select * from deleted 的。要从视图中删除的行已在执行初始更新语句时插入 “deleted” 高速缓存。然后，触发器对表进行扫描，报告已从 t12view 视图中删除的行。

CURSOR SCAN 运算符

CURSOR SCAN 运算符只显示在其上创建了 instead-of 触发器的视图上的 delete 或 update （即 delete view-name where current of *cursor_name*）定位语句中。因此，它仅作为 INSTEAD-OF TRIGGER 运算符的子运算符显示。delete 或 update 定位语句仅访问当前已在其上定位游标的行。 CURSOR SCAN 运算符直接从 fetch cursor 语句的查询计划的 EMIT 运算符中读取当前的游标行。这些值将传递给 INSTEAD-OF TRIGGER 运算符，以便插入已插入或已删除的伪表中 （以下示例和上一示例使用的表相同）。

```
declare curs1 cursor for select * from t12view
go
open curs1
go
fetch curs1

c1          c2
__________ _________
         1          2

(1 row affected)
set showplan on
go
update t12view set c1 = 3
where current of curs1

QUERY PLAN FOR STATEMENT (at line 1).

  STEP 1
    The type of query is SELECT.

2 operator(s) under root

  |ROOT:EMIT Operator (VA = 2)
  |
```

```
  |  |INSTEAD-OF TRIGGER Operator (VA = 1)
  |  | Using Worktable1 for internal storage.
  |  | Using Worktable2 for internal storage.
  |  |
  |  |  |CURSOR SCAN Operator (VA = 0)
  |  |  | FROM EMIT OPERATOR
```

上述示例中的 showplan 输出与上一个 INSTEAD-OF TRIGGER 运算符示例中的输出基本相同，只有一点区别。CURSOR SCAN 运算符作为 INSTEAD-OF TRIGGER 运算符的子运算符显示，而不是作为视图的基础表的扫描显示。

CURSOR SCAN 通过访问游标读取的结果来获取要插入伪表的值。这是通过 FROM EMIT OPERATOR 消息传达的。

```
QUERY PLAN FOR STATEMENT 1 (at line 3).

  STEP 1
    The type of query is SELECT.

1 operator(s) under root

   |ROOT:EMIT Operator (VA = 1)
   |
   |  |SCAN Operator (VA = 0)
   |  | FROM CACHE
```

上述 showplan 语句针对触发器语句。它与 INSTEAD-OF TRIGGER 示例中的输出相同。

deferred_index 和 deferred_varcol 消息

```
The update mode is deferred_varcol.

The update mode is deferred_index.
```

这些 showplan 消息指示 Adaptive Server 可以将 update 命令作为延迟索引更新来处理。

更新一个或多个可变长度列时，Adaptive Server 使用 deferred_varcol 模式。此更新操作可按延迟模式或直接模式进行，具体使用哪种模式，取决于只在运行时间才可用的消息。

当索引是唯一的或者可能在更新过程中发生更改时， Adaptive Server 使用 deferred_index 模式。在此模式下， Adaptive Server 按直接模式删除索引条目，但按延迟模式插入索引条目。

第 3 章

显示查询优化策略和估计值

本章介绍由 set 命令的查询优化选项输出的消息。

生成文本格式消息的 *set* 命令

查询优化程序或查询执行层可以生成诊断输出。要以文本格式生成诊断输出，请使用：

```
set option
  {  {show | show_lop | show_managers | show_log_props |
     show_parallel | show_histograms | show_abstract_plan |
     show_search_engine | show_counters | show_best_plan |
     show_code_gen | show_pio_costing | show_lio_costing |
     show_pll_costing | show_elimination | show_missing_stats}
    {normal | brief | long | on | off} }...
```

注释 每个指定的选项必须后跟 normal、 brief、 long、 on 或 off 之一。 on 和 normal 是等效的。每个 show 选项必须包括这些选项之一（normal、 brief 等等）；通过用逗号分隔每个选项或选项对可在一个 set 选项命令中指定多个选项。

有关使用 set 选项的示例，请参见第 109 页的 “诊断使用情景”。

表 3-1：生成文本格式消息的优化程序 set 命令

选项	定义
show	显示一个合理的详细信息集合，该集合取决于选择的 {normal | brief | long | on | off}
show_lop	显示所用的逻辑运算符
show_managers	显示优化期间所用的数据结构管理器
show_log_props	显示求值的逻辑属性

选项	定义
show_parallel	显示并行查询优化的详细信息
show_histograms	显示与 SARG/ 连接列关联的直方图的处理
show_abstract_plan	显示抽象计划的详细信息
show_search_engine	显示连接顺序算法的详细信息
show_counters	显示优化计数器
show_best_plan	显示优化程序选定的最佳查询计划的详细信息
show_code_gen	显示代码生成的详细信息
show_pio_costing	显示物理输入 / 输出 （读写磁盘）评估值
show_lio_costing	显示逻辑输入 / 输出 （读写内存）评估值
show_pll_costing	显示与并行执行开销相关的估计值
show_elimination	显示分区排除
show_missing_stats	显示 SARG/ 连接列中缺少的有用统计信息的详细信息

生成 XML 格式消息的 *set* 命令

可以以 XML 文档格式重新生成诊断。这更易于前端工具解读文档。如果启用了 XML 选项，则可以使用 Adaptive Server 中的本机 XPath 查询处理器查询此输出。

查询优化程序或查询执行层可以生成诊断输出。要生成诊断输出的 XML 文档，请使用以下 set plan 命令：

```
set plan for
    {show_exec_xml, show_opt_xml, show_execio_xml,
    show_lop_xml, show_managers_xml, show_log_props_xml,
    show_parallel_xml, show_histograms_xml, show_final_plan_xml,
    show_abstract_plan_xml, show_search_engine_xml,
    show_counters_xml, show_best_plan_xml, show_pio_costing_xml,
    show_lio_costing_xml, show_elimination_xml}
    to {client | message} on
```

选项	定义
show_exec_xml	以 XML 形式获取编译的计划输出，显示每个查询计划运算符。
show_opt_xml	获取优化程序诊断输出，其中显示不同的组成部分，例如逻辑运算符、管理器输出、某些搜索引擎诊断和最佳查询计划。
show_execio_xml	获取计划输出以及估计的和实际的 I/O。 show_execio_xml 还包括查询文本。
show_lop_xml	以 XML 形式获取输出逻辑运算符树。
show_managers_xml	显示查询优化程序准备阶段中各组件管理器的输出。
show_log_props_xml	显示给定等效类的逻辑属性 （查询中的一个或多个关系组）。

选项	定义
show_parallel_xml	显示生成并行查询计划时与优化程序有关的诊断。
show_histograms_xml	显示与直方图和直方图合并有关的诊断。
show_final_plan_xml	获取计划输出。不包括估计的和实际的 I/O。 show_final_plan_xml 包括查询文本。
show_abstract_plan_xml	显示生成的抽象计划。
show_search_engine_xml	显示与搜索引擎相关的诊断。
show_counters_xml	显示计划对象构造 / 破坏计数器。
show_best_plan_xml	以 XML 形式显示最佳计划。
show_pio_costing_xml	以 XML 形式显示实际的物理输入 / 输出开销。
show_lio_costing_xml	以 XML 形式显示实际的逻辑输入 / 输出开销。
show_elimination_xml	以 XML 形式显示分区排除。
client	如果指定，则将输出发送到客户端。缺省情况下这是错误日志。但如果打开跟踪标志 3604，输出将发送到客户端连接。
message	如果指定，则将输出发送到内部消息缓冲区。

若要关闭某一选项，请指定：

```
set plan for
    {show_exec_xml, show_opt_xml, show_execio_xml, show_lop_xml,
    show_managers_xml, show_log_props_xml, show_parallel_xml,
    show_histograms_xml,show_final_plan_xml
    show_abstract_plan_xml, show_search_engine_xml,
    show_counters_xml, show_best_plan_xml, show_pio_costing_xml,
    show_lio_costing_xml, show_elimination_xml} off
```

关闭选项时，不需要指定目标流。

如果指定了 message，客户端应用程序必须使用名为 showplan_in_xml(query_num) 的内置函数从缓冲区中获得诊断信息。

query_num 指的是在缓冲区中高速缓存的查询数。当前，缓冲区中最多可高速缓存 20 个查询。当查询达到 20 个时，高速缓存会停止收集查询计划；它会忽略其余的查询计划。但是，消息缓冲区会继续收集查询计划。在超过 20 个查询后，您只能使用值 0 来显示整个消息缓冲区。

query_num 的有效值为 1 – 20、 -1 和 0 （零）。值 -1 指的是高速缓存中的最后一个 XML 文档；值 0 指的是整个消息缓冲区。

消息缓冲区可能会溢出。如果发生溢出，则无法记录所有 XML 文档，这可能造成 XML 文档不完整和无效。

当使用 showplan_in_xml 访问消息缓冲区时，在执行后会清空缓冲区。

您可能要使用 set textsize 设置最大文本大小，因为 XML 文档会作为文本列输出，并且如果该列不够大，文档将会被截断。例如，使用以下命令将文本大小设置为 100000 个字节：

```
set textsize 100000
```

当 set plan 与 off 一起发出时，如果已关闭所有跟踪选项，则将关闭所有 XML 跟踪。否则，仅关闭指定的选项。以前打开的其它选项仍然有效，并会在指定的目标流中继续跟踪。当您发出另一个 set plan 选项时，之前的选项会与当前选项联合，但目标流将无条件地切换到新的目标流。

使用 show_execio_xml 诊断查询计划

show_execio_xml 包括诊断信息，您可能会发现这些信息有助于调查有问题的查询。 show_execio_xml 提供的信息包括：

- 查询计划的版本级别。计划的每个版本都唯一地进行标识。这是计划的第一个版本：

  ```
  <planVersion>1.0</planVersion>
  ```

- 批处理或存储过程中的语句号，以及语句在初始文本中的行号。查询中的语句号是 2，行号是 6：

  ```
  <statementNum>2</statementNum>
  <lineNum>6</lineNum>
  ```

- 查询的抽象计划。例如，下面是查询 select * from titles 的抽象计划：

```
<abstractPlan>
        <![CDATA[
                ( i_scan titleidind titles ) ( prop titles ( parallel 1
) ( prefetch 8 ) ( lru ) )
]]>
        </abstractPlan>
```

- 逻辑 I/O、物理 I/O 和 CPU 开销：

  ```
  <costs>
     <lio> 2 </lio>
     <pio> 2 </pio>
     <cpu> 18 </cpu>
  </costs>
  ```

  可以使用以下公式估计总开销 （25、 2 和 0.1 是常量）：

  25 X *pio* + 2 X *lio* + 0.1 X *cpu*

- 估计的执行资源使用情况，包括查询计划使用的线程数和辅助扫描描述符数。
- 查询引擎查看的计划数和它确定为有效的计划数，查询在查询引擎中花费的总时间 （以毫秒为单位），查询引擎确定第一个合法计划所花费的时间，以及在优化过程中使用的过程高速缓存量。

```
<optimizerMetrics>
    <optTimeMs>6</optTimeMs>
    <optTimeToFirstPlanMs>3</optTimeToFirstPlanMs>
    <plansEvaluated>1</plansEvaluated>
    <plansValid>1</plansValid>
    <procCacheBytes>140231</procCacheBytes>
</optimizerMetrics>
```

- 上次对当前表运行 update statistics 的时间，以及查询引擎是否对给定列使用了估计常量 （如果统计信息可用，则可以进行更准确的估计）。本部分包括有关缺少统计信息的列的信息：

```
<optimizerStatistics>
    <statInfo>
        <objName>titles</objName>
        <columnStats>
            <column>title_id</column>
            <updateTime>Oct  5 2006  4:40:14:730PM</updateTime>
        </columnStats>
        <columnStats>
            <column>title</column>
            <updateTime>Oct  5 2006  4:40:14:730PM</updateTime>
        </columnStats>
    </statInfo>
</optimizerStatistics>
```

- 运算符树，包括表和索引扫描以及有关高速缓存策略和 I/O 大小的信息 （insert、 update 和 delete 具有目标表的相同信息）。运算符树还显示是以 “直接” 模式还是 “延迟” 模式执行 update。 exchange 运算符包括有关查询使用的生产者进程和消耗程序进程数的信息。

```
<TableScan>
    <VA>0</VA>
    <est>
        <rowCnt>18</rowCnt>
        <lio>2</lio>
        <pio>2</pio>
        <rowSz>218.5555</rowSz>
    </est>
    <varNo>0</varNo>
    <objName>titles</objName>
```

```
    <scanType>TableScan</scanType>
    <partitionInfo>
        <partitionCount>1</partitionCount>
    </partitionInfo>
    <scanOrder> ForwardScan </scanOrder>
    <positioning> StartOfTable </positioning>
    <dataIOSizeInKB>8</dataIOSizeInKB>
    <dataBufReplStrategy> LRU </dataBufReplStrategy>
</TableScan>
```

以 XML 格式显示高速缓存计划

show_cached_plan_in_xml 可帮助跟踪语句高速缓存中的查询性能。对于给定查询， show_cached_plan_in_xml （由其对象 ID 或者 SSQLID 和 PlanID 标识）返回以下内容：

- 标题节，其中包含有关高速缓存语句的信息，例如语句 ID、对象 ID 和文本：

```
<?xml version="1.0" encoding="UTF-8"?>
<query>
    <statementId>1328134997</statementId>
<text>
<![CDATA[SQL Text: select name from sysobjects where id = 10]]>
</text>
```

  如果 PlanID 设置为 0， show_cached_plan_in_xml 会显示与高速缓存语句关联的所有可用计划的输出。

- 计划节，其中包含计划 ID 和以下小节：
  - 参数 — 返回计划状态，用于编译查询的参数以及导致最低性能的参数值。

```
<planId>11</planId>
<planStatus> available </planStatus>
<execCount>1371</execCount>
<maxTime>3</maxTime>
<avgTime>0</avgTime>
<compileParameters/>
<execParameters/>
```

  - opTree — 返回运算符树、行计数以及每个运算符的逻辑 I/O (lio) 和物理 I/O (pio) 估计值。 opTree 小节返回查询计划和优化程序估计值，例如 lio、 pio 和行计数。

以下是 Emit 运算符的输出示例。

```
<opTree>
      <Emit>
      <VA>1</VA>
      <est>
          <rowCnt>10</rowCnt>
          <lio>0</lio>
          <pio>0</pio>
          <rowSz>22.54878</rowSz>
      </est>
      <act>
          <rowCnt>1</rowCnt>
      </act>
      <arity>1</arity>
          <IndexScan>
              <VA>0</VA>
              <est>
                  <rowCnt>10</rowCnt>
                  <lio>0</lio>
                  <pio>0</pio>
                  <rowSz>22.54878</rowSz>
              </est>
              <act>
                  <rowCnt>1</rowCnt>
                  <lio>3</lio>
                  <pio>0</pio>
              </act>
              <varNo>0</varNo>
              <objName>sysobjects</objName>
              <scanType>IndexScan</scanType>
              <indName>csysobjects</indName>
              <indId>3</indId>
              <scanOrder> ForwardScan </scanOrder>
              <positioning> ByKey </positioning>
              <perKey>
                  <keyCol>id</keyCol>
                  <keyOrder> Ascending </keyOrder>
              </perKey>
              <indexIOSizeInKB>2</indexIOSizeInKB>
              <indexBufReplStrategy> LRU </indexBufReplStrategy>
              <dataIOSizeInKB>2</dataIOSizeInKB>
              <dataBufReplStrategy> LRU </dataBufReplStrategy>
           </IndexScan>
      </Emit>
  <opTree>
```

- execTree — 返回查询计划以及运算符内部详细信息。根据运算符的不同，详细信息也有所不同。以下是 Emit 运算符的输出示例。

```
    <Emit>
    <Details>
    <VA>5</VA>
        <Vtuple Label="Output Vtuple">
        <collection Label="Columns (#2)">
        <Column>
        <0x0x1462d2838) type:GENERIC_TOKEN len:0 offset:0
valuebuf:0x(nil) status:(0x00000008 (STATNULL))
(constant:0x0x1462d24c0 type:INT4 len:4 maxlen:4 constat:(0x0004
(VARIABLE), 0x0002 (PARAM)))
        </Column>
        <Column>
        (0x0x1462d2878) type:GENERIC_TOKEN len:0 offset:0
valuebuf:0x(nil) status:(0x00000008 (STATNULL))
(constant:0x0x1462d26e8 type:INT4 len:4 maxlen:4 constat:(0x0004
(VARIABLE), 0x0002 (PARAM))
        </Column>
        <Collection>
        <Collection Label="Evals">
        <EVAL>
        constp:0x0x1462d2290 status:0 E_ASSIGN
        </EVAL>
        <EVAL>
        constp:0x0x1462d2348 status:0 E_ASSIGN
        </EVAL>
        <EVAL>
        constp:0x(nil) status:0 E_END
        </EVAL>
        </Collection>
        </Vtuple>
    </Details>
```

诊断使用情景

对于以下示例，如果设置 dbcc traceon(3604)，则会将跟踪信息发送到客户端连接。如果设置 dbcc traceon (3605)，则会将跟踪信息发送到错误日志。对于 Adaptive Server 15.0.2 版及更高版本，您可以使用 set switch on。例如：

```
set switch on 3604
set switch on 3605
```

Adaptive Server 15.0 之前的版本中的优化跟踪选项 (dbcc traceon/off(302,310,317)) 不再受支持。

使用 dbcc traceon(3604) 或 set switch on print_output_to_client 可将本应进入错误日志的跟踪输出发送到客户端进程。使用 dbcc traceon(3605) 或 set switch on print_output_to_errorlog 可将跟踪输出发送到错误日志以及客户端进程。

情景 A

若要以跟踪输出的形式将执行计划 XML 发送到客户端，请使用：

```
set plan for show_exec_xml to client on
```

然后运行需要计划的查询：

```
select id from sysindexes where id < 0
```

情景 B

若要获得执行计划，请使用 showplan_in_xml 函数。您可以从批处理或存储过程中的最后一个查询或前 20 个查询中的任何一个查询获得输出。

```
set plan for show_opt_xml to  message on
```

按以下方式运行查询：

```
select id from sysindexes where id < 0
select name from sysobjects where id > 0
go

select showplan_in_xml(0)
go
```

该示例生成两个 XML 文档作为文本流。只要在 Adaptive Server 中启用 XML 选项，您就可以通过此内置函数运行 XPath 查询。

```
select xmlextract("/", showplan_in_xml(-1))
go
```

这将允许对最后一个查询所生成的 XML 文档运行 XPath 查询 “/”。

情景 C

若要设置多个选项，请使用：

```
set plan for show_exec_xml, show_opt_xml to client on
go

select name from sysobjects where id > 0
go
```

这会设置来自优化程序和查询执行引擎中的输出，以将结果发送至客户端，就像在常规跟踪中所进行的操作一样。

```
set plan for show_exec_xml off
go
select name from sysobjects where id > 0
go
```

因为 show_opt_xml 保留为打开，所以优化程序的诊断信息仍然可用。

情景 D

在批处理中运行一组查询时，您可以请求用于最后一个查询的优化程序计划。

```
set plan for show_opt_xml to message on
go
declare @v int
select @v = 1
select name from sysobjects where id = @v
go

select showplan_in_xml(-1)
go
```

showplan_in_xml() 也可以是同一批处理的一部分，因为其运行方式是一样的。不记录有关 showplan_in_xml() 函数的任何消息。

若要创建存储过程，请使用：

```
create proc PP as
declare @v int
select @v = 1
select name from sysobjects where id = @v
go

exec PP
go

select showplan_in_xml(-1)
go
```

如果该存储过程调用了另一个存储过程，被调用的存储过程进行了编译，并且优化程序诊断已打开，那么您也会获得用于新语句集的优化程序诊断。如果已打开 show_execio_xml 并且仅执行被调用的存储过程，情况也是如此。

情景 E

若要查询 XML 文档形式的查询执行计划的 showplan_in_xml() 函数输出，请使用：

```
set plan for show_exec_xml to message on
go

select name from sysobjects
go

select  case when
'/Emit/Scan[@Label="Scan:myobjectss"]' xmltest
showplan_in_xml(-1)
then "PASSED" else "FAILED" end
go

set plan for show_exec_xml off
go
```

情景 F

使用 show_final_plan_xml 可以配置 Adaptive Server 将查询计划显示为 XML 输出。此输出不包括实际的 LIO 开销、 PIO 开销或行计数。启用 show_final_plan_xml 后，便可以从最近运行的查询 （查询 ID 为 -1）选择查询计划。要启用 show_final_plan_xml，请使用：

```
set plan for show_final_plan_xml to message on
```

运行您的查询，例如：

```
use pubs2
go
select * from titles
go
```

使用 showplan_in_xml 参数选择最近运行的查询的查询计划：

```
select showplan_in_xml(-1)
```

set 命令的权限

sa_role 拥有对上述 set 命令的完全访问权限。

对于其他用户，系统管理员必须授予和撤消新的 set tracing 权限，才能允许用于 XML 的 set option 和 set plan，以及 dbcc traceon/off （3604 和 3605）。

有关详细信息，请参见 《Adaptive Server 参考手册：命令》中的 grant 命令说明。

分析动态参数

利用 Adaptive Server，您可以在运行查询之前先分析动态参数 （用问号指示），从而帮助您避免生成低效率的查询计划。

使用以下项分析动态参数：

- @@*lwpid* 全局变量 — 返回与动态 SQL prepare 语句对应的最近准备的轻量过程的对象 ID。
- @@*plwpid* 全局变量 — 返回与动态 SQL prepare 语句对应的下一个最近准备的轻量过程的对象 ID。
- show_dynamic_params_in_xml — 显示动态 SQL 语句中的参数的相关信息。请参见 《参考手册：构件块》。

通过将 @@*plwpid* 提供的值用作 show_dynamic_params_in_xml *object_id* 参数的值， Adaptive Server 可显示有关查询中的动态参数的信息。继续细化参数，直至找到能提供最佳查询计划的参数。

在分析查询的动态参数之前禁用语句高速缓存。语句高速缓存会重用查询计划来避免编译。

show_dynamic_params_in_xml 的输出类似于：

```
<!ELEMENT query (parameter*)>
<!ELEMENT parameter (number, type, column?)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT column (#PCDATA)>
```

文档的根元素是 <query>，它可以有零个或多个 <parameter> 元素。每个 <parameter> 元素均包括：

- number — 动态参数在语句中的位置 （从 1 开始）。
- type — 数据类型。
- column — 与动态参数关联的表和列的名称 （采用 *table.column* 格式）（如果存在此关联）。如果不存在任何关联，则输出中不显示任何列信息。

动态参数示例分析

分析动态参数的典型用例是：

1 禁用语句高速缓存：

```
set statement_cache off
```

2 准备要检查的包含动态参数的语句。

3 选择 @@*plwpid* 以确定最新轻量过程的对象 ID。

4 运行 show_dynamic_params_in_xml，将 @@*plwpid* 的值用作 *object_id* 的值，并在名为 *xmldoc1* 的局部变量中显示结果。

5 关闭该语句。

6 启用语句高速缓存：

```
set statement_cache on
```

7 分析 *xmldoc1* 中的结果，并为这些参数选择值。

第 4 章

查找慢速运行的查询

Adaptive Server 包括 set show_sqltext、set tracefile 和 set export_options 参数，您可以使用这些参数收集有关运行速度慢的查询的诊断信息，而无需事先启用 showplan 或其它检查参数。

将诊断信息保存到跟踪文件中

启用跟踪功能后， set tracefile 便会将当前会话的所有 SQL 文本保存到指定文件中，每个 SQL 文本批处理都会附加到上一个批处理之后。

启用跟踪的语法为：

set tracefile *file_name* [off] [for *spid*

禁用跟踪的语法为：

set tracefile off [for *spid*]

其中：

- *file_name* — 是保存 SQL 文本的文件的完整路径。如果不指定目录路径， Adaptive Server 将在 *$SYBASE* 中创建该文件。

  注释 如果 *file_name* 包含数字和字母之外的特殊字符 （“:”、“/”等），则必须用引号将 *file_name* 引起来。例如，以下 *file_name* 必须用引号引起来，因为目录结构中包含 “/”：

  ```
  set tracefile '/tmp/mytracefile.txt' for 25
  ```

  如果 *file_name* 不包含特殊字符，并且您要将其保存到 *$SYBASE* 中，则不需要用引号将其引起来。例如，以下 *file_name* 不需要用引号引起来：

  ```
  set tracefile mytracefile.txt
  ```

- off — 为此会话或 spid 禁用跟踪。
- *spid* — 要将其 SQL 文本保存至跟踪文件的服务器进程 ID。只有具有 SA 或 SSO 角色的用户才能为其它 spid 启用跟踪。不能保存系统任务 （如管家或端口管理器）的 SQL 文本。

示例

- 此示例为当前会话打开一个名为 *sql_text_file* 的跟踪文件：

```
set tracefile '/var/sybase/REL1502/text_dir/sql_text_file'
```

来自 set showplan、 set statistics io 和 dbcc traceon(100) 的后续输出将保存到 *sql_text_file* 中。

- 此示例未指定目录路径，所以跟踪文件将保存到 *$SYBASE/sql_text_file* 中：

```
set tracefile 'sql_text_file' for 11
```

在 spid 11 上运行的任何 SQL 都将保存到该跟踪文件中。

- 此示例为 spid 86 保存 SQL 文本：

```
set tracefile
'/var/sybase/REL1502/text_dir/sql_text_file' for 86
```

- 此示例禁用 set tracefile：

```
set tracefile off
```

下面是对 set tracefile 的限制：

- 不能保存系统任务 （如管家或端口管理器）的 SQL 文本。
- 您必须具有 sa 或 sso 角色，或被授予 set tracing 权限，才能运行启用或禁用跟踪。
- 不允许 set tracefile 将现有文件打开以作为跟踪文件。
- 在进行 SA 或 SSO 会话期间，如果为特定 spid 启用 set tracfile，则后续执行的所有跟踪命令都将对该 spid 而不对 SA 或 SSO spid 生效。
- 如果 Adaptive Server 在写入跟踪文件时文件空间不足，它将关闭文件并禁用跟踪。
- 如果 isql 会话为某个 spid 启动跟踪，但 isql 会话在未禁用跟踪的情况下退出，则另一个 isql 会话可以开始跟踪此 spid。
- 跟踪只在为其启用跟踪的会话中进行，而不在启用跟踪的会话中进行。
- 不能从一个 sa 或 sso 会话中同时跟踪多个会话。如果试图为已打开一个跟踪文件的会话再打开一个跟踪文件， Adaptive Server 将发出错误消息：`tracefile is already open for this session.`

- 不能从多个 sa 或 sso 会话中跟踪同一会话。
- 当被跟踪的会话退出或禁用跟踪时，存储跟踪输出的文件会关闭。
- 在为跟踪分配资源之前，请记住，所有跟踪都要求每个引擎具有一个文件描述符。

设置将诊断信息保存到跟踪文件的选项

可将 set tracefile 与其它可提供诊断信息以便更好地了解慢速查询的 set 命令和选项一起使用。以下是用于将诊断信息保存到文件的 set 命令和选项：

- set show_sqltext [on | off]
- set showplan [on | off]
- set statistics io [on | off]
- set statistics time [on | off]
- set statistics plancost [on | off]

以下是 set 选项：

- set option show [normal | brief | long | on | off]
- set option show_lop [normal | brief | long | on | off]
- set option show_parallel [normal | brief | long | on | off]
- set option show_search_engine [normal | brief | long | on | off]
- set option show_counters [normal | brief | long | on | off]
- set option show_managers [normal | brief | long | on | off]
- set option show_histograms [normal | brief | long | on | off]
- set option show_abstract_plan [normal | brief | long | on | off]
- set option show_best_plan [normal | brief | long | on | off]
- set option show_code_gen [normal | brief | long | on | off]
- set option show_pio_costing [normal | brief | long | on | off]
- set option show_lio_costing [normal | brief | long | on | off]
- set option show_log_props [normal | brief | long | on | off]
- set option show_elimination [normal | brief | long | on | off]

正在跟踪哪些会话？

使用 sp_helpapptrace 可以确定 Adaptive Server 正在跟踪哪些会话。sp_helpapptrace 会返回 Adaptive Server 正在跟踪的所有会话的服务器进程 ID (spid)、对其进行跟踪的会话的 spid 以及跟踪文件的名称。

sp_helpapptrace 的语法为：

```
sp_helpapptrace
```

sp_helpapptrace 会返回以下列：

- traced_spid — 您正在跟踪的会话的 spid。
- tracer_spid — traced_spid 正在跟踪的会话的 spid。如果 tracer_spid 会话已退出，则输出 “exited”。
- trace_file — 跟踪文件的完整路径。

例如：

```
sp_helpapptrace
 traced_spid   tracer_spid         trace_file
--------------   -------------     ----------
11               exited             /tmp/myfile1
13               14                 /tpcc/sybase.15_0/myfile2
```

重新绑定跟踪

如果一个会话正在跟踪另一个会话，但该会话在未禁用跟踪的情况下退出， Adaptive Server 将允许新会话与早期的跟踪重新绑定。这意味着，不需要 sa 或 sso 就能完成它们启动的每个跟踪，但可以启动跟踪会话、退出，然后再重新绑定到此跟踪会话

显示 SQL 文本

set show_sqltext 可用于为特定查询、存储过程、游标和动态准备的语句输出 SQL 文本。在执行查询以收集 SQL 会话的诊断信息之前，无需启用 set show_sqltext （与 set showplan on 等命令的处理方式相同）。相反，可以在命令运行时启用它，以帮助确定未正常执行的查询并诊断其存在的问题。

在启用 show_sqltext 之前，必须先启用 dbcc traceon，以使输出显示为标准输出：

```
dbcc traceon(3604)
```

set show_sqltext 的语法为：

```
set show_sqltext {on | off}
```

例如，以下语句将启用 show_sqltext：

```
set show_sqltext on
```

启用 set show_sqltext 后， Adaptive Server 便会将您输入的每个命令或系统过程的所有 SQL 文本都输出为标准输出。根据运行的命令或系统，此输出可进行扩展。

例如，如果运行 sp_who， Adaptive Server 将输出与此系统过程关联的所有 SQL 文本 （为了节省空间，输出采用缩写形式）：

```
sp_who
2007/02/23 02:18:25.77
SQL Text: sp_who
Sproc: sp_who, Line: 0
Sproc: sp_who, Line: 20
Sproc: sp_who, Line: 22
Sproc: sp_who, Line: 25
Sproc: sp_who, Line: 27
Sproc: sp_who, Line: 30
Sproc: sp_who, Line: 55
Sproc: sp_who, Line: 64
Sproc: sp_autoformat, Line: 0
Sproc: sp_autoformat, Line: 165
Sproc: sp_autoformat, Line: 167
Sproc: sp_autoformat, Line: 177
Sproc: sp_autoformat, Line: 188
. . .
Sproc: sp_autoformat, Line: 326
Sproc: sp_autoformat, Line: 332
SQL Text: INSERT
#colinfo_af(colid,colname,usertype,type,typename,collength,maxlength,autoform
```

```
at,selected,selectorder,asname,mbyte) SELECT
c.colid,c.name,t.usertype,t.type,t.name,case when c.length < 80 then 80 else
c.length end,0,0,0,0,c.name,0 FROM tempdb.dbo.syscolumns c,tempdb.dbo.systypes
t WHERE c.id=1949946031 AND c.usertype=t.usertype
Sproc: sp_autoformat, Line: 333
Sproc: sp_autoformat, Line: 334
. . .
Sproc: sp_autoformat, Line: 535
Sproc: sp_autoformat, Line: 0
Sproc: sp_autoformat, Line: 393
Sproc: sp_autoformat, Line: 395
. . .
Sproc: sp_autoformat, Line: 686
Sproc: sp_autoformat, Line: 688
SQL Text: UPDATE #colinfo_af SET maxlength=(SELECT
isnull(max(isnull(char_length(convert(varchar(80),fid)),4)),1) FROM
#who1result ), autoformat = 1, mbyte=case when usertype in (24, 25, 34, 35) then
1 else 0 end WHERE colname='fid'
Sproc: sp_autoformat, Line: 689
Sproc: sp_autoformat, Line: 690
. . .
Sproc: sp_autoformat, Line: 815
Sproc: sp_autoformat, Line: 818
SQL Text: SELECT
fid=right(space(80)+isnull(convert(varchar(80),fid),'NULL'),3),
spid=right(space(80)+isnull(convert(varchar(80),spid),'NULL'),4),
status=SUBSTRING(convert(varchar(80),status),1,8),
loginame=SUBSTRING(convert(varchar(80),loginame),1,8),
origname=SUBSTRING(convert(varchar(80),origname),1,8),
hostname=SUBSTRING(convert(varchar(80),hostname),1,8),
blk_spid=right(space(80)+isnull(convert(varchar(80),blk_spid),'NULL'),8),
dbname=SUBSTRING(convert(varchar(80),dbname),1,6),
tempdbname=SUBSTRING(convert(varchar(80),tempdbname),1,10),
cmd=SUBSTRING(convert(varchar(80),cmd),1,17),
block_xloid=right(space(80)+isnull(convert(varchar(80),block_xloid),'NULL'),1
1) FROM #who1result  order by fid, spid, dbname



Sproc: sp_autoformat, Line: 819
Sproc: sp_autoformat, Line: 820
Sproc: sp_autoformat, Line: 826
Sproc: sp_who, Line: 68
Sproc: sp_who, Line: 70
fid  spid  status   loginame     origname    hostname  blk_spid  dbname
tempdbnamecmd      block_xloid
```

```
---  ----  --------  --------   --------   --------  --------    -------------
------------       -----------
0    2     sleeping  NULL       NULL       NULL       0        master tempdb
DEADLOCK TUNE      0
0    3     sleeping  NULL       NULL       NULL       0        master tempdb
ASTC HANDLER       0
0    4     sleeping  NULL       NULL       NULL       0        master tempdb
CHECKPOINT SLEEP   0
0    5     sleeping  NULL       NULL       NULL       0        master tempdb
HK WASH            0
0    6     sleeping  NULL       NULL       NULL       0        master tempdb
HK GC           0
0    7     sleeping  NULL       NULL       NULL       0        master tempdb
HK CHORES          0
0    8     sleeping  NULL       NULL       NULL       0        master tempdb
PORT MANAGER       0
0    9     sleeping  NULL       NULL       NULL       0        master tempdb
NETWORK HANDLER    0
0    10    sleeping  NULL       NULL       NULL       0        master tempdb
LICENSE HEARTBEAT  0
0    1     running   sa         sa         echo       0        master tempdb
INSERT             0

(10 rows affected)
(return status = 0)
```

若要禁用 show_sqltext，请输入：

```
set show_sqltext off
```

对 show_sqltext 的限制

- 您必须具有 sa 或 sso 角色才能运行 show_sqltext。
- 不能使用 show_sqltext 为触发器输出 SQL 文本。
- 不能使用 show_sqltext 显示绑定变量或视图名。

保留会话设置

Adaptive Server 的缺省行为是在触发器或系统过程运行完成后重置它们设置的任何 set 参数更改。通过启用 set export_options，您可以将系统过程或触发器设置的会话设置导出到存储过程或触发器的父项 （或发出方）。

在本例中， Adaptive Server 将 set showplan on 导出到 outer_proc，而不将其导出到父存储过程：

```
create proc inner_proc
as
    set showplan on
    select * from titles
    set export_options on
     go

    create proc outer_proc
    as
    exec inner_proc
    go
```

第 5 章

并行查询处理

本章将对并行查询处理进行深入说明。

垂直、水平和管道并行度

Adaptive Server 支持在查询执行过程中应用水平和垂直并行度。垂直并行度是指利用不同系统资源 （如 CPU、磁盘等）同时运行多个运算符的能力。水平并行度是指对数据的指定部分运行同一运算符的多个实例的能力。

数据的分区方式会极大地影响水平并行度的效率。在可操作的决策支持系统 (DSS) 查询中，由于要处理大量的数据，因此对数据进行逻辑分区将非常有用。

有关在 Adaptive Server 上进行分区的详细论述，请参见 《Transact-SQL 用户指南》中的第 10 章 “对表和索引进行分区”以及 《性能和调优系列：物理数据库调优》指南。了解不同类型的分区是理解本章的前提条件。

Adaptive Server 还支持管道并行度。管道是垂直并行度的一种形式，在该形式中，中间结果将传递给查询树中更高级的运算符。一个运算符的输出将用作另一个运算符的输入。用作输入的运算符可与提供数据的运算符同时运行，这是管道并行度的一个重要元素。仅当有多种资源（如磁盘和 CPU）可用时，才应使用并行度。如果您的系统未配置为多项资源可串联工作，使用并行度将带来负面影响。另外，数据跨磁盘资源分布的方式必须为：数据的逻辑分区与并行设备上的物理分区紧密相联。并行系统的最大挑战是如何正确控制并行度的粒度。如果并行度过于精细，则通信和同步开销将抵消从并行操作中获得的所有益处。而并行度过于粗糙又会影响适当的缩放。

从并行处理获益的查询

将 Adaptive Server 配置为采用并行查询处理时，查询优化程序将评估每个查询，以确定它是否适合并行执行。如果适合，并且优化程序确定并行查询计划能比串行计划更快地输出结果，则查询将被分成几个计划片段以进行同时处理。结果组合在一起并传递给客户端，所需时间比将查询作为单个片段进行串行处理的时间短。

并行查询处理可提高以下项的性能：

- 扫描大量页但只返回相对很少行的 select 语句，例如使用分组或拆组集合的表扫描或聚簇索引扫描。
- 扫描大量页、但具有只返回一小部分行的 where 子句的表扫描或聚簇索引扫描。
- 包含 union、 order by 或 distinct 的 select 语句，原因是这些查询操作可利用并行排序或并行散列。
- 由优化程序选择重新格式化策略的 select 语句，因为这些语句能以并行模式填充工作表并可利用并行排序。
- join 查询。

受网络限制，返回大型无序结果集的命令可能不会从并行处理中受益。多数情况下，从数据库返回结果的速度比在合并结果后通过网络返回给客户端的速度要快。

不支持并行 DML，例如 insert、 delete 和 update，因此它们不会从并行度中获益。

启用并行度

要配置 Adaptive Server 以应用并行度，请启用 number of worker processes 和 max parallel degree 参数。

要获得优化的性能，请了解其它哪些配置参数还会影响 Adaptive Server 生成的计划的质量。

number of worker processes

在启用并行度之前，请先通过设置配置参数 number of worker processes 来配置 Adaptive Server 的可用工作进程（也称为线程）数。Sybase 建议将 number of worker processes 的值设置为高峰负载时所需总数的 1.5 倍。您可以使用 max parallel degree 配置参数来计算近似数，该数目表示可用于任何查询的工作进程总数。根据到 Adaptive Server 的连接数和要同时运行的近似查询数，您可以使用以下规则，来粗略估计可能随时需要的工作进程数的值：

[number of worker processes] = [max parallel degree] X [要并行运行查询的并发连接数] X [1.5]

例如，要将工作进程数设置为 40：

```
sp_configure "number of worker processes", 40
```

对线程数的任何运行期调整都可能对查询性能产生负面影响。 Adaptive Server 始终尝试优化线程使用，但它可能已提交到需要增加资源的计划，因此它在使用较少的线程来运行时无法保证线性递减。

如果查询处理器的工作进程不够，则该处理器将在运行期间尝试调整查询计划。如果需要最小工作进程数而又没有，则查询将中止，并显示以下错误消息：

```
用于执行并行查询的工作进程数不够。增大配置参数 “number of
worker processes” 的值
```

max parallel degree

使用 max parallel degree 配置参数可配置查询并行度的最大量。此参数确定 Adaptive Server 在处理给定查询时所使用的最大线程数。例如，要将 max parallel degree 设置为 10，请输入：

```
sp_configure "max parallel degree", 10
```

与 Adaptive Server 15.0 之前的版本不同，查询优化程序并不完全强制此参数值。整个强制过程将消耗大量优化时间。 Adaptive Server 接近所需的 max parallel degree 设置，并仅在语义方面需要时才会超过该值。

max resource granularity

max resource granularity 的值配置查询可以使用的系统资源的最大百分比。在 Adaptive Server version 15.0 和更高版本中， max resource granularity 只影响过程高速缓存。缺省情况下， max resource granularity 为 10%。但是，执行时并不强制使用此值；它只是作为查询优化程序的指导。将 max resource granularity 设置为较低的值时，查询引擎可避免内存密集型策略（例如，基于散列的算法）。

要将 max resource granularity 设置为 5%，请输入：

```
sp_configure "max resource granularity", 5
```

如果查询处理器的搜索引擎已消耗超过配置的过程高速缓存百分比，且如果它已找到至少一个完整计划，则搜索引擎超时并为查询使用当前最佳计划。

如果在达到 max resource granularity 的值（以过程高速缓存的百分比形式表示）之前，查询处理器找不到完整计划，则搜索引擎会继续搜索，直到它找到下一个完整计划。但是，如果搜索引擎达到完整过程高速缓存的 50% 且找不到计划，它会中止查询编译以避免关闭服务器。

max repartition degree

Adaptive Server 必须对中间数据进行动态重新分区，以与其它操作数的分区方案相匹配或执行有效的分区排除。 max repartition degree 控制 Adaptive Server 可执行的动态重新分区量。如果 max repartition degree 的值过高，中间分区数将过大，系统会被争用资源的工作进程所占用，并最终导致性能下降。 max repartition degree 的值将强制使用为所有中间数据创建的最大分区数。重新分区是一种 CPU 密集型操作。max repartition degree 的值不应超过 Adaptive Server 引擎的总数。

如果所有表和索引均未分区， Adaptive Server 将在对数据重新分区后使用 max repartition degree 的值来提供要创建的分区数。当该值设置为 1（缺省设置）时， max repartition degree 的值将设置为联机引擎的数目。

当使用 force 选项对表或索引执行并行扫描时，使用 max repartition degree。

```
select * from customers (parallel)
```

例如，如果 customers 表未分区，且使用了 force 选项，则 Adaptive Server 将尝试找出该表或索引的固有分区度 （本示例中为 1）。它使用为服务器配置的引擎数，或基于表或索引中页数的最佳程度，但不超过 max repartition degree 的值。

要将 max repartition degree 设置为 5：

```
sp_configure "max repartition degree", 5
```

max scan parallel degree

当分区表或索引中的数据高度倾斜时， max scan parallel degree 配置参数仅用于向后兼容性。如果此参数的值大于 1， Adaptive Server 将使用此值执行基于散列的扫描。max scan parallel degree 的值不能超过 max parallel degree 的值。

prod-consumer overlap factor

prod-consumer overlap factor 会影响可在查询计划中创建的管道并行度的量。缺省值为 20%，这意味着如果父子关系中的两个运算符由单独的工作进程运行，则会有 20% 的重叠。其余 80% 的操作是有序的。这会影响 Adaptive Server 使用两个计划片段的方式。请考虑分组操作下的扫描运算符示例。在此情况下，如果扫描运算符需要 *N1* 秒，而分组运算符需要 *N2* 秒，则两个运算符的响应时间为：

0.2 * max (N1, N2) + 0.8 * (N1 + N2)

设置此参数时，请考虑 Adaptive Server 在其上运行的联机引擎的数量以及要运行的查询的复杂性。一般情况下，首先使用线程资源对多个分区进行扫描。然后，如果有未使用的线程资源，将其用于加快垂直管道并行度。值不要超过 50。

min pages for parallel scan

max pages for parallel scan 控制可并行访问的表和索引。如果表中的页数低于此值，将以串行方式访问该表。 sqm_page_size 的缺省值为 200。虽然表和表索引是以串行方式访问的，但 Adaptive Server 会尝试在适合的情况下对数据进行重新分区，并在适合的情况下以并行方式进行扫描。

max query parallel degree

max query parallel degree 定义要用于给定查询的工作进程数。只有当您不需要以全局方式启用并行时，此参数才适用。您必须将工作进程数配置为大于 0 的值，但 max query parallel degree 必须设置为 1。

当 max query parallel degree 设置为大于 1 的值时，不会以并行方式编译查询。不过，您可以指定并行提示，使用抽象计划以并行方式编译一个或多个查询。

使用 use parallel *N* 可定义要用于给定查询的并行度的量。或者，还可以用 create plan 指定查询和要用于查询的工作进程数。

在会话级控制并行度

可使用 set 选项限定会话期间、存储过程或触发器中的并行度。这些选项可用于并行查询的调优试验，也可用于限定非关键查询以串行模式运行，以便留出工作进程供其它任务使用。

表 5-1：会话级并行度控制参数

参数	功能
parallel_degree	为会话、存储过程或触发器中的查询设置最大工作进程数。它替换 max parallel degree 配置参数，但必须小于或等于 max parallel degree 的值。
scan_parallel_degree	为特定会话、存储过程或触发器期间某个基于散列的扫描设置最大工作进程数。它覆盖 max scan parallel degree 配置参数且必须小于或等于 max scan parallel degree 的值。
resource_granularity	覆盖全局值 max resource granularity 并将其设置为会话特定的值，这将对 Adaptive Server 是否使用内存密集型操作产生影响。
repartition_degree	为会话设置 max repartition degree 的值。它是出于语义需要而对任何中间数据流重新分区的最大程度。

如果您指定的值对任一 set 选项来说都过大，则会使用相应配置参数的值，并显示一条消息来报告实际使用的值。如果会话期间， set parallel_degree、 set scan_parallel_degree、 set repartition_degree 或 set resource_granularity 生效，则所执行的任何存储过程的计划均不会放在过程高速缓存中。这些 set 选项生效时执行的过程可能会生成次佳的计划。

set 命令示例

本例将所有当前会话中开始的查询限定到 5 个工作进程：

```
set parallel_degree 5
```

此命令生效时，多于 5 个分区的表上的任何查询都不能使用基于分区的扫描。

要去除会话限制，可使用：

```
set parallel_degree 0
```

或：

```
set scan_parallel_degree 0
```

要以串行模式运行后续查询，可使用：

```
set parallel_degree 1
```

或：

```
set scan_parallel_degree 1
```

要将资源粒度设置为系统中可用资源总量的 25%，请使用：

```
set resource_granularity 25
```

这对于 repartition degree 来说也是如此；您可以将它的值设置为 5。但是，该值不能超过 max parallel degree 的值。

```
set repartition_degree 5
```

控制查询并行度

对 select 命令 from 子句的 parallel 扩展用于设置 select 语句中所用的工作进程数。指定的并行度不能大于使用 sp_configure 设置的值或由 set 命令控制的会话限制。如果指定的值大于它们，则指定值无效，优化程序使用 set 或 sp_configure 的限制值。

select 语句的语法为：

```
select ...
from tablename [( [index index_name]
[parallel [degree_of_parallelism | 1 ]]
[prefetch size] [lru|mru] ) ] ,
tablename [( [index index_name]
[parallel [degree_of_parallelism | 1]
[prefetch size] [lru|mru] ) ] ...
```

查询级 parallel 子句示例

若要指定单个查询的并行度，可在表名后加上 `parallel`。下例以串行模式执行：

```
select * from huge_table (parallel 1)
```

下例指定在查询中要使用的索引，并将并行度设置成 2：

```
select * from huge_table (index ncix parallel 2)
```

有选择地使用并行度

并非所有查询都会从并行度中获益。一般情况下，优化程序决定不会从并行度中获益的查询并尝试以串行方式运行它们。如果查询处理器在此类情况下出现错误，通常是由于统计信息出现偏差或是因不完善建模导致的错误开销。经验会告诉您查询运行得更好还是更差，您可以决定打开还是关闭 parallel。

如果您打开 parallel，且已标识要以串行模式运行的查询，则可附加抽象计划提示，如下所示：

```
select count(*) from sysobjects
plan "(use parallel 1)"
```

通过创建查询计划可实现相同的效果：

```
create plan "select count(*) from sysobjects"
"use parallel 1"
```

但是，如果您注意到并行度是资源密集型的或者它不会生成执行正常的查询计划，可选择性地使用它。要为所选的复杂查询启用并行度：

1　根据第 125 页的 “number of worker processes”中的规则将工作进程数设置为大于 0 的数。例如，要配置 10 个工作进程，请执行：

```
sp_configure "number of worker processes", 10
```

2　将 max query parallel degree 设置为大于 1 的值。开始时，将其设置为将用于 max parallel degree 的值。

```
sp_configure "max query parallel degree", 10
```

3　强制查询使用并行计划的首选方法是使用抽象计划语法：

use parallel *N*

其中， *N* 小于 max query parallel degree 的值。

要编写最多使用 5 个线程的查询，请使用：

```
select count (*), S1.id from sysobjects S1, sysindexes S2
where S1.id = S2.id
group by S1.id
plan
"(use parallel 5)"
```

此查询指示优化程序使用 5 个工作进程 （如果优化程序可以这样做）。此方法的唯一缺点是您必须改变应用程序中的实际查询。要避免此问题，请使用 create plan：

```
create plan
"select count(*), S1.id from sysobjects S1, sysindexes S2
where S1.id = S2.id
group by S1.id"
"(use parallel 5)"
```

要在全局范围内打开 abstract plan load 选项，请输入：

sp_configure “abstract plan load”, 1

有关使用抽象计划的详细信息，请参见第 297 页的 “创建和使用抽象计划”。

将并行度用于大量分区

在配置分区的可管理性时，以及在表现出很少或没有表现出并行的物理或逻辑设备上创建分区时，本节中的信息同样适用。

为了进行本次讨论，您已决定使用域分区 （代表一年的每周）对表进行分区。此处的问题是查询优化程序不了解基础磁盘系统响应 52 向并行扫描的方式。优化程序必须决定扫描表的最佳方式。如果已配置足够的工作进程，则优化程序将使用 52 个线程扫描表，这可能会导致严重的性能问题且甚至可能比串行扫描还要慢。

为防止出现此情况，首先要查明支持的并行度的确切量。如果您了解用于此表的设备，则可在 UNIX 系统上使用以下命令，其中基础设备称为 */dev/xx*：

```
time dd if=/dev/xx of=/dev/null bs=2k skip=8 count = 102400 &
```

假定时间记录为 *x*。

现在同时运行两个相同命令：

```
time dd if=/dev/xx of=/dev/null bs=2k skip=8 count = 102400 &
time dd if=/dev/xx of=/dev/null bs=2k skip=8 count = 102400 &
```

此时，假定时间为 *y*。在线性递增时， *x* 等同于 *y*，这可能无法实现。以下等式足以说明：

x <= *y* <= (*N**x)/*k*

其中，*N* 是同时启动的会话数，而 *k* 是标识可接受改进水平的常量。*k* 的较好近似值可以为 1.4，这意味着只要并行扫描提供的指标有 40% 优于串行扫描所提供的，即允许并行扫描。

表 5-2：并行扫描指标

线程数	性能指标	可接受 k=1.4
1	200s	
2	245s	245 <= (200*2)/1.4 ； 即 245<=285.71
4	560s	560 <= (200*4)/1.4 ； 即 560<=571.42
5	725s	725 <= (200*5)/1.4 ； 即 725<=714.28

表 5-2 表明磁盘子系统无法在四个并发访问后正常执行；性能数值降至由 *k* 建立的可接受限制以下。一般情况下，会读取足够的数据块以允许任何出现偏差的读数。

确定 4 个线程为最佳后，通过将该提示按以下方式使用 sp_chgattribute 绑定到对象来提供该提示：

sp_chgattribute <*tablename*>, “plldegree”, 4

这指示查询优化程序最多使用 4 个线程。如果查询优化程序找不到足够的资源，可选择使用较少的线程 （即少于 4 个）。此机制同样适用于索引。例如，如果名为 auth_ind 的索引位于 authors 上且您希望使用两个线程对它进行访问，请使用：

```
sp_chgattribute “authors.auth_ind”, “plldegree”, 4
```

必须从当前数据库中运行 sp_chgatttribute。

并行查询结果不同的情况

如果某查询不包括标量集合，或不需要最终的排序步骤，则对于这一查询，并行执行和串行运行所返回的结果的顺序可能不同，并且对这一查询的后续并行执行所返回的结果的顺序也会不同。不同工作进程的相对速度会导致结果集顺序不同。由于高速缓存中的页及锁争用等情况的不同，各并行扫描的运行也会不同。并行查询返回的结果集始终是相同的，只是顺序不同。

注释　如果需要一个可靠的结果顺序，可使用 order by 或以串行模式运行查询。

此外，由于多工作进程读取数据页的频率影响，在未完成集合或最终排序时，下列两类查询访问同一数据可能返回不同的结果。这两个参数为：

- 使用 set rowcount 的查询
- 将列选进某个局部变量而不使用充分限制查询子句的查询

使用 *set rowcount* 的查询

在向客户端返回一定数量的行后， set rowcount 选项会停止继续处理。使用串行处理时，只要查询计划相同，重复执行所返回的结果都将是一致的。在串行模式下，假定使用同一查询计划，则对于给定的 rowcount 值，会以同一顺序返回同样的行，因为单个进程每次是以同一顺序读取数据页的。使用并行查询时，所返回的结果和行集的顺序可能不同，这是因为工作进程与其它进程访问页的时间不同。要获得一致的结果，可使用执行最终排序步骤的子句，或以串行模式运行查询。

设置局部变量的查询

以下查询在 select 语句中设置一个局部变量的值：

```
select @tid = title_id from titles
where type = "business"
```

由于 where 子句与 titles 表中的多个行匹配，因此局部变量的值始终根据查询所返回的最后一个匹配行进行设置。串行处理中该值始终是相同的，但对于并行查询处理，其结果取决于最后结束的工作进程。要获得一致的结果，可使用一个执行最终排序步骤的子句，或以串行模式运行查询，或增加子句以使查询参数只选择单个的行。

了解并行查询计划

要了解 Adaptive Server 中的并行查询处理，关键在于了解并行查询计划中的基本构件块。

注释 请参见第 2 章 “使用 showplan”，本章介绍如何采用基于文本的格式显示批处理或存储过程中的每个 SQL 语句的查询计划。

已编译的查询计划包含一个执行运算符树，这些运算符与查询的关系语义极为相似。每个查询运算符都使用特定的算法实现一种关系操作。例如，称为 nested-loop join 的查询运算符可实现关系 join 操作。在 Adaptive Server 中，并行度的主运算符为 exchange 运算符 （是一个不会实现任何关系操作的控制运算符）。 exchange 运算符用于创建可处理数据片段的新工作进程。在优化期间， Adaptive Server 会有策略地放置 exchange 运算符，以创建可并行运行的运算符树片段。位于 exchange 运算符 （直到下一个 exchange 运算符）下的所有运算符都将由复制该运算符树片段的工作线程执行，以并行生成数据。 exchange 运算符随后可将该数据重新分配给它在查询计划中的上一级父运算符。 exchange 运算符处理数据的管道输送和重新路由。

在以下几节中，“度”这个字将用于两个不同的上下文。当提到表或索引的 “N 度”时，它是指包含在表或索引中的分区数。当提到 “某项操作的度”或 “配置参数的度”时，它是指中间数据流中生成的分区数。

以下示例显示了查询处理器中的运算符如何与在 pubs2 数据库中运行的以下查询串行工作。表 titles 对列 pub_id 应用三路散列分区。

```
select * from titles
QUERY PLAN FOR STATEMENT 1 (at line 1).

1 operator(s) under root


The type of query is SELECT.

ROOT:EMIT Operator

    |SCAN Operator
    |  FROM TABLE
    |  titles
    |  Table Scan.
    |  Forward Scan.
    |  Positioning at start of table.
    |  Using I/O Size 2 Kbytes for data pages.
    |  With LRU Buffer Replacement Strategy for data
       pages.
```

正如本例所述， titles 表由 scan 运算符扫描，该运算符的详细信息显示在 showplan 输出中。 emit 运算符从 scan 运算符读取数据并将其发送给客户端应用程序。一个给定查询可创建由此类运算符构成的具有任意复杂度的树。

启用并行度时，Adaptive Server 可以通过在 scan 运算符之上使用 exchange 运算符来执行简单的并行扫描。 exchange 会产生三个工作进程 （基于三个分区），每个工作进程都会扫描表中三个不相交的部分，并将输出发送给消耗程序进程。位于树顶部的 emit 运算符不会识别该扫描是并行执行的。

示例 A：

```
   select * from titles
Executed in parallel by coordinating process and 3 worker processes.

4 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

    |EXCHANGE Operator (Merged)
    |Executed in parallel by 3 Producer and 1 Consumer processes.
```

```
|
|    |EXCHANGE:EMIT Operator
|    |
|    |    |RESTRICT Operator
|    |    |
|    |    |    |SCAN Operator
|    |    |    |  FROM TABLE
|    |    |    |  titles
|    |    |    |  Table Scan.
|    |    |    |  Forward Scan.
|    |    |    |  Positioning at start of table.
|    |    |    |  Executed in parallel with a 3-way partition scan.
|    |    |    |  Using I/O Size 2 Kbytes for data pages.
|    |    |    |  With LRU Buffer Replacement Strategy for data pages.
```

称作 EXCHANGE:EMIT 的运算符被放在 EXCHANGE 运算符之下，用于传送数据。请参见第 136 页的 “EXCHANGE 运算符”。

Adaptive Server 并行查询执行模型

并行查询执行模型的一个重要组成部分是 EXCHANGE 运算符。您可以从查询的 showplan 输出中查看它。

EXCHANGE 运算符

EXCHANGE 运算符标记生产者运算符和使用者运算符间的边界（EXCHANGE 运算符之下的运算符产生数据，而位于它之上的运算符则消耗数据）。示例 A 显示了对 titles 表进行并行扫描 (`select * from titles`)，EXCHANGE: EMIT 和 SCAN 运算符产生数据。下例对此进行了简要说明。

```
select * from titles

The type of query is SELECT.

ROOT:EMIT Operator

    |EXCHANGE Operator (Merged)
    |Executed in parallel by 3 Producer and 1 Consumer
           processes.
    |
```

```
|     |EXCHANGE:EMIT Operator
|     |
|     |     |RESTRICT Operator
|     |     |
|     |     |     |SCAN Operator
|     |     |     |  FROM TABLE
|     |     |     |  titles
|     |     |     |  Table Scan.
```

在本例中，一个消耗程序进程从管道 （用作跨进程边界传送数据的介质）读取数据，并将数据传递给 emit 运算符，该运算符又会将结果发送至客户端。 exchange 运算符还生成工作进程，这些进程称为生产者线程。 exchange:emit 运算符将数据写入由 exchange 运算符管理的管道。

图 5-1：将线程绑定到查询计划中的计划片段

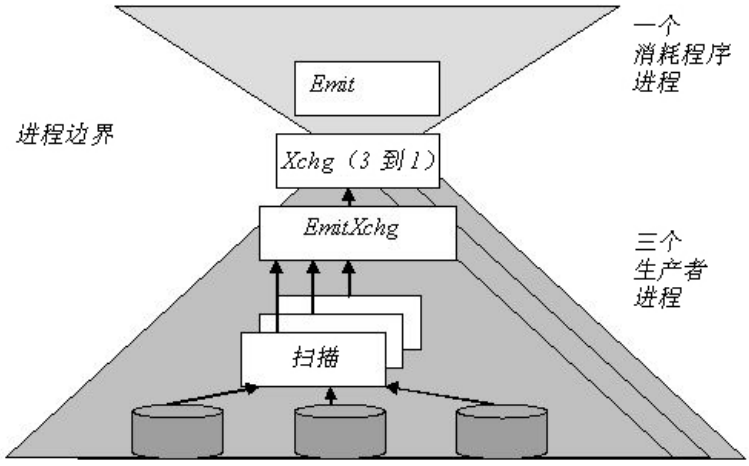


图 5-1 显示了生产者进程和消耗程序进程之间的进程边界。该查询计划中有两个计划片段。包含 scan 和 exchange:emit 运算符的计划片段通过三路复制，并由三对一的 exchange 运算符将其写入管道。emit 运算符和 exchange 运算符由单个进程运行，这意味着仅对该计划片段复制一次。

管道管理

exchange 运算符管理四类管道，这些管道按照其拆分和合并数据流的方式进行区分。通过在 showplan 输出 （其中显示了生产者和使用者的数目）中查看 exchange 运算符的说明，可以确定由该运算符管理的管道类型。下面对这四种管道类型进行了说明。

多对一

在此情况下， exchange 运算符将生成多个生产者线程来向一个管道写入数据，并只有一个使用者任务读取该管道中的数据。在上一示例中，exchange 运算符实现的就是多对一的 exchange。多对一的 exchange 运算符可以保留原有顺序，这一技术尤其适用于以下情况：为 order by 子句执行并行排序，并且结果数据流被合并，以生成最终顺序。 showplan 输出显示多个生产者进程以及一个消耗程序进程。

```
|EXCHANGE Operator (Merged)
        |Executed in parallel by 3 Producer and 1
         Consumer processes
```

一对多

在此情况下，存在一个生产者线程和多个使用者线程。生产者线程根据查询优化时设计的分区方案将数据写入多个管道中，然后将数据传送给各个管道。每个使用者线程都从一个分配的管道读取数据。这种数据拆分方式可保留数据的顺序。 showplan 输出显示一个生产者进程以及多个消耗程序进程：

```
|EXCHANGE Operator (Repartitioned)
      |Executed in parallel by 1 Producer
        and 4 Consumer processes
```

多对多

“多对多”意味着存在多个生产者和多个使用者。每个生产者向多个管道写入数据，而每个管道又拥有多个使用者。每个流都将写入管道。每个使用者线程都从一个分配的管道读取数据。

```
 |EXCHANGE Operator (Repartitioned)
    |Executed in parallel by 3 Producer and 4
      Consumer processes
```

复制型 exchange 运算符

在此情况下，生产者线程将它的所有数据写入 exchange 运算符配置的每个管道中。生产者线程会生成源数据的大量副本 （该数量由查询优化程序指定），其数目等于 exchange 运算符中的管道数。每个使用者线程都从一个分配的管道读取数据。 showplan 输出显示了这一情况，如下所示：

```
|EXCHANGE (Replicated)
    |Executed in parallel by 3 Producers and 4
     Consumer processes
```

工作进程模式

并行查询计划由多个不同的运算符组成，其中至少有一个 exchange 运算符。在运行期，并行查询计划会绑定到一组服务器进程上，这组进程会一起以并行方式执行该查询计划。

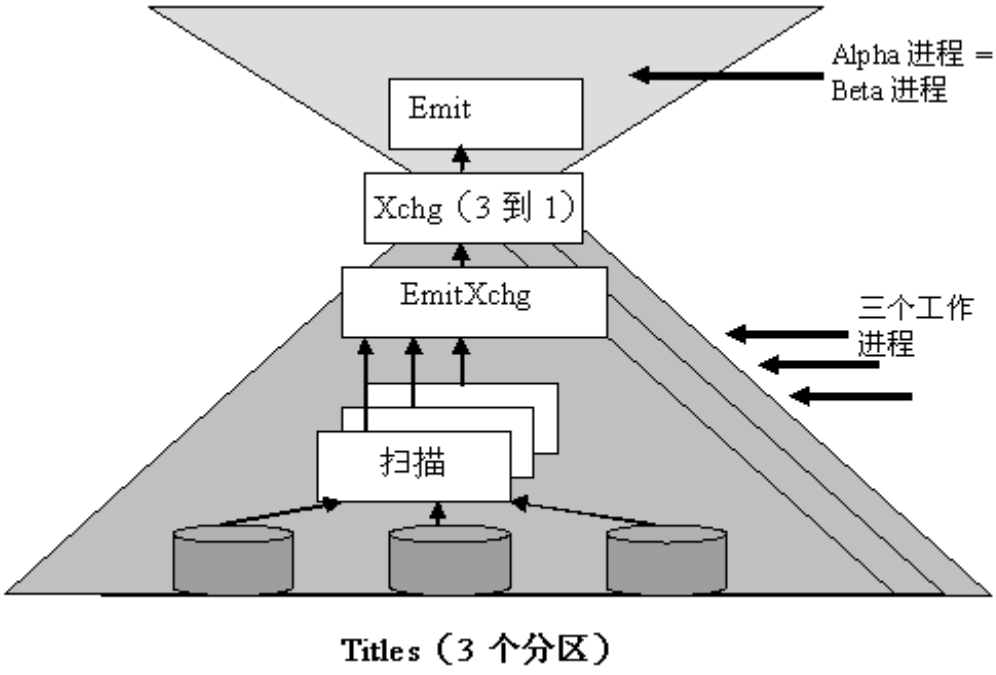
与用户连接相关联的服务器进程称为 *alpha 进程*，这是因为它是启动并行执行的源进程。具体来说，执行并行查询计划时所涉及的每个工作进程都是由 alpha 进程生成的。

除生成工作进程外， alpha 进程还初始化计划执行过程中所涉及的所有工作进程。此外，它还创建并损坏工作进程交换数据所必需的管道。alpha 进程实际上是执行并行查询计划的全局事务协调器。

在运行期， Adaptive Server 将计划中的每个 exchange 运算符与一组工作进程相关联。这些工作进程会执行位于 exchange 运算符正下方的查询计划片段。

对于示例 A （如第 136 页的 “EXCHANGE 运算符”中所述）中的查询，exchange 运算符与 3 个工作进程相关联。每个工作进程都会执行由 exchange:emit 运算符及 scan 运算符构成的计划片段。

图 5-2：包含一个 exchange 运算符的查询执行计划

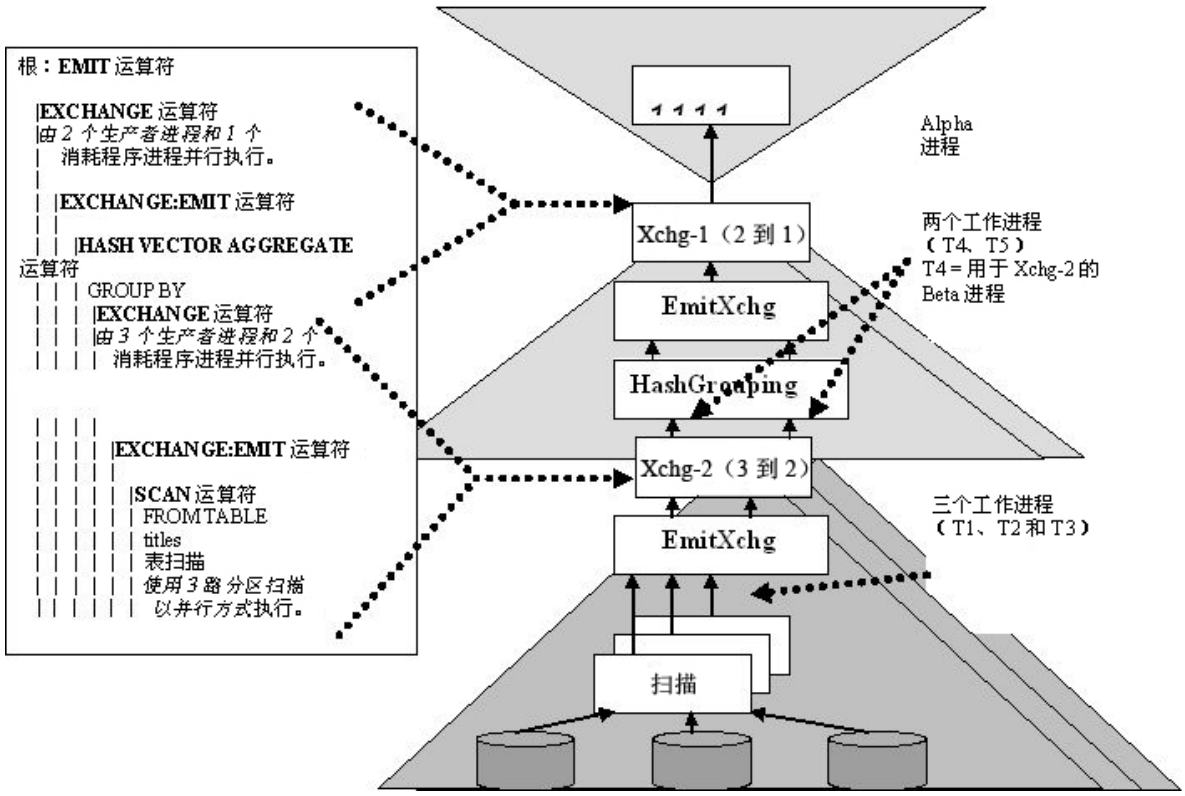


每个 exchange 运算符也与一个服务器进程相关联，该进程称为 *beta 进程*，它可以是 alpha 进程，也可以是工作进程。与给定的 exchange 运算符相关联的 beta 进程是执行 exchange 运算符下的计划片段的局部事务协调器。在上一示例中， beta 进程与 alpha 进程相同，原因是要执行的计划仅包含一个级别的 exchange 运算符。

接下来，我们将使用此查询来说明当查询计划中包含多个 exchange 运算符时的情形：

```
select count(*),pub_id, pub_date
    from titles
group by pub_id, pub_date
```

图 5-3：包含两个 exchange 运算符的查询执行计划



在图 5-3 中，存在两级 exchange 运算符，分别标记为 EXCHANGE-1 和 EXCHANGE-2。工作进程 T4 是与 exchange 运算符 EXCHANGE-2 相关联的 beta 进程。

beta 进程可在局部协调 exchange 运算符下的计划片段的执行；它分派工作进程所需的查询计划信息，并同步计划片段的执行。

除 alpha 进程和 beta 进程外，在并行查询计划的执行过程中，还会涉及另一种进程，称为 *gamma 进程*。

给定的并行查询计划在运行期会绑定到唯一的 alpha 进程、一个或多个 beta 进程以及至少一个 gamma 进程。任何 Adaptive Server 并行计划都需要至少两个不同的进程 （alpha 和 gamma）才能并行执行。

要找出 exchange 运算符与工作进程之间的映射，并确定哪个进程是 alpha 进程以及哪些进程是 beta 进程，请使用 dbcc traceon(516)：

```
=======Thread to XCHg Map BEGINS=======
ALFA thread spid: 17
XCHG = 2                          <- refers to Xchg-2
Comp Count = 2 Exec Count = 2
     Range Adjustable
   Consumer XCHG = 5
   Parent thread spid: 34         <- refers to T4
     Child thread 0: spid: 37      <- refers to T1
     Child thread 1: spid: 38     <- refers to T2
     Child thread 2: spid: 36     <- refers to T3
   Scheduling level: 0
XCHG = 5                          <- refers to Xchg-1
   Comp Count = 3 Exec Count = 3
Bounds Adjustable
   Consumer XCHG -1
   Parent thread spid:17        <- refers to Alpha
     Child thread 0: spid: 34    <- refers to T4
     Child thread 1: spid: 35    <- refers to T5

Scheduling level: 0
=======Thread to XCHg Map BEGINS=======
```

在 SQL 操作中使用并行度

可使用最能反映应用程序的需要的任意方式来对表或索引进行分区。Sybase 建议将分区置于使用不同物理磁盘的多个段上，以便提供足够的 I/O 并行度。例如，您可以基于表中某些列的散列、某些范围或属于某一分区的值列表来创建明确定义的分区。散列、范围和列表分区属于“基于语义的”分区类别，在给定某行的情况下，可以确定该行所属的分区。

循环分区与其分区没有任何语义关联。一个行可能出现在它的任何一个分区中。所选择的要分区的列与使用的分区类型会对应用程序的性能产生重大影响。分区可以视为基数较低的索引；必须对其定义分区的列将基于应用程序中的查询。

查询处理引擎及其运算符可以利用 Adaptive Server 的分区策略。对表和索引定义的分区称为静态分区。此外， Adaptive Server *动态地*对数据进行重新分区，以满足像连接、矢量集合、区分、联合等关系操作的需要。重新分区以流模式执行，且不会与任何存储相关联。重新分区与发出 alter table repartition 命令不同，后者执行的是静态分区。

一个查询计划由多个查询执行运算符组成。在 Adaptive Server 中，运算符分为两类：

- 不区分属性的运算符包括 scans、 union alls 和 scalar aggregation。基础分区不会影响不区分属性的运算符。
- 区分属性的运算符 （例如， join、 distinct、 union 和 vector aggregation），它们允许使用基于语义的分区，将针对给定数量的数据的一项操作分为多个较小的操作，以作用于该数据的多个较小的片段。之后，只需使用 union all 便可提供最终的结果集。 union all 通过执行多对一的 exchange 运算符实现。

以下几节将讨论这两类运算符，其中的示例将使用下表并通过足够的数据来触发并行处理。

```
create table RA2(a1 int, a2 int, a3 int)
```

不区分属性的操作的并行度

本节讨论不区分属性的操作，包括 scan（串行和并行）、scalar aggregation 和 union all。

表扫描

对于水平并行度，必须至少对查询中的一个表进行分区，或者配置参数 max repartition degree 的值必须大于 1。如果将 max repartition degree 设置为 1， Adaptive Server 将提示使用联机引擎的数目。当 Adaptive Server 运行水平并行度时，它将并行运行一个或多个运算符的多个版本。运算符的每个副本都在其分区上运行，该分区可通过静态方式创建，也可在执行过程中动态生成。

串行表扫描

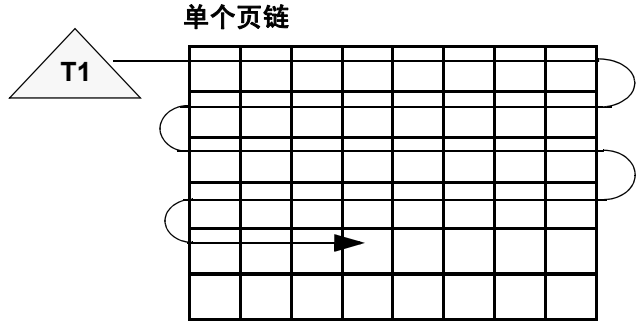
下面的示例显示查询的串行执行，其中，将使用 table scan 运算符对表 RA2 进行扫描。此操作的结果将传送给 emit 运算符，该运算符又会将结果转移到客户端。

```
select * from RA2
QUERY PLAN FOR STATEMENT 1 (at line 1).

1 operator(s) under root

The type of query is SELECT.
```

```
ROOT:EMIT Operator

     |SCAN Operator
     |  FROM TABLE
     |  RA2
     |  Table Scan.
     |  Forward Scan.
     |  Positioning at start of table.
     |  Using I/O Size 2 Kbytes for data pages.
     |  With LRU Buffer Replacement Strategy for data
        pages.
```

在 15.0 之前的版本中，除非使用 force 选项，否则它不会尝试使用基于散列的扫描来并行扫描未分区的表。图 5-4 显示了由单个任务 T1 在串行模式下对所有页锁定表执行的扫描。如果执行物理 I/O 时所需页不在缓存中，该任务会沿表的页链读取各页。

图 5-4：串行任务扫描数据页



并行表扫描

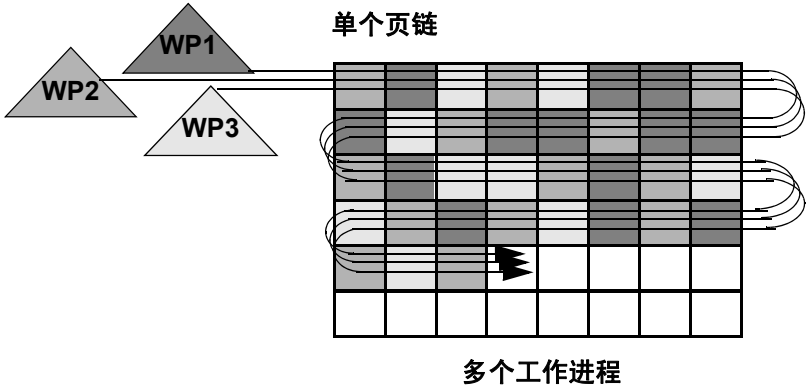
您可以使用 Adaptive Server force 选项对未分区的表强制并行表扫描。在此情况下， Adaptive Server 使用基于散列的扫描。

基于散列的表扫描

图 5-5 显示了在基于散列的表扫描期间，三个工作进程如何分工，共同访问所有页锁定表的数据页。每个工作进程对每页都执行一次逻辑 I/O，但只检查三分之一页上的行，如图中不同阴影行所示。仅当用户强制并行度时，才可使用基于散列的表扫描。请参见第 183 页的 “分区倾斜”。

即使只有一个引擎，使用并行访问也能使查询受益，因为在其它工作进程等待 I/O 时可以有一个进程在执行。如果存在多个引擎，则某些工作进程可同时运行。

图 5-5：多个工作进程扫描未分区的表



基于散列的扫描会增加扫描的逻辑 I/O，因为各工作进程必须访问每个页才能按页 ID 实现散列。而对于仅数据锁定表，基于散列的扫描既可按扩充 ID 也可按分配页 ID 实现散列，这样将只有一个工作进程扫描页，从而不会增加逻辑 I/O。

基于分区的表扫描

如果您按以下方式对该表进行分区：

```
alter table RA2 partition by range(a1, a2)
(p1 values <= (500,100), p2 values <= (1000, 2000))
```

使用以下查询， Adaptive Server 可能选择对表进行并行扫描。仅当存在足够多的页以进行扫描，并且各个分区大小都很相似，足以使查询从并行度中获益时，才可选择并行扫描。

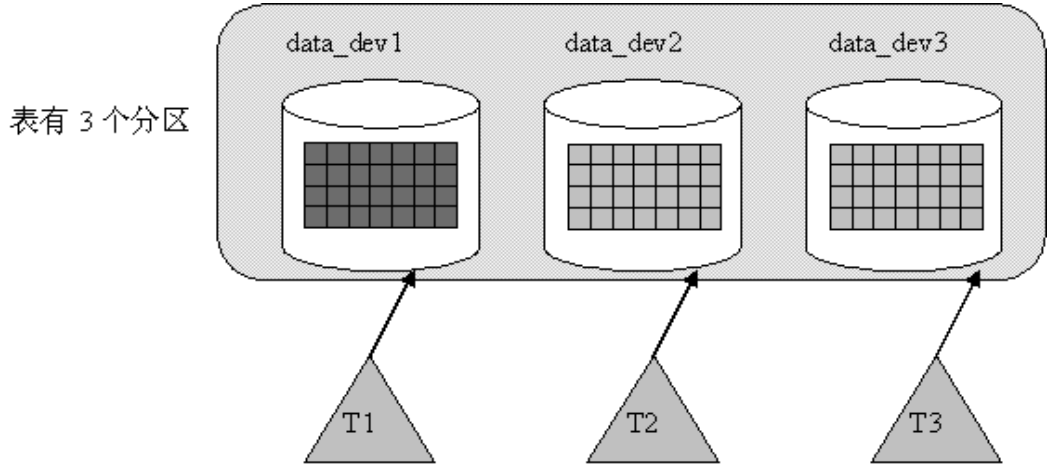
```
select * from RA2

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2
worker processes.

3 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

     |EXCHANGE Operator (Merged)
     |Executed in parallel by 2 Producer and 1 Consumer
          processes.
```

```
|
|    |EXCHANGE:EMIT Operator
|    |
|    |    |SCAN Operator
|    |    |  FROM TABLE
|    |    |  RA2
|    |    |  Table Scan.
|    |    |  Forward Scan.
|    |    | Positioning at start of table.
|    |    | Executed in parallel with a 2-way
|    |    |    partition scan.
|    |    | Using I/O Size 2 Kbytes for data pages.
|    |    |  With LRU Buffer Replacement Strategy
|    |    |    for data pages.
```

对表进行分区后，showplan 输出将包括两个附加的运算符：exchange 和 exchange:emit。该查询包括两个工作进程，每个工作进程都扫描给定的分区，并将数据传递给 exchange:emit 运算符，如第 137 页的图 5-1 中所示。

图 5-6 显示了查询如何针对某个在三个物理磁盘上有三个分区的表进行扫描。即使只有一个引擎，使用并行处理也能使查询获益，因为在其它工作进程休眠等待 I/O 或等待其它进程将其所持有的锁释放时，可以有一个工作进程在执行。如果有多个引擎可用，则工作进程便可同时在多个引擎上运行。此类配置可获得极佳的执行效果。

图 5-6：多工作进程访问多个分区



索引扫描

与表类似，索引也可以是分区的，也可以是未分区的。本地索引继承表的分区策略。每个本地索引分区仅扫描一个分区中的数据。全局索引的分区策略不同于基表的分区策略，它们引用一个或多个分区。

全局非聚簇索引

Adaptive Server 在所有表分区策略中都支持未分区的非聚簇全局索引。支持全局索引，以与 Adaptive Server 15.0 之前的版本兼容；全局索引还适用于 OLTP 环境。索引和数据分区可驻留在相同的存储区域中，也可驻留在不同的存储区域中。

使用散列对全局非聚簇索引执行非覆盖扫描

要在按范围分区的表 RA2 上创建未分区的全局非聚簇索引，请输入：

```
create index RA2_NC1 on RA2(a3)
```

此查询包含使用索引键 a3 的谓词：

```
select * from RA2 where a3 > 300
QUERY PLAN FOR STATEMENT 1 (at line 1).
. . . . . .
The type of query is SELECT.

ROOT:EMIT Operator

        |EXCHANGE Operator (Merged)
        |Executed in parallel by 3 Producer and 1
         Consumer processes.
        |
        |   |EXCHANGE:EMIT Operator
        |   |
        |   |   |SCAN Operator
        |   |   | FROM TABLE
        |   |   | RA2
        |   |   | Index : RA2_NC1
        |   |   | Forward Scan.
        |   |   | Positioning by key.
        |   |   | Keys are:
        |   |   | a3 ASC
        |   |   | Executed in parallel with a 3-way
                    hash scan.
        |   |   | Using I/O Size 2 Kbytes for index
                    leaf pages.
        |   |   | With LRU Buffer Replacement Strategy
                    for index leaf pages.
        |   |   | Using I/O Size 2 Kbytes for data
                    pages.
        |   |   | With LRU Buffer Replacement Strategy
                    for data pages.
```

Adaptive Server 应用的索引扫描使用索引 RA2_NC1，该索引使用由 exchange 运算符生成的三个生产者线程。每个生产者线程都将扫描所有的限定叶页，对限定数据的行 ID 应用散列算法并访问它所属的数据页。在此示例中，并行度出现在数据页级上。

图 5-7：对全局非聚簇索引执行基于散列的并行扫描

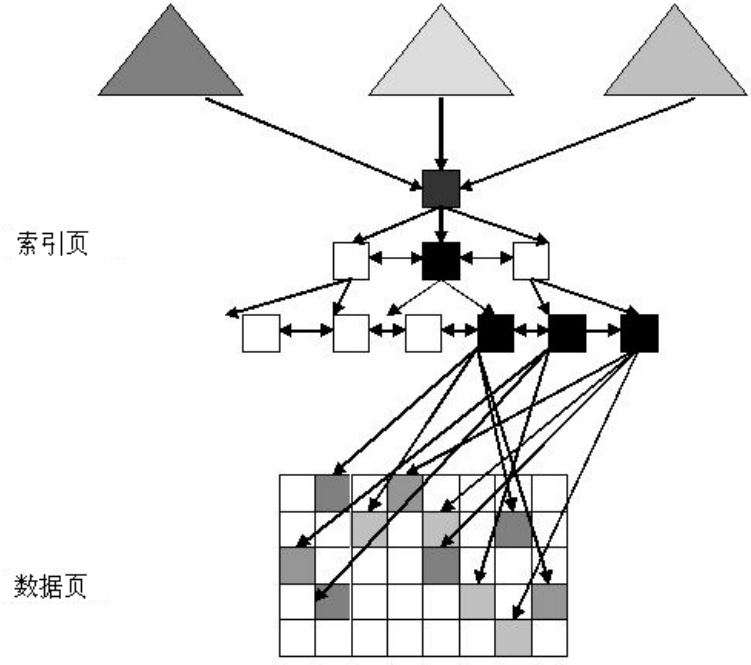
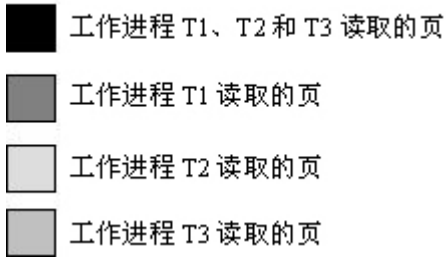


图 5-7 的图例：



如果查询不需要访问数据页，则不会并行执行查询。但是，必须向查询中添加分区列；因此，它将成为一个非覆盖扫描：

```
select a3 from RA2 where a3 > 300

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2
worker processes.
3 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

    |EXCHANGE Operator (Merged)
    |Executed in parallel by 2 Producer and 1 Consumer
processes.

    |
    |   |EXCHANGE:EMIT Operator
    |   |
    |   |   |SCAN Operator
    |   |   |  FROM TABLE
    |   |   |  RA2
    |   |   |  Index : RA2_NC1
    |   |   |  Forward Scan.
    |   |   |  Positioning by key.
    |   |   |  Keys are:
    |   |   |    a3 ASC
    |   |   |  Executed in parallel with a 2-way hash
                 scan.
    |   |   |  Using I/O Size 2 Kbytes for index leaf
                 pages.
    |   |   |  With LRU Buffer Replacement Strategy for
                index leaf pages.
    |   |   |  Using I/O Size 2 Kbytes for data pages.
    |   |   |  With LRU Buffer Replacement Strategy for
                data pages.
```

使用非聚簇全局索引的覆盖扫描

如果存在包含分区列的非聚簇索引，Adaptive Server 则无需访问数据页，且查询将以串行模式执行：

```
create index RA2_NC2 on RA2(a3,a1,a2)

select a3 from RA2 where a3 > 300

QUERY PLAN FOR STATEMENT 1 (at line 1).
```

```
1 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

     | SCAN Operator
     | FROM TABLE
     | RA2
     | Index : RA2_NC2
     |  Forward Scan.
     |  Positioning by key.
     |  Index contains all needed columns. Base table
        will not be read.
     |  Keys are:
     |    a3 ASC
     |  Using I/O Size 2 Kbytes for index leaf pages.
     |  With LRU Buffer Replacement Strategy for index
        leaf pages.
```

聚簇索引扫描

对所有页锁定表设置聚簇索引后，将不允许使用基于散列的扫描策略。唯一允许的策略为分区扫描。 Adaptive Server 将在必要时使用分区扫描。对于仅数据锁定表，聚簇索引通常是一个位置索引，它与非聚簇索引的行为相似。所有关于所有页锁定表上的非聚簇索引的论述也同样适用于仅数据锁定表上的聚簇索引。

本地索引

Adaptive Server 支持聚簇和非聚簇本地索引。

分区表上的聚簇索引

本地聚簇索引允许多个线程并行扫描每个数据分区，因而可极大地提高性能。要利用此并行度，可使用分区聚簇索引。对于本地索引，每个分区中的数据都是单独排序的。每个数据分区中的信息都符合创建分区时建立的边界，因此可以在整个表中实现唯一索引键。

唯一的聚簇本地索引具有下列限制：

- 索引列必须包含所有分区列。
- 分区列的顺序必须与索引定义的分区键相同。
- 唯一的聚簇本地索引不能包含在具有多个分区的循环表中。

分区表上的非聚簇索引

Adaptive Server 支持分区表上的本地非聚簇索引。

但是，在使用本地索引时仍有些许不同。当对本地非聚簇索引执行覆盖索引扫描时，由于索引页也已经过分区，因此 Adaptive Server 仍可使用并行扫描。

为说明其中的差异，下面的示例将创建一个本地非聚簇索引：

```
create index RA2_NC2L on RA2(a3,a1,a2) local index

select a3 from RA2 where a3 > 300

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2
worker processes.

3 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

     |EXCHANGE Operator (Merged)
     |Executed in parallel by 2 Producer and 1 Consumer
processes.

     |
     |    |EXCHANGE:EMIT Operator
     |    |
     |    |    |SCAN Operator
     |    |    |  FROM TABLE
     |    |    |  RA2
     |    |    |  Index : RA2_NC2L
     |    |    |  Forward Scan.
     |    |    |  Positioning by key.
     |    |    |  Index contains all needed columns. 基本功能
                      table will not be read.
     |    |    |  Keys are:
     |    |    |    a3 ASC
     |    |    |  Executed in parallel with a 2-way
                   partition scan.
     |    |    |  Using I/O Size 2 Kbytes for index leaf
                      pages.
     |    |    |  With LRU Buffer Replacement Strategy
                      for index leaf pages.
```

有时， Adaptive Server 会选择对本地索引执行基于散列的扫描。当需要不同的并行度时，或当分区中的数据存在倾斜，以致应首选基于散列的并行扫描时，便会发生这种情况。

标量集合

Transact-SQL 标量集合操作可以串行模式执行，也可并行执行。

两阶段标量集合

在并行标量集合中，集合操作使用两个标量集合运算符分两阶段执行。在第一阶段中，较低的标量集合运算符执行数据流的集合。通过使用多对一的 exchange 运算符，第一阶段中的标量集合结果将合并，该数据流将进行第二次集合。

对于 count(*) 集合，第二阶段的集合将对标量执行 sum。下一示例的 showplan 输出中重点介绍了这一情况。

```
select count(*) from RA2

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2
worker processes.

5 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

   |SCALAR AGGREGATE Operator
   |  Evaluate Ungrouped SUM OR AVERAGE AGGREGATE.
   |
   |    |EXCHANGE Operator (Merged)
   |    |Executed in parallel by 2 Producer and 1
            Consumer processes.

   |    |
   |    |    |EXCHANGE:EMIT Operator
   |    |    |
   |    |    |    |SCALAR AGGREGATE Operator
   |    |    |    |  Evaluate Ungrouped COUNT AGGREGATE.
   |    |    |    |
   |    |    |    |    |SCAN Operator
   |    |    |    |    |  FROM TABLE
   |    |    |    |    |  RA2
   |    |    |    |    |  Table Scan.
   |    |    |    |    |  Forward Scan.
   |    |    |    |    |  Positioning at start of table.
   |    |    |    |    |  Executed in parallel with a
                              2-way partition scan.
   |    |    |    |    |  Using I/O Size 2 Kbytes for data
```

```
                                pages.
    |   |   |   |   |   With LRU Buffer Replacement
                              Strategy for data pages.
```

串行集合

Adaptive Server 也可以选择以串行模式执行集合。如果要集合的数据量不足，无法保证提高性能，则串行集合可能成为首选技术。在执行串行集合时，扫描结果将使用多对一的 exchange 运算符进行合并。下面的示例对此进行了说明，其中还增加了一个选择性的谓词，用于最小化流入标量集合运算符的数据量。在此情况下，并行执行集合可能会毫无意义。

```
select count(*) from RA2 where a2 = 10

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2
worker processes.

4 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

    |SCALAR AGGREGATE Operator
    |  Evaluate Ungrouped COUNT AGGREGATE.
    |
    |   |EXCHANGE Operator (Merged)
    |   |Executed in parallel by 2 Producer
            and 1 Consumer processes.

    |   |
    |   |   |EXCHANGE:EMIT Operator
    |   |   |
    |   |   |   |SCAN Operator
    |   |   |   |  FROM TABLE
    |   |   |   |  RA2
    |   |   |   |  Table Scan.
    |   |   |   |  Forward Scan.
    |   |   |   |  Positioning at start of table.
    |   |   |   |  Executed in parallel with a 2-way
                     partition scan.
    |   |   |   |  Using I/O Size 2 Kbytes for data
                     pages.
    |   |   |   |  With LRU Buffer Replacement
                     Strategy for data pages.
```

union all

union all 运算符使用同名的物理运算符来实现。 union all 是一种相当简单的操作，仅在查询要移动大量数据时才应并行使用它。

并行 *union all*

要生成并行的 union all，唯一的条件是它的操作数必须具有相同的度，而无论这些操作数具有何种分区类型。以下示例 （使用表 HA2）显示了要并行处理的 union all 运算符。 exchange 运算符的位置高于 union all 运算符，这表示它要由多个线程共同处理：

```
create table HA2(a1 int, a2 int, a3 int)
partition by hash(a1, a2) (p1, p2)

select * from RA2
union all
select * from HA2

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2
worker processes.

The type of query is SELECT.

ROOT:EMIT Operator

    |EXCHANGE Operator (Merged)
    |Executed in parallel by 2 Producer and 1 Consumer
     processes.

    |
    |   |EXCHANGE:EMIT Operator
    |   |
    |   |   |UNION ALL Operator has 2 children.
    |   |   |
    |   |   |   |SCAN Operator
    |   |   |   |  FROM TABLE
    |   |   |   |  RA2
    |   |   |   |  Table Scan.
. . . . . . . . . . . . . . . . . . . .
    |   |   |   |  Executed in parallel with a 2-way
                     partition scan.
. . . . . . . . . . . . . . . . . . . .
    |   |   |
    |   |   |   |SCAN Operator
    |   |   |   |  FROM TABLE
```

```
    |   |   |   |  HA2
    |   |   |   |  Table Scan.
    . . . . . . . . . . . . . . . . . .
    |   |   |   |  Executed in parallel with a 2-way
                   partition scan.
```

串行 *union all*

在下一示例中，来自 union 运算符每一侧的数据分别由每一侧的选择性谓词加以限制。要通过 union all 运算符发送的数据量将很少，足以使 Adaptive Server 决定不并行运行联合。相反，通过在 union 的每一侧都放置 2 对 1 的 exchange 运算符，将对表 RA2 和 HA2 的每次扫描进行组织。结果操作数随后由 union all 操作符并行处理：

```
select * from RA2
where a2 > 2400
union all
select * from HA2
where a3 in (10,20)
Executed in parallel by coordinating process and 4
worker processes.

7 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

     |UNION ALL Operator has 2 children.
     |
     |   |EXCHANGE Operator (Merged)
     |   |Executed in parallel by 2 Producer and 1
               Consumer processes.

     |   |
     |   |   |EXCHANGE:EMIT Operator
     |   |   |
     |   |   |   |SCAN Operator
     |   |   |   |  FROM TABLE
     |   |   |   |  RA2
     |   |   |   |  Table Scan.
     |   |   |   |  Executed in parallel with a 2-way
                      partition scan.
     |
     |   |EXCHANGE Operator (Merged)
     |   |Executed in parallel by 2 Producer and 1
```

```
                    Consumer processes.

|   |
|   |   |EXCHANGE:EMIT Operator
|   |   |
|   |   |   |SCAN Operator
|   |   |   |  FROM TABLE
|   |   |   |  HA2
|   |   |   |  Table Scan.
|   |   |   |  Executed in parallel with a 2-way
                  partition scan.
```

区分属性的操作的并行度

区分属性的操作包括连接、矢量集合和联合。

join

如果要并行连接两个表， Adaptive Server 会尝试使用基于语义的分区来提高连接操作的效率，具体取决于要连接的数据量和每个操作数所具有的分区类型。如果要连接的数据量很少，但要为每个表扫描的页数很大，则 Adaptive Server 将使每一侧的并行流串行化，从而以串行模式来运行连接。在此情况下，查询优化程序会确定并行运行连接操作是次优的。通常， join 运算符所使用的一个或两个操作数可能是任何中间操作数，这与其它 join 或分组运算符类似，但使用的示例仅将扫描显示为操作数。

具有相同的有用分区的表

每个连接操作数的分区的有用性仅与 join 谓词相关。如果两个表具有相同的分区，并且分区列是 join 谓词的子集，则可以说这两个表是等分区的。例如，如果您使用以下命令创建另一个表 RB2，使其分区类似于 RA2 的分区：

```
create table RB2(b1 int, b2 int, b3 int)
partition by range(b1,b2)
(p1 values <= (500,100), p2 values <= (1000, 2000))
```

然后将 RB2 和 RA2 连接起来，扫描和连接可并行执行，无需再进行重新分区。 Adaptive Server 可以连接 RA2 的第一个分区与 RB2 的第一个分区，然后再连接 RA2 的第二个分区与 RB2 的第二个分区。这称为等分区连接，仅当两个表在列 a1、 b1 和 a2、 b2 上连接时，才可以执行此连接，如下所示：

```
select * from RA2, RB2
where a1 = b1 and a2 = b2 and a3 < 0

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2
worker processes.
```

```
7 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

     |EXCHANGE Operator (Merged)
     |Executed in parallel by 2 Producer
          and 1 Consumer processes.

     |
     |    |EXCHANGE:EMIT Operator
     |    |
     |    |    |NESTED LOOP JOIN Operator
                    (Join Type:Inner Join)
     |    |    |
     |    |    |    |RESTRICT Operator
     |    |    |    |
     |    |    |    |    |SCAN Operator
     |    |    |    |    |  FROM TABLE
     |    |    |    |    |  RB2
     |    |    |    |    |  Table Scan.
     |    |    |    |    |  Forward Scan.
     |    |    |    |    |  Positioning at start of table.
     |    |    |    |    |  Executed in parallel with a
                              2-way partition scan.
     |    |    |
     |    |    |    |RESTRICT Operator
     |    |    |    |
     |    |    |    |    |SCAN Operator
     |    |    |    |    |  FROM TABLE
     |    |    |    |    |  RA2
     |    |    |    |    |  Table Scan.
     |    |    |    |    |  Forward Scan.
     |    |    |    |    |  Positioning at start of table.
     |    |    |    |    |  Executed in parallel with a
                              2-way partition scan.
```

exchange 运算符显示在 nested-loop join 之上。这意味着 exchange 将生成两个生产者线程：第一个将扫描 RA2 和 RB2 的第一个分区并执行 nested-loop join ； 第二个扫描 RA2 和 RB2 的第二个分区以执行 nested-loop join。这两个线程使用多对一 （本示例中为二对一）的 exchange 运算符来合并结果。

具有相同有用分区的表之一

以下示例使用 alter table 命令，将表 RB2 重新分区为 b1 列上的三路散列分区。

```
alter table RB2 partition by hash(b1) (p1, p2, p3)
```

现在，我们举一个稍加修改的 join 查询示例，如下所示：

```
select * from RA2, RB2 where a1 = b1
```

表 RA2 上的分区无用，因为分区列不是要连接的列的子集（即给定连接列 a1 的值后，无法指定它属于哪个分区）。但是，RB2 上的分区是有帮助的，因为它与 RB2 的连接列 b1 相匹配。在此情况下，查询优化程序将通过 RA2 中 a1 列（连接列，后跟一个三路的 merge join）上的散列分区，对表 RA2 重新分区，以与 RB2 的分区相匹配。位于 RA2 扫描之上的多对多（2 对 3）exchange 运算符将执行此动态重新分区。merge join 运算符之上的 exchange 运算符使用多对一（本示例中为 3 对 1）的 exchange 运算符来合并结果。此查询的 showplan 输出如以下示例所示：

```
select * from RA2, RB2 where a1 = b1

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 5
worker processes.

10 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator
   |EXCHANGE Operator (Merged)
   |Executed in parallel by 3 Producer and 1 Consumer
       processes.

   |
   |   |EXCHANGE:EMIT Operator
   |   |
   |   |   |MERGE JOIN Operator (Join Type: Inner
   |   |   |          Join)
   |   |   | Using Worktable3 for internal storage.
   |   |   |  Key Count: 1
   |   |   |  Key Ordering: ASC
   |   |   |
   |   |   |   |SORT Operator
   |   |   |   |Using Worktable1 for internal storage.
   |   |   |   |
   |   |   |   |   |EXCHANGE Operator (Repartitioned)
   |   |   |   |   |Executed in parallel by 2 Producer
   |   |   |   |        and 3 Consumer processes.
```

```
    |   |   |   |   |
    |   |   |   |   |   |EXCHANGE:EMIT Operator
    |   |   |   |   |   |
    |   |   |   |   |   |   |RESTRICT Operator
    |   |   |   |   |   |   |
    |   |   |   |   |   |   |   |SCAN Operator
    |   |   |   |   |   |   |   |  FROM TABLE
    |   |   |   |   |   |   |   |  RA2
    |   |   |   |   |   |   |   |  Table Scan.
    |   |   |   |   |   |   |   |  Forward Scan.
    |   |   |   |   |   |   |   |  Positioning at start
                                     of table.
    |   |   |   |   |   |   |   |  Executed in parallel
                                     with a 2-way
                                     partition scan.
    |   |   |
    |   |   |   |SORT Operator
    |   |   |   |Using Worktable2 for internal storage.
    |   |   |   |
    |   |   |   |   |SCAN Operator
    |   |   |   |   |  FROM TABLE
    |   |   |   |   |  RB2
    |   |   |   |   |  Table Scan.
    |   |   |   |   |  Forward Scan.
    |   |   |   |   |  Positioning at start of table.
    |   |   |   |   |  Executed in parallel with a
                          3-way partition scan.
```

两个都具有无用分区的表

下一示例使用 join，其中两侧的表的本机分区无用。表 RA2 上的分区应用于列 （a1、 a2），而 RB2 上的分区应用于列 (b1)。 join 谓词位于不同的列集上，因而两个表的分区无法起到作用。一种选择是在连接两侧进行动态重新分区。通过使用 M 对 N （2 对 3）的 exchange 运算符对表 RA2 进行重新分区，Adaptive Server 会选择表 RA2 的 a3 列，因为该列包含在与表 RB2 的连接中。同样，也对表 RB2 的 b3 列应用 3 路重新分区。按照 join 谓词，连接的重新分区操作数是等分区的，这意味着将连接每一侧的相应分区。通常，当需要在 join 运算符的两侧都进行重新分区时，Adaptive Server 将使用基于散列的分区方案。

```
select * from RA2, RB2 where a3 = b3

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 8
worker processes.

12 operator(s) under root
```

```
The type of query is SELECT.

ROOT:EMIT Operator

    |EXCHANGE Operator (Merged)
    |Executed in parallel by 3 Producer and 1 Consumer
         processes.

    |
    |   |EXCHANGE:EMIT Operator
    |   |
    |   |   |MERGE JOIN Operator
                (Join Type:Inner Join)
    |   |   | Using Worktable3 for internal storage.
    |   |   | Key Count: 1
    |   |   | Key Ordering: ASC
    |   |   |
    |   |   |   |SORT Operator
    |   |   |   |Using Worktable1 for internal
                    storage.
    |   |   |   |
    |   |   |   |   |EXCHANGE Operator (Repartitioned)
    |   |   |   |   |Executed in parallel by 2
                       Producer and 3 Consumer
                       processes.

    |   |   |   |   |
    |   |   |   |   |   |EXCHANGE:EMIT Operator
    |   |   |   |   |   |
    |   |   |   |   |   |   |RESTRICT Operator
    |   |   |   |   |   |   |
    |   |   |   |   |   |   |   |SCAN Operator
    |   |   |   |   |   |   |   |  FROM TABLE
    |   |   |   |   |   |   |   |  RA2
    |   |   |   |   |   |   |   |  Table Scan.
    |   |   |   |   |   |   |   |  Forward Scan.
    |   |   |   |   |   |   |   |  Positioning at
                                      start of table.
    |   |   |   |   |   |   |   |  Executed in
                                       parallel with
                                       a 2-way
                                       partition scan.
    |   |   |
    |   |   |   |SORT Operator
    |   |   |   |Using Worktable2 for internal
                     storage.
```

```
    |   |   |   |
    |   |   |   |   |EXCHANGE Operator (Repartitioned)
    |   |   |   |   |Executed in parallel by 3
                        Producer and 3 Consumer
                        processes.

    |   |   |   |   |
    |   |   |   |   |   |EXCHANGE:EMIT Operator
    |   |   |   |   |   |
    |   |   |   |   |   |   |SCAN Operator
    |   |   |   |   |   |   |  FROM TABLE
    |   |   |   |   |   |   |  RB2
    |   |   |   |   |   |   |  Table Scan.
    |   |   |   |   |   |   |  Forward Scan.
    |   |   |   |   |   |   |  Positioning at start
                                   of table.
    |   |   |   |   |   |   |  Executed in parallel
                                    with a 3-way
                                     partition scan.
```

通常，所有连接（包括 nested-loop、merge 和 hash 连接）的行为方式都相似。 nested-loop joins 显示一个异常，指示无法在 nested-loop join 的内部重新分区。之所以发生此限制，是因为在 nested-loop join 中，连接谓词的列值是从外部推到内部的。

复制型 *join*

当需要使用 nested-loop join 索引时，复制型 join 非常有用。请考虑以下情况：在某大表的连接列上设有有用的索引，但其分区无用，且该表连接到一个已分区或未分区的小表。可将小表复制 N 路到内部表的分区中，其中 N 是大表的分区数。大表的每个分区会与小表连接，由于连接内部不需要 exchange 运算符，因而允许执行索引 nested-loop join。

```
create table big_table(b1 int, b2 int, b3 int)
partition by hash(b3) (p1, p2)

create index big_table_nc1 on big_table(b1)

create table small_table(s1 int, a2 int, s3 int)

select * from small_table, big_table
where small_table.s1 = big_table.b1

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 3
worker processes.

7 operator(s) under root
```

```
The type of query is SELECT.

ROOT:EMIT Operator

    |EXCHANGE Operator (Merged)
    |Executed in parallel by 2 Producer and 1
         Consumer processes.

    |
    |   |EXCHANGE:EMIT Operator
    |   |
    |   |   |NESTED LOOP JOIN Operator (Join Type:
              Inner Join)
    |   |   |
    |   |   |   |EXCHANGE Operator (Replicated)
    |   |   |   |Executed in parallel by 1 Producer
                    and 2 Consumer processes.

    |   |   |   |
    |   |   |   |   |EXCHANGE:EMIT Operator
    |   |   |   |   |
    |   |   |   |   |   |SCAN Operator
    |   |   |   |   |   |  FROM TABLE
    |   |   |   |   |   |  small_table
    |   |   |   |   |   |  Table Scan.
    |   |   |
    |   |   |   |SCAN Operator
    |   |   |   |  FROM TABLE
    |   |   |   |  big_table
    |   |   |   |  Index : big_table_nc1
    |   |   |   |  Forward Scan.
    |   |   |   |  Positioning by key.
    |   |   |   |  Keys are:
    |   |   |   |    b1 ASC
    |   |   |   |  Executed in parallel with a
                      2-way hash scan.
```

并行重新格式化

并行重新格式化在您使用 nested-loop join 时特别有用。通常，重新格式化是指将嵌套连接的内部实现为一个工作表，然后根据连接谓词创建索引。使用并行查询和 nested-loop join，当连接列上不存在有用的索引或 nested-loop join 是查询的唯一可用选项（根据服务器 / 会话 / 查询级设置）时，重新格式化也很有用。这是 Adaptive Server 的一个重要选项。外部可能包含有用的分区，如果不包含，则可以对其重新分区来创建有用的分区。但对于 nested-loop join 的内部，任何重新分区都意味着必须将表重新格式化为一个使用新分区策略的工作表。对 nested-loop join 的内部扫描必须随后访问该工作表。

在下一示例中，对表 RA2 和 RB2 的重新分区将分别应用于列 （a1、 a2）和 （b1、 b2）。运行查询时，该会话的 merge 和 hash join 都将关闭。

```
select * from RA2, RB2 where a1 = b1 and a2 = b3

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 12
worker processes.

17 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

    |SEQUENCER Operator has 2 children.
    |
    |   |EXCHANGE Operator (Merged)
    |   |Executed in parallel by 4 Producer
         and 1 Consumer processes.

    |   |
    |   |   |EXCHANGE:EMIT Operator
    |   |   |
    |   |   |   |STORE Operator
    |   |   |   |  Worktable1 created, in allpages
                     locking mode, for REFORMATTING.
    |   |   |   |  Creating clustered index.
    |   |   |   |
    |   |   |   |   |INSERT Operator
    |   |   |   |   |  The update mode is direct.
    |   |   |   |   |
    |   |   |   |   |   |EXCHANGE Operator
                          (Repartitioned)
    |   |   |   |   |   |Executed in parallel by
                           2 Producer and 4
                           Consumer processes.
    |   |   |   |   |   |
    |   |   |   |   |   |   |EXCHANGE:EMIT Operator
    |   |   |   |   |   |   |
    |   |   |   |   |   |   |   |RESTRICT Operator
    |   |   |   |   |   |   |   |
    |   |   |   |   |   |   |   |   |SCAN Operator
    |   |   |   |   |   |   |   |   |  FROM TABLE
    |   |   |   |   |   |   |   |   |  RB2
    |   |   |   |   |   |   |   |   |  Table Scan.
    |   |   |   |   |   |   |   |   |  Executed in
```

```
                                                parallel
                                                with a
                                                2-way
                                                partition
                                                scan.
|   |   |   |   |
|   |   |   |   |  TO TABLE
|   |   |   |   |  Worktable1.
|
|   |EXCHANGE Operator (Merged)
|   |Executed in parallel by 4 Producer
      and 1 Consumer processes.

|   |
|   |   |EXCHANGE:EMIT Operator
|   |   |
|   |   |   |NESTED LOOP JOIN Operator
              (Join Type:Inner Join)
|   |   |   |
|   |   |   |   |EXCHANGE Operator (Repartitioned)
|   |   |   |   |Executed in parallel by 2
                    Producer and 4 Consumer
                    processes.
|   |   |   |   |
|   |   |   |   |   |EXCHANGE: EMIT Operator
|   |   |   |   |   |
|   |   |   |   |   |   |RESTRICT Operator
|   |   |   |   |   |   |
|   |   |   |   |   |   |   |SCAN Operator
|   |   |   |   |   |   |   |  FROM TABLE
|   |   |   |   |   |   |   |  RA2
|   |   |   |   |   |   |   |  Table Scan.
|   |   |   |   |   |   |   |  Executed in
                                    parallel with
                                    a 2-way
                                    partition scan.
|   |   |   |   |SCAN Operator
|   |   |   |   |  FROM TABLE
|   |   |   |   |  Worktable1.
|   |   |   |   |  Using Clustered Index.
|   |   |   |   |  Forward Scan.
|   |   |   |   |  Positioning by key.
```

顺序运算符会执行其所有子运算符 （但最后一个子运算符除外），然后执行最后一个子运算符。在此情况下，顺序运算符将执行第一个子运算符，该运算符会使用列 b1 和 b3 上的四路散列分区，将表 RB2 重新格式化为一个工作表。还将对表 RA2 执行四路重新分区，以与该工作表的存储分区相匹配。

串行连接

有时，可能会因为要连接的数据量而使运行并行连接毫无意义。如果您运行的查询与前面的连接查询相似，而现在每个表 （RA2 和 RB2）上都设置了谓词，以致要连接的数据量不足，则可以用串行模式来执行连接。在此情况下，这些表的分区方式无关紧要。查询仍能从表的并行扫描中获益。

```
select * from RA2, RB2 where a1=b1 and a2 = b2
and a3 =  0 and  b2 = 20



QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 4
worker processes.

11 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

    |MERGE JOIN Operator (Join Type: Inner Join)
    | Using Worktable3 for internal storage.
    |  Key Count: 1
    |  Key Ordering: ASC
    |
    |   |SORT Operator
    |   | Using Worktable1 for internal storage.
    |   |
    |   |   |EXCHANGE Operator (Merged)
    |   |   |Executed in parallel by 2 Producer and
                 1 Consumer processes.

    |   |   |
    |   |   |   |EXCHANGE:EMIT Operator
    |   |   |   |
    |   |   |   |   |RESTRICT Operator
    |   |   |   |   |
    |   |   |   |   |   |SCAN Operator
    |   |   |   |   |   |  FROM TABLE
    |   |   |   |   |   |  RA2
```

```
|   |   |   |   |   |  Table Scan.
|   |   |   |   |   |  Executed in parallel with
                           a 2-way partition scan.
|   |SORT Operator
|   |Using Worktable2 for internal storage.
|   |
|   |   |EXCHANGE Operator (Merged)
|   |   |Executed in parallel by 2 Producer and
              1 Consumer processes.

|   |   |
|   |   |   |EXCHANGE:EMIT Operator
|   |   |   |
|   |   |   |   |RESTRICT Operator
|   |   |   |   |
|   |   |   |   |   |SCAN Operator
|   |   |   |   |   |  FROM TABLE
|   |   |   |   |   |  RB2
|   |   |   |   |   |  Table Scan.
|   |   |   |   |   |  Executed in parallel with
                            a 2-way partition scan.
```

半连接

展平 in/exist 子查询时将产生半连接，这种连接的行为方式与常规内部连接相同。但是，复制型连接不能用于半连接，原因是在此情况下，外部行会出现多次匹配。

外部连接

在外部连接的并行处理方面，不考虑复制型连接。其它一切方面的行为方式均与常规内部连接相似。另一个不同点是，在属于外部组的外部连接中，不对任何表执行分区排除。

矢量集合

矢量集合是指带有 group-by 的查询。 Adaptive Server 可通过多种不同的方式来执行矢量集合。此处未描述实际算法；以下几节仅说明并行计算的技术。

内分区的矢量集合

如果任何基关系或中间关系需要分组，并在 group by 子句中列的子集或其本身上进行分区，则将在每个分区上并行执行分组操作，并使用简单的 N 对 1 exchange 将生成的已分组流合并。这是因为给定的组不能出现在多个流中。同样的限制也适用于在任何 SQL 查询上所执行的分组，前提是对分组列或其子集应用基于语义的分区。此并行矢量集合的方法称为内分区集合。

以下查询使用并行 in-partitioned 矢量集合，原因是范围分区在列 a1 和 a2 上定义，且后一列还正好是需要执行集合的列。

```
select count(*), a1, a2 from RA2 group by a1,a2

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2
worker processes.

4 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

    |EXCHANGE Operator (Merged)
    |Executed in parallel by 2 Producer and
        1 Consumer processes.

    |
    |   |EXCHANGE:EMIT Operator
    |   |
    |   |   |HASH VECTOR AGGREGATE Operator
    |   |   |  GROUP BY
    |   |   |  Evaluate Grouped COUNT AGGREGATE.
    |   |   | Using Worktable1 for internal storage.
    |   |   |
    |   |   |   |SCAN Operator
    |   |   |   |  FROM TABLE
    |   |   |   |  RA2
    |   |   |   |  Table Scan.
    |   |   |   |  Forward Scan.
    |   |   |   |  Positioning at start of table.
    |   |   |   |  Executed in parallel with a 2-way
                       partition scan.
    |   |   |   |  Using I/O Size 2 Kbytes for data
                       pages.
    |   |   |   |  With LRU Buffer Replacement
                        Strategy for data pages.
```

重新分区的矢量集合

有时，对表或中间结果分区可能不会对分组操作产生帮助。通过对源数据重新分区以与分组列匹配，并随后应用并行矢量集合，并行执行分组操作可能还是值得的。下面对此情况进行了说明，其中分区应用于列（a1、a2），但查询需要对列 a1 执行矢量集合。

```
select count(*), a1 from RA2 group by a1

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 4
worker processes.

6 operator(s) under root


The type of query is SELECT.

ROOT:EMIT Operator

    |EXCHANGE Operator (Merged)
    |Executed in parallel by 2 Producer and 1 Consumer
processes.

    |
    |   |EXCHANGE:EMIT Operator
    |   |
    |   |   |HASH VECTOR AGGREGATE Operator
    |   |   |  GROUP BY
    |   |   |  Evaluate Grouped COUNT AGGREGATE.
    |   |   | Using Worktable1 for internal storage.
    |   |   |
    |   |   |   |EXCHANGE Operator (Repartitioned)
    |   |   |   |Executed in parallel by 2 Producer
                      and 2 Consumer processes.

    |   |   |   |
    |   |   |   |   |EXCHANGE:EMIT Operator
    |   |   |   |   |
    |   |   |   |   |   |SCAN Operator
    |   |   |   |   |   |  FROM TABLE
    |   |   |   |   |   |  RA2
    |   |   |   |   |   |  Table Scan.
    |   |   |   |   |   |  Forward Scan.
    |   |   |   |   |   |  Positioning at start of
                               table.
    |   |   |   |   |   |  Executed in parallel with
                                a 2-way partition scan.
```

两阶段矢量集合

对于上一示例中的查询，重新分区的开销可能是巨大的。另一种可行的方案为，先执行一级分组，使用 N 对 1 的 exchange 运算符合并数据，随后再执行另一级分组。这种方式称为*两阶段*矢量集合。根据分组列的重复数，Adaptive Server 可通过 N 对 1 的 exchange（可降低第二级分组操作的开销）来减少数据流的基数。

```
select count(*), a1 from RA2 group by a1

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2
worker processes.

5 operator(s) under root


The type of query is SELECT.

ROOT:EMIT Operator

    |HASH VECTOR AGGREGATE Operator
    |  GROUP BY
    |  Evaluate Grouped SUM OR AVERAGE AGGREGATE.
    | Using Worktable2 for internal storage.
    |
    |   |EXCHANGE Operator (Merged)
    |   |Executed in parallel by 2 Producer and
            1 Consumer processes.
    |   |
    |   |   |EXCHANGE:EMIT Operator
    |   |   |
    |   |   |   |HASH VECTOR AGGREGATE Operator
    |   |   |   |  GROUP BY
    |   |   |   |  Evaluate Grouped COUNT AGGREGATE.
    |   |   |   | Using Worktable1 for internal
                        storage.
    |   |   |   |
    |   |   |   |   |SCAN Operator
    |   |   |   |   |  FROM TABLE
    |   |   |   |   |  RA2
    |   |   |   |   |  Table Scan.
    |   |   |   |   |  Executed in parallel with
                           a 2-way partition scan.
```

串行矢量集合

与前面的某些示例相似，如果通过谓词来限制流入分组运算符的数据量，则并行执行该查询的意义可能不大。在此情况下，将并行扫描分区，并使用 N 对 1 的 exchange 运算符来串行化数据流，然后再执行串行矢量集合：

```
select count(*), a1, a2 from RA2
where a1 between 100 and 200
group by a1, a2

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and
     2 worker processes.

4 operator(s) under root


The type of query is SELECT.

ROOT:EMIT Operator

   |HASH VECTOR AGGREGATE Operator
   |  GROUP BY
   |  Evaluate Grouped COUNT AGGREGATE.
   | Using Worktable1 for internal storage.
   |
   |    |EXCHANGE Operator (Merged)
   |    |Executed in parallel by 2 Producer and 1
             Consumer processes.
   |    |
   |    |    |EXCHANGE:EMIT Operator
   |    |    |
   |    |    |    |SCAN Operator
   |    |    |    |  FROM TABLE
   |    |    |    |  RA2
   |    |    |    |  Positioning at start of table.
   |    |    |    |  Executed in parallel with a 2-way
                        partition scan.
```

您不能始终对分区列进行分组，或利用已在分组列上进行分区的表。查询优化程序可确定以下哪种方式更为合适：重新分区且并行执行分组；合并分区表中的数据流并使用串行模式或两阶段集合来执行分组。

distinct

具有 distinct 操作的查询与不带集合分量的分组矢量集合相同。例如：

```
select distinct a1, a2 from RA2
```

等同于：

```
select a1, a2 from RA2 group by a1, a2
```

所有适用于矢量集合的方法在此处同样适用。

使用 *in* 列表的查询

Adaptive Server 使用优化的技术来处理 in 列表。这是一种常见的 SQL 构造。因此，其构造类似于：

```
col in (value1, value2,..valuek)
```

等同于：

```
col = value1 OR col = value2 OR .... col = valuek
```

in 列表中的值被置于内存中的特殊表内，并已经过排序删除了重复项。随后，使用索引 nested-loop join 将该表连接回基表。下面的示例对此进行了说明，其中 in 列表中包含两个值，它们与 or 列表中的两个值相对应：

```
SCAN Operator
FROM OR List
OR List has up to 2 rows of OR/IN values.

select * from RA2 where a3 in (1425, 2940)

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2
      worker processes.

6 operator(s) under root


The type of query is SELECT.

ROOT:EMIT Operator

  |EXCHANGE Operator (Merged)
  |Executed in parallel by 2 Producer and 1
       Consumer processes.

  |
  |   |EXCHANGE:EMIT Operator
  |   |
  |   |   |NESTED LOOP JOIN Operator (Join Type:
              Inner Join)
  |   |   |
```

```
|   |   |   |SCAN Operator
|   |   |   |  FROM OR List
|   |   |   |  OR List has up to 2 rows of OR/IN
                    values.
|   |   |
|   |   |   |RESTRICT Operator
|   |   |   |
|   |   |   |   |SCAN Operator
|   |   |   |   |  FROM TABLE
|   |   |   |   |  RA2
|   |   |   |   |  Index : RA2_NC1
|   |   |   |   |  Forward Scan.
|   |   |   |   |  Positioning by key.
|   |   |   |   |  Keys are:
|   |   |   |   |    a3 ASC
|   |   |   |   |  Executed in parallel with a
                       2-way hash scan.
```

使用 *or* 子句的查询

Adaptive Server 使用分离性谓词 （如 or 子句）并单独应用分离的每一侧，以限定一组行 ID (RID)。在分离每一侧所应用的分离性谓词集必须是可索引的。此外，分离每一侧的分离性谓词内部不能再进一步分离；也就是说，对分离性和结合性子句应用任意深度的嵌套毫无意义。在下一示例中，将在同一列上采用分离性谓词 （可在不同的列上设置分离性谓词，前提是您具有的索引可执行开销不大的扫描），但这些谓词可能限定多个重叠的数据行集。 Adaptive Server 在分离的每一侧单独应用谓词，并限定一组行 ID。这些行 ID 随后将易于排除重复项。

```
select a3 from RA2 where a3 = 2955 or a3 > 2990

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2
worker processes.

8 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

  |EXCHANGE Operator (Merged)
  |Executed in parallel by 2 Producer and 1
       Consumer processes.

  |
  |   |EXCHANGE:EMIT Operator
  |   |
  |   |   |RID JOIN Operator
```

```
|   |   | Using Worktable2 for internal storage.
|   |   |
|   |   |   |HASH UNION Operator has 2 children.
|   |   |   | Using Worktable1 for internal storage.
|   |   |   |
|   |   |   |   |SCAN Operator
|   |   |   |   |  FROM TABLE
|   |   |   |   |  RA2
|   |   |   |   |  Index : RA2_NC1
|   |   |   |   |  Forward Scan.
|   |   |   |   |  Positioning by key.
|   |   |   |   |  Index contains all needed
                       columns. Base table will not
                       be read.
|   |   |   |   |  Keys are:
|   |   |   |   |    a3 ASC
|   |   |   |   |  Executed in parallel with a
                     2-way hash scan.
|   |   |   |   |
|   |   |   |   |SCAN Operator
|   |   |   |   |  FROM TABLE
|   |   |   |   |  RA2
|   |   |   |   |  Index : RA2_NC1
|   |   |   |   |  Forward Scan.
|   |   |   |   |  Positioning by key.
|   |   |   |   |  Index contains all needed
                       columns. Base table will
                       not be read.
|   |   |   |   |  Keys are:
|   |   |   |   |    a3 ASC
|   |   |   |   |  Executed in parallel with a
                     2-way hash scan.
|   |   |   |   |RESTRICT Operator
|   |   |   |   |
|   |   |   |   |   |SCAN Operator
|   |   |   |   |   |  FROM TABLE
|   |   |   |   |   |  RA2
|   |   |   |   |   |  Using Dynamic Index.
|   |   |   |   |   |  Forward Scan.
|   |   |   |   |   |  Positioning by Row IDentifier
                         (RID.)
|   |   |   |   |   |  Using I/O Size 2 Kbytes for
                         data pages.
|   |   |   |   |   |  With LRU Buffer Replacement
                          Strategy for data pages.
```

示例通过索引 RA2_NC1 使用了两个单独的索引扫描，该索引在列 a3 上定义。随后，将在行 ID 的限定集合中检查重复的行 ID，并最终将其连接回基表。请注意代码行 `Positioning by Row Identifier (RID)`。可为分离的每一侧使用不同的索引，但前提条件是谓词是可索引的。为确定这一点，一种简单的方法是使用分离的每一侧单独运行查询，以确保谓词是可索引的。如果求索引交集比对表执行单个扫描看上去开销要大，则 Adaptive Server 可能不会选择求索引交集。

使用 *order by* 子句的查询

如果查询因使用了 order by 子句而要求对输出排序，则 Adaptive Server 可并行应用 sort。首先，如果存在某些可用的固有排序，Adaptive Server 将尝试避免排序操作。如果 Adaptive Server 强制排序，它将检查并确定是否可以并行执行排序。为此， Adaptive Server 可以对现有数据流重新分区，也可以使用现有的分区方案，然后对每个要素流应用排序。将使用 N 对 1 的顺序合并结果数据，并保留 exchange 运算符。

```
select * from RA2 order by a1, a2

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2
worker processes.

4 operator(s) under root

The type of query is SELECT.

ROOT:EMIT Operator

   |EXCHANGE Operator (Merged)
   |Executed in parallel by 2 Producer and
         1 Consumer processes.

   |
   |   |EXCHANGE:EMIT Operator
   |   |
   |   |   |SORT Operator
   |   |   | Using Worktable1 for internal storage.
   |   |   |
   |   |   |   |SCAN Operator
   |   |   |   |  FROM TABLE
   |   |   |   |  RA2
   |   |   |   |  Index : RA2_NC2L
   |   |   |   |  Forward Scan.
   |   |   |   |  Positioning at index start.
   |   |   |   |  Executed in parallel with a
                        2-way partition scan.
```

根据要排序的数据量和可用的资源， Adaptive Server 可能对数据流重新分区，以使其分区度高于该数据流的当前分区度，从而提高 sort 操作的执行速度。排序的分区度取决于并行执行 sort 所带来的益处是否明显大于重新分区所产生的开销。

子查询

Adaptive Server 会使用不同的方法来降低处理子查询的开销。并行优化取决于子查询的类型：

- 实现子查询 — 在实现步骤中不考虑并行查询方法。
- 展平子查询 — 仅当将子查询展平为常规内部连接或半连接时，才会考虑并行查询优化。
- 嵌套子查询 — 对于包含子查询的查询的最外层查询块考虑执行并行操作；内部、嵌套的查询始终串行执行。这意味着，嵌套子查询中的所有表都将被串行访问。在下面的示例中，将并行访问表 RA2，但在访问子查询前，将使用 2 对 1 的 exchange 运算符对该表进行序列化。将并行访问子查询中的表 RB2。

```
select count(*) from RA2 where not exists
(select * from RB2 where RA2.a1 = b1)

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2
worker processes.

8 operator(s) under root


The type of query is SELECT.

ROOT:EMIT Operator

   |SCALAR AGGREGATE Operator
   |  Evaluate Ungrouped COUNT AGGREGATE.
   |
   |    |SQFILTER Operator has 2 children.
   |    |
   |    |    |EXCHANGE Operator (Merged)
   |    |    |Executed in parallel by 2 Producer
                 and 1 Consumer processes.

   |    |    |
   |    |    |    |EXCHANGE:EMIT Operator
```

```
|   |   |   |
|   |   |   |   |RESTRICT Operator
|   |   |   |   |
|   |   |   |   |   |SCAN Operator
|   |   |   |   |   |  FROM TABLE
|   |   |   |   |   |  RA2
|   |   |   |   |   |  Index : RA2_NC2L
|   |   |   |   |   |  Forward Scan.
|   |   |   |   |   |  Executed in parallel with
                          a 2-way partition scan.
|   |
|   |  Run subquery 1 (at nesting level 1).
|   |
|   |  QUERY PLAN FOR SUBQUERY 1 (at nesting
       level 1 and at line 2).
|   |
|   |   Correlated Subquery.
|   |   Subquery under an EXISTS predicate.
|   |
|   |   |SCALAR AGGREGATE Operator
|   |   |  Evaluate Ungrouped ANY AGGREGATE.
|   |   |  Scanning only up to the first
                qualifying row.
|   |   |
|   |   |   |SCAN Operator
|   |   |   |  FROM TABLE
|   |   |   |  RB2
|   |   |   |  Table Scan.
|   |   |   |  Forward Scan.
|   |
|   |  END OF QUERY PLAN FOR SUBQUERY 1.
```

下面的示例演示展平为半连接的 in 子查询。 Adaptive Server 将该子查询转换为内连接，以便在按连接顺序移动表时提供更大的灵活性。从下面可以看出，最初位于子查询中的表 RB2 现在可进行并行访问。

```
select * from RA2 where a1 in (select b1 from RB2)

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 5
worker processes.

10 operator(s) under root


The type of query is SELECT.
```

```
ROOT:EMIT Operator

   |EXCHANGE Operator (Merged)
   |Executed in parallel by 3 Producer and 1 Consumer
processes.

   |
   |   |EXCHANGE:EMIT Operator
   |   |
   |   |   |MERGE JOIN Operator (Join Type: Inner Join)
   |   |   | Using Worktable3 for internal storage.
   |   |   |  Key Count: 1
   |   |   |  Key Ordering: ASC
   |   |   |
   |   |   |   |SORT Operator
   |   |   |   | Using Worktable1 for internal
                     storage.
   |   |   |   |
   |   |   |   |   |SCAN Operator
   |   |   |   |   |  FROM TABLE
   |   |   |   |   |  RB2
   |   |   |   |   |  Table Scan.
   |   |   |   |   |  Executed in parallel with a
                        3-way partition scan.
   |   |   |
   |   |   |   |SORT Operator
   |   |   |   | Using Worktable2 for internal
                    storage.
   |   |   |   |
   |   |   |   |   |EXCHANGE Operator (Merged)
   |   |   |   |   |Executed in parallel by 2
                      Producer and 3 Consumer
                      processes.

   |   |   |   |   |
   |   |   |   |   |   |EXCHANGE:EMIT Operator
   |   |   |   |   |   |
   |   |   |   |   |   |   |RESTRICT Operator
   |   |   |   |   |   |   |
   |   |   |   |   |   |   |   |SCAN Operator
   |   |   |   |   |   |   |   |  FROM TABLE
   |   |   |   |   |   |   |   |  RA2
   |   |   |   |   |   |   |   |  Index : RA2_NC2L
   |   |   |   |   |   |   |   |  Forward Scan.
   |   |   |   |   |   |   |   |  Positioning at
                                    index start.
```

```
    |   |   |   |   |   |   |   | Executed in
                                    parallel with
                                    a 2-way
                                     partition scan.
```

select into 子句

包含 select into 子句的查询会创建一个新表以在其中存储该查询的结果集。Adaptive Server 会优化 select into 命令的基查询部分，其优化方式与它处理标准查询时的方式相同，二者都会考虑并行和串行访问方法。并行执行的 select into 语句：

- 使用 select into 语句中指定的列创建新表。
- 在新表中创建 N 个分区，其中 N 是优化程序为查询中的 insert 操作选择的并行度。
- 使用 N 个工作进程在新表中填充查询结果。
- 如果不需要特定的目标分区，则取消对新表的分区。

并行执行 select into 语句比等效的串行查询计划需要更多的步骤。以下是并行执行的简单 select into：

```
select * into RAT2 from RA2

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2
worker processes.

4 operator(s) under root

The type of query is INSERT.

ROOT:EMIT Operator

   |EXCHANGE Operator (Merged)
   |Executed in parallel by 2 Producer and 1
       Consumer processes.

   |
   |   |EXCHANGE:EMIT Operator
   |   |
   |   |   |INSERT Operator
   |   |   |  The update mode is direct.
   |   |   |
   |   |   |   |SCAN Operator
   |   |   |   |  FROM TABLE
```

```
|   |   |   |  RA2
|   |   |   |  Table Scan.
|   |   |   |  Forward Scan.
|   |   |   |  Positioning at start of table.
|   |   |   |  Executed in parallel with a 2-way
                  partition scan.
|   |   |
|   |   |  TO TABLE
|   |   |  RAT2
|   |   |  Using I/O Size 2 Kbytes for data
               pages.
```

Adaptive Server 不会尝试提高对表 RA2 扫描所产生的数据流的分区度，而是将其并行插入目标表中。目标表最初使用分区度为 2 的循环分区创建。插入后，将对表取消分区。

如果要插入的数据集不够大，Adaptive Server 可能选择串行插入此数据。对源表的扫描仍可并行执行。目标表随后会被创建为未分区的表。

select into 允许指定目标分区。在此情况下，将使用该分区创建目标表，并且 Adaptive Server 会找出插入数据的最优方式。如果目标表的分区方式必须与源数据的分区方式相同，并且要插入的数据也足够多，则将并行执行 insert 运算符。

下一示例显示了源表和目标表的相同分区，并说明 Adaptive Server 是如何识别此情况并选择不对源数据重新分区的。

```
select * into new_table
partition by range(a1, a2)
(p1 values <= (500,100), p2 values <= (1000, 2000))
from RA2

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 2
worker processes.

4 operator(s) under root


The type of query is INSERT.

ROOT:EMIT Operator

   |EXCHANGE Operator (Merged)
   |Executed in parallel by 2 Producer and 1 Consumer
processes.

   |
```

```
|    |EXCHANGE:EMIT Operator
|    |
|    |    |INSERT Operator
|    |    |  The update mode is direct.
|    |    |
|    |    |    |SCAN Operator
|    |    |    |  FROM TABLE
|    |    |    |  RA2
|    |    |    |  Table Scan.
|    |    |    |  Forward Scan.
|    |    |    |  Positioning at start of table.
|    |    |    |  Executed in parallel with a 2-way
                          partition scan.
|    |    |
|    |    |    |  TO TABLE
|    |    |    |  RRA2
|    |    |    |  Using I/O Size 16 Kbytes for data
                       pages.
```

如果源分区与目标表的分区不匹配，则必须对源数据重新分区。下一示例对此进行了说明，其中使用 2 对 2 的 exchange 运算符将数据从范围分区转换为散列分区，在对数据重新分区后，它使用两个工作进程并行执行插入操作。

```
select * into HHA2
partition by hash(a1, a2)
(p1, p2)
from RA2

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 4
worker processes.

6 operator(s) under root


The type of query is INSERT.

ROOT:EMIT Operator

|EXCHANGE Operator (Merged)
|Executed in parallel by 2 Producer and 1
      Consumer processes.

|
|    |EXCHANGE:EMIT Operator
|    |
```

```
|   |   |INSERT Operator
|   |   |  The update mode is direct.
|   |   |
|   |   |   |EXCHANGE OperatorEXCHANGE Operator (
                Merged)
|   |   |   |Executed in parallel by 2 Producer
                and 2 Consumer processes.

|   |   |   |
|   |   |   |   |EXCHANGE:EMIT Operator
|   |   |   |   |
|   |   |   |   |   |SCAN Operator
|   |   |   |   |   |  FROM TABLE
|   |   |   |   |   |  RA2
|   |   |   |   |   |  Table Scan.
|   |   |   |   |   |  Forward Scan.
|   |   |   |   |   |  Positioning at start of table.
|   |   |   |   |   |  Executed in parallel with a
                            2-way partition scan.
|   |   |
|   |   |  TO TABLE
|   |   |  HHA2
|   |   |  Using I/O Size 16 Kbytes for data
                pages.
```

insert/delete/update

在 Adaptive Server 中， insert、 delete 和 update 操作以串行模式执行。但是，除查询中用于限定要删除或更新的行的目标表外，所有其它表均可并行访问。

```
delete from RA2
where exists
(select * from RB2
where RA2.a1 = b1 and RA2.a2 = b2)

QUERY PLAN FOR STATEMENT 1 (at line 1).
Executed in parallel by coordinating process and 3
worker processes.

9 operator(s) under root

The type of query is DELETE.

ROOT:EMIT Operator
```

```
|DELETE Operator
|  The update mode is deferred.
|
|   |NESTED LOOP JOIN Operator (Join Type:Inner Join)
|   |
|   |   |SORT Operator
|   |   | Using Worktable1 for internal storage.
|   |   |
|   |   |   |EXCHANGE Operator (Merged)
|   |   |   |Executed in parallel by 3 Producer
                  and 1 Consumer processes.

|   |   |   |
|   |   |   |   |EXCHANGE:EMIT Operator
|   |   |   |   |
|   |   |   |   |   |RESTRICT Operator
|   |   |   |   |   |
|   |   |   |   |   |   |SCAN Operator
|   |   |   |   |   |   |  FROM TABLE
|   |   |   |   |   |   |  RB2
|   |   |   |   |   |   |  Table Scan.
|   |   |   |   |   |   |  Forward Scan.
|   |   |   |   |   |   |  Positioning at start of
                              table.
|   |   |   |   |   |   |  Executed in parallel with
                              a 3-way partition scan.
|   |   |   |   |   |   |  Using I/O Size 2 Kbytes
                              for data pages.
|   |   |   |   |   |   |  With LRU Buffer Replacement
                              Strategy for data pages.
|   |
|   |   |RESTRICT Operator
|   |   |
|   |   |   |SCAN Operator
|   |   |   |  FROM TABLE
|   |   |   |  RA2
|   |   |   |  Index : RA2_NC1
|   |   |   |  Forward Scan.
|   |   |   |  Positioning by key.
|   |   |   |  Keys are:
|   |   |   |    a3 ASC
|
|  TO TABLE
|  RA2
|  Using I/O Size 2 Kbytes for data pages.
```

应对要删除的表 RB2 串行执行扫描和删除。但是，对表 RA2 并行执行了扫描。此情况同样适用于 update 或 insert 语句。

分区排除

语义分区的一个优点是，查询处理器能够利用该分区并在编译时间排除 range、 hash 和 list 分区。对于 hash 分区，只能使用等同性谓词，而对于 range 和 list 分区，等同性和非等同性谓词均可用于排除分区。例如，请考虑表 RA2，其语义分区在列 a1、 a2 上定义 （其中 p1 值 <= (500,100)， p2 值 <= (1000, 2000)）。如果在 a1 列或在 a1 和 a2 列上设置了谓词，则可以执行某些分区排除。例如，以下语句不限定任何数据：

```
select * from RA2 where a1 > 1500
```

可以在 showplan 输出中查看此情况。

```
QUERY PLAN FOR STATEMENT 1 (at line 1).
.............................
|   |   |SCAN Operator
|   |   |  FROM TABLE
|   |   |  RA2
|   |   |  [ Eliminated Partitions : 1 2 ]
|   |   |  Index : RA2_NC2L
```

短语 `Eliminated Partitions` 依据分区的创建方式来标识分区并分配一个序号以便确认。对于表 RA2，由 p1 （其中，（a1、 a2） <= (500, 100)）表示的分区被视为分区号一，而由 p2（其中，（a1、a2）> (500, 100) 且 <= (1000, 2000)）表示的分区被标识为分区号二。

请考虑对散列分区表执行等同性查询，其中散列分区中的所有键都具有一个 equality 子句。可通过采用表 HA2 来说明这一点，该表在列 （a1、a2）上进行了两路散列分区。列号将引用分区在 sp_help 输出中列出的顺序。

```
select * from HA2 where a1 = 10 and a2 = 20

QUERY PLAN FOR STATEMENT 1 (at line 1).
.............................

|SCAN Operator
|  FROM TABLE
|  HA2
|  [ Eliminated Partitions : 1 ]
|  Table Scan.
```

分区倾斜

在确定是否可以使用并行分区扫描时，分区倾斜可发挥重要的作用。Adaptive Server 分区倾斜定义为分区的最大大小与平均大小的比率。请考虑具有 4 个分区的表，这些分区的大小分别为 20、20、35 和 80 页。分区的平均大小为 (20 + 20 + 35 + 85)/4 = 40 页。最大的分区包含 85 页，因此分区倾斜应计算为 85/40 = 2.125。对于分区扫描，执行并行扫描的开销与对最大分区进行扫描的开销相同。与此相反，基于散列的分区可能会非常迅速，这是因为每个工作进程均可以在一个页号或分配单元上应用散列，并仅扫描属于它的数据部分。倾斜分区会导致性能降低，这种影响并不总是发生在扫描级别，而是在对数据建立更复杂运算符 （如若干 join 操作）时。在此类情况下，出错空间呈指数增加。

对表运行 sp_help 可查看分区倾斜：

```
sp_help HA2

........
name   type partition_type partitions  partition_keys
------ -------------------- -------------- -----------
HA2    base table          hash                    2 a1, a2

partition_name partition_id pages       segment
create_date
-------------------------- ------------ ----------- --
------------
--------------------------
HA2_752002679               752002679        324 default
Aug 10 2005  2:05PM
HA2_768002736               768002736        343 default
Aug 10 2005  2:05PM

Partition_Conditions
--------------------
NULL

Avg_pages   Max_pages   Min_pages   Ratio(Max/Avg)

Ratio(Min/Avg)
----------- ----------- ----------- ------------------
---------
----------------------------
333         343         324                     1.030030

                              0.972973
```

或者，通过查询 systabstats 系统目录 （其中列出每个分区中的页数），也可以计算倾斜。

查询为何不并行运行

Adaptive Server 会在以下情况中以串行模式运行查询：

- 数据不足，无法从并行访问中获益。
- 查询中不包含等同性连接谓词，例如：

```
select * from RA2, RB2
where a1 > b1
```

- 资源 （如线程或内存）不足，无法并行运行查询。
- 对全局非聚簇索引使用覆盖扫描。
- 在无法展平的嵌套子查询内部访问表和索引。

运行期调整

如果运行期没有足够的工作进程可用，则执行引擎将尝试减少计划中 exchange 运算符所使用的工作进程数。

- 首先，尝试减少查询计划中某些 exchange 运算符所使用的工作进程，而不是依靠对查询进行串行重新编译。根据查询计划的语义，某些 exchange 运算符可进行调整，而某些则不可以。某些还对可以调整的方式进行了限制。
- 并行查询计划需要在保证一个最低的工作进程数时才能运行。如果没有足够的工作进程可用，将以串行模式重新编译查询。如果无法重新编译，则查询会中止并将生成相应的错误消息。

它通过以下两种方式实现此目的：

Adaptive Server 支持串行重新编译以下类型的查询：

- 即席的 select 查询，但 select into、 alter table 和 execute immediate 查询除外。
- 存储过程，但 select into 和 alter table 查询除外。

识别和管理运行期调整

Adaptive Server 提供了两种机制来帮助您观察查询计划的运行期调整：

- 发生运行期调整时，使用 set process_limit_action 可以中止批处理或过程。
- 发生运行期调整并且 showplan 有效时， showplan 将输出调整的查询计划。

使用 *set process_limit_action*

使用 set 命令的 process_limit_action 选项可在会话或存储过程级别监控调整的查询计划的使用。将 process_limit_action 设置为 “abort”时，如果需要调整的查询计划，则 Adaptive Server 将记录错误 11015 并中止查询。将 process_limit_action 设置为 “warning”时， Adaptive Server 将记录错误 11014，但仍执行查询。例如，在运行期调整查询时，下面的命令将中止批处理：

```
set process_limit_action abort
```

通过在错误日志中检查错误 11014 和 11015 所发生的次数，可以确定 Adaptive Server 使用调整的查询计划代替优化的查询计划的程度。要解除限制并允许运行期调整，请使用：

```
set process_limit_action quiet
```

请参见 《参考手册：命令》中的 set。

使用 *showplan*

使用 showplan 时， Adaptive Server 将在运行给定查询前显示该查询的优化计划。如果查询计划包含并行处理并进行了运行期调整， showplan 会显示此信息，后跟调整的查询计划：

```
AN ADJUSTED QUERY PLAN IS BEING USED FOR STATEMENT 1
BECAUSE NOT ENOUGH WORKER PROCESSES ARE CURRENTLY
AVAILABLE.

ADJUSTED QUERY PLAN:
```

当使用 set noexec 时， Adaptive Server 不会尝试执行查询，因此从不会显示运行期计划。

减小运行期调整的可能性

若有减少运行期调整的次数，请增加可用于并行查询的工作进程数。为此，可使用以下任一选项增大系统的工作进程总数，或者限制或取消非关键查询的并行执行：

- set parallel_degree，可对并行度设置会话级限制，或者
- 查询级 parallel 1 和 parallel N 子句，可限制各个语句使用的工作进程数。

要减少系统过程的运行期调整次数，请在改变服务器级或会话级的并行度后重新编译这些过程。请参见 《Adaptive Server 参考手册：过程》中的 sp_recompile。

第 6 章

积极和消极集合

本章介绍 Adaptive Server 中的积极和消极集合。

概述

集合处理是 DBMS 环境中最有用的操作之一。它汇总具有集合值的大量数据，其中包括：

- 指定的一组行中某一列的最小值、最大值、总和或平均值
- 与条件匹配的行计数
- 其它统计函数

在 SQL 中，集合处理是使用集合函数 min()、max()、count()、sum() 和 avg() 以及 group by 和 having 子句执行的。SQL 语言可实现两种集合处理类型，即*矢量集合*和*标量集合*。select-project-join (SPJ) 查询说明了这两类集合处理：

```
select r1, s1
from r, s
where r2 = s2
```

矢量集合

在矢量集合中， SPJ 结果集在 group by 子句表达式中分组，然后 select 子句集合函数会应用于每个组。查询会针对每组生成一个结果行：

```
select r1, sum (s1)
from r, s
where r2 = s2
group by r1
```

标量集合

在标量集合中，没有 group by 子句，整个 SPJ 结果集由同一 select 子句集合函数集合为一个组。查询将生成一个结果行：

```
select sum (s1)
from r, s
where r2 = s2
```

积极集合

积极集合会转换查询的内部表示 （如上面已介绍的查询），将集合视为以增量方式执行来处理查询：首先，在本地针对每个表中较小的本地子群生成中间集合结果，然后在连接后进行全局行处理，进而组合本地集合结果以生成最终结果集。

这些对任何数据集都返回同一结果集的查询是*派生表*对上述矢量和标量集合示例进行的 SQL 级重写。这些查询说明了 Adaptive Server 对查询的内部表示执行的积极集合转换。

矢量集合

```
select r1, sum(sum_s1 * count_r)
from
    (select
    r1,r2,
    count_r = count(*)
    from r
    group by r1,r2
    )gr
    ,
    (select
        s2,
        sum_s1 = sum(s1)
    from s
    group by s2
    )gs
where r2-s2
group by r1
```

标量集合

```
select sum(sum_s1 * count_r)
from
    (select
        r2,
        count_r = count(*)
    from r
    group by r2
    )gr
    ,
    (select
```

```
            s2,
            sum_s1 = sum(s1)
        from s
        group by s2
        )gs
    where r2 = s2
```

积极集合计划由优化程序生成并进行开销计算，可将其选作最佳计划。相对于原始 SQL 查询而言，这种形式的高级查询优化可对性能增益的大小进行排序。

集合和查询处理

从查询处理 (QP) 的角度来看，集合既是开销极高的操作，也是对查询性能具有重要影响的操作。

- 集合通常计算集合值。与标量集合相比，矢量集合开销较高，因为必须将行分为一组才能获取组的集合结果；通常情况下，这意味着要通过排序或散列这两种开销极高的操作对行进行重新排序。
- 在将减少基数的运算符 （如过滤器）应用到输入集后执行集合操作可降低集合的开销，并因而提高整体查询性能。
- 在将增加基数的运算符 （如连接与联合）应用到输入集之前执行集合操作会降低集合的开销，并因而提高整体查询性能。
- 早期执行集合操作可通过减少相应的输入集基数来降低父运算符的开销，并进而提高整体查询性能。
- 当分组列具有相对较少的不同值组合时，集合可显著减少结果集中输入集的基数。
- 由于已分组 （例如，当集合按分组列排序时），集合输入集的某些属性可降低矢量集合的开销；在标量集合中，通过按集合列排序的行，可计算 min 或 max，而无需访问每个输入行。
- 计划片段物理属性对集合开销具有很大影响。

集合的 Naive QP 实现可按 SQL 查询的指示将标准或矢量集合运算符置于其查询块的 SPJ 部分之上。然而，有些代数转换会保留查询的语义并允许集合位于运算符树中的其它位置：

- 将集合向下朝着叶的方向推动，以实现早期集合 （称为积极集合）。
- 将集合向上朝着根的方向拉动，以实现延迟集合 （称为消极集合）。

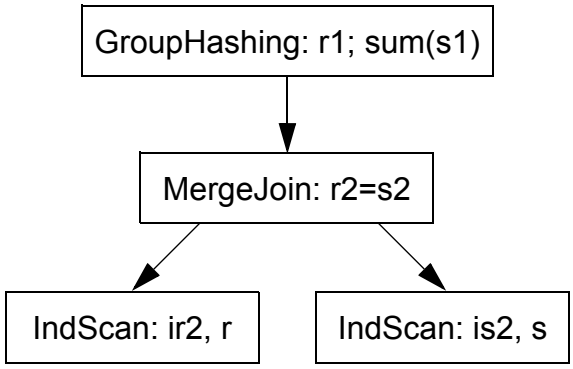
通过此类转换获得的计划在性能方面存在着很大的差异。积极集合可以显著减少中间结果的基数，这对于分布式查询处理 (DQP) 而言更为重要。此类大小顺序基数可降低跨节点数据传输的开销，从而克服了 DQP 相对于传统 QP 而言的主要缺点。

Adaptive Server 15.0.2 及更高版本对查询计划的叶实现积极集合，这意味着对扫描运算符实现积极集合。

以下查询说明了积极集合的 QP 影响：

```
select r1, sum(s1)
from r,s
where r2 = s2
group by r1
```

图 6-1：典型的查询执行计划

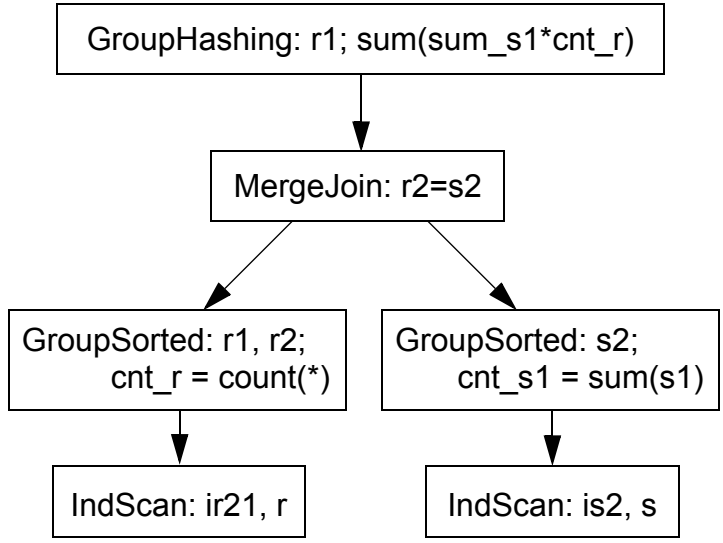


在 r(r2) 和 s(s2) 上执行的两个索引扫描提供了“r2=s2”合并连接所需的排序。按照查询的指定，对连接执行基于散列的分组。

优化程序还会生成执行积极集合 （也称为*向下推动分组*、*提前分组* 或 *积极分组*）的查询计划。使用派生表的转换的 SQL 表示为：

```
select r1, sum(sum_s1 * cnt_r)
from
    (select r1, r2, cnt_r = count(*)
            from r
            group by r1, r2
    ) as gr
    ,
    (select s2, sum_s1 = sum(s1)
            from s
            group by s2
    ) as gs
where r2 = s2
group by r1
```

图 6-2：可能的积极集合计划



两个积极 GroupSorted 运算符根据本地分组列来分组。GroupSorted 运算符可应用到由于某原因（除了是集合函数参数的原因之外）而突出的任何列。这些列包括：

- group by 子句中的主分组列
- 尚未应用的谓词所需要的列

若要放置开销较低的 GroupSorted 运算符，子计划片段必须对所有本地分组列提供排序；因此 `ir21` 索引在 `r(r2, r1)` 之上。

示例

联机数据存档

实现积极集合的最具有说服力的原因是联机数据存档，它是分布式查询处理 (DQP) 安装，其中，最近的 OLTP 可读写数据在 Adaptive Server 上，只读历史数据在另一服务器 （Adaptive Server 或 ASIQ）上。

以下视图 v 可向决策支持系统 (DSS) 应用程序提供对 ase_tab 中的本地 Adaptive Server 数据的透明访问，并且允许其通过组件集成服务 (CIS) proxy_asiq_tab 对 ASIQ 服务器上的远程历史数据进行透明访问。

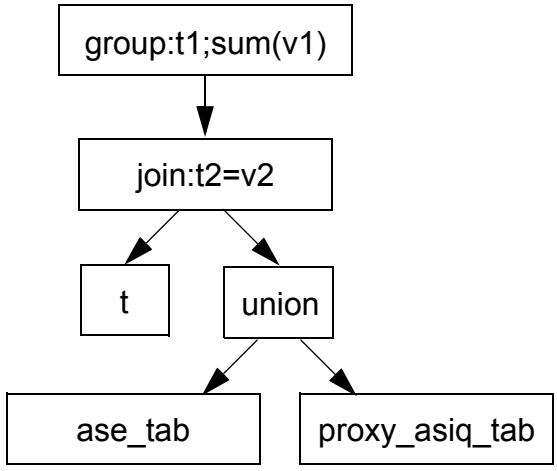
```
create view v(v1, v2)
as
select a1, a2 from ase_tab
union all
select q1, q2 from proxy_asiq_tab
```

DSS 应用程序忽略数据的分布式特性，并将此 union-in-view 表用作其复杂查询的基本表，并且通常会使用集合：

```
select t1, sum(v1)
from t,v
where t2=v2
group by t1
```

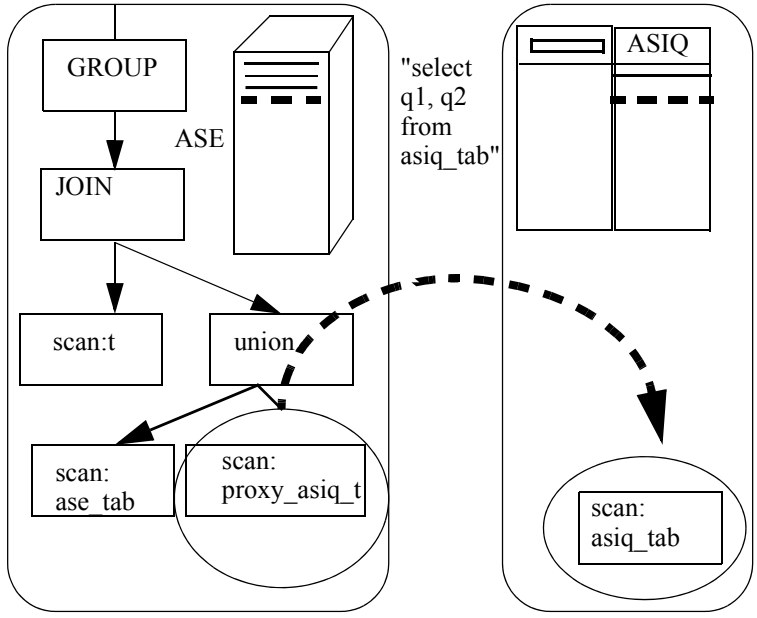
进行视图和联合解析后，将获得以下运算符树：

图 6-3：SQL 查询重写



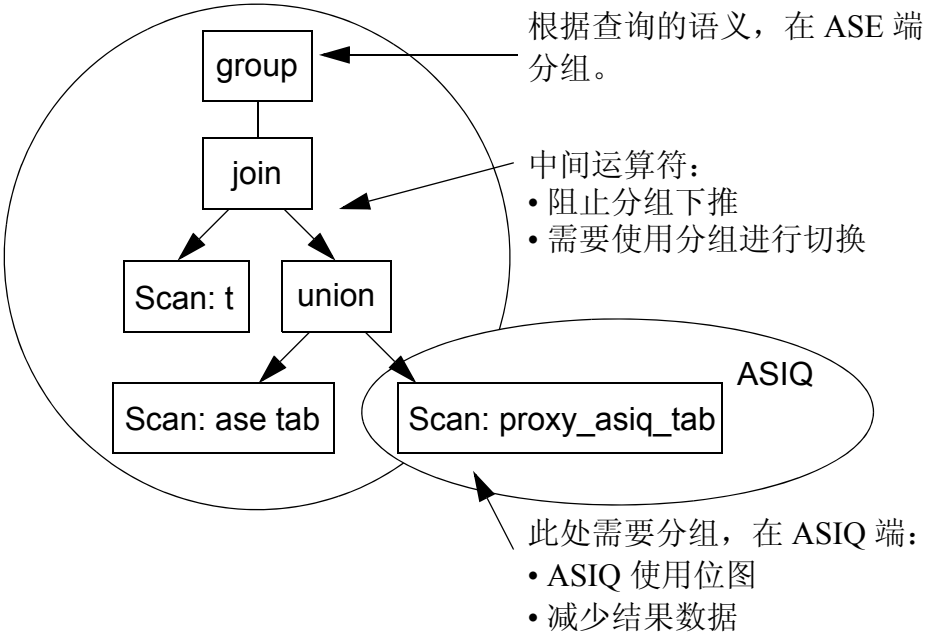
由于此树使用 CIS 代理表，所以 CIS 层使用专门的远程扫描运算符来生成计划片段并向远程节点传送计划片段。

图 6-4：次优典型 CIS 行为



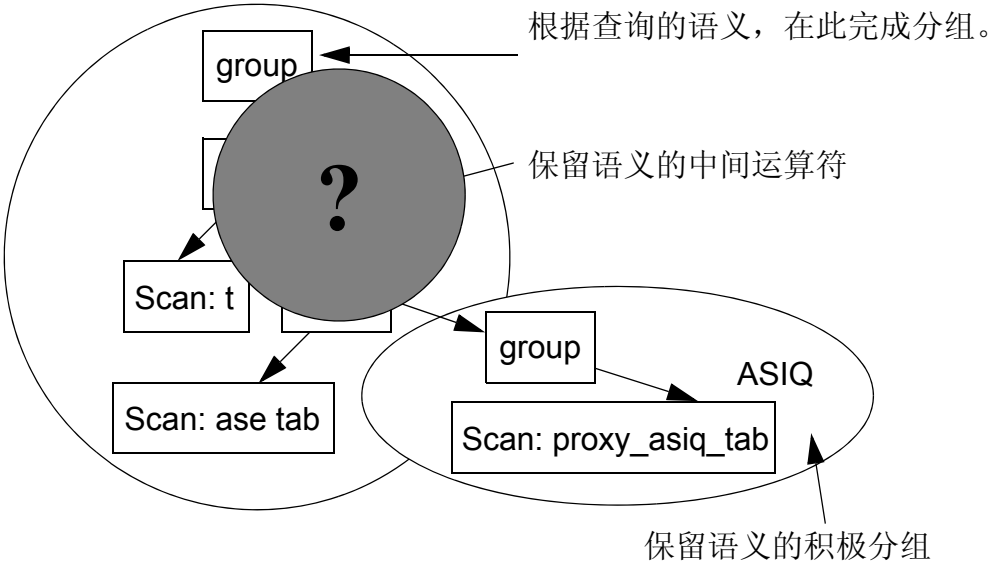
此机制之所以是次优机制，原因如下：整个历史表通过 CIS 层传送到 Adaptive Server 端，引发了极高的网络开销；此外，没有使用基于位图的高级 ASIQ 分组算法。

图 6-5：集合的典型处理



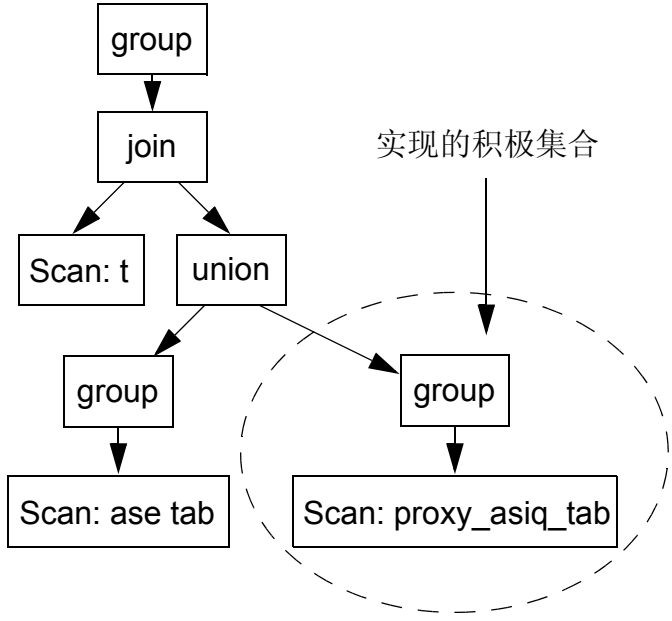
理想情况下，在运算符树上执行转换，在 ASIQ 端进行分组，所以只传输集合数据。

图 6-6：所需的集合处理布局



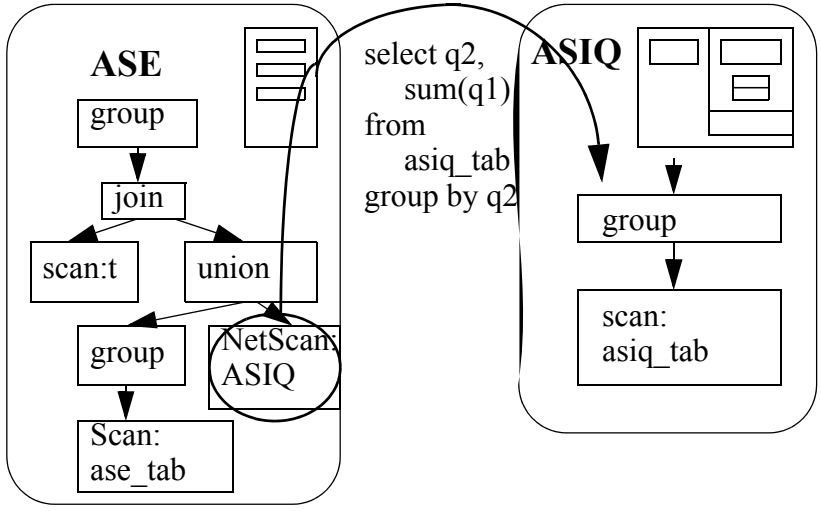
在此示例中，在组和 CIS 代理之间有两个运算符：join 和 union。下一转换将分组推到 join 和 union 下，从而实现*积极集合*：

图 6-7：积极集合



现在，分组与 CIS 代理相邻。 CIS 层可向 ASIQ 发送已分组的查询并返回集合数据。

图 6-8：包含积极集合的最佳 CIS 行为



DSS/DQP

在分布式环境中高效执行复杂的集合 DSS 查询不仅对联机数据存档是一个挑战；通常，它在联机数据存档 DSS/DQP 中也是一个常见的 DSS/DQP 问题：

- 典型查询涉及复杂的连接和联合，在查询树的顶部执行集合操作。
- 数据跨节点分布，必须将中间结果从生产者节点传送到消耗者节点，以进行进一步的处理。

单个节点 DSS

虽然上述示例一般适用于 DQP，特别是联机数据存档，但积极集合对性能的影响不仅仅是在各 DQP 节点之间传送中间结果。

积极集合通过减少中间结果集提高了集合复杂查询的性能。因为集合复杂查询很常见，所以积极集合可在所有 DSS 应用程序中提高 Adaptive Server 的性能。

使用积极集合

积极集合是一种内部查询处理功能。在启用积极集合时，无需在 SQL 级别上进行任何更改；使用集合的查询会自动枚举基于积极集合的计划，并由优化程序进行开销计算。

启用积极集合

积极集合由 advanced_aggregation 优化程序设置控制，缺省情况下在所有优化目标中处于关闭状态，但 allrows_dss 除外，在此优化目标中，此设置处于打开状态。可以在连接或查询级别，启用、禁用积极集合或将其重置为优化程序目标的缺省值。

例如，在连接级别上启用：

```
set advanced_aggregation on
```

在查询级别上启用：

```
select r1, sum (s1)
from r, s
where r2 = s2
group by r1
plan
"(use advanced_aggregation on)"
```

或者，如果优化目标设置为 allrows_dss，则会隐式启用积极集合。在下例中，抽象计划在查询级别设置 allrows_dss：

```
select r1, sum (s1)
from r, s
where r2 = s2
group by r1
plan
"(use optgoal allrows_dss)"
```

检查积极集合

启用积极集合时，优化程序将根据估计的开销最低的计划是否使用积极集合来确定开销。

showplan 集合的输出：

```
1> select r1, sum(s1)
2> from r, s
3> where r2=s2
4> group by r1
5> go

QUERY PLAN FOR STATEMENT 1 (at line 1).
STEP 1
The type of query is SELECT.
6 operator(s) under root
|ROOT:EMIT Operator
|
| |HASH VECTOR AGGREGATE Operator
| | GROUP BY
| | Evaluate Grouped SUM OR AVERAGE AGGREGATE.
| | Using Worktable2 for internal storage.
| | Key Count: 1
| |
| | |MERGE JOIN Operator (Join Type:Inner Join)
| | | Using Worktable1 for internal storage.
| | | Key Count: 1
| | | Key Ordering: ASC
| | |
| | | |GROUP SORTED Operator
| | | | Evaluate Grouped COUNT AGGREGATE.
| | | | |
| | | | |SCAN Operator
| | | | | FROM TABLE
| | | | | r
| | | | | Index : ir21
| | | | | Forward Scan.
| | | | | Positioning at index start.
| | | | | Index contains all needed columns. Base table will not be read.
| | | | | Using I/O Size 2 Kbytes for index leaf pages.
| | | | | With LRU Buffer Replacement Strategy for index leaf pages.
| | | |
| | | |GROUP SORTED Operator
| | | | Evaluate Grouped SUM OR AVERAGE AGGREGATE.
| | | | |
| | | | |SCAN Operator
| | | | | FROM TABLE
```

```
| | | | | s
| | | | | Index : is21
| | | | | Forward Scan.
| | | | | Positioning at index start.
| | | | | Index contains all needed columns. Base table will not be read.
| | | | | Using I/O Size 2 Kbytes for index leaf pages.
| | | | | With LRU Buffer Replacement Strategy for index leaf pages.
r1
----------- -----------
          1           2
          2           4
(2 rows affected)
```

当查询对 r 和 s 的连接执行矢量集合时，在所有情形下都需要查询树顶部的 hash vector aggregate 运算符。但是， r 和 s 的扫描上的 group sorted 运算符不是查询的一部分；它们执行积极集合。

当 advanced_aggregation 为 off 时，计划不包含积极集合运算符 group sorted：

```
1> set advanced_aggregation off
2> go
1> select r1, sum(s1)
2> from r, s
3> where r2=s2
4> group by r1
5> go


QUERY PLAN FOR STATEMENT 1 (at line 1).
STEP 1
The type of query is SELECT.
4 operator(s) under root
|ROOT:EMIT Operator
|
| |HASH VECTOR AGGREGATE Operator
| | GROUP BY
| | Evaluate Grouped SUM OR AVERAGE AGGREGATE.
| | Using Worktable2 for internal storage.
| | Key Count: 1
| |
| | |MERGE JOIN Operator (Join Type:Inner Join)
| | | Using Worktable1 for internal storage.
| | | Key Count: 1
| | | Key Ordering: ASC
| | |
| | | |SCAN Operator
| | | | FROM TABLE
```

```
| | | | r
| | | | Index : ir21
| | | | Forward Scan.
| | | | Positioning at index start.
| | | | Index contains all needed columns. Base table will not be read.
| | | | Using I/O Size 2 Kbytes for index leaf pages.
| | | | With LRU Buffer Replacement Strategy for index leaf pages.
| | |
| | | |SCAN Operator
| | | | FROM TABLE
| | | | s
| | | | Index : is21
| | | | Forward Scan.
| | | | Positioning at index start.
| | | | Index contains all needed columns. Base table will not be read.
| | | | Using I/O Size 2 Kbytes for index leaf pages.
| | | | With LRU Buffer Replacement Strategy for index leaf pages.
r1
----------- -----------
          1           2
          2           4
(2 rows affected)
```

使用抽象计划强制实现积极集合

当子计划片段提供本地分组列的排序时，优化程序会适时枚举开销较低的基于 GroupSorted 的积极集合计划。

此限制可避免优化搜索空间和时间的增加。但在某些情况下，基于散列的积极集合会生成开销最低的计划。在这种情况下，可以使用抽象计划来强制实现积极集合。必须启用 advanced_grouping 才能使用此类抽象计划；否则，将拒绝积极集合抽象计划。

在上面的示例中，如果 r 在 (r1, r2) 上没有索引，并且 r 较大，但具有的 r1--r2 值区分对较少，则通过 r 与积极分组的散列连接是最佳计划，由以下抽象计划强制执行：

```
1> select r1, sum(s1)
2> from r, s
3> where r2=s2
4> group by r1
5> plan
6> "(group_hashing
7>         (h_join
8>                 (group_hashing
```

```
9>                      (t_scan r)
10>                 )
11>                 (t_scan s)
12>         )
13> )"
14> go


QUERY PLAN FOR STATEMENT 1 (at line 1).
Optimized using the Abstract Plan in the PLAN clause.
STEP 1
The type of query is SELECT.
5 operator(s) under root
|ROOT:EMIT Operator
|
| |HASH VECTOR AGGREGATE Operator
| | GROUP BY
| | Evaluate Grouped SUM OR AVERAGE AGGREGATE.
| | Using Worktable3 for internal storage.
| | Key Count: 1| |
| | |HASH JOIN Operator (Join Type: Inner Join)
| | | Using Worktable2 for internal storage.
| | | Key Count: 1
| | |
| | | |HASH VECTOR AGGREGATE Operator
| | | | GROUP BY
| | | | Evaluate Grouped COUNT AGGREGATE.
| | | | Using Worktable1 for internal storage.
| | | | Key Count: 2
| | | |
| | | | |SCAN Operator
| | | | | FROM TABLE
| | | | | r
| | | | | Table Scan.
| | | | | Forward Scan.
| | | | | Positioning at start of table.
| | | | | Using I/O Size 2 Kbytes for data pages.
| | | | | With LRU Buffer Replacement Strategy for data pages.
| | | |
| | | | |SCAN Operator
| | | | | FROM TABLE
| | | | | s
| | | | | Table Scan.
| | | | | Forward Scan.
| | | | | Positioning at start of table.
| | | | | Using I/O Size 2 Kbytes for data pages.
| | | | | With LRU Buffer Replacement Strategy for data pages.
```

```
r1
----------- -----------
1           2
2           4

(2 rows affected)
```

散列矢量集合运算符应抽象计划的请求积极地集合 r 的扫描。

第 7 章

控制优化

本章描述了查询处理选项，这些选项将影响查询处理器对连接顺序、索引、 I/O 大小以及高速缓存策略的选择。

特殊优化技术

Sybase 建议，在使用本章讨论的工具之前，应阅读 《性能和调优系列：基础知识》。该书可帮助您理解本章中的内容。

应小心使用优化技术，因为这些技术能够让您覆盖由 Adaptive Server 查询处理器做出的决策，如果使用不当，可能会对性能造成极大的负面影响。您应了解这不但会影响您个人查询的性能，还可能会影响整个系统的性能。

大多数情况下， Adaptive Server 基于开销的高级查询处理器可制定出色的查询计划。但是，当查询处理器没有选择可获得最佳性能的恰当索引或选择了一个次优的连接顺序时，您必须控制查询的访问方法。使用优化技术，您可以进行此项控制。

此外，在进行调优时，您最好注意不同的连接顺序、 I/O 大小或高速缓存策略的效果。您可以利用其中一些优化选项来指定查询处理或访问策略，而不必进行开销很大的重新配置。

Adaptive Server 提供了可影响查询优化的工具和查询子句，以及帮助您理解查询处理器为何如此选择的高级查询分析工具。

注释 本章提供了对某些优化问题的解决方法。如果这些解决方法不能充分地解决这些问题，请与 Sybase 技术支持部门联系。

查看当前的优化程序设置

sp_options 允许您查看以下选项的当前优化程序设置：

- set plan dump / load
- set plan exists check
- set forceplan
- set plan optgoal
- set [optCriteria]
- set plan opttimeoutlimit
- set plan replace
- set statistics simulate
- set metrics_capture
- set prefetch
- set parallel_degree number
- set process_limit_action
- set resource_granularity number
- set scan_parallel_degree number
- set repartition_degree number

sp_options 可查询 sysoptions 虚设表，该表存储有关每个 set 选项、其类别及其当前和缺省设置的信息。 sysoptions 还包含可提供每个选项的详细信息的位图。

sp_options 的语法为：

```
sp_options [ [show | help
                [, option_name | category_name |null
                    [, dflt | non_dflt | null
                        [, spid] ] ] ] ]
```

其中：

- show — 列出所有选项的当前值和缺省值，并按这些值的类别进行分组。发出带指定选项名的 sp_options show 会向您显示单个选项的当前值和缺省值。您还可以指定会话 ID，并指定是要查看具有缺省设置的选项，还是要查看具有非缺省设置的选项。
- help — 显示用法信息。通过发出不带参数的 sp_options 可得到相同的结果。
- null — 表示要查看其设置的选项。
- dflt | non_dflt | null — 表示是显示具有缺省设置的选项，还是显示具有非缺省设置的选项。
- *spid* — 指定会话 ID。使用会话 ID 查看其它会话设置。

例如，要显示如下所示的当前优化程序设置，请输入：

```
sp_options show
Category:Query Tuning
name
          currentsetting     defaultsetting         scope
-------------------------------------------------------------------------
          ------------------ --------------------  ---------
optlevel
          ase_default        ase_current            3
optgoal
          allrows_mix        allrows_mix            3
opttimeoutlimit
          10                 10                     2
repartition_degree
          1                  1                      2
scan_parallel_degree
          0                  1                      2
resource_granularity
          10                 10                     2

. . .

outer_join_costing:outer join row counts and histogramming
          0                  0                      7
join_duplicate_estimates:avoid overestimates of dups in joins
```

```
          0                   0                       7
imdb_costing:0 PIO costing for scans for in-memory database
          1                   1                       7
auto_temptable_stats:auto generation of statistics for #temptables
          0                   0                       7
use_mixed_dt_sarg_under_specialor:allow special OR in case of mixed . . .
          0                   0                       7
timeout_cart_product:timeout queries involving cartesian product and more . . .
          0                   0                       7

(81 rows affected)
```

有关详细信息，请参见 《Adaptive Server 参考手册：过程》。

任何用户都可以查询 sysoptions：

您还可以使用字符串处理或转换。例如，如果选项是数值，则可以通过输入下列内容来查询 sysoptions：

```
if (isnumeric(currentsetting))
    select@int_val = convert(int, currentsetting)
    ...
else
    select@char_val = currentsetting
    ...
```

有关 sysoptions 的详细信息，请参见 《Adaptive Server 参考手册：表》。

设置优化级别

缺省情况下， Adaptive Server 不启用一些与性能相关的优化程序设置，您必须使用 set 命令启用它们。由于不启用这些优化程序设置，因此升级到最新版本的 Adaptive Server 时，已有效运行的任何应用程序不会受优化程序更改的影响，也不会因这些更改而获益。

Adaptive Server 允许您在全局 （即，在服务器中设置优化级别）和会话级别设置优化级别。启用这些更改可提高许多应用程序的查询性能，但 Sybase 建议进行更多性能测试。

使用 *@@optlevel* 全局变量确定当前优化级别设置：

```
select @@optlevel
```

优化设置根据每个 Adaptive Server 版本所适用的优化更改进行组织。表 7-1 介绍了这些设置：

表 7-1：优化级别

参数	说明
ase_current	启用当前及之前版本中的所有优化程序更改
ase_default	禁用自 1503 ESD #1 版本以来的所有优化程序更改
ase1503esd2	启用 15.0.3 ESD #2 版本及之前版本中的所有优化程序更改
ase1503esd3	启用 15.0.3 ESD #3 版本及之前版本中的所有优化程序更改

缺省情况下，以下优化级别条件处于启用状态：

表 7-2：缺省情况下启用的优化条件

设置	说明
cr421607	支持 NULL=NULL 合并和散列连接键
cr467566	允许抽象计划和语句高速缓存一起工作
cr487450	改善 distinct 针对多表外连接和半连接的开销
cr497066	通过观察 isnull 的参数来推断其可为空性
cr500736	支持合并连接和散列连接键中的 nocase 排序顺序列
cr531199	增加优化程序考虑的有用的嵌套循环连接计划数
cr534175	如有可能，仅计算一次嵌套子查询中的 group by 工作表
cr544485	将 distinct 视图的子查询连接谓词标记为 SARG
cr545059	减少缓冲区管理器优化排序使用
cr545180	如果存在有用的索引，则避免在不使用 SARG 的情况下重新格式化
cr545379	禁止对用户强制执行的索引扫描重新格式化
cr545585	覆盖扫描 CPU 开销过大
cr545653	避免内部表缓冲区估计不足
cr545771	改善多表外连接和半连接的开销
cr546125	允许对隐式可更新游标进行非唯一索引扫描
cr552795	消除重新格式化期间不必要的重复行
cr562947	允许游标表扫描
data_page_prefetch_costing	添加聚簇行偏差
mru_buffer_costing	MRU 的清洗大小缓冲区限制

启用 ase1503esd2 时，将会启用这些优化级别条件：

表 7-3：通过 ase1503esd2 启用的优化条件

设置	说明
cr556728	便于小表之间的合并连接
cr559034	避免首选非覆盖索引扫描，而不是覆盖索引扫描
allow_wide_top_sort	允许顶部排序超出 max row size
avoid_bmo_sorts	避免仅用于缓冲区管理器优化的排序
conserve_tempdb_space	使估计的临时数据库低于资源粒度
distinct_exists_transform	将 distinct 转换为半连接
join_duplicate_estimates	避免对连接重复的估计过高
outer_join_costing	外连接行计数和直方图
search_engine_timeout_factor	对复杂 select 语句打开游标命令时花费很长时间
timeout_cart_product	涉及笛卡儿乘积和超过 5 个表的超时查询。

表 7-4 列出了在启用 ase1503esd3 或 ase_current 时启用的优化条件

表 7-4：通过 ase1503esd3 和 ase_current 启用的优化条件

设置	说明
auto_template_stats	自动生成临时表的统计信息
use_mixed_dt_sarg_under_specialor	允许在 in 或 or 列表对混合数据类型 SARG 使用特殊 or

缺省情况下，以下优化条件处于禁用状态：

表 7-5：缺省情况下禁用的优化条件

设置	说明
full_index_filter	消除非覆盖全文索引扫描策略
no_stats_distinctness	允许重复估计，而不包含统计信息

使用以下方法之一设置优化和条件级别：

- 在会话级 — 使用 set plan optlevel 设置当前会话的优化级别。这会将会话配置为使用 Adaptive Server 当前及之前版本的所有优化更改：

```
set plan optlevel ase_current
```

- 对于单个登录 — 使用 sp_modifylogin 设置登录的优化级别。sp_modifylogin 会调用用户创建的存储过程来定义优化级别。例如，如果创建此存储过程以启用 ase1503esd2 优化级别，但禁用 cr545180 优化级别：

  ```
  create proc login_proc
  as
  set plan optlevel ase1503esd2
  set cr545180 off
  go
  ```

  您可以将这些优化设置应用于任何登录。这会将 login_proc 的设置应用于用户 joe：

  ```
  sp_modifylogin joe, 'login script', login_proc
  ```

- 在服务器中 — 使用 sp_configure “optimizer level” 参数来全局设置优化级别。这会启用 Adaptive Server 当前版本之前的所有优化程序更改：

  ```
  sp_configure 'optimizer level', 0, 'ase_current'
  ```

- 在抽象计划中 — 使用 optlevel 抽象计划设置优化程序级别。此示例会为当前计划启用 Adaptive Server 当前版本之前的所有优化程序更改：

  ```
  select name from sysdatabases
  plan '(use optlevel ase_current)'
  ```

- 在使用 set 命令的会话期间 — 使用 set 命令更改当前会话的优化程序条件。优化程序条件更改可提高一些查询的性能，同时也会降低其它查询的性能。您可以通过在精细级别 （或者在查询级别）应用优化程序条件级别来获得更好的性能。Adaptive Server 按 CR 编号表示一些优化程序条件级别，而其它较新的优化程序更改通过描述性名称指定。使用 sp_options 可查看当前版本中可用选项的列表。这会启用 CR 545180 的优化条件：

  ```
  set CR545180 on
  ```

注释　启用优化条件时， Adaptive Server 会保留以前的优化级别。

优化程序诊断实用程序

通过 sp_opt_querystats 系统过程，您可以分析 Adaptive Server 优化程序生成的查询计划和影响其对查询计划的选择的因素。此分析可帮助确定查询或执行环境中的元素是否影响 Adaptive Server 执行查询的方式以及它的性能。无需运行选定的查询即可执行分析。

sp_opt_querystats 输出包括：

- showplan 生成的查询计划
- 启用的跟踪标志和开关
- set statistics io 生成的查询的 I/O 活动
- 针对查询所涉及的任何表找到的缺少的统计信息
- 由优化程序计算的估计计划开销
- 由优化程序计算的最终计划和开销估计值
- 查询的抽象计划
- 执行结果集 （例如，如果未启用 noexec）时的查询结果
- set option *show* 生成的查询的逻辑运算符树
- set statistics time 生成的查询执行时间
- 执行查询后， set statistics time 生成的查询执行时间

将 Adaptive Server 配置为运行 sp_opt_querystats

1 安装 Job Scheduler。请参见 《Job Scheduler 用户指南》。

注释 这可能要求您重新启动 Adaptive Server。

2 如果 Adaptive Server 未命名，请设置服务器名称然后重新启动 Adaptive Server：

```
sp_addserver server_name, local
```

3 运行 sp_opt_querystats 的任何登录都必须具有 js_user_role 角色，并且必须使用非空口令登录 Adaptive Server。

sp_opt_querystats 需要 sa_role：

```
grant role role_name to login_name
```

4　在 loopback 服务器上为运行 sp_opt_querystats 的所有用户创建外部登录：

```
sp_addexternlogin loopback, <login>, <password>,
<password>
```

5　Sybase 建议您将 maximum job output 的值设置为 1000000，以确保 Job Scheduler 能够捕获您的查询的所有诊断输出。

注释　如果 sp_opt_querystats 截断输出，请增加 maximum job output 的值。增加此值不会消耗额外的资源。

运行 sp_opt_querystats

使用具有 js_user_role 的登录名登录 Adaptive Server。运行 sp_opt_querystats 分析查询（诊断信息以 [BEGIN QUERY ANALYSIS] 短语开始，以 [END QUERY ANALYSIS] 结束）。

语法为：

sp_opt_querystats "*query_text*" | help [, "*diagnostic_options*" | null [, *database_name*]]

例如：

```
sp_opt_querystats "select * from pubs2..authors where
au_id = "172-32-1176"
```

请参见 《参考手册：过程》。

指定查询处理器选择

Adaptive Server 允许通过在查询批处理或查询文本中包括命令来指定以下优化选项：

- 连接中表的顺序
- 在连接优化期间同时计算的表数量
- 访问表所使用的索引
- I/O 大小
- 高速缓存策略
- 并行度

有些情况下，查询处理器不选择最佳计划。有时，查询处理器选择的计划比 “最佳”计划的开销稍高，因此您必须在维护强制选项的开销与次佳计划的较慢性能之间进行权衡。

指定连接顺序、索引、I/O 大小或高速缓存策略的命令，与 statistics io 和 showplan 之类的查询报告命令相结合，可帮助您确定查询处理器做出其选择的原因。

警告！ 应谨慎使用在本章中说明的这些选项。有些情况下，强制查询计划可能并不适合，而且可能导致性能很差。如果在您的应用程序中包含这些选项，请定期检查查询计划、 I/O 统计信息和其它性能数据。

这些选项一般用于调优和试验操作的工具，而不用作优化问题的长期解决方法。

指定连接中的表顺序

Adaptive Server 可优化连接顺序以最大程度地减少 I/O。多数情况下，查询处理器选择的顺序与 select 命令中的 from 子句的顺序并不匹配。若要强制 Adaptive Server 按列出的顺序访问各表，请使用：

```
set forceplan [on|off]
```

查询处理器仍将为每个表选择最佳访问方法。如果使用 forceplan 并指定连接顺序，则查询处理器可对表使用不同的索引，而不是使用不同的表顺序，否则它可能无法使用现有索引。

如果其它查询分析工具让您怀疑查询处理器未选择最佳连接顺序，那么您可以使用此命令作为调试辅助手段。始终通过使用 set statistics io on 并比较使用和不使用 forceplan 这两种情况下的 I/O，检验强制执行的顺序是否减少了 I/O 和逻辑读取数。

如果使用 forceplan，您的例行性能维护检查应包括检验使用 forceplan 的查询和过程是否仍需要该选项来提高性能。

您可以在存储过程的文本中包括 forceplan。

set forceplan 仅强制执行连接顺序，而不是连接类型。没有用于指定连接类型的命令；可以在服务器级或会话级禁用合并连接。

可以在会话级禁用散列连接。并且要记住，抽象计划允许完整的计划指定，包括连接顺序和连接类型。

请参见第 12 章 “创建和使用抽象计划”和第 226 页的 “启用和禁用合并连接”。

强制执行连接顺序具有以下风险：

- 误用可能会导致开销极大的查询。始终用 statistics io 在使用和未使用 forceplan 的情况下充分测试查询。
- 它需要进行维护。必须定期检查包括 forceplan 的查询和存储过程。此外，每个新版本的 Adaptive Server 都可能会消除引起合并索引强制的问题，因此，每次安装新版本时，都应检查使用强制查询计划的所有查询。

在使用 forceplan 之前：

- 检查 showplan 的输出以确定是否按预期方式使用了索引键。
- 使用 set option show normal 查找其它优化问题。
- 对索引运行 update statistics。
- 使用 update statistics 为查询中未建索引的搜索子句的搜索参数添加统计信息，尤其是复合索引中与次键相匹配的搜索参数。
- 使用 set 选项 show_missing_stats on 查找可能需要统计信息的列。
- 如果查询连接的表超过四个，请使用 set table count，这可能会生成改进的连接顺序。

  请参见第 215 页的 “指定查询处理器所考虑的表的个数”。

指定查询处理器所考虑的表的个数

在 15.0 之前的版本中， Adaptive Server 通过每次考虑二至四种排列优化连接。 15.0 及更高版本不将查询处理器限制为二至四种排列。相反，新搜索引擎引入了超时机制，以避免在优化查询方面花费过多时间。本节稍后讨论的 set table count 设置仍然影响搜索引擎查看的初始连接顺序，因此在确实发生超时时，该设置会影响最终连接顺序。如果您怀疑在搜索引擎超时时选择的是低效率连接顺序，请使用 set table count 增加考虑的表数，这会影响搜索引擎在启动排列时考虑的初始连接顺序。

Adaptive Server 仍通过每次考虑二至四个表的排列来优化连接，但如果您怀疑为连接查询选择了低效率连接顺序，可同时使用 set table count 增加考虑的表数：

```
set table count int_value
```

有效值为 0-8 ；缺省值为 0。

例如，若要指定每次优化四个表，请使用：

```
set table count 4
```

减小该值时，会减少查询处理器考虑所有可能的连接顺序的机会。增加在确定连接顺序期间每次考虑的表数会显著增加优化查询所需的时间。

由于每增加一个表，都会增加优化查询所需的时间，因此当改进的连接顺序所节省的执行时间超出额外的优化时间时，最适合使用 set table count。以下是一些示例：

- 如果您认为更佳的连接顺序可缩短查询优化和执行的总时间，尤其是对于计划位于过程高速缓存中后预计多次执行的存储过程
- 在保存抽象计划供以后使用时

使用 statistics time 检查分析和编译时间，使用 statistics io 检验改进的连接顺序是否已减少物理和逻辑 I/O。

如果增加 table count 会改进连接优化，但将占用 CPU 的时间增加到了无法接受的程度，请重写查询中的 from 子句，按 showplan 输出指示的连接顺序指定表，并使用 forceplan 运行该查询。确保您的例行性能维护检查包括检验您强制执行的连接顺序是否仍会改进性能。

指定查询索引

您可以使用 select、update 和 delete 语句中的 (index *index_name* 子句来指定用于查询的索引。也可通过指定表的名称，强制查询执行一次表扫描。语法为：

```
select select_list
   from table_name [correlation_name]
      (index {index_name | table_name } )
      [, table_name ...]
   where ...


delete table_name
   from table_name [correlation_name]
   (index {index_name | table_name }) ...


update table_name set col_name = value
   from table_name [correlation_name]
   (index {index_name | table_name})...
```

例如：

```
select pub_name, title
    from publishers p, titles t (index date_type)
    where p.pub_id = t.pub_id
    and type = "business"
    and pubdate > "1/1/93"
```

当您怀疑查询处理器选择了一个次优的查询计划时，在查询中指定索引可能有非常有用。确实指定索引时：

- 始终检查查询的 statistics io，以确定您选择的索引需要的 I/O 是否少于查询处理器的选择。
- 测试查询子句的整个有效值范围，尤其是当您：
  - 对具有倾斜数据分配的表上的查询进行调优
  - 执行范围查询，因为这些查询的访问方法对范围的大小很敏感

只在测试后确定指定的索引选项可提高查询性能时使用 (index *index_name*)。在查询中包括索引指定后，请定期检验生成的计划是否仍优于查询处理器做出的其它选择。

如果非聚簇索引的名称与表名称相同，则指定表名会导致使用非聚簇索引。您可以使用 select *select_list* from *tablename* (0) 强制执行表扫描。

指定索引具有以下风险：

- 数据分配中的更改可能会使强制索引的效率低于其它选择。
- 删除索引意味着，指定该索引的所有查询和过程都将输出一个指示此索引并不存在的信息性消息。将使用最佳替代访问方法优化查询。
- 维护工作量增加，因为您必须定期检查使用此选项的所有查询。此外，每个新版本的 Adaptive Server 都可能会消除引起合并索引强制的问题，因此每次安装新版本时，都应检查使用强制索引的所有查询。
- 在优化使用索引的查询时，该索引必须存在。不能创建一个索引然后在同一个批处理的查询中使用它。

在指定查询中的索引之前：

- 检查 showplan 输出中的 “Keys are” 消息，以确保按预期方式使用索引键。
- 使用 dbcc traceon(3604) 或 set option show normal 查找其它优化问题。
- 对索引运行 update statistics。

- 如果索引是一个组合索引，并且索引中的次键被用作搜索参数，则对它们运行 update statistics。这样做可以极大地改善查询处理器的开销估计值。为常用于搜索子句的其它列创建统计信息也可以改善估计值。
- 使用 set 选项 show_missing_stats on 查找可能需要统计信息的列。

指定查询的 I/O 大小

如果 Adaptive Server 配置用于缺省数据高速缓存或指定的数据高速缓存中的大 I/O，则查询处理器可能决定对以下对象使用大 I/O：

- 扫描整个表的查询
- 使用聚簇索引的范围查询，如使用 >、<、> *x* 和 < *y*、between 以及 like “*charstring %*” 的查询
- 扫描大量索引叶页的查询

如果表或索引使用的高速缓存被配置为 16K I/O，则一次 I/O 最多可同时读取 8 页。每个命名数据高速缓存可拥有多个池，每个池的 I/O 大小不同。指定查询的 I/O 大小将使得此查询的 I/O 在一个为该大小配置的池中进行。请参见 《系统管理指南，卷 2》，以获取有关配置指定的数据高速缓存的信息。

若要指定与查询处理器选择的 I/O 大小不同的大小，请将 prefetch 指定添加到 select、 delete 或 update 语句的 index 子句中。语法为：

```
select select_list
   from table_name
     ( [index {index_name | table_name} ]
         prefetch size)
     [, table_name ...]
where ...


delete table_name from table_name
   ( [index {index_name | table_name} ]
       prefetch size)
...


update table_name set col_name = value
   from table_name
     ( [index {index_name | table_name} ]
         prefetch size)
...
```

有效预取大小取决于页大小。如果在对象使用的数据高速缓存中不存在指定大小的池，则查询处理器将选择最佳可用大小。

如果在 au_lname 上有一个聚簇索引，则此查询在扫描数据页时将执行 16K I/O：

```
select *
from authors (index au_names prefetch 16)
    where au_lname like "Sm%"
```

如果某查询通常执行大 I/O，并且您要使用 2K I/O 检查其 I/O 性能，则可以指定 2K 大小：

```
select type, avg(price)
from titles (index type_price prefetch 2)
    group by type
```

注释　对大 I/O 的引用位于一台逻辑页为 2K 的服务器上。如果服务器的逻辑页大小为 8K，则用于 I/O 的基本单位为 8K。如果服务器的逻辑页大小为 16K，则用于 I/O 的基本单位为 16K。

索引类型和大 I/O

当使用 prefetch 指定 I/O 大小时，此指定可能会影响数据页和叶级索引页。表 7-6 说明了这些影响。

表 7-6：访问方法和预取

访问方法	执行的大 I/O
表扫描	数据页
聚簇索引	仅数据页，用于所有页锁定表 DOL 锁定表的数据页和叶级索引页
非聚簇索引	非聚簇索引的数据页和叶页

showplan 报告了用于数据页和叶级页的 I/O 大小。

请参见第 50 页的 “I/O 大小消息”。

当无法遵循 *prefetch* 指定时

大多数情况下，在指定查询中的 I/O 大小后，查询处理器会将此 I/O 大小用于查询计划中。然而，有些时候这个指定不会被遵循，不论是对于全部的查询工作还是对于单个的大 I/O 请求。

在以下情形中，您不能对查询使用大 I/O：

- 高速缓存未按指定大小的 I/O 进行配置。查询处理器会替换可用的最佳大小。
- 已使用 sp_cachestrategy 对表或索引禁用大 I/O。

在以下情形中，您不能对单个缓冲区使用大 I/O：

- 包括在此 I/O 请求中的任何页位于高速缓存的另一个池中。
- 页位于分配单元的第一个扩充上。此扩充持有用于分配单元的分配页，并只有七个数据页。
- 池内没有请求 I/O 大小的可用缓冲区。

无法执行大 I/O 时， Adaptive Server 将对查询所需的范围内的特定页执行 2K I/O。

若要确定是否遵循了 prefetch 指定，请使用 showplan 显示查询计划，并使用 statistics io 查看查询的 I/O 结果。sp_sysmon 报告对每个高速缓存的大 I/O 的请求和拒绝情况。

请参见 《性能和调优系列：使用 sp_sysmon 监控 Adaptive Server》。

设置 *prefetch*

缺省情况下，每当配置了大 I/O 池并且查询处理器确定大 I/O 会减少查询开销时，查询都会使用大 I/O。若要在会话中禁用大 I/O，请使用：

```
set prefetch off
```

若要重新启用一个大 I/O，请使用：

```
set prefetch on
```

如果使用 sp_cachestrategy 关闭了某对象的大 I/O，则 set prefetch on 不会覆盖该设置。

如果使用 set prefetch off 关闭了某会话的大 I/O，则无法通过在 select、delete 或 insert 语句中指定预取大小来覆盖该设置。

set prefetch 命令在其运行的批处理内即可生效，所以您可以在存储过程中包括此命令，以影响该过程中的查询的执行。

指定高速缓存策略

对于扫描表的数据页或非聚簇索引的叶级的查询 （覆盖查询），Adaptive Server 查询处理器选择以下两种高速缓存替换策略之一：“读取和放弃”最近使用最多的 (MRU) 策略或最近使用最少的 (LRU) 策略。

请参见 《性能和调优系列：物理数据库调优》。

查询处理器可为以下对象选择 MRU 策略：

- 执行表扫描的任何查询
- 使用聚簇索引的范围查询
- 扫描非聚族索引叶级的覆盖查询
- 嵌套循环连接中的内部表 （如果内部表大于高速缓存）
- 嵌套循环连接的外部表，因为只需要读取它一次
- 合并连接中的两个表。

若要影响对象的高速缓存策略，请执行以下操作：

- 在 select、 update 或 delete 语句中指定 lru 或 mru
- 使用 sp_cachestrategy 禁用或重新启用 mru 策略

如果指定了 MRU 策略且一个页已位于数据高速缓存中，则此页将被置于高速缓存的 MRU 端，而不是置于清洗标记处。

指定高速缓存策略只影响数据页和索引的叶页。根页和中间页始终使用 LRU 策略。

在 *select*、 *delete* 和 *update* 语句中

您可以使用 select、delete 或 update 命令中的 lru 或 mru 来指定查询的 I/O 大小。（您只根据已正确配置的高速缓存获取大小。例如，如果指定 4K，但 Adaptive Server 未使用 4K 页大小，则命令会返回 2K）：

```
select select_list
    from table_name
        (index index_name prefetch size [lru|mru])
        [, table_name ...]
where ...


delete table_name from table_name (index index_name
    prefetch size [lru|mru]) ...
```

```
update table_name set col_name = value
    from table_name (index index_name
        prefetch size [lru|mru]) ...
```

例如，要将 LRU 替换策略添加到 16K I/O 指定中，请输入：

```
select au_lname, au_fname, phone
    from authors (index au_names prefetch 16 lru)
```

请参见第 218 页的 “指定查询的 I/O 大小”。

控制大 I/O 和高速缓存策略

sysindexes 表中的状态位表明对大 I/O prefetch 或 MRU 替换策略，您是否应考虑使用表或索引。缺省情况下，二者都将启用。若要禁用或重新启用这些策略，请使用 sp_cachestrategy：

```
sp_cachestrategy dbname , [ownername.]tablename
    [, indexname | "text only" | "table only"
    [, { prefetch | mru }, { "on" | "off"}]]
```

例如，若要禁用 authors 表的 au_name_index 的大 I/O prefetch 策略，请输入：

```
sp_cachestrategy pubtune, authors, au_name_index,
   prefetch, "off"
```

重新启用 titles 表的 MRU 替换策略：

```
sp_cachestrategy pubtune, titles, "table only",
   mru, "on"
```

只有系统管理员或对象所有者才能更改或查看对象的高速缓存策略状态。

获取高速缓存策略的有关信息

若要查看对于某一给定对象生效的高速缓存策略，请用相应的数据库名和对象名执行 sp_cachestrategy：

```
sp_cachestrategy pubtune, titles
object name      index name       large IO MRU
---------------- ---------------- -------- --------
titles           NULL             ON       ON
```

showplan 输出显示用于每个对象的高速缓存策略，包括工作表。

异步日志服务

异步日志服务 (ALS) 可提高 Adaptive Server 的可扩展性，并为高端对称多处理器系统的日志记录子系统提供更高的吞吐量。

如果您的引擎数少于 4 个，则无法使用 ALS ；如果在联机引擎数少于 4 个的情况下尝试使用 ALS，则会显示错误消息。

您可以使用 sp_dboption 存储过程启用、禁用或配置 ALS：

```
sp_dboption <db Name>, "async log service",
"true|false"
```

发出 sp_dboption 后，必须在为其设置 ALS 选项的数据库中发出 checkpoint：

```
sp_dboption "mydb", "async log service", "true"
use mydb
checkpoint
```

可以使用 checkpoint 来标识一个或多个数据库，或者使用 all 子句：

```
checkpoint [all | [dbname[, dbname[, dbname.....]]]
```

若要禁用 ALS，请输入：

```
sp_dboption "mydb", "async log service", "false"
use mydb
checkpoint
-------------
```

在禁用 ALS 前，请确保数据库中没有活动用户。如果在禁用 ALS 时数据库中有活动用户，则会看到下面的错误消息：

```
错误 3647：无法将数据库置于单用户模式。请等到所有用户都从数据库
注销，然后发出 CHECKPOINT 以禁用 “async log service”。
```

使用 sp_helpdb 查看是否已在指定数据库中启用 ALS：

```
sp_helpdb "mydb"
----------
mydb             3.0 MB sa                 2
       July 09, 2002
       select into/bulkcopy/pllsort, trunc log on chkpt,
           async log service
```

有关这些存储过程的详细信息，请参见 《Adaptive Server 参考手册：过程》。

理解用户日志高速缓存 (ULC) 的体系结构

Adaptive Server 的日志记录体系结构包含用户日志高速缓存 (ULC)，从而使每个任务都拥有自己的日志高速缓存。其它任何任务都无法向此高速缓存写入信息，每当事务生成日志记录时，该任务将继续向用户日志高速缓存写入信息。当提交或中止事务时，或者当日志高速缓存已满时，ULC 将被刷新到由所有当前任务共享的通用日志高速缓存中，该通用日志高速缓存随后再写至磁盘。

刷新 ULC 是提交或中止操作的第一部分，所需的每个后续步骤都可能导致延迟或增加争用：

1 在最后一页日志页上获得锁。

2 如有必要，分配新的日志页。

3 将日志记录从 ULC 复制到日志高速缓存。

  步骤 2 和步骤 3 中的进程要求在最后一页日志页上上锁，这样可以防止任何其它任务向日志高速缓存写入信息或者执行提交或中止操作。

4 将日志高速缓存刷新到磁盘。

  步骤 4 要求重复扫描日志高速缓存，以便对脏缓冲区发出 write 命令。

  重复扫描会引起对该日志绑定到的缓冲区高速缓存螺旋锁的争用。在大量的事务负载下，对该螺旋锁的争用现象可能会相当突出。

何时使用 ALS

如果您的系统运行 4 个或更多个联机引擎，则可对至少具有以下性能问题之一的任何指定数据库启用 ALS：

- 对最后一个日志页的争用很严重

  当“Task Management”报告部分中的 sp_sysmon 输出显示非常高的值时，则可以确定最后一个日志页正处于争用状态。例如：

```
Task Management              per sec       per xact    count   % of total
-----------------------   ---------    ----------   --------  --------
Log Semaphore Contention        58.0           0.3       34801    73.1%
```

- 日志设备中存在未充分利用的带宽

注释　仅在确定了一个具有很高的事务要求的数据库后再使用 ALS，因为对多个数据库设置 ALS 可能会导致吞吐量和响应时间的意外变化。要对多个数据库配置 ALS，请先检查吞吐量和响应时间是否令人满意。

使用 ALS

有两个线程将扫描脏缓冲区 （充满尚未写入到磁盘的数据的缓冲区），复制数据，然后将数据写入到日志中。这两个线程是：

- 用户日志高速缓存 (ULC) 刷新器 — ULC 刷新器是一个系统任务线程，专门用于将任务的用户日志高速缓存刷新到通用日志高速缓存。当任务准备好提交时，用户向刷新器队列中输入提交请求。每个条目都有一个句柄，ULC 刷新器可通过句柄访问将该请求排入队列的任务的 ULC。 ULC 刷新器任务始终监控着刷新器队列，负责从队列中删除请求并通过将 ULC 页刷新到日志高速缓存来满足请求。
- 日志写入器 — ULC 刷新器完成将 ULC 页刷新到日志高速缓存的操作后，它会将任务请求排入到唤醒队列中。日志写入器将巡查日志高速缓存中的脏缓冲区链，如果发现脏缓冲区，则发出 write 命令，同时还监控那些页面已全部写入磁盘的任务的唤醒队列。由于日志写入器巡查脏缓冲区链，因此它知道缓冲区何时可以向磁盘写入。

启用和禁用合并连接

缺省情况下，对于 allrows mix 和 allrows_dss optgoal，会在服务器级启用合并连接；对于其它 optgoals （包括 allrows_oltp），则在服务器级禁用合并连接。当禁用合并连接时，服务器开销只用于未禁用的其它连接类型。若要启用全服务器范围的合并连接，请将 enable merge join 设置为 1。Adaptive Server 15.0 之前版本中的 enable sort-merge joins and JTC 配置参数不影响 15.0 及更高版本的查询处理器。

set merge_join on 命令将覆盖服务器级设置，以允许在会话或存储过程中使用合并连接。

若要启用合并连接，请使用：

```
set merge_join on
```

若要禁用合并连接，请使用：

```
set merge_join off
```

启用和禁用散列连接

缺省情况下，仅当运行 allrows_dss optgoal 时才启用散列连接。若要覆盖服务器级设置，并允许在会话或存储过程中使用散列连接，请使用 set hash_join on。

若要启用散列连接，请使用：

```
set hash_join on
```

若要禁用散列连接，请使用：

```
set hash_join off
```

启用和禁用*连接*传递闭包

在 Adaptive Server 15.0 和更高版本中，连接传递闭包始终处于启用状态，无法禁用。搜索引擎使用超时机制来避免在优化方面花费过多时间。尽管超时设置不再影响查询处理器的传递闭包的实际使用，但它仍会影响发生超时时，搜索引擎开始排列的初始连接顺序。如果您怀疑在超时时选择了次优连接顺序，您会发现此讨论非常有用。

缺省情况下，连接传递闭包不在服务器级启用，因为它可能会增加优化时间。您可以使用 set jtc on 在会话级启用连接传递闭包。此会话级命令会覆盖 enable sort-merge joins and JTC 配置参数的服务器级设置（适用于 Adaptive Server 15.0 之前的版本）。

对于执行速度很快的查询来说，即使其中涉及了若干表，连接传递闭包也可能会增加优化时间，但对执行开销的改善非常小。例如，在将连接传递闭包用于此查询时，每加入一个表，连接数量可能都会成倍增加：

```
select * from t1, t2, t3, t4, ... tN
where t1.c1 = t2.c1
and t1.c1 = t3.c1
and t1.c1 = t4.c1
...
and t1.c1 = tN.c1
```

但是，对于很大的表的连接，在连接传递闭包添加的连接顺序上花费的额外优化时间可能会得到可显著改善响应时间的连接顺序。

使用 set statistics time 查看 Adaptive Server 优化查询所用的时间。如果使用 set jtc on 运行查询会大幅增加优化时间，但同时可通过选择更好的连接顺序来改善查询执行情况，请检查 showplan、set option show_search_engine normal 或 set option show_search_engine long 输出。显式地将有用的连接顺序添加到查询文本。在不使用连接传递闭包的情况下运行查询，以获得改善的执行时间，因为省去了检查连接传递闭包生成的所有可能的连接顺序所增加的优化时间。

也可启用连接传递闭包并保存受益查询的抽象计划。如果接下来执行这些带有从启用的已保存计划进行装载的查询，则已保存的执行计划将被用于优化此查询，并且优化时间极短。

有关使用抽象计划和配置全服务器范围的连接传递闭包的详细信息，请参见 Performance and Tuning: Optimizer and Abstact Plans （《性能和调优：优化程序和抽象计划》）。

控制文字参数化

Adaptive Server 15.0.1 及更高版本允许您自动将 SQL 查询中的实际值转换为参数说明 （与变量类似）。

要在全服务器范围内启用或禁用 enable literal autoparam，请使用：

```
sp_configure "enable literal autoparam", [0 | 1]
```

其中， 1 会自动将实际值转换为参数说明，而 0 （缺省值）则禁用此功能。

使用以下命令在会话级设置文字参数化：

```
set literal_autoparam [off | on]
```

在 Adaptive Server 15.0.1 版之前的版本中，如果两个查询除一个或多个实际值不同外，其它部分完全相同，则语句高速缓存中将存储两个单独的查询计划，或者在 sysqueryplans 中额外生成两行。例如，单独存储以下查询的查询计划，即使它们几乎是完全相同的：

```
select count(*) from titles where total_sales > 100
select count(*) from titles where total_sales > 200
```

示例

如果启用自动文字参数化，则上面引用的 `select count(*)` 示例的 SQL 文本将转换为：

```
select count(*) from titles where total_sales > @@@V0_INT
```

其中， `@@@V0_INT` 是内部生成的参数名称，表示实际值 100 和 200。

SQL 文本中文字值的所有实例都将由内部生成的参数所取代。例如：

```
select substring(name, 3, 4) from sysobjects where name in
    ('systypes', 'syscolumns')
```

将转换为：

```
select substring(name, 3, 4) from sysobjects where name in
    (@@@V0_VCHAR1,@@@V1_VCHAR1)
```

替换实际值 `3`、 `4`、 `systypes` 和 `syscolumns` 的任何值组合会转换为使用相同参数的同一 SQL 文本，并在您启用语句高速缓存时共享同一查询计划。

自动文字参数化：

- 缩短了第二次执行 （和执行后续）查询时的编译时间，而无论查询中的实际值是什么。
- 减少了 SQL 文本存储空间量，包括语句高速缓存中的内存使用率以及抽象计划和查询指标的 sysqueryplans 中的行数。

- 减少了用于存储查询计划的过程高速缓存量。
- 启用后会在 Adaptive Server 内自动发生：您不需要更改向 Adaptive Server 提交查询的应用程序。

自动文字参数化的使用问题包括：

- Adaptive Server 仅对 select、 delete、 update 和 insert 进行文字参数化。对于 insert 语句， Adaptive Server 仅参数化 insert ... select 语句，而不参数化 insert ... values 语句。
- Adaptive Server 不对包括派生表的查询中的文字进行参数化。
- 由于 group by 和 order by 子句中的 （“`id + 1`”）表达式， Adaptive Server 不对与 `select id + 1 from sysobjects group by id + 1` 或 `select id + 1 from sysobjects order by id + 1` 类似的查询进行参数化。
- 对于包含的文本长度超过 16384 个字节的 SQL 语句， Adaptive Server 不在语句高速缓存中对其进行高速缓存 （不高速缓存超过 16K 的 SQL 语句）。将 SQL 语句中的文字转换为变量时，可能会显著增加 SQL 文本的大小 （包含大量文字时尤其如此）。启用自动文字参数化可能会导致 Adaptive Server 不对本应该高速缓存的某些 SQL 语句进行高速缓存。
- 如果两个 SQL 语句相同，但它们的文字值具有不同的数据类型，则不会将它们转换为匹配的 SQL 文本。例如，以下两个 SQL 语句返回相同的结果，但由于它们使用不同的数据类型，因此对它们进行参数化的方式不同：

  ```
  select name from sysobjects where id = 1
  select name from sysobjects where id = 1.0
  ```

  这些语句的参数化版本为：

```
select name from sysobjects where id = @@@V0_INT
select name from sysobjects where id = @@@V0_NUMERIC
```

提出查询的并行度建议

select 命令中 from 子句的 parallel 和 *degree_of_parallelism* 扩展允许用户限制扫描中使用的工作进程数。

为了执行 parallel 分区扫描， *degree_of_parallelism* 必须等于或大于分区数。对于并行索引扫描，可为 *degree_of_parallelism* 指定任意值。

select 语句的语法为：

```
select...
    [from {tablename}
      [(index index_name
        [parallel [degree_of_parallelism | 1]]
        [prefetch size] [lru|mru])],
      {tablename} [([index_name]
        [parallel [degree_of_parallelism | 1]
          [prefetch size] [lru|mru])] ...
```

表 7-7 显示了如何将 index 和 parallel 关键字结合使用以获得串行或并行扫描。

表 7-7：对串行和并行执行的优化程序提示

使用：	指定：
(index *tablename* parallel *N*)	并行分区扫描
(index *index_name* parallel *N*)	并行索引扫描
(index *tablename* parallel 1)	串行表扫描
(index *index_name* parallel 1)	串行索引扫描
(parallel *N*)	并行扫描，由优化程序选择表扫描或索引扫描
(parallel 1)	串行扫描，由优化程序选择表扫描或索引扫描

在为合并连接中的表指定并行度时，会影响用于表的扫描和合并连接的并行度。

如果已使用 set parallel_degree 1 命令在会话级禁用并行处理，或者使用 parallel degree 配置参数在服务器级禁用并行处理，则无法使用 parallel 选项。parallel 选项无法覆盖这些设置。

如果指定的 *degree_of_parallelism* 大于配置的最大并行度，则 Adaptive Server 会忽略提示。

如果下列任何条件成立，优化程序将忽略要求指定并行度的提示：

- 在游标的定义中使用了 from 子句。
- 在子查询的内部查询块的 from 子句中使用了 parallel，并且优化程序在子查询展平期间未将表移动到最外面的查询块。
- 该表是一个视图、系统表或是虚拟表。

- 该表是外部连接的内部表。
- 查询对表指定了 exists、 min 或 max。
- max scan parallel degree 配置参数的值设置为 1。
- 指定了未分区的聚簇索引，或者未分区的聚簇索引是唯一的并行选项。
- 包括了非聚簇索引。
- 用 OR 策略处理了查询。
- 对更新或插入操作使用了 select 语句。

查询级别 *parallel* 子句示例

若要指定单个查询的并行度，可在表名后加上 parallel。下例以串行模式执行：

```
select * from titles (parallel 1)
```

下例指定查询中使用的索引，并将并行度设置为 5：

```
select * from titles
    (index title_id_clix parallel 5)
where ...
```

若要强制执行一个表扫描，请使用表名来代替索引名。

优化目标

Adaptive Server 允许您选择最适合您的查询环境的查询优化目标：

- fastfirstrow — 优化查询，以便 Adaptive Server 尽快返回前几行。
- allrows_oltp — 优化查询，以便 Adaptive Server 使用有限数量的优化条件 （请参见第 233 页的 “优化条件”）查找好的查询计划。allrows_oltp 对纯粹的 OLTP 查询最有用。
- allrows_mixed — 优化查询，以便 Adaptive Server 使用大多数可用的优化技术 （包括 merge_join 和 parallel）来查找最佳查询计划。作为缺省策略的 allrows_mixed 在混合查询环境中最有用。

- allrows_dss — 优化查询，以便 Adaptive Server 使用所有可用的优化技术查找最佳查询计划，包括散列连接、高级集合处理和深度树计划。 allrows_dss 在 DSS 环境中最有用。

设置优化目标

您可以在服务器、会话或查询级别设置优化目标。查询级优化目标高于会话级优化目标，会话级优化目标又高于服务器级优化目标，这意味着您可以在每个级别设置不同的优化目标。

在服务器级

若要在服务器级设置优化目标，您可以执行以下操作：

- 使用 sp_configure 命令
- 修改 Adaptive Server 配置文件中的 optimization goal 配置参数

例如，若要将服务器的优化级别设置为 fastfirstrow，请输入：

```
sp_configure "optimization goal", 0, "fastfirstrow"
```

在会话级

若要在会话级设置优化目标，请使用 set plan optgoal。例如，若要将会话的优化目标设置为 allrows，请输入：

```
set plan optgoal allrows_oltp
```

若要检验会话级的当前优化目标，请输入：

```
select @@optgoal
```

在查询级

若要在查询级设置优化目标，请使用 select 或其它 DML 命令。例如，若要将当前查询的优化目标更改为 allrows_oltp，请输入：

```
select * from A order by A.a plan "(use optgoal allrows_oltp)"
```

只能在查询级别指定当您将 fastfirstrow 设置为优化目标时， Adaptive Server 可快速返回的行数。例如，可输入：

```
select * from A order by A.a plan "(use optgoal fastfirstrow 5)"
```

一些例外

一般来讲，可以使用 select、 update 和 delete 语句设置查询级别优化目标。然而：

- 虽然可以在 select ... insert 语句中设置优化目标，但不能在纯 insert 语句中设置查询级别优化目标。
- fastfirstrow 仅与 select 语句相关；与其它 DML 语句一起使用时，会导致错误。

优化条件

您可以为每个会话设置特定的优化条件。优化条件表示在 Adaptive Server 创建查询计划时，可能会考虑或可能不会考虑的特定算法或关系技术。通过将单个优化条件设置为打开或关闭状态，您可以调优当前会话的查询计划。

注释　每个优化目标都具有每个优化条件的缺省设置。重置优化条件可能会干扰当前优化目标的缺省设置并生成错误消息，但 Adaptive Server 会应用新设置。

Sybase 建议，如果必须调优特定查询，则*只能在极少情况下谨慎地*设置单个优化条件。覆盖优化目标设置可能会使查询管理变得过于复杂。始终在设置任何现有会话级 optgoal 设置*之后*设置优化条件；显式 optgoal 设置可能会使优化条件恢复为其缺省值。

请参见第 235 页的 “缺省优化条件”。

设置优化条件

使用 set 命令启用或禁用单个条件。

例如，若要启用散列连接算法，请输入：

```
set hash_join 1
```

若要禁用散列连接算法，请输入：

```
set hash_join 0
```

若要启用一个选项并禁用另一选项，请输入：

```
set hash_join 1, merge_join 0
```

条件说明

此处介绍的多数条件可决定特定查询引擎运算符是否可在优化程序选择的最终计划中使用。

优化条件有：

- hash_join — 确定查询处理器是否可使用散列连接算法。散列连接可能会消耗更多运行时资源，但如果连接列没有有用的索引，或与连接表中的行数的乘积相比较而言，有大量的行符合连接条件，则散列连接非常有用。
- hash_union_distinct — 确定查询处理器是否可使用 hash union distinct 算法，该算法在多数行都不同时效率较低。

- merge_join — 确定查询处理器是否可以使用 merge join 算法，该算法依赖已排序的输入。如果对合并键上的输入进行排序 （例如从索引扫描中）， merge_join 最有用。如果对输入进行排序要求使用 sort 运算符，则会降低 merge_join 的有用性。
- merge_union_all — 确定查询处理器是否可对 union all 使用合并算法。merge_union_all 保留联合输入的结果行的排序。如果输入已经过排序且父运算符 （如合并连接）受益于该排序，则 merge_union_all 尤其有用。否则， merge_union_all 可能需要排序运算符，而这会降低效率。
- merge_union_distinct — 确定查询处理器是否可对 union 使用合并算法。merge_union_distinct 与 merge_union_all 类似，但它不保留重复行。merge_union_distinct 需要已排序的输入并提供已排序的输出。
- multi_table_store_ind — 确定查询处理器是否可对多表连接的结果进行重新格式化。使用 multi_tablet_store_ind 可能会增加工作表的使用率。
- nl_join — 确定查询处理器是否可使用嵌套循环连接算法。
- opportunistic_distinct_view — 确定查询处理器在强制执行差别运算时是否可使用更为灵活的算法。
- parallel_query — 确定查询处理器是否可使用并行查询优化。
- store_index — 确定查询处理器是否可使用重新格式化，这可能会增加工作表的使用率。
- append_union_all — 确定查询处理器是否可使用 append union all 算法。
- bushy_search_space — 确定查询处理器是否可使用 bushy-tree-shaped 查询计划，这可能会增加搜索空间，但会提供更多查询计划选项以改进性能。
- distinct_hashing — 确定查询处理器是否可使用散列算法来消除重复项，如果与行数相比，不同值很少，则此算法非常高效。
- distinct_sorted — 确定查询处理器是否可使用单传算法来消除重复项。distinct_sorted 依赖排序的输入流，如果其输入未经过排序，则可能会增加排序运算符数。
- group-sorted — 确定查询处理器是否可使用随即分组算法。 group-sorted 依赖分组列上排序的输入流，并在其输出中保持该顺序。
- distinct_sorting — 确定查询处理器是否可使用排序算法来消除重复项。如果输入未经过排序 （例如，如果没有索引），并且排序算法生成的输出顺序可能有用，则 distinct_sorting 会很有用；例如在合并连接中。

- group_hashing — 确定查询处理器是否可使用 group hashing 算法来处理集合。
- index_intersection — 确定查询处理器是否可将多次索引扫描的交集用作搜索空间中的查询计划的一部分。

如果禁用关系运算符的所有算法，则查询处理器会重新启用一种缺省算法。例如，如果禁用所有连接算法 （nl_join、 m_join 和 h_join），则查询处理器会启用 nl_join。

查询处理器还可出于语义原因重新启用 nl_join：例如，如果连接表未通过等值连接建立连接。

缺省优化条件

每个优化目标 （fastfirstrow、 allrows_oltp、 allrows_mixed、 allrows_dss）对每个优化条件都有一个缺省设置 （开 (1) 或关 (0)）。例如，对于 fastfirstrow 和 allrows_oltp， merge_join 的缺省设置为关 (0)，对于 allrows_mixed 和 allrows_dss，其缺省设置为开 (1)。有关每个优化条件的缺省设置的列表，请参见表 7-8。

Sybase 建议，在更改优化条件前，应重置优化目标并评估性能。仅在必须调优特定查询时才更改优化条件。

表 7-8：优化条件的缺省设置

优化条件	fastfirstrow	allrows_oltp	allrows_mixed	allrows_dss
append_union_all	1	1	1	1
bushy_search_space	0	0	0	1
distinct_sorted	1	1	1	1
distinct_sorting	1	1	1	1
group_hashing	1	1	1	1
group_sorted	1	1	1	1
hash_join	0	0	0	1
hash_union_distinct	1	1	1	1
index_intersection	0	0	0	1
merge_join	0	0	1	1
merge_union_all	1	1	1	1
multi_gt_store_ind	0	0	0	1
nl_join	1	1	1	1
opp_distinct_view	1	1	1	1
parallel_query	1	0	1	1
store_index	1	1	1	1

限制优化时间

可使用 optimization timeout limit 配置参数来限制 Adaptive Server 用于优化查询的时间。 optimization timeout limit 将 Adaptive Server 可用于优化查询的时间指定为用于处理查询的总时间的百分比。

仅当出现以下情况时，才会激活超时：

- 已将至少一个完整计划保留为最佳计划，以及
- 已超出优化超时限制。

使用 sp_configure 在服务器级设置 optimization timeout limit。例如，要将优化时间限制为总查询处理时间的 10%，请输入：

```
sp_configure "optimization timeout limit", 10
```

若要在会话级设置 optimization timeout limit，请使用：

```
set plan optimeoutlimit n
```

此命令会覆盖服务器设置。

缺省值为 10% ；您可以指定从 1 到 1000 的任何值。

对于存储过程编译中的服务器级缺省超时值，在服务器级有一个单独的配置参数 optimization timeout limit。缺省值为 40% ；您可以指定从 1 到 4000 的任何值。

有关 optimization timeout limit 的详细信息，请参见第 14 页的 “限制优化查询所用时间”。

控制并行优化

以并行方式执行查询的目的是获取最快的响应时间，即便这可能增加服务器的总工作量。

为了启用和控制并行处理， Adaptive Server 提供了以下配置参数：

- number of worker processes
- max parallel degree
- max resource granularity
- max repartition degree

除 number of worker processes 外，以上每个参数都可在服务器级和会话级设置。若要查看参数的当前会话级值，请使用 select 命令。例如，若要查看 max resource granularity 的当前值，请输入：

```
select @@resource_granularity
```

注释　在会话级设置或查看时，这些参数不包括 “max”。

number of worker processes

使用 number of worker processes 为所有同时运行的并行查询指定 Adaptive Server 可随时使用的工作进程的最大数目。

number of worker processes 只是个全服务器范围的配置参数；使用 sp_configure 可设置该参数。例如，若要将工作进程的最大数目设置为 200，请输入：

```
sp_configure "number of worker processes", 200
```

指定可用于并行处理的工作进程数

使用 max parallel degree 可指定每个查询所允许的工作进程的最大数目。可在服务器或会话级配置 max parallel degree。

例如，若要在服务器级将 max parallel degree 设置为 60，请输入：

```
sp_configure "max parallel degree", 60
```

若要在会话级将 max parallel degree 设置为 60，请输入：

```
set parallel_degree 60
```

max parallel degree 的值必须等于或小于 number of worker processes 的当前值。将 max parallel degree 设置为 1 将关闭并行处理 — Adaptive Server 以串行方式扫描所有表和索引。若要启用并行分区扫描，请将 max parallel degree 设置为等于或大于您要查询的表中的分区数。

max resource granularity

使用 max resource granularity 可指定 Adaptive Server 可分配给单个查询的总内存的百分比。您可以在服务器级或会话级设置此参数。

例如，若要在服务器级将 max resource granularity 设置为 35%，请输入：

```
sp_configure "max resource granularity", 35
```

若要在会话级将 max resource granularity 设置为 35%，请输入：

```
set resource_granularity 35
```

此参数的值会影响查询优化程序对查询运算符的选择。如果将 max resource granularity 设置为较低的值，则无法选择许多基于散列和排序的运算符。 max resource granularity 还影响调度算法。

max repartition degree

使用 max repartition degree 可提供查询处理器可用于对数据流进行分区的一些工作进程的建议。您可以在服务器或查询级别设置 max repartition degree。

注释 max repartition degree 的值只是一个建议；由查询处理器来确定最佳数量。

当查询的表未经分区时， max repartition degree 最有用，但对结果数据流进行分区可允许并发 SQL 操作，从而提高性能。

例如，若要在服务器级将 max repartition degree 设置为 15，请输入：

```
sp_configure "max repartition degree", 15
```

若要在会话级将 max repartition degree 设置为 15，请输入：

```
set repartition_degree 15
```

max repartition degree 的值不能超过 max parallel degree 的当前值。Sybase 建议将此参数的值设置为等于或小于可并行工作的 CPU 或磁盘系统数。

用于小表的并发优化

对于包含 15 页或更少页的 DOL 锁定表，如果表中存在有用的索引，则 Adaptive Server 不考虑表扫描。它而是始终选择与查询中任何可优化搜索参数匹配且开销最低的索引。索引扫描要求的锁定将提供较高的并发性并降低死锁的可能性，但比表扫描要求稍多一些的 I/O。

如果小表的并发性不成问题，并且您想要优化 I/O，请使用 sp_chgattribute 禁用并发优化。例如，若要禁用表的并发优化，请执行以下命令：

```
sp_chgattribute tiny_lookup_table,
  "concurrency_opt_threshold", 0
```

通过禁用并发优化，查询处理器可在需要较少 I/O 时选择表扫描。

也可以增加表的并发优化阈值。以下命令将表的并发优化阈值设置为 30 页：

```
sp_chgattribute lookup_table,
  "concurrency_opt_threshold", 30
```

并发优化阈值的最大值为 32,767。将该值设置为 -1 会对任意大小的表强制执行并发优化；当选择表扫描而不是选择索引访问，并且由此引起的锁定导致争用或死锁现象增加时，此设置可能会很有用。

当前设置存储在 systabstats.conopt_thld 中，并作为 optdiag 输出的一部分显示。

更改锁定方案

并发优化只影响 DOL 锁定表。表 7-9 显示了更改锁定方案的影响。

表 7-9：变更表对并发优化设置的影响

更改自	对存储值的影响
从所有页锁定更改为仅数据锁定	设置为 15，即缺省值
从仅数据锁定更改为所有页锁定	设置为 0
从一个仅数据锁定方案更改为另一个仅数据锁定方案	保持的配置值

第 8 章

游标的优化

本章讨论与游标有关的性能问题。游标是一种一次访问一行 （或几行，如果使用 set cursors rows） SQL select 语句结果的机制。由于游标使用与普通面向集的 SQL 不同的模式，因此游标使用内存和持有锁的方式会影响应用程序的性能。游标性能问题具体包括页级和表级锁定、网络资源及处理指令的开销。

定义

游标是与 select 语句关联的符号名。使用它可一次访问一行 select 语句的结果。图 8-1 显示了一个正在访问 authors 表的游标。

图 8-1：游标示例

您可以将游标视为 select 语句的结果集上的 “句柄”。使用它可一次检查并可能一次操作一行。

面向集的编程与面向行的编程

SQL 被认为是一种面向集的语言。 Adaptive Server 在面向集的模式下运行时效率极高。根据 ANSI SQL 标准，必须采用游标；需要它们时，它们可提供非常强大的功能。但它们对系统性能有负面影响。

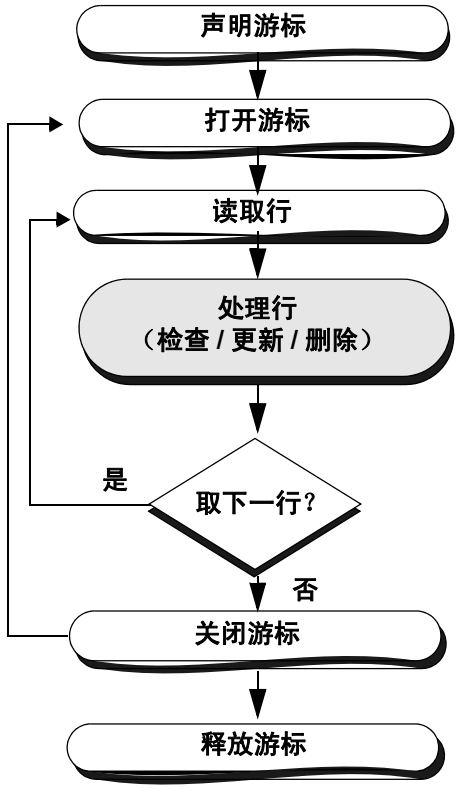
例如，此查询对所有符合 where 子句中条件的行执行相同的操作：

```
update titles
    set contract = 1
where type = 'business'
```

优化程序会找到最有效的方式来执行更新操作。相反，游标会检查每一行，并对符合条件的行进行单行更新。应用程序先为 select 语句声明一个游标，然后打开游标，读取一行，处理该行，然后转至下一行，依此类推。依据当前行中的不同值，应用程序可能会执行完全不同的操作，而且服务器对游标应用资源的总体利用可能不如服务器的结果集级别的操作有效。但游标的灵活性要好于面向集的编程。

图 8-2 显示了使用游标的相关步骤。游标的功能是到达中间的方框，在那里，用户或应用程序代码会对行进行检查，并根据行的值决定下一步操作。

图 8-2：游标流程图



示例

以下是一个游标的简单示例，上图中所示的“处理行”步骤以伪代码形式表示：

```
declare biz_book cursor
    for select * from titles
    where type = 'business'
go
open biz_book
go
fetch biz_book
go
/* Look at each row in turn and perform
** various tasks based on values,
```

```
** and repeat fetches, until
** there are no more rows
*/
close biz_book
go
deallocate cursor biz_book
go
```

依据行的内容，用户可能需删除当前行：

```
delete titles where current of biz_book
```

或更新当前行：

```
update titles set title="The Rich
    Executive's Database Guide"
where current of biz_book
```

每一阶段所需的资源

游标占用内存并需要表、数据页和索引页上的锁。打开游标时，系统会为游标分配内存，用以存储生成的查询计划。打开游标时， Adaptive Server 持有意图表锁，有时还会持有行锁或页锁。图 8-3 显示在游标操作期间锁的持续时间。

图 8-3：游标语句对资源的占用

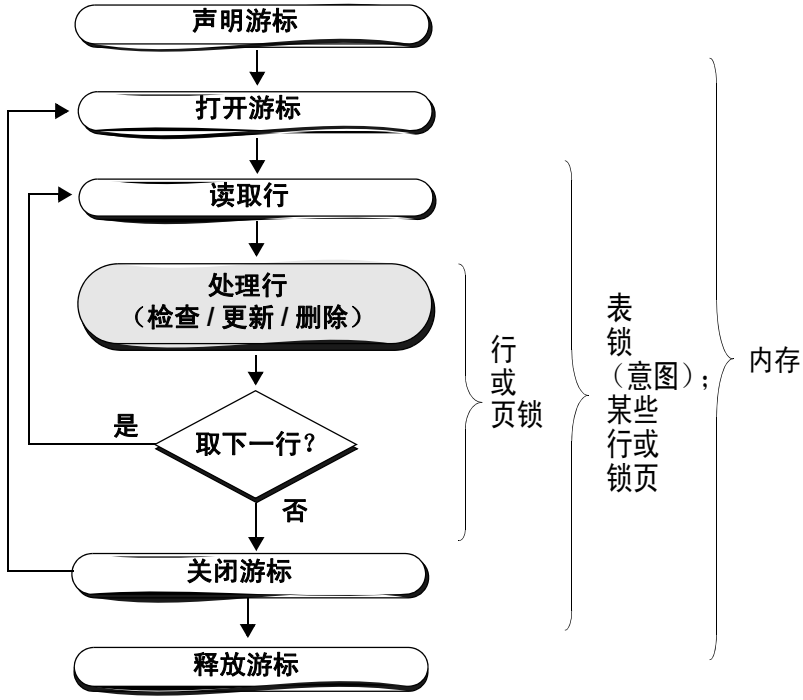


图 8-3 和表 8-1 中的内存资源说明指的是 isql 或 Client-Library™ 发送的查询的即席游标。对于其它类型的游标，锁是相同的，但根据所用游标类型的不同，内存分配和释放也稍有不同，如第 246 页的 “内存占用及执行游标”中所述。

表 8-1：isql 和 Client-Library 客户端游标对锁和内存的占用

游标命令	资源占用
declare cursor	声明游标时， Adaptive Server 仅使用足够存储查询文本的内存。
open	打开游标时， Adaptive Server 会为游标分配内存，以存储生成的查询计划。服务器会优化查询，遍历索引，并建立内存变量。除非需要建立工作表，否则服务器此时还不会访问行。但它会建立所需的表级锁 （意图锁）。行和页锁定行为取决于隔离级别、服务器配置和查询类型。 有关详细信息，请参见 Performance and Tuning Series: Locking and Concurrency Control（《性能和调优系列：锁定和并发控制》）。
fetch	在执行 fetch 时， Adaptive Server 会获取所需的行，并将指定的值读入游标变量或将该行发送到客户端。如果游标需要持有行或页的锁，则在 fetch 将游标移出行或页、或者游标被关闭之前，锁始终被持有。依据游标写入方式的不同，锁可为共享锁或更新锁。
close	关闭游标时， Adaptive Server 会释放锁和一些内存分配。必要时，可再次打开游标。
deallocate cursor	释放游标时， Adaptive Server 会释放该游标占用的其余内存资源。若要重新使用游标，请再次声明它。

内存占用及执行游标

表 8-1 中的 declare cursor 和 deallocate cursor 的说明指的是 isql 或 Client-Library 发送的即席游标。其它类型游标分配内存的方式与之不同：

- 对于在存储过程*上*声明的游标，只在 declare cursor 时分配少量内存。在存储过程上声明的游标由 Client-Library 或预编译程序发送，这些游标称为 execute cursors （执行游标）。
- 对于在存储过程*内*声明的游标，已有可供存储过程使用的内存，declare 语句不需要额外的内存。

游标模式

有两种游标模式：只读模式和更新模式。如其名字所示，只读游标只能显示来自 select 语句的数据；更新游标则可用于执行定位型更新和删除。

只读模式使用共享页锁或共享行锁。如果 read committed with lock 设置为 0，并且查询在隔离级别 1 运行，则它使用瞬间锁，并且在下次获取前，不持有页锁或行锁。

指定 for read only 时，或者游标的 select 语句使用 distinct、group by、union 或集合函数以及在某些情形下使用 order by 子句时，只读模式有效。

更新模式使用更新页锁或更新行锁。它在下列情况下有效：

- 指定 for update 时。
- select 语句不包括 distinct、group by、union、子查询、集合函数或 at isolation read uncommitted 子句时。
- 指定 shared 时。

如果指定了 *column_name_list*，则仅那些列是可更新的。

有关详细信息，请参见 Performance and Tuning Series: Locking and Concurrency Control （《性能和调优系列：锁定和并发控制》）。

声明游标时应指定游标模式。如果 select 语句包括特定选项，则即使您将游标声明为更新模式，也无法对其进行更新。

游标对索引的使用及要求

游标中使用查询时，它需要或选择的索引可能与在游标外部使用的相同查询所需要或选择的索引不同。

所有页锁定表

对于只读游标，以隔离级别 0 运行的查询 （脏读）需要唯一索引。隔离级别 1 或 3 的只读游标应能生成与游标外部的 select 语句相同的查询计划。

可更新游标的索引要求意指可更新游标可使用与只读游标不同的查询计划。可更新游标有下列索引要求：

- 如果没有将游标声明为更新模式，则最好使用唯一索引，而不要使用表扫描或非唯一索引。
- 如果将游标声明为更新模式而*没有* for update of 列表，则所有页锁定表上需要唯一索引。如果不存在唯一索引，将出现错误。
- 如果将游标声明为更新模式，且有 for update of 列表，则在所有页锁定表上只能选择*不含*任何来自该列表的列的唯一索引。如果没有符合条件的唯一索引，将出现错误。

当涉及到游标时，包含 IDENTITY 列的索引可视为唯一索引，即使未将该索引声明为唯一索引。在某些情况下，必须向索引添加 IDENTITY 列以使索引唯一，否则优化程序可能会被迫为游标查询选择次佳的查询计划。

DOL 锁定表

在 DOL 锁定表中，固定行 ID 用于游标扫描定位，所以对脏读或可更新游标不要求使用唯一索引。可更新游标中存在不同查询计划的唯一原因是，如果 for update of 列表中仅包含来自有用索引的列，则表扫描会很有用。

为避免 Halloween 问题进行表扫描

Halloween 问题是一种更新异常情况，当使用游标的客户端更新游标结果集行的一列，而该列又定义了行从表中返回的顺序时，就会出现这种问题。例如，如果游标要使用 last_name、 first_name 的索引，并且要更新这些列中的一个，则该行可能会在结果集中再次出现。

为避免 DOL 锁定表上出现 Halloween 问题，当 for update 子句的列列表中包含有其它用途的索引的列时， Adaptive Server 会选择表扫描。

对于未使用 for update 子句声明的隐式可更新游标和 for update 子句中的列列表为空的游标，更新该游标使用的索引中的列的游标可能会遇到 Halloween 问题。

比较有无游标情况下的系统性能

本节检验按以下两种不同方式写入的存储过程的性能：

- 不使用游标 — 此过程扫描表三次，并更改每本书的价格。
- 使用游标 — 此过程只扫描表一次。

在这两个示例中， titles(title_id) 上都有一个唯一索引。

无游标的存储过程举例

以下举例说明无游标的存储过程：

```
/* Increase the prices of books in the
** titles table as follows:
**
** If current price is <= $30, increase it by 20%
** If current price is > $30 and <= $60, increase
** it by 10%
** If current price is > $60, increase it by 5%
**
** All price changes must take effect, so this is
** done in a single transaction.
*/

create procedure increase_price
as

    /* start the transaction */
    begin transaction
    /* first update prices > $60 */
    update titles
        set price = price * 1.05
        where price > $60

    /* next, prices between $30 and $60 */
    update titles
        set price = price * 1.10
    where price > $30 and price <= $60

    /* and finally prices <= $30 */
    update titles
    set price = price * 1.20
    where price <= $30

    /* commit the transaction */
    commit transaction

return
```

有游标的存储过程举例

此过程对基础表所作的更改与无游标写入的过程对基础表所作的更改相同，但它使用游标代替了面向集的编程。读取、检查和更新每行时，都应在相应数据页上持有一个锁。而且，如注释所示，由于不存在显式事务，因此每个更新操作都随时提交。

```
/* Same as previous example, this time using a
** cursor.Each update commits as it is made.
*/
create procedure increase_price_cursor
as
declare @price money

/* declare a cursor for the select from titles */
declare curs cursor for
    select price
    from titles
    for update of price

/* open the cursor */
open curs

/* fetch the first row */
fetch curs into @price

/* now loop, processing all the rows
** @@sqlstatus = 0 means successful fetch
** @@sqlstatus = 1 means error on previous fetch
** @@sqlstatus = 2 means end of result set reached
*/
while (@@sqlstatus != 2)
begin
    /* check for errors */
    if (@@sqlstatus = 1)
    begin
        print "Error in increase_price"
        return
    end

    /* next adjust the price according to the
    ** criteria
    */
    if @price > $60
    select @price = @price * 1.05
    else
    if @price > $30 and @price <= $60
    select @price = @price * 1.10
```

```
        else
        if @price <= $30
        select @price = @price * 1.20

        /* now, update the row */
        update titles
        set price = @price
        where current of curs

        /* fetch the next row */
        fetch curs into @price
    end

    /* close the cursor and return */
    close curs
    return
```

您认为是执行了三次表扫描的那个过程性能好，还是使用游标只进行了一次扫描的那个过程性能好呢？

有游标与无游标的性能比较

表 8-2 显示了从一个有 5000 行的表搜集的统计信息。尽管只进行了一次表扫描，但游标代码要长 4 倍以上。

表 8-2：一个有 5000 行的表的执行时间示例

过程	访问方法	时间
increase_price	使用三次表扫描	28 秒
increase_price_cursor	使用游标，一次表扫描	125 秒

此类测试的结果可能会有很大差异。在网络繁忙、有大量活动数据库用户及多用户访问同一张表的系统上，测试结果的差异体现得最明显。

除锁定外，游标参与的网络活动比集合操作要多，并导致了处理指令的开销。应用程序需要就查询的每个结果行与 Adaptive Server 通信。这就是游标代码要花费比扫描表三次的代码长得多的时间才能完成的原因。

游标性能问题包括：

- 页级和表级锁定
- 网络资源
- 处理指令的开销

如果有一个集级编程与之相当，即使包括多次表扫描，它也可能是更可取的方法。

使用只读游标锁定

以下是一段游标代码，可用于显示在游标生命期内的每一点所建立的锁。下例中使用了一个所有页锁定表。执行图 8-4 中的代码，然后在箭头处暂停，执行 sp_lock，然后检查到位的锁。

图 8-4：只读游标和锁定试验输入

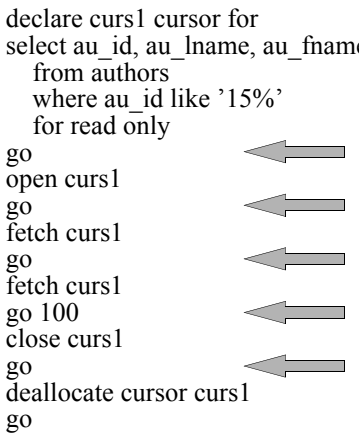


表 8-3 显示了结果。

表 8-3：游标在数据页和索引页上持有的锁

事件	数据页
执行 declare 后	没有与游标相关的锁。
执行 open 后	authors 上的共享意图锁。
执行第一次 fetch 后	authors 上的共享意图锁和 authors 中某页上的共享页锁。
执行 100 次读取后	authors 上的共享意图锁和 authors 中不同页上的共享页锁。
执行 close 后	没有与游标相关的锁。

如果在获取最后一行结果集后发出另一 fetch 命令，则会释放最后一页上的锁，因此将不存在与游标相关的锁。

对于 DOL 锁定表：

- 如果游标查询在隔离级别 1 运行，并且 read committed with lock 设置为 0，则不会看到任何页锁或行锁。值从页或行中复制完后，锁随即被释放。
- 如果 read committed with lock 设置为 1，或如果查询以隔离级别 2 或 3 运行，则在表 8-3 指示共享页锁处会看到共享页锁或共享行锁。如果表使用数据行锁定方案，则 sp_lock 报告将包括所读取行的 ID。

隔离级别和游标

打开游标时，会对游标的查询计划进行编译和优化。您不能打开游标，然后使用 set transaction isolation level 更改游标运行的隔离级别。

由于使用隔离级别 0 的游标的编译方法与使用其它隔离级别的游标的编译方法不同，因此不能以隔离级别 0 打开一个游标，然后以隔离级别 1 或 3 打开或读取它。同样，也不能以隔离级别 1 或 3 打开一个游标，然后以隔离级别 0 读取。试图以不兼容级别读取游标会引发错误消息。

一旦以特定的隔离级别打开游标，要更改隔离级别，就必须首先释放游标。游标打开时更改隔离级别的影响如下：

- 尝试关闭游标并以另一个隔离级别重新打开它，将会失败并会产生错误消息。
- 在未经过从关闭到重新打开游标这一过程的情况下，尝试更改隔离级别对使用中的隔离级别没有任何影响，且不会产生错误消息。

可在游标中包含 at isolation 子句，以指定隔离级别。下例中的游标可以隔离级别 1 声明，并从隔离级别 0 读取，因为查询计划与隔离级别兼容：

```
declare cprice cursor for
select title_id, price
    from titles
    where type = "business"
    at isolation read uncommitted
```

分区堆表和游标

对未分区堆表的游标扫描可读取直至最后对表执行的插入操作之前的所有数据 （也包括最后插入的数据），即使插入活动发生在游标扫描开始后。

如果堆表已分区，则数据可以插入到许多页链中的一个页链。其物理插入点可能在游标扫描当前位置的前方或后方。这意味着针对分区表的游标扫描*无法*保证可扫描到最后对该表进行的插入操作。

注释 如果游标操作要求所有插入都在单个页链末端进行，请*不要*对游标扫描中使用的表进行分区。

游标优化提示

以下是游标优化的几点提示：

- 使用游标，而不是即席查询来优化游标选择。
- 使用 union 或 union all，而不是 or 子句或 in 列表。
- 声明游标的意图。
- 在 for update 子句中指定列名。
- 如果要将行返回客户端，应读取多行。
- 在所有提交和回退中保持游标始终打开。
- 在单个连接上打开多个游标。

使用游标优化游标选择

独立 select 语句的优化方式可能与隐式或显式可更新游标中的同一 select 语句大不相同。开发使用游标的应用程序时，不要使用独立 select 语句，而应使用游标不断检查查询计划和 I/O 统计信息。特别要注意的是，可更新游标的索引限制要求使用的访问方法颇为不同。

使用 *union*，而不是 *or* 子句或 *in* 列表

游标不能使用由 OR 策略生成的行 ID 的动态索引。在独立 select 语句中使用 OR 策略的查询通常使用只读游标执行表扫描。可更新游标可能需要使用唯一索引，并且还需要依次访问每个数据行，以评估查询子句。

使用 union 的只读游标在其被声明后创建工作表，并对其排序，以去除重复值。读取是在工作表上进行的。使用 union all 的游标可返回重复值，且不需要有工作表。

声明游标的意图

始终声明游标的意图：只读或可更新。这样一来，就可对并发影响有更大的控制权。如果不指定意图，则 Adaptive Server 会替您作出决定，并且它往往会选择可更新游标。可更新游标使用更新锁，从而禁止其它更新锁或排它锁。如果更新活动更改一个索引列，优化程序可能需要为查询选择一个表扫描，从而可能导致棘手的并发问题。请务必为使用可更新游标的查询检查查询计划。

在 *for update* 子句中指定列名

对于有一些列在游标 select 语句的 for update 子句中列出的表，Adaptive Server 会获取所有这类表的页或行上的更新锁。如果 for update 子句未包含在游标声明中，则 from 子句中引用的所有表都会获取更新锁。

如下查询包含 for update 子句中的列名，但由于 for update 子句中提及了 price，因此它将只获取 titles 表上的更新锁。该表使用所有页锁定方案。authors 和 titleauthor 上的锁是共享页锁：

```
declare curs3 cursor
for
select au_lname, au_fname, price
    from titles t, authors a,
        titleauthor ta
where advance <= $1000
    and t.title_id = ta.title_id
    and a.au_id = ta.au_id
for update of price
```

表 8-4 显示了以下各项的影响：

- 省略整个 for update 子句 — 无 shared 子句
- 在 for update 子句中省略列名
- 在 for update 子句中包含要更新的列名
- 使用 for update of price 时，在 titles 表名的后面添加了 shared

在此表中，重点说明了两种 for update 子句的其它锁或限制性更强的锁。

表 8-4：for update 和 shared 子句对游标锁定的影响

子句	*titles*	*authors*	*titleauthor*
无	sh_page on data	sh_page on index sh_page on data	sh_page on data
for update	*updpage on index* updpage on data	*updpage on index* *updpage on data*	*updpage on data*
for update of price	updpage on data	sh_page on index sh_page on data	sh_page on data
for update of price + shared	sh_page on data	sh_page on index sh_page on data	sh_page on data

使用 *set cursor rows*

SQL 标准为游标指定了逐行读取方式，结果浪费了网络带宽。使用 set cursor rows 查询选项和 Open Client 的透明缓冲读取可提高性能：

```
ct_cursor(CT_CURSOR_ROWS)
```

为频繁执行的使用游标的应用程序选择返回的行数时，必须谨慎从事 — 应将它们调优以适应网络状况。

有关此过程的说明，请参见 《性能和调优系列：基础知识》。

在所有提交和回退中保持游标始终打开

ANSI 在每项事务结束时关闭游标。Transact-SQL 为必须符合 ANSI 行为的应用程序提供了 set 选项 close on endtran。但在缺省情况下，该选项是关闭的。除非必须符合 ANSI 的要求，否则应将此选项关闭，以保持并发性和吞吐量。

如果必须符合 ANSI 标准，请决定如何处理对 Adaptive Server 的影响。应在单个事务中执行大量更新或删除吗？还是应该保持事务简短呢？

如果选择保持事务简短，则关闭及打开游标可能会影响吞吐量，因为 Adaptive Server 需要在每次打开游标时重新实现结果集。如果选择在每次事务中执行更多工作，则可能会引发并发问题，因为查询将持有锁。

在单个连接上打开多个游标

一些开发人员使用两个或更多来自 DB-Library™ 的连接模拟游标。一个连接执行选择操作，另一个连接在相同表上执行更新或删除操作。这样做造成应用程序死锁的可能性非常大。例如：

- 连接 A 持有页的共享锁。只要存在 Adaptive Server 中待执行的行，共享锁就一直保持在当前页。
- 连接 B 请求同一页的排它锁，然后等待。
- 在调用去除共享锁所需的任何逻辑之前，应用程序等待连接 B 的成功完成。而这永远不会发生。

因为连接 A 从未请求连接 B 持有的锁，所以这不是服务器方死锁。

第 9 章

查询处理指标

概述

查询处理 (QP) 指标确定并比较查询执行过程中的实验计量值。执行查询时，此命令与 QP 指标中一组作为比较基础的已定义指标关联。

捕捉的指标包括：

- CPU 运行时间 — 执行查询所用的时间 （以毫秒为单位）。
- 经历时间 — 自编译后到执行结束的时间 （以毫秒为单位）。
- 逻辑 I/O — 逻辑 I/O 读取次数。
- 物理 I/O — 物理 I/O 读取次数。
- 计数 — 一个查询的执行次数。
- 中断计数 — 因超过资源限制而导致 Resource Governor （资源管理器）中断查询的次数。

除计数和中断计数外，每个指标都具有三个值：最小值、最大值、平均值。

执行 QP 指标

可以在服务器级别或会话级别激活和使用 QP 指标。

在服务器级别，将 sp_configure 与 enable metrics capture 选项一起使用。将即席语句的 QP 指标直接捕获到系统目录中，而将存储过程中语句的 QP 指标保存在过程高速缓存中。在刷新了语句高速缓存中的存储过程或查询后，将分别捕获的指标写入系统目录中。

```
sp_configure "enable metrics capture", 1
```

在会话级别，使用 set metrics_capture on/off：

```
set metrics_capture on/off
```

访问指标

QP 指标始终在缺省组 （即各个数据库中的组 1）中捕获。使用 sp_metrics 'backup' 可将保存的 QP 指标从缺省运行组移动到备份组。通过对 sysquerymetrics 视图使用 select 语句和 order by 来访问指标信息。有关详细信息，请参见第 260 页的 “sysquerymetrics 视图”。

还可以使用数据操作语言 (DML) 语句对指标信息进行排序并标识特定查询以进行评估。请参见属于 Adaptive Server Enterprise 文档集一部分的《组件集成服务用户指南》中的第 2 章 “了解组件集成服务”。

sysquerymetrics 视图

字段	定义
uid	用户 ID
gid	组 ID
id	唯一 ID
hashkey	SQL 查询文本上的散列键
sequence	SQL 代码文本需要多行时行的序列号
exec_min	最短执行时间
exec_max	最长执行时间
exec_avg	平均执行时间
elap_min	最短经历时间
elap_max	最长经历时间

字段	定义
elap_avg	平均经历时间
lio_min	最小逻辑 I/O
lio_max	最大逻辑 I/O
lio_avg	平均逻辑 I/O
pio_min	最小物理 I/O
pio_max	最大物理 I/O
pio_avg	平均物理 I/O
cnt	已经执行的查询次数
abort_cnt	由于超过资源限制 Resource Governor （资源管理器）中止查询的次数
qtext	查询文本

此视图中的平均值使用以下公式计算得出：

```
new_avg = (old_avg * old_count + new_value )/ (old_count + 1) = old_avg +
round((new_value - old_avg)/(old_count + 1))
```

sysquerymetrics 视图示例如下：

select * from sysquerymetrics

```
uid   gid   hashkey   id   sequence   exec_min
exec_max   exec_avg   elap_min   elap_max   elap_avg   lio_min
lio_max   lio_avg   pio_min   pio_max   pio_avg   cnt   abort_cnt
qtext
----------- ----------- ----------- ----------- ----------- -----------
----------- ----------- ----------- ----------- ----------- -----------
----------- ----------- ----------- ----------- ----------- -----------
--------------------------------------------------------------------------------
1   1   106588469   480001710   0   0
0   0   16   33   25   4
4   4   0   4   2   2   0
select distinct c1 from t_metrics1 where c2 in (select c2 from t_metrics2)
```

上例显示了一条 SQL 语句记录。该语句的查询文本为 select distinct c1 from t_metrics1 where c2 in (select c2 from t_metrics2)：

- 该语句目前已执行两次 (cnt = 2)。
- 最短经历时间为 16 毫秒；最长经历时间为 33 毫秒，平均经历时间为 25 毫秒
- 所有执行时间都为 0，这可能是由于 CPU 执行时间小于 1 毫秒所致。
- 最大物理 I/O 为 4，这与最大逻辑 I/O 一致。但是，由于在第二次运行时数据已存在于高速缓存中，因此最小物理 I/O 为 0。逻辑 I/O 等于 4，无论数据是否在内存中，它都应为静态的

使用指标

使用 QP 指标所生成的信息来标识：

- 查询性能衰退
- 一批正在运行的查询中开销最大的查询
- 最常运行的查询

如果您拥有了有关可能引发问题的查询的信息，则可以对这些查询进行调优以提高效率。

例如，标识和调优开销大的查询可能比调优同一批处理中开销小的查询更有效。

您还可以标识最常运行的查询，并调优它们以提高效率。

打开查询指标可能会涉及额外的 I/O 以用于每个执行的查询，因此可能会对性能造成影响。但是，也要考虑上面提到的优点。您可能要通过监控表而不是打开指标来收集统计信息。

QP 指标和监控表均可用于收集统计信息。但是，您可以使用 QP 指标代替监控表，以便将集合历史查询信息收集到持久目录，而不是收集监控表中的瞬时信息。

示例

可以使用 QP 指标来标识要调优的特定查询以及可能的性能衰退。

标识开销最大的语句

通常，为了查找开销最大的语句作为调优候选项， sysquerymetrics 会提供 CPU 执行时间、经历时间、逻辑 I/O 和物理 I/O 作为度量选项。例如，典型的度量基于逻辑 I/O。使用以下查询来查找会产生过多 IO 的语句作为调优候选项：

```
select lio_avg, qtext from sysquerymetrics order by lio_avg
lio_avg qtext
-----------
-----------------------------------------------------
2
select c1, c2 from t_metrics1 where c1 = 333
4
select distinct c1 from t_metrics1 where c2 in (select c2 from t_metrics2)
6
```

```
select count(t_metrics1.c1) from t_metrics1, t_metrics2,
t_metrics3 where (t_metrics1.c2 = t_metrics2.c2 and
t_metrics2.c2 = t_metrics3.c2 and t_metrics3.c3 = 0)
164
select min(c1) from t_metrics1 where c2 in (select t_metrics2.c2 from
t_metrics2, t_metrics3 where (t_metrics2.c2 = t_metrics3.c2 and t_metrics3.c3
= 1))

(4 rows affected)
```

以上结果的最后一个语句即为最佳调优候选项，该语句的平均逻辑 IO 值最大 (164)。

标识最常用的语句以进行调优

如果某个查询经常使用，则调优可提高其性能。将 select 语句与 order by 一起使用来标识最常用的查询：

```
select elap_avg, cnt, qtext from sysquerymetrics order by cnt
elap_avg cnt
qtext
----------- -----------
--------------------------------------------------
0           1
select c1, c2 from t_metrics1 where c1 = 333
16          2

select distinct c1 from t_metrics1 where c2 in (select c2 from t_metrics2)
24          3

select min(c1) from t_metrics1 where c2 in (select t_metrics2.c2 from
t_metrics2, t_metrics3 where (t_metrics2.c2 = t_metrics3.c2 and t_metrics3.c3
= 1))

78          4

select count(t_metrics1.c1) from t_metrics1, t_metrics2, t_metrics3 where
(t_metrics1.c2 = t_metrics2.c2 and t_metrics2.c2 = t_metrics3.c2 and
t_metrics3.c3 = 0)

(4 rows affected)
```

以上结果的最后一个语句即为最佳调优候选项，该语句的值最大 (78)。

标识可能的性能衰退

在某些情况下，当服务器升级至较新的版本时， QP 指标可用于比较性能。要在服务器版本升级后标识性能可能出现某种程度的下降的查询，请执行以下操作：

1 将旧服务器中的 QP 指标备份到备份组：

```
sp_metrics 'backup', '@gid'
```

2 在新服务器上启用 QP 指标：

```
sp_configure "enable metrics capture", 1
```

3 比较新旧服务器上输出的 QP 指标，以识别任何可能存在性能衰退问题的查询。

清除指标

使用 sp_metrics 'flush' 可将内存中的所有集合指标刷新到系统目录中。内存中所有语句的集合指标都将设置为零。

```
sp_metrics 'drop', '@gid' [, '@id'
```

若要删除一个条目，请使用：

```
sp_metrics 'drop', '<gid>', '<id>'
```

您还可以基于某些指标条件，使用 filter 从系统目录中删除 QP 指标：

sp_metrics 'filter', '@gid', [, '@predicate']

此示例删除组 1 中 lio_max < 100 的所有 QP 指标：

```
sp_metrics 'filter','1','lio_max < 100'
```

限制查询指标捕获

以下配置参数为目录中的捕获设置查询指标阈值：

- metrics lio max
- metrics pio max
- metrics elap max
- metrics exec max

利用这些参数，您可以在将指标信息写入目录之前过滤出普通指标。

缺省情况下，这些配置参数设置为 0 （关闭）。

例如，如果不想捕获其 lio 小于 10 的那些查询计划，请使用：

```
sp_configure 'metrics lio max', 10
```

如果不设置这些配置参数中的任何一个， Adaptive Server 会将查询指标捕获到系统表中。但是，如果设置其中任何一个配置参数， Adaptive Server 仅使用那些非零配置参数作为阈值，以便确定是否捕获查询指标。

例如，如果将 metrics elap max 设置为非零值，但不设置其它任何参数，则仅在经历时间大于配置的值时才捕获查询指标。因为其它三个配置参数设置为 0，所以它们不充当用于捕获指标的阈值。

了解 *sysquerymetrics* 中的 UID

当查询中未被用户名限定的所有表名都归数据库所有者所有时，sysquerymetrics 的用户 ID (UID) 为 0。

示例 1

```
select * from t1 where c1 = 1
```

t1 由数据库所有者所有并且由不同用户共享。无论哪个用户发出查询，0 都是 sysquerymetrics 中条目的 UID。

示例 2

```
select * from t2 where c1 = 1
```

在这种情况下，t2 由 *user1* 所有。*user1* 的 UID 用于 sysquerymetrics 中的条目，因为 t2 未经数据库所有者限定，也不归其所有。

示例 3

```
select * from u1.t3 where c1 = 1
```

这里， t3 由 *u1* 所有，且由 *u1* 限定，所以使用 UID 0。

这增加了在用户 ID 之间共享的指标数，从而减少 sysqueryplans 中的条目数。将自动集合具有不同用户 ID 的相同查询的指标。打开跟踪标志 15361 可使用发出查询的用户的 UID。

注释 当至少涉及到一个表时，将捕获 insert...selec、 /update、 delete 语句的 QP 指标。不包括 CIS 相关的查询和 insert...values 语句。

第 10 章

利用统计信息来提高性能

精确的统计信息对于查询优化至关重要。有些情况下，对非前导索引键的列添加统计信息也能提高查询性能。本章解释如何以及何时使用管理统计信息的命令。

Adaptive Server 中维护的统计信息

Adaptive Server 中维护有以下这些关键优化程序统计信息：

- 每个分区的统计信息 — 表行计数；表页计数。由于 systabstats 目录的关系，未分区表被认为具有一个分区。可以在 systabstats 中找到每个分区的统计信息。
- 每个索引的统计信息：索引行计数 — 索引高度；索引叶页计数。每个索引分区在本地索引中都有单独的 systabstats 行。全局索引（被认为是对应一个分区的分区索引）具有一个 systabstats 行。每个索引的统计信息：可以在 systabstats 中找到索引行计数。
- 每个列的统计信息：数据分布。可以在 sysstatistics 中找到每个列的统计信息。

- 列中各组的统计信息 — 密度信息。可以在 sysstatistics 中找到列中各组的统计信息。
- 每个分区的统计信息 —
  - 列统计信息 — 每列的数据分布；列中各组的密度。可以在 sysstatistics 中找到列统计信息。

在本章中，密度是一个表示给定列值的唯一性的统计测量值，而直方图是给定关系列的值的分布的统计表示形式。

统计信息的重要性

Adaptive Server 基于开销的优化程序使用查询中指定的表、索引、分区和列的相关统计信息来估计查询开销。它选择优化程序确定的开销最低的访问方法。但是当统计信息不精确时，开销估计就可能会不准确。

有些统计信息 （例如页数或某个表的行数）在查询处理过程中会被更新。其它统计信息 （如针对列的直方图）仅在执行 update statistics 或创建索引时才会更新。

如果查询执行速度慢，并且您试图通过技术支持部门或 Internet 上的 Sybase 新闻组寻求帮助，那么首先可能会被问到的一个问题就是：“您是否运行了 update statistics？”对于存在统计信息的每一列，可使用 optdiag 命令来查看 update statistics 上次运行的时间：

```
Last update of column statistics:Aug 31 2004
4:14:17:180PM
```

统计信息维护可能用到的另一个命令是 delete statistics。删除一个索引并不意味着删除该索引的统计信息。如果索引删除后列中键值的分布改变，而一些查询仍然应用这些统计信息，则过时的统计信息可能会影响查询计划。

从全局索引生成的直方图统计信息比本地索引生成的直方图统计信息更为准确。对于本地索引，将为每个分区创建统计信息，然后使用近似值将其合并以创建全局直方图，就像合并每个分区中的重叠直方图单元那样。对于全局索引，则不会执行使用合并估计的合并步骤。大多数情况下，对本地索引使用 update statistics 没有任何问题。但是，如果涉及分区表的查询中出现严重估计错误，则可通过创建和删除列的全局索引而不是更新本地索引的统计信息来提高直方图准确性。

非二进制字符集直方图插值

Adaptive Server 15.0.2 及更高版本允许选择性估计与二进制字符集具有相同的准确性，而无需过多的直方图梯级数。这有益于诸如下面使用范围谓词的查询：

```
select * from t1 where charcolumn > "LMC0021" and
charcolumn <= "LMC0029"
```

如果指定的域在同一直方图单元内，则 Adaptive Server 能够更精确地估计此选择性。

在 Adaptive Server 15.0.2 之前的版本中，只有缺省二进制字符集受益于直方图插值，该直方图插值用于估计范围谓词的选择性。对于所有其它字符集， Adaptive Server 对直方图单元的选择性估计为 50%。这通常需要 Adaptive Server 使用字符列直方图的大量直方图单元，才能减少与此估计相关的错误。

更新统计信息

update statistics 命令可更新直方图和密度等与列相关的统计信息。如果在这些列中，索引中键的分布进行更改的方式影响了查询对索引的使用，则它们的统计信息必须更新。

运行 update statistics 会占用系统资源。与其它维护任务类似， Sybase 建议将该任务安排在服务器负载较轻的时候运行。特别要指出的是，update statistics 要求进行表扫描或索引的页级扫描，可能会增加 I/O 争用和使用 CPU 执行排序，并且使用数据和过程高速缓存。使用这些资源会对服务器上运行的查询产生负面影响。

运行此任务时，使用采样功能可减少资源要求并提高灵活性。

注释　采样不会影响 update statistics *table_name index_name* 参数，而只影响未建索引列的 update index statistics 和 update all statistics 以及 update statistics *table_name (column_list)*。

此外，某些 update statistics 命令需要共享锁，这可能会阻止更新。请参见第 285 页的 “扫描类型、排序要求与锁定”。

也可以将 Adaptive Server 配置为在对系统资源影响最小时自动运行 update statistics。请参见第 273 页的 “自动更新统计信息”。

对未建索引的列添加统计信息

当创建一个索引时，将对该索引的前导列生成一个直方图。前几章中的例子已经显示了其它列的统计信息是如何提高优化程序统计信息的精确性的。

只要您的维护日程表留有时间使统计信息保持更新，就可以考虑为经常被用作搜索参数的几乎所有列添加统计信息。

特别要指出的是，当组合索引的次列在搜索参数或连接中与前导索引键一起使用时，对这些列添加统计信息可能会大大增加开销估计。

对更新代理表和视图上的统计信息的限制

如果对代理表创建索引，系统性能可能会下降；您无法运行 create index，因为 Adaptive Server 无法将该命令传播给远程视图。但是，如果将代理表映射到用户表， Adaptive Server 则可以通过对代理表的连接列创建索引来优化查询计划。

如果每次执行该命令时视图返回的行数变化较大，则收集代理表的统计信息时性能将会下降。

如果使用的是代理表或视图，则 Sybase 有以下建议：

- 不要对在视图上定义的代理表运行 update statistics。
- 如果对在视图上定义的代理表运行 update statistics，请通过删除然后重新创建代理表来提高性能。不能使用 delete statistics 删除当前统计信息，因为 systabstats 会保留其初始行计数。
- 将视图转换为永久表。
- 尝试使用命令行跟踪标志 318 （强制重新格式化）。

update statistics 命令

update statistics 命令会创建统计信息 （如果某特定列没有统计信息）或替换现有统计信息。这些统计信息存储在 systabstats 和 sysstatistics 系统表中：

```
update statistics table_name
[[ partition data_partition_name ] [ (column_list ) ] |
index_name [ partition index_partition_name ] ]
[ using step values ]
[ with consumers = consumers] [, sampling=percent]
```

```
update index statistics
table_name [[ partition data_partition_name ] |
[ index_name [ partition index_partition_name ] ] ]
[ using step values ]
[ with consumers = consumers] [, sampling=percent]

update all statistics table_name
[ partition data_partition_name ]
[ sp_configure histogram tuning factor, <value>

update table statistics
table_name [partition data_partition_name ]

delete [ shared ] statistics table_name
[ partition data_partition_name ]
[( column_name[, column_name ] ...)]
```

- 对于 update statistics 命令：
  - *table_name* — 为表中每个索引的前导列生成统计信息。
  - *table_name index_name* — 为索引的所有列生成统计信息。
  - *partition_name* — 只为该分区生成统计信息。
  - *partition_name table_name* (*column_name*) — 为该分区上该表中的指定列生成统计信息。
  - *table_name* (*column_name*) — 仅为该列生成统计信息。
  - *table_name* (*column_name*, *column_name*...) — 为集合中的前导列生成一个直方图，并为前缀子集生成多列密度值。
  - *using step values* — 标识使用的梯级数。缺省值为 20 梯级。要更改缺省梯级数，请使用 sp_configure。
  - sampling = *percent* — 采样百分比的数值，如 05 表示 5%， 10 表示 10%，依此类推。采样整数介于零 (0) 和一百 (100) 之间。
- 对于 update index statistics 命令：
  - *table_name* — 为表中所有索引的所有列生成统计信息。
  - *partition_name table_name* — 为该分区上的表的所有索引中的所有列生成统计信息。
  - *table_name index_name* — 为此索引中的所有列生成统计信息。

- 对于 update all statistics 命令：
  - *table_name* — 为某个表的所有列生成统计信息。
  - *table_name partition_name* — 为分区上某个表中的所有列生成统计信息。
  - *using step values* — 标识使用的梯级数。缺省值为 20 梯级。要更改缺省梯级数，请使用 sp_configure。

    sp_configure 语法包括 histogram tuning factor，它为直方图梯级数带来了更多选择。 histogram tuning factor 的缺省值为 20。有关 sp_configure 的信息，请参见 《系统管理指南，卷 1》中的 5 章“设置配置参数”。

对 *update statistics* 使用采样

Adaptive Server 的优化程序使用数据库上的统计信息来设置和优化查询。若要生成最佳结果，这些统计信息必须是最新的。

对数据集 （例如表）运行 update statistics 命令，以便为索引中的所有列或表中的所有列更新指定索引或列中有关键值分布的信息。此命令可修正列级统计信息的直方图和密度值。优化程序随后将会使用这些结果来计算最佳查询计划。

使用采样方法运行 update statistics 命令，当维护窗口较小但数据集较大时，这可以减少 I/O 和时间。如果正在更新的是处于连续使用状态、被截断并重新填充的大型数据集或表，则您可能希望进行一次统计采样，以减少时间和 I/O 的大小。因为采样不会更新密度值，所以应首先运行完整 update statistics，然后再使用采样来获得准确的密度值。

请谨慎使用采样，因为结果并非完全准确。请针对 I/O 的节省来平衡对直方图值的更改。

采样不会更新由非采样 update statistics 命令创建的密度值。由于密度变化极慢，因此用采样计算得出的近似值替换精确密度通常不会改善估计值。由采样 update statistics 命令创建的密度值将会更新。 Sybase 建议使用此非采样 update statistics 命令建立精确密度，此命令后面可以跟无数个采样 update statistics 命令。若要让采样 update statistics 命令更新密度，请在将 update statistics 用于采样之前删除列统计信息。

在确定是否使用采样时，请考虑数据集的大小、所面临的时间约束以及直方图生成的结果是否达到需要的精度。

采样时使用的百分比取决于您的需要。请测试不同的百分比，直到得到一个反映特定数据集的最精确信息的结果；例如：

```
update statistics authors(auth_id) with sampling = 5 percent
```

使用下面的命令设置全服务器范围的采样百分比：

```
sp_configure 'sampling percent', 5
```

此命令针对 update statistics 将全服务器范围内的采样百分比设置为 5%，这样，不使用*采样*语法也可以执行 update statistics。百分比可以是零 (0) 到一百 (100) 之间的值。

自动更新统计信息

Adaptive Server 基于开销的查询处理器使用查询中指定的表、索引和列的相关统计信息来估计查询开销。查询处理器会选择它确定的开销最小的访问方法。但是，如果统计信息不精确，则开销估计可能会不准确。可以运行 update statistics 来确保统计信息为最新，但是，运行 update statistics 会产生相关的开销，因为此操作会消耗系统资源，如 CPU、缓冲池、排序缓冲区和过程高速缓存。

可以将 update statistics 设置为在最适合您的节点的时间自动运行，以避免妨碍您的系统。可以使用 datachange 函数来确定 update statistics 的最佳运行时间。datachange 还可帮助确保避免不必要地运行 update statistics。可以使用 Job Scheduler 模板来确定对象、日程表、优先级和触发 update statistics 的 datachange 阈值，以确保仅在查询处理器生成更高效的计划时使用关键资源。

因为 update statistics 是资源密集型任务，所以运行 update statistics 的决定应基于一组特定的条件。可以帮助您确定 update statistics 的合适运行时间的一些关键参数包括：

- 自上次运行 update statistics 以来，有多少数据特性已更改？这称为 datachange 参数。
- 是否有足够的资源可用来运行 update statistics。其中包括空闲 CPU 周期数等资源，并且还要确保在运行 update statistics 期间不会发生关键联机活动。

数据更改是一个关键指标，它用于测量自上次运行 update statistics 以来更改的数据量。该指标由 datachange 函数跟踪。使用该指标和资源可用性标准，可以使运行 update statistics 的过程自动进行。Job Scheduler 提供了自动运行 update statistics 的机制，并且它还包括可自定义的模板，可用于确定何时运行 update statistics。这些输入包括 update statistics 的所有参数、datachange 阈值和运行 update statistics 的时间。Job Scheduler 按较低优先级运行 update statistics，因此它不会影响同时运行的关键作业。

datachange 函数

datachange 函数用于测量自上次运行 update statistics 以来的数据分布更改量。具体而言，它测量给定对象、分区或列中发生的 insert、 update 和 delete 次数，并帮助您确定运行 update statistics 是否有益于查询计划。

datachange 的语法为：

```
select datachange(object_name, partition_name, colname)
```

其中：

- *object_name* — 是对象名。此对象被认为位于当前数据库中。*object_name* 不能为空。
- *partition_name* — 是数据分区名称。它可以是空值。
- *colname* — 是请求 datachange 的列的名称。它可以是空值。

这些参数都是必需的。

datachange 以表或分区 （如果已指定分区）中总行数的百分比的形式表示。百分比值可以大于 100 %，原因是对象的更改次数可以大于表中的行数，特别是在对表的 delete 和 update 次数非常多的情况下。

下面的一组示例说明了 datachange 函数的各种用法：这些示例使用以下信息：

- 对象名为 “O”。
- 分区名为 “P”。
- 列名为 “C”。

传递有效对象、分区和列名

包括对象、分区和列名时报告的值由指定分区中指定列的 datachange 值除以分区中的行数计算得出。该结果以百分比的形式表示：

```
datachange = 100 * (data change value for column C/ rowcount (P))
```

使用空分区名

如果包括空分区名，则 datachange 值由所有分区中列的 datachange 值的总和除以表中的行数计算得出。该结果以百分比的形式表示：

```
datachange = 100 * (Sum(data change value for (O, P(1-N) , C))/rowcount(O)
```

其中， P(1-*N*) 表示该值是通过对所有分区求和得出的。

使用空列名

如果包括空列名，则 datachange 报告的值由指定分区上带直方图的所有列的 datachange 的最大值除以分区中的行数计算得出。该结果以百分比的形式表示：

```
datachange = 100 * (Max(data change value for (O, P, Ci))/rowcount(P)
```

其中， *i* 的值随带直方图的列的不同而变化 （例如， sysstatistics 中的 formatid 102）。

空分区名和列名

如果包括空分区名和列名，则 datachange 的值由对所有分区上的直方图求和的所有列的 datachange 的最大值除以表中的行数计算得出。该结果以百分比的形式表示：

```
datachange = 100 * ( Max(data change value for (O, NULL, Ci))/rowcount(O)
```

其中， *i* 为 1 至带直方图的列的总数 （例如， sysstatistics 中的 formatid 102）。

下面演示了 datachange 如何收集统计信息：

```
create table matrix(col1 int, col2 int)
go
insert into matrix values (234, 560)
go
update statistics matrix(col1)
go
insert into matrix values(34,56)
go
select datachange ("matrix", NULL, NULL)
go

------
50.000000
```

matrix 中的行数为 2。自上次执行 update statistics 命令以来更改的数据量为 1，因此 datachange 百分比为 100 * 1/2 = 50%。

datachange 计数器全部保存在内存中。这些计数器由管家定期刷新到磁盘或在运行 sp_flushstats 时刷新到磁盘。

配置自动 *update statistics*

通过以下方式自动更新统计信息：

- 利用 Job Scheduler 定义 update statistics 作业
- 将 update statistics 作业定义为自行管理安装的一部分
- 创建用户定义的脚本

本文档不讨论如何创建用户定义的脚本。

使用 Job Scheduler 更新统计信息

Job Scheduler 包含更新统计信息模板，可以用该模板创建对表、索引、列或分区运行 update statistics 的作业。 datachange 函数确定表或分区中更改量达到预定义阈值的时间。可在配置该模板时确定此阈值。

模板：

- 对特定表、分区、索引或列运行 update statistics。可以利用这些模板定义希望 update statistics 运行的 datachange 值。
- 根据您在创建作业时所确定的阈值，在服务器级运行 update statistics，这样可以配置 Adaptive Server 以扫描服务器上所有数据库中的可用表，并更新所有表的统计信息。

若要配置 Job Scheduler 以自动运行 update statistics，请执行以下操作（下面所列各章出自 《Job Scheduler 用户指南》）：

1 安装和设置 Job Scheduler （第 2 章 “配置和运行 Job Scheduler”）

2 安装模板所需的存储过程 （第 4 章 “使用模板调度作业”）。

3 安装模板。 Job Scheduler 提供专门用于自动更新统计信息的模板（第 4 章 “使用模板调度作业”）。

4 配置模板。用于自动更新统计信息的模板位于 “统计信息管理”文件夹中。

5 调度作业。在定义了要跟踪的索引、列或分区后，还可以创建确定 Adaptive Server 何时运行作业的日程表，以确保仅在不影响性能的情况下运行 update statistics。

6 确定成功或失败。利用 Job Scheduler 基本结构可以确定自动运行 update statistic 是成功还是失败。

可以使用此模板为 update statistics 命令的各个选项提供值，例如采样百分比、消耗程序数目和梯级数等。另外，还可以为 datachange 函数、页计数和行计数提供阈值。如果包括这些可选值，它们将用于确定 Adaptive Server 何时运行以及是否应该运行 update statistics。如果任何表、列、索引或分区的当前值超过阈值， Adaptive Server 便会运行 update statistics。Adaptive Server 检测到对某列运行了 update statistics。在下次执行前，将重新编译过程高速缓存中任何引用该表的查询。

Adaptive Server 何时运行 *update statistics*？

update statistics 命令有多种形式 （update statistics、 update index statistics 等）；可以根据需要使用这些不同的形式。

必须指定三个阈值：rowcount、 pagecount 和 datachange。虽然可以忽略 NULL 或 0 值，但这些值不会阻止命令的运行。

表 10-1 描述了 Adaptive Server 将根据您提供的参数自动运行 update statistics 的几种情况。

表 10-1：Adaptive Server 何时自动运行 update statistics？

如果用户	Job Scheduler 执行的操作
将 datachange 阈值指定为零或 NULL	在预定时间运行 update statistics。
仅为表指定一个大于零的 datachange 阈值，但不请求采用 update index statistics 形式	获取表的所有索引，并获取每个索引的前导列。如果任何前导列的 datachange 值大于或等于该阈值，则运行 update statistics。
指定表和索引的阈值，但不请求 update index statistics 形式	获取索引前导列的 datachange 值。如果 datachange 值大于或等于阈值，则运行 update statistics。
仅为表指定一个阈值，并请求 update index statistics 形式	获取表的所有索引，并获取每个索引的前导列。如果任何前导列的 datachange 值超过该阈值，则运行 update statistics。
指定表和索引的阈值，并请求采用 update index statistics 形式	获取索引前导列的 datachange 值。如果 datachange 值大于或等于阈值，则运行 update statistics。
指定表和一个或多个列的阈值 （忽略任何索引或请求 update index statistics 形式）	获取每个列的 datachange 值。如果任何列的 datachange 值大于或等于该阈值，则运行 update statistics。

datachange 函数可编译表中的更改数，并将其显示为占表中总行数的百分比。可以使用该编译信息来创建确定 Adaptive Server 何时运行 update statistics 的规则。运行该命令的最佳时间可以基于目标的任何数值：

- 表中的更改百分比
- 可用的 CPU 周期数
- 维护窗口期间

运行 update statistics 后， datachange 计数器将重置为零。对于 insert 和 delete，在分区级 （而非对象级）跟踪 datachange 计数；对于 update，则在列级跟踪该计数。

使用 *datachange* 的 update statistics 的示例

可以编写脚本，以便在列、表或分区级检查指定的更改数据量。可以根据 datachange 函数收集的众多变量来决定何时运行 update statistics：CPU 使用率、表中的更改百分比、分区中的更改百分比等。

在下面的示例中， authors 表经过分区，而且用户希望在 author_ptn2 分区中 city 列的数据更改大于或等于 50% 时运行 update statistics：

```
select  @datachange = datachange("authors","author_ptn2", "city")
if @datachange >= 50
begin
         update statistics authors partition author_ptn2(city)
end
go
```

用户也可以指定在系统空闲 （或其它任何参数）时执行该脚本。

在下面的示例中，如果 authors 表的 city 列的数据更改大于或等于 100%（该示例中的表没有分区），用户将触发 update statistics：

```
select  @datachange = datachange("authors",NULL, "city")
if @datachange > 100
begin
         update statistics authors (city)
end
go
```

列统计信息及统计信息维护

直方图按每个列保存，而不是按每个索引保存。这对管理统计信息具有某些提示：

- 如果某列出现在多个索引中，update statistics、update index statistics 或 create index 将更新该列的直方图以及所有前缀子集的密度统计信息。

  update all statistics 更新表中所有列的直方图。

- 删除索引不会删除索引的统计信息，因为即使索引不存在，优化程序也可以使用列级统计信息来估计开销。

  若要在删除索引后删除统计信息，必须使用 delete statistics 显式删除它们。

如果统计信息对查询处理器有用，并且要在没有索引的情况下保留统计信息，则需要使用 update statistics，以便为键值分布随时间变化的索引指定列名。

- 截断表不会删除 sysstatistics 中的列级统计信息。在许多情况下，表被截断并且重新装载相同数据。

  由于 truncate table 不会删除列级统计信息，因此如果数据相同，则在重新装载表后不必运行 update statistics。

  如果使用具有不同键值分布的数据重新装载表，则需运行 update statistics。

- 通过在 create index 的 with statistics 子句中将梯级数指定为 0，您可以删除并重新创建索引，而不影响索引统计信息。该 create index 命令不会影响 sysstatistics 中的统计信息：

  ```
  create index title_id_ix on titles(title_id)
      with statistics using 0 values
  ```

  这使您能够重新创建索引而不会覆盖用 optdiag 编辑过的统计信息。

- 如果两个用户试图同时在一个表中使用相同的列来创建索引，其中一个命令可能会由于试图在 sysstatistics 中输入一个重复键值而失败。

- 对多分区表的某个分区中的列执行 update statistics 会更新该分区的统计信息，而且还会更新该列的全局直方图。这可以通过以行加权的方式合并每个分区中该列的直方图来完成，以便为该列创建全局直方图。

- 在指定分区的情况下，为某列更新多分区表中的统计信息将会为该列更新表的每个分区的统计信息，最后，需要合并该列的分区直方图，以便为该列创建全局直方图。

- 在编译过程中，优化程序仅为多分区表使用全局直方图，而不会读取分区直方图。此方法可避免在编译期间合并分区直方图的开销，而在 DDL 期间执行所有合并操作。

创建和更新列统计信息

对未建索引的列创建统计信息可以提高许多查询的性能。优化程序可以在 where 或 having 子句中使用任何列上的统计信息，以帮助估计表中的行数，这些行与该表中一组完整的查询子句相匹配。

添加对于索引次列以及在搜索参数中频繁使用的未建索引列的统计信息，将会显著改善优化程序的估计。

在数据修改期间维护大量索引的代价是巨大的。每次对表执行 insert 和 delete 操作时，都必须更新表的每个索引，并且所做更新可能影响一个或多个索引。

为没有创建索引的列生成统计信息可以为优化程序提供更多信息，这些信息可用来估计查询要读取的页数，同时在数据修改期间不会产生处理索引更新的开销。

对于在 where 或 having 子句的搜索参数中使用的任何列以及在 join 子句中指定的任何列，优化程序可以应用统计信息。

使用以下命令创建并维护列统计信息：

- 当与列名一起使用时， update statistics 不用创建索引就能生成该列的统计信息。有关语法的信息，请参见第 283 页的 “使用 update statistics 为列添加统计信息”。

  优化程序可以使用这些列统计信息，从而更加精确地估计引用该列的查询开销。

- 与索引名一起使用时， update index statistics 将创建或更新索引中所有列的统计信息。有关语法的信息，请参见第 283 页的 “使用 update index statistics 为次列添加统计信息”。

  如果与表名一起使用， update index statistics 将更新所有索引列的统计信息。

- update all statistics 创建或更新表中所有列的统计信息。有关语法的信息，请参见第 283 页的 “使用 update all statistics 为所有列添加统计信息”。

适合创建统计信息的列是：

- 经常用作 where 和 having 子句中的搜索参数的列。
- 组合索引中包含的列，它们不是索引中的前导列，但有助于估计查询需要返回的数据行数。

其它统计信息何时可能有用

若要确定其它统计信息何时有用，请使用 set option 命令和 set statistics io 运行查询。如果 set 命令显示的“要返回的行”及 I/O 估计值与 statistics io 显示的实际 I/O 之间存在巨大差异，请检查这些查询以找出其它统计信息可以在哪些方面改善估计值。尤其注意了解搜索参数与连接列的缺省密度值的使用方法。

set option show_missing_stats 命令可以输出使用了直方图的列的名称和使用了多属性密度的列中的各组。这对指出其它统计信息在哪些方面有用至关重要。

示例 1

```
set option show_missing_stats long
go
dbcc traceon(3604)
go
DBCC execution completed.If DBCC printed error
messages, contact a user with System Administrator (SA)
role.
select * from part, partsupp
where p_partkey = ps_partkey and p_itemtype =
ps_itemtype
go
NO STATS on column part.p_partkey
NO STATS on column part.p_itemtype
NO STATS on column partsupp.pa_itemtype
NO STATS on density set for E={p_partkey, p_itemtype}
NO STATS on density set for F={ps_partkey, ps_itemtype}
- - - - - - - - - - - - - - -
(200 rows affected)
```

可以使用 show_final_plan_xml 选项获取相同信息。 set plan 使用客户端选项和跟踪标志 3604 在客户端获取输出。这与 set plan 的消息选项的使用方法不同。

示例 2

```
dbcc traceon(3604)
DBCC execution completed.If DBCC printed error
messages, contact a user with System Administrator (SA)
role.
set plan for show_final_plan_xml to client on
go
select * from part, partsupp
where p_partkey = ps_partkey and p_itemtype =
ps_itemtype
go
<?xml version="1.0" encoding="UTF-8"?>
<query>
```

```
        <planVersion> 1.0 </planVersion>
- - - - - - - - - - - - - - - - - -
<optimizerStatistics>
    <statInfo>
        <objName>part</objName>
        <missingHistogram>
            <column>p_partkey</column>
            <column>p_itemtype</column>
        </missingHistogram>
        <missingDensity>
            <column>p_partkey</column>
            <column>p_itemtype</column>
        </missingDensity>
    </statInfo>
    <statInfo>
<objName>partsupp</objName>
        <missingHistogram>
            <column>ps_partkey</column>
            <column>ps_itemtype</column>
        </missingHistogram>
        <missingDensity>
            <column>ps_partkey</column>
            <column>ps_itemtype</column>
        </missingDensity>
    </statInfo>
</optimizerStatistics>
```

对 part 和 partsupp 使用 update statistics 可创建 p_partkey 和 p_itemtype 的统计信息，进而对前导列 (p_partkey) 创建直方图并对 （p_partkey、p_itemtype）创建密度。同样可对 p_itemtype 创建直方图。使用：

```
update statistics part(p_partkey, p_itemtype)
go
update statistics part(p_itemtype)
go
```

因为 partsupp 在 ps_partkey 上具有直方图，所以您可以对 ps_itemtype 创建直方图并对 （ps_itemtype、 ps_partkey）创建密度。用于密度的列可以不经过排序。

```
update statistics partsupp(ps_itemtype, ps_partkey)
```

如果此过程成功，再次运行该查询时，将不会看到示例 1 中所示的“NO STATS”消息。

使用 *update statistics* 为列添加统计信息

若要为 titles 表中的 price 列添加统计信息，请输入：

```
update statistics titles (price)
```

若要为某列指定直方图梯级数，请使用：

```
update statistics titles (price)
   using 50 values
```

以下命令为 titles.pub_id 列添加一个直方图，并为前缀子集 pub_id；pub_id、 pubdate；和 pub_id、 pubdate、 title_id 生成密度值：

```
update statistics titles(pub_id, pubdate, title_id)
```

但是，此命令不会在 `pubdate` 和 `title_id` 上创建直方图，因为需要直方图的每个列都需要单独的 update statistics 命令。

注释　运行带有表名的 update statistics 时，只为索引的前导列更新直方图与密度，而不会更新未建索引的列的统计信息。要维护这些统计信息，请运行 update statistics 并指定列名，或运行 update all statistics。

使用 *update index statistics* 为次列添加统计信息

要创建或升级索引中所有列的统计信息，请使用 update index statistics。语法为：

update index statistics
table_name [[partition *data_partition_name*] |
[*index_name* [partition *index_partition_name*]]]
[using step values]
[with consumers = consumers] [, sampling = *percent*]

使用 *update all statistics* 为所有列添加统计信息

要创建或更新表中所有列的统计信息，请使用 update all statistics。语法为：

update all statistics *table_name*
[partition *data_partition_name*]

为直方图选择梯级数

缺省情况下，每个直方图的梯级数为 20，这为值均匀分配的列提供了良好的性能和建模。较高的梯级数可以提高以下列的 I/O 估计精确度：

- 具有大量高重复值的列
- 具有非均匀分布或值倾斜分布的值的列
- 在 like 查询中使用前导通配符进行查询的列

直方图调优因子缺省值 20 会在当前请求的梯级值 （缺省值 20）与由于因子 (20 * 20 = 400) 而增加的梯级之间自动选择梯级值，以便 Adaptive Server 自动选择最佳的梯级值以补偿上面的情形。覆盖梯级值时应该将由直方图调优因子引入的较大梯级数考虑在内。

注释 如果从 Adaptive Server 11.9 之前的版本更新数据库，则缺省情况下，梯级数将为 distribution page 上使用的梯级数。

如果将梯级数增加到超过良好查询优化所需的数量，将会降低 Adaptive Server 的性能，这主要是由于存储和使用统计信息所需的空间量造成的。增加梯级数将：

- 增加 sysstatistics 所需的磁盘存储空间
- 增加查询优化期间读取统计信息所需的高速缓存空间
- 如果梯级数非常大，则要求更多的 I/O

查询优化期间，直方图使用从过程高速缓存中借用的空间。优化查询后，此空间将立即被释放。

选择梯级数

如果表有 5000 行，并且列中的每个值仅有一个匹配行，则可能需要请求 5000 个梯级以获得直方图，从而使每个不同的值都具有一个频率单元。实际梯级数不是 5000 ；而是不同值的数目加一 （对于密集频率单元），或者值数目的两倍加一 （对于稀疏频率单元）。

当存在大量高重复值时， sp_configure 选项 histogram tuning factor 将自动选择参数中较大的梯级数。

在 Adaptive Server 15.0 及更高版本中，histogram tuning factor 的缺省值为 20。如果请求的梯级数为 50，则 update statistics 最多可以创建 20 * 50 = 1000 个梯级。这个较大的梯级数仅用于直方图分布倾斜的情况（具有大量高重复的域值）。但是，对于唯一列， update statistics 只使用 50 个梯级来表示直方图。若要最有效地使用直方图，请指定相对较低的梯级数并允许直方图调优因子确定更多的梯级是否会对优化有帮助。例如，不要使用缺省梯级计数 1000 指定所有直方图要使用的 1000 个梯级，而是最好指定 50 个缺省梯级并将直方图调优因子指定为 20。这样，Adaptive Server 即可在 50 到 1000 个梯级范围内确定最佳梯级计数以表示分布。

扫描类型、排序要求与锁定

表 10-2 显示了在 update statistics 期间执行的扫描类型、获取的锁类型以及何时需要排序。

表 10-2：更新统计信息期间的扫描、排序和锁定

update statistics 指定	执行的扫描和排序	锁定
表名		
所有页锁定表	表扫描，加上每个非聚簇索引的叶级扫描	级别 1；共享的意图表锁，当前页的共享锁
DOL 锁定表	表扫描，加上每个非聚簇索引和聚簇索引（如果存在）的叶级扫描	级别 0；脏读
表名和聚簇索引名		
所有页锁定表	表扫描	级别 1；共享的意图表锁，当前页的共享锁
DOL 锁定表	叶级索引扫描	级别 0；脏读
表名和非聚簇索引名		
所有页锁定表	叶级索引扫描	级别 1；共享的意图表锁，当前页的共享锁
DOL 锁定表	叶级索引扫描	级别 0；脏读
表名和列名		
所有页锁定表	表扫描；创建工作表和排序工作表	级别 1；共享的意图表锁，当前页的共享锁
DOL 锁定表	表扫描；创建工作表和排序工作表	级别 0；脏读

对未建索引的列或非前导列进行排序

对于未建索引的列以及不是索引中前导列的列， Adaptive Server 会执行串行表扫描，以便将列值复制到工作表中。然后它会对工作表进行排序，以生成直方图。如果不指定 with consumers 子句，排序将以串行方式执行。

请参见第 5 章 “并行查询处理”。

update index statistics 期间的锁定、扫描和排序

update index statistics 命令可生成一系列更新统计信息操作，这些操作与索引级和列级等效命令使用相同的锁定、扫描和排序。例如，如果 salesdetail 表在 salesdetail(stor_id, ord_num, title_id) 上有名为 sales_det_ix 的非聚簇索引，则此命令：

```
update index statistics salesdetail
```

将执行以下 update statistics 操作：

```
update statistics salesdetail sales_det_ix
update statistics salesdetail (ord_num)
update statistics salesdetail (title_id)
```

update all statistics 期间的锁定、扫描和排序

update all statistics 命令为表中的每个索引生成一系列 update statistics 操作，然后为所有未建索引的列生成一系列 update statistics 操作。

使用 *with consumers* 子句

update statistics 的 with consumers 子句设计用于独立磁盘冗余阵列 (RAID) 设备上的分区表，这种设备在 Adaptive Server 看来只是一个 I/O 设备，但它可以提供并行排序所需的高吞吐量。有关信息，请参见第 5 章 “并行查询处理”。

减少 *update statistics* 对并发进程的影响

由于 update statistics 对 DOL 锁定表使用脏读 （事务隔离级别为 0），因此您可以在服务器上的其它任务处于活动状态时执行它，而且不会妨碍对表和索引的访问。更新索引中前导列的统计信息时，只要求对索引进行叶级扫描，而不要求排序，因此更新这些列的统计信息不会过多影响并发性能。

不过，更新未建索引的列和非前导列的统计信息 （需要表扫描、工作表和排序）可能影响并发处理。

- 排序是 CPU 密集型操作。使用串行排序或更少数量的工作进程，可将 CPU 利用率降至最低。也可以使用执行类为 update statistics 设置优先级。

  请参见 《性能和调优系列：基础知识》。

- 合并排序运行所需的高速缓存空间是从数据高速缓存中得到的，同时还需要一些过程高速缓存空间。将 number of sort buffers 设置为较小值可减少缓冲区高速缓存中使用的空间。

  如果将 number of sort buffers 设置为较大值，它将占用数据高速缓存中更多的空间，并且还可能导致从过程高速缓存刷新存储过程，因为在合并排序值时需要使用过程高速缓存空间。可装入指定的排序缓冲区的每行所需的过程高速缓存大约为 100 字节。例如，如果指定了 500 个 2K 排序缓冲区，且每个 2K 缓冲区大约可容纳 200 行，则需要 200 * 100 * 500 字节的过程高速缓存以支持排序。如果全部 500 个数据高速缓存缓冲区都由排序运行填充，则本示例需要大约 5000 个 2K 过程高速缓存缓冲区。

创建用于排序的工作表也会占用 tempdb 中的空间。

使用 *delete statistics* 命令

在 Adaptive Server 11.9 之前的版本中，删除索引会删除该索引的分布页。从 11.9.2 版本开始，对列级统计信息的维护在用户的明确控制下进行，而且即使索引不存在，优化程序也可以使用列级统计信息。 delete statistics 命令允许删除特定列的统计信息。

如果创建了索引，但由于它对数据访问没有帮助或者由于数据修改期间产生索引维护开销而需要删除该索引，则必须确定：

- 索引中的统计信息对优化程序是否有用。
- 此索引列中的键值分布是否随以后进行的插入行和删除行而改变。

  如果改变了键值分布，则需定期运行 update statistics 以保存有用的统计信息。

下面的示例将删除 titles 表中 price 列的统计信息：

```
delete statistics titles(price)
```

注释 delete statistics 只从 sysstatistics 中删除行；而不会从 systabstats 中删除行。您不能删除 systabstats 中描述分区行计数、集群比、页计数等内容的行。但是，如果使用 optdiag simulate statistics 将任何模拟 systabstats 行添加到 sysstatistics，则会删除这些行。

当行计数可能不准确时

行数、转移的行数和删除的行数的行计数值可能不准确，特别是当查询处理包含许多 rollback 命令时。如果工作量非常过大，并且管家清洗任务没有经常运行，则统计信息更有可能不准确。

运行 update statistics 可纠正 systabstats 中的计数。运行管家清洗任务或执行 sp_flushstats 时，行计数值将保存在 systabstats 中。

注释 必须将配置参数 housekeeper free write percent 设置为 1 或更大值，以便能够刷新管家统计信息。

运行 dbcc checktable 或 dbcc checkdb 可更新内存中的这些行计数值。

第 11 章

抽象计划简介

概述

Adaptive Server 可以为查询生成一个抽象计划，并将文本和与之相关的抽象计划保存到 sysqueryplans 系统表中。通过快速散列方法，引入 SQL 查询可以和保存的查询文本相比较，如果发现有匹配，则相应保存的抽象计划将被用来执行查询。

抽象计划用专门为此目的而创建的语言描述一个查询的执行计划。该语言包含运算符以指定可由优化程序生成的选择和操作。例如，若要使用 title_id_ix 索引指定对 titles 表的索引扫描，抽象计划应如下：

```
(i_scan title_id_ix titles)
```

要将此抽象计划用于查询，您可以修改查询文本并添加 PLAN 子句：

```
select * from titles where title_id = “On Liberty”
plan
“(i_scan title_id_ix titles)”
```

此替代方法需要更改 SQL 文本；但是，第一段中所介绍的方法（即，提供查询抽象计划的基于 sysqueryplans 的方法）并不涉及更改查询文本。

抽象计划为系统管理员和性能调优员提供了一种方法，使其可以保护服务器的整体性能不受查询计划更改的影响。以下原因可引起查询计划的更改：

- Adaptive Server 影响优化程序选择和查询计划的软件升级
- 更改查询计划的 Adaptive Server 新增功能
- 更改调优选项，如并行程度、表分区或索引

抽象计划的主要用途是提供一种在发生重大系统更改前后捕获查询计划的方法。然后可通过对比重大系统更改前后的查询计划集来确定查询计划更改的效果。其它用途包括：

- 搜索特定类型的计划，如表扫描或重新格式化
- 搜索使用特定索引的计划
- 为执行效果差的查询指定完整或部分计划
- 保存需要较长优化时间的查询计划

抽象计划为那些必须在批处理或查询中指定以影响优化程序决策的选项提供了替代方案。使用抽象计划，不用修改 SQL 语句的语法即可对 SQL 语句的优化产生影响。将查询文本与存储的文本进行匹配需要一些处理开销，使用保存的计划可减少查询优化开销。

管理抽象计划

系统管理员和数据库所有者可利用一整套完整的系统过程来管理计划和计划组。个人用户可以查看、删除和复制他们所运行的查询计划。

有关详细信息，请参见第 14 章 “使用系统过程管理抽象计划”。

查询文本和查询计划之间的关系

对于大多数 SQL 查询，有许多可能的查询执行计划。SQL 描述预期的结果集，但不描述该结果集应如何从数据库中获得。下例是一个连接三个表的查询：

```
select t1.c11, t2.c21
from t1, t2, t3
where t1.c11 = t2.c21
and t1.c11 = t3.c31
```

有多种不同的可能连接顺序，同时根据表中存在的索引的不同，还有多种可能的访问方法，包括表扫描、索引扫描和重新格式化策略等。每个连接都可以使用嵌套循环连接或合并连接。这些选择由优化程序的查询开销算法确定，而不包括在查询本身或在查询中指定。

在捕获抽象计划时，查询用通常使用的方法进行优化，只是优化程序也生成了抽象计划，并将查询文本和抽象计划保存到 sysqueryplans 中。

对影响查询计划的选项的限制

Adaptive Server 提供了影响优化程序选择的其它选项：

- 会话级选项，如强制执行连接顺序的 set forceplan 或指定用于查询的最大工作进程数的 set parallel_degree
- 用来影响索引选择、高速缓存策略和并行度的可包含在查询文本中的选项

使用 set 命令或对查询文本添加提示时有一些限制：

- 并非能对所有查询计划步骤都产生影响，例如子查询连接。
- 某些查询生成工具不支持查询中的选项，或要求所有查询都是独立于供应商的。

完整和部分计划

抽象计划可以是描述全部查询处理步骤和选项的完整计划，也可以是部分计划。部分计划可能指定某个索引将用于某一特定表的扫描，而不指定其它访问方法。例如：

```
select t1.c11, t2.c21
from t1, t2, t3
where t1.c11 = t2.c21
and t1.c11 = t3.c31
plan
"(i_scan t3_c31_ix t3)"
```

完整抽象计划包括：

- 连接类型，可以是用于嵌套循环连接的 nl_join、用于合并连接的 m_join，或用于散列连接的 h_join。
- 连接顺序。
- 扫描类型，用于表扫描的 t_scan 或用于索引扫描的 i_scan。
- 为通过索引扫描访问的表选择的索引名。
- 扫描属性：用于查询中每个表的并行度、 I/O 大小和高速缓存策略。

上述查询的抽象计划指定了连接顺序、查询中每个表的访问方法和每个表的扫描属性：

```
select t1.c11, t2.c21
from t1, t2, t3
where t1.c11 = t2.c21
and t1.c11 = t3.c31
plan
"(i_scan t3_c31_ix t3)"

(nl_join ( nl_join
    ( t_scan t2 )
    ( i_scan t1_c11_ix t1 )
    )
    ( i_scan t3_c31_ix t3 )
)
( prop t3
    ( parallel 1 )
    ( prefetch 16 )
    ( lru )
)
( prop t1
```

```
        ( parallel 1 )
        ( prefetch 16 )
        ( lru )
    )
    ( prop t2
        ( parallel 1 )
        ( prefetch 16 )
        ( lru )
    )
```

如果启用抽象计划转储模式，查询文本和抽象计划对将保存在 sysqueryplans 中：

```
select t1.c11, t2.c21
from t1, t2, t3
where t1.c11 = t2.c21
and t1.c11 = t3.c31
plan
“(i_scan t3_c31_ix t3)”
```

创建部分计划

在捕获了抽象计划后，将生成并存储完整抽象计划。可编写部分计划，以便只对优化程序所做选择的一部分产生影响。如果上述查询未使用 t3 的索引，但查询计划的其它所有部分是最佳的，则可使用 create plan 命令为此查询创建一个部分计划。这一部分计划仅指定 t3 的索引选择：

```
create plan
"select t1.c11, t2.c21
from t1, t2, t3
where t1.c11 = t2.c21
and t1.c11 = t3.c31"
"( i_scan t3_c31_ix t3 )"
```

也可用 plan 子句为 select、 delete、 update 和其它可优化的命令创建抽象计划。如果启用抽象计划转储模式，查询文本和 AP 对将保存在 sysqueryplans 中。

有关详细信息，请参见第 12 章 “创建和使用抽象计划”。

抽象计划组

安装 Adaptive Server 后，有两个抽象计划组：

- ap_stdout，缺省情况下用于捕获计划
- ap_stdin，缺省情况下用于计划关联

系统管理员可为 ap_stdout 启用全服务器范围的计划捕获，以便可以捕获所有查询的所有查询计划。全服务器范围的计划关联使用来自 ap_stdin 的查询和计划。如果某些查询需要特殊调优计划，它们可被设为全服务器范围内可用。

系统管理员或数据库所有者可创建附加计划组，从一个组向另一个组复制计划以及比较两个不同组之间的计划。

对抽象计划的捕获及抽象计划与查询的关联始终发生在当前活动的计划组的环境中。用户可使用会话级 set 命令来启用计划捕获和关联。

可使用抽象计划组的方法有：

- 查询调优员可在为测试目的而创建的组中创建抽象计划，而不会影响系统中其他用户的计划。
- 使用计划组，用查询计划前后计划集的变化以确定系统变更或升级对查询优化所产生的影响。

有关启用计划捕获和关联的详细信息，请参见第 12 章 “创建和使用抽象计划”。

抽象计划如何与查询关联

抽象计划保存后，查询中的所有空白 （制表符、多个空格和回车，用于终止 — 样式注释的回车除外）被裁剪为单个空格，并为空白经过裁剪的 SQL 语句计算散列键值。经裁剪的 SQL 语句和散列键被存储在 sysqueryplans 中，同时存储的还有抽象计划、唯一的计划 ID、用户 ID 和当前抽象计划组 ID。

启用抽象计划关联后，将计算传入 SQL 语句的散列键，此值用于在当前关联组中搜索具有相应用户 ID 的匹配查询和抽象计划。抽象计划的完整关联键包括：

- 当前用户的用户 ID
- 当前关联组的组 ID
- 完整查询文本

发现匹配的散列键后，将用已保存查询的完整文本与待执行的查询进行比较，如果匹配则采用。

由用户 ID、组 ID 和查询文本构成的关联键组合表示：对于某一给定用户，同一抽象计划组中不能有两个具有相同的查询文本但查询计划不同的查询。

高速缓存语句中的抽象计划

在 Adaptive Server 15.7 及更高版本中，您可以在语句高速缓存中保存抽象计划信息。

在这个包括抽象计划的示例中，散列表会保存 `select * from t1 plan '(use optgoal allrows_mix)'`，如 `SQL TEXT` 行中所示：

```
1> select * from t1 plan '(use optgoal allrows_mix)'
2> go
1> dbcc prsqlcache
2> go

Start of SSQL Hash Table at 0x0x1474c9050

Memory configured:1000 2k pages        Memory used:17 2k pages


Bucket# 243 address 0x0x1474c9f80

SSQL_DESC 0x0x1474cd070
ssql_name *ss0626156152_0290084701ss*
ssql_hashkey 0x0x114a575d       ssql_id 626156152
ssql_suid 1             ssql_uid 1      ssql_dbid 1      ssql_spid 0
ssql_status 0x0xa0      ssql_parallel_deg 1
ssql_isolate 1          ssql_tranmode 32
ssql_keep 0             ssql_usecnt 1   ssql_pgcount 6
ssql_optgoal allrows_mix    ssql_optlevel ase_default
SQL TEXT:select * from t1 plan '(use optgoal allrows_mix)'

End of SSQL Hash Table
```

第 12 章

创建和使用抽象计划

使用 set 命令捕获抽象计划并将传入的 SQL 查询与保存的计划关联。在会话期间，任何用户都可以发出会话级命令来捕获和装载计划，系统管理员可启用全服务器范围抽象计划捕获和关联。此外，本章还将介绍如何用 SQL 指定抽象计划。

使用 *set* 命令捕获和关联计划

在会话级，任何用户都可以使用 set plan dump 和 set plan load 命令启用和禁用抽象计划的捕获和使用。 set plan replace 可确定现有计划是否被更改的计划覆盖。

启用和禁用抽象计划模式会在该包括该命令的批处理结束时生效（类似 showplan）。因此，在运行查询之前，应在单独的批处理中更改此模式：

```
set plan dump on
go
/*queries to run*/
go
```

存储过程中使用的任何 set plan 命令都不会影响它们所在的过程 （但延迟编译影响的那些语句除外），但在这些过程结束后仍然有效。

使用 *set plan dump* 启用计划捕获模式

set plan dump 命令可激活和停用抽象计划的捕获。可用不带组名的 set plan dump 命令将计划保存到缺省组 ap_stdout 中：

```
set plan dump on
```

若要在特定抽象计划组中开始捕获计划，请指定组名。以下示例将 dev_plans 组设置为捕获组：

```
set plan dump dev_plans on
```

发出 set 命令前，您指定的组必须存在。系统过程 sp_add_qpgroup 可创建抽象计划组；只有系统管理员或数据库所有者才能创建抽象计划组。抽象计划组一旦存在，任何用户都可将计划转储到该组中。

有关创建计划组的信息，请参见第 347 页的 “创建组”。

若要使计划捕获功能失效，请使用：

```
set plan dump off
```

不必指定组名即可结束捕获模式。任何时候保存或匹配抽象计划时，只能有一个抽象计划组处于活动状态。如果当前正在将计划保存到组，请关闭计划转储模式，然后为新组重新启用它，如下所示：

```
set plan dump on /*save to the default group*/
go
/*some queries to be captured */
go
set plan dump off
go
set plan dump dev_plans on
go
/*additional queries*/
go
```

在 set plan dump 生效时使用 use database 命令会禁用计划转储模式。

将查询与存储计划相关联

set plan load 命令可激活和停用查询与存储的抽象计划的关联。

若要使用缺省组 ap_stdin 启动关联模式，请使用：

```
set plan load on
```

若要使用另一抽象计划组启用关联模式，请指定该组名：

```
set plan load test_plans on
```

对于计划关联，同一时间只能有一个抽象计划组处于活动状态。如果计划关联对某个组来说处于活动状态，则必须停用当前组，并为新组激活计划关联，如下所示：

```
set plan load test_plans on
go
/*some queries*/
go
set plan load off
go
set plan load dev_plans on
go
```

在 set plan load 生效时使用 use database 命令会禁用计划装载模式。

计划捕获期间使用替换模式

计划捕获模式处于活动状态时，可通过启用或禁用 set plan replace 来选择是否用相同查询的计划替换现有计划。若要激活计划替换模式，请使用：

```
set plan replace on
```

不要使用 set plan replace 指定组名；它会影响当前的活动捕获组。

禁用计划替换：

```
set plan replace off
```

在 set plan replace 生效时使用 use database 命令会禁用计划替换模式。

何时使用替换模式

捕获计划时，如果查询与已保存计划具有相同的查询文本，则不会替换现有计划，除非已启用 replace 模式。如果已为特定查询捕获了抽象计划，并且正在对数据库进行可影响优化程序选择的物理更改，请替换现有计划，以便保存这些更改。

可能需要计划替换的一些操作有：

- 添加或删除索引，或者更改索引中的键或键排序
- 更改表上的分区
- 添加或删除缓冲池
- 更改影响查询计划的配置参数

大多数情况下，不要启用 plan load 模式。当计划关联处于活动状态时，任何计划说明都可用作优化程序的输入。例如，如果某个完整的查询计划包括 prefetch 属性和 2K 的 I/O 大小，已创建了 16K 的缓冲池，并且想要替换计划中的预取说明，则不要启用 plan load 模式。

当表中的数据分配发生变化时，或者在重建索引、更新统计信息或更改锁定方案之后，最好检查查询计划并替换一些抽象计划。

同时使用 *dump*、 *load* 和 *replace* 模式

无论 replace 模式是否处于活动状态，都可同时激活 plan dump 和 plan load 模式。

对同一组使用 *dump* 和 *load*

如果在未启用 replace 模式的情况下，对同一组启用了 dump 和 load：

- 如果存在对查询有效的计划，通常会装载和使用它，以优化查询。
- 如果存在的计划无效（例如，由于索引被删除），则将生成并使用新计划以优化查询，但不保存新计划。
- 如果只存在部分计划，则会生成完整的计划，但不替换现有的部分计划。
- 如果不存在用于查询的计划，则生成一个计划，并将其保存。

如果同时也启用了 replace 模式：

- 如果存在对查询有效的计划，通常会装载和使用它，以优化查询。
- 如果计划无效，则生成并使用新计划，以优化查询，同时替换原有计划。
- 如果该计划只是部分计划，则生成和使用完整的计划，同时替换现有的部分计划。部分计划中的说明用作优化程序的输入。
- 如果不存在用于查询的计划，则生成一个计划，并将其保存。

对不同组使用 *dump* 和 *load*

如果在未启用 replace 模式的情况下，对一个组启用 dump，而对另一个组启用 load：

- 如果在装载组中存在对查询有效的计划，则装载并使用此计划。该计划会保存在转储组中，除非该转储组中已存在一个用于查询的计划。
- 如果装载组中的计划无效，则生成一个新计划。该新计划会保存在转储组中，除非该转储组中已存在一个用于查询的计划。
- 如果装载组中的计划只是部分计划，则生成完整的计划，并保存在转储组中，除非该组中已存在一个计划。部分计划中的说明用作优化程序的输入。
- 如果装载组中不存在用于查询的计划，则生成此计划，并保存在转储组中，除非该转储组中已存在一个用于查询的计划。

如果同时也启用了 replace 模式：

- 如果在装载组中存在对查询有效的计划，则装载并使用此计划。
- 如果装载组中的计划无效，则生成并使用新计划，以优化查询。新计划保存在转储组中。
- 如果装载组中的计划只是部分计划，则生成完整的计划，并保存在转储组中。部分计划中的说明用作优化程序的输入。
- 如果装载组中不存在用于查询的计划，则生成一个新计划。新计划保存在转储组中。

对某些 set 参数的编译期更改

在 Adaptive Server 15.0.2 之前的版本中， set 参数在执行或重新编译存储过程后生效。Adaptive Server 15.0.2 及更高版本允许您在编译时使用优化程序 set 参数，以影响存储过程或批处理中的优化程序。

注释　这一更改的行为可能会影响结果集的组成。 Sybase 建议，在生产系统中使用 15.0.2 版 set 参数之前，先检查这些参数创建的结果集。

从存储过程返回前，必须重置 set 参数，否则后续存储过程的执行可能会受到影响。如果计划将此更改传播给后续存储过程，请使用 export_options 参数。

Adaptive Server 更改了以下参数的编译期行为：

- distinc_sorted
- distinct_sorting
- distinct_hashing
- group_sorted
- group_hashing
- bushy_space_search
- parallel_query
- order_sorting
- nl_join
- merge_join
- hash_join
- append_union_all
- merge_union_all
- merge_union_distinct
- hash_union_distinct
- store_index
- index_intersection
- index_union
- multi_table_store_ind
- opportunistic_distict_view
- advanced_aggregation
- replicated_partition
- group_inserting
- basic_optimization
- auto_query_tuning
- query_tuning_mem_limit
- query_tuning_time_limit
- set plan optgoal

set plan exists check 选项

在查询计划关联期间使用 exists check 模式可在用户需要抽象计划组中的少于 20 个查询的抽象计划时提高性能。如果少数查询需要使用计划来改进优化，则启用 exists check 模式可以加快所有无抽象计划的查询的执行速度，因为它们不用在 sysqueryplans 中检查计划。

同时启用 set plan load 和 set exists check 时，会为用户高速缓存装载组中最多 20 个查询的散列键。如果装载组包含 20 个以上的查询，则禁用 exists check 模式。每个引入查询被散列；如果其散列键未存储在抽象计划高速缓存中，则不存在此查询的计划，不进行搜索。这样可将加快所有无保存计划的查询的编译速度。

语法为：

```
set plan exists check {on | off}
```

启用计划散列键高速缓存前，必须启用装载模式。

系统管理员可以使用配置参数 abstract plan cache 来配置全服务器范围的计划散列键高速缓存。若要启用全服务器范围的计划高速缓存，请使用：

```
sp_configure "abstract plan cache", 1
```

对抽象计划使用其它 *set* 选项

您可以将其它 set 调优选项与 set plan dump、 show_abstract_plan 和 set plan load 结合使用。

使用 show_abstract_plan 查看计划

set option show_abstract_plan 可显示当前在 TDS 连接上运行的最佳抽象计划。它在优化后和执行前显示计划，是唯一一个不依赖跟踪标志 3604 或 3605 的 set option show_ 命令。

显示最终计划的抽象计划与查看 showplan 输出类似：它向用户提供信息，但抽象计划不保存在 sysqueryplans 中，并且如果您使用抽象计划 load 模式，则不会使用抽象计划。

以下示例显示当前在 TDS 连接上运行的最佳抽象计划：

```
1> set option show_abstract_plan on
2> go
1> select r1, sum(s1)
2> from r, s
3> where r2=s2
4> group by r1
The Abstract Plan (AP) of the final query execution plan:
( group_sorted ( nl_join ( i_scan ir12 r ) ( i_scan is21 s ) ) ) ( prop r
(parallel 1 ) ( prefetch 2 ) ( lru ) ) ( prop s ( parallel 1 ) ( prefetch 2 )
(lru ) )
To experiment with the optimizer behavior, this AP can be modified and then
passed to the optimizer using the PLAN clause:
SELECT/INSERT/DELETE/UPDATE ...
PLAN '( ... )'.
r1
----------- -----------
          1           2
          2           4

(2 rows affected)
```

使用 *showplan*

如果 showplan 已启用，并且已用 set plan load 启用了抽象计划关联模式，则 showplan 命令会在该语句 showplan 输出的开始位置输出匹配的抽象计划的计划 ID：

```
QUERY PLAN FOR STATEMENT 1 (at line 1).
Optimized using an Abstract Plan (ID : 832005995).
```

如果使用带 plan 子句的 SQL 语句运行查询，则 showplan 会显示：

```
Optimized using the Abstract Plan in the PLAN clause.
```

使用 *noexec*

您可以使用 noexec 模式捕获抽象计划，而不实际执行查询。如果 noexec 模式生效，则会优化查询并保存抽象计划，但不返回查询结果。

若要在捕获抽象计划时使用 noexec 模式，应在启用 noexec 模式之前，执行所有必需的过程 （如 sp_add_qpgroup）和其它 set 选项（如 set plan dump）。下例显示一组典型的步骤：

```
sp_add_qpgroup pubs_dev
go
set plan dump pubs_dev on
go
set noexec on
go
select type, sum(price) from titles group by type
go
```

使用 *fmtonly*

使用 fmtonly set 可以获取用于在存储过程中捕获计划的类似行为，而不实际执行存储过程。

```
sp_add_qpgroup pubs_dev
go
set plan dump pubs_dev on
go
set fmtonly on
go
exec stored_proc(...)
go
```

使用 *forceplan*

如果 set forceplan on 命令生效，并且查询关联也为该会话启用，则当使用完整抽象计划来优化查询时， forceplan 将被忽略。如果部分计划未完全指定连接顺序：

- 首先，按指定将抽象计划中的表排序。
- 其余表按照 from 子句中指定的顺序排序。
- 表的两列被合并。

全服务器范围的抽象计划捕获和关联模式

系统管理员可使用以下配置参数启用全服务器范围的计划捕获、关联和替换模式：

- abstract plan dump — 允许转储到缺省抽象计划捕获组 ap_stdout。
- abstract plan load — 允许从缺省抽象计划装载组 ap_stdin 进行装载。
- abstract plan replace — 当同时启用计划转储模式时，启用计划替换。
- abstract plan cache — 启用抽象计划散列 ID 的高速缓存； abstract plan load 也必须被启用。请参见第 303 页的 “set plan exists check 选项”。

缺省情况下，这些配置参数设置为 0，表示捕获模式和关联模式被禁用。若要启用某一模式，请将配置参数值设置为 1：

```
sp_configure "abstract plan dump", 1
```

对任何全服务器范围的抽象计划模式的启用都是动态的；不必重新启动服务器。

全服务器范围的捕获和关联允许系统管理员为服务器上的所有用户捕获所有计划。不能在会话级覆盖全服务器范围的模式。

使用 SQL 创建计划

可用以下方式为查询直接指定抽象计划：

- 使用 create plan 命令
- 将 plan 子句添加到 select、 insert...select、 update、 delete 和 return 命令，以及添加到 if 和 while 子句

有关编写计划的信息，请参见第 13 章 “抽象查询计划指南”。

使用 *create plan*

create plan 命令可指定查询的文本和要为查询保存的抽象计划。

下例创建了一个抽象计划：

```
create plan
   "select avg(price) from titles"
"(scalar_agg
    (i_scan type_price_ix titles)
)"
```

此计划被保存在当前活动的计划组中。也可指定组名：

```
create plan
   "select avg(price) from titles"
"(scalar_agg
    (i_scan type_price_ix titles)
)"
into dev_plans
```

如果当前计划组或您指定的计划组中已存在指定查询的计划，则您必须先启用 replace 模式才能覆盖现有计划。

为了查看用于您创建的计划的计划 ID，create plan 可将该 ID 作为变量返回。首先必须声明变量。以下为一个返回计划 ID 的例子：

```
create plan
   "select avg(price) from titles"
"(scalar_agg
    (i_scan type_price_ix titles)
)"
into dev_plans
and set @id

select @id
```

使用 create plan 时，并未执行计划中的查询。这意味着：

- 尚未分析查询文本，因此此查询未经过检查以确定 SQL 语法是否有效。
- 这些计划尚未经过检查以确定抽象计划语法是否有效。
- 这些计划尚未经过检查以确定是否与 SQL 文本兼容。

若要避免出现错误和问题，请在启用 showplan 的情况下立即执行指定查询。

使用 *plan* 子句

可将 plan 子句用于以下 SQL 语句，来指定用于执行查询的计划：

- select
- insert...select
- delete
- update
- if
- while
- return

以下为一个指定用于执行查询的计划的例子：

```
select avg(price) from titles
    plan
"(scalar_agg
    (i_scan type_price_ix titles
)"
```

为查询指定抽象计划时，使用指定的计划执行查询。如果已启用 showplan，则输出以下消息：

```
Optimized using the Abstract Plan in the PLAN clause.
```

如果将 plan 子句用于查询，则 SQL 文本和计划语法方面的任何错误以及计划和 SQL 文本间的不匹配都会作为错误被报告出来。例如，下面的计划对查询使用错误的抽象计划运算符：

```
/* wrong operator!*/
select * from t1,t2
where c11 = c21
plan
"(union
 (t_scan t1)
 (t_scan t2)
)"
```

此计划返回以下消息：

```
Abstract Plan (AP) Warning:An error occurred while applying the AP:
(union (t_scan t1) (t_scan2))
to the SQL query:
select * from t1, t2
where c11 = c21
Failed to apply the top operator 'union' of the following AP fragment:
```

```
(union (t_scan t1) (t_scan t2))
The query contains no union that matches the 'union' AP operator at this point.
The following template can be used as a basis for a valid AP:
(also_enforce (join (also_enforce (scan t1)) (also_enforce (scan t2)))
)
The optimizer will complete the compilation of this query; the query will be
executed normally.
```

仅当启用了计划捕获时，使用 plan 子句指定的计划才保存在 sysqueryplans 中。如果当前捕获组中已存在查询计划，可启用 replace 模式以替换现有计划。

第 13 章

抽象查询计划指南

本章涵盖了编写抽象计划时可能会用到的一些指南。

概述

抽象计划可用来指定所需的查询执行计划。对于强制连接顺序或者指定索引、 I/O 大小或其它查询执行选项的会话级和查询级选项，抽象计划为它们提供了替代方法。对这些会话级和查询级选项的描述详见第 12 章 “创建和使用抽象计划”。

有几个优化决策不能使用查询文本中的 set 命令或子句来指定，例如：

- 实现指定关系运算符的算法；例如 NLJ 与 MJ 与 HJ，或 GroupSorted 与 GroupHashing 与 GroupInserting
- 子查询连接
- 用于展平子查询的连接顺序
- 重新格式化

许多情况下，发出 T-SQL 命令时，不能包括 set 命令或更改查询文本。抽象计划提供了一种可替代的、更完整的影响优化程序决策的方法。

抽象计划是未包括在查询文本中的关系代数表达式。它们存储在系统目录中，并基于传入查询的文本与传入查询相关联。

抽象计划语言

抽象计划语言是使用这些运算符的关系代数：

- distinct — 描述重复项消除的逻辑运算符。
  - distinct_sorted — 描述可用的基于排序的重复项消除的物理运算符。
  - distinct_sorting — 描述基于排序的重复项消除的物理运算符。
  - distinct_hashing — 描述基于散列的重复项消除的物理运算符。
- group — 描述矢量集合的逻辑运算符。
  - group_sorted — 描述可用的基于排序的矢量集合的物理运算符。
  - group_hashing — 描述基于散列的矢量集合的物理运算符。
  - group_inserting — 描述基于聚簇索引插入的矢量集合的物理运算符。
- join — 使用嵌套循环连接、合并连接或散列连接描述内连接、外连接和存在连接的一般连接和高级逻辑连接运算符。
  - nl_join — 指定一个嵌套循环连接，包括所有内连接、外连接和存在连接。
  - m_join — 指定合并连接，包括内连接和外连接。
  - h_join — 指定散列连接，包括所有内连接、外连接和存在连接。
- union — 一个逻辑 union 运算符。它描述 union 和 union all 的 SQL 结构。
  - append_union_all — 实现 union all 的物理运算符。它会一个接一个地附加子结果集。
  - merge_union_all — 实现 union all 的物理运算符。它会合并在每个子级中排序的投影的子集上的子结果集，并保留该顺序。
  - merge_union_distinct — 实现 union [distinct] 的物理运算符。一个基于合并的重复项删除算法。
  - hash_union_distinct — 实现 union [distinct] 的物理运算符。一个基于合并的重复项删除算法。
- scalar_agg — 描述标量集合的逻辑运算符。

- scan — 转换行流中的存储表 （抽象计划派生表）的逻辑运算符。它允许不限制访问方法的部分计划。
  - i_scan — 实现 scan 的物理运算符。它指示优化程序对指定表使用索引扫描。
  - t_scan — 实现 scan 的物理运算符。它指示优化程序对指定表使用全表扫描。
  - m_scan — 实现 scan 的物理运算符。它指示优化程序对指定的表 index union 和 / 或 index intersection 使用多索引表扫描。
- store — 描述抽象计划派生表在存储工作表中的实现的物理运算符。
- store_index — 描述抽象计划派生表在聚簇索引存储工作表中的实现的物理运算符；优化程序会选择有用的键列。
- sort — 描述抽象计划派生表的排序的物理运算符；优化程序会选择有用的键列。
- nested — 描述嵌套子查询的放置和结构的过滤器。
- xchg — 描述抽象计划派生表的随即重新分区的物理运算符。抽象计划提供目标分区度，但优化程序会选择有用的目标分区。

以下附加抽象计划关键字用于分组和标识：

- sequence — 在序列需要多个步骤时对元素进行分组。
- hints — 对某个部分计划的一组提示进行分组。
- prop — 为表引入一组扫描属性：prefetch、 lru|mru 和 parallel。
- table — 在使用相关名时以及在子查询或视图中标识表。
- work_t — 标识工作表。
- in — 和 table 一起用于标识在子查询 (subq) 或视图 (view) 中指名的表。
- subq — 在嵌套运算符中使用，以指示嵌套子查询的连接点，以及引入子查询的抽象计划。

对于其新的对等形式，仍接受所有旧抽象计划运算符 （如 g_join）。

查询、访问方法和抽象计划

对任意特定表，某个特定查询可能有多种访问方法：使用不同索引的索引扫描、表扫描、 OR 策略和重新格式化。

以下简单查询可选择若干访问方法：

```
select * from t1
where c11 > 1000 and c12 < 0
```

以下抽象计划指定了三种不同的访问方法：

- 使用索引 i_c11：

  ```
  (i_scan i_c11 t1)
  ```

- 使用索引 i_c12：

  ```
  (i_scan i_c12 t1)
  ```

- 进行全表扫描：

  ```
  (t_scan t1)
  ```

- 进行多次扫描；即，根据复杂子句对表的多个索引进行联合或交集运算 （因此下面的示例中使用了较复杂的查询）：

```
select * from t1
where (c11 > 1000 or c12 < 0) and (c12 > 1000 or c112 < 0)
plan
"(m_scan t1)"
```

抽象计划可以是为查询指定了所有优化程序选择的完整计划，或者抽象计划可以指定这些选择的一个子集 （例如在查询中用于单张表的索引），但不能是表的连接顺序。例如，使用部分抽象计划，可以指定上述查询应使用某个索引并让优化程序在 i_c11 和 i_c12 之间选择，但不能执行全表扫描。在索引名的位置使用空的小括号：

```
(i_scan () t1)
```

此外，查询可以使用 2K 或 16K I/O，或者以串行或并行方式执行。

派生表

派生表通过对查询表达式求值来定义并且不同于常规表，因为它既没有在系统目录中被描述，也没有存储在磁盘上。在 Adaptive Server 中，派生表可以是 SQL 派生表或抽象计划派生表。

- SQL 派生表 — 由一个或多个表通过计算查询表达式来定义。SQL 派生表在定义它的查询表达式中使用，并且仅在查询期间存在。请参见 《Transact-SQL 用户指南》。
- 抽象计划派生表 — 在查询处理、查询优化和查询执行中使用的派生表。抽象计划派生表与 SQL 派生表的区别在于：它是作为抽象计划的一部分而存在的，最终用户看不到它。

标识表

抽象计划必须明确命名某个查询的所有表，以便在抽象计划中命名的表可以链接到它在 SQL 查询中的出现位置。大多数情况下，只需表名即可。如果查询用数据库和所有者名限定表名，则也需使用数据库和所有者名完全标识抽象计划中的表。例如，下面的示例使用了非限定的表名：

```
select * from t1
```

抽象计划也使用非限定名称 (t_scan t1)。如果查询中提供了数据库名称或所有者名称：

```
select * from pubs2.dbo.t1
```

抽象计划必须进行限定 (t_scan pubs2.dbo.t1)。不过，相同的表可能在同一查询中出现多次，如下例所示：

```
select * from t1 a, t1 b
```

相关名 （上述示例中的 a 和 b）用来标识 SQL 中的两个表。在抽象计划中， table 运算符将每个相关名与表的出现位置相关联：

```
(join
        (t_scan (table (a t1)))
        (t_scan (table (b t1)))
)
```

您还可以使用更简要的抽象计划，该计划仅使用相关名：

```
(join
  (t_scan a)
  (t_scan b)
)
```

在视图和子查询中，表名也可能是不明确的，因此 table 运算符可同时用于视图和子查询中的表。

对于子查询，in 和 subq 运算符通过子查询的句法包含的表名来限定它。在下例中，同一个表用于外部查询和子查询：

```
select *
from t1
where c11 in (select c12 from t1 where c11 > 100)
```

抽象计划可明确标识这些表：

```
(join
    (t_scan t1)
    (i_scan i_c11_c12 (table t1 (in (subq 1))))
)
```

对于视图， in 和 view 运算符可提供标识。下例中的查询将引用视图中使用的一个表：

```
create view v1
as
select * from t1 where c12 > 100
select t1.c11 from t1, v1
    where t1.c12 = v1.c11
```

下面是抽象计划：

```
(join
    (t_scan t1)
    (i_scan i_c12 (table t1 (in (view v1))))
)
```

在 Adaptive Server 生成的抽象计划中，仅对需要使用限定的表名称来消除名称不确定性的表生成视图或子查询限定的表名称。对于其它表，仅生成名称。

在用户创建的抽象计划中，需要使用视图或子查询限定的表名称，以防出现不确定性；这两种语法的其它形式也是可以接受的。

标识索引

i_scan 运算符需要两个操作数：索引名称和表名称，如下所示：

```
(i_scan i_c12 t1)
```

若要指定某个索引应被使用，但不指定该索引，可用空的小括号替换索引名：

```
(i_scan () t1)
```

指定连接顺序

Adaptive Server 通过连接两个表并将来自该连接的抽象计划派生表连接到连接顺序中的下一个表，来执行三个或更多个表的连接。此抽象计划派生表属于一个行流，出自查询执行中一个以前的嵌套循环连接。

以下查询连接三个表：

```
select *
from t1, t2, t3
where c11 = c21
    and c12 = c31
    and c22 = 0
    and c32 = 100
```

下面的示例使用 join 运算符显示连接算法的二进制性质。以下计划指定连接顺序 t2、 t1、 t3：

```
(join
    (join
        (scan t2)
        (scan t1)
    )
    (scan t3)
)
```

t2-t1 连接的结果随后连接到 t3。下例中的 scan 运算符让优化程序选择表扫描或索引扫描。

用于连接的速记注释

通常，一个 *N* 向深度嵌套循环左连接 （顺序为 t1、 t2、 t3...、 tN-1、 *tN*）的描述如下：

```
(join
    (join
        ...
            (join
                (join
                    (scan t1)
                    (scan t2)
                )
                (scan t3)
            )
        ...
        (scan tN-1)
    )
    (scan tN)
)
```

以下表示法可用作 nl_join 运算符的速记形式：

```
(nl_join
    (scan t1)
    (scan t2)
    (scan t3)
    ...
    (scan tN-1)
    (scan tN)
)
```

连接顺序示例

优化程序可从多个计划中为此三向连接查询进行选择：

```
select *
from t1, t2, t3
where c11 = c21
    and c12 = c31
    and c22 = 0
    and c32 = 100
```

以下是一些示例：

- 在 t2 中使用 c22 作为搜索参数，并和 c11 的 t1 连接，然后和 c31 的 t3 连接：

```
(nl_join
    (i_scan i_c22 t2)
    (i_scan i_c11 t1)
    (i_scan i_c31 t3)
)
```

- 使用 t3 的搜索参数和连接顺序 t3、 t1、 t2：

```
(nl_join
    (i_scan i_c32 t3)
    (i_scan i_c12 t1)
    (i_scan i_c21 t2)
)
```

- 执行 t2 的全表扫描，如果 t2 较小且适合放入高速缓存中，则仍使用连接顺序 t3、 t1、 t2：

```
(nl_join
    (i_scan i_c32 t3)
    (i_scan i_c12 t1)
    (t_scan t2)
)
```

- 如果 t1 非常大，并且 t2 和 t3 分别限定 t1 的大部分，但二者合起来却是非常小的一部分，则此计划指定星型连接：

```
(nl_join
    (i_scan i_c22 t2)
    (i_scan i_c32 t3)
    (i_scan i_c11_c12 t1)
)
```

连接运算符在实现任意外连接、内连接和存在连接方面是通用的；优化程序会根据查询语义选择正确的连接语义。

执行方法和抽象计划之间的匹配

对连接顺序和连接类型有某些限制，这取决于查询类型。一个示例是外部连接，例如：

```
select *
from t1 left join t2
on c11 = c21
```

在连接处理过程中， Adaptive Server 要求外连接的外部成员是外部表。因此，以下抽象计划是非法的：

```
(join
    (scan t2)
    (scan t1)
)
```

尝试使用此计划会生成错误消息， AP 应用程序将失败，并且优化程序会尽最大努力尝试完成查询编译。

使用视图为查询指定连接顺序

可使用抽象计划来强制合并视图的连接顺序。以下示例可创建执行 t2 和 t3 的连接的视图：

```
create view v2
as
select *
from t2, t3
where c22 = c32
```

以下查询执行一个和视图中 t2 的连接：

```
select * from t1, v2
where c11 = c21
    and c22 = 0
```

以下抽象计划指定了连接顺序 t2、 t1、 t3：

```
(nl_join
    (scan t2)
    (scan t1)
    (scan t3)
)
```

由于表名是明确的，因此不需要视图限定。但是，以下抽象计划也是合法的，并且具有相同的含义：

```
(nl_join
     (scan (table t2(in(view v2))))
     (scan t1)
     (scan (table t3 (in (view v2))))
)
```

下例将连接视图中的 t3：

```
select * from t1, v2
where c11 = c31
    and c32 = 100
```

以下计划使用连接顺序 t3、t1、t2：

```
(join
    (scan t3)
    (scan t1)
    (scan t2)
)
```

在此示例中，如果 set forceplan 无法影响查询的连接顺序，则可根据需要使用抽象计划来影响该顺序。

指定连接类型

Adaptive Server 可执行嵌套循环连接、合并连接或散列连接。 join 运算符允许优化程序根据开销自由选择最佳连接算法。若要指定嵌套循环连接，请使用 nl_join 运算符；若要指定合并连接，请使用 m_join 运算符，若要指定散列连接，请使用 h_join 运算符。Adaptive Server 捕获的抽象计划始终包含用于指定算法的运算符，而非 join 运算符。

以下查询将指定一个 t1 和 t2 之间的连接：

```
select * from t1, t2
    where c12 = c21 and c11 = 0
```

以下抽象计划指定一个嵌套循环连接：

```
(nl_join
    (i_scan i_c11 t1)
    (i_scan i_c21 t2)
)
```

嵌套循环计划使用索引 i_c11 来限制使用搜索子句的扫描，然后使用连接列上的索引来执行与 t2 的连接。

以下合并连接计划使用不同的索引：

```
(m_join
    (i_scan i_c12 t1)
    (i_scan i_c21 t2)
)
```

合并连接使用连接列上的索引 i_c12 和 i_c21 作为合并键。以下查询执行全合并连接并且不需要排序。

合并连接还可使用 i_c11 上的索引来仅选择匹配行，但随后需要排序才能提供所需的顺序。

```
(m_join
    (sort
        (i_scan i_c11 t1)
    )
    (i_scan i_c21 t2)
)
```

最后，此计划在内侧执行散列连接和全表扫描：

```
(h_join
   (i_scan i_c11 t1)
   (t_scan t2)
)
```

指定部分计划和提示

有时并不需要完整计划，例如，如果某个查询计划的唯一问题是优化程序选择了表扫描而不是使用非聚簇索引，则抽象计划只能指定索引选择，而由优化程序进行其它决策。

对于以下查询，优化程序可以选择 t3 的表扫描而不使用 i_c31：

```
select *
from t1, t2, t3
where c11 = c21
    and c12 < c31
    and c22 = 0
    and c32 = 100
```

由优化程序生成的以下计划将指定连接顺序 t2、t1、t3。但是，此计划指定了 t3 的表扫描：

```
(nl_join
    (i_scan i_c22 t2)
    (i_scan i_c11 t1)
    (t_scan t3)
)
```

可对这一完整计划进行修改以指定改用 i_c31：

```
(nl_join
    (i_scan i_c22 t2)
    (i_scan i_c11 t1)
    (i_scan i_c31 t3)
)
```

但是，只指定部分抽象计划是更为灵活的解决办法。当该查询中的其它表数据变化时，优化的连接顺序可以更改。部分计划只能指定一个部分计划项目。对于 t3 的索引扫描，部分计划比较简单：

```
(i_scan i_c31 t3)
```

优化程序为 t1 和 t2 选择连接顺序和访问方法。

使用逻辑运算符而不是物理运算符时，抽象计划是部分计划。例如，尽管以下抽象计划包含整个查询，但它仍是部分计划，因为它允许优化程序选择连接算法和访问方法：

```
(join
   (scan t1)
   (scan t2)
   (scan t3)
)
```

部分计划在顶部可能也是不完整的，因为抽象计划的根可能只包含查询的一部分。如果出现这种情况，则优化程序会完成该计划：

```
(nl_join
   (t_scan t1)
   (t_scan t2)
)
```

但是，抽象计划中提供的计划片段在叶以上都必须是完整的。例如，以下抽象计划“hash join t1 outer to something”是非法的。

```
(h_join
   (t_scan t1)
   ()
)
```

对多个提示分组

有时会需要多个计划片段。例如，可能希望指定某个索引应用于查询中的每个表，而让优化程序选择连接顺序。需要多个提示时，可使用 hints 运算符对它们进行分组：

```
(hints
    (i_scan () t1)
    (i_scan () t2)
    (i_scan () t3)
)
```

在这种情况下， hints 运算符仅起语法作用；它不会影响扫描的顺序。

对于提示的内容没有限制。部分连接顺序可以与部分访问方法混合。以下提示指定，在连接顺序中， t2 在 t1 外部， t3 的扫描应使用索引，而优化程序可以为 t3 选择索引，为 t1 和 t2 选择访问方法，为 t3 选择位置：

```
(hints
    (join
        (scan t2)
        (scan t1)
    )
    (i_scan () t3)
)
```

关于使用提示的不一致和非法计划

可以使用提示来说明不一致的计划，例如以下指定对立连接顺序的计划：

```
(hints
    (join
        (scan t2)
        (scan t1)
    )
    (join
        (scan t1)
        (scan t2)
    )
)
```

当执行与计划相关联的查询时，该查询不能被编译，并且会导致错误。

其它不一致的提示不会引发例外，但可以使用任意指定的访问方法。以下计划为同一个表指定索引扫描和表扫描：

```
(hints
    (t_scan t3)
    (i_scan () t3)
)
```

在这种情况下，两种方法均可选择，并且行为是不确定的。

为子查询创建抽象计划

子查询在 Adaptive Server 中有多种解析方法，抽象计划将反映查询执行步骤：

- 实现 — 执行子查询并将结果存储在工作表或内部变量中。请参见第 325 页的 “实现子查询”。
- 展平 — 查询展平到连接，其中，表位于主查询中。请参见第 326 页的 “展平子查询”。
- 嵌套 — 对每个外部查询行执行一次子查询。请参见第 327 页的 “嵌套子查询”。

抽象计划不允许选择基本子查询解析方法。这是基于规则的决策，在查询优化期间不能更改。但是，抽象计划可用来影响外部查询和内部查询的计划。在嵌套子查询中，抽象计划也可用来选择子查询在外部查询中嵌套的位置。

实现子查询

以下查询包括一个可以实现的非相关子查询：

```
select *
from t1
where c11 = (select count(*) from t2)
```

抽象计划的第一步实现子查询中的标量集合。第二步使用该结果扫描 t1：

```
( sequence
    (scalar_agg
        (i_scan i_c21 t2)
    )
    (i_scan i_c11 t1)
)
```

展平子查询

某些子查询可被展平到连接中。join、 nl_join、 m_join 和 h_join 运算符让优化程序来检测何时需要存在连接。例如，以下查询包括一个由 exists 引入的子查询：

```
select * from t1
where c12 > 0
    and exists (select * from t2
            where t1.c11 = c21 and c22 < 100)
```

查询的语义要求 t1 和 t2 之间的一个存在连接。连接顺序 t1、t2 由优化程序解释为半连接，并且对于 t1 中的每个限定行，对 t2 的扫描在 t2 的第一个匹配行处停止：

```
(join
    (scan t1)
    (scan t2)
)
```

连接顺序 t2、 t1 要求采用其它方法来确保不出现重复：

```
(join
    (distinct
       (scan t2)
    )
    (scan t1)
)
```

利用此抽象计划，优化程序可决定使用：

- t2.c21 上的唯一索引 （如果存在）及常规连接。
- 唯一重新格式化策略 （如果不存在唯一索引）。在此情况下，查询可能会使用 c22 上的索引来选择行并放入工作表。
- 重复排除的排序优化策略，执行常规连接并选择结果然后放入工作表，再对工作表进行排序。

抽象计划不需要指定最后两个选项所需的工作表的创建和扫描。

更改展平子查询中的连接顺序的示例

以下查询可被展平到一个存在连接中：

```
select *
from t1, t2
where c11 = c21
    and c21 > 100
   and exists (select * from t3 where c31 != t1.c11)
```

“!=”相关可能导致 t3 的扫描非常费时。如果连接顺序是 t1、 t2，则 t3 在连接顺序中的最佳位置取决于 t1 和 t2 的连接是增加还是减少行数以及据此得出的费时的表扫描需执行的次数。如果优化程序查找 t3 的正确连接顺序失败，则当连接将减少 t3 必须被扫描的次数时，可使用以下抽象计划：

```
(nl_join
    (scan t1)
    (scan t2)
    (scan t3)
)
```

如果连接将增加 t3 需要被扫描的次数，则此抽象计划将在连接前执行 t3 的扫描：

```
(nl_join
    (scan t1)
    (scan t3)
    (scan t2)
)
```

嵌套子查询

如果出现以下情况，可在抽象计划中显式描述嵌套子查询：

- 将提供子查询的抽象计划。
- 将指定子查询连接到主查询的位置。

抽象计划允许影响用于子查询的查询计划，还允许更改外部查询中子查询的连接点。

nested 运算符指定子查询在外部查询中的位置。子查询 “嵌套”在特定抽象计划派生表中。优化程序将选择一点，其中所有用于外部查询的相关列都是可用的，并且它将估计子查询需要被执行的最低次数。

以下 SQL 语句包含相关表达式子查询：

```
select *
from t1, t2
where c11 = c21
    and c21 > 100
    and c12 = (select c31 from t3
                where c32 = t1.c11)
```

抽象计划显示了嵌套在 t1 扫描中的子查询：

```
(nl_join
    (nested
        (i_scan i_c12 t1)
        (subq
            (scalar_agg
                (scan t3)
            )
        )
    )
    (i_scan i_c21 t2)
)
```

集合在第 2 章 “使用 showplan”中进行了介绍。scalar_agg 抽象计划运算符是必需的，因为所有抽象计划（甚至是部分计划）在叶以上都必须是完整的。

子查询标识和连接

SQL 查询中的子查询使用其基础表与抽象计划子查询进行匹配。表已明确标识，子查询也是如此。例如：

```
select
   (select c11 from t1 where c12 = t3.c32), c31
from t3
where
   c32 > (select c22 from t2 where c21 = t3.c31)
plan
"(nested
    (nested
        (t_scan t3)
        (subq
             (i_scan i_c11_c12 t1)
   )
)
    (subq
        (i_scan i_c21 t2)
   )
)"
```

但是，当表名不明确时，则需要子查询的标识以解决表名不明确的问题。

子查询用数字按其前导开括号 “(” 中的顺序来标识。

下面的示例有两个子查询；两个都指向表 t1：

```
select 1
from t1
where
   c11 not in (select c12 from t1)
   and c11 not in (select c13 from t1)
```

在抽象计划中，从 c12 突出的子查询被命名为 “1”，从 c13 突出的子查询被命名为 “2”。

```
(nested
    (nested
        (t_scan t1)
        (subq
           (scalar_agg
                (i_scan i_c11_c12 (table t1 (in (subq 1))))
           )
        )
    )
    (subq
        (scalar_agg
            (i_scan i_c13 (table t1 (in (subq 2))))
        )
    )
)
```

在以下查询中，第二个子查询嵌套在第一个子查询中：

```
select * from t1
where c11 not in
    (select c12 from t1
     where c11 not in
     (select c13 from t1)
```

在这种情况下，从 c12 中突出的子查询也被命名为 “1”，从 c13 中突出的子查询也被命名为 “2”。

```
(nested
   (t_scan t1
   (subq
      (scalar_agg
         (nested
            (i_scan i_c12 (table t1 (in (subq 1))))
            (subq
```

```
                (scalar_agg
                   (i_scan i_c21 (table t1 (in (subq 2))))
                 )
              )
           )
        )
   )
)
```

更多子查询示例：读取顺序与连接

nested 运算符将抽象计划派生表作为第一个操作数，将嵌套子查询作为第二个操作数。这允许简单的连接顺序和子查询位置的垂直读取：

```
select *
from t1, t2, t3
where c12 = 0
    and c11 = c21
    and c22 = c32
    and 0 < (select c21 from t2 where c22 = t1.c11)
```

在此计划中，连接顺序是 t1、 t2、 t3，其中子查询被嵌套在 t1 的扫描中：

```
(nl_join
    (nested
        (i_scan i_c11 t1)
        (subq
            (t_scan (table t2 (in (subq 1)))
        )
    )
    (i_scan i_c21 t2)
    (i_scan i_c32 t3)
)
```

修改子查询嵌套

如果修改子查询的连接点，则必须选择一个所有相关列都可用的点。此查询与外部查询中的两个表都相关：

```
select *
from t1, t2, t3
where c12 = 0
    and c11 = c21
    and c22 = c32
    and 0 < (select c31 from t3 where c31 = t1.c11
                    and c32 = t2.c22)
```

此计划使用连接顺序 t1、t2、t3，其中子查询嵌套在 t1-t2 的连接中：

```
(nl_join
    (nested
        (nl_join
            (i_scan i_c11_c12 t1)
            (i_scan i_c22 t2)
        )
        (subq
            (t_scan (table t3 (in (subq 1))))
        )
    )
    (i_scan i_c32 t3)
)
```

由于子查询需要两个外部表中的列，因此对 t1 的扫描或 t2 的扫描嵌套该子查询可能不正确；此类错误在优化过程中会得到更正，并且不出现提示。

但是，以下抽象计划会使合法请求对三表连接嵌套子查询：

```
(nested
  (nl_join
    (i_scan i_c11_c12 t1)
    (i_scan i_c22 t2)
    (i_scan i_c32 t3)
  )
  (subq
    (t_scan (table t3 (in (subq 1))))
  )
)
```

用于具体处理视图的抽象计划

多数情形下，视图处理会合并主查询中的视图定义。但是，有时候需要实现视图，例如在自连接时：

```
create view v3(cc31, sum_c32)
as
select c31, sum(c32)
from t3
group by c31

select *
from v3 a, v3 b
where a.c31 = b.c31
```

在这种情况下，抽象计划会公开该工作表并存储实现该工作表的运算符。该工作表的两个扫描通过其相关名进行标识：

```
(sequence
  (store
    (group_sorted
      (i_scan i_c31 t3)
    )
  )
  (m_join
  (sort
    (t_scan (work_t (a Worktable)))
  ( sort
    (t_scan (work_t (b Worktable)))
  )
  )
)
```

下一节介绍抽象计划中的矢量集合的处理。

包含集合的查询的抽象计划

以下查询返回一个标量集合：

```
select max(c11) from t1
```

存在实现标量集合的物理运算符，因此，优化程序没有选择权。但是，选择 c11 上的索引将允许 max() 优化：

```
(scalar_agg
  (i_scan ic11 t1)
)
```

由于标量集合是顶部抽象计划运算符，因此删除它与使用以下部分计划会产生相同的结果：

```
(i_scan ic11 t1)
```

当 scalar_agg 抽象计划是子查询的一部分时，通常需要该抽象计划，并且该抽象计划还必须包含父查询。

矢量集合有所不同，因为可使用多个物理运算符来实现组逻辑运算符，这意味着优化程序需要做出选择。因此，抽象计划可强制执行它。

```
select max(c11)
from t1
group by c12
```

以下抽象计划示例会强制执行三个矢量集合算法中的每一个：

注释　group_sorted 需要对分组列进行排序，所以它需要使用索引。

```
(group_sorted
  (i_scan i_c12 t1)
)
(group_hashing
  (t_scan t1)
)

(group_inserting
  (t_scan t1)
)
```

包含联合的查询的抽象计划

union 抽象计划运算符描述包含联合的 SQL 查询的计划：

```
select*
from
  t1,
  (select * from t2
  union
  select * from t3
 ) u(u1, u2)
where c11=u1
plan
"(nl_join
 (union
  (t_scan t2)
  (t_scan t3)
 )
 (i_scan i_c11 t1)
)"
```

SQL 中有两种类型的 union：union distinct 和 union [all]。 union [all] 是缺省值。

m_union_distinct 和 h_union_distinct 抽象计划运算符可强制执行基于合并或基于散列的 UNION DISTINCT 重复项删除操作。将这些运算符与 UNION ALL 结合使用是非法的。基于合并的算法需要每个联合子级中涵盖所有联合投影列的排序。

在下面的示例中，由 (c11, c12) 组合索引为第一个子级提供所需的排序，由 sort 为第二个子级提供所需的排序。

```
select c11, c12 from t1
union distinct
select c21, c22 from t2
plan
"(m_union distinct
 (i_scan i_c11_c12 t1)
 (sort
  (t_scan t2)
 )
)"
```

union_all 和 m_union_all 抽象计划运算符可强制执行基于附加或基于合并的 UNION ALL。将这些运算符与 UNION DISTINCT 结合使用是非法的。合并算法本身不需要排序；它会将子级中的任何有用排序提供给父级。

在下面的示例中，由两个 i_scan 运算符提供的排序被其 m_union_all 父级提供给上面的 m_join。

```
select *
from
  t1,
  (select c21, c22 from t2
   union
   select c31, c32 from t3
) u(u1, u2)
where c11=u1
plan
"(m_join
 (m_union_all
   (i_scan i_c21 t2)
   (i_scan i_c31 t3)
 )
 (i_scan i_c11 t1)
)"
```

在查询需要排序时使用抽象计划

ORDER BY 查询中显式需要排序，或者基于合并的运算符（如 m_join、m_union_distinct 和 group_sorted）中隐式需要排序。

排序由 sort 抽象计划运算符显式生成（优化程序对已知需要排序的所有列生成排序键），或由索引列上的 i_scan 隐式生成。

所有需要排序的基于合并的运算符都在其结果中保持该排序，以供也需要该排序的父级使用。

在下面的示例中，t1 的 i_scan 提供 m_join 所需的排序。t2 的 i_scan 和对 t3 的扫描的排序提供了 m_union_distinct 所需的排序。此排序还提供 m_join 所需的排序。最后，不需要顶部排序，因为 ORDER BY 所需的排序由 m_join 提供。

```
select *
from
  t1,
  (select c21, c22 from t2
    union distinct
    select c31, c32 from t3
) u(u1, u2)
where c11=u1
order by c11, u2
plan
"(m_join
    (m_union_distinct
        (i_scan i_c21_c22 t2)
        (sort
            (t_scan t3)
        )
    )
    (i_scan i_c11 t1)
)"
```

指定重新格式化策略

在此查询中， t2 很大，而且没有索引：

```
select *
from t1, t2
where c11 > 0
    and c12 = c21
    and c22 = 0
```

指定对 t2 的重新格式化策略的抽象计划是：

```
(nl_join
    (t_scan t1)
    (store_index
        (t_scan t2)
    )
)
```

必须将 store_index 抽象计划运算符放在 nl_join 的内侧。可将它放在任何抽象计划之上；不再有单表扫描限制。仍接受旧的 `(scan (store... ))` 语法。

指定 OR 策略

OR 策略使用一组索引扫描来限制每个 OR 术语的扫描，然后将生成的行 ID 通过 UnionDistinct 运算符传递到 get，其中 RidJoin 来自表，元组与唯一行 ID 相对应。

m_scan （多扫描）抽象计划运算符强制执行索引联合，因此形成 OR 策略：

```
select * from t1
where c11 > 10 or c12 > 100
plan
"(m_scan t1)"
```

未指定 *store* 运算符时

将元组的流存储到工作表中以满足某个算法（Sort、GroupInserting 等）的运算符内需求，被视为算法的实现详细信息，因此不在抽象计划中公开。

抽象计划仅公开出于交互运算符原因创建的工作表，如自连接实现视图。这种情况下，没有一个运算符需要工作表。原因在于计划的全局性质，更确切地说就是，中间派生表计算一次但使用两次。

并行处理的抽象计划

并行扫描的分区表会生成元组的分区流。不同的运算符对并行处理具有特定的要求。例如，在所有连接中，要么必须均分两个子级，要么必须复制一个子级。

抽象计划 xchg 运算符强制优化程序对其子派生表进行 *n* 向即时重新分区。抽象计划仅提供分区度。优化程序选择最有用的分区列和样式（散列、范围、列表或循环）。

在下面的示例中，假定在 join 列上对 t1 和 t2 进行双向和三向散列分区，并且 i_c21 是本地索引：

```
select *
from t1, t2
where c11=c21
```

以下抽象计划对 t1 进行三向重新分区，执行三向并行 nl_join、序列化结果，并将单个数据流返回到客户端：

```
(xchg 1
  (nl_join
    (xchg 3
      (t_scan t1)
    )
    (i_scan i_c21 t2)
  )
)
```

不必指定 t2 的并行扫描。它可进行三向散列分区，并且由于它与 xchg-3 进行连接，因此没有其它计划是合法的。

以下抽象计划可对 t1 和 t2 进行并行扫描和排序，将它们分区后，对它们进行序列化以用于 m_join：

```
(m_join
    (xchg 1
        (sort
            (t_scan t1)
        )
    )
    (xchg 1
        (sort
            (t_scan t2)
        )
    )
)
(prop t1 (parallel 2))
(prop t2 (parallel 3))
```

并行抽象计划构造用于确保优化程序选择具有本机分区度的并行扫描。

有关编写抽象计划的提示

以下是一些有关编写和使用抽象计划的附加提示：

- 查看当前查询计划以及查看与所需编写计划具有相同查询执行步骤的计划。与从头编写一个完整计划相比，修改一个现有计划通常更为简单。
  - 捕获用于查询的计划。
  - 使用 sp_help_qplan 显示 SQL 文本和计划。
  - 编辑此输出来生成 create plan 命令，或使用 plan 子句将一个已编辑的计划连接到 SQL 查询。
- 当大多数优化程序决策是合适的，但只有某一部分 （例如一个索引选择）需要改进时，最好为查询调优指定部分计划。

  通过使用部分计划，当其它表中的数据更改时，优化程序可以为这些表选择其它路径。
- 保存后，抽象计划将变为静态的。数据量和数据分配可能发生变化，导致保存的抽象计划不再是最优的。

通过添加索引、对表进行分区或添加缓冲池而进行的后续调优更改意味着某些保存的计划在当前条件下执行情况可能不够好。大多数情况下，请使用可解决特定问题的少量抽象计划。

执行定期计划检查以检验保存的计划是否仍优于优化程序将要选择的计划。

在查询级别使用抽象计划

您可以使用抽象计划强制查询处理器选择的查询计划允许多个查询级设置。有关在查询级别使用抽象计划的详细信息，请参见第 7 章“控制优化”。

优化条件可由以下 set 语句在会话级处理：

```
set
    nl_join|merge_join|hash_join|...
    on | off
```

use ... 抽象计划语法接受在抽象计划派生表前使用任何数目的 use 表单。在 Adaptive Server 15.0 之前的版本中， optgoal 和 opttimeout 不能位于派生表的同一抽象计划中。例如，需要从查询中的 optgoal 语句中分离以下语句：

```
select ...
    plan
     "(use opttimeoutlimit 10) (i_scan r)"
```

但是，可通过以下方式在同一抽象计划中包括多个语句：

- 使用多个 use 语句。例如：

  ```
  select ...
      plan
      "(use optgoal allrows_dss) (use nl_join off)
  (...)"
  ```

- 将多个项目放在一个 use 表单中。例如：

  ```
  select ...
      plan
      "(use (optgoal allrows_dss) (nl_join off))
  (...)"
  ```

在查询级别，将优化目标 (opt_goal) 或超时 (opttimeout) 设置与 use ... 抽象计划语法结合使用。在会话级，将以下设置与 set plan ... 语法结合使用：

- 优化目标。
- 优化超时

例如，将 r 连接到 s 外部并启用 hash_join 而不设置优化目标 (opt_goal)：

```
select ...
>
>            plan
>
            "(use hash_join on)
>
>             (join (scan r) (scan s))"
```

本例使用 opt_goal 和 allrows_oltp 语句，但启用 hash_join：

```
select ...
>
>     plan
>
>      "(use opt_goal allrows_oltp)(use hash_join
on)"
```

在查询级别设置优化目标和优化条件时， use 语句的顺序不会影响结果。

- 先设置抽象计划优化目标，这会为优化条件设置优化目标缺省值。
- 您可以设置抽象计划优化 （它将取代优化目标缺省条件），然后设置优化目标。

抽象计划和优化程序条件的运算符名称的一致性

算法的名称因您在抽象计划中使用它们的方式和在 set 命令中使用它们的方式而异。例如，散列连接在抽象计划中称为 h_join，但在 set 命令中称为 hash_join。 Adaptive Server 在扩展的抽象计划语法中同时接受这两种关键字。例如：

```
select ...

plan

"(h_join (t_scan r) (t_scan s))"
```

等效于：

```
select ...

   plan

   "(hash_join (t_scan r) (t_scan s))"
```

以及：

```
select ...
plan
"(use h_join on)"
```

以及：

```
select ...
plan
"(use hash_join on)"
```

当表抽象计划存在时，表抽象计划优化：

```
select ..
from r, s, t
...
plan
"(use hash_join off)
(h_join (t_scan r) (t_scan s))"
```

查询对 r 和 s 扫描使用 hash_join ；但对于与 t 的连接，它不使用由 use 抽象计划表单指定的 hash_join，因为表抽象计划中未指定它。

扩展优化程序条件 *set* 语法

set *opt criteria* 语句接受 on/off/default，其中 default 表示您将当前优化目标设置用于此优化条件 （有关完整的 set 语法，请参见 《参考手册：命令》）。

比较更改前后的计划

使用抽象查询计划评估 Adaptive Server 软件升级或系统调优更改对查询计划的影响。进行更改之前必须保存计划，然后执行升级或调优更改，最后再次保存计划并比较计划。基本步骤组合为：

1 通过将配置参数 abstract plan dump 设置为 1，可启用全服务器范围的捕获模式。随后所有计划都将捕获到缺省组 ap_stdout 中。

2 允许足够的时间用于捕获的计划以表示系统中运行的大多数查询。可以通过检查 sysqueryplans 中 ap_stdout 组的行数是否稳定来检查是否正生成附加计划：

```
select count(*) from sysqueryplans where gid = 2
```

3 使用 sp_copy_all_qplans 将所有计划从 ap_stdout 复制到 ap_stdin（或其它某个组，如果您不希望使用全服务器范围的计划装载模式）。

4 使用 sp_drop_all_qplans 从 ap_stdout 中删除所有查询计划。

5 执行升级或调优更改。

6 留出足够的时间以便将计划捕获到 ap_stdout。

7 使用 sp_cmp_all_qplans 的 diff 模式参数，在 ap_stdout 和 ap_stdin 中比较计划。例如，以下查询比较 ap_stdout 和 ap_stdin 中的所有计划：

```
sp_cmp_all_qplans ap_stdout, ap_stdin, diff
```

这只显示两组中关于不同计划的信息。

启用全服务器范围的捕获模式的效果

启用全服务器范围的捕获模式时，可被优化的全部查询的计划被保存在服务器的所有数据库中。一些可能的系统管理影响有：

- 捕获计划后，会将计划保存到 sysqueryplans 中，并生成日志记录。计划和日志记录所需的空间量取决于 SQL 语句和查询计划的大小和复杂程度。检查每个数据库 （其中的用户是活动的）中的空间。

  可能需要频繁执行事务日志转储，特别是在全服务器范围捕获的早期中正生成许多新计划时。

- 如果用户执行 master 数据库中的系统过程，并且在装载 installmaster 时启用了全服务器范围的计划捕获，则可在系统过程中优化的语句的计划将保存在 master.sysqueryplans 中。

  对启用计划捕获后创建的任何用户定义的过程也是如此。如果 master 中的空间有限，您可能需要在登录时为所有用户 （包括系统管理员）提供缺省数据库。

- sysqueryplans 表使用数据行锁定来减少锁争用。但是，在保存大量新计划时，可能会对执行性能产生轻微影响。

- 当启用全服务器范围的捕获模式时，使用 bcp 将在 master 数据库中保存查询计划。如果使用大量表或视图执行 bcp，请检查 sysqueryplans 和 master 中的事务日志。

复制计划所需的时间和空间

如果您在 ap_stdout 中有大量查询计划，在开始复制前，请确保 system 段中有足够的空间来复制它们。使用 sp_spaceused 检查 sysqueryplans 的大小，使用 sp_helpsegment 检查系统段的大小。

复制计划也需要事务日志中的空间。

sp_copy_all_qplans 为将要复制的组中每个计划调用 sp_copy_qplan。如果由于空间不足或其它问题导致 sp_copy_all_qplans 在任意时刻失败，则已成功复制的任何计划仍将保留在目标查询计划组中。

存储过程的抽象计划

为使抽象计划可被捕获以用于可在存储过程中优化的 SQL 语句：

- 在启用计划捕获或计划关联模式时这些过程必须已创建。（这会将该过程的文本保存在 sysprocedures 中。）
- 执行过程时必须已启用了计划捕获模式，并且过程必须从磁盘而不是过程高速缓存中读取。

以下步骤系列将为过程中可被优化的所有语句捕获查询文本和抽象计划：

```
set plan dump dev_plans on
go
create procedure myproc as ...
go
exec myproc
go
```

如果该过程在高速缓存中，并且当前没有捕获该过程的计划，请执行 with recompile 过程。同样，在使用某一抽象查询计划执行了某个存储过程后，在过程高速缓存中该计划将被使用，因此除非从磁盘中读取该过程，否则不发生抽象计划关联。

您可以使用 set fmtonly on 来捕获存储过程的计划，而不实际执行存储过程中的语句。

过程和计划所有权

当启用计划捕获模式时，存储过程中可被优化的语句的抽象计划将被保存，同时保存过程所有者的用户 ID。

在计划关联模式期间，存储过程的关联基于过程所有者的用户 ID，而不是执行过程的用户。这意味着为某一过程创建了抽象查询计划后，所有有权执行此过程的用户都将使用同一抽象计划。

具有可变执行路径和优化的过程

执行存储过程将为每个可优化的语句保存抽象计划，即使该存储过程包含控制流语句也是如此，根据过程参数或其它条件，这些控制流语句可能会导致不同的语句运行。

Adaptive Server 在编译而不是执行存储过程时装载和保存抽象计划。

当 Adaptive Server 编译存储过程时 （通常是在首次运行存储过程时），它会为每个优化的语句保存一个抽象计划。 Adaptive Server 不影响抽象计划捕获，也不影响存储过程是否包含导致执行不同语句 （具体取决于该过程的参数）的控制流语句。

如果您通过不同的参数 （使用不同的代码路径）再次运行该查询（未经重新编译），那么由于 Adaptive Server 已在之前的编译中优化和保存了所有语句的计划，因此这些计划以及此不同代码路径中的语句的抽象计划均可用，并且基于之前的存储过程的运行参数值，无论这些语句是否已执行。

但是，根据条件或参数的不同，过程的抽象计划不解决由具有以不同方式优化的语句的过程引起的问题。例如，下面是一个用户为 between 子句提供低值和高值的过程，其中包含类似如下的查询：

```
select title_id
from titles
where price between @lo and @hi
```

根据参数的不同，最佳计划可以是索引访问，也可以是表扫描。抽象计划可指定任一访问方法，具体取决于初次执行该过程时使用的参数。如果重新启动服务器，则在执行查询或存储过程时保存在 tempdb 中的抽象计划将会丢失。

即席查询与抽象计划

抽象计划捕获可保存 SQL 查询的完整文本，并且抽象计划关联基于 SQL 查询的完整文本。如果用户提交即席 SQL 语句，而不是使用存储过程或 Embedded SQL，将为查询子句的每个不同组合保存抽象计划。这可能产生大量的抽象计划。

例如，如果用户使用 select 语句检查特定 title_id 的价格，则会为每个语句保存一个抽象计划。以下两个查询分别生成一个抽象计划：

```
select price from titles where title_id = "T19245"
select price from titles where title_id = "T40007"
```

此外，每个用户都有一个计划，即，如果多个用户检查 title_id “T40007”，则会为每个用户 ID 保存一个计划。

如果此类查询包括在存储过程中，则有两个好处：

- 只为查询保存一个抽象计划，例如：

  ```
  select price from titles where title_id =
  @title_id
  ```

- 计划是以拥有该存储过程的用户的用户 ID 保存的，并且基于过程所有者 ID 进行抽象计划关联。

使用 Embedded SQL，唯一的抽象计划用主变量保存：

```
select price from titles
where title_id = :host_var_id
```

第 14 章

使用系统过程管理抽象计划

本章介绍了用于抽象计划的系统过程的基本功能和用途。有关各个过程的详细信息，请参见 《参考手册：过程》。

管理抽象计划组

可以使用系统过程来创建、删除、重命名抽象计划组并提供有关信息。

创建组

使用 sp_add_qpgroup 可以创建和命名抽象计划组。除非使用缺省捕获组 ap_stdout，否则必须创建一个计划组才能开始捕获计划。例如，若要在一个名为 dev_plans 的组中开始保存计划，必须先创建该组，然后发出 set plan dump 命令，指定组名：

```
sp_add_qpgroup dev_plans
set plan dump dev_plans on
/*SQL queries to capture*/
```

只有系统管理员或数据库所有者才能添加抽象计划组。组创建完毕后，所有用户都可从组中转储或装载计划。

删除组

使用 sp_drop_qpgroup 可以删除抽象计划组。

下列限制适用于 sp_drop_qpgroup：

- 只有系统管理员或数据库所有者才能删除抽象计划组。
- 不能删除含有计划的组。若要从组中删除所有计划，请使用 sp_drop_all_qplans 命令，并指定组名。
- 不能删除缺省抽象计划组 ap_stdin 和 ap_stdout。

以下命令将删除 dev_plans 计划组：

```
sp_drop_qpgroup dev_plans
```

获取组信息

sp_help_qpgroup 输出有关抽象计划组的信息，或有关数据库中所有抽象计划组的信息。

如果使用 sp_help_qpgroup 时不带组名，它将输出所有抽象计划组的名称、组 ID 和每组中的计划数：

```
      sp_help_qpgroup
Query plan groups in database 'pubtune'
 Group                          GID          Plans
 ------------------------------ ----------- -----------
 ap_stdin                                 1           0
 ap_stdout                                2           2
 p_prod                                   4           0
 priv_test                                8           1
 ptest                                    3          51
 ptest2                                   7         189
```

如果使用 sp_help_qpgroup 时带组名，该报告将提供指定组中有关计划的统计信息。下例报告了关于 ptest2 组的信息：

```
sp_help_qpgroup ptest2
Query plans group 'ptest2', GID 7

 Total Rows  Total QueryPlans
 ----------- ----------------
         452              189
sysqueryplans rows consumption, number of query
plans per row count
```

```
 Rows        Plans
 ----------- -----------
           5           2
           3          68
           2         119
Query plans that use the most sysqueryplans rows
 Rows        Plan
 ----------- -----------
           5  1932533918
           5  1964534032
 Hashkeys
 -----------
         123
There is no hash key collision in this group.
```

报告单独组的信息时， sp_help_qpgroup 将报告：

- 抽象计划总数以及 sysqueryplans 表中的总行数。
- sysqueryplans 中具有多行的计划数。这些计划从具有最大行数的计划开始，以降序顺序列出。
- 有关散列键数目和散列键冲突数目的信息。在整个查询中，抽象计划通过散列算法与查询相关联。

当系统管理员或数据库所有者执行 sp_help_qpgroup 时，该过程将报告数据库中或指定组中的所有计划。当任何其他用户执行 sp_help_qpgroup 时，它只报告该用户所拥有的计划。

sp_help_qpgroup 提供了多种报告模式。它们包括：

模式	返回的信息
full	组中的行数和计划数、使用两行或多行的计划数、最长计划的行数和计划 ID、散列键数以及散列键冲突信息。这是缺省报告模式。
stats	full 报告中除散列键信息以外的所有信息。
hash	组中的行数和抽象计划数、散列键数以及散列键冲突信息。
list	组中的行数和抽象计划数，以及每个查询 / 计划对的以下信息：散列键、计划 ID、查询的前几个字符以及计划的前几个字符。
queries	组中的行数和抽象计划数，以及每个查询的以下信息：散列键、计划 ID 以及查询的前几个字符。
plans	组中的行数和抽象计划数，以及每个计划的以下信息：散列键、计划 ID 以及计划的前几个字符。
counts	组中的行数和抽象计划数，以及每个计划的以下信息：行数、字符数、散列键、计划 ID、查询的前几个字符。

下例显示 counts 模式的输出：

```
sp_help_qpgroup ptest1, counts
Query plans group 'ptest1', GID 3

 Total Rows  Total QueryPlans
 ----------- ----------------
          48               19

Query plans in this group

Rows  Chars     hashkey     id          query
----- --------- ----------- ----------- ----------------------------
    3       623  1801454852   876530156 select title from titles ...
    3       576   476063777   700529529 select au_lname, au_fname...
    3       513   444226348   652529358 select au1.au_lname, au1....
    3       470   792078608   716529586 select au_lname, au_fname...
    3       430   789259291   684529472 select au1.au_lname, au1....
    3       425  1929666826   668529415 select au_lname, au_fname...
    3       421   169283426   860530099 select title from titles ...
    3       382   571605257   524528902 select pub_name from publ...
    3       355   845230887   764529757 delete salesdetail where ...
    3       347   846937663   796529871 delete salesdetail where ...
    2       379  1400470361   732529643 update titles set price =...
```

重命名组

系统管理员或数据库所有者可使用 sp_rename_qpgroup 重命名抽象计划组。下例将把组的名称从 dev_plans 改为 prod_plans：

```
sp_rename_qpgroup dev_plans, prod_plans
```

新组名不能与现有组名相同。

查找抽象计划

使用 sp_find_qplan 可搜索查询文本和计划文本，以查找与给定模式匹配的计划。

下例查找查询中包括字符串 “from titles”的所有计划：

```
sp_find_qplan "%from titles%"
```

下例将搜索执行表扫描的所有抽象计划：

```
sp_find_qplan "%t_scan%"
```

当系统管理员或数据库所有者执行 sp_find_qplan 时，该过程将检查和报告所有用户拥有的计划。当其他用户执行该过程时，sp_find_qplan 只搜索和报告这些用户自己拥有的计划。

若要只搜索一个抽象计划组，请指定组名。下例只搜索组 test_plans，查找使用特定索引的所有计划：

```
sp_find_qplan "%i_scan title_id_ix%", test_plans
```

对于每个匹配的计划，sp_find_qplan 都将输出组 ID、计划 ID、查询文本和抽象计划文本。

管理单独抽象计划

可使用系统过程来输出单独计划的查询和文本，也可复制、删除或比较单独计划，或更改与特定查询相关联的计划。

查看计划

使用 sp_help_qplan 可报告单个抽象计划。它提供三种可指定的报告类型：brief、full 和 list。brief 报告仅输出查询和计划的前 78 个字符；使用 full 可查看整个查询和计划，或使用 list 仅显示查询和计划的前 20 个字符。

下例将输出缺省 brief 报告：

```
sp_help_qplan 588529130
gid         hashkey     id
----------- ----------- -----------
          8  1460604254   588529130
query
```

```
 ------------------------------------------------------------------
 select min(price) from titles
 plan
------------------------------------------------------------------
 (plan
    (i_scan type_price titles)
    ()
)
(prop titles
    (parallel ...
```

系统管理员或数据库所有者可使用 sp_help_qplan 报告数据库中的任何计划。其他用户只能查看他们自己拥有的计划。

sp_help_qpgroup 报告组中的所有计划。请参见第 348 页的 “获取组信息”。

将计划复制到其它组

使用 sp_copy_qplan 可将抽象计划从一个组复制到另一个现有组。下例将从当前组中将计划 ID 为 316528161 的计划复制到 prod_plans 组：

```
sp_copy_qplan 316528161, prod_plans
```

sp_copy_qplan 可检验该查询不存在于目标组中。如果可能存在冲突，sp_copy_qplan 将运行 sp_cmp_qplans 检查目标组中的计划。除了由 sp_cmp_qplans 输出的消息外， sp_copy_qplan 也将在下列情况下输出消息：

- 尝试复制的查询和计划已存在于目标组中
- 组中的其它计划有相同的用户 ID 和散列键
- 组中的其它计划有相同的散列键，但查询不同

如果存在散列键冲突，计划将被复制。如果目标组中已存在该计划或如果它将引起关联键冲突，则计划不会被复制。由 sp_copy_qplan 输出的消息包含目标组中计划的计划 ID，因此您可以使用 sp_help_qplan 来检查查询和计划。

系统管理员或数据库所有者可以复制任何抽象计划。其他用户只能复制自己拥有的计划。初始计划和组不受 sp_copy_qplan 的影响。已复制的计划被分配给一个新的计划 ID、目标组 ID 以及运行生成该计划的查询的用户 ID。

删除单独抽象计划

使用 sp_drop_qplan 可以删除单个抽象计划。下例将删除指定的计划：

```
sp_drop_qplan 588529130
```

系统管理员或数据库所有者可以删除数据库中的任何抽象计划。其他用户只能删除自己拥有的计划。

若要查找抽象计划 ID，可使用 sp_find_qplan 在查询或计划中搜索使用某种模式的计划，或使用 sp_help_qpgroup 列出组中的计划。

比较两个抽象计划

假设有两个计划 ID， sp_cmp_qplans 将比较两个抽象计划和相关联的查询。例如：

```
sp_cmp_qplans 588529130, 1932533918
```

sp_cmp_qplans 首先输出一条消息报告查询的比较结果，然后输出关于计划的第二条消息，如下所示：

- 对于两个查询，其一为：
  - The queries are the same.
  - The queries are different.
  - The queries are different but have the same hash key.
- 对于计划：
  - The query plans are the same.
  - The query plans are different.

下例将比较查询和计划都匹配的两个计划：

```
sp_cmp_qplans 411252620, 1383780087
The queries are the same.
The query plans are the same.
```

下例将比较查询匹配但计划不同两个计划：

```
sp_cmp_qplans 2091258605, 647777465
The queries are the same.
The query plans are different.
```

sp_cmp_qplans 将返回状态值，显示比较结果。

表 14-1：sp_cmp_qplans 的返回状态值

返回值	含义
0	查询文本和抽象计划相同。
+1	查询和散列键不同。
+2	查询不同但散列键相同。
+10	抽象计划不同。
100	一个或两个计划 ID 不存在。

系统管理员或数据库所有者可以比较数据库中的任何两个抽象计划。其他用户只能比较自己拥有的计划。

更改现有计划

使用 sp_set_qplan 可以更改现有计划 ID 的抽象计划，而不更改该 ID 或查询文本。 sp_set_qplan 只能用于计划文本不超过 255 个字符的情况。

```
sp_set_qplan 588529130, "(i_scan title_ix titles)"
```

系统管理员或数据库所有者可以更改任何已保存查询的抽象计划。其他用户只能修改自己拥有的计划。

当执行 sp_set_qplan 时，将不会根据查询文本检查抽象计划来确定新计划对该查询是否有效，或表和索引是否存在。若要测试计划的有效性，可执行关联的查询。

也可以使用 create plan 和 plan 子句为某一查询指定抽象计划。请参见第 306 页的 “使用 SQL 创建计划”。

管理组中的所有计划

可以使用系统过程将一个抽象计划组中的所有计划复制到另一个组，比较两个组和报告中的所有抽象计划，以及删除组中的所有抽象计划。

复制组中的所有计划

使用 sp_copy_all_qplans 可以将一个抽象计划组中的所有计划复制到另一个组。下例从 test_plans 组中将所有计划都复制到 helpful_plans 组：

```
sp_copy_all_qplans test_plans, helpful_plans
```

执行 sp_copy_all_qplans 前， helpful_plans 组必须已存在。它可以包含其它计划。

sp_copy_all_qplans 通过执行 sp_copy_qplan 复制组中的每一计划，因为 sp_copy_qplan 可能会失败，所以复制某一计划可能会因同样的原因而失败。请参见第 353 页的 “比较两个抽象计划”。

每个计划都作为独立的事务复制，复制单个计划失败不会导致 sp_copy_all_qplans 失败。如果由于某种原因 sp_copy_all_qplans 失败，并且必须重新启动，则可看到关于已成功复制计划的一组消息，说明它们已存在于目标组中。

每个复制的计划都被分配一个新的计划 ID。复制的计划具有初始用户 ID。要复制抽象计划并分配新用户 ID，必须使用 sp_export_qpgroup 和 sp_import_qpgroup。请参见第 358 页的 “导入和导出计划组”。

系统管理员或数据库所有者可以复制数据库中的任何计划。其他用户只能复制自己拥有的计划。

比较组中的所有计划

使用 sp_cmp_all_qplans 可以比较两个组中的所有抽象计划并报告下列信息：

- 两组中相同的计划数
- 具有相同关联键、不同抽象计划的计划数
- 存在于一个组而不存在于其它组中的计划数

下例将比较 ap_stdout 和 ap_stdin 中的计划：

```
sp_cmp_all_qplans ap_stdout, ap_stdin
If the two query plans groups are large, this might take some
time.
Query plans that are the same
 count
 -----------
         338
Different query plans that have the same association key
```

```
 count
 -----------
          25
Query plans present only in group 'ap_stdout' :

 count
 -----------
           0
Query plans present only in group 'ap_stdin' :

 count
 -----------
           1
```

通过额外指定报告模式参数，sp_cmp_all_qplans 可提供详细的信息，包括 ID、查询以及组中各查询的抽象计划。模式参数可用于获取所有计划的详细信息，或获取只具有特定差异类型的计划的详细信息。

表 14-2：sp_cmp_all_qplans 的报告模式

模式	报告的信息
counts	以下计划的计数：相同的计划、具有相同关联键但属于不同组的计划，以及在一个组中但不在另一个组中的计划。这是缺省报告模式。
brief	counts 所提供的信息、各个组中计划不同但关联键相同的抽象计划 ID，以及在一个组中但不在另一个组中的计划 ID。
same	所有计数以及查询和计划匹配的所有抽象计划的 ID、查询和计划。
diff	所有计数以及查询和计划不同的所有抽象计划的 ID、查询和计划。
first	所有计数以及在第一个计划组中但不在第二个计划组中的所有抽象计划的 ID、查询和计划。
second	所有计数以及在第二个计划组中但不在第一个计划组中的所有抽象计划的 ID、查询和计划。
offending	所有计数以及关联键不同或不同时存在于两个组中的所有抽象计划的 ID、查询和计划。它是 diff、 first 和 second 模式的组合。
full	所有计数以及所有抽象计划的 ID、查询和计划。它是 same 和 offending 模式的组合。

下例显示了 brief 报告模式：

```
sp_cmp_all_qplans ptest1, ptest2, brief
If the two query plans groups are large, this might take some time.
Query plans that are the same

 count
 -----------
          39
Different query plans that have the same association key

 count
 -----------
```

```
           4

    ptest1    ptest2

 id1         id2
 ----------- -----------
   764529757  1580532664
   780529814  1596532721
   796529871  1612532778
   908530270  1724533177
Query plans present only in group 'ptest1' :

 count
 -----------
           3


 id
 -----------
   524528902
  1292531638
  1308531695

Query plans present only in group 'ptest2' :

 count
 -----------
           1


 id
 -----------
  2108534545
```

删除组中的所有抽象计划

使用 sp_drop_all_qplans 可以删除组中的所有抽象计划。下例将删除 dev_plans 组中的所有抽象计划：

```
sp_drop_all_qplans dev_plans
```

在系统管理员或数据库所有者执行 sp_drop_all_qplans 时，将从指定组中删除属于所有用户的所有计划都。当其他用户执行该过程时，它只影响该用户自己拥有的计划。

导入和导出计划组

使用 sp_export_qpgroup 和 sp_import_qpgroup 可在 sysqueryplans 和用户表之间复制计划组。这将允许系统管理员或数据库所有者：

- 将抽象计划从一个数据库复制到同一服务器上的另一个数据库
- 创建一个能利用 bcp 从当前服务器复制到另一个服务器的表
- 向同一数据库中的现有计划分配不同的用户 ID

将计划导出到用户表

使用 sp_export_qpgroup 可以将特定用户的所有计划从抽象计划组复制到用户表。下例将从 fast_plans 组复制数据库所有者 (dbo) 拥有的计划，并创建一个名为 transfer 的表：

```
sp_export_qpgroup dbo, fast_plans, transfer
```

sp_export_qpgroup 使用 select...into 来创建与 sysqueryplans 具有相同列和数据类型的表。如果在数据库中没有启用 select into/bulkcopy/pllsort 选项，可指定其它数据库名称。以下命令在 tempdb 中创建导出表：

```
sp_export_qpgroup mary, ap_stdout, "tempdb..mplans"
```

使用 bcp 可将表复制到另一个服务器上的表中。这些计划也可导入到同一服务器上另一个数据库的 sysqueryplans 中，或使用不同组名或用户 ID 导入到同一数据库的 sysqueryplans 中。

从用户表导入计划

使用 sp_import_qpgroup 可将计划从由 sp_export_qpgroup 创建的表复制到 sysqueryplans 的组中。下例将计划从表 tempdb.mplans 复制到 ap_stdin 中，并为数据库所有者分配用户 ID：

```
sp_import_qpgroup "tempdb..mplans", dbo, ap_stdin
```

不能将计划复制到已包含指定用户计划的组中。

索引

英文

A

B

C

D

F

G

H

J

K

L

M

N

P

Q

S

T

W

X

Y

Z