SYBASE®

Transact-SQL® 用户指南

Adaptive Server® Enterprise

15.5

文档 ID：DC32970-01-1550-01

最后修订日期：2009 年 11 月

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

目录

关于本手册

读者

本手册为 《Transact-SQL 用户指南》，介绍 SQL 关系数据库语言的增强版 Transact-SQL®。《Transact-SQL 用户指南》适用于初学者及有其他 SQL 实践经验的人员。

如何使用本手册

不熟悉 SQL 的 Sybase® Adaptive Server® Enterprise 数据库管理系统的用户可以将本指南视为教科书，从零开始学习。刚接触 SQL 的用户应该集中精力于本手册的第一部分。第二部分讲述了高级主题。

本手册可用作熟悉其它版本 SQL 的人员的复习工具，以及用作了解 Transact-SQL 增强功能的指南。SQL 专家应该研究 Transact-SQL 已添加到标准 SQL 的功能，特别是关于存储过程的资料。

相关文档

Adaptive Server® Enterprise 文档集包括：

- 针对所用平台的发行公告 — 包含未能及时写入手册的最新信息。

  最新版本的发行公告可能已经推出。若要了解本产品 CD 发行以后增加的重要产品或文档信息，请使用 Sybase® Product Manuals 网站。

- 针对所用平台的安装指南 — 介绍所有 Adaptive Server 产品及相关 Sybase 产品的安装、升级和某些配置过程。
- New Feature Summary （《新增功能摘要》）— 介绍 Adaptive Server 中的新功能，为支持这些功能所增加的系统更改，以及可能会影响现有应用程序的更改。
- Active Messaging Users Guide（《Active Messaging 用户指南》）— 介绍如何使用 Active Messaging 功能捕获 Adaptive Server Enterprise 数据库中的事务 （数据更改），并将它们作为事件实时传递给外部应用程序。
- 《组件集成服务用户指南》— 说明如何使用组件集成服务功能来连接远程 Sybase 和非 Sybase 数据库。
- 针对所用平台的 《配置指南》— 提供执行特定配置任务的操作说明。
- 《词汇表》— 定义 Adaptive Server 文档中使用的技术术语。
- 《Historical Server 用户指南》— 介绍如何使用 Historical Server 从 Adaptive Server 获取性能信息。

- 《Adaptive Server Enterprise 中的 Java》— 介绍在 Adaptive Server 数据库中如何安装 Java 类，如何将它们用作数据类型、函数及存储过程。
- 《Job Scheduler 用户指南》— 提供有关如何使用命令行或图形用户界面 (GUI) 在本地或远程 Adaptive Server 上安装、配置、创建和调度作业的操作说明。
- 《迁移技术指南》— 介绍了迁移到不同版本 Adaptive Server 的策略和工具。
- 《Monitor Client Library 程序员指南》— 介绍如何编写访问 Adaptive Server 性能数据的 Monitor Client Library 应用程序。
- 《Monitor Server 用户指南》— 介绍如何使用 Monitor Server 从 Adaptive Server 获取性能统计信息。
- Monitoring Tables Diagram （《监控表框图》）— 以张贴画的形式阐明监控表及其实体关系。大图只在印刷版本中提供；采用 PDF 格式时提供缩略图。
- Performance and Tuning Series 《性能和调优系列》— 是一套系列丛书，介绍如何调节 Adaptive Server 以获得最优性能：
  - Basics （《基础知识》）— 包含通晓和研究 Adaptive Server 中的性能问题需具备的基础知识。
  - Improving Performance with Statistical Analysis （《利用统计分析改进性能》）— 介绍 Adaptive Server 如何存储和显示统计信息，以及如何使用 set statistics 命令分析服务器统计信息。
  - Locking and Concurrency Control （《锁定和并发控制》）— 介绍如何使用锁定方案来改进性能，以及如何选择索引以最大限度地减少并发。
  - Monitoring Adaptive Server with sp_sysmon （《使用 sp_sysmon 监控 Adaptive Server》）— 讨论如何使用 sp_sysmon 监控性能。
  - Monitoring Tables （《监控表》）— 介绍如何从 Adaptive Server 监控表中查询统计信息和诊断信息。
  - Physical Database Tuning （《物理数据库调优》）— 介绍如何管理物理数据放置、为数据分配的空间以及临时数据库。
  - Query Processing and Abstract Plans（《查询处理和抽象计划》）— 介绍优化程序如何处理查询以及如何使用抽象计划更改某些优化程序计划。

- 《快速参考指南》— 这是一本袖珍手册，完整地列出了各种命令、函数、系统过程、扩展系统过程、数据类型和实用程序的名称和语法 （该手册在以 PDF 格式阅读时采用正常大小）。
- 《参考手册》— 是一系列丛书，包含详细的 Transact-SQL® 信息：
  - 《构件块》— 讨论数据类型、函数、全局变量、表达式、标识符、通配符和保留字。
  - 《命令》— 提供了命令的文档资料。
  - 《过程》— 介绍系统过程、目录存储过程、系统扩展存储过程和 dbcc 存储过程。
  - 《表》— 讨论系统表、监控表及 dbcc 表。
- 《系统管理指南》—
  - 《卷 1》— 介绍了系统管理的基本知识，包括配置参数、资源问题、字符集和排序顺序的说明以及诊断系统问题的操作说明。《卷 1》的第二部分深入讨论了安全性管理。
  - 《卷 2》— 包括管理物理资源、镜像设备、配置内存和数据高速缓存、管理多处理器服务器和用户数据库、装入和卸下数据库、创建和使用段、使用 reorg 命令及检查数据库一致性的操作说明和指导。《卷 2》的后半部分介绍了如何备份和恢复系统数据库及用户数据库。
- System Tables Diagram （《系统表框图》）— 以张贴画的形式阐明系统表及其实体关系。大图只在印刷版本中提供；采用 PDF 格式时提供缩略图。
- 《Transact-SQL 用户指南》— 提供有关 Transact-SQL 这一 Sybase 关系数据库语言增强版的文档资料。本指南可作为数据库管理系统入门用户的教科书，还包含 pubs2 和 pubs3 示例数据库的详细说明。
- Troubleshooting: Error Messages Advanced Resolutions （《故障排除：错误消息高级解析》）— 包括您可能遇到的问题的故障排除步骤。此处讨论的问题是 Sybase 技术支持部门的员工最常听到的问题。
- 《加密列用户指南》— 介绍了如何利用 Adaptive Server 配置和使用加密列。
- 《内存数据库用户指南》— 介绍了如何配置和使用内存数据库。
- Using Adaptive Server Distributed Transaction Management Features （《使用 Adaptive Server 分布式事务管理功能》）— 介绍如何在分布式事务处理环境中配置、使用 Adaptive Server DTM 功能以及如何排除其中的故障。

- 《将 Backup Server 与 IBM® Tivoli® Storage Manager 配合使用》— 介绍如何设置和使用 IBM Tivoli Storage Manager 以创建 Adaptive Server 备份。
- 《在高可用性系统中使用 Sybase 故障切换》— 提供有关使用 Sybase 的故障切换功能将 Adaptive Server 配置为高可用性系统中的协同服务器的操作说明。
- Unified Agent and Agent Management Console （《Unified Agent 和 Agent Management Console》）— 介绍用于提供管理、监控和控制分布式 Sybase 资源的运行时服务的 Unified Agent。
- 《实用程序指南》— 提供有关在操作系统级别执行的 Adaptive Server 实用程序 （如 isql 和 bcp）的文档资料。
- 《Web 服务用户指南》— 介绍如何配置、使用 Adaptive Server Web 服务以及如何排除其中的故障。
- 《适用于 CICS、 Encina 和 TUXEDO 的 XA 接口集成指南》— 提供有关将 Sybase DTM XA 接口与 X/Open XA 事务管理器配合使用的说明。
- 《Adaptive Server Enterprise 中的 XML 服务》— 介绍了 Sybase 本机 XML 处理器、 Sybase 基于 Java 的 XML 支持及数据库中的 XML，并提供了 XML 服务中可用的查询和映射函数的文档资料。

其它信息来源

使用 Sybase Getting Started CD、SyBooks™ CD 和 Sybase Product Manuals 网站可以了解有关产品的更多信息：

- Getting Started CD 包含 PDF 格式的发行公告和安装指南，还可能包含 SyBooks CD 中未收纳的其它文档或更新信息。它随软件一起提供。若要阅读或打印 Getting Started CD 上的文档，需要使用 Adobe Acrobat Reader，该软件可以使用 CD 上提供的链接从 Adobe Web 站点免费下载。
- SyBooks CD 含有产品手册，它随软件一起提供。基于 Eclipse 的 SyBooks 浏览器使您能够以易于使用的、基于 HTML 的格式阅读手册。

  有些文档可能是以 PDF 格式提供的，您可以通过 SyBooks CD 上的 PDF 目录访问这些文档。若要阅读或打印 PDF 文件，您需要使用 Adobe Acrobat Reader。

  有关安装和启动 SyBooks 的说明，请参见 Getting Started CD 上的《SyBooks 安装指南》或 SyBooks CD 上的 *README.txt* 文件。

- Sybase Product Manuals 网站是 SyBooks CD 的联机版本，您可以使用一种标准 Web 浏览器来访问它。除了产品手册之外，还可以找到有关 EBFs/Maintenance （EBF/ 维护）、 Technical Documents （技术文档）、 Case Management （案例管理）、 Solved Cases （解决的案例）、Newsgroups（新闻组）和 Sybase Developer Network（Sybase 开发员网络）的链接。

  要访问 Sybase Product Manuals 网站，请转到位于 http://www.sybase.com/support/manuals/ 的“产品手册”(Product Manuals)。

Web 上的 Sybase 认证

Sybase 网站上的技术文档不断在更新。

❖ **查找有关产品认证的最新信息**

1. 将 Web 浏览器定位到位于 http://www.sybase.com/support/techdocs/ 的“技术文档”(Technical Documents)。
2. 单击“认证报告”(Certification Report)。
3. 在“认证报告”(Certification Report) 过滤器中选择相应的产品、平台和时间范围，然后单击“查找”(Go)。
4. 单击“认证报告”(Certification Report) 标题显示此报告。

❖ **查找有关组件认证的最新信息**

1. 将 Web 浏览器定位到位于 http://certification.sybase.com/ 的“可用性和认证报告”(Availability and Certification Reports)。
2. 在“按基本产品搜索”(Search by Base Product) 下选择产品系列和产品，或在“按平台搜索”(Search by Platform) 下选择平台和产品。
3. 选择“搜索”(Search) 以显示所选项目的可用性和认证报告。

❖ **创建 Sybase 网站（包括支持页）的个人化视图**

建立 MySybase 配置文件。 MySybase 是一项免费服务，它允许您创建 Sybase Web 页的个人化视图。

1. 将 Web 浏览器定位到位于 http://www.sybase.com/support/techdocs/ 的“技术文档”(Technical Documents)。
2. 单击“我的 Sybase”(MySybase) 并创建 MySybase 配置文件。

Sybase EBF 和软件维护

❖ **查找有关 EBF 和软件维护的最新信息**

1 将 Web 浏览器定位到位于 http://www.sybase.com/support 的“Sybase 支持页”(Sybase Support Page)。

2 选择“EBF/ 维护”(EBFs/Maintenance)。如果出现提示信息，请输入您的 MySybase 用户名和口令。

3 选择一个产品。

4 指定时间范围并单击“查找”(Go)。即会显示 EBF/ 维护版本的列表。

锁形图标表示因为您没有注册为“技术支持联系人”(Technical Support Contact)，因此您没有某些 EBF/ 维护版本的下载授权。如果您尚未注册，但拥有 Sybase 代表提供的或通过支持合同获得的有效信息，请单击“编辑角色”(Edit Roles) 将“技术支持联系人”(Technical Support Contact) 角色添加到 MySybase 配置文件中。

5 单击信息图标可显示 EBF/ 维护报告，单击产品说明可下载软件。

约定

以下各部分将说明在本手册中使用的约定。

SQL 是一种形式自由的语言。没有规定每一行中的单词数量或者必须换行的地方。然而，为便于阅读，本手册中所有示例和大多数语法语句都经过了格式设置，以便语句的每个子句都在一个新行上开始。有多个成分的子句会扩展到其它行，这些行会有缩进。复杂命令使用修正的 Backus Naur Form (BNF) 表示法进行了格式处理。

表 1 说明本手册中出现的语法语句的约定：

表 1：本手册的字体和语法约定

元素	示例
命令名、过程名、实用程序名和其它关键字用 sans serif 字体显示。	select sp_configure
数据库名和数据库类型用 sans serif 字体显示。	master 数据库
书名采用正常字体并加书名号；文件名、变量和路径名用斜体显示。	《系统管理指南》 *sql.ini* 文件 *column_name* *$SYBASE/ASE* 目录

变量 （即代表您要填充的值的词语）作为查询或语句的一部分出现时用斜体的 Courier 字体显示。	`select column_name`
	`from table_name`
	`where search_conditions`

键入小括号作为命令的一部分。	compute *row_aggregate* (*column_name*)

元素	示例
双冒号加等号表示语法是用 BNF 表示法编写的。请勿输入此符号。表示“被定义为”。	`::=`
大括号表示必须至少选择括号中的一个选项。不要输入大括号。	`{cash, check, credit}`
中括号表示可以选择括号中的一个或多个选项，也可不选。不要输入中括号。	`[cash | check | credit]`
逗号表示可以选择任意多个显示的选项。可输入逗号作为命令的一部分来分隔选项。	`cash, check, credit`
竖线 (|) 表示只可选择所显示的选项中的一个。	`cash | check | credit`
省略号 (...) 表示可以将最后一个单元*重复*任意多次。	`buy thing = price [cash | check | credit] [, thing = price [cash | check | credit]]...` 您必须至少购买一件产品，并给出其价格。可以选择一种付款方式：中括号中的选项之一。还可选择购买其它物品：可根据需要购买任意数量的物品。对于要买的每种产品，给出其名称、价格和付款方式（可选）。

- 语法语句（显示命令的语法和所有选项）显示如下：

  sp_dropdevice [*device_name*]

  对于具有多个选项的命令：

  select *column_name*
  from *table_name*
  where *search_conditions*

  在语法语句中，关键字（命令）采用常规字体，而标识符为小写。斜体表示用户提供的内容。

- 说明 Transact-SQL 命令用法的示例如下：

  ```
  select * from publishers
  ```

- 计算机输出的示例如下：

```
pub_id    pub_name                city          state
------    --------------------    -----------   -----
0736      New Age Books           Boston        MA
0877      Binnet & Hardley        Washington    DC
1389      Algodata Infosystems    Berkeley      CA

(3 rows affected)
```

本手册中的大多数示例都用小写显示。不过，输入 Transact-SQL 关键字时可以忽略大小写。例如， SELECT、 Select 和 select 之间没有区别。

Adaptive Server 是否区分数据库对象 （如表名）的大小写，取决于安装在 Adaptive Server 上的排序顺序。通过重新配置 Adaptive Server 的排序顺序，可改变单字节字符集的区分大小写设置。有关详细信息，请参见《系统管理指南》。

辅助功能特性

此文档具有针对辅助功能进行了专门设计的 HTML 版本。可以利用适应性技术 （如屏幕阅读器）浏览 HTML 文档，也可以用屏幕放大器进行查看。

Adaptive Server HTML 文档已经过测试，符合美国政府 “第 508 节辅助功能”的要求。符合 “第 508 节”的文档一般也符合非美国的辅助功能原则，如针对网站的 World Wide Web 协会 (W3C) 原则。

注释 可能需要配置辅助功能工具，以便获得最佳使用效果。某些屏幕阅读器按照大小写来辨别文本，例如将 “ALL UPPERCASE TEXT”看作首字母缩写，而将 “MixedCase Text”看作单词。对工具进行配置，规定语法约定，您可能会感觉更方便。有关工具的信息，请查阅文档。

有关 Sybase 如何支持辅助功能的信息，请参见位于 http://www.sybase.com/accessibility 的 “Sybase 辅助功能” (Sybase Accessibility)。“Sybase 辅助功能” (Sybase Accessibility) 站点包括指向 “第 508 节”和 W3C 标准相关信息的链接。

如果需要帮助

对于购买了支持合同的客户安装的每一个 Sybase 产品，都会有一位或多位指定人员获得与 Sybase 技术支持部门联系的授权。如果使用手册或联机帮助不能解决问题，可让指定人员与 Sybase 技术支持部门联系或与所在区域的 Sybase 子公司联系。

第 1 章

SQL 构件块

Adaptive Server 中的 SQL

SQL （结构化查询语言）是在关系数据库系统中使用的高级语言。SQL 最初由 IBM 的 San Jose Research Laboratory 在二十世纪七十年代末开发成功，现经改编后已用于很多关系数据库管理系统。美国国家标准协会 (ANSI) 和国际标准化组织 (ISO) 已将其批准为正式关系查询语言标准。

Transact-SQL 是 Sybase 的 SQL 扩展，它与 IBM SQL 和 SQL 的大多数其它商业实施兼容。它提供重要的额外功能和函数，如汇总计算、存储过程 （预定义的 SQL 语句）和错误处理。

SQL 不仅包括用于查询数据库 （从数据库中检索数据）的命令，也包括用于创建新数据库和**数据库对象**、添加新数据、修改现有数据的命令，还包括其它函数。

注释 如果已在 Adaptive Server 上启用 Java，可在数据库中安装和使用 Java 类。可使用标准 Transact-SQL 命令调用 Java 操作和存储 Java 类。有关此功能的完整说明，请参见 《Adaptive Server Enterprise 中的 Java》。

查询、数据修改和命令

在 SQL 中，**查询**使用 select 命令请求数据。例如，下面的查询请求列出居住在加利福尼亚州的作者：

```
select au_lname, city, state
from authors
where state = "CA"
```

数据修改指使用 insert、delete 或 update 命令对数据进行添加、删除或更改。例如：

```
insert into authors (au_lname, au_fname, au_id)
values ("Smith", "Gabriella", "999-03-2346")
```

其它 SQL 命令 （如删除表或添加用户）执行管理操作。例如：

```
drop table authors
```

每个命令或 SQL 语句都以指定要执行的基本操作的**关键字** （如 insert）开始。很多 SQL 命令还具有一个或多个**关键字短语**或**子句**，它们用于定制命令以满足特定需要。运行查询时， Transact-SQL 显示结果。如果没有数据满足查询中所指定的标准，将显示一条消息指明此情况。由于数据修改语句和管理语句不检索数据，因此，它们不显示结果。Transact-SQL 提供消息，告知您是否已执行了数据修改或其它命令。

表、列和行

在关系数据库管理系统中，用户访问和修改存储在表中的数据。SQL 是为数据库管理的关系模型而专门设计的。

表中的每行 （即每条记录）描述的是一段数据 — 人员、公司、销售或其它事项。每列 （即每个字段）描述的是数据的一个特性 — 人员姓名或地址、公司名称或总裁、产品销售量。

关系数据库由彼此相关的一组表构成。数据库经常包括很多表。

关系操作

关系系统中的基本查询操作是选择 （也称为限制）、投影和连接。这些操作可通过 SQL 的 select 命令全组合起来。

选择是表中行的子集。在 select 查询中，可指定限制条件。例如，您可能只想查看居住在加利福尼亚州的所有作者的行。

```
select *
from authors
where state = "CA"
```

投影是表中列的子集。例如，下面的查询仅显示所有作者的姓名和所在城市，而省略街道地址、电话号码和其它信息。

```
select au_fname, au_lname, city
from authors
```

连接通过比较指定字段中的值将两个或多个表中的行进行链接。例如，假设有一个包含作者信息的表，其中包括列 au_id （作者标识号）和 au_lname （作者姓氏）。第二个表包含书籍的扉页信息，其中包括列出书籍作者的 ID 号 (au_id) 的列。可连接 authors 表和 titles 表，测试各表的 au_id 列中的值是否相等。只要存在匹配项，就会创建一个新行 （包含来自两个表的列），并将其作为连接结果的一部分进行显示。连接通常与投影和选择结合使用，这样可仅显示选定匹配行的所选列。

```
select *
from authors, publishers
where authors.city = publishers.city
```

编译对象

Adaptive Server 使用**编译对象**来包含有关各数据库的重要信息，并帮助您访问和处理数据。编译对象是需要 sysprocedures 表中条目的任何对象，包括：

- 检查约束
- 缺省值
- 规则
- 存储过程
- 扩展存储过程
- 触发器
- 视图

- 函数
- 计算列
- 分区条件

编译对象是从**源文本**创建的，源文本是描述和定义编译对象的 SQL 语句。创建编译对象时， Adaptive Server：

1 对源文本进行语法分析，捕捉任何语法错误，生成分析树。

2 使分析树规范化，以创建规范树，它以二叉树格式表示用户语句。这就是编译对象。

3 将编译对象存储在 sysprocedures 表中。

4 将源文本存储在 syscomments 表中。

注释 如果要从 Adaptive Server 的 11.5 版或更早版本升级，并已从 syscomments 中删除了源文本，则必须在升级前恢复已删除的部分。为此，请执行所用平台的安装文档中的步骤。

保存或恢复源文本

在 Adaptive Server 11.5 版以及该版本之前的 SQL Server® 版本中，源文本保存在 syscomments 中，这样它就可以返回给执行 sp_helptext 的用户。因为这样做的唯一目的就是保存源文本，而用户经常从 syscomments 表中删除此源文本以节省磁盘空间，并从该公共区域删除机密信息。然而，删除源文本可能导致在将来升级 Adaptive Server 时出现问题。

如果编译对象在 syscomments 表中没有匹配的源文本，可使用下列任一方法将源文本恢复到 syscomments：

- 从备份装载源文本。
- 手工重新创建源文本。
- 重新安装创建了编译对象的应用程序。

检验和加密源文本

Adaptive Server 的 11.5 版本和更高版本能够检验是否存在源文本，并且可加密所选文本。处理源文本时，可使用如下命令：

- sp_checkresource — 检验每个编译对象的 syscomments 中是否存在源文本。
- sp_hidetext — 加密 syscomments 表中编译对象的源文本，防止随意查看。
- sp_helptext — 如果 syscomments 中存在源文本，则显示它，否则通知您缺少源文本。
- dbcc checkcatalog — 通知您缺少源文本。

命名约定

Adaptive Server 识别的字符由安装语言和缺省字符集进行部分限定。因此，服务器中的 SQL 语句和数据里所允许的字符因安装不同而变化，并且部分取决于缺省字符集中的定义。

SQL 语句必须符合精确的语法和结构规则，并可包含运算符、常量、SQL 关键字、特殊字符和**标识符**。标识符是服务器中的数据库对象，如数据库名或表名。对于 SQL 语句的某些部分，命名约定有所不同。运算符、常量、SQL 关键字和 Transact-SQL 扩展必须遵循比标识符更严格的命名限制，它们本身不能包含运算符和特殊字符。不过，可按更自由的规则来命名标识符 （服务器中包含的数据）。

下面各节介绍了可用于语句各部分的字符集。关于标识符的那一节还介绍了数据库对象的命名约定。

SQL 数据字符

SQL 数据字符集是一个较大的字符集，SQL 语言字符和标识符字符都取自于它。 Adaptive Server 的字符集中的任何字符 （包括单字节字符和多字节字符）都可用于数据值。

SQL 语言字符

SQL 关键字、 Transact-SQL 扩展和特殊字符 （如**比较运算符** > 和 <）只能由 7 位 ASCII 值 （A–Z、 a–z、 0–9）和下列 ASCII 字符表示：

表 1-1：在 SQL 中使用的 ASCII 字符

字符	说明
;	（分号）
(	（左括号）
)	（右括号）
,	（逗号）
:	（冒号）
%	（百分号）
-	（减号）
?	（问号）
’	（单引号）
"	（双引号）
+	（加号）
_	（下划线）
*	（星号）
/	（斜杠）
	（空格）
<	（小于号运算符）
>	（大于号运算符）
=	（等号运算符）
&	（与符号）
|	（竖线）
^	（脱字符）
[	（左方括号）
]	（右方括号）
@	（at 符号）
~	（否定符号）
!	（感叹号）
$	（美元符号）
#	（数字符号）
.	（句点）

标识符

数据库对象的命名约定适用于整个 Adaptive Server 软件和文档。大多数用户定义的标识符最长可达 255 字节，其它标识符最长只能为 30 字节。这两种情况下字节限制都与是否使用多字节字符无关。表 1-2 指定了不同标识符类型的字节限制。

表 1-2：对于标识符的字节限制

255 字节限制的标识符	30 字节限制的标识符
表名	游标名
列名称	服务器名称
索引名	主机名
视图名	登录名
用户定义的数据类型	密码
触发器名	主机进程标识
缺省名	应用程序名
规则名	初始语言名
约束名	字符集名
存储过程名	用户名称
变量名	组名
JAR 名称	数据库名
LWP 或动态语句名	高速缓存名
函数名	逻辑设备名
时间范围名	段名
函数名	会话名
应用程序环境名	执行类名
	引擎名
	停顿标记名

您必须声明标识符的第一个字符为 Adaptive Server 使用的字符集定义中的字母字符。也可使用 @ 符号或 _（下划线）。 @ 符号在作为标识符的第一个字符时，指示**局部变量**。

临时表的名称必须以 “#”（井号）（如果是在 tempdb 外创建的）或 “tempdb” 开始。当您创建一个名称长度需要小于 238 字节的临时表时，Adaptive Server 会为该表名添加 17 个字节的后缀以确保其唯一。如果您创建一个名称长度大于 238 字节的临时表， Adaptive Server 只使用前 238 个字节，然后添加 17 个字节的后缀。

第一个字符之后，标识符可以包含字母、数字或者字符 $、 #、 @、 _、 ¥（日元符号）或 £（英镑符号）。但不能在所命名的对象开头连用两个 @ 符号，如 “@@myobject”。此命名约定专用于**全局变量**，全局变量是系统定义的变量，由 Adaptive Server 自动更新。

在安装服务器时设置 Adaptive Server 是否区分大小写，并只能由系统管理员进行更改。若要查看服务器设置，请执行：

```
sp_helpsort
```

在不区分大小写的服务器上，标识符 MYOBJECT、 myobject 与 MyObject（以及所有其它大小写组合）被认为是相同的。只能创建这些对象之一，但可使用任何大小写组合来表示此对象。

标识符中不允许出现空格，也不能使用任何 SQL 保留的关键字。保留字列在 《Adaptive Server 参考手册》中。

可使用函数 valid_name 确定 Adaptive Server 是否可接受已创建的标识符。此函数的语法为：

(*character_expression* [, *maximum_length*]

如果您没有使用可选的第二个参数 *maximum_length*， valid_name 对于长度大于 30 个字节的标识符将返回 0。如果您使用了 *maximum_length*，且标识符的长度大于 *maximum_length* 参数， valid_name 将返回 0，该标识符无效。不过，您可以在 *maximum_length* 中使用任何不大于 255 的数字；如果使用 255， valid_length 对于长度大于 255 个字节的标识符返回 0。

例如：

```
select valid_name ("string", 255)
```

string 是要检查的标识符。如果 *string* 不是有效的标识符，则 Adaptive Server 返回 0（零）。如果 *string* 是有效的标识符， Adaptive Server 将返回一个非零数字。如果使用了非法字符或 *string* 长度超过 255 个字节， Adaptive Server 将返回 0。

多字节字符集

在多字节字符集中，有更多的字符可以用于标识符中。例如，在安装有日文系统的服务器上，可以使用以下字符类型作为标识符的第一个字符：Zenkaku 或 Hankaku Katakana、Hiragana、Kanji、Romaji、Cyrillic、Greek 或者 ASCII。

虽然在日文系统中，将 Hankaku Katakana 字符用作标识符是合法的，但建议不要在异构系统中使用它们。这些字符不能在 EUC-JIS 和 Shift-JIS 字符集之间转换。

一些 8 位的欧洲字符也是如此。例如，字符 “Œ”（OE 连字）是 Macintosh 字符集的一部分（代码点 0xCE），但 ISO 8859-1 (iso_1) 字符集中没有该字符。如果在从 Macintosh 字符集转换为 ISO 8859-1 字符集的数据中存在 “Œ”，则会导致转换错误。

如果对象标识符包含无法转换的字符，客户端就无法直接访问该对象。

分隔标识符

分隔标识符是加双引号的对象名称。使用分隔标识符可避免对象名称的某些限制。可用双引号分隔表名、视图名和列名；但不能将双引号用于其它数据库对象。

分隔标识符可以是保留字，可用非字母字符开头，并且也可以包括其它情况下不允许的字符。分隔标识符不能超过 253 个字节。任何带引号的标识符的第一个字符不能为井号 (#)。（Adaptive Server 11.5 和所有更高版本都有此限制。）

在创建或引用分隔标识符之前，必须执行：

```
set quoted_identifier on
```

这使 Adaptive Server 能够识别分隔标识符。每次在语句中使用带引号的标识符时，都必须将其括在双引号中。例如：

```
create table "1one"(col1 char(3))
select * from "1one"
create table "include spaces" (col1 int)
```

注释　当分隔标识符与系统过程一起使用时，它们不能与 bcp 一起使用，某些前端产品不支持此类用法，并有可能产生意外结果。

如果启用了 quoted_identifier 选项，则不要在字符串或日期字符串两侧加双引号，而应使用单引号。如果用双引号分隔这些字符串，则会导致 Adaptive Server 将其视作标识符。例如，若要在 *1onetable* 的 *col1* 中插入一个字符串，请使用：

```
insert "1one"(col1) values ('abc')
```

而不应使用：

```
insert "1one"(col1) values ("abc")
```

若要将单引号插入到列中，请使用两个连续的单引号。例如，若要将字符 “a’b” 插入到 col1 中，请使用：

```
insert "1one"(col1) values('a''b')
```

包括引号的语法

在 quoted_identifier 选项设置为 on 时，如果语句的语法要求带引号的字符串包含一个标识符，则不需要用双引号将标识符括起来。例如：

```
create table '1one' (c1 int)
```

引号包含在表 '1one' 的名称中：

```
select object_id('1one')

-----------------------
  896003192
```

您可以通过使用双重引号，在带引号的标识符中包括一个嵌入的双引号：

```
create table "embedded""quote" (c1 int)
```

但是，在语句语法要求对象名表示为字符串时，无需使用双重引号：

```
select object_id('embedded"quote')
```

方括号括起来的分隔标识符

Sybase 也支持用方括号括起来的标识符，其行为与用引号括起来的标识符的行为相同，不同之处在于不需要借助 set quoted_identifier on 来使用它们。

```
create table [bracketed identifier](c1 int)
```

方括号的这种改善增强了平台兼容性。

唯一性约定和限定约定

在数据库中，数据库对象的名称不必唯一。但在表中，列名与索引名必须唯一，并且同一数据库中其它对象名对各所有者而言必须唯一。Adaptive Server 中的数据库名称必须唯一。

如果创建列使所用的列名在表中不唯一，或用在同一数据库中已使用的名称创建另一数据库对象 （如表、视图或存储过程）， Adaptive Server 就会输出错误消息。

可通过添加限定表或列的其它名称来唯一标识表或列。可使用数据库名称、所有者名称和表名或视图名 （对于列）来创建唯一 ID。

例如，如果用户 “sharon” 拥有 pubs2 数据库中的 authors 表，则该表中 city 列的唯一标识符为：

```
pubs2.sharon.authors.city
```

这种命名语法也适用于其它数据库对象。可按类似方式引用任何对象：

```
pubs2.dbo.titleview

dbo.postalcoderule
```

如果 set 命令的 quoted_identifier 选项为 on，则可以用双引号括起限定对象名的各部分。对于每个需要引号的限定符，都应单独使用一对引号。例如，使用：

```
database.owner."table_name"."column_name"
```

而不能使用：

```
database.owner."table_name.column_name"
```

在 create 语句中，不是始终都允许使用完整命名语法，因为除了当前所在的数据库外，不能在其它数据库中创建视图、过程、规则、缺省值或触发器。命名约定的语法为：

```
[[database.]owner.]object_name
```

或者

```
[owner.]object_name
```

owner 的缺省值是当前用户，而 *database* 的缺省值是当前数据库。在除 create 语句外的任何 SQL 语句中引用对象时，无需用数据库名称和所有者名称限定它， Adaptive Server 首先查看您拥有的所有对象，然后查看**数据库所有者**拥有的对象。只要 Adaptive Server 具有足以识别对象的信息，就不需要键入其名称的每个元素。可以省略中间元素并用句点来表示它们的位置：

```
database..table_name
```

在上述示例中，如果使用此语法来创建表，则必须包括开始元素。如果省略开始元素，就会创建名为 ..mytable 的表。命名约定能防止在此类表上执行某些操作，如游标更新。

在同一语句中限定列名和表名时，要对每个列名和表名使用相同的命名缩写；这些命名缩写求值为字符串且必须匹配，否则，将返回错误。下面是列名具有不同条目的两个示例。第二个示例不运行，因为列名的语法与表名的语法不匹配。

```
select pubs2.dbo.publishers.city
from pubs2.dbo.publishers

city
-----------------------
Boston
Washington
Berkeley

select pubs2.sa.publishers.city
from pubs2..publishers

The column prefix "pubs2.sa.publishers" does not match
a table name or alias name used in the query.
```

远程服务器

您可以执行远程 Adaptive Server 上的存储过程。存储过程的结果显示在调用该过程的终端上。标识远程服务器和存储过程的语法为：

```
[execute] server.[database].[owner].procedure_name
```

如果远程过程调用 (RPC) 是批处理中的第一个语句，可省略关键字 execute。如果 RPC 之前有其它 SQL 语句，则必须使用 execute 或 exec。必须给出服务器名和存储过程名。如果省略数据库名称，则 Adaptive Server 会在缺省数据库中查找 procedure_name。如果给出数据库名称，除非您拥有该过程或数据库所有者拥有该过程，否则也必须给出过程所有者的名称。

下列语句执行位于 GATEWAY 服务器上的 pubs2 数据库中的存储过程 byroyalty：

语句	注释
`GATEWAY.pubs2.dbo.byroyalty` `GATEWAY.pubs2..byroyalty`	byroyalty 由数据库所有者拥有。
`GATEWAY...byroyalty`	在 pubs2 是缺省数据库时使用。
`declare @var int` `exec GATEWAY...byroyalty`	在此语句不是批处理中的第一个语句时使用。

有关将 Adaptive Server 设置为可进行远程访问的信息，请参见《系统管理指南：卷 1》中的第 15 章 “管理远程服务器”。远程服务器名称（前一示例中的 GATEWAY）必须与本地 Adaptive Server 的 *interfaces* 文件中的服务器名称相匹配。如果 *interfaces* 中的服务器名称使用大写字母，则 RPC 中也必须使用大写字母。

Adaptive Server 中的表达式

表达式由运算符分隔开的一个或多个常量、文字、函数、列标识符和变量组成，用于返回单个值。表达式可有几种类型，包括**算术表达式**、**关系表达式**、**逻辑表达式**（或**布尔表达式**）和**字符串表达式**。在某些 Transact-SQL 子句中，可在表达式中使用子查询。在表达式中可使用 case 表达式。

可在表达式中使用小括号将元素分组。将 “*expression*” 当作语法语句中的变量提供时，就视其为简单表达式。只有在逻辑表达式可接受时，才指定 “逻辑表达式”。

算术表达式和字符表达式

算术表达式和字符表达式的一般模式为：

{*constant* | *column_name* | *function* | (*subquery*)
| (case_expression)}
[{*arithmetic_operator* | *bitwise_operator* |
string_operator | *comparison_operator* }
{*constant* | *column_name* | *function* | (*subquery*)
| case_expression}]...

运算符优先级

运算符的优先级如下，其中 1 是最高级别， 6 是最低级别：

1　一元运算符 （单个参数） - + ~

2　* /%

3　二元运算符 （两个参数） + - & | ^

4　not

5　and

6　or

当表达式中的所有运算符属于同一级别时，执行顺序为从左向右。可使用小括号来更改执行顺序 — 嵌套在最里面的表达式将最先执行。

算术运算符

Adaptive Server 使用以下算术运算符：

表 1-3：算术运算符

运算符	含义
+	加法
–	减法
*	乘法
/	除法
%	模运算 （Transact-SQL 扩展）

加法、减法、除法和乘法可用于精确数值列、近似数值列和货币类型列。

模运算符 （可用在除 money 和 numeric 之外的所有精确数值列上）可求出两个数相除后得到的余数。例如，使用整数：21 % 11 = 10，因为 21 除以 11 等于 1 余 10。可以通过 numeric 或 decimal 数据类型获得非整数结果：1.2 % 0.07 = 0.01，因为 1.2 / 0.07 = 17 * 0.07 + 0.01。可以通过 float 和 real 数据类型的计算获得类似的结果：1.2e0 % 0.07 = 0.010000。

在对混合数据类型 （例如， float 和 int）执行算术运算时， Adaptive Server 将按照特定的规则来确定结果类型。有关详细信息，请参见 第 6 章 “使用和创建数据类型”。

逐位运算符

逐位运算符是用于 integer 数据类型的 Transact-SQL 扩展。这些运算符将每个整数操作数转换为二进制表示形式，然后逐列对操作数求值。值 1 对应 true ； 值 0 对应 false。

表 1-4 和表 1-5 汇总了操作数 0 和 1 的结果。如果两个操作数均为 NULL，逐位运算符将返回 NULL：

表 1-4：逐位运算真值表

& （与）	1	0
1	1	0
0	0	0

| （或）	1	0
1	1	1
0	1	0

^ （异或）	1	0
1	0	1
0	1	0

~ （非）	
1	FALSE
0	0

下列示例使用两个 tinyint 参数：A = 170 （二进制形式为 10101010）和 B = 75 （二进制形式为 01001011）。

表 1-5：逐位运算示例

运算	二进制形式	结果	解释
(A & B)	10101010 01001011 ---------- 00001010	10	如果 A 和 B 均为 1，结果列等于 1。否则，结果列等于 0。
(A | B)	10101010 01001011 ---------- 11101011	235	如果 A 或 B 任一为 1，或二者均为 1，则结果列等于 1。否则，结果列等于 0。
(A ^ B)	10101010 01001011 ---------- 11100001	225	如果 A 或 B 任一为 1，但二者不同时为 1，则结果列等于 1。
(~A)	10101010 ---------- 01010101	85	所有的 1 都变为 0，所有的 0 都变为 1。

字符串并置运算符

字符串运算符 + 可并置两个或多个字符或二进制的表达式。例如：

```
select Name = (au_lname + ", " + au_fname)
    from authors
```

在列标题 “Name”下显示作者姓名，按照先姓后名的顺序，并在姓后加一个逗号；例如，“Bennett, Abraham”。

```
select "abc" + "" + "def"
```

返回字符串 “abc def”。空字符串在所有 char、 varchar、 nchar、 nvarchar 和 text 并置中以及在 varchar 插入和赋值语句中都被解释为一个空格。

当并置非字符、非二进制表达式时，使用 convert：

```
select "The date is " +
    convert(varchar(12), getdate())
```

比较运算符

Adaptive Server 使用以下比较运算符：

表 1-6：比较运算符

运算符	含义
=	等于
>	大于
<	小于
>=	大于或等于
<=	小于或等于
<>	不等于
!=	不等于 （Transact-SQL 扩展）
!>	不大于 （Transact-SQL 扩展）
!<	不小于 （Transact-SQL 扩展）

在比较字符数据时，< 表示更接近服务器排序顺序的开头，而 > 表示更接近服务器排序顺序的结尾。在不区分大小写的排序顺序中，大写字母和小写字母相等。使用 sp_helpsort 可以查看 Adaptive Server 的排列顺序。为便于比较，尾随空白将被忽略。

在比较日期时，< 表示早于，> 表示晚于。

为所有使用比较运算符的字符，以及 date 和 time 数据加上单引号或双引号：

```
= "Bennet"
  "May 22 1947"
```

非标准运算符

下列运算符是 Transact-SQL 扩展：

- 模运算符：%
- 否定比较运算符：!>, !<, !=
- 逐位运算符：~, ^, |, &
- 连接运算符：*= 和 =*

字符表达式比较

Adaptive Server 将字符常量表达式视为 varchar。将它们与非 varchar 变量或列数据进行比较时，将在比较中使用数据类型优先级规则 （即，将较低优先级的数据类型转换为较高优先级的数据类型）。如果不支持隐式数据类型转换，则必须使用 convert 函数。有关受支持和不受支持的转换的详细信息，请参见 《参考手册：构件块》。

在比较 char 表达式和 varchar 表达式时，应按照数据类型优先规则：将“较低级”的数据类型转换为 “较高级”的数据类型。为便于比较，将所有 varchar 表达式都转换为 char （即添加尾随空白）。

空字符串

空字符串 (“”) 或 (‘’) 在 varchar 数据的 insert 或赋值语句中被解释为一个空格。当并置 varchar、 char、 nchar 或 nvarchar 数据时，空字符串也被解释为一个空格。例如，下例中的语句存储为 “abc def”：

```
"abc" + "" + "def"
```

空字符串从不会求值为 NULL。

引号

在 char 或 varchar 条目中，有两种方法可以指定文字引号。第一种方法是多使用一个同种类型的引号。这称为 “转义”引号。例如，如果在某个字符条目开头用了一个单引号，但要将一个单引号作为该条目的一部分，则可使用两个单引号：

```
'I don''t understand.'
```

下面是一个包含内部双引号和单引号的示例。单引号不必转义，但双引号必须转义：

```
"He said, ""It's not really confusing."""
```

第二种方法是给引号加上相反类型的引号。也就是说，给包含双引号的条目加上单引号 （或相反）。下面是一些示例：

```
'George said, "There must be a better way."'
"Isn't there a better way?"
'George asked, "Isn"t there a better way?"'
```

若要使在屏幕上一行末尾处换行的字符串延续，可在转入下一行之前输入反斜杠 (\)。

注释 如果 quoted_identifier 选项设置为 on，则不要用双引号括起字符或日期数据。必须使用单引号，否则 Adaptive Server 将数据视为标识符。有关带引号的标识符的详细信息，请参见第 9 页的 “分隔标识符”。

关系表达式和逻辑表达式

逻辑表达式或关系表达式返回 TRUE、 FALSE 或 UNKNOWN。一般模式为：

expression *comparison_operator* [any | all] *expression*

expression [not] in *expression*

[not] exists *expression*

expression [not] between *expression* and *expression*

expression [not] like "*match_string*" [escape "*escape_character*"]

not expression like "*match_string*" [escape "*escape_character*"]

expression is [not] null

not *logical_expression*

logical_expression {and | or} *logical_expression*

any、 *all* 和 *in*

any 与 <、> 或 = 以及子查询一起使用。如果子查询中检索到的任何值都与外层语句的 where 或 having 子句中的值相匹配，它就会返回结果。 all 与 < 或 > 以及子查询一起使用。当在子查询中检索到的所有值都小于 (<) 或大于 (>) 外层语句的 where 或 having 子句中的值时，它就会返回结果。有关详细信息，请参见第 5 章 “子查询：在其它查询中使用查询”。

当第二个表达式返回的任何值与第一个表达式中的值相匹配时， in 就会返回结果。第二个表达式必须是用小括号括起来的子查询或值列表。 in 等效于 = any。

and 和 *or*

and 连接两个表达式，并在二者均为真时返回结果。 or 连接两个或多个条件，并在任一条件为真时返回结果。

如果在一个语句中使用了多个逻辑运算符， and 将先于 or 求值。使用小括号可更改执行顺序。

表 1-7 显示了逻辑运算的结果，其中包括那些含空值的表达式：

表 1-7：逻辑表达式真值表

and	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
NULL	UNKNOWN	FALSE	UNKNOWN
or	**TRUE**	**FALSE**	**NULL**
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
NULL	TRUE	UNKNOWN	UNKNOWN
not			
TRUE	FALSE		
FALSE	TRUE		
NULL	UNKNOWN		

结果 UNKNOWN 表示一个或多个表达式求值为 NULL，并且不能确定运算结果是 TRUE 还是 FALSE。

Transact-SQL 扩展

Transact-SQL 改进了 SQL 的功能，并将用户必须求助于编程语言来完成所需任务的场合减少到最小。Transact-SQL 的功能超出了 ISO 标准以及 SQL 的很多商业版本。

将大部分 Transact-SQL 改进 （也称为扩展）功能汇总如下。《参考手册：命令》中提供了各命令的 Transact-SQL 扩展。

compute 子句

Transact-SQL compute 子句扩展与行集合函数（sum、 max、 min、 avg、 count 和 count_big）一起使用来计算汇总值。包含了 compute 子句的查询在显示结果中带有明细行和汇总行。这些报告几乎与任何具有报告生成器的数据库管理系统 (DBMS) 生成的报告类似。 compute 在结果中将汇总值显示为附加行，而不是作为新列显示。有关 compute 子句的说明，请参见第 3 章 “使用集合、分组和排序”。

控制流语言

Transact-SQL 提供控制流语言，它可用作任何 SQL 语句或批处理的一部分。它提供以下结构：begin...end、 break、 continue、 declare、 goto label、 if...else、 print、 raiserror、 return、 waitfor 和 while。局部变量可用 declare 和所赋的值定义。系统提供了很多预定义的全局变量。

Transact-SQL 也支持 case 表达式，这种表达式包括关键字 case、 when、 then、 coalesce 和 nullif。 case 表达式代替标准 SQL 的 if 语句。在使用了值表达式的任何地方都可以使用 case 表达式。

存储过程

最重要的 Transact-SQL 扩展之一是能够创建存储过程。**存储过程**是按某一名称存储的 SQL 语句和可选的控制流语句的集合。存储过程的创建者也可定义在执行存储过程时所要提供的参数。

能够编写自己的存储过程极大地提高了 SQL 数据库语言的功能、效率和灵活性。由于执行计划在运行存储过程后保存，因此在以后运行存储过程时，要比独立语句快得多。

Adaptive Server 提供的存储过程称为**系统过程**，帮助 Adaptive Server 进行系统管理。第 17 章 “使用存储过程”论述了系统过程并说明如何创建存储过程。《参考手册：过程》中详细论述了系统过程。

您可在远程服务器上执行存储过程。所有 Transact-SQL 扩展都支持存储过程的返回值、用户定义的存储过程返回状态以及将参数从过程传递到其调用方的功能。

扩展存储过程

扩展存储过程 (ESP) 具有存储过程的接口，但它不包含 SQL 语句和控制流语句，而是执行已编译成动态链接库 (DLL) 的过程语言代码。

编写 ESP 函数所使用的过程语言可以是能调用 C 语言函数并能处理 C 数据类型的任何语言。

ESP 允许 Adaptive Server 在关系数据库管理系统 (RDBMS) 外执行任务，以响应在数据库内部发生的事件。例如，可使用 ESP 发送电子邮件通知或整个网络范围的广播，以响应关系数据库管理系统 (RDBMS) 内发生的事件。

有一些由 Adaptive Server 提供的 ESP，称为**系统扩展存储过程**。xp_cmdshell 是其中之一，它允许从 Adaptive Server 内执行操作系统命令。第 18 章 “使用扩展存储过程”中对 ESP 进行了说明。请参见《参考手册：过程》中的第 3 章 “系统扩展存储过程”。

ESP 由名为 XP Server™ 的 Open Server™ 应用程序实现，该应用程序与 Adaptive Server 运行在同一台计算机上。远程执行存储过程称为远程过程调用 (RPC)。 Adaptive Server 和 XP Server 通过 RPC 进行通信。安装 Adaptive Server 时自动安装 XP Server。

触发器

触发器是一个存储过程，在尝试进行特定更改时它指示系统采取一项或多项操作。触发器能防止对数据进行错误的、未授权的或不一致的更改，有助于保持数据库完整性。

触发器也可保护参照完整性：强制使用不同表中数据之间的关系规则。用户尝试用 insert、 delete 或 update 命令修改数据时，触发器就会起作用。

触发器最深可嵌套 16 层，并可调用本地或远程存储过程或其它触发器。

缺省值和规则

Transact-SQL 提供维护实体完整性 （确保为要求值的每列都提供值）和域完整性 （确保列中的每个值都属于该列的合法值集）的关键字。缺省值和规则定义了在数据输入和修改过程中起作用的完整性约束。

缺省值是链接到特定列或数据类型的值，如果输入数据时未提供任何值，则由系统插入缺省值。规则是用户定义的、链接到特定列或数据类型的完整性约束，在输入数据时强制执行这些规则。有关规则和缺省值的论述，请参见第 14 章 “为数据定义缺省值和规则”。

错误处理和 *set* 选项

为 Transact-SQL 程序员提供了许多错误处理技术，包括捕获存储过程的返回状态、定义存储过程的自定义返回值、从过程向其调用方传递参数以及从全局变量 （如 *@@error*）获得报告等功能。与控制流语言组合使用的 raiserror 和 print 语句可向 Transact-SQL 应用程序的用户提供错误消息。开发人员可本地化 print 和 raiserror，以便使用不同语言。

set 选项用于在调试 Transact-SQL 程序时自定义结果显示、显示处理统计信息和提供其它诊断帮助。除 showplan 和 char_convert 外，所有 set 选项都立即生效。

下面几个段落列出了可用的 set 选项。有关详细信息，请参见 《参考手册：命令》。

- parseonly、 noexec、 prefetch、 showplan、 rowcount、 nocount 和 tablecount 控制查询的执行方式。 statistics 选项可在每次查询后显示性能统计信息。 flushmessage 用于确定 Adaptive Server 何时将消息返回给用户。请参见 《实用程序指南》中的第 2 章 “从命令行使用交互式 isql”。
- arithabort 用于确定在发生算术溢出和数值截断错误时 Adaptive Server 是否中止查询。 arithignore 用于确定在查询导致算术溢出时 Adaptive Server 是否输出警告消息。有关详细信息，请参见第 27 页的 “算术错误”。
- offsets 和 procid 用于 DB-Library™，以解释来自 Adaptive Server 的结果。
- datefirst、 dateformat 和 language 影响日期函数、日期顺序和消息显示。
- char_convert 控制 Adaptive Server 与客户端间的字符集转换。

- textsize 控制着用 select 语句返回的 text 或 image 数据的大小。请参见第 505 页的 “用于 text、 unitext 和 image 数据的文本函数”。
- cursor rows 和 close on endtran 影响 Adaptive Server 处理游标的方式。请参见第 614 页的 “使用游标读取数据行”。
- identity_insert 可允许或禁止影响表的标识列的插入。请参见第 240 页的 “由于其它原因产生的间隔”。
- chained 和 transaction isolation level 控制 Adaptive Server 处理事务的方式。请参见第 695 页的 “选择事务模式和隔离级别”。
- self_recursion 允许 Adaptive Server 处理引起自行触发的触发器。请参见第 662 页的 “触发器自递归”。
- ansinull、ansi_permissions 和 fipsflagger 控制 Adaptive Server 是否对非标准 SQL 的使用进行标记。 string_rtruncation 控制在截断 char 或 nchar 字符串时 Adaptive Server 是否引发例外错误。请参见第 25 页的 “符合 ANSI 标准”。
- quoted_identifier 控制 Adaptive Server 是否将加双引号的字符串当作标识符。有关详细信息，请参见第 9 页的 “分隔标识符”。
- role 控制授予用户的角色。有关角色的信息，请参见 《系统管理指南：卷 1》中的第 13 章 “Adaptive Server 中的安全性管理快速入门”。

SQL 的其它 Adaptive Server 扩展

Transact-SQL 的其它独特的或与众不同的功能包括：

- 以下是 SQL 搜索条件的扩展：模运算符 (%)、否定比较运算符（!>、 !< 和 !=）、逐位运算符 （–、 ^、 | 和 &）、连接运算符 （*= 和 =*）、通配符 （[ ] 和 -）以及 not 运算符 (^)。请参见第 2 章 “查询：从表中选择数据”。
- 对 group by 子句和 order by 子句的限制较少。请参见第 3 章 “使用集合、分组和排序”。
- 子查询，它几乎可用在允许使用表达式的任何位置。请参见第 5 章 “子查询：在其它查询中使用查询”。
- 临时表和其它临时数据库对象，它们仅在当前工作会话期间存在，之后消失。请参见第 7 章 “创建数据库和表”。

- 建立在 Adaptive Server 所提供的数据类型基础上的、用户定义的数据类型。请参见第 6 章 “使用和创建数据类型”和第 14 章 “为数据定义缺省值和规则”。
- 可以使用 insert 将表中数据插入到同一表中。请参见第 8 章 “添加、更改、传输和删除数据”。
- 可以从一个表中提取数据，然后用 update 命令将其放到另一表中。请参见第 8 章 “添加、更改、传输和删除数据”。
- 可以在 delete 语句中使用连接来删除基于其它表中数据的数据。请参见第 8 章 “添加、更改、传输和删除数据”。
- 一种快速方式，它用 truncate table 命令删除指定表中所有行并回收这些行所占用的空间。请参见第 8 章 “添加、更改、传输和删除数据”。
- 标识列，它提供系统生成的、能唯一标识表中各行的值。请参见第 8 章 “添加、更改、传输和删除数据”。
- 通过视图进行更新和选择。与 SQL 的大多数其它版本不同，Transact-SQL 对通过视图检索数据没有任何限制，对通过视图更新数据的限制也较少。请参见第 12 章 “视图：限制访问数据”。
- 提供数十种内置函数。请参见第 16 章 “在查询中使用内置函数”。
- 为 create index 命令提供一些选项，可用来调优由索引所确定的性能的各个方面，并可控制重复键和重复行的处理。请参见第 13 章 “创建表的索引”。
- 控制当用户尝试在唯一索引中输入重复键或在表中输入重复行时所发生的操作。请参见第 13 章 “创建表的索引”。
- 用于 integer 和 bit 类型的列的逐位运算符。请参见第 44 页的 “逐位运算符”和第 6 章 “使用和创建数据类型”。
- 支持 text 与 image 数据类型。请参见第 6 章 “使用和创建数据类型”。
- 可访问 Sybase 和非 Sybase 数据库。使用组件集成服务，可在远程异构服务器中的各表之间实现下列类型的操作：如同本地表一样访问远程表，执行连接，在表之间传送数据，维护参照完整性，使应用程序 （如 PowerBuilder®）能够透明地访问异构数据，以及使用本机远程服务器的功能。有关详细信息，请参见 《组件集成服务用户指南》。

符合 ANSI 标准

关系数据库管理系统的标准仍在不断发展中。这些标准已被或正被 ISO 和若干国家 / 地区的标准化组织所采用。 SQL86 是这些标准中的第一个标准。该标准已被 SQL89 代替，而 SQL89 又被 ANSI SQL 所代替。ANSI SQL 是当前的标准，它定义了一致性的三个级别：初级、中间级和完整级别。在美国，国家标准技术研究所 (NIST) 定义了过渡级别，它介于初级和中间级之间。

由这些标准定义的某些行为与现有的 SQL Server 和 Adaptive Server 应用程序不兼容。 Transact-SQL 提供了 set 选项，可用来切换这些行为。

缺省情况下，会为所有 Embedded SQL™ 预编译应用程序启用一致行为。需要与 SQL 标准行为一致的其它应用程序可使用表 1-8 中的选项来符合初级 ANSI SQL。有关设置这些选项的详细信息，请参见 《参考手册：命令》中的 set。

表 1-8：为符合初级 ANSI SQL 设置命令标志

选项	设置
ansi_permissions	on
ansinull	on
arithabort	off
arithabort numeric_truncation	on
arithignore	off
chained	on
close on endtran	on
fipsflagger	on
quoted_identifier	on
string_rtruncation	on
transaction isolation level	3

下列各节介绍了标准行为与缺省的 Transact-SQL 行为之间的区别。

美国联邦信息处理标准 (*FIPS*) *标志程序*

为使客户编写出必须符合 ANSI SQL 标准的应用程序， Adaptive Server 提供了 set fipsflagger 选项。启用此选项后，包含在初级 ANSI SQL 中不允许使用的 Transact-SQL 扩展的所有命令都生成一条信息性消息。此选项不会禁用这些扩展。发出非 ANSI SQL 命令时，处理完成。

链式事务和隔离级别

Adaptive Server 提供符合 SQL 标准的 “链式”事务行为作为一个选项。在链式模式中，所有的数据检索命令和修改命令 （delete、 insert、 open、 fetch、 select 和 update）都隐式开始一个**事务**。由于这种行为与很多 Transact-SQL 应用程序不兼容，因此 Transact-SQL 类型 （或 “非链式”）的事务保持缺省值。

可使用 set chained 选项启动链式事务模式。set transaction isolation level 选项用于控制事务隔离级别。有关详细信息，请参见第 22 章 “事务：维护数据一致性和恢复”。

标识符

为符合初级 ANSI SQL，标识符不能：

- 以井号 (#) 开头
- 超过 18 个字符
- 包含小写字母

Adaptive Server 支持在表名、视图名和列名中使用分隔标识符。分隔标识符是用双引号括起来的对象名称。使用它们可避免对象名称的某些限制。

使用 set quoted_identifier 选项识别分隔标识符。此选项设置为 “打开”时，双引号中括起来的所有字符都被视为标识符。由于此行为与很多现有应用程序不兼容，所以此选项的缺省设置为 “关闭”。

SQL 标准样式的注释

在 Transact-SQL 中，注释由 “/*”和 “*/”分隔，并可以嵌套。Transact-SQL 也支持 SQL 标准样式注释，它们由两个连在一起的减号开头的任何字符串、注释和终止性换行符组成：

```
select "hello" -- this is a comment
```

完全支持 Transact-SQL “/*”和 “*/”注释分隔符，但无法识别 Transact-SQL 注释中的 “--”。

右截断字符串

string_rtruncation set 选项控制无提示地截断字符串以符合 SQL 标准。将此选项设置为 “打开”，以禁止无提示地截断并强制执行 SQL 标准行为。

update 和 *delete* 语句所要求的权限

ansi_permissions set 选项用于确定 delete 和 update 语句所要求的权限。此选项设置为 “打开”时， Adaptive Server 会对这些语句应用更严格的 ANSI SQL 权限要求。由于此行为与很多现有应用程序不兼容，所以此选项的缺省设置为 “关闭”。

算术错误

arithabort 和 arithignore set 选项按如下所述来符合 ANSI SQL 标准：

- arithabort arith_overflow 指定出现除零错误或精度损失后的行为。缺省设置为 arithabort arith_overflow on，它将回退发生错误的整个事务。如果不包含事务的批处理发生了这种错误，则 arithabort arith_overflow on 将不回退批处理中以前的命令，但 Adaptive Server 也不会执行批处理中产生错误的语句之后的语句。

  如果设置了 arithabort arith_overflow off，Adaptive Server 将中止导致错误的语句，但会继续处理事务或批处理中的其它语句。

- arithabort numeric_truncation 用于指定在精确数值类型导致标度损失后的行为。缺省设置为 “打开”，它将中止导致错误的语句，但会继续处理事务或批处理中的其它语句。如果设置了 arithabort numeric_truncation off，Adaptive Server 就会截断查询结果并继续进行处理。为符合 ANSI SQL 标准，输入 set arithabort numeric_truncation on。

- arithignore arith_overflow 确定 Adaptive Server 是否在出现除零错误或精度损失后显示消息。缺省设置为 “关闭”，在出现这些错误后显示警告消息。如果设置 arithignore arith_overflow on，则会取消发生这些错误后的警告消息。为符合 ANSI SQL 标准，请输入 set arithignore off。

  注释　有关在 JDBC 代码中处理警告的信息，请参见 jConnect Programmers Reference Guide 《jConnect 程序员参考指南》。该信息位于 “Programming Information”（编程信息）一章的 “Handling error messages”（处理错误消息）一节中。查找主题 “Handling numeric overflows that are returned as warnings from ASE”（处理 ASE 中作为警告返回的数值溢出）。

同义关键字

为符合 SQL 标准而添加的若干关键字与现有 Transact-SQL 关键字同义。

表 1-9：与 ANSI 兼容的关键字同义词

当前语法	附加语法
commit tran、 commit transaction、 rollback tran、 rollback transaction	commit work、 rollback work
any	some
grant all	grant all privileges
revoke all	revoke all privileges
max (*expression*)	max ([all | distinct]) *expression*
min (*expression*)	min ([all | distinct]) *expression*
user_name() *内置函数*	user *关键字*

空值处理

set 选项 ansinull 用于确定在 SQL 的等于 (=) 或不等于 (!=) 比较计算和集合函数中，空值操作数的求值是否符合 SQL 标准。此选项不影响 Adaptive Server 对其它类型 SQL 语句 （如 create table）中的空值的求值方式。

Adaptive Server 登录帐号

每个 Adaptive Server 用户都必须具有用唯一登录名和口令所标识的登录帐号。此帐号由系统安全员建立。登录帐号具有如下特点：

- 登录名，它在该服务器上是唯一的。
- 口令。
- 缺省数据库 （可选）。如果定义了缺省数据库，用户可在定义的数据库中启动每个 Adaptive Server 会话，而无须发出 use 命令。如果未定义缺省数据库，则要在 master 数据库中启动每个会话。
- 缺省语言 （可选）。它指定显示提示和消息所使用的语言。如果未定义语言，则使用安装时设置的 Adaptive Server 的缺省语言。
- 全名 （可选）。这是用户的全名，对文档记录和身份识别非常有用。

组成员资格

在 Adaptive Server 中，可使用组在数据库中一次授予或撤消多个用户的权限。例如，如果在销售部门工作的每个员工都需要访问某些表，则可将所有这些用户放入一个称为 “销售”的组中。数据库所有者可将特定访问权限授予该组，而不必将权限分别授予每个用户。

在数据库内创建组，而不是在整个服务器上创建组。数据库所有者负责创建组并为其指派用户。您始终是 “public”组的成员，该组包括 Adaptive Server 上的所有用户。您也可以是另一个组的成员。可使用 sp_helpuser 确定用户所在的组：

```
sp_helpuser user_name
```

角色成员资格

在 Adaptive Server 中，系统安全员可定义和创建角色，作为一次向多个用户授予或撤消服务器范围内权限的简便方法。例如，文秘人员可能需要在几个数据库中的表上进行插入和选择，但他们可能不需要更新这些表。系统安全员可定义一个称为 “clerical_user_role”的角色，并将此角色授予每个文秘人员。然后，数据库对象所有者可向 “clerical_user_role”授予所需的特权。

角色可在角色层次中进行定义，其中某角色 （如 “office_manager_role”）包含了 “clerical_user_role”。在某层次中被授予角色的用户自动拥有该层次中更低角色的所有权限。例如，办公室主管可执行文秘人员允许执行的所有操作。层次可包括系统角色或用户定义的角色。

若要确定有关指派给您的角色的详细信息，请使用：

- sp_displayroles — 确定指派给您的所有角色，无论它们是否处于激活状态。
- sp_activeroles — 确定指派给您的哪些角色处于激活状态。如果指定了 expand_down 参数， Adaptive Server 可显示当前活动角色中所包含的所有角色。

语法为：

```
sp_displayroles user_name
sp_activeroles expand_down
```

有关角色的详细信息，请参见 《系统管理指南：卷 1》中的第 13 章 “Adaptive Server 中的安全性管理快速入门”。

有关您的 Adaptive Server 帐户的信息

通过使用以下项可获取有关自己的 Adaptive Server 登录帐户的信息：

```
sp_displaylogin
```

Adaptive Server 返回以下信息：

- 您的服务器用户 ID
- 登录名
- 您的全名
- 授予您的任何角色 （无论它们当前是否处于激活状态）
- 帐户是否已被锁定
- 上一次更改口令的日期

口令更改

最好定期更改口令。系统安全员可将 Adaptive Server 配置成按预置的一定间隔要求您更改口令。如果您的服务器属于这种情况，到了应该更改口令的时间时， Adaptive Server 会通知您。

注释 如果使用远程服务器，则必须在本地服务器上更改口令*之前*，在所访问的所有远程服务器上更改口令。有关详细信息，请参见第 32 页的 “远程服务器上的口令更改”。

可随时使用 sp_password 更改口令：

```
sp_password old_passwd, new_passwd
```

创建新口令时：

- 其长度至少必须为六字节。
- 可以使用任何可打印字母、数字或符号。
- 口令的最大长度为 30 字节。如果口令超过 30 个字节， Adaptive Server 仅使用前 30 个字符。

指定口令时，如果属于以下情况，则需要加引号：

- 口令中包含 A–Z、a–z、0–9、 _、#、有效的单字节或多字节字母字符、变音字母字符以外的任何字符。
- 口令以数字 0–9 开头。

以下示例说明如何将口令 “terrible2”更改为 “3blindmice”：

```
sp_password terrible2, "3blindmice"
```

返回状态 0 表示口令已更改。有关 sp_password 的详细信息，请参见《参考手册：过程》。

远程登录名

如果获得了访问远程服务器及该服务器上相应数据库的权限，可使用 RPC 在远程 Adaptive Server 上执行存储过程。远程执行存储过程称为**远程过程调用 (RPC)**。

访问远程服务器：

- 本地服务器必须知道远程服务器。系统安全员执行 sp_addserver 时会发生这种情况。
- 必须在远程服务器上获取登录名。系统管理员执行 sp_addremotelogin 时会发生这种情况。
- 必须将您作为用户添加到远程服务器上的相应数据库中。数据库所有者执行 sp_adduser 时会发生这种情况。

《参考手册：过程》中论述了这些过程。

当能够访问远程服务器时，就可通过用远程服务器名称限定存储过程名称来执行 RPC。例如，若要在 GATEWAY 服务器上执行 sp_help，请输入：

```
GATEWAY...sp_help
```

或者，要完全限定过程名称，包括含有过程及其所有者名称的数据库名称，请输入：

```
GATEWAY.sybsystemprocs.dbo.sp_help
```

另外，如果系统安全员已将本地和远程服务器设置成可使用基于网络的安全服务，则在执行 RPC 时，下面的一个或多个功能可能会起作用：

- 相互鉴定 — 本地服务器通过检索远程服务器的认证并用安全性机制对其进行检验来鉴定远程服务器。通过这种服务，两台服务器的认证都得以鉴定和检验。
- 通过加密获得消息保密性 — 消息在发送给远程服务器时被加密，从远程服务器传来的结果也被加密。
- 消息完整性 — 对服务器之间的消息进行检查以查看数据是否被篡改。

如果使用的是基于网络的安全性的统一登录功能，则系统安全员可在远程服务器上使用 sp_remoteoption，将您确定为不需要向远程服务器提供口令的可信赖用户。如果使用的是 Open Client™ Client-Library™/C，则可用 ct_remote_pwd 指定用于远程服务器的口令。

请参见 《系统管理指南：卷 1》中的第 16 章 “外部鉴定”。

远程服务器上的口令更改

在本地服务器上更改口令*之前*，必须先在所访问的所有远程服务器上更改口令。如果先在本地服务器上更改口令，当发出 RPC 在远程服务器上执行 sp_password 时，此命令就会失败，因为本地口令与远程口令不匹配。

在远程服务器上更改口令的语法为：

remote_server...sp_password *old_passwd*, *new_passwd*

例如：

```
GATEWAY...sp_password terrible2, "3blindmice"
```

有关在本地服务器上更改口令的信息，请查看第 30 页的 “口令更改”。

isql 实用程序

通过独立的实用程序 isql，可直接从操作系统使用 Transact-SQL。

必须先在 Adaptive Server 中设置帐户或登录名。若要使用 isql，请在操作系统提示符处键入如下命令：

```
isql -Uhoratio -Ptallahassee -Shaze -w300
```

其中：

- “horatio”是用户
- “tallahassee”是口令
- “haze”是要连接到的 Adaptive Server 的名称

-w 参数表示 isql 输出的宽度为 300 个字符。登录名和口令区分大小写。

如果启动 isql 时没有使用登录名，则 Adaptive Server 会假定登录名与操作系统名称相同。有关为 isql 指定服务器登录名和其它参数的详细信息，请参见针对所用平台的 《实用程序指南》。

注释　访问 isql 时，不要在命令行上使用 -P 选项。而要等待出现 isql 口令提示，以避免其它用户看到您的口令。

启动 isql 后，可看到：

```
1>
```

此时可以开始发出 Transact-SQL 命令。

若要使用组件集成服务连接非 Sybase 数据库，可用 connect to 命令。有关详细信息，请参见 《组件集成服务用户指南》。另请参见 《参考手册：命令》中的 connect to...disconnect。

缺省数据库

创建 Adaptive Server 帐号后，就可能已指派了登录时要连接的缺省数据库。例如，缺省数据库可能是样本数据库 pubs2。如果未指派缺省数据库，则会连接到 **master 数据库**。

可将缺省数据库更改为具有使用权限的任何数据库，或更改为允许 guest 用户的任何数据库。具有 Adaptive Server 登录名的任何用户都可以是 guest。若要更改缺省数据库，请使用 sp_modifylogin，《参考手册：过程》中对其进行了说明。

若要更改为 pubs2 数据库 （本手册中多数示例都使用的是该数据库），请输入：

```
1> use pubs2
2> go
```

输入单词 “go”，它独占一行，前面不要有空白或制表符。它是命令终结符；告知 Adaptive Server 您已完成了键入操作，并且做好了执行命令的准备。

通常，本手册中演示的 Transact-SQL 语句示例不包括 isql 实用程序使用的行提示符，也不包括终结符 go。

使用 *isql* 的基于网络的安全服务

有关 Interactive SQL 以及实用程序 dbisq 的信息，请参见 《实用程序指南》。

可指定 isql 的 -V 选项来使用基于网络的安全服务，如统一登录。使用统一登录时，可用第三方提供商提供的安全性机制对您进行鉴定，然后登录到 Adaptive Server，而无须指定登录名或口令。其它可使用的安全服务 （如果第三方安全性机制支持它们）包括：

- 数据保密性
- 数据完整性
- 相互鉴定
- 数据源检查
- 数据回放检测
- 顺序混乱检测

有关为使用基于网络的安全性而可以指定的选项的详细信息，请参见 《系统管理指南：卷 1》中的第 16 章 “外部鉴定”。

isql 注销

可随时键入以下命令注销 isql：

```
quit
```

或者

```
exit
```

其中的任何一个命令都会使您返回到操作系统提示符处，并且不需要终结符 “go”。

有关 isql 的详细信息，请参见 《实用程序指南》。

显示 SQL 文本

set show_sqltext 可用于为即席查询、存储过程、游标和动态预准备语句输出 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
```

一旦启用 set show_sqltext， 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过程，此输出可能非常详尽。

若要禁用 show_sqltext，请输入：

```
set show_sqltext off
```

对 show_sqltext 的限制

- 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。

pubs2 和 *pubs3* 样本数据库

本手册中的多数示例使用的是 pubs2 样本数据库。在已特别注明的地方，使用的是 pubs3 数据库。可在自己的工作站上尝试使用任何示例。

在屏幕上看到的查询结果可能与其本手册中的结果不完全一致。这是由于为清晰起见或减少页面占用空间，某些示例已被重新设定格式 （例如，重新调整了列）。

若要使用 create 或数据修改语句更改样本数据库，可能需要从系统管理员处获得其它权限。如果确实更改了样本数据库， Sybase 建议为了便于将来用户的使用，请将其返回到原始状态。如果在恢复样本数据库时需要帮助，可向系统管理员求助。

样本数据库内容

样本数据库 pubs2 包括以下表：publishers、 authors、 titles、 titleauthor、 roysched、sales、salesdetail、stores、discounts、au_pix 和 blurbs。样本数据库 pubs3 增加了 store_employees，但不包括 au_pix。 pubs3 是 pubs2 的更新版本，可用于参照完整性示例。它的表与 pubs2 中定义的表稍有不同。

下面是各表的简要说明：

- publishers 包含三家出版公司的标识号、名称、所在城市和州。
- authors 包含每个作者的标识号、姓名、地址信息和合同状态。
- titles 包含每本书的书籍 ID、书名、类型、出版者 ID、价格、预付款、版税、年销售额、注释和出版日期。
- titleauthor 将 titles 和 authors 表链接在一起。它包含每本书的书目 ID、作者 ID、作者顺序和作者间的版税分配。
- roysched 列出单元销售范围和每个范围有关的版税。版税是来自销售额的净收入的若干百分比。
- sales 记录书店 ID、订单号和售书日期。它是 salesdetail 中明细行的主表。
- salesdetail 记录 titles 表中各书目的书店销售额。
- stores 按书店 ID 列出书店。
- store_employees 列出 stores 表中各书店的雇员。
- discounts 列出书店的三种折扣。
- au_pix 包含使用 image 数据类型以二进制形式表示的作者照片。只有 pubs2 中有 au_pix。
- blurbs 包含使用 text 数据类型对书籍进行的较长说明。

有关 pubs2 数据库的说明，请参见附录 A “pubs2 数据库”有关 pubs3 的说明，请参见附录 B “pubs3 数据库”。

第 2 章

查询：从表中选择数据

select 命令使用一个称作**查询**的过程检索数据库表的行和列中存储的数据。查询有三个主要部分：select 子句、 from 子句和 where 子句。

本章重点讨论基本的单表 select 语句。许多章节都包含有示例语句，可用来练习编写查询。如果还希望集成其它 Transact-SQL 功能（如连接、子查询和集合等），可在本书后面部分找到更具综合性的查询示例。

什么是查询？

SQL 查询用来从数据库中请求数据并接收结果。此进程 （又称为**数据检索**）用 select 语句来表达。可将其用于在一个或多个表中检索 rows 子集的**选择**，也可将其用于在一个或多个表中检索 columns 子集的**投影**。

以下是有关 select 语句的一个简单示例：

```
select select_list
from table_list
where search_conditions
```

select 子句指定要检索的列。 from 子句指定要搜索的表。 where 子句指定要看到表中的哪些行。例如，以下 select 语句将从 pubs2 数据库的 authors 表中查找居住在 Oakland 的作者的姓名。

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

此查询的结果将以分栏格式显示：

```
au_fname        au_lname
--------------  -----------
Marjorie        Green
Dick            Straight
Dirk            Stringer
Stearns         MacFeather
Livia           Karsen

(5 rows affected)
```

select 语法

select 语法可能比前一示例中所示的更简单或更复杂。一个简单的 select 语句只包含 select 子句；一般情况下都包括 from 子句，但在需要从表中检索数据的 select 语句中则必须要包括该子句。所有其它子句 （包括 where 子句）都是可选的。

《参考手册：命令》中介绍了 select 语句的完整语法。

TOP *unsigned_integer* 是一个旨在限制结果集中行数的选项，利用它可以指定要查看的行数。有关该功能的信息和示例，请参见 《参考手册：命令》。 TOP 还可用于 delete 和 update 命令中以实现相同的目的。

在 select 语句中要按上述顺序使用这些子句。例如，如果语句中包括 group by 子句和 order by 子句，group by 子句必须位于 order by 子句之前。

如果引用的对象有歧义，则需要限定数据库对象的名称。如果在多个表中有多个列名为 “name”，此时就必须用数据库名、所有者名或表名来限定 “name”。例如：

```
select au_lname from pubs2.dbo.authors
```

因为本章中的示例仅涉及单表查询，所以语法模型和示例中的列名通常没有用它们所属的表、所有者或数据库的名称来限定。为增强可读性，这些元素都被省略了；但包含限定符也并不错。本章剩余部分详细分析了 select 语句的语法。

本章仅说明了 select 命令的语法所包括的某些子句和关键字。下列子句在其它章节中讨论：

- 有关 group by、 having、 order by 和 compute 的说明，请参见第 3 章“使用集合、分组和排序”。
- 有关 into 的说明，请参见第 7 章“创建数据库和表”。
- 有关 at isolation 的说明，请参见第 22 章“事务：维护数据一致性和恢复”。

有关 holdlock、noholdlock 和 shared 关键字（用于在 Adaptive Server 中处理锁定）以及 index 子句的说明，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》）中的第 4 章“Using Locking Commands”（使用锁定命令）。有关 for read only 和 for update 子句的信息，请参见 《参考手册：命令》中的 declare cursor 命令。

注释　for browse 子句仅用于 DB-Library 应用程序中。有关详细信息，请参见 《Open Client DB-Library/C 参考手册》。另请参见第 632 页的“用浏览模式代替游标”。

检查 *select* 语句中的标识符

当存储过程或触发器的源文本存储在系统表 syscomments 中时，使用 select * 的查询将存储在 syscomments 中，同时扩展在 select * 中引用的列列表。

例如，包含列 col1 和 col2 的表中的 select * 将存储为：

```
select <table>.col1, <table>.col2 from <table>
```

列列表检查标识符 （表名、列名等）是否符合标识符的规则。

例如，如果一个表包括列 col1 和 2col，第二个列名以数字开头，该列名只能用括号括起来才能用在 create table 语句中。

当在存储过程或触发器中对此表执行 select * 时， syscomments 中的文本与以下内容类似：

```
select <table>.col1, <table>[2col] from <table>
```

对于用于扩展 select * 的文本中的所有标识符，当标识符不符合标识符规则时，将会添加括号。

必须用括号括起标识符，以确保 Adaptive Server 在升级至更新版本时可以使用该 SQL 文本。

使用 select 子句选择列

select 子句中的各项构成了选择列表。如果选择列表包含列名、列组或通配符 (*)，数据将按它们在表中存储的顺序 （即 create table 顺序）被检索出来。

使用 *select* * 选择所有列

星号 (*) 从 from 子句指定的所有表中选择所有列名。如果希望看到表中的所有列，用星号可节省键入时间并减少错误的发生。* 将按 create table 的顺序检索数据。

选择表中所有列的语法为：

```
select *
from table_list
```

以下语句将从 publishers 表中检索所有列，并按 create table 的顺序显示这些列。该语句将检索所有行，因为它不包含 where 子句：

```
select *
from publishers
```

结果如下所示：

```
pub_id pub_name               city          state
------   --------------       ---------     -----
0736     New Age Books        Boston        WA
0877     Binnet & Hardley     Washington    DC
1389     Algodata Infosystems Berkeley      CA

(3 rows affected)
```

如果在 select 关键字后按顺序列出表中的所有列名，将得到完全相同的结果：

```
select pub_id, pub_name, city, state
from publishers
```

在一个查询中也可多次使用星号 “*”：

```
select *, *
from publishers
```

此查询将每列的列名和每段列数据都显示两次。像列名一样，也可用表名限定星号。例如：

```
select publishers.*
from publishers
```

但是，因为 select * 将查找表中当前的所有列，所以对表结构的更改（如添加、删除或重命名列）将自动修改 select * 的结果。分别列出各列可对结果进行更精确的控制。

选择特定列

若要只选择表中的特定列，请使用：

```
select column_name[, column_name]...
    from table_name
```

用逗号分隔列名，例如：

```
select au_lname, au_fname
from authors
```

重新安排列顺序

在 select 子句中列出列名的顺序将决定列的显示顺序。下列示例将显示如何指定列顺序，显示 publishers 表的全部三行中的出版社名称和标识号。第一个示例先输出 pub_id，再输出 pub_name ；第二个示例与此输出顺序相反。信息一样，但顺序不同。

```
select pub_id, pub_name
from publishers

pub_id   pub_name
-----    ---------------
0736     New Age Books
0877     Binnet & Hardley
1389     Algodata Infosystems

(3 rows affected)

select pub_name, pub_id
from publishers

pub_name                       pub_id
---------------------          ------
New Age Books                  0736
Binnet & Hardley               0877
Algodata Infosystems           1389

(3 rows affected)
```

在查询结果中重命名列

显示查询结果时，每列的缺省标题是在创建列时给定的名称。出于显示目的，可通过下列方法之一重命名列标题，而不只是选择列表中的列名称。

column_heading = *column_name*

column_name column_heading

column_name as *column_heading*

这将为列提供一个替代名称。例如，若要在前一个查询中将 pub_name 更改为 “Publisher”，可键入下列任一语句：

```
select Publisher = pub_name, pub_id from publishers
select pub_name Publisher, pub_id from publishers
select pub_name as Publisher, pub_id from publishers
```

这些语句的结果如下所示：

```
Publisher                 pub_id
----------------------    ------
New Age Books             0736
Binnet & Hardley          0877
Algodata Infosystems      1389

(3 rows affected)
```

使用表达式

select 语句也可包括一个或多个**表达式**来操纵检索的数据。

```
select expression [, expression]...
   from table_list
```

表达式可以是常量、列名、函数、子查询或 case 表达式间的任意组合，它们由算术或逐位运算符和小括号连接。

如果列表中的任何表或列名不符合有效标识符的规则，可将 quoted_identifier 选项设置为打开，然后用双引号将此标识符括起来。

列标题中带引号的字符串

如果用引号将整个标题括起来，则可在列标题中包括任何字符，甚至可以包括空白。不需要将 quoted_identifier 选项设置为打开。如果列标题没有用引号括起，则它必须符合标识符规则。以下两个查询会产生同样的结果：

```
select "Publisher's Name" = pub_name from publishers
select pub_name "Publisher's Name" from publishers

Publisher's Name
----------------
New Age Books
Binnet & Hardley
Algodata Infosystems

(3 rows affected)
```

带引号的列标题中还可使用 Transact-SQL 保留字。例如，以下查询使用保留字 sum 作为列标题，它也是有效的：

```
select "sum" = sum(total_sales) from titles
```

带引号的列标题的长度不能大于 255 个字节。

注释　在 create table、 alter table、 select into 或 create view 语句中将列名用引号括起来之前，必须 set quoted_identifier on。

查询结果中的字符串

到目前为止，在所看到的 select 语句生成的结果中，显示的都是数据库中的数据。也可编写查询，以使结果中包含字符串。

用单引号或双引号将要包括的字符串括起来，然后用逗号将其与选择列表中的其它元素分隔开。如果字符串中含有撇号，则要使用双引号，否则它会被解释为单引号。

下面是一个带字符串的查询：

```
select "The publisher's name is", Publisher = pub_name
from publishers

                                        Publisher
-------------------------       --------------------
The publisher's name is         New Age Books
The publisher's name is         Binnet & Hardley
The publisher's name is         Algodata Infosystems

(3 rows affected)
```

选择列表中计算的值

可用来自数值列的数据或来自选择列表的数值常量数据进行计算。

逐位运算符

逐位运算符是可用于整数的 Transact-SQL 扩展。这些运算符将每个整数操作数转换为其二进制表示形式，然后逐列对操作数求值。值 1 对应 true ； 值 0 对应 false。

表 2-1 显示逐位运算符。

表 2-1：逐位运算符

运算符	含义
&	逐位与 （两个操作数）
|	逐位或 （两个操作数）
^	逐位异或 （两个操作数）
~	逐位非 （一个操作数）

请参见 《参考手册：构件块》中的第 4 章 “表达式、标识符和通配符”。

算术运算符

表 2-2 显示了可用的算术运算符。

表 2-2：算术运算符

运算符	运算
+	加法
-	减法
/	除法
*	乘法
%	模

除模运算符外，可在任何数值列 （bigint、 int、 smallint、 tinyint、 unsigned bigint、 unsigned int、 unsigned smallint、 numeric、 decimal、 float 或 money）使用任何算术运算符。

模运算符 （可对所有整数类型列使用）可求出两个数相除后得到的余数。例如，使用整数：21 % 11 = 10，因为 21 除以 11 等于 1 余 10。可以通过 numeric 或 decimal 数据类型获得非整数结果：1.2 % 0.07 = 0.01，因为 1.2 / 0.07 = 17 * 0.07 + 0.01。可以通过 float 和 real 数据类型的计算获得类似的结果：1.2e0 % 0.07 = 0.010000。

使用日期函数可对 date/time 列执行某些算术运算。有关信息，请参见第 16 章 “在查询中使用内置函数”。可在选择列表中对列名和数值常量的任意组合使用所有这些运算符。例如，若要查看 titles 表中计划销售增长为 100% 的所有书籍情况，请输入：

```
select title_id, total_sales, total_sales * 2
from titles
```

下面是执行结果：

```
title_id     total_sales
--------     -----------       ---------
BU1032              4095            8190
BU1111              3876            7752
BU2075             18722           37444
BU7832              4095            8190
MC2222              2032            4064
MC3021             22246           44492
MC3026              NULL            NULL
PC1035              8780           17560
PC8888              4095            8190
PC9999              NULL            NULL
PS1372               375             750
PS2091              2045            4090
PS2106               111             222
PS3333              4072            8144
PS7777              3336            6672
TC3218               375             750
TC4203             15096           30192
TC7777              4095            8190

(18 rows affected)
```

请注意 total_sales 列和计算列中的空值。空值没有显式指定的值。对空值所执行的任何算术运算，其结果都为 NULL。例如，若要赋予计算列一个标题 “proj_sales”，请输入：

```
select title_id, total_sales,
     proj_sales = total_sales * 2
from titles
```

```
title_id     total_sales     proj_sales
---------    -----------     -----------
BU1032              4095            8190
 ....
```

试着将字符串 （如 “Current sales =”和 “Projected sales are”）添加到 select 语句中。用来生成计算列的列不必出现在选择列表中。例如，在下列示例查询中显示的 total_sales 列只是用来将其列值与来自 total_sales * 2 列的值进行比较。若要只看到计算列，请输入：

```
select title_id, total_sales * 2
from titles
```

如果没有相关常量，算术运算符也可直接用于指定列的数据值。例如：

```
select title_id, total_sales * price
from titles
```

```
title_id
--------     ----------
BU1032        81,859.05
BU1111        46,318.20
BU2075        55,978.78
BU7832        81,859.05
MC2222        40,619.68
MC3021        66,515.54
MC3026             NULL
PC1035       201,501.00
PC8888        81,900.00
PC9999             NULL
PS1372         8,096.25
PS2091        22,392.75
PS2106           777.00
PS3333        81,399.28
PS7777        26,654.64
TC3218         7,856.25
TC4203       180,397.20
TC7777        61,384.05

(18 rows affected)
```

计算列也可来自多个表。在本手册的连接和子查询章节中包含了有关多表查询的信息。

下面是一个有关连接的示例，该查询将某一销路售出的心理学书籍的数量 salesdetail 表中的 qty 列）与书的价格 （titles 表中的 price 列）相乘。

```
select salesdetail.title_id, stor_id, qty * price
from titles, salesdetail
where titles.title_id = salesdetail.title_id
and titles.title_id = "PS2106"

title_id         stor_id
---------------- -----------  ------
PS2106           8042         210.00
PS2106           8042         350.00
PS2106           8042         217.00

(3 rows affected)
```

算术运算符优先级

如果表达式中包含多个算术运算符，则先计算乘法、除法和模运算，再计算加法和减法。如果表达式中的所有算术运算符都属于同一优先级级别，则执行顺序为从左向右。小括号中的表达式的优先级高于任何其它运算。

例如，以下 select 语句将书的销售总数乘以其价格来计算总销售额，然后减去作者预付款的一半。

```
select title_id, total_sales * price - advance / 2
from titles
```

首先计算 total_sales 和 price 的乘积，因为运算符是乘法。接下来，将把预付款除以 2，然后将结果从 total_sales * price 中减去。

为避免误会，可使用小括号。以下查询与前一查询具有同样的含义，其结果也一样，但它更易于理解：

```
select title_id,(total_sales * price) - (advance /2)
from titles

title_id
--------     ----------
BU1032        79,359.05
BU1111        43,818.20
BU2075        50,916.28
BU7832        79,359.05
MC2222        40,619.68
MC3021        59,015.54
MC3026             NULL
PC1035       198,001.00
PC8888        77,900.00
PC9999             NULL
PS1372         4,596.25
```

```
PS2091        1,255.25
PS2106       -2,223.00
PS3333       80,399.28
PS7777       24,654.64
TC3218        4,356.25
TC4203      178,397.20
TC7777       57,384.05

(18 rows affected)
```

使用小括号可以更改执行顺序；小括号里的计算会首先处理。如果嵌套使用小括号，则最内层嵌套的计算的优先级最高。例如，如果使用小括号在执行除法之前强制执行减法运算，则上述示例的结果和含义都会更改：

```
select title_id, (total_sales * price - advance) /2
from titles

title_id
--------   ----------------------
BU1032                  38,429.53
BU1111                  20,659.10
BU2075                  22,926.89
BU7832                  38,429.53
MC2222                  20,309.84
MC3021                  25,757.77
MC3026                       NULL
PC1035                  97,250.50
PC8888                  36,950.00
PC9999                       NULL
PS1372                     548.13
PS2091                  10,058.88
PS2106                  -2,611.50
PS3333                  39,699.64
PS7777                  11,327.32
TC3218                     428.13
TC4203                  88,198.60
TC7777                  26,692.03

(18 rows affected)
```

选择 *text*、*unitext*、*image* 值

text、unitext 和 image 的值可能非常大。如果选择列表包括 text、unitext 和 image 和值，则对返回数据长度的限制取决于全局变量 *@@textsize* 的设置。*@@textsize* 的缺省设置取决于用来访问 Adaptive Server 的软件；isql 的缺省值是 32K。若要更改此值，请使用 set 命令：

```
set textsize 25
```

利用此 *@@textsize* 设置，包括 text 列的 select 语句将只显示数据的前 25 个字节。

注释　如果选择 image 数据，返回值将包括字符 “0x”，它表示数据是十六进制数据。这两个字符将被视为 *@@textsize* 的一部分。

若要将 *@@textsize* 重新设置为 Adaptive Server 缺省值，请使用：

```
set textsize 0
```

当数据长度小于 textsize 时，缺省显示是数据的实际长度。有关 text、unitext 和 image 数据类型的详细信息，请参见第 6 章 “使用和创建数据类型”。

使用 *readtext*

readtext 命令提供了一种检索 text、unitext 和 image 值的方法，可以只检索某列数据的选定部分。readtext 需要表和列的名称、文本指针、列中的起始偏移量以及要检索的字符或字节数。以下示例将在 blurbs 表的 copy 列中查找六个字符：

```
declare @val binary(16)
select @val = textptr(copy) from blurbs
where au_id = "648-92-1872"
readtext blurbs.copy @val 2 6 using chars
```

在此示例中，声明了 *@val* 局部变量后，readtext 将显示 copy 列的第 3 到第 8 个字符，因为偏移量是 2。

Adaptive Server 将把可能会非常大的 text、unitext 和 image 数据存储在一个特殊结构中，而不是存储在表中。同时将分配一个指向数据实际存储页的文本指针 (textptr)。使用 readtext 检索数据时，实际上检索的是 textptr，它是一个 16 字节的 varbinary 字符串。若要避免此类情况，可声明一个局部变量来容纳 textptr，然后使用带 readtext 的变量，如上述示例所示。

有关 readtext 命令的深入论述，请参见第 505 页的 “用于 text、unitext 和 image 数据的文本函数”。

选择列表摘要

select 列表可以包括 *（以 create table 顺序排列的所有列）、以任意顺序排列的列名列表、字符串、列标题和包括算术运算符的表达式。也可包括集合函数，相关论述详见第 3 章 “使用集合、分组和排序”。下面是一些 select 列表，它们是用样本数据库 pubs2 中的表进行试验的：

```
select titles.*
from titles

select Name = au_fname, Surname = au_lname
from authors

select Sales = total_sales * price,
ToAuthor = advance,
ToPublisher = (total_sales * price) - advance
from titles

select "Social security #", au_id
from authors

select this_year = advance, next_year = advance
   + advance/10, third_year = advance/2,
    "for book title #", title_id
from titles

select "Total income is",
Revenue = price * total_sales,
"for", Book# = title_id
from titles
```

利用 *distinct* 消除重复查询结果

可选的 distinct 关键字将从 select 语句的缺省结果中消除重复行。

为了与其它 SQL 实现兼容，Adaptive Server 语法允许使用 all 来显式请求所有行。 all 是 select 语句的缺省值。如果未指定 distinct，缺省情况下，将会得到包括重复值的所有行。

例如，下面是未使用 distinct 时， titleauthor 表中所有作者标识代码的搜索结果：

```
select au_id
from titleauthor

au_id
-----------
172-32-1176
213-46-8915
213-46-8915
238-95-7766
267-41-2394
267-41-2394
274-80-9391
409-56-7008
427-17-2319
472-27-2349
486-29-1786
486-29-1786
648-92-1872
672-71-3249
712-45-1867
722-51-5454
724-80-9391
724-80-9391
756-30-7391
807-91-6654
846-92-7186
899-46-2035
899-46-2035
998-72-3567
998-72-3567

(25 rows affected)
```

其中列出一些重复内容。使用 distinct 可消除它们。

```
select distinct au_id
from titleauthor

au_id
-----------
172-32-1176
213-46-8915
238-95-7766
267-41-2394
274-80-9391
```

```
409-56-7008
427-17-2319
472-27-2349
486-29-1786
648-92-1872
672-71-3249
712-45-1867
722-51-5454
724-80-9391
756-30-7391
807-91-6654
846-92-7186
899-46-2035
998-72-3567

(19 rows affected)
```

distinct 将多个空值视为重复值。换言之，当 select 语句中包含 distinct 时，不论遇到多少空值，都只返回一个 NULL。

使用 order by 子句时， distinct 可以返回多个值。有关详细信息，请参见第 102 页的 “与 select distinct 一起使用的 order by 和 group by”。

利用 *from* 子句指定表

在涉及表或视图数据的所有 select 语句中都需要有 from 子句。用它来列出包含在选择列表和 where 子句中包括的列的所有表和视图。如果 from 子句包括多个表或视图，可用逗号分隔它们。

一个查询最多可以引用 50 个表和 46 个工作表（如由集合函数创建的那些表）。 50 个表的限制包括：

- from 子句中列出的表 （或表上的视图）。
- 对同一个表的多个引用的每个实例 （自连接）
- 在子查询中引用的表
- 用 into 创建的表
- from 子句中列出的视图所引用的基表

from 语法如下所示：

```
select select_list
    [from [[database.]owner.]{table_name |view_name}
            [holdlock | noholdlock] [shared]
        [, [[database.]owner.]{table_name | view_name}
            [holdlock | noholdlock] [shared]]...]
```

表名的长度可在 1 到 255 个字节之间。可以使用字母、@、# 或 _ 作为第一个字符。随后的字符可以是数字、字母或者 @、#、$、_、¥ 或 £。临时表的名称必须以 “#”（井号）（如果是在 tempdb 外创建的）或 “tempdb..” 开始。临时表的名称长度不可多于 238 个字节，因为 Adaptive Server 将把一个 17 字节的内部数字后缀附加到其后，以确保该名称是唯一的。有关详细信息，请参见第 7 章 “创建数据库和表”。

在 from 子句中始终允许表和视图的完整命名语法：

```
database.owner.table_name
    database.owner.view_name
```

但是，仅当名称发生混淆时才有必要使用完整命名语法。

可以赋给表名相关名，以节省键入时间。通过在表名之后提供相关名，从而在 from 子句中分配相关名，如下所示：

```
select p.pub_id, p.pub_name
from publishers p
```

对该表的所有其它引用 （例如在 where 子句中）必须也使用该相关名。相关名不能以数字开头。

使用 *where* 子句选择行

select 语句中的 where 子句指定搜索条件，用来确定检索哪些行。一般格式为：

```
select select_list
    from table_list
    where search_conditions
```

where 子句中的搜索条件或限定包括：

- 比较运算符 （=、 <、 > 等等）

  ```
  where advance * 2 > total_sales * price
  ```

- 范围 （between 和 not between）

  ```
  where total_sales between 4095 and 12000
  ```

- 列表 （in、 not in）

```
where state in ("CA", "IN", "MD")
```

- 字符匹配 （like 和 not like）

```
where phone not like "415%"
```

- 未知值 （is null 和 is not null）

```
where advance is null
```

- 搜索条件的组合 （and、 or）

```
where advance < 5000 or total_sales between 2000
  and 2500
```

where 关键字还可引入：

- 连接条件 （请参见第 4 章 “连接：从若干表中检索数据”）
- 子查询 （请参见第 5 章 “子查询：在其它查询中使用查询”）

注释 可以在 text 列使用的唯一 where 条件是 like （或 not like）。

有关搜索条件的详细信息，请参见 《参考手册：命令》。

比较运算符

Transact-SQL 使用下列比较运算符：

表 2-3：比较运算符

运算符	含义
=	等于
>	大于
<	小于
>=	大于或等于
<=	小于或等于
<>	不等于
!=	不等于 （Transact-SQL 扩展）
!>	不大于 （Transact-SQL 扩展）
!<	不小于 （Transact-SQL 扩展）

这些运算符用在以下语法中：

where expression comparison_operator expression

expression 是常量、列名、函数、子查询、 case 表达式或其任意组合，它们由算术运算符或逐位运算符连接。在比较字符数据时， < 表示在排序顺序中位于当前字符之前， > 表示在排序顺序中位于当前字符之后。使用 sp_helpsort 可以显示 Adaptive Server 的排序顺序。

出于比较目的，尾随空白将被忽略。例如，“Dirk”与 “Dirk ”是相同的。在比较日期时， < 表示早于， > 表示晚于。用撇号或引号括起所有的 char、 nchar、 unichar、 unitext、 varchar、 nvarchar、 univarchar、 text 和 date/time 数据。有关输入 date/time 型数据的详细信息，请参见第 8 章 “添加、更改、传输和删除数据”。

下面是一些使用比较运算符的 select 语句的示例：

```
select *
from titleauthor
where royaltyper < 50

select authors.au_lname, authors.au_fname
from authors
where au_lname > "McBadden"

select au_id, phone
from authors
where phone != "415 658-9932"

select title_id, newprice = price * $1.15
from  pubs2..titles
where advance > 5000
```

not 否定一个表达式。下列两个查询均查找预付款小于 $5500 的所有商业和心理学方面的书籍。请注意否定逻辑运算符 (not) 和否定比较运算符 (!>) 在位置上的区别。

```
select title_id, type, advance
from titles
where (type = "business" or type = "psychology")
and not advance >5500

select title_id, type, advance
from titles
where (type = "business" or type = "psychology")
and advance !>5500

title_id  type            advance
--------  ------------    --------
BU1032    business        5,000.00
BU1111    business        5,000.00
BU7832    business        5,000.00
PS2091    psychology      2,275.00
PS3333    psychology      2,000.00
```

```
PS7777    psychology       4,000.00

(6 rows affected)
```

范围 （*between* 和 *not between*）

使用 between 可以关键字指定包括的范围。

例如，若要查找销量在 4095 和 12,000 （包括 4095 和 12,000）之间的所有书籍，查询的编写方式如下：

```
select title_id, total_sales
from titles
where total_sales between 4095 and 12000

title_id  total_sales
------    -----------
BU1032           4095
BU7832           4095
PC1035           8780
PC8888           4095
TC7777           4095

(5 rows affected)
```

可用大于 (>) 和小于 (<) 运算符指定一个排除的范围：

```
select title_id, total_sales
from titles
where total_sales > 4095 and total_sales < 12000

title_id  total_sales
------    -----------
PC1035           8780

(1 row affected)
```

not between 将查找指定范围之外的所有行。若要查找销售额在 $4095 到 $12,000 范围之外的所有书籍，请键入：

```
select title_id, total_sales
from titles
where total_sales not between 4095 and 12000

title_id     total_sales
--------     -----------
BU1111              3876
BU2075             18722
```

```
MC2222              2032
MC3021             22246
PS1372               375
PS2091              2045
PS2106               111
PS3333              4072
PS7777              3336
TC3218               375
TC4203             15096

(11 rows affected)
```

列表 （*in* 和 *not in*）

in 关键字允许选择与任一组值相匹配的值。表达式可以是常量或列名，值列表可以是一组常量或一个子查询。

例如，若要列出居住在 California、 Indiana 或 Maryland 的所有作者的姓名及其所在州名，可使用：

```
select au_lname, state
from authors
where state = "CA" or state = "IN" or state = "MD"
```

或者，如果希望用较少的键盘敲击次数而得到同样的结果，可使用 in。用逗号将 in 关键字后的各项分隔开，并用小括号将其括起来。用单引号或双引号将 char、 varchar、 unichar、 unitext、 univarchar 和 datetime 值括起来。例如：

```
select au_lname, state
from authors
where state in ("CA", "IN", "MD")
```

下面是来自两个查询的结果：

```
 au_lname         state
 -----------      -----
 White            CA
 Green            CA
 Carson           CA
 O'Leary          CA
 Straight         CA
 Bennet           CA
 Dull             CA
 Gringlesby       CA
 Locksley         CA
 Yokomoto         CA
```

```
DeFrance        IN
Stringer        CA
MacFeather      CA
Karsen          CA
Panteley        MD
Hunter          CA
McBadden        CA

(17 rows affected)
```

in 关键字的最重要用途或许是用在嵌套查询（又称为**子查询**）中。有关子查询的完整论述，请参见第 5 章 “子查询：在其它查询中使用查询”。下例提供了一种思路，介绍用嵌套查询和 in 关键字可实现哪些功能。

假设希望知道对合著书籍收到的总版税低于 50% 的作者的姓名。authors 表给出了作者姓名， titleauthor 表给出了版税信息。通过使用 in 将两个表放到一起（但并不在同一个 from 子句中将这两个表列出），即可提取所需信息。以下查询：

- 对 titleauthor 表进行搜索，以找出从任一本书得到的版税少于 50% 的所有作者的 au_id。
- 从 authors 表中选择 au_id 与 titleauthor 的查询结果相匹配的所有作者的姓名。结果显示有多个作者满足上述版税少于 50% 的分类条件。

```
select au_lname, au_fname
from authors
where au_id in
(select au_id
   from titleauthor
   where royaltyper < 50)

au_lname          au_fname
--------------    ------------
Green             Marjorie
O'Leary           Michael
Gringlesby        Burt
Yokomoto          Akiko
MacFeather        Stearns
Ringer            Anne

(6 rows affected)
```

not in 将会查找与列表中的各项不匹配的作者。以下查询将查找作者姓名，他们至少从一本书中得到的版税不少于 50%。

```
select au_lname, au_fname
from authors
```

```
where au_id not in
(select au_id
   from titleauthor
   where royaltyper < 50)

au_lname          au_fname
---------------   ------------
White             Johnson
Carson            Cheryl
Straight          Dick
Smith             Meander
Bennet            Abraham
Dull              Ann
Locksley          Chastity
Greene            Morningstar
Blotchet-Halls    Reginald
del Castillo      Innes
DeFrance          Michel
Stringer          Dirk
Karsen            Livia
Panteley          Sylvia
Hunter            Sheryl
McBadden          Heather
Ringer            Albert
Smith             Gabriella

(18 rows affected)
```

匹配模式

可以在 where 子句中使用通配符，从而可以根据共同特征来搜索未知字符或对数据进行分组。以下各部分利用 SQL 和 Transact-SQL 对模式匹配进行说明。有关模式匹配的详细信息，请参见 《参考手册：命令》。

匹配字符串：*like*

like 关键字搜索与某一模式相匹配的字符串。 like 可用于 char、 varchar、 nchar、 nvarchar、 unichar、 unitext、 univarchar binary、 varbinary、 text 和 date/time 类型的数据。

like 的语法为：

{where | having} [not]
column_name [not] like "*match_string*"

match_string 可包含表 2-4 中的符号：

表 2-4：匹配字符串的特殊符号

符号	含义
%	与 0 个或多个字符的任意字符串相匹配。
_	与单个字符相匹配。

[*specifier*]	中括号将范围或集合括起来，如 [a – f] 或 [abcdef]。*specifier* 可采用两种形式：
	• *rangespec1-rangespec2*：
	  • *rangespec1* 表示字符范围的开始
	  • – 是一个特殊字符，它表示一个范围。
	  • *rangespec2* 表示字符范围的结束。
	• *set*：可由任意值的离散集合组成，可以按任何顺序排列，如 [a2bR]。范围 [a – f] 和集合 [abcdef] 以及 [fcbdae] 返回同样的一组值。
	分类符区分大小写。

[^*specifier*]	尖号 (^) 位于分类符之前，表示不包括。 [^a – f] 指 “不在 a–f 的范围内”； [^a2bR] 指 “不是 a、 2、 b 或 R”。

可将列数据与常量、变量或其它包含**通配符** （如表 2-4 所示）的列相匹配。使用常量时，要用引号将匹配字符串括起来。例如，可以对 authors 表中的数据使用 like：

- like “Mc%”将搜索每个以 “Mc”开始的名称 （如 McBadden）。
- like “%inger”将搜索每个以 “inger”结尾的名称 （如 Ringer、Stringer）。
- like “%en%”将搜索每个包含 “en”的名称 （如 Bennet、 Green、McBadden）。
- like “_heryl”将搜索每个以 “heryl”结尾且长度为六个字母的名称 （如 Cheryl）。
- like “[CK]ars[eo]n”将搜索 “Carsen”、“Karsen”、“Carson”和 “Karson” (Carson)。
- like “[M-Z]inger”将搜索所有以 “inger”结尾且以 M 到 Z 之间任意单个字母开始的名称 （如 Ringer）。
- like “M[^c]%”将搜索所有以 “M”开始且第二个字母不是 “c”的名称。

以下查询将在 authors 表中查找区号为 415 的所有电话号码：

```
select phone
from authors
where phone like "415%"
```

可以在 text 列上使用的唯一 where 条件是 like。此查询将在 blurbs 表中查找 copy 列中包含单词 “computer” 的所有行：

```
select * from blurbs
where copy like "%computer%"
```

Adaptive Server 将不带 like 的通配符解释为文字，而不将其解释为一种模式；它们只代表其本身的值。下列查询尝试查找仅由四个字符 “415%” 组成的任何电话号码。它不会查找以 415 开始的电话号码。

```
select phone
from authors
where phone = "415%"
```

将 like 用于 datetime 值时， Adaptive Server 将把这些值转换为标准的 datetime 格式，然后转换为 varchar 或 univarchar。由于标准存储格式不包括秒或毫秒，所以不能用 like 和某一模式来搜索秒或毫秒。

最好使用 like 来搜索 date 和 time 值，因为这些数据类型条目可包含多种日期分量。例如，如果将值 “9:20” 插入名为 arrival_time 的 datetime 列，以下查询将无法找到该值，因为 Adaptive Server 将把该条目转换为 “Jan 1 1900 9:20AM”：

```
where arrival_time = "9:20"
```

但是，以下子句可找到值 9:20：

```
where arrival_time like "%9:20%"
```

您也可以将 date 和 time 数据类型用于 like 事务。

使用 *not like*

用于 like 的通配符都可以用于 not like。例如，若要在 authors 表中查找所有区号 *不是* 415 的电话号码，可使用下列查询之一：

```
select phone
from authors
where phone not like "415%"
```

```
select phone
from authors
where not phone like "415%"
```

使用 *not like* 和 ^ 获得不同的结果

不能总是用 like 和否定通配符 [^] 来重复 not like 模式。带否定通配符的匹配字符串将被分步计算 （每次一个字符）。如果在计算过程的任一点匹配失败，它将被排除。

例如，以下查询在数据库中查找其名称以 “sys”开头的系统表：

```
select name
from sysobjects
where name like "sys%"
```

如果共有 32 个对象，且 like 找到 13 个与模式匹配的名称，那么 not like 将找到 19 个与模式不匹配的对象。

```
where name not like "sys%"
```

如下所示的模式可能不会产生同样的结果：

```
like [^s][^y][^s]%
```

因为从结果中排除了所有首字母是 “s” *或* 第二个字母是 “y” *或* 第三个字母是 “s” 的名称及系统表名称，所以最终结果可能是 14，而不是 19。

将通配符用作文字字符

可通过将通配符转义并将其作为文字搜索来搜索通配符。在 like 匹配字符串中，可通过两种方式将通配符作为文字使用：方括号和 escape 子句。匹配字符串也可以是包含通配符的表中的一个变量或一个值。

方括号 （Transact-SQL 扩展）

可将方括号用于百分号、下划线、左括号和右括号。若要搜索破折号（而不是用它来指定范围），应在一组括号中将破折号用作第一个字符。

表 2-5：使用方括号搜索通配符

like 子句	搜索
like "5%"	5 以及后跟的包含 0 个或多个字符的任意字符串
like "5[%]"	5%
like "_n"	an、 in、 on 等等
like "[_]n"	_n
like "[a-cdf]"	a、 b、 c、 d 或 f
like "[-acdf]"	-、 a、 c、 d 或 f
like "[[]"	[
like "[]]"	]

escape 子句 （符合 SQL 标准）

使用 escape 子句可在 like 子句中指定转义字符。转义字符必须是单个字符串。可使用服务器缺省字符集中的任意字符。

表 2-6：使用 escape 子句

like 子句	搜索
like "5@%" escape "@"	5%
like "*_n" escape "*"	_n
like "%80@%%" escape "@"	包含 80% 的字符串
like "*_sql**%" escape "*"	包含 _sql* 的字符串
like "%#####_#%%" escape "#"	包含 ##_% 的字符串

转义字符仅在本身的 like 子句中才有效，而对同一语句中的其它 like 子句没有影响。

在转义字符之后，只有通配符 （_、%、[、] 和 [^]）及转义字符本身才有效。转义字符只影响其后面的字符。如果字符在某一模式中作为转义字符出现两次，则该字符串必须包含四个连续的转义字符 （请参见表 2-6 中最后一个示例）。否则，Adaptive Server 将引发一个 SQLSTATE 错误条件并返回一条错误消息。

指定多个转义字符将引发一个 SQLSTATE 错误条件，Adaptive Server 将返回一条错误消息：

```
like "%XX_%" escape "XX"
like "%XX%X_%" escape "XX"
```

通配符和方括号的交互作用

在方括号中转义字符将保留其自身的含义，这一点与通配符不同。在 escape 子句中不要使用现有通配符作为转义字符，因为：

- 如果将 “_” 或 “%” 指定为转义字符，它将在 like 子句内失去其本身含义，而仅作为一个转义字符。
- 如果将 “[” 或 “]” 指定为转义字符，方括号的 Transact-SQL 含义将在该 like 子句中禁用。
- 如果将 “-” 或 “^” 指定为转义字符，则在方括号中它将失去本身的含义，而仅作为一个转义字符。

使用尾随空白和 %

Adaptive Server 将把 like 子句中 “%”后的多个尾随空白截断为一个尾随空白。like ‘‘% ’’（百分比符号后跟两个空格）与 ‘‘X ’’（一个空格）、‘‘X ’’（两个空格）、‘‘X   ’’（三个空格）或任意数量的尾随空格相匹配。

在列中使用通配符

可在列中或列名中使用通配符。如果希望在 pubs2 数据库中创建一个名 special_discounts 的表，为打折销售制订一个价格方案，可输入：

```
create table special_discounts
id_type char(3), discount int)
insert into special_discounts
values("BU%", 10)
...
```

表中应该包含如下数据：

```
id_type discount
------- -----------
BU%              10
PS%              12
MC%              15
```

下列查询将在 where 子句的 id_type 中使用通配符：

```
select title_id, discount, price, price -
(price*discount/100)
from special_discounts, titles
where title_id like id_type
```

下面是该查询的结果：

```
title_id   discount      price

 -------- ----------- -------------- --------------
BU1032             10          19.99          17.99
BU1111             10          11.95          10.76
BU2075             10           2.99           2.69
BU7832             10          19.99          17.99
PS1372             12          21.59          19.00
PS2091             12          10.95           9.64
PS2106             12           7.00           6.16
PS3333             12          19.99          17.59
PS7777             12           7.99           7.03
```

```
MC2222              15            19.99           16.99
MC3021              15             2.99            2.54
MC3026              15             NULL            NULL

(12 rows affected)
```

这将允许复杂的模式匹配而不必构造一系列 or 子句。

字符串和引号

在输入或搜索字符和日期型数据 （char、 nchar、 unichar、 varchar、 nvarchar、 univarchar、 datetime、 smalldatetime、 bigdatetime、 bigtime、 date 和 time 数据类型）时，必须将其用单引号或双引号括起来。

有关字符数据的详细信息，请参见第 1 章 “SQL 构件块”，有关 date/time 数据类型的详细信息，请参见第 6 章 “使用和创建数据类型”。

“未知”值：NULL

NULL 在列中意味着该列没有被赋予条目。该列的数据值为 “未知”或 “不可用”。

NULL *不同于* “零”或 “空白”。利用空值可区分是特意在数值列输入零 （对字符列则为空白）还是无条目 （对数值列和字符列均为 NULL）。

在允许空值的列中：

- 如果未输入任何数据， Adaptive Server 将自动输入 “NULL”。
- 用户可显式输入*不带*引号的 “NULL”或 “null”。

如果在某个字符列中键入 “NULL”时带引号，它将被认为是数据而非空值。

查询结果将显示单词 NULL。例如， titles 表的 advance 列允许空值。通过检查该列中的数据，可判断一本书是*尚未*根据协议支付预付款 （在预付款列中 MC2222 行为零），还是输入数据时预付款金额*未知* （在预付款列中 MC3026 行为 NULL）。

```
select title_id, type, advance
from titles
where pub_id = "0877"

title_id     type             advance
--------     ----------       ---------
```

```
MC2222      mod_cook            0.00
MC3021      mod_cook       15,000.00
MC3026      UNDECIDED           NULL
PS1372      psychology      7,000.00
TC3218      trad_cook       7,000.00
TC4203      trad_cook       4,000.00
TC7777      trad_cook       8,000.00

(7 rows affected)
```

测试某一列是否为空值

在 where、 if 和 while 子句中用 is null （如第 15 章 “使用批处理和控制流语言”中所述）将列值与 NULL 进行比较，然后根据比较结果选择它们或执行某一特定操作。只有返回 TRUE 值的列才被选定或产生指定操作；返回值为 FALSE 或 UNKNOWN 的列则不然。

以下示例只选择 advance 少于 $5000 或为 NULL 的行：

```
select title_id, advance
from titles
where advance < $5000 or advance is null
```

Adaptive Server 将以不同的方式处理空值，具体取决于使用的**运算符**和要比较的值的类型。通常，空值的比较结果是 UNKNOWN，因为无法确定 NULL 是等于 （或不等于）给定值或另一个 NULL。如果 *expression* 是求值结果为 NULL 的任意列、变量或文字 （或这些项的组合），则下列情况将返回 TRUE：

- expression is null
- expression = null
- expression = *@x*，其中，*@x* 是包含 NULL 的变量或参数。此例外为用空的缺省参数来编写存储过程提供了方便。
- *expression* != *n*，其中 *n* 是不包含 NULL 的文字，*expression* 的求值结果为 NULL。

当表达式求值结果不为 NULL 时，这些表达式的否定形式将返回 TRUE：

- *expression* is not null
- *expression* != null
- *expression* != *@x*

如果使用关键字 like 和 not like 而非运算符 = 和 !=，则情况相反。以下比较将返回 TRUE：

- expression not like null

而以下比较将返回 FALSE：

- *expression* like null

请注意，这些表达式的最右侧文字为空或是包含 NULL 的变量或参数。如果比较运算的最右侧是表达式 （例如 *@nullvar* + 1），则整个表达式的求值结果为 NULL。

空列值不与其它空列值相连接。在 where 子句中将空列值与其它空列值进行比较时，无论比较运算符是什么，始终返回 UNKNOWN，且结果中不包括行。例如，以下查询不返回 column1 在两个表中都包含 NULL 的结果行 （虽然它可能返回其它行）：

```
select column1
from table1, table2
where table1.column1 = table2.column1
```

当与 NULL 连用时，下列运算符将返回结果：

- = 返回包含 NULL 的所有行。
- != 或 <> 返回*不*包含 NULL 的所有行。

如果为了与 SQL 兼容而将 set ansinull 置于打开状态，则当 = 和 != 运算符与 NULL 连用时，它们不会返回结果。无论 set ansinull 选项是何值，下列运算符在与 NULL 连用时都不会返回值：<、 <=、 !<、 >、 >=、 !>。

Adaptive Server 可确定某列值为 NULL。因此，以下表达式将被视为 true：

```
column1 =  NULL
```

但是，下列比较可能永远也不会确定，因为 NULL 意味着 “存在一个未知值”：

```
where column1 > null
```

不应假定两个未知值是一样的。

此逻辑同样适用于在 where 子句中使用两个列名的情况 （即连接两个表时）。类似于 “where column1 = column2” 的子句不会返回列中包含空值的行。

也可使用以下模式查找空值或非空值：

```
where column_name is [not] null
```

例如：

```
where advance < $5000 or advance is null
```

titles 表中的某些行包含不完整的数据。例如，一本名为 The Psychology of Computer Cooking (title_id = MC3026) 的书已被列入计划，但是其标题、标题标识号和可能的出版社还未确定，此时在 price、 advance、 royalty、 total_sales 和 notes 列中就会出现空值。因为在比较中空值不与任何内容匹配，所以某个查找所有标题标识号且书籍预付款少于 $5000 的查询将不会找到 The Psychology of Computer Cooking。

```
select title_id, advance
from titles
where advance < $5000

title_id  advance
--------  ----------
MC2222          0.00
PS2091      2,275.00
PS3333      2,000.00
PS7777      4,000.00
TC4203      4,000.00

(5 rows affected)
```

以下查询将查找预付款少于 $5000 *或* 在 advance 列中有空值的书籍：

```
select title_id, advance
from titles
where advance < $5000
  or advance is null

title_id  advance
--------  ----------
MC2222          0.00
MC3026          NULL
PC9999          NULL
PS2091      2,275.00
PS3333      2,000.00
PS7777      4,000.00
TC4203      4,000.00

(7 rows affected)
```

有关 create table 语句中 NULL 的信息以及 NULL 和缺省值之间关系的信息，请参见第 7 章 “创建数据库和表”。有关向表中插入空值的信息，请参见第 8 章 “添加、更改、传输和删除数据”。

FALSE 和 UNKNOWN 之间的区别

FALSE 和 UNKNOWN 之间有一个重要的逻辑区别：false 的相反值（“not false”）是 true，而 UNKNOWN 的相反值仍然是 UNKNOWN。例如，
“1 = 2”的求值结果为 false，而相对的 “1 != 2”的求值结果为 true。但“not unknown”仍然是 unknown。如果在比较运算中包括空值，则不能通过否定表达式来获得行的相对集合或相对真值。

用一个值替代 NULL

利用内置函数 isnull 可用一特定值替代空值。进行替代仅出于显示目的，实际列值不受影响。语法为：

isnull(*expression*, *value*)

例如，使用以下语句从 titles 中选择所有行，并在 notes 列中用“unknown”值显示所有空值。

```
select isnull(notes, "unknown")
from titles
```

计算结果为 NULL 的表达式

如果任何操作数为空，则带有算术运算符或逐位运算符的表达式求值结果为 NULL。例如，如果 column1 为 NULL，则以下表达式的求值结果也为 NULL：

```
1 + column1
```

并置字符串和 NULL

如果将字符串与 NULL 并置，则表达式的求值结果为字符串。例如：

```
select "abc" + NULL + "def"

-----
abcdef
```

系统生成的 NULL

在 Transact-SQL 中，系统生成的 NULL （例如，通过 convert 之类的系统函数得到的结果）与用户分配的 NULL 具有不同的行为。例如，在以下语句中，用户提供的 NULL 和 1 在进行 “不等于”比较时将返回 TRUE：

```
if (1 != NULL) print "yes" else print "no"

yes
```

但与系统生成的 NULL 进行同样的比较时将返回 UNKNOWN：

```
if (1 != convert(integer, NULL))
print "yes" else print "no"

no
```

为了获得更为一致的行为，使用 set ansinull on。这时，系统生成的 NULL 和用户提供的 NULL 都使比较计算返回 UNKNOWN。

带有逻辑运算符的连接条件

逻辑运算符 and、 or 和 not 用来在 where 子句中连接搜索条件。语法为：

{where | having} [not]
column_name join_operator column_name

其中， *join_operator* 是一个比较运算符， *column_name* 是在比较中使用的列。如果有歧义，可限定列名。

and 连接两个或两个以上的条件，并且仅当*全部*条件都成立时才返回结果。例如，以下查询仅查找作者姓为 Ringer、名为 Anne 的行。它不会查找作者姓名为 Albert Ringer 的行。

```
select *
from authors
where au_lname = "Ringer" and au_fname = "Anne"
```

or 也连接两个或两个以上的条件，只要其中*任何一个*条件成立就返回结果。以下查询将在 au_fname 列中搜索包含 Anne 或 Ann 的行。

```
select *
from authors
where au_fname = "Anne" or au_fname = "Ann"
```

可指定多达 252 个 and 和 or 条件。

not 否定它后面所跟的表达式。以下查询将选择居住在 California 以外的所有作者：

```
select * from authors
where not state = "CA"
```

如果在某条语句中使用了多个逻辑运算符，通常先对 and 运算符求值，然后再对 or 运算符求值。可使用小括号更改执行顺序。例如：

```
select * from authors
where (city = "Oakland" or city = "Berkeley") and state
= "CA"
```

逻辑运算符的优先级

算术和逐位运算符在逻辑运算符之前处理。如果在某条语句中使用了多个逻辑运算符，则首先对 not 求值，然后是 and，最后是 or。有关详细信息，请参见第 14 页的 “逐位运算符”。

例如，以下查询将在 titles 表中查找*全部*商业方面的书籍而不管其预付款是多少，还将查找预付款大于 $5500 的全部心理学方面的书籍。预付款条件只适用于心理学书籍，因为 and 将先于 or 进行处理。

```
select title_id, type, advance
from titles
where type = "business" or type = "psychology"
  and advance > 5500

title_id  type       advance
--------  ----------  ----------
BU1032    business     5,000.00
BU1111    business     5,000.00
BU2075    business    10,125.00
BU7832    business     5,000.00
PS1372    psychology   7,000.00
PS2106    psychology   6,000.00

(6 rows affected)
```

若要更改查询含义，可加上小括号以强制先求 or 的值。以下查询将查找预付款大于 $5500 的所有商业和心理学方面的书籍：

```
select title_id, type, advance
from titles
where (type = "business" or type = "psychology")
  and advance > 5500

title_id  type         advance
--------  ----------   ---------
BU2075    business     10,125.00
PS1372    psychology    7,000.00
PS2106    psychology    6,000.00

(3 rows affected)
```

第 3 章

使用集合、分组和排序

本章介绍 sum、 avg、 count、 count(*)、 count_big、 count_big(*)、 max 和 min 集合函数，使用这些函数可以对查询中所检索到的数据进行汇总。本章还讨论如何使用 group by、 having 和 order by 子句将数据按类别和子群进行组织。还讨论了两个 Transact-SQL 扩展，compute 子句和 union 运算符。

如果您的 Adaptive Server 不区分大小写，则可以参见 《参考手册：命令》中的 group by 和 having 子句以及 compute 子句来了解有关区分大小写如何影响这些子句返回的数据。

集合函数的使用

集合函数为：sum、 avg、 count、 min、 max、 count_big、 count(*) 和 count_big(*)。使用**集合函数**可以计算并汇总数据。例如，若要得出 pubs2 数据库的 titles 表中的书籍销售量，请输入：

```
select sum(total_sales)
from titles

-------------
       97746

(1 row affected)
```

请注意，示例中的集合列没有列标题。

集合函数将要对其值进行运算的列名作为参数。可以将集合函数应用于表中的所有行、由 where 子句指定的表的子集或者表中的一组或多组行。在应用集合函数的每组行处， Adaptive Server 都生成一个值。

以下是集合函数的语法：

aggregate_function ([all | distinct] *expression*)

expression 通常为列名。但它也可是常量、函数或由算术运算符或逐位运算符连接的列名、常量和函数的任何组合。还可以在表达式中使用 case 表达式或子查询。

例如，使用以下语句可以计算价格加倍后所有书籍的平均价格：

```
select avg(price * 2)
from titles

-------------
        29.53

(1 row affected)
```

应用集合函数前，可以将可选关键字 distinct 用于 sum、 avg、 count、 min 和 max 以消除重复值。缺省值为 all，它对所有行执行运算。

表 3-1 中显示了集合函数的语法及其产生的结果：

表 3-1：集合函数的语法和结果

集合函数	结果
sum([all | distinct] *expression*)	表达式中 （不同）值的总和。
avg([all | distinct] *expression*	表达式中 （不同）值的平均值。
count([all | distinct] *expression*)	以 integer 形式返回的表达式中 （不同）非空值的数量。
count_big [all | distinct] *expression*	以 bigint 形式返回的表达式中 （不同）非空值的数量。
count(*)	integer 形式的选定行数量。
count_big(*)	bigint 形式的选定行数量。
max (*expression*)	表达式中的最大值。
min (*expression*)	表达式中的最小值。

可以在选择列表中使用集合函数 （如前例所示），或在 having 子句中使用集合函数。有关 having 子句的信息，请参见第 94 页的 “选择数据组：having 子句”。

不能在 where 子句中使用集合函数，但多数在选择列表中有集合函数的 select 语句都包括限制应用集合的行的 where 子句。在本节前面给出的示例中，每个集合函数都为整个表生成一个汇总值。

如果 select 语句包括 where 子句，但不包括 group by 子句（请参见第 81 页的 “将查询结果分组：group by 子句”），则集合函数为行的子集生成一个值，称为**标量集合**。但是， select 语句也可以在其选择列表中包括列（Transact-SQL 扩展），这样就为结果表中的每行都重复此值。

下面的查询只返回商业书籍的平均预付款和总销售额，在其前有一列名为 “advance and sales”：

```
select "advance and sales", avg(advance),
sum(total_sales)
from titles
where type = "business"

-----------------  ----------------- -----------
advance and sales          6,281.25       30788

(1 row affected)
```

集合函数和数据类型

可以对任何类型的列使用集合函数，但以下情况例外：

- sum 和 avg 只能用于数值列 — bigint、 int、 smallint、 tinyint、 unsigned bigint、 unsigned int、 unsigned smallint、 decimal、 numeric、 float 和 money。
- 不能对 bit 数据类型使用 min 和 max。
- 不能对 text 和 image 数据类型使用 count(*) 和 count_big(*) 以外的集合函数。

例如，可以使用 min （最小值）得出字符类型列中的最小值 （字母表中最靠前的那个值）：

```
select min(au_lname)
from authors

---------------------------
Bennet

(1 row affected)
```

但是，不能计算文本列内容的平均值：

```
select avg(au_lname)
from authors

Msg 257, Level 16, State 1:
---------------------------
(1 row affected)
Line 1:
Implicit conversion from datatype 'VARCHAR' to 'INT' is
not allowed.  Use the CONVERT function to run this
query.
```

count 与 *count(*)*

count 得出表达式中非空值的数量，而 count(*) 得出表中的总行数。以下语句得出书籍的总数：

```
select count(*)
from titles

------------------
                18


(1 row affected)
```

count(*) 返回指定表中的行数 （不排除重复行）。它将每行都计算在内，包括具有空值的行。

与其它集合函数一样，您可以将 count(*) 与选择列表中的其它集合、与 where 子句等组合使用：

```
select count(*), avg(price)
from titles
where advance > 1000

---------- ---------
        15     14.42

(1 row affected)
```

带有 *distinct* 的集合函数

可选关键字 distinct 只能用于 sum、avg、count_big 和 count。使用 distinct 时， Adaptive Server 会在执行计算之前消除重复值。

如果使用 distinct，则不能在参数中包括算术表达式。参数只能使用列名。 distinct 放在列名前的小括号中。例如，若要查找作者所在的不同城市数目，请输入：

```
select count(distinct city)
from authors

-------------
           16

(1 row affected)
```

若要准确计算所有商业书籍的平均价格，请省略 distinct。以下语句返回所有商业书籍的平均价格：

```
select avg(price)
from titles
where type = "business"

-------------
        13.73

(1 row affected)
```

但是，如果两本或更多书具有相同的价格，并且使用 distinct，则相同的价格只计算一次：

```
select avg(distinct price)
from titles
where type = "business"

-------------
        11.64

(1 row affected)
```

空值和集合函数

- Adaptive Server 忽略对其运行集合函数的列中的任何空值，以使该函数有效 （count(*) 和 count_big(*) 除外，这两个函数包括空值）。如果将 ansinull 设置为 on，则每次忽略空值时，Adaptive Server 都返回错误消息。请参见 《参考手册：命令》。

例如， titles 表中预付款的 count 与书名的 count 不同，因为 advance 列中有空值：

```
select count(advance)
from titles

-------------
           16

(1 row affected)

select count(title)
from titles

-------------
           18

(1 row affected)
```

如果列中的所有值都为空，则 count 返回 0。如果没有行满足 where 子句中指定的条件，则 count 返回 0。其它函数都返回 NULL。示例如下：

```
select count(distinct title)
from titles
where type = "poetry"

-------------
            0

(1 row affected)

select avg(advance)
from titles
where type = "poetry"

-------------
          NULL

(1 row affected)
```

使用统计集合

集合函数可汇总数据库中一组行中的数据。简单的集合函数（例如 sum、 avg、 max、 min、 count_big 和 count）只允许用于 select 列表、having 和 order by 子句，以及 select 语句的 compute 子句。这些函数可汇总数据库中一组行中的数据

Adaptive Server Enterprise 支持统计集合函数，这些函数允许对数值数据进行统计分析。统计集合函数包括 stddev、 stddev_samp、 stddev_pop、 variance、 var_samp 和 var_pop。

这些函数（包括 stddev 和 variance）都是真正的集合函数，因为它们可以计算查询的 group by 子句确定的一组行的值。如同其它基本集合函数（例如 max 或 min）一样，它们在计算时会忽略输入中的空值。此外，无论是否正在分析表达式的域，所有方差和标准偏差的计算都使用 IEEE 双精度浮点标准。

如果对任何方差或标准偏差函数的输入为空集，则每个函数都将返回空值作为其结果。如果对任何方差或标准偏差函数的输入为单个值，则每个函数都将返回 0 作为其结果。

标准偏差和方差

统计集合函数（及其别名）如下：

- stddev_pop（也称为 stdevp）— 总体的标准偏差。计算对每个组行求值的所提供值表达式的总体标准偏差（如果指定了 distinct，则已删除在复制之后仍保留的各行），其定义为总体方差的平方根。
- stddev_samp（也称为 stdev、 stddev）— 样本的标准偏差。计算对每个组行求值的所提供值表达式的总体标准偏差（如果指定了 distinct，则已删除在复制之后仍保留的各行），其定义为样本方差的平方根。
- var_pop（也称为 varp）— 总体的方差。计算对每个组行求值的值表达式的总体方差（如果指定了 distinct，则已删除在复制之后仍保留的各行），总体方差定义为值表达式与值表达式均值之差的平方和，再除以组中的行数。
- var_samp（也称为 var、 variance）— 样本的方差。计算对组中各行求值的值表达式的样本方差（如果指定了 distinct，则已删除在复制之后仍保留的各行），样本方差定义为与值表达式均值之差的平方和，然后再除以组中的行数减 1。

有关语法和用法信息，请参见《参考手册：构件块》。

统计集合

统计集合类似于 avg 集合，原因如下：

- 语法为：

  var_pop ([all | distinct] *expression*)

- 只有包含数值数据类型的表达式是有效的。
- Null 值不参与计算。
- 仅当没有数据参与计算时，结果才为 NULL。
- distinct 或 all 子句可以位于表达式之前 （缺省为 all）。
- 可以将统计集合用作矢量集合 （使用 group by）和标量集合 （不使用 group by），或用于 compute 子句。

然而，与 avg 集合不同的是，统计集合的结果如下：

- 始终为 float 数据类型 （即，双精度浮点），然而对于 avg 集合，结果的数据类型与表达式的数据类型相同 （存在例外）。
- 0.0 表示单个数据点。

公式

图 3-1：总体相关的统计集合函数的公式

定义均值为 μ (var_pop) 的总数为 *n* 的总体方差的公式如下所示。总体标准偏差 (stddev_pop) 为此值的正平方根。

$$\left(\sigma^2 = \frac{\sum (x_i - \mu)^2}{n} \right)$$

σ^2 = 方差
n = 总数
$\mu = x_i$ 的均值

图 3-2：样本相关的统计集合函数的公式

定义均值为 $\bar{x}$ (var_samp) 的样本数为 *n* 的总体方差的无偏估计的公式如下所示。样本标准偏差 (stddev_samp) 是此值的正平方根。

$$s^2 = \frac{\sum (x_i - \bar{x})^2}{n - 1}$$

s^2 = 方差
n = 样本数
$\bar{x} = x_i$ 的均值

这两个公式之间的主要区别是除以 *n-1* 而不是 *n*。

这两个函数相似，但用途不同：

- var_samp — 在希望对样本求值时使用，样本是总体的一个子集，作为整个总体的代表
- var_pop — 在拥有总体的全部可用数据时使用，或在 *n* 值太大，以致于 *n* 和 *n-1* 之间的差异可忽略不计时使用

将查询结果分组：*group by* 子句

group by 子句将查询的输出划分为若干组。可以按一个或多个列名进行分组，也可以按在表达式中使用数值数据类型所计算得出的列结果进行分组。与集合一起使用时， group by 检索每个子群中的计算结果，并会返回多行。

group by 列 （或表达式）的最大数量未受到显式限制。 group by 结果的唯一限制是 group by 列加上集合结果的宽度不超过 64K。

注释　不能对 text、 unitext 或 image 数据类型使用 order by 列。

尽管使用 group by 时可以不带集合，但这样的结构限制了功能，并可能产生混淆的结果。下例按书籍类型对结果进行分组：

```
select type, advance
     from titles
group by type
```

```
type              advance
------------         ---------
popular comp          7,000.00
popular comp          8,000.00
popular comp              NULL
business              5,000.00
business              5,000.00
business             10,125.00
mod_cook                  .00
mod_cook             15,000.00
trad_cook             7,000.00
trad_cook             4,000.00
trad_cook             8,000.00
UNDECIDED                 NULL
```

```
psychology            7,000.00
psychology            2,275.00
psychology            6,000.00
psychology            2,000.00
psychology            4,000.00

(18 rows affected)
```

对 advance 列使用集合后，查询返回每组的和：

```
select type, sum(advance)
   from titles
group by type

type
------------ ------------------------
popular_comp                15,000.00
business                    25,125.00
mod_cook                    15,000.00
trad_cook                   19,000.00
UNDECIDED                        NULL
psychology                  21,275.00

(6 rows affected)
```

使用集合的 group by 子句中的汇总值称为**矢量集合**，与标量集合相对，后者在只返回一行时产生 （请参见第 73 页的 “集合函数的使用”）。

请参见 《参考手册：命令》。

group by 和 SQL 标准

group by 的 SQL 标准比 Sybase 标准更具有限制性。 SQL 标准要求：

- 选择列表中的列必须在 group by 表达式中，或者它们必须是集合函数的参数。
- group by 表达式只能包含选择列表中的列名，而不能包含仅用作矢量集合的参数的那些列名。

```
set fipsflagger on
```

有关 fipsflagger 选项的详细信息，请参见 《参考手册：命令》中的 set 命令。

许多 Transact-SQL 扩展 （在以下的节中描述）放宽了这些限制。但是，结果集越复杂，理解起来可能越困难。如果按如下设置 fipsflagger 选项，就会出现警告消息，说明使用了 Transact-SQL 扩展：

用 group by 对组进行嵌套

通过在 group by 子句中列出多个列可以对组进行嵌套。用 group by 建立集后，即应用了集合。以下语句得出平均价格和书籍总销售额，先按出版社标识号分组，然后按类型分组：

```
select pub_id, type, avg(price), sum(total_sales)
from titles
group by pub_id, type
pub_id type
------ ------------ ------ -------
0736 business 2.99 18,722
0736 psychology 11.48 9,564
0877 UNDECIDED NULL NULL
0877 mod_cook 11.49 24,278
0877 psychology 21.59 375
0877 trad_cook 15.96 19,566
1389 business 17.31 12,066
1389 popular_comp 21.48 12,875

(8 rows affected)
```

可以在组内嵌套组。 group by 列 （或表达式）的最大数量未受到显式限制。 group by 结果的唯一限制是 group by 列加上集合结果的宽度不超过 64K。

使用 *group by* 在查询中引用其它列

SQL 标准规定 group by 子句必须包含来自 select 列表中的项。但是，使用 Transact-SQL 可以在 group by 或 select 列表中指定任何有效的列名，无论它们是否采用集合。

通过以下扩展， Sybase 解除了对包括 group by 的查询的 select 列表中可以包括或忽略的内容的限制。

- 选择列表中的列并不限于分组列和用于矢量集合的列。
- group by 所指定的列并不限于选择列表中的非集合列。

矢量集合必须伴有 group by 子句。 SQL 标准要求 select 列表中的非集合列与 group by 列相匹配。但是，上述第一条允许在查询的 select 列表中指定附加的扩展列。

例如， SQL 的许多版本不允许在 select 列表中包括扩展的 title_id 列，但在 Transact-SQL 中这是合法的：

```
select type, title_id, avg(price), avg(advance)
from titles
group by type
type title_id

------------ -------- ----- -------
business BU1032 13.73 6,281.25
business BU1111 13.73 6,281.25
business BU2075 13.73 6,281.25
business BU7832 13.73 6,281.25
mod_cook MC2222 11.49 7,500.00
mod_cook MC3021 11.49 7,500.00
UNDECIDED MC3026 NULL NULL
popular_comp PC1035 21.48 7,500.00
popular_comp PC8888 21.48 7,500.00
popular_comp PC9999 21.48 7,500.00
psychology PS1372 13.50 4,255.00
psychology PS2091 13.50 4,255.00
psychology PS2106 13.50 4,255.00
psychology PS3333 13.50 4,255.00
psychology PS7777 13.50 4,255.00
trad_cook TC3218 15.96 6,333.33
trad_cook TC4203 15.96 6,333.33
trad_cook TC7777 15.96 6,333.33
(18 rows affected)
```

上述示例仍然基于 type 列集合 price 和 advance 列，但其结果也显示每组中包括的书籍的 title_id。

上述第二个扩展允许对未指定为查询的选择列表中的列的那些列进行分组。这些列不出现在结果中，但矢量集合仍然计算其汇总值。例如：

```
select state, count(au_id)
from authors
group by state, city

state
----- --------
CA 2
CA 1
CA 5
CA 5
CA 2
CA 1
CA 1
CA 1
```

```
CA 1
IN 1
KS 1
MD 1
MI 1
OR 1
TN 1
UT 2
(16 rows affected)
```

此示例虽然不显示每组所包含的城市，但同时按 state 和 city 对矢量集合结果进行分组。因此，结果可能会造成误解。

您可能会认为以下查询应该产生与前一查询相似的结果，因为似乎只有矢量集合才记录每行每个城市的数量：

```
select state, count(au_id)
from authors
group by city
```

但是，其结果却大大不同。由于未将 group by 同时用于 state 和 city 列，查询会记录每个城市的数量，但将每行中该城市的记录内容显示在 authors 中，而不是将其按城市组成一个结果行。

```
state
----- -----------
CA 1
CA 5
CA 2
CA 1
CA 5
KS 1
CA 2
CA 2
CA 1
CA 1
TN 1
OR 1
CA 1
```

```
MI 1
IN 1
CA 5
CA 5
CA 5
MD 1
CA 2
CA 1
UT 2
UT 2
(23 rows affected)
```

在包括 where 子句或连接的复杂查询中使用 Transact-SQL 扩展时，结果可能会变得更加难以理解。要避免因 group by 而产生令人混淆或误解的结果， Sybase 建议使用 fipsflagger 选项来标识包含 Transact-SQL 扩展的查询。有关详细信息，请参见第 82 页的 “group by 和 SQL 标准”。

请参见 《参考手册：命令》。

使用外连接和集合扩展列

如果同时使用外连接和集合扩展列，且如果集合扩展列来自外连接的内部表，则它们可能导致查询的结果集等于外连接的结果集。

外连接通过使用 Sybase 外连接运算符 =* 或 *= 来连接两个表中的列。这些符号为 Sybase Transact-SQL 扩展语法。它们不是 ANSI SQL 符号，而 OUTER JOIN 不是 Transact-SQL 中的关键字。本节仅涉及 Sybase 语法。

出于外连接的目的，星号一边所指定的列是来自外部表的外部列。

尽管集合扩展列使用集合函数 （max、 min），但它不包括在查询的 group by 子句中。

例如，若要创建结果包含空值行的外连接，请输入：

```
select publishers.pub_id, titles.price
from publishers, titles
where publishers.pub_id *= titles.pub_id
and titles.price > 20.00
```

```
pub_id price
------ ----------------- ------
0736 NULL
0877 20.95
0877 21.59
1389 22.95
(4 rows affected)
```

同样，若要创建结果包括空值行的外连接和集合列，请输入：

```
select publishers.pub_id, max(titles.price)
from publishers, titles
where publishers.pub_id *= titles.pub_id
and titles.price > 20.00
group by publishers.pub_id
pub_id
--------------- --------
0736 NULL
0877 21.59
1389 22.95
(3 rows affected)
```

若要创建带有集合扩展列的外连接和集合列 （其结果包含空值行），请输入：

```
select publishers.pub_id, titles.title_id,
max(titles.price)
from publishers, titles
where publishers.pub_id *= titles.pub_id
and titles.price > 20.00
group by publishers.pub_id
--------------- --------
....
(54 rows affected)
```

表达式和 *group by*

另一个 Transact-SQL 扩展允许按不包括集合函数的表达式进行分组。例如：

```
select avg(total_sales), total_sales * price
from titles
group by total_sales * price

---------      -------------
     2045          22,392.75
     2032          40,619.68
     4072          81,399.28
     NULL               NULL
     4095          61,384.05
    18722          55,978.78
      375           7,856.25
    15096         180,397.20
     3876          46,318.20
      111             777.00
     3336          26,654.64
     4095          81,859.05
    22246          66,515.54
     8780         201,501.00
      375           8,096.25
     4095          81,900.00

(16 rows affected)
```

允许使用 “total_sales * price” 表达式。

不能对列标题 （也称为**别名**）执行 group by，但仍可以在选择列表中使用它。以下语句会产生错误消息：

```
select Category = type, title_id, avg(price),
avg(advance)
from titles
group by Category
------------------
Msg 207, Level 16, State 4:
Line 1:
Invalid column name 'Category'
Msg 207, Level 16, State 4:
Line 1:
Invalid column name 'Category'
```

group by 子句应是 “group by type”，而不是 “group by Category”。

```
select Category = type, title_id, avg(price),
avg(advance)
from titles
group by type
--------------
          21.48
          13.73
          11.49
          15.96
           NULL
13.50

(6 rows affected)
```

在嵌套集合中使用 *group by*

另一个 Transact-SQL 扩展允许在标量集合中嵌套矢量集合。例如，要使用非嵌套集合得出所有类型书籍的平均价格，请输入：

```
select avg(price)
from titles
group by type

---------------
 NULL
13.73
11.49
21.48
13.50
15.96

(6 rows affected)
```

在 max 函数中嵌套平均价格会生成一组书籍的最高平均价，按类型分组：

```
select max(avg(price))
from titles
group by type

-------------
        21.48

(1 row affected)
```

根据定义， group by 子句适用于最内层的集合 — 本例中为 avg。

空值和 *group by*

如果分组列包含一个空值，则该空值所在行在结果中将自成一组。如果分组列包含多个空值，则这些空值组成一个组。下例使用 group by 和 advance 列，其中包含一些空值：

```
select advance, avg(price * 2)
from titles
group by advance
```

```
advance
-----------------  -----------------
               NULL                NULL
               0.00               39.98
            2000.00               39.98
            2275.00               21.90
            4000.00               19.94
            5000.00               34.62
            6000.00               14.00
            7000.00               43.66
            8000.00               34.99
           10125.00                5.98
           15000.00                5.98

(11 rows affected)
```

如果使用 count(*column_name*) 集合函数，则对包含空值的列执行 group by 会对分组行的计数返回零，因为 count(*column_name*) 不将空值计算在内。在大多数情况下，应该改用 count(*)。下例对 titles 表中的 price 列（包含空值）进行分组和计数，并显示 count(*) 用于比较：

```
select price, count(price), count(*)
from titles
group by price
```

```
price
------------- ----- -----
        NULL     0     2
        2.99     2     2
        7.00     1     1
        7.99     1     1
       10.95     1     1
       11.95     2     2
       14.99     1     1
       19.99     4     4
       20.00     1     1
       20.95     1     1
       21.59     1     1
```

```
       22.95     1     1

(12 rows affected)
```

where 子句和 *group by*

可以在带有 group by 的语句中使用 where 子句。执行任何分组之前会消除不满足 where 子句中的条件的行。以下是一个示例：

```
select type, avg(price)
from titles
where advance > 5000
group by type

type
-------------    --------
business            2.99
mod_cook            2.99
popular_comp       21.48
psychology         14.30
trad_cook          17.97

(5 rows affected)
```

只有预付款超过 $5000 的行才包括在用于生成查询结果的组中。

但是， Adaptive Server 处理选择列表和 where 子句中额外列的方法似乎是矛盾的。例如：

```
select type, advance, avg(price)
from titles
where advance > 5000
group by type

type           advance
-------------  ---------  --------
business        5,000.00      2.99
business        5,000.00      2.99
business       10,125.00      2.99
business        5,000.00      2.99
mod_cook            0.00      2.99
mod_cook       15,000.00      2.99
popular_comp    7,000.00     21.48
popular_comp    8,000.00     21.48
popular_comp        NULL     21.48
psychology      7,000.00     14.30
psychology      2,275.00     14.30
```

```
psychology        6,000.00     14.30
psychology        2,000.00     14.30
psychology        4,000.00     14.30
trad_cook         7,000.00     17.97
trad_cook         4,000.00     17.97
trad_cook         8,000.00     17.97

(17 rows affected)
```

查看 advance （扩展的）列的结果时，会发现查询似乎忽略 where 子句。 Adaptive Server 仍然只使用那些满足 where 子句的行来计算矢量集合，但也显示包括在选择列表中的任何扩展列的所有行。要进一步限制这些行在结果中出现，请使用 having 子句 （稍后在本章中描述）。

group by 和 *all*

group by 子句中的关键字 all 是 Transact-SQL 增强。它只有在使用它的 select 语句中也包括 where 子句时才有意义。

如果使用 all，则即使某些组没有任何满足搜索条件的行，查询结果也包括由 group by 子句生成的所有组。如果不使用 all，则包括 group by 的 select 语句不显示其中没有满足条件的行的组。

以下是一个示例：

```
select type, avg(advance)
from titles
where advance > 1000 and advance < 10000
group by type

type
------------  ------------------------
business                     5,000.00
popular_comp                 7,500.00
psychology                   4,255.00
trad_cook                    6,333.33

(4 rows affected)

select type, avg(advance)
from titles
where advance > 1000 and advance < 10000
group by all type

type
------------  ------------------------
UNDECIDED                        NULL
```

```
business                       5,000.00
mod_cook                           NULL
popular_comp                   7,500.00
psychology                     4,255.00
trad_cook                      6,333.33

(6 rows affected)
```

第一个语句只生成要求预付款在 $1000 和 $10,000 之间的书籍的组。由于所有现代烹调书籍的预付款都不在此范围之内，因此结果中没有 mod_cook 类型的组。

第二个语句会生成所有类型的组，包括现代烹调和 “UNDECIDED”，即使现代烹调组并不包含任何满足 where 子句中指定条件的行也是如此。Adaptive Server 对于所有没有满足条件的行的组都返回 NULL 结果。

不带 *group by* 的集合

根据定义，标量集合适用于表中所有行，对于每个函数都为整个表生成一个值。允许包括带矢量集合的扩展列的 Transact-SQL 扩展也允许包括带标量集合的扩展列。例如，查看 publishers 表：

```
pub_id pub_name             city            state
------ -------------------  --------------- -----
0736   New Age Books        Boston          MA
0877   Binnet & Hardley     Washington      DC
1389   Algodata Infosystems Berkeley        CA
```

它包含三行。以下查询基于表的每行生成一个三行的标量集合：

```
select pub_id, count(pub_id)
from publishers

pub_id
---------- ---------
0736               3
0877               3
1389               3

(3 rows affected)
```

因为 Adaptive Server 将 publishers 视为一个组，所以标量集合适用于（单组）表。除标量集合外，结果对于在选择列表中包括的每列还显示表的每行。

where 子句对标量集合和对矢量集合的作用方式相同。 where 子句限制包括在集合汇总值中的列，但对于在选择列表中指定的每个扩展列，它不影响在结果中出现的行。例如：

```
select pub_id, count(pub_id)
from publishers
where pub_id < "1000"

pub_id
-------------- -----------
0736                     2
0877                     2
1389                     2

(3 rows affected)
```

像 group by 的其它 Transact-SQL 扩展一样，此扩展提供的结果可能难以理解，特别是对于大型表的查询或使用多表连接的查询更是如此。

选择数据组：*having* 子句

使用 having 子句显示或拒绝由 group by 子句所定义的行。 having 子句为 group by 子句设置条件的方式和 where 为 select 子句设置条件的方式相同，不同的是 where 不能包括集合，而 having 则经常包括集合。下例是合法的：

```
select title_id
from titles
where title_id like "PS%"
having avg(price) > $2.0
```

但下例是不合法的：

```
select title_id
from titles
where avg(price) > $20
--------------------
Msg 147, Level 15, State 1
Line 1:
An aggregate function may not appear in a WHERE clause
unless it is in a subquery that is in a HAVING clause,
and the column being aggregated is in a table named in
a FROM clause outside of the subquery.
```

having 子句可以引用任何出现在选择列表中的项。

以下语句是带有集合函数的 having 子句的示例。它按类型对 titles 表中的行进行分组，但排除只包括一本书的组：

```
select type
from titles
group by type
having count(*) > 1

type
----------------
business
mod_cook
popular_comp
psychology
trad_cook

(5 rows affected)
```

以下是不带集合的 having 子句的示例。它按类型对 titles 表分组，只显示以字母 “p”开头的那些类型：

```
select type
from titles
group by type
having type like "p%"

type
------------
popular_comp
psychology

(2 rows affected)
```

在 having 子句中包括多个条件时，请用 and、or 或 not 将条件进行组合。例如，要按出版社对 titles 表进行分组，并且要只包括预付款总额已经支付 $15,000 以上、其平均书价低于 $18 且其标识号 (pub_id) 大于 0800 的出版社，则语句为：

```
select pub_id, sum(advance), avg(price)
from titles
group by pub_id
having sum(advance) > 15000
  and avg(price) < 18
  and pub_id > "0800"

pub_id
------  ---------------- ----------------
0877          41,000.00            15.41

(1 row affected)
```

having、 group by 和 where 子句如何交互作用

在查询中包括 having、 group by 和 where 子句时，每个子句影响行的顺序决定最终的结果：

- where 子句排除不满足其搜索条件的行。
- 对于 group by 表达式中的每个唯一值， group by 子句都将剩余的行集中为一组。
- 在选择列表中指定的集合函数计算每组的汇总值。
- having 子句从最终结果中排除不满足其搜索条件的行。

以下查询说明了在一个包含集合的 select 语句中 where、 group by 和 having 子句的用法：

```
select stor_id, title_id, sum(qty)
from salesdetail
where title_id like "PS%"
group by stor_id, title_id
having sum(qty) > 200
```

```
stor_id  title_id
-------  --------  -----------
5023     PS1372            375
5023     PS2091          1,845
5023     PS3333          3,437
5023     PS7777          2,206
6380     PS7777            500
7067     PS3333            345
7067     PS7777            250

(7 rows affected)
```

查询按以下顺序执行：

1 where 子句只标识带有以 “PS”开头 （心理学书籍）的 title_id 的行，

2 group by 按相同的 stor_id 和 title_id 收集行，

3 sum 集合计算每组销售书籍的总数，并且

4 having 子句从最终结果中排除书籍总数不超过 200 本的组。

所有以前的 having 示例都遵循 SQL 标准，这些标准规定 having 表达式中的列必须只有一个值，而且这些列必须在选择列表或 group by 子句中。但是， having 的 Transact-SQL 扩展允许列或表达式不在选择列表中，也允许其不在 group by 子句中。

下例确定每种书籍类型的平均价格，但排除那些总销售额未超过 $10,000 的类型，即使 sum 集合没有出现在结果中也是如此。

```
select type, avg(price)
from titles
group by type
having sum(total_sales) > 10000

type
------------  ----------
business           13.73
mod_cook           11.49
popular_comp       21.48
trad_cook          15.96

(4 rows affected)
```

扩展表现为仿佛列或表达式是选择列表的一部分，而不是显示结果的一部分。如果使用 having 包含了非集合列，但它不是选择列表或 group by 子句的一部分，则此查询生成的结果与本章前面描述的 “扩展”列扩展相似。例如：

```
select type, avg(price)
from titles
group by type
having total_sales > 4000

type
------------  ----------
business           13.73
business           13.73
business           13.73
mod_cook           11.49
popular_comp       21.48
popular_comp       21.48
psychology         13.50
trad_cook          15.96
trad_cook          15.96

(9 rows affected)
```

与扩展列不同， total_sales 列不出现在最终结果中，但对每种类型所显示的行数取决于每种书籍的 total_sales。查询指出三个 business、一个 mod_cook、两个 popular_comp、一个 psychology 以及两个 trad_cook 书籍的总销售额都超过 $4000。

如上所述， Adaptive Server 处理扩展列的方式看起来似乎是查询忽略最终结果中的 where 子句。若要使 where 条件影响扩展列的结果，请在 having 子句中重复这些条件。例如：

```
select type, advance, avg(price)
from titles
where advance > 5000
group by type
having advance > 5000

type           advance
-------------  ---------  --------
business       10,125.00      2.99
mod_cook       15,000.00      2.99
popular_comp    7,000.00     21.48
popular_comp    8,000.00     21.48
psychology      7,000.00     14.30
psychology      6,000.00     14.30
trad_cook       7,000.00     17.97
trad_cook       8,000.00     17.97

(8 rows affected)
```

使用不带 *group by* 的 *having*

带有 having 子句的查询也应该有 group by 子句。如果省略 group by，则 where 子句未排除的所有行作为一个组返回。

因为未在 where 与 having 子句之间分组，所以它们不能相互独立地运行。 having 与 where 作用方式相同，因为它影响的是单个组中而不是多组中的行，只是 having 子句仍可以使用集合。

下例按照以下方式使用 having 子句：计算平均价格，从结果中排除预付款大于 $4,000 的书籍，并生成价格低于平均价格的结果：

```
select title_id, advance, price
from titles
where advance < 4000
having price > avg(price)

title_id        advance    price
-------------  ---------  --------
BU1032          5,000.00     19.99
BU7832          5,000.00     19.99
MC2222              0.00     19.99
PC1035          7,000.00     22.95
```

```
PC8888          8,000.00     20.00
PS1372          7,000.00     21.59
PS3333          2,000.00     19.99
TC3218          7,000.00     20.95

(8 rows affected)
```

还可以使用带 Transact-SQL 扩展的 having 子句，它允许从其选择列表中包括集合的查询中省略 group by 子句。这些标量集合函数为作为单个组的表计算值，而不是为表中的多组计算值。

下例中省略了 group by 子句，这使得集合函数计算整个表的值。 having 子句从结果组中排除不匹配的行。

```
select pub_id, count(pub_id)
from publishers
having pub_id < "1000"

pub_id
------ ----------------
0736                  3
0877                  3

(2 rows affected)
```

对查询结果进行排序：*order by* 子句

使用 order by 子句可以按一个或多个（最多 31 个）列对查询结果进行排序。每个排序不是升序 (asc) 就是降序 (desc)。如果未指定任何排序方式，则缺省为 asc。以下查询按 pub_id 对结果进行排序：

```
select pub_id, type, title_id
from titles
order by pub_id

pub_id  type            title_id
------  ------------    --------
0736    business        BU2075
0736    psychology      PS2091
0736    psychology      PS2106
0736    psychology      PS3333
0736    psychology      PS7777
0877    UNDECIDED       MC3026
0877    mod_cook        MC2222
0877    mod_cook        MC3021
```

```
0877    psychology      PS1372
0877    trad_cook       TC3218
0877    trad_cook       TC4203
0877    trad_cook       TC7777
1389    business        BU1032
1389    business        BU1111
1389    business        BU7832
1389    popular_comp    PC1035
1389    popular_comp    PC8888
1389    popular_comp    PC9999

(18 rows affected)
```

多列 如果在 order by 子句中指定多个列，则 Adaptive Server 对排序进行嵌套。以下语句对 stores 表中的行进行排序：首先按 stor_id 以降序排序，然后按 payterms （以升序，因为未指定 desc）排序，最后按 country （也是升序）排序。 Adaptive Server 在任何组中首先对空值进行排序。

```
select stor_id, payterms, country
from stores
order by stor_id desc, payterms
```

```
stor_id payterms     country
------- ------------ ------------
8042    Net 30       USA
7896    Net 60       USA
7131    Net 60       USA
7067    Net 30       USA
7066    Net 30       USA
6380    Net 60       USA
5023    Net 60       USA

(7 rows affected)
```

列位置号 可使用选择列表中列的**位置号**代替列名。可以将列名和选择列表号混合使用。以下两个语句产生的结果都与前面语句相同。

```
select pub_id, type, title_id
from titles
order by 1 desc, 2, 3
```

```
select pub_id, type, title_id
from titles
order by 1 desc, type, 3
```

SQL 的大多数版本都要求 order by 项出现在选择列表中，但 Transact-SQL 没有此类限制。即使该列未出现在选择列表中，也可以按 title 对前面查询的结果进行排序。

注释　不能对 text、 unitext 或 image 列使用 order by。

集合函数　在 order by 子句中允许使用集合函数，但是它们必须遵循某种语法，以避免哪个 order by 列取决于 union 表达式的不确定性。但是，联合中的列名从联合的第一（最左边）部分派生而来。这意味着 order by 子句仅使用在联合的第一部分中指定的列名。

例如，以下语法有效，因为清楚地指定了由 order by 键标识的列：

```
select id, min(id) from tab
union
select id, max(id) from tab
ORDER BY 2
```

但是，下例产生错误消息：

```
select id+2 from sysobjects
union
select id+1 from sysobjects
order by id+1
------------
Msg 104, Level 15, State1:
Line 3:
Order-by items must appear in the select list if the
statement contains set operators.
```

如果通过交换 union 两边的内容重新安排语句，则语句会正确执行：

```
select id+1 from sysobjects
union
select id+2 from sysobjects
order by id+1
```

空值　使用 order by 可以将空值排在其它所有值之前。

混合大小写类型的数据　order by 子句对混合大小写类型的数据的影响取决于 Adaptive Server 上安装的排序顺序。基本选项为二进制、字典顺序和不区分大小写。 sp_helpsort 显示服务器的排序顺序。请参见 《系统管理指南：卷 1》中的第 9 章 “配置字符集、排序顺序和语言”。

限制　Adaptive Server 不允许 order by 列表中有子查询或变量。

不能对 text、 unitext 或 image 列使用 order by。

order by 和 group by

可以使用 order by 子句以特定方式对 group by 的结果进行排序。

将 order by 子句置于 group by 子句之后。例如，若要得出每种类型书籍的平均价格，并按平均价格对结果进行排序，则语句为：

```
select type, avg(price)
from titles
group by type
order by avg(price)

type
---------- ------------
UNDECIDED           NULL
mod_cook          11.49
psychology        13.50
business          13.73
trad_cook         15.96
popular_comp      21.48

(6 rows affected)
```

与 select distinct 一起使用的 order by 和 group by

如果 select 列表中没有 order by 或 group by 列，则带有 order by 或 group by 的 select distinct 查询可以返回重复值。例如：

```
select distinct pub_id
from titles
order by type

pub_id
------
0877
0736
1389
0877
1389
0736
0877
0877

(8 rows affected)
```

如果查询的 order by 或 group by 子句包括不在 select 列表中的列，Adaptive Server 就会将这些列作为隐藏列添加在正处理的列中。order by 或 group by 子句中列出的列包括在对不同行的测试中。为符合 ANSI 标准，请在 select 列表中包括 order by 或 group by 列。例如：

```
select distinct pub_id, type
from titles
order by type

pub_id type
------ ------------
0877   UNDECIDED
0736   business
1389   business
0877   mod_cook
1389   popular_comp
0736   psychology
0877   psychology
0877   trad_cook

(8 rows affected)
```

汇总数据组：*compute* 子句

compute 子句是 Transact-SQL 扩展。将其与行集合一起使用以生成显示分组汇总的小计的报告。此类报告通常由报告发生器生成，称为控制中断报告，因为汇总值出现在 compute 子句中所指定的分组控制 （“中断”）下的报告中。

这些汇总值作为附加行出现在查询结果中，这与 group by 子句的集合结果不同，后者作为新列出现。

使用 compute 子句可以用单个 select 语句查看明细行和汇总行。可以计算子群的汇总值，还可以计算同一组的多个行集合 （请参见第 106 页的 “行集合和 compute”）。

compute 的一般语法为：

```
compute row_aggregate(column_name)
    [, row_aggregate(column_name)]...
    [by column_name [, column_name]...]
```

可以用于 compute 的行集合为 sum、 avg、 min、 max 和 count 以及 count_big。sum 和 avg 只能用于数值列。与 order by 子句不同，不能使用选择列表中列的位置编号来代替列名。

注释 不能在 compute 子句中使用 text、 unitext 或 image 列。

系统测试可能会因查询的 compute 子句中集合过多而失败。每个 compute 子句最多可以容纳 127 个集合，而如果 compute 子句包含的集合超过 127 个，则系统在您尝试执行查询时会生成错误消息。

由于 avg() 集合实际上是 sum() 集合和 count() 集合的组合，因此在向 127 这一限制值进行计数时，每个 avg() 集合以两个集合计。

以下是两个查询及其结果。第一个查询使用 group by 和集合。第二个查询使用 compute 和行集合。请注意结果的区别。

```
select type, sum(price), sum(advance)
from titles
group by type

type
------------   -------    ----------
UNDECIDED         NULL         NULL
business         54.92    25,125.00
mod_cook         22.98    15,000.00
popular_comp     42.95    15,000.00
psychology       67.52    21,275.00
trad_cook        47.89    19,000.00

(6 rows affected)

select type, price, advance
from titles
order by type
compute sum(price), sum(advance) by type

type         price                     advance
------------ ------------------------ --------
UNDECIDED    NULL                      NULL

Compute Result:
---------------------- -------------------------
                     NULL                      NULL

type         price                advance
------------ -------------------- ----------
business     2.99                 10,125.00
```

```
business    11.95                   5,000.00
business    19.99                   5,000.00
business    19.99                   5,000.00

Compute Result:
----------------------- ------------------------
                  54.92                25,125.00
type         price                     advance
------------ ----------------------- ---------
mod_cook      2.99                    15,000.00
mod_cook     19.99                         0.00

Compute Result:
----------------------- ------------------------
                  22.98                15,000.00

type          price               advance
------------- ------------------- ------------
popular_comp  NULL                        NULL
popular_comp  20.00                   8,000.00
popular_comp  22.95                   7,000.00

Compute Result:
----------------------- ------------------------
                  42.95                15,000.00

type         price                     advance
------------ ----------------------- --------
psychology    7.00                     6,000.00
psychology    7.99                     4,000.00
psychology   10.95                     2,275.00
psychology   19.99                     2,000.00
psychology   21.59                     7,000.00

Compute Result:
----------------------- ------------------------
                  67.52                21,275.00

type         price                     advance
------------ ----------------------- --------
trad_cook    11.95                    4,000.00
trad_cook    14.99                    8,000.00
trad_cook    20.95                    7,000.00

Compute Result:
----------------------- ------------------------
```

```
                                47.89                19,000.00

(24 rows affected)
```

将每个汇总值视为一行。

行集合和 *compute*

在表 3-2 中列出了与 compute 一起使用的行集合：

表 3-2：集合如何用于 compute 语句

行集合	结果
sum	表达式中值的总和
avg	表达式中值的平均值
max	表达式中的最大值
min	表达式中的最小值
count	integer 形式的选定行数
count_big	bigint 形式的选定行数

这些行集合与可以用于 group by 的集合相同，不同的是没有等同于 count(*) 的行集合函数。若要得出由 group by 和 count(*) 生成的汇总信息，请使用不带 by 关键字的 compute 子句。

compute 子句的规则

- Adaptive Server 不允许将 distinct 关键字用于行集合。
- compute 子句中的列必须出现在选择列表中。
- 不能在同一语句中将 select into （请参见第 7 章 “创建数据库和表”）用作 compute 子句，因为包括 compute 的语句不生成常规行。
- 不能在 insert 语句中的 select 语句中使用 compute 子句，原因同上：包括 compute 的语句不生成常规行。
- 如果使用带有 by 关键字的 compute，则还必须使用 order by 子句。在 by 之后列出的列必须与在 order by 后面列出的列相同或为其子集，且必须采用相同的从左向右的顺序，以相同表达式开始，且不跳过任何表达式。

  例如，假设 order by 子句为：

  ```
  order by a, b, c
  ```

compute 子句可以是任何或所有以下形式：

```
compute row_aggregate (column_name) by a, b, c
compute row_aggregate (column_name) by a, b
compute row_aggregate (column_name) by a
```

compute 子句不能是以下任何形式：

```
compute row_aggregate (column_name) by b, c
compute row_aggregate (column_name) by a, c
compute row_aggregate (column_name) by c
```

必须在 order by 子句中使用列名或表达式；不能按列标题进行排序。

- 可以使用不带 by 的 compute 关键字生成总和、总计数等等。如果使用不带 by 的 compute 关键字，则 order by 是可选的。第 110 页的“生成总和：不带 by 的 compute”中对不带 by 的 compute 关键字进行了讨论。

在 *compute* 后指定多列

在 by 关键字后列出多列会将组分成子群，并将指定的行集合应用于每个分组级别，从而影响查询。例如，以下查询得出每个出版社的心理学书籍的价格之和：

```
select type, pub_id, price
from titles
where type = "psychology"
order by type, pub_id, price
compute sum(price) by type, pub_id

type         pub_id  price
------------ ------- -------------
psychology     0736           7.00
psychology     0736           7.99
psychology     0736          10.95
psychology     0736          19.99
Compute Result:
---------------
          45.93

type         pub_id  price
------------ ------- -------------
 psychology   0877          21.59
```

```
Compute Result:
---------------
          21.59

(7 rows affected)
```

使用多个 *compute* 子句

通过包括多个 compute 子句，可以在同一语句中使用不同的集合。以下查询与前面查询类似，但按出版社将心理学书籍的价格之和相加：

```
select type, pub_id, price
from titles
where type = "psychology"
order by type, pub_id, price
compute sum(price) by type, pub_id
compute sum(price) by type
```

```
type        pub_id  price
----------- ------- --------------
psychology    0736           7.00
psychology    0736           7.99
psychology    0736          10.95
psychology    0736          19.99

Compute Result:
---------------
          45.93

type        pub_id  price
 ---------- ------- --------------
 psychology    0877         21.59

Compute Result:
---------------
          21.59

Compute Result:
---------------
           67.52

(8 rows affected)
```

将一个集合应用于多列

一个 compute 子句可以将同一集合应用于若干列。以下查询分别得出每种类型烹调书籍的价格和预付款之和：

```
select type, price, advance
from titles
where type like "%cook"
order by type
compute sum(price), sum(advance) by type

type      price            advance
--------- ---------------- ---------------
mod_cook              2.99       15,000.00
mod_cook             19.99            0.00

Compute Result:
--------------- ---------------
          22.98       15,000.00

type      price            advance
--------- ---------------- ---------------
trad_cook            11.95        4,000.00
trad_cook            14.99        8,000.00
trad_cook            20.95        7,000.00

Compute Result:
--------------- ---------------
          47.89       19,000.00

(7 rows affected)
```

请记住，集合所适用的列也必须在选择列表中。

在同一 *compute* 子句中使用不同集合

可以在同一 compute 子句中使用不同集合：

```
select type, pub_id, price
from titles
where type like "%cook"
order by type, pub_id
compute sum(price), max(pub_id) by type
```

```
type         pub_id  price
------------ ------- --------------
mod_cook     0877             2.99
mod_cook     0877            19.99

Compute Result:
--------------- ----
          22.98 0877

type         pub_id  price
------------ ------- --------------
trad_cook    0877            11.95
trad_cook    0877            14.99
trad_cook    0877            20.95

Compute Result:
--------------- ----
          47.89 0877

(7 rows affected)
```

生成总和：不带 *by* 的 *compute*

可以使用不带 by 的 compute 关键字生成总和、总计数，等等。

以下语句得出价格高于 $20 的所有类型书籍的价格和预付款的总和：

```
select type, price, advance
from titles
where price > $20
compute sum(price), sum(advance)
```

```
type          price            advance
------------ ----------------  -------------
popular_comp            22.95      7,000.00
psychology              21.59      7,000.00
trad_cook               20.95      7,000.00
```

```
Compute Result:
--------------- ---------
          65.49 21,000.00

(4 rows affected)
```

可以在同一查询中使用带 by 的 compute 和不带 by 的 compute。以下查询按类型得出价格和预付款之和，然后计算所有类型书籍的价格和预付款的总和。

```
select type, price, advance
from titles
where type like "%cook"
order by type
compute sum(price), sum(advance) by type
compute sum(price), sum(advance)

type         price                advance
-----------  -----------------    ------------
mod_ cook                 2.99       15,000.00
mod_cook                 19.99            0.00

Compute Result:
--------------- ---------
          22.98 15,000.00

type         price                advance
-----------  -----------------    ------------
trad_cook                11.95        4,000.00
trad_cook                14.99        8,000.00
trad_cook                20.95        7,000.00

Compute Result:
--------------- ---------
          47.89 19,000.00

Compute Result:
--------------- ---------
          70.87 34,000.00

(8 rows affected)
```

组合查询：*union* 运算符

union 运算符将两个或更多个查询的结果组合为一个结果集。使用 union 的 Transact-SQL 扩展可以：

- 在 insert 语句的 select 子句中使用 union。
- 当 select 语句中存在 union 时，在 select 语句的 order by 子句中指定新的列标题。

union 运算符的语法如下：

query1
[union [all] *queryN*] ...
[order by clause]
[compute clause]

其中：

- *query1* 为：

```
select select_list
[into clause]
[from clause]
[where clause]
[group by clause]
[having clause]
```

- *queryN* 为：

```
select select_list
[from clause]
[where clause]
[group by clause]
[having clause]
```

例如，假设有包含如下所示数据的两个表：

图 3-3：合并查询的联合

表 T1	
a char(4)	b int
abc def ghi	1 2 3

表 T2	
a char(4)	b int
ghi jkl mno	3 4 5

图 3-1 显示两个表 T1 和 T2。 T1 显示两列 “a， char(4)” 和 “b， char(4)”。 T2 包含两列：“a” char(4) 和 “b” int。每个表有三行：在 T1 中，第 1 行在 “a” 列中显示 “abc”，在 “b” 列中显示 “1”。 T1 的第 2 行在 “a” 列中显示 “def”，在 “b” 列中显示 “2”。第 3 行在 “a” 列中显示 “ghi”，在 “b int” 列中显示 “3”。表 T4 的第 1 行在 “a” 列中显示 “ghi”，在 “b” 列中显示 “1”；第 2 行在 “a” 列中显示 “jkl”，在 “b” 列中显示 “2”；第 3 行在 “a” 列中显示 “mno”，在 “b(int)” 列中显示 “3”。以下查询在两个表之间创建 union：

```
create table T1 (a char(4), b int)
insert T1 values ("abc", 1)
insert T1 values ("def", 2)
insert T1 values ("ghi", 3)
create table T2 (a char(4), b int)
insert T2 values ("ghi", 3)
insert T2 values ("jkl", 4)
insert T2 values ("mno", 5)
select * from T1
union
select * from T2

a     b
----  ---------
abc           1
def           2
ghi           3
jkl           4
mno           5

(5 rows affected)
```

缺省情况下， union 运算符从结果集中删除重复行。使用 all 选项以包括重复行。还请注意，结果集中的列与 T1 中的列名称相同。可以在 Transact-SQL 语句中使用任意数量的 union 运算符。例如：

```
x union y union z
```

缺省情况下， Adaptive Server 从左向右计算包含 union 运算符的语句的值。可以使用小括号指定不同的计算顺序。

例如，以下两个表达式不等同：

```
x union all (y union z)

(x union all y) union z
```

在第一个表达式中，删除了 *y* 与 *z* 的联合中的重复值。然后，在该集合与 *x* 的联合中，*不*删除重复值。在第二个表达式中， *x* 与 *y* 的联合中包括重复值，但在随后与 *z* 的后续联合中删除了它们； all 不影响此语句的最终结果。

union 查询的准则

使用 union 语句时：

- union 语句中的所有选择列表必须具有相同数量的表达式 （如列名、算术表达式和集合函数）。以下语句无效，因为第一个选择列表比第二个选择列表长：

```
create table stores_east
(stor_id char(4) not null,
stor_name varchar(40) null,
stor_address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null,
payterms varchar(12) null)
select stor_id, city, state from stores
union
select stor_id, city from stores_east
drop table stores_east
```

- 所有表中的相应列或单个查询中使用的列的任何子集必须具有相同的数据类型，或两种数据类型之间必须可以进行隐式数据转换，否则应该提供显式转换。例如，在 char 数据类型的列与 int 数据类型的列之间不能执行 union 运算，除非提供显式转换。但是，在 money 数据类型的列与 int 数据类型的列之间可以执行 union 运算。请参见《参考手册：命令》中的 union 和 《参考手册：构件块》中的第 1 章“系统数据类型和用户定义的数据类型”。
- 必须以相同顺序将相应的列置于 union 语句的各个查询中，因为 union 按照查询中给定的顺序对各列逐一进行比较。例如，假设有以下表：

图 3-4：比较列的联合

表 T3	
a	b
int	char(4)
1	abc
2	def
3	ghi

表 T4	
a	b
char(4)	int
abc	1
def	2
ghi	3

图 T3 显示两个表 T3 和 T4。 T3 有两列：“a” int 和 “b” char(4)。T4 包含两列：“a” char(4) 和 “b” int。每个表有三行：第 1 行在 “a” 列中显示 “1”，在 “b” 列中显示 “abc”。第 2 行在 “a” 列中显示 “2”，在 “b” 列中显示 “def”。第 3 行在 “a” 列中显示 “3”，在 “b char” 列中显示 “ghi”。表 T4 的第 1 行在 “a” 列中显示 “abc”，在 “b” 列中显示 “1”；第 2 行在 “a” 列中显示 “def”，在 “b” 列中显示 “2”；第 3 行在 “a” 列中显示 “ghi”，在 “b(int)” 列中显示 “3”。

输入此查询：

```
select a, b from T3
union
select b, a from T4
```

此查询生成：

```
a           b
---------   ---
          1 abc
          2 def
```

```
        3  ghi

(3 rows affected)
```

但是，以下查询会导致错误消息，因为相应列的数据类型不兼容：

```
select a, b from T3
union
select a, b from T4
drop table T3
drop table T4
```

在 union 语句中组合使用不同 （但兼容）的数据类型 （如 float 和 int）时， Adaptive Server 将其转换为精度最高的数据类型。

- Adaptive Server 从 union 语句中*第一个*单个查询中提取由 union 生成的表中的列名。因此，要为结果集定义新的列标题，请在第一个查询中这样做。此外，若要按新名称引用结果集中的列 （例如在 order by 语句中），请在第一个 select 语句中以这种方式引用它。

  以下查询是正确的：

  ```
  select Cities = city from stores
  union
  select city from authors
  order by Cities
  ```

将 *union* 与其它 Transact-SQL 命令一起使用

将 union 语句与其它 Transact-SQL 命令一起使用时：

- union 语句中的第一个查询可以包含 into 子句，该子句创建用于保存最终结果集的表。例如，以下语句创建名为 results 的表，该表包含表 publishers、 stores 和 salesdetail 的联合：

  ```
  use mastersp_dboption pubs2, "select into", true
  use pubs2
  checkpoint
  select pub_id, pub_name, city into results
  from publishers
  union
  select stor_id, stor_name, city from stores
  union
  select stor_id, title_id, ord_num from salesdetail
  ```

  只能在第一个查询中使用 into 子句；如果它出现在其它任何位置，都会显示错误消息。

- 只能在 union 语句的末尾使用 order by 和 compute 子句，以定义最终结果的顺序或计算汇总值。不能在组成 union 语句的单个查询中使用它们。特别是，不能在 insert...select 语句中使用 compute 子句。
- 只能在单个查询中使用 group by 和 having 子句；不能使用它们影响最终结果集。
- 还可以在 insert 语句中使用 union 运算符。例如：

```
create table tour (city varchar(20), state char(2))
insert into tour
    select city, state from stores
    union
    select city, state from authors
drop table tour
```

- 从 Adaptive Server 版本 12.5 开始，可以在 create view 语句中使用 union 运算符。但是，如果使用 Adaptive Server 的早期版本，则不能在 create view 语句中使用 union 运算符。
- 不能对 text 和 image 列使用 union 运算符。
- 不能在涉及 union 运算符的语句中使用 for browse 子句。

第 4 章

连接：从若干表中检索数据

连接操作通过以下方法对两个或多个表 （或视图）进行比较：指定各个表的列，逐行比较这些列中的值，然后链接具有匹配值的行。然后，它在新表中显示结果。连接中指定的表可在同一数据库或不同数据库中。

可声明多个连接作为子查询，它们也可包括两个或多个表。请参见第 5 章 “子查询：在其它查询中使用查询”。

如果启用了 “组件集成服务”，则可执行通过远程服务器的连接。有关详细信息，请参见 《组件集成服务用户指南》。

连接如何工作

连接两个或多个表时，被比较的列必须具有相似值 （即，使用相同或相似数据类型的值）。

有几种类型的连接，如，等值连接、自然连接和外连接。最常见的是基于等同性的等值连接。以下连接查找居住在同一城市的作者姓名和出版社名称：

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city = publishers.city
au_fname      au_lname      pub_name
--------      --------      --------------------
Cheryl        Carson        Algodata Infosystems
Abraham       Bennet        Algodata Infosystems

(2 rows affected)
```

因为查询得到两个单独的表 （publishers 和 authors）中包含的信息，所以需要一个连接来检索所需信息。下列语句使用 city 列作为链接，将 publishers 和 authors 表连接在一起：

```
where authors.city = publishers.city
```

连接语法

可将连接嵌入 select、 update、 insert、 delete 或子查询中。其它连接限制条件和子句可遵循连接条件。连接使用如下语法：

start of select, update, insert, delete, or subquery
from {*table_list* | *view_list*}
where [not]
[*table_name*.| *view_name*.]*column_name join_operator*
[*table_name*.| *view_name*.]*column_name*
[{and | or} [not]
[*table_name*.|*view_name*.]*column_name join_operator*
[*table_name*.|*view_name*.]*column_name*]...

End of select, update, insert, delete, or subquery

连接和关系模型

连接操作是数据库管理的关系模型的特点。除其它任何功能外，连接使关系数据库管理系统区别于其它类型的数据库管理系统。

在通常称为网络和层次系统的结构化数据库管理系统中，数据值间的关系被预定义。建立数据库后，在数据中进行有关非预期关系的查询将变得很困难。

在关系数据库管理系统中，数据值间的关系在数据库定义中保持未声明状态。处理数据时 （当**查询**数据库而不是创建它时），数据值间的关系成为显式状态。可提出想到的有关存储在数据库中数据的任何问题，而不管该数据库建立时的用途。

根据称为**规范化规则**的优良数据库设计规则，每个表都应描述一类实体 — 人员、地点、事件或物品。这就是要比较有关两类或多类实体的信息时需要连接操作的原因。不同表中存储的数据间关系通过使其连接来发现。

此规则的必然结果是：连接操作在将新类型的数据添加到数据库方面获得无限的灵活性。始终可创建包含有关不同类型实体的数据的新表。如果新表某字段上具有的值与现有一个或多个表中某些字段上的值相似，可通过连接将其链接到其它表上。

如何使连接结构化

连接语句如同选择语句一样，都以关键字 select 开始。 select 关键字后命名的列是要按其所需顺序包括在查询结果中的列。先前示例指定了包含 authors 表中作者姓名和 publishers 表中出版社名称的列：

```
select au_fname, au_lname, pub_name
from authors, publishers
```

不必通过表名限定列 au_fname、 au_lname 和 pub_name，因为有关列属于哪个表很明确。但用于连接比较的 city 列确实需要限定，因为 authors 和 publishers 表中都有此名称的列：

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city = publishers.city
```

虽然每个 city 列都不在结果中输出，但 Adaptive Server 需要表名来执行比较。

若要指定将查询涉及表的所有列都包括在结果中，请和 select 一起使用星号 (*)。例如，要在前述连接查询中包括 authors 和 publishers 中的所有列，则语句为：

```
select *
from authors, publishers
where authors.city = publishers.city
au_id       au_lname au_fname phone        address
city       state postalcode contract pub_id pub_name
city       state
----------- -------- -------- ------------ --------------------
---------- ----- ---------- -------- ------ --------------------
---------- -----
238-95-7766 Carson   Cheryl   415 548-7723 589 Darwin Ln.
Berkeley   CA    94705      1        1389   Algodata Infosystems
Berkeley   CA
409-56-7008 Bennet   Abraham  415 658-9932 223 Bateman St
Berkeley   CA    94705      1        1389   Algodata Infosystems
Berkeley   CA

(2 rows affected)
```

显示器显示每个表总计为 2 行 13 列。因为行宽的原因，每行占用显示器上多个水平线。只要使用 “*”，结果中的列就按创建该表的 create 语句中规定的顺序显示。

连接的选择列表和结果不需要包括被连接的两个表中的列。例如，要查找与一个出版社位于同一城市的作者姓名，则此查询不需要包括 publishers 中的任何列：

```
select au_lname, au_fname
from authors, publishers
where authors.city = publishers.city
```

如同任何 select 语句一样，选择列表中的列名和 from 子句中的表名必须由逗号分隔。

from 子句

使用 from 子句指定要连接的表和视图。这是表示 Adaptive Server 需要连接的子句。可按任何顺序列出表或视图。仅当使用 select * 指定选择列表时，表的顺序影响显示结果。

一个查询最多可以引用 50 个表和 46 个工作表（如由集合函数创建的那些表）。 50 个表的限制包括：

- from 子句中列出的表 （或表上的视图）。
- 对同一个表的多个引用的每个实例 （自连接）
- 在子查询中引用的表
- 用 into 创建的表
- from 子句中列出的视图所引用的基表

以下示例连接 titles 和 publishers 表中的列，并将在 California 出版的所有书籍的价格加倍：

```
begin tran
update titles
  set price = price * 2
  from titles, publishers
  where titles.pub_id = publishers.pub_id
  and publishers.state = "CA"
rollback tran
```

有关涉及多于两个表或视图的连接的信息，请参见第 134 页的 “连接两个以上的表”。

如第 2 章 “查询：从表中选择数据”所述，表名或视图名可由所有者和数据库名称限定，并且为了方便可给出相关名。例如：

```
select au_lname, au_fname
from pubs2.blue.authors, pubs2.blue.publishers
where authors.city = publishers.city
```

可按与表完全相同的方式连接视图，并且只要使用表，就要使用视图。第 12 章 “视图：限制访问数据”论述各个视图；本章仅使用其示例中的表。

where 子句

使用 where 子句可以确定结果中包括的行。 where 指定在 from 子句中指定的表和视图间的连接。如果列名所属的表或视图存在歧义，则限定列名。例如：

```
where authors.city = publishers.city
```

此 where 子句给出要连接的列名 （如有必要，由表名限定）和连接运算符 — 通常为等号，有时为 “大于号”或 “小于号”。有关 where 子句语法的详细信息，请参见第 2 章 “查询：从表中选择数据”。

注释 如果省略连接的 where 子句，将生成意外结果。如果没有 where 子句，则所论述的任何连接查询目前将生成 69 行，而不是 2 行。第 126 页的 “如何处理连接”将解释此结果。

除链接不同表中的列的条件外，连接语句中的 where 子句还可包括其它条件。也就是说，可在同一个 SQL 语句中包括连接操作和 select 操作。有关示例，请参见第 126 页的 “如何处理连接”。

连接运算符

基于等同性匹配列的连接称为**等值连接**。**等值连接**的更精确定义在第 126 页的 “等值连接和自然连接”中给出，并给出不是基于等同性的连接的示例。

等值连接使用如下比较运算符：

表 4-1：连接运算符

运算符	含义
=	等于
>	大于
>=	大于或等于
<	小于
<=	小于或等于
!=	不等于
!>	小于或等于
!<	大于或等于

使用关系运算符的连接合称为 **theta 连接**。用于**外连接**的另一组连接运算符也将在本章后面详细讨论。外连接运算符是 Transact-SQL 扩展，如表 4-2 中所示：

表 4-2：外连接运算符

运算符	操作
*=	结果中包括第一个表中的所有行，而不仅是连接列匹配的行。
=*	结果中包括第二个表中的所有行，而不仅是连接列匹配的行。

连接列中的数据类型

被连接的列必须具有相同或兼容的数据类型。比较其数据类型无法隐式转换的列时，使用 convert 函数。虽然被连接的列经常具有相同的名称，但它们不需具有相同名称。

如果与用于连接的数据类型兼容， Adaptive Server 自动将其转换。例如， Adaptive Server 在任何数值类型列 （bigint、 int、 smallint、 tinyint、 unsigned bigint、 unsigned int、 unsigned smallint、 decimal 或 float）中和在任何字符类型和日期类型列 （char、 varchar、 unichar、 univarchar、 nchar、 nvarchar、 datetime、 date 和 time）中进行转换。有关数据类型转换的详细信息，请参见第 16 章 “在查询中使用内置函数”和 《参考手册：构件块》中的第 1 章 “系统数据类型和用户定义的数据类型”。

连接和 *text* 与 *image* 列

对于包含 text 或 image 值的列，无法使用连接。然而，可使用 where 子句比较两个表的文本列的长度。例如：

```
where datalength(textab_1.textcol) >
datalength(textab_2.textcol)
```

如何处理连接

知道如何处理连接有助于对它们进行了解，并可在错误声明了连接或获得意外结果时查出原因。本节用概念术语描述连接的处理。 Adaptive Server 使用的实际过程更为复杂。

从概念上讲，处理连接的第一步是形成表的**笛卡儿乘积**，即每个表中各行的所有可能组合。在两个表的笛卡儿乘积中，行数等于第一个表中的行数乘第二个表中的行数。

authors 表和 publishers 表的笛卡儿乘积为 69 （23 个作者乘以 3 个出版社）。可使用任何查询查看笛卡儿乘积，该查询包括选择列表中多个表的列， from 子句中的多个表，并且没有 where 子句。例如，如果在任何先前示例使用的连接中，省略 where 子句，则 Adaptive Server 将 23 个作者与 3 个出版者组合，并返回全部 69 行。

笛卡儿乘积不包含任何特别有用的信息。实际上，这完全是误解，因为它暗示数据库中的每个作者都与该数据库中的每个出版社有关系 — 这完全不正确。

这是必须在连接中包括 where 子句的原因，该子句指定了要匹配的列和匹配基础。它也可能包括其它限制条件。 Adaptive Server 形成笛卡儿乘积后，它通过使用 where 子句中的条件消除了不满足连接的行。

例如，所引用示例 （authors 表和 publishers 表的笛卡儿乘积）中的 where 子句从结果中消除了作者与出版社不在同一城市的所有行：

```
where authors.city = publishers.city
```

等值连接和自然连接

基于等于 (=) 的连接称为**等值连接**。等值连接比较被连接列中的值是否相等，然后在结果中包括所连接表中的所有列。

以下最初的查询是等值连接的一个示例：

```
select *
from authors, publishers
where authors.city = publishers.city
```

在该语句的结果中， city 列出现两次。按照定义，等值连接的结果包含两个相同列。因为通常重复相同信息没有意义，可通过重述查询消除这些列之一。此结果称为**自然连接**。

在 city 列上产生 publishers 和 authors 的自然连接的查询是：

```
select publishers.pub_id, publishers.pub_name,
  publishers.state, authors.*
from publishers, authors
where publishers.city = authors.city
```

publishers.city 列未出现在结果中。

自然连接的另一个示例是：

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city = publishers.city
```

可使用多个连接运算符来连接两个以上的表或连接两个以上的列对。这些“连接表达式”通常使用 and 连接，虽然也可以使用 or。

下面是由 and 连接的两个连接示例。首先按书籍类型排序，列出有关书籍的信息（书籍类型、作者和书名）。多个作者的书籍具有多个列表，每一个与一个作者对应。

```
select type, au_lname, au_fname, title
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
order by type
```

第二个查找位于相同州和城市的作者姓名和出版社名称：

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city = publishers.city
and authors.state = publishers.state
```

使用其它条件的连接

连接查询的 where 子句可包括选择标准以及连接条件。例如，要检索预付款超过 $7500 的所有书籍的书名和出版者，其语句为：

```
select title, pub_name, advance
from titles, publishers
where titles.pub_id = publishers.pub_id
and advance > $7500
```

```
title                          pub_name              advance
-----------------------------  --------------------  ---------
You Can Combat Computer Stress!  New Age Books        10,125.00
The Gourmet Microwave          Binnet & Hardley      15,000.00
Secrets of Silicon Valley      Algodata Infosystems   8,000.00
Sushi, Anyone?                 Binnet & Hardley       8,000.00

(4 rows affected)
```

被连接的列（titles 和 publishers 中的 pub_id）不需要显示在选择列表中，因此不会出现在结果中。

可将任意多个选择标准包括在连接语句中。选择标准顺序和连接条件并不重要。

未基于等同性的连接

两列中连接值的条件不必相等。您可以使用任何其它比较运算符：不等于 (!=)、大于 (>)、小于 (<)、大于或等于 (>=) 以及小于或等于 (<=)。Transact-SQL 也提供运算符 !> 和 !<，它们分别与 <= 和 >= 等效。

下面大于连接的示例按字母顺序，查找居住在 New Age Books 所在的马萨诸塞州之后各州的 New Age 作者。

```
select pub_name, publishers.state,
   au_lname, au_fname, authors.state
from publishers, authors
where authors.state > publishers.state
and pub_name = "New Age Books"
```

```
pub_name         state   au_lname       au_fname         state
-------------    ------  -------------- -----------      -----
New Age Books    MA      Greene         Morningstar      TN
New Age Books    MA      Blotchet-Halls Reginald         OR
New Age Books    MA      del Castillo   Innes            MI
```

```
New Age Books     MA       Panteley        Sylvia           MD
New Age Books     MA       Ringer          Anne             UT
New Age Books     MA       Ringer          Albert           UT

(6 rows affected)
```

下例根据书籍的总销售额，使用 >= 连接和 < 连接查找 roysched 表中正确的 royalty。

```
select t.title_id, t.total_sales, r.royalty
from titles t, roysched r
where t.title_id = r.title_id
and t.total_sales >= r.lorange
and t.total_sales < r.hirange

title_id       total_sales      royalty
--------       -----------      -------
BU1032               4095           10
BU1111               3876           10
BU2075               1872           24
BU7832               4095           10
MC2222               2032           12
MC3021              22246           24
PC1035               8780           16
PC8888               4095           10
PS1372                375           10
PS2091               2045           12
PS2106                111           10
PS3333               4072           10
PS7777               3336           10
TC3218                375           10
TC4203              15096           14
TC7777               4095           10

(16 rows affected)
```

自连接和相关名

在一个表的同一列中比较值的连接称为**自连接**。要区分出现在表中的这两个角色，使用别名或相关名。

例如，可使用自连接找出在 California Oakland 居住的作者，他们住在相同的邮政编码区域。由于此查询包括 authors 表与其自身的连接，所以 authors 表以两种角色出现。若要区分这些角色，可临时在 from 子句中任意为 authors 表给出两个不同的相关名 — 如 au1 和 au2。这些相关名限定此查询其余部分的列名。自连接语句如下所示：

```
select au1.au_fname, au1.au_lname,
au2.au_fname, au2.au_lname
from authors au1, authors au2
where au1.city = "Oakland" and au2.city = "Oakland"
and au1.state = "CA" and au2.state = "CA"
and au1.postalcode = au2.postalcode
au_fname     au_lname    au_fname    au_lname
---------    ----------- --------    --------
Marjorie     Green       Marjorie    Green
Dick         Straight    Dick        Straight
Dick         Straight    Dirk        Stringer
Dick         Straight    Livia       Karsen
Dirk         Stringer    Dick        Straight
Dirk         Stringer    Dirk        Stringer
Dirk         Stringer    Livia       Karsen
Stearns      MacFeather  Stearns     MacFeather
Livia        Karsen      Dick        Straight
Livia        Karsen      Dirk        Stringer
Livia        Karsen      Livia       Karsen

(11 rows affected)
```

在 from 子句中列出别名的顺序与在选择列表中引用它们的顺序相同，如本例所示。根据查询，如果以不同顺序列出它们，则结果可能含糊不清。

要在结果中消除作者匹配自己并且除作者顺序相反外其它都相同的行，可将如下条件加入自连接查询中：

```
select au1.au_fname, au1.au_lname,
au2.au_fname, au2.au_lname
from authors au1, authors au2
where au1.city = "Oakland" and au2.city = "Oakland"
and au1.state = "CA" and au2.state = "CA"
and au1.postalcode = au2.postalcode
and au1.au_id < au2.au_id

au_fname    au_lname     au_fname    au_lname
```

```
---------   -----------  ---------   ---------
Dick        Straight     Dirk        Stringer
Dick        Straight     Livia       Karsen
Dirk        Stringer     Livia       Karsen

(3 rows affected)
```

现在它明显指出 Dick Straight、 Dirk Stringer 和 Livia Karsen 都具有相同邮政编码。

不均等连接

不均等连接在限制由自连接返回的行方面非常有用。在如下示例中，不均等连接和自连接查找不同价格的两种或多种便宜书籍（少于 $15）的类别：

```
select distinct t1.type, t1.price
from titles t1, titles t2
where t1.price < $15
and t2.price < $15
and t1.type = t2.type
and t1.price != t2.price
type         price
----------   -----
business      2.99
business     11.95
psychology    7.00
psychology    7.99
psychology   10.95
trad_cook    11.95
trad_cook    14.99

(7 rows affected)
```

表达式 “not *column_name* = *column_name*” 与 “*column_name* != *column_name*” 等效。

以下示例使用不均等连接，并结合使用自连接。它查找 titleauthor 表中的所有行，该表有两个或多个行具有相同 title_id 但不同的 au_id 编号（即具有多个作者的书籍）。

```
select distinct t1.au_id, t1.title_id
from titleauthor t1, titleauthor t2
where t1.title_id = t2.title_id
```

```
and t1.au_id != t2.au_id
order by t1.title_id
au_id            title_id
-----------      --------
213-46-8915      BU1032
409-56-7008      BU1032
267-41-2394      BU1111
724-80-9391      BU1111
722-51-5454      MC3021
899-46-2035      MC3021
427-17-2319      PC8888
846-92-7186      PC8888
724-80-9391      PS1372
756-30-7391      PS1372
899-46-2035      PS2091
998-72-3567      PS2091
267-41-2394      TC7777
472-27-2349      TC7777
672-71-3249      TC7777

(15 rows affected)
```

对于 titles 中的每本书，下例查找具有相同类型但不同价格的所有其它书籍：

```
select t1.type, t1.title_id, t1.price, t2.title_id,
t2.price
from titles t1, titles t2
where t1.type = t2.type
and t1.price != t2.price
```

解释不均等连接的结果时务必要小心。例如，您可能会认为可以使用不均等连接查找居住在某城市 （无出版社）的作者：

```
select distinct au_lname, authors.city
from publishers, authors
where publishers.city != authors.city
```

然而，此查询却查找所居住城市无出版社 （指全部出版社）的作者。正确 SQL 语句是如下的子查询：

```
select distinct au_lname, authors.city
from publishers, authors
where authors.city not in
(select city from publishers
where authors.city = publishers.city)
```

不均等连接和子查询

有时不均等连接查询限制条件不完全，因而必须由子查询代替。例如，假定要列出居住某城市（无出版者）的作者姓名。为了清晰表达，还要将此查询限定为作者姓氏的首字母为“A”、“B”或“C”。不均等连接查询可以为：

```
select distinct au_lname, authors.city
from publishers, authors
where au_lname like "[ABC]%"
and publishers.city != authors.city
```

结果不是所提问题的答案：

```
au_lname             city
-----------------    ------------
Bennet               Berkeley
Carson               Berkeley
Blotchet-Halls       Corvallis

(3 rows affected)
```

系统解释这种形式的 SQL 语句的含义：“查找所居住城市有*某个*出版社的作者的姓名。”根据姓氏排除的所有作者都符合条件，包括居住在 Berkeley 的作者， Berkeley 是出版社 Algodata Infosystems 的所在地。

这种情况下，系统处理连接的方式（在评估其它条件之前首先查找每个符合条件的组合）导致此查询返回意外结果。必须使用子查询获得所需结果。子查询首先消除不合格的行，然后执行剩余限制条件。

下面是正确语句：

```
select distinct au_lname, city
from authors
where au_lname like "[ABC]%"
and city not in
(select city from publishers
where authors.city = publishers.city)
```

现在，可以得到所需的结果：

```
au_lname             city
-------------        ------------
Blotchet-Halls       Corvallis

(1 row affected)
```

有关子查询的详细信息，请参见第 5 章“子查询：在其它查询中使用查询”。

连接两个以上的表

pubs2 的 titleauthor 表提供一个很好的示例，有助于理解连接两个以上的表非常有帮助的情况。若要查找特定类型的所有书籍书名及其作者姓名，请使用：

```
select au_lname, au_fname, title
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
and titles.type = "trad_cook"
au_lname       au_fname     title
-------------- -----------  ------------------------
Panteley       Sylvia       Onions, Leeks, and Garlic: Cooking
                            Secrets of the Mediterranean
Blotchet-Halls Reginald     Fifty Years in Buckingham Palace
                            Kitchens
O'Leary        Michael      Sushi, Anyone?
Gringlesby     Burt         Sushi, Anyone?
Yokomoto       Akiko        Sushi, Anyone?

      (5 rows affected)
```

from 子句中的表之一 titleauthor 不向结果输出任何列。被连接的任何列（au_id 和 title_id）都未出现在结果中。虽然如此，仅在使用 titleauthor 作为中间表时才可以实现此连接。

在相同语句中也可连接两个以上的列对。例如，下面一个查询，显示 title_id 和它的总销售额及它们所在范围，以及版税：

```
select titles.title_id, total_sales, lorange, hirange,
royalty
from titles, roysched
where titles.title_id = roysched.title_id
and total_sales >= lorange
and total_sales < hirange

title_id    total_sales   lorange hirange  royalty
--------    -----------   ------- -------  -------
 BU1032            4095         0    5000       10
 BU1111            3876         0    4000       10
 BU2075           18722     14001   50000       24
 BU7832            4095         0    5000       10
 MC2222            2032      2001    4000       12
 MC3021            2224     12001   50000       24
 PC1035            8780      4001   10000       16
 PC8888            4095         0    5000       10
 PS1372             375         0   10000       10
```

```
 PS2091            2045      1001     5000       12
 PS2106             111         0     2000       10
 PS3333            4072         0     5000       10
 PS7777            3336         0     5000       10
 TC3218             375         0     2000       10
 TC4203           15096      8001    16000       14
 TC7777            4095         0     5000       10

 (16 rows affected)
```

在同一语句中有多个连接运算符，并且连接两个以上的表或连接两个以上的列对时，“连接表达式”几乎总是用 and 连接，如前面示例中所示。但是，也可以使用 or 连接。

外连接

包括所有行而不管是否有匹配行的连接称为**外连接**。 Adaptive Server 支持左外连接和右外连接。例如，以下查询通过 titles 和 titleauthor 表的 title_id 列连接这两个表。

```
select *
from titles, titleauthor
where titles.title_id *= titleauthor.title_id
```

Sybase 支持 Transact-SQL 和 ANSI 外连接。 Transact-SQL 外连接（如先前示例所示）使用 *= 命令表示左外连接，使用 =* 命令表示右外连接。 Transact-SQL 外连接由 Sybase 创建，作为 Transact-SQL 语言的一部分。有关 Transact-SQL 外连接的详细信息，请参见第 152 页的 “Transact-SQL 外连接”。

ANSI 外连接使用关键字 left join 和 right join 分别表示左连接和右连接。 Sybase 执行 ANSI 外连接语法以完全符合 ANSI 标准。有关 ANSI 外连接的详细信息，请参见第 137 页的 “ANSI 内部和外连接”。下面按 ANSI 外连接重写先前示例：

```
select *
from titles left join titleauthor
on titles.title_id = titleauthor.title_id
```

内部表和外部表

术语**外部表**和**内部表**说明外连接中表的位置：

- 在*左连接*中，**外部表**和**内部表**分别是左表和右表。外部表和内部表也分别称为行保留表和空值提供表。
- 在*右连接*中，外部表和内部表分别是右表和左表。

例如，在下面查询中， T1 是外部表， T2 是内部表：

```
T1 left join T2
T2 right join T1
```

或使用 Transact-SQL 语法：

```
T1 *= T2
T2 =* T1
```

外连接限制

如果表是外连接的内部成员，则无法参与外连接子句*和*常规连接子句。以下查询失败，因为 salesdetail 表是外连接和常规连接子句的一部分：

```
select distinct sales.stor_id, stor_name, title
from sales, stores, titles, salesdetail
where qty > 500
and salesdetail.title_id =* titles.title_id
and sales.stor_id = salesdetail.stor_id
and sales.stor_id = stores.stor_id

Msg 303, Level 16, State 1:
Server 'FUSSY', Line 1:
The table 'salesdetail' is an inner member of an outer-
join clause. This is not allowed if the table also
participates in a regular join clause.
```

如果要知道销售超过 500 册书籍的商店名称，必须使用另一个查询。如果提交具有外连接的查询，且带有外连接的内部表中列的限定条件，则可能不会得到预期结果。查询中的限定条件没有限制返回的行数，而是影响包含空值的行。对于不符合限定条件的行，空值显示在那些行的内部表列上。

用于外连接的视图

如果定义带有外连接的视图，然后使用外连接的内部表中列的限定条件查询该视图，则可能不会得到预期结果。查询返回内部表的所有行。不满足限定条件的行在这些行的相应列中显示为空值。

下列规则确定了可通过连接视图对列进行更新的类型：

- 连接视图中不允许使用 delete 语句。
- 在用 with check option 创建的连接视图中不允许使用 insert 语句。
- 在用 with check option 创建的连接视图中，允许使用 update 语句。如果任何受影响的列出现在 where 子句中，或出现在包含来自多个表的列的表达式中，更新将会失败。
- 如果通过连接视图来插入或更新行，所有受影响的列都必须属于同一基表。

ANSI 内部和外连接

下面是连接表的 ANSI 语法：

```
left_table [inner | left [outer] | right [outer]] join right_table
on left_column_name  = right_column_name
```

左表和右表间的连接结果称为**连接表**。连接表在 from 子句中定义。例如：

```
select titles.title_id, title, ord_num, qty
from titles left join salesdetail
on titles.title_id = salesdetail.title_id
title_id title              ord_num                qty
--------------------------- ---------------------- -----
BU1032  The Busy Executive   AX-532-FED-452-2Z7   200
BU1032  The Busy Executive   NF-123-ADS-642-9G3   1000
. . .
TC7777  Sushi, Anyone?       ZD-123-DFG-752-9G8   1500
TC7777  Sushi, Anyone?       XS-135-DER-432-8J2   1090
(118 rows affected)
```

ANSI 连接语法允许编写如下任一项目：

- **内连接**，连接表仅包括满足 on 子句条件的内部表和外部表的行。有关信息，请参见第 140 页的 “ANSI 内连接”。对于不满足 on 子句条件的外部表中的行，包含内连接的查询所得到的结果集中并不包含任何空值行。

- **外连接**，连接表包括外部表中的所有行，不论外部表是否满足 on 子句条件。如果某行不满足 on 子句条件，则内部表的值将作为空值存储在连接表中。 ANSI 外连接中的 where 子句将限制查询结果中的行。有关详细信息，请参见第 143 页的 “ANSI 外连接”。

  注释 也可使用 ANSI 语法连接视图。

Sybase ANSI 连接语法不支持 using 子句。

ANSI 连接的相关名和列引用规则

下列是专用于 ANSI 连接的相关名和列引用规则。有关相关名的详细信息，请参见第 130 页的 “自连接和相关名”。

- 如果表或视图使用引用列或视图的相关名，它必须始终使用同一相关名，而不能使用该表名或视图名。即，不能在查询中用相关名命名表后又使用其表名。以下示例正确使用相关名 t 指定已指定其 pub_id 列的表：

  ```
  select title, t.pub_id, pub_name
  from titles t left join publishers p
  on t.pub_id = p.pub_id
  ```

  然而，以下示例未在查询的 on 子句中使用 titles 表 (t.pub_id) 的相关名，而是错误地使用了表名，因而后来生成错误消息：

  ```
  select title, t.pub_id, pub_name
  from titles t left join publishers p
  on titles.pub_id = p.pub_id
  Msg 107, Level 15, State 1:
  Server 'server_name', Line 1:
  The column prefix 't' does not match with a table
  name or alias name used in the query. Either the
  table is not specified in the FROM clause or it has
  a correlation name which must be used instead.
  ```

- on 子句中指定的限制可引用：
  - 在连接表的引用中指定的列
  - 在 ANSI 连接 （如嵌套连接）中包含的连接表中指定的列
  - 在外部查询块中指定的表的子查询中的相关名

- 在 on 子句中指定的条件*不能*引用*包含*另一 ANSI 连接的 ANSI 连接中引入的列（通常当第二连接与第一连接进行连接生成连接表时）。

  下面是非法列引用生成错误的一个示例：

```
select *
from titles left join titleauthor
on titles.title_id=roysched.title_id    /*join #1*/
left join roysched
on titleauthor.title_id=roysched.title_id    /*join #2*/
where titles.title_id != "PS7777"
```

  第一个左连接无法引用 roysched.title_id 列，因为此列直到第二连接后才被引入。可正确重写此查询为：

```
select *
from titles
left join (titleauthor
left join roysched
on titleauthor.title_id = roysched.title_id) /*join #1*/
on titles.title_id = roysched.title_id      /*join #2*/
where titles.title_id != "PS7777"
```

  另一个示例为：

```
select title, price, titleauthor.au_id, titleauthor.title_id, pub_name,
publishers.city
from roysched, titles
left join titleauthor
on roysched.title_id=titleauthor.title_id
left join authors
on titleauthor.au_id=roysched.au_id, publishers
```

  在此查询中， roysched 表或 publishers 表都不是任意左连接的一部分。因此，任何左连接都无法引用 roysched 或 publishers 表作为其 on 子句条件的一部分。

ANSI 内连接

生成仅包括满足限制条件连接表行的结果集的连接称为**内连接**。连接表中不包括不满足连接限制条件的行。如果要求连接表包括一个表的所有行，而不管它们是否满足限制条件，使用外连接。有关详细信息，请参见第 143 页的 “ANSI 外连接”。

Adaptive Server 支持使用 Transact-SQL 内连接和 ANSI 内连接。使用 Transact-SQL 内连接的查询用逗号分隔被连接的表，并在 where 子句中列出连接比较和限制条件。例如：

```
select au_id, titles.title_id, title, price
from titleauthor, titles
where titleauthor.title_id = titles.title_id
and price > $15
```

有关编写 Transact-SQL 内连接的信息，请参见第 121 页的 “如何使连接结构化”。

ANSI 标准 inner joins 语法为：

```
select select_list
    from table1 inner join table2
        on join_condition
```

例如，下面的示例使用 inner join，等同于上述 Transact-SQL 连接：

```
select au_id, titles.title_id, title, price
from titleauthor inner join titles
on titleauthor.title_id = titles.title_id
and price > 15
au_id         title_id  title                        price
-----------   --------  -------------------------    -----
213-46-8915   BU1032    The Busy Executive's Datab   19.99
409-56-7008   BU1032    The Busy Executive's Datab   19.99
. . .
172-32-1176   PS3333    Prolonged Data Deprivation   19.99
807-91-6654   TC3218    Onions, Leeks, and Garlic:   20.95
(11 rows affected)
```

编写连接的两个方法 ANSI 或 Transact-SQL 等同。例如，下列查询生成的结果集间没有区别：

```
select title_id, pub_name
from titles, publishers
where titles.pub_id = publishers.pub_id
```

和

```
select title_id, pub_name
from titles left join publishers
```

```
on titles.pub_id = publishers.pub_id
```

内连接可以是 update 或 delete 语句的一部分。例如，下列是对于在 California 出版的所有书目价格乘以 1.25 的查询：

```
begin tran
update titles
  set price = price * 1.25
  from titles inner join publishers
  on titles.pub_id = publishers.pub_id
  and publishers.state = "CA"
```

内连接的连接表

ANSI 连接指定在查询中要连接的表或视图。ANSI 连接中指定的表引用构成连接表。例如，以下查询的连接表包括 title、 price、 advance 和 royaltyper 列：

```
select title, price, advance, royaltyper
from titles inner join titleauthor
on titles.title_id = titleauthor.title_id
title          price     advance      royaltyper
-----------    -------   ----------   ----------
The Busy...    19.99     5,000.00     40
The Busy...    19.99     5,000.00     60
. . .
Sushi, A...    14.99     8,000.00     30
Sushi, A...    14.99     8,000.00     40
(25 rows affected)
```

如果连接表用作 ANSI 内连接中的表引用，则它成为**嵌套**内连接。ANSI 嵌套内连接遵循与 ANSI 外连接相同的规则。

查询在联合的每一侧可引用最多 50 个用户表（或 14 个工作表），包括：

- from 子句中列出的基表或视图
- 对同一表的每个相关引用 （自连接）
- 在子查询中引用的表
- 视图或嵌套视图引用的基表
- 用 into 创建的表

ANSI 内连接的 *on* 子句

ANSI 内连接的 on 子句指定表或视图连接时所使用的条件。虽然可在表的任何列上连接，但如果这些列编入索引可能会更好地提高性能。通常必须使用限定符 （表名或相关名）唯一地标识列及其所属表。例如：

```
from titles t left join titleauthor ta
on t.title_id = ta.title_id
```

此 on 子句消除了没有匹配的 title_id 的两个表中的行。有关相关名的详细信息，请参见第 130 页的 “自连接和相关名”。

此 on 子句通常对 ANSI 连接表进行比较，如以下查询的第三行和第四行所示：

```
select title, price, pub_name
from titles inner join publishers
on titles.pub_id = publishers.pub_id
and total_sales > 300
```

在此 on 子句中指定的连接限制条件从连接表中删除销售额不大于 300 的所有行。 on 子句也可指定搜索参数，如此查询的第四行所示。

无论限制条件放置在 on 子句还是 where 子句中， ANSI 内连接对结果集的限制都类似 （除非它们嵌套到外连接中）。即，下列查询生成相同结果集：

```
select stor_name, stor_address, ord_num, qty
from salesdetail inner join stores
on salesdetail.stor_id = stores.stor_id
where qty > 3000
```

和

```
select stor_name, stor_address, ord_num, qty
from salesdetail inner join stores
on salesdetail.stor_id = stores.stor_id
and qty > 3000
```

如果限制条件放置在 where 子句中，查询通常更易读；它明确指示用户结果集中包括哪些连接表的行。

ANSI 外连接

生成包括外部表所有行的连接表的连接 （不管 on 子句是否生成匹配行），则称为**外连接**。内连接和等值连接生成的结果集仅包括在连接子句中存在匹配值的表中的行。然而，有时不但要包括匹配行，而且要包括在第二个表中没有匹配行的一个表中的行。此类连接为外连接。在外连接中，对于外连接的内部表，具有空值的连接表中包括不满足 on 子句条件的行。内部表也称为空值提供成员。

Sybase 建议您的应用程序使用 ANSI 外连接，其原因在于 ANSI 外连接明确指定了是 on 还是 where 子句包含谓词。

本节仅论述 ANSI 外连接；有关 Transact-SQL 外连接的信息，请参见第 152 页的 “Transact-SQL 外连接”。

注释　包含 ANSI 外连接的查询不能包含 Transact-SQL 外连接，反之亦然。然而，具有 ANSI 外连接的查询可引用包含 Transact-SQL 外连接的视图，反之亦然。

ANSI 外连接语法为：

```
select select_list
    from table1 {left | right} [outer] join table2
    on predicate
    [join restriction]
```

左连接将保留在连接子句左侧列出的表引用的所有行；右连接将保留在连接子句右侧列出的表引用的所有行。在左连接中，左表引用称为外部表或行保留表。

以下示例确定居住城市与出版社所在的城市相同的作者：

```
select au_fname, au_lname, pub_name
from authors left join publishers
on authors.city = publishers.city
au_fname   au_lname    pub_name
---------  ----------  -------------
Johnson    White       NULL
Marjorie   Green       NULL
Cheryl     Carson      Algodata Infosystems
. . .
Abraham    Bennet      Algodata Infosystems
. . .
Anne       Ringer      NULL
Albert     Ringer      NULL
(23 rows affected)
```

结果集包括 authors 表中的所有作者。当作者与他们的出版者不居住同一城市时， pub_name 列中的值就为空。只有 Cheryl Carson 和 Abraham Bennet 与他们的出版者居住在同一城市，这两位作者在 pub_name 列中产生非空值。

可通过在 from 子句中调换表位置，将左外连接重写为右外连接。另外，如果 select 语句指定 “select *”，则必须编写所有列名的显式列表，否则结果集中的列可能与重写查询顺序不同。

下面将先前示例重写为右外连接，这样生成与上述左外连接相同的结果集：

```
select au_fname, au_lname, pub_name
from publishers right join authors
on authors.city = publishers.city
```

谓词应放在 *on* 还是放在 *where* 子句？

ANSI 外连接的结果集取决于在 on 还是 where 子句中放置限制条件。on 子句定义连接表的结果集和此连接表具有空值的行； where 子句定义要在结果集中包括的连接表的行。

在连接条件中使用 on 还是 where 子句，取决于要在结果集中包括的内容。下列示例有助于决定将谓词放置在 on 还是 where 子句中。

外部表上的谓词限制

下列查询在 where 子句中放置了外部表的限制条件。因为限制条件应用于外连接的结果，所以它删除条件*非*真的所有行：

```
select title, titles.title_id, price, au_id
from titles left join titleauthor
on titles.title_id = titleauthor.title_id
where titles.price > $20.00

title                title_id price      au_id
-------------------- -------- ---------- -----------------
But Is It User F...PC1035        22.95 238-95-7766
Computer Phobic ...PS1372        21.59 724-80-9391
Computer Phobic ...PS1372        21.59 756-30-7391
Onions, Leeks, a...TC3218        20.95 807-91-6654
(4 rows affected)
```

有四行满足标准并且结果集中仅包括这些行。

然而，如果将外部表的此限制条件移到 on 子句中，则结果集包括满足 on 子句条件的所有行。外部表中不满足条件的行为空扩展行：

```
select title, titles.title_id, price, au_id
from titles left join titleauthor
on titles.title_id = titleauthor.title_id
and titles.price > $20.00
title                  title_id   price   au_id
---------------------  ---------  ------  ---------------
The Busy Executive's   BU1032     19.99   NULL
Cooking with Compute   BU1111     11.95   NULL
You Can Combat Compu   BU2075      2.99   NULL
Straight Talk About    BU7832     19.99   NULL
Silicon Valley Gastro  MC2222     19.99   NULL
The Gourmet Microwave  MC3021      2.99   NULL
The Psychology of Com  MC3026      NULL   NULL
But Is It User Friend  PC1035     22.95   238-95-7766
Secrets of Silicon Va  PC8888     20.00   NULL
Net Etiquette          PC9999      NULL   NULL
Computer Phobic and    PS1372     21.59   724-80-9391
Computer Phobic and    PS1372     21.59   756-30-7391
Is Anger the Enemy?    PS2091     10.95   NULL
Life Without Fear      PS2106      7.00   NULL
Prolonged Data Depri   PS3333     19.99   NULL
Emotional Security:    PS7777      7.99   NULL
Onions, Leeks, and Ga  TC3218     20.95   807-91-6654
Fifty Years in Buckin  TC4203     11.95   NULL
Sushi, Anyone?         TC7777     14.99   NULL
(19 rows affected)
```

将限制条件移动到 on 子句向结果集中增加了 15 个空值行。

通常，如果查询使用外部表的限制条件，并且想要结果集仅删除限制条件为假的行，可能应在 where 子句中放置限制条件，以限制结果集的行。如果外部表谓词在 on 子句上，则它们不用于索引键。

在 on 还是 where 子句中放置外部表的限制条件最终取决于需要查询返回的信息。如果只要结果集仅包括限制条件为真的行，应在 where 子句中放置限制条件。然而，如果需要结果集包括外部表的所有行，而不管它们是否满足限制条件，应在 on 子句中放置限制条件。

内部表的限制

以下查询在 where 子句中包括内部表的限制条件：

```
select title, titles.title_id, titles.price, au_id
from titleauthor left join titles
on titles.title_id = titleauthor.title_id
where titles.price > $20.00

title            title_id   price    au_id
```

```
-------------    --------   -----    -----------
But Is It U...   PC1035     22.95    238-95-7766
Computer Ph...   PS1372     21.59    724-80-9391
Computer Ph...   PS1372     21.59    756-30-7391
Onions, Lee...   TC3218     20.95    807-91-6654
(4 rows affected)
```

因为 where 子句的限制条件在连接*后*应用于结果集，所以它删除限制条件*非*真的所有行。换句话说， where 子句对于所有提供的空值为非真并将其删除。将其限制条件放置在 where 子句的连接是一个有效内连接。

然而，如果将限制条件移动到 on 子句中，则它适用于连接过程中，并在生成连接表时利用它。在这种情况下，结果集包括内部表中限制条件为真的所有行，以及外部表的所有行 （如果这些行不满足限制标准则为空扩展行）：

```
select title, titles.title_id, price, au_id
from titleauthor left join titles
on titles.title_id = titleauthor.title_id
and price > $20.00

title      title_id   price           au_id
---------  ---------  -----------     -----------
NULL       NULL       NULL            172-32-1176
NULL       NULL       NULL            213-46-8915
. . .
Onions,    TC3218     20.95           807-91-6654
. . .
NUL        NULL       NULL            998-72-3567
NULL       NULL       NULL            998-72-3567
(25 rows affected)
```

此结果集包括先前示例未包括的 21 行。

通常，如果查询需要对内部表的限制条件 （例如，上述查询中的 “and price > $20.00”），可能要将此条件放置到 on 子句中；这样将保留外部表的行。如果在 where 子句中包括内部表的限制条件，则结果集可能不包括外部表的行。

与在外部表上放置限制标准一样，将内部表的限制条件放置到 on 还是 where 子句上，最终取决于所需的结果集。如果仅对限制条件为真的行感兴趣，并且不包括外部表中行的全集，则将限制条件放置到 where 子句中。然而，如果需要结果集包括外部表的所有行，而不管它们是否满足限制条件，应在 on 子句中放置限制条件。

在内部和外部表中包括的限制

下列查询的 where 子句中的限制条件包括内部表和外部表：

```
select title, titles.title_id, price, price*qty, qty
from salesdetail left join titles
on titles.title_id = salesdetail.title_id
where price*qty > $30000.00
```

```
title               title_id   price  qty
-----------------   --------   -----  ---------  -----
Silicon Valley Ga   MC2222     19.99  40,619.68  2032
But Is It User Fr   PC1035     22.95  45,900.00  2000
But Is It User Fr   PC1035     22.95  45,900.00  2000
But Is It User Fr   PC1035     22.95  49,067.10  2138
Secrets of Silico   PC8888     20.00  40,000.00  2000
Prolonged Data De   PS3333     19.99  53,713.13  2687
Fifty Years in Bu   TC4203     11.95  32,265.00  2700
Fifty Years in Bu   TC4203     11.95  41,825.00  3500
(8 rows affected)
```

在 where 子句中放置限制条件消除了结果集中的下列行：

- 限制条件 “price*qty>$30000.0”为假的行
- 限制条件 “price*qty>$30000.0”为未知（因为 price 为空）的行

若要保留外部表的不匹配行，请将限制条件移动到 on 子句，如下例所示：

```
select title, titles.title_id, price, price*qty, qty
from salesdetail left join titles
on titles.title_id = salesdetail.title_id
and price*qty > $30000.00
title               title_id   price  qty
-----------------   --------   -----  ---------  -----
NULL                NULL       NULL   NULL       75
NULL                NULL       NULL   NULL       75
. . .
Secrets of Silico   PC8888     20.00  40,000.00  2000
. . .
NULL                NULL       NULL   NULL       300
NULL                NULL       NULL   NULL       400
(116 rows affected)
```

此查询在结果集中保留 salesdetail 表的所有 116 行，如果该表中的行不满足限制条件则为空扩展行。

放置包括内部和外部表的限制条件的位置取决于所需的结果集。如果只对限制条件为真的行感兴趣，则将限制条件放置在 where 子句中。然而，如果要包括外部表的所有行，而不管它们是否满足限制条件，应在 on 子句中放置限制条件。

嵌套 ANSI 外连接

嵌套外连接使用一个外连接的结果集作为另一个外连接的表引用。例如：

```
select t.title_id, title, ord_num, sd.stor_id, stor_name
from (salesdetail sd
left join titles t
on sd.title_id = t.title_id)   /*join #1*/
left join stores
on sd.stor_id = stores.stor_id /*join #2*/
```

```
title_id  title         ord_num  stor_id  stor_name
--------  ------------  -------  -------  ------------------------
TC3218    Onions, L...  234518   7896     Fricative Bookshop
TC7777    Sushi, An...  234518   7896     Fricative Bookshop
. . .
TC4203    Fifty Yea...  234518   6380     Eric the Read Books
MC3021  The Gourmet...  234518   6380     Eric the Read Books
(116 rows affected)
```

在此示例中， salesdetail 和 titles 表间的连接表在逻辑上首先生成，然后，在 salesdetail.stor_id 等于 stores.stor_id 时，与 stores 表的列进行连接。在语义上，连接中嵌套的每级都创建一个连接表，然后用于下一连接。

在上述查询中，因为第一个外连接成为第二个外连接的运算符，所以此查询为**左嵌套外连接**。

下例显示右嵌套外连接：

```
select stor_name, qty, date, sd.ord_num
from salesdetail sd left join (stores /*join #1 */
left join sales on stores.stor_id = sales.stor_id) /*join #2 */
on stores.stor_id = sd.stor_id
where date > "1/1/1990"
```

```
stor_name      qty  date                  ord_num
------------   ---- ------------------    --------------------
News & Brews   200  Jun 13 1990 12:00AM  NB-3.142
News & Brews   250  Jun 13 1990 12:00AM  NB-3.142
News & Brews   345  Jun 13 1990 12:00AM  NB-3.142
. . .
Thoreau Read   1005 Mar 21 1991 12:00AM  ZZ-999-ZZZ-999-0A0
Thoreau Read   2500 Mar 21 1991 12:00AM  AB-123-DEF-425-1Z3
Thoreau Read   4000 Mar 21 1991 12:00AM  AB-123-DEF-425-1Z3
```

在此示例中，第二个连接 （stores 和 sales 表间）在逻辑上首先生成，然后与 salesdetail 表连接。因为第二个外连接用作第一个外连接的表引用，所以此查询为**右嵌套外连接**。

如果第一个外连接 （“from salesdetail...”）的 on 子句失败，则它向第二个外连接的 stores 和 sales 表提供空值。

嵌套外连接中的小括号

嵌套外连接生成带有或不带有小括号的相同结果集。如果连接使用小括号构建，具有多个外连接的较大查询对于用户可更加易读。

在嵌套外连接中的 on 子句

在嵌套外连接中的 on 子句的位置确定逻辑上首先处理的连接。从左到右读取，第一个 on 子句是要定义的第一个连接。

在下列示例中，第一个连接 （小括号中）中 on 子句的位置表示它是第二个连接的表引用，所以它被 *首先*定义，生成与 authors 表连接的表引用：

```
select title, price, au_fname, au_lname
from (titles left join titleauthor
on titles.title_id = titleauthor.title_id)   /*join #1*/
left join authors
on titleauthor.au_id = authors.au_id      /*join #2*/
and titles.price > $15.00
title            price      au_fname      au_lname
---------------  ---------  ------------  -------------
The Busy Exe...  19.99      Marjorie      Green
The Busy Exe...  19.99      Abrahame      Bennet
. . .
Sushi, Anyon...  14.99      Burt          Gringlesby
Sushi, Anyon...  14.99      Akiko         Yokomoto
(26 rows affected)
```

然而，如果 on 子句在其它位置，则连接按不同序列求值，但仍然生成相同结果集 （这仅具有解释性的意义；如果连接表在逻辑上按不同顺序生成，则它不太可能生成相同结果集）：

```
select title, price, au_fname, au_lname
from titles left join
(titleauthor left join authors
on titleauthor.au_id = authors.au_id)        /*join #2*/
on titles.title_id = titleauthor.title_id    /*join #1*/
and au_lname like"Yokomoto"

title                    price        au_fname          au_lname
----------------------   -------      -----------       -----------

The Busy Executive's     19.99        Marjorie          Green
The Busy Executive's     19.99        Abraham           Bennet
. . .
```

```
Sushi, Anyone?          14.99       Burt            Gringlesby
Sushi, Anyone?          14.99       Akiko           Yokomoto

(26 rows affected)
```

第一个连接（查询的最后一行）的 on 子句位置表示*第二个*左连接是第一个连接的表引用，所以它被首先执行。即，第二个左连接的结果与 titles 表连接。

用连接顺序依赖性转换外连接

几乎为 Adaptive Server 早期版本编写的所有 Transact-SQL 外连接，在 12.0 和更高版本的 Adaptive Server 上运行时将生成相同的结果集。然而，有一类外连接查询，其结果集取决于优化过程中选择的连接顺序。根据查询中谓词求值的位置，使用更高版本的 Adaptive Server 发出这些查询时，它们可生成不同的结果集。它们返回的结果集由将谓词分配给连接的 ANSI 规则确定。

谓词引用的所有表被处理后，才能对谓词求值。即，在以下查询中，只有在处理 titles 表后，才能对谓词 “and titles.price > 20”求值：

```
select title, price, au_ord
from titles, titleauthor
where titles.title_id *= titleauthor.title_id
and titles.price > 20
```

12.0 以前版本的 Adaptive Server 中的谓词根据下列语义求值：

- 如果谓词在外连接的内部表上被求值，则谓词具有 on 子句的语义。
- 如果使用表（对于所有外连接都为外部表或不依赖于连接顺序）对谓词求值，则谓词具有 where 子句的语义。

注释 在生产环境中运行 Adaptive Server 之前，确保使用跟踪标志 4413 启动它，并且运行您认为可能依赖于 12.0 版之前 Adaptive Server 中连接顺序的任何查询。运行依赖于 12.0 版之前 Adaptive Server 中连接顺序的查询时，用跟踪标志 4413 启动的 Adaptive Server 发出类似于下面的消息：

```
Warning: The results of the statement on line %d are
join-order independent. Results may differ on
pre-12.0 releases, where the query is potentially
join-order dependent.
```

确保解决应用程序对由 12.0 版本之前 Adaptive Server 生成的连接顺序查询的结果集所具有的依赖性。

依赖于外连接的连接顺序何时发生影响？

一般情况下，依赖于查询的连接顺序不会发生任何问题，因为谓词通常仅引用：

- 外部表，它使用 where 子句语义求值
- 内部表，它使用 on 子句语义求值
- 内部表及其所依赖的表

这些表不会生成依赖于外连接的连接顺序。然而，具有下列任何特性的 Transact-SQL 查询转换为 ANSI 外连接后，它们可生成不同结果集：

- 谓词包含 or 语句并引用外连接的内部表和不依赖于此内部表的另一个表
- 引用内部表属性的谓词就是不在内部表的连接顺序依赖性中的表的谓词
- 在子查询中作为相关引用来引用的内部表

下列示例演示了用连接顺序依赖性将 Transact-SQL 查询转换为 ANSI 外连接查询的问题。

示例：

```
select title, price, authors.au_id, au_lname
from titles, titleauthor, authors
where titles.title_id =* titleauthor.title_id
and titleauthor.au_id = authors.au_id
and (titles.price is null or authors.postalcode = '94001')
```

此查询不依赖于连接顺序，因为外连接引用 titleauthor 和 titles 表，并且可以根据如下三个连接顺序将 authors 表与这些表连接：

- authors、 titleauthors、 titles （作为 on 子句的一部分）
- titleauthors、 authors、 titles （作为 on 子句的一部分）
- titleauthors、 titles、 authors （作为 where 子句的一部分）

此查询生成以下消息：

```
Warning: The results of the statement on line 1 are join-order independent.
Results may differ on pre-12.0 releases, where the query is potentially join-
order dependent. Use trace flag 4413 to suppress this warning message.
```

下面是等同的 ANSI 示例：

```
select title, price, authors.au_id, au_lname
from titles right join
(titleauthor inner join authors
on titleauthor.au_id = authors.au_id)
on titles.title_id = titleauthor.title_id
```

```
where (titles.price is null or authors.postalcode = '94001')
```

另一个示例：

```
select title, au_fname, au_lname, titleauthor.au_id, price
from titles, titleauthor, authors
where authors.au_id *= titleauthor.au_id
and titleauthor.au_ord*titles.price > 40
```

此查询依赖于连接顺序，原因与先前示例相同。下面是等同的 ANSI 示例：

```
select title, au_fname, au_lname, titleauthor.au_id, price
from titles,(authors left join titleauthor
on titleauthor.au_id = authors.au_id)
where titleauthor.au_ord*titles.price > 40
```

Transact-SQL 外连接

Transact-SQL 包括左外连接和右外连接的语法。*左外连接* (*=) 选择第一个表中满足语句限制条件的所有行。如果与连接条件相匹配，第二个表产生值。否则，第二个表产生空值。

例如，以下左外连接列出所有作者并查找其城市中的出版社 （如果有）：

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city *= publishers.city
```

右外连接 (=*) 选择第二个表中满足语句限制条件的所有行。如果与连接条件相匹配，第一个表产生值。否则，第一个表产生空值。

注释 在 having 子句中不能包括 Transact-SQL 外连接。

表是外连接的内部成员或外部成员。如果连接运算符为 *=，则第二个表为内部表；如果连接运算符为 =*，则第一个表为内部表。可将内部表的某一列比作常量，并将其使用于外连接中。例如，若要查找销售超过 4000 册的 title，请使用以下命令：

```
select qty, title from salesdetail, titles
where qty > 4000
and titles.title_id *= salesdetail.title_id
```

然而，外连接的内部表也无法编入常规连接子句。

先前示例使用连接来查找与出版社处于同一城市的作者姓名，返回两个姓名：Abraham Bennet 和 Cheryl Carson。要在结果中包括所有作者，而不管是否与出版者居住在同一城市，则使用外连接。外连接的查询和结果如下所示：

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city *= publishers.city

au_fname     au_lname         pub_name
---------    --------------   ---------------
Johnson      White            NULL
Marjorie     Green            NULL
Cheryl       Carson           Algodata Infosystems
Michael      O'Leary          NULL
Dick         Straight         NULL
Meander      Smith            NULL
Abraham      Bennet           Algodata Infosystems
Ann          Dull             NULL
Burt         Gringlesby       NULL
Chastity     Locksley         NULL
Morningstar  Greene           NULL
Reginald     Blotche-Halls    NULL
Akiko        Yokomoto         NULL
Innes        del Castillo     NULL
Michel       DeFrance         NULL
Dirk         Stringer         NULL
Stearns      MacFeather       NULL
Livia        Karsen           NULL
Sylvia       Panteley         NULL
Sheryl       Hunter           NULL
Heather      McBadden         NULL
Anne         Ringer           NULL
Albert       Ringer           NULL

(23 rows affected)
```

比较运算符 *= 使外连接区别于普通连接。这个左外连接指示 Adaptive Server 在结果中包括 authors 表的所有行，而无论在 publishers 表中的 city 列上是否有匹配。结果显示对于列出的多数作者没有匹配数据，所以这些行在 pub_name 列中包含 NULL。

右外连接使用比较运算符 =* 指定，它表示将在结果中包括第二个表的所有行，而不管在第一个表中是否有匹配数据。

在以前所示的外连接中替换此运算符，得出如下结果：

```
select au_fname, au_lname, pub_name
```

```
from authors, publishers
where authors.city =* publishers.city
au_fname     au_lname    pub_name
---------    ---------   ---------------
NULL         NULL        New Age Books
NULL         NULL        Binnet & Hardley
Cheryl       Carson      Algodata Infosystems
Abraham      Bennet      Algodata Infosystems

(4 rows affected)
```

可通过将其用作常量来进一步限制外连接。这意味着可精确地限定要查看的值，并使用外连接列出未进行剪切的行。首先查看等值连接，然后将其与外连接比较。例如，若要从任何商店中查找销售量超过 500 册的书目，请使用如下查询：

```
select distinct salesdetail.stor_id, title
from titles, salesdetail
where qty > 500
and salesdetail.title_id = titles.title_id
stor_id
title

------- ----------------------------------------
5023    Sushi, Anyone?
5023    Is Anger the Enemy?
5023    The Gourmet Microwave
5023    But Is It User Friendly?
5023    Secrets of Silicon Valley
5023    Straight Talk About Computers
5023    You Can Combat Computer Stress!
5023    Silicon Valley Gastronomic Treats
5023    Emotional Security: A New Algorithm
5023    The Busy Executive's Database Guide
5023    Fifty Years in Buckingham Palace Kitchens
5023    Prolonged Data Deprivation: Four Case Studies
5023    Cooking with Computers: Surreptitious Balance Sheets
7067    Fifty Years in Buckingham Palace Kitchens

(14 rows affected)
```

另外，要显示任何商店中销售量未超过 500 册的书目，可使用外连接查询：

```
select distinct salesdetail.stor_id, title
from titles, salesdetail
where qty > 500
and salesdetail.title_id =* titles.title_id
```

```
stor_id     title

-------   ----------------------------------------
NULL    Net Etiquette
NULL    Life Without Fear
5023    Sushi, Anyone?
5023    Is Anger the Enemy?
5023    The Gourmet Microwave
5023    But Is It User Friendly?
5023    Secrets of Silicon Valley
5023    Straight Talk About Computers
NULL    The Psychology of Computer Cooking
5023    You Can Combat Computer Stress!
5023    Silicon Valley Gastronomic Treats
5023    Emotional Security: A New Algorithm
5023    The Busy Executive's Database Guide
5023    Fifty Years in Buckingham Palace Kitchens
7067    Fifty Years in Buckingham Palace Kitchens
5023    Prolonged Data Deprivation: Four Case Studies
5023    Cooking with Computers: Surreptitious Balance Sheets
NULL    Computer Phobic and Non-Phobic Individuals:
        Behavior Variations
NULL    Onions, Leeks, and Garlic: Cooking Secrets of the
        Mediterranean

 (19 rows affected)
```

可用一个简单子句限制内部表。下例列出与出版社处于同一城市的作者，但不包括作者 Cheryl Carson，他通常作为居住在出版社所在城市的作者列出：

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city =* publishers.city
and authors.au_lname != "Carson"
```

```
au_fname     au_lname    pub_name
---------    ---------   ---------------
NULL         NULL        New Age Books
NULL         NULL        Binnet & Hardley
Abraham      Bennet      Algodata Infosystems

(3 rows affected)
```

重新分配的连接

重新分配的连接允许将本地和远程表之间的连接重新分配到远程服务器。远程系统使用动态创建的引用回本地表的代理表执行连接。使用远程系统执行连接，可避免大量的网络通信量

使用重新分配的连接

本地表 ls1 和远程表 rl1 之间的连接会导致系统将以下查询发送到远程服务器：

```
select a,b,c
from localserver.testdb.dbo.ls1 t1, rl1 t2
where t1.a = t2.a
```

发送到远程服务器的语句包含对本地系统上的本地表的完全限定引用。远程服务器将为此表动态创建临时代理表定义，或者使用包含匹配映射的现有代理表。远程服务器之后将执行连接，并将结果集返回到本地服务器。

此重新分配的连接的计划将显示为：

```
              Emit
                |
              DXCHG
                |
              JOIN
             /      \
       DXCHG        Scan (rl1)
       /
Scan (ls1)
```

此计划有两个分布式交换运算符。本地表 ls1 的扫描上的 DXCHG 代表远程服务器将结果拉到远程系统时的开销。计划顶部的 DXCHG 代表将连接结果移回到本地系统。

配置重新分配的连接

必须为所涉及的每个远程服务器明确启用重新分配的连接。必须完成此配置，因为远程服务器必须能够使用组件集成服务 (CIS) 连接回本地服务器。

配置重新分配的连接：

1　使用 sp_serveroption 启用要发送到该远程服务器的重新分配的连接：

```
sp_serveroption  servername, "relocated joins",true
```

2　为了使 Adaptive Server 能够建立返回到本地服务器的 CIS 连接，请检验远程服务器上的以下各项：

- 远程服务器有进入本地服务器的接口
- 存在 sysservers 条目 （通过 sp_addserver 添加）
- 已配置外部登录

3　如果正在使用动态创建的代理表，则在接收到重新分配的连接时，会在 tempdb 中创建代理表。启用 ddl in tran 以确保 tempdb 允许代理表：

```
sp_dboption tempdb,"ddl in tran",true
```

空值如何影响连接

被连接的表或视图中的空值决不能互相匹配。由于 bit 列不允许为空值，所以当内部表中的 bit 列上没有匹配值时，在外连接中显示值为 0。

空与任何其它值的连接结果为空。因为空值代表未知或不适用的值，Transact-SQL 没有理由认为一个未知值与另一值相匹配。

仅使用一个外连接，就可检测被连接的一个表中的列是否存在空值。下面是两个表，每个表在参与连接的列中都有空。左外连接在第一个表中显示空值。

图 4-1：外连接中的空值

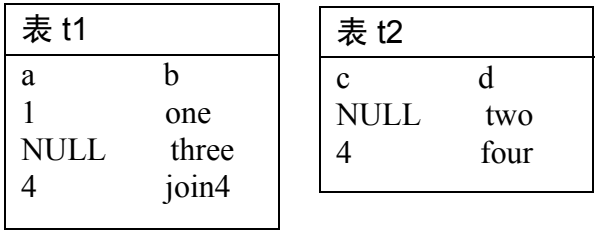


图 6-1 显示两个表，表 t1 和表 t2。表 t1 包含两个列“a”和“b”。表 t2 包含两个列“c”和“d”。表 t1 包含三行：第 1 行在“a”列中显示“1”，在“b”列中显示“one”。第 2 行在“a”列中显示“NULL”，在“b”列中显示“three”。第 3 行在“a”列中显示“4”，在“b”列中显示“join4”。表 t2 有 2 行：第 1 行在“c”列中显示“NULL”，在“d”列中显示“two”，第 2 行在“c”列中显示“4”，在“d”列中显示“four”。

下面是左外连接：

```
select *
from t1, t2
where a *= c
```

```
a            b       c            d
-----------  ------  -----------  ------
          1  one            NULL  NULL
       NULL  three            NULL  NULL
          4  join4               4  four

(3 rows affected)
```

结果很难区分出是数据中的空值还是代表失败连接的空值。在被连接的数据中出现空值时，通常使用常规连接将其从结果中省略。

确定要连接的表列

sp_helpjoins 列出两个表或视图中适于连接的列。它的语法为：

```
sp_helpjoins table1, table2
```

例如，下面说明如何使用 sp_helpjoins 查找 titleauthor 和 titles 间适于连接的列：

```
sp_helpjoins titleauthor, titles
first_pair

---------------------------- --------------------
title_id                     title_id
```

sp_helpjoins 显示的列对有两个来源。首先， sp_helpjoins 检查当前数据库中的 syskeys 表，查看是否已使用 sp_foreignkey 在两个表上定义任何外键，然后，检查是否已用 sp_commonkey 在两个表上定义了任何公用键。如果它未在其中找到任何公用键，此过程应用较少的限制标准来识别可能被适当连接的任何键。它检查具有相同用户数据类型的键，如果仍找不到，它将检查具有相同名称和数据类型的列。

请参见 《参考手册：过程》。

第 5 章

子查询：在其它查询中使用查询

子查询是一个 select 语句，它嵌套于另一个 select、 insert、 update 或 delete 语句中，位于条件语句或另一个子查询内。

也可将子查询表示为连接操作。请参见第 4 章 “连接：从若干表中检索数据”。

子查询工作方式

子查询又称为**内部查询**，它出现于其它 SQL 语句的 where 或 having 子句中，或出现于语句的选择列表中。子查询可用来处理表达为其它查询的结果的查询请求。包含子查询的语句会依据子查询的 select 列表的求值来处理来自一个表的行，它可与外部查询引用相同的表，也可以引用不同的表。在 Transact-SQL 中，子查询还可用于允许使用表达式的几乎任何地方，条件是子查询返回一个单值。case 表达式也可包含子查询。

例如，下面的子查询列出版税分配大于 $75 的所有作者的姓名：

```
select au_fname, au_lname
from authors
where au_id in
   (select au_id
    from titleauthor
    where royaltyper > 75)
```

包含一个或多个子查询的 select 语句有时也称为**嵌套查询**或**嵌套的 select 语句**。

您可将连接明确地表示为包含子查询的多个 SQL 语句。其它的问题也可以只通过子查询来解决。有些人认为子查询更易于理解。而其它 SQL 用户则尽可能地避免使用子查询。您可以选择自己喜欢的方式。

不返回值的子查询的结果为 NULL。如果子查询返回 NULL，则此查询失败。

请参见 《参考手册：命令》。

子查询限制

子查询存在如下限制：

- 除在 exists 子查询中外 （通常以 (*) 代替单个列名），*subquery_select_list* 只能由一个列名组成。不要指定多个列名。如果列名所属的表或视图存在歧义，应用表名或视图名限定列名。
- 子查询可嵌套于外层 select、insert、update 或 delete 语句的 where 或 having 子句中，或嵌套于另一个子查询或选择列表中。或者，也可编写多个包含子查询的语句作为连接； Adaptive Server 将这种语句作为连接来处理。
- 在 Transact-SQL 中，子查询可用于允许使用表达式的几乎任何地方，条件是子查询返回一个单值。
- 子查询可出现在可使用表达式的几乎任何地方。 SQL 派生表因此可以在使用了子查询的任何地方用于该子查询的 from 子句。
- 不能在 order by、 group by 或 compute by 列表中使用子查询。
- 不能在子查询中包括 for browse 子句。
- 不能在子查询中包括 union 子句，除非它在该子查询内是派生表表达式的一部分。有关使用 SQL 派生表的详细信息，请参见第 9 章 “SQL 派生表”。
- 由比较运算符引入的内部子查询中的选择列表只能包含一个表达式或列名，并且子查询必须返回单个值。在外层语句的 where 子句中指定的列与在子查询选择列表中指定的列必须是连接兼容的。
- 不能在子查询中包含 text、 unitext 或 image 数据类型。
- 子查询不能在内部处理其结果，也就是说子查询不能包含 order by 子句、 compute 子句或 into 关键字。
- 在由 declare cursor 定义的可更新游标的 select 子句中不允许有相关（重复）子查询。

- 嵌套不能超过 50 层。
- 在 union 运算符两边最多可有 50 个子查询。
- 子查询的 where 子句可包含集合函数，但条件是该子查询必须在外层查询的 having 子句中，并且集合值必须为外层查询的 from 子句中的某表上的一列。
- 对来自子查询的结果表达式的限制与其它任何表达式的限制相同。表达式的最大长度是 16K。请参见《参考手册：构件块》中的第 4 章“表达式、标识符和通配符”。

使用子查询的示例

假设要查找与 *Straight Talk About Computers* 价格相同的书籍。首先，查找 *Straight Talk* 的价格：

```
select price
from titles
where title = "Straight Talk About Computers"

price
-------------
       $19.99

(1 row affected)
```

将第一个查询的结果用于第二个查询中，以查找与 *Straight Talk* 价格相同的所有书籍：

```
select title, price
from titles
where price = $19.99

title                                          price
-----------------------------------------      -----
The Busy Executive's Database Guide            19.99
Straight Talk About Computers                  19.99
Silicon Valley Gastronomic Treats              19.99
Prolonged Data Deprivation: Four Case Studies 19.99

(4 rows affected)
```

使用子查询只需一步即可得到相同的结果：

```
select title, price
from titles
where price =
     (select price
    from titles
    where title = "Straight Talk About Computers")

title                                          price
---------------------------------------        -----
The Busy Executive's Database Guide            19.99
Straight Talk About Computers                  19.99
Silicon Valley Gastronomic Treats              19.99
Prolonged Data Deprivation: Four Case Studies 19.99

(4 rows affected)
```

限定列名

语句中的列名由在同级的 from 子句中引用的表隐式限定。下例中，表名 publishers 隐式限定外层查询 where 子句中的 pub_id 列。在子查询的选择列表中对 pub_id 的引用由该子查询的 from 子句（即 titles 表）进行限定：

```
select pub_name
from publishers
where pub_id in
   (select pub_id
    from titles
    where type = "business")
```

下面是隐式假定生效时的查询：

```
select pub_name
from publishers
where publishers.pub_id in
   (select titles.pub_id
    from titles
    where type = "business")
```

显式地规定表名永远都不会错，可通过使用显式限定来覆盖有关表名的隐式假定。

带相关名的子查询

如第 4 章 “连接：从若干表中检索数据”所述，自连接中需要使用表相关名，因为与其自身连接的表以两种不同的角色出现。也可在嵌套查询（这些查询的内层查询与外层查询引用同一个表）中使用相关名。

例如，可使用以下子查询来查找与 Livia Karsen 居住在同一城市的作者：

```
select au1.au_lname, au1.au_fname, au1.city
from authors au1
where au1.city in
   (select au2.city
    from authors au2
    where au2.au_fname = "Livia"
    and au2.au_lname = "Karsen")
```

```
au_lname     au_fname     city
-----------  ---------    -------
Green        Marjorie     Oakland
Straight     Dick         Oakland
Stringer     Dirk         Oakland
MacFeather   Stearns      Oakland
Karsen       Livia        Oakland

(5 rows affected)
```

显式相关名清楚地显示：在子查询中对 authors 的引用与在外部查询中对 authors 的引用并不相同。

如果没有显式相关，则子查询为：

```
select au_lname, au_fname, city
from authors
where city in
   (select city
    from authors
    where au_fname = "Livia"
    and au_lname = "Karsen")
```

或者，可将上述查询和其它语句（其中的子查询与外部查询引用同一个表）表述为自连接：

```
select au1.au_lname, au1.au_fname, au1.city
from authors au1, authors au2
where au1.city = au2.city
and au2.au_lname = "Karsen"
and au2.au_fname = "Livia"
```

重述为连接的子查询返回结果的顺序可能会有所不同；此外，此连接可能需要使用 distinct 关键字来消除重复项。

多重嵌套

子查询可包含一个或多个子查询。在一个语句中最多可嵌套 50 个子查询。

例如，若要查找参与编写了至少一本计算机畅销书的作者的姓名，请输入：

```
select au_lname, au_fname
from authors
where au_id in
   (select au_id
    from titleauthor
    where title_id in
      (select title_id
       from titles
       where type = "popular_comp") )

au_lname                au_fname
----------------------- ------------
Carson                  Cheryl
Dull                    Ann
Locksley                Chastity
Hunter                  Sheryl

(4 rows affected)
```

最外层查询选择所有的作者名。下一层查询查找作者 ID，最内层查询返回标题 ID 号 PC1035、 PC8888 和 PC9999。

也可将此查询表示为一个连接：

```
select au_lname, au_fname
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
and type = "popular_comp"
```

update、 *delete* 和 *insert* 语句中的子查询

可在 update、 delete 和 insert 语句以及 select 语句中嵌套子查询。

注释　运行本节中的示例查询将会更改 pubs2 数据库。若要获得样本数据库的原始副本，请向系统管理员寻求帮助。

以下查询将 New Age Books 出版的所有书籍的价格加倍。此语句会更新 titles 表；其子查询引用 publishers 表。

```
update titles
set price = price * 2
where pub_id in
   (select pub_id
    from publishers
    where pub_name = "New Age Books")
```

使用连接的等效 update 语句为：

```
update titles
set price = price * 2
from titles, publishers
where titles.pub_id = publishers.pub_id
and pub_name = "New Age Books"
```

可用以下嵌套 select 语句删除所有商业书籍的销售记录：

```
delete salesdetail
where title_id in
   (select title_id
    from titles
    where type = "business")
```

使用连接的等效 delete 语句为：

```
delete salesdetail
from salesdetail, titles
where salesdetail.title_id = titles.title_id
and type = "business"
```

条件语句中的子查询

可在条件语句中使用子查询。可按下例所示，重新编写上述删除所有商业书籍销售记录的子查询，以在删除这些记录*之前*对它们进行检查：

```
if exists (select title_id
    from titles
    where type = "business")
begin
    delete salesdetail
    where title_id in
      (select title_id
       from titles
       where type = "business")
end
```

用子查询替代表达式

在 Transact-SQL 中， select、 update、 insert 或 delete 语句中几乎所有可使用表达式的地方都可用子查询来替代。例如，子查询可与外连接的内部表中的列进行比较。

可将子查询用于 order by 列表中，或者将其用作 insert 语句的 values 列表中的一个表达式。

以下语句显示如何查找由居住在加利福尼亚的作者所著并在那里出版的书籍的标题和类型：

```
select title, type
from titles
where title in
   (select title
    from titles, titleauthor, authors
    where titles.title_id = titleauthor.title_id
    and titleauthor.au_id = authors.au_id
    and authors.state = "CA")
and title in
   (select title
    from titles, publishers
    where titles.pub_id = publishers.pub_id
    and publishers.state = "CA")

title                                   type
-----------------------------------   ----------
The Busy Executive's Database Guide   business
Cooking with Computers:
```

```
     Surreptitious Balance Sheets     business
Straight Talk About Computers         business
But Is It User Friendly?              popular_comp
Secrets of Silicon Valley             popular_comp
Net Etiquette                         popular_comp

(6 rows affected)
```

以下语句选择销量在 5000 册以上的书籍的书名，并列出其价格以及最贵书籍的价格：

```
select title, price,
    (select max(price) from titles)
    from titles
    where total_sales > 5000

title                                price
-----------------------------------  -----  ------
You Can Combat Computer Stress!       2.99   22.95
The Gourmet Microwave                 2.99   22.95
But Is It User Friendly?             22.95   22.95
Fifty Years in Buckingham Palace
    Kitchens                         11.95   22.95

(4 rows affected)
```

子查询类型

子查询分为两种基本类型：

- **表达式子查询**由无修饰词的比较运算符引入，它必须返回单个值，可用于 SQL 中几乎所有允许使用表达式的地方。
- **定量判定子查询**对由 in 引入的列表、或由经 any 或 all 修饰的比较运算符引入的列表进行操作。定量判定子查询返回 0 或更多的值。此类子查询也用作存在测试，由 exists 引入。

每种类型的子查询或是无关的或是相关的（重复的）。

- **无关子查询**可按独立查询那样进行求值。在概念上，子查询的结果在主语句或外层查询中被替代。但这并*不是* Adaptive Server 对含有子查询的语句的实际处理方式。无关子查询也可表达为连接，并由 Adaptive Server 将其作为连接来处理。
- **相关子查询**不可作为独立查询进行求值，但可引用外部查询的 from 列表中列出的表的列。本章最后将详细论述相关子查询。

表达式子查询

表达式子查询包括：

- select 列表中的子查询 （由 in 引入）
- 由比较运算符 （=、 !=、 >、 >=、 <、 <=）连接的 where 或 having 子句中的子查询

表达式子查询的一般格式是：

[select、 insert、 update、 delete 语句或子查询开始]

```
where expression comparison_operator (subquery)
```

[select、 insert、 update 或 delete 语句或子查询结束]

表达式由子查询或由算术运算符或逐位运算符连接的列名、常量和函数的任何组合组成。

comparison_operator 为以下运算符之一：

运算符	含义
=	等于
>	大于
<	小于
>=	大于或等于
<=	小于或等于
!=	不等于
<>	不等于
!>	不大于
!<	不小于

如果在外层语句的 where 或 having 子句中使用列名，应确保 *subquery_select_list* 中的列名与它是可以连接的。

由无修饰词的比较运算符 （即后面没有 any 或 all 的比较运算符）引入的子查询必须求出一个单值。如果此类子查询返回多个值， Adaptive Server 会返回一个错误消息。

例如，假设每个出版社仅位于一个城市中。若要查找与 Algodata Infosystems 居住在同一城市的作者的姓名，可编写一个含有子查询（由比较运算符 = 引入）的语句：

```
select au_lname, au_fname
from authors
where city =
   (select city
```

```
    from publishers
    where pub_name = "Algodata Infosystems")

au_lname        au_fname
--------------- --------------
Carson          Cheryl
Bennet          Abraham

(2 rows affected)
```

使用标量集合函数来确保单值

由无修饰词的比较运算符引入的子查询通常包括标量集合函数，因为它们返回单值。

例如，若要查找价格高于当前最低价格的书籍的书名：

```
select title
from titles
where price >
   (select min(price)
    from titles)

title
--------------------------------------------------
The Busy Executive's Database Guide
Cooking with Computers: Surreptitious Balance
     Sheets
Straight Talk About Computers
Silicon Valley Gastronomic Treats
But Is It User Friendly?
Secrets of Silicon Valley
Computer Phobic and Non-Phobic Individuals:
     Behavior Variations
Is Anger the Enemy?
Life Without Fear
Prolonged Data Deprivation: Four Case Studies
Emotional Security: A New Algorithm
Onions, Leeks, and Garlic: Cooking Secrets of the
     Mediterranean
Fifty Years in Buckingham Palace Kitchens
Sushi, Anyone?

(14 rows affected)
```

在表达式子查询中使用 *group by* 和 *having*

因为由无修饰词的比较运算符引入的子查询必须返回一个单值，所以它们不能包括 group by 和 having 子句，除非您知道 group by 和 having 子句将返回单值。

例如，以下查询查找价格高于 trad_cook 类别中最低价书的书名：

```
select title
from titles
where price >
   (select min(price)
    from titles
    group by type
    having type = "trad_cook")
```

在表达式子查询中使用 *distinct*

由无修饰词的比较运算符引入的子查询通常包括 distinct 关键字，以确保返回单值。

例如，如果不包含 distinct，以下子查询将由于返回多个值而失败：

```
select pub_name from publishers
    where pub_id =
     (select distinct pub_id
      from titles
      where pub_id = publishers.pub_id)
```

定量判定子查询

定量判定子查询是 where 或 having 子句中由 any、all、in 或 exists 连接的子查询，返回 0 和更大值的列表。 any 或 all 子查询运算符修饰比较运算符。

定量判定子查询共分三类：

- any/all 子查询。由有修饰词的比较运算符引入的子查询 （可能包括 group by 或 having 子句）采取以下一般格式：

  [select、 insert、 update、 delete 语句或子查询开始]

  ```
  where expression comparison_operator [any | all]
      (subquery)
  ```

  [select、 insert、 update 或 delete 语句或子查询结束]

- in/not in 子查询。由 in 或 not in 引入的子查询采取以下一般格式：

  [select、 insert、 update、 delete 语句或子查询开始]

  ```
  where expression [not] in (subquery)
  ```

  [select、 insert、 update 或 delete 语句或子查询结束]

- exists/not exists 子查询。由 exists 或 not exists 引入的子查询是采取以下一般格式的存在测试：

  [select、 insert、 update、 delete 语句或子查询开始]

  ```
  where [not] exists (subquery)
  ```

  [select、 insert、 update 或 delete 语句或子查询结束]

尽管 Adaptive Server 允许在定量判定子查询中使用关键字 distinct，但它总是按子查询中不包括 distinct 那样对其进行处理。

带 *any* 和 *all* 的子查询

关键字 all 和 any 修饰引入子查询的比较运算符。

如果在子查询中将 any 和 <、 > 或 = 一同使用时，则当在子查询中检索到与外层语句的 where 或 having 子句中的值相匹配的任意值时，将返回结果。

如果在子查询中将 all 和 < 或 > 一同使用，则只有在子查询中检索到与外层语句的 where 或 having 子句中的值相匹配的所有值时，才返回结果。

any 和 all 的语法为：

{where | having} [not]
expression comparison_operator {any |all} (*subquery*)

以 > 比较运算符为例：

- > all 表示大于每个值，或大于最大值。例如， > all (1, 2, 3) 表示大于 3。
- > any 表示大于至少一个值，或大于最小值。因此， >any (1, 2, 3) 表示大于 1。

如果使用 all 引入子查询，而比较运算符不返回任何值，则整个查询将失败。

all 和 any 可以互相转换。例如，您可能会问 “哪些书的预付款比 New Age Books 出版的任何一本书的预付款都多？”

可这样解释这个问题，以使 SQL “转换”更清楚：“哪些书的预付款比 New Age Books 支付的最高预付款还多？”此时，需要 all 关键字，*而非* any 关键字：

```
select title
from titles
where advance > all
   (select advance
    from publishers, titles
    where titles.pub_id = publishers.pub_id
    and pub_name = "New Age Books")

title
----------------------------------------
The Gourmet Microwave

(1 row affected)
```

对于每个书目，外部查询从 titles 表中获取标题和预付款值，并将它们与从子查询中返回的 New Age Books 所付预付款金额进行比较。外层查询查找列表中的最大值，并确定所考虑的标题是否得到更多的预付款。

> *all* 表示大于所有值

> all 运算符表示：引入子查询的列中的值必须大于子查询为符合外层查询条件的行返回的每个值。

例如，若要查找价格高于 mod_cook 类别中最高价书籍的书名：

```
select title from titles where price > all
    (select price from titles
    where type = "mod_cook")

title
--------------------------------------------------
But Is It User Friendly?
Secrets of Silicon Valley
Computer Phobic and Non-Phobic Individuals:
    Behavior Variations
Onions, Leeks, and Garlic:Cooking Secrets of
    the Mediterranean

(4 rows affected)
```

但是，如果内部查询返回的结果集中包含 NULL，则查询返回 0 行。这是因为 NULL 代表 “未知值”，而要判断出比较的值是否大于未知值是不可能的。

例如，若要查找价格高于 popular_comp 类别中最高价书籍的书名：

```
select title from titles where price > all
    (select price from titles
     where type = "popular_comp")

title
--------------------------------------------------

(0 rows affected)
```

未返回任何行，这是因为子查询发现其中一本书 (*Net Etiquette*) 的价格为空值。

= *all* 表示等于每个值

= all 运算符表示：引入子查询的列中的值必须等于子查询为符合外层查询条件的行返回的值列表中的每个值。

例如，以下查询通过查找邮政编码来找出居住在同一个城市的作者：

```
select au_fname, au_lname, city
from authors
where city = all
     (select city
      from authors
      where postalcode like "946%")
```

> *any* 表示至少大于一个值

> any 表示：引入子查询的列中的值必须至少大于子查询为符合外层查询条件的行返回的多个值中的一个。

下例由经 any 修饰的比较运算符引入。它查找预付款值大于 New Age Books 所付任意预付款金额的每本书的标题。

```
select title
from titles
where advance > any
   (select advance
    from titles, publishers
    where titles.pub_id = publishers.pub_id
    and pub_name = "New Age Books")

title
--------------------------------------------------
The Busy Executive's Database Guide
Cooking with Computers: Surreptitious Balance
    Sheets
```

```
You Can Combat Computer Stress!
Straight Talk About Computers
The Gourmet Microwave
But Is It User Friendly?
Secrets of Silicon Valley
Computer Phobic and Non-Phobic Individuals:
    Behavior Variations
Is Anger the Enemy?
Life Without Fear
Emotional Security: A New Algorithm
Onions, Leeks, and Garlic: Cooking Secrets of
    the Mediterranean
Fifty Years in Buckingham Palace Kitchens
Sushi, Anyone?

(14 rows affected)
```

对于外部查询选择的每个标题，内部查询都会查找 New Age Books 所付预付款金额的列表。外层查询会查看该列表中的所有值，并确定所考虑的标题的预付款是否比这些值中的任意一个多。也就是说，本示例查找预付款大于或等于 New Age Books 所付*最低*预付款数的书籍的标题。

如果子查询不返回任何值，则整个查询失败。

= *any* 表示等于一些值

= any 运算符是一个存在性检查；它等效于 in。例如，若要查找与任意出版社处于同一城市的作者，可使用 = any 或 in：

```
select au_lname, au_fname
from authors
where city = any
   (select city
    from publishers)
select au_lname, au_fname
from authors
where city in
   (select city
    from publishers)

au_lname        au_fname
--------------- --------------
Carson          Cheryl
Bennet          Abraham

(2 rows affected)
```

但是，!= any 运算符与 not in 不同。!= any 运算符表示 “not = a *or* not = b *or* not = c”；而 not in 表示 “not = a *and* not = b *and* not = c”。

例如，若要查找所居住城市无出版社的作者的姓名：

```
select au_lname, au_fname
from authors
where city != any
   (select city
    from publishers)
```

结果包括所有 23 个作者。这是因为每个作者都居住于无出版社的*某个*城市，且每个作者只居住在一个城市中。

内层查询会找到有出版社的所有城市，然后外层查询为*每个*城市查找不居住于此的作者。

如果在同一个查询中改用 not in，则会出现以下情况：

```
select au_lname, au_fname
from authors
where city not in
   (select city
    from publishers)
```

```
au_lname                au_fname
--------------          ------------
White                   Johnson
Green                   Marjorie
O'Leary                 Michael
Straight                Dick
Smith                   Meander
Dull                    Ann
Gringlesby              Burt
Locksley                Chastity
Greene                  Morningstar
Blotchet-Halls          Reginald
Yokomoto                Akiko
del Castillo            Innes
DeFrance                Michel
Stringer                Dirk
MacFeather              Stearns
Karsen                  Livia
Panteley                Sylvia
Hunter                  Sheryl
McBadden                Heather
Ringer                  Anne
Ringer                  Albert
```

```
(21 rows affected)
```

这些正是所需的结果。其中包括除 Cheryl Carson 和 Abraham Bennet 之外的所有作者，这两位作者居住于 Berkeley，那里是 Algodata Infosystems 的所在地。

使用 !=all 可得到同样的结果，它等效于 not in：

```
select au_lname, au_fname
from authors
where city != all
   (select city
    from publishers)
```

与 *in* 结合使用的子查询

由关键字 in 引入的子查询返回一个 0 和更大值的列表。例如，以下查询会查找出版过商业书籍的出版社的名称：

```
select pub_name
from publishers
where pub_id in
   (select pub_id
    from titles
    where type = "business")

pub_name
----------------------------------------
New Age Books
Algodata Infosystems

(2 rows affected)
```

此语句分两个步骤求值。内部查询返回出版过商业书籍的出版社的标识号，即 1389 和 0736。然后在外部查询中替代这些值，外部查询会在 publishers 表中查找与这两个标识号相匹配的名称。此查询如下所示：

```
select pub_name
from publishers
where pub_id in ("1389", "0736")
```

使用子查询表达这个查询的另一种方法是：

```
select pub_name
from publishers
where "business" in
   (select type
    from titles
```

```
        where pub_id = publishers.pub_id)
```

请注意，外部查询中 where 关键字之后的表达式可以是常量和列名。可以使用其它类型的表达式，例如常量和列名的组合。

上述查询与许多其它子查询一样，也可表达为一个连接查询：

```
select distinct pub_name
from publishers, titles
where publishers.pub_id = titles.pub_id
and type = "business"
```

此查询及表达后的子查询都会查找出版过商业书籍的出版社。尽管可能需要使用 distinct 关键字来消除重复项，但它们都同样正确并产生同样的结果。

但是，对于这个问题或类似问题，使用连接查询而不使用子查询的一个好处是：连接查询在结果中显示来自多个表的列。例如，若要在结果中包括商业书籍的标题，请使用以下连接形式：

```
select pub_name, title
from publishers, titles
where publishers.pub_id = titles.pub_id
and type = "business"
```

```
pub_name             title
--------------------   ----------------------------------------
Algodata Infosystems   The Busy Executive's Database Guide
Algodata Infosystems   Cooking with Computers: Surreptitious
                       Balance Sheets
New Age Books          You Can Combat Computer Stress!
Algodata Infosystems   Straight Talk About Computers

(4 rows affected)
```

下面是可以用子查询或连接查询表达的语句的另一个示例："查找居住于加利福尼亚，并且从某本书中获得的版税低于 30% 的所有第二作者的姓名。"使用子查询时，语句为：

```
select au_lname, au_fname
from authors
where state = "CA"
and au_id in
   (select au_id
    from titleauthor
    where royaltyper < 30
    and au_ord = 2)

au_lname                  au_fname
------------------------ ------------
```

```
MacFeather                Stearns

(1 row affected)
```

外部查询产生一个列表，列出 15 位居住于加利福尼亚的作者。然后，对内部查询求值，产生一个列表，列出符合条件的作者的 ID。

在内层查询和外层查询的 where 子句中都可以包含多个条件。

使用连接时，查询表达如下：

```
select au_lname, au_fname
from authors, titleauthor
where state = "CA"
and authors.au_id = titleauthor.au_id
  and royaltyper < 30
  and au_ord = 2
```

连接始终可以表达为子查询。子查询通常可以表达为连接。

与 *not in* 结合使用的子查询

由关键字短语 not in 引入的子查询也返回一个 0 和更大值的列表。 not in 表示 “not = a *and* not = b *and* not = c”。

以下查询查找*尚未*出版过商业书籍的出版社的名称，与第 178 页的“与 in 结合使用的子查询”中的示例相反：

```
select pub_name from publishers
where pub_id not in
   (select pub_id
    from titles
    where type = "business")

pub_name
----------------------------------------
Binnet & Hardley

(1 row affected)
```

除 not in 替代了 in 外，此查询与先前的查询相同。但是，不能将此语句转换为连接，“不均等”连接会查找出版过*某*一非商业书籍的出版社的名称。有关解释不基于等同性的连接含义的难点的详细论述，请参见第 4 章 “连接：从若干表中检索数据”。

使用包含 NULL 的 *not in* 的子查询

使用 not in 的子查询会为外层查询中的每一行都返回一组值。如果外部查询中的值不在内部查询返回的结果集中，则 not in 求值结果为 TRUE，并且外部查询将所考虑的记录放在结果中。

但是，如果内部查询返回的结果集中不包含匹配值，而是包含 NULL，则 not in 返回 UNKNOWN。这是因为 NULL 代表“未知值”，而要判断出所查找的值是否在包含未知值的集合中是不可能的。外部查询会放弃此行。例如：

```
select pub_name
    from publishers
    where $100.00 not in
        (select price
         from titles
         where titles.pub_id = publishers.pub_id)

pub_name
------
New Age Books

(1 row affected)
```

New Age Books 是唯一未出版过价格为 100 美元的书籍的出版社。查询结果中不包括 Binnet & Handley 和 Algodata Infosystems，这是因为它们都各自出版了一本价格未定的书。

与 *exists* 结合使用的子查询

将 exists 关键字与子查询一起使用可以测试子查询的某些结果是否存在：

{where | having} [not] exists (*subquery*)

也就是说，外部查询的 where 子句会测试子查询返回的行是否存在。实际上，此类子查询并不产生任何数据，而只是返回一个 TRUE 或 FALSE 值。

例如，以下查询查找出版社业书籍的所有出版社的名称：

```
select pub_name
from publishers
where exists
     (select *
      from titles
      where pub_id = publishers.pub_id
      and type = "business")
```

```
pub_name
----------------------------------------
New Age Books
Algodata Infosystems

(2 rows affected)
```

为使此查询的解析在概念上更清晰，应依次考虑每个出版社的名称。此值是否可使子查询至少返回一行？也就是说，它是否可使存在测试的求值结果为 TRUE？

在前面查询的结果中，第二个出版社的名称为 Algodata Infosystems，其标识号为 1389。titles 表中是否存在 pub_id 为 1389、且 type 为“business”的任何行？如果存在，“Algodata Infosystems”应为所选值之一。为其它每个出版社名称重复同样的过程。

由 exists 引入的子查询与其它子查询的不同之处在于：

- 关键字 exists 不优先于列名、常量或其它表达式。
- exists 子查询的求值结果为 TRUE 或 FALSE，而不返回任何数据。
- 子查询的选择列表通常包含星号 (*)。不需要指定列名，因为只需测试符合子查询中指定条件的行是否存在。此外，由 exists 引入的子查询的选择列表规则与标准选择列表规则相同。

exists 关键字非常重要，因为通常没有非子查询的表达方法可以替代它。实际上，由 exists 引入的子查询始终为相关子查询 （请参见第 186 页的“使用相关子查询”）。

尽管不可将某些由 exists 表达的查询表示为其它任何方式，但却可用 exists 表示使用 in 或者由 any 或 all 修饰的比较运算符的所有查询。以下是一些关于使用 exists 的语句及其等效替代方法的示例。

以下两种方法用于查找与出版社处于同一城市的作者：

```
select au_lname, au_fname
from authors
where city = any
   (select city
    from publishers)
```

```
select au_lname, au_fname
from authors
where exists
   (select *
    from publishers
    where authors.city = publishers.city)
```

```
au_lname            au_fname
```

```
--------------    --------------
Carson            Cheryl
Bennet            Abraham

(2 rows affected)
```

以下两个查询用于查找由位于以字母 “B”开头的城市中的出版社所出版的书籍标题：

```
select title
from titles
where exists
   (select *
    from publishers
    where pub_id = titles.pub_id
    and city like "B%")

select title
from titles
where pub_id in
   (select pub_id
    from publishers
    where city like "B%")

title
--------------------------------------------------
You Can Combat Computer Stress!
Is Anger the Enemy?
Life Without Fear
Prolonged Data Deprivation: Four Case Studies
Emotional Security: A New Algorithm
The Busy Executive's Database Guide
Cooking with Computers: Surreptitious Balance
    Sheets
Straight Talk About Computers
But Is It User Friendly?
Secrets of Silicon Valley
Net Etiquette

(11 rows affected)
```

与 *not exists* 结合使用的子查询

not exists 与 exists 十分相似，不同之处在于：当子查询不返回任何行时满足在其中所使用的 where 子句。

例如，若要查找*不*出版社业书籍的出版社的名称，查询应为：

```
select pub_name
from publishers
where not exists
   (select *
    from titles
    where pub_id = publishers.pub_id
    and type = "business")
pub_name
----------------------------------------
Binnet & Hardley

(1 row affected)
```

此查询查找无销售额的书籍的标题：

```
select title
from titles
where not exists
   (select title_id
    from salesdetail
    where title_id = titles.title_id)

title
----------------------------------------
The Psychology of Computer Cooking
Net Etiquette

(2 rows affected)
```

使用 *exists* 查找交集与差集

可使用由 exists 和 not exists 引入的子查询进行两种集理论操作：交集和差集。两个集合的交集包含两个初始集合共有的所有元素。而差集则包含仅属于第一个集合的元素。

city 列上 authors 和 publishers 的交集为作者和出版社所同处的城市的集合：

```
select distinct city
from authors
where exists
  (select *
   from publishers
   where authors.city = publishers.city)

city
--------------------
Berkeley

(1 row affected)
```

city 列上 authors 和 publishers 的差集为有作者居住而没有出版社的城市的集合（即除 Berkeley 外的所有城市）：

```
select distinct city
from authors
where not exists
  (select *
   from publishers
   where authors.city = publishers.city)

city
--------------------
Gary
Covelo
Oakland
Lawrence
San Jose
Ann Arbor
Corvallis
Nashville
Palo Alto
Rockville
Vacaville
Menlo Park
Walnut Creek
San Francisco
Salt Lake City

(15 rows affected)
```

使用 SQL 派生表的子查询

SQL 派生表可用于子查询 from 子句中。例如，以下查询会查找出版过商业书籍的出版社的名称：

```
select pub_name from publishers
   where "business" in
      (select type from
         (select type from titles, publishers
             where titles.pub_id = publishers.pub_id)
       dt_titles)
```

在上例中， dt_titles 是最内层 select 语句定义的 SQL 派生表。

SQL 派生表可用于子查询 （只要子查询合法）的 from 子句中。有关 SQL 派生表的详细信息，请参见第 9 章 “SQL 派生表”。

使用相关子查询

可通过执行一次子查询并将结果值替换进外层查询的 where 子句来对前面的多个查询求值；它们是无关子查询。在包含重复子查询 （或**相关子查询**）的查询中，子查询的值取决于外层查询。子查询重复执行，即对外层查询选择每一行执行一次。

此示例查找获得一本书 100% 版税的所有作者的姓名：

```
select au_lname, au_fname
from authors
where 100 in
   (select royaltyper
    from titleauthor
    where au_id = authors.au_id)

au_lname        au_fname
--------------  ----------
White           Johnson
Green           Marjorie
Carson          Cheryl
Straight        Dick
Locksley        Chastity
Blotchet-Hall   Reginald
del Castillo    Innes
Panteley        Sylvia
Ringer          Albert

(9 rows affected)
```

与前面的大多数子查询不同，对此语句中子查询的解析不能独立于主查询。它需要一个 authors.au_id 值，而此值为一个变量 — 当 Adaptive Server 检查 authors 表的不同行时，它会发生变化。

对前面查询的求值方法如下：Transact-SQL 通过替换内部查询中每行的值，来考虑 authors 表中的每一行，以得到将包含于结果中的内容。例如，假设 Transact-SQL 首先检查 Johnson White 的行。然后，authors.au_id 得到值 “172-32-1176”，即 Transact-SQL 为内部查询替换的值：

```
select royaltyper
from titleauthor
where au_id = "172-32-1176"
```

结果为 100，所以外部查询求值结果为：

```
select au_lname, au_fname
from authors
where 100 in (100)
```

由于 where 条件成立，因此 Johnson White 的行包括在结果中。如果对 Abraham Bennet 行执行同样的过程，会看到此行并未包括在结果中。

包含 Transact-SQL 外连接的相关子查询

Adaptive Server 12.5 和更高版本对包含 Transact-SQL 外连接的相关子查询的处理不同于较早版本的 Adaptive Server。有关详细信息，请参见 “连接：从若干表中检索数据”。下面是一个查询示例，它将相关变量用作 Transact-SQL 外连接的外部成员：

```
select t2.b1, (select t2.b2 from t1 where t2.b1 *= t1.a1) from t2
```

Adaptive Server 的较早版本使用跟踪标志 298 来显示这些查询的错误消息。依据跟踪标志 298 是否打开，以及查询将相关变量用作外连接的内部成员还是外部成员， Adaptive Server 显示了表 5-1 中描述的行为：

表 5-1：在 Adaptive Server 较早版本中的行为

查询类型	跟踪标志 298 关闭	跟踪标志 298 打开
相关变量是外连接的内部成员	不允许：产生错误消息 11013	没有错误
相关变量是外连接的外部成员	没有错误	不允许：产生错误消息 301

Adaptive Server 中跟踪标志 298 的行为与之相反。因为，Adaptive Server versions 12.5 和更高版本在预处理程序阶段将 Transact-SQL 外连接转换为 ANSI 外连接，当允许这样的查询运行时，有可能产生不同结果。在 298 跟踪标志打开时，允许运行包含 Transact-SQL 外连接的相关子查询，这与 Sybase 历史跟踪标志的用法一致。对于 12.5 和更高版本，跟踪标志 298 的行为如下：

表 5-2：在 Adaptive Server 12.5 和更高版本中的行为

查询类型	跟踪标志 298 关闭	跟踪标志 298 打开
相关变量是外连接的内部成员	不允许：产生错误消息 11013	不允许：产生错误消息 11013
相关变量是外连接的外部成员	不允许：产生错误消息 11055	没有错误

注释 Adaptive Server 已将错误消息 301 改为错误消息 11055，但消息文本仍然相同。

带相关名的相关子查询

可使用相关子查询来查找由多个出版社出版的书籍的类型：

```
select distinct t1.type
from titles t1
where t1.type in
   (select t2.type
    from titles t2
    where t1.pub_id != t2.pub_id)

type
--------------------
business
psychology

(2 rows affected)
```

在以下查询中需要相关名，来区分 titles 表在其中出现的两个角色。此嵌套查询等效于自连接查询：

```
select distinct t1.type
from titles t1, titles t2
where t1.type = t2.type
and t1.pub_id != t2.pub_id
```

带比较运算符的相关子查询

表达式子查询可以是相关子查询。例如，若要查找销量低于心理学书目平均销量的心理学书籍的：

```
select s1.ord_num, s1.title_id, s1.qty
from salesdetail s1
where title_id like "PS%"
and s1.qty <
   (select avg(s2.qty)
    from salesdetail s2
    where s2.title_id = s1.title_id)
```

```
ord_num            title_id     qty
------------------ --------     ---
91-A-7             PS3333        90
91-A-7             PS2106        30
55-V-7             PS2106        31
AX-532-FED-452-2Z7 PS7777       125
BA71224            PS7777       200
NB-3.142           PS2091       200
NB-3.142           PS7777       250
NB-3.142           PS3333       345
ZD-123-DFG-752-9G8 PS3333       750
91-A-7             PS7777       180
356921             PS3333       200

(11 rows affected)
```

外部查询依次选择 sales 表 （或 s1）中的行。子查询计算外层查询进行选择时考虑的每次销售的平均数量。对于 s1 中的每个可能值，Transact-SQL 会计算子查询的值，如果数量小于计算出的平均值，则将所考虑的记录放入结果中。

有时，相关子查询与 group by 语句相仿。若要查找价格高于同类书籍平均价格的书籍的标题，则查询为：

```
select t1.type, t1.title
from titles t1
where t1.price >
   (select avg(t2.price)
    from titles t2
    where t1.type = t2.type)
```

```
type         title
---------    -------------------------------------
business     The Busy Executive's Database Guide
business     Straight Talk About Computers
mod_cook     Silicon Valley Gastronomic Treats
```

```
popular_comp But Is It User Friendly?
psychology   Computer Phobic and Non-Phobic
             Individuals:Behavior Variations
psychology   Prolonged Data Deprivation:Four Case
             Studies
trad_cook    Onions, Leeks, and Garlic:Cooking
             Secrets of the Mediterranean

(7 rows affected)
```

对于 t1 的每个可能值，Transact-SQL 都会对子查询求值，如果该行的价格值高于计算出的平均值，则将该行包含在结果中。不必显式地按类型分组，因为计算平均价格的行由子查询中的 where 子句进行限制。

having 子句中的相关子查询

定量判定子查询可以是相关子查询。

在此例中，外部查询的 having 子句中的相关子查询用于查找最高预付款高于给定组中平均预付款两倍的书籍的类型：

```
select t1.type
from titles t1
group by t1.type
having max(t1.advance) >= any
   (select 2 * avg(t2.advance)
   from titles t2
   where t1.type = t2.type)

type
----------
mod_cook

(1 row affected)
```

上述子查询对外部查询中定义的每个组求一次值，即对书籍的每个类型求一次值。

第 6 章

使用和创建数据类型

数据类型定义表中每列保存的信息的种类和存储方式。当您定义列时，可使用 Adaptive Server 系统数据类型，或可创建并使用用户定义的数据类型。

Transact-SQL 数据类型的工作原理

在 Transact-SQL 中，数据类型指定表列、存储过程参数以及局部变量的信息类型、大小与存储格式。例如， int （整数）数据类型用于存储正负 2^{31} 范围内的整数，而 tinyint （微整数）数据类型只用于存储 0 到 255 之间的整数。

Adaptive Server 提供几种系统数据类型和两种用户定义的数据类型（timestamp 和 sysname）。可使用 sp_addtype 建立基于系统数据类型的用户定义的数据类型。

声明列、局部变量或参数时，必须指定系统数据类型或用户定义的数据类型。下例使用系统数据类型 char、 numeric 和 money 定义 create table 语句中的列：

```
create table sales_daily
    (stor_id char(4),
    ord_num numeric(10,0),
    ord_amt money)
```

下一个示例使用 bit 系统数据类型定义 declare 语句中的局部变量：

```
declare @switch bit
```

后面各章将更详细地描述如何使用本章中所述的数据类型声明列、局部变量和参数。使用 sp_help 可确定为现有表中的列定义了何种数据类型。

使用系统提供的数据类型

表 6-1 列出了系统提供的用于不同类型信息的数据类型、 Adaptive Server 可识别的同义词以及每种数据类型的范围和存储大小。尽管 Adaptive Server 允许系统数据类型以大写或小写方式输入，但它们是以小写字符输出的。大多数由 Adaptive Server 提供的数据类型并不是保留字，且可用于命名其它对象。

表 6-1：Adaptive Server 系统数据类型

数据类型种类	同义词	范围	存储的字节数
精确数值：整数			
bigint		-2^{63} 到 2^{63} - 1 之间的整数 （从 -9,223,372,036,854,775,808 到 +9,223,372,036,854,775,807，包括这两个值）。	8
int	integer	2^{31} -1 (2,147,483,647) 到 -2^{31} (-2,147,483,648	4
smallint		2^{15} -1 (32,767) 到 -2^{15} (-32,768)	2
tinyint		0 到 255 （不允许使用负数）	1
unsigned bigint		0 到 18,446,744,073,709,551,615 之间的整数	8
unsigned int		0 到 4,294,967,295 之间的整数	4
unsigned smallint		0 到 65535 之间的整数	2
精确数值：小数			
numeric (p, s)		10^{38} -1 到 -10^{38}	2 到 17
decimal (p, s)	dec	10^{38} -1 到 -10^{38}	2 到 17
近似数值			
float (precision)		与计算机有关	缺省精度 < 16 时为 4，缺省精度 >= 16 时为 8
double precision		与计算机有关	8
real		与计算机有关	4
货币			
smallmoney		214,748.3647 到 -214,748.3648	4
money		922,337,203,685,477.5807 到 -922,337,203,685,477.5808	8

数据类型种类	同义词	范围	存储的字节数
日期 / 时间			
smalldatetime		1900 年 1 月 1 日至 2079 年 6 月 6 日	4
datetime		1753 年 1 月 1 日至 9999 年 12 月 31 日	8
date		0001 年 1 月 1 日至 9999 年 12 月 31 日	4
time		12:00:00AM 到 11:59:59:999PM	4
bigdatetime		0001 年 1 月 1 日到 9999 年 12 月 31 日；12:00:00.000000 AM 到 11:59:59.999999 PM	8
bigtime		12:00:00.000000 AM 到 11:59:59.999999 PM	8
字符			
char(n)	character	页大小	n
varchar(n)	character varying、 char varying	页大小	实际条目长度
unichar	Unicode 字符	页大小	*n* * @@*unicharsize*（@@*unicharsize* 等于 2）
univarchar	Unicode 字符 varying、 char varying	页大小	实际字符数 * @@*unicharsize*
nchar(n)	national character、national char	页大小	*n* * @@*ncharsize*
nvarchar(n)	nchar varying、national char varying、national character varying	页大小	@@*ncharsize* * 字符数
text		2^{31} -1 (2,147,483,647) 个字节或更少	未初始化时为 0；初始化后是 2K 的倍数
unitext		1,073,741,823 个 Unicode 字符或更少	未初始化时为 0；初始化后是 2K 的倍数
二进制			
binary(n)		页大小	*n*
varbinary(n)		页大小	实际条目长度
image		2^{31} -1 (2,147,483,647) 个字节或更少	未初始化时为 0；初始化后是 2K 的倍数
位			
bit		0 或 1	1（1 个字节最多容纳 8 个 bit 列）

精确数值类型：整数

Adaptive Server 提供了数据类型 bigint、 int、 smallint、 tinyint、 unsigned bigint、unsigned int 和 unsigned smallint 来存储整数。这些类型是精确数值类型，它们在算术运算中保留其精确性。

基于要存储数字的预期大小在各种整数类型中选择。内部存储大小视数据类型的不同会有所变化。

仅当值处于要转换为的类型的范围内时，才支持从任一整数类型到其它整数类型的隐式转换。

无符号的整数数据类型允许您为现有的整数类型扩展正数的范围，而不增加所需的存储大小。也就是说，这些数据类型的有符号版本既可以向负数方向扩展，也可以向正数方向扩展 （例如，从 -32 到 +32）。但是，无符号版本只能向正数方向扩展。表 6-2 描述了这些数据类型的有符号和无符号版本的范围。

表 6-2：有符号和无符号数据类型的范围

数据类型	有符号数据类型的范围	数据类型	无符号数据类型的范围
bigint	-2^{63} 到 2^{63} - 1 之间的整数 （从 -9,223,372,036,854,775,808 到 +9,223,372,036,854,775,807，包括这两个值）	unsigned bigint	0 到 18,446,744,073,709,551,615 之间的整数
int	-2^{31} 到 2^{31} - 1 之间的整数 （从 -2,147,483,648 到 2,147,483,647，包括这两个值）	unsigned int	0 到 4,294,967,295 之间的整数
smallint	-2^{15} 到 2^{15} -1 之间的整数 （从 -32,768 到 32,767，包括这两个值）	unsigned smallint	0 到 65535 之间的整数

精确数值类型：小数

对于包含小数点的数字，可使用精确数值类型 numeric 和 decimal。将存储在 numeric 和 decimal 列中的数据进行压缩以节省磁盘空间，并在算术运算后将数据的精确度保留到最低有效位。除了下面这种情况以外，numeric 和 decimal 类型在其它方面都相同：仅标度为 0 的 numeric 类型可以用于标识列。

精确数值类型可接受两种可选参数， precision 和 scale，这两个参数加有小括号，并以逗号分隔：

datatype [(*precision* [, *scale*])]

Adaptive Server 将每种 precision 和 scale 的组合都定义为一种不同的数据类型。例如， numeric(10,0) 和 numeric(5,0) 是两种不同的数据类型。精度和标度确定了可以存储在 decimal 或 numeric 列中的值的范围：

- precision 指定了能够在该列中存储的最大小数位数。它包括小数点左右两侧的所有位数。可将精度指定为 1 至 38 位范围内的一个值，或者使用缺省的 18 位精度。
- scale 指定了能够存储到小数点右侧的最大位数。标度必须小于或等于 precision。可将标度指定为 0 至 38 位范围内的一个值，或者使用缺省的 0 位标度。

标度为 0 的精确数值类型显示时没有小数点。不能输入超过列的精度或标度范围的值。

numeric 或 decimal 列的存储大小取决于其精度。 1 或 2 位数列的最小存储要求为 2 个字节。精度每增加 2 位，存储大小就增加 1 个字节，最多可增加 17 个字节。

近似数值数据类型

数值类型 float、 double precision 和 real 用于存储在算术运算中允许舍入的数值数据。

近似数值数据类型将按二进制小数存储的实数的近似表示存储为二进制 traction。只要显示、输出、在主机间传送或在计算中使用近似数值，数字就会丢失精度。 isql 只显示小数点后的 6 位有效数字，并舍入其余数字。有关精度和近似数值数据类型的详细信息，请参见 《参考手册》。

可将近似数值类型用于包含各类数值的数据。它们支持所有集合函数和所有算术运算。

real 和 double precision 类型是在操作系统提供的类型的基础上建立的。 float 类型接受小括号内的可选精度。精度为 1 至 15 的 float 列存储为 real ；而具有更高精度的那些列存储为 double precision。这三种类型的范围和存储精度都与所使用的计算机有关。

货币数据类型

货币数据类型 money 和 smallmoney 用于存储货币数据。尽管 Adaptive Server 没有提供将一种货币转换为另一种货币的方法，但可将这些数据类型用于美元和其它十进制货币。可以将 money 和 smallmoney 数据用于除 modulo 外的所有算术运算以及所有集合函数。

money 和 smallmoney 都精确到货币单位的万分之一，但显示时它们将数值向上舍入，只保留两位小数。缺省的输出格式是在每三位后放置一个逗号。

日期和时间数据类型

使用 datetime 和 smalldatetime 数据类型存储 1753 年 1 月 1 日至 9999 年 12 月 31 日的日期和时间信息。将 date 用于 0001 年 1 月 1 日至 9999 年 12 月 31 日的日期，将 time 用于 12:00:00 AM 至 11:59:59:999 的时间。必须将不在此范围的日期作为 char 或 varchar 值进行输入、存储和处理。

- datetime 列可保存从 1753 年 1 月 1 日到 9999 年 12 月 31 日之间的日期。在支持 1/300 秒精度级别的平台上， datetime 值可精确到该级别。存储大小为 8 个字节：4 个字节用于存储自 1900 年 1 月 1 日这一基准日期以来的天数，另外 4 个字节用于存储当天的时间。
- smalldatetime 列可保存从 1900 年 1 月 1 日到 2079 年 6 月 6 日之间的日期，其精确度可以精确到分钟。存储大小为 4 个字节：2 个字节用于存储 1900 年 1 月 1 日以后的天数，另外 2 个字节用于存储自午夜以后的分钟数。
- bigdatetime 列可保存从 0001 年 1 月 1 日到 9999 年 12 月 31 日的日期以及 12:00:00.000000 AM 到 11:59:59.999999 PM 之间的时间。存储大小为 8 个字节。 bigdatetime 值精确到微秒。 bigdatetime 的内部表示是一个 64 位整数，其中包含自 0000 年 1 月 1 日以来所逝去的微秒数。
- bigtime 列可保存从 12:00:00.000000 AM 到 11:59:59.999999 PM 之间的时间。存储大小为 8 个字节。 bigtime 值精确到微秒。 bigtime 的内部表示是一个 64 位整数，其中包含自午夜以来所逝去的微秒数。
- date 是实际值，由单引号或双引号括起来的日期部分组成。该列可以保存从 0001 年 1 月 1 日至 9999 年 12 月 31 日的日期。存储大小为 4 个字节。
- time 是实际值，由单引号或双引号括起来的时间部分组成。该列可以保存从 12:00:00AM 到 11:59:59:999PM 的时间。存储大小为 4 个字节。

将日期和时间信息加以单引号或双引号。可用大写或小写字母输入，并可在数据分量间包含空格。 Adaptive Server 可识别多种数据输入格式，详细信息可参见第 8 章 “添加、更改、传输和删除数据”。但 Adaptive Server 拒绝接受诸如 0 或 00/00/00 之类的值，不会将这些值识别为日期。

日期的缺省显式格式为 “Apr 15 1987 10:23PM”。对于其它日期显示样式，可使用 convert 函数进行转换。尽管 Adaptive Server 可能会舍入或截断毫秒值，但也可使用内置日期函数对 datetime 值执行某些算术运算，除非您使用 time 数据类型。

对于 bigdatetime 和 bigtime，显示值会反映微秒精度。 bigdatetime 和 bigtime 的缺省显示格式可容纳此提高的精度。

- hh:mi:ss.zzzzzzAM 或 PM
- hh:mi:ss.zzzzzz
- mon dd yyyy
  hh:mi:ss.zzzzzz
- yyyy-mm-dd
  hh:mi:ss.zzzzzz

字符数据类型

使用字符数据类型存储由单引号或双引号内输入的字母、数字和符号组成的字符串。可使用 like 关键字来搜索字符串以查找特定的字符，并使用内置字符串函数来处理其内容。可使用 convert 函数将由数字组成的字符串转换为精确和近似数值数据类型，然后将其用于算术运算。

char(*n*) 数据类型存储固定长度的字符串，而 varchar(*n*) 数据类型以英语等单字节字符集存储可变长度的字符串。它们对应的国家 / 地区字符 nchar(*n*) 和 nvarchar(*n*)，以日语等多字节字符集存储固定和可变长度的字符串。 unichar 和 univarchar 数据类型存储常量大小的 Unicode 字符。可使用 n 指定最大字符数或使用一个字符的缺省列长度。对于大小超过页大小的字符串，可使用 text 数据类型。

表 6-3：字符数据类型

数据类型	存储
char(n)	固定长度数据，如社会保险号或邮政编码
varchar(n)	名称等长度上可能变化很大的数据
unichar	与 char 同等的固定长度的 Unicode 数据
univarchar	与 varchar 同等的在长度上可能变化很大的 Unicode 数据
nchar(n)	多字节字符集中的固定长度数据
nvarchar(n)	多字节字符集中的可变长度数据
text	将多达 2,147,483,647 个字节的可输出字符存储在各数据页的链接列表中
unitext	将多达 1,073,741,823 个 Unicode 字符存储在各数据页的链接列表中

除非设置 string_rtruncation on，否则 Adaptive Server 将条目截断到指定列的长度，而不显示警告或错误消息。有关详细信息，请参见 《参考手册》中的 set 命令。空字符串 “ ” 或 ‘ ’ 将存储为单个空格，而不是 NULL。因此， “abc” + “ ” + “def” 等于 “abc def”，而不等于 “abcdef”。

固定和可变长度列的行为稍有不同：

- 固定长度列中的数据将被用空白填充到列长度。对于 char 和 unichar 数据类型，存储大小为 *n* 个字节， (unichar = n*@@unicharsize) ；对于 nchar，存储大小为平均国家 / 地区字符长度的 *n* 倍 (*@@ncharsize*)。创建允许空值的 char、 unichar 或 nchar 列时，Adaptive Server 将其转换为 varchar、 univarchar 或 nvarchar 列，并对这些数据类型使用存储规则。但对 char 和 nchar 变量和参数就并非如此。
- 可变长度列中的数据将被去掉尾随空白，其存储大小即为数据的实际长度。对于 varchar 或 univarchar 列，它是字符数；对于 nvarchar 列，它是字符数乘以平均字符长度。可变长度字符数据要求的空间可能比固定长度数据要少，但访问它的速度会稍慢。

unichar 数据类型

unichar 和 univarchar 数据类型支持 Adaptive Server 中 Unicode 的 UTF-16 编码。这些数据类型不依赖于 char 和 varchar 数据类型，但会镜像它们的行为。

例如，对 char 和 varchar 操作的内置函数也可对 unichar 和 univarchar 进行操作。但 unichar 和 univarchar 只存储 UTF-16 字符，且与 char 和 varchar 不同，它们与缺省字符集 ID 或缺省排序顺序 ID 没有关系。

每个 unichar/univarchar 字符要求两个字节存储空间。unichar/univarchar 列的声明是 16 位 Unicode 值的数量。下例将创建一个带有 unichar 列的表，该列的 10 个 Unicode 值要求 20 个字节的存储空间：

```
create table unitbl (unicol unichar(10))
```

正如 char/varchar 列的长度， unichar/univarchar 列的长度也要受数据页大小的限制。

Unicode 代理对将使用两个 16 位 Unicode 值的存储空间，也就是四个字节。声明用于存储 Unicode 代理对的列时，要注意这一特点。缺省情况下， Adaptive Server 会正确处理代理，不会拆分对。处理截断 Unicode 数据的方式与处理 char 和 varchar 数据的方式类似。

可在任何使用 char 表达式的位置使用 unichar 表达式，包括比较运算符、连接、子查询等等。但是， unichar 和 char 的混合型表达式的执行方式与 unichar 相同。可参与此类运算的 Unicode 值的数量不能超过 unichar 字符串的最大大小。

规范化过程会修改 Unicode 数据，因此在数据库中，给定的抽象字符序列只有一种表示法。通常，后接组合读音符的字符将被预组合形式所代替。性能可因此而得到显著优化。缺省情况下，服务器假设所有 Unicode 数据都应规范化。

关系表达式

所有至少包括一个 unichar 或 univarchar 表达式的关系表达式都基于缺省 Unicode 排序顺序。如果一个表达式是 unichar，而另一个是 varchar（nvarchar、 char 或 nchar），则后者会被隐式转换为 unichar。

下列表达式最常在 where 子句中使用，在该子句中它可与逻辑运算符结合使用。

比较 Unicode 字符数据时，“less than”表示更接近缺省 Unicode 排序顺序的开头，而 “greater than”表示更接近其结尾。“Equality”表示 Unicode 缺省排序顺序对两值不加区别 （尽管它们无需相同）。例如，必须将预先组合字符 视为等于字母 e 后接 U+0302 的组合序列。如果启用了 Unicode 规范化功能 （缺省值），将自动规范化 Unicode 数据，服务器也因此永远不会看到未规范化的数据。

表 6-4：关系表达式

expr1 op_compare [any | all] (subquery)	将 any 或 all 与比较运算符和作为子查询的 expr2 一起使用，会隐式调用 min 或 max。例如，“expr1> any expr2”实际意味着 “expr1> min(expr2)”。
expr1 [not] in (expression list) expr1 [not] in (subquery)	in 运算符检查与 expr2 中的每个元素是否相等，这些元素可以是常量列表或子查询结果。
expr1 [not] between expr2 and expr3	between 运算符用于指定范围。它实际上是 “expr1 = expr2 and expr1<= expr3”的简写形式。
expr1 [not] like "match_string" [escape"esc_charî]	like 运算符用于指定要匹配的模式。与 Unicode 数据匹配的模式的语义和与规则字符数据匹配的模式的语义相同。如果 expr1 是 unichar 列名，则 “match_string”可能是 unichar 字符串或 varchar 字符串。在后一种情况下，varchar 和 unichar 之间会发生隐式转换

连接运算符

连接运算符出现的方式与比较运算符相同。实际上，连接中可使用任何比较运算符。至少包括一个 unichar 类型表达式的表达式基于缺省的 Unicode 排序顺序。如果一个表达式的类型是 unichar，而另一个类型是 varchar （nvarchar、 char 或 nchar），则后者会被隐式转换为 unichar。

Union 运算符

union 运算符对 unichar 数据的运算方式和它对 varchar 数据的运算方式非常相似。各个查询中的相应列必须可隐式转换为 unichar，或必须使用显式转换。

子句和修饰符

unichar 和 univarchar 列用于 group by 和 order by 子句时，将根据缺省 Unicode 排序顺序判断等同性。使用 distinct 修饰符时，也是如此。

text 数据类型

text 数据类型将多达 2,147,483,647 个字节的可输出字符存储在各数据页的链接列表中。每页最多可存储 1800 个字节的数据。

为节省存储空间，可将 text 列定义为 NULL。用非空值的 insert 或 update 初始化 text 列时， Adaptive Server 会指派一个文本指针，并分配一个完整的 2K 数据页用于保存值。

不能将 text 数据类型用于以下各项：

- 用于存储过程的参数 （作为传递给这些参数的值）或用于局部变量
- 用于远程过程调用 (RPC) 的参数
- 用于 order by、 compute、 group by 或 union 子句
- 用于索引
- 用于子查询或连接
- 用于 where 子句，除非带有关键字 like
- 同 + 并置运算符一起使用

如果使用通过 “组件集成服务”连接的数据库，则在处理 text 数据类型的方式上会有些不同。有关详细信息，请参见 《组件集成服务用户指南》。

有关 text 数据类型的详细信息，请参见第 322 页的 “更改 text、 unitext 和 image 数据”。

unitext 数据类型

可变长度的 unitext 数据类型最多可容纳 1,073,741,823 个 Unicode 字符（2,147,483,646 个字节）。可以在使用 text 数据类型的任何位置，使用具有相同语义的 unitext。无论 Adaptive Server 的缺省字符集是什么，unitext 列都采用 UTF-16 编码存储。

unitext 数据类型使用的存储机制与 text 相同。为节省存储空间，可将 unitext 列定义为 NULL。用非空值的 insert 或 update 子句初始化 unitext 列时， Adaptive Server 会指派一个文本指针，并分配一个完整的 2K 数据页用于保存值。

不能将 unitext 数据类型用于以下各项

- 用于存储过程的参数 （作为传递给这些参数的值）或用于局部变量
- 用于远程过程调用 (RPC) 的参数
- 用于 order by、 compute、 group by 或 union 子句

- 用于索引
- 用于子查询或连接
- 用于 where 子句，除非带有关键字 LIKE
- 同并置运算符 (+) 一起使用。

unitext 的优点包括：

- 大 Unicode 字符数据。除了 unichar 和 univarchar 数据类型，Adaptive Server 还提供了完整的 Unicode 数据类型支持，该支持最适用于增量多语种应用程序。
- unitext 以 UTF-16 存储数据，它是 Windows 和 Java 环境的本机编码。

有关 Unitext 数据类型的详细信息，请参见第 322 页的 “更改 text、unitext 和 image 数据”。

二进制数据类型

二进制数据类型采用类似十六进制的表示法存储原始二进制数据 （如图片）。二进制数据以字符 “0x” 开头，包括数字与大小写字母 A 到 F 的任意组合。 binary 和 varbinary 数据中 “0x” 后的两位数字表示数值的类型：“00” 表示正数，“01” 表示负数。

如果输入值不包括 “0x”， Adaptive Server 会假定该值为 ASCII 值并将其转换。

注释 Adaptive Server 对不同平台的二进制类型采取不同的处理方式。对于真十六进制数据，使用 hextoint 和 inttohex 函数。请参见第 16 章 “在查询中使用内置函数”。

使用 binary(n) 和 varbinary(n) 数据类型存储多达 255 个字节长度的数据。每个存储字节可保存 2 个二进制位。用 *n* 指定列的长度，或使用 1 个字节的缺省长度。如果输入了一个长于 *n* 的值， Adaptive Server 会在不显示警告或错误的情况下将该条目截断为指定的长度。

- 对于预计其中所有条目在长度上都大致相等的数据，可使用固定长度二进制类型 binary(n)。由于 binary 列中的条目会被用零填充到列长度，因此它们可能会比 varbinary 列中的条目要求占用的存储空间更大，但访问它们的速度会稍快。

- 对于预计长度会变化很大的数据，使用可变长度二进制类型 varbinary(n)。存储大小是输入的数据值的实际大小，不是列长度。尾随零将被截断。

创建允许空值的 binary 列时，Adaptive Server 会将其转换为 varbinary 列，并使用该数据类型的存储规则。

可用 like 关键字搜索二进制字符串，并用内置字符串函数对其进行运算。由于输入某个特定值的确切格式取决于所使用的硬件，因此*在不同平台上进行涉及二进制数据的计算可能会产生不同的结果。*

image 数据类型

使用 image 数据类型将较大的二进制数据块存储到外部数据页上。image 列可在与表的其它数据存储分离的数据页的链接列表上存储多达 2,147,483,647 个字节的数据。

用非空值的 insert 或 update 初始化 image 列时，Adaptive Server 会指派一个文本指针，并分配一个完整的 2K 数据页用于保存值。每页最多可存储 1800 个字节。

为了节省存储空间，可将 image 列定义为 NULL。要添加 image 数据，而又不想在事务日志中记录大块的二进制数据，可使用 writetext。有关 writetext 的详细信息，请参见 《参考手册》。

不能将 image 数据类型用于以下各项：

- 用于存储过程的参数 （作为传递给这些参数的值）或用于局部变量
- 用于远程过程调用 (RPC) 的参数
- 用于 order by、 compute、 group by 或 union 子句
- 用于索引
- 用于子查询或连接
- 用于 where 子句，除非带有关键字 like
- 同 + 并置运算符一起使用
- 用于触发器的 if update 子句

如果使用通过 “组件集成服务”连接的数据库，则在处理 image 数据类型的方式上会有些不同。有关详细信息，请参见 《组件集成服务用户指南》。

有关 image 数据类型的详细信息，请参见第 322 页的 “更改 text、unitext 和 image 数据”。

bit 数据类型

将 bit 列用于真 / 假或是 / 否类型的数据。bit 列保存 0 或 1。可接受 0 或 1 以外的其它整数值，但始终会将这些值解释为 1。存储大小为 1 个字节。表中的多个 bit 数据类型被收集到字节中。例如，7-bit 列将占用 1 个字节；而 9-bit 列则要占用 2 个字节。

数据类型为 bit 的列不能为 NULL，且其中不能有索引。syscolumns 系统表中的 status 列用于指示 bit 列的唯一偏移位置。

timestamp 数据类型

对于可在 Open Client™ DB-Library 应用程序中浏览到的表中的列，用户定义的 timestamp 数据类型是必需的。

每次插入或更新包含 timestamp 列的行时， timestamp 列都会自动更新。一个表只能有一列 timestamp 数据类型。名为 timestamp 的列将自动具有系统数据类型 timestamp。它的定义为：

```
varbinary(8) "NULL"
```

因为 timestamp 是用户定义数据类型，所以不能用它定义其它用户定义数据类型。必须全部用小写字母将其输入为 “timestamp”。

sysname 和 *longsysname* 数据类型

sysname 和 longsysname 是用于系统表的用户定义数据类型。sysname 定义为：

```
varchar(30) "NOT NULL"
```

longsysname 定义为：

```
varchar(255) "NOT NULL"
```

可以将列、参数或变量声明为 sysname 类型。或者，您也可以使用 sysname 或 longsysname 的基本类型创建用户定义的数据类型。

然后使用该**用户定义的数据类型**创建列。有关详细信息，请参见 第 208 页的 “创建用户定义的数据类型”。

数据类型之间的转换

Adaptive Server 可自动处理许多从一种数据类型到另一种数据类型的转换。这些转换称作隐式转换。可用 convert、 inttohex 和 hextoint 函数显式请求其它转换。其它转换，无论是显式转换还是自动转换，都因数据类型间不兼容而无法执行。

例如，为进行比较， Adaptive Server 将把 char 表达式自动转换为 datetime，如果可将它们作为 datetime 值解释。但为方便显示，必须使用 convert 函数将 char 转换为 int。同样，如果想要 Adaptive Server 将其视作字符数据，以便将 like 关键字与其一同使用，则必须对整数数据使用 convert。

convert 函数的语法是：

convert (*datatype*, *expression*, [*style*])

在下例中，为显示以数字 2 开头的所有销售额， convert 使用 char 数据类型显示 total_sales 列：

```
select title, total_sales
from titles
where convert(char(20), total_sales) like "2%"
```

可选的 style 参数用于将 datetime 值转换为 char 或 varchar 数据类型，以获得多种日期显示格式。

有关 convert、 inttohex 和 hextoint 函数的详细信息，请参见第 16 章 “在查询中使用内置函数”。

混合型算术和数据类型层次

对具有不同数据类型的值执行算术运算时， Adaptive Server 必须确定结果的数据类型，在某些情况下，还要确定结果的长度和精度。

每个系统数据类型都有**数据类型层次**，数据类型层次存储在 systypes 系统表中。用户定义的数据类型继承了它们所基于的系统类型的层次。

下列查询按照层次来排列数据库中的数据类型。除了下面显示的信息之外，查询结果还将包括有关数据库中任意用户定义的数据类型的信息：

```
select name, hierarchy
from systypes
order by hierarchy

name            hierarchy
```

```
---------- ---------
floatn              1
float               2
datetimn            3
datetime            4
real                5
numericn            6
numeric             7
decimaln            8
decimal             9
moneyn             10
money              11
smallmoney         12
smalldatet         13
intn               14
uintn              15
bigint             16
ubigint            17
int                18
uint               19
smallint           20
usmallint          21
tinyint            22
bit                23
univarchar         24
unichar            25
unitext            26
sysname            27
varchar            27
nvarchar           27
longsysnam         27
char               28
nchar              28
timestamp          29
varbinary          29
binary             30
text               31
image              32
date               33
time               34
daten              35
timen              36
bigdatetim         37
bigtime            38
bigdatetim         39
bigtimen           40
extended t         99
```

注释　u<int type> 是一种内部表示形式。无符号类型的正确语法是 unsigned {int | integer | bigint | smallint }

数据类型层次决定了使用不同数据类型的值所进行的计算的结果。最终所得的值将被指派给最靠近列表顶部的数据类型。

在下例中， sales 表中的 qty 与 roysched 表中的 royalty 相乘。 qty 是 smallint，其层次为 20 ； royalty 是 int，其层次为 18。因此，结果的数据类型是 int。

```
smallint(qty) * int(royalty) = int
```

本例将层次为 18 的 int 类型与层次为 19 的 unsigned int 类型相乘，结果的数据类型为 int：

```
int(10) * unsigned int(5) = int(50)
```

注释　使用混合模式表达式时，无符号的整数总是升级到有符号的数据类型。如果无符号的整数值不在有符号的整数范围内， Adaptive Server 将发出转换错误消息。

请参见 《参考手册：构件块》，了解有关数据类型层次结构的更多信息。

使用 *money* 数据类型

如果要组合 money 与文字或变量，且需要 money 类型的结果，可使用 money 文字或变量：

```
create table mytable
(moneycol money,)
insert into mytable values ($10.00)
select moneycol * $2.5 from mytable
```

如果要组合 money 与列值中的 float 或 numeric 数据类型，可使用 convert 函数：

```
select convert (money, moneycol * percentcol)
    from debits, interest
drop table mytable
```

确定精度和标度

对于 numeric 和 decimal 类型，精度和标度的每一种组合都是一个不同的 Adaptive Server 数据类型。如果对两个 numeric 或 decimal 值执行算术运算，其中 n1 精度为 p1，标度为 s1， n2 精度为 p2，标度为 s2，则 Adaptive Server 所确定的结果的精度和标度将如表 6-5 中所示：

表 6-5：算术运算后的精度和标度

运算	精度	标度
n1 + n2	max(s1, s2) + max(p1 -s1, p2 - s2) + 1	max(s1, s2)
n1 - n2	max(s1, s2) + max(p1 -s1, p2 - s2) + 1	max(s1, s2)
n1 * n2	s1 + s2 + (p1 - s1) + (p2 - s2) + 1	s1 + s2
n1 / n2	max(s1 + p2 + 1, 6) + p1 - s1 + p2	max(s1 + p2 - s2 + 1, 6)

创建用户定义的数据类型

可使用 SQL 的 Transact-SQL 增强功能命名并设计自己的数据类型，作为对系统数据类型的补充。用户定义的数据类型是根据系统数据类型定义的。可以给经常使用的数据类型定义命名。这使自定义适用于列的数据类型变得更加容易。

注释 若要在多个数据库中使用用户定义数据类型，请在 model 数据库中创建该数据类型。该用户定义的数据类型就会对所有新创建的数据库生效。

一旦定义了数据类型，就可以在数据库的任一列中使用。例如， tid 用作多个 pubs2 表中列的数据类型：titles.title_id、 titleauthor.title_id、sales.title_id 和 roysched.title_id.

用户定义的数据类型的优点是，可将规则和缺省值绑定到它们上面以便在多个表中使用。有关该主题的详细信息，请参见第 14 章 “为数据定义缺省值和规则”。

使用 sp_addtype 创建用户数据类型。它将创建的用户数据类型名、创建它的 Adaptive Server 提供的数据类型及可选的空、非空或标识规范作为参数。

可使用除 timestamp 外的任何系统数据类型建立用户定义的数据类型。用户定义的数据类型可与其所基于的系统数据类型拥有相同的数据类型层次。与 Adaptive Server 提供的数据类型不同，用户定义的数据类型名区分大小写。

sp_addtype 的语法为：

```
sp_addtype datatypename,
      phystype [(length) | (precision [, scale])]
   [, "identity" |nulltype]
```

定义 tid：

```
sp_addtype tid, "char(6)", "not null"
```

如果参数包含空白或某种形式的标点符号，或如果参数是 null 以外的关键字 （例如， identity 或 sp_helpgroup），则必须使用单引号或双引号将其引起来。在本例中，因为有括号，所以需为 char(6) 加引号，但因为有空白，所以需为 “not null”加引号。不需要为 tid 加引号。

使用 *identity* 属性创建用户定义的数据类型

可使用 sp_addtype 创建具有 identity 属性的用户定义的数据类型，但必须仅为标度为 0 的数值数据类型指定 identity 属性。

指定长度、精度和标度

建立基于某种 Adaptive Server 数据类型的用户定义的数据类型时，必须指定额外的参数：

- char、 nchar、 varchar、 nvarchar、 binary 以及 varbinary 数据类型要求将长度加以括号。如果未提供长度， Adaptive Server 将假设其为缺省的 1 个字符的长度。
- float 数据类型要求将精度加以括号。如果未提供精度， Adaptive Server 将使用所用平台的缺省精度。
- numeric 和 decimal 数据类型要求精度和标度加以括号，并用逗号分隔。如果未提供精度和标度， Adaptive Server 将使用缺省的精度 18 和标度 0。

在 create table 语句中加入用户定义的数据类型时，不能更改长度、精度或标度规范。

指定空值类型

空值类型决定用户定义数据类型如何处理空值。可创建带有 “null”、“NULL”、“nonull”、“NONULL”、“not null”或 “NOT NULL”空值类型的用户定义数据类型。 bit 和 identity 类型不允许空值。

如果省略了空值类型， Adaptive Server 将使用为数据库定义的空值模式（缺省为 “NOT NULL”）。为与 SQL 标准兼容，可使用 sp_dboption 将 allow nulls by default 选项设置为真。

在 create table 语句中加入用户定义的数据类型时，可覆盖空值类型。

将规则和缺省值与用户定义的数据类型相关联

创建用户定义的数据类型后，可使用 sp_bindrule 和 sp_bindefault 将规则和缺省值与数据类型相关联。使用 sp_help 输出列有规则、缺省值和其它与数据类型相关联的信息的报告。

对规则和缺省值的论述详见第 14 章 “为数据定义缺省值和规则”。有关系统过程及其语法的完整信息，请参见 《参考手册》。

创建具有 IDENTITY 属性的用户定义的数据类型

要创建具有 IDENTITY 属性的用户定义的数据类型，可使用 sp_addtype。新类型必须基于标度为 0 的物理类型 numeric 或任何整数类型：

```
sp_addtype typename, "numeric (precision, 0)",
"identity"
```

下例创建一个具有 IDENTITY 属性的用户定义数据类型 IdentType：

```
sp_addtype IdentType, "numeric(4,0)", "identity"
```

从 IDENTITY 类型创建列时，可在 create 或 alter table 语句中指定 identity 或 not null，或者两者都不指定。该列自动继承 IDENTITY 属性。

以下是从 IdentType 用户定义的类型创建 IDENTITY 列的三种不同方法：

```
create table new_table (id_col IdentType)
drop table new_table
create table new_table (id_col IdentType identity)
drop table new_table
create table new_table (id_col IdentType not null)
drop table new_table
```

注释　如果试图从 IDENTITY 类型创建允许空值的列，则 create table 或 alter table 语句将失败。

从用户定义的数据类型创建 IDENTITY 列

可从不具有 IDENTITY 属性的用户定义的数据类型创建 IDENTITY 列。

用户定义的类型必须具有标度为 0 的物理数据类型 numeric，或者具有任何整数类型，且必须定义为 not null。

删除用户定义的数据类型

若要删除用户定义数据类型，请执行：

sp_droptype *typename*

注释　不能删除任何表中正在使用的数据类型。

获取有关数据类型的信息

使用 sp_help 显示有关系统数据类型或用户定义的数据类型的属性的信息。报告指示了从其创建数据类型的基本类型、是否允许空值、绑定到数据类型的任何规则和缺省值的名称、是否具有 IDENTITY 属性。

下例显示了有关 money 系统数据类型和 tid 用户定义数据类型的信息：

```
sp_help money
Type_name  Storage_type Length Prec  Scale
---------- ------------ ------ ----- -----
money      money             8 NULL  NULL
Nulls   Default_name  Rule_name   Identity
-----   ------------  ---------   --------
    1   NULL          NULL            NULL
(return status = 0)

sp_help tid
Type_name  Storage_type Length Prec  Scale
---------- ------------ ------ ----- -----
tid        varchar           6 NULL  NULL
Nulls   Default_name  Rule_name   Identity
-----   ------------  ---------   --------
    0   NULL          NULL                0

(return status = 0)
```

第 7 章

创建数据库和表

本章阐述如何创建数据库和表。

何为数据库和表？

数据库在一组数据库对象中存储信息 （数据），如互相关联的表。表是拥有含单个数据项的相关列的行集合。创建数据库和表时应定义数据的组织方式。这一过程称为*数据定义*。

Adaptive Server 数据库对象包括：

- 表
- 规则
- 缺省值
- 存储过程
- 触发器

- 视图
- 参照完整性约束
- 检查完整性约束
- 函数
- 计算列
- 分区条件

本章讲述数据库和表的创建、修改和删除，包括完整性约束。

列和**数据类型**用于定义表中包括的数据类型，本章也将就此展开讨论。索引用于说明表中数据的组织方式。它们是 Adaptive Server 所不考虑的数据库对象，并不列在 sysobjects 中。对索引的论述详见第 13 章 “创建表的索引”。

强制实现数据库的数据完整性

数据完整性指数据库中数据的正确性与完全性。为强制实现数据的完整性，可约束或限制用户可在数据库中插入、删除或更新的数据值。例如， pubs2 和 pubs3 数据库中的数据完整性要求：titles 表中的书名必须对应 publishers 表的一个出版者。不能将无有效出版者的书插入 titles，因为它会破坏 pubs2 或 pubs3 的数据完整性。

Transact-SQL 为在数据库中强制实现完整性提供了几种机制，如规则、缺省值、索引和触发器。通过这些机制可以维护以下类型的数据完整性：

- 要求 — 要求表列的每一行中都必须包含一个有效值；不允许使用空值。可以使用 create table 语句限制列的空值。
- 检查或有效性 — 限制或限定插入到表列中的数据值。可使用触发器或规则来强制实现此类完整性。
- 唯一性 — 在任何两个表行中，不能有一个或多个表列具有相同的非空值。可用索引来强制实现此类完整性。
- 参照 — 插入到表列中的数据必须在其它表列或相同表的其它列中已拥有匹配数据。一个单表最多可拥有 192 个参照。

数据库中数据值的一致性也属于数据完整性 （如第 22 章 “事务：维护数据一致性和恢复”所述）。

除使用规则、缺省值、索引和触发器外， Transact-SQL 还提供了一系列的**完整性约束**作为 create table 语句的一部分，用以强制实现 SQL 标准定义的数据完整性。这些完整性约束将在本章的以后部分加以阐述。

数据库中的权限

是否可以创建和删除数据库和数据库对象，取决于您的权限或特权。通常，系统管理员或数据库所有者会依据您的工作和所需要的功能为您设置权限。在特定的系统或数据库中，每个用户的这些权限可以不同。

可通过执行如下命令确定您的权限：

```
sp_helprotect user_name
```

user_name 为您的 Adaptive Server 登录名。

为使数据库对象试验尽可能便捷， pubs2 和 pubs3 数据库在其 sysusers 系统表中包含一个 **guest** 用户名。创建 pubs2 和 pubs3 的脚本赋予“guest”多种权限。

“guest”机制意味着登录到 Adaptive Server 的任何人（即列在 master..syslogins 中的任何人）都有权访问 pubs2 和 pub3，有权创建和删除如表、索引、缺省值、规则或过程等对象。“guest”用户名也允许使用某些存储过程、创建用户定义的数据类型、查询数据库以及修改其中的数据。

要使用 pubs2 或 pubs3 数据库，可发布 use 命令。Adaptive Server 会检查 pubs2.sysusers 或 pubs3..sysusers 中是否列有您的姓名。如果没有，则认为您为访客，对您没有任何操作。如果您的姓名列在 pubs2 或 pubs3 的 sysusers 表中，则 Adaptive Server 承认您的身份，可能给您一些不同于“guest”身份的权限。

注释　本章中的所有示例都假定将您视为“guest”。

大多数用户都可通过先前描述的“guest”机制查看 master 数据库中的系统表。对于在 master 数据库中无法识别名称的用户，允许其进入，但将其视为“guest”用户。在安装用于创建 master 数据库的脚本时，将在 master 数据库中添加“guest”用户。

数据库所有者“dbo”可使用 sp_adduser 向任何用户数据库添加“guest”用户。系统管理员可自动成为所用任意数据库中的数据库所有者。有关详细信息，请参见《系统管理指南：卷 1》中的第 13 章“Adaptive Server 中的安全性管理快速入门”。

使用和创建数据库

数据库是相关表和其它数据库对象 （视图、索引等）的集合。

安装 Adaptive Server 时，它包含以下**系统数据库**：

- master — 从整体上控制用户数据库和 Adaptive Server 操作。
- sybsystemprocs — 包含系统存储过程。
- sybsystemdb — 包含有关分布式事务的信息。
- tempdb — 存储临时对象，包括创建的具有名称前缀 “tempdb..” 的临时表。
- model — Adaptive Server 将其用作创建新用户数据库的模板。

此外，系统管理员可安装以下可选数据库：

- pubs2 — 包含表示发布操作的数据的样本数据库。可使用此数据库测试服务器的连接并了解 Transact-SQL。 Adaptive Server 文档中的大多数示例都使用 pubs2 数据库。
- pubs3 — pubs2 的一个使用参照完整性示例的版本。 pubs3 有一个使用自参照列的表 store_employees。 pubs3 的 sales 表中还包括 IDENTITY 列。此外，其主表中的主键使用非聚簇唯一索引， titles 表有一个 numeric 数据类型的示例。
- interpubs — 与 pubs2 类似，但包含法文和德文数据。
- jpubs — 与 pubs2 类似，但包含日文数据。安装日文模块后，即可使用。

这些可选数据库为用户数据库。用户的全部数据都存储在用户数据库中。 Adaptive Server 通过系统表来管理每一个数据库。 master 数据库和其它数据库中的**数据字典**表均被视为系统表。

选择数据库：*use*

可以使用 use 命令来访问现有数据库。它的语法为：

```
use database_name
```

例如，若要访问 pubs2 数据库，请输入：

```
use pubs2
```

只有当您是 pubs2 中的已知用户时，此命令才允许您访问 pubs2 数据库。否则，会出现错误消息。

在登录到 Adaptive Server 上时，系统可能会自动将您连接到 master 数据库，因此，要使用其它数据库，应发出 use 命令。您或系统管理员可以使用 sp_modifylogin 来更改最初连接到的数据库。但只有系统管理员才可以更改另一用户的缺省数据库。

创建用户数据库：*create database*

如果系统管理员授予您使用 create database 的权限，则您可创建新数据库。创建新数据库时，必须正在使用 master 数据库。在许多公司中，由系统管理员创建全部的数据库。数据库的创建者即为其所有者。为您创建数据库的另一用户可通过 sp_changedbowner 转交数据库的所有权。

数据库所有者负责赋予用户访问数据库的权限，以及负责授予或撤消某些其它权限。在某些组织中，数据库所有者也有责任维护数据库的定期备份，并在系统出现故障时重新装载数据库。数据库所有者可使用 setuser 命令，来临时获得其他任何用户对数据库的权限。

因为为每个数据库分配了相当大的空间，所以即使它只包含少量数据，您也可能无权使用 create database。

最简单的 create database 形式如下：

```
create database database_name
```

若要创建名为 newpubs 的新数据库，请先确保使用的是 master 数据库而非 pubs2，然后输入以下命令：

```
use master
create database newpubs
drop database newpubs
use pubs2
```

Adaptive Server 上的数据库名称必须是唯一的，并且必须遵循第 7 页的“标识符”中描述的标识符规则。 Adaptive Server 最多可管理 32,767 个数据库。一次只能创建一个数据库。任何数据库的最大段数均为 32。

Adaptive Server 创建一个新的数据库，作为 model 数据库的副本，其中包含属于每个用户数据库的系统表。

新数据库的创建记录在 master 数据库表 sysdatabases 和 sysusages 中。

请参见 《参考手册：命令》和 《参考手册：表》。

本章介绍除 with override 外的所有 create database 选项。有关 with override 的信息，请参见 《系统管理指南：卷 2》中的第 6 章 “创建和管理用户数据库”。

on 子句

可以使用可选的 on 子句指定数据库的存储位置及为其分配的空间大小（以兆字节为单位）。如果使用关键字 default，则会将数据库分配给一个可用的数据库设备，该设备位于 master 数据库表 sysdevices 中指示的缺省数据库设备池中。使用 sp_helpdevice 可查看缺省列表中的设备。

注释 系统管理员可以根据性能统计信息和其它考虑因素进行一些存储分配。创建数据库之前，应向系统管理员核实分配情况。

要为即将存储在此缺省位置处的数据库指定 5MB 大小，可使用 on default = *size*：

```
use master
create database newpubs
on default = 5
drop database newpubs
use pubs2
```

要为数据库指定其它位置，应给出要存储此数据库的数据库设备的逻辑名。一个数据库可以存储于多个数据库设备上，每个设备都有不同的空间量。

下例创建 newpubs 数据库，并在 pubsdata 上为其分配 3MB 空间，而在 newdata 上为其分配 2MB 空间：

```
create database newpubs
on pubsdata = 3, newdata = 2
```

如果省略 on 子句和大小，则数据库创建在 sysdevices 指示的缺省数据库设备池中，并占用 2MB 空间。

数据库分配的大小介于 2MB 到 2^{23}MB 之间。

log on 子句

除非要创建很小且不太重要的数据库，否则务必使用 create database 的 log on database_device 扩展。这样可将事务日志放到单独的数据库设备上。之所以要将日志放到单独的设备上，是因为以下几种原因：

- 它允许使用 dump transaction 而不是 dump database，从而节省了时间和空间。
- 允许建立固定大小的日志，防止它与其它数据库活动争用空间。

将日志放在与数据表所在的不同的*物理*设备上的其它原因是：

- 提高性能。
- 确保在硬盘出现故障时能完全恢复。

以下命令将 newpubs 的日志放在逻辑设备 pubslog 上，其大小为 1MB：

```
create database newpubs
on pubsdata = 3, newdata = 2
log on pubslog = 1
```

注释　在使用 log on 扩展时，数据库事务日志将放在名为“logsegment”的段上。若要为现有日志添加更多的空间，请使用 alter database；在某些情况下，应使用 sp_extendsegment。有关详细信息，请参见《参考手册：命令》、《参考手册：过程》或《系统管理指南：卷 2》中的第 8 章“创建和使用段”。

事务日志所需的设备大小因更新活动量和事务日志转储频率不同而有所不同。作为一般准则，为日志分配的空间应为分配的数据库空间的 10-25%。

for load 选项

可选 for load 子句调用 create database 的精简形式，这种形式只能用于装载数据库转储。for load 选项用于从介质故障中恢复，或用于将数据库从一台计算机移动到另一台计算机。请参见《参考手册：命令》和《系统管理指南：卷 2》中的第 12 章“备份和恢复用户数据库”。

quiesce database 命令

也可以使用以下命令，将数据库置于休眠模式：

```
quiesce database
```

此命令可挂起和恢复对指定的一系列数据库进行的更新。

变更数据库的大小

如果数据库的分配存储空间已全部填满，则不能再向其添加新数据或进行更新。将始终保留现有数据。如果为数据库分配的空间过小，则数据库所有者可用 alter database 命令增加空间。缺省情况下， alter database 权限授予给数据库所有者，且不能移交。若要使用 alter database，必须正在使用 master 数据库。

缺省情况下，增加缺省空间池中的 2MB 空间。以下语句为缺省数据库设备上的 newpubs 增加 2MB 空间：

```
alter database newpubs
```

完整的 alter database 语法允许将数据库扩展指定的兆字节数 （最小 1MB），并允许指定增加存储空间的位置：

```
alter database database_name
    [on {default | database_device} [= size]
        [, database_device [= size]]...]
    [log on {default | database_device} [ = size ]
        [, database_device [= size]]...]
    [with override]
    [for load]
```

请参见 《参考手册：命令》。

alter database 命令中的 on 子句与 create database 中的 on 子句相似。for load 子句与 create database 中的 for load 子句相似，并只能用于使用 for load 子句创建的数据库。

若要在数据库设备 pubsdata 上为 newpubs 增加 2MB 的分配空间，并在数据库设备 newdata 上为其增加 3MB 的分配空间，请输入：

```
alter database newpubs
on pubsdata = 2, newdata = 3
```

当使用 alter database 在已由数据库使用的设备上分配更多空间时，该设备上所有已有段均可使用增加的空间片段。已映射到现有段的所有对象现在均可扩展到增加的空间内。任何数据库的最大段数均为 32。

当使用 alter database 在未由数据库使用的设备上分配空间时，system 和 default 段被映射到新设备。若要更改此段映射，请使用 sp_dropsegment 从设备中删除不想要的段。请参见 《参考手册》

注释 使用 sp_extendsegment 会自动取消系统与缺省段的映射。

有关 with override 的信息，请参见 《系统管理指南：卷 2》中的第 6 章 “创建和管理用户数据库”。

删除数据库

drop database 命令用于删除数据库。 drop database 可从 Adaptive Server 删除数据库及其所有内容，释放为其分配的存储空间，并从 master 数据库删除对它的引用。

语法为：

drop database *database_name* [, *database_name*]...

不能删除使用中的数据库，即由其它用户打开进行读写操作的数据库。

如前述，可用一个命令删除多个数据库。例如：

```
drop database newpubs, newdb
```

可以使用 drop database 删除损坏的数据库。如果 drop database 不起作用，可以使用 dbcc dbrepair 修复损坏的数据库，然后再删除它。

创建表

创建表时，应为其列命名并为每个列提供数据类型。也可指定特定行能否存在空值，或指定表中各列的完整性约束。每个数据库可有 2,000,000,000 个表。

对象名称或标识符的长度限制如下：常规标识符为 255 个字节，分隔标识符为 253 个字节。此限制适用于大多数用户定义的标识符，包括表名、列名和索引名等。由于这些扩展限制，一些系统表 （目录）和内置函数也进行了扩展。

对于变量，“@”计为 1 个字节，它允许的名称长度为 254 个字节。

每个表的最大列数

表中的最大列数取决于许多因素，包括服务器的逻辑页大小以及表是为所有页配置的还是为仅数据锁定配置的等。请参见 《参考手册：命令》中的 create table。

创建表示例

可以使用上一节中创建的 newpubs 数据库来试用这些示例。否则，这些更改会影响其它数据库，如 pubs2 或 pubs3。

create table 的最简单形式如下：

```
create table table_name
(column_name datatype)
```

例如，若要创建名为 names 且包含 *some_name* 列 （固定长度为 11 字节）的表，请输入：

```
create table names
(some_name char(11))
drop table names
```

如果使用 set quoted_identifier on 命令，则表名和列名均可为分隔标识符。否则，它们必须遵循第 7 页的 “标识符”中描述的标识符规则。在同一个表中，列名必须唯一，但同一数据库的不同表中可使用相同的列名。

每个列都必须有数据类型。上例中列名后的单词 “char”是指该列的数据类型，即该列将包含的值的类型。数据类型在第 6 章 “使用和创建数据类型”中论述。

数据类型后面的小括号中的数字决定了可在列中存储的最大字节数。某些数据类型的最大长度可由用户指定。而其它数据类型则使用系统定义的长度。

请用小括号将列名列表括起来，并在每个列定义后面以逗号分隔。最后一个列定义后不需要加逗号。

注释 如果缺省值是 create table 语句的一部分，则不能在缺省值中使用变量。

有关 create table 的完整描述，请参见 《参考手册》。

选择表名

create table 命令可在当前打开的数据库中建立新表。对于每个用户，表名必须是唯一的。

创建临时表有两种途径，一种途径是在 create table 语句中的表名前加井号 (#)，另一种途径是指定表名前缀 “tempdb..”。有关详细信息，请参见第 230 页的 “使用临时表”。

您可使用任何由您创建且未限定名称的表或其它对象。也可使用由数据库所有者创建且未限定名称的对象 （只要有相应的权限）。这些规则适用于所有用户，包括系统管理员和数据库所有者。

不同用户可创建相同名称的表。例如，名为 “jonah” 的用户和名为 “sally” 的用户可各自创建一个名为 info 的表。拥有这两个表权限的用户必须将它们分别限定为 jonah.info 和 sally.info。Sally 必须将对 Jonah 的表的引用限定为 jonah.info，而对自己的表的引用只需使用 info。对象名称或标识符的长度限制如下：常规标识符为 255 个字节，分隔标识符为 253 个字节。

对于变量，“@” 计为 1 个字节，它允许的名称长度为 254 个字节。

将表创建于不同的数据库中

正如 create table 语法所示，通过用其它数据库的名称限定表名，可在当前数据库以外的数据库中创建表。但是，您必须是要创建表的数据库的授权用户，且必须具有其中的 create table 权限。

如果使用的是 pubs2 或 pubs3，并且存在另一个名为 newpubs 的数据库，则可以在 newpubs 中创建一个名为 newtab 的表，方法如下：

```
create table newpubs..newtab (col1 int)
```

不能在当前数据库以外的数据库中创建其它数据库对象，如视图、规则、缺省值、存储过程和触发器。

create table 语法

create table 语句：

- 定义表中的每一列。
- 提供列名和数据类型，指定每个列对空值的处理方式。
- 指定哪列 （如果有）有 IDENTITY 属性。
- 定义列级完整性约束和表级完整性约束。每个表定义的每一列和每个表可有多个约束。

例如，对 pubs2 数据库中的 titles 表执行的 create table 语句为：

```
create table titles
(title_id tid,
title varchar(80) not null,
type char(12),
pub_id char(4) null,
price money null,
advance money null,
royalty int null,
total_sales int null,
notes varchar(200) null,
pubdate datetime,
contract bit not null)
```

请参见 《参考手册：命令》。

以下几节介绍表定义的组成部分：系统提供的数据类型、用户定义的数据类型、空类型和 IDENTITY 列。

注释 对 create table 的 on *segment_name* 扩展允许将表放到现有段上。*segment_name* 指向一个特定的数据库设备或一组数据库设备。在段上创建表之前，请向系统管理员或数据库所有者索取您可以使用的段列表。由于性能原因或其它方面的考虑，某些段可能会被分配给特定的表或索引。

使用 IDENTITY 列

IDENTITY 列包含对应每一行的一个值，由 Adaptive Server 自动生成，在表内唯一地标识该行。

每个表只能有一个 IDENTITY 列。可在创建表时用 create table 或 select into 语句定义 IDENTITY 列，或在以后用 alter table 语句添加。

可通过在 create table 语句中指定关键字 identity 而不是 null 或 not null 来定义 IDENTITY 列。IDENTITY 列的数据类型必须为 numeric 且标度为零，或者为任何整数类型。请在新表中将 IDENTITY 列定义为任何所需精度（1 到 38 位）：

```
create table table_name
    (column_name numeric(precision ,0) identity)
```

可能的最大列值为 $10^{精度} - 1$。例如，以下命令创建一个表，其 IDENTITY 列允许的最大值为 $10^5 - 1$ （即 9999）：

```
create table sales_daily
    (sale_id numeric(5,0) identity,
    stor_id char(4) not null)
```

可以使用 auto identity 数据库选项和 size of auto identity 配置参数来创建自动 IDENTITY 列。若要在非唯一索引中包括 IDENTITY 列，请使用 identity in nonunique index 数据库选项。

注释　缺省情况下， Adaptive Server 以值 1 开始编写行号，并随着行的增加而连续编写行号。某些活动 （如手动插入、删除或事务回退、服务器关机或故障）会使 IDENTITY 列的值出现间隔。 Adaptive Server 提供了几种控制标识间隔的方法，详见第 233 页的 “管理表中的标识间隔”中的介绍。

创建用户定义数据类型的 IDENTITY 列

可以使用用户定义的数据类型创建 IDENTITY 列。用户定义的数据类型的基础类型必须为 numeric 且标度为零，或者为任何整数类型。如果使用 IDENTITY 属性来创建用户定义的数据类型，则在创建该列时不必重复 identity 关键字。

下例显示一个有 IDENTITY 属性的用户定义数据类型：

```
sp_addtype ident, "numeric(5)", "identity"
```

下例显示一个基于 ident 数据类型的 IDENTITY 列：

```
create table sales_monthly
```

```
(sale_id ident, stor_id char(4) not null)
```

如果用户定义的类型创建为 not null，则必须在 create table 语句中指定 identity 关键字。不能从允许空值的用户定义数据类型创建 IDENTITY 列。

引用 IDENTITY 列

创建引用 IDENTITY 列的表列时，应确保任何被引用列与 IDENTITY 列具有相同的数据类型定义。例如，在 pubs3 数据库中， sales 表被定义为以 ord_num 列作为 IDENTITY 列：

```
create table sales
(stor_id char(4) not null
      references stores(stor_id),
ord_num numeric(6,0) identity,
date datetime not null,
unique nonclustered (ord_num))
```

ord_num IDENTITY 列被定义为唯一约束，它需要使用该约束来引用 salesdetail 中的 ord_num 列。 salesdetail 定义如下：

```
create table salesdetail
(stor_id char(4) not null
     references storesz(stor_id),
ord_num numeric(6,0)
     references salesz(ord_num),
title_id tid not null
     references titles(title_id),
qty smallint not null,
discount float not null)
```

向 sales 中插入一行后，向 salesdetail 中插入一行的简单方法是使用 *@@identity* 全局变量，将 IDENTITY 列值插入 salesdetail。 @@identity 全局变量存储最近生成的 IDENTITY 列值。例如：

```
begin tran
insert sales values ("6380", "04/25/97")
insert salesdetail values ("6380", @@identity, "TC3218", 50, 50)
commit tran
```

此示例属于一个事务，因为这两次插入操作互相依赖、互为基础。例如，如果 sales 插入失败，则 *@@identity* 的值不相同，从而向 salesdetail 中插入错误的行。因为这两次插入同属于一个事务，因此如果一个失败，整个事务都会被拒绝。

有关 IDENTITY 列的详细信息，请参见第 313 页的 “用 @@identity 检索 IDENTITY 列值”。有关事务的信息，请参见第 22 章 “事务：维护数据一致性和恢复”。

使用 *syb_identity* 引用 IDENTITY 列

定义了 IDENTITY 列后，就无需记住实际的列名。必要时，可在对表的 select、 insert、 update 或 delete 语句中使用由表名限定的 syb_identity 关键字。

例如，以下查询选择 *sale_id* 等于 1 的行：

```
select * from sales_monthly
    where syb_identity = 1
```

自动创建 “隐藏的” IDENTITY 列

系统管理员可使用 auto identity 数据库选项自动将 10 位的 IDENTITY 列包含在新表中。要在数据库中打开此功能，可使用：

```
sp_dboption database_name, "auto identity", "true"
```

用户每次创建新表而未指定 primary 键、unique 约束或 IDENTITY 列时，Adaptive Server 都会自动定义一个 IDENTITY 列。使用 select * 检索表中的所有列时， IDENTITY 列是不可见的。必须在选择列表中显式地包含列名 SYB_IDENTITY_COL （全部为大写字母）。如果启用了 “组件集成服务”，则代理表的自动 IDENTITY 列称为 OMNI_IDENTITY_COL。

若要设置自动 IDENTITY 列的精度，可使用 size of auto identity 配置参数。例如，若要将 IDENTITY 列的精度设置为 15，请使用：

```
sp_configure "size of auto identity", 15
```

允许在列中使用空值

可为每一列指定是否允许空值。空值与 “零” 或 “空白” 不同。NULL 意味着尚未输入内容，通常暗指 “值未知” 或 “值不适用”。它表示由于某种原因，用户没有输入任何内容。例如， titles 表的 price 列中的空值不表示该书是免费的，而只表示价格未知，或尚未设置。

如果用户在使用关键字 null 定义的列中未输入任何内容，则 Adaptive Server 提供值 “NULL”。使用关键字 null 定义的列也接受用户显式输入的 NULL，而无论其属于何种数据类型。在字符列中输入空值时应小心。如果将 “null” 一词放在单引号或双引号中， Adaptive Server 会将该条目解释为字符串，而非 NULL 值。

如果在 create table 语句中省略 null 或 not null，则 Adaptive Server 使用为数据库定义的空值模式 （缺省情况下为 NOT NULL）。使用 sp_dboption 将 allow nulls by default 选项设置为 true。

必须在定义为 NOT NULL 的列中输入条目；否则，Adaptive Server 会显示一条错误消息。

将列定义为 NULL，会为未知数据提供一个占位符。例如，在 titles 表中， price、 advance、 royalty 和 total_sales 均设置为允许 NULL。

但 title_id 和 title 不能这样设置，因为这些列中如果缺少条目则是无意义的，会引起混乱。没有标题的价格毫无意义，但没有价格的标题则只意味着价格尚未确定，或此价格不可用。

当列中的信息对于其它列的意义至关重要时，应在 create table 中使用 not null。

与空值相关的约束和规则

不能定义一个允许空值的列，接着又用禁止空值的约束或规则覆盖此定义。例如，如果一个列定义指定 NULL，规则指定：

```
@val in (1,2,3)
```

隐式或显式 NULL 与规则不发生冲突。此列定义覆盖此规则，即使规则指定：

```
@val is not null
```

有关约束的详细信息，请参见第 241 页的 “为表定义完整性约束”。规则在第 14 章 “为数据定义缺省值和规则”中详细介绍。

缺省值和空值

对 NULL 和 NOT NULL 列均可使用缺省值 （即当不输入条目时，系统自动提供的值）。缺省值与输入条目的处理方式相同。但是，不能为 NOT NULL 列指定 NULL 缺省值。可使用 create table 的 default 约束或使用 create default 将空值指定为缺省值。default 约束将在本章的后面部分介绍； create default 详见第 14 章 “为数据定义缺省值和规则”。

如果在创建列时指定 NOT NULL，并且没有为其创建缺省值，则当用户在插入操作期间未在该列中输入任何内容时，将显示一条错误消息。此外，用户不能对以 NULL 为值的列进行 insert 或 update 操作。

表 7-1 说明当用户未指定列值或显式输入 NULL 值时，列的缺省值及其空值类型之间的交互作用。三个可能结果是空值、缺省值或一条错误消息。

表 7-1：列定义和空缺省值

列定义	用户输入	结果
已定义空和缺省值	不输入值	使用缺省值
	输入 NULL 值	使用 NULL
已定义 Null，未定义缺省值	不输入值	使用 NULL
	输入 NULL 值	使用 NULL
非空，已定义缺省值	不输入值	使用缺省值
	输入 NULL 值	使用 NULL
非空，未定义缺省值	不输入值	错误
	输入 NULL 值	错误

Null 需要可变长度数据类型

只有具有可变长度数据类型的列才能存储空值。用固定长度的数据类型创建 NULL 列时， Adaptive Server 会将其转换为相应的可变长度数据类型。关于类型更改， Adaptive Server 不作通知。

表 7-2 列出固定长度数据类型及 Adaptive Server 将它们转换到的可变长度数据类型。某些可变长度的数据类型 （如 moneyn）是保留的数据类型；不能使用它们来创建列、变量或参数。

表 7-2：固定长度数据类型向可变长度数据类型的转换

原始的固定长度数据类型	转换为
char	varchar
nchar	nvarchar
unichar	univarchar
binary	varbinary
datetime	datetimn
float	floatn
bigint、 int、 smallint、 tinyint、	intn
unsigned bigint、 unsigned int 和 unsigned smallint	uintn
decimal	decimaln
numeric	numericn
money 和 smallmoney	moneyn

在 char、 nchar、 unichar 和 binary 列中输入的数据遵循可变长度列的规则，而不是用空格或零填补至列规范的完整长度。

text、*unitext* 和 *image* 列

使用 insert 和 NULL 创建的 text、unitext 和 image 列未初始化，它们不包含任何值。它们不占用存储空间，不能用 readtext 或 writetext 访问。

使用 update 将 NULL 值写入 text、unitext 或 image 列时，将初始化该列，将一个指向该列的有效文本指针插入表中，并为该列分配一个 2K 的数据页。列初始化后，即可通过 readtext 和 writetext 命令进行访问。请参见《参考手册：命令》。

使用临时表

临时表创建于 tempdb 数据库中。若要创建临时表，必须对 tempdb 有 create table 权限。 create table 权限缺省授予数据库所有者。

若要生成临时表，应在 create table 语句中的表名前使用井号 (#) 或“tempdb..”。

临时表分为两种：

- 可在 Adaptive Server 会话之间共享的表

  通过将 tempdb 指定为 create table 语句中表名的一部分，可以创建可共享临时表。例如，下列语句创建可在 Adaptive Server 会话之间共享的临时表：

  ```
  create table tempdb..authors
  (au_id char (11))
  drop table tempdb..authors
  ```

  Adaptive Server 不更改以此方式创建的临时表名。该表会一直存在，直至当前会话结束或者表的所有者使用 drop table 将其删除。

- 只能由当前 Adaptive Server 会话或过程访问的表

  通过在 create table 语句中的表名前面指定井号 (#)，可以创建非共享临时表。例如：

  ```
  create table #authors
  (au_id char (11))
  ```

  该表会一直存在，直至当前会话或过程结束，或者其所有者用 drop table 将其删除。

如果不在表名前使用井号或 “tempdb..”，且当前未使用 tempdb，则表被创建为永久表。永久表会一直保留在数据库中，直至被其所有者显式删除。

此语句创建非共享临时表：

```
create table #myjobs
(task char(30),
start datetime,
stop datetime,
notes varchar(200))
```

可以使用此表保留今日杂事和任务的列表、开始和结束时间的记录以及可能有的任何注释。在当前工作会话结束时，将自动删除此表及其数据。临时表不可恢复。

可将规则、缺省值和索引与临时表建立关联，但不能在临时表上创建视图，或将触发器与之建立关联。创建临时表时，只有在用户定义的数据类型存在于 tempdb..systypes 中时，才可使用该数据类型。

要仅为当前会话向 tempdb 添加对象，则应在使用 tempdb 时执行 sp_addtype。若要永久性地添加对象，请在 model 中执行 sp_addtype，然后重新启动 Adaptive Server，以将 model 复制到 tempdb 中。

确保临时表名唯一

为确保临时表名对于当前会话是唯一的， Adaptive Server：

- 必要时，将表名截断为 238 个字节，包括井号 (#)
- 附加一个 17 位的数字后缀，它对于 Adaptive Server 会话来说是唯一的

以下示例显示一个创建为 #temptable 并存储为 #temptable00000050010721973 的表：

```
use pubs2
go
create table #temptable (task char(30))
go
use tempdb
go
select name from sysobjects where name like
    "#temptable%"
go

name
------------------------------
#temptable00000050010721973

(1 row affected)
```

在存储过程中处理临时表

存储过程可引用在当前会话期间创建的临时表。在存储过程中，不能先创建临时表、将其删除，然后再用相同名称创建新的临时表。

名称以 “#”开头的临时表

退出存储过程时，在该存储过程中创建的带有以 “#”开头的名称的临时表将会消失。单个过程能够：

- 创建临时表
- 将数据插入表中
- 在表上运行查询
- 调用引用该表的其它过程

因为要创建引用临时表的过程，临时表必须存在，所以要遵循以下步骤：

1 使用 create table 创建临时表。

2 创建访问临时表的过程，但不创建生成该表的过程。

3 删除临时表。

4 创建一个过程，该过程创建表并调用在第 2 步中创建的过程。

名称以 *tempdb..* 开头的临时表

从存储过程内部，可使用 create table tempdb..tablename 创建不带 # 前缀的临时表。过程完成后这些表不消失，所以它们可被独立过程引用。可按以上步骤创建这些表。

警告！ *仅当*要在用户和会话之间共享表时，才应该在存储过程中创建具有 “tempdb..”前缀的临时表。创建和删除临时表的存储过程应使用 # 前缀来避免无意的共享。

临时表的一般规则

名称以 # 开头的临时表应遵循如下限制：

- 不能在这些表上创建视图。
- 不能将触发器与这些表建立关联。
- 无法判断出是哪个会话或过程创建了这些表。

这些限制不适用于在 tempdb 中创建的可共享的临时表。

应用于这两类临时表的规则如下：

- 可将规则、缺省值和索引与临时表建立关联。当临时表消失时，在临时表上创建的索引随之消失。
- 系统过程 （如 sp_help）可使用临时表，但条件是必须从 tempdb 调用它们。
- 在临时表中不能使用用户定义的数据类型，除非该数据类型存在于 tempdb 中；即，除非自上次 Adaptive Server 重新启动以后，该数据类型已显式创建于 tempdb 中。
- 要对临时表执行 select into 操作，不必将 select into/bulkcopy 选项设置为 on。

管理表中的标识间隔

IDENTITY 列包含一个由 Adaptive Server 生成的、表中每一行的唯一 ID 号。由于缺省情况下由服务器生成 ID 号，因此可能偶尔会使 ID 号有很大的间隔。 identity_gap 参数可控制特定表中的 ID 号及其可能的间隔。

缺省情况下，Adaptive Server 会在内存中分配一个 ID 号块，而不是将每个 ID 号按需写入磁盘，这样做需要更多的处理时间。服务器会将每个块的最大号写入表的对象分配映射 (OAM) 页。当前分配的号码块已使用或 “烧毁”后，此号码将用作下一个块的起始点。该块的其它号码保留在内存中，但不保存到磁盘上。当号码被分配到内存，随后因为被指派到行中或因为某些反常事件 （如系统故障）而被从内存中删除时，这些号码就认为被 “烧毁”了。

分配 ID 号码块减少了对表的争用，从而提高了性能。但是，在 ID 号码全部指派完之前，如果服务器出现故障或用 no wait 关机，则未使用的号码被烧毁。当服务器再次运行时，它会以先前写入磁盘的块的最高号码为基础，开始为下一号码块编号。依据出现故障前被指派到行中的已分配号码的多少， ID 号可能会有很大的间隔。

标识间隔也可能由转储和装载活动数据库产生。转储时，数据库对象被保存到 OAM 页中。如果当前正在使用某个对象，则 maximum used identity value 不在 OAM 页中，因此不会对其进行转储。

控制标识间隔的参数

Adaptive Server 提供了用于控制标识号间隔的参数 （如表 7-3 所述）。

表 7-3：控制标识间隔的参数

参数名	范围	使用	说明
identity_gap	特定于表	create table 或 select into	为特定表创建特定大小的 ID 号块。覆盖该表的 identity burning set factor。使用 identity grab size。
identity burning set factor	全服务器范围的	sp_configure	指出要为每个块分配的总可用 ID 号码的百分比。使用 identity grab size。如果将表的 identity_gap 设置为 1 或更高值，则 identity burning set factor 对该表无影响。 burning set factor 用于 identity_gap 设置为 0 的所有表。 设置 identity burning set factor 时，以十进制形式表示数字，然后乘以 10,000,000 (10^7) 以得到用于 sp_configure 的正确值。例如，要每次释放 15% (.15) 的可能 IDENTITY 列值，指定的值应为 .15 乘以 10^7 （即 1,500,000）： sp_configure "identity burning set factor", 1500000
identity grab size	全服务器范围的	sp_configure	为每一进程保留连续的 ID 号码块。使用 identity burning set factor 和 identity_gap。

identity burning set factor 与 *identity_gap* 的比较

使用 identity_gap 参数可控制特定表的标识间隔的大小 （如下例所示）。在下例中，已创建了名为 books 的表，列有书店中的所有书籍。我们希望每本书都有唯一的 ID 号，并希望 Adaptive Server 能自动生成 ID 号。

使用 *identity burning set factor* 的示例

为 books 表定义 IDENTITY 列时，我们使用了缺省的数值 (18, 0)，提供总共 999,999,999,999,999,999 个 ID 号。对于 identity burning set factor 配置参数，我们使用缺省设置 5000 （999,999,999,999,999,999 的 0.05%），即表示 Adaptive Server 分配 500,000,000,000,000 个号码的号码块。

服务器在内存中分配第一批 500,000,000,000,000 个号码，并将该块的最高号码 (500,000,000,000,000) 存储在表的 OAM 页中。当所有号码均被指派到行中或烧毁后，服务器会取来下一个号码块 （下一批 500,000,000,000,000 个号码），此块从 500,000,000,000,001 开始，并将 1,000,000,000,000,000 存储为该块的最高号码。

假定，在行号 500,000,000,000,022 后，服务器出现故障。只有号码 1 到 500,000,000,000,022 被用作 books 的 ID 号。号码 500,000,000,000,023 到 1,000,000,000,000,000 被烧毁。当服务器再次运行时，会创建以存储在表的 OAM 页中的最高号加一 (1,000,000,000,000,001) 开始的 ID 号，留下 499,999,999,999,978 个号码间隔。

若要为特定表缩减这样大的标识号码间隔，可以按下面的 “使用 identity_gap 的示例”中所述来设置标识间隔。

使用 *identity_gap* 的示例

在此示例中，我们用 identity_gap 值 1000 创建 books 表。这样会覆盖能产生 500,000,000,000,000 个 ID 号码的号码块的全服务器范围 identity burning set factor 设置。更改后，服务器会在内存中分配 1000 个 ID 号码的号码块。

服务器分配第一批的 1000 个号码，并将该块的最高号码 (1000) 存储到磁盘中。所有号码均被使用后，服务器会取来下一批 1000 个号码，以 1001 开始，并将 2000 作为最高号码存储。

现在，假定在行号 1002 之后停电了，这会导致服务器出现故障。则只有号码 1000 到 1002 被使用，而号码 1003 到 2000 丢失。当服务器再次运行时，会创建以存储在表的 OAM 页中的最高号加一 (2000) 开始的 ID 号，只留下 998 个号码间隔。

通过为表设置 identity_gap 而不使用全服务器范围的 table burning set factor，可极大地减少 ID 号码的间隔。但是，如果此值设置得过低，会使性能降低。服务器每次都必须将块的最高号码写入磁盘，因此性能受到影响。例如，如果 identity_gap 设置为 1，即表示一次分配一个 ID 号，服务器必须在每次创建行时都写入新号码，这样会在表上出现页锁争用，从而可能会降低性能。必须找到可获得最佳性能、而对您的具体情况来说间隔值又最低的最佳设置。

设置特定于表的标识间隔

使用 create table 或 select into 创建表时，应设置特定于表的标识间隔。

用 *create table* 设置标识间隔

语法为：

```
select IdNum into newtable
with identity_gap=20
from mytable
--------
```

例如：

```
create table mytable (IdNum numeric(12,0) identity)
with identity_gap = 10
```

此语句创建一个带 identity 列的名为 mytable 的表。标识间隔设置为 10，表示将在内存中以十个 ID 号的块为单位分配 ID 号。如果服务器出现故障或没有等待就关闭服务器，那么分配给行的最后一个 ID 号与分配给行的下一个 ID 号之间的差距为十个号码。

用 *select into* 设置标识间隔

如果用 select into 语句从具有特定标识间隔设置的表创建表，则新表不继承父表的标识间隔设置。而使用 identity burning set factor 设置。若要为新表指定一个特定的 identity_gap 设置，则应在 select into 语句中指定标识间隔。可以为新表指定一个与父表相同或不同的标识间隔。

例如，若要通过具有标识间隔的现有表 (mytable) 来创建新表 (newtable)，请使用以下命令：

```
select IdNum into newtable
with identity_gap = 20
from mytable
```

更改特定于表的标识间隔

若要更改特定表的标识间隔，请使用 sp_chgattribute：

```
sp_chgattribute "table_name", "identity_gap", set_number
```

其中：

- *table_name* 为要更改标识间隔的表的名称。
- identity_gap 表示要更改标识间隔。
- *set_number* 是标识间隔的新大小。

例如：

```
sp_chgattribute "mytable", "identity_gap", 20
```

若要更改 mytable 以使用 identity burning set factor 设置而不是 identity_gap 设置，请将 identity_gap 设置为 0：

```
sp_chgattribute "mytable", "identity_gap", 0
```

显示特定于表的标识间隔信息

若要查看表的 identity_gap 设置，请使用 sp_help。

例如， identity_gap 列 （接近输出末尾）的零值表示未设置特定于表的标识间隔。 mytable 使用全服务器范围的 identity burning set factor 值。

```
sp_help mytable

Name    Owner    Object_type    Create_date
------- -------- -------------- ----------------------------
mytable dbo      user table     Nov 29 2004 1:30PM

(1 row affected)
Column_name     Type   Length Prec Scale Nulls Default_name   Rule_name
        Access_Rule_name  Computed_Column_object   Identity


--------------- --------------- ------ ---- ----- ----- ---------------
    --------------- --------
IdNum           numeric              6   12     0     0 NULL     NULL
        NULL              NULL                                  1
Object does not have any indexes.


No defined keys for this object.
name    type       partition_type   partitions   partition_keys
```

```
------- ---------- ---------------- ------------ ---------------
mytable base table roundrobin                   1 NULL

partiton_name       partition_id   pages     segment   create_date
-----------------   -------------- --------- --------- -----------------------
-
mytable_1136004047  1136004047            1  default     Nov 29 2004 1:30PM

partition_conditions
--------------------
NULL

Avg_pages   Max_pages   Min_pages   Ratio(Max/Avg)         Ration(Min/Avg)
----------- ----------- ----------- ---------------------- -------------------
          1           1           1               1.000000                1.000
000


                        Lock scheme Allpages
                        The attribute 'exp_row_size' is not applicable to tables
                        with
                        allpages lock scheme.
                        The attribute 'concurrency_opt_threshold' is not appli-
                        cable to
                        tables with allpages lock scheme.

                        exp_row_size reservepagegap fillfactor
                        max_rows_per_page identity_gap


                        ------------ ------------- ---------- ---------------
                        -- ------------


                                    1               0          0
                          0            0
                        concurrency_opt_threshold   optimistic_index_lock   de
                        alloc_first_txtpg
                        --------------------------- ----------------------- --
                        ------------------
                                                0                           0
                                          0
                        (return status = 0)
```

如果将 mytable 的 identity_gap 更改为 20，则该表的 sp_help 输出在 identity_gap 列中显示 20。此设置会覆盖全服务器范围的 identity burning set factor 值。

```
sp_help mytable

Name    Owner    Object_type    Create_date
------- -------- -------------- ----------------------------
mytable dbo      user table     Nov 29 2004 1:30PM

(1 row affected)
Column_name     Type   Length Prec Scale Nulls Default_name   Rule_name
        Access_Rule_name  Computed_Column_object   Identity


--------------- --------------- ------ ---- ----- ----- ---------------
    --------------- --------
IdNum           numeric              6   12     0     0 NULL     NULL
        NULL              NULL                                  1
Object does not have any indexes.


No defined keys for this object.
name    type       partition_type   partitions   partition_keys
------- ---------- ---------------- ------------ ---------------
mytable base table roundrobin                  1 NULL

partiton_name       partition_id   pages     segment   create_date
-----------------   -------------- --------- --------- ------------------------
-
mytable_1136004047  1136004047            1  default     Nov 29 2004 1:30PM

partition_conditions
--------------------
NULL


Avg_pages   Max_pages   Min_pages   Ratio(Max/Avg)        Ration(Min/Avg)
----------- ----------- ----------- --------------------- -------------------
          1           1           1              1.000000                1.000
000
Lock scheme Allpages
The attribute 'exp_row_size' is not applicable to tables with
allpages lock scheme.
The attribute 'concurrency_opt_threshold' is not applicable to
tables with allpages lock scheme.

exp_row_size reservepagegap fillfactor max_rows_per_page identity_gap
```

```
------------ -------------- ---------- ----------------- ------------
           1              0          0                 0            0
concurrency_opt_threshold   optimistic_index_lock   dealloc_first_txtpg
--------------------------- ----------------------- --------------------
                          0                       0                    0
(return status = 0)
```

由于其它原因产生的间隔

手动插入到 IDENTITY 列中、删除行、 identity grab size 配置参数的值以及事务回退都可使 IDENTITY 列值产生间隔。这些间隔不受 identity burning set factor 配置参数设置的影响。

例如，假设有一带有如下值的 IDENTITY 列：

```
select syb_identity from stores_cal

 id_col
 -------
      1
      2
      3
      4
      5

(5 rows affected)
```

可删除 IDENTITY 列值在 2 和 4 之间的所有行，使列值出现间隔：

```
delete stores_cal
where syb_identity between 2 and 4
select syb_identity from stores_cal

id_col
------
      1
      5

(2 rows affected)
```

为表设置 identity_insert on 后，表所有者、数据库所有者或系统管理员可手动插入任何大于 5 的合法值。例如，如果插入值 55，则会使 IDENTITY 列值产生较大的间隔：

```
insert stores_cal
(syb_identity, stor_id, stor_name)
```

```
values (55, "5025", "Good Reads")
select syb_identity from stores_cal

id_col
 -------
       1
       5
      55

(3 rows affected)
```

如果随后将 identity_insert 设置为 off，则 Adaptive Server 在下一次插入时为 IDENTITY 列指派值 55 + 1 （即 56）。如果回退包含 insert 语句的事务， Adaptive Server 将放弃值 56，并在下一次插入时使用值 57。

当表插入达到 IDENTITY 列的最大值时

可插入表中的最大行数取决于 IDENTITY 列的精度设置。如果某表达到该限制，则可以使用较高精度重新创建该表，或者该表的 IDENTITY 列未用于参照完整性，则可以使用 bcp 实用程序删除间隔。有关详细信息，请参见第 314 页的 “达到 IDENTITY 列的最大值”。

为表定义完整性约束

Transact-SQL 为维护数据库中的数据完整性提供了两种方法：

- 定义规则、缺省值、索引和触发器
- 定义 create table 完整性约束

选择哪种方法取决于您的需求。完整性约束的优点表现在：在创建表的进程中一步完成完整性控件的定义 （如 SQL 标准规定），并可简化创建这些完整性控件的过程。但是，与缺省值、规则、索引和触发器相比，完整性约束的范围较为有限，并且综合性也较差。

例如，触发器对参照完整性的处理方式比在 create table 中声明的方法更为复杂。 create table 语句定义的完整性约束是针对该表的；不能将它们绑定到其它表上，并且只能使用 alter table 对其进行删除或更改。即使在同一张表上，约束也不能包含子查询或集合函数。

这两种方法不是互斥的。可以把完整性约束与缺省值、规则、索引与触发器结合使用。在应用中可灵活选择最佳方法。本部分介绍 create table 完整性约束。缺省值、规则、索引和触发器将在后面的章节中加以介绍。

可创建以下类型的约束：

- unique 和 primary key 约束要求在一个表中，任意两行的特定列中均不能有相同的值。此外， primary key 约束要求列的任何行中都不能有空值。
- 参照完整性 (references) 约束要求特定列中插入的数据应在指定表和列中有与之相匹配的数据。使用 sp_helpconstraint 查找表的被引用表。
- check 约束限制插入到列中的数据值。

也可通过以下方法来强制实现数据完整性：即限制在列中使用空值（null 或 not null 关键字）；为列提供缺省值（default 子句）。有关 null 和 not null 关键字的信息，请参见第 227 页的 “允许在列中使用空值”。

有关为表定义的任何约束的信息，请参见第 292 页的 “使用 sp_helpconstraint 查找表的约束信息”。

警告！ 不要定义或变更系统表的约束定义。

指定表级或列级约束

可在表级或列级声明完整性约束。列级约束和表级约束之间存在差异，但用户很少会注意到这一差异。即，仅当修改列中的值时才会检查列级约束；而如果对行进行任何修改，都会检查表级约束，而不管是否更改了有关的列。

列级约束应放在列名和数据类型后面，但在分隔逗号前面。表级约束应作为单独的逗号分隔子句输入。 Adaptive Server 同等对待表级约束和列级约束；两种方式同等有效。

但是，必须将操作于多个列的约束声明为表级约束。例如，以下 create table 语句有一个操作于两个列 （pub_id 和 pub_name）的 check 约束：

```
create table my_publishers
(pub_id      char(4),
pub_name     varchar(40),
constraint my_chk_constraint
    check (pub_id in ("1389", "0736", "0877")
        or pub_name not like "Bad News Books"))
```

可将作用于一个列的约束声明为列级约束，但这并不是必需的。例如，如果上面的 check 约束只使用一个列 (pub_id)，则可将该约束放到该列上：

```
create table my_publishers
(pub_id     char(4) constraint my_chk_constraint
        check (pub_id in ("1389", "0736", "0877")),
pub_name     varchar(40))
```

在这两种情况下， constraint 关键字和伴随的 *constraint_name* 都是可选的。 check 约束在第 249 页的 “指定检查约束”中介绍。

注释　不能使用 check 约束发出 create table，然后在同一批处理或过程中将数据插入到表中。可以将创建和插入语句分放在两个不同的批处理或过程中，或者使用 execute 分别执行操作。

为约束创建错误消息

可创建错误消息并将其绑定到约束，方法为：使用 sp_addmessage 创建消息，然后使用 sp_bindmsg 将其绑定到约束。

例如：

```
sp_addmessage 25001,
     "The publisher ID must be 1389, 0736, or 0877"
sp_bindmsg my_chk_constraint, 25001
insert my_publishers values
     ("0000", "Reject This Publisher")

Msg 25001, Level 16, State 1:
Server 'snipe', Line 1:
The publisher ID must be 1389, 0736, or 0877
Command has been aborted.
```

若要更改约束的消息，请绑定一条新消息。新消息会替换旧消息。

使用 sp_unbindmsg 将消息与约束解除绑定；使用 sp_dropmessage 删除用户定义的消息。

例如：

```
sp_unbindmsg my_chk_constraint
sp_dropmessage 25001
```

若要更改消息文本，但保留相同的错误号，请解除绑定消息，使用 sp_dropmessage 删除它，然后使用 sp_addmessage 再次添加它，并使用 sp_bindmsg 绑定它。

创建检查约束后

创建检查约束之后，描述检查约束的**源文本**存储在 syscomments 系统表的 text 列中。*不要从 syscomments 中删除此信息*；否则将来升级 Adaptive Server 时可能会出现问题。出于安全考虑，可以使用 sp_hidetext 来加密 syscomments 中的文本 （如 《参考手册：过程》）中所述。请参见第 3 页的 “编译对象”。

指定缺省列值

定义任何列级完整性约束之前，可用 default 子句为该列指定缺省值。**缺省子句**可为某列指派缺省值，它是 create table 语句的一部分。当用户不输入列的值时， Adaptive Server 会插入该缺省值。

可将下列值用于 default 子句：

- *constant_expression* — 指定要用作列缺省值的常量表达式。它不能包含任何列或其它数据库对象的名称，但可以包括不引用数据库对象的内置函数。该缺省值必须与列的数据类型兼容。
- user — 指定 Adaptive Server 插入用户名作为缺省值。若要使用此缺省值，列的数据类型必须为 char(30) 或 varchar(30)。
- null — 指定 Adaptive Server 插入空值作为缺省值。不能为不允许空值的列定义此缺省值 （使用 notnull 关键字）。

例如，以下 create table 语句定义两个列缺省值：

```
create table my_titles
(title_id        char(6),
title            varchar(80),
price            money       default null,
total_sales      int         default 0)
```

表中的每个列只能包括一个 default 子句。

使用 default 子句指派缺省值比两步 Transact-SQL 方法要简单。在 Transact-SQL 中，可使用 create default 声明缺省值，然后使用 sp_bindefault 将其绑定到列。

指定唯一约束和主键约束

可声明 unique 或 primary key 约束，来确保一个表中的任意两行的指定列中没有相同的值。这两个约束创建唯一索引，以强制实现此数据完整性。然而， primary key 约束比 unique 约束的限制性更强。带 primary key 约束的列不能包含 NULL 值。通常表的 primary key 约束应与其它表上定义的参照完整性约束结合使用。

SQL 标准的 unique 约束定义规定，列定义不允许使用空值。缺省情况下，如果在列定义中省略 null 或 not null， Adaptive Server 会将列定义为不允许空值（如尚未用 sp_dboption 进行更改）。在 Transact-SQL 中，可以将列定义为在 unique 约束下允许空值，因为用于强制约束的唯一索引允许插入空值。

注释　不要将唯一和主键完整性约束与 sp_primarykey、 sp_foreignkey 和 sp_commonkey 定义的信息相混淆。唯一和主键约束实际上会创建索引来定义表列的唯一或主键属性。 sp_primarykey、 sp_foreignkey 和 sp_commonkey 为通过创建索引和触发器强制的表列定义键 （在 syskeys 表中）的逻辑关系。

缺省情况下， unique 约束创建唯一的非聚簇索引； primary key 约束创建唯一的聚簇索引。可声明带有任意类型约束的聚簇或非聚簇索引。

例如，以下 create table 语句使用表级 unique 约束，来确保 stor_id 和 ord_num 列中没有哪两行具有相同的值：

```
create table my_sales
(stor_id     char(4),
ord_num      varchar(20),
date         datetime,
unique clustered (stor_id, ord_num))
```

一个表只能有一个聚簇索引，所以只能指定一个 unique clustered 或 primary key clustered 约束。

强制数据完整性时，可使用 unique 和 primary key 约束来创建唯一索引（包括 with fillfactor、with max_rows_per_page 和 on *segment_name* 选项）。但是，索引可提供附加功能。有关索引及其选项的信息 （包括聚簇索引与非聚簇索引之间的差别），请参见第 13 章 “创建表的索引”。

指定参照完整性约束

参照完整性指用于管理表间关系的方法。创建表时，可定义约束，以确保插入特定表中的数据在另一表中有匹配值。

可在表中定义的引用有三种类型：对另一个表的引用、从另一个表的引用和自身引用 （即同一个表中的引用）。

以下两个来自 pubs3 数据库的表说明了声明参照完整性的工作方式。第一个表 stores 是 “被参照”的表：

```
create table stores
(stor_id     char(4) not null,
stor_name    varchar(40) null,
stor_address varchar(40) null,
city         varchar(20) null,
state        char(2) null,
country      varchar(12) null,
postalcode   char(10) null,
payterms     varchar(12) null,
unique nonclustered (stor_id))
```

第二个表 store_employees 是 “引用”表，因为它包含对 stores 表的引用。它还包含自身参照：

```
create table store_employees
(stor_id     char(4) null
       references stores(stor_id),
emp_id       id not null,
mgr_id       id null
       references store_employees(emp_id),
emp_lname    varchar(40) not null,
emp_fname    varchar(20) not null,
phone        char(12) null,
address      varchar(40) null,
city         varchar(20) null,
state        char(2) null,
country      varchar(12) null,
postalcode   varchar(10) null,
unique nonclustered (emp_id))
```

store_employees 表中定义的引用强制执行下列限制：

- store_employees 表中指定的任何商店都必须包括在 stores 表中。references 约束规定任何插入到 store_employees 的 stor_id 列中的值必须已存在于 my_stores 的 stor_id 列中，从而强制执行这一限制。
- 所有管理人员都必须有员工标识号。 references 约束规定任何插入到 mgr_id 列中的值必须已存在于 emp_id 列中，从而强制执行这一限制。

表级或列级参照完整性约束

可在列级或表级定义参照完整性约束。前例中的参照完整性约束是使用 create table 语句中的 references 关键字在列级定义的。

定义表级参照完整性约束时，应包括 foreign key 子句和一个或多个列名的列表。 foreign key 指定当前表中列出的列是外键，这些外键的目标键是随后的 references 子句中列出的列。例如：

```
constraint sales_detail_constr
     foreign key (stor_id, ord_num)
     references my_salesdetail(stor_id, ord_num)
```

foreign key 语法只允许用于表级约束，而不允许用于列级约束。有关详细信息，请参见第 242 页的 “指定表级或列级约束”。

在列级或表级定义参照完整性约束后，可以使用 sp_primarykey、sp_foreignkey 和 sp_commonkey 在 syskeys 系统表中定义键。

表中允许的最大参照数

表中允许的最大参照数为 192。可使用 sp_helpconstraint 来检查表的参照（如第 292 页的 “使用 sp_helpconstraint 查找表的约束信息”所述）。

为交叉引用约束使用 *create schema*

不能创建参照不存在的表的表。若要创建两个或多个相互参照的表，请使用 create schema。

模式是特定用户拥有的对象以及这些对象的关联权限的集合。如果 create schema 语句中的任何语句失败，整个命令将作为一个单元回退，无生效的语句。

create schema 语法为：

create schema authorization *authorization name*
create_object_statement
  [create_object_statement ...]
[*permission_statement* ...]

例如：

```
create schema authorization dbo
    create table list1
        (col_a char(10) primary key,
        col_b char(10) null
        references list2(col_A))
    create table list2
        (col_A char(10) primary key,
```

```
    col_B char(10) null
    references list1(col_a))
```

创建参照完整性约束的一般规则

在表中定义参照完整性约束时：

- 确保拥有对被引用表的 references 权限。请参见 《系统管理指南》中的第 17 章 “管理用户权限”。
- 确保被参照的列受被引用表中唯一索引的约束。可使用 unique 或 primary key 约束或者 create index 语句来创建唯一索引。例如，stores 表中被引用的列定义如下：

  ```
  stor_id char(4) primary key
  ```

- 确保用在参照定义中的列有匹配的数据类型。例如，在 my_stores 和 store_employees 中的 stor_id 列都是使用 char(4) 数据类型创建的。store_employees 中的 mgr_id 和 emp_id 列都是用 id 数据类型创建的。
- 只有在通过 primary key 约束将被引用表中的列指定为主键时，才可在 references 子句中省略列名。
- 在被引用表中，不能更新与引用表中的值相匹配的列值或删除与引用表中的值相匹配的行。应该先从引用表中执行删除或更新，然后再从被引用表中执行删除。

  同样，也不能对被引用表使用 truncate table 命令。而应首先截断引用表，然后再截断被引用表。
- 在删除被引用表之前，必须先删除引用表；否则将违反约束。
- 使用 sp_helpconstraint 查找表的被引用表。

参照完整性约束为强制数据完整性提供了比使用触发器更为简单的方式。但是，触发器为强制表间的参照完整性提供附加能力。有关详细信息，请参见第 20 章 “触发器：强制实施参照完整性”。

指定检查约束

可声明 check 约束来限制用户可在表列中插入的值。检查约束对于检查有限的具体值域的应用程序非常有用。 check 约束指定了值在插入表之前必须通过的 *search_condition*。 *search_condition* 可包括：

- 使用 in 引入的一系列常量表达式
- 用 between 引入的一系列常量表达式
- 使用 like 引入的一组条件，可以包含通配符

表达式可包括算术运算符和 Transact-SQL 内置函数。*search_condition* 不能包含子查询、集合函数说明或目标说明。

例如，以下语句确保只有某些特定的值才能输入 pub_id 列：

```
create table my_new_publishers
(pub_id      char(4)
        check (pub_id in ("1389", "0736", "0877",
             "1622", "1756")
        or pub_id like "99[0-9][0-9]"),
pub_name     varchar(40),
city         varchar(20),
state        char(2))
```

列级检查约束仅可参照定义约束的列；不能引用表中的其它列。表级检查约束可参照表中任意列。 create table 允许列定义中有多个检查约束。

因为检查约束不替代列定义，所以，如果列定义允许空值，就不能使用检查约束来禁止空值。如果对允许空值的列声明检查约束，即使 *search_condition* 中不包含 NULL 值，也可以在列中显式或隐式地插入 NULL 值。例如，假定对允许空值的表列定义下列检查约束：

```
check (pub_id in ("1389", "0736", "0877", "1622",
"1756"))
```

您仍可将 NULL 插入该列。因为以下表达式始终判断为真，所以列定义覆盖检查约束：

```
col_name != null
```

设计使用参照完整性的应用程序

设计使用参照完整性功能的应用程序时：

- 不创建不必要的参照约束。表中具有的参照约束越多，需要参照完整性的语句在表上运行得越慢。
- 在表上尽可能少地使用自参照约束。
- 对于检查有限的特定值范围的应用程序，应使用 check 约束而不是 references 约束。由于没有参照，使用 check 约束时不再要求 Adaptive Server 扫描其它表以完成查询。因此，对这样的表运行查询的速度快于对使用参照的表运行查询。

  例如，该表使用 check 约束将作者限定在加利福尼亚州：

  ```
  create table cal_authors
  (au_id id not null,
  au_lname varchar(40) not null,
  au_fname varchar(20) not null,
  phone char(12) null,
  address varchar(40) null,
  city varchar(20) null,
  state char(2) null
       check(state = "CA"),
  country varchar(12) null,
  postalcode char(10) null)
  ```

- 将普通扫描的外键索引绑定到各自的高速缓存中，以优化性能。唯一索引在声明为主键的列上自动创建。当相应的外键被更新或插入时，通常选择这些索引扫描被引用表。
- 将候选键的多行更新保持在最低限度。
- 将参照完整性查询放入使用约束检查的过程中。约束检查被编入执行计划；当参照约束改变时，具有该约束的过程在执行时被自动重新编译。
- 如果不能将参照完整性查询内置于过程中，且频繁地在特别批处理中运行参照完整性，那么将系统目录 sysreferences 绑定到其高速缓存上。当 Adaptive Server 必须重新编译参照完整性查询时，

  这可提高性能。

- 创建具有参照约束的表之后，在使用该表运行查询之前，使用 set showplan, noexec on 对其进行测试。showplan 输出指示运行查询所需的辅助扫描描述符的数量；扫描描述符管理在其上运行查询时表的扫描。如果辅助扫描描述符的数量很大，则重新设计该表使其使用较少扫描描述符，或者增加 number of auxiliary scan descriptors 配置参数的值。

如何设计和创建表

本节给出一个 create table 语句的示例，可用其创建自己的练习用表。如果没有 create table 权限，则咨询系统管理员或您使用的数据库的所有者。

创建一个表通常意味着创建与其相关的索引、缺省值和规则。还经常涉及自定义数据类型、触发器和视图。

您可以创建表，在创建索引、缺省值、规则、触发器或视图之前，输入一些数据并使用该表一段时间。这允许您查看最常用的事务，最常输入的数据。

不过，同时设计表及其所有组成部分通常是效率最高的方式。以下概要介绍了您要遵循的步骤。您会发现在实际创建表及其相应对象之前，在纸上草绘计划是最简单的方式。

首先，计划表的设计：

1　决定在表中需要哪些列以及各列的数据类型、长度、精度与标度。

2　在定义该表用于何处*之前*，应创建所有新用户定义的数据类型。

3　决定哪一列应成为 IDENTITY 列 （如果有）。

4　决定哪些列能接受空值，哪些列不能。

5　决定需要在表中的列上增加何种完整性约束或列缺省值 （如果有）。这也包括决定何时使用列约束与缺省值代替缺省值、规则、索引与触发器来保证数据完整性。

6　决定是否需要缺省值与规则，如果需要，何处需要何种类型的缺省值与规则。考虑列的 NULL 与 NOT NULL 状态与缺省值及规则之间的关系。

7　决定在何处需要何种索引。对索引的论述详见第 13 章 “创建表的索引”。

现在，创建表及其相关对象：

1　使用 create table 和 create index 创建表及其索引。

2　使用 create default 和 create rule 来创建缺省值和规则。对这些命令的论述详见第 14 章 “为数据定义缺省值和规则”。

3　使用 sp_bindefault 和 sp_bindrule 来绑定任何缺省值和规则。如果在 create table 语句中使用了用户定义的数据类型，而该类型已被捆绑了缺省值或规则，则它们将自动生效。对这些系统过程的论述详见第 17 章 “使用存储过程”。

4　使用 create trigger 创建触发器。对触发器的论述详见第 20 章 “触发器：强制实施参照完整性”。

5　使用 create view 创建视图。对视图的论述详见第 12 章 “视图：限制访问数据”。

制作设计草图

称为 friends_etc 的表用于本章和后续章节中，例示创建索引、缺省值、规则、触发器等的方法。其中包含您朋友的名字、地址、电话号码和个人信息。表中没有定义任何列缺省值或完整性约束。

如果另一用户已创建了 friends_etc 表，则您在计划按照这些示例执行并创建配合 friends_etc 一起使用的对象之前向系统管理员或数据库所有者核实情况。 friends_etc 的所有者必须删除其索引、缺省值、规则和触发器，以免在创建这些对象时发生冲突。

表 7-4 显示 friends_etc 表的建议结构以及用于各列的索引、缺省值和规则。

表 7-4：示例表设计

列	数据类型	是否允许 Null?	索引	缺省值	规则
pname	nm	NOT NULL	nmind （组合）		
sname	nm	NOT NULL	nmind （组合）		
address	varchar(30)	NULL			
city	varchar(30)	NOT NULL		citydflt	
state	char(2)	NOT NULL		statedflt	
zip	char(5)	NULL	zipind	zipdflt	ziprule
phone	p#	NULL			phonerule
age	tinyint	NULL			agerule
bday	datetime	NOT NULL		bdflt	
gender	bit	NOT NULL		gndrdflt	
debt	money	NOT NULL		gndrdflt	
notes	varchar(255)	NULL			

创建用户定义的数据类型

前两个列用于个人的名 （第一个）和姓。它们定义为 nm 数据类型。在创建表之前，请创建数据类型。对于 phone 列的 p# 数据类型同样如此：

```
execute sp_addtype nm, "varchar(30)"
execute sp_addtype p#, "char(10)"
```

nm 数据类型允许带有最大长度为 30 字节的可变长度字符条目。p# 数据类型允许带有固定长度为 10 字节的 char 数据类型。

选择能接受空值的列

除被指派了用户定义的数据类型的列外，每一列都有显式的 NULL 或 NOT NULL 条目。不需要在表定义中指定 NOT NULL，因为缺省如此。为了可读，此表被设计成显式地标明 NOT NULL。

缺省 NOT NULL 意味着列要求有条目，如此表中的两个名称列。没有名称，其它的数据就没有意义。另外， gender 列必须为 NOT NULL，因为不能对 bit 列使用 NULL 值。

如果将某个列指定为 NULL 并绑定了缺省值，则在输入中没有指定其它值时，输入该缺省值而不是 NULL。如果一列被指定为 NULL，而捆绑到该列的规则没有指定 NULL，当没有值输入时，列的定义将替换规则。列可以同时拥有缺省值和规则。缺省值和规则之间关系的论述详见第 14 章 “为数据定义缺省值和规则”。

定义表

请编写以下 create table 语句：

```
create table friends_etc
(pname        nm             not null,
sname         nm             not null,
address       varchar(30)    null,
city          varchar(30)    not null,
state         char(2)        not null,
postalcode    char(5)        null,
phone         p#             null,
age           tinyint        null,
bday          datetime       not null,
gender        bit            not null,
debt          money          not null,
notes         varchar(255)   null)
```

现在已为个人名和姓、地址、城市、州 / 省、邮政编码、电话号码、年龄、生日、性别、债务信息和注释定义了列。以后，会为此表创建规则、缺省值、索引、触发器和视图。

通过查询结果来创建新表：*select into*

可以使用 select into 命令，根据 select 语句的选择列表中指定的列以及 where 子句中指定的行来创建新表。 into 子句对创建测试表 （作为现有表副本的新表）以及用一个大表制作出几个较小表很有用。

select 和 select into 子句以及 delete 和 update 子句可启用 TOP 功能。TOP 选项是一个无符号的整数，可通过它来限制在目标表中插入的行数。它实现了与其它平台之间的兼容性。有关语法和用法信息，请参见 《参考手册：命令》。

仅当 select into/bulkcopy/pllsort 数据库选项设置为 on 时，才可对永久表使用 select into。系统管理员可以使用 sp_dboption 打开此选项。请使用 sp_helpdb 来查看是否已打开此选项。

将 select into/bulkcopy/pllsort 数据库选项设置为 on 时， sp_helpdb 及其结果如下所示：下例使用 8K 的页大小。

```
    sp_helpdb pubs2

name      db_size    owner     dbid created        status
--------- ---------- --------- ---- -------------- --------------
pubs2     20.0 MB     sa           4 Apr 25, 2005   select
     into/bulkcopy/pllsort, trunc log on chkpt, mixed log and data

device_fragments    size           usage         created      free kbytes
------------------- -------------- ------------- ----------   ------------
master              10.0MB         data and log  Apr 13 2005          1792
pubs_2_dev         10.0MB         data and log  Apr 13 2005         9888

device                  segment
----------------------- -----------------------
master                  default
master                  logsegment
master                  system
pubs_2_dev              default
pubs_2_dev              logsegment
pubs_2_dev              system
pubs_2_dev              seg1
pubs_2_dev              seg2
```

sp_helpdb output 指示是将该选项设置为 on 还是 off。只有系统管理员或数据库所有者才能设置这些数据库选项。

如果 select into/bulkcopy/pllsort 数据库选项为 on，则可使用 select into 子句来建立新的永久表，而不必使用 create table 语句。可 select into 临时表，即使 select into/bulkcopy/pllsort 选项不是 on 也是如此。

注释　因为 select into 是最少记录操作，所以在 select into 之后使用 dump database 来备份数据库。不能遵循最低限度记录操作转储事务。

不象显示部分表的视图，用 select into 创建的表是分离的、独立的实体。有关详细信息，请参见第 12 章 “视图：限制访问数据”。

新表基于在选择列表中指定的列、在 from 子句中命名的表以及在 where 子句中指定的行。新表的名称在数据库中必须是唯一的，并且必须符合标识符规则。

带有 into 子句的 select 语句允许基于现有定义和数据定义一个表并放入数据，而不必经历通常的数据定义进程。

以下示例显示了一个 select into 语句及其结果。使用四列表 publishers 中的两个列，创建了称为 newtable 的表。因为此语句不包括 where 子句，所以将 publishers 的所有行 （但仅指定的两个列）中的数据都复制到 newtable 中。

```
select pub_id, pub_name
into newtable
from publishers

(3 rows affected)
```

“3 rows affected”指的是插入到 newtable 中的三行。 newtable 如下所示：

```
select *
from newtable

pub_id  pub_name
------  ------------------------------------
0736    New Age Books
0877    Binnet & Hardley
1389    Algodata Infosystems
(3 rows affected)
```

新表包含 select 语句的结果。成为数据库的一部分，就像其父表一样。

通过在 where 子句中放置一个假条件，可创建一个不带数据的框架 (skeleton) 表。例如：

```
select *
into newtable2
from publishers
where 1=2
```

```
(0 rows affected)

select *
from newtable2

pub_id    pub_name          city      state
------    --------------    --------  -----

(0 rows affected)
```

由于 1 不可能等于 2，因此不会在新表中插入行。

也可以使用 select into 和集合函数来创建包含摘要数据的表：

```
select type, "Total_amount" = sum(advance)
into #whatspent
from titles
group by type

(6 rows affected)

select * from #whatspent

type         Total_amount
 ------------ ------------------------
UNDECIDED                        NULL
business                    25,125.00
mod_cook                    15,000.00
popular_comp                15,000.00
psychology                  21,275.00
trad_cook                   19,000.00

(6 rows affected)
```

始终为 select into 结果表中的任意列提供名称，该结果表是从集合函数或任何其它表达式得出的。以下为示例：

- 算术集合，例如， *amount* * 2
- 并置，例如， lname + fname
- 函数，例如 lower(lname)

下面是使用并置的一个示例：

```
select au_id,
    "Full_Name" = au_fname + ' ' + au_lname
into #g_authortemp
from authors
where au_lname like "G%"

(3 rows affected)
```

```
select * from #g_authortemp

 au_id       Full_Name
 ----------- -------------------------
 213-46-8915 Marjorie Green
 472-27-2349 Burt Gringlesby
 527-72-3246 Morningstar Greene

(3 rows affected)
```

因为函数允许空值，所以从 convert() 或 isnull() 之外的函数得出的表中的任意列也允许空值。

检查错误

select into 是个两步骤的操作。第一步创建新表，第二步将指定行插入新表。

由于未记录 select into 操作，因此无法在用户定义的事务中发出这些操作，也无法回退这些操作。

如果 select into 语句在创建新表后失败，Adaptive Server 将*不会*自动删除此表或释放此表的第一个数据页。这意味着，发生错误之前在第一页插入的所有行都将保留在该页上。检查 select into 语句后全局变量 *@@error* 的值，以确保没有错误发生。

如果 select into 操作中发生错误，请使用 drop table 来删除新表，然后重新发出 select into 语句。

对 IDENTITY 列使用 *select into*

本节描述对包含 IDENTITY 列的表使用 select into 命令的特殊规则。

选择一个 IDENTITY 添加到新表

若要选择现有的 IDENTITY 列并添加到新表中，请在 select 语句的 *column_list* 中包括该列名 （或 syb_identity 关键字）：

```
select column_list
    into table_name
    from table_name
```

以下示例基于 stores_cal 表中的列来创建一个新表 stores_cal_pay30：

```
select record_id, stor_id, stor_name
into stores_cal_pay30
from stores_cal
where payterms = "Net 30"
```

新列继承 IDENTITY 属性，除非下列任一条件为真：

- IDENTITY 列被多次选择。
- IDENTITY 列被选择作为表达式的一部分。
- select 语句包含 group by 子句、集合函数、 union 运算符或连接。

多次选择 IDENTITY 列

一张表不能有多个 IDENTITY 列。如果多次选择某一 IDENTITY 列，它将在新表中定义为 NOT NULL。该 IDENTITY 列将不继承 IDENTITY 属性。

在以下示例中，在 stores_cal_pay60 中将 record_id 列 （按名称选择一次，又按 syb_identity 关键字选择一次）定义为 NOT NULL：

```
select syb_identity, record_id, stor_id, stor_name
into stores_cal_pay60
from stores_cal
where payterms = "Net 60"
```

使用 *select into* 添加新 IDENTITY 列

若要在 select into 语句中定义新 IDENTITY 列，请在 into 子句之前添加列定义。该定义包括列的精度，但不包括其标度：

```
select column_list
      identity_column_name = identity(precision)
   into table_name
   from table_name
```

以下示例通过 discounts 表创建一个新表 new_discounts，并添加一个新 IDENTITY 列 id_col：

```
select *, id_col=identity(5)
into new_discounts
from discounts
```

如果 *column_list* 包括一个现有 IDENTITY 列，并且添加了新 IDENTITY 列的描述， select into 语句将失败。

定义一个必须计算其值的列

IDENTITY 列值由 Adaptive Server 生成。基于 IDENTITY 列生成，且其值必须经计算而不是生成得来的新列，不能继承 IDENTITY 属性。

如果表的 select 语句包括一个 DENTITY 列作为表达式的一部分，则必须计算生成的列值。如果表达式中的任何列都允许 NULL 值，则新列就被创建为 NULL。否则，为 NOT NULL。

在以下示例中，将 new_id 列被创建为 NOT NULL，该列是通过将 record_id 的值增加 1000 计算出来的：

```
select new_id = record_id + 1000, stor_name
into new_stores
from stores_cal
```

如果 select 语句包含 group by 子句或集合函数，则还将计算列值。如果 IDENTITY 列是集合函数的参数，那么结果列被创建为 NULL。否则，为 NOT NULL。

IDENTITY 列通过联合或连接选择到表中

IDENTITY 列的值唯一地标识表的每一行。然而，如果表的 select 语句包含联合或连接，则单独的行可在结果集中多次出现。通过联合或连接选入表中的 IDENTITY 列不保留 IDENTITY 属性。如果表中包含 IDENTITY 列和 NULL 列的联合，新列将定义为 NULL。否则，为 NOT NULL。

有关详细信息，请参见第 225 页的 “使用 IDENTITY 列”和第 322 页的 “更新 IDENTITY 列”。请参见 《参考手册：命令》。

改变现有表

使用 alter table 命令更改现有表的结构。利用它您可以：

- 添加列和约束
- 更改列缺省值
- 添加 null 列或 non-null 列
- 删除列和约束
- 更改锁定方案
- 分区或未分区表

- 转换列数据类型
- 转换现有列的 null 缺省值
- 增加或减小列长度

也可以更改表的分区属性。有关语法和用法信息，请参见第 10 章 “对表和索引进行分区”。请参见 《参考手册：命令》。

例如，缺省情况下， authors 表的 au_lname 列使用 varchar(50) 数据类型。若要将 au_lname 变更为使用 varchar(60)，请输入：

```
alter table authors
modify au_lname varchar(60)
```

注释 不能将变量用作属于 alter table 语句一部分的缺省值的参数。

删除、修改和添加非空列可能执行数据复制，这意味着需要空间和锁定方案。请参见第 270 页的 “数据复制”。

修改的表的页链将继承表的当前配置选项 （例如，如果 fillfactor 设置为 50%，则新页将拥有此相同的 fillfactor）。

注释 Adaptive Server 对 alter table 操作进行 （页分配的）部分日志记录。但是，由于 alter table 作为事务执行，因此运行 alter table 后无法转储事务日志；必须转储数据库以确保其可恢复。如果在 alter table 操作期间服务器遇到任何问题， Adaptive Server 将回退该事务。

alter table 在修改表模式时将获取排它表锁。命令完成后，将立即释放此锁。

alter table 不引发任何触发器。

使用 *select* * 的对象不列出对表的更改

如果数据库拥有的任何对象 （存储过程、触发器等）对删除列的表执行 select *，将出现一条错误消息，列出缺少的列。即使使用 with recompile 选项创建对象，上述情况同样会发生。例如，如果从 authors 表删除了 postalcode 列，所有对此表执行 select * 的存储过程都将发出以下错误消息：

```
Msg 207, Level 16, State 4:
Procedure 'columns', Line 2:
Invalid column name 'postalcode'.
```

```
(return status = -6)
```

如果添加新列并且随后运行包含 select * 的对象，此消息将不出现；在这种情况下，新列不出现在输出中。

必须删除并重新创建引用删除列的任何对象。

对远程表使用 *alter table*

可使用 alter table 修改使用 “组件集成服务 (CIS)” 的远程表。修改远程表之前，输入下列命令以确保 CIS 运行：

```
sp_configure "enable cis"
```

如果启用了 CIS，则此命令的输出为 “1”。缺省情况下，安装 Adaptive Server 时将启用 CIS。

请参见 《系统管理指南：卷 1》中的第 5 章 “设置配置参数”。有关使用 CIS 的信息，请参见 《组件集成服务用户指南》。

添加列

以下命令在 authors 表中添加名为 author_type 的非空列 （它包括常量 “primary_author” 作为缺省值）和名为 au_publisher 的空列：

```
alter table authors
add author_type varchar(20)
default "primary_author" not null,
au_publisher varchar(40) null
```

添加列将附加列 ID

alter table 在表中添加一列，其列 ID 比当前最大列 ID 大 1。例如，表 7-5 列出 salesdetail 表的缺省列 ID：

表 7-5：salesdetail 表的列 ID

列名	stor_id	ord_num	title_id	qty	discount
Col ID	1	2	3	4	5

以下命令将 store_name 列附加到 salesdetail 表的末端，其列 ID 为 6：

```
alter table salesdetail
add store_name varchar(40)
default
```

```
"unknown" not null
```

如果添加其它列，其列 ID 将为 7。

注释 因为添加和删除列时表的列 ID 会更改，所以应用程序不应依赖于列 ID。

添加 NOT NULL 列

可向表添加 NOT NULL 列。这意味着常量表达式和非空值将在添加列时被放置到该列中。这也确保了对所有现有行在创建表时用指定的常量表达式填充新列。

如果用户向 NOT NULL 列输入值失败，Adaptive Server 将发出错误消息。

以下命令将列 owner 添加到 stores 表中，并且缺省值为 “unknown”：

```
alter table stores
add owner_lname varchar(20)
default "unknown" not null
```

添加 NULL 列时，缺省值可以是常量表达式，但仅当添加 NOT NULL 列时 （如上例所示），缺省值才可以是常量值。

添加约束

要向现有列添加约束，可使用：

alter table *table_name*
add constraint *constraint_name* {*constraint_clause*}

其中：

- *table_name* 是要添加约束的表名称。
- *constraint_name* 是要添加的约束。
- *constraint_clause* 是约束强制的规则。

例如，若要在 titles 表中添加约束以禁止预付款超过 10,000，请使用以下命令：

```
alter table titles
add constraint advance_chk
check (advance < 10000)
```

如果用户试图在 titles 表中插入大于 10,000 的值，Adaptive Server 将生成类似于以下内容的错误消息：

```
Msg 548, Level 16, State 1:
Line 1:Check constraint violation occurred,
dbname = 'pubs2',table name= 'titles',
constraint name = 'advance_chk'.
Command has been aborted.
```

添加约束不会影响现有数据。同样，如果添加带有缺省值的新列并对该列指定约束，将不会根据约束对缺省值进行检验。

有关删除约束的信息，请参见第 264 页的 “删除约束”。

删除列

使用以下语法从现有表删除列：

alter table *table_name*
　　drop *column_name* [, *column_name*]

其中：

- *table_name* 是包含要删除的列的表。
- *column_name* 是要删除的列。

可使用单个 alter table 语句删除任何数量的列。但是，不能删除表中剩余的最后一列 （例如，如果从含有五个列的表中删除了四个列，则不能再删除剩余的一个列）。

例如，若要从 titles 表删除 advance 和 contract 列，请使用以下命令：

```
alter table titles
drop advance, contract
```

当 alter table 删除列时，它将重建表的所有索引。

删除列将对列 ID 进行重新编号

从表中删除列时，alter table 将对列 ID 重新进行编号。如果列 ID 大于删除的列编号，这些列将向上移动一个列 ID 以填充删除的列留下的间隙。例如， titleauthor 表包含下列列名和列 ID：

表 7-6：titleauthor 列 ID

列名	au_id	title_id	au_ord	royaltyper
列 ID	1	2	3	4

如果要从表中删除 au_ord 列，请使用以下命令：

```
alter table titleauthor drop au_ord
```

titleauthor 现在具有以下列名和列 ID：

表 7-7：删除 au_ord 后的列 ID

列名	au_id	title_id	royaltyper
列 ID	1	2	3

royaltyper 的列 ID 现在为 3。删除 au_ord 后， title_id 和 royaltyper 上的非聚簇索引也进行了重排。同样，不同系统目录中的列 ID 的所有实例均进行了重新编号。

通常，您不会注意到列 ID 的重新编号。

注释 因为添加和删除列时将对表的列 ID 进行重新编号，所以应用程序不应依赖于列 ID。如果存储过程或应用程序依赖于列 ID，应对它们重新进行编写，以使其能够访问正确的列 ID。

删除约束

要删除约束，可使用：

```
alter table table_name
    drop constraint constraint_name
```

table_name 和 *constraint_name* 与上面 “添加约束”中所述内容具有相同的值。

例如，若要删除前面创建的约束，请使用以下命令：

```
alter table titles
drop constaint advance_chk
```

有关表约束的详细信息，请参见第 292 页的 “使用 sp_helpconstraint 查找表的约束信息”。

修改列

要修改现有列，可使用：

```
alter table table_name
    modify column_name datatype [null | not null]
    [null | not null]
    [, column_name...]
```

其中：

- *table_name* 是包含要修改列的表。
- *column_name* 是要修改的列。
- *data_type* 是要将列修改成的数据类型。

可使用 alter table 语句修改任何数量的列。

例如，以下命令将 titles 表中 type 列的数据类型由 char(12) 更改为 varchar(20)，并使其可为空：

```
alter table titles
modify type varchar(20) null
```

警告！ 拥有的对象 （存储过程、触发器等）可能取决于具有特定数据类型的列。修改列之前，应确保所有引用该列的对象能够在修改后成功运行。使用 sp_depends 来确定表的相关对象。

可以进行转换的数据类型

只能转换可隐式或显式转换为新数据类型的数据类型，或者转换在 Transact-SQL 中包含显式转换函数的数据类型。有关受支持的数据类型转换的列表，请参见 《参考手册：构件块》中的第 1 章 “系统数据类型和用户定义的数据类型”。如果试图进行非法数据类型修改，Adaptive Server 将发出错误消息并中止该操作。

注释　不能将现有列数据类型转换为 timestamp 数据类型，也不能将使用 timestamp 数据类型的列修改为任何其它数据类型。

如果多次发出相同的 alter table...modify 命令，Adaptive Server 将发出类似于以下内容的消息：

```
Warning: ALTER TABLE operation did not affect column 'au_lname'.
Msg 13905, Level 16, State 1:
Server 'SYBASE1', Line 1:
Warning: no columns to drop, add or modify. ALTER TABLE 'authors' was aborted.
```

修改表可能会妨碍以前转储的成功批量复制

修改列的长度或数据类型可能会妨碍成功使用批量复制在表的旧转储中进行复制。旧表模式可能与新表模式不兼容。在对列的长度或数据类型进行修改之前，应进行检验以确保它不会妨碍在表的以前转储中进行复制。

减少列长度可能会截断数据

如果减少列的长度，应确保减少的列长度不会导致截断数据。例如，可使用 alter table 将 titles 表中 title 列的长度从 varchar(80) 减少到 varchar(2)，但数据就没有意义了：

```
select title from titles

title
-----
Bu
Co
Co
Em
Fi
Is
Li
Ne
On
Pr
Se
Si
St
Su
Th
Th
Th
Yo
```

Adaptive Server 仅当 set string_rtruncation 选项打开时才发出有关截断列数据的错误消息。如果需要截断字符数据，可设置适当的字符串截断选项并修改列以减少其长度。

修改 datetime 列

如果将列从 char 数据类型修改为 datetime、 smalldatetime 或 date 数据类型，则无法指定输出中月、日和年的出现顺序。也无法指定输出中使用的语言。相反，两项设置均被指定为缺省值。但可使用 set dateformat 或 set language 变更输出以与存储在列中的信息设置相匹配。同样，Adaptive Server 也不支持将列从 smalldatetime 修改为 char 数据类型。请参见 《参考手册：命令》。

修改列的 NULL 缺省值

如果仅更改列的 NULL 缺省值，则不需要指定列的数据类型。例如，以下命令将 authors 表中 address 列从 NULL 修改为 NOT NULL：

```
alter table authors
modify address not null
```

如果修改了某列并指定数据类型为 NOT NULL，只要所有行均没有 NULL 值该操作即成功。但如果其中任何行有 NULL 值，操作则失败并回退所有不完整的事务。例如，因为 titles 表中 *The Psychology of Computer Cooking* 包含 NULL 值，所以下列语句失败：

```
alter table titles
modify advance numeric(15,5) not null

Attempt to insert NULL value into column 'advance',
table 'pubs2.dbo.titles';
column does not allow nulls. Update fails.
Command has been aborted.
```

若要成功运行此命令，请对表进行更新，将修改列的所有 NULL 值更改为 NOT NULL，然后重新发出此命令。

修改具有精度或标度的列

在修改列标度之前检查数据长度。

如果 alter table 命令导致列值损失精度 （假定从 numeric(10,5) 变为 numeric(5,5)）， Adaptive Server 将中止该语句。如果此语句是批处理的一部分，此选项打开时批处理也将中止。

如果 alter table 命令导致列值损失标度 （假定从 numeric(10, 5) 变为 numeric(10,3)），行将被截断且不发出任何警告。无论 arithabort numeric_truncation 开关与否，上述情况都会发生。

如果 arithignore arith_overflow 被打开且 alter table 导致数值溢出，Adaptive Server 将发出警告。但如果 arithignore arith_overflow 是关闭的，当 alter table 导致数值溢出时， Adaptive Server 不会发出警告。缺省情况下，安装 Adaptive Server 时， arithignore arith_overflow 是关闭的。

注释 在生产环境中发出可能截断列长度的命令之前，确保已查看数据截断规则并完全了解其含义。应首先在一组测试列上执行这些命令。

修改 *text*、 *unitext* 和 *image* 列

text 列可以转换为：

- [n]char
- [n]varchar
- unichar
- univarchar

unitext 列可以转换为：

- [n]char
- [n]varchar
- unichar
- univarchar
- binary
- varbinary

image 列可以转换为：

- varbinary
- binary

不能将 char、 varchar、 unichar 和 univarchar 数据类型列修改为 text 或 unitext 列。如果将 text 或 unitext 转换为 char、 varchar、 unichar 或 univarchar，则列的最大长度由页大小控制。如果未指定列长度， alter table 将使用缺省长度一字节。如果要修改多字节字符 text、 unitext 或 image 列，并且没有指定足够大的列长度以包含该数据，Adaptive Server 将根据列的长度截断该数据。

添加、删除和修改 IDENTITY 列

本节说明使用 alter table 添加、删除和修改 IDENTITY 列。有关 IDENTITY 列的概述，请参见第 225 页的 “使用 IDENTITY 列”。

添加 IDENTITY 列

只能添加缺省值为 NOT NULL 的 IDENTITY 列。不能指定新 IDENTITY 列的缺省子句。

要将 IDENTITY 列添加到表，可在 alter table 语句中指定 identity 关键字：

```
alter table table_name add column_name
    numeric(precision ,0) identity not null
```

以下示例将 IDENTITY 列 record_id 添加到 stores 表中：

```
alter table stores
    add record_id numeric(5,0) identity not null
```

将 IDENTITY 列添加到表时， Adaptive Server 从 1 开始为每行指派一个唯一的顺序值。如果表包含大量行，此进程可能会花费大量时间。如果行数超过列所允许的最大值 （在此情况下为 10^5 - 1 或 99,999）， alter table 语句失败。

用户定义的数据类型

可以使用用户定义的数据类型来创建 IDENTITY 列。用户定义的数据类型必须为 numeric 类型，且标度为零。

删除 IDENTITY 列

可以象删除其它任何列一样删除 IDENTITY 列。例如，要删除在上一节中创建的 IDENTITY 列：

```
alter table stores
drop record_id
```

但是，以下限制适用于删除标识列：

- 如果在数据库中打开了 sp_dboption “identity in nonunique index”，则必须先删除所有索引，然后删除 IDENTITY 列，再重新创建这些索引。

  如果 IDENTITY 列是隐藏的，则必须首先使用 syb_identity 关键字标识该列。请参见第 227 页的 “使用 syb_identity 引用 IDENTITY 列”。

- 若要从已打开 set identity_insert 的表中删除 IDENTITY 列，请发出 sp_helpdb 以确定是否打开了 set identity_insert。

必须发出以下命令来关闭 set identity_insert 选项：

```
set identity_insert table_name off
```

删除 IDENTITY 列，然后添加新的 IDENTITY 列，再输入以下命令以打开 set identity_insert 选项：

```
set identity_insert table_name on
```

修改 IDENTITY 列

可以修改 IDENTITY 列的大小以增加其范围。如果当前范围太小或由于服务器关闭而使范围用尽时，则有必要进行上述操作。

例如，可通过输入以下命令来增加 record_id 的范围：

```
alter table stores
modify record_id numeric(9,0)
```

通过为目标数据类型指定较小的精度可减小范围。如果表中的 IDENTITY 值对于目标 IDENTITY 列范围来说太大，将引发算术转换，alter table 将终止该语句。

不能使用需要数据复制的 alter table 命令将非空 IDENTITY 列添加到分区表。对于分区表，数据复制并行完成，无法保证唯一的 IDENTITY 值。

数据复制

Adaptive Server 仅在更改表模式前必须将数据从表中临时复制出来时，才会执行数据复制。如果表具有索引， Adaptive Server 将在数据复制完成时重建索引。

注释 在 alter table 执行数据复制时，包含表的数据库必须已打开 select into/bulkcopy/pllsort。请参见 《参考手册：命令》。

Adaptive Server 在下列情况下执行数据复制：

- 删除一列。
- 修改某一列的任何一个属性：
  - 数据类型 （不包括增加 varchar、 varbinary 或 NULL char 或 NULL binary 列的长度）。
  - 从 NULL 到 NOT NULL，或相反。

  - 减小长度。减小列的长度时，也许您事先并不知道长度减小的列是否能够容下所有数据。例如，如果将 au_lname 减小到 varchar(30)，它可能包含一个需要 varchar (35) 的名称。减小列的数据长度时， Adaptive Server 会首先执行数据复制以确保列长度的更改成功。
- 减小编号列的长度 （例如，从 tinyint 到 int）。当此列中某一行的值为 NOT NULL 时， Adaptive Server 执行数据复制。
- 添加 NOT NULL 列。

在下列情况下， alter table 不执行数据复制：

- 更改 varchar 或 varbinary 列的长度。
- 更改用户定义的数据类型 ID，但物理数据类型不变。例如，如果站点有两种数据类型， mychar1 和 mychar2，它们有不同的用户定义数据类型，但物理数据类型相同，则将 mychar1 改为 mychar2 时，不会发生数据复制。
- 将可变长度列的 NULL 缺省值从 NOT NULL 更改为 NULL。

要确定 alter table 是否执行数据复制：

1　将 showplan 设置为打开，报告 Adaptive Server 是否将执行数据复制。

2　将 noexec 设置为打开，确保不执行任何操作。

3　如果不需要任何数据复制，则执行 alter table 命令；仅执行目录更新以反映 alter table 命令所做的更改。

修改 *exp_row_size*

在执行数据复制时，也可更改 exp_row_size，它用于指定每行的空间。仅在被修改的表模式包含可变长度列时，才能更改 exp_row_size，并且只能在被修改表的 sysindexes 中的 maxlen 和 minlen 值指定的范围内进行修改。

如果列中有多个固定长度的列，则只能将 exp_row_size 改为 0 或 1。如果从表中删除所有可变长度列，则必须将 exp_row_size 指定为 0 或 1。此外，如果没有为 alter table 命令提供 exp_row_size，则使用旧的 exp_row_size。如果表仅包含固定长度的列且旧的 exp_row_size 与修改模式不兼容， Adaptive Server 将发出错误消息。

不能将 exp_row_size 子句与其它任何 alter table 子句一同使用 （例如，定义约束、更改锁定方案等）。也可使用 sp_chgattribute 来更改 exp_row_size。请参见 《参考手册：命令》。

修改锁定方案和表模式

如果 alter table 执行数据复制，则也可同时使用一个更改表的锁定方案的命令。例如，若要修改 authors 表的 au_lname 列并将此表的锁定方案从所有页锁定更改为数据行锁定，请使用以下命令：

```
alter table authors
modify au_lname varchar(10)
lock datarows
```

但不能使用 alter table 更改具有聚簇索引的表的表模式和锁定方案。如果表具有聚簇索引，则可以：

1 删除索引。

2 修改表模式并在同一语句中更改锁定方案 （如果表模式的更改也包括数据复制）。

3 重建聚簇索引。

也可以发出 alter table 命令来更改锁定方案，然后发出另一个 alter table 命令来更改表模式。

变更具有用户定义的数据类型的列

可使用 alter table 添加、删除或修改使用用户定义数据类型的列。

添加具有用户定义数据类型的列

添加具有用户定义数据类型的列的语法，与添加具有系统定义数据类型的列相同。例如，要使用 usertype 数据类型将列添加到 pubs2 的 authors 表：

```
alter table titles
add newcolumn usertype not null
```

用户指定的 NULL 或 NOT NULL 缺省值的优先级高于用户定义数据类型所指定的缺省值。也就是说，如果添加了一列并指定 NOT NULL 为缺省值，则新列的缺省值为 NOT NULL，即使用户定义数据类型指定为 NULL。如果没有指定 NULL 或 NOT NULL，则使用用户定义数据类型指定的缺省值。

添加 not null 列时，必须提供缺省子句，除非用户定义数据类型已绑定了缺省值。

如果用户定义数据类型指定了 IDENTITY 列属性 （精度和标度），则该列被作为 IDENTITY 列添加。

删除具有用户定义数据类型的列

删除具有用户定义数据类型的列的方式与删除具有系统定义数据类型的列相同。

修改具有用户定义数据类型的列

修改列使其包含用户定义数据类型的语法与修改列使其包含系统定义数据类型的语法相同。例如，要修改 authors 表的 au_lname，使其使用用户定义的 newtype 数据类型：

```
alter table authors
modify au_lname newtype(60) not null
```

如果没有指定缺省值为 NULL 或 NOT NULL，则列使用用户定义数据类型指定的缺省值。

修改表并不影响绑定到列的任何现行规则或缺省设置。但是，如果指定了新规则或缺省设置，则任何旧规则或绑定到用户定义数据类型的缺省设置都将被删除。如果以前没有任何旧规则或缺省设置绑定到列，则应用任何用户定义的规则和缺省设置。

不能将现有的列改为 IDENTITY 列。只能使用具有 IDENTITY 列属性（精度和标度）的用户定义数据类型修改现有的 IDENTITY 列。

alter table 生成的错误和警告

在运行 alter table 时出现的大多数错误都会告诉您模式结构阻止了所请求命令的执行 （例如，如果尝试删除属于索引一部分的列）。在重新发出命令前，必须解决指向与受影响列相关的模式对象的错误或警报。要报告错误情况：

1　将 showplan 设置为打开。

2　将 noexec 设置为打开。

3　执行 alter table 命令。

在更改命令以解决任何报告出的错误后，请将 showplan 和 noexec 设置为关闭，以便 Adaptive Server 能够执行实际操作。

实际运行命令时， alter table 会检测出并报告某些错误 （例如，要删除存在参照约束的某列）。只能在执行语句时，才能查明所有运行时数据相关的错误 （例如，数值溢出错误、字符截断错误等）。必须更改命令以适应可用的数据，或修改数据值以便使用语句指定的、必需的目标数据类型。若要查明这些错误，必须在运行命令时将 noexec 关闭。

alter table modify 生成的错误和警告

有些错误仅由 alter table modify 命令生成。虽然 alter table modify 用于将列转换为兼容的数据类型，但如果要转换的列具有某些限制，则 alter table 可能产生错误。

注释 确保在发出命令前了解修改数据类型的潜在影响。通常，只将 alter table modify 用于可转换数据类型之间的隐式转换。这样可确保 insert 和 update 语句处理期间必需的隐藏转换不会因为数据类型不兼容而失败。

例如，如果将 second_advance 列添加到数据类型为 int 的 titles 表中，并在 second_advance 上创建聚簇索引，则无法将此列修改为 char 数据类型。这将使 int （‘1’、‘2’、‘3’）。用已排序数据重建索引后，数据值应按排序顺序排列。但此示例中，数据类型已经从 int 更改为 char，且不再按 char 数据类型的排序顺序排列。因此， alter table 命令将在重建索引阶段失败。

如某些列属于聚簇索引的索引键列的一部分，则在为其选择新数据类型时，要格外谨慎。 alter table modify 必须指定一个在其数据复制阶段结束后，不会与已修改数据值的排序顺序发生冲突的目标数据类型。

如果在包含约束的列中将数据类型修改为不兼容的数据类型， alter table modify 也会发出警告消息。例如，如果尝试将数据类型 char 修改为数据类型 int，且该列包含一个约束，则 alter table modify 发出以下警告：

```
Warning: a rule or constraint is defined on column ‘new_col’ being modified.
Verify the validity of rules and constraints after this ALTER TABLE operation.
```

此警告说明约束需要的值与修改的列 new_col 数据类型之间可能存在数据类型不一致：

```
Warning: column ‘new_col’ is referenced by one or more rules or constraints.
Verify the validity of the rules/constraints after this ALTER TABLE operation.
```

如果尝试将数据类型为 char 的数据插入此表中，此操作将失败并显示以下消息：

```
Msg 257, Level 16, State 1: Line 1: Implicit conversion from datatype ‘CHAR’ to
‘INT’ is not allowed.  Use the CONVERT function to run this query.
```

检查约束的定义期望列的数据类型为 int，但要插入的数据的数据类型是 char。

此外，不能将数据作为类型 int 插入，因为列现在使用的数据类型为 char。 insert 返回以下消息：

```
Msg 257, Level 16, State 1: Line 1: Implicit conversion from datatype 'INT' to
'CHAR' is not allowed.  Use the CONVERT function to run this query.
```

modify 操作非常灵活，但必须谨慎使用。一般情况下，修改为可隐式转换数据类型的操作不会出错。修改为可显式转换数据类型的操作可能导致表模式中的不一致。修改列的数据类型前，请使用 sp_depends 确定所有列级依赖性。

if exists()...alter table 生成的脚本

如果包括如下结构的脚本中说明的表未包含指定的列，则该脚本可能会出错：

```
if exists (select 1 from syscolumns
    where id = object_id("some_table")
        and name = "some_column")
    begin
        alter table some_table drop some_column
    end
```

在此示例中， some_table 中*必须*有 some_column，批处理才能成功。

如果 some_table 中有 some_column，则第一次运行批处理时， alter table 将删除该列。在后续的执行中，批处理不会进行编译。

Adaptive Server 在预处理此批处理时引发这些错误，这些错误与常规 select 试图访问不存在的列时引发的错误类似。使用需要数据复制的子句修改表模式时，将引起这些错误。如果添加空列并使用上述结构，则 Adaptive Server 不会提示这些错误。

为避免在修改表模式时出现此类错误，请将 alter table 包括在 execute immediate 命令中：

```
if exists (select 1 from syscolumns
       where id = object_id("some_table")
         and name = "some_column")
begin
    exec ("alter table some_table drop
        some_column")
end
```

因为 execute immediate 语句只能在 if exists() 函数成功后运行，所以 Adaptive Server 在编译此脚本时不会引发任何错误。

还必须将 execute immediate 结构用于 alter table 的其它使用场合（例如，更改锁定方案），以及该命令不要求数据复制的任何其它情况。

重命名表和其它对象

若要重命名表和其它数据库对象 （列、约束、数据类型、视图、索引、规则、缺省值、过程和触发器），请使用 sp_rename。

必须拥有对象，才能重命名它。不能更改系统对象或系统数据类型的名称。数据库所有者可更改任何用户对象的名称。另外，要更改名称的对象必须在当前数据库中。

若要重命名数据库，请使用 sp_renamedb。请参见 《参考手册：过程》。

sp_rename 的语法为：

sp_rename *objname*, *newname*

例如，若要将 friends_etc 的名称更改为 infotable，请使用以下命令：

```
sp_rename friends_etc, infotable
```

若要重命名列，请使用：

sp_rename "*table.column*", *newcolumnname*

新列名中必须去掉表名前缀，否则新名将不被接受。

要更改索引的名称，请使用：

sp_rename "*table.index*", *newindexname*

不要将表名包括在新名称中。

若将用户数据类型 tid 的名称更改为 t_id，请使用：

```
exec sp_rename tid, "t_id"
```

重命名相关对象

重命名对象时，也必须更改任何相关过程、触发器或视图的文本以反映新对象名。原始对象名仍出现在查询结果中，直到更改了过程、触发器或视图的名称并对它们进行了编译。最安全的做法是在执行 sp_rename 时更改所有**相关**对象的定义。可以使用 sp_depends 来获取相关对象的列表。

可使用 defncopy 实用程序将过程、触发器、规则、缺省设置和视图的定义复制到操作系统文件中。编辑此文件以改正对象名，然后使用 defncopy 将这些定义复制回 Adaptive Server 中。请参见 《实用程序指南》。

删除表

可以使用 drop table 从数据库中删除表。

```
drop table [[database.]owner.] table_name
    [, [[database.]owner.] table_name]...
```

此命令从数据库中删除指定的表，连同表中的内容以及相关的所有索引与特权。原来绑定到该表的规则和缺省值将不再绑定，其它方面均无变化。

只有表的所有者才能删除它。然而，在用户或前端程序对表进行读写操作时，任何人都不能将删除它。不能对任何系统表使用 drop table 命令，无论是在 master 数据库还是在用户数据库中。

可以删除另一数据库中的表，只要您是表的所有者。

如果使用 delete 删除表中的所有行，或对它使用 truncate table 命令，则在使用 drop 删除表之前，该表将始终存在。

drop table 和 truncate table 权限不能移交给其它用户。

计算列

本节介绍了一项增强功能，可通过它来创建计算列、计算列索引和基于函数的索引，从而提供更简便的数据操纵和更快捷的数据访问。有关基于函数的索引的详细信息，请参见第 421 页的 “使用基于函数的索引编制索引”。

- 计算列是由表达式定义的，无论是从同一行中的常规列还是从函数、算术运算符和 XML 路径查询等等。

  表达式可以是确定性的，也可以是非确定性的。对于相同的一组输入，确定性表达式始终返回相同的结果。

- 可以创建实现计算列的索引，就像它们是常规列一样。

计算列与基于函数的索引类似，它们都允许创建表达式的索引。

计算列和基于函数的索引的区别

计算列和基于函数的索引在某些方面存在区别：

- 计算列为表达式和索引功能提供了简写形式；而基于函数的索引不提供任何简写形式。
- 基于函数的索引允许直接创建表达式索引；而要创建计算列索引，必须先创建计算列。

- 计算列可以是确定性的，也可以是非确定性的，而基于函数的索引必须是确定性的。“确定性”是指如果表达式中的输入值相同，返回值也必须是相同的。有关此属性的详细信息，请参见第 281 页的“确定性属性”。
- 可以创建计算列的聚簇索引，但不能创建基于函数的聚簇索引。

实现计算列与非实现计算列之间的区别

- 计算列可以是已实现的，也可以是未实现的。当插入或更新基列时，会对实现的列进行预求值并将其存储在表中。与实现列相关的值同时存储在数据行和索引行中。任何对实现列的后续访问都不需要重新求值，因为可以访问它的预求值结果。一旦实现了列，每次访问该列都将返回相同的值。
- 未实现的列有时称为虚拟列；在访问虚拟列时，这些列将变为实现列。如果列是虚拟列 （即未实现的列），每次访问该列时，必须求出它的结果值。这意味着，如果虚拟计算列表达式基于非确定性表达式或调用非确定性表达式，则每次访问它时，可能会返回不同的值。访问虚拟计算列时，还可能会出现运行时异常 （如域错误）。

使用计算列

计算列允许为表达式创建简写形式 （如 “Pay”表示 “Salary + Commission”）并使该列变为可索引的列，前提是其数据类型是可索引的。不能索引的数据类型包括：

- text
- unitext
- image
- Java 类
- bit

计算列用于提高应用程序开发和维护效率。通过将表达式逻辑集中放在表定义 (DDL) 中并为表达式指定有意义的别名，计算列可大大提高 DML 的简便性和可读性。仅通过修改计算列定义，即可更改表达式。

必须对以下列进行索引时，计算列特别有用：列的定义表达式是非确定性表达式或函数，或者调用非确定性表达式或函数。例如， getdate() 函数始终返回当前日期，因此它是非确定性函数。要使用 getdate() 对列进行索引，您可以生成一个实现计算列，然后对其进行索引：

```
create table rental
      (cust_id int, start_date as getdate()materialized, prod_id in))

create index ind_start_date on rental (start_date)
```

组合和分解数据类型

计算列的一项重要功能是，它们可用于组合和分解复杂数据类型。可以使用计算列通过简单元素来生成复杂数据类型（组合），或者从复杂数据类型中提取一个或多个元素（分解）。复杂数据类型通常是由各个元素或片段组成的。在定义表时，可以定义自动分解或组合这些复杂数据类型。例如，假定要在表中存储 XML “order”文档以及下面的一些关系型元素：*order_no*、*part_no* 和 *customer*。通过使用 create table，您可以定义计算列提取：

```
CREATE TABLE orders(xml_doc IMAGE,
order_no COMPUTE xml_extract("order_no", xml_doc)MATERIALIZED,
part_no COMPUTE xml_extract ("part_no", xml_doc)MATERIALIZED,
customer COMPUTE xml_extract("customer", xml_doc)MATERIALIZED)
```

每次将新的 XML 文档插入表中时，会自动将文档的关系型元素提取到计算列中。

或者，若要将每个行中的关系型数据表示为 XML 文档，请使用表定义中的计算列来指定关系型数据到 XML 文档的映射。例如，若要定义一个表，请使用以下命令：

```
CREATE TABLE orders
(order_no INT,part_no INT, quantity SMALLINT, customer VARCHAR(50))
```

然后，若要返回每个行中关系型数据的 XML 表示，请使用 alter table 来添加计算列：

```
ALTER TABLE orders
ADD order_xml COMPUTE order_xml(order_no, part_no, quantity, customer)
```

最后，使用 select 语句以 XML 格式返回每个行：

```
SELECT order_xml FROM orders
```

用户定义的排序

计算列支持复杂数据类型的 comparison、order by 和 group by 排序，如 XML、text、unitext、image 和 Java 类。可以使用计算列来提取复杂数据的关系型元素，这些元素可用于定义排序。

也可以使用计算列将数据转换为不同的格式，以自定义数据表示来进行数据检索。这称为用户定义的排序顺序。例如，以下查询按服务器的缺省字符集的顺序和排序顺序（通常为 ASCII 字母顺序）返回结果：

```
SELECT name, part_no, listPrice FROM parts_table ORDER BY name
```

可以使用计算列按不区分大小写的格式表示查询结果（例如基于特殊情况的首字母缩写词的顺序，与证券市场符号的顺序一样），也可以使用非缺省的系统排序顺序。要将数据转换为不同的格式，请使用内置函数 sortkey 或用户定义的排序顺序函数。

例如，可以使用用户定义的函数 Xform_to_myorder() 来添加名为 *name_in_myorder* 的计算列：

```
ALTER TABLE parts_table add name_in_myorder COMPUTE
     Xform_to_myorder(name)MATERIALIZED
```

下面的查询以自定义的格式返回结果：

```
SELECT name, part_no, listPrice FROM parts_table ORDER BY name_in_myorder
```

通过使用此方法，您可以实现转换的排序数据并创建其索引。

如果愿意，可以使用数据操纵语言 (DML) 来执行相同的操作：

```
SELECT name, part_no, listPrice FROM parts_table
       ORDER BY Xform_to_myorder(name)
```

不过，通过使用计算列方法，您可以实现转换的排序数据并创建其索引，这可提高查询的性能。

决策支持系统 (DSS)

典型的决策支持系统应用程序要求在数据分析过程中进行大量的数据操纵、关联以及归类。此类应用程序经常在查询中使用表达式和函数，通常需要用户定义的特殊排序。通过使用计算列和基于函数的索引，可以简化此类应用程序中所需的任务并提高性能。

计算列示例

计算列是由表达式定义的。可通过合并同一行中的常规列来生成表达式。表达式可以包含函数、算术运算符、 case 表达式、同一表中的其它列、全局变量、 Java 对象和路径表达式。例如：

```
CREATE TABLE parts_table
  (part_no Part.Part_No, name CHAR(30),
  descr TEXT, spec IMAGE, listPrice MONEY,
      quantity INT,
   name_order COMPUTE name_order(part_no)
   version_order COMPUTE part_no>>version,
   descr_index COMPUTE des_index(descr),
   spec_index COMPUTE xml_index(spec)
   total_cost COMPUTE quantity*listPrice
)
```

在本节中：

- part_no 是一个 Java object 列，它表示指定的部件号。
- desc 是一个 text 列，它包含指定部分的详细描述。
- spec 是一个 image 列，它存储分析的 XML 流对象。

- name_order 是一个计算列，它是由用户定义的函数 XML() 定义的。
- version_order 是一个计算列，它是由 Java 类定义的。
- descr_index 是一个计算列，它是由 des_index()（它生成 text 数据的索引键）定义的。
- spec_index 是一个计算列，它是由 xml_index()（它生成 XML 文档的索引键）定义的。
- total_cost 是一个计算列，它是由算术表达式定义的。

计算列的索引

只要可以为结果的数据类型编制索引，您就可以对计算列创建索引，就好像这些列是常规列一样。计算列索引和基于函数的索引提供了一种创建复杂数据类型（如 XML、text、unitext、image 和 Java 类）索引的方法。

例如，以下代码示例对计算列创建聚簇索引，就好像这些列是常规列一样：

```
CREATE CLUSTERED INDEX name_index on part_table(name_order)
CREATE INDEX adt_index on parts_table(version_order)
CREATE INDEX xml_index on parts_table(spec_index)
CREATE INDEX text_index on parts_table(descr_index)
```

创建或更新索引时，Adaptive Server 将对计算列进行求值，并使用结果来建立或更新索引。

确定性属性

本节说明了确定性属性的性质及其对计算列和基于函数的索引的影响。

什么是确定性属性？

所有表达式和函数要么是确定性的，要么是非确定性的：

- 如果使用一组相同的输入值对确定性表达式和函数进行求值，它们始终返回相同的结果。
- 每次对非确定性表达式或函数进行求值时，它们可能会返回不同的结果，甚至使用一组相同的输入值来调用它们时也是如此。

“确定性属性”是定义计算列或基于函数的索引键的表达式的属性，因而定义了计算列或基于函数的索引键本身。

示例

- 以下表达式是确定性的：

  ```
  c1 * c2
  ```

- getdate 函数是非确定性的，因为它始终返回当前日期。

什么会影响确定性属性？

确定性属性取决于表达式是否包含任何非确定性的元素，如各种系统函数、用户定义的函数以及全局变量。

函数是否为确定性的函数取决于函数编码：

- 如果函数调用非确定性的函数，它本身可能是非确定性的。
- 如果函数的返回值取决于输入值以外的因素，则函数可能是非确定性的。

确定性属性如何影响计算列？

Adaptive Server 中共有两种类型的计算列：

- 虚拟计算列
- 实现计算列

这两种类型的计算列的主要区别是，当查询引用虚拟计算列时，每次查询访问它时，都会对其进行求值。

当插入数据行或更新任何基列时，实现计算列的结果将存储在表中。在查询中引用实现计算列时，不会重新对其进行求值。将使用其预求值结果。

- 在将非实现计算列 （即虚拟计算列）用作索引键后，该计算列将变为实现计算列。
- 仅当更新实现计算列的基列之一时，才会重新对其进行求值。

示例

以下示例说明了非确定性计算列和索引键的用途和使用风险。本文中的所有示例仅用于说明目的。

示例 1

```
CREATE TABLE Renting
     (Cust_ID INT, Cust_Name VARCHAR(30),
     Formatted_Name COMPUTE format_name(Cust_Name),
     Property_ID INT,Property_Name COMPUTE
     get_pname(Property_ID), start_date COMPUTE
     today_date()MATERIALIZED, Rent_due COMPUTE
     rent_calculator(Property_ID, Cust_ID,
     Start_Date))
```

此示例中的 Renting 表存储各种房产的租赁信息。它包含以下字段：

表 7-8：Renting 表中的字段

字段	定义
Cust_ID	客户 ID
Cust_Name	客户名称
Formatted_Name	标准格式的客户名称
Property_ID	租赁的房产 ID
Property_Name	标准格式的房产信息
Start_Date	租赁开始日期
Rent_Due	今天到期的租金

Formatted_Name、 Property_Name、 Start_Date 和 Rent_Due 被定义为计算列。

- Formatted_Name — 将客户名称转换为标准格式的虚拟计算列。由于其输出仅取决于输入 Cust_Name，因此 Formatted_Name 是确定性的。
- Property_Name — 从另一个表 Property 中检索房产名称的虚拟计算列。 Property 被定义为：

  ```
  CREATE TABLE Property
       (Property_ID INT, Property_Name VARCHAR(30),
        Address VARCHAR(50), Rate INT)
  ```

  为基于输入 ID 来获取房产名称，函数 get_pname 调用一个 JDBC 查询：

  ```
  SELECT Property_Name from Property where
  Property_ID=input_ID
  ```

  计算列 Property_Name 看起来是确定性的，但它实际上是非确定性的，因为其返回值取决于表 Property 中存储的数据以及输入值 Property_ID。

- Start_Date — 用户定义的非确定性函数，它以 varchar(15) 返回当前日期。它被定义为实现函数。因此，每次插入新记录时都会重新对其进行求值，并且将该值存储在 Renting 表中。
- Rent_Due — 非确定性的虚拟计算列，它基于房产的租赁价格、客户的折扣状态以及租赁的天数来计算当前到期的租金。

确定性属性如何影响虚拟计算列

- Adaptive Server 确保可重复读取确定性的虚拟计算列，即使从定义上讲，每次引用虚拟计算列时，都会对其进行求值。例如：

  ```
  SELECT Cust_ID, Property_ID from Renting
       WHERE Formatted_Name ='RICHARD HUANG'
  ```

  如果表中的数据没有发生变化，该语句始终返回相同的结果。

- Adaptive Server 不保证可重复读取非确定性的虚拟计算列。例如：

  ```
  SELECT Cust_Name, Rent_Due from renting
      where Cust_Name= 'RICHARD HUANG'
  ```

  在此查询中，对于不同的天数， Rent_Due 列将返回不同的结果。该列具有连续时间属性，它的值是两次租赁付款间隔的时间长度的函数。

  非确定性属性在此处很有用，但必须谨慎使用。例如，如果误将 Start_Date 定义为虚拟计算列，并且输入了相同的查询，所有房产租金就会为零：Start_Date 的求值结果始终为当前日期，因此，Rental_Days 的数量始终为零。

  类似地，如果误将非确定性的计算列 Rent_Due 定义为预求值列（通过将其声明为实现列或将其用作索引键），房产租金就会为零。将只对其求值一次 （在插入记录时），并且租赁天数为零。每次引用该列时，将返回该值。

确定性属性如何影响实现计算列

Adaptive Server 可确保重复读取实现计算列，无论其确定性属性如何，这是因为在查询中引用它们时，不会重新对其进行求值。相反，Adaptive Server 使用预求值的结果。

确定性的实现计算列始终具有相同的值，但通常会重新对其进行求值。

非确定性的实现计算列必须遵循以下规则：

- 每次对相同的计算列进行求值时，可能会返回不同的结果，甚至使用一组相同的输入。

- 引用非确定性的预求值计算列时，将使用预求值的结果，这可能会与当前求值结果不同。换句话说，在非确定性的预求值计算列中使用过去的数据，而非当前数据。

在示例 1 中， Start_Date 是非确定性的实现计算列。取决于插入行的日期，其结果会发生变化。例如，如果租期从 “02/05/04” 开始，则会在列中插入 “02/05/04”，以后引用 Start_Date 时，将使用该值。如果决定以后 （例如， 06/05/04）引用该值，并且每次查询该列时都会对表达式进行求值，查询将继续返回 “02/05/04”，而不是像所希望的那样返回 “06/05/04”。

示例 2

假定您使用示例 1 中创建的表 Renting，并且创建虚拟计算列 Property_Name 的索引，从而将 Property_Name 转换为实现计算列。然后，插入一条新记录：

```
Property_ID=10
```

插入此新记录时，将从表 Property 中调用 getpname(10) 以执行该 JDBC 查询：

```
SELECT Property_Name from Property where Property_ID=10
```

该查询返回 “Rose Palace”，它存储在数据行中。除非有人发出以下查询更改了房产名称，否则所有一切都会正常运行：

```
UPDATE Property SET Property_Name ='Red Rose Palace'
     where Property_ID = 10
```

该查询返回 “Rose Palace”，因此， Adaptive Server 存储 “Rose Palace”。对表 Property 执行的此 update 命令将使 Renting 表中存储的 Property_Name 值无效，也必须将其更新为 “Red Rose Palace”。因为 Property_Name 是在表 Renting 中的 Property_ID 列上定义的，而不是在表 Property 中的 Property_Name 列上定义的，所以，不会自动对其进行更新。以后引用 Property_Name 时，可能会产生不正确的结果。

若要避免出现这种情况，请在表 Property 上创建一个触发器：

```
CREATE TRIGGER my_t ON Property FOR UPDATE AS
     IF UPDATE(Property_Name)
     BEGIN
     UPDATE Renting SET Renting.Property_ID=Property.Property_ID
          FROM Renting, Property
          WHERE Renting.Property_ID=Property.Property_ID
     END
```

在此触发器更新 Property 表中的 Property_Name 列时，它还会更新 Renting.Property_ID 列，这是 Property_Name 的基列。此自动更新会触发 Adaptive Server 重新对 Property_Name 进行求值，并更新数据行中存储的值。每次 Adaptive Server 更新 Property 表中的 Property_Name 列时，都会刷新 Renting 表中的 Property_Name 实现计算列，并且该列显示正确的值。

确定性属性如何影响基于函数的索引？

与计算列不同，基于函数的索引键必须是确定性的。从概念上讲，计算列仍然是一个列，在对其进行求值和存储后，它不要求重新进行求值。但是，每次函数或表达式在查询中出现时，必须重新对其进行求值。除非对于一组相同的输入，函数的求值结果始终相同，否则，不能使用预求值的数据，如索引数据。

- Adaptive Server 在内部将基于函数的索引键表示为隐藏的实现计算列。基于函数的索引键值存储在数据行和索引页上，因此，它具有实现计算列的所有属性。
- Adaptive Server 假定所有基于函数或表达式的索引键都是确定性的。在查询中引用它们时，将使用索引页中已存储的预求值结果；不会对索引键重新进行求值。
- 仅当更新基于函数的索引键的基列时，才会更新此预求值的结果。
- 不要像示例 2 中那样，使用非确定性的函数作为索引。否则，可能会出现意想不到的结果。

给用户分配权限

grant 和 revoke 命令控制 Adaptive Server 命令和对象保护系统。可使用 grant 命令将各种权限分配给用户、组和角色，并使用 revoke 命令撤消这些权限，包括：

- 创建数据库
- 在数据库中创建对象
- 访问表、视图和列
- 执行存储过程

某些命令可由所有用户随时使用，不需要任何权限。其它命令只能由具有某种身份 （例如，系统管理员）的用户使用，且这种权限不能移交。

能否分配可授予和撤消的命令权限是由以下因素决定的：每个用户的身份 （如系统管理员、数据库所有者或数据库对象所有者）以及是否授权特定用户将该权限授予其他用户。

所有者不会自动获得其它用户拥有的对象权限。但数据库所有者或系统管理员可通过使用 setuser 命令临时使用对象所有者的身份并编写相关的 grant 或 revoke 语句来获得任何权限。

可以使用 grant 和 revoke 来分配以下两种类型的权限：**对象访问权限**和**对象创建权限**。

对象访问权限控制访问某些数据库对象的某些命令的使用。例如，必须获得对 authors 表使用 select 命令的权限。对象访问权限由对象所有者授予或撤消。

若为 Mary 和 Joe 授予对象访问权限以便对 titles 表执行 insert 和 delete 命令，请使用以下命令：

```
grant insert, delete
on titles
to mary, joe
```

对象创建权限控制创建对象的命令的使用。这些权限只能由系统管理员或数据库所有者授予。

例如，若要撤消 Mary 在当前数据库中创建表和规则的权限，请使用以下命令：

```
revoke create table, create rule
from mary
```

有关使用 grant 和 revoke 的完整信息，请参见 《参考手册：命令》和《系统管理指南：卷 1》中的第 17 章 “管理用户权限”。

系统安全员也可使用角色，简化授予和撤消访问权限的任务。

例如，系统安全员可创建角色，请求对象所有者授权给每一角色，并根据雇员在公司中的职责将这些用户定义的角色授予他们，而不是让对象所有者将每一对象的特权分别授予每一雇员。系统安全员还可以撤消授予雇员的用户定义角色。

有关用户定义的角色的完整信息，请参见 《系统管理指南：卷 1》中的第 17 章 “管理用户权限”。

获得有关数据库和表的信息

Adaptive Server 包含许多可用于获得有关数据库、表和其它数据库对象的信息的过程和函数。本节描述其中一些。请参见 《参考手册：过程》和 《参考手册：构件块》。

获得有关数据库的帮助

sp_helpdb 可报告有关指定数据库或所有 Adaptive Server 数据库的信息。它报告分配给数据库的每个片段的名称、大小和使用情况。

```
sp_helpdb [dbname]
```

下例显示有关 pubs2 的报告；它使用 8K 的页大小。

```
sp_helpdb pubs2

name      db_size    owner     dbid created        status
--------- ---------- --------- ---- -------------- --------------
pubs2     20.0 MB     sa            4 Apr 25, 2005   select
     into/bulkcopy/pllsort, trunc log on chkpt, mixed log and data

device_fragments    size           usage          created      free kbytes
------------------- -------------- -------------- ----------   ------------
master              10.0MB         data and log  Apr 13 2005          1792
pubs_2_dev         10.0MB         data and log  Apr 13 2005         9888

device                  segment
----------------------- -----------------------
master                  default
master                  logsegment
master                  system
pubs_2_dev              default
pubs_2_dev              logsegment
pubs_2_dev              system
pubs_2_dev              seg1
pubs_2_dev              seg2
```

sp_databases 列出服务器中的所有数据库。例如：

```
sp_databases
database_name     database_size  remarks
----------------- -------------  ------------
master                     5120  NULL
model                      2048  NULL
pubs2                      2048  NULL
pubs3                      2048  NULL
```

```
sybsecurity                 5120  NULL
sybsystemprocs             30720  NULL
tempdb                      2048  NULL

(7 rows affected, return status = 0)
```

若要查明数据库的所有者，请使用 sp_helpuser：

```
sp_helpuser dbo

Users_name    ID_in_db Group_name   Login_name
------------- -------- ------------ ------------
dbo                  1 public       sa
```

可以使用 db_id() 和 db_name() 来确定当前数据库。例如：

```
select db_name(), db_id()

------------------------------ ------
master                              1
```

获得有关数据库对象的帮助

Adaptive Server 提供系统过程、目录存储过程和内置函数，它们可返回有关数据库对象 （如表、列和约束）的有用信息。

对数据库对象使用 *sp_help*

使用 sp_help 可显示有关指定的数据库对象（即 sysobjects 中列出的任何对象）、指定的数据类型 （在 systypes 中列出）或当前数据库中的所有对象和数据类型的信息。

sp_help [*objname*]

下面是 publishers 表的输出：

```
Name                Owner         Object_type     Create_date
----------------    -----------   -------------   ------------------------------
-
publishers          dbo           user table      Nov 9 2004 9:57AM

(1 row affected)
Column_name Type     Length   Prec  Scale   Nulls   Default_name   Rule_name
----------- -------  ------   ----- ------- ------- -------------- ----------
pub_id      char          4    NULL  NULL        0  NULL           pub_idrule
pub_name    varchar      40    NULL  NULL        1  NULL           NULL
city        varchar      20    NULL  NULL        1  NULL           NULL
state       char          2    NULL  NULL        1  NULL           NULL
```

```
Access_Rule_name    Computed_Column_object     Identity
------------------- -------------------------  ------------
NULL                NULL                                  0
NULL                NULL                                  0
NULL                NULL                                  0
NULL                NULL                                  0

Object has the following indexes
index_name index_keys  index_description  index_max_rows_per_page
---------- ----------  ---------- ------- -------------
pubind     pub_id      clustered, unique              0

index_fill_factor  index_reservepagegap   index_created      index_local
------------------ ---------------------- ----------------   ----------------
                0                      0  Nov 9 2004 9:58AM  Global Index
(1 row affected)
index_ptn_name   index_ptn_segment
---------------- --------------------
pubind_416001482  default

(1 row affected)
keytype  object      related_object  related_keys
-------  ----------  --------------  ------------
primary  publishers  -- none --      pub_id, *, *, *, *, *
foreign  titles      publishers      pub_id, *, *, *, *, *

(1 row affected)

name         type        partition_type     partitions     partition_keys
--------     --------    ------------------ -------------- ------------------
publishers   base table  roundrobin                     1   NULL

partition_name        partition_id  pages   segment     Create_date
------------------    ------------  ------  ----------- ----------------------
publishers_416001482  416001482         1   default     Nov 9 2004 9:58AM

Partition_Conditions
---------------------
NULL

Avg_pages   Max_pages   Min_pages   Ratio

(return status = 0)

No defined keys for this object.
```

```
name    type       partition_type   partitions   partition_keys
------- ---------- ---------------- ------------ ---------------
mytable base table roundrobin                  1 NULL

partiton_name       partition_id   pages     segment   create_date
-----------------   -------------- --------- --------- -----------------------
-
mytable_1136004047  1136004047            1  default     Nov 29 2004 1:30PM

partition_conditions
--------------------
NULL

Avg_pages   Max_pages   Min_pages   Ratio(Max/Avg)        Ration(Min/Avg)
----------- ----------- ----------- --------------------- -------------------
          1           1           1              1.000000                1.000
000
Lock scheme Allpages
The attribute 'exp_row_size' is not applicable to tables with
allpages lock scheme.
The attribute 'concurrency_opt_threshold' is not applicable to
tables with allpages lock scheme.

exp_row_size reservepagegap fillfactor max_rows_per_page identity_gap
 ------------ -------------- ---------- ----------------- ------------
            1              0          0                 0            0
(1 row affected)
concurrency_opt_threshold   optimistic_index_lock   dealloc_first_txtpg
--------------------------- ----------------------- --------------------
                          0                       0                    0
(return status = 0)
```

如果执行 sp_help 而没有提供对象名，则生成的报告将显示 sysobjects 中的每个对象，以及其名称、所有者和对象类型。也显示 systypes 中的每个用户定义的数据类型及其名称、存储类型、长度、是否允许空值，以及绑定到它的任何缺省值或规则。该报告也注明是否已经为表或视图定义了任何主键或**外键**列。

sp_help 列出表中的任何索引，包括通过定义唯一或主键约束创建的索引。但不提供关于为表定义的完整性约束的任何信息。使用 sp_helpconstraint 可显示有关任何完整性约束的信息。

使用 *sp_helpconstraint* 查找表的约束信息

sp_helpconstraint 报告有关为表指定的声明参照完整性约束的信息，包括约束名称和缺省值的定义、唯一或主键约束、参照或检查约束。sp_helpconstraint 也报告与指定表相关联的引用的数目。

它的语法为：

sp_helpconstraint [*objname*] [, detail]

objname 是被查询的表的名称。如果不包括表名，则 sp_helpconstraint 显示与当前数据库中每个表相关联的引用的数目。如果包括表名，则 sp_helpconstraint 报告与该表相关联的完整性约束的名称、定义和数目。detail 选项也返回有关约束的用户或错误消息的信息。

例如，假定为 pubs3 中的 store_employees 表运行 sp_helpconstraint。

```
name                          defn
----------------------------  --------------------------------
store_empl_stor_i_272004000   store_employees FOREIGN KEY
                              (stor_id) REFERENCES stores(stor_id)
store_empl_mgr_id_288004057   store_employees FOREIGN KEY
                              (mgr_id) SELF REFERENCES
                              store_employees(emp_id)
store_empl_2560039432         UNIQUE INDEX( emp_id) :
                              NONCLUSTERED, FOREIGN REFERENCE

(3 rows affected)

Total Number of Referential Constraints: 2
Details:
-- Number of references made by this table: 2
-- Number of references to this table: 1
-- Number of self references to this table: 1

Formula for Calculation:
Total Number of Referential Constraints
= Number of references made by this table
+ Number of references made to this table
- Number of self references within this table
```

若要找出与当前数据库中的任何表相关联的参照约束的最大数目，请运行 sp_helpconstraint 并且不指定表名，例如：

```
sp_helpconstraint

id          name                      Num_referential_constraints
----------- ------------------------ ----------------------------
   80003316 titles                                              4
   16003088 authors                                             3
```

```
 176003658 stores                                          3
 256003943 salesdetail                                     3
 208003772 sales                                           2
 336004228 titleauthor                                     2
 896006223 store_employees                                 2
  48003202 publishers                                      1
 128003487 roysched                                        1
 400004456 discounts                                       1
 448004627 au_pix                                          1
 496004798 blurbs                                          1

(11 rows affected)
```

在此报告中， pubs3 数据库中表 titles 具有最大数目的参照约束。

查找表占用的空间

可以使用 sp_spaceused 来查明表占用的空间：

sp_spaceused [*objname*]

sp_spaceused 计算并显示表或聚簇或非聚簇索引使用的行和数据页的数量。

要显示 titles 表使用空间的报告：

```
sp_spaceused titles

name    rows   reserved  data  index_size  unused

------- -----  --------- ----- ---------   ------
titles  18     48 KB     6 KB  4 KB        38 KB

(0 rows affected)
```

如果没有将对象名包括在内，则 sp_spaceused 显示所有数据库对象使用空间的汇总。

列出表、列和数据类型

目录存储过程用于在表形式的系统表中检索信息。可为某些参数提供通配符。

按以下格式使用 sp_tables 时，它列出数据库中所有的用户表：

sp_tables @table_type = "'TABLE'"

sp_columns 返回数据库中一个或多个表的任意或所有列的数据类型。可使用通配符获得多个表或列的信息。

例如，下列命令返回名称中含 “sales”的所有表中的、包括字符串 “id”的所有列的信息：

```
sp_columns "%sales%", null, null, "%id%"
```

```
table_qualifier table_owner
       table_name     column_name
       data_type type_name  precision  length  scale radix  nullable
       remarks

ss_data_type colid
---------------  -----------
      ----------       -----------
      ---------  --------- ---------  ------  ----- -----  --------
      -------

------------ -----
pubs2              dbo
       sales           stor_id
       1          char      4           4       NULL  NULL    0
 NULL

47            1
pubs2            dbo
       salesdetail     stor_id
       1         char        4           4       NULL  NULL    0
 NULL

4             1
pubs2          dbo
        salesdetail     title_id
        12      varchar      6           6       NULL  NULL 0
     NULL

39            3

(3 rows affected, return status = 0)
```

查找对象名和 ID

可以使用 object_id() 和 object_name() 来确定对象的 ID 和名称。例如：

```
select object_id("titles")

----------
  208003772
```

对象名和 ID 存储在 sysobjects 系统表中。

第 8 章

添加、更改、传输和删除数据

在创建数据库、表和索引后，可以将数据置于表中并处理它，即根据需要添加、更改和删除数据。

简介

用于添加、更改或删除数据的命令称为**数据修改语句**。这些命令有：

- insert — 在表中添加新行。
- update — 更改表中的现有行。
- writetext — 添加或更改 text、 unitext 和 image 数据，而无需在系统的事务日志中写入冗长的更改内容。
- delete — 从表中删除特定行。
- truncate table — 从表中删除所有行。

请参见 《参考手册：命令》。

也可使用批量复制实用程序 bcp，通过从文件中传输数据，来将数据添加到表中。请参见 《实用程序指南》。

在每个语句中，可使用 insert、 update 或 delete 修改某个表中的数据。Transact-SQL 对这些命令的增强作用表现在，能根据其它表 （甚至其它数据库）中的数据进行修改。

数据修改命令也能执行于视图上，但是有一些限制。请参见第 12 章 “视图：限制访问数据”。

权限

数据修改命令不必每个人都可用。数据库所有者和数据库对象所有者能使用 grant 和 revoke 命令，指定能够修改数据的用户。

可将使用数据修改命令的任意组合的权限或特权授予单个用户、组或公众。《系统管理指南：卷 1》中的第 17 章 “管理用户权限”中讨论了相关权限。。

参照完整性

insert、 update、 delete、 writetext 和 truncate table 允许在不更改其它表中的相关数据的情况下对数据进行更改，但这可能会导致不一致。

例如，如果更改 authors 表中 “Sylvia Panteley”的 au_id 条目，则必须同样更改 titleauthor 表中的条目，以及数据库中所有其它有包含该值的列的表。否则，将找不到 Panteley 女士所撰写书籍的名称之类的信息，因为无法建立与她的 au_id 列的连接。

在数据库中，保持数据修改在所有表中的一致性称为**参照完整性**。管理它的一个方法是为表定义参照完整性约束。另一种方法是创建称为触发器的特殊过程，当对特定表或列发出 insert、 update 和 delete 命令时，这些触发器就会生效 （truncate table 命令不会由触发器或参照完整性约束捕获）。请参见第 20 章 “触发器：强制实施参照完整性”和第 7 章 “创建数据库和表”。

要从参照完整性表中删除数据，首先更改被引用的表，然后更改引用表。

事务

受每条数据修改语句影响的每行的新旧状态，被写入事务日志。这意味着，如果通过发出 begin transaction 命令开始事务，而后意识有错误发生并回退事务，则数据库恢复到以前的状态。

注释　不能回退在远程 Adaptive Server 上，通过远程过程调用 (RPC) 所做的更改。

当 select/into bulkcopy 数据库选项设定为 false 时， writetext 是一个例外。

writetext 的缺省操作模式*不*记录事务。这可避免事务日志被极长的数据块填满， text、 unitext 和 image 字段可能包含此类数据块。要记录此命令所作更改的日志，必须使用 writetext 命令的 with log 选项。

第 22 章 “事务：维护数据一致性和恢复”中对事务进行了更完整的论述。

使用样本数据库

如果按照本章中的示例操作， Sybase 建议从 pubs2 或 pubs3 数据库的原始副本开始，并在完成时使其返回该状态。请寻求系统管理员的帮助，获得任一这些数据库的原始副本。

为了防止所做的任何更改变成永久性更改，可以将输入的所有语句置于一个事务中，然后在完成本章后终止该事务。例如，键入以下语句启动事务：

```
begin tran modify_pubs2
```

此事务名为 modify_pubs2。可以通过键入以下语句，随时取消此事务并使数据库恢复为其在事务开始前的状态：

```
rollback tran modify_pubs2
```

数据类型输入规则

Adaptive Server 提供的几种数据类型具有特殊的数据输入和搜索规则。有关数据类型的详细信息，请参见第 7 章 “创建数据库和表”。

char、*nchar*、*unichar*、*univarchar*、*varchar*、*nvarchar*、*unitext* 和 *text*

所有 character、 text、 date 和 time 数据在作为文字输入时必须用单引号或双引号引起来。如果 set 命令的 quoted_identifier 选项设定为 on，则使用单引号。如果使用双引号， Adaptive Server 视文本为标识符。

字符文字可以是任意长度，而不管数据库的逻辑页大小是多少。如果文字长于 16 KB（16384 字节），Adaptive Server 会将其视为 text 数据，此类数据在隐式和显式转换为其它数据类型时存在某些限制性规则。请参见 《参考手册：构件块》中的第 1 章 “系统数据类型和用户定义的数据类型”，以了解 character 和 text 数据类型的不同行为。

如果将字符数据插入 char、 nchar、 unichar、 univarchar、 varchar 或 nvarchar 列中，并且为该列指定的长度小于插入数据长度，则会截断输入内容。将 string_rtruncation 选项设置为 on 可在出现这种情况时收到警告消息。

注释 此截断规则适用于所有字符数据，无论字符数据是驻留在列、变量还是文字字符串中。

有两种方法可以指定字符输入中的文字引号：

- 使用两个引号。例如，如果在某个字符条目开头用了一个单引号，并要将另一个单引号作为该条目的一部分，则可使用两个单引号：'I don’t 'understand.' '；对于双引号：“He said, “ “It’s not really confusing.” ”
- 用相对类型的引号来括带引号的内容。也就是说，用单引号括起包含双引号的条目，反之亦然。例如：“George said, 'There must be a better way.' “'

如果输入的字符串超出屏幕宽度，可在转到下一行之前输入反斜杠 (\)。

使用在第 2 章 “查询：从表中选择数据”中描述的 like 关键字和通配符，可搜索 character、 text 和 datetime 数据。

请参见 《参考手册：构件块》中的第 1 章 “系统数据类型和用户定义的数据类型”，以了解有关插入 text 数据的详细信息以及有关 character 数据中尾随空白的信息。

日期和时间

Adaptive Server 允许使用数据类型 datetime、 smalldatetime、 date、 time、 bigdatetime 和 bigtime。

日期和时间数据的显示和输入格式提供了范围广泛的日期输出格式，并且可以识别多种不同的输入格式。显示和输入格式是分别控制的。缺省的显示格式提供类似 “Apr 15 1997 10:23PM” 的输出。 convert 命令提供了用于显示秒和毫秒的选项，以及按其它日期分量顺序显示日期的选项。有关显示日期值的详细信息，请参见第 16 章 “在查询中使用内置函数”。

Adaptive Server 识别范围广泛的日期数据输入格式。始终忽略大小写，并且空格可在日期分量间的任何地方出现。当输入 datetime 和 smalldatetime 值时，始终用单引号或双引号将它们括起来。如果 quoted_identifier 选项设置为 on，则使用单引号；如果使用双引号，Adaptive Server 视文本为标识符。

Adaptive Server 分别识别数据的两种日期和时间选项，因此时间可以位于日期之前或之后。可以忽略任何一部分，在这种情况下， Adaptive Server 使用缺省值。缺省的日期和时间为 January 1, 1900, 12:00:00:000AM。

对于 datetime，可使用的最早日期为 1753 年 1 月 1 日；最晚日期为 9999 年 12 月 31 日。对于 smalldatetime，可使用的最早日期为 1900 年 1 月 1 日；最晚日期为 2079 年 6 月 6 日。对于 bigdatetime，可输入的最早日期为 0001 年 1 月 1 日，最晚日期为 9999 年 12 月 31 日。如果日期比这些日期早或晚，则必须将其作为 char 或 unichar 或者 varchar 或 univarchar 值进行输入、存储和处理。 Adaptive Server 拒绝在这些范围内不能识别为日期的所有值。

对于 date，可使用的最早日期为 0001 年 1 月 1 日，最晚日期为 9999 年 12 月 31 日。如果日期比这些日期早或晚，则必须将其作为 char 或 unichar ； 或者 varchar 或 univarchar 值进行输入、存储和处理。Adaptive Server 拒绝在这些范围内不能识别为日期的所有值。

对于 time，最早时间为 12:00AM，最晚时间为 11:59:59:999。对于 bigtime，最早时间为 12:00:00.000000 AM，最晚时间为 11:59:59.999999PM。

输入时间

各时间部分的顺序对于数据的时间部分非常重要。首先输入小时；然后分；然后秒；然后毫秒；最后 AM （或 am）或 PM (pm)。 12AM 是午夜， 12PM 是正午。若要使数字表示时间，值中必须包含一个冒号，或者包含 AM 或 PM 标记。smalldatetime 仅精确到分钟。time 精确到毫秒。

毫秒前可以带一个冒号或句点。如果前面带冒号，数字就表示千分之多少秒。如果前面带句点，单个数字位表示十分之多少秒，两个数字位表示百分之多少秒，三个数字位表示千分之多少秒。

例如，“12:30:20:1”表示 12:30 过二十又千分之一秒； “12:30:20.1”表示 12:30 过二十又十分之一秒。

可接受的时间数据格式为：

```
14:30
14:30[:20:999]
14:30[:20.9]
4am
4 PM
[0]4[:30:20:500]AM
```

bigdatetime 和 bigtime 的显示和输入格式包括微秒。必须按如下方式指定时间：

hours[:minutes[:seconds[.microseconds]] [*AM* | *PM*]

hours[:minutes[:seconds[number of milliseconds]] [*AM* | *PM*]

使用 12 AM 表示午夜，使用 12 PM 表示正午。 bigtime 值必须包含一个冒号，或者 AM 或 PM 标记。 AM 或 PM 可以采用大写、小写或大小写混合的方式输入。

对秒数的说明可以包括一个小数点，后面跟小数部分；也可以包括一个冒号，后面跟毫秒数。例如，“12:30:20:1”表示 12:30 过二十秒又一毫秒； “12:30:20.1”表示 12:30 过二十又十分之一秒。

若要存储包括微秒的 bigdatetime 或 bigtime 时间值，请使用小数点指定字符串文字。“00:00:00.1”表示午夜过十分之一秒； “00:00:00.000001”表示午夜过百万分之一秒。在冒号之后指定秒数部分的任何值仍然表示毫秒数。例如，“00:00:00:5”表示 5 毫秒。

输入日期

当以带分隔符的数字字符串形式输入日期时， set dateformat 命令指定日期分量的顺序 （月、日和年）。根据指定语言的缺省日期格式， set language 也会影响日期的格式。缺省语言为 us_english，缺省日期格式为 *mdy*。请参见 《参考手册：命令》。

注释　dateformat 只影响以带分隔符的数字形式输入的日期，例如 “4/15/90” 或 “20.05.88”。它不影响月份以字母格式提供的日期 （例如 “April 15, 1990”）或没有分隔符的日期 （例如 “19890415”）。

日期格式

Adaptive Server 识别三种基本的日期格式，如下所述。每种格式必须用引号引起来，并且可在前面或后面加上时间规范，如第 300 页的 “输入时间”中所述。

- 月份按字母格式输入。
  - 用字母指定日期的有效格式为：

```
Apr[il] [15][,] 1997
Apr[il] 15[,] [19]97
Apr[il] 1997 [15]
[15] Apr[il][,] 1997
15 Apr[il][,] [19]97
15 [19]97 apr[il]
[15] 1997 apr[il]
1997 APR[IL] [15]
1997 [15] APR[IL]
```

  - 如当前语言规范所规定，月份可以是三个字符的缩写，也可以是完整的月名称。
  - 逗号是可选的。
  - 忽略大小写。
  - 如果只指定年份的最后两位数字，则小于 50 的值解释为 “20yy”，大于或等于 50 的值解释为 “19yy”。
  - 只有在日被忽略，或需要的不是缺省世纪时，才输入世纪。
  - 如果日丢失， Adaptive Server 将该月的第一天作为缺省值。
  - 以字母格式指定月份时，将忽略 dateformat 设置 （请参见 《参考手册：命令》）。

- 月份在带有斜杠 (/)、连字符 (-) 或句点 (.) 分隔符的字符串中以数字格式输入。
  - 必须指定月、日、年。
  - 字符串必须是如下格式：

    <num> <sep> <num> <sep> <num> [<time spec>]

    或：

    [<time spec>] <num> <sep> <num> <sep> <num>
  - 对日期分量值的解释取决于 dateformat 的设置。如果顺序与设置不匹配，则不会将值解释为日期 （由于值超出了范围），或者对值进行错误的解释。例如，“12/10/08”可解释为六个不同日期之一，具体取决于 dateformat 设置。请参见 《参考手册：命令》。
  - 若要采用 *mdy* dateformat 格式输入 “1997 年 4 月 15 日”，可以使用以下这些格式：

    ```
    [0]4/15/[19]97
    [0]4-15-[19]97
    [0]4.15.[19]97
    ```
  - 用 “/” 作为分隔符的其它输入顺序如下所示，也可以使用连字符或句点：

    ```
    15/[0]4/[19]97 (dmy)
    1997/[0]4/15 (ymd)
    1997/15/[0]4 (ydm)
    [0]4/[19]97/15 (myd)
    15/[19]97/[0]4 (dym)
    ```
- 以 4 位、 6 位或 8 位无分隔符的字符串形式给出日期，或以空字符串形式给出日期，或仅给出时间值而没有日期值。
  - 对于此输入格式， dateformat 总是被忽略。
  - 如果给出四位数字，则该字符串被解释为年，月设定为一月，天为月的第一天。不能忽略世纪。
  - 6 位或 8 位字符串总是解释为 *ymd* ；月和天必须总是两位。以下格式是可识别的：[19]960415.
  - 空字符串 (“”) 或缺少日期被解释为基准日期，即 1900 年 1 月 1 日。例如，像 “4:33” 这样没有日期的时间值被解释为 “1900 年 1 月 1 日上午 4:33”。

- 当 weekday 或 dw 与 datename 一起使用时， set datefirst 命令指定一周内的第几天（星期日、星期一等等）；当与 datepart 一起使用时，该命令指定一个相应的数字。用 set language 更改语言也影响日期的格式，取决于该语言缺省的周的第一天。对于缺省语言 us_english，缺省的 datefirst 设置是 Sunday=1、Monday=2，等等；而对其它语言则为 Monday=1、 Tuesday=2 等等。可在每个会话期用 set datefirst 更改缺省行为。请参见《参考手册：命令》。

搜索日期和时间

可以将 like 关键字和通配符用于 datetime、 malldatetime、 bigdatetime、bigtime、 date 和 time 数据，也可以用于 char、 unichar、 nchar、varchar、 univarchar、 nvarchar 以及 text 和 unitext。在将 like 用于 date 和 time 值时， Adaptive Server 首先将日期转换为标准的 date/time 格式，然后转换为 varchar 或 univarchar。由于 datetime 和 smalldatetime 的标准显示格式不包括秒或毫秒，因此，无法使用 like 和匹配模式来搜索秒或毫秒。使用类型转换函数 convert 来搜索秒和毫秒。

在搜索 datetime、 bigtime、 bigdatetime 或 smalldatetime 值时，最好使用 like，因为这些类型的数据输入可能包含多种日期分量。例如，如果将值 “9:20”插入名为 arrival_time 的列，则以下子句将找不到它，因为 Adaptive Server 将此输入内容转换为 “Jan 1, 1900 9:20AM”：

```
where arrival_time = "9:20"
```

但是，以下子句将能够找到它：

```
where arrival_time like "%9:20%"
```

这也同样适用于使用 date 和 time 数据类型的情况。

如果使用了 like，并且月份中的日期小于 10，则必须在月份和日期之间插入两个空格，以便与 datetime 值的 varchar 转换形式相匹配。同样，如果小时小于 10，转换结果将在年份和小时之间放置两个空格。 like May 2% 子句在 “May”和 “2”之间包含一个空格，它可找出从 5 月 20 日到 5 月 29 日之间的所有日期，但找不到 5 月 2 日。由于 datetime 值仅在进行 like 比较时才转换为 varchar，因此不需要在进行其它日期比较时插入额外的空格，仅在使用 like 时才需要这样做。

binary、 *varbinary* 和 *image*

在将 binary、 varbinary 或 image 数据作为文字输入时，必须为数据附加前缀 “0x”。例如，若要输入 “FF”，请键入 “0xFF”。但是，不要用引号括起以 “0x” 开头的数据。

二进制文字可以是任意长度，而不管数据库的逻辑页大小是多少。如果文字的长度小于 16 KB（16384 字节），Adaptive Server 会将这样的文字视为 varbinary 数据。如果文字的长度超过 16 KB， Adaptive Server 会将这样的文字视为 image 数据。

如果将 binary 数据插入到某个列中，并且为该列指定的长度小于插入数据的长度，将截断输入内容而不显示警告消息。

如果 binary 或 varbinary 列的长度为 10，则表明为 10 个字节，每个字节存储 2 个十六进制数字。

当在 binary 或 varbinary 列上创建缺省值时，在其前面加上 “0x”。

请参见 《参考手册：构件块》中的第 1 章 “系统数据类型和用户定义的数据类型”，以了解 binary 数据类型和 image 数据类型的不同行为以及有关十六进制值中尾随零的信息。

money 和 *smallmoney*

输入的带有 E 符号的货币值被解释为 float。当一个条目作为 money 或 smallmoney 值存储时，可能会导致拒绝该条目或丧失该条目的某些精度。

输入 money 和 smallmoney 值时，可以在其前面加上货币符号，如美元符号 ($)、日元符号 (¥) 或英镑符号 (£)，也可以不加。若要输入负值，请在货币符号后加上一个负号。在条目中不要包含逗号。

不能带逗号输入 money 或 smallmoney 值，尽管 money 或 smallmoney 数据的缺省输出格式为每三个数字后放一个逗号。当显示 money 或 smallmoney 值时，它们将舍入到最接近的分。除了模运算外的所有算术运算都可用于 money。

float、 *real* 和 *double precision*

可以将近似数值类型 （float、 real 和 double precision）作为后跟可选指数的尾数输入。尾数可包含一个正号或负号以及一个小数点。字符“e”或 “E”后的指数可包含一个符号，但不能有小数点。

为了计算近似数值数据， Adaptive Server 将尾数乘以 10 的给定指数次幂。表 8-1 显示了 float、 real 和 double precision 数据的示例：

表 8-1：计算数值数据

输入的数据	尾数	指数	值
10E2	10	2	$10 * 10^2$
15.3e1	15.3	1	$15.3 * 10^1$
-2.e5	-2	5	$-2 * 10^5$
2.2e-1	2.2	-1	$2.2 * 10^{-1}$
+56E+2	56	2	$56 * 10^2$

列的二进制精度决定了尾数中所允许的二进制位的最大位数。对于 float 列，可以指定高达 48 位的精度；对于实数和双精度列，其精度取决于计算机。如果值超过了列的二进制精度， Adaptive Server 将该条目标记为一个错误。

decimal 和 *numeric*

精确数值类型 （dec、 decimal 和 numeric）以可选的正号或负号开头，并且可以包含小数点。精确数值数据的值取决于列的小数 *precision* 和 *scale*，可以用以下语法指定：

datatype [(*precision* [, *scale*])]

Adaptive Server 将每一种精度和标度的组合都作为一种不同的数据类型来对待。例如， numeric (10,0) 和 numeric (5,0) 是两种不同的数据类型。精度和标度确定了可以存储在 decimal 或 numeric 列中的值的范围：

- 精度指定了能够在该列中存储的最大小数位数。它包括小数点左右两侧的所有位数。可以将精度范围指定为 1 至 38 位，或者使用缺省的 18 位精度。
- 标度指定了能够存储到小数点右侧的最大位数。标度必须小于或等于精度。可在 0 至 38 位范围之间指定标度，或者使用缺省的 0 位标度。

如果值超过了列的精度或标度， Adaptive Server 将该条目标记为一个错误。以下是一些有效 dec 和 numeric 数据的示例：

表 8-2：数值数据的有效精度和标度

输入的数据	数据类型	精度	标度	值
12.345	numeric(5.3)	5	3	12.345
-1234.567	dec(8,4)	8	4	-1234.567

以下输入会导致错误，因为它们超过了列的精度或标度：

表 8-3：数值数据的无效精度和标度

输入的数据	数据类型	精度	标度
1234.567	numeric(3,3)	3	3
1234.567	decimal(6)	6	1

整数类型及其无符号形式

正如上一节中所述，可以将带有 E 符号的数字值插入到 bigint、 int、 smallint、 tinyint、 unsigned bigint、 unsigned int 和 unsigned smallint 列中。

timestamp

不能将数据插入 timestamp 列中。必须通过在列中键入 “NULL” 来插入显式空值，或通过提供跳过 timestamp 列的列列表来使用隐式空值。每次插入或更新后， Adaptive Server 更新 timestamp 值。有关详细信息，请参见第 307 页的 “插入数据到指定列”。

添加新数据

可以使用 insert 命令以两种方式将行添加到数据库中；使用 values 关键字或 select 语句：

- values 关键字指定新行中部分或全部列的值。使用 values 关键字的 insert 命令的简化语法为：

  insert *table_name*
  values (*constant1*, *constant2*, ...)

- 可以在 insert 语句中使用 select 语句，从一个或多个表（上限为 50 个表，包括要插入的表）中取值。使用 select 语句的 insert 命令的简化语法为：

  ```
  insert table_name
      select column_list
      from table_list
      where search_conditions
  ```

  注释　不能在位于 insert 语句内部的 select 语句中使用 compute 子句，因为包含 compute 的语句不会生成常规行。

  使用 insert 来添加 text、 unitext 或 image 值时，所有数据将写入事务日志中。可以使用 writetext 命令来添加这些值，而不记录可能包含 text、 unitext 或 image 值的长数据块。请参见第 307 页的 “插入数据到指定列”和第 322 页的 “更改 text、 unitext 和 image 数据”。

使用 *values* 添加新行

以下 insert 语句在 publishers 表中添加一个新行，从而为该行中的每个列指定一个值：

```
insert into publishers
values ("1622", "Jardin, Inc.", "Camden", "NJ")
```

请注意，键入数据值的顺序应与初始 create table 语句中列名称的顺序相同，即，首先是 ID 号，然后是名称，接着是城市，最后是州。 values 数据用括号括起来，并且所有的字符数据用单引号或双引号引起。

对添加的每一行，使用单独的 insert 语句。

插入数据到指定列

可通过仅指定某些列及其数据，将数据添加到某行的这些列。不包含在列列表中的所有其它列，必须定义为允许空值。被跳过的列可接受缺省值。如果被跳过的列有绑定缺省值，则使用缺省值。

您可能非常希望使用这种形式的 insert 命令，在行中插入除 text、 unitext 或 image 以外的所有其它值，然后使用 writetext 插入长数据值，以免在事务日志中存储这些值。也可使用这种形式的命令跳过 timestamp 数据。

例如，如果仅在 pub_id 和 pub_name 列中添加数据，您需要使用如下命令：

```
insert into publishers (pub_id, pub_name)
values ("1756", "The Health Center")
```

列出列名的顺序必须与列出值的顺序一致。下面的示例产生与前一个示例相同的结果：

```
insert publishers (pub_name, pub_id)
values("The Health Center", "1756")
```

两个 insert 语句都将 “1756”放到标识号列中，并将 “The Health Center”放到出版社名称列中。因为 publishers 中的 pub_id 列具有唯一索引，所以无法执行两条 insert 语句；第二次尝试插入 pub_id 值 “1756”会产生错误消息。

以下 select 语句显示已添加到 publishers 中的行：

```
select *
from publishers
where pub_name = "The Health Center"

pub_id  pub_name            city    state
------- ------------------   ------  -------
1756    The Health Center    NULL    NULL
```

Adaptive Server 将 Null 值输入 city 和 state 列中，因为在 insert 语句中没有给这些列赋值，并且 publisher 表允许这些列中包含 Null 值。

限制列数据：规则

可创建一个规则并将其绑定到列或用户定义的数据类型。规则约束能或不能添加的数据类型。

publishers 表的 pub_id 列是一个示例。一个名为 pub_idrule 的规则绑定到该列，该规则指定可接受的 publisher 标识号。可接受的 ID 是 “1389”、“0736”、“0877”、“1622”和 “1756”，或任何以 “99”开头的四位数字。如果输入任何其它数字，则会显示错误消息。

出现此类错误消息时，您可能需要使用 sp_helptext 来查看规则定义：

```
sp_helptext pub_idrule

---------
         1

(1 row affected)

text
```

```
------------------------------------------------------
create rule pub_idrule
as @pub_id in ("1389", "0736", "0877", "1622", "1756")
or @pub_id like "99[0-9][0-9]"

(1 row affected)
```

有关特定规则的一般详细信息，请使用 sp_help。或用表名称作为参数，使用 sp_help，查看是否有任何列拥有规则。请参见第 14 章 “为数据定义缺省值和规则”。

使用 NULL 字符串

只有在以下情况下，列才会包含空值：在 create table 语句中为列指定了 NULL，在列中显式输入了 NULL （无引号），或者尚未在列中输入任何数据。应避免将字符串 "NULL"（带引号）当作字符列的数据输入。而应使用 “N/A”、“none”或类似值。要显式输入值 NULL，请*不要*使用单引号或双引号。

将 NULL 显式插入列中：

```
values({expression | null}
    [, {expression | null}]...)
```

以下示例显示了两个等效的 insert 语句。在第一个语句中，用户将 NULL 显式插入 t1 列中。在第二个语句中，Adaptive Server 为 t1 提供了 NULL 值，因为用户没有指定显式列值：

```
create table test
(t1 char(10) null, t2 char(10) not null)
insert test
values (null, "stuff")
insert test (t2)
values ("stuff")
```

NULL 不是空字符串

空字符串 （“ ”或 ‘ ’）总是作为单个空格存储在变量和列数据中。以下并置语句相当于 “abc def”，而不是 “abcdef”：

```
"abc" + "" + "def"
```

空字符串从不会求值为 NULL。

将 NULL 插入不允许 NULL 的列中

若要使用 select 将数据从某些字段为空值的表中插入不允许空值的表中，必须为初始表中的所有 NULL 条目提供替代值。例如，为了将数据插入不允许空值的 advances 表中，以下示例用 “0”替代 NULL 字段：

```
insert advances
select pub_id, isnull(advance, 0) from titles
```

如果不使用 isnull 函数，此命令就会将带有非空值的所有行插入 advances，并对在 titles 的 advance 列中包含 NULL 的所有行产生错误消息。

如果无法对数据进行这种替代，则不能将包含空值的数据插入已有 NOT NULL 规定的列。

添加所有列都没有值的行

当仅给行中某些列指定值时，没有值的列可出现下列四种情况之一：

- 如果列或用户定义的列数据类型存在缺省值，则输入缺省值。有关详细信息，请参见第 14 章 “为数据定义缺省值和规则”或《参考手册：命令》。
- 如果创建表时为列指定了 NULL，并且列或数据类型不存在缺省值，则输入 NULL。请参见《参考手册：命令》。
- 如果列具有 IDENTITY 属性，则输入一个唯一、连续的值。
- 如果创建表时没有为列指定 NULL 并且不存在缺省值，则 Adaptive Server 拒绝该行并显示错误消息。

表 8-4 显示了在这些情况下会看到的情况：

表 8-4：没有值的列

缺省值存在对于列或数据类型	列已定义为 NOT NULL	列已定义为允许 NULL	列是 IDENTITY
有	缺省值	缺省值	下一个顺序值
无	错误消息	NULL	下一个顺序值

可以使用 sp_help 来显示有关指定表、缺省值或系统表 sysobjects 中列出的任何其它对象的报告。要查看缺省值定义，请使用 sp_helptext。

将列的值更改为 NULL

若要将列值设置为 NULL，请使用 update 语句：

```
set column_name = {expression | null}
[, column_name = {expression | null}]...
```

例如，要查找所有 title_id 为 TC3218 的行并使用 NULL 替换 advance，请使用以下命令：

```
update titles
set advance = null
where title_id = "TC3218"
```

Adaptive Server 为 IDENTITY 列生成的值

在将行插入包含 IDENTITY 列的表中时， Adaptive Server 会自动生成列值。不要在列列表中包括 IDENTITY 列的名称，也不要在值列表中包括 IDENTITY 列的值。

此 insert 语句将新行添加到 sales_daily 表。列列表不包含 IDENTITY 列 row_id：

```
insert sales_daily (stor_id)
values ("7896")
```

注释　在此示例中，也可以省略列名 stor_id。服务器可以识别 IDENTITY 列并使用 insert 插入下一个标识值，而无需用户输入列名。例如，下表包含三个列，但 insert 语句为两个列指定值，而没有指定列名：

```
create table idtext (a int, b numeric identity, c
char(1))
-------------------
(1 row affected)

insert idtext values(98,"z")
-------------------
(1 row affected

insert idtest values (99, "v"))
--------------------

(1 row affected)
select * from idtest
---------------------
98     1        z
99     2         v

(2 rows affected)
```

下列语句显示添加到 sales_daily 中的行。 Adaptive Server 自动为 row_id 生成下一个顺序值 2：

```
select * from sales_daily
where stor_id = "7896"

sale_id      stor_id
-------      -------
      1      7896

(1 row affected)
```

将数据显式插入 IDENTITY 列

有时，可能想要将特定值插入 IDENTITY 列，而不接受服务器生成的值。例如，可能想要插入表的第一行的 IDENTITY 值为 101，而不是 1，或者可能需要重新插入误删除的行。

表的所有者可将值显式插入 IDENTITY 列中。如果数据库所有者和系统管理员获得了表所有者授予的显式权限，或通过 setuser 命令作为表所有者操作，他们能将值显式插入 IDENTITY 列。

插入数据前，将表的 identity_insert 选项设置为 “on”。在会话中，一次只能针对数据库中的一个表将 identity_insert 设置为 “on”。

以下示例指定 IDENTITY 列的源值为 101：

```
set identity_insert sales_daily on
insert sales_daily (syb_identity, stor_id)
values (101, "1349")
```

insert 语句列出了每个列，其中包含指定了值的 IDENTITY 列。当 identity_insert 选项设置为 “on”时，表的每条 insert 语句都必须指定显式列列表。值列表必须指定 IDENTITY 列值，因为 IDENTITY 列不允许空值。

将 identity_insert 设置为 “off”后，能自动插入 IDENTITY 列值，不必与以前一样指定 IDENTITY 列。后续插入根据在将 identity_insert 设置为 “on”后指定的显式值，使用 IDENTITY 值。例如，如果为 IDENTITY 列指定了 101，则后续的插入应为 102、 103，等等。

注释 Adaptive Server 不强制插入值具有唯一性。可在列的声明精度允许的范围内指定任何正整数。为确保只接受唯一列值，可在插入任何行前，在 IDENTITY 列上创建唯一索引。

用 @@*identity* 检索 IDENTITY 列值

使用 @@*identity* 全局变量检索插入 IDENTITY 列的最后一个值。每次 insert 或 select into 尝试向表中插入行时， @@*identity* 的值就会更改。如果 insert 或 select into 语句失败，或如果包含它的事务被回退，则 @@*identity* 不还原为其以前的值。如果语句影响没有 IDENTITY 列的表，则将 @@*identity* 设置为 0。

如果语句插入多行，则 @@*identity* 显示插入 IDENTITY 列的最后一个值。

存储过程或触发器内的 @@*identity* 值不会影响该存储过程或触发器以外的值。例如：

```
select @@identity

---------------------------------------
                                     101

create procedure reset_id as
    set identity_insert sales_daily on
    insert into sales_daily (syb_identity, stor_id)
         values (102, "1349")
    select @@identity
select @@identity
execute reset_id

---------------------------------------
                                     102

select @@identity

---------------------------------------
                                     101
```

保留 IDENTITY 列值块

identity grab size 配置参数允许每个 Adaptive Server 进程保留一个 IDENTITY 列值块，以便插入到具有 IDENTITY 列的表中。此配置参数减少了在插入隐式标识值时， Adaptive Server 引擎必须持有内部同步结构的次数。例如，将保留值的数量设置为 20：

```
sp_configure "identity grab size", 20
```

当用户执行插入包含 IDENTITY 列的表的操作时， Adaptive Server 为该用户保留一个 20 个 IDENTITY 列值的块。因此，在当前会话期间，用户插入到表中的下一 20 行具有顺序的 IDENTITY 列值。如果在第一个用户进行插入时，第二个用户在同一表中插入行， Adaptive Server 将为第二个用户保留下一个包含 20 个 IDENTITY 列值的块。

例如，假定已创建以下包含 IDENTITY 列的表，并且将 identity grab size 设置为 10：

```
create table my_titles
(title_id   numeric(5,0)    identity,
title       varchar(30)     not null)
```

用户 1 将这些行插入到 my_titles 表中：

```
insert my_titles (title)
values ("The Trauma of the Inner Child")insert my_titles
(title)

values ("A Farewell to Angst")
insert my_titles (title)
values ("Life Without Anger")
```

Adaptive Server 允许用户 1 有一个含 10 个顺序 IDENTITY 值的块，例如， title_id 编号 1–10。

当用户 1 将行插入 my_titles 时，用户 2 开始将行插入 my_titles。Adaptive Server 授予用户 2 保留 IDENTITY 值的下一可用块，也就是说，值 11–20。

如果用户 1 只输入三个 “title”，然后注销 Adaptive Server，那么剩余的七个保留的 IDENTITY 值将丢失。结果是表的 IDENTITY 值中出现一个间隔。要避免 identity grab size 设置太高，因为这可能导致 IDENTITY 列编号中出现间隔。

达到 IDENTITY 列的最大值

可以插入 IDENTITY 列的最大值是 10 精度 - 1。如果不指定 IDENTITY 列的精度， Adaptive Server 将使用 numeric 列的缺省精度 （18 位）。

一旦 IDENTITY 列达到最大值，插入语句将返回错误消息并终止当前事务。发生这种情况时，使用下列方法之一可解决问题。

修改 IDENTITY 列的最大值

可以通过 alter table 命令中的修改操作更改任何 IDENTITY 列的最大值。

```
alter table my_titles
modify title_id, numeric (10,0)
```

此操作对表执行数据复制操作并重建所有表索引。

创建具有更大精度的新表

如果表包含用于参照完整性的 IDENTITY 列，则必须保留 IDENTITY 列值的当前编号。

1　使用 create table 创建新表，该表除了 IDENTITY 列精度值更大外，与旧表相同。

2　使用 insert into 将数据从旧表复制到新表。

用 bcp 对表的 IDENTITY 列重新编号

如果表不包含用于参照完整性的 IDENTITY 列，并且如果编号序列中有间隔，则可以对 IDENTITY 列进行重新编号以消除间隔，从而提供更多的插入空间。

若要按顺序对 IDENTITY 列值重新编号并删除间隔，请使用 bcp 实用程序：

1　在操作系统命令行上，使用 bcp 拷出数据。例如：

```
bcp pubs2..mytitles out my_titles_file -N -c
```

-N 指示 bcp 不将 IDENTITY 列值从表中复制到主机文件。 -c 指示 bcp 使用字符模式。

2　在 Adaptive Server 中，创建一个与旧表相同的新表。

3　从操作系统命令行中，使用 bcp 将数据复制到新表中：

```
bcp pubs2..mynewtitles in my_titles_file -N -c
```

-N 指示 bcp 当从主机文件装载数据时，让 Adaptive Server 分配 IDENTITY 列值。 -c 指示 bcp 使用字符模式。

4　在 Adaptive Server 中，删除旧表，并使用 sp_rename 将新表的名称更改为旧表的名称。

如果 IDENTITY 列是连接的主键，可能需要更新其它表中的外键。

缺省情况下，在将数据批量复制到有 IDENTITY 列的表时， bcp 为每行分配一个临时的 IDENTITY 列值 0。在将每行插入表时，服务器从下一可用的值开始，为其分配一个唯一的、连续的 IDENTITY 列值。若要为每行输入显式 IDENTITY 列值，请指定 -E 标志。请参见 《实用程序指南》。

使用 *select* 添加新行

要从一个或多个其它表中将值提取到某个表中，请在 insert 语句中使用 select 子句。 select 子句能将值插入某行的某些或所有列。

如果要从现有表中提取一些值，仅为某些列插入值可能会非常方便。然后，可以使用 update 将值添加到其它列。

在将值插入表的某些但不是全部列前，要确保存在缺省值或不插入值的列已经指定为 NULL。否则， Adaptive Server 返回错误消息。

当从一个表向另一个表插入行时，两个表必须具有兼容的结构；也就是说，匹配列的数据类型必须相同或者是 Adaptive Server 可在两者间自动转换的数据类型。

注释 如果要插入的任何数据是空值，则不能将数据从允许空值的表插入不允许空值的表。

如果列在其 create table 语句中具有相同的顺序，则不需要在两个表中指定列名称。假定有一个名为 newauthors 的表，它包含一些作者信息行，这些行具有与 authors 中信息相同的格式。将 newauthors 中的所有行添加到 authors 中：

```
insert authors
select *
from newauthors
```

若要根据某表中的数据将行插入另一个表，那么在各自的 create table 语句中不必按相同的顺序列出两个表中的列。可以使用 insert 或 select 语句排序列，以使它们相互匹配。

例如，假定用于创建 authors 表的 create table 语句包含顺序依次为 au_id、 au_fname、 au_lname 和 address 的列；而 newauthors 包含 au_id、address、 au_lname 和 au_fname 列。在 insert 语句中必须使列顺序匹配。可以采用下列两种方法之一实现此目的：

```
insert authors (au_id, address, au_lname, au_fname)
select * from newauthors
```

或者

```
insert authors
select au_id, au_fname, au_lname, address
    from newauthors
```

如果两个表中列的顺序不能匹配，则 Adaptive Server 不能完成或不能正确完成 insert 操作，并会将数据放入错误的列。例如，可能在 au_lname 列中得到 “address” 数据。

使用计算列

可以在 insert 语句内的 select 语句中使用计算列。例如，假定名为 tmp 的表包含一些 titles 表的新行，其中包含一些过期的数据 – price 数字需要加倍。增加价格并在 titles 中插入 tmp 行的语句如下所示：

```
insert titles
select title_id, title, type, pub_id, price*2,
    advance, total_sales, notes, pubdate, contract
from tmp
```

对列执行计算时，不能使用 select * 语法。每列都必须在选择列表中单独命名。

插入数据到某些列

可以使用 select 语句将数据添加到行的某些列，但不是所有列，如同使用 values 子句一样。只需在 insert 子句中，指定数据要添加到的列。

例如，authors 表中的某些 author 没有 title，因此在 titleauthor 表中没有相应的条目。若要将它们的 au_id 编号从 authors 表中取出来，并将其插入 titleauthor 表中作为占位符，请尝试以下语句：

```
insert titleauthor (au_id)
select au_id
    from authors
    where au_id not in
    (select au_id from titleauthor)
```

此语句不合法，因为 title_id 列需要一个值。不允许空值，并且没有指定缺省值。可以通过使用一个常量为 titles_id 输入伪值 “xx1111”，如下所示：

```
insert titleauthor (au_id, title_id)
select au_id, "xx1111"
    from authors
    where au_id not in
    (select au_id from titleauthor)
```

titleauthor 表现在包含四个新行，这些新行在 au_id 列中具有条目，在 title_id 列中具有伪条目，其它两列为空值。

从相同表中插入数据

可以基于相同表中的其它数据将数据插入表中。实质上，这意味着复制一行的全部或部分。

例如，能将新行插入 publishers 表中，该表基于同一表中已经存在行的值。请确保遵循有关 pub_id 列的规则：

```
insert publishers
select "9999", "test", city, state
    from publishers
    where pub_name = "New Age Books"

 (1 row affected)

select * from publishers

 pub_id  pub_name              city        state
------- --------------------- -------     ------
0736    New Age Books         Boston      MA
0877    Binnet & Hardley      Washington  DC
1389    Algodata Infosystems  Berkeley    CA
9999    test                  Boston      MA

(4 rows affected)
```

该示例插入两个常量（“9999”和“test”）以及满足查询的行的 city 和 state 列的值。

更改现有数据

可以使用 update 命令来更改表中的单行、行组或所有行。如同在所有数据修改语句中，一次只可更改一个表中的数据。

update 指定了要更改的一行或多行以及新数据。新数据可以是指定的常量或表达式，或者是从其它表中拖出的数据。

如果 update 语句违反完整性约束，则不发生更新，并产生错误消息。例如，如果更新影响了表的 IDENTITY 列，或某个要添加的值的数据类型错误，或更新违反了为所涉及的列或数据类型之一定义的规则，则会取消该更新。

Adaptive Server 允许多次执行更新单个行的 update 命令。当然， update 语句的处理方法决定了单个语句进行的更新不累积。也就是说，如果 update 语句两次修改同一行，则第二次更新将不以第一次更新后的新值为基础，而是以初始值为基础。结果是不可预知的，因为它们取决于处理的顺序。

有关对更新视图的限制，请参见第 12 章 “视图：限制访问数据”。

注释　将记录 update 命令。如果更改较大的 text、 unitext 或 image 数据块，请尝试使用 writetext 命令，将不会记录该命令。同样，每个 update 语句大约限制为 125K。请参见第 322 页的 “更改 text、unitext 和 image 数据”中对 writetext 的论述。

请参见 《参考手册：命令》。

将 *set* 子句用于 *update*

set 子句指定了列和更改的值。 where 子句确定要更新哪一行或哪些行。如果没有 where 子句，则*所有*行的指定列都将更新为 set 子句中给定的值。

注释　在尝试本节中的示例前，应确保知道如何重新安装 pubs2 数据库。有关安装 pubs2 数据库的指导，请参见所用平台的安装和配置指南。

例如，如果 publishers 表中的所有 publishing house 将其 head office 搬到 Atlanta 和 Georgia，则可以使用以下方法来更新表：

```
update publishers
set city = "Atlanta", state = "GA"
```

同样，可以使用以下方法将所有出版社的名称更改为 NULL：

```
update publishers
set pub_name = null
```

也可以在更新中使用计算列值。若要将 titles 表中的所有价格加倍，请使用：

```
update titles
set price = price * 2
```

由于没有 where 子句，因此对价格的更改适用于表中所有行。

在 *set* 子句中对变量赋值

可在 update 语句的 set 子句中对变量赋值，也可以同样方式，在 select 语句中对它们赋值。将变量用于 update 减少了锁争用，并减少了额外的 select 语句与 update 一起使用时可出现的 CPU 消耗。

以下示例使用声明变量来更新 titles 表：

```
declare @price money
select @price = 0
update titles
    set total_sales = total_sales + 1,
    @price = price
    where title_id = "BU1032"
select @price, total_sales
    from titles
    where title_id = "BU1032"

                          total_sales
 ----------------------- -----------
                   19.99        4096

(1 row affected)
```

请参见 《参考手册：命令》。有关声明变量的详细信息，请参见 第 476 页的 “局部变量”。

将 *where* 子句用于 *update*

where 子句指定要更新哪些行。以一个不太可能发生的事情为例，将 Northern California 改名为 Pacifica （简称 PC），并且 Oakland 的人们投票将他们的城市改名为 College town，则可以使用以下方法为所有地址现已过时的前 Oakland 居民更新 authors 表：

```
update authors
set state = "PC", city = "College Town"
where state = "CA" and city = "Oakland"
```

您必须编写另外一个语句来为 Northern California 州其它城市居民更改州名。

将 *from* 子句用于 *update*

使用 from 子句将数据从一个或多个表取进要更新的表。

例如，本章的前面给出了一个示例，即在 titleauthor 表中为没有 title 的 author 插入一些新行，填充 au_id 列，并对其它列使用伪值和空值。Dirk Stringer 是这些 author 之一，他编写了 *The Psychology of Computer Cooking* 一书，在 titles 表中给这本书分配一个 title 标识号。在 titleauthor 表中，可以通过为他添加一个 title 标识号来修改他的行：

```
update titleauthor
set title_id = titles.title_id
from titleauthor, titles, authors
    where titles.title =
    "The Psychology of Computer Cooking"
    and authors.au_id = titleauthor.au_id
    and au_lname = "Stringer"
```

没有使用 au_id 连接的 update 将更改 titleauthor 表中的所有 title_id，因此它们与 *The Psychology of Computer Cooking* 的标识号相同。如果两个表在结构上相同，只是一个表包含 NULL 字段和一些空值，而另一个表包含 NOT NULL 字段，则不能使用 select 将 NULL 表中的数据插入到 NOT NULL 表中。也就是说，不允许 NULL 的字段不能通过从允许 NULL 的字段中进行选择而更新 （如果有任何数据为 NULL）。

作为 update 语句中的 from 子句的替代，还可以使用符合 ANSI 的子查询。

使用连接执行更新

以下示例连接 *titles* 和 *publishers* 表中的列，并将在 California 出版的所有书籍的价格加倍：

```
update titles
  set price = price * 2
  from titles, publishers
  where titles.pub_id = publishers.pub_id
  and publishers.state = "CA"
```

更新 IDENTITY 列

必要时，可以使用用表名限定的 syb_identity 关键字更新 IDENTITY 列。例如，此 update 语句查找 IDENTITY 列等于 1 的行，并将店铺的名称更改为 “Barney’s”。

```
update stores_cal
set stor_name = "Barney's"
where syb_identity = 1
```

更改 *text*、 *unitext* 和 *image* 数据

如果不想在数据库事务日志中存储长文本值，请使用 writetext 来更改 text、 unitext 或 image 值。不要使用 update 命令，它也可用于 text、 unitext 或 image 列，因为会始终记录 update 命令。缺省模式下，writetext 命令不被记录。

注释 若要在其缺省的不记入日志的状态下使用 writetext，系统管理员必须使用 sp_dboption 将 select into/bulkcopy/pllsort 设置为 on。这将允许插入不记入日志的数据。使用 writetext 后，必须转储数据库。对数据库进行不记入日志的更改后，不能使用 dump transaction。

---

writetext 命令完全覆盖它所影响的列中的所有数据。列必须已经包含有效的文本指针。

可以使用 textvalid() 函数来检查有效指针：

```
select textvalid("blurbs.copy", textptr(copy))
from blurbs
```

有两种创建文本指针的方法：

- 使用 insert 将实际数据插入到 text、 unitext 或 image 列中
- 用数据或 NULL update 列

“已初始化”的 text 列使用 2K 存储空间，即使只存储两个字。当用 insert 将显式或隐式空值放入 text 列中时， Adaptive Server 通过不初始化文本列来节省空间。下列代码片段插入一个有空文本指针的值，检查是否存在一个文本指针，然后更新 blurbs 表。在文本中嵌入说明性注释：

```
/* Insert a value with a text pointer. This could
** be done in a separate batch session. */
insert blurbs (au_id) values ("267-41-2394")
/* Check for a valid pointer in an existing row.
** Use textvalid in a conditional clause; if no
** valid text pointer exists, update 'copy' to null
** to initialize the pointer. */
if (select textvalid("blurbs.copy", textptr(copy))
    from blurbs
    where au_id = "267-41-2394") = 0
begin
    update blurbs
       set copy = NULL
       where au_id = "267-41-2394"
end
/*
** use writetext to insert the text into the
** column. The next statements put the text
** into the local variable @val, then writetext
** places the new text string into the row
** pointed to by @val. */
declare @val varbinary(16)
select @val = textptr(copy)
    from blurbs
    where au_id = "267-41-2394"
writetext blurbs.copy @val
    "This book is a must for true data junkies."
```

有关在本示例中使用的批处理文件和控制流语言的详细信息，请参见第 15 章 “使用批处理和控制流语言”。

增量传输数据

transfer 命令允许增量传输数据，并且在需要时，可以将数据增量传输到不同产品。在 Adaptive Server 15.5 之前的版本中，只能将所有表从一个 Adaptive Server 传输到另一个。

注释 在您购买、安装和注册内存数据库许可证或在您安装 Risk Analytics Platform (RAP) 产品后， Adaptive Server 即会启用数据传输功能。

增量数据传输：

- 允许从标记为增量传输的 Adaptive Server 表中导出数据，其中只包括自以前的传输之后更改的数据。
- 允许读取表数据，而无需获取常规锁，无需保证任何行检索顺序，并且不会干扰其它正在进行的读取或更新操作。
- 允许将所选行写入输出文件 （可以是命名管道），其格式为定义的接收方：IQ (Sybase IQ)、 ASE (Adaptive Server Enterprise)、批量复制 (bcp) 或字符编码输出。将不加密传输所有选定行，并且缺省情况下，会在传输行中的任何加密列之前先对其解密。要写入的文件必须对运行 Adaptive Server 的计算机可见 （文件可以是 Adaptive Server 可作为本地文件打开的 NFS 文件）。
- 保留合格表的传输历史记录，并且允许在不需要时删除传输历史记录。
- 从未声明符合增量传输条件的表导出数据时存在某些限制。
- 从指示的表传输所有行。目前无法选择某些列，可选择表中的分区或传输 SQL 查询的结果。

将表标记为增量传输

必须将表标记为可以参与增量传输。可以标记除系统表和工作表以外的任何表。可以在创建表时或以后使用 alter table 指定合格性。还可以使用 alter table 删除表的合格性。

在合格表中：

- 如果某行自上一次传输以后发生了更改，以及如果更改现有行或插入新行的任何事务在传输开始之前已提交，则会传输该行。

  这需要额外的存储空间来存储每行，这由行中隐藏的 8 字节列实现。

- 保留用于传输每个表的附加信息。此类信息包括传输的行集和行数的标识信息、传输的开始和结束时间、传输的数据格式以及目标文件的完整路径。

删除表的合格性会删除为支持增量传输而添加的任何行更改，并删除该表的任何已保存的传输历史记录。

从目标文件传输表

使用 transfer table 命令可将数据从外部文件中包含的表装载到 Adaptive Server 中。请参见 《参考手册：命令》。

注释　Adaptive Server 15.5 版必须使用其内部格式来导入表。

要装载的表不需要唯一主索引，除非您知道要装载的数据已基于表中已有的数据发生更改 （可以装载新数据，没有任何限制）。但是，当行复制表中已有的数据，并且您不希望复制数据时，装载数据将成为问题。为避免出现此问题，唯一主索引允许 Adaptive Server 查找并删除新行替换的旧行。

要装载的表必须将唯一索引作为其主键 （所有页锁定表的聚簇索引或仅数据锁定表的位置索引）。唯一索引允许 transfer 检测重复键插入尝试，并将内部 insert 命令转换为 update 命令。没有该索引， Adaptive Server 将无法检测重复的主键。插入更新行会导致：

- 在表具有任何其它唯一索引并且要插入的行重复该索引中的键时，部分或所有传输操作失败。
- 插入成功，但表包含两个或多个使用此主键的行。

以下示例将 pubs2.titles 表从位于 */sybase/data* 中的外部 *titles.tmp* 文件传输到 Adaptive Server 中：

```
transfer table titles from '/sybase/data/titles.tmp' for ase
```

不能对 transfer table...from 使用用于 transfer table.. .to 的所有参数。不适合从文件装载数据的参数会产生错误，并且传输命令会停止。 Adaptive Server 15.5 版包括 from 参数的保留以备后用的参数，但如果您在语法中包括这些参数， transfer 会忽略它们。 from 参数的参数是：

- column_order=*option* （不适用于使用 for ase 进行的装载；保留以备后用）
- column_separator=*string* （不适用于使用 for ase 进行的装载；保留以备后用）
- encryption={true | false}（不适用于使用 for ase 进行的装载；保留以备后用）
- progress=*nnn*
- row_separator=*string* （不适用于使用 for ase 进行的装载；保留以备后用）

将 Adaptive Server 数据类型转换为 IQ

表 8-5 显示 Adaptive Server 数据类型在 Sybase IQ 中的表示形式。在传输数据类型时， Adaptive Server 会进行任何必要的转换以将其数据转换为指示的 IQ 格式

表 8-5：Adaptive Server 到 IQ 数据类型的转换

Adaptive Server		IQ	
数据类型	大小 （字节）	数据类型	大小 （字节）
bigint unsigned bigint	8	bigint unsigned bigint	8
int unsigned int	4	int unsigned int	4
smallint	2	smallint	2
unsigned smallint	2	int	2
tinyint	1	tinyint	1
numeric(P,S) decimal(P,S)	2 – 17	numeric(P,S) decimal(P,S)	2 – 26
double precision	8	double	8
real	4	real	4
float(P)	4, 8	float(P)	4, 8
money	8	money IQ 将此存储为 numeric(19.4)	16
smallmoney	4	smallmoney IQ 将此存储为 numeric(10,4)	8
bigdatetime datetime	8	datetime	8
smalldatetime	4	smalldatetime	8
date	4	date	4
time	4	time	8
bigtime	8	time	8
char(N)	1 – 16296	char(N)	1 – 16296
char(N) (null)	1 – 16296	char(N) (null)	1 – 16296
varchar(N) (null)	1 – 16296	varchar(N) (null)	1 – 16296
unichar(N)	1– 8148	binary(N*2)	1 – 16296
unichar(N) null univarchar(N) (null)	1– 8148	varbinary(N*2) (null)	1 – 16296
binary(N)	1 – 16296	binary(N)	1 – 16296
varbinary(N)	1 – 16296	varbinary(N)	1 – 16296
binary(N) null varbinary(N) (null)	1 – 16296	varbinary(N) (null)	1 – 16296
bit	1	bit	1
timestamp	8	varbinary(8) null	8

在将 Adaptive Server 数据类型转换为 IQ 数据类型时应考虑以下情况：

- 在 IQ 和 Adaptive Server 上定义相同的精度和标度。
- float 的存储大小是 4 或 8 字节，具体取决于精度。如果没有为精度提供值， Adaptive Server 将 float 存储为 double precision，但 IQ 将它存储为 real。Adaptive Server 不会将浮点数据转换为其它格式以传输到 IQ。如果必须使用近似数值类型，请将它们指定为 double 或 real，而非 float。
- Adaptive Server 中 char、 unichar 或 binary 数据类型的列的最大长度取决于您的安装的页大小。表 8-5 中指定的最大大小是 16k 页中可能的最大列。
- Adaptive Server 中每个字符通常需要两个字节来存储 Unicode 字符。因为 IQ 不包括 Unicode 数据类型，所以 Adaptive Server 将 unichar(N) 作为 binary(N X 2) 传输到 IQ。但 Adaptive Server 不转换 Unicode 字符：Adaptive Server 使用 NULL(0x00) 填补 Unicode 字符串以将它们传输到 IQ。
- IQ 没有本机 Unicode 数据类型。 Adaptive Server 将 Unicode 字符串作为每个 Unicode 字符长两个字节的二进制数据传输到 IQ。例如，Adaptive Server 中的 unichar(40) 转换为 IQ 中的 binary(80)。传输后，IQ 无法将 Unicode 数据显示为字符串。

存储传输信息

此节介绍 spt_TableTransfer 和 monTableTransfer 表。

表传输结果存储在 spt_TableTransfer 表中并从其中检索。从为此命令指定的表中检索的成功传输结果将用作后续传输的缺省值。例如，如果发出以下命令 （包括行和列分隔符）：

```
transfer table mytable for csv
```

下次传输表 mytable 时， transfer 命令会缺省使用 for csv 和相同的行和列分隔符。

每个数据库具有自己的 spt_TableTransfer 版本。表只存储同一数据库中标记为增量传输的表的表传输历史记录。

max transfer history 配置参数控制 Adaptive Server 在每个数据库的 spt_TableTransfer 表中保留的传输历史记录条目。请参见 《系统管理指南：卷 1》中的第 5 章 “设置配置参数”。

数据库所有者使用 sp_setup_table_transfer 创建 spt_TableTransfer 表。sp_setup_table_transfer 不采用任何参数，并在当前数据库中运行。如果 spt_TableTransfer 不存在， sp_setup_table_transfer 会创建它。

spt_TableTransfer 表存储有关成功和失败传输的历史信息。它不存储有关当前正在进行的传输的信息。

spt_TableTransfer 是用户表，不是系统表。它不是在您创建 Adaptive Server 时创建的，但如果您没有使用 sp_setup_transfer_table 手动创建它， Adaptive Server 会自动在具有符合传输条件的表的任何数据库中创建它（手动创建它可以避免 Adaptive Server 自动创建表时可能出现的意外错误）。

sp_help 将增量传输作为表属性进行报告。

spt_TableTransfer 中的列包括：

列	数据类型	说明
end_code	unsigned smallint not null	传输的结束状态。 0 – 成功 错误代码 – 失败
id	int not null	已传输表的对象 ID。
ts_floor	bigint not null	开始事务时间戳。
ts_ceiling	bigint not null	在其后未提交行并因而未传输的事务时间戳。
time_begin	datetime not null	• 传输的开始日期和时间 • 如果 transfer 在实现前失败，则为 Adaptive Server 开始设置命令的时间。否则为命令将第一个数据发送到输出文件的时间。
time_end	datetime not null	• 传输的结束日期和时间 • 如果 transfer 命令失败，则为失败时间。否则为命令完成数据发送并关闭文件的时间
row_count	bigint not null	传输的行数。
byte_count	bigint not null	已写入的字节数。
sequence_id	int not null	一个跟踪此传输的数字，它对于表的每次传输具有唯一性。
col_order	tinyint not null	一个代表输出的列顺序的数字： • 1 — id • 2 — offset • 3 — name • 4 — name_utf8

列	数据类型	说明
output_to	tinyint not null	一个代表输出格式的数字： • 1 — ase • 2 — bcp • 3 — csv • 4 — iq
tracking_id	int null	一个客户提供的可选跟踪 ID。如果不与 tracking_id = nnn 一起使用，此列将为空。
pathname	varchar (512) null	输出文件名
row_sep	varchar(64) null	用于 for csv 的行分隔符字符串。
col_sep	varchar(64) null	用于 for csv 的列分隔符。

monTableTransfer 监控表提供：

- Adaptive Server 当前在内存中保存其信息的表的历史传输信息。这适用于 Adaptive Server 最近重新启动后访问的任何表，除非没有为 Adaptive Server 配置足够大的内存来保存所有当前表信息
- 有关当前正在进行的传输和 Adaptive Server 在内存中保存其信息的表的已完成传输的信息。其中包括有关以下表的信息：
  - 标记为增量传输的表
  - 重新启动 Adaptive Server 后至少一个传输中涉及的表。
  - 不将其说明用于其它表的表 （monTableTransfer 不在每个数据库中搜索以前传输的每个表； monTableTransfer 将自己限制为自最近传输之后继续跟踪的一组活动表）。

请参见 《参考手册：表》中的 monTableTransfer。

例外和错误

如果您试图执行以下操作， Adaptive Server 会生成错误消息：

- 传输不存在的表。
- 传输不是表的对象。
- 传输不属于自己的表，并且您没有被授予传输权限和 sa_role 特权。
- 在传输期间解密包含加密列的表中的列，但没有解密这些列的特定权限。

- 在表包含 text 或 image 列时传输 for iq
- 传输 for ase，但指定的列顺序不是 offset。
- 传输 for bcp，但指定的列顺序不是 id。
- 发出指定系统目录的 alter table...set transfer table on 命令。

传输失败的其它原因是：

- 请求的文件无法关闭。
- 无法传输包含行外列 （存储在行外的 text、 unitext、 image 和 Java 列）的表。
- 未能打开文件。确保目录存在且 Adaptive Server 在目录中具有 write 权限。
- Adaptive Server 在传输中无法获取足够的内存用于每个表的已保存数据。如果发生这种情况，需增加 Adaptive Server 可用的内存量。

增量数据传输的示例会话

此教程介绍如何使用 transfer 命令将数据传输到外部文件，更改表中的数据，然后再次使用 transfer 命令重新填充此外部文件中的表，并演示 transfer 如何将数据附加到文件中，而不覆盖该文件。

注释　为便于说明，此示例将数据传输出去，然后再传输到同一表中。在典型用户方案中，会将一个表中的数据传输出去，然后再传输到另一个表中。

1　运行 sp_setup_table_transfer 以创建 spt_TableTransfer 表，该表存储传输历史记录：

```
sp_setup_table_transfer
```

2　配置 max transfer history。缺省值为 10，这表示 Adaptive Server 为每个标记为增量传输的表保留 10 个成功传输和 10 个不成功传输。此示例将 max transfer history 的值从 10 更改为 5：

```
sp_configure 'max transfer history', 5

Parameter Name                Default     Memory Used
Config Value    Run Value              Unit
Type                      Instance Name
-------------------------     -------     -----------
--------------   --------------------   -------
```

```
---------------          ---------------
max transfer history      10              0
5                5                       bytes
dynamic                  NULL
```

3 创建 transfer_example 表，该表已启用 transfer 属性并使用数据行锁：

```
create table transfer_example (
f1 int,
f2 varchar(30),
f3 bigdatetime,
primary key (f1)
) lock datarows
   with transfer table on
```

4 运行以下脚本来使用示例数据填充 transfer_example 表：

```
set nocount on
declare @i int, @vc varchar(1024), @bdt bigdatetime
select @i = 1
while @i <= 10
begin
   select @vc = replicate(char(64 + @i), 3 * @i)
   select @bdt = current_bigdatetime()
   insert into transfer_example values ( @i, @vc,
@bdt )
   select @i = @i + 1
end
set nocount off
```

该脚本生成以下数据：

```
select * from transfer_example
order by f1
f1       f2                              f3
-------- ------------------------------- --------------------------
   1      AAA                             Jul 17 2009  4:40:14.465789PM
   2      BBBBBB                          Jul 17 2009  4:40:14.488003PM
   3      CCCCCCCCC                       Jul 17 2009  4:40:14.511749PM
   4      DDDDDDDDDDDD                    Jul 17 2009  4:40:14.536653PM
   5      EEEEEEEEEEEEEE                  Jul 17 2009  4:40:14.559480PM
   6      FFFFFFFFFFFFFFFFF               Jul 17 2009  4:40:14.583400PM
   7      GGGGGGGGGGGGGGGGGGGG            Jul 17 2009  4:40:14.607196PM
   8      HHHHHHHHHHHHHHHHHHHHHHHH        Jul 17 2009 4:40:14.632152PM
   9      IIIIIIIIIIIIIIIIIIIIIIIII    Jul 17 2009  4:40:14.655184PM
  10      JJJJJJJJJJJJJJJJJJJJJJJJJJJJ  Jul 17 2009  4:40:14.678938PM
```

5　使用 for ase 格式将 transfer_example 数据传输到外部文件。

```
transfer table transfer_example
to 'transfer_example-data.ase'
for ase

(10 rows affected)
```

数据传输过程将在 spt_TableTransfer 中创建以下历史记录：

```
select id, sequence_id, end_code, ts_floor, ts_ceiling, row_count
from spt_TableTransfer
where id = object_id('transfer_example')
id        sequence_id      end_code     ts_floor       ts_ceiling
row_count
---------   -----------       --------      -------------   -----------
--------
592002109  1                0            0              5309
10
```

6　禁用 transfer_example 的 transfer 属性以演示接收表无需启用 transfer 属性便可接收增量数据 （数据库必须启用 select into 才能运行 alter table）。

```
alter table transfer_example
set transfer table off
```

在 alter table 命令运行后， spt_TableTransfer 为空：

```
select id, sequence_id, end_code, ts_floor, ts_ceiling, row_count
from spt_TableTransfer
where id = object_id('transfer_example')
id        sequence_id      end_code     ts_floor       ts_ceiling
row_count
---------   -----------       --------      -------------   -----------
--------

(0 rows affected
```

7　更新 transfer_example 以将其字符数据设置为 no data 并指定其 bigdatetime 列中的日期和时间，以便可以检验表是否包含原始数据：

```
update transfer_example
set f2 = 'no data',
f3 = 'Jan 1, 1900 12:00:00.000001AM'

(10 rows affected)
```

update 后， transfer_example 包含以下数据。

```
select * from transfer_example
```

```
order by f1
f1          f2                              f3
----------- ------------------------------- --------------------------
1           no data                         Jan  1 1900 12:00:00.000001AM
2           no data                         Jan  1 1900 12:00:00.000001AM
3           no data                         Jan  1 1900 12:00:00.000001AM
4           no data                         Jan  1 1900 12:00:00.000001AM
5           no data                         Jan  1 1900 12:00:00.000001AM
6           no data                         Jan  1 1900 12:00:00.000001AM
7           no data                         Jan  1 1900 12:00:00.000001AM
8           no data                         Jan  1 1900 12:00:00.000001AM
9           no data                         Jan  1 1900 12:00:00.000001AM
10          no data                         Jan  1 1900 12:00:00.000001AM

(10 rows affected)
```

8 将示例数据从外部文件传输到 transfer_example 中。虽然 transfer_example 不再标记为增量传输，您仍可以将数据传输到表中。因为它具有唯一主索引，所以传入行会替换现有数据，且不会产生重复键错误：

```
transfer table transfer_example
from 'transfer_example-data.ase'
for ase

(10 rows affected)
```

9 选择 transfer_example 中的所有数据，以检验传入数据是否替换了已更改数据。 transfer 替换了 transfer_example.f2 和 transfer_example.f3 表的内容，这两个表包含原来为它们创建的数据，且存储在 *transfer_example-data.ase* 输出文件中。

```
select * from transfer_example
order by f1
f1      f2                              f3
------- ------------------------------- ----------------------------
   1    AAA                             Jul 17 2009  4:40:14.465789PM
   2    BBBBBB                          Jul 17 2009  4:40:14.488003PM
   3    CCCCCCCCC                       Jul 17 2009  4:40:14.511749PM
   4    DDDDDDDDDDD                     Jul 17 2009  4:40:14.536653PM
   5    EEEEEEEEEEEEEE                  Jul 17 2009  4:40:14.559480PM
   6    FFFFFFFFFFFFFFFFF               Jul 17 2009  4:40:14.583400PM
   7    GGGGGGGGGGGGGGGGGGGG            Jul 17 2009  4:40:14.607196PM
   8    HHHHHHHHHHHHHHHHHHHHHHHH        Jul 17 2009  4:40:14.632152PM
   9    IIIIIIIIIIIIIIIIIIIIIIIII       Jul 17 2009  4:40:14.655184PM
  10    JJJJJJJJJJJJJJJJJJJJJJJJJJJJ    Jul 17 2009  4:40:14.678938PM
```

10 为 transfer_example 重新启用 transfer，以便后续传输缺省使用以前的参数。

```
alter table transfer_example
set transfer table on
(10 rows affected)
```

使用新行替换数据

如果更改了一些行的键值，则在使用增量传输时，下一个表 transfer 会将已更改的键数据行视为新数据，并且只替换其键未更改的行的数据。

1 transfer_example 使用 f1 列作为主键列。 Adaptive Server 使用此列确定传入行是否包含新数据，或它是否替换现有行。

例如，如果通过向键为 3、5 和 7 的行的各个值添加 10 来替换它们：

```
update transfer_example
set f1 = f1 + 10
where f1 in (3,5,7)
(3 rows affected)
```

transfer_example 现在包括键为 13、15 和 17 的行，transfer 将其视为新行。在向 transfer_example 中传输相同数据时， transfer 会插入键为 3、 5 和 7 的行，并保留键为 13、 15 和 17 的行。

```
transfer table transfer_example
from 'transfer_example-data.ase'
for ase
(10 rows affected)
```

2 检验 f2 和 f3 的行 3 中的数据是否与行 13 中的数据相同，行 5 中的数据是否与行 15 中的数据相同，行 7 中的数据是否与行 17 中的数据相同：

```
select * from transfer_example
order by f1
f1     f2                              f3
------- ------------------------------- ----------------------------
1       AAA                              Jul 17 2009  4:40:14.465789PM
2       BBBBBB                           Jul 17 2009  4:40:14.488003PM
3       CCCCCCCCC                        Jul 17 2009  4:40:14.511749PM
4       DDDDDDDDDDDD                     Jul 17 2009  4:40:14.536653PM
5       EEEEEEEEEEEEEEE                  Jul 17 2009  4:40:14.559480PM
6       FFFFFFFFFFFFFFFFF                Jul 17 2009  4:40:14.583400PM
7       GGGGGGGGGGGGGGGGGGG              Jul 17 2009  4:40:14.607196PM
8       HHHHHHHHHHHHHHHHHHHHHHHHH        Jul 17 2009  4:40:14.632152PM
9       IIIIIIIIIIIIIIIIIIIIIIIII        Jul 17 2009  4:40:14.655184PM
```

```
10      JJJJJJJJJJJJJJJJJJJJJJJJJJJJJ  Jul 17 2009  4:40:14.678938PM
13      CCCCCCCCC                      Jul 17 2009  4:40:14.511749PM
15      EEEEEEEEEEEEEEE                Jul 17 2009  4:40:14.559480PM
17      GGGGGGGGGGGGGGGGGGGG           Jul 17 2009  4:40:14.607196PM

(13 rows affected)
```

3 将 transfer_example 再次传输出来：传输所有的 13 行。 Adaptive Server 将传输进来的键为 3、5 和 7 的行视为新行，因为它们替换了现有行 （此示例使用跟踪 ID 值 101）：

```
transfer table transfer_example
to 'transfer_example-data-01.ase'
for ase
with tracking_id = 101
(13 rows affected)
```

4 修改行以显示增量 transfer 只传输上一次传输之后修改的行 （此更新影响 3 行）。

```
update transfer_example
set f3 = current_bigdatetime()
where f1 > 10
(3 rows affected)
```

5 将表再次传输出来以检验只传输了 3 个已更改行。无需指定 for ase：Adaptive Server 将上一次传输中设置的此参数用作缺省值。

```
transfer table transfer_example
to 'transfer_example-data-02.ase'
with tracking_id = 102
(3 rows affected)
```

6 使用步骤 3 中的 tracking_id 查看传输信息：

```
select id, sequence_id, end_code, ts_floor, ts_ceiling, row_count
from spt_TableTransfer
where id = object_id('transfer_example')
and tracking_id = 101
id          sequence_id   end_code    ts_floor             ts_ceiling
row_count
-----------   -----------    --------     --------------------   ------
--------------
592002109    3             0           5309                  5716
13

(1 row affected)
```

删除数据

delete 适用于单行和多行操作。

where 子句指定要删除哪些行。delete 语句中没有提供 where 子句时，将删除表中的所有行。请参见《参考手册：命令》。

将 *from* 子句用于 *delete*

在 delete 关键字后面紧跟可选的 from 是为了与其它 SQL 版本兼容。delete 语句中第二位的 from 子句是一项特殊的 Transact-SQL 功能，它允许从一个或多个表中选择数据，并从第一个命名的表中删除相应的数据。在 from 子句中选定的行指定 delete 的条件。

假设一个复杂公司业务的结果是另一出版社获得 College Town 市（前 Oakland）的所有作者及其书籍。您必须立即从 titles 表中删除所有这些书籍，但您不知道它们的 title 或标识号。仅有的信息是“author”（作者）的姓名和地址。

可以通过查找 authors 表中以 Big Bad Bay City 作为 town 的行中的 author 标识号，并使用这些标识号查找这些书在 titleauthor 表中的 title 标识号来删除 titles 中的行。也即，查找 titles 表中要删除的行需要一个三向连接。

这三个表都包含在 delete 语句的 from 子句中。然而，只有 titles 表中满足 where 子句的行才被删除。必须使用单独的 delete 语句来删除非 titles 表中的相应行。

以下是所需的语句：

```
delete titles
from authors, titles, titleauthor
where titles.title_id = titleauthor.title_id
and authors.au_id = titleauthor.au_id
and city = "Big Bad Bay City"
```

pubs2 数据库中的 deltitle 触发器禁止实际执行此删除操作，因为它不允许删除在 sales 表中有销售记录的任何 title。

从 IDENTITY 列删除

对于包含 IDENTITY 列的表，可以在 delete 语句中使用 syb_identity 关键字。例如，以下语句删除 row_id 等于 1 的行：

```
delete sales_monthly
where syb_identity = 1
```

删除 IDENTITY 列中的相应行后，可能需要消除表的 IDENTITY 列的编号顺序中的间隔。请参见第 315 页的 “用 bcp 对表的 IDENTITY 列重新编号”。

删除表中的所有行

使用 truncate table 可删除表中的所有行。truncate table 几乎总是比不带条件的 delete 语句快，因为 delete 记录每次更改，而 truncate table 只记录整个数据页的重新分配。 truncate table 可立即释放表数据和索引占用的所有空间。然后自由空间可被任何对象使用。所有索引的分布页也重新分配。给表添加新行后，运行 update statistics。

与 delete 一样，用 truncate table 腾空的表，连同其索引和其它相关对象都保留在数据库中，除非输入 drop table 命令。

如果另一表有通过参照完整性约束引用它的行，则不能使用 truncate table。先删除外表的行，或截断外表，然后再截断主表。请参见第 248 页的 “创建参照完整性约束的一般规则”。

truncate table 语法

truncate table 的语法是：

```
truncate table [ [ database.]owner.]table_name
    [ partition partition_name ]
```

例如，要删除 sales 中的所有数据，请输入：

```
truncate table sales
```

有关 partition 子句的信息，请参见第 10 章 “对表和索引进行分区”。

与 drop table 相似，缺省情况下，授予表的所有者使用 truncate table 的权限，但不能转让该权限。

truncate table 命令不会由 delete 触发器捕获。请参见第 20 章 “触发器：强制实施参照完整性”。

第 9 章

SQL 派生表

SQL 派生表由一个或多个表通过查询表达式求值来定义。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。它不在系统目录中描述或不存储在磁盘上。

与抽象计划派生表的区别

不要将 SQL 派生表与抽象计划派生表混淆。抽象计划派生表用于查询处理、查询优化和查询执行中。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，最终用户无法看到它。

SQL 派生表的工作方式

如下例所示， SQL 派生表是通过由嵌套的 select 语句组成的派生表表达式创建的，它返回 pubs2 数据库的 publishers 表中城市的列表：

```
select city from (select city from publishers)
cities
```

该 SQL 派生表名为 cities，并且具有一个标题为 city 的列。该 SQL 派生表由嵌套的 select 语句定义并且只在查询期间存在，它将返回以下结果：

```
city
--------------------
Boston
Washington
Berkeley
```

SQL 派生表的优点

如果只想查看在科罗拉多编写的书的标题，则可以创建如下视图：

```
create view vw_colorado_titles as
     select title
     from titles, titleauthor, authors
     where titles.title_id = titleauthor.title_id
     and titleauthor.au_id = authors.au_id
     and authors.state ="CO"
```

可以重复使用存储在内存中的视图 vw_colorado_titles，以显示其结果：

```
select * from vw_colorado_titles
```

一旦不再需要该视图，它就被删除：

```
drop view vw_colorado_titles
```

如果查询结果只需要一次，则可改用 SQL 派生表：

```
select title
from (select title
      from titles, titleauthor, authors
      where titles.title_id = titleauthor.title_id
      and titleauthor.au_id = authors.au_id and
      authors.state = "CO") dt_colo_titles
```

创建的 SQL 派生表名为 dt_colo_titles。 SQL 派生表只在查询期间存在，这与临时表不同，后者在整个会话期间都存在。

在前面的示例 （查询结果只需要一次）中，视图不如 SQL 派生表查询适合，因为视图更复杂，不但需要 create 和 drop 语句，而且需要 select 语句。只为一个查询而创建视图，其好处远远抵不上由此带来的诸如处理系统目录之类的管理负担。SQL 派生表通过使查询可以自动创建非持久表，而无需删除表或将内容插入到系统目录中，消除了这一开销。因此，不需要执行任何管理任务。执行使用了多次的 SQL 派生表相当于将视图与高速缓存的定义一起使用的查询。

SQL 派生表和优化

表示为一条 SQL 语句的查询可以比用两条或更多 SQL 语句表示的查询更好地利用优化程序。 SQL 派生表能够确保简明地用单个步骤表示查询，而不使用 SQL 派生表则可能要使用若干 SQL 语句和临时表，特别是在必须存储中间集合结果的情况下。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
```

集合结果取自 SQL 派生表 dt_1 和 dt_2，并且在两个 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。

SQL 派生表语法

用于 SQL 派生表的查询表达式在 select 或 select into 命令的 from 子句中指定 （代替表名或视图名）：

from_clause ::=
    from *table_reference* [,*table_reference*]...

table_reference ::=
    table_view_name | *ANSI_join*

table_view_name ::=
    {*table_view_reference* | *derived_table_reference*}
    [holdlock | noholdlock]
    [readpast]
    [shared]

table_view_reference ::=
[[*database*.]*owner*.]{*table_name* | *view_name*}
[[as] *correlation_name*]
[index {*index_name* | *table_name* }]
[parallel [*degree_of_parallelism*]]
[prefetch size]
[lru | mru]

derived_table_reference ::=
derived_table [as] *correlation_name*
['(' *derived_column_list*')']

derived_column_list ::= *column_name* [',' *column_name*] ...

derived_table ::= '(' *select* ')'

派生表表达式类似于 create view 语句中的 select 并且遵守相同的规则，但有以下例外情况：

- 在派生表表达式中允许使用临时表，但在表达式是 create view 语句的一部分时除外。
- 在派生表表达式中允许使用局部变量，但在表达式是 create view 语句的一部分时除外。不能在派生表表达式中向一个变量赋值。
- 不能在作为 create view 语句一部分的派生表语法 （其中的派生表在游标中引用）中使用变量。
- correlation_name （必须跟在派生表表达式的后面以指定 SQL 派生表的名称）可以省略派生列列表，而视图不能具有未命名的列：

```
select * from
   (select sum(advance) from total_sales) dt
```

请参见 《参考手册：命令》中 create view 的 “用法”部分的 “视图的限制”。

派生列列表

如果派生列列表不包括在 SQL 派生表中，则 SQL 派生表列的名称必须匹配在派生表表达式的目标列中指定的列名称。如果列名未在派生表表达式的目标列表中指定 （如在派生表表达式的目标列表中提供了常量表达式或集合的情况），则 SQL 派生表中的结果列没有名称。服务器返回错误 11073 “A derived table expression may not have null column names...”

如果派生列列表包括在 SQL 派生表中，则它必须为派生表表达式的目标列表中的所有列指定名称。这些列名必须用于查询块中，代替 SQL 派生表的自然列名。这些列必须按它们在派生表表达式中出现的顺序列出，并且在派生列列表中不能多次指定一个列名。

相关 SQL 派生表

不支持相关 SQL 派生表 （不属于 ANSI 标准）。例如，不支持以下查询，因为它在用于 dt_publishers1 的派生表表达式内引用了 SQL 派生表 dt_publishers2：

```
select * from
   (select * from titles where titles.pub_id =
      dt_publishers2.pub_id) dt_publishers1,
   (select * from publishers where city = "Boston")
      dt_publishers2
where dt_publishers1.pub_id = dt_publishers2.pub_id
```

同样，不支持以下查询，因为用于 dt_publishers 的派生表表达式引用 publishers_pub_id 列，而该列在 SQL 派生表范围之外：

```
select * from publishers
  where pub_id in (select pub_id from
                      (select pub_id from titles
                      where pub_id = publishers.pub_id)

                   dt_publishers)
```

以下查询使用了正确的引用，因此受到支持：

```
select * from publishers
  where pub_id in (select pub_id from
                      (select pub_id from titles)
                   dt_publishers
                     where pub_id = publishers.pub_id)
```

使用 SQL 派生表

SQL 派生表可以用作使用混合 SQL 子句和运算符的大型集成查询的一部分。

嵌套

查询可以使用大量的嵌套派生表表达式，即定义 SQL 派生表的 SQL 表达式。在下例中，最内层的派生表表达式定义 SQL 派生表 dt_1，用于构成派生表表达式的 select 定义 SQL 派生表 dt_2。

```
select postalcode
   from (select postalcode
            from (select postalcode
                    from authors) dt_1) dt_2
```

嵌套的程度被限制为 25。

使用 SQL 派生表的子查询

可在子查询 from 子句中使用 SQL 派生表。例如，以下查询会查找出版过商业书籍的出版社的名称：

```
select pub_name from publishers
   where "business" in
      (select type from
         (select type from titles, publishers
             where titles.pub_id = publishers.pub_id)
       dt_titles)
```

在上例中， dt_titles 是最内层 select 语句定义的 SQL 派生表。

SQL 派生表可用于子查询 （只要子查询合法）的 from 子句中。有关子查询的详细信息，请参见第 5 章 “子查询：在其它查询中使用查询”。

联合

在派生表表达式中允许使用 union 子句。例如，以下查询生成 sales 和 sales_east 表的 stor_id 和 ord_num 列的内容：

```
select * from
   (select stor_id, ord_num from sales
    union
    select stor_id, ord_num from sales_east)
dt_sales_info
```

在上例中，两个 select 操作的联合定义 SQL 派生表 dt_sales_info。

子查询中的联合

在派生表表达式的子查询中允许使用 union 子句。下例在 SQL 派生表的子查询中使用 union 子句，以列出在 sales 和 sales_east 表中列出的商店售出的书的标题：

```
select title_id from salesdetail
   where stor_id in
      (select stor_id from
         (select stor_id from sales
          union
          select stor_id from sales_east)
       dt_stores)
```

使用 SQL 派生表重命名列

如果包括派生列列表以用于 SQL 派生表，则它跟在 SQL 派生表的名称之后并且用小括号括起来，如下例中所示：

```
select dt_b.book_title, dt_b.tot_sales
   from (select title, total_sales
            from titles) dt_b (book_title, tot_sales)
   where dt_b.book_title like "%Computer%"
```

在上例中，使用派生列列表将派生表表达式中的列名 title 和 total_sales 分别重命名为 book_title 和 tot_sales。 book_title 和 tot_sales 列名用于其余查询。

注释　SQL 派生表不能具有未命名的列。

常量表达式

如果在派生表表达式的目标列表中未指定列名（如将常量表达式用于列名的情况），则 SQL 派生表中的结果列没有名称：

```
1> select * from
2> (select title_id, (lorange + hirange)/2
3> from roysched) as dt_avg_range
4> go

title_id
--------- -----------
BU1032    2500
BU1032    27500
PC1035    1000
PC1035    2500
```

您可以使用派生列列表为派生表表达式的目标列表指定列名：

```
1> select * from
2> (select title_id, (lorange + hirange)/2
3> from roysched) as dt_avg_range (title, avg_range)
4> go

title     avg_range
--------- -----------
BU1032    2500
BU1032    27500
PC1035    1000
PC1035    2500
```

或者，您可以通过重命名派生表表达式的目标列表中的列来指定列名：

```
1> select * from
2> (select title_id, (lorange + hirange)/2 avg_range
3> from roysched) as dt_avg_range
4> go

title     avg_range
--------- -----------
BU1032    2500
BU1032    27500
PC1035    1000
PC1035    2500
```

注释　如果在派生列列表和派生表表达式的目标列表中都指定了列名，则按派生列列表命名结果列。派生列列表中的列名优先于派生表表达式的目标列表中指定的名称。

如果在 create view 语句中使用某一常量表达式，则必须为该常量表达式的结果指定列名。

集合函数

派生表表达式可以使用集合函数，如 sum、 avg、 max、 min、 count_big 和 count。下例从 SQL 派生表 dt_a 选择列 pub_id 和 adv_sum。在派生表表达式中通过对 titles 表的 advance 列使用 sum 函数创建第二个列。

```
select dt_a.pub_id, dt_a.adv_sum
   from (select pub_id, sum(advance) adv_sum
         from titles group by pub_id) dt_a
```

如果您在 create view 语句中使用集合函数，则必须为集合结果指定列名。

使用 SQL 派生表连接

下例说明 SQL 派生表和一个现有表之间的连接。该连接由 where 子句指定。连接的两个表分别是 dt_c（一个 SQL 派生表）和 publishers（pubs2 数据库中的现有表）。

```
select dt_c.title_id, dt_c.pub_id
   from (select title_id, pub_id from titles) as dt_c,
        publishers
   where dt_c.pub_id = publishers.pub_id
```

下例说明两个 SQL 派生表之间的连接。连接的两个表分别为 dt_c 和 dt_d。

```
select dt_c.title_id, dt_c.pub_id
   from (select title_id, pub_id from titles)
         as dt_c,
        (select pub_id from publishers)
         as dt_d
   where dt_c.pub_id = dt_d.pub_id
```

涉及 SQL 派生表的外连接也可能发生。 Sybase 支持左外连接和右外连接。下例说明两个 SQL 派生表之间的左外连接。

```
select dt_c.title_id, dt_c.pub_id
   from (select title_id, pub_id from titles)
         as dt_c,
        (select title_id, pub_id from publishers)
         as dt_d
   where dt_c.title_id *= dt_d.title_id
```

下例说明派生表表达式中的左外连接。

```
select dt_titles.title_id
   from (select * from titles, titleauthor
         where titles.title_id *= titleauthor.title_id)
dt_titles
```

从 SQL 派生表创建表

可将从 SQL 派生表获取的数据插入一个新表，如下例中所示。

```
select pubdate into pub_dates
   from (select pubdate from titles) dt_e
            where pubdate = "450128 12:30:1PM"
```

上例中将来自 SQL 派生表 dt_e 的数据插入到新表 pub_dates 中。

将视图用于 SQL 派生表

下例使用 SQL 派生表 dt_colo_pubs 创建视图 view_colo_publishers，以显示位于科罗拉多的出版社：

```
create view view_colo_publishers (Pub_Id, Publisher,
City, State)
as select pub_id, pub_name, city, state
from
(select * from publishers where state="CO")
dt_colo_pubs
```

如果用于派生表表达式的 insert 规则和权限设置遵守用于 create view 语句的 select 部分的 insert 规则和权限设置，则可通过包含 SQL 派生表的视图插入数据。例如，以下 insert 语句通过 view_colo_publishers 视图将某一行插入该视图所基于的 publishers 表：

```
insert view_colo_publishers
values ('1799', 'Gigantico Publications', 'Denver',
'CO')
```

您还可以通过使用 SQL 派生表的视图更新现有数据：

```
update view_colo_publishers
set Publisher = "Colossicorp Industries"
where Pub_Id = "1699"
```

注释　您必须指定视图定义的列名，而不是基础表的列名。

使用 SQL 派生表的视图按标准方式删除：

```
drop view view_colo_publishers
```

相关属性

不能从 SQL 派生表表达式引用超过 SQL 派生表的范围的相关属性。例如，以下查询会导致错误：

```
select * from publishers
   where pub_id in
      (select pub_id from
         (select pub_id from titles
          where pub_id = publishers.pub_id)
       dt_publishers)
```

在上例中，列 publishers.pub_id 在 SQL 派生表表达式中被引用，但它在 SQL 派生表 dt_publishers 的范围外。

第 10 章

对表和索引进行分区

本章介绍如何创建和管理数据和索引分区。

概述

分区可提高性能并帮助管理数据。分区对管理大型表和索引尤为有用，可将它们划分为较小、更易管理的片。分区像大规模的索引一样，利用它可以更快、更便捷地访问数据。

每个分区可以驻留在单独的段上。分区是数据库对象，可单独进行管理。例如，您可以在分区级别装载数据和创建索引。不过，分区对最终用户是透明的，无论表是否已进行分区，最终用户都可以使用同样的 DML 命令选择、插入和删除数据。

分区是并行处理的基础，并行处理可显著提高性能。

Adaptive Server 支持水平分区，在这种分区中，选定的表行可以在磁盘设备之间分配。根据分区策略，单个表或索引行被分配到分区。可以通过以下几种方法对表进行分区：

- 循环 — 通过随机分配。
- 散列 — 使用散列函数确定行分配 （基于语义的分区）。
- 域 — 根据数据行中处于指定的值范围内的值 （基于语义的分区）。
- 列表 — 根据数据行中与指定值相匹配的值 （基于语义的分区）。

注释 基于语义的分区是单独授权的。若要在授权的节点上启用语义分区，请将 enable semantic partitioning 配置参数的值设为 1。请参见 《系统管理指南：卷 1》中的第 5 章 “设置配置参数”。

分区有如下优点：

- 提高可伸缩性。
- 提高性能 — 在不同分区上可同时运行多个 I/O，在多个分区上可同时运行多个 CPU 的多个线程。
- 加快响应速度。
- 分区对于应用程序是透明的。
- 超大型数据库 (VLDB) 支持 – 可同时扫描超大型表的多个分区。
- 域分区可管理历史数据。

注释 缺省情况下， Adaptive Server 用单个分区和循环分区策略创建表。这些表称为 “未分区表”，以便更好地区分未用分区语法 （缺省情况下）创建或修改的表和用分区语法创建的表。

从 Adaptive Server 12.5.x 和更早版本升级

在从 12.5.x 和更早版本升级时， Adaptive Server 15.0 和更高版本会将所有数据库表 （包括 12.5.x 版中受支持的片分区表）全部更改为未分区表。索引不会发生变化；它们仍是全局性的和未分区的。

若要对数据库表或分区索引进行重新分区，请按照本章中的说明操作。

数据分区

数据分区是独立的数据库对象，具有唯一的分区 ID。它是表的子集，共享基表的列定义以及参照和完整性约束。若要最大限度地提高 I/O 并行度， Sybase 建议将每个分区绑定到一个不同的段，然后再将每个段绑定到一台不同的存储设备。

分区键

每个语义分区表都有一个分区键，用于确定如何将各数据行分配到不同的分区。分区键由一个分区键列或多个键列组成。这些键列中的值决定实际的分区分配。

域分区表和散列分区表的分区键中的键列可多达 31 个。列表分区的分区键中可只有一个键列。循环分区表没有分区键。

除以下类型外，您可以指定任何类型的分区键列：

- text、 image 和 unitext
- bit
- Java 类
- 计算列

如果表包含使用上述数据类型的列，可以对这些表进行分区，但分区键列必须是受支持的数据类型。

索引分区

索引和表一样，可以进行分区。在 Adaptive Server 15.0 之前的版本中，所有索引都是全局性的。在 Adaptive Server 15.0 中，既可以创建本地索引也可以创建全局索引。

索引分区是用索引 ID 和分区 ID 的唯一组合进行标识的、独立的数据库对象；它是索引的子集，驻留在段或其它存储设备上。

Adaptive Server 支持本地和全局索引。

- *本地索引* — 只跨一个数据分区中的数据。对于语义分区表，本地索引的分区按照其基表进行均分；即表和索引共享相同的分区键和分区类型。

  对于具有本地索引的所有分区表，每个本地索引分区有且仅有一个相对应的数据分区。

- *全局索引* — 跨表中的所有数据分区。 Sybase 仅支持未分区的全局索引。未分区表上的所有未分区索引都是全局性的。

可以在分区表中混合分区和未分区索引：

- 分区表可以包含分区和未分区索引。
- 未分区表只能包含未分区的全局索引。

分区 ID

与对象 ID 类似，分区 ID 是伪随机数。分区 ID 和对象 ID 都是从同一数字空间分配的。索引或数据分区用索引 ID 和分区 ID 的唯一组合来标识。

锁定和分区

Adaptive Server 在执行任何 DDL 命令时，都会根据需要以共享或排它模式锁定整个表，即使在该操作只影响某些分区时也是如此。 Adaptive Server 不锁定单个分区。

分区类型

Adaptive Server 支持以下四种数据分区类型：

- 域分区
- 散列分区
- 列表分区
- 循环分区

域分区

域分区表或索引中的行根据分区键列中的值在分区之间分配。每行的分区列值将与一组上限和下限值进行比较，以确定该行所属的分区。

- 每个分区都有一个在创建分区时由 values <= 子句指定的包含的上限。
- 除第一个分区外，其它分区具有非包含的下限，这是由临近的下一个分区上的 values <= 子句隐式指定的。

域分区对 OLTP 和决策支持环境中的高性能应用程序尤为有用。选择域时要仔细地将行平均分配到所有分区，因此，了解分区键列的数据分配对于在分区之间均匀地平衡负载十分重要。

域分区是有顺序的，也就是说每个后续分区的边界都必须高于前一个分区。

散列分区

对于散列分区，Adaptive Server 使用散列函数为每行指定分区分配。您选择的是分区键列，而 Adaptive Server 选择可控制分区分配的散列函数。

散列分区非常适用于：

- 具有许多分区的大型表，尤其是在决策支持环境中。
- 对散列键列进行有效等同性搜索
- 没有特定顺序的数据，例如，字母数字产品代码密钥

如果选择了适当的分区键，则散列分区在所有分区之间平均分配数据。但是，如果所选的分区键不合适 （例如，键在多个行具有相同值），则可能会产生倾斜的数据，从而导致行在分区之间的分配不均衡。

列表分区

和域分区一样，列表分区按语义分配行；即，根据分区键列中的实际值进行分配。列表分区只有一个键列。分区键列中的值将与一组用户提供的值进行比较，以确定各行所属的分区。分区键必须与为分区指定的某个值完全匹配。

每个分区的值列表必须至少包含一个值，值列表在所有分区间必须是唯一的。在每个列表分区中可指定多达 250 个值。列表分区没有顺序。

循环分区

在循环分区中， Adaptive Server 不使用分区标准。循环分区表没有分区键。 Adaptive Server 以循环方式将行分配给每个分区，使每个分区所包含的行数相差无几，以实现平衡负载的目的。由于没有分区键，因此行在所有分区之间随机分配。

支持循环分区主要是为了与早于 Adaptive Server 15.0 的版本兼容。此外，循环分区还提供了：

- 多个插入点以供将来插入时使用
- 使用并行度提高性能的方法
- 执行管理任务 （如更新统计信息和截断各分区上的数据）的方法

组合分区键

对于语义分区表，每个表或索引都有一个分区键。对于域分区表或散列分区表，分区键可以是具有多达 31 个键列的组合键。如果散列分区表具有组合分区键， Adaptive Server 会获取所有分区键列中的值并用系统提供的散列函数散列计算结果数据流。

如果域分区表具有多个分区键列，则 Adaptive Server 会将每个数据行中相应的分区键列值与每个分区的上限和下限进行比较。每个分区边界都是包含一个或多个值的列表，该列表中的值与分区键列一一对应。

Adaptive Server 在首次创建表时，会按指定顺序将分区键值与边界进行比较。如果第一个键值满足分区的分配标准，则将该行分配给该分区并且不计算其它键值。如果第一个键值不满足分配标准，则计算后续键值，直到满足分配标准为止。因此， Adaptive Server 为确定分区分配，少则可能会计算一个分区键值，多则会计算所有键值。

例如，假设 key1 和 key2 是 my_table 的分区列。该表由以下三个分区组成：p1、 p2 和 p3。 p1 的声明上限为 (a, b)， p2 的为 (c, d)， p3 的则为 (e, f)。

```
if key1 < a, then the row is assigned to p1
if key1 = a, then

    if key2 < b or key2 = b, then the row is assigned to p1
if key1 > a or (key1 = a and key2 > b), then
    if key1 < c, then the row is assigned to p2
    if key1 = c, then
        if key2 < d or key2 = d, then the row is assigned to p2
```

```
if key1 > c or (key1 = c and key2 > d), then
    if key1 < e, then the row is assigned to p3
    if key1 = e, then
        if key2 < f or key2 = f, then the row is assigned to p3
        if key2 > f, then the row is not assigned
```

假设 pubs2 中的 roysched 表是按域进行分区的。分区列是高范围 (hirange) 和版税 (royalty)。有三个分区：p1、 p2 和 p3。 p1 的上限为 (5000, 14)， p2 的上限为 (10000, 10)， p3 的上限为 (100000, 25)。

可以使用 alter table 在 roysched 表中创建分区：

```
alter table roysched partition
       by range (hirange, royalty)
   (p1 values <= (5000, 14),
   p2 values <= (10000, 10),
   p3 values <= (100000, 25))
```

Adaptive Server 按如下方式对行进行分区：

- 具有以下分区键值的行将分配给 p1：(4001, 12)、(5000, 12)、 (5000, 14)、 (3000, 18)。
- 具有以下分区键值的行将分配给 p2：(6000, 18)、(5000, 15)、 (5500, 22)、 (10000, 10)、 (10000, 9)。
- 具有以下分区键值的行将分配给 p3：(10000, 22)、(80000, 24)、 (100000, 2)、 (100000, 16)。

Adaptive Server 用类似的方式计算具有两个以上分区键列的表。

分区清理

基于语义的分区可允许 Adaptive Server 在执行搜索时排除某些分区。例如，基于域的分区包含一些行，这些行的分区键是离散值集。当某一查询谓词 （where 子句）基于这些分区键时， Adaptive Server 可快速确定特定分区中的行是否满足该查询。此行为称为分区清理或分区排除，它可以在执行期间节省可观的时间和资源。

- 对于域和列表分区 — Adaptive Server 可对单个表上的分区键列的等同性 (=) 和域 （>、 >=、 < 和 <=）谓词应用分区清理。
- 对于散列分区 — Adaptive Server 只能对单个表上的等同性谓词应用分区清理。

- 对于域、列表和散列分区 — Adaptive Server 不能对包含 “不等于” (!=) 子句的谓词应用分区清理，也不能对在分区列上包含表达式的复杂谓词应用分区清理。

例如，假设 pubs2 中的 roysched 表在 hirange 和 royalty 上进行分区（请参见第 356 页的 “组合分区键”）。 Adaptive Server 可对以下查询使用分区清理：

```
select avg(royalty) from roysched
where hirange <= 10000 and royalty < 9
```

分区清理过程将 p1 和 p2 标识为符合此查询的唯一分区。这样就不需要扫描 p3 分区，而只需扫描 p1 和 p2，因此 Adaptive Server 可以更高效地返回查询结果。

在下面这些示例中， Adaptive Server 不能使用分区清理：

```
select * from roysched
where hirange != 5000
select * from roysched
where royalty*0.15 >= 45
```

注释 在串行执行模式中，分区清理只适用于扫描、插入、删除和更新，而不适用于其它运算符。在并行执行模式中，分区清理适用于所有运算符。

索引和分区

索引有助于 Adaptive Server 定位数据。通过指向表列数据在磁盘上的位置，索引可加快数据检索过程。可以创建全局索引和本地索引，这两种索引还可以是聚簇或非聚簇的。

在聚簇索引中，物理数据的存储顺序与索引相同，索引的最低级别包含实际的数据页。在非聚簇索引中，物理数据的存储顺序与索引不同，索引的最低级别包含指向数据页上的行的指针。

无论是否在 create index 命令中指定 “本地”索引，语义分区表上的聚簇索引始终是本地索引。循环表上的聚簇索引既可以是全局性的，也可以是本地的。

全局索引

全局索引跨一个或多个分区中的数据，这些索引不按照基表均分。由于 Sybase 仅支持未分区全局索引，因此全局索引跨所有分区。

对于早于 15.0 的 Adaptive Server 版本，在其中的分区表中创建的索引均是全局索引。支持全局索引是为了与 Adaptive Server 的早期版本兼容，而且全局索引在 OLTP 环境中尤为有用。

Adaptive Server 支持下列类型的全局索引：

- 循环表和未分区表上的聚簇索引
- 所有类型的表上的非聚簇索引

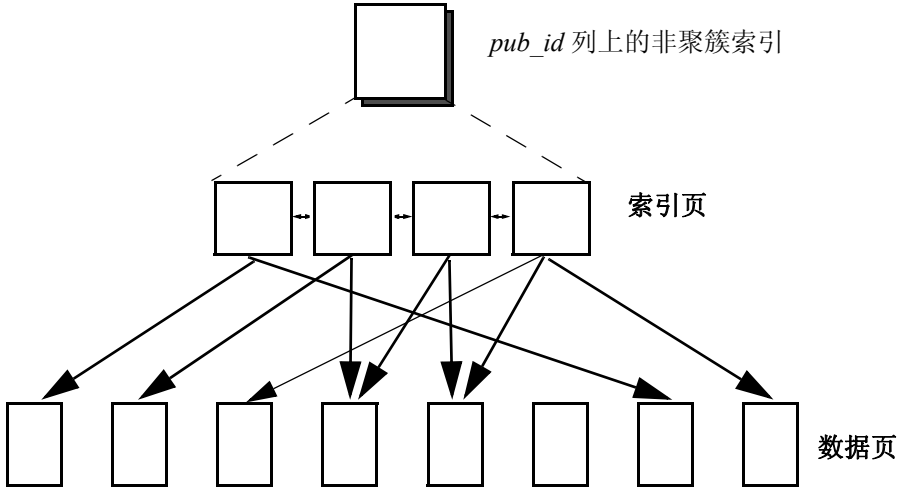
未分区表上的全局非聚簇索引

图 10-1 中的示例显示了缺省的非聚簇索引配置，该配置受 Adaptive Server 12.5 和更高版本的支持。

若要在未分区表 publishers 上创建此索引，请输入：

```
create nonclustered index publish5_idx on
    publishers(pub_id)
```

图 10-1：未分区表上的全局非聚簇索引



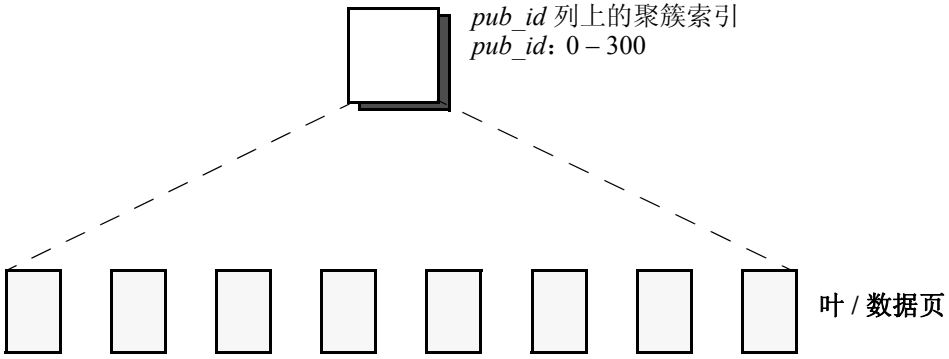
未分区表上的全局聚簇索引

图 10-2 显示了缺省的聚簇索引配置。该表和索引是未分区的。

若要在未分区表 publishers 上创建此索引，请输入：

```
create clustered index publish4_idx
    on publishers(pub_id)
```

图 10-2：未分区表上的全局聚簇索引



循环分区表上的全局聚簇索引

Adaptive Server 仅支持循环分区表上的未分区聚簇全局索引。这种类型的索引与 Adaptive Server 12.5.x 中的分区表一致。

注释 如果循环表的数据分区超过 255 个，则不能在该表上创建全局索引，必须创建本地索引。

未分区索引允许对所有分区进行全表扫描。由于叶索引页在 APL 表上也是数据页，因此当所有数据分区都驻留在同一段时，此索引最有用。必须在数据分区键上创建该索引。

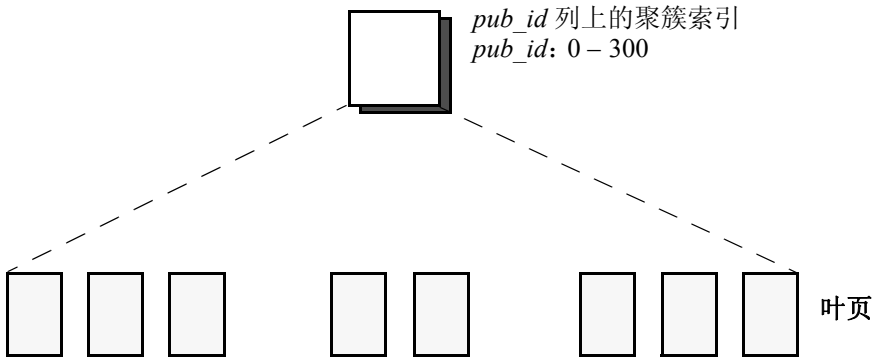
对于下例 （请参见图 10-3 ）中，在 pubs2 中的 publishers 表上创建了三个循环分区。在创建分区之前，请确保删除了全部索引。

```
alter table publishers partition 3
```

若要在循环分区表 publishers 上创建聚簇索引，请输入：

```
create clustered index publish1_idx
    on publishers(pub_id)
```

图 10-3：循环分区表上的全局聚簇索引



分区表上的全局非聚簇索引

可以为所有分区表策略创建非聚簇和未分区全局索引。

索引和数据分区可驻留在相同或不同的段中。可以在表中的任何可索引列上创建索引。

图 10-4 中的示例在 pub_name 列上建立索引；在 pub_id 列上对表进行分区。

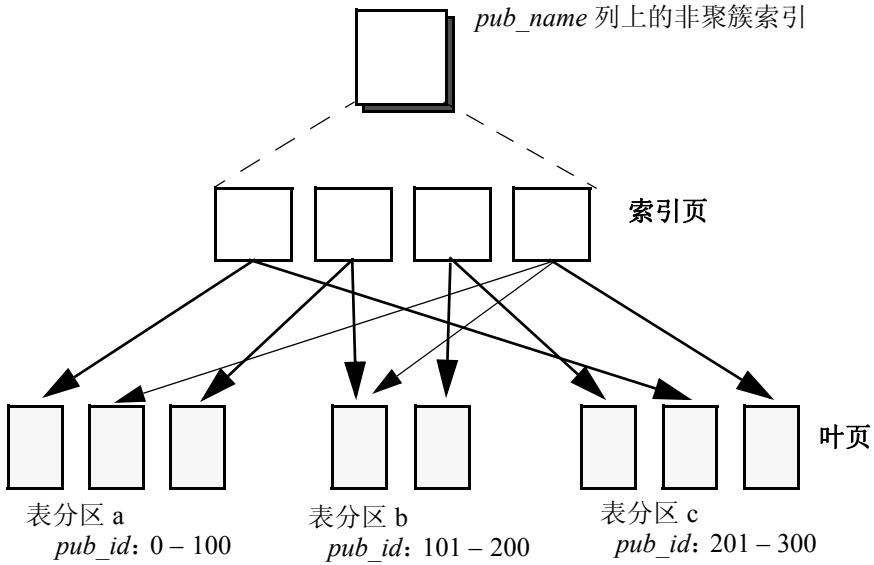
在此示例中，使用 alter table 对 publishers 进行重新分区，从而使该表在 pub_id 列上有三个域分区。

```
alter table publishers partition by range(pub_id)
(a values <= ("100"),
b values <= ("200"),
c values <= ("300"))
```

若要在 pub_name 列上创建全局非聚簇索引，请输入：

```
create nonclustered index publish2_idx
    on publishers(pub_name)
```

图 10-4：分区表上的全局非聚簇索引

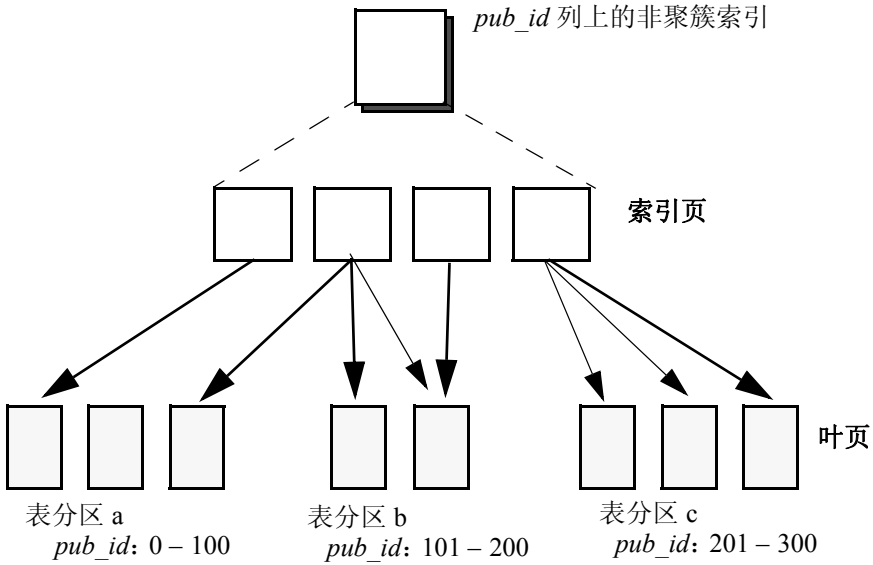


在图 10-5 的示例中，在 pub_id 列上建立索引；此外还在 pub_id 列上对表进行分区。

若要在 pub_id 列上创建全局非聚簇索引，请输入：

```
create nonclustered index publish3_idx
    on publishers(pub_id)
```

图 10-5：分区表上的全局非聚簇索引



本地索引

所有本地索引都按照基表的数据分区均分；也就是说它们继承基表的分区类型和分区键。每个本地索引只跨一个数据分区。可以在域、散列、列表和循环分区表上创建本地索引。本地索引允许多个线程并行扫描每个数据分区，这可以大幅提高性能。

本地聚簇索引

在对表进行分区时，根据值将行分配到分区，但不对数据进行排序。在创建本地索引时，将分别对每个分区进行排序。

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。

图 10-6 显示了分区表上分区聚簇索引的示例。在 pub_id 列上创建索引，并在 pub_id 上为表建立索引。此示例可在 pub_id 列上实现唯一性。

若要在域分区表 publishers 上创建此表，请输入：

```
create clustered index publish6_idx
    on publishers(pub_id)
    local index p1, p2, p3
```

图 10-6：本地聚簇索引 — 唯一

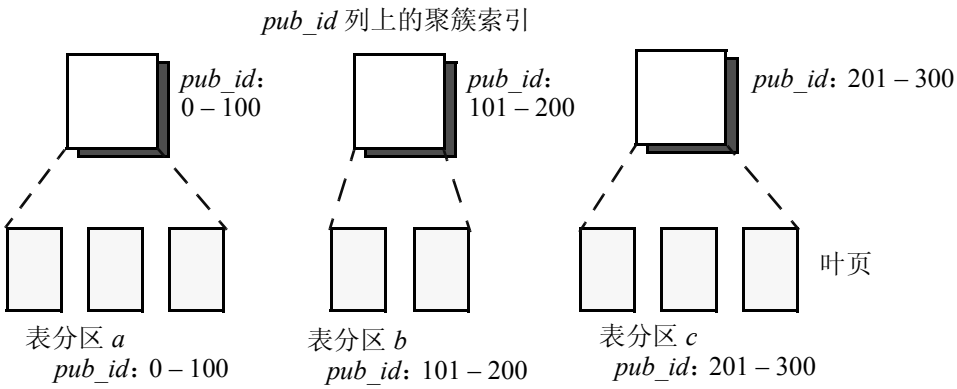
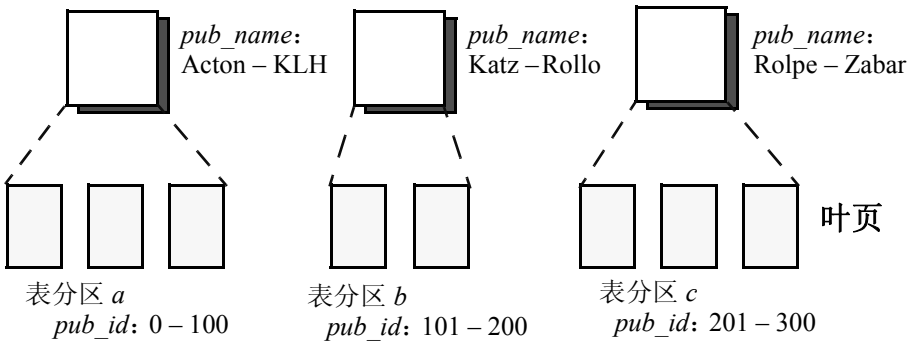


图 10-7 中的示例在 pub_name 列上建立索引。它不能实现唯一性。请参见第 366 页的 “保证唯一索引”。

若要在域分区表 publishers 上创建此示例，请输入：

```
create clustered index publish7_idx
    on publishers(pub_name)
    local index p1, p2, p3
```

图 10-7：本地聚簇索引 – 不唯一



本地非聚簇索引

可以在任何一组可索引列上定义本地非聚簇索引。

使用 publishers 表 （在 pub_id 列上按域对该表进行了分区，如第 361 页的 “分区表上的全局非聚簇索引”中所述）可以在 pub_id 和 city 列上创建一个分区非聚簇索引：

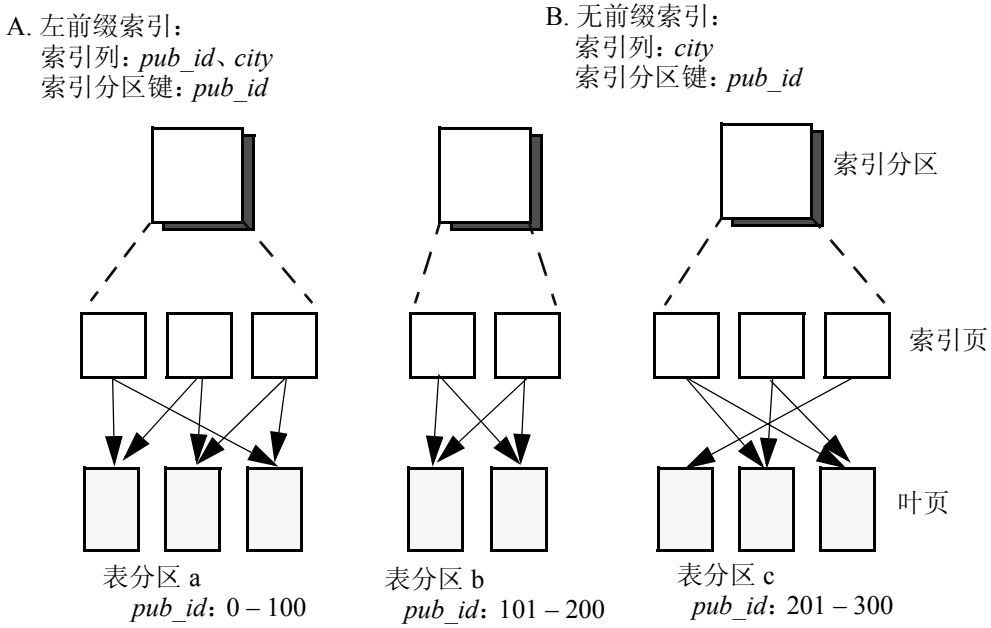
```
create nonclustered index publish8_idx (A)
    on publishers(pub_id, city)
    local index p1, p2, p3
```

还可以在 city 列上创建一个分区非聚簇索引：

```
create nonclustered index publish9_idx (B)
    on publishers(city)
    local index p1, p2, p3
```

图 10-8 显示了非聚簇本地索引的两个示例。每个示例的图形说明是相同的。但在示例 A 中可以实现唯一性；在示例 B 中则不能实现唯一性。请参见第 366 页的 “保证唯一索引”。

图 10-8：本地非聚簇索引



保证唯一索引

唯一索引可确保任意两行均没有相同的索引值（包括 NULL）。在创建索引时，如果数据已经存在，系统将检查重复值，以后每次使用 insert 或 update 增加或修改数据时也会进行检查。有关创建唯一索引的详细信息，请参见第 13 章 “创建表的索引”。

由于全局索引未进行分区，因此可以使用 unique 关键字轻松实现其唯一性。本地索引进行了分区；因此，实现唯一性需要其它一些约束。

要实现本地索引的唯一性，分区键须满足下列条件：

- 必须是索引键的子集
- 必须与索引键具有相同的序列

例如，可以在下列实例中强制实现唯一性：

- 表在 column1 上按散列、列表或域进行分区，其本地索引的索引键在 column1 上。
- 表在 column1 上按散列、列表或域进行分区，其本地索引的索引键在 column1 和 column2 上。请参见图 10-8 中的示例 A。
- 表在 column1 和 column3 上按散列、列表或域进行分区。本地索引具有下列索引键：
  - column1、 column3 或
  - column1、 column2、 column3 或
  - column0、 column1、 column3、 column4

  具有下列索引键的索引不能实现唯一性：column3 或 column1、column3。

不能对具有本地索引的循环分区表实现唯一性。

创建和管理分区

对表进行分区可将表数据划分为更小、更容易管理的片。数据驻留在分区上，而表则在逻辑上联合了各个分区。在使用诸如 select、 insert 和 delete 之类的 DML 命令时，不能指定分区。在使用 DML 命令访问表时，分区是透明的。

启用分区

语义分区是经过授权的功能。若要在授权的节点处启用语义分区，请输入：

```
sp_configure 'enable semantic partitioning', 1
```

循环分区始终可用，它不受 enable semantic partitioning 值的影响。

启用语义分区命令可执行常用的管理和维护操作，如：

- 创建和截断表 — create table、 truncate table
- 改变表以更改锁定或修改方案 — alter table
- 创建索引 — create index
- 更新统计信息 — update statistics
- 重组表页以符合聚簇索引并充分利用空间 — reorg rebuild

对磁盘进行分区

在对表或索引进行分区之前，必须准备好磁盘设备和段或要用于分区的其它存储设备。

可以为一个段分配多个分区，但一个分区只能分配给一个段。通过给每个段分配一个分区，并将设备绑定到各个段，可确保最大限度地利用并行和分区的优势。

分区任务的一般顺序如下：

1 使用 disk init 初始化新的数据库设备。 disk init 会将物理磁盘设备或操作系统文件映射到逻辑数据库设备名。例如：

```
use master
disk init
name ="pubs_dev1",
physname = "SYB_DEV01/pubs_dev",
size = "50M"
```

有关初始化磁盘的详细信息，请参见 《系统管理指南：卷 1》中的第 7 章 “初始化数据库设备”。

2 使用 alter database 将新设备分配给包含要分区的表或索引的数据库。例如：

```
use master
alter database pubs2 on pubs_dev1
```

3 [可选] 使用 sp_addsegment 在数据库中定义段。以下示例假定已经用与创建 pubs_dev1 类似的方法创建了 pubs_dev2、 pubs_dev3 和 pubs_dev4。

```
use pubs2
sp_addsegment seg1, pubs2, pubs_dev1
sp_addsegment seg2, pubs2, pubs_dev2
sp_addsegment seg3, pubs2, pubs_dev3
sp_addsegment seg4, pubs2, pubs_dev4
```

4 从要分区的表中删除所有索引。例如：

```
use pubs2
drop index salesdetail.titleidind,
    salesdetail.salesdetailind
```

5 使用 sp_dboption 将表或索引数据成批复制到新分区中。例如：

```
use master
sp_dboption pubs2,"select into", true
```

6 使用 alter table 对表进行重新分区，或使用 create table 创建一个新的分区表；使用 create index 创建一个新的分区索引；或使用 select into 根据现有表创建一个新的分区表。

例如，若要对 pubs2 中的 salesdetail 表进行重新分区，请输入：

```
use pubs2
alter table salesdetail partition by range (qty)
    (smsales values <= (1000) on seg1,
    medsales values <= (5000) on seg2,
```

```
    lgsales values <= (10000) on seg3)
```

7　在分区表上重新创建索引。例如，在 salesdetail 表上：

```
use pubs2
create nonclustered index titleidind
    on salesdetail (title_id)
create nonclustered index salesdetailind
    on salesdetail (stor_id)
```

创建数据分区

本节介绍如何使用 create table 创建域、散列、列表和循环分区表。请参见《参考手册：命令》。

创建域分区表

以下示例创建一个名为 fictionsales 的域分区表；它有四个分区，每个分区表示一年中的一个季度。为了获得最佳性能，每个分区分别驻留在一个段上：

```
create table fictionsales
    (store_id int not null,
    order_num int not null,
    date datetime not null)
partition by range (date)
    (q1 values <= ("3/31/2004") on seg1,
    q2 values <= ("6/30/2004") on seg2,
    q3 values <= ("9/30/2004") on seg3,
    q4 values <= ("12/31/2004") on seg4)
```

分区键列是 date。q1 分区驻留在 seg1 上，包含 date 值在 3/31/2004 之前的所有行。q2 分区驻留在 seg2 上，包含日期值从 4/1/2004 到 6/30/2004 的所有行。 q3 和 q4 也是按类似的方式进行分区。

如果尝试插入晚于 “12/31/2004” 的 date 值，则会出现错误且插入失败。这样，通过限制可插入到表中的行，域条件可充当表的检查约束。

若要确保包含所有的值（最大可达到数据类型的最大值），请使用 MAX 关键字作为最后创建的分区的上限。例如：

```
create table pb_fictionsales
    (store_id int not null,
    order_num int not null,
    date datetime not null)
partition by range (order_num)
```

```
(low values <= (1000) on seg1,
mid values <= (5000) on seg2,
high values <= (MAX) on seg3)
```

对域分区表的分区键和边界值的限制

分区边界必须依照分区创建顺序按升序排列。也就是说，第二个分区的上限必须高于第一个分区的上限，依此类推。

此外，分区边界值必须与对应的分区键列的数据类型兼容。例如，varchar 与 char 兼容。如果边界值的数据类型与其对应的分区键列的数据类型不同， Adaptive Server 会将该边界值转换为相应分区键列的数据类型，但以下情况例外：

- 不允许显式转换。以下示例尝试执行从 varchar 到 int 的非法转换。

  ```
  create table employees(emp_names varchar(20))
         partition by range(emp_name)
     (p1 values <= (1),
     p2 values <= (10))
  ```

- 不允许进行导致数据丢失的隐式转换。在以下示例中，如果 Adaptive Server 将边界值转换为 integer 值，则舍入假定可能会导致数据丢失。分区边界与分区键的数据类型不兼容。

  ```
  create table emp_id (id int)
         partition by range(id)
     (p1 values <= (10.5),
     p2 values <= (100.5))
  ```

  在以下示例中，分区边界与分区键的数据类型兼容。 Adaptive Server 直接将边界值转换为 float 值。不需要舍入，因而支持转换。

  ```
  create table id_emp (id float)
         partition by range(id)
     (p1 values <= (10),
     p2 values <= (100))
  ```

- 不允许从非二进制数据类型转换为二进制数据类型。不允许进行此转换的示例如下所示：

  ```
  create table newemp (name binary)
         partition by range(name)
     (p1 values <= ("Maarten"),
     p2 values <= ("Zymmerman"))
  ```

创建散列分区表

以下示例创建一个具有三个散列分区的表：

```
create table mysalesdetail
    (store_id char(4) not null,
    ord_num varchar(20) not null,
    title_id tid not null,
    qty smallint not null,
    discount float not null)
partition by hash (ord_num)
(p1 on seg1, p2 on seg2, p3 on seg3)
```

散列分区表很容易创建和维护。 Adaptive Server 会选择散列函数并尝试在分区之间平均分配行。

利用散列分区，可保证所有行都属于某一分区。不存在插入和更新无法找到分区的情况，而域或列表分区表可能会出现这种情况。

创建列表分区表

列表分区用于控制各行映射到特定分区的方式。列表分区没有顺序，因而对于较小的基数值很有用。每个分区值列表必须至少有一个值，而任何值都不能出现在多个列表中。

以下示例创建一个具有两个列表分区的表：

```
create table my_publishers
(pub_id char(4) not null,
pub_name varchar(40) null,
city varchar(20) null,
state char(2) null)
partition by list (state)
(west values ('CA', 'OR', 'WA') on seg1,
east value ( 'NY', 'NJ') on seg2)
```

如果尝试用 state 列中不是列表所提供的值插入行，则会失败。类似地，如果尝试用不是列表所提供的键列值更新现有行，也会失败。与域分区表一样，每个列表中的值充当对整个表的检查约束。

创建循环分区表

此分区策略不使用任何分区标准，因此是随机的。循环分区表没有分区键。

以下示例指定循环分区：

```
create table currentpublishers
(pub_id char(4) not null,
pub_name varchar(40) null,
city varchar(20) null,
state char(2) null)
partition by roundrobin 3 on (seg1)
```

所有面向分区的实用程序和管理任务都可用于循环分区表，无论是否已授权或配置语义分区。

创建分区索引

本节介绍如何使用 create index 创建分区索引。请参见 《参考手册：命令》。

可以在串行或并行模式下创建索引。在并行模式下，必须在循环分区表上创建全局索引。请参见 Performance and Tuning Series: Query Processing and Abstract Plans （《性能和调优系列：查询处理和抽象计划》）中的第 7 章 “Controlling Optimization”（控制优化）。

创建全局索引

只能为循环分区表创建全局聚簇索引。 Adaptive Server 支持所有类型分区表上的全局未分区非聚簇索引。

可以使用 Adaptive Server 版本 12.5.x 和更早版本所支持的语法，在分区表上创建聚簇和非聚簇全局索引。

创建全局索引

在分区表上创建索引时， Adaptive Server 会自动创建全局索引。如果：

- 在任意分区表上创建非聚簇索引，并且未包含 local index 关键字。例如，在散列分区表 mysalesdetail （详见第 371 页的 “创建散列分区表”）上输入：

  ```
  create nonclustered index ord_idx on mysalesdetail
  ```

- 在循环分区表上创建聚簇索引，并且未包含 local index 关键字。例如，在循环分区表 currentpublishers （详见第 372 页的 “创建循环分区表”）上输入：

  ```
  create clustered index pub_idx on currentpublishers
  ```

创建本地索引

Adaptive Server 支持所有类型分区表上的本地聚簇索引和本地非聚簇索引。本地索引继承基表的分区类型、分区列和分区边界。

对于域、散列和列表分区表，无论 create index 语句中是否包含关键字 local index， Adaptive Server 始终会创建本地聚簇索引。

以下示例在分区表 mysalesdetail （请参见第 371 页的 “创建散列分区表”）上创建本地聚簇索引。在聚簇索引中，索引行的物理顺序必须与数据行的相同；只能为每个表创建一个聚簇索引。

```
create clustered index clust_idx
    on mysalesdetail(ord_num) local index
```

以下示例在分区表 mysalesdetail 上创建本地非聚簇索引。由 title_id 对该索引进行分区。可以为每个表创建多达 249 个非聚簇索引。

```
create nonclustered index nonclust_idx
       on mysalesdetail(title_id)
   local index p1 on seg1, p2 on seg2, p3 on seg3
```

在已分区表上创建聚簇索引

如果满足下列条件，则可在分区表上创建聚簇索引：

- select into/bulkcopy/pllsort 数据库选项设置为 true，并且
- 可用的工作线程数与分区数同样多。

注释　在循环分区表上创建全局索引之前，请确保服务器配置为并行运行。

为了加快恢复过程，应在创建聚簇索引之后转储数据库。

在已分区表上创建聚簇索引之前，可与系统管理员或数据库所有者联系。

从现有表创建分区表

若要从现有表创建分区表，请使用 select into 命令。可以使用带有 into_clause 的 select 创建域、散列、列表或循环分区表。从中进行选择的表可以是分区或未分区的。请参见 《参考手册：命令》。

注释 在应用程序中，可以使用带有 into_clause 的 select 在 tempdb 中创建临时分区表。

例如，若要从 salesdetail 表创建分区表 sales_report，请输入：

```
select * into sales_report partition by range (qty)
(smallorder values <= (500) on seg1,
bigorder values <= (5000) on seg2)
from salesdetail
```

变更数据分区

可以使用 alter table 命令执行以下操作：

- 将未分区表更改为多分区表
- 向列表或域分区表添加一个或多个分区
- 对表进行重新分区以使用其它分区类型
- 对表进行重新分区以使用其它分区键或边界
- 对表进行重新分区以使用其它分区数
- 对表进行重新分区以将分区分配给其它段

请参见 《参考手册：命令》。

❖ **对表进行重新分区**

对表进行重新分区的常规步骤如下：

1 如果在重新分区过程中分区键或类型发生变化，则删除表上的所有索引。

2 使用 alter table 对表进行重新分区。

3 如果在重新分区过程中分区键或类型发生了变化，则在表上重新创建索引。

将未分区表更改为分区表

以下示例将未分区表 titles 更改为具有三个域分区的表：

```
alter table titles partition by range (total_sales)
(smallsales values <= (500) on seg1,
mediumsales values <= (5000) on seg2,
bigsales values <= (25000) on seg3)
```

向分区表添加分区

可以向列表或域分区表添加分区，但不能向散列或循环分区表添加分区。以下示例使用现有分区键列向域分区表添加一个新的分区：

```
alter table titles add partition
(vbigsales values <= (40000) on seg4)
```

注释　只能将分区添加到现有基于域的分区的高端。如果已在分区上定义了 values >= (MAX)，则不能添加新分区。

向列表或域分区表添加分区时并不复制数据。新创建的分区是空的。

更改分区类型

注释　在更改分区类型之前，必须先删除所有索引。

在第 375 页的 “向分区表添加分区”中，titles 表是按域进行分区的。以下示例中，在 title_id 列上按散列对 titles 进行了重新分区：

```
alter table titles partition by hash(title_id)
    3 on (seg1, seg2, seg3)
```

可以再次在 total_sales 列上按域对 titles 进行重新分区。

```
alter table titles partition
       by range (total_sales)
    (smallsales values <= (500) on seg1,
    mediumsales values <= (5000) on seg2,
    bigsales values <= (25000) on seg3)
```

更改分区键

注释 在更改分区键之前，必须先删除所有索引。

在第 375 页的 “更改分区类型”中， titles 表在 total_sales 列上按域进行重新分区。以下示例更改分区键，但不更改分区类型。

```
alter table titles partition by range(pubdate)
    (q1 values <= ("3/31/2006"),
    q2 values <= ("6/30/2006"),
    q3 values <= ("9/30/2006"),
    q1 values <= ("12/31/2006"))
```

取消对循环分区表的分区

可以使用带有 unpartition 子句的 alter table 根据分区的循环表创建未分区的循环表，但条件是所有分区都位于同一段上，并且表上没有索引。在将大量数据装载到最终用作未分区表的表中时，此功能十分有用。请参见第 381 页的 “使用分区装载表数据”。

使用 *partition* 参数

可以使用 partition *number_of_partitions* 参数将未分区的循环表更改为具有指定数量分区的循环分区表。 Adaptive Server 会将现有的全部数据放在第一个分区中。其余的分区在创建时为空，它们与第一个已有分区位于同一段上。随后插入的数据会根据循环策略在所有分区之间分配。

如果初始分区上有本地索引， Adaptive Server 会在新分区上创建空本地索引。如果在创建表时声明了段， Adaptive Server 会将新分区放在该段上；否则，会将分区置于在表和索引级别指定的缺省段上。

例如，可以对 pubs2 中的 discounts 表使用 partition *number_of_partitions* 以创建三个循环分区：

```
alter table discounts partition 3
```

注释 仅支持使用带有 partition 子句的 alter table 创建循环分区，不支持创建其它类型的分区。

变更分区键列

修改分区键列时要小心，须遵守以下这些规则：

- 不能删除属于分区键的列。不属于分区键的列可以删除。
- 如果为域分区表更改属于分区键的列的数据类型，则该分区的边界将转换为新的数据类型，但以下情况例外：
  - 显式转换
  - 导致数据丢失的隐式转换。
  - 非二进制数据类型向二进制数据类型的转换

  有关不受支持的转换的详细信息和示例，请参见第 370 页的 “对域分区表的分区键和边界值的限制”。

在某些情况下，如果修改分区键列的数据类型，则可能会在分区之间重新分配数据：

- 对于域分区 — 如果某些分区键值接近于分区边界，则数据类型转换可能会导致这些行迁移到其它分区。

  例如，假设分区键的原有数据类型为 float，并且该数据类型转换为 integer。分区边界是：`p1 values <= (5), p2 values <= (10)`。结果，某一行的分区键从 5.5 转换为 5，并且该行从 p2 迁移到了 p1。
- 对于域分区 — 如果排序顺序因分区键数据类型更改而更改，则所有数据行都将根据新的排序顺序进行重新分区。例如，如果分区键数据类型从 varchar 更改为 datetime，则排序顺序也会更改。

  如果尝试变更分区键列的数据类型，则 alter table 会失败，并且在转换后，新的边界不会使所需的升序顺序保持不变，或者有些行不适合新的分区。

  有关详细信息，请参见 《系统管理指南：卷 1》的第 9 章 “配置字符集、排序顺序和语言”中的 “处理可疑分区”。
- 对于散列分区 — 分区键数据类型的数据值和存储大小均用于生成散列值。因此，更改散列分区键的数据类型可能会导致重新分配数据。

配置分区

为了提高性能，可以对分区进行配置。分区的配置参数包括：

- number of open partitions — 指定 Adaptive Server 一次可访问的分区数。缺省值为 500。
- partition spinlock ratio — 指定用于防止并发访问打开的分区的螺旋锁数。缺省值为 10。

请参见 《系统管理指南：卷 1》中的第 5 章 “设置配置参数”。

在分区表中执行更新、删除和插入

在分区表中，可以完全像在未分区表中那样更新、插入和删除数据，且使用的语法相同。但不能在 update、 insert 和 delete 语句中指定分区。

在分区表中，数据驻留在分区中，表在逻辑上联合了各个分区。特定数据行具体存储在哪个分区对用户是透明的。 Adaptive Server 通过内部逻辑和表的分区策略组合来确定要访问哪些分区。

对于尝试插入不符合任何表分区的行的任何事务， Adaptive Server 都会将其中止。在循环或散列分区表中，所有行都符合分区。在域或列表分区表中，只有满足分区标准的行才符合分区。

- 对于域分区表 — 如果在插入数据行时，其值超过为表定义的范围的上限，则该操作会中止，除非指定了 MAX 范围。如果指定了 MAX 范围，则所有行的上限均符合分区。
- 对于列表分区表 — 如果在插入数据行时，其分区列值不符合分区标准，则该操作会失败。

如果更新数据行的分区键列时，其键列值不再满足任何分区的分区标准，则更新会中止。有关详细信息，请参见第 379 页的 “更新分区键列中的值”。

更新分区键列中的值

对于语义分区表，更新分区键列中的值可能导致数据行从一个分区移至另一分区。

如果数据行必须移至其它分区，则 Adaptive Serverey 会以延迟模式更新分区键列。延迟更新分为两步：首先从原始分区中删除该行，然后再将其插入新分区。

对仅数据锁定 (DOL) 表执行此类操作会导致行 ID (RID) 发生更改，从而导致扫描异常。例如，可以创建一个表并在列 a 上按域对该表进行分区：

```
create table test_table (a int) partition by range (a)
    (partition1 <= (1),
    partition2 <= (10))
```

该表在 partition2 中具有单独的一行。分区键列值为 2。 partition1 为空。假定有以下语法：

```
Transaction T1:
      begin tran
    go
    update table set a = 0
    go

Transaction T2:
    select count(*) from table isolation level 1
    go
```

更新 T1 会导致从 partition2 中删除该行，并将其插入 partition1 中。但此时不会提交 delete 和 insert。因此， T2 中的 select count(*) 不会阻塞 partition1 中未提交的 insert，而是会阻塞 partition2 中未提交的 delete。如果提交了 T1，则 T2 不会看到已提交的 delete 并返回计数值零 (0)。

在未分区的 DOL 表上，可以在 inserts 和 deletes 中看到此行为。仅在更新分区键值以便行从一个分区移至另一分区时， updates 才出现此行为。请参见 Performance and Tuning Series: Physical Database Tuning（《性能和调优系列：物理数据库调优》）中的第 1 章 “Controlling Physical Data Placement”（控制物理数据放置），和 Performance and Tuning Series: Locking and Concurrency Control （《性能和调优系列：锁定和并发控制》）中的第 5 章 “Indexes”（索引）。

显示有关分区的信息

使用 sp_helpartition 可查看有关分区的信息。例如，若要查看有关 publishers 中 p1 分区的信息，请输入：

```
sp_helpartition publishers, null, p1
```

请参见 《参考手册：过程》。

使用函数

可以使用几种函数显示分区信息。请参见 《参考手册：构件块》，了解完整的语法和用法信息。

- data_pages — 返回表、索引或分区所使用的页数。
- reserved_pages — 返回为表、索引或分区保留的页数。
- row_count — 估计表或分区中的行数。
- used_pages — 返回表、索引或分区所使用的页数。与 data_pages 不同， used_pages 包含内部结构所使用的页数。
- partition_id — 返回指定索引的指定分区的分区 ID。
- partition_name — 返回与指定索引和分区 ID 对应的分区名。

示例

以下示例返回指定数据库中 ID 为 31000114 的对象所使用的页数。这些页数包含索引页数。

```
data_pages(5, 31000114)
```

以下示例返回 testtable_ptn1 分区的相应分区 ID。

```
select partition_id( 搕 esttableî, testtable_ptn1î)
```

以下示例返回分区 ID 1111111111 （属于索引 ID 为 0 的基表）的相应分区名称。

```
select partition_name(0, 1111111111)
```

截断分区

可以删除分区中的所有信息，而不会影响其它分区中的信息。例如，若要从 fictionsales 表的 q1 和 q2 分区中删除所有行，请输入：

```
truncate table fictionsales partition q1
truncate table fictionsales partition q2
```

请参见 《参考手册：命令》。

使用分区装载表数据

即使表最终将用作未分区的表，您也可以使用分区来加快装载大量表数据的过程。

使用循环分区方法，并将所有分区放在同一段上。

步骤如下：

1　创建一个空表，并将该表分为 *n* 个分区。例如，输入：

```
create table currentpublishers
(pub_id char(4) not null,
pub_name varchar(40) null,
city varchar(20) null,
state char(2) null)
partition by roundrobin 3 on (seg1)
```

2　使用 *partition_id* 选项运行 bcp in。将预排序数据复制到每个分区。例如，若要将 *datafile1.dat* 复制到 currentpublishers 的第一个分区，请输入：

```
bcp pubs2..currentpublishers:1 in datafile1.dat
```

3　使用 alter table unpartition 取消对该表的分区。例如，输入：

```
alter table currentpublishers unpartition
```

4　使用带有以下子句的 created clustered index 创建一个聚簇索引：with sorted_data。例如，输入：

```
create clustered index pubnameind
    on currentpublishers(pub_name)
    with sorted_data
```

创建分区时，Adaptive Server 会在 syspartitions 表中为每个分区放置一个条目。带有 *partition_id* 选项的 bcp in 会按 syspartitions 中列出的顺序将数据装载到每个分区。在创建聚簇索引之前取消对该表的分区可使此顺序保持不变。

更新分区统计

Adaptive Server 查询处理器在查询中使用有关表、索引、分区和列的统计信息估计查询开销。查询处理器会选择它确定的开销最小的访问方法。但要实现此目的，它必须具有精确的统计信息。

有些统计信息在查询期间更新，有些则只在运行 update statistics 命令或创建索引时更新。

update statistics 通过为分区的本地索引的每个主要属性创建直方图并为组合属性创建密度，可帮助 Adaptive Server 制定最佳决策。在分区表中添加、更改或删除大量数据后，可使用 update statistics。

发出 update statistics 和 delete statistics 的权限缺省情况下授予表的所有者，并且不可移交。使用 update statistics 命令可以更新各数据和索引分区的统计信息。产生分区相关信息的 update statistics 命令有：

- update statistics
- update table statistics
- update all statistics
- update index statistics
- delete statistics

例如，若要更新 titles 表（在第 375 页的 “将未分区表更改为分区表”中创建）的 smallvalues 分区的统计信息，请输入：

```
update statistics titles partition smallvalues
```

请参见《参考手册：命令》。

第 11 章

虚拟散列表

注释 虚拟散列表只能在 Linux pSeries 中使用。

可以使用非聚簇索引或聚簇索引对仅数据锁定表执行基于散列的索引扫描。在此类扫描期间，各工作进程会浏览索引的较高级别并读取索引的叶级页。然后，每个工作进程在单独散列表中按数据页 ID 或键值散列，以确定要处理的数据页或数据行。

通过虚拟散列表可以更高效地组织表，因为它不需要单独的散列表。虚拟散列表可存储行，以便查询处理器可以使用散列键确定行 ID （基于行的序号）以及数据的位置。因为它不使用单独的散列表存储信息，因此称其为 “虚拟”散列表。

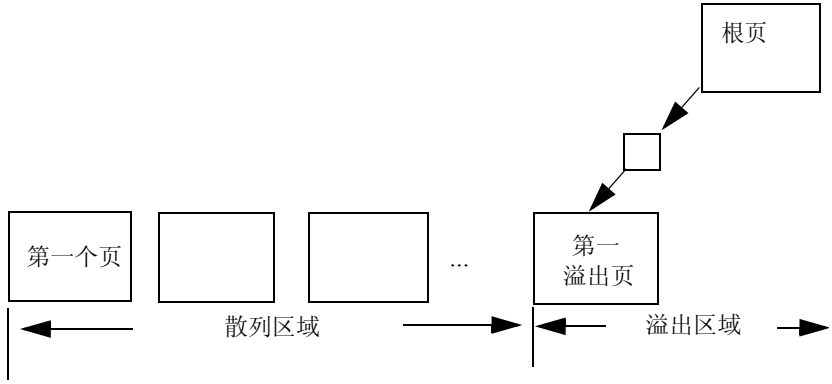
对于必须更有效地使用其中央处理单元 (CPU) 的系统，虚拟散列表是一个不错的选择。

对用于查找的表或行位置不发生变化的表使用聚簇索引或非聚簇索引会占用大量系统资源。在级别 2 和级别 3 （L2 和 L3） CPU 体系结构得到最新改进后，必须利用高速缓存才能利用实际 CPU 计算能力。如果不利用高速缓存，则 CPU 会浪费不必要的周期来等待可用内存。对于聚簇或非聚簇索引，服务器每次执行索引级搜索时都会错过一些行，这会消耗很多 CPU 周期。虚拟散列表可通过计算散列键值而不是执行搜索来访问行位置模式。

虚拟散列表的结构

虚拟散列表包含两个区域：“散列”区域和 “溢出”区域。散列区域存储散列行，溢出区域存储剩余行。可以使用 B 树聚簇索引，通过常规聚簇索引扫描访问溢出区域。

图 11-1：使用根页访问溢出区域



虚拟散列表的第一个数据页、根页和第一个溢出页在您创建表时创建。SYSINDEXES.indroot 是溢出聚簇区域的根页。此页下的第一个叶页是第一个溢出页。 SYSINDEXES.indfirst 指向第一个数据页，因此表扫表在表的起始位置开始并扫描整个表。

创建虚拟散列表

要创建虚拟散列表，请指定散列区域的最大值。下面是 create table 的部分语法；新参数以粗体显示：

```
create table [database.[owner].]table_name
. . .
        | {unique | primary key}

using
 clustered
        (column_name [asc | desc] [{, column_name [asc | desc]}...])=

(hash_factor [{, hash_factor}...])
                with max num_hash_values key
```

其中：

- using clustered — 指示您要创建虚拟散列表。列列表将被视为此表的键列。
- *column_name* [asc | desc] — 由于行是基于其散列函数来放置的，因此不能将 [asc | desc] 用于散列区域。如果提供虚拟散列表的键列顺序，该顺序将只用于溢出聚簇区域。
- *hash_factor* — 用于虚拟散列表的散列函数所必需的参数。对于散列函数，每个键列都需要一个散列因子。这些因子与键值结合使用，以便为特定行生成散列值。
- with max *num_hash_values* key — 可以使用的最大散列值数。定义此散列函数输出的上限。

注释　虚拟散列表具有以下限制：

- 虚拟散列表的行必须唯一。虚拟散列表不允许多个行具有相同的键列值，因为 Adaptive Server 不能将某行保留在散列区域中，而将具有相同键列值的另一行保留在溢出聚簇区域中。
- 必须在不能存储任何其它对象 （如表或列）的排它段上创建每个虚拟散列表。

确定 *hash_factor* 的值

可将第一个键的散列因子保持为 1。其余所有键列的散列因子大于散列区域中允许的前一个键与其散列因子之乘积的最大值。

对于第一个键列的散列因子大于 1 的表， Adaptive Server 允许其页上具有较少的行。例如，如果表的第一个键列的散列因子为 5，则在页上的每一行之后，留给后四行的空间将保持为空。为了支持此功能，Adaptive Server 所需空间量为表空间的五倍。

如果键列的值大于或等于下一个键列的散列因子，会将当前行插入溢出聚簇区域，以避免散列区域中出现冲突。

例如， t 是具有键列 id 和 age 并且对应的散列因子为 (10,1) 的虚拟散列表。由于行 (5, 5) 和 (2, 35) 的散列值为 55，因此可能导致散列冲突。

但是，由于值 35 大于等于 10 （下一个键列 id 的散列因子），因此 Adaptive Server 会将第二行存储在溢出聚簇区域中，以避免散列区域中出现冲突。

在另一个示例中，如果 u 是主索引和散列因子为 (id1, id2, id3) = (125, 25, 5) 并且 *max hash_value* 为 200 的虚拟散列表：

- 行 (1,1,1) 的散列值为 155，因此会将该行存储在散列区域中。

- 行 (2,0,0) 的散列值为 250，因此会将该行存储在溢出聚簇区域中。
- 行 (0,0,6) 的散列因子为 6 x 5，该值大于等于 25，因此会将该行存储在溢出聚簇区域中。
- 行 (0,7,0) 的散列因子为 7 x 25，该值大于等于 125，因此会将该行存储在溢出聚簇区域中。

示例

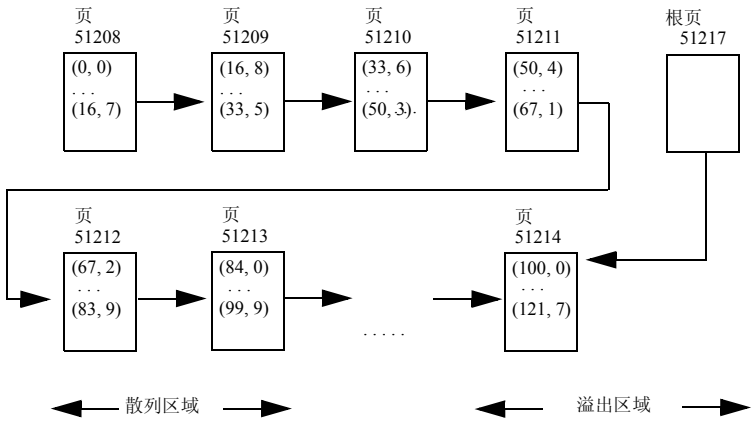
以下示例在 pubs2 数据库的 order_seg 段中创建名为 orders 的虚拟散列表：

```
create table orders(
id int,
age int,
primary key using clustered (id,age) = (10,1) with max
1000 key)
on order_seg
```

数据布局为：

- order_seg 段开始于 ID 为 51200 的页。
- 第一个数据对象分配映射 (OAM) 页的 ID 为 51201。
- 逻辑页大小为 2048 字节
- 如果逻辑页大小为 2048 字节，则每页的最大行数为 168。
- 行宽为 10。
- 溢出聚簇区域的根索引页为 51217。

图 11-2：数据的页布局



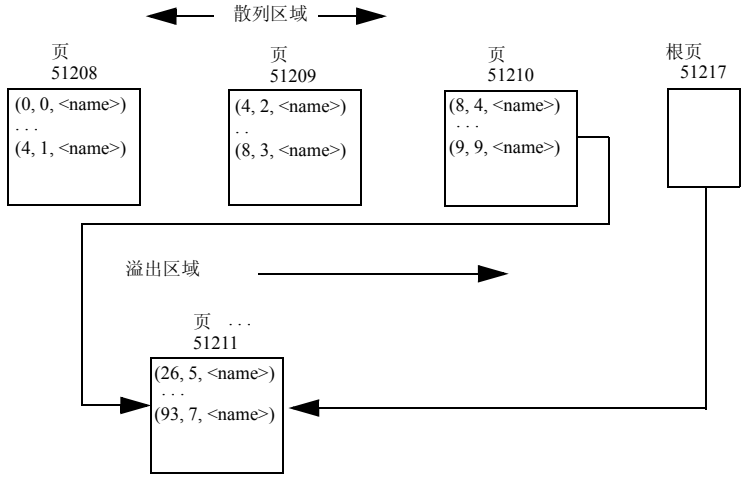
以下示例在 pubs2 数据库的 order_seg 段中创建名为 orders 的虚拟散列表：

```
create table orders(
id int default NULL,
age int,
primary key using clustered (id,age) = (10,1) with max
100 key,
name varchar(30)
)
on order_seg
```

数据布局为：

- order_seg 段开始于 ID 为 51200 的页。
- 逻辑页大小为 2048 字节
- 第一个数据 OAM 页的 ID 为 51201。
- 每页的最大行数为 42。
- 如果逻辑页大小为 2048 字节，则行宽为 45。
- 溢出聚簇区域的根索引页为 51217。

图 11-3：数据的页布局



对虚拟散列表的限制

下面是对虚拟散列表的限制：

- 不支持 truncate table。应改用 delete from table *table_name*。
- SQL92 不允许一个关系中的两个唯一约束具有相同的键列，因此 Adaptive Server 不支持对相同的键列 （用作虚拟散列表的键列）应用主键或唯一键约束。
- 由于在创建表之后无法创建虚拟散列聚簇索引，因此也无法删除虚拟散列聚簇索引。
- 必须在排它段上创建虚拟散列表。不能将指派给段用于创建虚拟散列表的磁盘设备与其它段共享。换句话说，必须首先创建特殊设备，然后在该设备上创建排它段。
- 不能在同一排它段上创建两个虚拟散列表。 Adaptive Server 支持每个数据库具有 32 个不同的段。缺省段、系统段和日志段这三个段为保留段，所以每个数据库的最大虚拟散列表数为 29 个。
- 不能对虚拟散列表使用 alter table 或 drop clustered index 命令。
- 虚拟散列表必须使用所有页锁定 (APL)。
- 虚拟散列表的键列和散列因子必须使用 int 数据类型。
- 虚拟散列表中不能包括 text 或 image 列，也不能包括数据类型基于 text 或 image 数据类型的列。
- 不能创建分区的虚拟散列表。

不能创建以下类型的虚拟散列表：

- 经常执行插入和更新操作
- 是分区表
- 频繁使用表扫描
- 溢出区域中的数据行比散列区域中的数据行多。在这种情况下更适合使用 B 树，而不是虚拟散列表。

命令更改

第 384 页的 “创建虚拟散列表”中介绍了对 create table 命令的更改。

- dbcc checktable — 除了执行常规检查外， checktable 还会验证散列区域中数据和 OAM 页的布局是否正确。
  - 不在为每个布局的 OAM 页保留的扩充中分配数据页。
  - 只在分配单元 (AU) 的第一个扩充中分配 OAM 页。
- dbcc checkstorage — 如果用于非散列表的除第一个数据页之外的任何数据页为空，则报告软故障。然而， dbcc checkstorage 不为虚拟散列表的散列区域报告此软故障。虚拟散列表的散列区域中的任何数据页均可为空。

查询处理器更改

仅在包含的用于所有键列的搜索参数包含等号限定符 （例如， `where id=2`）时，查询处理器才使用虚拟散列索引。如果查询处理器使用虚拟散列索引，则它在 showplan 输出中包含类似以下内容的行：

```
Using Virtually Hashed Index.
```

如果查询处理器选择虚拟散列索引，则它在索引选择输出中包含类似以下内容的行：

```
Unique virtually hashed index found, returns 1 row, 1
pages
```

监控计数器更改

为虚拟散列表增加了以下监控计数器：

- am_srch_hashindex — 计算使用虚拟散列聚簇索引执行搜索的次数

系统过程更改

以下是为了支持虚拟散列表而对系统过程所做的更改：

- sp_addsegment — 不能在已经具有排它段的设备上创建段。如果试图这样做，则会显示类似以下内容的错误消息：

  ```
  A segment with a virtually hashed table exists on
  device orders_dat.
  ```

- sp_extendsegment — 不能在已经具有排它段的设备上扩展段，并且不能在已经具有其它段的设备上扩展排它段。

  例如，如果试图在 *orders.dat* 设备上扩展段 orders_seg，但是该设备上已经具有排它段，则会显示类似以下内容的错误消息：

  ```
  A segment with a virtually hashed table exists on
  device orders.dat.
  ```

  如果试图在 *orders.dat* 设备上扩展排它段 orders_seg，但是该设备上已经具有其它段，则会显示类似以下内容的错误消息：

  ```
  You cannot extend a segment with a virtually hashed
  table on device orders.dat, because this device has
  other segments.
  ```

- sp_placeobject — 不能对虚拟散列表使用 sp_placeobject，并且不能在排它段上放置其它对象。例如，如果尝试在段上放置名为 orders 的虚拟散列表，则会显示类似以下内容的错误消息：

  ```
  sp_placeobject is not allowed for orders, as it is a
  virtually hashed table.
  ```

  如果尝试在包含虚拟散列表的 myseg 段上放置对象 t，则会显示类似以下内容的错误消息：

  ```
  Segment myseg has a virtually hashed table;
  therefore, you cannot place an object on this
  segment.
  ```

- sp_chgattribute — 不允许更改虚拟散列表的属性。例如，如果试图按照如下方式更改表 order_line （它是虚拟散列表）的属性：

  ```
  sp_chgattribute 'order_line', 'exp_row_size', 1
  ```

  Adaptive Server 将发出类似以下内容的错误消息：

  ```
  sp_chgattribute is not allowed for order_line, as it
  is a virtually hashed table.
  ```

- sp_help — 对于虚拟散列表：
  - 使用下面的消息报告表是虚拟散列表：

    ```
    Object is Virtually Hashed
    ```

  - 使用以下语法以消息形式报告表的 hash_key_factors：

    ```
    column_1:hash_factor_1,
    column_2:hash_factor_2...,
    max_hash_key=max_hash_value
    ```

    例如：

```
attribute_class         attribute       int_value
char_value                              comments
--------------------    ---------------  ------------------------
---------------------------------------    -----------
misc table info         hash key factors                    NULL
id:10.0, id2:1.0, max_hash_key=1000.0           NULL
```

第 12 章

视图：限制访问数据

视图是作为对象存储在数据库中的命名 select 语句。通过视图可以显示一个或多个表中行或列的子集。可以通过在其它 Transact-SQL 语句中调用视图的名称来使用相应的视图。在特定数据库中，可以使用视图来关注、简化和自定义每个用户查看表的效果。通过只允许用户访问他们所需要的数据，视图还提供了一种安全性机制。

视图的工作方式

视图是一种查看一个或多个表中数据的替代方法。

例如，假定您正在进行一个关于 Utah 州的项目。可以创建一个视图，在其中仅列出居住在 Utah 州的作者：

```
create view authors_ut
as select * from authors
where state = "UT"
```

要显示 authors_ut 视图，请输入：

```
select * from authors_ut
```

当向 authors 表中添加或从中删除居住在 Utah 州的作者时，authors_ut 视图就会反映出更新后的 authors 表。

视图派生自一个或多个真实的表，而这些表的数据物理存储在数据库中。从中派生视图的表叫做基表或基础表。视图也可以从另一视图派生。

视图的定义以从中派生它的基表的方式存储在数据库中。没有单独的数据副本与这种存储的定义相关联。您所看到的数据存储在基础表中。

视图看起来与数据库中的任何其它表完全一样。可以用显示和操作任何其它表的方法来显示和操作视图。在通过视图进行查询方面没有任何限制，在修改视图方面的限制也比较少。本章后面部分对例外情况进行了说明。

当通过视图修改数据时，实际上是在更改底层基表中的数据。反之，底层基表中数据的改变会自动地在其派生的视图中反映出来。

视图的优点

可以使用视图来关注、简化和自定义每个用户查看数据库的效果；视图还提供了一种便于使用的安全措施。对数据库的结构进行了更改，但用户都愿意使用他们已习惯的数据库结构时，视图也非常有用。

可以使用视图来：

- 关注每个用户感兴趣的数据和该用户所承担的任务。用户不感兴趣的数据可以从视图中省略。
- 将常用的连接、投影和选择定义为视图，这样在每次操作该数据时，用户就不需要指定所有的条件和限制。
- 为不同的用户显示不同的数据 （即使他们同时使用同一数据）。这一优点在不同兴趣、不同技术水平的用户共享相同数据库时尤为重要。

安全性

通过视图，用户只能查询和修改他们所能看到的数据。数据库的其余部分既不可见也不可访问。

使用 grant 和 revoke 命令，可将每个用户对数据库的访问限制在指定的数据库对象 （包括视图）上。如果视图、所有表以及从中派生该视图的其它视图属于同一用户，则该用户可以授予其他用户使用该视图的权限，而禁止他们使用该视图的基础表和基础视图。这是一个简单而有效的安全机制。请参见 《系统管理指南：卷 1》中的第 17 章 “管理用户权限”。

通过定义不同的视图并有选择地授予对它们的权限，可以将用户限制到不同的数据子集上。例如：

- 可以将访问限制到基表中行的子集，即与值相关的子集。例如，可以定义只包含商业与心理学书籍的行的视图，使某些用户无法看到有关其它类型的书籍的信息。
- 可以将访问限制到基表中列的子集，即与值不相关的子集。例如，可以定义包含 titles 表中的所有行但不包含 royalty 和 advance 列的视图。
- 可以将访问限制到由基表中的部分行与列组成的子集。
- 可以将访问限制到能连接多个基表的行。例如，可以定义一个连接 titles、 authors 和 titleauthor 的视图以显示作者的姓名和他们编写的图书。不过，此视图将隐藏作者的个人信息以及图书的价格信息。
- 可以将访问限制到基表中数据的统计信息。例如，通过 category_price 视图，用户只能访问各类图书的平均价格。
- 可以将访问限制到另一个视图的子集或视图与基表的组合上。例如，通过 hiprice_computer 视图，用户可以访问满足 hiprice 视图定义中的限定的计算机图书的书名与价格。

要创建视图，用户必须从数据库所有者处获得 create view 权限，并且必须对视图定义中引用的任何表或视图拥有适当的权限。

如果视图引用不同数据库中的对象，视图的用户必须是各个数据库的有效用户或访客。

如果其他用户在您拥有的某个对象上创建了视图，则必须了解谁能通过哪些视图看到哪些数据。例如：数据库所有者已经授予 “haroldîcreate view 权限，而 “maude”已授予 “harold”在她拥有的表中进行 select 的权限。有了这些权限后，“harold”可以创建可从 “maude”拥有的表中选择所有列和行的视图。如果 “maude”撤消 “harold”从其表中执行 select 操作的权限，则 “harold”仍然可以通过其已经创建的视图查看 “maude”的数据。

逻辑数据独立性

如果需要对真实表的结构进行更改，视图可以使用户不受这种更改的影响。

例如，假定您使用 select into 将 titles 表拆分为如下两个新基表，然后删除 titles 表，以调整数据库的结构：

```
titletext (title_id, title, type, notes)
titlenumbers (title_id, pub_id, price, advance,
```

```
    royalty, total_sales, pub_date)
```

通过连接两个新表的 title_id 列，可以 “重新生成”旧的 titles 表。可以创建一个视图来连接这两个新表。甚至可以将其命名为 titles。

以前引用基表 titles 的任何查询或存储过程现在则引用视图 titles。对于用户来说， select 操作与以前完全相同。只在新视图中进行检索的用户甚至不需要知道数据库的结构已经改变。

遗憾的是，视图只提供部分逻辑独立性。由于某些限制，在新的 titles 上不能执行某些数据修改语句。

视图示例

第一个示例是从 titles 表中派生的视图。假定您只对价格高于 $15 并且预付款超过 $5000 的图书感兴趣。以下这个简单的 select 语句将查找符合这些条件的行：

```
select *
from titles
where price > $15
  and advance > $5000
```

现在，假定您要对这组数据执行大量的检索和更新操作。当然可以将在先前查询中所显示的条件与要发出的任何命令进行组合。不过，为了方便起见，可以创建一个只显示感兴趣的记录的视图：

```
create view hiprice
as select *
from titles
where price > $15
  and advance > $5000
```

Adaptive Server 接收到此命令后，实际并不执行关键字 as 后面的 select 语句。而是将 select 语句 （视图 hiprice 的定义）存储在系统表 syscomments 中。还会在 sysobjects 和 syscolumns 中为视图所包含的每一列建立条目。

现在，当显示 hiprice 或对其进行操作时， Adaptive Server 会将该语句与存储的 hiprice 的定义相组合。例如，可像更改任何其它表一样更改 hiprice 中的所有价格：

```
update hiprice
set price = price * 2
```

Adaptive Server 将在系统表中查找视图定义，并将 update 命令转换为以下语句：

```
update titles
set price = price * 2
where price > $15
  and advance > $5000
```

也就是说， Adaptive Server 从视图定义中得知要更新的数据在 titles 中。它还知道只应在符合条件 （指在视图定义中给出的 price 和 advance 列上的条件和更新语句中的条件）的行中增加价格。

发出更新 hiprice 的命令后，可以在视图或 titles 表中看到其结果。反之，如果已创建了视图并发出了第二个 update 语句 （此语句直接对基表进行操作），则通过该视图也能看到更改后的价格。

如果用不同的行都符合视图条件的这种方式来更新视图的基础表，就会影响此视图。例如，假定要想将 *You Can Combat Computer Stress* 一书的价格增加到 $25.95。由于此书符合视图定义语句中的限定条件，因此会将其视为视图的一部分。

但是，如果通过添加列来更改视图基础表的结构，新列将*不会*出现在用 select * 子句定义的视图中，除非删除并重新定义该视图。这是因为原视图定义中的星号只考虑原有的列。

创建视图

对于每个用户来说，视图名称在已经存在的表和视图中必须唯一。如果设置了 set quoted_identifier on，就可以在视图名称上使用分隔标识符。否则，视图名称必须符合第 7 页的 “标识符”中给出的标识符规则。

可以在其它视图和引用视图的过程上建立视图。可以在视图上定义主键、外键和公用键。但不能将规则、缺省值或触发器与视图相关联，也不能在视图上建立索引。不能在临时表上创建临时视图或视图。

create view 语法

如第 396 页的 “视图示例”中的 create view 示例所述，不需要在视图定义语句的 create 子句中指定任何列名。 Adaptive Server 为视图的列提供了与 select 语句的选择列表中所引用的列相同的名称和数据类型。可以用星号 (*) 指定选择列表，如示例所示，也可以是基表中全部或部分列名的列表。请参见 《参考手册：命令》。

要建立不含重复行的视图，可以使用 select 语句的 distinct 关键字来确保视图中的每一行都唯一。但是，无法更新 distinct 视图。

随时都可以指定列名。但是，如果出现下列任何情况，则*必须*在 create 子句中为视图中的*每*一列指定列名：

- 视图中的一列或多列是从算术表达式、集合、内置函数或常量派生的。
- 两个或多个视图列拥有相同的名称。此现象经常出现，因为视图的定义包含连接，而被连接的列具有相同的名称。
- 要为视图中的列指定一个与派生该列的列不同的名称。

  也可以在 select 语句中对列进行重命名。无论是否对视图列进行重命名，它都继承从中派生它的列的数据类型。

下面是一个视图定义语句，它使视图中的列名不同于它在基础表中的名称：

```
create view pub_view1 (Publisher, City, State)
as select pub_name, city, state
from publishers
```

下面是另一种创建相同的视图但在 select 语句中对列进行重命名的方法：

```
create view pub_view2
as select Publisher = pub_name,
City = city, State = state
from publishers
```

下一节中的视图定义语句示例说明了将列名包含在 create 子句中的其余规则。

注释 不能在视图定义中使用局部变量。

将 *select* 语句和 *create view* 一起使用

create view 语句中的 select 语句用于定义视图。在所创建的视图的 select 语句中，必须拥有对任何引用的对象进行 select 操作的权限。

可以通过任何复杂的 select 语句用多个表和其它视图创建视图。

对视图定义中的 select 语句有一些限制：

- 不能包含 order by 或 compute 子句。
- 不能包含 into 关键字。
- 不能引用临时表。

带有投影的视图定义

要创建一个包含 titles 表中所有行但只含列的一个子集的视图，请输入：

```
create view titles_view
as select title, type, price, pubdate
from titles
```

create view 子句中不包含任何列名。视图 titles_view 继承了选择列表中给定的列名。

带有计算列的视图定义

下面是一个视图定义语句，该语句使用从 price、 royalty 和 total_sales 列生成的计算列创建视图：

```
create view accounts (title, advance, amt_due)
as select titles.title_id, advance,
(price * royalty/100) * total_sales
from titles, roysched
where price > $15
and advance > $5000
and titles.title_id = roysched.title_id
and total_sales between lorange and hirange
```

因为通过将 price、 royalty 和 total_sales 相乘得到的列没有可继承的名称，所以必须将列的列表包含在 create 子句中。计算列命名为 amt_due。计算列在 create 子句中的位置必须与计算该列时所用的表达式在 select 子句中的位置相同。

带有集合或内置函数的视图定义

包含集合或内置函数的视图定义必须将列名包含在 create 子句中。例如：

```
create view categories1 (category, average_price)
as select type, avg(price)
from titles
group by type
```

当出于安全考虑而创建视图时，应谨慎使用集合函数和 group by 子句。对于可以包含在带有 group by 的 select 中的列，由于 Transact-SQL 扩展不对其进行限制，因此可能会导致视图返回的信息多于所需要的信息。例如：

```
create view categories2 (category, average_price)
as select type, avg(price)
from titles
where type = "business"
```

在上述示例中，您可能想用视图将其结果限定为 “business” 类别，但结果中具有其它类别的信息。有关 group by 和此 group by Transact-SQL 扩展的详细信息，请参见第 81 页的 “将查询结果分组：group by 子句”。

带有连接的视图定义

可以创建从多个基表派生的视图。下面是从 authors 和 publishers 表派生的视图的示例。此视图包含与出版社处于同一城市的作者的姓名和所在的城市，以及每个出版社的名称和所在城市。

```
create view cities (authorname, acity, publishername,
pcity)
as select au_lname, authors.city, pub_name,
publishers.city
from authors, publishers
where authors.city = publishers.city
```

用于外连接的视图

如果定义带有外连接的视图，然后使用外连接的内部表中列的限定条件查询该视图，则查询的行为表现得就像该限定条件是视图的 WHERE 子句的一部分，而不是视图中外连接的 ON 子句的一部分一样。因此，该限定条件只适用于外连接完成后的行。例如，如果满足外连接条件，则限定条件适用于 NULL 扩展行，并相应地消除行。

下列规则确定了可通过连接视图对列进行更新的类型：

- 连接视图中不允许使用 delete 语句。
- 在用 with check option 创建的连接视图中不允许使用 insert 语句。
- 在用 with check option 创建的连接视图中，允许使用 update 语句。如果任何受影响的列出现在 where 子句中，或者出现在包含来自多个表的列的表达式中，更新将会失败。
- 如果通过连接视图来插入或更新行，所有受影响的列都必须属于同一基表。

从其它视图派生的视图

可以根据其它视图来定义视图，如下面的示例所示：

```
create view hiprice_computer
as select title, price
from hiprice
where type = "popular_comp"
```

distinct 视图

可以确保视图中包含的行是唯一的，如下面的示例所示：

```
create view author_codes
as select distinct au_id
from titleauthor
```

如果某行的所有列值都与另一行包含的相同列值相匹配，则该行就是另一行的重复行。两个空值被认为是相同的。

Adaptive Server 在第一次访问视图时，会将 distinct 要求应用到视图定义中，然后再进行任何投影或选择。视图的外观和操作与任何数据库表类似。如果选择 distinct 视图的投影（即只选择视图的某些列但选择其所有行），则得到的结果似乎重复。但视图自身内的每行仍然唯一。例如，假定创建了一个 distinct 视图 myview，它包含三个列，即 a、b 和 c，其中包含下列值：

a	b	c
1	1	2
1	2	3
1	1	0

输入此查询后：

```
select a, b from myview
```

结果如下所示：

```
a    b
---  ---
1    1
1    2
1    1

(3 rows affected)
```

第一行与第三行似乎重复。但基础视图行仍然唯一。

包含 IDENTITY 列的视图

可以通过在视图的 select 语句中列出列名 （或 syb_identity 关键字），定义包含 IDENTITY 列的视图。例如：

```
create view sales_view
as select syb_identity, stor_id
from sales_daily
```

但是，不能使用 *identity_column_name* = identity(*precision*) 语法向视图中添加新的 IDENTITY 列。

可以使用 syb_identity 关键字选择视图中的 IDENTITY 列，除非视图：

- 多次选择 IDENTITY 列
- 通过 IDENTITY 列计算新列
- 包含一个集合函数
- 连接多个表中的列
- 将 IDENTITY 列作为表达式的一部分包含

在上述任何情况下， Adaptive Server 不将该列视为视图的 IDENTITY 列。在对视图执行 sp_help 时，此列显示的 “Identity” 值为 0。

在下面的示例中， row_id 列不被认为是 store_discounts 视图的 IDENTITY 列，因为 store_discounts 连接两个表中的列：

```
create view store_discounts
as
select stor_name, discount
from stores, new_discounts
where stores.stor_id = new_discounts.stor_id
```

定义视图时，基础列会保留 IDENTITY 属性。通过视图更新行时，不能为 IDENTITY 列指定新值。通过视图插入行时， Adaptive Server 为 IDENTITY 列生成新的顺序值。为列的基表设置 identity_insert on 后，只有表所有者、数据库所有者或系统管理员才能显式地在 IDENTITY 列中插入值。

创建视图后

创建视图后，描述视图的**源文本**存储在 syscomments 系统表的 text 列中。*切勿从 syscomments 中删除此信息。*而应使用 sp_hidetext 对 syscomments 中的文本加密。请参见 《参考手册：过程》和第 3 页的“编译对象”。

验证视图的选择标准

通常， Adaptive Server 不对视图的 insert 和 update 语句进行检查，以确定受影响的行是否在该视图的范围内。语句可以将行插入到底层基表中，但不能插入到视图中，语句还可以更改现有行使其不再符合视图的选择标准。

使用 with check option 子句创建视图时，会根据视图的选择标准对通过视图执行的每个 insert 和 update 进行验证。所有通过视图插入或更新的行都必须可以通过该视图来查看，否则该语句失败。

下面是用 with check option 创建的视图 stores_ca 的示例。此视图包含有关位于 California 的商店的信息，但不包含有关位于任何其它州的商店的信息。此视图是通过选择 stores 表中 state 值为 “CA” 的所有行来创建的：

```
create view stores_ca
as select * from stores
where state = "CA"
with check option
```

尝试通过 stores_ca 插入行时， Adaptive Server 会检验新行是否在该视图的范围内。下面的 insert 语句失败，因为新行的 state 值为 “NY”而不是 “CA”：

```
insert stores_ca
values ("7100", "Castle Books", "351 West 24 St.", "New
York", "NY", "USA", "10011", "Net 30")
```

尝试通过 stores_cal 更新行时， Adaptive Server 将检验该更新操作是否会导致该行从视图中消失。下面的 update 语句失败，因为它要将 state 的值从 “CA”更改为 “MA”。更新后，就不能再通过该视图看到该行了。

```
update stores_ca
set state = "MA"
where stor_id = "7066"
```

从其它视图派生的视图

如果视图是用 with check option 创建的，则从 “基”视图派生的所有视图都必须满足其检查选项要求。必须能通过基视图查看通过派生视图插入的每一行。通过派生视图更新的每一行都必须能通过基视图查看。

以从 stores_cal 派生的视图 stores_cal30 为例。此新视图包含位于 California 且付款方式为 “Net 30”的商店的有关信息：

```
create view stores_cal30
as select * from stores_ca
where payterms = "Net 30"
```

因为 stores_cal 是用 with check option 创建的，所以通过 stores_cal30 插入或更新的所有行都必须可通过 stores_cal 查看。 state 值不是 “CA”的任何行都会被拒绝。

stores_cal30 本身并没有 with check option 子句。这意味着可以通过 stores_cal30 插入或更新 payterms 值不是 “Net 30”的行。即使无法再通过 stores_cal30 查看该行，以下 update 语句仍会成功：

```
update stores_cal30
set payterms = "Net 60"
where stor_id = "7067"
```

通过视图检索数据

通过视图检索数据时， Adaptive Server 会进行检查，以确保在语句中的任何位置所引用的所有数据库对象都存在，并且它们在语句的环境中均有效。如果检查成功通过， Adaptive Server 就会将此语句与存储的视图定义相组合，然后转换为视图基础表上的一个查询。这一过程称为**视图解析**。

以下面的视图定义语句及其查询为例：

```
create view hiprice
as select *
from titles
where price > $15
and advance > $5000
select title, type
from hiprice
where type = "popular_comp"
```

在内部， Adaptive Server 将 hiprice 上的查询与其定义相组合，并将查询转换为：

```
select title, type
from titles
where price > $15
and advance > $5000
and type = "popular_comp"
```

通常，可以像处理真实的表一样用任何方法查询任何视图。对于视图，可通过任意组合使用连接、 group by 子句、子查询和其它查询方法。不过，如果视图是用外连接或集合函数定义的，则在查询该视图时可能会得到意想不到的结果。请参见第 401 页的 “从其它视图派生的视图”。

注释　可以对视图中的 text 和 image 列使用 select。但在视图中不能使用 readtext 和 writetext。

视图解析

定义视图时， Adaptive Server 将检验 from 子句中的所有表或视图是否存在。通过视图查询时也会执行类似的检查。

从定义视图到在语句中使用视图这段时间内，情况可能会发生变化。例如，视图定义的 from 子句中列出的一个或多个表或视图可能已被删除。或者视图定义的 select 子句中列出的一个或多个列可能已被重命名。

为了完全解析视图， Adaptive Server 将检验：

- 从中派生视图的所有表、视图和列仍然存在。
- 视图列所依赖的每一列的数据类型尚未被更改成不兼容的类型。
- 如果语句为 update、 insert 或 delete，则并不会违反修改视图的限制。这些内容将在第 408 页的 “通过视图修改数据”中进行讨论。

如果上述任何检查失败， Adaptive Server 都会发出错误消息。

重定义视图

Adaptive Server 允许重定义视图，而无需强制重定义依赖于它的其它视图，除非重定义过程使 Adaptive Server 不能转换相关视图。

例如， authors 表和三个可能的视图显示如下。每个后续视图都是通过其前面的视图来定义的：view2 是通过 view1 创建的，view3 是通过 view2 创建的。因此， view2 依赖于 view1， view3 依赖于前面两个视图。

每个视图的名称后面都跟随创建它时所使用的 select 语句。

view1：

```
create view view1
as select au_lname, phone
from authors
where postalcode like "94%"
```

view2：

```
create view view2
as select au_lname, phone
from view1
where au_lname like "[M-Z]%"
```

view3：

```
create view view3
as select au_lname, phone
```

```
from view2
where au_lname = "MacFeather"
```

这些视图所基于的 authors 表由以下列组成：au_id、 au_lname、 au_fname、 phone、 address、 city、 state 和 postalcode。

可以删除 view2 并用另一个视图来代替它，同样命名为 view2，但选择标准方面略有不同，如：

```
create view view2
as select au_lname, phone
from view3
where au_lname like "[M-P]"
```

依赖于 view2 的 view3 仍然有效并且不需要重新定义。如果使用的查询引用 view2 或 view3，则视图解析照常发生。

如果重定义 view2 使其不能派生 view3，则 view3 就会无效。例如，如果 view2 的另一个新版本只包含一个列 au_lname，而不包含 view3 所要求的那两个列，则不能再使用 view3，因为它不能从其所依赖的对象派生 phone 列。

但是， view3 仍存在，可以通过删除 view2 然后重新创建带有 au_lname 和 phone 列的 view2 来再次使用 view3。

简而言之，只要相关视图的 select 列表有效，就可在不影响相关视图的情况下更改中间视图的定义。如果违反了此规则，引用了无效视图的查询就会产生错误消息。

重命名视图

可以使用 sp_rename 来重命名视图：

```
sp_rename objname, newname
```

例如，要将 titleview 重命名为 bookview，请输入：

```
sp_rename titleview, bookview
```

重命名视图时应遵循下述约定：

- 确保新名称符合用于第 7 页的 “标识符”中所讨论的标识符的规则。
- 只能更改自己拥有的视图的名称。数据库所有者可以更改任何用户的视图的名称。
- 确保视图位于当前数据库中。

改变或删除基础对象

可以更改视图的基础对象的名称。例如，如果视图引用了名为 new_sales 的表，并将该表重命名为 old_sales，则该视图将使用重命名后的表。

但是，如果视图所引用的表已删除，当有人试图使用该视图时，Adaptive Server 就会产生错误消息。如果创建了新的表或视图来替代已删除的表或视图，则该视图将再次变为可用。

如果用 select * 子句定义视图，然后用添加列的方法来改变其基础表的结构，就不会显示新列。这是因为首次创建视图时，星号速记符得到了解释和扩展。要查看新列，可删除该视图然后重新创建它。

通过视图修改数据

虽然 Adaptive Server 对通过视图检索数据没有限制，并且与 SQL 的其它版本相比， Transact-SQL 对通过视图修改数据的限制也较少，但是各种数据修改操作都应遵循下列规则：

- 不允许进行对视图中的计算列或内置函数进行引用的 update、insert 或 delete 操作。
- 不允许执行对包含集合或行集合的视图进行引用的 update、insert 或 delete 操作。
- 不允许进行对 distinct 视图进行引用的 insert°¢delete 和 update 操作。
- 除非基础表或视图中的所有 NOT NULL 列都包含在用来插入新行的视图中，否则不允许使用 insert 语句。Adaptive Server 无法为基础对象中的 NOT NULL 列提供值。
- 如果视图中有 with check option 子句，则通过该视图（或通过任何派生的视图）插入或更新的所有行都必须符合该视图的选择标准。
- delete 语句不得用于多表视图。
- insert 语句不得用于使用 with check option 子句创建的多表视图。
- update 语句可用于使用 with check option 的多表视图。如果任何受影响的列出现在 where 子句中，或者出现在包含来自多个表的列的表达式中，更新将会失败。
- insert 和 update 语句不得用于多表 distinct 视图。

- update 语句不能指定 IDENTITY 列的值。为列的基表设置 identity_insert on 后，表所有者、数据库所有者或系统管理员可以在 IDENTITY 列中 insert 显式值。
- 如果通过多表视图来插入或更新行，所有受影响的列都必须属于同一基表。
- writetext 不得用于视图中的 text 和 image 列。

试图在视图中执行 update、insert 或 delete 操作时， Adaptive Server 要进行检查，以确保没有违反上述限制且没有违反数据完整性规则。

更新视图的限制

更新视图的限制适用于下述情况：

- 视图定义中的计算列
- 视图定义中的 group by 或 compute
- 基础对象中的空值
- 用 with check option 创建的视图
- 多表视图
- 带有 IDENTITY 列的视图

视图定义中的计算列

此限制适用于从计算列或内置函数派生的视图的列。例如，视图 accounts 中的 amt_due 列是计算列。

```
create view accounts (title_id, advance, amt_due)
as select titles.title_id, advance,
(price * royalty/100) * total_sales
from titles, roysched
where price > $15
  and advance > $5000
and titles.title_id = roysched.title_id
and total_sales between lorange and hirange
```

可通过 accounts 查看的行如下：

```
select * from accounts

title_id     advance     amt_due
--------     --------    ---------
```

```
PC1035      7,000.00    32,240.16
PC8888      8,000.00     8,190.00
PS1372      7,000.00       809.63
TC3218      7,000.00       785.63

(4 rows affected)
```

不允许对 amt_due 列执行 updates 和 inserts 操作，因为无法从 amt_due 列中的任何输入值推知价格、版税或年销售额的基础值。因为没有需要删除的基值，所以 delete 操作毫无意义。

视图定义中的 *group by* 或 *compute*

此限制适用于包含集合值的视图 （即，其定义中包含 group by 或 compute 子句的视图）中的所有列。下面是一个用 group by 子句定义的视图和通过它可查看的行：

```
create view categories (category, average_price)
as select type, avg(price)
from titles
group by type


select * from categories

category         average_price
-------------    -------------
UNDECIDED                 NULL
business                 13.73
mod_cook                 11.49
popular_comp             21.48
psychology               13.50
trad_cook                15.96

(6 rows affected)
```

在视图 categories 中 insert 行毫无意义。被插入的行属于哪个基础行组呢？因为无法从输入的任何值得知应该如何更改基础价格，所以不允许更新 average_price 列。

基础对象中的 NULL 值

当从中派生该视图的表或视图中包含一些 NOT NULL 列时，此限制适用于 insert 语句。

例如，假定视图下面的表的列中不允许有空值。通常，在通过视图对新行进行 insert 操作时，所有未包含在该视图内的基础表中的任何列的值都为空值。如果其中的一列或多列不允许有空值，则不允许通过该视图执行插入操作。

例如，在此视图中：

```
create view business_titles
as select title_id, price, total_sales
from titles
where type = "business"
```

基础表 titles 的 title 列中不允许有空值，所以不允许通过 business_view 使用 insert 语句。虽然视图中甚至不存在 title 列，但其禁止空值的限制使得在该视图中的任何插入操作都是非法的。

同样，如果 title_id 列具有唯一的索引，那些重复基础表中的任何值的更新或插入操作都会遭到拒绝，即使条目不重复视图中的任何值也是如此。

用 *with check option* 创建的视图

此限制用于确定通过有检查选项的视图可进行的修改的类型。如果视图具有 with check option 子句，则通过该视图插入或更新的每一行都必须能通过该视图查看。无论是直接地还是通过另一个派生视图间接地插入或更新视图都是如此。

多表视图

对于连接多个表中的列的视图，此限制用于确定通过它们可以进行的修改的类型。 Adaptive Server 禁止在多表视图上使用 delete 语句，但是允许使用其它系统所不允许的 update 和 insert 语句。

在下述情况下，可以对多表视图执行 insert 或更新操作：

- 该视图没有 with check option 子句。
- 所有被插入或更新的列都属于同一基表。

以下面的视图为例，该视图包含了 titles 和 publishers 中的列，但没有 with check option 子句：

```
create view multitable_view
as select title, type, titles.pub_id, state
from titles, publishers
where titles.pub_id = publishers.pub_id
```

单个 insert 或 update 语句*既*可以指定 titles 中的列的值，*也*可以指定 publishers 中的列的值：

```
update multitable_view
set type = "user_friendly"
where type = "popular_comp"
```

但是，此语句会因影响同属 titles 和 publishers 中的列而失败：

```
update multitable_view
set type = "cooking_trad",
state = "WA"
where type = "trad_cook"
```

带有 IDENTITY 列的视图

此限制用于确定可对包含 IDENTITY 列的视图进行的修改的类型。根据定义， IDENTITY 列是不可更新的。通过视图所进行的更新不能指定 IDENTITY 列的值。

对 IDENTITY 列的插入操作限制为：

- 表所有者
- 表所有者授权的数据库所有者或系统管理员
- 通过使用 setuser 命令来模拟表所有者的数据库所有者或系统管理员。

要通过视图来启用此类插入，可对该列的基表使用 set identity_insert on。对于通过其进行插入的视图，仅使用 set identity_insert on 是不够的。

删除视图

要从数据库中删除视图，请使用 drop view：

```
drop view [owner.]view_name [, [owner.]view_name]...
```

可见，一次可删除多个视图。只有视图所有者 （或数据库所有者）可以删除视图。

发出 drop view 后，就会从 sysprocedures、 sysobjects、 syscolumns、 syscomments、 sysprotects 和 sysdepends 中删除有关该视图的信息。对该视图拥有的所有特权也将被删除。

如果视图所依赖的表或另一个视图已被删除，当有人试图使用该视图时， Adaptive Server 将返回错误消息。如果创建新的表或视图来替代已被删除的表或视图，并且其名称与被删除的表或视图的名称相同，只要视图定义中所引用的列存在，该视图就会再次变为可用。

使用视图作为安全性机制

无论对视图基础表的权限如何，必须明确授予或撤消访问视图中数据子集的权限。授权访问视图而非基础表的用户无法看到视图中没有的基础表数据。

例如，可能想禁止某些用户访问 titles 表中有关销售额和销售量的列。可以创建不包含这些列的 titles 表视图，然后将该视图的权限授予所有用户，而将访问表的权限只授予销售部门。例如：

```
revoke all on titles to public
grant all on bookview to public
grant all on titles to sales
```

请参见 《系统管理指南：卷 1》中的第 17 章 “管理用户权限”。

获取有关视图的信息

系统过程、目录存储过程和 Adaptive Server 内置函数可从系统表中提供有关视图的信息。请参见 《参考手册：过程》。

使用 sp_help 和 *sp_helptext* 显示视图信息

使用 sp_help 报告有关视图的信息：

```
sp_help hiprice
---------------
```

系统安全员必须在已评估的配置中重新设置 allow select on syscomments.text column 配置参数。（有关详细信息，请参见 “词汇表”中的**已评估的配置**。）在这种情况下，只有视图的创建者或系统管理员才能通过 sp_helptext 来查看视图的文本。

要显示 create view 语句的文本，请执行 sp_helptext：

```
sp_helptext hiprice
# Lines of Text
-----------------
3

(1 row affected)

text

 ------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------

--Adaptive Server has expanded all '*' elements in the following statement

 create view hiprice as select

titles.title_id, titles.title, titles.type, titles.pub_id, titles.price,
titles.advance, titles.total_sales, titles.notes, titles.pubdate,
titles.contract from titles where price > $15 and advance > $5000

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

如果视图的源文本已用 sp_hidetext 加密， Adaptive Server 就会显示一条消息，提示您文本已被隐藏。请参见 《参考手册：过程》。

使用 *sp_depends* 列出相关对象

sp_depends 可列出视图或表在当前数据库中引用的所有对象，以及所有引用该视图或表的对象。

```
sp_depends titles

Things inside the current database that reference the
object.
object                type
-------------         ---------------------------
dbo.history_proc      stored procedure
dbo.title_proc        stored procedure
dbo.titleid_proc      stored procedure
dbo.deltitle          trigger
dbo.totalsales_trig   trigger
dbo.accounts          view
dbo.bookview          view
dbo.categories        view
dbo.hiprice           view
dbo.multitable_view   view
dbo.titleview         view

(return status = 0)
```

列出数据库中的所有视图

以下面的格式使用 sp_tables 时，该命令可列出数据库中的所有视图：

```
sp_tables @table_type = "'VIEW'"
```

查找对象名和 ID

系统函数 object_id() 和 object_name() 用来标识视图的 ID 和名称。例如：

```
select object_id("titleview")

----------
 480004741
```

对象的名称和 ID 存储在 sysobjects 表中。

第 13 章

创建表的索引

使用**索引**可以基于指定列中的值快速访问表中的数据。一个表可以有多个索引。索引对访问表中数据的用户是透明的； Adaptive Server 自动决定何时使用为表创建的索引。

有关如何设计索引来提高性能的信息，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》）。有关创建和管理分区索引的信息，请参见第 10 章 “对表和索引进行分区”。

索引如何工作

索引有助于 Adaptive Server 定位数据。通过指向表列数据在磁盘上的位置，索引可加速数据检索过程。例如，假设需要使用 stores 表中的存储标识号运行常用的查询。为防止 Adaptive Server 必须搜索表中的每一行 （如果 stores 表包含数百万行，这样做会非常耗时），可创建以下名为 stor_id_ind 的索引：

```
create index stor_id_ind
on stores (stor_id)
```

下次查询 stores 中的 stor_id 列时，stor_id_ind 索引会自动生效。换句话说，索引对用户是透明的。 SQL 不包括在查询中引用索引的语法。只能从表中创建或删除索引； Adaptive Server 决定是否对为该表提交的每个查询都使用索引。如果表中数据随着时间推移而发生变化， Adaptive Server 可以更改表的索引以反映这些变化。而且，这些更改对用户也是透明的。

Adaptive Server 支持以下类型的索引：

- **组合索引** — 这些索引涉及多个列。当两个或更多个列因其逻辑关系而最好作为一个单位进行搜索时，请使用此类型的索引。
- **唯一索引** — 这些索引不允许指定的列中的任意两行有相同的值。 Adaptive Server 在创建索引（如果数据已经存在）和每次添加数据时检查重复值。
- **聚簇索引**或**非聚簇索引** — 聚簇索引强制 Adaptive Server 不断地对表中的行进行排序和重排序，以使其物理顺序与逻辑（或索引）顺序始终保持一致。每个表只能有一个聚簇索引。非聚簇索引不要求行的物理顺序与其索引顺序相同。可以使用每个非聚簇索引以各种排序顺序访问数据。
- **本地索引** — 本地索引是一个索引子目录树，它仅编制一个数据分区的索引。可以对本地索引进行分区，并且对于所有类型的分区表都支持本地索引。
- **全局索引** — 全局索引编制表中所有数据分区的索引。循环分区表支持非分区全局聚簇索引，而所有类型的分区表都支持非聚簇全局索引。不能对全局索引进行分区。

第 10 章 “对表和索引进行分区”中描述了本地索引和全局索引。本章后面的部分详细描述其余类型的索引。

比较创建索引的两种方式

可以通过使用 create index 语句（在本章中描述），或通过使用 create table 命令的 unique 或 primary key 完整性约束来创建表的索引。但是，在下列方式中完整性约束受到限制：

- 不能创建非唯一索引。
- 不能使用由 create index 命令提供的选项来定制索引的工作方式。
- 只能使用 alter table 语句将这些索引作为约束删除。

如果应用程序需要这些功能，则应该使用 create index 创建索引。另外， unique 或 primary key 完整性约束为定义表的索引提供了一种更简单的方法。有关 unique 和 primary key 约束的信息，请参见第 7 章“创建数据库和表”。

使用索引指南

索引可加速数据的检索。向列添加索引通常会对查询是快速响应还是长时间等待有重要影响。

但是，建立索引要花费时间并占用存储空间。例如，当聚簇索引重建时，非聚簇索引也将自动重新创建。

此外，插入、删除或更新索引列中的数据所花费的时间比未索引列中的数据要长。但是，索引在检索性能方面的提高程度通常会超过这种开销。

何时建立索引

请使用以下常规指南：

- 如果打算在 IDENTITY 列中进行手动插入，请创建一个唯一索引以确保不会插入一个已使用的值。
- 对于经常按排序顺序访问的列 （即在 order by 子句中指定的列），可能应该编制其索引，以便 Adaptive Server 可以利用索引顺序。
- 对经常用于连接的列应始终编制其索引，因为如果列是按排序顺序存储的，系统就能更快地执行连接。
- 存储表的主键的列通常有聚簇索引，尤其是当它经常与其它表中的列连接时更是如此。请注意，每个表只能有一个聚簇索引。
- 对聚簇索引来说最好的选择是经常用于搜索值的范围的列。找到具有搜索范围内第一个值的行后，即可保证具有后续值的行物理相邻。聚簇索引对搜索单个值来说并没有太大优势。

何时不应编制索引

在某些情况下，索引不会发挥作用：

- 在查询中很少或从不引用的列并不从索引获益，因为系统很少必须根据这些列中的值搜索行。
- 相对于表中行数而言有许多重复值而几乎没有唯一值的列不能从索引中真正获益。

如果系统确实必须搜索未建索引的列，则搜索将逐行进行。执行这种扫描所花的时长与表中的行数成正比。

创建索引

在执行 create index 之前，请打开 select into：

```
sp_dboption,'select into', true
```

create index 的最简单形式为：

```
create index index_name
    on table_name (column_name)
```

若要对 authors 表中的 au_id 列创建索引，请执行：

```
create index au_id_ind
on authors(au_id)
```

索引名称必须符合标识符的规则。列和表名称指定要为其编制索引的列和包含该列的表。

对于 bit、 text 或 image 数据类型的列，不能创建索引。

只有表的所有者才能 create 或 drop 索引。表的所有者可以随时 create 或 drop 索引，无论表中是否有数据。通过限定表名，可以对另一数据库中的表创建索引。

create index 的语法

以下部分解释此命令的各种选项。有关创建和管理索引分区（包括使用 index_partition_clause）的信息，请参见第 10 章 “对表和索引进行分区”。

注释　使用 create index 的 on *segment_name* 扩展可以在指向特定数据库设备或数据库设备集合的段上放置索引。在段上创建索引之前，要获得可用段的列表，请与系统管理员或数据库所有者联系。出于性能原因或其它考虑事项，可能已将某些段分配给特定的表或索引。

为多列编制索引：组合索引

要对所有指定列中的组合值创建组合索引，必须指定两个或更多个列名。

最好将两列或更多列作为一个单位搜索时（例如，authors 表中作者的名和姓），请使用组合索引。在表名后的小括号内，按排序优先级顺序列出组合索引中要包括的列，如下所示：

```
create index auth_name_ind
on authors(au_fname, au_lname)
```

组合索引中列的顺序不必与 create table 语句中列的顺序相同。例如，在上述索引创建语句中，au_lname 和 au_fname 的顺序可以颠倒。

可以在单个组合索引中指定多达 31 个列。组合索引中的所有列必须在同一个表中。请参见 Performance and Tuning Series: Physical Database Tuning（《性能和调优系列：物理数据库调优》）中的第 4 章 “Table and Index Size”（表和索引大小）。

使用基于函数的索引编制索引

基于函数的索引包含一个或多个作为索引键的表达式。可直接对函数和表达式创建索引。

就像计算列一样，基于函数的索引对用户定义的排序和 DSS（决策支持系统）应用程序（它们经常需要进行密集型数据操作）很有用。基于函数的索引简化了这些应用程序中的任务并提高了性能。

有关用户定义的排序的详细信息，请参见第 278 页的 “使用计算列”。

有关计算列的详细信息，请参见第 277 页的 “计算列”。

基于函数的索引类似于计算列，其中都允许对表达式创建索引。

但是，存在显著的区别：

- 使用函数索引可以直接为表达式编制索引。并不首先创建列。
- 基于函数的索引必须是确定性的，并且不能引用全局变量，这与计算列不同。
- 可创建聚簇计算列索引，但不能创建基于函数的聚簇索引。

语法更改

用于创建基于函数的索引的语法包含一些新变量，以粗体显示：

```
create [unique] [clustered] | nonclustered] index index_name
on [[ database.] owner.] table_name
(column_expression [asc | desc] [, column_expression [asc |
desc]]...
```

在执行 create index 之前，必须先打开数据库选项 select into。

```
sp_dboption <dbname>, 'select into', true
```

请参见 《参考手册：命令》和 《参考手册：过程》。

使用 *unique* 选项

唯一索引不允许两行有相同的索引值 （包括 NULL）。在创建索引时，如果数据已经存在，系统将检查重复值，以后每次使用 insert 或 update 增加或修改数据时也会进行检查。

只有在数据本身具有唯一性时指定唯一索引才有意义。例如，不应根据 last_name 列建立唯一索引，因为即使在只有几百行的表中也可能有多个 “Smith”或 “王”存在。

但是，为存储社会保险号的列编制唯一索引很有意义。数据具有唯一性 — 每人都有一个不同的社会保险号。此外，唯一索引还用于完整性检查。例如，重复的社会保险号反映可能数据输入或政府部门出现某种错误。

如果尝试对包含重复值的数据创建唯一索引，则将中止该命令，而 Adaptive Server 显示给出第一个重复值的错误消息。不能对多行中都包含空值的列创建唯一索引；出于索引目的，这些空值被视为重复值。

如果尝试更改对其有唯一索引的数据，则结果取决于是否使用了 ignore_dup_key 选项。有关详细信息，请参见第 428 页的 “使用 ignore_dup_key 选项”。

可以对组合索引使用 unique 关键字。

在非唯一索引中包括 IDENTITY 列

identity in nonunique index 数据库选项自动在表的索引键中包括 IDENTITY 列，以使在该表上创建的所有索引都唯一。此选项使逻辑上非唯一的索引在内部唯一，并使其可以处理可更新游标和隔离级别 0 的读取。

若要启用 identity in nonunique indexes，请输入：

```
sp_dboption pubs2, "identity in nonunique index", true
```

表必须已经具有 IDENTITY 列， identity in nonunique index 数据库选项才能起作用，这可以通过 create table 语句或通过在创建表之前将 auto identity 数据库选项设置为 true 来实现。

通过 identity in nonunique index 在具有非唯一索引的表上使用游标和隔离级别 0 读取。唯一索引可确保下一次对该游标执行 fetch 时，游标定位于正确的行。

例如，将 identity in nonunique index 和 auto identity 设置为 true 之后，假设创建以下表 （没有索引）：

```
create table title_prices
(title varchar(80)  not null,
price  money        null)
```

sp_help 显示表包含 IDENTITY 列 SYB_IDENTITY_COL，它由 auto identity 数据库选项自动创建。如果对 title 列创建索引，则使用 sp_helpindex 检验索引是否自动包括 IDENTITY 列。

升序和降序排列索引列值

可使用 asc （升序）和 desc （降序）关键字对索引中的每一列指派一个排序顺序。缺省情况下，排序顺序是升序。

创建索引以使列的顺序与查询的 order by 子句中所指定的相同，这样就会消除查询处理过程中对列的排序。下列示例对 Orders 表创建索引。该索引有两列，第一列是 customer_ID，按升序排列，第二列是 date，按降序排列，这样就会最先列出最新订单：

```
create index nonclustered cust_order_date
    on Orders
    (customer_ID asc,
    date desc)
```

使用 *fillfactor*、 *max_rows_per_page* 和 *reservepagegap*

fillfactor、 max_rows_per_page 和 reservepagegap 是空间管理属性，适用于表和索引并影响用数据填充物理页的方式。请参见 《参考手册：命令》。表 13-1 总结了有关索引的空间管理属性的信息。

表 13-1：索引的空间管理属性摘要

属性	说明	使用	注释
fillfactor	指定创建索引时某页可填充空间的百分比。 fillfactor 低于 100% 会为插入到页中留出空间，不会立即导致页拆分。 优点在于： • 开始时页拆分较少。 • 减少对页的争用，因为页很多但页上的行较少。	只适用于仅数据锁定表的聚簇索引。	仅当对现有数据的表创建索引时，才使用 fillfactor 百分比。创建表后它不适用于页和插入。 如果未指定 fillfactor，则使用系统范围的缺省 fillfactor。最初，将其设置为 100%，但可使用 sp_configure 对其进行更改。
max_rows_per_page	指定每页允许的最大行数。 优点在于： • 通过限制每页的行数并增加页数可减少对页的争用。	仅适用于所有页锁定表。 此属性可以设置的最大值为 256。	从创建索引开始，max_rows_per_page 始终适用。如果未指定，缺省值即是页上将要容纳的最大行数。

属性	说明	使用	注释
reservepagegap	确定分配扩充时留空的页数。例如， reservepagegap 值为 16 是指分配扩充时，在 2 个扩充中共有 16 页，其中 1 页留作空页。 优点在于： • 可减少行的前移并降低维护活动 （如运行 reorg rebuild 和重建索引）的频率。	适用于所有锁定方案中的页。	如果未指定 reservepagegap，则分配扩充时不留空页。

以下语句将索引的 fillfactor 设置为 65%，并将每一分配扩充的 reservepagegap 设置为一个空页：

```
create index postalcode_ind2
    on authors (postalcode)
    with fillfactor = 10, reservepagegap = 8
```

计算列的索引

只要可以为结果的数据类型编制索引，您就可以对计算列创建索引，就好像这些列是常规列一样。计算列的索引提供一种对复杂数据类型 （如 XML、 text、 image 和 Java 类）创建索引的方式。

例如，以下代码示例对计算列创建聚簇索引，就好像这些列是常规列一样：

```
CREATE CLUSTERED INDEX name_index on parts_table(name_order)
CREATE INDEX adt_index on parts_table(version_order)
CREATE INDEX xml_index on parts_table(spec_index)
CREATE INDEX text_index on parts_table(descr_index)
```

创建或更新索引时， Adaptive Server 将对计算列进行求值，并使用结果来建立或更新索引。

基于函数的索引

利用基于函数的索引功能，可直接对函数和表达式创建索引。与计算列索引类似，此功能对用户定义的排序和 DDS 应用程序很有用。

下例使用表中的三列对广义索引键创建索引。

```
CREATE INDEX generalized_index on parts_table
     (general_key(part_no,listPrice,
part_no>>version)
```

在某些情况下，如果无法对单个列创建索引，则可以通过调用返回多列的组合值的用户定义函数来创建广义索引键。

使用聚簇或非聚簇索引

利用聚簇索引， Adaptive Server 可以随时对行进行排序，以使其物理顺序与其逻辑 （索引）顺序相同。聚簇索引的底级或**叶级**包含表的实际数据页。请在创建任何非聚簇索引之前创建聚簇索引，因为创建聚簇索引时会自动重建非聚簇索引。

每个表只能有一个聚簇索引。它通常根据**主键** （唯一标识行的一列或多列）来创建。

从逻辑上讲，数据库的设计将确定一个主键。可以使用 create table 或 alter table 语句指定 primary key 约束，从而创建索引并强制主键属性用于表列。用 sp_helpconstraint 可显示有关约束的信息。

同样，使用 sp_primarykey、 sp_foreignkey 和 sp_commonkey 可显式定义主键、外键和公用键 （经常被连接的键对）。但是，这些过程并不强制键关系。

使用 sp_helpkey 可显示有关定义的键的信息，使用 sp_helpjoins 可显示有关适于连接的列的信息。请参见 《参考手册：过程》。有关主键和外键的定义，请参见第 20 章 “触发器：强制实施参照完整性”。

对于非聚簇索引，行的物理顺序与其索引顺序不同。非聚簇索引的叶级包含指向数据页上的行的指针。更确切地说，每个叶页包含索引值以及指向该值所在行的指针。换句话说，非聚簇索引在索引结构与数据本身之间有一个额外的级别。

表中最多允许有 249 个非聚簇索引，通过每个此类索引都可以按不同的排序顺序访问数据。

使用聚簇索引查找数据几乎总是比使用非聚簇索引查找数据要快。另外，聚簇索引更适于检索多个具有连续键值的行的情况（即需要经常在列上搜索值的范围）。找到第一个**键值**所在的行之后，后继索引值所在的行在物理上一定相邻，因而不需要进一步搜索。

如果不使用关键字 clustered 和 nonclustered， Adaptive Server 将创建非聚簇索引。

以下是 titles 表中 title_id 列的 titleidind 索引的创建方法。要尝试此命令，必须先删除该索引：

```
drop index titles.titleidind
```

然后创建聚簇索引：

```
create clustered index titleidind
on titles(title_id)
```

如果您认为需要经常对 friends_etc 表中的人员（在第 7 章 “创建数据库和表”中按邮政编码创建的）进行排序，请对 postalcode 列创建非聚簇索引：

```
create nonclustered index postalcodeind
on friends_etc(postalcode)
```

此处唯一索引并无意义，因为某些联系人很可能具有相同的邮政编码。聚簇索引也不适合，因为邮政编码不是主键。

friends_etc 中的聚簇索引应是人员的名和姓列的组合索引，例如：

```
create clustered index nmind
on friends_etc(pname, sname)
```

对段创建聚簇索引

使用 create index 命令可以对指定的段创建索引。由于聚簇索引的叶级及其数据页在定义上相同，因此创建 clustered 索引并使用 on segment_name 扩展会将表从创建它的设备移动到命名段。

对段创建表或索引之前，请与系统管理员或数据库所有者联系；出于性能原因，可以保留某些段。

指定索引选项

当使用 insert 或 update 创建重复键或重复行时，索引选项 ignore_dup_key、 ignore_dup_row 和 allow_dup_row 控制发生的操作。表 13-2 显示根据索引类型要使用的选项。

表 13-2：索引选项

索引类型	选项
聚簇	ignore_dup_row | allow_dup_row
唯一聚簇索引	ignore_dup_key
非聚簇	无
唯一非聚簇索引	无

使用 *ignore_dup_key* 选项

如果尝试向具有唯一索引的列中插入重复值，则会取消该命令。通过对唯一索引包括 ignore_dup_key 选项，可避免此情况出现。

唯一索引可以是聚簇的或非聚簇的。开始数据输入时，任何插入重复键的尝试都被取消，并显示错误消息。取消后，所有当时活动的事务都可以继续执行，就像 update 或 insert 从未发生过一样。可以正常插入非重复键。

无论是否设置 ignore_dup_key，都不能对已经包括重复值的列创建唯一索引。如果尝试这样做， Adaptive Server 会输出错误消息以及重复值的列表。对列创建唯一索引之前，必须消除重复值。

下面是一个使用 ignore_dup_key 选项的示例：

```
create unique clustered index phone_ind
on friends_etc(phone)
with ignore_dup_key
```

使用 *ignore_dup_row* 和 *allow_dup_row* 选项

ignore_dup_row 和 allow_dup_row 是用于创建非唯一聚簇索引的选项。创建非唯一的非聚簇索引时，这些选项并不相关。由于 Adaptive Server 非聚簇索引在内部附加了唯一的行标识号，因此永远不会出现重复行，即使对于相同的数据值也是如此。

ignore_dup_row 和 allow_dup_row 相互排斥。

非唯一聚簇索引允许重复键，但不允许重复行，除非指定 allow_dup_row。

如果设置 allow_dup_row，则可以对包括重复行的表创建新的非唯一聚簇索引，随后可以 insert 或 update 重复行。

如果表中的任何索引都是唯一的，则唯一性要求 （最严格的要求）优先于 allow_dup_row 选项。因而， allow_dup_row 只适用于具有非唯一索引的表。如果表中任何列存在唯一聚簇索引，则不能使用此选项。

ignore_dup_row 选项消除一批数据中的重复项。当输入重复行时，Adaptive Server 忽略该行并取消该 insert 或 update，同时显示错误消息。取消后，任何当时可能已经活动的事务都将继续执行，就像插入或更新从未发生过一样。正常插入非重复行。

ignore_dup_row 只适用于具有非唯一索引的表：如果表中任何列存在唯一索引，则不能使用此关键字。

表 13-3 说明 allow_dup_row 和 ignore_dup_row 如何影响在表（包括重复行）中创建非唯一聚簇索引及向表中输入重复行的尝试。

表 13-3：索引中的重复行选项

选项	有重复	输入重复
未设置选项	create index 命令失败。	命令失败。
allow_dup_row 设置	命令完成。	命令完成。
设置了 ignore_dup_row	创建了索引，但删除了重复行；错误消息。	不插入 / 更新重复；错误消息；事务完成。

使用 *sorted_data* 选项

当表中的数据已处于排序顺序时 （例如，当使用 bcp 将已排序的数据复制到空表时）， create index 的 sorted_data 选项会加速索引的创建。速度提高对大型表尤其明显，对 1GB 以上的表，速度将提高到许多倍。

如果指定 sorted_data，但数据未处于排序顺序，则将显示错误消息并中止命令。

此选项只提高编制聚簇索引或唯一非聚簇索引的速度。但是，创建非唯一非聚簇索引会成功，除非存在具有重复键的行。如果存在具有重复键的行，则会显示错误消息并中止命令。

即使指定了 sorted_data，某些其它 create index 选项也需要排序。请参见 《参考手册：命令》。

使用 *on segment_name* 选项

on segment_name 子句指定要对其创建索引的数据库段名。可以对不同的段创建非聚簇索引，但不能对数据页创建。例如：

```
create index titleind
on titles(title)
on seg1
```

创建聚簇索引时，如果使用 segment_name，则包含该索引的表将移动到您所指定的段。对段创建表或索引之前，请与系统管理员或数据库所有者联系；出于性能原因，可以保留某些段。

删除索引

drop index 命令用于从数据库中删除索引。它的语法为：

drop index *table_name.index_name*
[, *table_name.index_name*]...

使用此命令时， Adaptive Server 将从数据库中删除指定的索引并回收其存储空间。

只有索引的所有者可删除它。 drop index 权限不能转移给其他用户。不能对 master 数据库或用户数据库的任何系统表使用 drop index 命令。

如果索引并不用于大多数或全部查询，则可能需要删除它。

要删除 friends_etc 表中的 phone_ind 索引，命令为：

```
drop index friends_etc.phone_ind
```

在执行规模类型不同的跨平台 load database 后使用 sp_post_xpload 检查并重建索引。

确定表上存在哪些索引

要查看表上存在的索引，可使用 sp_helpindex。以下是关于 friends_etc 表的报告：

```
sp_helpindex friends_etc

index_name      index_keys    index_description    index_max_rows_per_page
-------------   ----------   ----------------- --------------------------
nmind           pname,sname   clustered                                  0
postalcodeind   postalcode    nonclustered                               0

index_fillfactor  index_reservepagegap  index_created        index_local
----------------- --------------------  ---------------      ------------
               0                     0  May 24 2005 1:49PM   Global Index
               0                     0  May 24 2005 1:49PM   Global Index

(2 rows affected)

index_ptn_name            index_ptn_seg
------------------        ---------------
nmind_1152004104          default
postalcodeind_1152004104  default
(2 rows affected)
```

sp_help 在其报告的最后运行 sp_helpindex。

sp_statistics 返回表的索引列表。例如：

```
    sp_statistics friends_etc

table_qualifier               table_owner
         table_name                        non_unique
         index_qualifier                   index_name
         type   seq_in_index column_name                      collation
         cardinality pages
------------------------------- -------------------------------
        -------------------------------- ----------
        -------------------------------- ------------------------------
        ------ ------------ ------------------------------- ---------
        ----------- -----------
pubs2                          dbo
        friends_etc                            NULL
        NULL                                 NULL
             0           NULL NULL                              NULL
                   0           1
pubs2                            dbo
         friends_etc                                1
```

```
         friends_etc                      nmind
            1           1 pname                        A
                    0           1
pubs2                              dbo
         friends_etc                                 1
         friends_etc                       nmind
               1            2 sname                          A
                   0            1
pubs2                              dbo
         friends_etc                                 1
         friends_etc                      postalcodeind
             3           1  postalcode                       A
                NULL       NULL

(4 rows affected)

table_qualifier table_owner table_name index_qualifier index_name
       non_unique_type seq_in_index column_name collation index_id
       cardinality pages status status2
---------------- ---------  ---------- --------------- -----------
       ---------------- ------------ ----------- --------- ---------
       ----------- ----- ------ --------
pubs2            dbo         friends_etc friends_etc     nmind
                   1     1              1 pname      A
                    0          1        16       0
pubs2            dbo         friends_etc friends_etc     nmind
                   1     1              2 sname      A
                   0          1        16       0
pubs2            dbo         friends_etc friends_etc     postalcodeind
                   1     3              1 postalcode A
                NULL          NULL       0       0
pubs2             dbo         friends_etc NULL            NULL
                NULL     0      NULL NULL    NULL    0
                   0          1           0       0
(4 rows affected)
(return status = 0)
```

此外，如果在表名后接着输入“1”，则 sp_spaceused 将报告表及其索引所使用的空间量。例如：

```
sp_spaceused friends_etc, 1
index_name           size       reserved   unused
-------------------- ---------- ---------- ----------
nmind                2 KB       32 KB      28 KB
postalcodeind        2 KB       16 KB      14 KB
```

```
name         rowtotal   reserved     data    index_size      unused
------------ ---------- ------------ ------- --------------- --------
friends_etc  1          48 KB        2 KB    4 KB            42 KB

(return status = 0)
```

更新关于索引的统计信息

在 Adaptive Server 处理查询时， update statistics 命令通过保持索引中键值分布为最新，帮助 Adaptive Server 做出关于使用何种索引的最优决策。在索引列中添加、更改或删除了大量数据后使用 update statistics。

启用了组件集成服务后， update statistics 可生成远程表的准确分布统计信息。请参见 《组件集成服务用户指南》。

缺省情况下，授予表的所有者发出 update statistics 命令的权限，并且不能移交。它的语法为：

update statistics *table_name* [*index_name*]

如果不指定索引名，则该命令更新指定表中所有索引的分布统计信息。如果给定索引名，将仅对该索引更新统计信息。

通过使用 sp_helpindex 可查找索引名。以下演示如何列出 authors 表的索引：

```
sp_helpindex authors
```

```
index_name index keys            index_description  index_max_rows_per_page
---------- ----------------      ----------  ----------------------
auidind    au_id                 clustered, unique                        0
aunmind    au_lname, au_fname    nonclustered                             0

index_fillfactor  index_reservepagegap   index_created        index_local
----------------- ---------------------- -------------------- -------------
               0                      0   Apr 13 2005 10:30AM  Global Index
               0                      0   Apr 13 2005 10:30AM  Global Index
(2 rows affected)
index_ptn_name       index_ptn_seg
------------------   --------------
auidind_384001368    default
aunmind_384001368    default
(2 rows affected)
```

若要更新所有索引的统计信息，请键入：

```
update statistics authors
```

若要仅更新 au_id 列的索引的统计信息，请键入：

```
update statistics authors auidind
```

因为 Transact-SQL 不要求索引名在数据库中唯一，所以必须给出与索引相关联的表名。对现有数据创建索引时， Adaptive Server 自动运行 update statistics。

可以设置 update statistics，使它在最适合您节点时自动运行并避免在阻碍您的系统时运行。

第 14 章

为数据定义缺省值和规则

缺省值是在用户没有为某个列显式输入值时 Adaptive Server 插入到该列中的值。在数据库管理中，**规则**指定在特定列或给定用户定义数据类型的任意列中允许或不允许输入什么。可以使用缺省值和规则来维护数据库中的数据完整性。

缺省值和规则是怎样执行的

可以为表列或用户定义数据类型定义一个值，当用户未显式输入值时将自动插入该值。例如，可以创建一个值为 “???”或 “fill in later”的缺省值。也可以为该表列或数据类型定义一些规则，以限制用户可为其输入的值类型。

在关系数据库管理系统中，每个数据元素必须包含一些值，即使该值为空。如第 7 章 “创建数据库和表”中所述，某些列不接受空值。对于这些列，必须输入其它值，可以是用户显式输入的值，也可以是由 Adaptive Server 输入的缺省值。

规则强制实现数据完整性的方式不受列的数据类型的影响。规则可与一个特定列、几个特定列或用户定义数据类型相关联。

用户每次输入值时， Adaptive Server 根据绑定在特定列上的最新规则检查该值。不检查在规则创建并绑定以前输入的值。

作为缺省值和规则的替代，可以使用 create table 语句的 default 子句和 check 完整性约束来完成一些相同的任务。然而，这些项特定于每个表，无法将其绑定到其它表中的列或用户定义的数据类型上。有关完整性约束的详细信息，请参见第 7 章 “创建数据库和表”。

创建缺省值

在数据输入表之前或之后，可随时创建或删除缺省值。通常，要创建缺省值：

1 使用 create default 定义缺省值。

2 用 sp_bindefault 将缺省值绑定到相应表中的列或用户定义的数据类型。

3 通过插入数据测试绑定缺省值。

可用 drop default 删除缺省值，并用 sp_unbinddefault 撤消它们之间的关联。

创建并绑定缺省值时：

- 确保列对于缺省值足够大。例如，一个 char(2) 列不能持有 17 个字节的字符串，如 “Nobody knows yet”。
- 在对用户定义数据类型设置缺省值，并对该类型的单个列设置不同的缺省值时应小心。如果先绑定数据类型缺省值再绑定列缺省值，列缺省值将只在该命名列代替用户定义数据类型的缺省值。用户定义数据类型的缺省值被绑定到所有其它具有此数据类型的列上。

  但是，一旦把另一个缺省值绑定到因其类型而有了缺省值的列，则该列将不再受其数据类型所绑定的缺省值的影响。该问题将在第 437 页的 “绑定缺省值”中详细论述。
- 注意缺省值与规则不要发生冲突。要确保缺省值是规则所允许的；否则，缺省值可能被规则消除。

  例如，如果规则允许的输入值在 1 到 100 之间，而缺省值设为 0，则规则拒绝该缺省输入值。改变缺省值或者改变规则。

create default 语法

create default 的语法是：

```
create default [owner.]default_name
    as constant_expression
```

缺省值名必须遵循关于标识符的规则。只可在当前数据库中创建缺省值。

在一个数据库中，缺省值名对每一个用户必须是唯一的。例如，不能创建两个名为 phonedflt 的缺省值。然而，作为 “guest”，即使 dbo.phonedflt 已经存在，仍可创建 phonedflt，因为所有者名将每一个区分开了。

另一个示例：假设要创建一个缺省值 “Oakland”，它可用于 friends_etc 的 city 列，并且可能用于其它列或用户数据类型。若要创建该缺省值，请输入：

```
create default citydflt
as "Oakland"
```

继续该例，您可使用与将输入到人员表中的人相匹配的任何城市名。

将字符和日期常量用引号引起来；货币、整型和浮点常量不需要这样。二进制数据前面必须带有 “0x”，货币数据前面应带有美元符号 ($)，或代表您工作所在地区的逻辑缺省货币的任何货币符号。缺省值必须与列的数据类型兼容。例如，不可以用 “none”作为数值型列的缺省值，但是可以用 0。

如果在创建列时指定 NOT NULL 并且没有将一个缺省值与之相连，则无论何时有人未给该列输入值， Adaptive Server 都将给出出错信息。

通常，您在创建表时输入缺省值。然而，对于输入在一列或多列中有相同值的许多行的会话，在开始之前创建一个是适合于该会话的缺省值会带来很大方便。

注释　不能使用声明缺省值发出 create table，然后在同一批处理或过程中将数据插入表中。可以将创建和插入语句分放在两个不同的批处理或过程中，或者使用 execute 分别执行操作。

绑定缺省值

创建缺省值后，使用 sp_bindefault 将缺省值绑定到列或用户定义的数据类型。例如，如果创建以下缺省值：

```
create default advancedflt as "UNKNOWN"
```

现在，使用 sp_bindefault 将缺省值绑定到适当的列或用户定义数据类型上。

```
sp_bindefault advancedflt, "titles.advance"
```

只有当用户不向 titles 表的 advance 列添加条目时，缺省值才有效。不输入条目与输入空值是不同的。一个缺省值可与一个特定列、许多列或数据库中具有特定用户定义数据类型的所有列相关联。

注释　若要获取缺省值，必须对列列表 （不包括具有缺省值的列）发出 insert 或 update 命令。

以下限制适用于缺省值：

- 缺省值仅适用于新行。它并不追溯性地改变已存在的行。仅当无条目输入时缺省值才生效。如果给列赋予任何值，包括 NULL，缺省值都是无效的。
- 不能将缺省值绑定到系统数据类型。
- 不能绑定缺省值到 timestamp 列，因为 Adaptive Server 为 timestamp 列生成值。
- 不能将缺省值绑定到系统表。
- 可以将缺省值绑定到 IDENTITY 列或有 IDENTITY 属性的用户定义数据类型，但 Adaptive Server 将忽略这样的缺省值。当没有为 IDENTITY 列指定值而将一行插入表中时， Adaptive Server 分配一个比上一次分配的值大 1 的值。
- 如果列上已存在缺省值，那么在绑定新的缺省值之前必须将其删除。使用 sp_unbindefault 可删除用 sp_bindefault 创建的缺省值。使用 alter table 删除用 create table 创建的缺省值。

若要将 citydflt 绑定到 friends_etc 中的 city 列上，请输入：

```
sp_bindefault citydflt, "friends_etc.city"
```

由于嵌入式标点符号 （句点）的原因，应使用引号将表名和列名引起来。

如果为数据库的每个表中的所有 city 列创建了一个特殊的数据类型，并将 citydflt 绑定到该数据类型，则仅在 city 名适合时才显示 “Oakland”。例如，如果用户数据类型名为 citytype，则将 citydflt 绑定到它的方法为：

```
sp_bindefault citydflt, citytype
```

为防止现有列或特定用户数据类型继承新的缺省值，当将缺省值绑定到用户数据类型时，请使用 futureonly 参数。然而，将缺省值绑定到列时不要使用 futureonly。以下是创建一个新缺省值 “Berkeley”并将它绑定到数据类型 citytype 以仅供表的新列使用的方法：

```
create default newcitydflt as "Berkeley"
sp_bindefault newcitydflt, citytype, futureonly
```

对任何使用 citytype 的现有表列，“Oakland”继续作为缺省值出现。

如果表中的大多数人居住的地区使用相同的邮政编码，则可以创建一个缺省值来保存数据输入时间。以下即是一例，其绑定缺省值适用于 Oakland 的一个区域：

```
create default zipdflt as "94609"
sp_bindefault zipdflt, "friends_etc.postalcode"
```

以下是 sp_bindefault 的完整语法：

sp_bindefault *defname, objname* [, futureonly]

defname 是用 create default 创建的缺省值的名称。*objname* 是缺省值绑定的表、列或用户定义数据类型名。如果参数的形式不是 *table.column*，它将被认为是用户定义的数据类型。

指定的用户定义数据类型的所有列将与指定的缺省值相关联，除非使用可选参数 futureonly，该参数阻止该用户数据类型的现有列继承缺省值。

注释　不能在同一批处理过程中将缺省值绑定到列并使用它们。sp_bindefault 不能与调用缺省值的 insert 语句在同一批处理中。

解除绑定缺省值

解除绑定缺省值意味着断开其与特定列或用户定义数据类型的关联。解除绑定的缺省值仍保存在数据库中且将来可以使用。使用 sp_unbindefault 删除缺省值与列或数据类型之间的绑定。

可按如下方式，从 friends_etc 表的 city 列中解除绑定当前缺省值：

```
execute sp_unbindefault "friends_etc.city"
```

若要从用户定义数据类型 citytype 中解除绑定缺省值，请使用：

```
sp_unbindefault citytype
```

sp_unbindefault 的完整语法是：

sp_unbindefault *objname* [, futureonly]

如果给定的 *objname* 参数的形式不是 *table.column*，Adaptive Server 假设它为用户定义的数据类型。从用户定义的数据类型中解除绑定缺省值时，将从具有此类型的所有列中解除绑定该缺省值，除非指定可选 futureonly 参数，该参数禁止具有此类型的现有列解除与缺省值的绑定。

缺省值怎样影响 NULL 值

如果在创建列时指定 NOT NULL 并且没有为其创建缺省值，则无论何时有人插入一行并且没有在该列中输入值， Adaptive Server 都会产生错误消息。

删除了 NULL 列的缺省值后，每次增加一行而没有给该列输入值时，Adaptive Server 都将在该位置上插入 NULL。删除了 NOT NULL 列的缺省值后，当增加一行而没有给该列输入值时，将会产生出错信息。

表 14-1 说明缺省值的存在和 NULL 或 NOT NULL 列定义之间的关系。

表 14-1：列定义和空缺省值

列定义	用户输入	结果
已定义空和缺省值	没有值	使用缺省值
	NULL 值	使用 NULL
已定义 Null，未定义缺省值	没有值	使用 NULL
	NULL 值	使用 NULL
非空，已定义缺省值	没有值	使用缺省值
	NULL 值	错误
非空，未定义缺省值	没有值	错误
	NULL 值	错误

创建缺省值后

创建缺省值后，描述缺省值的**源文本**存储在 syscomments 系统表的 text 列上。*请勿删除此信息*；这样或许会使 Adaptive Server 的将来版本出现问题。而应使用 《参考手册》中说明的 sp_hidetext 对 syscomments 中的文本加密。有关详细信息，请参见第 3 页的 “编译对象”。

删除缺省值

要从整个数据库中删除缺省值，请使用 drop default 命令。在删除缺省值前，解除其与所有行和用户数据类型的绑定。（请参见第 439 页的 “解除绑定缺省值”。）如果试图删除仍旧绑定的缺省值，Adaptive Server 会显示出错信息且 drop default 命令会失败。

可按如下方式删除 citydflt。首先，解除绑定：

```
sp_unbindefault citydft
```

接下来，可以删除 citydft：

```
drop default citydflt
```

drop default 的完整语法是：

```
drop default [owner.]default_name
  [, [owner.]default_name] ...
```

缺省值只能由其所有者删除。请参见 《参考手册：过程》和 《参考手册：命令》。

创建规则

可通过规则指定，在特定的列中或具有用户定义数据类型的任意列中，用户能够输入哪些内容以及不能输入哪些内容。通常，要创建规则：

1　使用 create rule 创建规则。

2　使用 sp_bindrule 将规则绑定到一列或用户定义的数据类型。

3　通过插入数据测试绑定规则。仅通过使用 insert 或 update 命令进行测试，便可捕获在创建和绑定规则时产生的许多错误。

通过使用 sp_unbindrule 或将新规则绑定到列或数据类型，可解除规则到该列或数据类型的绑定。

create rule 语法

create rule 的语法是：

```
create rule [owner.]rule_name
   as condition_expression
```

规则名必须遵循标识符的规则。只可在当前数据库中创建规则。

在数据库中，规则名对每一个用户必须是唯一的。例如，一个用户不可以创建两个名为 socsecrule 的规则。然而，两个不同的用户都可以创建名为 socsecrule 的规则，因为所有者名将它们区分开了。

可按如下方式，创建允许五个不同 pub_id 号和一个伪值 （99 后跟任意两个数字）的规则：

```
create rule pub_idrule
as @pub_id in ("1389", "0736", "0877", "1622", "1756")
or @pub_id like "99[0-9][0-9]"
```

as 子句包含前缀为 “@”的规则参数名和规则自身的定义。参数指受 update 或 insert 语句影响的列值。

在前面的例子中，参数是 @pub_id，这是一个便于记忆的名字，因为此规则将绑定到 pub_id 列上。参数可以使用*任何*名称，但是第一个字符必须是 “@”。通过使用将规则绑定到的列或数据类型的名称，可以帮助您记住其用途。

规则定义可包括任何在 where 子句中合法的表达式，并可包括算术运算符、比较运算符、 like、 in、 between 等。然而，规则定义不能直接引用任何列或其它数据库对象。*可以*包括不引用数据库对象的内置函数。

以下示例创建一个规则，它强制输入值与特定 “样式”保持一致。在本示例中，在列中输入的每个值必须以 “415”开头，后跟 7 个其它字符：

```
create rule phonerule
as @phone like "415_______"
```

若要确保所输入的朋友年龄在 1 到 120 之间，但不可以是 17，请尝试以下方法：

```
create rule agerule
as @age between 1 and 120 and @age ! = 17
```

绑定规则

创建规则后，使用 sp_bindrule 将规则链接到列或用户定义数据类型。

以下是 sp_bindrule 的完整语法：

sp_bindrule *rulename*, *objname* [, futureonly]

rulename 是用 create rule 创建的规则的名称。*objname* 是要将规则绑定到的表和列或用户定义数据类型的名称。如果参数的形式不是 *table.column*，它将被认为是用户数据类型。

仅当将规则绑定到用户定义的数据类型时，才会使用可选的 futureonly 参数。指定的用户定义数据类型的所有列与指定规则相关联，除非指定 futureonly，此参数使该用户数据类型的现有列不继承规则。如果与给定用户定义的数据类型相关联的规则已被改变， Adaptive Server 为该用户定义的数据类型的现有列维护此改变的规则。

以下限制适用于规则：

- 不能将规则绑定到 text、 unitext、 image 或 timestamp 数据类型列上。
- Adaptive Server 不允许规则在系统表上。

绑定到列的规则

使用 sp_bindrule 以及规则名、用引号引起来的表名和列名，将规则绑定到列。以下是将 pub_idrule 绑定到 publishers.pub_id 的方法：

```
sp_bindrule pub_idrule, "publishers.pub_id"
```

以下是一个规则，它可确保输入的所有邮政编码将 946 作为前 3 位：

```
create rule postalcoderule946
as @postalcode like "946[0-9][0-9]"
```

可按如下方式，将它绑定到 friends_etc 中的 postalcode 列上：

```
sp_bindrule postalcoderule946,
"friends_etc.postalcode"
```

同一批处理过程中不能将规则绑定到列上并使用它。 sp_bindrule 不能与调用规则的 insert 语句的在同一批处理中。

绑定到用户定义数据类型的规则

不可以将规则绑定到系统数据类型上，但可以将其绑定到用户定义数据类型上。若要将 phonerule 绑定到名为 p# 的用户定义数据类型，请输入：

```
sp_bindrule phonerule, "p#"
```

规则的优先顺序

绑定到列的规则总是比绑定到用户数据类型的规则具有更高的优先级。将规则绑定到列后，将替换绑定到该列的用户数据类型的规则。但是，将规则绑定到数据类型，则不会替换绑定到该用户数据类型的列的规则。

只有在试图向用户定义数据类型的数据库列插入值或更新它时，绑定到用户定义数据类型的规则才被激活。由于规则不测试变量，因此不要将值赋予将被绑定到同一数据类型的规则拒绝的用户定义数据类型变量。

表 14-2 指出了将规则绑定到已存在规则的列和用户数据类型时的优先级：

表 14-2：规则的优先顺序

新规则绑定到	**旧规则绑定到**	
	用户数据类型	*列*
用户数据类型	替换旧规则	没有变化
列	替换旧规则	替换旧规则

在某些列上输入需要特殊临时约束的数据时，可创建一个新规则来帮助检测这些数据。例如，假设要向 friends_etc 表的 debt 列添加数据。已知今天要记录的所有 debt 都在 $5 和 $200 之间。为避免意外键入这些限制以外的金额，可创建如下规则：

```
create rule debtrule
as @debt = $0.00 or @debt between $5.00 and $200.00
```

@debt 规则定义允许输入 $0.00 以保留原来为此列定义的缺省值。

可按如下方式，将 debtrule 绑定到 debt 列上：

```
sp_bindrule debtrule, "friends_etc.debt"
```

规则和 NULL 值

不能定义一个允许空的列，接着又用禁止空值的规则覆盖此定义。例如，如果列定义指定 NULL 而规则指定如下，则隐式或显式的 NULL 并不违背规则：

```
@val in (1,2,3)
```

此列定义覆盖此规则，即使规则指定：

```
@val is not null
```

定义规则后

定义规则后，描述规则的**源文本**存储在 syscomments 系统表的 text 列中。*请勿删除此信息*；这样或许会使 Adaptive Server 的将来版本出现问题。而应使用 sp_hidetext 对 syscomments 中的文本加密。请参见 《参考手册：过程》和第 3 页的 “编译对象”。

解除绑定规则

解除绑定规则即断开其与特定列或用户定义数据类型的关联。解除绑定的规则定义仍存储在数据库中且将来可以使用。

有两种解除绑定规则的方法：

- 使用 sp_unbindrule 解除规则和列或用户定义数据类型之间的绑定。
- 使用 sp_bindrule 将新规则绑定到该列或数据类型。旧规则将自动解除绑定。

可按如下方式，从 friends_etc.debt 中解除 debtrule（或任何其它当前绑定的规则）关联：

```
sp_unbindrule "friends_etc.debt"
```

规则仍在数据库中，但与 friends_etc.debt 无关联。

若要从用户定义数据类型 p# 上解除绑定规则，请使用：

```
sp_unbindrule "p#"
```

sp_unbindrule 的完整语法是：

sp_unbindrule *objname* [, futureonly]

如果使用的 *objname* 参数的形式不是 *"table.column"*，则 Adaptive Server 假设它为用户定义数据类型。当从用户定义数据类型解除绑定规则时，该规则从该类型的所有列上解除绑定，除非：

- 给出可选的 futureonly 参数，它使该数据类型的现有列不会失去与规则的绑定，或
- 该用户定义数据类型的列上的规则已被改变，因此其当前值与绑定的规则不同。

删除规则

要从整个数据库中删除规则，请使用 drop rule 命令。在删除规则前，解除其与所有行和用户数据类型的绑定。如果试图删除仍旧绑定的规则，Adaptive Server 会显示出错信息且 drop rule 会失败。当然，要绑定新规则，不必先解除绑定接着再删除规则。只需在其位置上绑定一个新规则。

若要在解除绑定后删除 phonerule，请使用以下命令：

```
drop rule phonerule
```

drop rule 的完整语法是：

drop rule [*owner*.]*rule_name*
    [, [*owner*.]*rule_name*] ...

删除规则后，新数据输入先前由该规则控制的列时，将没有这些约束。不管怎样，已存在的数据都不会受到影响。

规则只能由其所有者删除。

获得有关缺省值和规则的信息

系统过程 sp_help 与表名一起使用时，会显示绑定到列的规则和缺省值。以下示例显示有关 pubs2 数据库中 authors 表的信息，包括规则和缺省值：

```
sp_help authors
```

sp_help 也报告绑定到用户定义数据类型的规则。若要检查规则是否绑定到用户定义数据类型 p#，请使用：

```
sp_help "p#"
```

sp_helptext 报告规则或缺省值的定义 （create 语句）。

如果缺省值或规则的源文本已使用 sp_hidetext 加密，则 Adaptive Server 会显示一条通知您文本已隐藏的消息。请参见 《参考手册：过程》。

如果系统安全员已经使用 sp_configure 重设了 allow select on syscomments.text column 参数 （这是在已评估的配置中运行 Adaptive Server 所必需的），则您必须是缺省值或规则的创建者或系统管理员，才能通过 sp_helptext 查看缺省值或规则的文本。请参见 《系统管理指南：卷 1》中的第 12 章 “安全性简介”。

第 15 章

使用批处理和控制流语言

Transact-SQL 允许交互地或在操作系统文件中将一系列语句组合为批处理。也可以通过编程结构使用 Transact-SQL 的控制流语言连接语句。

变量是一个被赋值的实体。在使用此变量的批处理或存储过程的执行期间，此值可以变化。 Adaptive Server 有两种类型的变量：**局部的**和**全局的**。局部变量是用户定义的，而全局变量是由系统提供并预定义的。

简介

为了说明这一点，本手册中的每个示例都包括单个语句。每次只向 Adaptive Server 提交一个语句，交互地输入语句并接收结果。

Adaptive Server 也可以交互地或通过文件将多个语句作为批处理提交。批处理或批处理文件是被一同提交的 Transact-SQL 语句，并作为一组依次执行。批处理用批结束符号来终止。对于 isql 实用程序，批结束符号是独占一行的 “go” 字。有关 isql 的详细信息，请参见 《实用程序指南》。

以下是一个包含两个 Transact-SQL 语句的批处理示例：

```
select count(*) from titles
select count(*) from authors
go
```

从技术角度上说，单个 Transact-SQL 语句可以构成一个批处理，但是通常认为批处理包含多个语句。通常将语句的批处理编写为操作系统文件，然后才提交给 isql。

Transact-SQL 提供称为控制流语言的特殊关键字，允许用户控制语句执行流。控制流语言可用于单个语句、批处理、存储过程或触发器。

不使用控制流语言，独立的 Transact-SQL 语句按其出现的顺序依序执行。在第 5 章 “子查询：在其它查询中使用查询”中论述的相关子查询是部分例外。控制流语言使用类似于程序的结构使语句相互连接和关联。

通过使用控制流语言 （如用于命令的条件执行的 if...else 和用于重复执行的 while），您可以优化和控制 Transact-SQL 语句的执行情况。Transact-SQL 控制流语言将标准 SQL 转换为更高级的编程语言。

批处理相关的规则

有一些规则约束哪些 Transact-SQL 语句可组合到单个批处理中。这些批处理规则如下：

- 引用数据库中的对象之前，请对该数据库发出 use 语句。例如：

  ```
  use master
  go
  select count(*)
  from sysdatabases
  go
  ```

- 在批处理中 *不能* 将以下数据库命令与其它语句组合使用：
  - create procedure
  - create rule
  - create default
  - create trigger
- 在批处理中 *可以* 将以下数据库命令与其它 Transact-SQL 语句组合使用：
  - create database （除非在单个批处理中不能创建数据库并创建或访问此新数据库中的对象）
  - create table
  - create index
  - create view

- 不能在同一批处理中将规则和缺省值绑定到列中，然后使用它们。sp_bindrule 和 sp_bindefault 不能与调用该规则或缺省值的 insert 语句在同一个批处理中。
- 不能在同一批处理中 drop 对象，然后又引用或重新创建它。
- 如果表已经存在，则不能在批处理中重新创建它，即使批处理中包括对表的存在性的测试。

  Adaptive Server 先编译批处理，然后执行它。编译期间， Adaptive Server 不对批处理所引用的对象 （如表和视图）进行权限检查。Adaptive Server 只在执行批处理时才检查权限。当 Adaptive Server 访问的不是当前数据库时情况例外。这种情况下， Adaptive Server 在编译时显示一条错误消息，不执行批处理中的任何语句。

  假定批处理中包含以下这些语句：

  ```
  select * from taba
  select * from tabb
  select * from tabc
  select * from tabd
  ```

  如果具有除第三条语句 (select * from tabc) 之外的所有语句的必要权限，则 Adaptive Server 会返回有关第三条语句的错误消息和所有其它语句的结果。

使用批处理的示例

本节中的示例说明使用 isql 实用程序格式的批处理，该实用程序具有明显的批结束符号，即独占一行的 “go”单词。以下是一个包含两个 select 语句的批处理：

```
select count(*) from titles
select count(*) from authors
go

-------------
          18

(1 row affected)
-------------
          23

(1 row affected)
```

可以在同一批处理中创建表然后引用它。以下批处理创建表，向其中插入一行并从中选择所有列：

```
create table test
(column1 char(10), column2 int)
insert test
values ("hello", 598)
select * from test
go

(1 row affected)
column1  column2
-------  -------
hello        598

(1 row affected)
```

只要后续语句中引用的对象仍在开始时打开的数据库中，就可以将 use 语句与其它语句结合使用。以下批处理从 master 数据库中选择一个表，然后打开 pubs2 数据库。批处理首先将 master 数据库标记为当前数据库；然后， pubs2 成为当前数据库。

```
use master
go
select count(*) from sysdatabases
use pubs2
go

-------------
           6

(1 row affected)
```

只要在同一批处理中不引用或重新创建删除的对象，就可以将 drop 语句与其它语句结合使用。下例将 drop 语句与 select 语句结合使用：

```
drop table test
select count(*) from titles
go
------------
         18

(1 row affected)
```

只要批处理中有一个语法错误，就不会执行其中任何语句。例如，以下批处理在最后的语句中有键入错误，其结果为：

```
select count(*) from titles
select count(*) from authors
slect count(*) from publishers
```

```
go

Msg 156, Level 15, State 1:
Line 3:
Incorrect syntax near the keyword 'count'.
```

与批处理规则相冲突的批处理也生成错误消息。以下是一些非法批处理的示例：

```
create table test
    (column1 char(10), column2 int)
insert test
    values ("hello", 598)
select * from test
create procedure testproc as
    select column1 from test
go

Msg 111, Level 15, State 7:
Line 6:
CREATE PROCEDURE must be the first command in a
query batch.


create default phonedflt as "UNKNOWN"
sp_bindefault phonedflt, "authors.phone"
go

Msg 102, Level 15, State 1:
Procedure 'phonedflt', Line 2:
Incorrect syntax near 'sp_bindefault'.
```

如果您已经在 use 语句所指定的数据库中，则可以执行下一个批处理。但是，如果试图从其它数据库（例如 master）中执行该批处理，将显示一条错误消息。

```
use pubs2
select * from titles
go

Msg 208, Level 16, State 1:
Server 'hq', Line 2:
titles not found. Specify owner.objectname or use
sp_help to check whether the object exists (sp_help may
produce lots of output)

drop table test
create table test
(column1 char(10), column2 int)
go
```

```
Msg 2714, Level 16, State 1:
Server 'hq', Line 2:
There is already an object named 'test' in the
database.
```

作为文件提交的批处理

可以通过操作系统文件向 isql 提交一个或多个 Transact-SQL 语句的批处理。一个文件可以包括多个批处理，即包括多个语句集合，每个语句集合以 “go”字结束。

例如，操作系统文件可能包含以下三个批处理：

```
use pubs2
go
select count(*) from titles
select count(*) from authors
go
create table hello
   (column1 char(10), column2 int)
insert hello
   values ("hello", 598)
select * from hello
go
```

以下是将此文件提交到 isql 实用程序时输出的结果：

```
-------------
           18

(1 row affected)
-------------
           23

(1 row affected)
column1      column2
---------    ---------
hello              598

(1 row affected)
```

有关在特定环境下运行存储在文件中的批处理的信息，请参见 《实用程序指南》中的 isql。

使用控制流语言

可以在批处理和存储过程中将控制流语言用于交互式语句。表 15-1 列出了控制流和相关关键字及其功能。

表 15-1：控制流和相关关键字

关键字	函数
if	定义条件执行。
…else	定义当 if 条件为假时的替代执行。
case	使用 when...then 语句代替 if...else 定义条件表达式。
begin	语句块的起点。
…end	语句块的终点。
while	条件为真时重复执行语句。
break	从下一外层 while 循环结束处退出。
…continue	重新启动 while 循环。
declare	声明局部变量。
goto label	跳转到语句块中的 label: 位置。
return	无条件退出。
waitfor	设置命令执行的延迟。
print	在用户屏幕上输出用户定义的消息或局部变量。
raiserror	在用户屏幕上输出用户定义的消息或局部变量，并在全局变量 *@@error* 中设置系统标志。
/* *comment* */ 或者 *--comment*	在 Transact-SQL 语句中任意位置插入注释。

if...else

关键字 if （无论有没有协同的 else）引入确定是否执行下一语句的条件。如果条件满足，也就是说条件返回 TRUE，则执行 Transact-SQL 语句。

else 关键字引入了一个替代 Transact-SQL 语句，在 if 条件返回 FALSE 时执行该语句。

if 和 else 的语法为：

```
if
        boolean_expression
        statement
    [else
        [if boolean_expression]
            statement ]
```

布尔表达式返回 TRUE 或 FALSE。它可以包括列名、常量、使用算术运算符或逐位运算符连接的列名和常量的任意组合、或子查询（只要子查询返回单个值）。如果布尔表达式包含 select 语句，则 select 语句必须用小括号括起来，并且必须返回单个值。

以下示例只使用 if：

```
if exists (select postalcode from authors
         where postalcode = "94705")
print "Berkeley author"
```

如果 authors 表中的一个或多个邮政编码的值为 “94705”，则输出消息 “Berkeley author”。因为使用了关键字 exists，所以此示例中的 select 语句返回单个值， TRUE 或 FALSE。 exists 关键字在此的功能与其在子查询中的功能相同。请参见第 5 章 “子查询：在其它查询中使用查询”。

以下示例使用 if 和 else，用于测试是否存在 ID 号大于 50 的用户创建对象。如果这些用户对象存在， else 子句将选择其名称、类型和 ID 号。

```
if (select max(id) from sysobjects) < 50
   print "There are no user-created objects in this
database."
else
   select name, type, id from sysobjects
   where id > 50 and type = "U"

(0 rows affected)

 name           type   id
------------    ----   ---------
 authors        U       16003088
 publishers     U       48003202
 roysched       U       80003316
 sales          U      112003430
 salesdetail    U      144003544
 titleauthor    U      176003658
 titles         U      208003772
 stores         U      240003886
 discounts      U      272004000
 au_pix         U      304004114
 blurbs         U      336004228
 friends_etc    U      704005539
 test           U      912006280
 hello          U     1056006793

(14 rows affected)
```

if...else 结构经常用在存储过程中，用来测试某个参数的存在性。

if 测试可嵌套在其它 if 测试中，在另一 if 中或在 else 后。if 测试中的表达式只能返回单个值。同样，每个 if...else 结构可以有一个 if 语句和一个 else 语句。要包括多个选择语句，请使用 begin...end 关键字。根据每个 if...else 结构包括的 select 语句（或其它语言结构）的复杂性，可嵌套的 if 测试的最大数目也会有所变化。

case 表达式

case 表达式简化了许多条件 Transact-SQL 结构。无需使用一系列 if 语句， case 表达式允许使用一系列条件，在条件满足时返回适当的值。case 表达式符合 ANSI SQL。

使用 case 表达式，可以：

- 简化查询并编写更有效的代码
- 在数据库所使用的格式 （如 int）和应用程序所使用的格式 （如 char）之间转换数据
- 返回列列表中的第一个非空值
- 编写避免除以零的查询

  注释　如果除数的求值结果为常量表达式，则 case 无法避免除数为 0 的错误，因为 Adaptive Server *先* 计算常量表达式，然后才执行 case 逻辑。这有时会导致除数为零。

  解决办法：

  - 使用 nullif()。例如：

    ```
    (x/nullif(@foo.0)0
    ```

  - 包括列值，以便它们相互约去，强制 Adaptive Server 为每行计算表达式。例如：

    ```
    (x/(@foo + (col1 - col1))
    ```

- 比较两个值，值不匹配时返回第一个值，值匹配时返回 NULL 值

case 表达式包括关键字 case、 when、 then、 coalesce 和 nullif。 coalesce 和 nullif 是 case 表达式的缩写形式。请参见 《参考手册：命令》。

在替代表示中使用 *case* 表达式

使用 case 表达式可以采用对用户更有意义的方式表示数据。例如，pubs2 数据库在 titles 表的 contract 列中存储 1 或 0 来指示书籍合同的状态。但是，在您的应用程序代码中或用于用户交互时，最好使用词语“Contract”或“No Contract”来指示书籍的状态。若要使用替代表示从 titles 表中选择类型，请使用以下命令：

```
select title, "Contract Status" =
    case
        when contract = 1 then "Contract"
        when contract = 0 then "No Contract"
    end
from titles
```

```
title                                Contract Status
-----                                ---------------
The Busy Executive's Database Guide  Contract
Cooking with Computers: Surreptitio  Contract
You Can Combat Computer Stress!      Contract
. . .
The Psychology of Computer Cooking   No Contract
. . .
Fifty Years in Buckingham Palace     Contract
Sushi, Anyone?                       Contract

(18 rows affected)
```

case 和除数为零

利用 case 表达式，可以编写避免除数为零（称为意外排除）的查询。例如，如果试图用 advance 列去除每本书的 total_sales 列，则当查询试图用 advance (0.00) 去除 title_id MC2222 的 total_sales (2032) 时，查询会导致除数为零：

```
select title_id, total_sales, advance,
total_sales/advance from titles
```

```
title_id   total_sales          advance
-------    -----------          ---------     ------
BU1032     4095                   5,000.00     0.82
BU1111     3876                   5,000.00     0.78
BU2075     18722                 10,125.00     1.85
BU7832     4095                   5,000.00     0.82

Divide by zero occurred.
```

可以使用 case 表达式，通过不允许在等式中包括零来避免出现这种情况。在此示例中，查询碰到 0 时，返回预定义的值，而不是执行此除法：

```
select title_id, total_sales, advance, "Cost Per Book" =
     case
           when advance != 0
           then convert(char, total_sales/advance)
           else "No Books Sold"
     end
from titles
```

```
title_id       total_sales     advance          Cost Per Book
--------       -----------     ----------       -------------
BU1032                4095       5,000.00                0.82
BU1111                3876       5,000.00                0.78
BU2075               18722      10,125.00                1.85
BU7832                4095       5,000.00                0.82
MC2222                2032           0.00       No Books Sold
MC3021               22246      15,000.00                1.48
MC3026                NULL           NULL       No Books Sold
. . .
TC3218                 375       7,000.00                0.05
TC4203               15096       4,000.00                3.77
TC7777                4095       8,000.00                0.51

(18 rows affected)
```

title_id MC2222 的除数为零情况不再会阻止查询运行。同样， MC3021 的空值也不会阻止查询的运行。

在 case 表达式中使用 rand() 函数

每次对引用 rand 和 getdate 等函数的表达式进行求值时，它们将生成不同的值。当您在某些 case 表达式中使用这些表达式时上述情况可能造成意外结果。例如， SQL 标准指定，具有以下形式的 case 表达式：

```
case expression
    when value1 then result1
    when value2 then result2
    when value3 then result3
...
end
```

与以下形式的 case 表达式等效：

```
case expression
    when expression=value1 then result1
    when expression=value2 then result2
```

```
    when expression=value3 then result3
...
end
```

此定义显式地要求在检查的每个 when 子句中反复对表达式进行求值。此 case 表达式的定义影响引用函数（如 rand 函数）的 case 表达式。例如，根据 SQL 标准，将以下 case 表达式：

```
select
CASE convert(int, (RAND() * 3))
       when 0 then "A"
       when 1 then "B"
       when 2 then "C"
       when 3 then "D"
   else "E"
end
```

定义为等效于以下内容：

```
 select
CASE
       when convert(int, (RAND() * 3)) = 0 then "A"
       when convert(int, (RAND() * 3)) = 1 then "B"
       when convert(int, (RAND() * 3)) = 2 then "C"
       when convert(int, (RAND() * 3)) = 3 then "D"
   else "E"
end
```

在此形式中，为每一 when 子句都生成一个新的 rand 值，并且 case 表达式频繁生成结果 “E”。

case 表达式结果

用于确定 case 表达式的数据类型的规则所基于的规则与确定 union 操作中列的数据类型的规则相同。 case 表达式具有一系列替代结果表达式（如以下示例中的 *R1*、 *R2*、 ...、 *Rn*），它们是由 then 和 else 子句指定的。例如：

```
case
      when search_condition1 then R1
      when search_condition2 then R2
      ...
      else Rn
   end
```

结果表达式 *R1*, *R2*, ..., *Rn* 的数据类型用于确定整个 case 的数据类型。union 列的数据类型用于指定 *n* 个表，并将表达式 *R1*, *R2*, ..., *Rn* 作为第 *i* 列，确定该数据类型的规则与确定 case 表达式的数据类型的规则相同。case 数据类型的确定方式与以下查询中的确定方式相同：

```
select...R1...from ...
union
select...R2...from...
union...
...
select...Rn...from...
```

并非所有数据类型都是兼容的，如果指定两个不兼容的数据类型 （例如， *char* 和 *int*）， Transact-SQL 查询将会失败。请参见 《参考手册：构件块》。

case 表达式要求至少返回一个非空值

至少有一个 case 表达式的结果必须返回非空值。以下查询：

```
select price,
    case
        when title_id like "%" then NULL
        when pub_id like "%" then NULL
    end
from titles
```

返回错误消息：

```
All result expressions in a CASE expression must not be
NULL
```

case

使用 case 表达式可以测试确定结果集的条件。

语法为：

```
case
      when search_condition1 then result1
      when search_condition2 then result2
      . . .
      when search_conditionn then resultn
      else resultx
   end
```

其中， *search_condition* 是逻辑表达式， *result* 是表达式。

如果 *search_condition1* 为真，则 case 的值为 *result1* ；如果 *search_condition1* 为假，则检查 *search_condition2*。如果 *search_condition2* 为真，则 case 的值为 *result2*，依此类推。如果搜索条件都为假，则 case 的值为 *resultx*。 else 子句是可选的。如果没有使用它，则缺省为 else NULL。 end 指示 case 表达式结束。

每个书店的每本书的总销售额保存在 salesdetail 表中。要显示一系列范围的书籍销售额，可以使用以下范围来跟踪每个书店中每本书的销售额：

- 销售量小于 1000 的书籍 （低销售额的书籍）
- 销售量在 1000 与 3000 之间的书籍 （中等销售额的书籍）
- 销售量大于 3000 的书籍 （高销售额的书籍）

编写以下查询：

```
select stor_id, title_id, qty, "Book Sales Catagory" =
     case
           when qty < 1000
              then "Low Sales Book"
           when qty >= 1000 and qty <= 3000
              then "Medium Sales Book"
           when qty > 3000
           then "High Sales Book"
     end
from salesdetail
group by title_id
```

```
stor_id     title_id      qty      Book Sales Catagory
-------     --------      ----     -------------------
5023        BU1032         200     Low Sales Book
5023        BU1032        1000     Low Sales Book
7131        BU1032         200     Low Sales Book
. . .
7896        TC7777          75     Low Sales Book
7131        TC7777          80     Low Sales Book
5023        TC7777        1000     Low Sales Book
7066        TC7777         350     Low Sales Book
5023        TC7777        1500     Medium Sales Book
5023        TC7777        1090     Medium Sales Book

(116 rows affected)
```

以下示例根据 author 的版税率 (*royaltyer*) 从 titleauthor 表中选择 title，并为每个 title 分配一个高、中或低版税值：

```
select title, royaltyper, "Royalty Category" =
   case
      when (select avg(royaltyper) from titleauthor tta
        where t.title_id = tta.title_id) > 60 then "High Royalty"
      when (select avg(royaltyper) from titleauthor tta
        where t.title_id = tta.title_id) between 41 and 59
      then "Medium Royalty"
      else "Low Royalty"
   end
from titles t, titleauthor ta
where ta.title_id = t.title_id
order by title
```

```
title                                 royaltyper   royalty Category
-------                               ----------   ----------------
But Is It User Friendly?              100          High Royalty
Computer Phobic and Non-Phobic Ind    25           Medium Royalty
Computer Phobic and Non-Phobic Ind    75           Medium Royalty
Cooking with Computers: Surreptiti    40           Medium Royalty
Cooking with Computers: Surreptiti    60           Medium Royalty
Emotional Security: A New Algorith    100          High Royalty
. . .
Sushi, Anyone?                        40           Low Royalty
The Busy Executive's Database Guide   40           Medium Royalty
The Busy Executive's Database Guide   60           Medium Royalty
The Gourmet Microwave                 75           Medium Royalty
You Can Combat Computer Stress!       100          High Royalty

(25 rows affected)
```

case 和值比较

这种形式的 case 用于值比较。它只允许进行两个值的等同性检查；而不允许其它的比较。

语法为：

```
case valueT
      when value1 then result1
      when value2 then result2
      . . .
      when valuen then resultn
      else resultx
   end
```

其中， *value* 和 *result* 是表达式。

如果 *valueT* 等于 *value1*，则 case 的值为 *result1*。如果 *valueT* 不等于 value1，则 *valueT* 与 *value2* 相比较。如果 *valueT* 等于 *value2*，则 case 的值为 *result2*，依此类推。如果 *valueT* 的值不等于 *value1* 到 *valuen*，则 case 的值为 *resultx*。

至少有一个结果必须是非空值。所有的结果表达式都必须是兼容的。同样，所有的 values 也必须都是兼容的。

上述语法等价于：

```
case
        when valueT = value1 then result1
        when valueT = value2 then result2
        . . .
        when valueT = valuen then resultn
        else resultx
    end
```

这与用于 case 和搜索条件的格式相同 （有关此语法的详细信息，请参见第 459 页的 “case”）。

以下示例从 titles 表中选择 title 和 pub_id，并基于 pub_id 指定每本书的出版社：

```
select title, pub_id, "Publisher" =
   case pub_id
     when "0736" then "New Age Books"
     when "0877" then "Binnet & Hardley"
     when "1389" then "Algodata Infosystems"
     else "Other Publisher"
   end
from titles
order by pub_id
```

```
title                          pub_id       Publisher
-----                          ------       -------------
Life Without Fear                0736       New Age Books
Is Anger the Enemy?              0736       New Age Books
You Can Combat Computer          0736       New Age Books
. . .
Straight Talk About Computers    1389       Algodata Infosystems
The Busy Executive's Database    1389       Algodata Infosystems
Cooking with Computers: Surre    1389       Algodata Infosystems

(18 rows affected)
```

这等效于以下使用 case 和搜索条件语法的查询：

```
select title, pub_id, "Publisher" =
   case
     when pub_id = "0736" then "New Age Books"
     when pub_id = "0877" then "Binnet & Hardley"
     when pub_id = "1389" then "Algodata Infosystems"
     else "Other Publisher"
   end
from titles
order by pub_id
```

coalesce

coalesce 检查一系列值（*value1*、*value2*、...、*valuen*）并返回第一个非空值。coalesce 的语法为：

coalesce(*value1*, *value2*, ..., *valuen*)

其中，*value1*, *value2*, ..., *valuen* 是表达式。如果 *value1* 为非空值，则 coalesce 的值为 *value1*；如果 *value1* 为空值，则检查 *value2*，依此类推。继续检查，直到找到非空值为止。第一个非空值成为 coalesce 的值。

使用 coalesce 时，Adaptive Server 在内部将其转换为以下格式：

```
case
    when value1 is not NULL then value1
    when value2 is not NULL then value2
    . . .
    when valuen-1 is not NULL then valuen-1
    else valuen
end
```

valuen-1 指倒数第二个值，在最后的值 *valuen* 前面。

以下示例使用 coalesce 来确定书店订购的书籍数量是低（大于 100 但小于 1000）还是高（大于 1000）：

```
select stor_id, discount, "Quantity" =
    coalesce(lowqty, highqty)
from discounts

stor_id discount    Quantity
------- --------   ---------   ----
NULL               10.500000   NULL
  NULL              6.700000    100
  NULL             10.000000   1001
8042                5.000000   NULL

(4 rows affected)
```

nullif

nullif 比较两个值；如果这两个值相等，则 nullif 返回空值。如果两个值不相等，则 nullif 返回其中第一个的值。这有利于查找以编码形式存储的任何遗失的、未知的或不适用的信息。例如，以前有时将未知值作为 -1 存储。使用 nullif，您可以用空值替代 -1 值，并获得 Transact-SQL 定义的空值行为。语法为：

nullif(*value1*, *value2*)

如果 *value1* 等于 *value2*， nullif 将返回 NULL。如果 *value1* 不等于 *value2*， nullif 将返回 *value1*。 *value1* 和 *value2* 是表达式，并且其数据类型必须是可比的。

使用 nullif 时， Adaptive Server 在内部将其翻译为以下格式：

```
case
      when value1 = value2 then NULL
      else value1
   end
```

例如， titles 表使用值 “UNDECIDED”来表示还没有确定类别的书。以下查询对 titles 表执行书籍类型的搜索；任何类型为 “UNDECIDED”的书籍都作为类型 NULL 返回 （以下输出出于显示目的重新设置了格式）：

```
select title, "type"=
   nullif(type, "UNDECIDED")
from titles

title                                    type
-----                                    --------
The Busy Executive's Database Guide      business
Cooking with Computers: Surreptiti       business
You Can Combat Computer Stress!          business
. . .
The Psychology of Computer Cooking       NULL
Fifty Years in Buckingham Palace K       trad_cook
Sushi, Anyone?                           trad_cook

(18 rows affected)
```

The Psychology of Computing 作为 “UNDECIDED”存储在表中，但查询将其作为类型 NULL 返回。

begin...end

begin 和 end 关键字将一系列语句包含在其中，以使控制流结构 （如 if...else）将其作为一个单元进行处理。涵括在 begin 和 end 中的一系列语句称为一个**语句块**。

begin...end 的语法为：

```
begin
        statement block
end
```

以下是一个示例：

```
if (select avg(price) from titles) < $15
begin
   update titles
   set price = price * 2

   select title, price
   from titles
   where price > $28
end
```

没有 begin 和 end， if 条件只应用于第一个 Transact-SQL 语句。第二个语句的执行独立于第一个语句。

begin...end 块可以嵌套在其它 begin...end 块中。

while 和 break...continue

while 设置重复执行语句或语句块的条件。只要指定条件为真，语句就会重复执行。

语法为：

```
while boolean_expression
    statement
```

在此示例中，只要平均价格低于 $30，就会重复执行 select 和 update 语句：

```
while (select avg(price) from titles) < $30
begin
   select title_id, price
   from titles
   where price > $20
   update titles
   set price = price * 2
```

```
end

(0 rows affected)

title_id    price
------      -------
PC1035        22.95
PS1372        21.59
TC3218        20.95

(3 rows affected)
(18 rows affected)
(0 rows affected)
title_id    price
------      -------
BU1032      39.98
BU1111      23.90
BU7832      39.98
MC2222      39.98
PC1035      45.90
PC8888      40.00
PS1372      43.18
PS2091      21.90
PS3333      39.98
TC3218      41.90
TC4203      23.90
TC7777      29.98

(12 rows affected)
(18 rows affected)
(0 rows affected)
```

break 和 continue 控制 while 循环中语句的执行。 break 导致从 while 循环中退出。执行在以 end 关键字作为循环结束标记之后的所有语句。 continue 导致 while 循环重新启动，并跳过此循环内 continue 之后的所有语句。 break 和 continue 经常由 if 测试激活。

break...continue 的语法为：

```
while boolean expression
    begin
            statement
        [statement]...
        break
        [statement]...
        continue
        [statement]...
    end
```

以下示例使用 while、 break、 continue 和 if 来平抑上述示例中导致的涨价情况。只要平均价格高于 $20，将其价格减半。然后选择最高价格。如果平均价格低于 40 美元，则退出 while 循环；否则将试图继续执行循环。仅在平均价格高于 20 美元时， continue 才允许执行 print。 while 循环结束后，将输出一条消息和价格最高的书籍列表。

```
while (select avg(price) from titles) > $20
begin
    update titles
        set price = price / 2
    if (select max(price) from titles) < $40
        break
    else
        if (select avg(price) from titles) < $20
            continue
    print "Average price still over $20"
end

select title_id, price from titles
    where price > $20

print "Not Too Expensive"
```

```
(18 rows affected)
(0 rows affected)
(0 rows affected)
Average price still over $20
(0 rows affected)
(18 rows affected)
(0 rows affected)

title_id    price
--------    -------
PC1035        22.95
PS1372        21.59
TC3218        20.95

(3 rows affected)
Not Too Expensive
```

如果嵌套了两个或多个 while 循环，则 break 退出到下一外层循环。首先，执行循环内部 end 之后的所有语句。然后，重新启动外部循环。

declare 和局部变量

局部变量用 declare 关键字来声明、命名和指定类型，并用 select 语句为其赋初始值。必须在同一个批处理或过程中声明、赋值并使用它们。

有关详细信息，请参见第 476 页的 “局部变量”。

goto

goto 关键字导致无条件地分岔转到用户定义标签。 goto 和标签可用于存储过程和批处理。标签的名称必须符合标识符规则，并且必须在首次给出时后接冒号。当与 goto 一起使用时，它不后接冒号。

语法如下：

label:
goto *label*

以下示例使用 goto 和标签、 while 循环和一个作为计数器的局部变量：

```
declare @count smallint
select @count = 1
restart:
print "yes"
select @count = @count + 1
while @count <=4
   goto restart
```

为了避免 goto 和标签之间的无穷循环，通常使 goto 依赖于 while 或 if 测试，或者其它一些条件。

return

return 关键字无条件地退出批处理或过程。它可用在批处理或过程中的任何位置。用于存储过程时， return 可以接收可选的参数以向调用者返回一个状态。 return 之后的语句不会执行。

语法为：

return [*int_expression*]

以下存储过程示例使用 return 以及 if...else 和 begin...end：

```
create procedure findrules @nm varchar(30) = null as
if @nm is null
begin
  print "You must give a user name"
```

```
  return
end
else
begin
   select sysobjects.name, sysobjects.id,
sysobjects.uid
   from sysobjects, master..syslogins
   where master..syslogins.name = @nm
   and sysobjects.uid = master..syslogins.suid
   and sysobjects.type = "R"
end
```

如果调用 findrules 时没有指定用户名参数，则 return 关键字将导致在将消息发送给用户屏幕之后退出此过程。如果指定了用户名，则从相应的系统表中检索用户所拥有的规则名称。

return 类似于在 while 循环中使用的 break 关键字。

有关使用返回值的示例，请参见 第 17 章 “使用存储过程”。

print

上一个示例中使用的 print 关键字在用户屏幕上显示用户定义的消息或局部变量的内容。必须在使用局部变量的批处理或过程中声明它。消息可以长达 255 个字节。

语法为：

print {*format_string* | *@local_variable* |
@@global_variable} [,*arg_list*]

例如：

```
if exists (select postalcode from authors
   where postalcode = "94705")
print "Berkeley author"
```

下面说明了如何使用 print 来显示局部变量内容：

```
declare @msg char(50)
select @msg = "What's up, doc?"
print @msg
```

print 可识别要输出的字符串中的占位符。格式字符串最多可包含 20 个任意顺序的唯一占位符。将消息文本发送到客户端时，使用 *format_string* 后接参数的格式化内容替换这些占位符。

为了在用其它语法结构将格式字符串翻译为某种语言时，可以对参数重新排序，要对占位符进行编号。参数所用占位符的显示形式如下：*%nn!*。它由百分比符号 (%) 和后接的 1 到 20 之间的整数以及后接的感叹号 (!) 组成。整数表示原始语言中占位符在字符串中的位置。“%1!”是原始版本中的第一个参数，“%2!”是第二个参数，依此类推。用这种方法指示参数的位置可以使翻译正确，即使参数出现在目标语言中的顺序与其在源语言中的顺序不同。

例如，假定以下是一条英文消息：

```
%1! is not allowed in %2!.
```

此消息的德语版本是：

```
%1! ist in %2! nicht zulässig.
```

此消息的日文版本是：

图 15-1：日文消息

%2! の中で %1! は許されません。

图 15-1 显示了一个日文短语，在该短语中的不同位置包含字符“%1!”。在此示例中，“%1!”在所有三种语言中都表示同一个参数，“%2!”在所有三种语言中也表示单个参数。此示例显示了翻译格式中有时需要对参数重新排序。

虽然占位符不必按数值顺序使用，但在格式字符串中使用占位符时不能跳过占位符编号。例如，不允许将占位符 1 和 3 放在一个格式字符串中，却不把占位符 2 也放在此字符串中。

可选的 *arg_list* 可以是一系列变量或常量。参数可以是除 text 或 image 以外的任何数据类型；在出现在最终的消息之前，它将被转换为 char 数据类型。如果没有提供任何参数列表，则格式字符串必须是要输出的没有任何占位符的消息。

替换后，*format_string* 加上所有参数的最大输出字符串长度是 512 个字节。

raiserror

raiserror 在用户屏幕上显示用户定义的错误或局部变量消息，并设置系统标志来记录已出现过错误这一情况。使用 print 时必须在使用其的批处理或过程中声明局部变量。消息可以长达 255 个字节。

以下是 raiserror 的语法：

```
raiserror error_number
    [{format_string | @local_variable}] [, arg_list]
    [extended_value  = extended_value  [{,
       extended_value  = extended_value }...]]
```

error_number 放置在全局变量 *error@@* 中，它存储 Adaptive Server 最近生成的错误号。用户定义的错误消息的错误号必须大于 17,000。如果 *error_number* 介于 17,000 和 19,999 之间，并且 *format_string* 缺失或为空 (“ ”)，则 Adaptive Server 从 master 数据库的 sysmessages 表中检索错误消息文本。这些错误消息主要供系统过程使用。

format_string 本身的长度限制在 255 个字节以内； *format_string* 加上所有参数的最大输出长度限制在 512 个字节以内。 raiserror 消息所用的局部变量必须是 char 或 varchar。*format_string* 或变量是可选的；如果不包括它们，则 Adaptive Server 使用缺省语言的 sysusermessages 中相应 *error_number* 的消息。使用 print 可以替代由 *format_string* 中 *arg_list* 所定义的变量或常量。

作为一种选择，您可以定义扩展错误数据以供 Open Client 应用程序使用 （在 raiserror 中包括 *extended_values* 时）。有关扩展错误数据的详细信息，请参见 Open Client 文档或 《参考手册：命令》。

如果希望将错误号存储在 *error@@* 中，请使用 raiserror 而不是 print。以下是如何在过程 findrules 中使用 raiserror 的示例：

```
raiserror 99999 "You must give a user name"
```

所有用户定义错误消息的严重级是 16，表示用户已经犯了非致命错误。

为 *print* 和 *raiserror* 创建消息

可用 sp_getmessage 从 sysusermessages 中调用 print 或 raiserror 所用的消息。使用 sp_addmessage 创建一组消息。

以下示例使用 sp_addmessage、 sp_getmessage 和 print 在 sysusermessages 中创建一条英文和德文消息、检索该消息并将其用于用户定义的存储过程，然后输出该消息。

```
/*
** Install messages
```

```
** First, the English (langid = NULL)
*/
set language us_english
go
sp_addmessage 25001,
  "There is already a remote user named '%1!' for remote
server '%2!'."
go
/* Then German*/
sp_addmessage 25001,
          "Remotebenutzername '%1!' existiert bereits
auf dem Remoteserver '%2!'.","german"
go
create procedure test_proc @remotename varchar(30),
            @remoteserver varchar(30)
as
        declare @msg varchar(255)
        declare @arg1 varchar(40)
        /*
        ** check to make sure that there is not
        ** a @remotename for the @remoteserver.
        */
        if exists (select *
            from master.dbo.sysremotelogins l,
                master.dbo.sysservers s
            where l.remoteserverid = s.srvid
                and s.srvname = @remoteserver
                and l.remoteusername = @remotename)
        begin
            exec sp_getmessage 25001, @msg output
            select @arg1=isnull(@remotename, "null")
            print @msg, @arg1, @remoteserver
            return (1)
        end
return(0)
go
```

也可以将用户定义的消息绑定到约束，如第 243 页的 “为约束创建错误消息”中所述。

要删除用户定义消息，使用 sp_dropmessage。要更改消息，首先使用 sp_dropmessage 将其删除，然后使用 sp_addmessage 再次添加它。

waitfor

waitfor 关键字指定执行语句块、存储过程或事务的特定时间、时间间隔或事件。

语法为：

```
waitfor {delay "time" | time "time" | errorexit |  processexit | mirrorexit}
```

其中， delay *time* 指示 Adaptive Server 要等到指定的时间段已经过去。time *time* 指示 Adaptive Server 要等到指定的时间，它使用 datetime 的有效数据格式之一。

但是，不能指定日期 – 不允许指定 datetime 值的日期部分。用 waitfor time 或 waitfor delay 指定的时间可包括小时、分钟和秒 – 最多为 24 小时。使用格式 “hh:mm:ss”。

例如，以下命令指示 Adaptive Server 等到下午 4:23：

```
waitfor time "16:23"
```

以下命令指示 Adaptive Server 等待 1 小时 30 分钟：

```
waitfor delay "01:30:00"
```

有关 time 值的格式的论述，请参见第 300 页的 “输入时间”。

errorexit 指示 Adaptive Server 一直等到进程异常终止。 processexit 一直等到进程因任何原因终止。 mirrorexit 一直等到读取或写入镜像设备操作失败。

可将 waitfor errorexit 用于注销异常终止进程的过程，从而释放系统资源，避免被受影响的进程占用。要找到受影响的进程，使用 sp_who 来检查 sysprocesses 表。

以下示例指示 Adaptive Server 一直等到下午 2:20。然后，它用走下一步棋来更新 chess 表，并执行名为 sendmessage 的存储过程，这会在 Judy 的一个表中插入一条消息，通知她 chess 表中现在包含新的走棋。

```
begin
waitfor time "14:20"
insert chess(next_move)
values("Q-KR5")
execute sendmessage "Judy"
end
```

若要在 10 秒钟之后而不是等到 2:20 向 Judy 发送消息，请替换上一示例中的 waitfor 语句：

```
waitfor delay "0:00:10"
```

执行 waitfor 命令后，在到达指定时间或发生指定事件之前，不能使用到 Adaptive Server 的连接。

注释

使用注释符号使注释附加到语句、批处理和存储过程。不会执行注释。

可以单独一行或在命令行结束处插入注释。可以使用两种注释样式：“斜线 - 星号”样式：

```
/* text of comment */
```

和“双连字符”样式：

```
-- text of comment
```

对于多行注释，不能使用双连字符样式。

注释不限制最大长度。

斜线 - 星号样式注释

/* 样式注释是 Transact-SQL 扩展。只要每条注释以“/*”开始并以“*/”结束，多行注释就是可接受的。“/*”和“*/”之间的所有内容都作为注释的一部分来处理。/* 形式允许嵌套。

多行注释惯用的样式是第一行以“/*”开始，后续各行以“**”开始。注释通常以“*/”结束：

```
select * from titles
/* A comment here might explain the rules
** associated with using an asterisk as
** shorthand in the select list.*/
where price > $5
```

以下过程包括几个注释：

```
/* this procedure finds rules by user name*/
create procedure findmyrule @nm varchar(30) = null
as
if @nm is null
begin
  print "You must give a user name"
  return
  print "I have returned"
/* this statement follows return,
** so won't be executed */
end
else      /* print the rule names and IDs, and
          the user ID */
  select sysobjects.name, sysobjects.id,
     sysobjects.uid
  from sysobjects, master..syslogins
```

```
  where  master..syslogins.name = @nm
  and sysobjects.uid = master..syslogins.suid
  and sysobjects.type = "R"
```

双连字符样式注释

此注释样式以两个连续的连字符和随后的一个空格 (--) 开始并以换行符终止。因此，多行注释是不可能的。

Adaptive Server 不会将字符串文字中或 /* 样式注释中的两个连续连字符解释为注释开始的标记。

若要表示包含两个连续减号的表达式 （binary 后跟 unary），请在两个连字符之间放置一个空格或左小括号。

示例如下：

```
-- this procedure finds rules by user name
create procedure findmyrule @nm varchar(30) = null
as
if @nm is null
begin
  print "You must give a user name"
  return
  print "I have returned"
-- each line of a multiple-line comment
-- must be marked separately.
end
else        -- print the rule names and IDs, and
        -- the user ID
  select sysobjects.name, sysobjects.id,
     sysobjects.uid
  from sysobjects, master..syslogins
  where  master..syslogins.name = @nm
  and sysobjects.uid = master..syslogins.suid
  and sysobjects.type = "R"
```

局部变量

局部变量经常用作批处理或存储过程中 while 循环或 if...else 块的计数器。在存储过程中使用局部变量时，它们被声明为，在其执行过程中由该过程自动并且非交互地使用。几乎可以在任何 Transact-SQL 语法指示可以使用表达式的位置使用变量，例如 *char_expr*、 *integer_expression*、*numeric_expr* 或 *float_expr*。

声明局部变量

若要 declare （声明）局部变量的名称和数据类型，请使用：

declare @*variable_name datatype*
  [, @*variable_name datatype*]...

变量名前面必须带有 @ 符号，且必须符合标识符的规则。指定 text、image 或 sysname 外的用户定义数据类型或系统提供的数据类型。

从记忆和性能的角度看，这样编写更有效：

```
declare @a int, @b char(20), @c float
```

而不是编写为：

```
declare @a int
declare @b char(20)
declare @c float
```

局部变量和 *select* 语句

声明变量时，它的值为 NULL。使用 select 语句为局部变量赋值。语法如下：

select @*variable_name* = { *expression* |
  (*select_statement*) } [, @*variable* =
  { *expression* | (*select_statement*) } ...]
  [from *clause*] [where *clause*] [group by *clause*]
  [having *clause*] [order by *clause*] [compute *clause*]

使用 declare 语句时，这样编写更有效：

```
select @a = 1, @b = 2, @c = 3
```

而不是编写为：

```
select @a = 1
select @b = 2
select @c = 3
```

不要使用单个 select 语句为一个变量赋值，然后为另一个基于此变量值的变量赋值。这样做可能产生不可预料的结果。例如，如下两个查询都要查找 *@c2* 的值。第一个查询产生值 NULL，而第二个查询产生正确答案为 0.033333：

```
/* this is wrong*/
declare @c1 float, @c2 float
select @c1 = 1000/1000, @c2 = @c1/30
select @c1 , @c2

/* do it this way */
declare @c1 float, @c2 float
select @c1 = 1000/1000
select @c2 = @c1/30
select @c1 , @c2
```

不能使用 select 语句给变量赋值，同时为用户返回数据。下例中的第一个 select 语句给局部变量 *@veryhigh* 赋以最高价格；需要第二个 select 语句显示值：

```
declare @veryhigh money
select @veryhigh = max(price)
   from titles
select @veryhigh
```

如果为变量赋值的 select 语句返回多个值，则最后返回的值将赋给此变量。以下查询给变量赋以 “select advance from titles” 最后返回的值。

```
declare @m money
select @m = advance from titles
select @m

(18 rows affected)
------------------------
                8,000.00

(1 row affected)
```

赋值语句指示 select 语句影响 （返回）了多少行。

如果给变量赋值的 select 语句没有返回任何值，则该语句没有更改变量的值。

局部变量可以用作 print 或 raiserror 的参数。

如果在 update 语句中使用变量，请参见第 319 页的 “将 set 子句用于 update”。

局部变量和 *update* 语句

可以直接在 update 语句中给变量赋值。无需用 select 语句为变量赋值。声明变量时，它的值为 NULL。语法如下：

```
update [[database.]owner.] {table_name | view_name}
    set [[[database.]owner.]{table_name. | view_name.}]
            column_name1 =
          {expression1 | null | (select_statement)} |
            variable_name1 = {expression1 | null}
        [, column_name2 = {expression2 | null |
            (select_statement)}]... |
        [, variable_name2 = {expression2 | null}]...
    [from [[database.]owner.]{table_name | view_name}
            [, [[database.]owner.] {table_name |
                    view_name}]]...
    [where search_conditions]
```

局部变量和子查询

为局部变量赋值的子查询*必须*仅返回一个值。下面是一些示例：

```
declare @veryhigh money
select @veryhigh = max(price)
   from titles
if @veryhigh > $20
   print "Ouch!"
declare @one varchar(18), @two varchar(18)
select @one = "this is one", @two = "this is two"
if @one = "this is one"
   print "you got one"
if @two = "this is two"
   print "you got two"
else print "nope"
declare @tcount int, @pcount int
select @tcount = (select count(*) from titles),
   @pcount = (select count(*) from publishers)
select @tcount, @pcount
```

局部变量和 *while* 循环以及 *if...else* 块

以下示例在 while 循环的计数器中使用局部变量，以便与 if 语句中的 where 子句匹配以及设置和重新设置 select 语句中的值：

```
/* Determine if a given au_id has a row in au_pix*/
/* Turn off result counting */
set nocount on
/* declare the variables */
declare @c int,
        @min_id varchar(30)
/*First, count the rows*/
select @c = count(*) from authors
/* Initialize @min_id to "" */
select @min_id = ""
/* while loop executes once for each authors row */
while @c > 0
begin
        /*Find the smallest au_id*/
        select @min_id = min(au_id)
            from authors
            where au_id > @min_id
        /*Is there a match in au_pix?*/
        if exists (select au_id
            from au_pix
            where au_id = @min_id)
            begin
            print "A Match!%1!", @min_id
      end
      select @c = @c -1  /*decrement the counter */
end
```

变量和空值

声明局部变量时给其赋值 NULL，也可用 select 语句为其赋以空值。NULL 的特殊含义要求空值变量和其它空值之间的比较遵循特殊的规则。

表 15-2 显示了在空值列和空值表达式之间使用不同的比较运算符的比较结果。表达式可以是变量、文字或变量、文字以及算术运算符的组合。

表 15-2：比较空值

比较的类型	使用 = 运算符	使用 <、 >、 <=、 !=、 !<、 !> 或 <> 运算符
column_value 与 *column_value* 相比	FALSE	FALSE
column_value 与 *expression* 相比	TRUE	FALSE
expression 与 *column_value* 相比	TRUE	FALSE
expression 与 *expression* 相比	TRUE	FALSE

例如，如下测试：

```
declare @v int, @i int
if @v = @i select "null = null, true"
if @v > @i select "null > null, true"
```

显示了只有第一个比较返回 TRUE：

```
-----------------
null = null, true

(1 row affected)
```

下例返回 titles 表中所有 advance 为值 NULL 的行：

```
declare @m money
select title_id, advance
from titles
where advance = @m
title_id  advance
-------- ----------------
MC3026              NULL
PC9999              NULL

(2 rows affected)
```

全局变量

全局变量是系统支持的预定义变量。与局部变量不同的是，它们在名称前有两个 @ 符号；例如 *@@error*。两个 @ 符号是定义全局变量所用标识符的一部分。

用户不能在 select 语句中，直接创建全局变量，也不能直接更新全局变量的值。如果用户声明一个与某**全局变量**同名的局部变量，则此变量作为局部变量处理。

事务和全局变量

一些全局变量提供用于事务的信息。

用 @@*error* 检查错误

@@*error* 全局变量通常用于检查当前用户会话中最近执行的批处理的错误状态。如果最后的事务成功，则 @@*error* 包含 0；否则 @@*error* 包含系统生成的最后一个错误号。类似 if @@error != 0 后接 return 的语句，会导致在出错时退出。

每个 Transact-SQL 语句，包括 print 语句和 if 测试，重新设置 @@*error* 以便状态检查必须立即跟踪存在问题的批处理。

@@*sqlstatus* 全局变量对 @@*error* 输出没有影响。有关详细信息，请参见第 482 页的 “检查最后的 fetch 的状态”。

用 @@*identity* 检查 IDENTITY 值

@@*identity* 包含当前用户会话中最后插入 IDENTITY 列的值。每次 insert、 select into 或 bcp 要向表中插入一行时，设置 @@*identity* 的值。insert、 select into 或 bcp 语句失败或包含它的事务回退不会影响 @@*identity* 的值。@@*identity* 保留最后插入 IDENTITY 列的值，即使向 IDENTITY 列插入该值的语句提交失败。

如果语句插入多行，则 @@*identity* 反映插入的最后一行的 IDENTITY 值。如果受影响的表不包含 IDENTITY 列，则 @@*identity* 被设置为 0。

用 @@*trancount* 检查事务嵌套级

@@*trancount* 包含当前用户会话中事务的嵌套级。批处理中的每个 begin transaction 将增加事务计数。在链式事务模式中查询 @@*trancount* 时，因为查询自动开始一个事务，所以其值不会为 0。

用 @@*transtate* 检查事务状态

@@*transtate* 包含当前用户会话执行某个语句之后事务的当前状态。但是，与 @@*error* 不同， @@*transtate* 不会被每个批处理清除。@@*transtate* 可以包含表 15-3 中的值：

表 15-3：@@transtate 值

值	含义
0	事务正在进行：正在处理显式或隐式事务；前一语句已成功执行。
1	事务已经成功：事务已完成且已提交其更改。
2	语句被中止：前一语句已中止；对事务无影响。
3	事务被中止：事务已中止且已回退任何更改。

@@transtate 只因执行错误而更改。语法和编译错误不会影响 *@@transtate* 的值。有关显示事务中的 *@@transtate* 的示例，请参见第 691 页的 “检查事务的状态”。

用 @@nestlevel 检查嵌套级

@@nestlevel 包含用户会话中当前执行的嵌套级别，初始值为 0。每当存储过程或触发器调用另一个存储过程或触发器，嵌套级别就会增加一级。创建高速缓存的语句时，嵌套级别也会增加一级。如果超过了最大值 16，事务中止。

检查最后的 *fetch* 的状态

@@sqlstatus 包含当前用户会话中最后的 fetch 语句导致的状态信息。*@@sqlstatus* 可能包含以下值：

表 15-4：@@sqlstatus 值

值	含义
0	fetch 语句成功完成。
1	fetch 语句出错。
2	结果集中不再有数据。当前游标位置在结果集中的最后一行并且客户端对该游标提交了 fetch 语句时，就会出现这一警告。

@@sqlstatus 对 *@@error* 输出没有影响。例如，以下批处理通过使 fetch *@@error* 语句出错，将 *@@sqlstatus* 设置为 1。但是， *@@error* 反映错误消息的编号，而不是 *@@sqlstatus* 输出：

```
declare csr1 cursor
for select * from sysmessages
for read only

open csr1

begin
```

```
    declare @xyz varchar(255)
    fetch csr1 into @xyz
    select error = @@error
    select sqlstatus = @@sqlstatus
end

Msg 553, Level 16, State 1:
Line 3:
The number of parameters/variables in the FETCH INTO
clause does not match the number of columns in cursor
'csr1' result set.
```

此时，将 *@@error* 全局变量设置为 553，即最后生成的错误的编号。*@@sqlstatus* 设置为 1。

@@fetch_status 返回最近的 fetch 的状态：

表 15-5：@@fetch_status

值	含义
0	fetch 操作成功
-1	fetch 操作失败
-2	留作将来使用

受 *set* 选项影响的全局变量

set 选项可以自定义结果的显示、显示处理的统计信息并提供其它调试 Transact-SQL 程序所用的诊断帮助。

表 15-6 列出了包含与 set 命令控制的会话选项有关的信息的全局变量。

表 15-6：包含会话选项的全局变量

全局变量	说明
@@char_convert	如果字符集转换无效，则包含 0。如果字符集转换有效，则包含 1。
@@client_csexpansion	返回从服务器字符集转换为客户端字符集时使用的扩展因子。例如，如果它包含一个值 2，则服务器字符集中的字符在转换为客户端字符集后最多可以占用原来字节数的两倍。
@@cursor_rows	为可滚动游标专门设计的全局变量。该变量显示游标结果集中的总行数。它返回值 –1：
@@datefirst	使用 set datefirst n 进行设置，其中， n 是介于 1 和 7 之间的值。返回 *@@datefirst* 的当前值，它指示为每个星期指定的第一天，表示为 tinyint。 在 Adaptive Server 中，缺省值为星期日（根据 us_language 缺省值），您可以通过指定 `set datefirst 7` 来进行设置。有关这些设置和值的详细信息，请参见 set 命令的 datefirst 选项。

全局变量	说明
@@isolation	包含 Transact-SQL 程序的当前隔离级别。 *@@isolation* 采用活动级别的值（0、 1 或 3）。
@@options	包含以十六进制形式表示的会话的 set 选项。
@parallel_degree	包含当前的最大并行度设置。
@@rowcount	包含最后的查询所影响的行数。任何不返回行的命令 （例如 if、 update 或 delete 语句）将 *@@rowcount* 设置为 0。对于游标， *@@rowcount* 表示从游标结果集返回到客户端 （直到最后一个 fetch 命令）的累积行数。
@scan_parallel_degree	包含非聚簇索引扫描的当前最大并行度设置。
@@lock_timeout	使用 set lock wait n 进行设置。它返回当前 *lock_timeout* 设置 （以毫秒为单位）。 *@@lock_timeout* 返回值 n。缺省值为无超时。如果在会话开始时没有执行 set lock wait n， *@@lock_timeout* 将返回 -1。
@@textsize	包含 select 返回的 text、 unitext 或 image 数据的字节数限制。 isql 的缺省限制为 32K 字节；缺省值取决于客户端软件。它可以用 set textsize 为某个会话进行更改。
@@tranchained	包含 Transact-SQL 程序的当前事务模式。 *@@tranchained* 为非链接模式则返回 0，为链式模式则返回 1。

@@options 全局变量包含会话的 set 选项的十六进制表示形式。表 15-7 列出了 *@@options* 使用的 set 选项和值。

表 15-7：@@options 的 set 选项和值

数字值	十六进制值	*set* 选项
4	0x04	showplan
5	0x05	noexec
6	0x06	arithignore
8	0x08	arithabort
13	0x0D	control
14	0x0E	offsets
15	0x0F	statistics io 和 statistics time
16	0x10	parseonly
18	0x12	procid
20	0x14	rowcount
23	0x17	nocount
77	0x4D	opt_sho_fi
78	0x4E	select
79	0x4F	set tracefile

有关会话选项的详细信息，请参见 《参考手册：命令》中的 set。

全局变量中的语言和字符集信息

表 15-8 列出了包含有关语言和字符集的信息的全局变量。请参见《系统管理指南：卷 1》中的第 9 章“配置字符集、排序顺序和语言”。

表 15-8：用于语言和字符集的全局变量

全局变量	说明
@@char_convert	如果字符集转换无效，则包含 0。如果字符集转换有效，则包含 1。
@@client_csid	包含客户端的字符集 ID。如果客户端字符集从未初始化，则设置为 -1；否则，它包含 syscharsets 中最近使用的客户端字符集的 *id*。
@@client_csexpansion	返回从服务器字符集转换为客户端字符集时使用的扩展因子。例如，如果它包含值 2，则服务器字符集中的字符在转换为客户端字符集后，最多可能会占用原来字节数的两倍。
@@client_csname	包含客户端的字符集名。如果客户端字符集从未初始化，则设置为 NULL；否则，将包含最近使用的字符集的名称。
@@langid	定义当前所用语言的本地语言 ID，该值在 syslanguages.langid 中指定。
@@language	定义当前使用的语言的名称，如 syslanguages.name 中所指定。
@@maxcharlen	包含缺省字符集中多字节字符的最大长度 （以字节计）。
@@ncharsize	包含国家 / 地区字符的平均长度 （以字节计）。

监控系统活动的全局变量

许多全局变量报告自 Adaptive Server 最近一次启动以来，系统发生的活动。 sp_monitor 显示一些全局变量的当前值。

表 15-9 以 sp_monitor 返回的顺序列出了监控系统活动的全局变量。请参见《参考手册：过程》。

表 15-9：监控系统活动的全局变量

全局变量	说明
@@authmech	只读全局变量，设置为用于鉴定用户的机制。
@@connections	包含登录或试图登录的次数。
@@cpu_busy	包含自 Adaptive Server 上次启动起，CPU 执行 Adaptive Server 工作所花费的时间 （以时钟周期计）。
@@idle	包含 Adaptive Server 自上次启动起空闲的时间 （以时钟周期计）。
@@io_busy	包含 Adaptive Server 执行输入和输出操作所花费的时间 （以时钟周期计）。
@@packet_errors	包含 Adaptive Server 发送和接收包时所出现的错误数。
@@monitors_active	减少 sp_sysmon 显示的消息数。
@@pack_received	包含 Adaptive Server 自上次启动起所读取的输入包的数目。
@@pack_sent	包含 Adaptive Server 自上次启动起所写入的输出包的数目。
@@total_errors	包含 Adaptive Server 在进行读取操作时出现的错误数。

全局变量	说明
@@total_read	包含 Adaptive Server 自上次启动起所读取的磁盘的数目。
@@total_write	包含 Adaptive Server 自上次启动起所写入的磁盘的数目。

存储在全局变量中的优化程序和分区信息

以下全局变量报告 Adaptive Server 版本 15 和更高版本的优化和分区信息。

表 15-10：包含优化程序和分区信息的全局变量

全局变量	说明
@@optgoal	返回查询优化的当前优化目标设置。
@@opttimeoutlimit	返回查询优化的当前优化超时限制设置。
@@repartition_degree	返回当前的动态重新分区度设置。
@@resource_granularity	返回查询优化的最大资源使用情况提示设置。

存储在全局变量中的服务器信息

表 15-11 列出了包含 Adaptive Server 的杂类信息的全局变量。

表 15-11：包含 Adaptive Server 信息的全局变量

全局变量	说明
@@bulkarraysize	返回使用批量复制接口进行传输之前缓冲到本地服务器内存中的行数。它仅用于组件集成服务，以使用 select into 将行传输到远程服务器。请参见 《组件集成服务用户指南》。
@@bulkbatchsize	返回使用批量接口通过 select into *proxy_table* 传输到远程服务器的行数。仅用于组件集成服务，以便通过 select into 将行传输到远程服务器。有关详细信息，请参见 《组件集成服务用户指南》。
@@boottime	显示服务器的启动时间。
@@cis_version	包含组件集成服务的日期和版本 （如果已经安装）。
@@cmpstate	确定高可用性环境中主协同服务器的当前模式。有关返回的值，请参见 《在高可用性系统中使用 Sybase 故障切换》中的表 4-1。
@@guestuid	guest 服务器用户 ID。其值为 -1。
@@guestuserid	guest 用户 ID。其值为 2。
@@invaliduserid	无效用户 ID。其值为 -1
@@invalidusid	无效服务器用户 ID。其值为 -2。

全局变量	说明
@@max_connections	包含在当前计算机环境下可同时与 Adaptive Server 连接的最大数目。使用 number of user connections 配置参数，可以将 Adaptive Server 的连接数量配置为少于或等于 *@@max_connection* 的任意值。
@@maxpagesize	显示服务器页大小。其值为 2048、 4096、 8128 或 16384。
@@maxuserid	最大用户 ID。其值为 2147483647。
@@max_precision	返回由服务器设置的 decimal 和 numeric 数据类型所使用的精度级别。该值固定为常数 38。
@@maxuid	最大服务器用户 ID。其值为 2147483647。
@@mingroupid	最小组用户 ID。最小值为 16384。
@@minsuid	最小服务器用户 ID。最小值为 -32768。
@@minuserid	最小用户 ID。最小值为 -32768。
@@maxgroupid	最大组用户 ID。最大值为 1048576
@@maxuserid	最大用户 ID。最大值为 2147483647
@@procid	包含当前执行过程的存储过程 ID。
@@probesuid	probe 用户 ID。其值为 2。
@@remotestate	返回高可用性环境中主协同服务器的当前模式。有关返回的值，请参见 《在高可用性环境中使用 Sybase 故障切换》中的表 4-1。
@@servername	包含此 Adaptive Server 的名称。使用 sp_addserver 定义服务器名，并重新启动 Adaptive Server。
@@spid	包含当前进程的服务器进程 ID 号。
@@ssl_ciphersuite	如果当前连接未使用 SSL 协议，则返回 NULL ； 否则，它返回当前连接上的 SSL 握手选择的密码成套程序名称。
@@tempdbid	返回为会话指定的临时数据库的有效临时数据库 ID (dbid)。
@@thresh_hysteresis	包含激活阈值所要求自由空间的减小量。此数量也称作停滞值 （以 2K 数据库页为单位）。它确定可在数据库段上放置阈值的接近程度。
@@timeticks	包含每个时钟周期中的微秒数。每个时钟周期的时间量与计算机有关。
@@version	包含 Adaptive Server 当前发布版的日期。
@@version_number	以整数形式返回 Adaptive Server 当前发布版的完整版本 （例如， 12503）。
@@boottime	显示 Adaptive Server 的启动时间。

全局变量以及 *text*、 *unitext* 和 *image* 数据

text、 unitext 和 image 的值可能非常大。如果选择列表中包括 text 和 image 值，返回数据的长度限制取决于 *@@textsize* 全局变量的设置，它包含 select 返回的 text 或 image 数据的字节数限制。 isql 的缺省限制为 32K 字节；缺省值取决于客户端软件。使用 set textsize 更改会话所用的值。

表 15-12 列出了包含文本指针信息的全局变量。这些全局变量是基于会话的变量。有关文本指针的详细信息，请参见第 322 页的 “更改 text、unitext 和 image 数据”。

表 15-12：存储在全局变量中的文本指针信息

全局变量	说明
@@textcolid	包含 *@@textptr* 所引用列的列 ID。
@@textdbid	包含数据库的数据库 ID，该数据库含有其列被 *@@textptr* 引用的对象。
@@textobjid	包含对象的对象 ID，该对象含有被 *@@textptr* 引用的列。
@@textdataptnid	返回在其中插入或更新数据的文本分区的分区 ID。
@@textptnid	返回包含 @@textptr 引用的列的数据分区的分区 ID。
@@textptr	包含进程最后插入或更新的 text 或 image 列的文本指针。（不要将此变量与 textptr 函数混淆。）
@@textts	包含 *@@textptr* 所引用列的文本时间戳。

第 16 章

在查询中使用内置函数

内置函数是对从数据库返回信息的 SQL 的 Transact-SQL 扩展。请参见 《参考手册：构件块》。

返回数据库信息的系统函数

系统函数返回数据库中的特殊信息。许多系统函数提供查询系统表的快捷方式。

系统函数的一般语法是：

select *function_name*(*argument*[*s*])

可以在 select 列表、 where 子句以及允许使用表达式的任何地方使用系统函数。

例如，若要找到以 “harold” 登录的同事的用户标识号，请键入：

```
select user_id("harold")
```

假定 “harold” 的用户 ID 为 13，则结果为：

```
-------------
           13

(1 row affected)
```

通常，函数名可指示返回信息的类型。

系统函数 user_name 采用一个 ID 号作为其参数并返回用户名：

```
select user_name(13)
---------
harold

(1 row affected)
```

若要查找当前用户名 （即您的名称），请省略该参数：

```
select user_name()

---------
dbo

(1 row affected)
```

Adaptive Server 按如下方式处理用户 ID：

- 系统管理员在自己使用的任何数据库中，通过假定**服务器用户 ID** 为 1 可以成为数据库所有者。
- 服务器用户 ID 1 总是赋予 “guest”用户。
- 在数据库中，数据库所有者的 user_name 始终是 “dbo”；其用户 ID 为 1。
- 在数据库中，“guest”用户 ID 始终为 2。

《系统管理指南：卷 2》的第 10 章 “检查数据库一致性”和 《参考手册：命令》中介绍了 dbcc 命令。

表 16-1 列出了每个系统函数的名称、采用的参数及其返回的结果。请参见 《参考手册：构件块》。

表 16-1：系统函数、参数及结果

函数	参数	结果
audit_event_name	((*event_id*)	返回审计事件的编号。
col_length	(*object_name*, *column_name*)	返回已定义的列长度。使用 datalength 来查看实际的数据大小。
col_name	(*object_id*, *column_id* [, *database_id*])	返回指定的列 ID 和表 ID 的列名。
curunreservedpgs	(*dbid*, *lstart*, *unreservedpgs*)	返回磁盘区段中的可用页数。如果数据库已打开，将从内存中取值；如果未使用数据库，则从 sysusages 的 unreservedpgs 列中取值。
data_change	(*expression*)	返回指定列或字符串的实际长度 （以字节表示）

函数	参数	结果
datalength	(*expression*)	以字节为单位返回表达式的长度。*expression* 通常是一个列名。如果 *expression* 是字符常量，则必须用引号引起来。
data_pages	([*dbid*,] *obj_id* , {*indid* | *partitionid* })	返回表 (*doampg*) 或者索引 (*ioampg*) 所使用的页数。其结果不包括用于内部结构的页。在对 sysindexes 表运行查询时可以使用这一函数。
db_id	([*database_name*])	返回数据库 ID 号。 *database_name* 必须是字符表达式；如果它是常量表达式，则必须用引号引起来。如果未提供 *database_name*， db_id 将返回当前数据库的 ID 号。
db_name	([*database_id*])	返回数据库的名称。 *database_id* 必须是一个数值表达式。如果未提供 *database_id*， db_name 将返回当前数据库的名称。
host_id	( )	返回客户端进程 （不是 Adaptive Server 进程）的主机进程 ID。
host_name	( )	返回当前客户端进程 （不是 Adaptive Server 进程）的主计算机名。
index_col	(*object_name*, *index_id* , *key_* #[, *user_id*])	返回索引列的名称；如果 *object_name* 不是表名或视图名，则返回 NULL。索引列名称最多可以包含 255 个字节。
isnull	(*expression1*, *expression2*)	当求得的 *expression1* 的值为 NULL 时，用 *expression2* 中指定的值替代。这些表达式的数据类型必须隐式转换，或使用 convert 函数进行转换。
lct_admin	({{ "lastchance" | "logfull" | "unsuspend"} , *database_id* } | "reserve", *log_pages* })	管理日志段的最后机会阈值。 lastchance 在指定数据库中创建最后机会阈值。 logfull 在指定的数据库已超过最后机会阈值时返回 1，否则返回 0。 unsuspend 将唤醒数据库中挂起的任务，并在超过最后机会阈值时禁用该阈值。 reserve 将返回成功转储指定大小的事务日志所需的可用日志页数。
identity_burn_max	(*table_name*)	跟踪给定表的 identity burn max 值。此函数只返回值；不执行更新。

函数	参数	结果
is_quiesced	((*dbid*)	指示数据库是否处于 quiesce database 模式。如果数据库已停顿，则 is_quiesced 返回 1，否则返回 0。
lockscheme	(*object_id* [*, db_id*])	以字符串形式返回指定对象的锁定方案。
mut_excl_roles	(*role1, role2* [membership | activation])	返回有关两个角色之间互斥性的信息。
next_identity	(*table_name*)	检索下一个 insert 可用的下一个标识值。
pagesize	(*object_name* [, *index_name*])	以字节为单位返回指定对象的页大小。
object_id	((*object_name*)	返回对象 ID。
object_name	(*object_id* [, *database_id*])	返回指定对象 ID 的对象名。对象名最多可为 255 字节。
Proc_role	("*role_name*")	返回有关是否已授予用户指定角色的信息。
reserved_pages	(*dbid*, *objid* [, *indid* [, *partitionid*]])	返回分配给表或索引的页数，包括用于内部结构的报告页。
row-count	(*dbid*, *objid*, [, *partitionid*])	返回表的行数 （估计值）。
suser_id	([*server_user_name*])	返回服务器用户在 syslogins 中的 ID 号。如果未提供 *server_user_name*，它将返回当前用户的服务器 ID。
suser_name	([*server_user_id*])	返回服务器用户的名称。服务器用户的 ID 存储在 syslogins 中。如果未提供 *server_user_id*，它将返回当前用户的名称。
tsequal	(timestamp, timestamp2)	比较 timestamp 值，以防止更新被选择用于浏览而被修改的行。 timestamp 是浏览行的时间戳； timestamp2 是存储行的时间戳。允许不调用 DB-Library 即可使用浏览模式。
tran_dumpable_status	("*database_name*")	返回 true/false，指示是否允许执行 dump transaction。
used_pages	(*dbid*, *objid*, [, *indid* [, *partitionid*]])	返回表及其聚簇索引所用的总页数。
user		返回用户的名称。
user_id	([([*user_name*])	返回用户的 ID 号。报告当前数据库的 sysusers 中的编号。如果未提供 *user_name*，它将返回当前用户的 ID。
user_name	([*user_id*])	基于当前数据库中的用户 ID，返回用户名。如果未提供 *user_id*，它将返回当前用户的名称。

函数	参数	结果
valid_name	(*character_expression* [, *maximum_length*]	如果 *character_expression* 不是有效标识符，则返回 0，如果是有效标识符，则返回非 0 数字。 第二个可选参数变量是一个整数，其值范围为 > 零 (0) 且 <= 255。缺省值为 30。 如果标识符长度大于 *maximum_length* 参数， valid_name 将返回 0，该标识符无效。 有关有效标识符的定义，请参见第 1 章 “SQL 构件块”。
valid_user	((*server_user_id*)	如果指定 ID 至少在该 Adaptive Server 上的一个数据库中是有效用户或别名，则返回 1。您必须具有 sa_role 或 sso_role 角色才能以自己之外的 *server_user_id* 使用该函数。

当系统函数的参数为可选时，会采用当前的数据库、主计算机、服务器用户或数据库用户。除 user 之外，即使参数为 NULL，内置函数也始终使用小括号。

使用系统函数的示例

本节中的示例使用以下系统函数：

- col_length
- datalength
- isnull
- user_name

col_length

该查询查找 titles 表中 title 列的长度 （查询包括 “x=”，因此结果有一个列标题）：

```
select x = col_length("titles", "title")

 x
--------
      80

(1 row affected)
```

datalength

col_length 查找定义的列长度，与之相反， datalength 报告每行中所存储数据的实际长度 （以字节表示）。因为 varchar、 nvarchar、 univarchar、 varbinary、 text、 unitext 和 image 数据类型可以存储可变长度 （但不存储尾随空白），所以对于这些数据类型可以使用该函数。任何 NULL 数据的 datalength 都将返回 NULL。当声明 char 值可为 NULL 时， Adaptive Server 在内部将其存储为 varchar。所有其它数据类型将报告其已定义长度。下面是查找 publishers 表中 pub_name 列的长度的示例：

```
select Length = datalength(pub_name), pub_name
from publishers

Length  pub_name
------  ------------------------
13      New Age Books
16      Binnet & Hardley
20      Algodata Infosystems

(3 rows affected)
```

isnull

该查询查找所有书目的平均价格，同时将 price 中所有 NULL 条目的值替换为 “$10.00”：

```
select avg(isnull(price,$10.00))
from titles

------------
       14.24

(1 row affected)
```

user_name

以下查询查找 sysusers 中的行 （其中名称等于对用户 ID 1 应用系统函数 user_name 的结果）：

```
select name
from sysusers
where name = user_name(1)
name
------------------------
dbo

(1 row affected)
```

用于字符串或表达式的字符串函数

字符串函数用于对字符串或表达式进行多种运算。少数字符串函数可用于二进制数据及字符数据。也可以并置二进制数据、字符串或表达式。

字符串内置函数返回通常在字符数据运算中所需的值。字符串函数名不是关键字。

字符串函数语法的一般形式为：

```
select function_name(arguments)
```

可以并置二进制表达式或字符表达式，如下所示：

```
select (expression + expression [+ expression]...)
```

在并置非字符表达式和非二进制表达式时，必须使用 convert 函数：

```
select "The price is " + convert(varchar(12),price)
from titles
```

大多数字符串函数只能用于 char、 nchar、 unichar、 varchar、 univarchar 和 nvarchar 数据类型以及隐式转换为 char、unichar 或 varchar、univarchar 的数据类型。少数字符串函数也可用于 binary 和 varbinary 数据。patindex 也可用于 text、 unitext、 char、 nchar、 unichar、 varchar、 nvarchar 和 univarchar 列。

可以并置 binary 和 varbinary 列以及 char、 unichar、 nchar、 varchar、 univarchar 和 nvarchar 列。如果用 char、 nchar、 nvarchar 和 varchar 并置 unichar 和 univarchar，结果将是 unichar 或 univarchar。但是，*不能* 并置 text、 unitext 或 image 列。

可以嵌套字符串函数并在允许使用表达式的任何地方使用它们。如果将常量用于字符串函数，则应使用单引号或双引号将其括起来。

每种函数还可接受隐式地转换为指定类型的参数。例如，接受近似数值表达式的函数还可接受整数表达式。 Adaptive Server 会自动将参数转换成所需类型。

表 16-2 列出了字符串函数中使用的参数。如果函数使用多个同一类型的表达式，则参数将被编号为 *char_expr1*、 *char_expr2*，等等。

表 16-2：字符串函数中使用的参数

参数类型	可替换为
char_expr	类型为 char、 unichar、 varchar、 univarchar、 nchar 或 nvarchar 的字符型列名、变量或常量表达式。解释中将指明接受 text 列名的函数。常数表达式必须用引号括起来。
uchar_expr	类型为 unichar 或 univarchar 的字符型列名、变量或常量表达式。解释中将指明接受 text 列名的函数。常数表达式必须用引号括起来。
expression	二进制或字符类型的列名、变量或常量表达式。可以是 char、 varchar、 nchar、 nvarchar、 unichar 或 univarchar 数据，就 *char_expr* 而言，再加上 binary 或 varbinary。
pattern	char、 nchar、 varchar 或 nvarchar 数据类型的字符表达式，其中可能包括 Adaptive Server 支持的任何模式匹配的通配符。
approx_numeric	任何近似数值类型 （*float*、 *real* 或 *double precision*）的列名、变量或常量表达式。
integer_expr	任意整数 （例如 bigint、 tinyint、 smallint 或 int、 unsigned bigint、 unsigned int 或 unsigned smallint）、列名、变量或常量表达式。应用它们时，要注意它们的最大大小范围。
start	一个 *integer_expr*。
length	一个 *integer_expr*。

表 16-3 列出了函数名、参数和结果。

表 16-3：字符串函数、参数及结果

函数	参数	结果
ascii	(*char_expr*)	返回表达式中第一个字符的 ASCII 代码。
char	(*integer_expr*)	将单字节的 integer 值转换为 character 值。 char 通常用作 ascii 的倒数。 *integer_expr* 必须介于 0 和 255 之间。该函数返回 char 数据类型。如果所得值是一个多字节字符的第一个字节，则该字符可能尚未定义。
charindex	(*expression1*, *expression2*)	在 *expression2* 中搜索首次出现的 *expression1* 并返回表示其起始位置的整数。如果未找到 *expression1*，将返回 0。如果 *expression1* 包含通配符， charindex 会将它们视为文字。
char_length	((*char_expr*)	返回一个表示字符表达式中字符数的整数，或返回一个 text 或 unitext 值。对于表列中的可变长度数据， char_length 将返回字符数。对于固定长度数据，将返回列的已定义长度。对于多字节字符集，表达式中的字符数通常小于字节数；使用 datalength 系统函数可确定字节数。
difference	(*char_expr1*, *char_expr2*)	返回一个整数，表示两个 soundex 值之间的差值。请参见下面的 soundex。
n	(*character_expression*, *integer_expression*)	返回字符串最左侧指定数目的字符。
len	((*string_expression*)	返回指定字符串表达式 （不包括尾随空白）的字符数 （而不是字节数）。
lower	((*char_expr*)	将大写转换为小写。返回一个字符值。

函数	参数	结果
ltrim	((*char_expr*)	删除字符表达式中的前导空白。而且只删除 SQL 特殊字符说明中等于空格字符的值。
patindex	(“%*pattern*%”, *char_expr* [using {bytes | chars | characters}])	返回一个整数，表示指定字符表达式中首次出现的 *pattern* 的起始位置；如果未找到 *pattern*，则返回 0。缺省情况下，patindex 返回的偏移以字符表示。若要返回以字节表示的偏移（多字节字符串），请指定 using bytes。*pattern* 的前后必须添加通配符 %（搜索第一个或最后一个字符的情况除外）。有关可用于 *pattern* 的通配符的说明，请参见第 43 页的 “查询结果中的字符串”。 patindex 可用于 text 和 unitext 数据。
replicate	(*char_expr*, *integer_expr*)	返回与 *char_expr* 具有相同数据类型的字符串，该字符串包含重复指定次数的同一表达式，或包含重复多次，直至其大小可占据 255 字节空间的同一表达式，取其次数少者。
reverse	((*expression*)	返回字符或二进制表达式的逆序形式；如果 *expression* 是 “abcd”，它将返回 “dcba”；如果 *expression* 是 0x12345000，则返回 0x00503412。
right	(*expression, integer_expr*)	返回字符或二进制表达式中从右端起具有指定字符数的部分。返回值的数据类型与字符表达式的数据类型相同。
rtrim	((*char_expr*)	删除尾随空白。而且只删除 SQL 特殊字符定义中等于空格字符的值。
soundex	((*char_expr*)	对于由有效单字节或双字节罗马字母的邻接序列所组成的字符串，它将返回四个字符的 soundex 代码。
space	((*integer_expr*)	返回由指定数目的单字节空格所组成的字符串。
str	(*approx_numeric* [, *length* [, decimal]])	返回浮点数的字符表示形式。 *length* 设置要返回的字符数（包括小数点、小数点左右两侧的所有位数及空白）；decimal 设置要返回的小数位数。 *length* 和 decimal 是可选参数。如果给出这两个参数，就必须使用非负数。*length* 的缺省值为 10； decimal 的缺省值为 0。str 将舍入数字的小数部分，使结果不超出指定的 *length*。
str_replace	("*string_expression1*", "*string_expression2*", "*string_expression3*")	将第一个字符串表达式 (*string_expression1*) 中出现的第二个字符串表达式 (*string_expression2*) 的所有实例替换为第三个表达式 (*string_expression3*)。
stuff	(*char_expr1*, *start*, *length*, *char_expr2*)	在 *char_expr1* 的 *start* 处删除 *length* 字符，然后将 *char_expr2* 插入到 *char_expr1* 的 *start* 处。若要在不插入其它字符的情况下删除字符， *char_expr2* 应该是 NULL，而不是表示单个空格的 “ ”。
substring	(*expression*, *start*, *length*)	返回字符或二进制字符串的一部分。 *start* 指定子字符串开始的字符位置。 *length* 指定子字符串中的字符数目。

函数	参数	结果
to_unichar	((*integer_expr*)	返回具有整数表达式值的 unichar 表达式。如果整数表达式介于 0xD800 和 0xDFFF 之间，将会引发 *sqlstate* 例外、输出错误并中止操作。如果整数表达式介于 0 和 0xFFFF 之间，将返回一个 Unicode 值。如果整数表达式介于 0x10000 和 0x10FFFF 之间，将返回一个代理对。
upper	((*char_expr*)	将小写转换为大写。返回一个字符值。
uhighsurr	(*uchar_expr*, *start*)	如果 *start* 处的 Unicode 值是代理对的上半部分 （应首先出现在对中），则返回 1。否则返回 0。该函数允许编写用于处理代理的显式代码。应特别指出，如果子字符串以 uhighsurr() 为真的 Unicode 字符开头，则提取至少具有 2 个 Unicode 值的子字符串，因为 substr() 不能只提取具有 1 个 Unicode 值的子字符串。substr() 不提取半个代理对。
ulowsurr	(*uchar_expr*, *start*)	如果 *start* 处的 Unicode 值是代理对的下半部 （应出现在对中的第二部分），则返回 1。否则返回 0。该函数允许在 substr()、 stuff() 和 right() 所执行的调整周围显式代码。应特别指出，如果子字符串以 ulowsurr() 为真的 Unicode 值结尾，将提取少 1 （或多 1）个字符的子字符串，这是因为 substr() 不会提取包含不匹配代理对的字符串。
uscalar	((*uchar_expr*)	返回表达式中第一个 Unicode 字符的 Unicode 标量值。如果第一个字符 *不是* 代理对的高阶半部，则值的范围将在 0 和 0xFFFF 之间。如果第一个字符 *是* 代理对的高阶半部，则第二个值必须是低阶半部，并且返回值的范围在 0x10000 和 0x10FFFF 之间。如果在包含不匹配代理对半部的 uchar_expr 上调用该函数，将引发 SQLSTATE 例外、输出错误并中止操作。

使用字符串函数的示例

本节中的示例使用以下系统函数：

- charindex、 patindex
- str
- stuff
- soundex、 difference
- substring

charindex、*patindex*

charindex 和 patindex 函数返回指定模式的起始位置。它们都使用两个参数，但是稍有不同，因为 patindex 可以使用通配符，但 charindex 不可以。 charindex 仅可用于 char、 nchar、 unichar、 univarchar、 varchar、 nvarchar、 binary 和 varbinary 列； patindex 可用于 char、 nchar、 unichar、 univarchar、 varchar、 nvarchar、 text 和 unitext 列。

两个函数都使用两个参数。第一个参数是任意的位置模式。使用 patindex 时，必须在该模式的前后添加百分号，但在您希望该模式为列的首字符 （省略前面的 %）或者最后字符 （省略后面的 %）时除外。对于 charindex，该模式不能包括通配符。第二个参数是字符表达式，通常是 Adaptive Server 在其中搜索指定模式的列名。

若要使用这两个函数查找“wonderful”模式在 titles 表的 notes 列的某行中的开始位置，请输入：

```
select charindex("wonderful", notes),
    patindex("%wonderful%", notes)
from titles
where title_id = "TC3218"

------------ -------------
          46            46

(1 row affected)
```

如果不对要搜索的行加以限制，该查询将返回表的所有行，并将不包含该模式的行报告为零值。在下面的示例中， patindex 将查找以 “sys” 开始并且第四个字符是 “a”、“b”、“c”、或 “d” 的 sysobjects 的所有行：

```
select name
from sysobjects
where patindex("sys[a-d]%", name) > 0
name
------------------------------
sysalternates
sysattributes
syscolumns
syscomments
sysconstraints
sysdepends

(6 rows affected)
```

str

str 函数可将数字转换成字符，其可选参数用于指定数字的长度 （包括符号、小数点和小数点左右两侧的位数）和小数点后的位数。

将长度和小数参数设置为 str 正数。缺省长度为 10。缺省小数为 0。该长度应足以容纳小数点和数字符号。结果的小数部分将舍入到指定的长度之内。但是，如果数字的整数部分超出了该长度， str 将返回一行具有指定长度的星号。

例如：

```
select str(123.456, 2, 4)


--
**

(1 row affected)
```

如果 *approx_numeric* 较短，则在指定长度中右对齐，如果 *approx_numeric* 较长，则被截断到指定的小数位数。

stuff

stuff 函数将一个字符串插入到其它字符串中。它删除 *expr1* 起始位置处中指定长度的字符。然后将 *expr2* 字符串插入到 *expr1* 字符串的起始位置处。如果起始位置或长度为负，则返回 NULL 字符串。

如果起始位置超出 *expr1* 的长度，也将返回 NULL 字符串。如果要删除的长度超出 *expr1* 的长度，将删除 *expr1* 中的全部字符

```
select stuff("abc", 2, 3, "xyz")

----
axyz

(1 row affected)
```

若要使用 stuff 来删除某个字符，应该用 NULL（而不是空引号）来替换 *expr2*。使用 “ ” 指定空字符会将其替换为空格。

```
select stuff("abcdef", 2, 3, null)

---
aef

(1 row affected)

select stuff("abcdef", 2, 3, "")
```

```
----
a ef

(1 row affected)
```

soundex、*difference*

soundex 函数可将字符串转换为四位代码以便于比较。在比较中，忽略元音。非字母字符将终止 soundex 求值。该函数始终会返回一个值。这两个名称有相同的 soundex 代码：

```
select soundex ("smith"), soundex ("smythe")

----- -----
S530  S530
```

difference 函数将比较两个字符串的 soundex 值，并评估二者之间的相似性，最后返回从 0 到 4 的值。最佳匹配值是 4。例如：

```
select difference("smithers", "smothers")

---------
        4

(1 row affected)

select difference("smothers", "brothers")

 ---------
         2

(1 row affected)
```

substring

下面的示例使用 substring 函数。它显示每个作者的姓以及名的首字母，如 “Bennet A”。

```
select au_lname, substring(au_fname, 1, 1)
from authors
```

substring 返回字符或二进制字符串的一部分。

substring 总是带有三个参数。第一个参数是字符或二进制字符串、列名或包括列名的字符串值表达式。第二个参数指定子字符串应该开始的位置。第三个参数指定要返回的字符串的长度 （以字符数表示）。

substring 的语法如下所示：

substring(*expression*, *start*, *length*)

例如，下面显示了如何指定字符串常量 “abcdef”的第二个、第三个和第四个字符：

```
select x = substring("abcdef", 2, 3)

x
---------
bcd

(1 row affected)
```

以下示例显示如何从二进制字段中提取较低的四位数，其中的每个位置都表示两个二进制位：

```
select substring(xactid,5,2) from syslogs

------
0x0000
0x0001
0x0001
0x0001
0x0001
0x0001
0x0001
0x0001
0x0001
0x0001
0x0001

(11 rows affected)
```

其它字符串函数的示例

剩余的大部分字符串函数都易于使用和理解。

表 16-4：字符串函数示例

语句	结果
select right("abcde", 3)	cde
select right("abcde", 6)	abcde
select right(0x12345000, 3)	0x345000
select right(0x12345000, 6)	0x12345000
select upper("torso")	TORSO
select ascii("ABC")	65

并置

使用 + 字符串并置运算符，可以并置二进制或字符串表达式（组合两个或更多的字符或二进制字符串、字符或二进制数据或它们的组合）。并置的字符串的最大长度是 16384 字节。

并置字符串时，每个字符表达式都必须用单引号或双引号括起来。

并置语法为：

select (*expression* + *expression* [+ *expression*]...)

下面是组合两个字符串的方法：

```
select ("abc" + "def")

-------
abcdef

(1 row affected)
```

此查询在列标题 *Moniker* 下以先姓后名的顺序显示加利福尼亚作者的姓名，并在姓后面加一个逗号和空格：

```
select Moniker = (au_lname + ", " + au_fname)
from authors
where state = "CA"

Moniker
-----------------------------------------------
White, Johnson
Green, Marjorie
Carson, Cheryl
O'Leary, Michael
Straight, Dick
Bennet, Abraham
Dull, Ann
Gringlesby, Burt
Locksley, Chastity
Yokomoto, Akiko
Stringer, Dirk
MacFeather, Stearns
Karsen, Livia
Hunter, Sheryl
McBadden, Heather

(15 rows affected)
```

若要并置数值或 datetime 数据类型，必须使用 convert 函数：

```
select "The due date is " + convert(varchar(30),
    pubdate)
from titles
where title_id = "BU1032"
--------------------------------------
The due date is Jun 12 1986 12:00AM

(1 row affected)
```

并置和空字符串

Adaptive Server 将空字符串 （“ ”或 ‘ ’）作为单个空格求值。以下语句：

```
select "abc" + "" + "def"
```

生成：

```
abc def
```

嵌套字符串函数

可以嵌套字符串函数。例如，若要显示每位作者的姓以及名的首字母，并在姓后加一个逗号且在名后加一个句点，请输入：

```
select (au_lname + "," + " " + substring(au_fname, 1,
1) + ".")
from authors
where city = "Oakland"

-----------------------------------------
Green, M.
Straight, D.
Stringer, D.
MacFeather, S.
Karsen, L.

(5 rows affected)
```

若要显示标价在 $20 以上的书籍的 pub_id 和每个 title_id 的前两个字符，请输入：

```
select substring(pub_id + title_id, 1, 6)
from titles
where price > $20
--------------
```

```
1389PC
0877PS
0877TC

(3 rows affected)
```

用于 *text*、 *unitext* 和 *image* 数据的文本函数

文本内置函数用于 text、 image 和 unitext 数据操作。表 16-5 列出了文本函数名、参数和结果：

表 16-5：用于 text、 unitext 和 image 数据的内置文本函数

函数	参数	结果
patindex	(“%*pattern*%”, *char_expr* [using {bytes | chars | characters}])	返回一个整数值，表示指定字符表达式中首次出现的 *pattern* 的起始位置；如果未找到 *pattern*，则返回 0。缺省情况下， patindex 返回的偏移以字符表示；若要返回以字节表示的多字节字符串的偏移，请指定 using bytes。 *pattern* 的前后必须添加通配符 % （搜索第一个或最后一个字符的情况除外）。有关可用于 *pattern* 的通配符的说明，请参见第 43 页的 “查询结果中的字符串”。
textptr	((*text_columname*)	返回文本指针值 （16 字节的 varbinary 值）。
textvalid	(“*table_name..col_name*”, *textpointer*)	检查给定的 text 指针是否有效。 text、 unitext 或 image 列的标识符必须包含表名。指针有效时返回 1，无效时返回 0。

注释　有关将这些函数用于 unitext 数据时的限制，请参见 《参考手册：构件块》中各个函数的 “用法”部分。

datalength 也可用于 text 列。有关 datalength 的信息，请参见第 489 页的 “返回数据库信息的系统函数”。

set textsize 命令指定使用 select 语句返回的 text、 image 和 unitext 数据的限制 （用字节表示）。例如，该命令可将用 select 语句返回的 text、 image 和 unitext 数据的限制设置为 100 字节：

```
set textsize 100
```

当前设置存储在 *@@textsize* 全局变量中。缺省设置由客户端程序控制。若要重新设置缺省值，请发出：

```
set textsize 0
```

也可以使用 @@*textcolid*、 @@*textdbid*、 @@*textobjid*、 @@*textptr* 和 @@*textsize* 全局变量来处理 text、 unitext 和 image 数据。

readtext

如果只想检索某列数据的选定部分， readtext 命令提供了一种检索 text、 unitext 和 image 值的方法。 readtext 需要表和列的名称、文本指针、列中的起始偏移量以及要检索的字符或字节数。

holdlock 标志将锁定文本值以供读取，直到事务结束为止。其他用户可以读取该值，但不能修改。有关 at isolation 子句的说明，请参见第 22 章“事务：维护数据一致性和恢复”。

如果使用多字节字符集，则可以通过 using 选项选择是希望 readtext 将偏移和大小解释为字节，还是解释为字符。 chars 和 characters 都可以指定字符。当用于单字节字符集或 image 值（readtext 只能逐字节读取 image 值）时，该选项无效。如果未给出 using 选项，readtext 将按照您指定字节时的设置返回值。

Adaptive Server 必须确定要发送到客户端的字节数，才能响应 readtext 命令。当偏移和大小用字节数表示时，要确定返回文本中的字节数很简单。当偏移和大小用字符数表示时， Adaptive Server 必须计算返回到客户端的字节数。因此，在使用字符表示偏移和大小时，执行速度可能会较慢。只有在 Adaptive Server 使用多字节字符集时，才使用 using characters。该选项可以确保 readtext 不会返回部分字符。

在使用字节数表示偏移时， Adaptive Server 可能会在要返回的 text 数据的开头或末尾找到部分字符。如果是这样，则服务器会在将文本返回到客户端之前，用问号替代每个部分字符。

readtext 不能用于视图中的 text、 unitext、和 image 列。

对 *unitext* 列使用 *readtext*

在对定义为 unitext 数据类型的列发出 readtext 时， readtext *offset* 参数指定在开始读取 unitext 数据之前要跳过的字节数或 Unicode 值。 readtext *size* 参数指定要读取的字节数或 16 位 Unicode 值。如果指定 using bytes（缺省值），则在必要时， *offset* 和 *size* 值将被调整为始终以 Unicode 字符边界开始和结束。

如果使用 enable surrogate processing， readtext 将仅截断代理边界，并可能调整起始或终止位置。因此，如果对定义为 unitext 的列发出 readtext，则返回的字节数可能会少于指定的字节数。

在下面的示例中， unitext 列 ut 包含字符串 U+0101U+0041U+0042U+0043：

```
declare @val varbinary(16)
select @val = textptr(ut) from unitable
where i = 1
readtext foo.ut @val 1 5
```

该查询将返回值 U+0041U+0042。

offset 位置被调整为 2，因为 readtext 不能从 Unicode 字符的第二个字节开始。 Unicode 字符总是由偶数个字节组成。从第二个字节开始 （或以奇数字节结束）会使结果偏移一个字节，从而导致结果集不正确。

在上面的示例中，*size* 值被调整为 4，因为 readtext 无法读取第四个字符 U+0043 的部分字节。

在下面的查询中，启用了 enable surrogate processing，并且 ut 列包含字符串 U+d800dc00U+00c2U+dbffdeffU+d800dc00：

```
declare @val varbinary(16)
select @val = textptr(ut) from unitable
where i = 2
readtext unitable.ut @val 1 8
```

该查询将返回值 U+00c2U+dbffdeff。起始位置被重新设置为 4，并且实际的结果大小为 6 个字节而不是 8 个字节，原因是 readtext 未拆分代理对。代理对 （在此示例中为范围 d800..dbff 中的第一个值和范围 dc00..dfff 中的第二个值）需要 4 字节边界，但遵从 UTF-16 的 Unicode 规则不允许拆分这些 4 字节字符。

使用文本函数的示例

以下示例使用 textptr 函数来定位 text 列 copy，该列与 blurbs 表中的 title_id BU7832 相关联。文本指针 （16 字节二进制字符串）放在局部变量 *@val* 中并作为 readtext 命令的参数提供。 readtext 将从第二个字节开始返回 5 个字节 （偏移量为 1）。

```
declare @val binary(16)
select @val = textptr(copy) from blurbs
where au_id = "486-29-1786"
readtext blurbs.copy @val 1 5
```

textptr 将返回 16 字节的 varbinary 字符串。Sybase 建议将该字符串放在局部变量中 （如上例所示），并通过引用来使用它。

在前面的 declare 示例中，可以使用 @@*textptr* 全局变量作为使用 textptr 的替代方法：

```
readtext texttest.blurb @@textptr 1 5
```

当前 Adaptive Server 进程可以将 @@*textptr* 的值从最后一个 insert 或 update 设置为 *text*、 unitext 或 *image* 字段的值。由其它进程执行的插入和更新并不影响当前进程。

使用 convert 函数时，受支持的显式转换包括：从 text 到 char、 nchar、 unichar、 varchar、 univarchar 或 nvarchar，以及从 image 到 varbinary 或 binary。这种转换受字符数据类型的最大长度的限制，该长度由服务器逻辑页大小的最大列大小决定。如果未指定该数据类型的长度，则转换后的值将具有缺省的长度 （30 个字节）。

还允许在 text/image 和 unitext 之间进行隐式转换。不支持将 text、 unitext 或 image 隐式或显式地转换成上述数据类型之外的数据类型。

集合函数

集合函数生成的汇总值在查询结果中显示为新列。表 16-6 列出了集合函数、其参数以及它们返回的结果。

表 16-6：集合函数

函数	参数	结果
avg	(all | distinct) *expression*	返回所有 （不同）值的数字平均值
count	(all | distinct) *expression*	以 integer 形式返回 （不同）非空值的数量或行数
count_big	(all | distinct) *expression*	以 bigint 形式返回 （不同）非空值的数量或行数
max	((*expression*)	返回表达式中的最大值
min	((*expression*)	返回列中的最小值
sum	(all | distinct) *expression*	返回值的总和

示例如下：

```
select avg(advance), sum(total_sales)
from titles
where type = "business"

------------------------ -----------
                 6281.25       30,788

(1 row affected)
```

```
select count(distinct city) from authors

-----------
         16

(1 row affected)

select discount from salesdetail
compute max(discount)

discount
--------------------
           40.000000
           ...
           46.700000

Compute Result:
--------------------
           62.200000

(117 rows affect)

select min(au_lname) from authors
----------------------
Bennet

(1 row affected)
```

数学函数

数学内置函数返回通常在数学数据运算中所需的值。

数学函数的一般形式为：

function_name(*arguments*)

表 16-7 列出了用于内置数学函数的参数类型。

表 16-7：数学函数中使用的参数

参数类型	可替换为
approx_numeric	任何近似数值类型 （float、 real 或 double precision）的列名、变量、常量表达式或它们的组合
integer	任意整数类型 （tinyint、 smallint、 int、 bigint、 unsigned smallint、 unsigned int、 unsigned bigint）的列名、变量、常量表达式或它们的组合

参数类型	可替换为
numeric	任意精确数值 （numeric、 dec、 decimal、 tinyint、 smallint、 int、 bigint、 unsigned smallint、 unsigned int 或 unsigned bigint）、近似数值 （float、 real 或 double precision）或 money 类型的列、变量、常量表达式或它们的组合
power	任意精确数值、近似数值或 money 类型的列、变量、常量表达式或它们的组合

每种函数还可接受隐式地转换为指定类型的参数。例如，接受近似数值类型的函数还可接受整数类型。Adaptive Server 会将参数转换成所需类型。

如果函数使用多个同一类型的表达式，表达式将被编号 （例如，*approx_numeric1*、 *approx_numeric2*）。

表 16-8 列出了数学函数、采用的参数及其返回的结果。

表 16-8：数学函数

函数	参数	结果
abs	((numeric)	返回给定表达式的绝对值。结果与数值表达式属于同一类型并且具有相同的精度和标度。
acos	((*approx_numeric*)	返回已指定余弦值的角 （以弧度表示）。
asin	((*approx_numeric*)	返回已指定正弦值的角 （以弧度表示）。
atan	((*approx_numeric*)	返回已指定正切值的角 （以弧度表示）。
atn2	(*approx_numeric1*, *approx_numeric2*)	返回正切值是 (*approx_numeric1*/*approx_numeric2*) 的角 （以弧度表示）。
ceiling	((numeric)	返回大于或等于指定值的最小整数。其结果与数值表达式的类型相同。对于 numeric 和 decimal 表达式，其结果的精度与该表达式的精度相同，且标度为 0。
cos	((*approx_numeric*)	返回指定角 （以弧度表示）的三角余弦。
cot	((*approx_numeric*)	返回指定角 （以弧度表示）的三角余切。
degrees	((numeric)	将弧度转换为度。其结果与数值表达式的类型相同。对于 numeric 和 decimal 表达式，其结果的内部精度为 77，标度与该表达式的标度相同。当使用 money 数据类型时，如果将其在内部转换为 float，则可能导致精度损失。
exp	((*approx_numeric*)	返回指定值的指数值。
floor	(numeric)	返回小于或等于指定值的最大整数。其结果与数值表达式的类型相同。对于 numeric 或 decimal 类型的表达式，其结果的精度与表达式的精度相同，标度为 0。
lockscheme	(*object_name*) 或 (*object_id* [, *db_id*])	以字符串形式返回指定对象的锁定方案。
log	(*approx_numeric*)	返回指定值的自然对数。

函数	参数	结果
log10	(*approx_numeric*)	返回指定值的以 10 为底的对数。
pagesize	(*object_name* [, *index_name*])	以字节为单位返回指定对象的页大小。
pi	()	返回常数值 3.1415926535897931。
power	(numeric, power)	返回求 numeric 的 power 次幂的值。其结果与 numeric 的类型相同。对于 numeric 或 decimal 类型的表达式，其结果的精度是 77，标度与该表达式的标度相同。
radians	(*numeric_expr*)	将度转换为弧度。其结果与 numeric 的类型相同。对于 numeric 或 decimal 类型的表达式，其结果的内部精度是 77，标度与 numeric 表达式的标度相同。当使用 money 数据类型时，如果将其在内部转换为 float，则可能导致精度损失。
rand	([*integer*])	使用可选整数作为源值，返回 0 和 1 之间的随机浮动值。
round	(numeric, *integer*)	舍入 numeric 以使其具有 *integer* 个有效位数。正 *integer* 确定小数点右边的有效位数；负 *integer* 确定小数点左边的有效位数。其结果与数值表达式的类型相同，对于 numeric 和 decimal 表达式，其结果的内部精度等于第一个参数的精度加 1，其标度等于 numeric 表达式的标度。
sign	(numeric)	返回 numeric 的符号：正 (+1)、零 (0) 或负。其结果与 numeric 表达式属于同一类型并且具有相同的精度和标度。
sin	(*approx_numeric*)	返回指定角 （以弧度表示）的三角正弦。
square	(*numeric_expression*)	返回指定值 （表示为 float 类型）的平方值。
sqrt	(*approx_numeric*)	返回指定值的平方根。值必须是正数或零。
tan	(*approx_numeric*)	返回指定角 （以弧度表示）的三角正切。

使用数学函数的示例

数学内置函数对数值数据进行运算。某些函数需要整数数据，而其它函数则可以使用近似数值数据。很多函数可对精确数值、近似数值、money 和 float 类型进行运算。内部运算 float 类型的数据的精度缺省为六位小数。

为处理数学函数的域错误或范围错误，提供了错误陷阱。用户可以对 arithabort 和 arithignore 选项执行 set 来确定处理域错误的方式。有关这些选项的详细信息，请参见第 524 页的 “转换错误”。

表 16-9 显示了一些简单的数学函数示例。

表 16-9：数学函数的示例

语句	结果
select floor(123)	123
select floor(123.45)	123.000000
select floor(1.2345E2)	123.000000
select floor(-123.45)	-124.000000
select floor(-1.2345E2)	-124.000000
select floor($123.45)	123.00
select ceiling(123.45)	124.000000
select ceiling(-123.45)	-123.000000
select ceiling(1.2345E2)	124.000000
select ceiling(-1.2345E2)	-123.000000
select ceiling($123.45)	124.00
select round(123.4545, 2)	123.4500
select round(123.45, -2)	100.00
select round(1.2345E2, 2)	123.450000
select round(1.2345E2, -2)	100.000000

round(*numeric, integer*) 函数总返回值。如果 *integer* 为负数并超过了 *numeric* 中的有效位数，则 Adaptive Server 只舍入最高有效位。例如，以下语句将返回值 100.00：

```
select round(55.55, -3)
```

小数点右边零的个数与第一个参数的精度加 1 的标度相等。

日期函数

日期内置函数执行算术运算并显示有关 datetime、smalldatetime、date 和 time 值的信息。

Adaptive Server 将 datetime 数据类型的值在内部存储为两个四字节的整数。前 4 个字节存储基准日期（以 1900 年 1 月 1 日为基准日期）以前或以后的天数。基准日期是系统参考日期。 datetime 值不允许早于 1753 年 1 月 1 日。内部 datetime 表示中的另外四个字节存储一天中的时间，可精确到 1/300 秒。

date 数据类型存储为四个字节。基准日期的范围是从 0001 年 1 月 1 日到 9999 年 12 月 31 日。 time 数据类型涵盖从 12:00:00AM 到 11:59:59:999PM 的时间。

smalldatetime 数据类型存储的日期和时间精度低于 datetime。 smalldatetime 值存储为两个双字节整数。前两个字节存储 1900 年 1 月 1 日以后的天数。后两个字节存储午夜后的分钟数。日期的范围是从 1900 年 1 月 1 日到 2079 年 6 月 6 日，可精确到分。

缺省的日期显示格式如下所示：

```
Apr 15 1997 10:23PM
```

有关更改 datetime 或 smalldatetime 的显示格式的信息，请参见第 519 页的 “使用常规用途的转换函数：convert”。当输入 datetime、smalldatetime、 date 和 time 值时，请使用单引号或双引号将它们括起来。 Adaptive Server 可能会舍入或截断毫秒值。

Adaptive Server 可识别多种 datetime 数据条目格式。有关 datetime、smalldatetime、 date 和 time 值的详细信息，请参见第 7 章 “创建数据库和表”和第 8 章 “添加、更改、传输和删除数据”。

表 16-10 列出了日期函数及其生成的结果：

表 16-10：日期函数

函数	参数	结果
current_date	日期	返回当前日期
current_time	日期	返回当前时间
day	(*date_expression*)	返回一个整数，表示指定日期的 datepart 中的日子
datename	(*datepart*, *date*)	datetime、 smalldatetime、 date 或 time 值中以 ASCII 字符串表示的部分
datepart	(*datepart*, *date*)	datetime、 smalldatetime、 date 或 time 值（例如，月份）中以整数表示的部分

函数	参数	结果
datediff	(*datepart*, *date*, *date*)	两个日期中第二个和第一个日期间的时间量，该时间量已转换为指定的日期分量（如月、日、小时）
dateadd	(*datepart*, *number*, *date*)	通过向其它日期添加日期分量所产生的日期
getdate	()	返回当前的系统日期和时间。
getutcdate	()	返回一个日期，其值为协调通用时间（有时称为格林威治标准时间）。该值将不会随时区进行调整，也不会被高速缓存；每次调用该函数时，都将返回一个新值。
month	(*date_expression*)	返回一个整数，表示指定日期的 datepart 中的月份
year	(*date_expression*)	返回一个整数，表示指定日期的 datepart 中的年份

datename、 datepart、 datediff 和 dateadd 函数将**日期分量**（年、月、小时等）作为参数。 datename 函数在适当情况下产生 ASCII 值（如为一周中的某天）。

datepart 返回符合 ISO 标准 8601 的数字，此标准定义了一周中的第一天和一年中的第一周。根据 datepart 函数包含的是 calweekofyear 值、calyearofweek 值还是 caldayofweek 值，对同一时间单元返回的日期可能不同。例如，如果将 Adaptive Server 配置为以美国英语为缺省语言：

```
datepart(cyr, "1/1/1989")
```

返回 1988，而：

```
datepart(yy, "1/1/1989)
```

返回 1989。

出现这种不一致是由于该 ISO 标准将一年的第一周定义为包含周四*并且*以周一开始的第一周。

如果服务器以美国英语为缺省语言，则一周的第一天是周日，一年的第一周是包含 1 月 4 日的那一周。

表 16-11 列出了每个日期分量、其缩写（如果有）和该日期分量可能的整数值。

表 16-11：日期分量

日期分量	缩写	值
year	yy	1753–9999
quarter	qq	1–4
month	mm	1–12
week	wk	1–54
day	dd	1–31
dayofyear	dy	1–366
weekday	dw	1– 7（星期日 – 星期六）
hour	hh	0–23
minute	mi	0–59
second	ss	0–59
millisecond	ms	0–999

```
select datename (mm, "1997/06/16")
-----------
        June
(1 row affected)


select datediff (yy, "1984", "1997")
----------
     13

(1 row affected)

select dateadd (dd, 16, "1997/06/16")
------------------------------
           Jul  2 1997 12:00AM

(1 row affected)
```

注释　语言设置将影响 weekday 的值。

以下是一些周日期分量的示例：

```
select datepart(cwk,"1997/01/31")

-----------
          5

(1 row affected)

select datepart(cyr,"1997/01/15")

-----------
```

```
        1997

(1 row affected)
select datepart(cdw,"1997/01/24")
-----------
          5

(1 row affected)
```

表 16-12 列出了周数日期分量、其缩写和值。

表 16-12：周数日期分量

日期分量	缩写	值
calweekofyear	cwk	1–52
calyearofweek	cyr	1753–9999
caldayofweek	cdw	1 - 7 （在美国英语中 1 是星期一）

获取当前日期：*getdate*

getdate 函数以 Adaptive Server 内部格式产生 datetime 值的当前日期和时间。 getdate 采用 NULL 参数 ()。

若要查找当前的系统日期和时间，请键入：

```
select getdate()
--------------------------
Aug 19 1997 12:45PM

(1 row affected)
```

在设计报告中可使用 getdate，以便每次产生报告时输出当前日期和时间。 getdate 对记录帐户上事务的发生时间也是很有用。

若要使用毫秒显示日期，请使用 convert 函数。例如：

```
select convert(char(26), getdate(), 109)
--------------------------
Aug 19 1997 12:45:59:650PM

(1 row affected)
```

有关详细信息，请参见第 526 页的 “更改日期显示格式”。

查找作为数字或名称的日期分量

datepart 和 datename 函数可以整数或 ASCII 字符串形式产生 datetime、smalldatetime 或 date 值的指定分量（年、季度、日、小时等）。由于 smalldatetime 仅精确到分钟，因此当在这些函数中使用 smalldatetime 值时，秒和毫秒始终为 0。如果使用 time 数据类型，则会给出秒和毫秒。

以下示例假设日期为 8 月 12 日：

```
select datepart(month, getdate())
--------------

            8
(1 row affected)

select datename(month, getdate())
-------------
August

(1 row affected)
```

计算间隔或增量日期

datediff 函数计算指定的两个日期的第一个和第二个之间日期分量的时间，换句话说就是找到两个日期间的间隔。其结果是带符号的整数值（以日期分量表示），它等于 *date2* - *date1*。

以下查询使用日期 1990 年 11 月 30 日并查找 pubdate 和该日期之间的天数：

```
select pubdate, newdate = datediff(day, pubdate,
    "Nov 30 1990")
from titles
```

对于 titles 表中 pubdate 为 1990 年 10 月 21 日的行，上述查询产生的结果将是 40，即在 10 月 21 日和 11 月 30 日之间的天数。若要计算以月份表示的间隔，则查询应为：

```
select pubdate, interval = datediff(month, pubdate,
    "Nov 30 1990")
from titles
```

此查询对 pubdate 为 1990 年 10 月的行产生值 1，而对 pubdate 为 1990 年 6 月的行产生值 5。如果 datediff 函数中的第一个日期晚于第二个日期，则所得的值为负数。由于 titles 中有两个行的 pubdate 将通过使用 getdate 函数分配的值作为缺省值，因此这些值会被设置为 pubs 数据库的创建日期，并在上述两个查询中返回负值 (-65)。

如果两个日期参数中的一个或两者都是 smalldatetime 值，它们将在内部被转换成 datetime 值以用于计算。为了计算差值， smalldatetime 值中的秒和毫秒会被自动设置为 0。

添加日期间隔：*dateadd*

dateadd 函数向指定的日期添加间隔 （指定为整数）。例如，如果 titles 表中所有书的出版日期都延期 3 天，则可以使用以下语句获得新的出版日期：

```
select dateadd(day, 3, pubdate)
from titles

-------------------
Jun 15 1986 12:00AM
Jun 12 1988 12:00AM
Jul  3 1985 12:00AM
Jun 25 1987 12:00AM
Jun 12 1989 12:00AM
Jun 21 1985 12:00AM
Jul  6 1997  1:43PM
Jul  3 1986 12:00AM
Jun 25 1987 12:00AM
Jul  6 1997  1:43PM
Oct 24 1990 12:00AM
Jun 12 1989 12:00AM
Oct  8 1990 12:00AM
Jun 12 1988 12:00AM
Jun 12 1988 12:00AM
Oct 24 1990 12:00AM
Jun 21 1985 12:00AM
Jun 25 1987 12:00AM

(18 rows affected)
```

如果给定的日期参数是 smalldatetime，则结果也会是 smalldatetime。可使用 dateadd 向 smalldatetime 添加秒或毫秒，但只有在 dateadd 返回的结果日期更改了至少一分钟时，该操作才有意义。

数据类型转换函数

数据类型转换是将一种数据类型更改为另一种数据类型并重新格式化日期和时间信息。 Adaptive Server 提供了三个数据类型转换函数：convert、 inttohex 和 hextoint。

Adaptive Server 自动执行某些数据类型转换。这些转换称作**隐式转换**。例如，如果比较 char 表达式和 datetime 表达式，或比较 smallint 表达式和 int 表达式，或比较具有不同长度的 char 表达式， Adaptive Server 就会自动将一种数据类型转换为另一种数据类型。再举一例，如果将 datetime 值转换为 date 值，时间部分将被截断，只留下日期部分。如果将 time 值转换为 datetime 值，将在新的 datetime 值中添加 Jan 1, 1900 的缺省日期部分。如果将 date 值转换为 datetime 值，则将向 datetime 值添加缺省的时间部分 00:00:00:000。

必须使用一种内置数据类型转换函数显式请求其它数据类型转换。例如，在并置数值表达式之前，必须将其转换为字符表达式。如果尝试显式将 date 转换为 datetime，并且该值不在诸如 “Jan 1, 1000”之类的 datetime 范围内，将不允许这样的转换，并且显示错误消息。

对于某些数据类型，Adaptive Server 不允许将其隐式或显式转换为某些其它数据类型。例如，不能进行以下转换：将 smallint 数据转换为 datetime，将 datetime 数据转换为 smallint 或 binary，或将 varbinary 数据转换为 smalldatetime 或 datetime。进行不受支持的转换将导致出现错误消息。

有关受支持和不受支持的数据类型转换的详细信息，请参见 《参考手册：构件块》。

使用常规用途的转换函数：*convert*

常规转换函数 convert 可在多种数据类型间进行转换，并为日期和时间信息指定新的显示格式。它的语法为：

convert(*datatype, expression* [, *style*])

以下是在选择列表中使用 convert 的示例：

```
select title, convert(char(5), total_sales)
from titles
where type = "trad_cook"

title
------------------------------------     -----
Onions, Leeks, and Garlic:Cooking
    Secrets of the Mediterranean           375
Fifty Years in Buckingham Palace
```

```
    Kitchens                                  15096
Sushi, Anyone?                                 4095

(3 rows affected)
```

在以下示例中， total_sales 列 （一个 int 列）将被转换成 char(5) 列，以便可与 like 关键字一起使用：

```
select title, total_sales
from titles
where convert(char(5), total_sales) like "15%"
    and type = "trad_cook"

title
---------------------------------        -----
Fifty Years in Buckingham Palace
    Kitchens                                  15096

(1 row affected)
```

某些数据类型需要使用长度或精度和标度。如果不指定长度， Adaptive Server 将为字符和二进制数据使用缺省长度 30。如果不指定精度或标度， Adaptive Server 将分别使用缺省值 18 和 0。

转换规则

以下各节说明 Adaptive Server 在转换不同类型的信息时所遵循的规则。

将字符数据转换为非字符类型

可以将字符数据转换为非字符类型 （例如货币、日期和时间、精确数值或近似数值类型），但前提是它必须全部由对于该新类型有效的字符组成。前导空白将被忽略。但是，如果将包含一个或多个空白的 char 表达式转换为 datetime 表达式， Adaptive Server 就会将空白转换为 Sybase 缺省 datetime 值 “Jan 1, 1900”。

当数据包含不可接受的字符时， Adaptive Server 将生成语法错误。以下类型的字符将导致语法错误：

- 整数数据中的逗号或小数点
- 货币数据中的逗号
- 精确 / 近似数值数据或位流数据中的字母
- 日期 / 时间数据中的错误拼写月份名

支持 unichar/univarchar 和 datetime/smalldatetime 之间的隐式转换。

将一种字符类型转换为另一种字符类型

text 和 unitext 列可显式转换为 char、nchar、unichar、varchar、univarchar 或 nvarchar。但受到字符数据类型最大长度 （页大小）的限制。如果未指定长度，转换后的值将具有缺省的长度 （30 个字节）。

将 Unicode 字符数据类型转换为其它字符数据类型时，无对等值的字符将被替换为问号。

将数字转换为字符类型

可将精确和近似数值数据转换为字符类型。如果新类型太短，无法容纳整个字符串，则将生成空间不足的错误。例如，以下转换尝试以单字符类型存储由 5 个字符组成的字符串：

```
select convert(char(1), 12.34)
```

因为 char 数据类型仅限包括一个字符，而数值 12.34 需要 5 个字符才能成功转换，所以转换失败。

从货币类型转换或转换成货币类型时舍入

money 和 smallmoney 类型在小数点的右侧存储 4 个数字位，但为了便于显示，它们将上舍入到最接近的百分位 (.01)。当数据转换为 money 类型时，它将上舍入到第四个小数位。

如果可能，从 money 类型转换而来的数据将遵循相同的舍入规则。如果新类型是小数位少于 3 的精确数值，数据就会舍入到新类型的标度。例如，当 $4.50 转换为整数后，将得到 5：

```
select convert(int, $4.50)
-----------
          5

(1 row affected)
```

Adaptive Server 假定转换为 money 或 smallmoney 的数据是完整货币单位（如美元），而不是辅币单位 （如美分）。例如，整数值 5 在美国英语语言中将转换为 5 美元 （而不是 5 美分）的等值货币。

转换日期和时间信息

可以将可识别为日期的数据转换为 datetime、 smalldatetime、 time 和 date。月份名有误将导致语法错误。超出数据类型可接受范围的日期将导致算术溢出错误。

当 datetime 值转换为 smalldatetime 时，它们将上舍入到最接近的分。

从 unitext 转换或转换成 unitext

可以隐式地将其它字符和二进制数据类型转换成 unitext。也可以在 unitext 和其它数据类型之间进行显式转换。但是，转换结果会受到目标数据类型的最大长度的限制。如果 Unicode 字符边界上的目标缓冲区无法容纳 unitext 值，则会截断数据。如果启用了 enable surrogate processing，则决不会在值代理对的中间截断 unitext 值，这意味着在数据类型转换之后可能会返回更少的字节。例如，如果表 tb 的 unitext 列 ut 中存储了字符串 “U+0041U+0042U+00c2”（U+0041 表示 Unicode 字符 “A”）并且服务器的字符集设置为 UTF-8，则下面的查询会返回值 “AB”，这是因为 U+00C2 被转换为 2 个字节的 UTF-8 0xc382：

```
select convert(char(3), ut) from tb
```

表 16-13：从 unitext 转换和转换成 unitext

下面的数据类型可隐式转换*成* unitext	下面的数据类型可*从* unitext 隐式转换	下面的数据类型可*从* unitext 显式转换
char、 varchar、 unichar、 univarchar、 binary、 varbinary、 text、 image	text、 image	char、 varchar、 unichar、 univarchar、 binary、 varbinary

目前，alter table modify 命令不支持将 text、image 或 unitext 列作为被修改的列。若要从 text 迁移到 unitext 列，必须首先使用 bcp out 拷出现有数据并创建包含 unitext 列的表，然后再使用 bcp in 将数据放入新表中。只有用 -Jutf8 选项调用 bcp 时，此迁移路径才有效。

在数字类型之间转换

数据可以从一种数字类型转换为另一种数字类型。如果新类型是精确数值类型，且其精度或标度不足以容纳数据，就会发生错误。使用 arithabort 和 arithignore 选项可确定处理这些错误的方式。

注释 arithabort 和 arithignore 选项在 SQL Server 版本 10.0 中重新进行了定义。如果您在应用程序中使用这些选项，请检查它们是否能正确实现预期功能。请参见 《参考手册：构件块》。

转换二进制式数据

Adaptive Server binary 和 varbinary 数据是特定于平台的数据；您所使用的硬件类型将决定数据的存储和解释方式。一些平台认为前缀 “0x”后第一个字节最重要；而其它平台则认为第一个字节最不重要。

convert 函数将 Sybase 二进制数据作为一个字符串对待，而不是数值信息。当将二进制表达式转换为整数或整数表达式时， convert 不考虑字节顺序对二进制值的重要性。因此，在不同平台上可能会得到不同的转换结果。

在将二进制字符串转换为整数前， convert 将删去其前缀 “0x”。如果字符串包含的位数为奇数， Adaptive Server 将插入前导零。如果数据太长，不适于转换为整数类型， convert 将截断它。如果数据太短， convert 会添加前导零将数据左侧填平，然后在其右侧用零填充。

假定您要将字符串 0x00000100 转换为整数。该字符串在某些平台上代表数字 1；在其它平台上代表 256。 convert 返回 1 还是 256，取决于执行函数的平台。

转换十六进制数据

要获得跨平台可靠的转换结果，可使用 hextoint 和 inttohex 函数。

类似函数 hextobigint 和 biginttohex 可用于转换为 64 位的整数，以及从 64 位的整数进行转换。

hextoint 接受由数字和大小写字母 A–F （不论是否有前缀 “0x”）组成的文字或变量。以下都是 hextoint 的有效用法：

```
select hextoint("0x00000100FFFFF")
select hextoint("0x00000100")
select hextoint("100")
```

hextoint 删去字符串的前缀 “0x”。如果数据超过 8 位，则 hextoint 会截断它。如果数据少于 8 位， hextoint 将使数据向右对齐并用零填充。这样， hextoint 将返回独立于平台的等值整数。无论执行 hextoint 函数的平台是何种类型，上述表达式都将返回相同值 256。

inttohex 函数接受整数数据并返回没有前缀 “0x”的 8 字符**十六进制字符串**。无论使用何种平台， inttohex 总返回相同的结果。

将 *image* 数据转换为 *binary* 或 *varbinary*

使用 convert 函数可将 image 列转换为 binary 或 varbinary。这种转换受 binary 数据类型的最大长度 （即服务器的页大小）的限制。如果未指定长度，转换后的值将具有缺省的长度 （30 个字符）。

在二进制和数字或十进制类型之间转换

在 binary 和 varbinary 数据字符串中，“0x”之后的前两位表示 binary 类型：“00”表示正数，“01”表示负数。将 binary 或 varbinary 类型转换为 numeric 或 decimal 时，务必要在 “0x”位后指定 “00”或 “01”值；否则转换将失败。

例如，以下是将 binary 数据转换为 numeric 的方法：

```
select convert(numeric
(38, 18),0x00000000000000006b14bd1e6eea0000000000000000000000000000000000)

----------------------
123.456000000000000000
```

以下示例将同一 numeric 数据转换回 binary：

```
select convert(binary,convert(numeric(38, 18), 123.456))

------------------------------------------------------------------
0x00000000000000006b14bd1e6eea0000000000000000000000000000000000
```

转换错误

本节说明数据类型转换期间可能产生的错误类型。

算术溢出和除零错误

当 Adaptive Server 尝试将数字值除以零时，就会发生除零错误。当新类型的小数位数太少，无法容纳结果时，就会发生算术溢出错误。会发生此错误的转换过程包括：

- 显式或隐式转换为具有较低精度或标度的精确类型
- 显式或隐式转换超出货币或日期时间类型的可接受范围的数据
- 使用 hextoint 转换大于 4 个字节的字符串

无论是在隐式转换还是在显式转换中，发生算术溢出和除零错误都是很严重的问题。使用 arithabort arith_overflow 选项可确定 Adaptive Server 处理这些错误的方法。缺省设置为 arithabort arith_overflow on，它将回退发生错误的整个事务。如果将其设置为 arithabort arith_overflow off，Adaptive Server 将中止导致错误的语句，但会继续处理事务或批处理中的其它语句。可以使用 *@@error* 全局变量来检查语句结果。

使用 arithignore arith_overflow 选项可确定 Adaptive Server 是否在发生这些错误后显示消息。缺省设置为 off，它将在发生除零错误或精度损失时显示警告消息。如果设置 arithignore arith_overflow on，则会取消发生这些错误后的警告消息。可省略可选关键字 arith_overflow 而不会有任何影响。

标度错误

当显式转换导致标度损失时，将截断结果而不发出任何警告。例如，如果将 float、numeric 或 decimal 类型显式转换为 integer，Adaptive Server 会认为您确实要使结果成为整数，从而会截断小数点右侧的所有数字。

当隐式转换为 numeric 或 decimal 类型时，标度损失将导致标度错误。使用 arithabort numeric_truncation 选项可确定这种错误的严重程度。缺省设置为 arithabort numeric_truncation on，它将中止导致错误的语句，但会继续处理事务或批处理中的其它语句。如果设置了 arithabort numeric_truncation off， Adaptive Server 就会截断查询结果并继续进行处理。

域错误

当 convert 函数的参数超出其定义范围时，该函数就会产生域错误。这种错误极少发生。

二进制和整数类型之间的转换

binary 和 varbinary 类型存储十六进制式数据，这种数据由前缀 “0x”以及随后的数字和字母字符串组成。不同的平台会以不同的方式来解释这些字符串。例如，字符串 0x0000100 在认为字节 0 最重要的计算机上代表 65,536 而在认为字节 0 最不重要的计算机上代表 256。

cout_big 函数

count_big 是一个集合函数，它查找列中非空值的数量。以 bigint 形式返回 （不同）非空值的数量或选定的行数。

转换函数和隐式转换

二进制类型向整数类型的转换可显式进行 （使用 convert 函数），该转换也可隐式进行。数据的前缀 “0x”被删除，然后用零填充 （如果对于新类型太短）或被截断 （如果对于新类型太长）。

convert 和隐式数据类型转换在不同的平台上会以不同的方式对二进制数据求值。因此，在不同平台上可能会得到不同的结果。使用 hextoint 函数可进行与平台无关的十六进制字符串到整数的转换，而使用 inttohex 函数可进行与平台无关的整数到十六进制值的转换。

十六进制函数

使用 hextoint 函数可进行与平台无关的十六进制数据到整数的转换。hextoint 接受用引号括起来的有效十六进制字符串 （带有或不带有 “0x”前缀均可），也接受字符类型列或变量的名称。

hextoint 返回十六进制字符串的等值整数。对于给定的十六进制字符串，在任何平台上执行该函数都始终返回同一个等值整数。

inttohex 函数

使用 inttohex 函数可进行与平台无关的整数到十六进制字符串的转换。inttohex 接受任何求值为整数的表达式。对于给定的表达式，在任何平台上执行该函数都始终返回同一个等值十六进制数。

hextobigint 函数

使用 hextobigint 函数可进行与平台无关的十六进制数据到 64 位整数的转换。 hextobigint 接受用引号括起来的有效十六进制字符串 （带或不带“0x”前缀均可），也接受字符类型列或变量的名称。

hextobigint 返回十六进制字符串的 64 位等值整数。对于给定的十六进制字符串，在任何平台上执行该函数都始终返回同一个 64 位等值整数。

biginttohex 函数

使用 biginttohex 函数可进行与平台无关的 64 位整数到十六进制字符串的转换。 biginttohex 接受任何求值为 64 位整数的表达式。对于给定的表达式，在任何平台上执行该函数都始终返回同一个等值十六进制数。

将图像列转换为二进制类型

可使用 convert 函数将 image 列转换为 binary 或 varbinary。这种转换受 binary 数据类型的最大长度 （即服务器的页大小）的限制。如果未指定长度，转换后的值将具有缺省的长度 （30 个字符）。

将其它类型转换为 bit 类型

精确和近似 numeric 类型可以隐式转换为 bit 类型。字符类型需要使用 convert 函数显式转换。

所转换的表达式必须仅包括数字、小数点、货币符号以及加号或减号。使用其它字符会生成语法错误。

0 的 bit 等值为 0。任何其它数字的 bit 等值为 1。

更改日期显示格式

convert 的 *style* 参数提供了多种日期显示格式，可用于将 datetime 或 smalldatetime 数据转换为 char 或 varchar。作为 *style* 参数提供的数值参数确定数据的显示方式。年份可用 2 或 4 位显示。向 *style* 值添加 100 可获得 4 位的年，包括世纪 (*yyyy*)。

表 10-14 显示了 *style* 的可能值及可用的多种日期格式。将 *style* 和 smalldatetime 一起使用时，包括秒或毫秒的样式将在这些位置显示零。

表 16-14：用 style 参数转换日期格式。有关说明，请参见以下内容

不含世纪 (yy)	含世纪 (yyyy)	标准	输出
-	0 或 100	缺省值	*mon dd yyyy hh:mm* AM （或 PM）
1	101	美国	*mm/dd/yy*
2	2	SQL 标准	*yy.mm.dd*
3	103	英语 / 法语	*dd/mm/yy*
4	104	德语	*dd.mm.yy*
5	105		*dd-mm-yy*
6	106		*dd mon yy*
7	107		*mon dd, yy*
8	108		*HH:mm:ss*

不含世纪 (yy)	含世纪 (yyyy)	标准	输出
-	9 或 109	缺省值 + 毫秒	*mon dd yyyy hh:mm:sss* AM（或 PM）
10	110	美国	*mm-dd-yy*
11	111	日本	*yy/mm/dd*
12	112	ISO	*yymmdd*
13	113		*yy/dd/mm*
14	114		*mm/yy/dd*
15	115		*dd/yy/mm*
-	16 或 116		*mon dd yyyy HH:mm:ss*
17	117		*hh:mmAM*
18	118		*HH:mm*
19			*hh:mm:ss:zzzAM*
20			*HH:mm:ss:zzz*
21			*yy/mm/dd HH:mm:ss*
22			*yy/mm/dd hh:mm* AM （或 PM）
	23		*yyyy-mm-ddTHH:mm:ss*

表的说明： “mon”表示英文拼写的月份，“mm”表示月份或分钟。“HH”表示 24 小时制时钟值，“hh”表示 12 小时制时钟值。最后一行（第 23 行）包含文字 “T”以分离格式中的日期部分和时间部分。

缺省值 （样式 0 或 100 以及 9 或 109）始终返回世纪 (*yyyy*)。

下面是一个使用 convert 的 *style* 参数的示例：

```
select convert(char(12), getdate(), 3)
```

该语句会将当前日期转换为样式 3，即 *dd/mm/yy*。

在将 date 数据转换为字符类型时，将使用表 16-14 中的样式编号 1 到 7（101 到 107）或 10 到 12（110 到 112）来指定显示格式。缺省值为 100（*mon dd yyyy hh:mi*AM （或 PM））。如果将 date 数据转换为包含时间部分的样式，该时间部分将显示缺省值 0。

在将 time 数据转换为字符类型时，将使用样式编号 8 或 9 （108 或 109）来指定显示格式。缺省值为 100 （*mon dd yyyy hh:mi*AM （或 PM））。如果 time 数据转换为包含日期部分的样式，将显示缺省日期 Jan 1, 1900。

注释 convert 在其 *style* 参数为 NULL 时返回的结果与 convert 不带 *style* 参数时返回的结果相同。例如：

```
select convert(datetime, "01/01/01")
------------

Jan 1 2001 12:00AM

select convert(datetime, "01/01/01", NULL)
------------

Jan 1 2001 12:00AM
```

安全性函数

安全性函数返回有关安全服务和用户定义角色的信息。表 16-15 列出每个安全性函数的名称、使用的参数和返回结果。

表 16-15：安全性函数

函数	参数	结果
audit_event_name	(*event_id*)	返回对审计事件的说明。
is_sec_service_on	(*security_service_nm*)	确定是否启用了某个特定的安全服务。如果已启用该服务则返回 1；否则返回 0。
mut_excl_roles	(“*role_1*” , “*role_2*” [, " membership" |" activation"])	返回有关两个角色之间互斥性级别的信息。
proc_role	(“*role_name*”)	如果调用用户不具有或未激活指定的角色，则返回 0；如果调用用户已激活指定的角色，则返回 1；如果该用户直接或间接具有指定的角色但尚未激活该角色，则返回 2。
role_contain	(["*role1*", "*role2*"])	如果指定的第二个角色包含第一个角色，则返回 1。
role_id	(“*role_name*”)	返回指定角色名的角色 ID。
role_name	(*role_id*)	返回指定角色 ID 的角色名。
show_role	( )	如果有当前登录的活动角色 （sa_role、 sso_role、 oper_role、 replication_role 或 *role_name*），该函数就会返回这些角色。如果登录没有任何角色，将返回 NULL。
show_sec_services	( )	返回为当前会话启用的可用安全服务的列表。

请参见 《系统管理指南：卷 1》中的第 17 章 “管理用户权限”。

用户定义的 SQL 函数

使用 create function 可以创建包含以下项的标量函数：

- declare 语句，用于定义该函数的局部数据变量和游标。
- 赋给此函数的局部对象的值 （例如，使用 select 或 set 命令为表的局部标量和变量赋值）。
- 引用函数中已声明、打开、关闭和释放的局部游标的游标操作。
- 控制流语句。
- set 选项 （仅在函数范围内有效）。

Adaptive Server 不允许在向客户端返回数据的标量函数中使用 fetch 语句。您不能在该函数中包括下列语句：

- 向客户端返回数据的 select 或 fetch 语句。
- insert、 update 或 delete 语句。
- 实用程序命令，如 dbcc、 dump 和 load 命令。
- print 语句
- 引用 rand、 rand2、 getdate 或 newid 的语句。

可以包括仅为局部变量赋值的 select 或 fetch 语句。

请参见 《参考手册：命令》。

第 17 章

使用存储过程

存储过程是 SQL 语句或控制流语言的已命名集合。可以为常用功能创建存储过程以提高性能。 Adaptive Server 也提供执行管理任务和更新系统表的**系统过程**。

也可创建扩展的存储过程并用它从 Adaptive Server 中调用过程语言函数。请参见第 18 章 “使用扩展存储过程”。

存储过程如何工作

运行存储过程时， Adaptive Server 会准备一个计划，使该过程能非常快地执行。存储过程可以：

- 带参数
- 调用其它过程
- 把状态值返回给调用过程或批处理，以指明成功或失败，以及失败的原因
- 把参数的值返回给调用过程或批处理
- 在远程 Adaptive Server 上执行

编写存储过程的能力大大提高了 SQL 的功能、效率和灵活性。编译后的过程显著提高了 SQL 语句的性能与批处理能力。另外，如果您的服务器和远程服务器都设置为允许远程登录，则可以执行其它 Adaptive Server 上的存储过程。可以在本地 Adaptive Server 上编写触发器，当本地发生删除、更新或插入事件时，这些触发器执行远程服务器上的过程。

存储过程与普通的 SQL 语句和批 SQL 语句不同，它是预编译的。第一次运行过程时，Adaptive Server 查询处理器会对它进行分析，准备最终存储到**系统表**中的执行计划。随后，过程将按照存储在系统表中的计划执行。由于大多数查询处理工作已被执行，存储过程的执行几乎是瞬时的。

Adaptive Server 提供多种存储过程，作为方便用户使用的工具。存储在 sybsystemprocs 数据库中的名称以 “sp_” 开头的过程称为**系统过程**，因为它们插入、更新、删除并报告系统表中的数据。

创建并使用存储过程的示例

创建简单的、没有特殊功能 （如带参数）的存储过程的语法是：

```
create procedure procedure_name
    as SQL_statements
```

存储过程是数据库对象，其名字必须遵循标识符的规则。

除 create 语句外，可以包括任意数目和种类的 SQL 语句。请参见第 558 页的 “与存储过程关联的限制”。过程可以象一条列出数据库中所有用户名字的语句那样简单：

```
create procedure namelist
as select name from sysusers
```

若要执行存储过程，请使用关键字 execute 和存储过程的名称；或者，只要存储过程由其自己提交给 Adaptive Server 或者它是批处理中的第一条语句，仅用过程名就可以执行它。可以用下述任意方法执行 namelist：

```
namelist
execute namelist
exec namelist
```

要在远程 Adaptive Server 上执行存储过程，必须给出服务器的名称。远程过程调用的语法是：

```
execute server_name.[database_name].[owner].procedure_name
```

以下示例执行 GATEWAY 服务器上的 pubs2 数据库中的 namelist 过程：

```
execute gateway.pubs2..namelist
gateway.pubs2.dbo.namelist
exec gateway...namelist
```

仅当 pubs2 是缺省数据库时，最后的示例才生效。有关在 Adaptive Server 上设置远程过程调用的信息，请参见 《系统管理指南：卷 1》中的第 15 章 “管理远程服务器”。

仅当存储过程位于您的**缺省数据库**中时，数据库名才是可选的。仅当数据库所有者 （“dbo”）拥有该过程或者是您自己拥有它时，所有者名称才是可选的。您必须拥有执行该过程的**权限**。

一个过程可包含多条语句。

```
create procedure showall as
select count(*) from sysusers
select count(*) from sysobjects
select count(*) from syscolumns
```

执行过程时，每条命令的结果按相关语句在过程中的先后顺序显示。

```
showall
------------

           5

(1 row affected)

------------
          88

(1 row affected)

------------
         349

(1 row affected, return status = 0)
```

create procedure 命令执行成功后，过程名存储在 sysobjects 中，其**源文本**存储在 syscomments 中。

可以用 sp_helptext 显示过程的源文本：

```
sp_helptext showall
# Lines of Text
---------------
              1
```

```
(1 row affected)

text
----------------------------------------
create procedure showall as
select count(*) from sysusers
select count(*) from sysobjects
select count(*) from syscolumns

(1 row affected, return status = 0)
```

在用延迟名称解析创建的过程中，将存储 syscomments 中的文本，而不执行 select * 扩展。在该过程首次成功执行后， Adaptive Server 将执行 select * 扩展，然后用扩展的文本更新该过程的文本。由于 select * 扩展是在更新文本之前执行的，因此最终的文本将包含扩展的 select *，如下例所示：

```
create table t (a int, b int)
---------

set deferred_name_resolution on
---------------

create proc p as select * from t
------------------

sp_helptext p
-------------

 # Lines of Text
---------------

               1

(1 row affected)
text
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
-------------------------
create proc p as select * from t

(1 row affected)

(return status = 0)

exec p
------------
a           b
----------- -----------
```

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

sp_helptext p
-------------

 # Lines of Text
---------------
              1

(1 row affected)
text
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------

 /* Adaptive Server has expanded all '*' elements in the
following statement */

create proc p as select t.a, t.b from t

(1 row affected)

(return status = 0)
```

存储过程和权限

存储过程可用作一种安全机制，因为用户可以被授予执行存储过程的权限，即使该用户不拥有它所引用的表或视图的权限或执行具体命令的权限。请参见 《系统管理指南：卷 1》中的第 17 章 “管理用户权限”。

可保护存储过程的源文本免受未经授权的访问，方法为施加限制，使过程创建者和系统管理员才具有对 syscomments 表中 text 列的 select 权限。这种权限限制要求在**已评估的配置**中运行 Adaptive Server。若要实施这种限制，系统安全员必须用 sp_configure 重新设置 allow select on syscomments.text column 参数。请参见 《系统管理指南：卷 1》中的第 5 章 “设置配置参数”。

保护存储过程源文本免受访问的另一种方法是用 sp_hidetext 隐藏源文本。请参见 《参考手册：过程》。

存储过程和性能

数据库更改时，可以优化用于访问其表的初始查询计划，方法为用 sp_recompile 重新编译它们。这样就不必查找、删除和重新创建每个存储过程及触发器。下面的示例表示访问表 titles 的所有存储过程和触发器在下次执行时都要重新编译。

```
sp_recompile titles
```

请参见 《参考手册：过程》。

创建和执行存储过程

只能在当前数据库中创建过程。

在 Adaptive Server 15.5 之前的版本中，所有被引用对象在创建时必须存在。现在，延迟名称解析功能允许在首次执行存储过程时解析对象，但用户创建的数据类型对象除外。

延迟名称解析这一功能通过一个新的配置参数 deferred name resolution 和一个新的 set 参数 set deferred_name_resolution 来执行；前一个参数在服务级执行，后一个参数在连接级执行。

缺省行为是在执行前解析对象 （旧行为）。必须通过显式指定配置选项 deferred name resolution 或 set 参数来指定新行为。

有关这些参数的信息，请参见 《系统管理指南：卷 1》以及 《参考手册：命令》。

发出 create procedure 的权限在缺省情况下将授予数据库所有者，获得授权的所有者可将此权限移交给其他用户。请参见 《参考手册：命令》。

使用 *deferred_name_resolution*

在此功能有效 （通过使用 set 选项 deferred_name_resolution 设为有效或通过使用配置参数 deferred name resolution 设为全局有效）时，过程内的对象将在执行时而不是在创建时进行解析。利用此选项，您可以创建过程来引用在创建过程时不存在的对象。

例如，使用 deferred_name_resolution 可以创建一个引用不存在的表的过程。以下示例尝试在不使用 deferred_name_resolution 的情况下创建一个过程：

```
select * from non_existing_table
-----------
error message


Msg 208, Level 16, State 1:
Line 1:
non_existing_table not found. Specify owner.objectname
or use sp_help to check whether the object exists
(sp_help may produce lots of output).
```

不过，如果使用此选项，即可创建过程，而不会因缺少对象而引发错误。

```
set deferred_resolution_on
-----------------
create proc p as select * from non_existing_table
-----------------
```

注释　deferred_name_resolution 不在执行时解析用户定义的数据类型。此类数据类型在创建时解析，因此如果解析失败，将无法创建过程。

在创建时解析对象意味着在执行时 （而不是创建时）也会引发对象解析错误。

参数

参数是存储过程的参数。在 create procedure 语句中可选择性地声明一个或多个参数。在执行过程时，用户必须提供在 create procedure 语句中指定的每个参数的值。

参数名前面必须带有 @ 符号且必须符合标识符的规则，请参见第 7 页的 “标识符”。参数名局限于创建它们的过程中；同一参数名可用于其它过程。把含标点符号的任何参数值 （如用数据库名或所有者名限定的对象名）用单引号或双引号引起来。参数名 （包括 @ 符号）最长为 255 个字节。

必须给定参数的系统数据类型 （text、 unitext 或 image 除外）或用户定义的数据类型，和 （如果数据类型需要）长度或精度和标度 （在小括号中）。

以下是 pubs2 数据库的一个存储过程。给出作者的姓和名后，该过程将显示出该作者所撰写的任何书的书名和出版社的名称。

```
create proc au_info @lastname varchar(40),
  @firstname varchar(20) as
select au_lname, au_fname, title, pub_name
from authors, titles, publishers, titleauthor
where au_fname = @firstname
and au_lname = @lastname
and authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
and titles.pub_id = publishers.pub_id
```

现在，执行 au_info：

```
au_info Ringer, Anne

au_lname au_fname title                 pub_name
-------- -------- -------------------- ----------
Ringer   Anne     The Gourmet Microwave Binnet & Hardley
Ringer   Anne     Is Anger the Enemy?   New Age Books
(2 rows affected, return status = 0)
```

以下存储过程查询系统表。指定一个表名作为参数，则该过程将显示该表名、索引名和索引 ID。

```
create proc showind @table varchar(30) as
select table_name = sysobjects.name,
index_name = sysindexes.name, index_id = indid
from sysindexes, sysobjects
where sysobjects.name = @table
and sysobjects.id = sysindexes.id
```

添加列标题 （如 table_name）以增强结果的可读性。以下是执行此存储过程的可接受的语法形式：

```
execute showind titles
exec showind titles
execute showind @table = titles
execute GATEWAY.pubs2.dbo.showind titles
showind titles
```

只要语句是批处理中的唯一语句或第一条语句，最后一种语法形式（没有 exec 或 execute）就可接受。

以下是指定 *titles* 作为参数时，在 pubs2 数据库中执行 showind 的结果：

```
table_name  index_name  index_id
----------  ----------  ----------
titles      titleidind      0
titles      titleind        2
```

```
(2 rows affected, return status = 0)
```

如果以“*@parameter* = *value*”的形式提供参数，则可以以任意顺序提供它们。否则，必须以参数的 create procedure 语句的顺序提供它们。如果以“*@parameter* = *value*”的形式提供了一个值，则以这种方式提供所有后续参数。

此过程将显示 salesdetail 表的 qty 列的数据类型。

```
create procedure showtype @tabname varchar(18),
@colname varchar(18) as
  select syscolumns.name, syscolumns.length,
  systypes.name
  from syscolumns, systypes, sysobjects
  where sysobjects.id = syscolumns.id
  and @tabname = sysobjects.name
  and @colname = syscolumns.name
  and syscolumns.type = systypes.type
```

如果 *@tabname* 和 *@colname* 按名称指定，则执行此过程时可从 create procedure 语句中以不同的顺序给出：

```
exec showtype
@colname = qty , @tabname = salesdetail
```

在任何使用值表达式的存储过程中都可使用 case 表达式。以下示例检查 titles 表中所有书的销售记录：

```
create proc booksales @titleid tid
as
select title, total_sales,
case
when total_sales != null then "Books sold"
when total_sales = null then "Book sales not available"
end
from titles
where @titleid = title_id
```

例如：

```
booksales MC2222
title                                total_sales
-------------------------            -----------
Silicon Valley Gastronomic Treats    2032 Books sold

(1 row affected)
```

缺省参数

可以在 create procedure 语句中为参数指定缺省值。该值可以是任意常量，如果用户没有指定，它将作为该过程的参数。

以下是一个显示所有作者名字的过程，他们所写的书由以参数形式给定的出版者出版。如果没有提供出版者姓名，该过程将显示由 Algodata Infosystems 出版的书的作者。

```
create proc pub_info
  @pubname varchar(40) = "Algodata Infosystems" as
select au_lname, au_fname, pub_name
from authors a, publishers p, titles t, titleauthor ta
where @pubname = p.pub_name
and a.au_id = ta.au_id
and t.title_id = ta.title_id
and t.pub_id = p.pub_id
```

请注意，如果缺省值是含有嵌入空格或标点符号的字符串，则它必须用单引号或双引号引起来。

执行 pub_info 时，可以指定任意出版社的名称作为参数值。如果没有提供任何参数， Adaptive Server 将使用缺省值 Algodata Infosystems。

```
exec pub_info
au_lname        au_fname      pub_name
--------------  ------------  --------------------
Green           Marjorie      Algodata Infosystems
Bennet          Abraham       Algodata Infosystems
O'Leary         Michael       Algodata Infosystems
MacFeather      Stearns       Algodata Infosystems
Straight        Dick          Algodata Infosystems
Carson          Cheryl        Algodata  Infosystems
Dull            Ann           Algodata Infosystems
Hunter          Sheryl        Algodata Infosystems
Locksley        Chastity      Algodata Infosystems

(9 rows affected, return status = 0)
```

此过程 showind2 将 “titles” 指派为 *@table* 参数的缺省值：

```
create proc showind2
@table varchar(30) = titles as
select table_name = sysobjects.name,
    index_name = sysindexes.name, index_id = indid
from sysindexes, sysobjects
where sysobjects.name = @table
and sysobjects.id = sysindexes.id
```

列标题 （如 table_name）阐明了结果显示。以下是由 showind2 显示的 authors 表的内容：

```
showind2 authors

table_name  index_name       index_id
----------- -------------    ---------
authors     auidind                 1
authors     aunmind                 2

(2 rows affected, return status = 0)
```

如果用户不提供值， Adaptive Server 将使用缺省值 *titles*。

```
showind2

table_name  index_name  index_id
----------- ----------- ---------
titles      titleidind          1
titles      titleind            2

(2 rows affected, return status = 0)
```

如果需要参数却没有提供，且 create procedure 语句没有提供缺省值，则 Adaptive Server 将显示一条错误消息，其中列出了过程所需的参数。

在存储过程中使用缺省参数

如果您创建使用参数缺省值的存储过程，但用户发出该存储过程时误拼了参数名，则 Adaptive Server 将使用缺省值执行该存储过程并且不发出错误消息。例如，如果创建以下过程：

```
create procedure test @x int = 1
as select @x
```

它返回以下内容：

```
exec test @x = 2
go
----------------
                2
```

不过，如果您向此存储过程传递了一个不正确的参数，则它会返回不正确的结果集，但并不发出错误消息：

```
exec test @z = 4
go
-----------
          1
(1 row affected)
(return status = 0)
```

NULL 作为缺省参数

可以在 create procedure 语句中将空值声明为个别参数的缺省值：

```
create procedure procedure_name
    @param datatype [ = null ]
    [, @param datatype [ = null ]]...
```

这种情况下，如果用户没有提供参数， Adaptive Server 执行存储过程时不显示错误信息。

过程定义通过检查参数值是否为空，可以指定在用户未提供参数时要采取的操作。以下是一个示例：

```
create procedure showind3
@table varchar(30) = null as
if @table is null
    print "Please give a table name."
else
   select table_name = sysobjects.name,
     index_name = sysindexes.name,
     index_id = indid
   from sysindexes, sysobjects
   where sysobjects.name = @table
   and sysobjects.id = sysindexes.id
```

如果用户未给定参数， Adaptive Server 将把来自过程的信息输出到屏幕上。

有关将缺省值设置为空的其它示例，请用 sp_helptext 检查系统过程的源文本。

缺省参数中的通配符

如果过程使用带有关键字 like 的参数，则缺省值可包括通配符 %、_、[ ] 和 [^]。

例如，如果用户未提供参数，则可以修改 showind 以显示有关系统表的信息，如下所示：

```
create procedure showind4
@table varchar(30) = "sys%" as
select table_name = sysobjects.name,
    index_name = sysindexes.name,
    index_id = indid
from sysindexes, sysobjects
where sysobjects.name like @table
and sysobjects.id = sysindexes.id
```

使用多个参数

以下是 au_info 的变量，在两个参数中，该变量均使用带通配符的缺省值：

```
create proc au_info2
  @lastname varchar(30) = "D%",
  @firstname varchar(18) = "%" as
select au_lname, au_fname, title, pub_name
from authors, titles, publishers, titleauthor
where au_fname like @firstname
and au_lname like @lastname
and authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
and titles.pub_id = publishers.pub_id
```

如果执行不带参数的 au_info2，则将显示所有姓氏以 “D”开头的作者：

```
au_info2

au_lname au_fname title                        pub_name
-------- -------  -------------------------   -------------
Dull     Ann      Secrets of Silicon Valley   Algodata Infosystems
DeFrance Michel   The Gourmet Microwave       Binnet & Hardley

(2 rows affected)
```

如果参数有缺省值，则可在执行时省略这些参数，并从最后一个参数开始执行。不能跳过任何参数，除非为其提供的缺省值是 NULL。

注释　如果以 *@parameter = value* 的形式提供参数值，则可以按任意顺序提供参数值。也可省略已经为其提供缺省值的参数。如果以 *@parameter = value* 的形式提供了一个值，则以这种方式提供所有后续参数。

下面是一个两个参数的缺省值都已定义后省略第二个参数的示例，可以找到姓氏为 “Ringer”的所有作者的书籍及出版社，如下：

```
au_info2 Ringer

au_lname   au_fname    title                    Pub_name
--------   --------    ---------------------    ------------
Ringer     Anne        The Gourmet Microwave    Binnet & Hardley
Ringer     Anne        Is Anger the Enemy?      New Age Books
Ringer     Albert      Is Anger the Enemy?      New Age Books
Ringer     Albert      Life Without Fear        New Age Books

(4 rows affected)
```

如果用户执行存储过程且指定的参数数目超过过程所期望的数目，Adaptive Server 将忽略多余的参数。例如，针对 pubs2 数据库，sp_helplog 显示如下内容：

```
sp_helplog

In database 'pubs2', the log starts on device
'pubs2dat'.
```

如果错误地添加了某些无意义的参数， sp_helplog 的输出相同：

```
sp_helplog one, two, three

In database 'pubs2', the log starts on device
'pubs2dat'.
```

请记住 SQL 是形式自由的语言。没有规定每一行中的单词数量或者必须换行的地方。如果在执行系统过程后执行命令， Adaptive Server 将尝试执行系统过程，然后再执行命令。例如，如果执行：

```
sp_help checkpoint
```

Adaptive Server 将返回 sp_help 的输出，然后运行 checkpoint 命令。如果对过程参数使用分隔标识符，会产生意外结果。

过程组

在 create procedure 和 execute 语句中，过程名后可选的分号和整数可以将同名的过程组成一组，以便能用一条 drop procedure 语句一起删除它们。

同一应用程序中使用的过程经常用此方法分组。例如，可以创建一系列过程，名称分别为 orders;1、 orders;2，依次类推。下面的语句将删除整个组：

```
drop proc orders
```

用向其名称后附加分号和数字的方法将过程分组后，就不能单独删除它们。例如，以下语句是不允许的：

```
drop proc orders;2
```

要在已评估的配置中运行 Adaptive Server，必须禁止过程分组。这可以确保每个存储过程都拥有唯一的对象标识符并可被单个删除。若要禁止过程分组，系统安全员必须重新设置配置参数 allow procedure grouping。请参见 《系统管理指南：卷 1》中的第 5 章 “设置配置参数”。

在 *create procedure* 中使用 *with recompile*

在 create procedure 语句中，可选子句 with recompile 紧邻在 SQL 语句之前。它指示 Adaptive Server 不为该过程保存计划。每次执行过程时都要创建一个新计划。

如果没有 with recompile，则 Adaptive Server 存储其创建的执行计划。通常，此执行计划满足要求。

然而，如果提供给后续执行的数据或参数值发生变化，可能导致 Adaptive Server 创建一个与过程初次执行时创建的计划不同的执行计划。这种情况下， Adaptive Server 需要一个新的执行计划。

当您认为需要一个新计划时，请在 create procedure 语句中使用 with recompile。请参见 《参考手册：命令》。

在 *execute* 中使用 *with recompile*

在 execute 语句中，可选子句 with recompile 在所有参数之后。它指示 Adaptive Server 编译一个用于后续执行的新计划。

执行过程时，如果数据已发生很大更改，或者您以不规则方式提供参数（即，如果您有理由相信与过程存储在一起的计划可能不是此次执行的最优计划），请使用 with recompile。

多次使用 execute procedure *with recompile* 可能对过程高速缓存性能有不利影响。因为每次使用 *with recompile* 时都生成新的计划，所以，如果没有足够的空间用于这些新计划，有用的性能计划可能会在高速缓存中老化。

如果在 create procedure 语句中使用 select *，则过程不会选取任何已添加到表的新列（即使将 with recompile 选项用于 execute）。您必须删除该过程并重新创建它。

过程中的嵌套过程

当一个存储过程或触发器调用另一存储过程或触发器时，就会发生嵌套。嵌套级别在被调用的过程或触发器开始执行时会递增，而在其执行结束时递减。创建高速缓存的语句时，嵌套级别也会增加一级。超过 16 层的嵌套将导致过程失败。当前的嵌套级别存储在 *@@nestlevel* 全局变量中。

可用名称或变量名替换真正的过程名来调用其它过程。例如：

```
create procedure test1 @proc_name varchar(30)
    as exec @proc_name
```

使用存储过程的临时表

可在存储过程中创建和使用临时表，但临时表只在创建它的存储过程的持续期间内存在。过程完成后， Adaptive Server 将自动删除临时表。单个过程能够：

- 创建临时表
- 插入、更新或删除数据
- 在临时表上运行查询
- 调用引用临时表的其它过程

因为要创建引用临时表的过程，临时表必须存在，所以要遵循以下步骤：

1 使用 create table 语句或 select into 语句创建临时表。例如：

```
create table #tempstores
    (stor_id char(4), amount money)
```

注释 使用 set deferred_name_resolution 时无需此步骤；请参见 第 536 页的 “使用 deferred_name_resolution”。

2 创建访问临时表的过程 （而非创建它的过程）。

```
create procedure inv_amounts as
    select stor_id, "Total Due" = sum(amount)
    from #tempstores
    group by stor_id
```

3 删除临时表：

```
drop table #tempstores
```

如果使用 deferred_name_resolution，则无需此步骤。

4　创建一个过程，该过程创建表并调用在第 2 步中创建的过程：

```
create procedure inv_proc as
create table #tempstores
(stor_id char(4), amount money)
```

运行 inv_proc 过程时，它创建该表，但该表只在该过程的执行期间存在。试着将值插入 #tempstores 表或运行 inv_amounts 过程：

```
insert #tempstores
select stor_id, sum(qty*(100-discount)/100*price)
from salesdetail, titles
where salesdetail.title_id = titles.title_id
group by stor_id, salesdetail.title_id
exec inv_amounts
```

因为 #tempstores 表不再存在，所以无法完成此操作。

也可在存储过程中使用 create table tempdb..*tablename*... 创建不带 # 前缀的临时表。过程完成后这些表不消失，所以它们可被独立过程引用。按以上步骤创建这些表。

在存储过程中设置选项

几乎所有 set 命令选项都可以在存储过程中使用。在过程的执行期间 set 选项始终有效，在过程结束时大多数选项都会恢复为先前的设置。只有 dateformat、 datefirst、 language 和 role 选项不还原到其先前的设置。

然而，如果使用要求用户为对象所有者的 set 选项 （如 identity_insert），则不是对象所有者的用户将无法执行存储过程。

查询优化设置

可以使用 set export_options on 导出优化设置，如 set plan optgoal 和 set plan optcriteria。优化设置不是存储过程的本地设置；它们适用于整个用户会话。

注释　缺省情况下，为登录触发器启用 set export_options。

存储过程的参数

存储过程的参数的最大数目为 2048。但是，如果您执行具有大量参数的过程，系统性能可能会下降，因为查询处理引擎必须处理所有参数并且向内存复制参数值和从内存获取参数值。Sybase 建议您首先测试所编写的包含大量参数的所有存储过程，然后在生产环境中实现它们。

表达式、变量和参数的长度

无论页有多大，传递到存储过程的表达式、变量和参数的最大大小为 16384(16K) 字节。它可以是字符，也可以是二进制数据。无须使用 writetext 命令就可以将最大大小的变量、文字插入到文本列。

如果 Adaptive Server 已升级且先前版本使用较小的最大长度 Adaptive Server 的某些早期版本中，存储过程的表达式、变量和参数的最大大小为 255 字节。

因为页的最大大小增加，所以为受更小最大长度限制的 Adaptive Server 早期版本所写的脚本或存储过程现在可能会返回较大的字符串值。

因为值变大，所以 Adaptive Server 可能要截断字符串，或者如果字符串保存在另一变量中或被插入到列或字符串中，字符串可能溢出。

如果要修改现有表的列以增加字符列的长度，则必须修改使用这些列的数据的所有存储过程以反映新的长度。

```
select datalength(replicate("x", 500)),
    datalength("abcdefgh....255 byte long string.." +
    "xxyyzz ... another 255 byte long string")

----------- -----------
255         255
```

创建存储过程后

创建存储过程后，描述该过程的**源文本**存储在 syscomments 系统表的 text 列中。*不要从 syscomments 中删除此信息；*否则将来升级 Adaptive Server 时可能会出现问题。而应使用 sp_hidetext 对 syscomments 中的文本加密 （请参见 《参考手册：过程》以及第 3 页的 “编译对象”）。

执行存储过程

存储过程可以延时执行，也可以远程执行。

延时执行过程

waitfor 命令可以在指定的时间延迟存储过程的执行或者延迟一个指定的时间段。

例如，若要在半小时后执行过程 testproc，请输入：

```
begin
    waitfor delay "0:30:00"
    exec testproc
end
```

发出 waitfor 命令后，在到达指定时间或发生指定事件前，不能使用与 Adaptive Server 的该连接。

远程执行过程

可以在本地 Adaptive Server 上执行远程 Adaptive Server 的过程。两个服务器都正确配置后，只要把服务器名作为标识符的一部分就可以执行远程 Adaptive Server 上的任何过程。例如，若要在服务器 GATEWAY 上执行名为 remoteproc 的过程，请输入：

```
exec gateway.remotedb.dbo.remoteproc
```

请参见 《系统管理指南：卷 1》中的第 15 章 “管理远程服务器”，以了解有关如何为远程执行过程而配置本地和远程 Adaptive Server 的信息可以将批处理或含有远程过程的 execute 语句中的一个或多个值作为参数传递给远程过程。来自远程 Adaptive Server 的结果显示在本地终端上。

过程的返回状态可用于捕获和传输有关过程执行状态的信息消息。有关详细信息，请参见第 551 页的 “返回状态”。

警告！ 如果不启用组件集成服务， Adaptive Server 不会将远程过程调用 (RPC) 视为事务的一部分。因此，如果把 RPC 作为事务的一部分来执行然后回退该事务，则 Adaptive Server 不回退 RPC 导致的任何改变。如果启用了组件集成服务，则通过 set transactional rpc 和 set cis rpc handling 使用事务性 RPC。请参见 《参考手册：命令》。

使用 with *recompile*

Adaptive Server 12.5 版的文档的指出，如果在存储过程的各次执行中操作的数据不是一致的，则应当使用 with recompile 创建存储过程，这样可在每次执行时重新编译存储过程，而不是使用以前执行中的计划。在 Adaptive Server 15.0 版中使用此选项更为重要，而随着在 15.0.2 版及更高版本中引入延迟编译，这一点已变得至关重要。

如果因存储过程的多次同时执行而同时将存储过程的多个副本放入存储过程高速缓存，则使用以前执行中的查询计划的问题可能会加重。如果存储过程的不同执行使用了差别很大的数据集，则会在存储高速缓存中产生存储过程的两个或多个副本，其中每个副本使用的计划也有很大差别。存储过程的后续执行将使用按照最近使用最多的 (MRU) 算法选择出的副本。

此问题可在同一存储过程的不同执行时导致性能的大幅度波动。在 Adaptive Server 12.5 中可能会出现同一问题，但由于过程是使用计划可能相同的幻数进行优化的，因此很少会出现性能的大幅度波动。

故障排除

在解决存储过程的性能问题时，应使用 with recompile 确保在测试期间重新编译所使用的每个存储过程，这样在测试期间就不会使用以前编译的任何计划。

存储过程中的延迟编译

在首次执行存储过程后 （此时传递给变量的值是可用的）， Adaptive Server 将对其进行优化。 15.0.2 以前的 Adaptive Server 版本会先编译存储过程中的所有语句，然后再执行这些语句。这意味在存储过程内创建的局部变量的实际值或临时表的内容在优化期间将不可用。编译后的存储过程 （包括计划在内）会放入存储高速缓存中，这样下次执行时用户就无需再花时间对其进行编译和优化。

Adaptive Server 15.0.2 为存储过程增加了延迟编译功能，这样只有引用局部变量或临时表的存储过程准备好执行时才会对其进行编译。

先出现在存储过程中的语句 （如将值赋予局部变量或创建临时表的语句）将已执行。这意味着将根据已知值和临时表而不是幻数来优化语句。通过使用实际值，优化程序可以为针对给定数据集执行存储过程选择更好的计划。

只要所操作的数据与编译存储过程时所使用的数据类似，就可在存储过程的后续执行中重用同一计划。

由于该计划是专门针用首次执行时使用的值和数据进行优化的，因此对于具有不同值和数据集的存储过程的后续执行来说，它可能不是一项好计划。在 12.5 中这一点是事实，而在 15.0 中随着优化程序有更多的选项可供选择，这一点更是如此并且更为重要。

返回存储过程的信息

存储过程可返回以下类型的信息：

- 返回状态 — 表示存储过程是否成功完成。
- proc role 函数 — 检查过程是否被拥有 sa_role、sso_role 或 ss_oper 特权的用户执行。
- 返回参数 — 将参数值报告给调用者，然后调用方可以使用条件语句来检查返回的值。

返回状态和返回参数可以使存储过程模块化。由多个存储过程使用的一组 SQL 语句可以作为一个过程来创建，该过程将其执行状态或参数值返回给调用过程。例如，许多 Adaptive Server 系统过程执行同一个过程，以此来检验某些参数是否为有效的标识符。

远程过程调用（在远程 Adaptive Server 上运行的存储过程）也返回两种信息。如果执行语句的语法包含服务器、数据库、所有者名字和过程名，则下面的所有示例都可以远程执行。

返回状态

存储过程将报告一个**返回状态**，该返回状态指示存储过程是否成功完成，如果没有，将报告失败原因。当调用过程时，这个值可以保存在变量中，并在以后的 Transact-SQL 语句中使用。 Adaptive Server 定义的用于表示失败的返回状态值的范围是 -1 到 -99 ；您可以定义此范围外的自己的返回状态值。

以下是一个批处理的例子，它使用 execute 语句的形式返回状态：

```
declare @status int
execute @status = byroyalty 50
select @status
```

byroyalty 过程的执行状态存储在变量 *@status* 中。“50”是根据 titleauthor 表中的 royaltyper 列提供的参数。该示例用 select 语句将值输出；以后的示例将在条件子句中使用该返回值。

保留的返回状态值

Adaptive Server 保留 0 来指示返回成功，而用 -1 到 -99 之间的负数指示失败的各种原因。目前，0 和从 -1 到 -14 的数值已用于版本 12 及更高版本，如表 17-1 中所示：

表 17-1：保留的返回状态值

值	含义
0	过程执行时没有发生错误
-1	缺失对象
-2	数据类型错误
-3	进程被选作死锁牺牲品
-4	权限错误
-5	语法错误
-6	杂类用户错误
-7	资源错误，如空间不足
-8	非致命内部问题
-9	达到系统限制
-10	致命内部不一致性
-11	致命内部不一致性
-12	表或索引损坏
-13	数据库损坏
-14	硬件错误

值 -15 至 -99 留作 Adaptive Server 将来使用。

如果在执行时发生了多个错误，将返回绝对值最高的状态。

用户生成的返回值

可通过向 return 语句添加参数的方法，在存储过程中生成自己的返回值。可以使用 0 到 -99 范围外的任意整数。以下示例在书具有有效合同时返回 1，在所有其它情况下返回 2：

```
create proc checkcontract @titleid tid
as
if (select contract from titles where
        title_id = @titleid) = 1
```

```
   return 1
else
   return 2
```

例如：

```
checkcontract MC2222

(return status = 1)
```

下面的存储过程调用 checkcontract，并使用条件子句检查返回状态：

```
create proc get_au_stat @titleid tid
as
declare @retvalue int
execute @retvalue = checkcontract @titleid
if (@retvalue = 1)
   print "Contract is valid."
else
    print "There is not a valid contract."
```

以下是使用具有有效合同的书的 title_id 执行 get_au_stat 时产生的结果：

```
get_au_stat MC2222
Contract is valid
```

检查过程中的角色

如果存储过程执行系统管理任务或与安全有关的任务，则最好确保只有已被授予特定角色的用户才可执行它。过程执行时， proc_role 函数允许检查角色。如果该用户为指定角色则返回 1。角色名为 sa_role、sso_role，和 oper_role。

以下是在存储过程 test_proc 中使用 proc_role 的示例，要求调用方是系统管理员：

```
create proc test_proc
as
if (proc_role("sa_role") = 0)
begin
    print "You do not have the right role."
    return -1
end
else
    print "You have SA role."
    return 0
```

例如：

```
test_proc

You have SA role.
```

返回参数

存储过程可以把信息返回给调用方的另一种方法是*返回参数*。调用方可以使用条件语句检查返回的值。

当 create procedure 语句和 execute 语句都包含带有参数名的 output 选项时，过程将值返回给调用方。调用方可以是 SQL 批处理或者是其它存储过程。返回值可用于批处理或调用过程中的其它语句。如果返回参数用于属于批处理的 execute 语句，则在执行批处理中的后续语句前，将输出带有标题的返回值。

此存储过程执行两个整数的乘法运算 （第三个整数 *@result* 被定义为 output 参数）：

```
create procedure mathtutor
@mult1 int, @mult2 int, @result int output
as
select @result = @mult1 * @mult2
```

要使用 mathtutor 计算乘法问题，必须声明 *@result* 变量并将其包含在 execute 语句中。将 output 关键字添加到 execute 语句，显示返回参数的值。

```
declare @result int
exec mathtutor 5, 6, @result output

(return status = 0)

Return parameters:

-----------
         30
```

如果想通过提供三个整数来猜测答案并执行此过程，则可能无法看到相乘结果。过程中的 select 语句赋值但不输出：

```
mathtutor 5, 6, 32

(return status = 0)
```

output 参数的值必须作为变量而不是常量传递。以下示例声明变量 *@guess* 以存储传递到 mathtutor 的值，以供在 *@result* 中使用。Adaptive Server 输出返回参数：

```
declare @guess int
select @guess = 32
exec mathtutor 5, 6,
@result = @guess output

(1 row affected)
(return status = 0)

Return parameters:

@result
-----------
         30
```

无论返回参数值更改与否，都始终报告该值。注意：

- 上例中， output 参数 *@result 必须*以“*@parameter* = *@variable*”的形式传递。如果它不是传递的最后一个参数，则后续参数必须以“*@parameter* = *value*”的形式传递。
- *@result* 不必已在调用批处理中声明；它是要传递到 mathtutor 的参数的名称。
- 虽然 *@result* 的更改值通过 execute 语句中指派的变量 （此例为 *@guess*）返回给调用方，但它将在其本身的标题 *@result* 下显示。

要在 execute 语句后的条件子句中使用 *@guess* 的初始值，过程调用时必须把它用另一个变量名存储。以下示例通过在存储过程执行期间用 *@store* 持有变量值，并在条件子句中使用 *@guess* 的“新”返回值，说明了上述最后两项：

```
declare @guess int
declare @store int
select @guess = 32
select @store = @guess
execute mathtutor 5, 6,
@result = @guess output
select Your_answer = @store,
Right_answer = @guess
if @guess = @store
    print "Bingo!"
else
    print "Wrong, wrong, wrong!"

(1 row affected)
```

```
(1 row affected)
(return status = 0)

@result
-----------
         30

 Your_answer Right_answer
 ----------- ------------
          32           30

Wrong, wrong, wrong!
```

此存储过程检查并确定新书销售额是否导致作者的版税百分比发生变化（*@pc* 参数定义为 output 参数）：

```
create proc roy_check @title tid, @newsales int,
        @pc int output
as
declare @newtotal int
select @newtotal = (select titles.total_sales +
@newsales
from titles where title_id = @title)
select @pc = royalty from  roysched
   where @newtotal  >= roysched.lorange and
          @newtotal < roysched.hirange
   and roysched.title_id = @title
```

将值指派给变量 *percent* 后，以下 SQL 批处理调用 roy_check。返回参数在执行批处理中的下一条语句之前输出：

```
declare @percent int
select @percent = 10
execute roy_check "BU1032", 1050, @pc = @percent output
select Percent = @percent
go
(1 row affected)
(return status = 0)

@pc
-----------
         12
Percent
-----------
         12

(1 row affected)
```

下面的存储过程调用 roy_check 并在条件子句中使用 *percent* 的返回值：

```
create proc newsales @title tid, @newsales int
as
declare @percent int
declare @stor_pc int
select @percent = (select royalty from roysched, titles
        where roysched.title_id = @title
        and total_sales >= roysched.lorange
        and total_sales < roysched.hirange
        and roysched.title_id = titles.title_id)
select @stor_pc = @percent
execute roy_check @title, @newsales, @pc = @percent
  output
if
  @stor_pc != @percent
begin
  print "Royalty is changed."
  select Percent = @percent
end
else
  print "Royalty is the same."
```

如果使用与早期批处理中相同的参数执行此存储过程，则：

```
execute newsales "BU1032", 1050

Royalty is changed
Percent
-----------
         12

(1 row affected, return status = 0)
```

在上两个调用 roy_check 的示例中，*@pc* 是传递给 roy_check 的参数，而 *@percent* 是含有输出的变量。 newsales 执行 roy_check 时， *@percent* 中的返回值可能会根据传递的其它参数发生变化。要比较 *percent* 的返回值和 *@pc* 的初始值，必须在另一变量中存储初始值。上例将值保存在 *stor_pc* 中。

传递参数值

使用以下格式传递参数中的值：

@parameter = *@variable*

不能传递常量；要 “接收”返回值，必须有变量名。参数可以是除 text、 unitext 或 image 之外的任意 Adaptive Server 数据类型。

注释 如果存储过程要求多个参数，则在 execute 语句中最后传递返回值参数，或以 *@parameter* = *value* 形式传递所有的后续参数。

output 关键字

一个存储过程可返回多个值；每个值都必须在存储过程和调用语句中被定义为 output 变量。 output 关键字可以缩写为 out。

```
exec myproc @a = @myvara out, @b = @myvarb out
```

如果执行过程时指定 output，而在存储过程中没有用 output 定义参数，会出现错误信息。调用一个包含返回值说明而不要求用 output 返回值的过程是允许的。然而，将得不到返回值。存储过程的编写者控制用户可访问的信息，而用户控制他们的变量。

与存储过程关联的限制

以下是一些创建存储过程的附加限制：

- 不能将 create procedure 语句与同一批处理中的其它语句合并。
- create procedure 定义本身可包含任何数目和类型的 SQL 语句，但 use 和以下 create 语句除外：
  - create view
  - create default
  - create rule
  - create trigger
  - create procedure

- 可以在过程内创建其它数据库对象。可以引用本过程中创建的对象，只要它是在引用之前创建的。对象的 create 语句必须位于过程中语句的实际顺序的最前面。
- 在存储过程内，不能在创建某个对象并将其删除之后又以相同的名字创建新的对象。
- Adaptive Server 是在执行而非编译存储过程时创建在该过程中定义的对象的。
- 如果执行一个调用其它过程的过程，被调用过程可以访问第一个过程创建的对象。
- 可以在过程中引用临时表。
- 如果在过程中创建带 # 前缀的临时表，则该临时表只用于该过程，在退出过程后它将消失。用 create table tempdb..*tablename* 创建的临时表不会消失，除非已将它们显式删除。
- 一个存储过程最多有 255 个参数。
- 过程中局部变量和全局变量的最大数仅受可用内存的限制。

限定过程内的名字

在存储过程中，如果其他用户要使用该存储过程，则同 create table 和 dbcc 一起使用的对象名必须用对象所有者的名称来**限定**。在存储过程中同其它语句 （如 select 和 insert）一起使用的对象名无需限定，因为在过程编译时这些名称就已被解析。

例如，如果用户 “mary”（拥有表 marytab）想要其他用户执行使用该表的过程，则当该表与 select 或 insert 一起使用时，应当使用她自己的姓名限定该表的名称。之所以应用此规则是因为，对象名在过程编译时被解析，且存储为数据库 ID 或对象 ID 对。如果在运行时此对不可用，则对象将再次被解析；如果不使用所有者的姓名限定它，则服务器会查找用户 “mary”所拥有的名为 marytab 的表，而不查找执行存储过程的用户所拥有的名为 marytab 的表。如果服务器没有找到对象 ID “marytab”，则它查找数据库所有者所拥有的具有相同名称的对象。

这样，如果没有限定 marytab，且用户 “john”尝试执行该过程，则 Adaptive Server 将查找过程所有者 （在此例中为 “mary”）所拥有的名为 marytab 的表；如果该用户表不存在，则查找数据库所有者所拥有的 marytab 表。例如，如果删除了表 mary.marytab，则过程将引用 dbo.marytab。

- 如果无法用对象所有者的姓名限定用于 create table 的对象名，请使用 “dbo”或 “guest”限定该对象名。
- 如果存储过程由具有 sa_role 特权的用户执行，则该用户应将表名限定为 tempdb.dbo.mytab。
- 如果存储过程由没有 sa_role 特权的用户执行，则该用户应将表名限定为 tempdb.guest.mytab。如果临时数据库中的对象已使用缺省所有者的姓名进行限定，则在没有 sa_role 特权的用户执行存储过程时，如下所示的查询可能不会返回正确对象 ID：

  ```
  select object_id ('tempdb..mytab')
  ```

  若要在没有 sa_role 特权的情况下获取正确对象 ID，请使用 execute 命令：

  ```
  exec("select object_id('tempdb..mytab')")
  ```

重命名存储过程

使用 sp_rename 重命名存储过程。它的语法为：

sp_rename *objname*, *newname*

例如，若要将 showall 重新命名为 countall：

```
sp_rename showall, countall
```

新名必须遵循标识符的规则。只能修改您所拥有的存储过程的名字。数据库所有者可更改任何用户的存储过程的名称。该存储过程必须位于当前数据库中。

重命名过程引用的对象

如果重命名了过程所引用的任何对象，则必须删除该过程，然后重新创建它。所引用表或视图的名字已被修改的存储过程可能在一段时间内工作很好。事实上，它只工作到 Adaptive Server 编译它为止。进行重新编译的原因有多种，并且不会通知用户。

用 sp_depends 获取被过程引用的对象的报告。

使用存储过程作为安全机制

可以把存储过程作为安全机制来使用，以控制对表中信息的访问和执行数据修改的能力。例如，可以拒绝其他用户对您拥有的表使用 select 命令，并创建只允许这些用户查看某几行或某几列的存储过程。也可使用存储过程来限制 update、 delete 或 insert 语句。

拥有存储过程的用户必须同时拥有过程中使用的表或视图。如果没有授予系统管理员对其它用户表的权限，即使是系统管理员也不能创建对这些表执行操作的存储过程。

有关授予或撤消存储过程及其它数据库对象权限的信息，请参见 《系统管理指南：卷 1》中的第 17 章 “管理用户权限”。

删除存储过程

使用 drop procedure 删除存储过程。它的语法为：

```
drop proc[edure] [owner.]procedure_name
    [, [owner.]procedure_name] ...
```

如果其它存储过程调用已被删除的存储过程，则 Adaptive Server 将显示错误信息。然而，如果定义了一个同名的新过程以代替被删除的过程，则引用初始过程的其它过程可以成功调用它。

过程组 （即拥有相同名称，但却具有不同数字后缀的多个过程）可以用单个 drop procedure 语句删除。过程分组后，就不能单独删除组中的过程。

系统过程

系统过程是：

- 从系统表检索信息的快捷方式
- 执行数据库管理和包括更新系统表在内的其它任务的机制

大多数情况下，系统表*仅*通过存储过程更新。通过更改配置变量并发出 reconfigure with override 命令，系统管理员可以直接更新系统表。请参见 《系统管理指南：卷 1》中的第 17 章 “管理用户权限”。

系统过程名以 “sp_”开头。它们由 sybsystemprocs 数据库中的 installmaster 脚本在 Adaptive Server 安装期间创建。

执行系统过程

可从任意数据库运行系统过程。如果系统过程是在从 sybsystemprocs 数据库之外的数据库执行，则对系统表的任何引用都要映射到开始运行过程的数据库。例如，如果 pubs2 的数据库所有者在 pubs2 中运行 sp_adduser，则新用户将添加到 pubs2..sysusers。要在特定数据库上运行系统过程，须用 use 命令打开数据库并执行该过程，或用数据库名限定过程名。

如果系统过程的参数是对象名，且对象名由数据库名或所有者名限定，则整个名称必须用单引号或双引号引起来。

系统过程的权限

因为系统过程位于 sybsystemprocs 数据库中，所以其权限也在那里设置。某些系统过程只能由数据库所有者来运行。这些过程确保执行过程的用户是他们在其中执行该过程的数据库的所有者。

其它系统过程可由任何被授予 execute 权限的用户执行，但此权限必须在 sybsystemprocs 数据库中授予。这种情况有两种后果：

- 用户要么具有在所有数据库中执行某系统过程的权限，要么在任何数据库中都不具有这一权限。
- 用户数据库的所有者不能从自己的数据库内部直接控制该系统过程的权限。

系统过程类型

系统过程可按功能 （如审计、安全管理、数据定义等等）进行分组。以下部分列出了系统过程的类型。请参见 《参考手册：过程》。

用于审计的系统过程

这些系统过程用于：

- 控制审计设置
- 创建并管理持有审计队列的表

该类过程有：

sp_addauditrecord	sp_addaudittable	sp_audit	sp_displayaudit

请参见 Security Administration Guide: Volume 1 （《安全性管理指南，卷 1》）中的第 18 章 “Auditing”（审计）。

用于安全性管理的系统过程

这些系统过程用于：

- 在 Adaptive Server 上添加、删除和报告登录名
- 添加、删除并报告数据库中的用户、组、角色和别名
- 修改口令和缺省数据库
- 添加、删除和报告可以访问当前 Adaptive Server 的远程服务器
- 从远程服务器添加可以访问当前 Adaptive Server 的用户名

该类过程有：

- sp_activeroles
- sp_addalias
- sp_addgroup
- sp_addlogin
- sp_adduser
- sp_changegroup
- sp_displaylogin
- sp_displayroles
- sp_dropalias
- sp_dropgroup
- sp_droplogin
- sp_dropuser
- sp_encryption
- sp_errorlog
- sp_helpgroup
- sp_helprotect

- sp_helpuser
- sp_lmconfig
- sp_locklogin
- sp_logintrigger
- sp_modifylogin
- sp_password
- sp_passwordpolicy
- sp_post_xpload
- sp_role
- sp_spaceusage
- sp_tab_suspectptn
- sp_webservices

Windows 还可使用其它系统过程。该类过程有：

- sp_grantlogin
- sp_loginconfig
- sp_logininfo
- sp_processmail
- sp_revokelogin

用于远程服务器的系统过程

这些系统过程用于：

- 添加、删除和报告可以访问当前 Adaptive Server 的远程服务器
- 从远程服务器添加可以访问当前 Adaptive Server 的用户名

该类过程有：

- sp_addremotelogin
- sp_addserver
- sp_dropremotelogin
- sp_dropserver
- sp_helpremotelogin

- sp_helpserver
- sp_remoteoption
- sp_serveroption

如果安装了组件集成服务，则还可以使用其它系统过程。该类过程有：

- sp_addexternlogin
- sp_addobjectdef
- sp_autoconnect
- sp_defaultloc
- sp_dropexternlogin
- sp_dropobjectdef
- sp_helpexternlogin
- sp_helpobjectdef
- sp_passthru
- sp_remotesql

用于管理数据库的系统过程

这些系统过程用于：

- 改变数据库的所有者
- 显示和修改数据库选项

该类过程有：

- sp_changedbowner
- sp_dboption
- sp_dbremap
- sp_helpdb
- sp_helpmaplogin
- sp_listsuspectobject
- sp_maplogin
- sp_tempdb
- sp_renamedb

用于数据定义和数据库对象的系统过程

这些系统过程用于：

- 绑定和解除绑定规则和缺省值
- 添加、删除和报告主键、外键及公用键
- 添加、删除和报告用户定义的数据类型
- 重新定义数据库对象和用户定义的数据类型
- 重新优化存储过程和触发器
- 将对象绑定到数据高速缓存或将其解除绑定
- 报告数据库对象、用户定义的数据类型以及数据库对象、数据库、索引及表和索引使用空间之间的依赖性
- 获取有关命令语法的帮助

该类过程有：

- sp_addextendedproc
- sp_addtype
- sp_bindcache
- sp_bindefault
- sp_bindrule
- sp_cacheconfig
- sp_cachestrategy
- sp_chgattribute
- sp_checksource
- sp_commonkey
- sp_cursorinfo
- sp_depends
- sp_dropextendedproc
- sp_dropkey
- sp_droptype
- sp_estspace
- sp_foreignkey

- sp_freedll
- sp_help
- sp_helpartition
- sp_helpcache
- sp_helpcomputedcolumn
- sp_helpconstraint
- sp_helpextendedproc
- sp_helpindex
- sp_helpjoins
- sp_helpkey
- sp_helptext
- sp_hidetext
- sp_poolconfig
- sp_primarykey
- sp_procxmode
- sp_recompile
- sp_rename
- sp_spaceused
- sp_unbindcache
- sp_unbindcache_all
- sp_unbindefault
- sp_unbindrule
- sp_version

用于用户定义的消息的系统过程

这些系统过程用于：

- 将用户定义的消息添加到用户数据库的 sysusermessages 表中
- 从 sysusermessages 删除用户定义的消息
- 从 master 数据库的 sysusermessages 或 sysmessages 中检索消息，用于 print 和 raiserror 语句

该类过程有：

- sp_addmessage
- sp_altermessage
- sp_bindmsg
- sp_dropmessage
- sp_getmessage
- sp_unbindmsg

与语言有关的系统过程

这些系统过程用于：

- 管理字符设置
- 添加和删除替代语言

该类过程有：

- sp_addlanguage
- sp_checknames
- sp_helplanguage
- sp_helpsort
- sp_indsuspect
- sp_setlangalias

用于设备管理的系统过程

这些系统过程用于：

- 添加、删除和报告数据库及转储设备
- 添加、删除和扩展数据库段

该类过程有：

- sp_addsegment
- sp_addumpdevice
- sp_device_attr
- sp_diskdefault
- sp_dropdevice
- sp_dropsegment
- sp_extendsegment
- sp_flushstats
- sp_helpdevice
- sp_helplog
- sp_helpsegment
- sp_listener
- sp_ldapadmin
- sp_logdevice
- sp_logiosize
- sp_placeobject
- sp_version

用于备份和恢复的系统过程

这些系统过程用于：

- 创建和管理阈值
- 显示和设置恢复故障隔离模式
- 与 Backup Server ™ 通信

该类过程有：

- sp_addthreshold
- sp_dropthreshold
- sp_forceonline_db
- sp_forceonline_page
- sp_helpthreshold
- sp_listsuspect_page
- sp_listsuspect_db
- sp_modifythreshold
- sp_setsuspect_granularity
- sp_setsuspect_threshold
- sp_thresholdaction
- sp_volchanged

用于配置与调优的系统过程

这些系统过程用于：

- 修改和报告配置参数
- 报告系统使用情况
- 报告性能行为
- 监控 Adaptive Server 活动

该类过程有：

- sp_clearstats
- sp_configure
- sp_countmetadata
- sp_displaylevel
- sp_helpconfig
- sp_monitor
- sp_monitorconfig
- sp_reportstats

- sp_showplan
- sp_sysmon

用于系统管理的系统过程

这些系统过程用于：

- 报告锁和当前运行进程的用户
- 变更表或数据库的锁升级阈值
- 创建和报告资源限制及时间范围
- 创建和报告执行类及引擎组
- 计划 dbccdb 数据库的布局

该类过程有：

- sp_addengine
- sp_addexeclass
- sp_add_resource_limit
- sp_add_time_range
- sp_bindexeclass
- sp_clearpsexe
- sp_dbextend
- sp_dropexeclass
- sp_dropengine
- sp_dropglockpromote
- sp_drop_resource_limit
- sp_drop_time_range
- sp_familylock
- sp_help_resource_limit
- sp_lock
- sp_modify_resource_limit
- sp_modify_time_range
- sp_setpglockpromote

- sp_plan_dbccdb
- sp_setpsexe
- sp_showcontrolinfo
- sp_showexeclass
- sp_showpsexe
- sp_transactions
- sp_unbindexeclass
- sp_ssladmin
- sp_who

用于升级的系统过程

这些系统过程用于：

- 查找保留字
- 重新映射对象
- 检查对象的查询处理模式

该类过程有：

- sp_checkreswords
- sp_downgrade
- sp_remap

Sybase 提供的其它过程

Sybase 提供目录存储过程、系统扩展存储过程 （系统 ESP）和 dbcc 过程。

目录存储过程

目录存储过程是在表形式的系统表中检索信息的系统过程。目录存储过程有：

- sp_column_privileges
- sp_columns

- sp_databases
- sp_datatype_info
- sp_fkeys
- sp_pkeys
- sp_server_info
- sp_special_columns
- sp_sproc_columns
- sp_statistics
- sp_stored_procedures
- sp_table_privileges
- sp_tables

系统扩展存储过程

扩展存储过程 (ESP) 从 Adaptive Server 中调用过程语言函数。系统扩展存储过程是在安装时通过 installmaster 创建的，位于 sybsystemprocs 数据库中并由系统管理员所拥有。它们可在任何数据库中运行且名称以“xp_”开头。系统扩展存储过程有：

- xp_cmdshell
- xp_deletemail
- xp_enumgroups
- xp_findnextmsg
- xp_logevent
- xp_readmail
- xp_sendmai
- xp_startmai
- xp_stopmai

有关创建和使用 ESP 的详细信息，请参见第 18 章 “使用扩展存储过程”。

dbcc 过程

由 *installdbccdb* 创建的 dbcc 过程是存储过程，用来生成有关 dbcc checkstorage 产生的信息的报告。这些过程驻留于 dbccdb 数据库或替代数据库 dbccalt 中。 dbcc 过程有：

- sp_dbcc_alterws
- sp_dbcc_configreport
- sp_dbcc_createws
- sp_dbcc_deletedb
- sp_dbcc_deletehistory
- sp_dbcc_differentialreport
- sp_dbcc_evaluatedb
- sp_dbcc_exclusions
- sp_dbcc_help_fault
- sp_dbcc_faultreport
- sp_dbcc_fullreport
- sp_dbcc_recommendations
- sp_dbcc_runcheck
- sp_dbcc_statisticsreport
- sp_dbcc_summaryreport
- sp_dbcc_updateconfig

获取有关存储过程的信息

多个系统过程在系统表中提供关于存储过程的信息。

在第 561 页的 “系统过程”中对系统过程作了简要论述。请参见 《参考手册：过程》。

用 *sp_help* 获取报告

可以使用 sp_help 来获取关于存储过程的报告。例如，可以获取有关存储过程 byroyalty 的信息，该存储过程是 pubs2 数据库的一部分，如下所示：

```
sp_help byroyalty
```

```
Name       Owner   Object_type        Create_date
---------   ------  ----------------
byroyalty  dbo     stored procedure   Jul 27 2005 4:30PM

(1 row affected)
Parameter_name Type    Length   Prec  Scale  Param_order Mode
-------------- ------  ------   ----  -----  -----------
@percentage    int          4   NULL   NULL            1

(return status = 0)
```

在使用 sybsystemprocs 数据库时执行 sp_help，可以获取有关系统过程的帮助。

用 *sp_helptext* 查看过程的源文本

要显示 create procedure 语句的源文本，请执行 sp_helptext：

```
sp_helptext byroyalty
```

```
# Lines of Text
---------------
              1

(1 row affected)

text
--------------------------------------------------
create procedure byroyalty @percentage int
as
select au_id from titleauthor
where titleauthor.royaltyper = @percentage

(1 row affected, return status = 0)
```

使用 sybsystemprocs 数据库时执行 sp_helptext 可查看系统过程的源文本。

如果存储过程的源文本已使用 sp_hidetext 加密， Adaptive Server 就会显示一条消息，提示用户文本被隐藏。请参见 《参考手册：过程》。

用 *sp_depends* 标识相关对象

sp_depends 列出引用指定对象的所有存储过程或它所依赖的所有过程。

例如，以下命令列出了用户定义的存储过程 byroyalty 引用的所有对象：

```
sp_depends byroyalty

Things the object references in the current database.
object           type       updated      selected
---------------- ---------- ---------    --------
dbo.titleauthor  user table  no          no

(return status = 0)
```

下面的语句使用 sp_depends 列出引用表 titleauthor 的所有对象：

```
sp_depends titleauthor

Things inside the current database that reference the
object.

object          type
--------------  ------------------
dbo.byroyalty   stored procedure
dbo.titleview   view

(return status = 0)

Dependent objects that reference all columns in the
table. Use sp_depends on each column to get more
information. Columns referenced in stored procedures
views, or triggers are not included in this report.
........
(1 row affected)
(return status = 0)
```

如果重命名了过程所引用的任何对象，则必须删除并重新创建该过程。

将 *sp_depends* 与 *deferred_name_resolution* 一起使用

由于使用 deferred_name_resolution 依赖性信息创建的过程是在执行时创建的，因此当 sp_depends 执行用 deferred_name_resolution 创建但尚未执行的过程时，会出现一条消息：

```
sp_depends p
----------
The dependencies of the stored procedure cannot be
determined until the first successful execution.
(return status = 0)
```

在首次成功执行后，将创建依赖性信息并且 sp_depends 的执行会返回预期信息。

例如：

```
 set deferred_name_resolution on
-----------------

create procedure p as
select id from sysobjects
where id =


1sp_depends p
------------
The dependencies of the stored procedure cannot be
determined until the first successful execution.

(return status = 0)

exec p
id

 -----------

           1


(1 row affected)

(return status = 0)


sp_depends p
----------------


The object references in the current database.
```

```
 object                      type                    updated
         selected
 --------------------------- ----------------------
--------------
         ----------------
 dbo.sysobjects              system table            no
         no
(return status = 0)
```

用 *sp_helprotect* 标识权限

sp_helprotect 报告对存储过程 （或其它任何数据库对象）的权限。例如：

```
sp_helprotect byroyalty

grantor    grantee    type  action    object    column  grantable
---------  ---------  ----  --------- -------   -----   ---------
dbo        public     Grant Execute   byroyalty All     FALSE

(return status = 0)
```

第 18 章

使用扩展存储过程

扩展存储过程 (ESP) 提供了一种可以从 Adaptive Server 内部调用外部过程语言函数的机制。用户调用 ESP 所使用的语法与调用存储过程所使用的语法相同。不同之处在于 ESP 执行过程语言代码，而不是 Transact-SQL 语句。

概述

利用扩展存储过程可以从 Adaptive Server 内部动态装载和执行外部过程语言函数。每个 ESP 都与一个相应的函数相关联，当从 Adaptive Server 调用 ESP 时将执行该函数。

Adaptive Server 可以通过 ESP 在 Adaptive Server 外部执行任务，以响应 Adaptive Server 内部发生的事件。例如，可以创建一个用于销售证券的 ESP 函数。调用此 ESP 以响应当证券价格达到某一特定值时引发的触发器。或者，可以创建一个发送电子邮件通知或在网络范围内广播的 ESP 函数，以响应在相关数据库系统内部发生的事件。

对 ESP 来说，“过程语言”是一种编程语言，它能调用 C 语言函数并能使用 C 语言数据类型。

当某个函数作为 ESP 注册到数据库后，可以像调用存储过程一样从 isql、触发器、其它存储过程或客户端应用程序调用它。

ESP 可以：

- 接受输入参数
- 返回一个状态值，指明成功或失败以及失败的原因
- 返回输出参数的值
- 返回结果集

Adaptive Server 提供了一些系统 ESP。例如系统 ESP xp_cmdshell，它可以从 Adaptive Server 内部执行操作系统命令。您也可使用 Open Server 应用程序编程接口 (API) 的子集编写自己的 ESP。

XP Server

扩展存储过程由名为 XP Server 的 Open Server 应用程序实现，该应用程序与 Adaptive Server 运行在同一台计算机上。 Adaptive Server 和 XP Server 通过远程过程调用 (RPC) 进行通信。在独立的进程中运行 ESP 可防止 Adaptive Server 因 ESP 代码错误而失败。使用 ESP 而不使用 RPC 的好处在于，ESP 在 Adaptive Server 中的运行方式与存储过程的运行方式相同，不需要使用 Open Server 即可运行 ESP。

安装 Adaptive Server 时自动安装 XP Server。但是，如果打算开发 XP Server 库，则必须购买一个 Open Server 许可证。Adaptive Server 许可证中提供了运行 XP Server 的 dll 和命令所需的全部内容。

Adaptive Server 必须在 XP Server 运行时才能执行 ESP。 Adaptive Server 在 ESP 首次被调用时启动 XP Server，当 Adaptive Server 退出时关闭 XP Server。

在 Windows 上，如果 start mail session 配置参数设置为 1，XP Server 将在 Adaptive Server 启动时自动启动。

使用 CIS RPC 机制

除了通过节点处理器传送外，还可以使用 CIS RPC 机制执行 XP Server 过程。若要设置两个所需的选项 cis rpc handling 和 negotiated logins，请输入：

```
//to set 'cis rpc handling'//
sp_configure 'cis rpc handling', 1
//or at the session level//
set 'cis rpc handling' on

//to set 'negotiated logins'//
sp_serveroption XPServername, 'negotiated logins', true
```

如果未设置其中任一选项，例如，如果 cis rpc handling 是 on，但 negotiated logins 不是 true，则通过节点处理器传送 ESP，且不显示任何警告消息。

使用 sybesp_dll_version()

Sybase 建议将所有库装载到 XP Server 以实现函数 sybesp_dll_version()。该函数返回 DLL 使用的 Open Server API 版本。输入：

```
CS_INT sybesp_dll_version()
----------
CS_CURRENT_VERSION
```

CS_CURRENT_VERSION 是 Open Server API 中定义的宏。 DLL 可以使用该宏，而无需硬编码一个特定值。如果尚未实现该函数， XP Server 将不会尝试执行版本匹配，并在日志文件中输出错误消息 #11554。（有关错误消息的信息，请参见 《故障排除和错误消息指南》。）但是， XP Server 会继续装载 DLL。

如果存在版本不匹配的情况，XP Server 将在日志文件中输出错误消息 11555，但会继续装载 DLL。

手动启动 XP Server

通常，用户没有必要手动启动 XP Server，因为 Adaptive Server 会在收到第一个 ESP 会话请求时启动 XP Server。但是，如果您创建并调试自己的 ESP，可能会发现有必要使用 xpserver 实用程序从命令行手动启动 XP Server。有关 xpserver 的语法，请参见针对您的平台的《实用程序指南》。

动态链接库支持

将编译包含 ESP 代码的过程函数并将其链接到动态链接库 (DLL)，这些动态链接库将装载到 XP Server 的内存中以响应 ESP 的执行要求。动态链接库将始终保留在内存中，直到出现以下情况：

- XP Server 退出
- 调用了 sp_freedll 系统过程
- 使用 sp_configure 设置了 esp unload dll 配置参数

Open Server API

Adaptive Server 使用 Open Server API，它允许用户运行 Adaptive Server 提供的系统 ESP。用户也可使用 Open Server API 实现自己的 ESP。

表 18-1 列出了 ESP 开发所需的 Open Server 例程。有关这些例程的完整文档，请参见 《Open Server Server-Library/C 参考手册》。

表 18-1：用于 ESP 支持的 Open Server 例程

函数	用途
srv_bind	描述和绑定参数的程序变量
srv_descfmt	描述参数
srv_numparams	根据 ESP 客户端请求返回参数个数
srv_senddone	发送结果完成消息
srv_sendinfo	发送消息
srv_sendstatus	发送状态值
srv_xferdata	发送及接收参数或数据
srv_yield	挂起当前执行的线程并允许执行其它线程

创建和使用 ESP 的示例

当 ESP 函数编写、编译完成并链接到 DLL 后，可以使用 create procedure 命令的 as external name 子句为函数创建一个 ESP：

```
create procedure procedure_name [parameter_list]
    as external name dll_name
```

procedure_name 是 ESP 的名称，它必须与其在 DLL 中的实施函数名相同。 ESP 是数据库对象，其名称必须遵守标识符规则。

dll_name 是 DLL 的名称，实施函数存储在其中。

以下语句创建一个名为 getmsgs 的 ESP，它位于 *msgs.dll* 中。getmsgs ESP 不采用任何参数。本示例用于 Windows Adaptive Server：

```
create procedure getmsgs
as external name "msgs.dll"
```

在 Solaris Adaptive Server 上，语句是：

```
create procedure getmsgs
as external name "msgs.so"
```

这反映了 Solaris 的命名约定。

接下来的语句将创建一个名为 getonemsg 的 ESP，它也位于 *msgs.dll* 中。 getonemsg ESP 将消息号作为单一参数。

```
create procedure getonemsg @msg int
as external name "msgs.dll"
```

表 18-2 中总结了针对特定平台的 DLL 扩展命名约定。

表 18-2：DLL 扩展命名约定

平台	DLL 扩展
HP 9000/800 HP-UX	*.sl*
Sun Solaris	*.so*
Windows	*.dll*

Adaptive Server 创建了一个 ESP 后，将过程名称存储在 sysobjects 系统表中，将对象类型 “XP” 和包含 ESP 函数的 DLL 名称存储在系统表 syscomments 的 text 列中。

执行 ESP 如同执行用户定义的存储过程或系统过程。可使用关键字 execute 和存储过程的名称，或仅给出过程名称，只要它是自己提交到 Adaptive Server 的或是批处理中的第一条语句即可。例如，可以通过下列任一方式执行 getmsgs：

```
getmsgs
execute getmsgs
exec getmsgs
```

可以通过下列任一方式执行 getonemsg：

```
getonemsg 20
getonemsg @msg=20
execute getonemsg 20
execute getonemsg @msg=20
exec getonemsg 20
exec getonemsg @msg=20
```

ESP 和权限

在 ESP 中，可以像在常规存储过程中一样授予和撤消权限。

除了通常的 Adaptive Server 安全性以外，还可以使用 xp_cmdshell context 配置参数将 xp_cmdshell 的执行权限限制到具有系统管理特权的用户。使用该配置参数可以防止普通用户使用 xp_cmdshell 执行他们无权直接从命令行执行的操作系统命令。 xp_cmdshell 配置参数的行为因平台而异。

缺省情况下，用户必须使用 sa_role 来执行 xp_cmdshell。要授权其他用户使用 xp_cmdshell，可使用 grant 命令。可通过 revoke 撤消权限。无论 xp_cmdshell 环境设为 0 还是 1， grant 或 revoke 权限都适用。

ESP 和性能

由于 Adaptive Server 和 XP Server 驻留在同一台计算机上，当 XP Server 执行消耗大量资源的某一函数时，它们可能会影响彼此的性能。

可以使用 sp_configure 设置两个参数 esp execution priority 和 esp unload dll，通过设置 ESP 执行的优先级和释放 XP Server 内存来控制 XP Server 对 Adaptive Server 产生的影响。

设置优先级

使用 esp execution priority 将 XP Server 线程的优先级设置高一些，以便 Open Server 调度程序在运行队列中的其它线程之前运行它，或设置低一些，以便调度程序仅在没有其它线程运行时才运行 XP Server。 esp execution priority 的缺省值为 8，可将其设置为 0 到 15 之间的任意值。有关调度 Open Server 线程的信息，请参见 《Open Server Server-Library/C 参考手册》中关于多线程编程的讨论。

释放内存

当 ESP 装载 DLL 的请求结束后，通过将其从 XP Server 的内存中卸载，可使 XP Server 使用的内存量降到最低。为此，请设置 esp unload dll，以便在 ESP 执行完成后自动卸载 DLL。如果未设置 esp unload dll，可以使用 sp_freedll 显式释放 DLL。

不能卸载支持系统 ESP 的 DLL。

为 ESP 创建函数

对实现 ESP 的函数内容没有限制。唯一的要求是要以过程化编程语言编写，该语言应能够：

- 调用 C 语言函数
- 处理 C 语言数据类型
- 与 Open Server API 相链接

通过使用 Open Client API， ESP 函数可向 Adaptive Server （可以是最初调用该函数的 Adaptive Server，也可以是其它 Adaptive Server）发送请求。

但有一个例外， ESP 函数不应调用 Windows 上的 C 运行时信号例程。这会导致 XP Server 失败，因为 Open Server 不支持 Windows 上的信号处理。

用于 ESP 开发的文件

必须首先购买一个 Open Server 许可证，以便使用 Open Server 库进行开发。ESP 开发所需的头文件在 *$SYBASE/$SYBASE_OCS/include* 中。为了能在源文件中找到这些文件，应在源代码中包含以下内容：

- *ospublic.h*
- *oserror.h*

Open Server 库在 *$SYBASE/$SYBASE_OCS/lib* 中。第 587 页的“ESP 函数示例”中显示的示例程序的源代码在 *$SYBASE/$SYBASE_ASE/sample/esp* 中。

Open Server 数据结构

有三种数据结构对编写 ESP 函数非常有用：

- SRV_PROC
- CS_SERVERMSG
- CS_DATAFMT

SRV_PROC

所有 ESP 函数都被编码为能够接受单个参数及指向 SRV_PROC 结构的指针。 SRV_PROC 结构在函数及其调用进程之间传递信息。 ESP 开发人员无法直接处理此结构。

ESP 函数将 SRV_PROC 指针传递到 Open Server 例程，该例程获得参数类型和数据，并返回输出参数、状态代码和结果集。

CS_SERVERMSG

通过 *srv_sendinfo* 例程， Open Server 使用 CS_SERVERMSG 结构向客户端发送错误消息。有关 CS_SERVERMSG 的信息，请参见《Open Server-Library/C 参考手册》。

CS_DATAFMT

Open Server 使用 CS_DATAFMT 结构来描述数据值和程序变量。

Open Server 返回代码

Open Server 函数返回 CS_RETCODE 类型的代码。 ESP 函数最常见的 CS_RETCODE 值有：

- CS_SUCCEED
- CS_FAIL

简单 ESP 函数概述

考虑到与 Open Server API 的相互作用，一个 ESP 函数应有以下基本结构：

1 获得参数个数。

2 获得输入 / 输出参数值并将其绑定到局部变量。

3 使用输入参数值执行处理并将结果存储到局部变量中。

4 用相应的值初始化所有输出参数，将其与局部变量绑定并传送到客户端。

5 使用 srv_sendinfo() 将返回行发送到客户端。

6　使用 srv_sendstatus() 将状态发送到客户端。

7　使用 srv_senddone() 通知客户端处理已完成。

8　如果有错误情况，可使用 srv_sendinfo() 将错误消息发送到客户端。

有关 Open Server 例程的文档，请参见 《Open Server Server-Library/C 参考手册》。

多线程

由于 Open Server 目前是非抢先式的，同一服务器上运行的所有 ESP 必须互相让步，使用 Open Server srv_yield() 例程将其 XP Server 线程挂起并允许执行其它线程。

有关详细信息，请参见 《Open Server Server-Library/C 参考手册》中的多线程编程章节。

ESP 函数示例

xp_echo.c 具有一个 ESP，它接收用户提供的输入参数并将其回应给 ESP 客户端，它是调用 ESP 的进程。该示例包括 xp_message 函数（该函数发送消息和状态）和 xp_echo 函数 （该函数处理输入参数并执行回应）。可以使用该示例作为建立自己的 ESP 函数的模板。源代码在 *$SYBASE/$SYBASE_ASE/sample/esp* 中。

```
/*
** xp_echo.c
**
**      Description:
**              The following sample program is generic in
**              nature. It echoes an input string which is
**              passed as the first parameter to the xp_echo
**              ESP. This string is retrieved into a buffer
**              and then sent back (echoed) to the ESP client.
*/
#include <string.h>
#include <stdlib.h>
#include <malloc.h>
/* Required Open Server include files.*/
#include <ospublic.h>
#include <oserror.h>
/*
** Constant defining the length of the buffer that receives the
```

```
** input string. All of the Adaptive Server parameters related
** to ESP may not exceed 255 char long.
*/
#define ECHO_BUF_LEN    255
/*
** Function:
**      xp_message
**      Purpose: Sends information, status and completion of the
**      command to the server.
** Input:
**      SRV_PROC *
**      char *  a message string.
** Output:
**      void
*/
void xp_message
(
        SRV_PROC *srvproc,  /* Pointer to Open Server thread
                            control structure */
        char     *message_string /* Input message string */
)
{
        /*
        ** Declare a variable that will contain information
        ** about the message being sent to the SQL client.
        */
        CS_SERVERMSG    *errmsgp;
        /*
        ** A SRV_DONE_MORE instead of a SRV_DONE_FINAL must
        ** complete the result set of an Extended Stored
        ** Procedure.
        */
        srv_senddone(srvproc, SRV_DONE_MORE, 0, 0);
        free(errmsgp);
}
/*
** Function:xp_echo
** Purpose:
**      Given an input string, this string is echoed as an output
**      string to the corresponding SQL (ESP) client.
** Input:
**      SRV_PROC *
** Output
**      SUCCESS or FAILURE
*/
CS_RETCODE xp_echo
(
```

```
        SRV_PROC        *srvproc
)

{
        CS_INT          paramnum;  /* number of parameters */
        CS_CHAR         echo_str_buf[ECHO_BUF_LEN + 1];
                               /* buffer to hold input string */
        CS_RETCODE      result = CS_SUCCEED;
        CS_DATAFMT      paramfmt; /* input/output param format */
        CS_INT          len;      /* Length of input param */
        CS_SMALLINT     outlen;
        /*
        ** Get number of input parameters.*/
        */
        srv_numparams(srvproc, &paramnum);
        /*
        ** Only one parameter is expected.*/
        */
        if (paramnum != 1)
        {
                /*
                ** Send a usage error message.*/
                */
                xp_message(srvproc, "Invalid number of
                parameters");
                result = CS_FAIL;
        }
      else
        {
                /*
                ** Perform initializations.
                */
                outlen = CS_GOODDATA;
                memset(&paramfmt, (CS_INT)0,
                    (CS_INT)sizeof(CS_DATAFMT));
                /*
                ** We are receiving data through an ESP as the
                ** first parameter.So describe this expected
                ** parameter.
                */
                if ((result == CS_SUCCEED) &&
                  srv_descfmt(srvproc, CS_GET
                  SRV_RPCDATA, 1, &paramfmt) != CS_SUCCEED)
                {
                          result = CS_FAIL;
                }
```

```
/*
** Describe and bind the buffer to receive the
** parameter.


*/
if ((result == CS_SUCCEED) &&
  (srv_bind(srvproc, CS_GET, SRV_RPCDATA,
   1, &paramfmt,(CS_BYTE *) echo_str_buf,
   &len, &outlen) != CS_SUCCEED))
{
          result = CS_FAIL;
}
/* Receive the expected data.*/
if ((result == CS_SUCCEED) &&
  srv_xferdata(srvproc,CS_GET,SRV_RPCDATA)
  != CS_SUCCEED)
{
          result = CS_FAIL;
}               /*
** Now we have the input info and are ready to
** send the output info.
*/
if (result == CS_SUCCEED)
{
        /*
        ** Perform initialization.
        */
        if (len == 0)
                outlen = CS_NULLDATA;
        else
                outlen = CS_GOODDATA;


        memset(&paramfmt, (CS_INT)0,
          (CS_INT)sizeof(CS_DATAFMT));
        strcpy(paramfmt.name, "xp_echo");
        paramfmt.namelen = CS_NULLTERM;
        paramfmt.datatype = CS_CHAR_TYPE;
        paramfmt.format = CS_FMT_NULLTERM;
        paramfmt.maxlength = ECHO_BUF_LEN;
        paramfmt.locale = (CS_LOCALE *) NULL;
        paramfmt.status |= CS_CANBENULL;
        /*
        ** Describe the data being sent.
        */
        if ((result == CS_SUCCEED) &&
                srv_descfmt(srvproc, CS_SET,
```

```
                                SRV_ROWDATA, 1, &paramfmt)
                                != CS_SUCCEED)
                        {
                                result = CS_FAIL;
                        }
                        /*
                        ** Describe and bind the buffer that
                        ** contains the data to be sent.
                        */
                        if ((result == CS_SUCCEED) &&
                                (srv_bind(srvproc, CS_SET,
                                SRV_ROWDATA, 1,
                                &paramfmt, (CS_BYTE *)
                                echo_str_buf, &len, &outlen)
                                != CS_SUCCEED))
                        {
                                result = CS_FAIL;
                        }
                        /*
                        ** Send the actual data.
                        */
                        if ((result == CS_SUCCEED) &&
                                srv_xferdata(srvproc, CS_SET,
                                SRV_ROWDATA)!= CS_SUCCEED)
                        {
                                result = CS_FAIL;
                        }
                }
                /*
                ** Indicate to the ESP client how the
                ** transaction was performed.
                */
                if (result == CS_FAIL)
                        srv_sendstatus(srvproc, 1);
                else
                        srv_sendstatus(srvproc, 0);
                /*
                ** Send a count of the number of rows sent to
                ** the client.
                */
                srv_senddone(srvproc,(SRV_DONE_COUNT |
                  SRV_DONE_MORE), 0, 1);
        }
        return result;
}
```

建立 DLL

可以在服务器平台上使用能生成所需 DLL 的任何编译器。

有关使用 Open Server API 编译和链接函数的一般信息，请参见 Open Client/Server Supplement （《Open Client/Server 补充说明》）。

DLL 的搜索顺序

Windows 以下列顺序搜索 DLL：

1 调用应用程序的目录

2 当前目录

3 系统目录 (SYSTEM32)

4 PATH 环境变量中所列目录

UNIX 按照在 LD_LIBRARY_PATH 环境变量 （在 Solaris 上）、SHLIB_PATH （在 HP 上）或 LIBPATH （在 AIX 中）中所列的目录顺序搜索库。

如果 XP Server 在搜索路径中未能找到 ESP 函数库，它将尝试从 Windows 的 *$SYBASE/DLL* 或其它平台的 *$SYBASE/lib* 中装载它。

不支持 DLL 的绝对路径名。

样本 makefile (UNIX)

以下 makefile *make.unix* 用来为 UNIX 平台上的 xp_echo 程序创建动态链接共享库。它将生成一个文件，在 Solaris 上名为 *examples.so*，在 HP 上名为 *examples.sl*。源代码在 *$SYBASE/$SYBASE_ASE/sample/esp* 中，您可对其进行修改以供自己使用。

若要使用此 makefile 建立示例库，请输入：

```
make -f make.unix
```

```
#
# This makefile creates a shared library. It needs the open
# server header
# files usually installed in $SYBASE/include directory.
# This make file can be used for generating the template ESPs.
# It references the following macros:
#
# PROGRAM is the name of the shared library you may want to create.PROGRAM
= example.so
BINARY        = $(PROGRAM)
```

```
EXAMPLEDLL    = $(PROGRAM)

# Include path where ospublic.h etc reside.You may have them in
# the standard places like /usr/lib etc.

INCLUDEPATH    = $(SYBASE)/include

# Place where the shared library will be generated.
DLLDIR        = .

RM        = /usr/bin/rm
ECHO        = echo
MODE        = normal

# Directory where the source code is kept.
SRCDIR        = .

# Where the objects will be generated.
OBJECTDIR    = .

OBJS        = xp_echo.o

CFLAGS        = -I$(INCLUDEPATH)
LDFLAGS        =  $(GLDFLAGS) -Bdynamic

DLLLDFLAGS    =  -dy -G
#====================================================================

$(EXAMPLEDLL) : $(OBJS)
    -@$(RM) -f $(DLLDIR)/$(EXAMPLEDLL)
    -@$(ECHO) "Loading $(EXAMPLEDLL)"
    -@$(ECHO) " "
    -@$(ECHO) "    MODE:           $(MODE)"
    -@$(ECHO) "    OBJS:           $(OBJS)"
    -@$(ECHO) "    DEBUGOBJS:      $(DEBUGOBJS)"
    -@$(ECHO) " "

    cd $(OBJECTDIR); \
    ld -o $(DLLDIR)/$(EXAMPLEDLL) $(DEBUGOBJS) $(DLLLDFLAGS) $(OBJS)
    -@$(ECHO) "$(EXAMPLEDLL) done"
    exit 0

#====================================================================

$(OBJS) : $(SRCDIR)/xp_echo.c
    cd $(SRCDIR); \
    $(CC) $(CFLAGS) -o $(OBJECTDIR)/$(OBJS) -c xp_echo.c
```

样本定义文件

以下文件 *xp_echo.def* 必须与 *xp_echo.mak* 位于相同的目录下。它列出了在 EXPORTS 部分中将被用作 ESP 函数的所有函数。

```
LIBRARY   examples

CODE      PRELOAD MOVEABLE DISCARDABLE
DATA      PRELOAD SINGLE

EXPORTS
        xp_echo  .   1
```

注册 ESP

创建了一个 ESP 函数并将其链接到 DLL 后，可在数据库中将其作为 ESP 注册。这使得用户能够将该函数作为 ESP 来执行。

可使用下列任一方法注册 ESP：

- Transact-SQL create procedure 命令
- sp_addextendedproc

使用 *create procedure*

创建 ESP 的 create procedure 语法与建议的 ANSI SQL3 语法一致：

```
create procedure [owner.]procedure_name
    [[(]@parameter_name datatype [= default] [output]
    [, @parameter_name datatype [= default]
    [output]]...[)]][with recompile]
    as external name dll_name
```

procedure_name 是 ESP 的名称，与其在数据库中的名称一样。该名称必须与它支持的函数名相同。

参数声明同存储过程的声明一样，有关说明详见第 17 章 “使用存储过程”。

Adaptive Server 将忽略 with recompile 子句 （如果它包括在用于创建 ESP 的 create procedure 命令中）。

dll_name 是包含 ESP 支持函数的库名。

dll_name 可以被指定为一个无扩展名的名称（例如 *msgs*）或名称中带有针对特定平台的扩展名，如在 Windows 上为 *msgs.dll*，在 Solaris 上为 *msgs.so*。

在任一情况下，针对具体平台的扩展名被假定为是库的实际文件名的一部分。

由于 create procedure 在特定数据库中注册 ESP，因此应在调用命令前指定在其中注册 ESP 的数据库。如果尚未运行目标数据库，可通过 isql 利用 use *database* 命令指定数据库。

下列语句注册一个第 587 页的 “ESP 函数示例”中的示例所描述的 xp_echo 例程支持的 ESP，并假定该函数在一个名为 *examples.dll* 的 DLL 中编译。该 ESP 在 pubs2 数据库中注册。

```
use pubs2
create procedure xp_echo @in varchar(255)
as external name "examples.dll"
```

请参见 《参考手册：命令》。

使用 *sp_addextendedproc*

sp_addextendedproc 与 Microsoft SQL Server 使用的语法一致。可使用它替代 create procedure。语法为：

sp_addextendedproc *esp_name*, *dll_name*

esp_name 是 ESP 的名称。该名称必须与支持 ESP 的函数名匹配。

dll_name 是包含 ESP 支持函数的 DLL 名称。

sp_addextendedproc 必须由具有 sa_role 权限的用户在 master 数据库中执行。因此， sp_addextendedproc 始终在 master 数据库中注册 ESP，而不像 create procedure 那样在当前数据库中注册 ESP。与 create procedure 不同， sp_addextendedproc 不允许在 Adaptive Server 中检查参数或给定参数缺省值。

下列语句在 master 数据库中注册一个第 587 页的 “ESP 函数示例”中的示例所描述的 xp_echo 例程支持的 ESP，并假定该函数在一个名为 *examples.dll* 的 DLL 中编译：

```
use master
sp_addextendedproc "xp_echo", "examples.dll"
```

请参见 《参考手册：过程》。

删除 ESP

可以使用 drop procedure 或 sp_dropextendedproc 从数据库中删除 ESP。

drop procedure 的语法与存储过程语法相同：

drop procedure [*owner*.]*procedure_name*

下列命令将删除 xp_echo：

```
drop procedure xp_echo
```

sp_dropextendedproc 的语法是：

sp_dropextendedproc *esp_name*

例如：

```
sp_dropextendedproc xp_echo
```

这两种方法都通过删除系统表 sysobjects 和 syscomments 中对 ESP 的引用而将 ESP 从数据库中删除。它们不会影响基础 DLL。

重命名 ESP

因为 ESP 名称绑定到了其函数的名称，所以不能像重命名存储过程那样使用 sp_rename 重命名 ESP。更改 ESP 的名称：

1 利用 drop procedure 或 sp_dropextendedproc 删除 ESP。

2 重命名并重新编译支持函数。

3 使用 create procedure 或 sp_addextendedproc 以新名称重新创建 ESP。

执行 ESP

使用与执行常规存储过程相同的 execute 命令来执行 ESP。有关执行存储过程的语法和其它信息，请参见第 536 页的 “创建和执行存储过程”。

也可以远程执行 ESP。有关执行远程存储过程的信息，请参见第 549 页的 “远程执行过程”。

因为执行任何 ESP 都涉及 Adaptive Server 和 XP Server 之间的远程过程调用，所以在同一个 execute 命令中不能组合按值传递的参数和按名称传递的参数。所有参数必须全部按名称传递，或全部按值传递。这是执行扩展存储过程和常规存储过程的唯一区别之处。

ESP 可返回：

- 一个状态值，指明成功或失败以及失败的原因
- 输出参数的值
- 结果集

ESP 函数使用 srv_sendstatus Open Server 例程报告返回状态值。从 srv_sendstatus 返回的状态值因应用程序而异。但是零状态表明请求正常完成。

如果对于扩展存储过程没有参数声明列表， Adaptive Server 将忽略所有提供的参数，但不发出任何错误消息。如果您在执行 ESP 时提供的参数多于在声明列表中声明的参数，则 Adaptive Server 总会调用它们。为了避免在调用哪些参数上发生混淆，请检查声明列表上的参数是否与运行时提供的参数匹配。您还应在建立 Open Server 函数时检查指定的 ESP 中参数的个数。

ESP 函数使用 srv_descfmt、 srv_bind 和 srv_xferdata Open Server 例程返回输出参数的值和结果集。有关从 ESP 函数传递值的详细信息，请参见第 587 页的“ESP 函数示例”和 Open Server Server-Library/C Reference Manual （《Open Server Server-Library/C 参考手册》）。在 Adaptive Server 端，从 ESP 返回的值作为常规存储过程来处理。

系统 ESP

除 xp_cmdshell 外，还有多个支持 Windows 功能的系统 ESP，如 Adaptive Server 与 Windows 事件日志或邮件系统的集成。

所有系统 ESP 的名称都以 “xp_”开头。在 Adaptive Server 安装过程中，在 sybsystemprocs 数据库中创建了它们。由于系统 ESP 位于 sybsystemprocs 数据库中，其权限也在该处设置。但您可从任意数据库中运行系统 ESP。系统 ESP 有：

- xp_cmdshell
- xp_deletemail
- xp_enumgroups
- xp_findnextmsg
- xp_logevent
- xp_readmail
- xp_sendmail
- xp_startmail
- xp_stopmail

请参见 《参考手册：过程》中的第 3 章 “系统扩展存储过程”来了解有关系统 ESP 的信息，并且 Configuration Guide for Windows（《Windows 配置指南》）详细讨论了一些特定功能，如 MAPI 和事件日志集成 （使用针对 Windows 的系统 ESP 来实现）。

获取有关 ESP 的信息

使用 sp_helpextendedproc 可获取有关在当前数据库中注册的 ESP 的信息。

在没有参数的情况下， sp_helpextendedproc 显示数据库中的所有 ESP 及包含其相关函数的 DLL 名称。使用 ESP 名称作为参数，它仅为指定的 ESP 提供该信息。

```
sp_helpextendedproc getmsgs

ESP Name   DLL
--------   ---------
getmsgs    msgs.dll
```

由于系统 ESP 在 sybsystemprocs 数据库中，因此必须使用 sybsystemprocs 数据库才能显示它们的名称和 DLL：

```
use sybsystemprocs
sp_helpextendedproc

ESP Name      DLL
--------      ---------
xp_freedll    sybyesp
xp_cmdshell   sybyesp
```

如果某个 ESP 的源文本已使用 sp_hidetext 进行了加密，则 Adaptive Server 会显示一条通知您文本已隐藏的消息。请参见 《参考手册：过程》。

ESP 例外和消息

Adaptive Server 处理来自 XP Server 的所有消息和例外。除了将标准 ESP 消息发送给客户端外， Adaptive Server 还在日志文件中记录它们。不对来自用户定义 ESP 的用户定义消息进行记录，但会将其发送到客户端。

ESP 相关消息可由 XP Server、创建或处理 ESP 的系统过程或系统 ESP 生成。有关 ESP 相关消息的列表，请参见 《故障排除和错误消息指南》。

用户定义的 ESP 的函数可使用 srv_sendinfo Open Server 例程生成消息。请参见第 587 页的 “ESP 函数示例”中的示例 xp_message 函数。

手动启动 XP Server

通常，用户没有必要手动启动 XP Server，因为 Adaptive Server 会在收到第一个 ESP 会话请求时启动 XP Server。但是，如果您创建并调试自己的 ESP，可能会发现有必要使用 xpserver 实用程序从命令行手动启动 XP Server。有关 xpserver 的语法，请参见针对您的平台的《实用程序指南》。

第 19 章

游标：访问数据

游标一次访问 SQL select 语句结果中的一行或多行。使用游标可以修改或删除单独的行或一组行。

Adaptive Server 15.0 版显著更改了游标功能，引入了可滚动的只读游标。使用可滚动游标可以选择一行或多行，以及在行中向后和向前滚动。可滚动游标始终为只读游标。

使用不可滚动游标，无法返回到已选择的行，并且无法一次移动多行。

Sybase 继续支持缺省的仅向前游标，但建议在无需通过游标更新结果集时使用更方便灵活的可滚动游标。

有关支持游标的全局变量、命令和函数的详细信息，请参见 《参考手册：构件块》和 《参考手册：命令》。

使用游标选择行

游标与 select 语句相关，包括：

- **游标结果集** — 执行与游标关联的查询后返回的限定行的集合（表）。
- **游标位置** — 指向游标结果集中的行的指针。如果未将游标指定为只读，可以使用 update 或 delete 语句显式修改或删除该行。

下面两个属性指定游标的行为：

- 敏感性
- 可滚动性

游标有如下四种类型：

- 客户端游标
- 执行游标
- 服务器游标
- 语言游标

敏感性和可滚动性

声明游标时，可以使用以下两个关键字来指定敏感性：

- insensitive
- semi_sensitive

如果声明一个游标为 insensitive，则在打开游标时，该游标只按结果集原样来显示结果集；基础表中的数据更改不可见。如果声明游标为 semi_sensitive，则自打开游标后对基表所做的某些更改可能显示在结果集中。半敏感游标能否看到数据更改是不确定的。

缺省值为 semi_sensitive。

还有以下两个用于指定可滚动性的关键字：

- scroll
- no scroll

如果将游标指定为可滚动游标 （换句话说，使用 scroll 声明它），则可以按顺序或不按顺序获取结果行，并且可以重复扫描结果集。如果游标声明中出现选项 no scroll，则游标不可滚动；结果集在仅向前方向上显示，每次一行。

如果两个属性均未指定，则缺省值为 no scroll。

可将游标看作是 select 语句结果集上的 “句柄”。可以按顺序或不按顺序获取游标，具体取决于游标的可滚动性。

只能向前获取不可滚动游标，不能返回到已获取的行。可以向后或向前获取可滚动游标。

使用可滚动游标可以在游标打开的情况下，通过在 fetch 语句中指定选项 first、 last、 absolute、 next、 prior 或 relative，将游标位置设置到游标结果集中的任何位置。

若要获取结果集中的最后一行，请输入：

```
fetch last [from] <cursor_name>
```

或者，若要选择结果集中的特定行 （在此示例中为第 500 行），请输入：

所有可滚动游标都是只读的。所有可更新的游标都是不可滚动的。

游标类型

游标有如下四种类型：

- **客户端游标** — 通过 Open Client™ 调用 （或嵌入式 SQL）来声明。 Open Client 跟踪从 Adaptive Server 返回的行，并为应用程序缓冲它们。只能通过 Open Client 调用来更新和删除客户端游标结果集。客户端游标是最常用的游标类型。
- **执行游标** — 客户端游标的子集，其结果集由存储过程定义。存储过程可使用参数，通过 Open Client 调用来发送参数值。
- **服务器游标** — 在 SQL 中声明。如果在存储过程中使用服务器游标，执行存储过程的客户端不会注意到它们。执行 fetch 返回给客户端的结果与执行常规 select 返回的结果相同。
- **语言游标** — 不使用 Open Client 而在 SQL 中声明。与服务器游标相同，客户端不会注意到这类游标，返回给客户端的结果与常规 select 返回的结果格式相同。

为简化游标讨论，本手册中只给出了语言游标和服务器游标的示例。有关客户端游标或执行游标的示例，请参见 Open Client 或 Embedded SQL 文档。

游标范围

游标是否存在取决于其**范围**，即使用游标的环境：

- 用户会话中
- 存储过程中
- 触发器中

在用户会话内，游标只在用户结束会话之前存在。用户注销后，Adaptive Server 会释放在该会话中创建的游标。对其他用户启动的其它会话而言，此游标并不存在

如果 declare cursor 语句是某个存储过程或触发器的一部分，则在其中创建的游标将应用于该范围以及启动该存储过程或触发器的范围。但是，所有嵌套的存储过程或触发器将不能访问触发器内的 inserted 或 deleted 表上声明的游标。此类游标只可在声明它们的触发器范围内访问。一旦该存储过程或触发器完成， Adaptive Server 就释放在其中创建的游标。

游标名在给定范围内必须是唯一的。 Adaptive Server 只在运行时才检测特定范围内的名称冲突。如果只执行一个游标，存储过程或触发器就可以定义两个同名的游标。例如，以下存储过程有效，因为在其范围内只定义了一个 names_crsr 游标：

```
create procedure proc2 @flag int
as
if @flag > 0
    declare names_crsr cursor
    for select au_fname from authors
else
    declare names_crsr cursor
    for select au_lname from authors
return
```

游标扫描与游标结果集

Adaptive Server 使用何种方法创建游标结果集取决于游标和游标 select 语句的查询计划。如果工作表不是必需的，则 Adaptive Server 通过使用表的索引键在基表内定位游标来执行 fetch。此执行过程类似于 select 语句，不同的是它返回 fetch 所指定的行数。在执行 fetch 后， Adaptive Server 将游标定位到下一个有效索引键上，直到再次获取或关闭游标。

所有可滚动游标和 insensitive 不可滚动游标都需要工作表来保存游标结果集。有些查询还要求工作表生成游标结果集。若要检验某一特定游标是否使用工作表，请检查 set showplan, no exec on 语句的输出。

使用工作表时，用游标的 fetch 语句检索的行不一定反映实际基表行中的值。例如，用 order by 子句声明的游标通常要求创建一个工作表来为游标结果集的行排序。 Adaptive Server 不锁定与工作表中的行相对应的基表中的行，这样其它客户端就可更新这些基表行。从游标语句返回客户端的行与基表行不同。有关锁与游标如何协同工作的详细信息，请参见第 629 页的 “游标和锁定”。

一般来讲，对于缺省的和 semi_sensitive 游标，当通过该游标的 fetch 返回行时，就会生成游标结果集。这意味着处理游标 select 查询与处理常规 select 查询的方式相同。这一过程称为**游标扫描**，它不仅加快了周转时间，还无需读取应用程序不需要的行。

Adaptive Server 要求游标扫描使用表的唯一索引，特别是对于隔离级别为 0 的读取。如果表有 IDENTITY 列并且必须在其上创建非唯一索引，可使用 identity in nonunique index 数据库选项在表的索引键中包含一个 IDENTITY 列，以使表上创建的所有索引都是唯一索引。该选项使那些在逻辑上不唯一的索引在内部都唯一，并且允许用这些索引来处理隔离级别为 0 的读取的可更新游标。

如果游标所引用的无索引表都没有被另一个使当前行位置发生移动的进程所更新，就仍然可以使用这些游标。例如：

```
declare storinfo_crsr cursor
for select stor_id, stor_name, payterms
    from stores
    where state = "CA"
```

上述游标指定的表 stores 不含任何索引。只要尚未在 declare cursor 语句中指定 for update， Adaptive Server 就允许在无唯一索引的表上声明游标。如果 update 不更改行的位置，游标的位置就不会更改，除非执行下一个 fetch。

使游标变为可更新

如果游标是可更新的，就可以更新或删除该游标所返回的行。如果游标是只读的，则不能更新或删除它。缺省情况下， Adaptive Server 在指定游标为只读前会试图确定该游标是否可更新。

可以通过在 declare 语句中使用关键字 read only 或 update 来显式指定一个游标是否只读。将游标指定为只读可以确保 Adaptive Server 正确执行定位型更新。请确保被更新的表具有唯一索引。否则，Adaptive Server 会拒绝执行 declare cursor 语句。

所有可滚动游标和所有 insensitive 游标都是只读的。

下例为 pubs_crsr 游标定义一个可更新结果集：

```
declare pubs_crsr cursor
for select pub_name, city, state
from publishers
for update of city, state
```

上例包括 publishers 表的所有行，但它只显式将 city 和 state 列定义为可更新。

除非您计划通过游标更新或删除行，否则应将游标声明为只读。如果没有显式指定 read only 或 update，则当 select 语句*没有*包含以下任何结构时， semi_sensitive 不可滚动游标为隐式可更新：

- distinct 选项
- group by 子句
- Aggregate 函数
- Subquery
- union 运算符
- at isolation read uncommitted 子句

如果游标的 select 语句包含上述结构之一，则不能指定 for update 子句。如果将包括 order by 子句的某些类型的游标声明为其 select 语句的一部分， Adaptive Server 也会将游标定义为只读。有关 Adaptive Server 所支持的游标类型的信息，请参见第 603 页的 “游标类型”。

确定哪些列可被更新

可滚动游标和 insensitive 不可滚动游标是只读的。如果没有使用 for update 子句指定 *column_name_list*，则查询中的所有指定列都可更新。 Adaptive Serve 在扫描基表时尝试对可更新游标使用唯一索引。对于游标， Adaptive Server 将包含 IDENTITY 列的索引看作唯一索引，即使该游标并未如此声明。

使用 Adaptive Server 可以更新 *column_name_list* 中的列 （在游标的 select 语句的列列表中没有指定这些列），但这些列应是 select 语句中指定的表的一部分。但如果使用 for update 指定了 *column_name_list*，则只能更新该列表中的列。

在下例中，Adaptive Server 在 publishers 的 pub_id 列上使用唯一索引（即使 newpubs_crsr 的定义中不包括 pub_id）：

```
declare newpubs_crsr cursor
for select pub_name, city, state
from publishers
for update
```

如果没有指定 for update 子句， Adaptive Server 将选择任何唯一索引，尽管在指定表列没有唯一索引的情况下，它也可以使用其它索引或表扫描。但如果指定了 for update 子句， Adaptive Server 就必须使用为一列或多列定义的唯一索引才能扫描基表。如果不存在唯一索引， Adaptive Server 将返回一条错误消息。

多数情况下，只在 for update 子句的 *column_name_list* 中包括要更新的列。如果声明游标时使用了 for update 子句，而表只有一个唯一索引，则不能在 for update *column_name_list* 中包括它的列，因为 Adaptive Server 在游标扫描期间要用到它。如果表具有多个唯一索引，则可以在 for update *column_name_list* 中包括索引列，这样，Adaptive Server 就可以使用另一个可能不在 *column_name_list* 中的唯一索引来执行游标扫描。例如，下面 declare cursor 语句中使用的表在列 c3 上只有一个唯一索引，因此不应在 for update 列表中包括该列：

```
declare mycursor cursor
for select c1, c2, 3
from mytable
for update of c1, c2
```

但是，如果 mytable 具有多个唯一索引 （例如在列 c3 和 c4 上），就必须在 for update 子句中指定一个唯一索引，如下所示：

```
declare mycursor cursor
for select c1, c2, 3
from mytable
```

```
for update of c1, c2, c3
```

不能在 *column_name_list* 中同时包括 c3 和 c4。一般来讲，Adaptive Server 至少需要一个不在列表上的唯一索引键才能执行游标扫描。

允许 Adaptive Server 以这种方式在游标扫描中使用唯一索引，有助于防止一种称为 **Halloween 问题**的更新异常。如果客户端通过游标更新列，且该列 （即具有唯一索引的列）定义了行从基表返回的顺序，就会发生 Halloween 问题。例如，如果 Adaptive Server 使用索引访问一个基表，且该索引键被客户端更新，则更新的索引行可以在索引内移动，并且可被游标再次读取。该行看上去好像在结果集中出现了两次：一次是在索引键被客户端更新时，另一次是在更新的索引行在结果集中进一步下移时。

另一种避免 Halloween 问题的方法是：使用设置为 on 的 unique auto_identity index 数据库选项来创建表。请参见 Performance and Tuning Series: Query Processing and Abstract Plans （《性能和调优系列：查询处理和抽象计划》）中的第 8 章 “Optimization for Cursors”（游标优化）。

Adaptive Server 如何处理游标

使用游标访问数据时， Adaptive Server 将整个过程分为几个步骤：

- **声明游标** 一旦声明游标， Adaptive Server 便会创建游标的结构。但 Adaptive Server 不在游标声明阶段编译游标，直到打开游标为止。

  有关 declare cursor 的语法及说明，请参见第 611 页的 “declare cursor 语法”。请参见 《参考手册：命令》。

  下例声明一个缺省的不可滚动游标 business_crsr，用于查找 titles 表中所有商业书籍的书名和标识号。

  ```
  declare business_crsr cursor
  for select title, title_id
  from titles
  where type = "business"
  for update of price
  ```

  声明游标时，使用 for update 子句可确保 Adaptive Server 正确地执行定位型更新。在此示例中，可以使用游标更改价格。

  此示例声明可滚动游标 `authors_scroll_crsr`，用于查找 authors 表中来自加利福尼亚的作者。

```
declare authors_scroll_crsr scroll cursor
for select au_fname, au_lname
from authors
where state = 'CA'
```

因为可滚动游标是只读的，所以不能在游标声明中使用 for update 子句。

- **打开游标**　打开一个已在存储过程外声明的游标时， Adaptive Server 将编译该游标并生成优化查询计划。然后，它执行预备操作，以扫描游标中定义的行并为返回结果行做准备。

  如果在存储过程内声明游标， Adaptive Server 将在首次调用存储过程时编译游标。 Adaptive Server 还编译游标，生成优化查询计划并存储该计划以备后用。再次调用存储过程时，游标已经以编译格式存在。当打开游标时， Adaptive Server 只需执行预备操作，以执行扫描和返回结果集。

  注释　由于 Transact-SQL 语句是在游标的打开阶段编译的，因此任何与声明该游标相关的错误消息都会在游标打开阶段显示。

- **从游标中获取**　fetch 命令执行已编译的游标以返回满足游标中定义的条件的一行或多行。缺省情况下， fetch 命令只返回一行。

  在不可滚动游标中，第一个 fetch 返回满足游标搜索条件的第一行并存储游标的当前位置。第二个 fetch 使用第一个 fetch 的游标位置，返回满足搜索条件的下一行并存储其当前位置。每个后续的 fetch 都使用前一 fetch 的游标位置来定位下一游标行。

  在可滚动游标中，可以通过在 fetch 语句中指定 fetch 方向，获取任何行并将当前游标位置设置为结果集中的任何行。方向选项有 first、 last、 next、 prior、 absolute 和 relative。对于可滚动游标， fetch 可以在前后两个方向上执行，并且可以重复扫描结果集。

  可以使用 set cursor rows 来更改一条 fetch 命令所返回的行数。请参见第 617 页的 “通过每个 fetch 获取多行”。

  在下例中， fetch 命令显示 titles 表中包含一本商业书籍的第一行中的书名与标识号：

```
fetch business_crsr

title                                   title_id
-----------------------------------     --------
The Busy Executive's Database Guide     BU1032

(1 row affected)
```

再次运行 fetch business_crsr 将显示 titles 中下一本商业书籍的书名与标识号。

在下例中，对可滚动游标第一次执行 fetch 命令将显示 authors 表中包含来自加利福尼亚的作者的第十行：

```
fetch absolute 10 authors_scroll_crsr
au_fname au_lname
--------------------
Akiko Yokomoto
```

再次执行具有 prior 方向选项的 fetch 将返回第十行之前的那一行：

```
fetch prior authors_scroll_crsr
au_fname au_lname
------------------
Chastity Locksley
```

- **处理行** Adaptive Server 将更新或删除游标结果集中当前游标位置处的数据 （以及提供这些数据的相应基表中的这些数据）。这些操作是可选的。

  以下 update 语句将商业书籍的价格提高 5% ；它只影响 business_crsr 游标当前所指的书籍：

  ```
  update titles
  set price = price * .05 + price
  where current of business_crsr
  ```

  更新游标行会更改行数据或删除行。不能使用游标来插入行。通过游标执行的所有更新都会对游标结果集中包括的相应基表产生影响。

- **关闭游标** Adaptive Server 关闭游标结果集、删除所有剩余临时表，并释放游标结构占用的服务器资源。但是，它会为游标保留查询计划以便能够再次打开游标。使用 close 命令关闭游标。例如：

  ```
  close business_crsr
  ```

  当关闭一个游标然后重新打开它时， Adaptive Server 将重新创建游标结果，并将游标放在第一个有效行之前。这样就可根据需要多次处理游标结果集。可随时关闭游标，而不必处理整个结果集。

- **释放游标**　Adaptive Server 从内存删除查询计划并消除游标结构的所有跟踪。若要释放游标，请使用 deallocate cursor 命令。例如：

  ```
  deallocate cursor business_crsr
  ```

  关键字 cursor 在 Adaptive Server 15.0 中是可选的。

  游标必须重新声明后才能使用。

declare cursor 语法

declare cursor 语句必须位于该游标的任何 open 语句之前。在同一 Transact-SQL 批处理中，不能将 declare cursor 与其它语句结合使用，除非是在存储过程中使用游标。

select_statement 是定义游标结果集的查询。有关语法和用法信息，请参见 《参考手册：命令》。一般来讲， *select_statement* 几乎可以使用 Transact-SQL select 语句的全部语法和语义，包括关键字 holdlock。但它不能包含 compute、 for browse 或 into 子句。

游标敏感性

可以使用 insensitive 或 semi_sensitive 来显式指定游标敏感性。

insensitive 游标是在游标打开时获取的结果集的快照。当打开该游标时，就会创建一个内部工作表并用游标结果集完全填充。

基表上的所有锁都将释放，且当执行 fetch 时，只访问该工作表。如果对基表声明了该游标，则基表中的任何数据更改不会影响游标结果集。该游标是只读的，并且不能与 for update 一起使用。

在 semi_sensitive 游标中，基表中的某些数据更改可能在游标中显示。基表数据更改的可见性可能受以下因素影响：选择何种查询计划以及是否至少读取了一次数据行。

在使用工作表保存结果集以用于滚动用途方面， semi_sensitive 可滚动游标与 insensitive 游标相同。在 semi_sensitive 模式中，游标的工作表是在获取行时实现的，而不是在打开游标时实现的。仅在所有行都被读取过一次并复制到滚动工作表之后，结果集的成员资格才固定。

如果没有指定游标敏感性，则缺省值为 semi_sensitive。

即使将游标声明为 semi_sensitive，游标基表中数据更改的可见性还是取决于优化器所选择的查询计划。

即使已将该游标声明为 semi_sensitive，所有 sort 命令还是强制使该游标变为 insensitive，因为它要求表中的行进行排序后才能执行 sort。但可以在获取任何行之前填充工作表。

例如，如果 select 语句中包含 order by 子句，并且 order by 列无索引，则会在游标打开时完全填充工作表，不管是否将游标声明为 semi_sensitive。游标变为 insensitive。

一般来讲，尚未获取的行可以显示数据更改，而已获取的行则不能。

用 semi_sensitive 可滚动游标代替 insensitive 可滚动游标的主要好处是：结果集的第一行能迅速返回给用户，因为表锁是逐行应用的。如果获取一行并更新它，则该行将通过 fetch 成为工作表的一部分，且对基表进行更新。不需要等待结果集工作表被完全填充。

cursor_scrollability

可以使用 scroll 或 no scroll 来指定 cursor_scrollability。如果游标是可滚动的，可以通过反复读取任一行或多行在游标结果集中滚动，还可以反复扫描结果集。

所有可滚动游标都是只读的，在游标声明中，它们不能与 for update 一起使用。

read_only 选项

read_only 选项指定游标结果集不能更新。与此相反， for update 选项指定游标结果集是可更新的。可用在 *select_statement* 中定义为可更新的列列表指定 for update 后的 of *column_name_list*。

declare cursor 示例

以下 declare cursor 语句为 authors_crsr 游标定义一个结果集，其中包含所有不住在加利福尼亚的作者：

```
declare authors_crsr cursor
for select au_id, au_lname, au_fname
from authors
where state != 'CA'
for update
```

下例定义 stores_scrollcrsr 的一个 insensitive 可滚动结果集，其中包含位于加利福尼亚的书店：

```
declare storinfo_crsr insensitive scroll cursor
for select stor_id, stor_name, payterms
from stores
where state = "CA"
```

有关 declare cursor 的语法，请参见第 611 页的 “declare cursor 语法”。

若要声明名为 “C1” 的 insensitive 不可滚动游标，请输入：

```
declare C1 insensitive cursor for
select fname from emp_tab
```

若要声明名为 “C3” 的 insensitive 可滚动游标，请输入：

```
declare C3 insensitive scroll cursor for
select fname from emp_tab
```

若要获取第一行，请输入：

```
fetch first from <cursor_name>
```

还可以读取结果集中第一行的列。若要将它们放在 <*fetch_target_list*> 中所指定的变量中，请输入：

```
fetch first from <cursor_name> into
<fetch_target_list>
```

可以直接获取结果集中的第 20 行，不管游标的当前位置在哪里：

```
fetch absolute 20 from <cursor_name> into <fetch_target_list>
```

打开游标

声明游标后，必须打开它才能对行执行 fetch、 update 或 delete 操作。打开游标后， Adaptive Server 就可通过对 select 语句求值开始创建游标结果集，该语句定义游标并使其可供处理。 open 语句的语法是：

open *cursor_name*

不能打开已经打开的或尚未使用 declare cursor 语句定义的游标。可以重新打开已关闭的游标，以将游标位置重新设置到游标结果集的开头。

根据游标的类型和查询计划，打开游标时可能会创建一个工作表并填充它。

使用游标读取数据行

fetch 完成游标结果集并向客户端返回一行或多行。根据游标中定义的查询类型， Adaptive Server 通过直接扫描表或扫描由查询类型和游标类型生成的工作表，来创建游标结果集。

fetch 命令将不可滚动游标定位在游标结果集的第一行之前。如果表具有有效索引， Adaptive Server 将游标定位到第一个索引键处。

fetch 语法

first、 next、 prior、 last、 absolute 和 relative 指定可滚动游标的 fetch 方向。如果不指定任何关键字，则缺省值为 next。请参见 《参考手册：命令》。

如果使用 fetch absolute 或 fetch relative，请指定 *fetch_offset*。它可以是整数文字、标度为 0 的精确带符号的数值文字或标度为 0 的整数或数值数据类型的 Transact-SQL 局部变量。当游标位置超过了最后一行或在第一行之前时，不返回任何数据，也不产生任何错误。

当使用 fetch absolute 但 *fetch_offset* 大于或等于 0 时，从结果集第一行之前的位置计算偏移量。如果 fetch absolute 小于 0，则从结果集最后一行之后的位置计算偏移量。

如果使用 fetch relative，当 *fetch_offset* n 大于 0 时，游标位于当前位置之后的 *n* 行上；如果 *fetch_offset n>0*，游标位于当前位置之前的 *abs(n)* 行上。

例如，通过可滚动游标 *stores_scrollcrsr*，可以获取所需的任何行。此 fetch 将游标定位到结果集的第三行：

```
fetch absolute 3 stores_scrollcrsr
stor_id stor_name
---------------------------------------
7896 Fricative Bookshop
```

其后的 fetch prior 操作将游标定位到结果集的第二行：

```
fetch prior stores_scrollcrsr
stor_id stor_name
---------------------------------------
7067 News & Brews
```

再后的 fetch relative -1 将游标定位到结果集的第一行：

```
fetch relative -1 stores_scrollcrsr
stor_id stor_name
```

```
-------------------------
7066      Barnum's
```

生成游标结果集后，在不可滚动游标的 fetch 语句中， Adaptive Server 将游标在结果集中的位置移动一行。它从结果集检索数据并存储当前位置，从而允许执行其它 fetch，直到 Adaptive Server 到达结果集末端为止。

下例说明一个不可滚动游标。声明并打开 authors_crsr 游标后，可以对其结果集的第一行执行 fetch，如下所示：

```
fetch authors_crsr

au_id        au_lname             au_fname
-----------  -------------------- ---------------
341-22-1782  Smith                Meander

(1 row affected)
```

每个后续的 fetch 都从游标结果集中检索下一行。例如：

```
fetch authors_crsr

au_id        au_lname             au_fname
-----------  -------------------- ---------------
527-72-3246  Greene                Morningstar

(1 row affected)
```

对所有行执行 fetch 后，游标将指向结果集的最后一行。如果再次执行 fetch， Adaptive Server 将通过 *@@sqlstatus* 或 *@@fetch_status* 全局变量 （如第 616 页的 “检查游标状态”中所述）返回一则警告，指出不再有数据。游标的位置保持不变。

如果正在使用不可滚动游标，则不能获取已获取的行。关闭并重新打开该游标可重新生成游标结果集，并再次从头开始读取。

使用 *into* 子句

into 子句指定 Adaptive Server 将列数据返回到指定的变量中。*fetch_target_list* 必须由在前面声明的 Transact-SQL 参数或局部变量组成。

例如，声明变量 *@name*、*@city* 和 *@state* 后，可以从 pubs_crsr 游标读取行，如下所示：

```
fetch pubs_crsr into @name, @city, @state
```

还可以只读取结果集中第一行的列。若要将读取的列放入列表中，请输入：

```
fetch first from <cursor_name> into
<fetch_target_list>
```

检查游标状态

每次读取后， Adaptive Server 都返回一个状态值。可以通过全局变量 *@@sqlstatus*、 *@@fetch_status* 或 *@@cursor_rows* 来访问该值。*@@fetch_status* 和 *@@cursor_rows* 仅在 Adaptive Server 版本 15.0 和更高版本中受支持。

表 19-1 列出了 *@@sqlstatus* 的值及其含义：

表 19-1：@@sqlstatus 值

值	含义
0	fetch 语句成功完成。
1	fetch 语句出错。
2	结果集中不再有数据。如果当前游标位于结果集的最后一行上而客户端对该游标提交了 fetch 语句，就会出现这一警告。

表 19-2 列出了 *@@fetch_status* 的值及其含义：

表 19-2：@@fetch_status 值

值	含义
0	fetch 操作成功。
-1	fetch 操作失败。
-2	该值留作将来使用。

下例确定当前打开的 authors_crsr 游标的 *@@sqlstatus*：

```
select @@sqlstatus
---------
        0

(1 row affected)
```

下例确定当前打开的 authors_crsr 游标的 *@@fetch_status*：

```
select @@fetch_status

---------
        0

(1 row affected)
```

只有 fetch 语句能设置 *@@sqlstatus* 和 *@@fetch_status*。其它语句对 *@@sqlstatus* 没有影响。

@@cursor_rows 指示游标结果集中上次打开并获取的行数。第 617 页的 “@@cursor_rows 值”列出了 *@@cursor_rows* 的值及其含义。

表 19-3：@@cursor_rows 值

值	含义
-1	指示以下各项之一： • 游标是动态的。由于动态游标反映所有更改，因此符合游标条件的行数不断更改。您不能明确规定检索所有符合条件的行。 • 游标是半敏感的且可滚动，但尚未填充滚动工作表。结果集中符合条件的行数是未知的。
0	没有打开任何游标、上次打开的游标中没有符合条件的行，或者上次打开的游标已关闭或释放。
n	上次打开或读取的游标结果集被完全填充；返回的值 (n) 是结果集中的总行数。

通过每个 *fetch* 获取多行

可以使用 set cursor rows 命令更改 fetch 所返回的行数。但该选项不影响包含 into 子句的 fetch。

set cursor rows 的语法为：

set cursor rows *number* for *cursor_name*

number 指定游标的行数。 *number* 可以是无小数点的数值文字，也可以是 integer 类型的局部变量。声明的每个游标的缺省设置都为 1。无论游标是打开的还是关闭的，都可以为其对 cursor rows 选项执行 set。

例如，可以更改为 authors_crsr 游标读取的行数，如下所示：

```
set cursor rows 3 for authors_crsr
```

设置游标行数后， authors_crsr 的每个 fetch 将返回三行：

```
fetch authors_crsr

au_id        au_lname            au_fname
----------- ------------------- --------------
648-92-1872  Blotchet-Halls      Reginald
712-45-1867  del Castillo        Innes
722-51-5424  DeFrance            Michel

(3 rows affected)
```

游标定位到读取的最后一行上 （本例中为作者 Michel DeFrance）。

对于客户端应用程序而言，一次读取多行非常有效。如果读取多行， Open Client 或 Embedded SQL 将把发送给客户端应用程序的行放入缓冲区。虽然在客户端上看到的仍然是逐行访问，但实际上每个 fetch 减少了调用 Adaptive Server 的次数，从而提高了性能。

检查读取的行数

可以使用 *@@rowcount* 全局变量来监控到上一次读取为止返回给客户端的游标结果集行数。该变量显示任一时刻游标所见到的总行数。

在不可滚动游标中，一旦从游标结果集读取了所有行，*@@rowcount* 则代表该结果集中的总行数。总行数代表上次读取的游标中的最大 *@@cursor_rows* 值。

下例确定当前打开的 authors_crsr 游标的 *@@rowcount*：

```
select @@rowcount
----------
         5

(1 row affected)
```

在可滚动游标中， *@@rowcount* 没有最大值。每个获取操作都会使该值持续增加，不管获取的方向如何。

下例显示可滚动 insensitive 游标 `authors_scrollcrsr` 的 *@@rowcount* 值。假设结果集中有五行。游标打开后，*@@rowcount* 的初始值为 0：结果集的所有行都读取自基表并保存到工作表中。以下 fetch 示例中的所有行都是从工作表访问的。

```
fetch last authors_scrollcrsr   @@rowcount = 1
fetch first authors_scrollcrsr  @@rowcount = 2
fetch next authors_scrollcrsr   @@rowcount = 3
fetch relative 2 authors_scrollcrsr @@rowcount = 4
fetch absolute 3 authors_scrollcrs  @@rowcount = 5
fetch absolute -2 authors_scrollcrsr  @@rowcount = 6
fetch first authors_scrollcrsr  @@rowcount = 7
fetch absolute 0 authors_scrollcrsr  @@rowcount =7
(nodatareturned)
fetch absolute 2 authors_scrollcrsr  @@rowcount = 8
```

使用游标更新和删除行

如果游标是可更新的，可使用 update 或 delete 语句更新或删除行。Adaptive Server 通过检查定义游标的 select 语句来确定游标是否可更新。也可以使用 declare cursor 语句的 for update 子句将游标显式定义为可更新。有关详细信息，请参见第 606 页的 “使游标变为可更新”。

更新游标结果集的行

可以使用 update 语句的 where current of 子句更新当前游标位置处的行。对游标结果集所做的任何更新同样也会影响派生游标行的基表行。

update...where current of 的语法是：

```
update [[database.]owner.]{table_name | view_name}
    set [[[database.]owner.]{table_name.| view_name.}]
                column_name1  =
            {expression1 | NULL | (select_statement)}
        [, column_name2  =
            {expression2 | NULL | (select_statement)}]...
    where current of cursor_name
```

set 子句指定游标的结果集列名并赋予其新值。如果列出多个列名值对，必须用逗号将它们分开。

table_name 或 *view_name* 必须是定义游标的 select 语句的第一个 from 子句中指定的表或视图。如果该 from 子句引用多个表或视图 （使用连接），则只能指定实际被更新的表或视图。

例如，可以更新 pubs_crsr 游标当前所指向的行，如下所示：

```
update publishers
set city = "Pasadena",
    state = "CA"
where current of pubs_crsr
```

更新后，游标位置保持不变。只要另一个 SQL 语句没有移动该游标的位置，就可以继续更新该游标位置处的行。

Adaptive Server 允许更新游标的 *select_statement* 列列表中没有指定的列，但这些列必须是该语句中指定的表的一部分。但如果使用 for update 指定了 *column_name_list*，则只能更新该列表中的列。

删除游标结果集的行

可以使用 delete 语句的 where current of 子句删除当前游标位置处的行。从游标的结果集中删除一行时，该行将从基础数据库表中删除。使用游标时，每次只能删除一行。

delete...where current of 的语法为：

```
delete [from]
    [[database.]owner.]{table_name | view_name}
    where current of cursor_name
```

使用 delete...where current of 指定的 *table_name* 或 *view_name* 必须是在定义该游标的 select 语句的第一个 from 子句中指定的表或视图。

例如，若要删除 authors_crsr 游标当前所指向的行，请输入：

```
delete from authors
where current of authors_crsr
```

上例中的关键字 from 是可选的。

注释 不能从包含连接的 select 语句所定义的游标中删除行，即使该游标是可更新的。

从游标中删除一行后， Adaptive Server 将游标定位到游标结果集中被删除行的下一行之前。仍须使用 fetch 命令访问该行。如果被删除的行是游标结果集的最后一行， Adaptive Server 则将游标定位到结果集最后一行之后。

例如，删除上例中的当前行 （作者 Michel DeFrance）后，可获取游标结果集中的下三位作者 （假设 cursor rows 仍设置为 3）：

```
fetch authors_crsr

au_id        au_lname            au_fname
-----------  -------------------- ---------------
807-91-6654  Panteley            Sylvia
899-46-2035  Ringer              Anne
998-72-3567  Ringer              Albert

(3 rows affected)
```

不引用游标也可以从基表中删除行。对基表进行更改后，游标结果集随之更改。

关闭并释放游标

使用完游标结果集后，可以使用如下命令对其执行 close：

close *cursor_name*

关闭游标不会更改其定义。如果重新打开一个游标， Adaptive Server 将使用与以前相同的查询创建一个新的游标结果集。例如：

```
close authors_crsr
open authors_crsr
```

然后就可从其游标结果集的开头开始从 authors_crsr 获取。与该游标关联的所有条件 （如 set cursor rows 所定义的获取行数）仍然有效。

若要放弃游标，必须使用如下命令释放它：

deallocate cursor *cursor_name*

注释　在 Adaptive Server 15.0 及更高版本中，cursor 一词是可选的。

释放游标将释放与该游标相关联的所有资源，包括游标名。释放游标后，才能重新使用游标名。如果释放一个打开的游标， Adaptive Server 会自动关闭它。终止客户端与服务器的连接也会关闭并释放任何打开的游标。

可滚动的仅向前游标示例

仅向前 （缺省）游标

以下游标示例使用此查询：

```
select author = au_fname + " " + au_lname, au_id
from authors
```

查询结果如下：

```
author                      au_id
--------------------------  -----------
Johnson White               172-32-1176
Marjorie Green              213-46-8915
Cheryl Carson               238-95-7766
Michael O'Leary             267-41-2394
```

```
Dick Straight                274-80-9391
Meander Smith                341-22-1782
Abraham Bennet               409-56-7008
Ann Dull                     427-17-2319
Burt Gringlesby              472-27-2349
Chastity Locksley            486-29-1786
Morningstar Greene           527-72-3246
Reginald Blotchet Halls      648-92-1872
Akiko Yokomoto               672-71-3249
Innes del Castillo           712-45-1867
Michel DeFrance              722-51-5454
Dirk Stringer                724-08-9931
Stearns MacFeather           724-80-9391
Livia Karsen                 756-30-7391
Sylvia Panteley              807-91-6654
Sheryl Hunter                846-92-7186
Heather McBadden             893-72-1158
Anne Ringer                  899-46-2035
Albert Ringer                998-72-3567

(23 rows affected)
```

将游标与上述查询结合使用：

1 声明游标。

该 declare cursor 语句使用上面的 select 语句来定义游标：

```
declare newauthors_crsr cursor for
select author = au_fname + " " + au_lname, au_id
from authors
for update
```

2 打开游标：

```
open newauthors_crsr
```

3 使用游标获取行：

```
fetch newauthors_crsr

author                       au_id
-------------------------    -----------
Johnson White                172-32-1176

(1 row affected)
```

通过使用 set cursor rows 命令指定行数，可以每次获取多行：

```
set cursor rows 5 for newauthors_crsr
go
```

```
fetch newauthors_crsr

author                        au_id
-------------------------     -----------
Marjorie Green                213-46-8915
Cheryl Carson                 238-95-7766
Michael O'Leary               267-41-2394
Dick Straight                 274-80-9391
Meander Smith                 341-22-1782

(5 rows affected)
```

每个后续的 fetch 返回下五行：

```
fetch newauthors_crsr

author                        au_id
-------------------------     -----------
Abraham Bennet                409-56-7008
Ann Dull                      427-17-2319
Burt Gringlesby               472-27-2349
Chastity Locksley             486-29-1786
Morningstar Greene            527-72-3246

(5 rows affected)
```

游标现在位于作者 Morningstar Greene 处，这里是当前 fetch 的最后一行。

4　若要更改 Greene 的名字，请输入：

```
update authors
set au_fname = "Voilet"
where current of newauthors_crsr
```

游标停留在 Greene 女士的记录上，直到执行下一个 fetch 为止。

5　使用完游标后，可以关闭它：

```
close newauthors_crsr
```

如果重新对该游标执行 open， Adaptive Server 会重新运行上述查询并将游标放置到结果集的第一行之前。游标仍然设置为执行一次 fetch 返回五行。

6　使用 deallocate 命令删除游标：

```
deallocate cursor newauthors_crsr
```

释放游标后，才能重新使用游标名。

可滚动游标的示例表

本节中的示例由可滚动游标执行。若要生成表 19-4 中的数据，请执行：

```
select emp_id, fname, lname
from emp_tab
where emp_id > 2002000
```

基表 emp_tab 是一个数据行锁定表，表的 *emp_id* 字段上具有一个聚簇索引。“行位置”是一个假想列，其中的值代表每行在结果集中的位置。以下几节的示例中使用该表中的结果集，分别说明不敏感游标和半敏感游标。

表 19-4：执行 select 语句的结果

行位置	员工 ID	名字	姓氏
1	2002010	Mari	Cazalis
2	2002020	Sam	Clarac
3	2002030	Bill	Darby
4	2002040	Sam	Burke
5	2002050	Mary	Armand
6	2002060	Mickey	Phelan
7	2002070	Sam	Fife
8	2002080	Wanda	Wolfe
9	2002090	Nina	Howe
10	2002100	Sam	West

不敏感的可滚动游标

当声明并打开不敏感游标时，会创建一个工作表并用游标结果集完全填充它。基表上的锁被释放，只使用工作表来执行读取。

若要将游标 CI 声明为不敏感游标，请输入：

```
declare CI insensitive scroll cursor for
select emp_id, fname, lname
from emp_tb
where emp_id > 2002000

open CI
```

滚动工作表现在被填充了表 19-4 中显示的数据。若要将名字“Sam”更改为 “Joe”，请输入：

```
.....
```

```
update emp_tab set fname = "Joe"
where fname = "Sam"
```

现在，基表 emp_tab 中的四个 “Sam” 行消失，替换为四个 “Joe” 行。

```
fetch absolute 2 CI
```

该游标读取游标结果集的第二行，并返回第 2 行 “2002020, Sam, Clarac”。因为该游标是不敏感游标，所以它看不到更新的值，且返回行的值 （“Sam”，而非 “Joe”）与表 19-4 的第 2 行的值相同。

下一条命令向表 emp_tab 中再插入一个限定行 （即满足 declare cursor 中查询条件的行），但由于一个游标中的行成员资格是固定的，因此游标 CI 看不到所添加的行。输入：

```
insert into emp_tab values (2002101, "Sophie", "Chen", .., ..., ...)
```

以下 fetch 命令将游标滚动到工作表的末端，并读取结果集中的最后一行，结果返回行值 “2002100, Sam, West”。同理，因为游标是 insensitive，所以在游标 CI 的结果集中看不到 emp_tab 中插入的新行。

```
fetch last CI
```

半敏感的可滚动游标

半敏感的可滚动游标与 insensitive 游标类似，因为它们都使用工作表保存结果集以用于滚动目的。但在 semi_sensitive 模式中，游标的工作表是在获取行时实现的，而不是在打开游标时实现的。仅在所有行都被读取过一次并复制到滚动工作表之后，结果集的成员资格才固定。

若要将游标 CSI 声明为半敏感的可滚动游标，请输入：

```
declare CSI semi_sensitive scroll cursor for
select emp_id, fname, lname
from emp_tab
where emp_id > 2002000

open CSI
```

结果集的初始行包含表 19-4 中显示的数据。因为游标是半敏感的，所以在打开游标时，不会向工作表复制任何行。若要读取第一条记录，请输入：

```
fetch first CSI
```

该游标读取 emp_tab 的第一行并返回 “2002010, Mari, Cazalis”。该行被复制到工作表中。若要获取下一行，请输入：

```
fetch next CSI
```

该游标读取 emp_tab 的第二行并返回 “2002020, Sam, Clarac”。该行被复制到工作表中。若要将名字 “Sam” 替换为名字 “Joe”，请输入：

```
.......
update emp_tab set fname = "Joe"
where fname = "Sam"
```

基表 emp_tab 中的四个 “Sam” 行消失，变成了四个 “Joe” 行。若要只读取第二行，请输入：

```
fetch absolute 2 CSI
```

该游标读取结果集的第二行，并且返回员工 ID 2002020，但返回行的值是 “Sam” 不是 “Joe”。因为该游标是半敏感的，所以该行在更新之前已复制到工作表中，又因为返回的行来自结果集滚动工作表，所以游标看不到 update 语句所做的数据更改。

若要读取第四行，请输入：

```
fetch absolute 4 CSI
```

该游标读取结果集的第四行。由于第四行 (2002040, Sam, Burke) 是在 “Sam” 更新为 “Joe” 之后获取的，因此返回的员工 ID 2002040 是 “Joe, Burke”。第三行和第四行现在被复制到工作表中。

若要添加一个新行，请输入：

```
insert into emp_tab values (2002101, "Sophie", "Chen", .., ..., ...)
```

结果集中又添加一个限定行。该行在以下 fetch 语句中是可见的，因为游标是半敏感的，并且尚未获取最后一行。若要获取更新版本，请输入：

```
fetch last CSI
```

fetch 语句读取结果集中的 “2002101, Sophie, Chen”。

使用带有 last 选项的 fetch 后，将游标 CSI 的所有限定行复制到了工作表中。基表 emp_tab 上的锁定被释放，且游标 CSI 的结果集得以固定。emp_tab 中的任何进一步数据更改都不会影响 CSI 的结果集。

注释 锁定方案和事务隔离级别也会影响游标的可见性。上例基于缺省隔离级别，即级别 1。

在存储过程中使用游标

游标在存储过程中特别有用。通过游标，只用一个查询就能完成原本需要几个查询才能完成的任务。但所有游标操作必须在单个过程中执行。存储过程不能对未在该过程中声明的游标执行 open、fetch 或 close。在存储过程范围以外的游标将被视为未定义。请参见第 604 页的 “游标范围”。

例如，存储过程 au_sales 检查 sales 表，以查看某个作者的所有书籍是否很畅销。它使用游标来检查每一行，然后输出信息。如果不使用游标，则需要几个 select 语句才能完成同样的任务。在存储过程之外，不能将其它语句和 declare cursor 在同一批处理中执行。

```
create procedure au_sales (@author_id id)
as

/* declare local variables used for fetch */
declare @title_id tid
declare @title varchar(80)
declare @ytd_sales int
declare @msg varchar(120)

/* declare the cursor to get each book written
    by given author */
declare author_sales cursor for
select ta.title_id, t.title, t.total_sales
from titleauthor ta, titles t
where ta.title_id = t.title_id
and ta.au_id = @author_id

open author_sales
fetch author_sales
        into @title_id, @title, @ytd_sales
if (@@sqlstatus = 2)
begin
    print "We do not sell books by this author."
    close author_sales
    return
end

/* if cursor result set is not empty, then process
    each row of information */
while (@@sqlstatus = 0)
begin
    if (@ytd_sales = NULL)
    begin
```

```
                select @msg = @title +
                   " -- Had no sales this year."
                print @msg
            end
            else if (@ytd_sales < 500)
            begin
                select @msg = @title +
                    " -- Had poor sales this year."
                print @msg
            end
            else if (@ytd_sales < 1000)
            begin
                select @msg = @title +
                    " -- Had mediocre sales this year."
                print @msg
            end
            else
            begin
                select @msg = @title +
                    " -- Had good sales this year."
                print @msg
            end

            fetch author_sales into @title_id, @title,
            @ytd_sales
        end
```

例如：

```
    au_sales "172-32-1176"
Prolonged Data Deprivation:Four Case Studies -- Had good sales this year.

(return status = 0)
```

有关存储过程的详细信息，请参见第 17 章 “使用存储过程”。

游标和锁定

游标锁定方法与 Adaptive Server 的其它锁定方法类似。一般来讲，读取数据的语句（例如 select 或 readtext）在每个数据页上使用共享锁，以避免从未提交的事务读取已更改的数据。更新语句在其更改的每一页上使用**排它锁**。为减少死锁发生及提高并发性能，Adaptive Server 经常在排它锁前放置一个更新锁，这表示客户端想更改该页上的数据。

缺省情况下，对于可更新游标， Adaptive Server 在扫描通过 declare cursor 的 for update 子句引用的表或视图时使用更新锁。如果包括 for update 子句但列表为空，则在缺省情况下， *select_statement* 的 from 子句中引用的所有表和视图将接收更新锁。如果不包括 for update 子句，则引用的表和视图将接收共享锁。如果对于某些表您更愿意使用**共享锁**，只要将关键字 shared 添加到每个表名后的 from 子句中，就可以用共享锁代替更新锁。

在不敏感游标中，在工作表被完全填充后释放基表锁。在半敏感的可滚动游标中，在获取一次结果集的最后一行后释放基表锁。

注释　Adaptive Server 在游标的位置移出数据页时释放更新锁。因为应用程序为客户端游标将行放入缓冲区中，所以相应的服务器游标可能会被定位到与客户端游标不同的数据行和页上。在这种情况下，另一个客户端可以更新代表第一个客户端当前游标位置的行，即使第一个客户端使用了 for update 选项。

事务结束之前，事务中的游标一直持有它所获取的所有排它锁。当您使用 holdlock 关键字或 set isolation level 3 选项时共享锁或更新锁也是如此。但是，如果不设置 close on endtran，则该游标在事务结束后仍保持打开，而且其当前页锁保持有效。它还能在读取其它行的同时继续获取锁。

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》）。

游标锁定选项

在定义可更新游标时，指定 holdlock 或 shared 选项 （在 select 语句中）会产生以下几种结果：

- 如果两个选项都省略，只能读取当前读取的页上的数据。其他用户不能通过游标或其它方式更新您当前读取的页。其他用户可以在您的游标所使用的相同表上声明游标，但不能获取您当前读取的页上的更新锁。
- 如果指定 shared 选项，只能读取当前读取的页上的数据。其他用户不能通过游标或其它方式更新您当前读取的页。
- 如果指定 holdlock 选项，可以读取所有读取的页 （在当前事务中）或仅当前读取的页 （如果不在事务中）上的数据。其他用户不能通过游标或其它方式更新您当前读取的页或在当前事务中读取的页。其他用户可以在您的游标所使用的相同表上声明游标，但不能获取您当前读取的页或在当前事务中读取的页上的更新锁。
- 如果两个选项都指定，则可以读取所有读取的页 （在当前事务中）或仅当前读取的页 （如果不在事务中）上的数据。其他用户不能通过游标或其它方式更新您当前读取的页。

有关游标的信息

可以使用 sp_cursorinfo 来查找有关游标的名称、其当前状态以及其结果列的信息。下例显示有关 authors_crsr 的信息：

```
sp_cursorinfo 0, authors_crsr

Cursor name 'authors_crsr' is declared at nesting
    level '0'.
The cursor id is 327681
The cursor has been successfully opened 1 times
The cursor was compiled at isolation level 1.
The cursor is not open.
The cursor will remain open when a transaction is
    committed or rolled back.
The number of rows returned for each FETCH is 1.
The cursor is updatable.
There are 3 columns returned by this cursor.
The result columns are:
Name = 'au_id', Table = 'authors', Type = ID,
```

```
    Length = 11 (updatable)
Name = 'au_lname', Table = 'authors', Type =
    VARCHAR, Length = 40 (updatable)
Name = 'au_fname', Table = 'authors', Type =
    VARCHAR, Length = 20 (updatable)
```

下例显示有关可滚动游标的信息：

```
sp_cursorinfo 0, authors_scrollcrsr


Cursor name 'authors_scrollcrsr' is declared at nesting level '0'.

The cursor is declared as SEMI_SENSITIVE SCROLLABLE cursor.

The cursor id is 786434.

The cursor has been successfully opened 1 times.

The cursor was compiled at isolation level 1.

The cursor is currently scanning at a nonzero isolation level.

The cursor is positioned on a row.

There have been 1 rows read, 0 rows updated and 0 rows deleted through this
cursor.

The cursor will remain open when a transaction is committed or rolled back.

The number of rows returned for each FETCH is 1.

The cursor is read only.

This cursor is using 19892 bytes of memory.

There are 2 columns returned by this cursor.

The result columns are:

Name = 'au_fname', Table = 'authors', Type =VARCHAR, Length = 20 (not updatable)

Name = 'au_lname', Table = 'authors', Type = VARCHAR, Length = 40 (not updatable)
```

也可以使用全局变量 *@@sqlstatus*、*@@fetch_status*、*@@cursor_rows* 和 *@@rowcount* 来检查游标的状态。请参见第 616 页的 “检查游标状态”和第 618 页的 “检查读取的行数”。

请参见 《参考手册：过程》。

用浏览模式代替游标

浏览模式允许在表中每次搜索一行并更新其值。该模式用于那些使用 DB-Library 和主机编程语言的前端应用程序。浏览模式很有用，因为它提供了与 Open Server™ 应用程序和早期 Open Client™ Library 的兼容性。但不鼓励在最新的 Client-Library™ 应用程序（10.0.x 和更高版本）中使用它，因为游标可提供相同的功能，且可移植性和灵活性更好。此外，因为浏览模式是 Sybase 所特有的功能，所以不适于异构环境。

通常，要逐行更改表值，应使用游标更新数据。 Client-Library 应用程序可以使用 Client-Library 游标实现某些浏览模式功能，例如从表读取行的同时更新表。但是，游标可能会导致选定的表中发生锁争用问题。

有关浏览模式的详细信息，请参见 Open Client/Server 文档中的 dbqual 函数。

浏览表

若要在前端应用程序中浏览表，可在发送给 Adaptive Server 的 select 语句的末尾附加 for browse 关键字。

例如：

Start of select *statement in an Open Client application*

...

for browse

Completion of the Open Client application routine

如果表中的行带有时间戳，就可以在前端应用程序中浏览该表。

浏览模式限制

不能在包含 union 运算符的语句或游标声明中使用 for browse 子句。

不能在包括 for browse 选项的 select 语句中使用关键字 holdlock。

在浏览模式下，忽略 select 语句中的关键字 distinct。

为用于浏览的新表加盖时间戳

创建用于浏览的新表时，可在表定义中包括名为 timestamp 的列。会自动为此列分配 timestamp 数据类型。例如：

```
create table newtable(col1 int, timestamp,
    col3 char(7))
```

当插入或更新行时，Adaptive Server 会通过自动给 timestamp 列赋予一个唯一 varbinary 值来为其加盖时间戳。

为现有表加盖时间戳

若要准备一个现有表以用于浏览，请使用 alter table 来添加名为 timestamp 的列。例如：

```
alter table oldtable add timestamp
```

将一个包含空值的 timestamp 列添加到每个现有行中。若要生成时间戳，请更新每行，而无需指定新列值。

例如：

```
update oldtable
set col1 = col1
```

比较 *timestamp* 值

在前端应用程序中采用浏览模式时，可使用 tsequal 系统函数来比较时间戳。例如，以下语句更新 publishers 中已浏览的一行。它比较该行浏览版本中的 timestamp 列与存储版本中的十六进制时间戳。如果两个时间戳不相等，您就会收到一条错误消息，并且不会更新该行。

```
update publishers
set city = "Springfield"
where pub_id = "0736"
and tsequal(timestamp,0x0001000000002ea8)
```

不要在 where 子句中使用 tsequal 函数作为搜索参数。如果使用 tsequal，where 子句的其余部分应与单个行唯一匹配。只在 insert 和 update 语句中使用 tsequal 函数。如果将 timestamp 列用作搜索子句，则应像比较常规 varbinary 列那样来比较该列，即 *timestamp1 = timestamp2*。

连接游标处理和数据修改

本章的剩余部分将只介绍 Adaptive Server 11.9.2 至 12.5.3 版中的游标行为，尤其是仅数据锁定表上的游标的定位和修改规则。有关 12.5.3 以后版本中的游标行为的信息，请参见第 602 页的 “敏感性和可滚动性”。

可能影响游标位置的更新和删除

有两种类型的删除和更新会影响游标位置处的行：

- *定位型* 删除和更新使用 delete...where current of 或 update...where current of 来更改游标位置处的行
- *搜索型* 删除和更新是指更改游标位置处行中的值，但不包括 where current of 子句的任何 delete 或 update 查询

执行 *delete* 或 *update* 命令后没有连接的游标定位

对于单表查询上的游标， delete 或 update 命令对基表的行为取决于修改类型、基表的锁定方案以及基表的聚簇索引是否受影响等因素：

- 删除游标位置处的行的定位型或搜索型 delete 命令将游标定位到表的下一限定行上。所有页锁定表和仅数据锁定表都是如此。不允许随后的使用相同游标的定位型 update 或 delete，直到下一个 fetch 将游标定位到下一限定行。
- 不更改行位置的搜索型或定位型 update 命令保持游标的当前位置不变。下一个 fetch 返回下一个限定行。
- 所有页锁定表上的搜索型或定位型 update 命令可以更改行的位置；例如，假设它更新了聚簇索引的键列。游标不跟踪行；游标仍定位到下一行前的初始位置。不允许执行定位型更新，直到随后的 fetch 返回下一行。如果更新的行移到搜索顺序中靠后的位置，则游标可能会再次看到它。

  仅数据锁定表具有固定的行 ID，因此扩展更新或影响聚簇键的更新不会移动行的位置。游标仍定位到该行上，而下一个 fetch 返回下一个限定行。

这种游标定位行为从 11.9.2 之前的版本起一直未变。

更新和删除对连接游标的影响

在包含连接的游标的游标位置处的行上发出搜索型或定位型 delete 命令时，会出现以下两种结果之一：

- 搜索型删除隐式关闭该游标。下一个 fetch 返回错误 582：

  ```
  Cursor 'cursor_name' was closed implicitly
  because the current cursor position was deleted
  due to an update or a delete.The cursor scan
  position could not be recovered. This happens for
  cursors which reference more than one table.
  ```

- 定位型删除失败，并显示错误 592：

  ```
  The DELETE WHERE CURRENT OF to the cursor
  'cursor_name' failed because the cursor is on a
  join.
  ```

  游标仍保持打开，定位于同一行。

在游标的连接列上发出搜索型或定位型 update 命令时：

- 对所有页锁定表上的聚簇索引键的搜索型更新成功，但会隐式关闭该游标，因此下一个 fetch 返回错误 582。对聚簇索引键的搜索型更新不会关闭仅数据锁定表上的游标。
- 所有锁定方案的定位型更新以及不更改所有页锁定表上聚簇索引键的搜索型更新成功，并且不关闭该游标。

  游标位置取决于表的类型和列是否具有聚簇索引两个因素。有关详细信息，请参见第 634 页的 “执行 delete 或 update 命令后没有连接的游标定位”。

  当连接列更新时，将连接列放入缓冲区可能会影响返回的结果集。有关详细信息，请参见第 634 页的 “连接游标处理和数据修改”。

表 19-5 显示了 delete 和 update 命令是如何影响连接游标的。“保持打开”指示游标没有被更新关闭。游标仍定位到该行上，因此可以继续执行定位型更新，而下一个 fetch 也会成功。

表 19-5：连接游标中的删除和连接列更新的影响

	所有页锁定	仅数据锁定
delete 命令		
定位型直接删除	错误 592	错误 592
搜索型直接删除	错误 582	错误 582
搜索型延迟删除	错误 582	错误 582

	所有页锁定	仅数据锁定
影响聚簇索引的搜索型删除（延迟）	错误 582	错误 582
update 命令		
定位型直接更新	保持打开	保持打开
搜索型直接更新	保持打开	保持打开
搜索型延迟更新	保持打开	保持打开
影响聚簇索引的搜索型更新（延迟）	集合 582	保持打开

将连接列放入缓冲区对连接游标的影响

对于包含连接的查询上的游标，将在执行第一个 fetch 后缓冲连接顺序中的外部表的连接列、搜索参数和选择列表值。如果从连接顺序中的内部表返回多行，则后续读取将缓冲区内的值用于外部行。这意味着，通过游标更新连接列和搜索参数的应用程序的行为会随着为查询选择的连接顺序的不同而不同。将连接列放入缓冲区对所有页锁定表和仅数据锁定表都有影响。

注释 在 11.5.x 版中，所有页锁定表上的连接游标不会缓冲连接值。

游标扫描期间将列放入缓冲区的影响

如果在会话期间对连接列进行了更新，则根据为查询选择的连接顺序，执行连接的游标查询可能会产生多种结果。以下两个示例说明连接顺序是如何影响游标结果集的。

```
select * from dept

dept_id deptloc
------- -----------------
      1 Elm Street
      2 Acacia Drive
      3 Maple Lane
      4 Oak Avenue

select * from employee

dept_id empid
------- ---------
      1 172321176
      1 213468915
      2 341221782
      2 409567008
```

```
2 427172319
3 472272349
3 486291786
```

将连接列放入缓冲区示例

下面的语句声明一个游标并打开它，然后获取第一行：

```
declare j_curs cursor for
select d.dept_id, e.empid, d.deptloc
from dept d , employee e
where d.dept_id = e.dept_id
and d.dept_id = 2
open j_curs
fetch j_curs

dept_id     empid       deptloc
----------- ----------- --------------------
          2   341221782 Acacia Drive
```

如果 dept 被选作连接顺序中的外部列，则将 dept_id 的值 2 放入缓冲区。下面的 update 更改 dept 中连接列的 dept_id：

```
update dept set dept_id = 12
where current of j_curs
```

此 update 命令更改表行中存储的值，但不变更缓冲的值，这就使游标看不到对基表的这一更新的结果。如果 dept 中的连接列由非定位型更新所更新，也会产生同样的效果。

以下 update 更改 employee 行的 dept_id：

```
update employee set dept_id = 12
where current of j_curs
```

对表执行下一个 fetch 将返回结果集中的第二行：

```
fetch j_curs

dept_id     empid       deptloc
----------- ----------- --------------------
          2   409567008 Acacia Drive
```

可获取和更新 employee 中的所有匹配行。

如果此游标的连接顺序选择 employee 作为外部表，且在执行第一个 fetch 后更新了 dept 中的连接列，则第二个 fetch 将找不到匹配行。步骤依次为：

1 employee.dept_id 的值 2 在执行第一个 fetch 期间被放入缓冲区。

2 搜索型或定位型更新将 dept.dept_id 的值更改为 12。

3 在执行第二个 fetch 时，使用缓冲的 employee.dept_id 的值 2，并扫描内部表以查找匹配值；因为 dept.dept_id 已更改为 12，所以第二个 fetch 不返回行。

employee 中其余两个 dept_id 为 2 的行不能被游标读取。

将搜索参数放入缓冲区示例

由于将搜索参数放入缓冲区，因此游标结果对游标选择查询上的搜索参数更新的敏感性也取决于连接顺序。本例使用第 636 页的 “游标扫描期间将列放入缓冲区的影响”中所示的 dept 和 employee 表。下面的游标通过对 deptloc 列使用搜索参数，连接表的 dept_id 列：

```
declare c cursor for
select d.dept_id, empid, deptloc
from dept d , employee e
where d.dept_id = e.dept_id
and deptloc = "Acacia Drive"
```

对该表执行第一个 fetch 将返回以下行：

```
dept_id   empid       deptloc
--------- ----------- --------------
        2 41221782    Acacia Drive
```

以下定位型更新语句将当前行的 employee.dept_id 值更改为 4；必须对位于 Acacia Drive 的部门的每个员工执行该语句：

```
update employee set dept_id = 4
where current of c
```

以下定位型更新将 dept.deptloc 值从 “Acacia Drive” 更改为 “Pine Way”：

```
update dept set deptloc = "Pine Way"
where current of c
```

如果在执行第一个 fetch 后对 deptloc 发出 update，则结果集是否返回需要更改的所有行是不确定的；其结果取决于为查询选择的连接顺序：

- 如果选择 dept 作为外部表，则将 dept.dept_id 和 dept.deptloc 列、值 2 和 “Acacia Drive”放入缓冲区。执行第一个 fetch 后，对 deptloc 的 update 将更新基表而非缓冲区中的值，并且后续的 fetch 命令将返回其它结果集行。
- 如果选择 employee 作为外部表，则仅将 employee 表上的 dept_id 限定放入缓冲区。对 deptloc 列的 update 将会更改基表，并在下一次执行 fetch 以将搜索参数限定值 “Acacia Drive”应用到 deptloc 表时，不会返回任何行，即使表中仍有两个 employee.dept_id 为 2 的行也是如此。

在读取了 employee 表的所有行后对 deptloc 发出 update，则两个更新都完成，并且不依赖连接顺序。

将选择列表放入缓冲区的影响

当读取连接顺序中外部表的某一行时，该外部表的选择列表中的各列将被放入缓冲区。如果对选择列表中的某一列执行搜索型或定位型更新，且从内部表读取另一行，则缓冲区中的外部表的值会显示在游标结果行中。

建议

一般来讲，当游标扫描正在进行时，应用程序应尝试避免通过更新连接列或带有搜索子句和其它谓词的列来更改其值。

- 尽量避免更新游标的连接列或连接游标的搜索参数，除非十分确定连接顺序。
- 使用创建工作表的游标查询，然后对基表使用搜索型更新和删除。当游标选择查询要求一个工作表时，游标对基础表的更改始终是 insensitive。包含 order by、 distinct、 group by 或其它创建工作表的子句的游标不能用定位型更新来更新。
- 如果您正在使用游标同时更新两个连接列，可考虑用搜索型更新代替定位型更新。虽然进一步的读取可能返回缓冲区中的值而非数据行中已更改的值，但使用搜索型更新可以避免因连接顺序的选择而导致无法对匹配行执行 fetch 的可能性。对所有页锁定表中聚簇索引键的搜索型更新将隐式关闭游标。
- ANSI SQL 92 初级规范不允许使用游标更新连接列，因此该行为会因具体实现而有所不同。对于设计为可在不同数据库软件上移植的应用程序，必须考虑其各自的实现情况。

第 20 章

触发器：强制实施参照完整性

本章讨论触发器。触发器是在表中插入、删除或更新数据时起作用的存储过程。可以使用触发器执行多种自动操作，例如，对相关表进行级联更改，强制实施列限制，比较数据修改的结果和维护数据库中数据的参照完整性。

触发器的工作方式

触发器可*自动运行*。只要数据发生改变 （无论是数据录入人员进行数据录入还是应用程序执行的操作），触发器就会工作。触发器专门用于一种或多种数据修改操作 （update、 insert 和 delete），并对每个 SQL 语句执行一次。

例如，若要防止用户删除 publishers 表中的任何出版公司，可以使用以下触发器：

```
create trigger del_pub
on publishers
for delete
as
begin
    rollback transaction
```

```
        print "You cannot delete any publishers!"
    end
```

如果下次有人尝试删除 publishers 表中的行， del_pub 触发器将取消该删除操作，回退事务并输出一条消息。

只有在数据修改语句完成操作，并且 Adaptive Server 完成对所有数据类型、规则或完整性约束冲突的检查之后，触发器才 “触发”。触发器和引发它的语句将被当作单个事务，可从触发器中回退。如果 Adaptive Server 检测到严重错误，则回退整个事务。

触发器在以下情况中特别有用：

- 触发器可对数据库中的相关表进行级联更改。例如， titles 表中 title_id 列上的删除触发器可将 title_id 列用作唯一键，以此来定位表 titleauthor 和 roysched 中的匹配行，进而删除这些行。
- 触发器可以不接受或回退违反了参照完整性的更改，从而取消尝试修改数据的事务。当尝试插入的外键与其主键不匹配时，这种触发器就会发挥作用。例如，可以在 titleauthor 上创建一个插入触发器，如果新插入的 titleauthor.title_id 值在 titles.title_id 中没有匹配值，该触发器便会回退该插入操作。
- 触发器能强制实施比通过规则定义的限制更为复杂的限制。与规则不同，触发器能引用列或数据库对象。例如，触发器能够回退尝试将图书价格增加超过预付款 1% 的更新操作。
- 触发器可以执行简单的 “what if” 分析。例如，触发器能比较数据修改前后的表状态，并根据比较结果采取相应操作。

使用触发器与使用完整性约束的对比

作为使用触发器的一种替代方法，可使用 create table 语句的参照完整性约束在数据库表中强制实施参照完整性。然而，参照完整性约束 *不能*：

- 对数据库中的相关表进行级联更改
- 通过引用其它列或数据库对象强制实施复杂限制
- 执行 “what if” 分析

另外，由于参照完整性约束强制实施数据完整性，因此它不能回退当前事务。使用触发器则可以根据处理参照完整性的方式回退或继续事务。有关事务的信息，请参见第 22 章 “事务：维护数据一致性和恢复”。

如果应用程序需要执行上述任务之一，则请使用触发器。否则，请使用参照完整性约束来强制实施数据完整性。 Adaptive Server 在检查触发器之前将检查参照完整性约束，从而使违反该约束的数据修改语句不会再引发触发器。有关参照完整性约束的详细信息，请参见第 7 章 “创建数据库和表”。

创建触发器

触发器是一种数据库对象。创建触发器时，需指定应 “触发”或激活触发器的表和数据修改命令。然后指定触发器要执行的任何操作。

例如，每当有人尝试在 titles 表中插入、删除或更新数据时，以下触发器都会输出一条消息：

```
create trigger t1
on titles
for insert, update, delete
as
print "Now modify the titleauthor table the same way."
```

注释　除了名为 deltitle 的触发器外，本章所讨论的触发器均未包括在 Adaptive Server 附带的 pubs2 数据库中。要使用本章演示的示例，可通过键入 create trigger 语句来创建各个触发器示例。对于表中或列中的相同操作 （insert、 update 或 delete），每个新触发器将在不发出警告的情况下覆盖先前的触发器，然后自动删除旧触发器。

create trigger 语法

create 子句可创建和命名触发器。触发器的名称必须符合标识符规则。

on 子句给出了激活触发器的表的名称。此表有时称为**触发器表**。

虽然触发器可以引用其它数据库中的对象，但它是在当前数据库中创建的。限定触发器名的所有者名必须与表的所有者名相同。只有表所有者才能在表中创建触发器。如果表所有者是在 create trigger 子句或 on 子句中与表名一起给出的，则在其它子句中也必须指定该所有者。

for 子句指定触发器表中可激活触发器的数据修改命令。在前面的示例中，对 titles 执行的 insert、 update 或 delete 操作都会输出该消息。

SQL 语句可指定**触发器条件**和**触发器动作**。触发器条件可指定附加标准，用于决定 insert、delete 或 update 是否会导致执行触发器动作。可在 if 子句中用 begin 和 end 对多个触发器动作进行分组。

if update 子句可对指定列中的插入或更新进行测试。对于更新操作，如果列名包含在 update 语句的 set 子句中，则即使更新没有更改该列的值， if update 子句的求值结果也为 true。不要将 if update 子句与 delete 一起使用。可以指定多个列，也可以在 create trigger 语句中使用多个 if update 子句。由于在 on 子句中指定了表名，因此在使用 if update 时，不要在列名前面使用表名。请参见 《参考手册：命令》。

不允许在触发器中使用的 SQL 语句

由于触发器是作为事务的一部分执行的，因此在触发器中不允许使用下列语句：

- 所有 create 命令，包括 create database、 create table、 create index、 create procedure、 create default、 create rule、 create trigger 和 create view
- 所有 drop 命令
- alter table 和 alter database
- truncate table
- grant 和 revoke
- update statistics
- reconfigure
- load database 和 load transaction
- disk init、 disk mirror、 disk refit、 disk reinit、 disk remirror、 disk unmirror
- select into

使用触发器来维护参照完整性

触发器用于维护参照完整性，这可确保数据库中的重要数据 （如给定数据段的唯一标识符）始终精确无误，并在数据库更改后仍可使用。参照完整性是通过使用主键和外键来实现的。

主键是指其值可对行进行唯一标识的列或列组合。该值不能为空且必须有唯一索引。带有主键的表可以与其它表的外键相连接。主键表可以看作是具有**主 - 明细关系**的**主表**。一个数据库中可以有许多个这样的主 - 明细组。

可以使用 sp_primarykey 来标记主键。此命令会标记用于 sp_helpjoins 的键并将其添加到 syskeys 表中。

例如， title_id 列是 titles 的主键。它唯一地标识了 titles 中的图书，并与 titleauthor、 salesdetail 和 roysched 中的 title_id 相连接。 titles 表对于 titleauthor、 salesdetail 和 roysched 来说是主表。

第 730 页的 “pubs2 数据库框图”说明了 pubs2 表是如何相关的。第 739 页的 “pubs3 数据库框图”提供了 pubs3 数据库的相同信息。

外键是一个列或列组合，它们的值与主键相匹配。外键不必是唯一的。它通常与主键有着多对一的关系。外键值应是主键值的副本。这意味着，如果主键中某个值不存在，则在外键中也不应存在此值。外键的值可以为空；如果组合外键值有一部分为空，则整个外键的值一定为空。带有外键的表通常称作主表的**明细**表或**相关**表。

可以使用 sp_foreignkey 标记数据库中的外键。标记后的这些外键可用于 sp_helpjoins 以及引用 syskeys 表的其它过程。 titleauthor、 salesdetail 和 roysched 中的 title_id 列都是外键；这些表是明细表。大多数情况下，由于单个表允许的最大引用数为 200，因此可以使用第 246 页的 “指定参照完整性约束”中描述的参照约束在表与表之间强制实施参照完整性。如果某个表超过该限制或需要特殊的参照完整性，则应使用参照完整性触发器。

参照完整性触发器可保持外键值与主键值相匹配。当数据修改影响到键列时，触发器将使用名为**触发器测试表**的临时工作表来比较新列值与相关的键。在编写触发器时，应基于临时存储在触发器测试表中的数据进行比较。

根据触发器测试表测试数据修改

Adaptive Server 在触发器语句中使用两种特殊的表：deleted 表和 inserted 表。它们是触发器测试时使用的临时表。编写触发器时，可以使用这些表来测试数据修改的影响并设置触发器的动作条件。不能直接变更触发器测试表中的数据，但可以在 select 语句中使用这些表来检测 insert、update 或 delete 的影响。

- deleted 表存储执行 delete 和 update 语句期间受到影响的行的副本。在执行 delete 或 update 语句期间，相关的行将会从触发器表中删除并被传送到 deleted 表中。通常情况下， deleted 表与触发器表没有共用的行。
- inserted 表存储执行 insert 和 update 语句期间受到影响的行的副本。在执行 insert 或 update 期间，新行将被同时添加到 inserted 表与触发器表中。 inserted 表中的行是触发器表中新行的副本。以下触发器片段使用 inserted 表来测试对 titles 表中 title_id 列的更改：

```
if (select count(*)
    from titles, inserted
    where titles.title_id = inserted.title_id) !=
    @@rowcount
```

注释 inserted 和 deleted 表在事务日志中均显示为视图，但它们在 syslogs 中为虚设表。

update 操作实际上是先删除行然后再插入行；旧行先被复制到 deleted 表中；然后新行被复制到触发器表和 inserted 表中。下面的示例说明了在执行 insert、 delete 和 update 期间触发器测试表的条件：

设置触发器条件时，应使用适合数据修改操作的触发器测试表。在测试 insert 时引用 deleted 或在测试 delete 时引用 inserted 并不会导致错误；但这些触发器测试表将不包含任何行。

注释 对于每个查询，给定的触发器只引发一次。如果触发器动作取决于数据修改所影响的行数，应对多行数据修改操作进行测试 （如检查 *@@rowcount*），并采取适当操作。

以下触发器示例包含必要的多行数据修改。用于存储受最近数据修改操作 “影响的行数”的 *@@rowcount* 变量可测试多行 insert、 delete 或 update 操作。如果触发器中有任何其它 select 语句先于对 *@@rowcount* 的测试，则应使用局部变量来存储该值，以便以后进行检查。所有不返回值的 Transact-SQL 语句都会将 *@@rowcount* 重新设置为 0。

触发器插入示例

插入新的外键行时，应确保该外键与某个主键相匹配。触发器应检查插入行 （使用 inserted 表）与主键表中的行之间的连接，然后回退与主键表中的键不匹配的任何外键插入操作。

下面的触发器将 inserted 表中的 title_id 值与 titles 表中的相应值进行比较。它假定您插入一个外键条目且没有插入空值。如果连接失败，该事务将被回退。

```
create trigger forinsertrig1
on salesdetail
for insert
as
if (select count(*)
    from titles, inserted
    where titles.title_id = inserted.title_id) !=
    @@rowcount
/* Cancel the insert and print a message.*/
begin
    rollback transaction
    print "No, the title_id does not exist in
    titles."
end
/* Otherwise, allow it. */
else
  print "Added!All title_id 担 exist in titles."
```

@@rowcount 指的是添加到 salesdetail 表中的行数。它也是向 inserted 表中添加的行数。该触发器将连接 titles 和 inserted，以确定所有添加到 salesdetail 的 title_id 是否都存在于 titles 表中。如果由 select count(*) 查询确定的连接的行数与 *@@rowcount* 不同，则表明有一个或多个插入不正确，该事务将被取消。

对于插入被回退及被接受的情况，此触发器分别输出不同的消息。要测试第一种情况，可尝试使用以下 insert 语句：

```
insert salesdetail
values ("7066", "234517", "TC9999", 70, 45)
```

要测试第二种情况，请输入：

```
insert salesdetail
values ("7896", "234518", "TC3218", 75, 80)
```

触发器删除示例

如果删除了主键行，也应删除相关表中的相应外键行。这样可确保在删除主行时也删除了明细行，从而保持了参照完整性。如果不删除相关表中的相应行，数据库中最终可能会有一些明细行不能被检索或识别。要正确删除相关外键行，可使用触发器来执行级联 delete 操作。

级联删除的示例

对 titles 执行 delete 语句时，将从 titles 表中删除一行或多行，并将这些行添加到 deleted 表中。触发器可检查相关表 （titleauthor、 salesdetail 和 roysched）以查看其中是否有这样的行，即行的 title_id 与从 titles 中删除并且现存储在 deleted 表中的 title_id 相匹配。如果触发器找到任何这样的行，便会将其删除。

```
create trigger delcascadetrig
on titles
for delete
as
delete titleauthor
from titleauthor, deleted
where titleauthor.title_id = deleted.title_id
/* Remove titleauthor rows that match deleted
** (titles) rows.*/
delete salesdetail
from salesdetail, deleted
where salesdetail.title_id = deleted.title_id
/* Remove salesdetail rows that match deleted
** (titles) rows.*/
delete roysched
from roysched, deleted
where roysched.title_id = deleted.title_id
/* Remove roysched rows that match deleted
** (titles) rows.*/
```

限制删除的示例

在实际生活中，因保留历史记录的需要 （为了查看下架停售的书在销售期间的销售量）或由于明细行上的事务尚未完成，您可能希望保留某些明细行。一个编写良好的触发器应考虑以下情况。

防止主键删除

如果在 salesdetail 表中有关于某一主键的任何明细行，则随 pubs2 一起提供的 deltitle 触发器可以防止删除该主键。以下触发器保留了从 salesdetail 中检索行的功能：

```
create trigger deltitle
```

```
on titles
for delete
as
if (select count(*)
    from deleted, salesdetail
    where salesdetail.title_id =
    deleted.title_id) > 0
begin
    rollback transaction
    print "You cannot delete a title with sales."
end
```

此触发器通过将从 titles 中删除的一个或多个行与 salesdetail 表连接起来，来对这些行进行测试。如果找到连接，该事务将被取消。

同样，如果主表 titles 在 titleauthor 中有相关子项，则下面的限制删除可防止删除发生。它不计算 deleted 和 titleauthor 中的行数，而是通过检查确定 title_id 是否被删除。这种方法的效率更高，这是由性能因素决定的，因为它仅检查特定的行是否存在，而不是浏览整个表并计算所有行数。

记录发生的错误

下一个示例将 raiserror 用于错误消息 35003。raiserror 可通过设置系统标志来记录发生的错误。尝试运行此示例前，请将错误消息 35003 添加到 sysusermessages 系统表中：

```
sp_addmessage 35003, "restrict_dtrig - delete
failed:row exists in titleauthor for this title_id."
```

触发器是：

```
create trigger restrict_dtrig
on titles
for delete as
if exists (select * from titleauthor, deleted where
          titleauthor.title_id = deleted.title_id)
   begin
           rollback transaction
           raiserror 35003
           return
   end
```

要测试此触发器，请尝试使用以下 delete 语句：

```
delete titles
where title_id = "PS2091"
```

触发器更新示例

下面的示例从主表 titles 向相关表 titleauthor 和 roysched 进行级联更新。

```
create trigger cascade_utrig
on titles
for update as
if update (title_id)
begin
     update titleauthor
           set title_id = inserted.title_id
           from titleauthor, deleted, inserted
           where deleted.title_id =
titleauthor.title_id
     update roysched
           set title_id = inserted.title_id
           from roysched, deleted, inserted
           where deleted.title_id = roysched.title_id
     update salesdetail
           set title_id = inserted.title_id
           from salesdetail, deleted, inserted
           where deleted.title_id =
salesdetail.title_id
end
```

要测试此触发器，请假定将 *Secrets of Silicon Valley* 一书从 popular_comp 重新归类为心理学书籍。以下查询在 titleauthor、 roysched 和 titles 中将 title_id PC8888 更新为 PS8888。

```
update titles
set title_id = "PS8888"
where title_id = "PC8888"
```

限制更新触发器

主键是它的行和其它表中的外键行的唯一标识符。通常，不应更新主键。尝试更新主键时应特别慎重。在这种情况下，除非满足指定的条件，否则会回退更新操作以保护参照完整性。

Sybase 建议禁止对主键进行任何编辑更改。例如，可通过撤消对该列的所有权限来实现这种禁止。但是，若要仅在某些条件下禁止更新，请使用触发器。

使用日期函数的限制更新触发器

下面的触发器可防止在周末对 titles.title_id 进行更新。stopupdatetrig 中的 if update 子句允许您关注特定的列 titles.title_id。如果对该列中的数据进行修改，则会导致触发器运行。而对其它列中的数据进行更改则不会。当此触发器检测到违反触发器条件的更新时，它将取消更新并输出消息。如果想测试此触发器，可以用当前的星期数代替 “Saturday”或“Sunday”。

```
create trigger stopupdatetrig
on titles
for update
as
/* If an attempt is made to change titles.title_id
** on Saturday or Sunday, cancel the update. */
if update (title_id)
      and datename(dw, getdate())
      in ("Saturday", "Sunday")
begin
    rollback transaction
    print "We do not allow changes to"
    print "primary keys on the weekend."
end
```

具有多个动作的限制更新触发器

可以使用 if update 对多个列指定多个触发器动作。下面的示例对 stopupdatetrig 进行修改，使其包括附加的触发器动作来响应对 titles.price 或 titles.advance 的更新。除了防止在周末对主键进行更新外，只要标题的总收入不超过其预付款的总额，它就还防止更新该标题的价格或预付款。可以使用相同的触发器名，因为在重新创建触发器时，修改后的触发器会替代旧的触发器。

```
create trigger stopupdatetrig
on titles
for update
as
if update (title_id)
  and datename(dw, getdate())
  in ("Saturday", "Sunday")
  begin
    rollback transaction
    print "We do not allow changes to"
    print "primary keys on the weekend!"
    end
if update (price) or update (advance)
  if exists (select * from inserted
    where (inserted.price * inserted.total_sales)
    < inserted.advance)
    begin
      rollback transaction
```

```
            print "We do not allow changes to price or"
            print "advance for a title until its total"
            print "revenue exceeds its latest advance."
        end
```

下一个示例是在 titles 上创建的，它在满足下列任一条件时，可防止 update 操作：

- 用户尝试更改 titles 中的 title_id 主键值
- 在 publishers 中未找到 pub_id 相关键
- 目标列不存在或为空

运用此示例之前，请确保 sysusermessages 中存在以下错误消息：

```
sp_addmessage 35004, "titles_utrg - Update Failed:update of primary keys %1!
is not allowed."
sp_addmessage 35005, "titles_utrg - Update Failed:%1! not found in authors."
```

触发器如下所示：

```
create trigger title_utrg
on titles
for update as
begin
     declare @num_updated int,
             @col1_var varchar(20),
             @col2_var varchar(20)
/* Determine how many rows were updated. */
select @num_updated = @@rowcount
    if @num_updated = 0
    return
/* Ensure that title_id in titles is not changed. */
if update (title_id)
    begin
      rollback transaction
     select @col1_var = title_id from inserted
     raiserror 35004, @col1_var
     return
   end
 /* Make sure dependencies to the publishers table are accounted for. */
 if update(pub_id)
   begin
     if (select count(*) from inserted, publishers
         where inserted.pub_id = publishers.pub_id
         and inserted.pub_id is not null) != @num_updated
     begin
         rollback transaction
```

```
            select @col1_var = pub_id from inserted
            raiserror 35005, @col1_var
            return
        end
    end
/* If the column is null, raise error 24004 and rollback the
** trigger.If the column is not null, update the roysched table
** restricting the update. */
    if update(price)
        begin
            if exists (select count(*) from inserted
            where price = null)
        begin
            rollback trigger with
            raiserror 24004 "Update failed :Price cannot be null. "
        end
        else
        begin
            update roysched
            set lorange = 0,
            hirange = price * 1000
            from inserted
            where roysched.title_id =  inserted.title_id
        end
       end
end
```

要测试第一条错误消息 35004，请输入：

```
update titles
set title_id = "BU7777"
where title_id = "BU2075"
```

要测试第二条错误消息 35005，请输入：

```
update titles
set pub_id = "7777"
where pub_id = "0877"
```

要测试第三条错误消息 24004，请输入：

```
update titles
set price = 10.00
where title_id = "PC8888"
```

此查询失败，这是因为 titles 中的 price 列为空。如果该列不为空，它就会更新标题 PC8888 的价格，并为 roysched 表进行必要的重新计算。错误 24004 不在 sysusermessages 中，但在此示例中有效。它演示了代码中的 “rollback trigger with raiserror”部分。

更新外键

外键自身的更改或更新可能会导致错误。外键是主键的副本。切勿将二者设计为独立的。若要允许对外键进行更新，应创建一个触发器，使其能检查 master 表的更新，并在更新与主键不匹配时回退更新，以此来保护数据的完整性。

在下面的示例中，触发器将测试两个可能的失败源：title_id 不在 salesdetail 表中，或者它不在 titles 表中。

此示例使用嵌套的 if...else 语句。当 update 语句中 where 子句的值在 salesdetail 中没有匹配值时，第一个 if 语句将为真，即 inserted 表将不包含任何行，且 select 返回一个空值。如果此测试通过，则下一个 if 语句将确定 inserted 表中的新行是否与 titles 表中的任何 title_id 连接。如果任何行均未连接，则回退该事务，并输出错误消息。如果连接成功，则输出不同的消息。

```
create trigger forupdatetrig
on salesdetail
for update
as
declare @row int
/* Save value of rowcount. */
select @row = @@rowcount
if update (title_id)
  begin
    if (select distinct inserted.title_id
        from inserted) is null
      begin
        rollback transaction
        print "No, the old title_id must be in"
        print "salesdetail."
      end
    else
      if (select count(*)
          from titles, inserted
          where titles.title_id =
          inserted.title_id) != @row
        begin
          rollback transaction
          print "No, the new title_id is not in"
          print "titles."
        end
      else
        print "salesdetail table updated"
  end
```

多行注意事项

当触发器的功能是重新计算汇总值或提供即时计数时，多行注意事项尤为重要。

用于保持汇总值的触发器应包含 group by 子句或包含可执行隐式分组的子查询。这样，当插入、更新或删除多个行时，就会创建汇总值。由于 group by 子句会增加额外的开销，因此编写了下面的示例，以测试 *@@rowcount* 是否等于 1，如果是，则表示触发器表中只有一行受到影响。当 *@@rowcount* 等于 1 时，触发器动作即会生效，无需 group by 子句。

使用多行的插入触发器示例

每次添加新的 salesdetail 行时，下面的插入触发器都会更新 titles 表中的 total_sales 列。只要在记录销售量时向 salesdetail 表中添加了行，此触发器就会起作用。它将更新 titles 表中的 total_sales 列，以使 total_sales 等于它以前的值加上添加到 salesdetail.qty 的值。这保证了向 salesdetail.qty 中执行插入后的总数总是最新的。

```
create trigger intrig
on salesdetail
for insert as
    /* check value of @@rowcount */
if @@rowcount = 1
    update titles
      set total_sales = total_sales + qty
      from inserted
      where titles.title_id = inserted.title_id
else
    /* when @@rowcount is greater than 1,
       use a group by clause */
    update titles
      set total_sales =
        total_sales + (select sum(qty)
      from inserted
      group by inserted.title_id
      having titles.title_id = inserted.title_id)
```

使用多行的删除触发器示例

下一个示例为删除触发器，每次删除一个或多个 salesdetail 行时，此触发器都将更新 titles 表中的 total_sales 列。

```
create trigger deltrig
on salesdetail
for delete
as
    /* check value of @@rowcount */
if @@rowcount = 1
    update titles
      set total_sales = total_sales - qty
      from deleted
      where titles.title_id = deleted.title_id
else
    /* when rowcount is greater than 1,
       use a group by clause */
    update titles
      set total_sales =
        total_sales - (select sum(qty)
      from deleted
      group by deleted.title_id
      having titles.title_id = deleted.title_id)
```

只要从 salesdetail 表中删除了行，此触发器就会起作用。它将更新 titles 表中的 total_sales 列，以使 total_sales 等于它以前的值减去从 salesdetail.qty 减掉的值。

使用多行的更新触发器示例

每次更新 salesdetail 行中的 qty 字段时，下面的更新触发器都会更新 titles 表中的 total_sales 列。前面曾提到，更新是先插入再删除的操作。此触发器引用 inserted 和 deleted 触发器测试表。

```
create trigger updtrig
on salesdetail
for update
as
if update (qty)
begin
    /* check value of @@rowcount */
    if @@rowcount = 1
      update titles
          set total_sales = total_sales +
            inserted.qty - deleted.qty
```

```
                from inserted, deleted
                where titles.title_id = inserted.title_id
                and inserted.title_id = deleted.title_id
        else
        /* when rowcount is greater than 1,
           use a group by clause */
        begin
          update titles
                set total_sales = total_sales +
                  (select sum(qty)
                        from inserted
                        group by inserted.title_id
                        having titles.title_id =
                        inserted.title_id)
          update titles
                set total_sales = total_sales -
                  (select sum(qty)
                        from deleted
                        group by deleted.title_id
                        having titles.title_id =
                        deleted.title_id)
        end
    end
```

使用多行的条件插入触发器示例

如果仅因为某些数据修改是无法接受的，则不必回退所有数据修改。在触发器中使用相关子查询能够强制触发器逐行检查已修改的行。有关相关子查询的详细信息，请参见第 186 页的 “使用相关子查询”。触发器可以通过这样对不同的行采取不同的操作。

下面的触发器示例假定存在一个名为 junesales 的表，其 create 语句为：

```
create table junesales
(stor_id    char(4)     not null,
ord_num     varchar(20) not null,
title_id    tid         not null,
qty         smallint    not null,
discount    float       not null)
```

在 junesales 表中插入四行，以测试该条件触发器。在 junesales 中，有两行的 title_id 与 titles 表中已有的任何 title_id 均不匹配。

```
insert junesales values ("7066", "BA27619", "PS1372", 75, 40)
insert junesales values ("7066", "BA27619", "BU7832", 100, 40)
insert junesales values ("7067", "NB-1.242", "PSxxxx", 50, 40)
```

```
insert junesales values ("7131", "PSyyyy", "PSyyyy", 50, 40)
```

将数据从 junesales 插入到 salesdetail 中时，其语句如下：

```
insert salesdetail
select * from junesales
```

触发器 conditionalinsert 将逐行分析插入，并删除 titles 中没有 title_id 的行：

```
create trigger conditionalinsert
on salesdetail
for insert as
if
(select count(*) from titles, inserted
where titles.title_id = inserted.title_id)
    != @@rowcount
begin
  delete salesdetail from salesdetail, inserted
    where salesdetail.title_id = inserted.title_id
    and inserted.title_id not in
    (select title_id from titles)
  print "Only records with matching title_ids
    added."
end
```

此触发器可删除不需要的行。这一删除刚插入的行的功能取决于引发触发器时的处理顺序。首先，在上述表和 inserted 表中插入行；然后引发触发器。

回退触发器

可以使用 rollback trigger 语句或 rollback transaction 语句（如果触发器作为事务的一部分引发）来回退触发器。但是， rollback trigger 仅回退触发器的影响和导致触发器引发的语句；而 rollback transaction 则回退整个事务。例如：

```
begin tran
insert into publishers (pub_id) values ("9999")
insert into publishers (pub_id) values ("9998")
commit tran
```

如果第二个 insert 语句导致 publishers 上的触发器发出 rollback trigger，则只影响第二个 insert ； 不会回退第一个 insert。如果该触发器发出的是 rollback transaction，则两个 insert 语句都会被作为事务的一部分回退。

rollback trigger 的语法为：

```
rollback trigger
   [with raiserror_statement]
```

有关 rollback transaction 的语法，请参见第 22 章 “事务：维护数据一致性和恢复”。

raiserror_statement 语句会输出用户定义的错误消息，并设置系统标志来记录发生了错误这一情况。这样，在执行 rollback trigger 时，该语句便能够向客户端引发错误，从而使错误中包含的事务状态能够反映回退情况。例如：

```
rollback trigger with raiserror 25002
    "title_id does not exist in titles table."
```

有关 raiserror 的详细信息，请参见第 15 章 “使用批处理和控制流语言”。

下面的插入触发器示例与第 647 页的 “触发器插入示例”中描述的 forinsertrig1 触发器执行相似的任务。但在回退插入而不是事务时，此触发器使用 rollback trigger 而不是 rollback transaction 来引发错误。

```
create trigger forinsertrig2
on salesdetail
for insert
as
if (select count(*) from titles, inserted
    where titles.title_id = inserted.title_id) !=
    @@rowcount
   rollback trigger with raiserror 25003
     "Trigger rollback:salesdetail row not added
     because a title_id does not exist in titles."
```

执行 rollback trigger 时， Adaptive Server 将中止当前执行的命令并停止执行触发器的其余部分。如果发出 rollback trigger 的触发器嵌套在其它触发器内， Adaptive Server 将回退在这些触发器中完成的所有工作，一直到导致第一个触发器引发的那个更新操作为止 （包括该操作）。

如果通过批处理来执行包含 rollback transaction 语句的触发器，它们会中止整个批处理。在下面的示例中，如果 insert 语句引发的触发器包含 rollback transaction （如 forinsertrig1），则批处理将会中止，从而不会执行 delete 语句：

```
insert salesdetail values ("7777", "JR123",
    "PS9999", 75, 40)
delete salesdetail where stor_id = "7067"
```

如果包含 rollback transaction 语句的触发器是在**用户定义的事务**中引发的，则 rollback transaction 将回退整个批处理。在下面的示例中，如果 insert 语句引发的触发器包括 rollback transaction，则还会回退 update 语句：

```
begin tran
update stores set payterms = "Net 30"
    where stor_id = "8042"
insert salesdetail values ("7777", "JR123",
    "PS9999", 75, 40)
commit tran
```

有关用户定义的事务的信息，请参见第 22 章 “事务：维护数据一致性和恢复”。

Adaptive Server 将忽略在触发器外执行的 rollback trigger 语句，并且不发出与该语句关联的 raiserror。但在触发器外、事务内执行的 rollback trigger 将生成错误，进而导致 Adaptive Server 回退该事务，并中止当前的语句批处理。

全局登录触发器

可以使用 sp_logintrigger 设置在每个用户登录时执行的全局登录触发器。若要采取用户特定操作，请使用 sp_modifylogin 或 sp_addlogin 设置用户特定登录触发器。

注释 可通过设置跟踪标志 -T4073 来激活此选项。

请参见 《参考手册：过程》，了解 sp_logintrigger 完整的语法和用法信息。

嵌套触发器

触发器可嵌套的深度为 16 级。当前的嵌套级别存储在 *@@nestlevel* 全局变量中。嵌套功能是在安装时启用的。系统管理员可通过 allow nested triggers 配置参数来打开和关闭触发器嵌套功能。

启用嵌套触发器后，如果一个触发器所更改的表上有另一个触发器，则更改表的那个触发器就会引发第二个触发器，第二个触发器又会引发第三个触发器，依此类推。如果链中有任何触发器引发了无限循环，就会超出其嵌套级别，触发器将中止。可以使用嵌套触发器执行有用的管家功能，如存储受前一个触发器影响的行的备份副本。

例如，可以在 titleauthor 上创建一个触发器来保存被 delcascadetrig 触发器删除的 titleauthor 行的备份副本。 delcascadetrig 触发器生效后，如果删除 titles 中的 title_id “PS2091”，则也会从 titleauthor 中删除相对应的所有行。要保存这些数据，可以在 titleauthor 上创建一个 delete 触发器，以便将删除的数据保存到另一个表 del_save 中：

```
create trigger savedel
on titleauthor
for delete
as
insert del_save
select * from deleted
```

Sybase 建议按与顺序有关的序列使用嵌套触发器。使用单独的触发器对数据进行级联修改，如前面第 648 页的 “级联删除的示例”中描述的 delcascadetrig 的示例所述。

注释　如果将触发器放置在事务中，则当任一级别的一组嵌套触发器失败时，都会取消事务并回退所有数据修改。在触发器中使用 print 或 raiserror 语句可确定发生失败的位置。

无论触发器处于哪个嵌套级别，它包含的 rollback transaction 都可回退每个触发器的影响并取消整个事务。 rollback trigger 只影响嵌套触发器和导致初始触发器引发的数据修改语句。

触发器自递归

缺省情况下，触发器不递归调用自身。也就是说，更新触发器不会调用自身来响应在该触发器中对同一个表的二次更新。如果表中某一列上的更新触发器导致对另一列进行更新，则该更新触发器只引发一次。但是，可以通过打开 set 命令的 self_recursion 选项来允许触发器递归调用自身。要实现自递归，还必须启用 allow nested triggers 配置变量。

self_recursion 设置仅在当前客户端会话的持续时间内有效。如果该选项被设置为触发器的一部分，它的作用就会受到对它进行设置的触发器的范围的限制。如果设置了 self_recursion on 的触发器返回或导致另一个触发器引发，则此选项将还原为 off。触发器打开 self_recursion 选项后，如果其自身的动作可导致其再次引发，它就能重复循环，但不能超过 16 级嵌套的限制。

例如，假定 pubs2 中包含下表 new_budget：

```
select * from new_budget

unit            parent_unit     budget
--------------- --------------- -------
one_department  one_division    10
one_division    company_wide    100
company_wide    NULL            1000

(3 rows affected)
```

可创建一个触发器，使其可以在 budget 列每次被更改时递归更新 new_budget，如下所示：

```
create trigger budget_change
on new_budget
for update as
if exists (select * from inserted
            where parent_unit is not null)
begin
    set self_recursion on
    update new_budget
    set new_budget.budget = new_budget.budget +
        inserted.budget - deleted.budget
    from inserted, deleted, new_budget
    where new_budget.unit = inserted.parent_unit
        and new_budget.unit = deleted.parent_unit
end
```

如果通过将 one_department 单位的 budget 值增加 3 来更新 new_budget.budget， Adaptive Server 将执行如下操作 （假定已启用嵌套触发器）：

1　将 one_department 从 10 增加到 13 可引发 budget_change 触发器。

2　该触发器将 one_department 的父项 （此例中为 one_division）的 budget 值从 100 更新为 103，这将再次引发该触发器。

3　该触发器将 one_division 的父项 （此例中为 company_wide）从 1000 更新为 1003，这将第三次引发该触发器。

4　该触发器尝试更新 company_wide 的父项，但由于不存在任何父项（值为 “NULL”），因此最后一个 update 永远不会发生且不会引发触发器，自递归结束。可以通过查询 new_budget 来查看最终结果，如下所示：

```
select * from new_budget

unit            parent_unit     budget
--------------- --------------- -------
one_department  one_division    13
one_division    company_wide    103
company_wide    NULL            1003

(3 rows affected)
```

还可以通过其它方式递归执行触发器。触发器可调用存储过程，该存储过程执行的操作会导致触发器再次引发 （只有在启用嵌套触发器后，触发器才会被重新激活）。除非触发器内有限制递归次数的条件，否则将导致嵌套级别溢出。

例如，如果更新触发器调用一个执行更新操作的存储过程，则在将 nested triggers 设置为 off 时，该触发器和该存储过程将只执行一次。如果将 nested triggers 设置为 on，且触发器或过程中的某一条件要求更新次数超过 16，则循环将继续进行，直至该触发器或过程超过 16 级这个最大嵌套值为止。

与触发器相关的规则

编写触发器时，除了要预见多行数据修改、触发器回退和触发器嵌套的影响外，还需考虑下列因素。

触发器和权限

触发器是在特定表上定义的。只有表的所有者才具有表的 create trigger 和 drop trigger 权限。这些权限不能移交给其他人。

Adaptive Server 能接受对无权执行的操作进行尝试的触发器定义。存在这种触发器时，对触发器表进行修改的任何尝试都会中止，因为不正确的权限会导致该触发器在引发后失败。事务将被取消。必须修改权限或删除该触发器。

例如， Jose 拥有 salesdetail 并在上面创建了触发器。该触发器应在 salesdetail.qty 被更新时更新 titles.total_sales。但 Mary 是 titles 的所有者，且没有向 Jose 授予 titles 的权限。当 Jose 尝试更新 salesdetail 时，Adaptive Server 检测到该触发器并发现 Jose 没有对 titles 的权限，于是将更新事务回退。Jose 必须从 Mary 那里获得 titles.total_sales 的更新权限，或者删除 salesdetail 上的触发器。

触发器限制

Adaptive Server 对触发器施加以下限制：

- 一个表最多可以有三个触发器：一个更新触发器、一个插入触发器和一个删除触发器。
- 每个触发器只能应用于一个表上。但一个触发器可应用于用户的所有三个操作：update、 insert 和 delete。
- 虽然触发器可以引用视图或临时表，您也不能在视图或特定于会话的临时表上创建触发器。
- writetext 语句不能激活插入触发器或更新触发器。
- 虽然 truncate table 语句由于删除所有行而在实际上与不含 where 子句的 delete 相似，但它却不能引发触发器，因为未记录各个行的删除操作。
- 不能在临时对象 (@object) 上创建触发器或建立索引或视图

- 不能在系统表上创建触发器。如果尝试在系统表上创建触发器，Adaptive Server 将返回错误消息并取消该触发器。
- 不能使用从 inserted 或 deleted 表的 text 列或 image 列中选择的触发器。
- 如果启用了 “组件集成服务”，则在代理表上使用触发器将受到限制，原因是无法对正在插入、更新或删除的行 （通过 inserted 和 deleted 表）进行检查。可在代理表上创建能被调用的触发器。但是，由于 insert 被传递到远程服务器，因此已删除或插入的数据不会被写入到代理表的事务日志中。因而， inserted 和 deleted 表 （它们实际上是事务日志的视图）不包含代理表的数据。

隐式和显式空值

只要在选择列表或在 values 子句中为列赋值，对于 insert 语句来说， if update(*column_name*) 子句即为真。如果*显式*空值或缺省值为列赋值，则将激活触发器。而*隐式*空值则不会如此。

例如，假设您创建了下表：

```
create table junk
(a int null,
 b int not null)
```

然后编写下面的触发器：

```
create trigger junktrig
on junk
for insert
as
if update(a) and update(b)
        print "FIRING"

    /*"if update" is true for both columns.
      The trigger is activated.*/
insert junk (a, b) values (1, 2)

    /*"if update" is true for both columns.
      The trigger is activated.*/
insert junk values (1, 2)

    /*Explicit NULL:
      "if update" is true for both columns.
      The trigger is activated.*/
insert junk values (NULL, 2)
```

```
    /* If default exists on column a,
      "if update" is true for either column.
      The trigger is activated.*/
insert junk (b) values (2)

    /* If no default exists on column a,
      "if update" is not true for column a.
      The trigger is not activated.*/
insert junk (b) values (2)
```

仅使用以下子句便可产生同样的结果：

```
if update(a)
```

要创建禁止插入隐式空值的触发器，可使用：

```
if update(a) or update(b)
```

然后触发器中的 SQL 语句进行测试，以确定 *a* 或 *b* 是否为空。

触发器和性能

就性能而言，触发器的开销通常很低。运行触发器所需的时间大多用于引用内存或数据库设备中的其它表。

deleted 和 inserted 触发器测试表始终位于活动内存中。触发器所引用的其它表的位置将决定操作所需的时间。

触发器中的 *set* 命令

可以在触发器内使用 set 命令。在触发器执行期间，调用的 set 选项一直有效。执行完毕后，触发器将还原为其原来的设置。

重命名和触发器

要更改触发器所引用的对象的名称，必须先删除该触发器，然后重新创建它，这样才能使触发器的源文本反映所引用对象的新名称。使用 sp_depends 可获取触发器所引用的对象的报告。最安全的操作方法是不对触发器引用的任何表或视图进行重命名。

触发器提示

创建触发器时，应考虑以下提示：

- 假设您的 insert 或 update 触发器可调用某个存储过程，而该存储过程又会更新基表。如果将 nested triggers 配置参数设置为真，该触发器将进入无限循环。在执行 insert 或 update 触发器之前，应将 sp_configure “nested triggers” 设置为 false。
- 执行 drop table 时，任何与该表相关的触发器都将被删除。要保留任何此类触发器，可在删除该表之前使用 sp_rename 更改它们的名称。
- 使用 sp_depends 可查看触发器中所引用的表和视图的报告。
- 使用 sp_rename 可重命名触发器。
- 对于每个查询（单个数据修改语句，例如 insert 或 update），触发器只引发一次。如果查询在循环中重复，则触发器的引发次数将与查询的重复次数相同。

禁用触发器

insert、 update 和 delete 命令通常会引发它们所遇到的任何触发器，这会增加执行操作所需的时间。要在执行批量 insert、update 或 delete 操作期间禁用触发器，可以使用 alter table 命令的 disable trigger 选项。使用 disable trigger 选项可禁用与表关联的所有触发器，也可以指定要禁用的特定触发器。但在复制结束之后，已禁用的所有触发器都将被引发。

为尽快装载数据， bcp 不会引发规则和触发器。

要查找违反规则和触发器的所有行，可将数据复制到表中，然后运行用于测试规则或触发器条件的查询或存储过程。

alter table... disable trigger 使用以下语法：

```
alter table [database_name.[owner_name].]table_name
    {enable | disable } trigger [trigger_name]
```

其中， *table_name* 是要禁用触发器的表的名称， *trigger_name* 是要禁用的触发器的名称。例如，要禁用 pubs2 数据库中的 del_pub 触发器，可使用以下命令：

```
alter table pubs2
disable del_pubs
```

如果未指定触发器， alter table 会禁用表中定义的所有触发器。

装载完数据库后，可使用 alter table... enable trigger 重新启用触发器。要重新启用 del_pub 触发器，应发出：

```
alter table pubs2
enable del_pubs
```

注释 只有表所有者或数据库管理员才能使用禁用触发器功能。

如果触发器包含任何插入、更新或删除语句，则在禁用该触发器后，这些语句不会运行，这可能会影响表的参照完整性。

删除触发器

可以直接删除触发器，也可以通过删除与其关联的触发器表来删除触发器。

drop trigger 的语法如下：

```
drop trigger [owner.]trigger_name
    [, [owner.]trigger_name]...
```

删除表时， Adaptive Server 将删除与其相关的任何触发器。 drop trigger 权限缺省为触发器表的所有者，此权限不可转让。

获取有关触发器的信息

作为数据库对象，触发器按名称列在 sysobjects 中。 sysobjects 的 type 列用缩写形式 “TR”来标识触发器。以下查询可查找存在于数据库中的触发器：

```
select *
from sysobjects
where type = "TR"
```

每个触发器的源文本都存储在 syscomments 中。触发器的执行计划存储在 sysprocedures 中。以下几节所介绍的系统过程提供了系统表中有关触发器的信息。

sp_help

可以使用 sp_help 获取有关触发器的报告。例如，可以按以下方式获取有关 deltitle 的信息：

```
sp_help deltitle

Name         Owner   Type
------------ ------- -----------
deltitle     dbo     trigger
Data_located_on_segment  When_created
-----------------------  -----------------
not applicable           Jul 10 1997 3:56PM

(return status = 0)
```

还可以使用 sp_help 来报告触发器的状态 （已禁用或已启用）：

```
1> sp_help trig_insert
2> go
Name Owner
Type
-----------------------------------
trig_insert dbo
trigger
(1 row affected)
data_located_on_segment When_created
-----------------------------------
not applicable Aug 30 1998 11:40PM
Trigger enabled
(return status = 0)
```

sp_helptext

要显示触发器的源文本，可执行 sp_helptext，如下所示：

```
sp_helptext deltitle

# Lines of Text
---------------
              1
text
--------------------------------------------
create trigger deltitle
on titles
for delete
as
```

```
if (select count(*) from deleted, salesdetail
where salesdetail.title_id = deleted.title_id) >0
begin
      rollback transaction
      print "You can't delete a title with sales."
end
```

如果触发器的源文本已用 sp_hidetext 加密， Adaptive Server 会显示一条消息，提示用户该文本已被隐藏。请参见 《参考手册：过程》。

如果系统安全员已经用 sp_configure 重新设置了 allow select on syscomments.text column 参数 （这是在已评估的配置中运行 Adaptive Server 所必需的操作），则只有触发器的创建者或系统管理员才能通过 sp_helptext 查看触发器的源文本。（请参见 《系统管理指南：卷 1》中的第 12 章 “安全性简介”。

sp_depends

sp_depends 可列出引用了对象的触发器，或列出触发器所影响的所有表或视图。以下示例将说明如何使用 sp_depends 来获取 deltitle 触发器引用的所有对象的列表：

```
sp_depends deltitle

Things the object references in the current database.
object              type       updated  selected
----------------    ----------  ------- --------
dbo.salesdetail     user table  no      no
dbo.titles          user table  no      no
(return status = 0)
```

以下语句列出引用 salesdetail 表的所有对象：

```
sp_depends salesdetail
Things inside the current database that reference the
object.
object                          type
--------------------------  ----------------
dbo.deltitle                trigger
dbo.history_proc            stored procedure
dbo.insert_salesdetail_proc  stored procedure
dbo.totalsales_trig         trigger
(return status = 0)
```

第 21 章

使用 *instead of* 触发器

视图通常用于将逻辑数据库模式与物理模式分开。本章讨论可以在视图中定义的功能，以替换 update、 insert 或 delete 语句的标准操作。使用 instead of 触发器，可使所有视图 （包括那些通过其它方式不可更新的视图）得到更新。

概述

instead of 触发器是一些特殊的存储过程，可覆盖触发语句 （insert、update 和 delete）的缺省操作，并执行用户定义的操作。

就像 for 触发器一样，每次对特定视图执行数据修改语句时，就会执行 instead of 触发器。由于 for 触发器是在对表执行 insert/update/delete 语句后引发的，因此有时也称为 after 触发器。单个 instead of 触发器可应用于一个特定的触发操作：

```
instead of update
```

它也可以应用于多个操作，同一触发器可执行其中列出的所有操作：

```
instead of insert,update,delete
```

就像 for 触发器一样， instead of 触发器也可以在触发器处于活动状态时使用逻辑 inserted 表和 deleted 表存储修改的记录。这些表中的每一列都可以直接映射到触发器中所引用的基视图中的列。例如，如果名为 V1 的视图包含名为 C1、C2、C3 和 C4 的列，则 inserted 表和 deleted 表将包含所有这四个列的值，即使触发器只修改列 C1 和 C3 也是如此。Adaptive Server 会以内存驻留对象的形式自动创建和管理 inserted 表及 deleted 表。

instead of 触发器允许视图支持更新，并允许实现代码逻辑，该逻辑需要拒绝批处理的某些部分执行，而允许其它部分成功执行。

instead of 触发器在执行每个数据修改语句后仅引发一次。包含 while 循环的复杂查询可能多次重复执行 update 或 insert 语句，且每次都会引发 instead of 触发器。

Inserted 逻辑表和 deleted 逻辑表

deleted 和 inserted 表都是逻辑 （概念）表。 inserted 表是一种伪表，其行中含有 insert 语句的插入值； deleted 表也是一种伪表，其行中含有 update 语句的更新值 (after image)。 inserted 表和 deleted 表的模式与为其定义 instead of 触发器的视图 （即试图对其执行用户操作的视图）的模式相同。这两个表和该视图之间的区别在于 inserted 表和 deleted 表的所有列均可为空，但对应的视图列不能为空。例如，如果视图中包含数据类型为 char 的列，且 insert 语句提供了要插入的 char 值，则 inserted 表中对应列的数据类型为 varchar，且输入值将转换为 varchar。但是，在将该值添加到 inserted 表中时，不会截断值中的尾随空白。

如果指定值的数据类型与插入该值的对应列的数据类型不同，则会在内部将该值转换为此列的数据类型。如果转换成功，则会将转换的值插入表中，但如果转换失败，将中止该语句。在本例中，如果视图从表中选择一个整数列：

```
CREATE VIEW v1 AS SELECT intcol FROM t1
```

下面的 insert 语句会导致在 v1 上执行 instead of 触发器，因为值 1.0 可以成功转换为整数值 1：

```
INSERT INTO v1 VALUES (1.0)
```

但是，下一语句会导致出现例外，且在可以执行 instead of 触发器之前，该语句即被中止：

```
INSERT INTO v1 VALUES (1.1)
```

触发器可对 deleted 表和 inserted 表进行检查，以确定是否应执行触发器操作以及如何执行该操作才不会更改表本身。

deleted 表可与 delete 和 update 一起使用；而 inserted 表可与 insert 和 update 一起使用。

注释　instead of 触发器可将 inserted 表和 deleted 表创建为内存中的表或工作表。而 for 触发器可通过读取事务日志 syslogs 随即生成 inserted 表和 deleted 表的行。

LOB 数据类型

inserted 表或 deleted 表中数据类型为大对象 (LOB) 的列的各个值都存储在内存中，如果这两个表中包含很多行，并且这些行都包含较大的 LOB 值，则会占用大量内存。

触发器和事务

rollback trigger 和 rollback transaction 命令均用于 instead of 触发器。rollback trigger 可回退在由触发语句引发的任意或全部嵌套 instead of 触发器中执行的操作。 rollback transaction 可将在整个事务中执行的操作回退到最近的 savepoint。

嵌套和递归

就像 for 触发器一样， instead of 触发器也可以嵌套至 16 层。当前的嵌套级别存储在 *@@nestlevel* 中。缺省情况下，嵌套功能处于启用状态。系统管理员可使用配置参数 allow nested triggers 来更改触发器嵌套功能的 on 和 off 状态。如果已启用嵌套触发器，则在一个触发器所更改的表包含另一个触发器的情况下，第一个触发器会执行第二个触发器，而后者又可执行另一个触发器，依此类推，从而形成无限循环。在这种情况下，当超过嵌套层数时，即会结束处理并中止触发器。无论触发器处于哪个嵌套级别，它包含的 rollback transaction 都可回退每个触发器的影响并取消整个事务。 rollback trigger 只影响嵌套触发器和导致初始触发器执行的数据修改语句。

您可以交错嵌套 instead of 和 for 触发器。例如，对具有 instead of update 触发器的视图执行 update 语句会导致执行该触发器。如果触发器包含的 SQL 语句可更新其中已定义 for 触发器的表，则会引发该触发器。 for 触发器可以包含可更新另一个视图的 SQL 语句，该视图具有稍后执行的 instead of 触发器，依此类推。

递归

然而， instead of 和 for 触发器具有不同的递归行为。 for 触发器支持递归，而 instead of 触发器不支持递归。如果 instead of 触发器引用引发了该触发器的同一视图，则不会递归调用该触发器。相反，触发语句会直接应用于该视图；换句话说，就是将该语句解析为对该视图底层的基表的修改。在这种情况下，视图定义必须满足可更新视图的所有限制。如果该视图不可更新，则会产生错误。

例如，如果将触发器定义为某个视图的 instead of update 触发器，则在 instead of 触发器中对该视图执行 update 语句不会导致该触发器再次执行。该触发器会对视图执行 update 语句，就好像视图不具有 instead of 触发器一样。必须将 update 更改的列解析到单个基表。

instead of insert 触发器

可以在视图上定义 instead of insert 触发器来替换 insert 语句的标准操作。通常，会在视图上定义该触发器，以将数据插入一个或多个基表中。

视图 select 列表中的列可以为空，也可以不为空。如果 view 列不允许包含空值，则可将行插入视图中的 SQL 语句必须为该列提供值。除有效的非空值外，视图的非空列还接受显式空值。如果定义 view 列的表达式包含下列项，则 view 列允许包含空值：

- 对允许包含空值的任何基表列的引用
- 算术运算符
- 对函数的引用
- 具有可为空值的子表达式的 CASE
- NULLIF

存储过程 sp_help 可报告允许包含空值的 view 列。

引用具有 instead of insert 触发器的视图的 insert 语句必须为每个不允许包含空值的 view 列提供值。其中包括引用基表中未指定输入值的列的 view 列，例如：

- 基表中的计算列
- 基表中 identity insert 为 OFF 的标识列。

如果 instead of insert 触发器包含的针对基表执行的 insert 语句使用 inserted 表中的数据，则 insert 语句必须通过排除该语句 select 列表中的列来忽略这些类型列的值。对视图执行的 insert 语句可以为这些列生成虚值，但 instead of insert 触发器中的 insert 语句可以忽略这些值，而 Adaptive Server 将提供正确的值。

示例

insert 语句必须为映射到基表中标识列或计算列的 view 列指定值。但是，它可以提供占位符值。将对 instead of 触发器中将值插入基表的 insert 语句进行编写，以忽略所提供的值。

例如，下列语句将创建说明该过程的表、视图和触发器：

```
CREATE TABLE BaseTable
(PrimaryKey         int IDENTITY
Color               varchar (10) NOT NULL,
Material            varchar (10) NOT NULL,
TranTime            timestamp
)
----------------------
--Create a view that contains all columns from the base
table.
CREATE VIEW  InsteadView
AS SELECT PrimaryKey, Color, Material, TranTime
FROM BaseTable
----------------------
Create an INSTEAD OF INSERT trigger on the view.
CREATE TRIGGER InsteadTrigger on InsteadView
INSTEAD OF INSERT
AS
BEGIN
    --Build an INSERT statement ignoring
    --inserted.PrimaryKey and
    --inserted.TranTime.
INSERT INTO BaseTable
    SELECT Color, Material
    FROM inserted
END
```

直接引用 BaseTable 的 insert 语句无法为 PrimaryKey 和 TranTime 列提供值。例如：

```
--A correct INSERT statement that skips the PrimaryKey
--and TranTime columns.
INSERT INTO BaseTable (Color, Material)
            VALUES ('Red', 'Cloth')

--View the results of the INSERT statement.
SELECT PrimaryKey, Color, Material, TranTime
FROM BaseTable

--An incorrect statement that tries to supply a value
--for the PrimaryKey and TranTime columns.
INSERT INTO BaseTable
    VALUES (2, 'Green', 'Wood', 0x0102)

INSERT statements that refer to InsteadView, however,
    must supply a value for PrimaryKey:

--A correct INSERT statement supplying a dummy value for
--the PrimaryKey column.A value for TranTime is not
--required because it is a nullable column.
INSERT INTO InsteadView (PrimaryKey, Color, Material)
        VALUES (999, 'Blue', 'Plastic')
--View the results of the INSERT statement.
SELECT PrimaryKey, Color, Material, TranTime
FROM InsteadView
```

传递到 InsteadTrigger 的 inserted 表是使用不可为空的 PrimaryKey 列创建的，因此引用该视图的 insert 语句必须为该列提供值。值 999 将传递到 InsteadTrigger，但是 InsteadTrigger 中的 insert 语句未选择 inserted.PrimaryKey，因此将忽略该值。实际插入 BaseTable 中的行在 PrimaryKey 中的值为 2，而在 TranTime 中为 Adaptive Server 生成的 timestamp 值。

如果在具有 instead of insert 触发器的视图中引用具有 default 定义的 not null 列，则引用该视图的所有 insert 语句都必须为该列提供值。在创建传递到触发器的 inserted 表时，此值是必需的。必须具有相应的值约定，以通知触发器应使用缺省值。可能的约定是在 insert 语句中为 not null 列提供显式空值。在将显式空值插入已在其中定义了视图的表时，instead of insert 触发器会忽略该显式空值，从而将缺省值插入该表中。例如：

```
--Create a base table with a not null column that has
--a default
CREATE TABLE td1 (coll int DEFAULT 9 NOT NULL, col2 int)
```

```
--Create a view that contains all of the columns of
--the base table.
CREATE VIEW vtd1 as select * from td1
--create an instead of trigger on the view
CREATE TRIGGER vtd1insert on vtd1 INSTEAD OF INSERT AS
BEGIN
    --Build an INSERT statement that inserts all rows
    --from the inserted table that have a NOT NULL value
    --for col1.
    INSERT INTO td1 (col1,col2) SELECT * FROM inserted
    WHERE col != null

    --Build an INSERT statement that inserts just the
    --value of col2 from inserted for those rows that
    --have NULL as the value for col1 in inserted.In
    --this case, the default value of col1 will be
    --inserted.
    INSERT INTO td1 (col2) SELECT col2 FROM inserted
    WHERE col1 = null
END
```

instead of insert 触发器中的 deleted 表始终为空。

instead of update 触发器

通常，可在视图上定义 instead of update 触发器以修改一个或多个基表中的数据。

在引用具有 instead of update 触发器的视图的 update 语句中，视图中列的任何子集都可以出现在 update 语句的 set 子句中，无论这些列是非空列还是空列。

即使不可更新的视图列 （这些列引用基表中未指定输入值的列）也可以出现在 update 语句的 set 子句中。不可更新的列包括：

- 基表中的计算列
- 基表中 identity insert 设置为 off 的标识列
- timestamp 数据类型的基表列。

通常，当引用表的 update 语句试图设置计算列、 identity 列或 timestamp 列的值时会出现错误，因为 Adaptive Server 必须生成这些列的值。但是，如果 update 语句引用具有 instead of update 触发器的视图，则该触发器中所定义的逻辑能够绕过这些列并避免出现错误。为此， instead of update 触发器不能尝试更新基表中相应列的值。在触发器的定义中，将这些列从 update 语句的 set 子句中排除即可达到此目的。

因为 instead of update 触发器不必处理不可更新的插入列中的数据，因此该解决方案行之有效。 inserted 表包含未在 set 子句中指定的列的值，而这些列在发出 update 语句前就已存在。该触发器可以使用 if update (column) 子句测试特定列是否已更新。

instead of update 触发器应使用仅在 where 子句搜索条件下，为计算列、identity 列或 timestamp 列提供的值。视图上 instead of update 触发器用于处理计算列、 identity 列、 timestamp 列或缺省列的更新值的逻辑与应用于这些列类型的插入值的逻辑相同。

在对具有 instead of update 触发器的视图执行 update 语句时，对于限定为要更新的每一行， inserted 表和 deleted 表均包含一行。inserted 表中的行包含更新操作完成后列中的值，而 deleted 表中的行包含更新操作开始前列中的值。

instead of delete 触发器

instead of delete 触发器可以替换 delete 语句的标准操作。通常，可在视图上定义 instead of delete 触发器以修改基表中的数据。

delete 语句不指定对现有数据值的修改，而只指定要删除的行。 delete 表是一种伪表，其中包含的行具有 delete 语句的删除值，或 update 语句的更新前的值 (before image)。发送给 instead of delete 触发器的 deleted 表包含在发出 delete 语句前就存在的行的映像。 deleted 表的格式以为视图定义的 select 列表的格式为基础，可以转换为可空列的所有 not null 列除外。

传递给 delete 触发器的 inserted 表始终为空。

searched 与 positioned *update* 和 *delete*

update 和 delete 语句可以是 searched 或 positioned 命令。searched delete 是一个在 where 子句中包含可选谓词表达式的 SQL delete 语句，用于限定要删除的行。 searched update 是一个在 where 子句中包含可选谓词表达式的 SQL update 语句，用于限定要更新的行。

下面是一个 searched delete 语句示例：

```
DELETE myview WHERE myview.col1 > 5
```

此语句的执行方式为：检查 myview 中的所有行，并应用在 where 子句中指定的谓词 (myview.col1 > 5) 以确定应删除的行。

在 searched update 和 delete 语句中不允许使用连接。若要使用另一个表中的行来查找视图的限定行，请使用子查询。例如，不允许使用以下语句：

```
DELETE myview FROM myview, mytab
    where myview.col1 = mytab.col1
```

但允许使用包含子查询的等同语句：

```
DELETE myview WHERE col1 in (SELECT col1 FROM mytab)
```

仅对游标的结果集执行 Positioned update 和 delete 语句，且这些语句只影响单个行。例如：

```
DECLARE mycursor CURSOR FOR SELECT * FROM myview
OPEN mycursor
FETCH mycursor
DELETE myview WHERE CURRENT OF mycursor
```

positioned delete 语句仅会删除 myview 上当前已定位 mycursor 的行。

如果视图上包含 instead of 触发器，则始终会对 searched delete 或 update 限定语句（即没有连接的语句）执行该触发器。对于对 positioned delete 或 update 语句执行的 instead of 触发器，必须满足下列两个条件：

- 声明游标时 （即执行 declare cursor 命令时）， instead of 触发器必须存在。
- 定义游标的 select 语句只能访问视图，例如虽然 select 语句不包含连接，但它可以访问视图列的任何子集。

在对滚动游标执行 positioned delete 或 update 语句时，也会执行 instead of 触发器。但在使用**客户端游标**和 set cursor rows 命令时，不会引发 instead of 触发器。

客户端游标

可在应用程序中使用 Open Client™ Library 函数声明和获取客户端游标，以便处理游标。 Client Library 函数可以使用一个 fetch 命令从 Adaptive Server 中检索多个行并将这些行放入缓冲区，以便在执行后续 fetch 命令时一次返回一行给应用程序，而无需在读取完所有缓冲行之前，从 Adaptive Server 检索其它行。缺省情况下， Adaptive Server 会针对它所接收的每个 fetch 指令，将单个行返回给 Open Client Library 函数。但是， set cursor rows 命令可以更改 Adaptive Server 返回的行数。

对于对客户端游标执行的 Positioned update 和 delete 语句，set cursor rows 未能用于增加每个 fetch 返回的行数，从而导致执行 instead of 触发器。但是，如果 set cursor rows 增加每个 fetch 命令返回的行数，则仅在内部处理 declare cursor 期间未将游标标记为只读时才会执行 instead of 触发器。例如：

```
--Create a view that is read-only (without an instead
--of trigger) because it uses DISTINCT.
CREATE VIEW myview AS
SELECT DISTINCT (col1) FROM tab1

--Create an INSTEAD OF DELETE trigger on the view.
CREATE TRIGGER mydeltrig ON myview
INSTEAD OF DELETE
AS
BEGIN
    DELETE tab1 WHERE col1 in (SELECT col1 FROM deleted)
END

Declare a cursor to read the rows of the view
DECLARE cursor1 CURSOR FOR SELECT * FROM myview

OPEN cursor1

FETCH cursor1

--The following positioned DELETE statement will
--cause the INSTEAD OF TRIGGER, mydeltrig, to fire.
DELETE myview WHERE CURRENT OF cursor1

--Change the number of rows returned by ASE for
--each FETCH.
SET CURSOR ROWS 10 FOR cursor1

FETCH cursor1

--The following positioned DELETE will generate an
--exception with error 7732:"The UPDATE/DELETE WHERE
```

```
--CURRENT OF failed for cursor 'cursor1' because
--the cursor is read-only."
DELETE myview WHERE CURRENT OF cursor1
```

通过使用 set cursor rows，可以在 Adaptive Server 中的游标位置和应用程序中的游标位置之间创建断开的连接：Adaptive Server 定位在返回的 10 行的最后一行，但应用程序可以定位在这 10 行中的任意一行，因为 Open Client Library 函数可以在不向 Adaptive Server 发送信息的情况下将这些行放入缓冲区并滚动这些行。因为在应用程序发送 positioned delete 语句时， Adaptive Server 无法确定 cursor1 的位置，所以 Open Client Library 函数还会发送行 （cursor1 在应用程序中定位到该行）的列的子集值。这些值用于将 positioned delete 转换为 searched delete 语句。这意味着如果 col1 的值在 cursor1 的当前行中为 5，则 Adaptive Server 将使用诸如 'where col1 = 5' 的子句来查找该行。

在将 positioned delete 转换为 searched delete 时，游标必须是可更新的，就像不具有 instead of 触发器的表和视图上的游标一样。在上一个示例中，定义 cursor1 的 select 语句替换为定义 myview 的 select 语句：

```
DECLARE cursor1 CURSOR FOR SELECT * FROM myview
```

变成：

```
DECLARE cursor1 CURSOR FOR SELECT DISTINCT (col1)
FROM tab1
```

由于 select 列表中包含 distinct 选项，cursor1 不可更新，即该游标是只读的。这会导致处理 positioned delete 时出现 7732 错误。

如果因将视图替换为其定义 select 语句而产生的游标可更新，则无论是否使用 set cursor rows，都会引发 instead of 触发器。在 Adaptive Server 支持的其它类型的游标中，使用 set cursor rows 不会阻止引发 instead of 触发器。

获取有关触发器的信息

instead of 触发器信息的存储方式与 for 触发器信息的存储方式相同。

- 触发器的定义查询树存储在 sysprocedures 中。
- 每个触发器都有一个标识号 （对象 ID），它存储在 sysobjects 的新行中。应用触发器的视图的对象 ID 存储在触发器 sysobjects 行的 deltrig 列、 instrig 列和 updtrig 列中。触发器的对象 ID 存储在应用触发器的视图的 sysobjects 行的 deltrig 列、 instrig 列或 putrid 列中。
- 若要显示存储在 syscomments 中的触发器文本，请使用 sp_helptext。如果系统安全员已经使用 sp_configure 重新设置了列参数 allow select on syscomments.text，那么只有触发器的创建者或系统管理员才能通过 sp_helptext 查看触发器的文本。
- 使用 sp_help 可获取有关触发器的报告。 sp_help 会将 instead of 触发器报告为 instead of 触发器的 *object_type*。
- 使用 sp_depends 可以报告由 instead of 触发器引用的视图。sp_depends *view_name* 会将触发器名称及其类型报告为 instead of 触发器。

第 22 章

事务：维护数据一致性和恢复

事务将一组 Transact-SQL 语句进行组合，以便将其作为一个单元处理。组中所有的语句要么都执行，要么都不执行。

事务的工作方式

Adaptive Server 自动将所有数据修改命令 （包括单步更改请求）作为事务进行管理。缺省情况下，每个 insert、 update 和 delete 语句均视为单个事务。

但是，让我们考虑一下下面这种情况：Lee 必须对 authors、 titles 和 titleauthors 表进行一系列的数据检索和修改操作。在 Lee 执行操作时， Lil 开始更新 titles 表。Lil 的更新可能会导致与 Lee 所执行的操作产生不一致的结果。要防止发生这种情况， Lee 可以将相应的语句组合成一个事务，锁定她正在表中操作的部分，使 Lil 无法更新这部分内容。这样就可以让 Lee 基于准确的数据完成工作。 Lee 完成对表的更新之后， Lil 才能对表进行更新。

可以使用以下命令来创建事务：

- begin transaction — 标记事务块的起点。语法为：

  ```
  begin {transaction | tran} [transaction_name]
  ```

  transaction_name 是分配给事务的名称。它必须符合标识符的规则。仅在最外面的一对嵌套的 begin/commit 或 begin/rollback 语句中使用事务名。

- save transaction — 标记事务中的保存点：

  ```
  save {transaction | tran} savepoint_name
  ```

  savepoint_name 是分配给保存点的名称。它必须符合标识符的规则。

- commit — 提交整个事务：

  ```
  commit [transaction | tran | work]
  [transaction_name]
  ```

- rollback — 将事务回退到保存点或事务的起点：

  ```
  rollback [transaction | tran | work]
  [transaction_name | savepoint_name]
  ```

例如，Lee 开始为 *The Gourmet Microwave* 的两个作者更改版税分配。由于两次更新之间的数据库可能不一致，因此必须将两次更新组合为一个事务，如下面的示例所示：

```
begin transaction royalty_change

update titleauthor
set royaltyper = 65
from titleauthor, titles
where royaltyper = 75
and titleauthor.title_id = titles.title_id
and title = "The Gourmet Microwave"

update titleauthor
set royaltyper = 35
from titleauthor, titles
where royaltyper = 25
and titleauthor.title_id = titles.title_id
and title = "The Gourmet Microwave"

save transaction percentchanged

/* After updating the royaltyper entries for
** the two authors, insert the savepoint
** percentchanged, then determine how a 10%
** increase in the book's price would affect
** the authors' royalty earnings. */

update titles
set price = price * 1.1
where title = "The Gourmet Microwave"

select (price * total_sales) * royaltyper
```

```
from titles, titleauthor
where title = "The Gourmet Microwave"
and titles.title_id = titleauthor.title_id

/* The transaction is rolled back to the savepoint
** with the rollback transaction command. */

rollback transaction percentchanged

commit transaction
```

事务允许 Adaptive Server 保证：

- 一致性 — 同时进行的查询和更改请求不能相互冲突，并且用户永远不会看到或操作部分更改的数据。
- 恢复 — 在系统出现故障时，自动完成数据库恢复。

为了支持符合 SQL 标准的事务，Adaptive Server 允许为事务选择模式和隔离级别。要求符合 SQL 标准的事务的应用程序应在每个会话开始时设置这些选项。有关详细信息，请参见第 695 页的 “选择事务模式和隔离级别”。

事务和一致性

在多用户环境中， Adaptive Server 必须防止同时进行的查询和数据修改请求相互干涉。这一点十分重要，原因在于：如果查询处理的数据能够被另一用户的更新所更改，则查询结果可能不明确。

Adaptive Server 自动为每个事务设置相应的锁定级别。通过在 select 语句中包括 holdlock 关键字，可以逐查询地加强共享锁的限制性。

事务和恢复

事务既是工作单元又是恢复单元。事实上， Adaptive Server 将单步更改请求作为事务来处理，这意味着数据库可在出现故障时完全恢复。

Adaptive Server 恢复时间以分钟和秒计算。可以指定可接受的最大恢复时间。

将在第 712 页的 “事务的备份和恢复”中讨论与恢复和备份有关的 SQL 命令。

注释 将大量 Transact-SQL 命令组合为一个长期运行的事务可能会影响恢复时间。如果 Adaptive Server 在提交事务之前失败，则 Adaptive Server 必须撤消该事务，因此将花费更长的时间来恢复。

如果使用具有组件集成服务的远程数据库，则处理事务的方式有所不同。请参见 《组件集成服务用户指南》。

如果已经购买并安装了 Adaptive Server DTM 功能，则在多个服务器中更新数据的事务也将受益于事务一致性。请参见 Using Adaptive Server Distributed Transaction Features （《使用 Adaptive Server 分布式事务功能》）。

使用事务

begin transaction 和 commit transaction 命令指示 Adaptive Server 将任意数量的单独命令作为单个单元进行处理。 rollback transaction 撤消事务，并回退到起点或**保存点**。使用 save transaction 可在事务中定义一个保存点。

事务使您可以控制事务管理。另外，由于将 SQL 语句组合为一个单元进行处理，每个事务而非每个单独的命令导致每次系统开销，从而提高了性能。

任何用户都可以定义事务。任何事务命令都没有权限要求。

以下各节通过示例讨论了一般事务主题和事务命令。

在事务中允许使用数据定义命令

通过将 ddl in tran 数据库选项设置为 true，可以在事务中使用某些数据定义语言命令。如果在某个特定的数据库中 ddl in tran 为 true，则在该数据库的事务中可以发出 create table、 grant 和 alter table 等命令。如果在 model 数据库中 ddl in tran 为 true，则对于在 model 中将 ddl in tran 设置为 true 之后所创建的所有数据库而言，允许在其事务中发出上述命令。要检查 ddl in tran 的当前设置，请使用 sp_helpdb。

警告！请谨慎使用数据定义命令。在事务中使用数据定义语言命令的唯一合理情况是在执行 create schema 命令时。数据定义语言命令将锁定系统表，如 sysobjects。如果要在事务中使用数据定义语言命令，则应使事务保持简短。

在事务中应避免对 tempdb 使用数据定义语言命令，否则可能导致系统性能降低甚至停止。在 tempdb 中应始终将 ddl in tran 设置为 false。

要将 ddl in tran 设置为 true，请输入：

```
sp_dboption database_name,"ddl in tran", true
```

然后在该数据库中执行 checkpoint 命令。

第一个参数指定要在其中设置选项的数据库的名称。执行 sp_dboption 时，必须使用 master 数据库。任何用户都可以执行不带参数的 sp_dboption，以显示当前选项设置。但是，要设置选项，您必须是系统管理员或数据库所有者。

只有 sp_dboption 的 ddl in tran 选项设置为 true 时，才允许在事务中使用以下命令：

- create default
- create index
- create procedure
- create rule
- create schema
- create table
- create trigger
- create view
- drop default
- drop index

- drop procedure
- drop rule
- drop table
- drop trigger
- drop view
- grant
- revoke

不能在事务中使用更改 master 数据库或创建临时表的系统过程。

不要在事务中使用以下命令：

- alter database
- alter table...partition
- alter table...unpartition
- create database
- disk init
- dump database
- dump transaction
- drop database
- load transaction
- load database
- reconfigure
- select into
- update statistics
- truncate table

不允许在事务中使用的系统过程

不能在事务中使用以下系统过程：

- sp_helpdb、 sp_helpdevice、 sp_helpindex、 sp_helpjoins、 sp_helpserver、 sp_lookup 和 sp_spaceused （因为它们会创建临时表）
- sp_configure
- 更改 master 数据库的系统过程

开始和提交事务

begin transaction 和 commit transaction 命令可以包含任意数量的 SQL 语句和存储过程。这两个语句的语法为：

begin {transaction | tran} [*transaction_name*]

commit {transaction | tran | work} [*transaction_name*]

transaction_name 是分配给事务的名称。它必须符合标识符的规则。

关键字 transaction、 tran 和 work （在 commit transaction 中）具有相同含义；在使用时可以相互替换。但是， transaction 和 tran 是 Transact-SQL 扩展；只有 work 符合 SQL 标准。

以下是一个简例：

```
begin tran
    statement
    procedure
    statement
commit tran
```

如果事务当前不处于活动状态，则 commit transaction 不会影响 Adaptive Server。

回退和保存事务

如果提交事务之前必须取消此事务 （由于某些故障或用户更改），则必须撤消所有已经完成的语句或过程。有关在处理期间执行回退的影响，请参见第 706 页的表 22-2。

在执行 commit transaction 之前，随时可以使用 rollback transaction 命令取消或回退事务。使用保存点可以取消整个事务或部分事务。但是，不能取消已经提交的事务。

rollback transaction 的语法为：

```
rollback {transaction | tran | work}
    [transaction_name | savepoint_name]
```

保存点是用户放置在事务中的一个标记，用于表示事务可回退到的点。在提交整个批处理之前，通过将不需要的部分回退到某个保存点，可以只提交批处理中的某些部分。

通过在事务中放置一个 save transaction 命令可以插入一个保存点。语法为：

```
save {transaction | tran} savepoint_name
```

该保存点名必须符合标识符的规则。

如果 rollback transaction 命令不带 *savepoint_name* 或 *transaction_name*，则事务将回退到批处理中的第一个 begin transaction。

以下是使用 save transaction 和 rollback transaction 命令的方法：

```
begin tran
    statements                  组 A
    save tran mytran
    statements                  组 B
    rollback tran mytran        回退组 B
    statements                  组 C
    commit tran                 提交组 A 和组 C
```

发出 commit transaction 之前，只要不遇到另一个 begin transaction 语句，Adaptive Server 就会将所有后续语句视为事务的一部分。遇到该语句时， Adaptive Server 会将所有后续语句视为新的嵌套事务的一部分。请参见第 693 页的 “嵌套事务”。

如果事务当前不是处于活动状态，则 rollback transaction 或 save transaction 不会影响 Adaptive Server，也不会返回错误消息。

也可以在存储过程或触发器中使用 save transaction 来创建事务，使得可以将事务回退而不影响批处理或其它过程。例如：

```
create proc myproc as
begin tran
save tran mytran
statements
if ...
    begin
        rollback tran mytran
    /*
    ** Rolls back to savepoint.
```

```
        */
          commit tran
        /*
        ** This commit needed; rollback to a savepoint
        ** does not cancel a transaction.
        */
       end
    else
    commit tran
        /*
        ** Matches begin tran; either commits
        ** transaction (if not nested) or
        ** decrements nesting level.
        */
```

除非回退到某个保存点，否则只能在最外层的一对 begin/commit 或 begin/rollback 语句中使用事务名。

警告！ 在嵌套事务语句中使用事务名时，事务名可能被忽略或导致错误。如果要在从其它事务中调用的存储过程或触发器中使用事务，则不要使用事务名。

检查事务的状态

全局变量 *@@transtate* 跟踪事务的当前状态。 Adaptive Server 通过跟踪在语句执行之后发生的所有事务更改来确定要返回的状态。*@@transtate* 可能包含以下值：

表 22-1：@@transtate 值

值	含义
0	事务正在进行。事务有效；已成功执行了前一语句。
1	事务已经成功。事务已完成且已提交其更改。
2	语句已中止。前一语句已中止；对事务无影响。
3	事务已中止。事务已中止且已回退任何更改。

Adaptive Server 不会清除每一个语句后面的 *@@transtate*。在事务中，可以在语句 （如 insert）之后使用 *@@transtate* 来确定该语句已成功还是已中止，从而确定该语句对事务的影响。下面的示例在事务执行期间（insert 成功后）以及提交事务之后检查 *@@transtate*：

```
begin transaction
insert into publishers (pub_id) values ("9999")
```

```
(1 row affected)

select @@transtate

----------
         0

(1 row affected)

commit transaction
select @@transtate

----------
         1

(1 row affected)
```

下一个示例在 insert 失败 （由于规则冲突）以及事务回退之后检查 *@@transtate*：

```
begin transaction
insert into publishers (pub_id) values ("7777")

Msg 552, Level 16, State 1:
A column insert or update conflicts with a rule bound
to the column. The command is aborted. The conflict oc-
cured in database 'pubs2', table 'publishers', rule
'pub_idrule', column 'pub_id'.

select @@transtate

----------
         2

(1 row affected)

rollback transaction
select @@transtate

----------
         3

(1 row affected)
```

Adaptive Server 不会清除每一个语句后面的 *@@transtate*。只有在响应事务采取的操作时，它才会更改 *@@transtate*。语法和编译错误不会影响 *@@transtate* 的值。

嵌套事务

可以在其它事务中嵌套事务。当嵌套 begin transaction 和 commit transaction 语句时，实际上由最外层的语句对开始并提交事务。内部的语句对只跟踪嵌套级别。 Adaptive Server 直到发出与最外层 begin transaction 相匹配的 commit transaction 后才提交事务。通常，当包含 begin/commit 语句对的存储过程或触发器相互调用时，才发生这种事务“嵌套”。

@@trancount 全局变量可跟踪事务的当前嵌套级别。开头的隐式或显式 begin transaction 会将 *@@trancount* 设置为 1。每个后续的 begin transaction 会使 *@@trancount* 递增，而 commit transaction 则使其递减。引发一个触发器也会使 *@@trancount* 递增，并且可用一个可引发该触发器的语句来开始事务。除非 *@@trancount* 等于 0，否则不会提交嵌套事务。

例如，以下嵌套语句组执行到最后的 commit transaction，才由 Adaptive Server 提交：

```
begin tran
    select @@trancount
    /* @@trancount = 1 */
    begin tran
        select @@trancount
        /* @@trancount = 2 */
        begin tran
            select @@trancount
            /* @@trancount = 3 */
        commit tran
    commit tran
commit tran
select @@trancount
/* @@ trancount = 0 */
```

如果在嵌套 rollback transaction 语句时不包括事务名或保存点名，则将回退到最外层的 begin transaction 语句并取消该事务。

事务示例

以下示例演示如何指定事务：

```
begin transaction royalty_change
/* A user sets out to change the royalty split */
/* for the two authors of The Gourmet Microwave. */
/* Since the database would be inconsistent */
/* between the two updates, they must be grouped */
/* into a transaction. */
update titleauthor
set royaltyper = 65
from titleauthor, titles
where royaltyper = 75
and titleauthor.title_id = titles.title_id
and title = "The Gourmet Microwave"
update titleauthor
set royaltyper = 35
from titleauthor, titles
where royaltyper = 25
and titleauthor.title_id = titles.title_id
and title = "The Gourmet Microwave"
save transaction percent_changed
/* After updating the royaltyper entries for */
/* the two authors, the user inserts the */
/* savepoint "percent_changed," and then checks */
/* to see how a 10 percent increase in the */
/* price would affect the authorsí royalty */
/* earnings. */
update titles
set price = price * 1.1
where title = "The Gourmet Microwave"
select (price * royalty * total_sales) * royaltyper
from titles, titleauthor, roysched
where title = "The Gourmet Microwave"
and titles.title_id = titleauthor.title_id
and titles.title_id = roysched.title_id
rollback transaction percent_changed
/* The transaction rolls back to the savepoint */
/* with the rollback transaction command. */
/* Without a savepoint, it would roll back to */
/* the begin transaction. */
commit transaction
```

选择事务模式和隔离级别

Adaptive Server 提供以下选项来支持符合 SQL 标准的事务：

- **事务模式**可以设置是否使用隐式 begin transaction 语句开始事务。
- **隔离级别**是指事务执行期间数据可由其他用户访问的程度。

在每个需要符合 SQL 标准的事务的会话开始时设置这些选项。

以下几节将更详细地描述这些选项。

选择事务模式

Adaptive Server 支持以下事务模式：

- 符合 SQL 标准的模式，称为**链式**模式，它在执行任何数据检索或修改语句之前隐式开始一个事务。这些语句包括：delete、 insert、 open、 fetch、 select 和 update。必须仍然使用 commit transaction 或 rollback transaction 显式结束该事务。
- 缺省模式，称为**非链式**模式或 Transact-SQL 模式，该模式要求与 commit transaction 或 rollback transaction 语句成对地显式使用 begin transaction 语句来完成事务。

可以使用 set 命令的 chained 选项设置这两种模式中的任意一种。但是，不能在应用程序中混用这些事务模式。在不同的模式下，存储过程和触发器有不同的行为，如果要在一种模式下运行在另一种模式下创建的过程，则可能需要执行特殊操作。

使用链式模式时， SQL 标准要求每个 SQL 数据检索和数据修改语句出现在事务中。开始会话之后，或者提交或中止先前事务之后，将自动从第一个数据检索或数据修改语句开始一个事务。这是链式事务模式。

通过打开 set 语句的 chained 选项，可以为当前会话设置此模式：

但是，不能在事务中执行 set chained 命令。要返回到非链式事务模式，请将 chained 选项设置为 off。缺省事务模式为非链式模式。

在链式事务模式中， Adaptive Server 在以下数据检索或修改语句之前隐式执行一个 begin transaction 语句：delete、 insert、 open、 fetch、 select 和 update。例如，根据您所使用的模式，以下语句组可产生不同的结果：

```
insert into publishers
    values ("9906", null, null, null)
begin transaction
delete from publishers where pub_id = "9906"
rollback transaction
```

在非链式事务模式中， rollback 只影响 delete 语句，因此 publishers 仍包含所插入的行。在链式模式中， insert 语句隐式开始一个事务，而且回退操作将影响该事务开始之前涉及到的所有语句，包括 insert。

所有应用程序和即席用户查询均应了解其当前的事务模式。所用的事务模式取决于特定的查询或应用程序是否要符合 SQL 标准。使用链式事务的应用程序 （例如， Embedded SQL 预编译程序）应在每个会话开始时设置链式模式。

事务模式和嵌套事务

虽然链式模式使用数据检索或修改语句隐式开始事务，但是只能通过显式使用 begin transaction 语句来嵌套事务。一旦隐式开始第一个事务，则进一步的数据检索或修改语句将不再开始新的事务，直到第一个事务提交或中止。例如，在下面的查询中，第一个 commit transaction 在链式模式中提交所有更改；第二次提交则是不必要的：

```
insert into publishers
    values ("9907", null, null, null)
    insert into publishers
        values ("9908", null, null, null)
    commit transaction
commit transaction
```

注释 在链式模式中，无论数据检索或修改语句是否成功执行，该语句都将开始执行事务。甚至不访问任何表的 select 语句也会开始一个事务。

查找当前事务模式的状态

可以检查全局变量 *@@tranchained* 来确定 Adaptive Server 的当前事务模式。 select *@@tranchained* 对于非链式模式返回 0，对于链式模式返回 1。

选择隔离级别

ANSI SQL 标准为事务定义了四种隔离级别。每种隔离级别都指定在执行并发事务时禁止的操作种类。较高级别包括由较低级别施加的限制：

- 级别 0 — 确保由某个事务写入的数据能够表示实际数据。防止其它事务更改已经由一个未提交的事务修改 （通过 insert、 delete、 update 等）的数据。在提交该事务之前，其它事务不能修改这些数据。但是，其它事务仍然可以读取未提交的数据，从而导致**脏读**。

- 级别 1 — 防止脏读。当第一个事务修改某行时，如果第二个事务在第一个事务提交更改之前读取该行，则会发生脏读。如果第一个事务回退了更改，则第二个事务读取的信息就会成为无效信息。这是 Adaptive Server 支持的缺省隔离级别。
- 级别 2 — 防止**非重复读取**。当一个事务读取某行而另一个事务又对该行进行修改时就会发生此类读取行为。如果第二个事务提交其更改，则第一个事务随后会读取到与原读取结果不同的结果。

  Adaptive Server 在仅数据锁定表中支持此级别。在所有页锁定表中不支持此级别。
- 级别 3 — 确保某个事务读取的数据在该事务结束之前有效，从而防止出现**幻像行**。 Adaptive Server 通过 select 语句中的 holdlock 关键字支持此级别，它在指定数据上使用读取锁。在某个事务读取满足搜索条件的一组数据行，而第二个事务修改了其中的数据 （通过 insert、 delete、 update 等）时，便会出现幻像行。如果第一个事务按相同的搜索条件重复读取，它将获得一组不同的行。

通过使用 set 命令的 transaction isolation level 选项，可以为会话设置隔离级别。可以只对某一个查询强制实施隔离级别，而不是使用 select 语句的 at isolation 子句。例如：

```
set transaction isolation level 0
```

Adaptive Server 和 ANSI SQL 的缺省隔离级别

缺省情况下，Adaptive Server 事务隔离级别为 1。而 ANSI SQL 标准要求所有事务的缺省隔离级别为 3。这样可防止脏读、非重复读取和幻像行。为了强制实施此缺省隔离级别， Transact-SQL 提供了 set 语句中的 transaction isolation level 3 选项。此选项指示 Adaptive Server 在事务的所有 select 操作中应用 holdlock。例如：

```
set transaction isolation level 3
```

使用 transaction isolation level 3 的应用程序应该在每个会话开始时设置该隔离级别。但是，设置 transaction isolation level 3 将导致 Adaptive Server 在事务执行期间持有所有读取锁。如果同时还使用了链式事务模式，那么该隔离级别对任何隐式开始某个事务的数据检索或修改语句将继续有效。在这两种情况下，由于长时间持有更多的锁，因此会导致某些应用程序的并发问题。

要将会话返回到 Adaptive Server 缺省隔离级别，请输入：

```
set transaction isolation level 1
```

脏读

如果在每个会话开始时设置 transaction isolation level 0，那么访问同一数据时，不受脏读影响的应用程序会具有较好的并发性，并能减少死锁的可能性。例如，查找表中存储的所有储蓄帐号的即时平均余额的应用程序。由于它只要求当前平均余额的快照，而活动表中可能会对平均余额进行频繁的更改，因此该应用程序应使用隔离级别 0 查询该表。而其它要求数据一致性的应用程序 （例如在表中特定的帐号上存款和取款）则应避免使用隔离级别 0。

在隔离级别 0 执行的扫描不会为其扫描获得任何读取锁，因此它们不会阻止其它事务写入同一数据，反之亦然。但是，即使将隔离级别设置为 0，由于修改数据之前必须确保读取的数据正确以便维护数据库完整性，因此实用程序 （如 dbcc）和数据修改语句 （如 update）仍会为其扫描获取读取锁。

在隔离级别 0 执行的扫描不会获得任何读取锁，因此 0 级扫描的结果集可以在扫描过程中发生更改。如果扫描位置因基础表中的更改而丢失，必须使用唯一索引才能重新启动该扫描。如果没有唯一索引，该扫描可能会中止。

缺省情况下，如果在只读数据库之外的表上进行 0 级扫描，就需要使用唯一索引。可以通过强制 Adaptive Server 选择非唯一索引或表扫描来替代此要求，如下所示：

```
select * from table_name (index table_name)
```

基础表上的活动可能会在扫描完成之前中止扫描。

可重复读取

执行可重复读取的事务将锁定该事务执行期间读取的所有行或页。事务中的某个查询读取行之后，在可重复读取事务完成之前其它事务不能更新或删除这些行。但是，与可串行化事务不同，可重复读取事务不能通过执行域锁定来提供幻像保护。其它事务可以插入那些能由可重复读取事务读取的值，并且可以更新行以便它们与可重复读取事务的搜索条件相匹配。

执行可重复读取的事务将锁定该事务执行期间读取的所有行或页。事务中的某个查询读取行之后，在可重复读取事务完成之前其它事务不能更新或删除这些行。但是，与可串行化事务不同，可重复读取事务不能通过执行域锁定来提供幻像保护。其它事务可以插入那些能由可重复读取事务读取的值，并且可以更新行以便它们与可重复读取事务的搜索条件相匹配。

注释　只在仅数据锁定表中支持事务隔离级别 2。如果对所有页锁定表使用事务隔离级别 2（可重复读取），也将强制使用隔离级别 3（可串行化读取）。

要在会话级强制进行可重复读取，请使用：

```
set transaction isolation level 2
```

或者

```
set transaction isolation level repeatable read
```

要在查询中强制使用事务隔离级别 2，请使用：

```
select title_id, price, advance
from titles
at isolation 2
```

或者

```
select title_id, price, advance
from titles
at isolation repeatable read
```

只在事务级支持事务隔离级别 2。不能通过在 select 或 readtext 语句中使用 at isolation 子句将查询的隔离级别设置为 2。请参见第 700 页的 “更改查询的隔离级别”。

查找当前隔离级别的状态

全局变量 *@@isolation* 包含 Transact-SQL 会话的当前隔离级别。查询 *@@isolation* 会返回活动级别的值（0、1 或 3）。例如：

```
select @@isolation

--------
       1

(1 row affected)
```

更改查询的隔离级别

通过在 select 或 readtext 语句中使用 at isolation 子句，可以更改查询的隔离级别。at isolation 子句支持隔离级别 0、1 和 3，但不支持隔离级别 2。at isolation 的 read uncommitted、 read committed 和 serializable 选项表示的隔离级别如下表所示：

at isolation 选项	隔离级别
read uncommited	0
read committed	1
serializable	3

例如，以下两个语句分别在隔离级别 0 和 3 查询同一个表：

```
select *
from titles
at isolation read uncommitted
select *
from titles
at isolation serializable
```

at isolation 子句只在单个 select 和 readtext 查询或 declare cursor 语句中才有效。如果按以下方式使用 at isolation，则 Adaptive Server 将返回语法错误：

- 用于使用 into 子句的查询
- 在子查询内
- 用于 create view 语句中的查询
- 用于 insert 语句中的查询
- 用于使用 for browse 子句的查询

如果查询中有 union 运算符，则必须在最后一个 select 之后指定 at isolation 子句。

SQL-92 标准将 read uncommitted、 read committed 和 serializable 定义为 at isolation 和 set transaction isolation level 的选项。 Transact-SQL 扩展也允许为 at isolation 指定隔离级别 0、 1 或 3，但不能指定隔离级别 2。为简化隔离级别的讨论过程，本手册中的 at isolation 示例没有使用此扩展。

通过使用 select 语句的 holdlock 关键字，也可以强制实施隔离级别 3。但是，不能在指定了 at isolation read uncommitted 的查询中指定 noholdlock 或 shared。（如果在查询中指定 holdlock 和隔离级别 0， Adaptive Server 就会发出警告并忽略 at isolation 子句。）使用不同方法设置隔离级别时，holdlock 关键字优先于 at isolation 子句 （隔离级别 0 除外）， at isolation 优先于 set transaction isolation level 定义的会话级。

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》）。

隔离级别优先级

以下内容描述了应用于定义隔离级别的不同方法的优先规则：

1　holdlock、 noholdlock 和 shared 关键字优先于 at isolation 子句和 set transaction isolation level 选项 （隔离级别为 0 的情况除外）。例如：

```
/* This query executes at isolation level 3 */
select *
    from titles holdlock
    at isolation read committed
create view authors_nolock
    as select * from authors noholdlock
set transaction isolation level 3
/* This query executes at isolation level 1 */
select * from authors_nolock
```

2　at isolation 子句优先于 set transaction isolation level 选项。例如：

```
set transaction isolation level 2
/* executes at isolation level 0 */
select * from publishers
    at isolation read uncommitted
```

在同一查询中不能既使用 at isolation 的 read uncommitted 选项，又使用 holdlock、 noholdlock 和 shared 关键字。

3　set 命令的 transaction isolation level 0 选项优先于 holdlock、 noholdlock 和 shared 关键字。例如：

```
set transaction isolation level 0
/* executes at isolation level 0 */
select *
    from titles holdlock
```

Adaptive Server 在执行上述查询之前会发出警告。

游标和隔离级别

Adaptive Server 为游标提供了三种隔离级别：

- 级别 0 — Adaptive Server 在包含表示当前游标位置的行的基表页上不使用锁。游标不为其扫描获得任何读取锁，因此它们不会阻止其它应用程序访问相同数据。但是，在此隔离级别进行操作的游标是不可更新的，并且它们需要基表上的唯一索引，以确保其扫描的精确性。
- 级别 1 — Adaptive Server 在包含表示当前游标位置的行的基表页上使用共享锁或更新锁。在当前游标位置移出这些页 （fetch 语句的结果）或关闭游标之前，这些页将保持在锁定状态。如果使用索引来搜索基表行，则还会对相应的索引页应用共享锁或更新锁。这是 Adaptive Server 的缺省锁定行为。
- 级别 3 — 在事务中所读取的用于代表游标的任何基表页上，Adaptive Server 使用共享锁或更新锁。另外，事务结束之前将一直持有锁，而不是在不需要数据页时将其释放。如表或视图上的查询所指定， holdlock 关键字也将此锁定级别应用于基表。

游标不支持隔离级别 2。

除了在隔离级别 3 中使用 holdlock 之外，还可以使用 set transaction isolation level 为会话指定四种隔离级别之一。使用 set transaction isolation level 时，任何打开的游标都使用指定的隔离级别，除非将事务隔离级别设置为 2。在此情况下，游标使用隔离级别 3。也可以使用 select 语句的 at isolation 子句为特定的游标指定隔离级别 0、 1 或 3。例如：

```
declare commit_crsr cursor
for select *
from titles
at isolation read committed
```

此语句使游标在隔离级别 1 执行操作，而不管事务或会话的隔离级别是多少。如果在隔离级别 0 (read uncommitted) 声明游标，则 Adaptive Server 也将其定义为只读游标。不能在 declare cursor 语句中同时指定 for update 子句和 at isolation read uncommitted。

在您打开游标 （而不是声明游标）时， Adaptive Server 会根据以下情况确定游标的隔离级别：

- 如果使用 at isolation 子句声明游标，则该隔离级别将替换打开该游标的事务隔离级别。
- 如果*未* 使用 at isolation 声明游标，游标将使用将其打开的隔离级别。如果关闭再重新打开游标，则游标将获得事务的当前隔离级别。

Adaptive Server 在您声明游标时编译此游标的查询。此编译过程在隔离级别 0 与隔离级别 1 或 3 中是不同的。如果在隔离级别为 1 或 3 的事务中声明 **language** 或 **client** 游标，则在隔离级别为 0 的事务中打开该游标会导致错误。

例如：

```
set transaction isolation level 1
declare publishers_crsr cursor
    for select *
    from publishers
open publishers_crsr     /* no error */
fetch publishers_crsr
close publishers_crsr
set transaction isolation level 0
open publishers_crsr     /* error */
```

存储过程和隔离级别

不管事务或会话的隔离级别如何，Sybase 系统过程始终在隔离级别 1 执行操作。用户存储过程在执行该存储过程的事务隔离级别进行操作。如果存储过程中更改了隔离级别，则新的隔离级别只在执行该存储过程期间有效。

触发器和隔离级别

由于触发器由数据修改语句（如 insert）来触发，因此所有触发器都在事务隔离级别或隔离级别 1（取两者中较高的级别）下执行。因此，如果触发器在事务中以隔离级别 0 触发，那么 Adaptive Server 会在执行其第一条语句之前将触发器的隔离级别设置为 1。

符合 SQL 标准

要获取符合 SQL 标准的事务，必须在为后续事务更改模式和隔离级别的每个应用程序的开始位置 set chained 和 transaction isolation level 3 选项。如果应用程序使用游标，则还必须设置 close on endtran 选项。这些选项在第 709 页的 “在事务中使用游标”中进行了说明。

使用 *lock table* 命令提高性能

lock table 命令允许您显式请求，在访问表之前，在事务执行期间在表上放置表锁。在即时表锁可能减少获取大量行锁或页锁的开销并节省锁定时间时，这一点非常有用。此类情况的示例如下：

- 在同一事务中将对某个表进行多次扫描，并且每次扫描可能都需要获取许多个页锁或行锁。
- 扫描将会超过表的锁升级阈值，因此会尝试升级到表锁。

  如果未显式请求表锁，则扫描获取页锁或行锁，直到其达到表的锁升级阈值 （请参见 《参考手册：过程》），此时它将尝试获取表锁。

lock table 命令的语法

lock table 的语法为：

```
lock table table_name in {share | exclusive} mode
    [wait [no_of_seconds] | nowait]
```

wait/nowait 选项允许指定命令阻塞时要获取表锁所等待的时间长度 （请参见第 714 页的 “lock table 命令的 wait/nowait 选项”）。

使用 lock table 时应考虑以下问题：

可以

- 只在事务中发出 lock table。
- 不能在系统表中使用 lock table。
- 可以先在 share 模式中使用 lock table 将表锁定，然后使用它将锁升级到 exclusive 模式。
- 可以在同一事务中使用独立的 lock table 命令锁定多个表。
- 一旦获取表锁，使用 lock table 锁定的表与通过锁升级而不使用 lock table 命令锁定的表之间将没有任何区别。

在存储过程和触发器中使用事务

您可以像使用语句批处理一样在存储过程和触发器中使用事务。如果批处理或存储过程中的事务调用另一个包含事务的存储过程或触发器，则第二个事务将嵌套到第一个事务中。

使用链式模式时，第一个显式或隐式 begin transaction 将开始批处理、存储过程或触发器中的事务。每个后续的 begin transaction 使嵌套级别递增。每个后续的 commit transaction 使嵌套级别递减，直到嵌套级别达到 0 为止。然后 Adaptive Server 将提交整个事务。 rollback transaction 中止整个事务并回退到第一个 begin transaction，而不管嵌套级别和它所包含的存储过程与触发器的数目如何。

在存储过程和触发器中， begin transaction 语句数必须与 commit transaction 语句数相匹配。这也适用于使用链式模式的存储过程。第一个隐式开始事务的语句也必须有一个匹配的 commit transaction 语句。

存储过程中的 rollback transaction 语句不会影响最初调用该过程的过程或批处理中的后续语句。 Adaptive Server 会执行该存储过程或批处理中的后续语句。但是，触发器中的 rollback transaction 语句将中止批处理，因而不执行后续语句。

注释　触发器中的 rollback 语句：1) 回退事务，2) 完成触发器中后续语句，并 3) 中止批处理，因而*不*执行批处理中的后续语句。

例如，下面的批处理调用存储过程 myproc，它包括一个 rollback transaction 语句：

```
begin tran
update titles set ...
insert into titles ...
execute myproc
delete titles where ...
```

update 和 insert 语句被回退，并且事务被中止。Adaptive Server 继续批处理并执行 delete 语句。但是，如果表中具有包括 rollback transaction 的 insert 触发器，则将中止整个批处理，并且不执行 delete。例如：

```
begin tran
update authors set ...
insert into authors ...
delete authors where ...
```

存储过程中不同的事务模式或隔离级别有不同的要求，详见第 708 页的“事务模式和存储过程”中的说明。由于触发器是作为数据修改语句的一部分调用的，因而不受当前事务模式的影响。

错误和事务回退

影响数据完整性的错误会影响隐式或显式事务的状态：

- 严重级为 19 或更高的错误

  由于这些错误会终止用户与服务器的连接，因此用户事务进行中出现的严重级为 19 或更高的任何错误将中止事务并回退所有语句，直到最外层的 begin transaction。 Adaptive Server 始终在会话结束时回退所有未提交的事务。

- 影响数据完整性的数据修改命令中的错误 （请参见第 707 页的表 22-3）：
  - 算术溢出和除零错误 （使用 set arithabort arith_overflow 命令可以更改对事务的影响）
  - 权限冲突
  - 规则冲突
  - 重复键冲突

表 22-2 总结了在几种不同的环境中 rollback 如何影响 Adaptive Server 的处理。

表 22-2：回退如何影响处理

环境	*rollback* 的影响
仅事务	回退从事务开始后的所有数据修改。如果事务跨越多个批处理，则 rollback 将影响所有批处理。 执行回退后发出的所有命令。
仅存储过程	无。
事务中的存储过程	回退从事务开始后的所有数据修改。如果事务跨越多个批处理，则 rollback 将影响所有批处理。 执行回退后发出的所有命令。 存储过程产生错误消息 266：“Transaction count after EXECUTE indicates that a COMMIT or ROLLBACK TRAN is missing.”
仅触发器	触发器完成，但回退触发器的影响。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。
事务中的触发器	触发器完成，但回退触发器的影响。 回退从事务开始后的所有数据修改。如果事务跨越多个批处理，则 rollback 将影响所有批处理。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。
嵌套触发器	内部触发器完成，但回退所有触发器的影响。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。

环境	*rollback* 的影响
事务中的嵌套触发器	内部触发器完成，但回退所有触发器的影响。 回退从事务开始后的所有数据修改。如果事务跨越多个批处理，则 rollback 将影响所有批处理。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。

在存储过程和触发器中，begin transaction 语句数必须与 commit 语句数相匹配。包含不成对的 begin/commit 语句的过程或触发器在执行时将生成警告消息。这也适用于使用链式模式的存储过程：第一个隐式开始事务的语句必须有一个匹配的 commit 语句。

对于重复键错误和规则冲突，触发器完成 （除非还有 return 语句），并执行 print、 raiserror 等语句或执行远程过程调用。然后，将回退触发器和剩余的事务，并中止剩余的批处理。 rollback 语句不能回退在常规 SQL 事务 （没有使用 DB-Library 两阶段提交）中执行的远程过程调用。

表 22-3 总结了在几种不同环境中由重复键错误或规则冲突所导致的回退如何影响 Adaptive Server 的处理。

表 22-3：重复键错误 / 规则冲突导致的回退

环境	事务期间数据修改错误的影响
仅事务	当前命令被中止。不回退以前的命令，并执行后续命令。
存储过程中的事务	同上。
事务中的存储过程	同上。
仅触发器	触发器完成，但回退触发器的影响。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。
事务中的触发器	触发器完成，但回退触发器的影响。 回退从事务开始后的所有数据修改。如果事务跨越了多个批处理，则回退将影响所有的批处理。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。
嵌套触发器	内部触发器完成，但回退所有触发器的影响。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。
事务中的嵌套触发器	内部触发器完成，但回退所有触发器的影响。 回退从事务开始后的所有数据修改。如果事务跨越了多个批处理，则回退将影响所有的批处理。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。
事务中的错误将回退触发器	回退触发器的影响。回退从事务开始后的所有数据修改。如果事务跨越了多个批处理，则回退将影响所有的批处理。 触发器继续并导致重复键或规则错误。通常，触发器会回退其影响并继续执行，但是这种情况下不回退触发器的影响。 触发器完成之后，不执行批处理中剩余的任何命令。在下一个批处理恢复处理。

事务模式和存储过程

为使用非链式事务模式而编写的存储过程可能与其它使用链式模式的事务不兼容，反之亦然。例如，下面是一个使用链式事务模式的有效存储过程：

```
create proc myproc
as
insert into publishers
    values ("9996", null, null, null)
commit work
```

由于 commit 没有相应的 begin，因此使用非链式事务模式的程序调用此过程时会失败。您可能会遇到其它问题：

- 使用链式模式开始事务的应用程序可能无法创建长期事务或可能在整个会话期间持有数据锁，这会降低 Adaptive Server 的性能。
- 这些应用程序可能不会按预期次数嵌套事务。根据不同的事务模式，这会产生不同的结果。

一般说来，使用某种事务模式的应用程序应该调用为使用该模式而编写的存储过程。例外情况是， Sybase 系统过程（sp_procxmode 除外）可由使用任何事务模式的会话调用。（有关 sp_procxmode 的信息，请参见第 709 页的 “设置存储过程的事务模式”。）如果执行系统过程时没有活动的事务，则在整个过程执行期间 Adaptive Server 将关闭链式模式。返回之前， Adaptive Server 会重新将模式设置为初始设置。

Adaptive Server 在创建过程的会话中使用事务模式 （“链式”或 “非链式”）来标记所有过程。这有助于避免与事务相关的问题，这些事务使用某种模式来调用使用其它模式的事务。标记为 “链式”的存储过程不能在使用非链式事务模式的会话中执行，反之亦然。

触发器可在任何事务模式中执行。因为触发器始终被作为数据修改语句的一部分来调用，所以它们或者是链式事务的一部分 （如果会话使用链式模式），或者保持其当前事务模式。

警告！ 使用事务模式时，请务必了解每种设置对应用程序的影响。

设置存储过程的事务模式

使用 sp_procxmode 显示或更改存储过程的事务模式。例如，若要将存储过程 byroyalty 的事务模式更改为 “链式”，请输入：

```
sp_procxmode byroyalty, "chained"
```

sp_procxmode ìanymodeî 使存储过程在链式或非链式事务模式下运行。例如：

```
sp_procxmode byroyalty, "anymode"
```

使用不带任何参数值的 sp_procxmode 可显示当前数据库中所有存储过程的事务模式：

```
sp_procxmode

procedure name                transaction mode
--------------------------    --------------------
byroyalty                     Any Mode
discount_proc                 Unchained
history_proc                  Unchained
insert_sales_proc             Unchained
insert_salesdetail_proc       Unchained
storeid_proc                  Unchained
storename_proc                Unchained
title_proc                    Unchained
titleid_proc                  Unchained

(9 rows affected, return status = 0)
```

只能在非链式事务模式中使用 sp_procxmode。

要更改过程的事务模式，您必须是系统管理员、数据库所有者或过程所有者。

在事务中使用游标

缺省情况下， Adaptive Server 在通过 commit 或 rollback 结束事务时，不更改游标的状态（打开或关闭）。但是， SQL 标准会将一个打开的游标与其活动事务相关联。提交或回退该事务都将自动关闭与其关联的任何打开的游标。

为强制实施这种符合 SQL 标准的行为，Adaptive Server 提供了 set 命令的 close on endtran 选项。另外，如果将链式模式设置为 on，则 Adaptive Server 在打开游标时开始事务，在提交或回退最外层事务时关闭该游标。

例如，缺省情况下，以下语句序列将产生错误：

```
open test_crsr
commit tran
open test_crsr
```

如果将 close on endtran 或 chained 选项设置为 on，则在提交最外层事务之后游标的状态将从打开更改为关闭。这将允许重新打开游标。

注释 因为可以通过游标返回客户端应用程序缓冲行，并且允许用户在这些缓冲区中滚动，所以中止事务之后这些客户端应用程序不应该回滚。因为 close on endtran 选项或链式模式强制执行事务回退 （但客户端不知道），所以客户端高速缓存中的行可能变为无效行。

事务结束之前，事务中的游标一直持有它所获取的所有排它锁。使用 HOLDLOCK 关键字、 at isolation serializable 子句或 set isolation level3 选项时，这一点也适用于共享锁。

以下规则定义了通过事务游标进行更新的行为：

- 如果在显式事务中发生更新，则该更新将被视为该事务的一部分。如果提交该事务，则也将提交该事务所包括的任何更新。如果中止该事务，则将回退该事务所包括的任何更新。但在被中止的事务以外发生的由同一游标进行的更新不受此影响。
- 如果通过游标所作的更新发生在显式 （和指定的客户端）事务中，则 Adaptive Server 不会在游标关闭时提交这些更新。只有与该游标相关的事务结束时，才提交或回退待执行更新。
- 事务的提交或中止对不操纵结果行的 SQL 游标语句 （例如 declare cursor、 open cursor、 close cursor、 set cursor rows 和 deallocate cursor）没有任何影响。例如，如果客户端在某事务中打开一个游标，并中止了该事务，则该游标在中止之后将保持打开 （除非设置了 close on endtran 或使用了链式模式）。

但是，如果不设置 close on endtran，则该游标在事务结束后仍保持打开，而且其当前页锁保持有效。它在读取其它行时，也可以继续获取锁。

使用事务时应考虑的问题

在应用程序中使用事务时应该考虑以下问题：

- 不带事务名或保存点名的 rollback 语句会始终将语句回退到最外层的 begin transaction （显式或隐式）语句并取消事务。如果在发出 rollback 时当前没有任何事务，则该语句不会产生任何影响。

  在触发器或存储过程中，不带事务名或保存点名的 rollback 语句会将所有语句回退到最外层的 begin transaction （显式或隐式）。

- rollback 不会对用户生成任何消息。如果需要警告，请使用 raiserror 或 print 语句。

- 将大量 Transact-SQL 命令组合为一个长期运行的事务可能会影响恢复时间。如果长期事务执行期间 Adaptive Server 失败，则由于 Adaptive Server 必须首先撤消整个事务，恢复时间会增加。

- 在用户事务中具有的数据库数量可以与在安装 Adaptive Server 时安装的数据库数量一样多。例如，如果 Adaptive Server 具有 25 个数据库，则在用户事务中可以包括 25 个数据库。

- 远程过程调用 (RPC) 可以独立于任何包含它的事务执行。在标准事务 （即，不使用 Open Client DB-Library/C 两阶段提交或 Adaptive Server 分布式事务管理功能的事务）中，由远程服务器通过 RPC 执行的命令不能通过 rollback 回退，并且执行时不需要依赖 commit 。

- 事务不能跨越客户端应用程序和服务器之间的多个连接。例如，DB-Library/C 应用程序不能跨多个打开的 DBPROCESS 连接在一个事务中组合多个 SQL 语句。

- Adaptive Server 版本执行两次日志扫描：第一次扫描查找数据页解除分配和未保留页，第二次扫描查找日志页解除分配。这些扫描是内部优化，对用户透明，并且自动执行；您无法打开或关闭扫描。

  借助提交后优化， Adaptive Server 可以记住包含这些日志记录的“下一”日志页 （向后）。在提交后阶段，在处理完某一页的记录后， Adaptive Server 会移动到 “下一”要求提交后处理的页。在并发环境中，许多用户同时将各自的事务记录到 syslogs 中，提交后优化通过避免读取或扫描不必要的日志页，可以提高提交后操作的性能。

  该优化过程不会显示在任何诊断信息中。

事务的备份和恢复

对数据库所做的每次更改，无论是单个 update 语句还是一组 SQL 语句执行的结果，都会被记录在系统表 syslogs 中。此表称为**事务日志**。

不记录某些更改数据库的命令，例如 truncate table、批量复制到没有索引的表中、 select into、 writetext 和 dump transaction with no_log。

事务日志实时记录 update、 insert 或 delete 语句。事务开始时，会在日志中记录一个 begin transaction 事件。接收每个数据修改语句时，都将其记录到日志中。

对数据库本身做出任何更改之前，日志始终先记录所有的数据更改。这种日志称为前写式日志，可确保在出现故障时能够完全恢复数据库。

硬件或介质问题、系统软件问题、应用软件问题、程序指示的事务取消或用户取消事务都可能导致故障。

一旦出现上述任何故障，可以从使用 dump 命令生成的备份中恢复数据库副本，回放事务日志。

要从故障中恢复，因为部分事务不是精确的更改，所以发生故障时正在进行但尚未提交的事务必须撤消。如果不能保证完成的事务被写入到数据库设备中，则必须重新执行这些事务。

如果 Adaptive Server 发生故障时有处于活动状态的、长期运行但没有提交的事务，则撤消其更改所要求的时间与事务已经运行的时间相同。不包含与 begin transaction 相匹配的 commit transaction 或 rollback transaction 的事务便是其中一例。这样可防止 Adaptive Server 写入任何更改，但增加了恢复时间。

Adaptive Server 的动态转储允许在继续使用数据库的情况下备份数据库和事务日志。请经常备份数据库事务日志。备份数据的次数越频繁，发生系统故障时丢失的数据就越少。

虽然执行数据库备份的权限可以移交给其他用户，但是每个数据库所有者或具有 ss_oper 角色的用户负责使用 dump 命令来备份数据库及其事务日志。但是，缺省情况下，使用 load 命令的权限属于数据库所有者，而不能被移交。

一旦发出相应的 load 命令， Adaptive Server 就将全面处理恢复进程。Adaptive Server 也控制检查点间隔，检查点可保证所有更改的数据页都写入到数据库设备中。必要时，用户可使用 checkpoint 命令强制执行检查点。

有关备份和恢复的详细信息，请参见 《参考手册：命令》和 《系统管理指南：卷 2》的第 11 章 “制定备份和恢复计划”。

第 23 章

锁定命令和选项

Adaptive Server 提供多种用于锁定的命令和选项。本章概述了可用锁定的类型以及如何通过 Transact SQL 调用它们。

设置等待锁的时间限制

Adaptive Server 允许指定锁等待时间以确定命令等待多长时间才能获取锁：

- 可以使用 lock table 命令的 wait/nowait 选项指定获取表锁所等待的时间限制。
- 会话期间，可以使用 set lock 命令来指定会话期间发出的所有后续命令的锁等待时间。
- sp_configure 参数 lock wait period（与会话级设置 set lock wait nnn 一起使用）只适用于用户定义的表。这些设置对系统表没有任何影响。
- 在存储过程中，可以使用 set lock 命令来指定该存储过程中发出的所有后续命令的锁等待时间。
- 可以使用 sp_configure 的 lock wait period 选项来设置全服务器范围的锁等待时间。

lock table 命令的 *wait*/*nowait* 选项

在事务中， lock table 命令允许在不等待命令获取足够的行级锁或页级锁以升级到表锁的情况下，对表请求一个表锁。

lock table 命令包含 wait/nowait 选项，使用该选项可以指定在其它事务中的操作放弃对目标表的所有锁前，命令要等待的时间长度。

lock table 命令的语法是：

```
lock table table_name in {share | exclusive} mode
    [wait [no_of_seconds] | nowait]
```

以下事务中的命令将获取 titles 上的表锁的等待时间设置为 2 秒

```
lock table titles in share mode wait 2
```

如果等待时间在获得表锁之前到期，事务将继续并使用行或页锁定，就如同没有 lock table 一样，并且生成以下信息性消息 （错误号 12207）：

```
Could not acquire a lock within the specified wait
period.COMMAND level wait...
```

有关事务期间处理此错误消息的代码示例，请参见 《参考手册：命令》。

注释 如果使用 lock table...wait 时没有指定 *no_of_seconds*，则此命令将无限期地等待锁。

可以在会话级和系统级设置等待锁的时间限制，如以下各节所述。使用 lock table 命令设置的等待时间会替换这两种时间限制

nowait 选项等效于等待 0 秒的 wait 选项：lock table 或者立即获取表锁，或者生成上述信息性消息。如果没有获取此锁，如同没有 lock table 命令一样，事务将继续进行。

可以在会话级或存储过程中使用 set lock 命令来控制任务等待获取锁的时间长度。

系统管理员可以使用 sp_configure 选项 lock wait period 来设置获取锁的全服务器范围的时间限制。

设置会话级等待获取锁的时间限制

可以使用 set lock wait 来控制会话或存储过程中命令等待获取锁的时间长度。语法为：

```
set lock {wait no_of_seconds | nowait}
```

no_of_seconds 是一个整数。因此，以下示例将等待锁的会话级时间限制设置为 5 秒：

```
set lock wait 5
```

但有一个例外，如果 set lock wait 时间在命令获取锁之前到期，则此命令失败，包含此命令的事务被回退，并且生成以下错误消息：

```
Msg 12205, Level 17, State 2:
Server 'sagan', Line 1:
Could not acquire a lock within the specified wait pe-
riod. SESSION level wait period=300 seconds, spid=12,
lock type=shared page, dbid=9, objid=2080010441,
pageno=92300, rowno=0. Aborting the transaction.
```

如果事务中 lock table 命令设置的等待时间比 set lock wait 设置的等待时间长，将出现例外。在这种情况下，事务在超时之前使用 lock table 等待时间，如上一节所述。

set lock nowait 选项与等待时间为 0 秒的 set lock wait 选项等效。如果除 lock table 以外的命令不能立即获取请求的锁，则此命令失败，其事务被回退，并且生成上述错误消息。

如果同时设置了全服务器范围的锁等待限制和会话级锁等待限制，则会话级限制优先。如果没有使用 set lock wait 设置会话级等待时间，则使用服务器级等待时间。

设置全服务器范围的锁等待限制

系统管理员可以使用配置参数 lock wait period 来配置全服务器范围的锁等待限制。语法为：

```
sp_configure "lock wait period" [, no_of_seconds]
```

如果锁等待时间在命令获取锁之前到期，那么除非有覆盖的 set lock wait 或 lock table 等待时间，否则此命令失败，包含它的事务被回退，并且生成以下错误消息：

```
Msg 12205, Level 17, State 2:
Server 'wiz', Line 1:
Could not acquire a lock within the specified wait
```

```
period. SERVER level wait period=300 seconds, spid=12,
lock type=shared page, dbid=9, objid=2080010441,
pageno=92300, rowno=0. Aborting the transaction.
```

通过 set lock wait 或 lock table wait 输入的时间限制将覆盖服务器级的锁等待时间。因此，如果服务器级等待时间为 5 秒，而会话级等待时间为 10 秒，则 update 命令在命令失败和中止其事务前将等待 10 秒以获取锁。

缺省服务器级锁等待时间为 “wait forever”（永远等待）。若要在设置有限等待时间之后恢复缺省值，请使用 sp_configure 来设置 lock wait period 的值，如下所示：

```
sp_configure "lock wait period", 0, "default"
```

锁等待超时次数的有关信息

sp_sysmon 报告等待获取锁的任务在指定时间内没有获取锁的次数。

队列处理的 Readpast 锁定

Readpast 锁定是 select 和 readtext 命令以及数据修改命令 update、 delete 和 writetext 的可用选项。它指示命令自动跳过遇到的所有不兼容锁，而不会阻塞、终止或生成消息。它主要用于表中各行组成一个队列的情况。在这种情况下，许多任务可能访问表来处理排队的行。例如，这些行可能代表排队的客户或客户订单。给定的任务不处理队列中的特定成员，但可处理任何满足其选择标准的队列成员。

Readpast 语法

使用 Readpast 锁定的语法是：

```
{select | update | delete} ...
    from tablename [holdlock | noholdlock]
        [readpast] [shared]
        ...

readtext [[database.]owner.]table_name.column_name
        text_pointer offset size
    [holdlock | noholdlock] [shared] [readpast]
        ...
```

writetext [[*database*.]*owner*.]*table_name.column_name*
text_pointer [readpast] [with log] *data*

Readpast 查询期间的不兼容锁

对于 select 和 readtext 命令，不兼容锁是排它锁。因此， select 和 readpast 命令可以访问持有共享锁或更新锁的任何行或页。

对于 delete、 update 和 writetext 命令，任何类型的页锁或行锁都不兼容，因此：

- 在数据行锁定表中将跳过带有共享、更新或排它行锁的所有行，并且
- 在数据页锁定表中将跳过带有共享、更新或排它锁的所有页。

如果存在排它表锁，将阻塞指定 readpast 的所有命令，但在事务隔离级别 0 下执行的 select 命令除外。

所有页锁定表和 Readpast 查询

如果对所有页锁定表指定了 readpast 选项，此 readpast 选项将被忽略。此命令在为此命令或会话指定的隔离级别下执行：

- 如果隔离级别是 0，则执行脏读，且命令从锁定行返回值而不会阻塞。
- 如果隔离级别为 1 或 3，则在必须读取具有不兼容锁的页时，命令将阻塞。

带 Readpast 的隔离级别 *select* 查询的效果

Readpast 锁定被设计为在事务隔离级别 1 或 2 中使用。

会话级事务隔离级别和 Readpast

对于仅数据锁定表， readpast 在 select 命令中对表的影响如表 23-1 所示。

表 23-1：会话级隔离级别和 Readpast 的用法

会话隔离级别	效果
0， read uncommitted （脏读）	将忽略 readpast，并向用户返回包含未提交事务的行。输出警告消息。
1， read committed	将跳过带有不兼容锁的行或页；所读取的行或页上未持有锁。
2， repeatable read	将跳过带有不兼容锁的行或页；所读取的所有行或页上都持有共享锁，直到语句或事务结束为止。
3， serializable	将忽略 readpast，命令在级别 3 执行。命令将在带有不兼容锁的任何行或页上阻塞。

查询级的隔离级别和 Readpast

如果指定 readpast 的 select 命令还包括以下任何子句，这些命令将失败并显示错误消息：

- at isolation 子句，指定 0 或 read uncommitted
- at isolation 子句，指定 3 或 serializable
- 同一表上的 holdlock 关键字

如果指定 readpast 的 select 查询也指定了 at isolation 2 或 at isolation repeatable read，将在 readpast 表中持有共享锁，直到语句或事务完成为止。

包括 readpast 并指定 at isolation read uncommitted 的 readtext 命令发出警告消息，然后自动在隔离级别 0 下运行。

带有 *readpast* 和隔离级别的数据修改命令

如果会话的事务隔离级别为 0，则使用 readpast 的 delete、 update 和 writetext 命令不会发出警告消息。

- 对于数据页锁定表，这些命令将修改没有被不兼容锁锁定的所有页中的所有行。
- 对于数据行锁定表，它们影响没有被不兼容锁锁定的所有行。

如果会话的事务隔离级别是 3 （序列化读取），当遇到带有不兼容锁的行或页时，使用 readpast 的 delete、 update 和 writetext 命令将自动阻塞。

在事务隔离级别 2 （序列化读取）， delete、 update 和 writetext 命令：

- 修改没有被不兼容锁锁定的所有页中的所有行。
- 对于数据行锁定表，它们影响没有被不兼容锁锁定的所有行。

text、*unitext* 和 *image* 列及 Readpast

如果带有 readpast 选项的 select 命令遇到具有不兼容锁的文本列，则 Readpast 锁定将检索行，但返回具有空值的文本列。在这种情况下，由于列被锁定，因此包含空值的文本列和返回的空值之间没有区别。

如果将带有 readpast 选项的 update 命令应用于两个或多个文本列，并且检查的第一个文本列含有不兼容锁，则 Readpast 锁定将跳过此行。如果此列没有不兼容锁，则此命令将获取一个锁并修改此列。然后，如果行中的任何后续文本列带有不兼容锁，则此命令将阻塞，直到它可以获取锁并修改此列为止。

如果行中的任何文本列带有不兼容锁，则具有 readpast 选项的 delete 命令将跳过此行。

Readpast 锁定示例

以下示例说明了 Readpast 锁定。

若要跳过含有排它锁的所有行，请使用以下命令：

```
select * from titles readpast
```

若要只更新没有被其它会话锁定的行，请使用以下命令：

```
update titles
    set price = price * 1.1
    from titles readpast
```

若要对 titles 表而不是 authors 或 titleauthor 表使用 Readpast 锁定，请使用以下命令：

```
select *
    from titles readpast, authors, titleauthor
    where titles.title_id = titleauthor.title_id
    and authors.au_id = titleauthor.au_id
```

若要只删除没有在 stores 表中锁定的行，但允许扫描在 authors 表上阻塞，请使用以下命令：

```
delete stores from stores readpast, authors
where stores.city = authors.city
```

附 录 A

pubs2 数据库

本附录描述样本数据库 pubs2。该数据库包含表 publishers、authors、titles、titleauthor、salesdetail、sales、stores、roysched、discounts、blurbs 和 au_pix。

pubs2 数据库也列出用于创建这些表的主键与外键、规则、缺省值、视图、触发器和存储过程。

第 730 页的图 A-1 显示了 pubs2 数据库的框图。

有关安装 pubs2 的信息，请参见适用于所用平台的配置指南。

pubs2 数据库中的表

以下几节详细介绍 pubs2 的每个表。在这些表中，每个列标题指定了列名、数据类型（包括任何用户定义的数据类型）和空或非空状态。列标题还指定了影响该列的所有缺省值、规则、触发器和索引。

publishers 表

publishers 表包含 pubs2 数据库中出版社的出版社名称和 ID、城市和州。

publishers 定义如下：

```
create table publishers
(pub_id char(4) not null,
pub_name varchar(40) not null,
city varchar(20) null,
state char(2) null)
```

其主键为 pub_id：

```
sp_primarykey publishers, pub_id
```

其 pub_idrule 规则定义如下：

```
create rule pub_idrule
as @pub_id in
("1389", "0736", "0877", "1622", "1756")
or @pub_id like "99[0-9][0-9]"
```

authors 表

authors 表包含 pubs2 数据库中的姓名、电话号码、作者 ID 和其它有关作者的信息。

authors 定义如下：

```
create table authors
(au_id id not null,
au_lname varchar(40) not null,
au_fname varchar(20) not null,
phone char(12) not null,
address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null)
```

其主键为 au_id：

```
sp_primarykey authors, au_id
```

其对于 au_lname 和 au_fname 列的非聚簇索引定义如下：

```
create nonclustered index aunmind
on authors (au_lname, au_fname)
```

phone 列有以下缺省值：

```
create default phonedflt as "UNKNOWN"
sp_bindefault phonedft, "authors.phone"
```

以下视图使用 authors：

```
create view titleview
as
select title, au_ord, au_lname,
price, total_sales, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

titles 表

titles 表包含有关 pubs2 数据库中书目的书目 ID、标题、类型、出版社 ID、价格和其它信息。

titles 定义如下：

```
create table titles
(title_id tid not null,
title varchar(80) not null,
type char(12) not null,
pub_id char(4) null,
price money null,
advance money null,
total_sales int null,
notes varchar(200) null,
pubdate datetime not null,
contract bit not null)
```

其主键为 title_id：

```
sp_primarykey titles, title_id
```

其 pub_id 列为 publishers 表的外键：

```
sp_foreignkey titles, publishers, pub_id
```

其对于 title 列的非聚簇索引定义如下：

```
create nonclustered index titleind
on titles(title)
```

其 title_idrule 定义如下：

```
create rule title_idrule
as
@title_id like "BU[0-9][0-9][0-9][0-9]" or
@title_id like "[MT]C[0-9][0-9][0-9][0-9]" or
@title_id like "P[SC][0-9][0-9][0-9][0-9]" or
@title_id like "[A-Z][A-Z]xxxx" or
@title_id like "[A-Z][A-Z]yyyy"
```

type 列有以下缺省值：

```
create default typedflt as "UNDECIDED"
sp_bindefault typedflt, "titles.type"
```

pubdate 列有以下缺省值：

```
create default datedflt as getdate()
sp_bindefault datedflt, "titles.pubdate"
```

titles 使用以下触发器：

```
create trigger deltitle
on titles
for delete
as
if (select count(*) from deleted, salesdetail
where salesdetail.title_id = deleted.title_id) >0
begin
   rollback transaction
   print "You can't delete a title with sales."
end
```

以下视图使用 titles：

```
create view titleview
as
select title, au_ord, au_lname,
price, total_sales, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

titleauthor 表

titleauthor 表显示 pubs2 数据库中书目的作者 ID、书目 ID 和书目的版税（百分比）。

titleauthor 定义如下：

```
create table titleauthor
(au_id id not null,
title_id tid not null,
au_ord tinyint null,
royaltyper int null)
```

其主键为 au_id 和 title_id：

```
sp_primarykey titleauthor, au_id, title_id
```

其 title_id 和 au_id 列为 titles 和 authors 的外键：

```
sp_foreignkey titleauthor, titles, title_id
sp_foreignkey titleauthor, authors, au_id
```

其对于 au_id 列的非聚簇索引定义如下：

```
create nonclustered index auidind
on titleauthor(au_id)
```

其对于 title_id 列的非聚簇索引定义如下：

```
create nonclustered index titleidind
on titleauthor(title_id)
```

以下视图使用 titleauthor：

```
create view titleview
as
select title, au_ord, au_lname,
price, total_sales, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

以下过程使用 titleauthor：

```
create procedure byroyalty @percentage int
as
select au_id from titleauthor
where titleauthor.royaltyper = @percentage
```

salesdetail 表

salesdetail 表显示 pubs2 数据库中销售的书店 ID、订单 ID、书目编号、销售量和折扣。

salesdetail 定义如下：

```
create table salesdetail
(stor_id char(4) not null,
ord_num numeric(6.0),
title_id tid not null,
qty smallint not null,
discount float not null)
```

其主键为 stor_id 和 ord_num：

```
sp_primarykey salesdetail, stor_id, ord_num
```

其 title_id、 stor_id 和 ord_num 列为 titles 和 sales 的外键：

```
sp_foreignkey salesdetail, titles, title_id
sp_foreignkey salesdetail, sales, stor_id, ord_num
```

其对于 title_id 列的非聚簇索引定义如下：

```
create nonclustered index titleidind
on salesdetail (title_id)
```

其对于 stor_id 列的非聚簇索引定义如下：

```
create nonclustered index salesdetailind
on salesdetail (stor_id)
```

其 title_idrule 规则定义如下：

```
create rule title_idrule
as
@title_id like "BU[0-9][0-9][0-9][0-9]" or
@title_id like "[MT]C[0-9][0-9][0-9][0-9]" or
@title_id like "P[SC][0-9][0-9][0-9][0-9]" or
@title_id like "[A-Z][A-Z]xxxx" or
@title_id like "[A-Z][A-Z]yyyy"
```

salesdetail 使用以下触发器：

```
create trigger totalsales_trig on salesdetail
    for insert, update, delete
as
/* Save processing:return if there are no rows affected
*/
if @@rowcount = 0
    begin
      return
end
/* add all the new values */
/* use isnull:a null value in the titles table means
**              "no sales yet" not "sales unknown"
*/
update titles
    set total_sales = isnull(total_sales, 0) + (select
sum(qty)
    from inserted
    where titles.title_id = inserted.title_id)
    where title_id in (select title_id from inserted)
/* remove all values being deleted or updated */
update titles
    set total_sales = isnull(total_sales, 0) - (select
sum(qty)
    from deleted
    where titles.title_id = deleted.title_id)
    where title_id in (select title_id from deleted)
```

sales 表

sales 表包含 pubs2 数据库中销售的书店 ID、订单号和销售日期。

sales 定义如下：

```
create table sales
(stor_id char(4) not null,
ord_num varchar(20) not null,
date datetime not null)
```

其主键为 stor_id 和 ord_num：

```
sp_primarykey sales, stor_id, ord_num
```

其 stor_id 列为 stores 的外键：

```
sp_foreignkey sales, stores, stor_id
```

stores 表

stores 表包含 pubs2 数据库中书店的名称、地址、 ID 号和付款方式。

stores 定义如下：

```
create table stores
(stor_id char(4) not null,
stor_name varchar(40) not null,
stor_address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null,
payterms varchar(12) null)
```

其主键为 stor_id：

```
sp_primarykey stores, stor_id
```

roysched 表

roysched 表包含 pubs2 数据库中的版税 （定义为价格的百分比）。

roysched 定义如下：

```
create table roysched
title_id tid not null,
lorange int null,
hirange int null,
royalty int null)
```

其主键为 title_id：

```
sp_primarykey roysched, title_id
```

其 title_id 列为 titles 的外键：

```
sp_foreignkey roysched, titles, title_id
```

其对于 title_id 列的非聚簇索引定义如下：

```
create nonclustered index titleidind
on roysched (title_id)
```

discounts 表

discounts 表包含 pubs2 数据库中列出的书店折扣。

discounts 定义如下：

```
create table discounts
(discounttype varchar(40) not null,
stor_id char(4) null,
lowqty smallint null,
highqty smallint null,
discount float not null)
```

其主键为 discounttype 和 stor_id：

```
sp_primarykey discounts, discounttype, stor_id
```

其 stor_id 列为 stores 的外键：

```
sp_foreignkey discounts, stores, stor_id
```

blurbs 表

blurbs 表包含 pubs2 数据库中书籍的短评示例。

blurbs 定义如下：

```
create table blurbs
(au_id id not null,
copy text null)
```

其主键为 au_id：

```
sp_primarykey blurbs, au_id
```

其 au_id 列为 authors 的外键：

```
sp_foreignkey blurbs, authors, au_id
```

au_pix 表

author_pix 表包含 pubs2 数据库中作者的照片。

au_pix 定义如下：

```
create table au_pix
(au_id char(11) not null,
pic image null,
format_type char(11) null,
bytesize int null,
pixwidth_hor char(14) null,
pixwidth_vert char(14) null)
```

其主键为 au_id：

```
sp_primarykey au_pix, au_id
```

其 au_id 列为 authors 的外键：

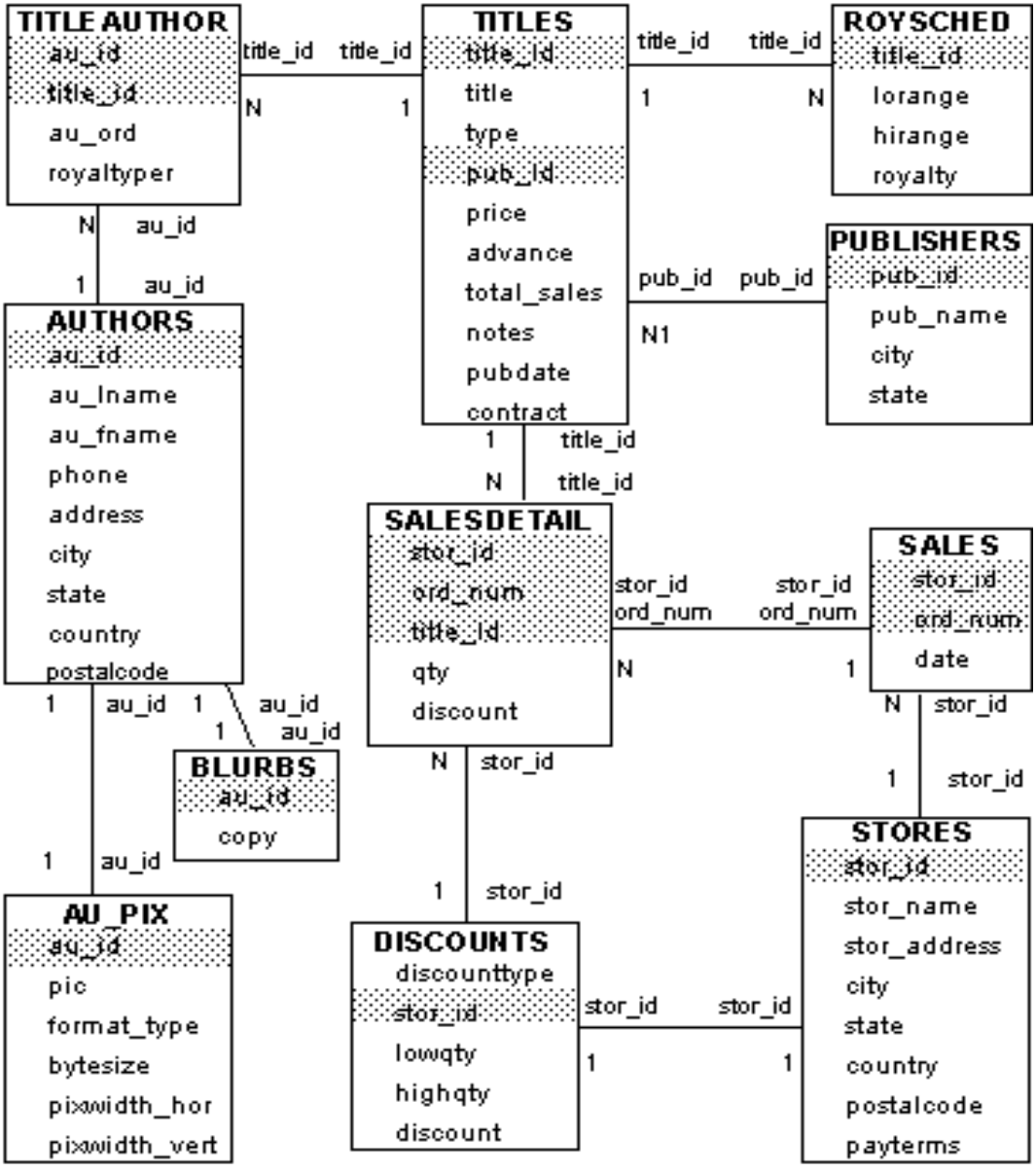
```
sp_foreignkey au_pix, authors, au_id
```

pic 列包含二进制数据。由于 image 数据 （六张图片， PICT、 TIF 和 Sunraster 文件格式各两幅）相当大，因此应该*只*在要使用或测试 image 数据类型时才运行 installpix2 脚本。提供 image 数据旨在展示 Sybase 如何存储 image 数据。 Sybase 不提供任何显示 image 数据的工具：从数据库中提取这些数据后，必须使用适当的屏幕图形工具来显示图像。

pubs2 数据库框图

此框图显示 pubs2 数据库中的表以及它们之间的某些关系。

图 A-1：pubs2 数据库框图



附 录 B

pubs3 数据库

本附录介绍样本数据库 pubs3。该数据库包含表 publishers、authors、titles、titleauthor、salesdetail、sales、stores、store_employees、roysched、discounts 和 blurbs。

它列出了用于创建每个表的主键与外键、规则、缺省值、视图、触发器和存储过程。

第 739 页的图 B-1 显示了 pubs3 数据库的框图。

有关安装 pubs3 的信息，请参见适用于所用平台的配置指南。

pubs3 数据库中的表

以下几节介绍 pubs3 的每个表。在这些表中，每个列标题指定了列名、数据类型（包括任何用户定义的数据类型）、空或非空状态以及如何使用参照完整性。列标题还指定了影响该列的所有缺省值、规则、触发器和索引。

publishers 表

publishers 表包含 pubs3 数据库中每个出版社的出版社 ID、名称、所在城市和省 / 市 / 自治区。

publishers 定义如下：

```
create table publishers
(pub_id char(4) not null,
pub_name varchar(40) not null,
city varchar(20) null,
state char(2) null,
unique nonclustered (pub_id))
```

其 pub_idrule 规则定义如下：

```
create rule pub_idrule
as @pub_id in
("1389", "0736", "0877", "1622", "1756")
or @pub_id like "99[0-9][0-9]"
```

authors 表

authors 表包含 pubs3 数据库中的姓名、电话号码和其它有关作者的信息。

authors 定义如下：

```
create table authors
(au_id id not null,
au_lname varchar(40) not null,
au_fname varchar(20) not null,
phone char(12) not null,
address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null,
unique nonclustered (au_id))
```

其对于 au_lname 和 au_fname 列的非聚簇索引定义如下：

```
create nonclustered index aunmind
on authors (au_lname, au_fname)
```

phone 列有以下缺省值：

```
create default phonedflt as "UNKNOWN"
sp_bindefault phonedft, "authors.phone"
```

以下视图使用 authors：

```
create view titleview
as
select title, au_ord, au_lname,
price, num_sold, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

titles 表

titles 表包含 pubs3 数据库中书目的名称、书目 ID、类型和其它信息。

titles 定义如下：

```
create table titles
(title_id tid not null,
title varchar(80) not null,
type char(12) not null,
pub_id char(4) null
    references publishers(pub_id),
price money null,
advance numeric(12,2) null,
num_sold int null,
notes varchar(200) null,
pubdate datetime not null,
contract bit not null,
unique nonclustered (title_id))
```

其对于 title 列的非聚簇索引定义如下：

```
create nonclustered index titleind
on titles(title)
```

其 title_idrule 定义如下：

```
create rule title_idrule
as
@title_id like "BU[0-9][0-9][0-9][0-9]" or
@title_id like "[MT]C[0-9][0-9][0-9][0-9]" or
@title_id like "P[SC][0-9][0-9][0-9][0-9]" or
@title_id like "[A-Z][A-Z]xxxx" or
@title_id like "[A-Z][A-Z]yyyy"
```

type 列有以下缺省值：

```
create default typedflt as "UNDECIDED"
sp_bindefault typedflt, "titles.type"
```

pubdate 列有以下缺省值：

```
create default datedflt as getdate()
sp_bindefault datedflt, "titles.pubdate"
```

titles 使用以下触发器：

```
create trigger deltitle
on titles
for delete
as
```

```
if (select count(*) from deleted, salesdetail
where salesdetail.title_id = deleted.title_id) >0
begin
   rollback transaction
   print "You can't delete a title with sales."
end
```

以下视图使用 titles：

```
create view titleview
as
select title, au_ord, au_lname,
price, num_sold, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

titleauthor 表

titleauthor 表包含 pubs3 数据库中的书目和作者 ID、版税百分比和有关书目和作者的其它信息。

titleauthor 定义如下：

```
create table titleauthor
(au_id  id not null
    references authors(au_id),
title_id tid not null
    references titles(title_id),
au_ord tinyint null,
royaltyper int null)
```

其对于 au_id 列的非聚簇索引定义如下：

```
create nonclustered index auidind
on titleauthor(au_id)
```

其对于 title_id 列的非聚簇索引定义如下：

```
create nonclustered index titleidind
on titleauthor(title_id)
```

以下视图使用 titleauthor：

```
create view titleview
as
select title, au_ord, au_lname,
price, num_sold, pub_id
from authors, titles, titleauthor
```

```
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

以下过程使用 titleauthor：

```
create procedure byroyalty @percentage int
as
select au_id from titleauthor
where titleauthor.royaltyper = @percentage
```

salesdetail 表

salesdetail 表包含 pubs3 数据库中的书店 ID、订单号和其它销售详细信息。

salesdetail 定义如下：

```
create table salesdetail
(stor_id char(4) not null
    references sales(stor_id),
ord_num numeric(6,0)
    references sales(ord_num),
title_id tid not null
    references titles(title_id),
qty smallint not null,
discount float not null)
```

其对于 title_id 列的非聚簇索引定义如下：

```
create nonclustered index titleidind
on salesdetail (title_id)
```

其对于 stor_id 列的非聚簇索引定义如下：

```
create nonclustered index salesdetailind
on salesdetail (stor_id)
```

其 title_idrule 规则定义如下：

```
create rule title_idrule
as
@title_id like "BU[0-9][0-9][0-9][0-9]" or
@title_id like "[MT]C[0-9][0-9][0-9][0-9]" or
@title_id like "P[SC][0-9][0-9][0-9][0-9]" or
@title_id like "[A-Z][A-Z]xxxx" or
@title_id like "[A-Z][A-Z]yyyy"
```

salesdetail 使用以下触发器：

```
create trigger totalsales_trig on salesdetail
    for insert, update, delete
as
/* Save processing:return if there are no rows affected
*/
if @@rowcount = 0
    begin
      return
end
/* add all the new values */
/* use isnull:a null value in the titles table means
**              "no sales yet" not "sales unknown"
*/
update titles
    set num_sold = isnull(num_sold, 0) + (select
sum(qty)
    from inserted
    where titles.title_id = inserted.title_id)
    where title_id in (select title_id from inserted)
/* remove all values being deleted or updated */
update titles
    set num_sold = isnull(num_sold, 0) - (select
sum(qty)
    from deleted
    where titles.title_id = deleted.title_id)
    where title_id in (select title_id from deleted)
```

sales 表

sales 表包含 pubs3 数据库中的书店 ID、订单号和销售日期。

sales 定义如下：

```
create table sales
(stor_id char(4) not null
     references stores(stor_id),
ord_num numeric(6,0) identity,
date datetime not null,
unique nonclustered (ord_num))
```

stores 表

stores 表包含 pubs3 数据库中的书店 ID、书店名称和其它有关书店的信息。

stores 定义如下：

```
create table stores
(stor_id char(4) not null,
stor_name varchar(40) not null,
stor_address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null,
payterms varchar(12) null,
unique nonclustered (stor_id))
```

store_employees 表

store_employees 表包含 pubs3 数据库中的书店、员工和管理员 ID 以及其它有关书店员工的信息。

store_employees 定义如下：

```
create table store_employees
(stor_id char(4) null
     references stores(stor_id),
emp_id id not null,
mgr_id id null
     references store_employees(emp_id),
emp_lname varchar(40) not null,
emp_fname varchar(20) not null,
phone char(12) null,
address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode varchar(10) null,
unique nonclustered (emp_id))
```

roysched 表

roysched 表包含 pubs3 表中的书目 ID、版税百分比和其它有关书目版税的信息。

roysched 定义如下：

```
create table roysched
title_id tid not null
    references titles(title_id),
lorange int null,
hirange int null,
royalty int null)
```

其对于 title_id 列的非聚簇索引定义如下：

```
create nonclustered index titleidind
on roysched (title_id)
```

discounts 表

discount 表包含 pubs3 数据库中的折扣类型、书店 ID、数量和折扣百分比。

discounts 定义如下：

```
create table discounts
(discounttype varchar(40) not null,
stor_id char(4) null
     references stores(stor_id),
lowqty smallint null,
highqty smallint null,
discount float not null)
```

blurbs 表

blurbs 表包含 pubs3 数据库中书籍的作者 ID 和短评。

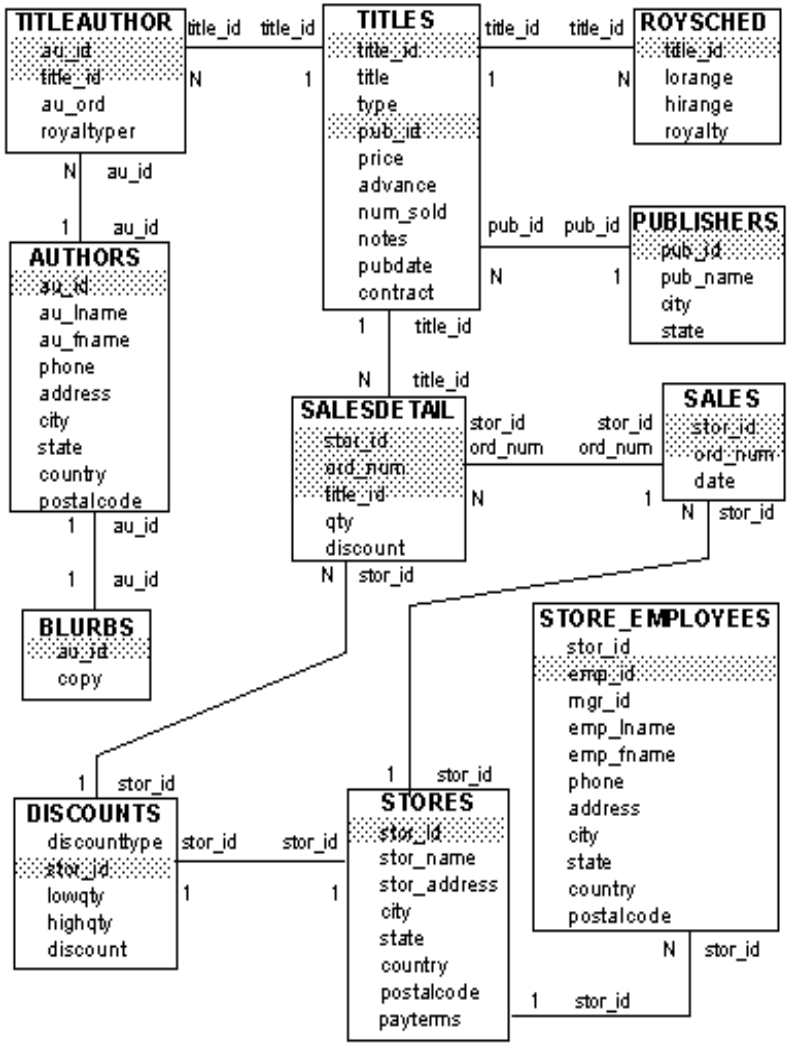
blurbs 定义如下：

```
create table blurbs
(au_id  id not null
    references authors(au_id),
copy text null)
```

pubs3 数据库框图

此框图显示 pubs3 数据库中的表以及它们之间的某些关系。

图 B-1：pubs3 数据库框图



索引

符号

英文

A

B

C

D

E

F

G

H

J

K

L

M

N

P

Q

R

S

T

W

X

Y

Z