Sybase® Adaptive Server™ Enterprise
参考手册

第二卷：命令

Adaptive Server Enterprise 版本 12
文档 ID：37418-01-1200-01
最后修订日期：1999 年 10 月

主要作者：Enterprise Data Studios Publications

参与作者：Anneli Meyer、 Evelyn Wheeler

文档 ID：37418-01-1200-01

本手册适用于 Sybase 数据库管理软件的 Adaptive Server Enterprise 版本 12 及所有后续版本，除非在新版本或技术说明中另有指定。本文档的信息若有变动，恕不另行通知。本手册中所述的软件按许可协议提供，因此必须按照协议条款来使用或复制该软件。

文档订购

要订购附加文档，美国和加拿大的客户可拨打客户服务部门电话 (800) 685-8225 或发传真至 (617) 229-9845。

其它国家的客户如持有美国许可协议，也可通过上述传真号与客户服务部门联系。所有其他国际客户应与 Sybase 子公司或当地分销商联系。

升级产品仅在软件的定期发布日提供。

Sybase 商标

有限权利

目录

关于本手册

6. Transact-SQL 命令

如需索引，参见第四卷 “表格和参考手册索引”。

图目录

表目录

关于本手册

*Adaptive Server Enterprise 参考手册*共有四卷，它是一本有关 Sybase® Adaptive Server™ Enterprise 和 Transact-SQL® 语言的指南。

第一卷 “*构件块*”介绍 Transact-SQL 的 “构件”：数据类型、内部函数、表达式和标识符、 SQLSTATE 错误，以及保留字。要成功地使用 Transact-SQL，您首先需要理解每个构件块的目的，并明确它们的使用会如何影响 Transact-SQL 语句的结果。

第二卷 “*命令*”提供有关创建语句时所使用的 Transact-SQL 命令的参考信息。

第三卷 “*过程*”提供有关系统过程，分类存储过程，扩展存储过程和 **dbcc** 存储过程的参考信息。所有过程都是使用 Transact-SQL 语句创建的。

第四卷 “*表格和参考手册索引*”提供有关系统表的参考信息。系统表中存储了有关服务器、数据库、用户的信息以及其它信息。它还提供 *dbccdb* 和 *dbccalt* 数据库中表的信息。此卷中还有一个索引，它包括了全部四卷中的主题。

有关本手册所针对的读者、相关文档、其它信息来源、本手册中使用的约定以及帮助的信息，参见第一卷中的 “关于本手册”。

如何使用本手册

本手册包括：

- 第 6 章 “Transact-SQL 命令”，提供每个 Transact-SQL 命令的参考信息。特别复杂的命令 （如 **select**）将分为几个小节来介绍。例如， **compute** 子句和 **select** 命令的 **group by** 及 **having** 子句都有相应的参考页。

6 Transact-SQL 命令

本章说明用于构造 Transact-SQL 语句的命令、子句和其它元素。

概述

表 6-1 将对本章中的命令进行简要说明。

表 6-1：Transact-SQL 命令

命令	说明
alter database	增加分配给数据库的空间量。
alter role	定义角色之间的互斥关系以及为角色添加、删除和更改口令。
alter table	添加新列；添加、更改或删除约束，更改约束；对现有表进行分区或取消分区。
begin...end	包括一系列 SQL 语句，以便使控制流语言 （例如 **if...else**）能够影响整个组的性能。
begin transaction	标记用户定义事务的起点。
break	导致从 **while** 循环退出。 **break** 通常由 **if** 测试激活。
case	用于写入条件值的 SQL 表达式。在任何可以使用值表达式的地方都可以使用 **case** 表达式。
checkpoint	将所有**脏**页 （自上次写入以来被更新的页）写入数据库设备。
close	使游标失效。
coalesce	用于写入条件值的 SQL 表达式。在任何可以使用值表达式的地方都可以使用 **coalesce** 表达式；它是 **case** 表达式的替代表达式。
commit	标记用户定义事务的终点。
compute Clause	生成摘要值，这些值在查询结果中显示为附加行。
connect to...disconnect	指定需要直通连接的服务器。
continue	使 **while** 循环重新启动。 **continue** 通常由 **if** 测试激活。
create database	创建新的数据库。
create default	如果在插入时没有显式提供值，则指定要在列 （或具有用户定义数据类型的所有列）中插入的值。
create existing table	确认当前远程表信息与存储在 *column_list* 中的信息相匹配，并验证基础对象是否存在。

表 6-1：Transact-SQL 命令 （续）

命令	说明
create index	创建表中一个或多个列的索引。
create plan	创建抽象查询计划。
create procedure	创建可以使用一个或多个用户提供参数的存储过程。
create proxy_table	创建代理表而无需指定列表。组件集成服务使用从远程表获取的元数据来得到列的列表。
create role	创建用户定义的角色。
create rule	为特定列或具有用户定义数据类型的任意列指定可接受值的域。
create schema	为数据库用户创建新的表、视图和权限集合。
create table	创建新表和可选的完整性约束。
create trigger	创建触发器，即一种经常用于强制完整性约束的存储过程。当用户试图在指定表上使用指定的数据修改语句时，触发器就会自动执行。
create view	创建视图，这是查看一个或多个表中数据的备选方法。
dbcc	数据库一致性检查程序 **(dbcc)** 检查数据库的逻辑和物理一致性。定期使用 **dbcc** 来进行周期性检查，或者在怀疑数据库损坏时用它来检查。
deallocate cursor	使游标不可访问并释放所有提交到该游标的内存资源。
declare	声明批处理或过程的局部变量的名称和类型。
declare cursor	定义游标。
delete	删除表中的行。
delete statistics	删除 *sysstatistics* 系统表中的统计信息。
disk init	使物理设备或文件可用于 Adaptive Server。
disk mirror	创建在主设备失败时可以立即取代的软件镜像。
disk refit	从 *sysdevices* 所包含的信息中重建 *master* 数据库的 *sysusages* 和 *sysdatabases* 系统表。在 **disk reinit** 之后使用 **disk refit**，以此作为 *master* 数据库恢复过程的一部分。
disk reinit	重建 *master* 数据库的 *sysdevices* 系统表。使用 **disk reinit**，以此作为 *master* 数据库恢复过程的一部分。
disk remirror	在磁盘镜像因被镜像的设备出现故障而停止，或是临时被 **disk unmirror** 命令禁用之后，重新启用磁盘镜像。
disk unmirror	禁用初始设备或其镜像，以便进行硬件维护或更改硬件设备。
drop database	从 Adaptive Server 中删除一个或多个数据库。

表 6-1：Transact-SQL 命令 （续）

命令	说明
drop default	删除用户定义的缺省值。
drop index	删除当前数据库中的表的索引。
drop procedure	删除用户定义的存储过程。
drop role	删除用户定义的角色。
drop rule	删除用户定义的规则。
drop table	从数据库中删除表定义及其所有数据、索引、触发器和权限说明。
drop trigger	删除触发器。
drop view	从当前数据库中删除一个或多个视图。
dump database	以可通过 **load database** 来读入的形式备份整个数据库，包括事务日志。转储和装载均通过 Backup Server 执行。
dump transaction	复制事务日志并删除不活动的部分。
execute	运行系统过程、用户定义的存储过程或动态构造的 Transact-SQL 命令。
fetch	从游标结果集中返回一个行或一组行。
goto 标签	分支到用户定义的标签。
grant	向用户或用户定义的角色分配权限。
group by 和 having 子句	用于在 **select** 语句中将表分为多个组，并且只返回与 having 子句中的条件匹配的组。
if...else	施加执行 SQL 语句的条件。
insert	向表或视图中添加新行。
kill	注销进程。
load database	装载用户数据库 （包括其事务日志）的备份副本。
load transaction	装载事务日志的备份副本。
lock table	显式锁定事务中的表。
nullif	用于写入条件值的 SQL 表达式。在任何可以使用值表达式的地方都可以使用 **nullif** 表达式；它是 **case** 表达式的替代表达式。
online database	在完成常规装载序列后将数据库标记为可公用，并且在需要时将装载的数据库和事务日志转储升级到当前版本的 Adaptive Server。
open	打开要进行处理的游标。
order by Clause	按排序顺序在指定列中返回查询结果。

表 6-1：Transact-SQL 命令 （续）

命令	说明
prepare transaction	由 DB-Library™ 在两阶段提交应用中用来查看服务器是否已准备好提交事务。
print	在用户屏幕上输出用户定义的消息。
quiesce database	挂起并重新开始对指定数据库列表进行的更新。
raiserror	在用户屏幕上输出用户定义的错误消息，并通过设置系统标识来记录已发生错误情况。
readtext	从指定的偏移开始，读取指定的字节数或字符数的 *text* 和 *image* 值。
reconfigure	**reconfigure** 命令当前不具有任何作用；将其包括在内是为了使现有脚本不经修改即可运行。在以前的版本中，执行 **sp_configure** 系统过程之后需要使用 **reconfigure** 来实施新的配置参数设置。
remove java	从数据库中删除一个或多个 Java-SQL 类、软件包或 JAR。在数据库中启用 Java 时使用。
reorg	根据使用的选项，回收页面上未使用的空间、删除行转移或将表中的所有行重新写入新页。
return	从批处理或过程中无条件退出，同时 （可选）提供返回状态。 **return** 之后的语句不会被执行。
revoke	从用户或角色中撤消权限或角色。
rollback	将用户定义的事务回退到事务内的上一个保存点或事务的起始点。
rollback trigger	回退用触发器完成的工作 （包括引发触发器的更新），并且发出可选的 **raiserror** 语句。
save transaction	在事务内设置保存点。
select	从数据库对象中检索行。
set	设置用户工作会话期间的 Adaptive Server 查询处理选项。可用于设置触发器或存储过程内的某些选项。还可用于激活当前会话中的角色或使其失效。
setuser	允许数据库所有者充当其它用户。
shutdown	关闭 Adaptive Server 或 Backup Server™。只有系统管理员才能发出该命令。
truncate table	删除表中所有的行。
union 运算符	返回一个包括两个或更多查询的结果的结果集。除非指定了 **all** 关键字，否则将从结果集中删除重复行。

表 6-1：Transact-SQL 命令 （续）

命令	说明
update	通过添加数据或修改现有数据来更改现有行中的数据；更新给定表的所有统计信息；更新已分区表中每个分区包含的页数的信息；更新有关指定索引中的键值分布的信息。
use	指定要使用的数据库。
waitfor	指定用于执行语句块、存储过程或事务的具体时间、时间间隔或事件。
where 子句	设置 **select**、 **insert**、 **update** 或 **delete** 语句中的搜索条件。
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。
writetext	允许对现有的 *text* 或 *image* 列进行不记日志的交互式更新。

alter database

功能

增加分配给数据库的空间量。

语法

```
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]
   [for proxy_update]]
```

关键字和选项

database_name — 是数据库的名称。数据库名可为文字、变量或存储过程参数。

on — 表示数据库扩展的大小和 / 或位置。如果日志和数据在不同的设备片段上，则对数据设备使用该子句，对日志设备使用 **log on** 子句。

default — 表示 **alter database** 可在任何缺省数据库设备（如 **sp_helpdevice** 所示）上放置数据库扩展。要指定数据库扩展的大小但不指定确切位置，使用下面命令：

```
on default = size
```

要更改数据库设备的状态为缺省状态，使用系统过程 **sp_diskdefault**。

database_device — 是放置数据库扩展的数据库设备的名称。一个数据库可以占用多个数据库设备，每个设备具有不同的空间量。用 **disk init** 向 Adaptive Server 添加数据库设备。

size — 是分配给数据库扩展的空间量（用兆字节表示）。最小的扩展是 1MB（512 个 2K 页）。缺省值是 2MB。

log on — 表示要为数据库的事务日志指定额外空间。**log on** 子句使用与 **on** 子句相同的缺省值。

with override — 强制 Adaptive Server 接受设备规格，即使这些规则混淆了同一设备上的数据和事务日志，并由此危及了数据库随时恢复的能力。如果试图在同一设备上混合日志和数据而不使用该子句，则 **alter database** 命令执行失败。如果混合了日志和数据，并使用 **with override**，尽管会受到警告，但是命令仍成功执行。

for load — 只在 **create database for load** 后使用，此时必须重新创建数据库（从转储进行装载）的空间分配和片段使用。

for proxy_update — 强制对代理数据库中的代理表重新进行同步。

示例

1.
```
alter database mydb
```

   为缺省数据库设备上的数据库 *mydb* 增加 1MB 空间。

2.
```
alter database pubs2
on newdata = 3
```

   在名为 *newdata* 的数据库设备上为分配给 *pubs2* 数据库的空间增加 3MB 大小。

3.
```
alter database production
on userdata1 = 10
log on logdev = 2
```

   为 *userdata1* 上的数据增加 10MB 的空间，并为 *logdev* 上的日志增加 2MB 的空间。

注释

限制

- 要使用 **alter database**，必须使用 *master* 数据库或在 *master* 数据库中执行存储过程。
- 如果 Adaptive Server 不能分配所请求空间，但它会尽可能地为每个设备分配接近请求的空间，并输出消息说明已经在每个数据库设备上分配的空间量。
- 只能在主设备上扩展 *master* 数据库。试图使用 **alter database** 把 *master* 数据库扩展到任何其它数据库设备将产生错误消息。下例是在主设备上修改 *master* 数据库的正确语句：

  ```
  alter database master on master = 1
  ```

- 任何数据库的最大设备片段数是 128。每次用 **create database** 或 **alter database** 在数据库设备上分配空间时，该分配表示一个设备片段，并作为 *sysusages* 中的一行输入。

- 如果在正在进行转储的数据库上使用 **alter database**，则 **alter database** 命令在转储结束后才能够完成。Adaptive Server 在转储时锁定数据库空间使用的内存映射。如果在内存映射被锁定时发出 **alter database** 命令，Adaptive Server 在转储完成后从磁盘更新映射。如果中断 **alter database**，Adaptive Server 将指导您运行 **sp_dbremap**。如果运行 **sp_dbremap** 失败，则在下一次重新启动前 Adaptive Server 不能使用已添加的空间。
- 可在脱机数据库上使用 **alter database on** *database_device*。

分配更多空间后备份 *master*

- 在每次使用 **alter database** 之后，用 **dump database** 命令来备份 *master* 数据库。这样会使 *master* 被破坏后恢复起来更加容易和安全。
- 如果使用 **alter database** 但无法备份 *master*，可用 **disk refit** 恢复更改。

在不同的设备上放置日志

- 如果已使用 **create database** 的 **log on** 扩展，要增加分配给事务日志的存储空间量，则在发出 **alter database** 命令时为 **log on** 子句中的日志设备赋予名称。
- 如果没有使用 **create database** 的 **log on** 扩展将日志放置在不同的设备上，则一旦硬盘发生崩溃就无法完全恢复。在这种情况下，可使用带有 **log on** 子句的 **alter database** 来扩展日志，然后使用 **sp_logdevice**。

获得有关空间使用的帮助

- 要查看数据库正在使用的设备片段的名称、大小和用法，执行 **sp_helpdb** *dbname*。
- 要查看当前数据库正在使用的空间量，执行 **sp_spaceused**。

system 和 *default* 段

- *system* 和 *default* 段被映射到 **alter database** 命令的 **on** 子句所包括的每一个新的数据库设备上。要取消映射这些段，使用 **sp_dropsegment**。
- 当使用 **alter database**（没有 **override**）在数据库正在使用的设备上扩展该数据库时，映射到该设备的段同时也得到扩展。如果使用了 **override** 子句，所有在 **on** 子句中命名的设备片段成为系统 / 缺省段，所有在 **log on** 子句中命名的设备片段成为日志段。

使用 *alter database* 唤醒处于 sleep 状态的进程

- 如果用户进程因为已经在日志段上达到最后机会阈值而被挂起，则使用 **alter database** 为日志段添加空间。当可用空间量超过了最后机会阈值时，进程被唤醒。

使用 *for proxy_update*

- 如果输入的 **for proxy_update** 子句没有其它选项，则不能扩展数据库的大小；但是，可以从代理数据库中删除代理表（如果有），并用从指定路径名获得的元数据重新创建代理表（该路径名在 **create database ... with default_location = 'pathname'** 期间指定）。
- 如果使用该命令和其它选项扩展数据库大小，则在实现大小扩展之后执行代理表同步。
- **alter database** 扩展的目的是为 DBA 提供使用方便的单步操作，通过该操作可在单个远程站点获得所有表的精确且最新的代理表示。
- 所有的外部数据源都支持此重新同步，而不仅仅是 HA 集群环境中的主服务器。而且，不需要用 **for proxy_update** 子句来创建数据库。如果用 **create database** 命令或 **sp_defaultloc** 指定了缺省存储位置，数据库中包含的元数据就可与远程存储位置上的元数据进行同步。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，数据库所有者具有 **alter database** 权限。系统管理员也可以变更数据库。

参见

命令	**create database**、**disk init**、**drop database**、**load database**
系统过程	**sp_addsegment**、**sp_dropsegment**、**sp_helpdb**、**sp_helpsegment**、**sp_logdevice**、**sp_renamedb**、**sp_spaceused**

alter role

功能

定义角色间的互斥性关系；添加、删除和更改角色的口令；指定口令有效期、口令最小长度和允许指定角色登录失败的最多次数。

语法

```
alter role role1 { add | drop } exclusive { membership
   | activation } role2

alter role role_name [add passwd "password" |
   drop passwd] [lock | unlock]

alter role { role_name | "all overrides" }
   set { passwd expiration | min passwd length |
   max failed_logins } option_value
```

关键字和选项

role1 — 是互斥性关系中的一个角色。

add — 用于在互斥性关系中添加角色；用于向角色添加口令。

drop — 用于删除互斥性关系中的角色；用于删除角色的口令。

exclusive — 使两个指定的角色相互排斥。

membership — 不允许同时授予用户两个角色。

activation — 允许同时授予一个用户两个角色，但不允许该用户同时激活两个角色。

role2 — 是互斥性关系中的另一个角色。

role_name — 是要为其添加、删除或更改口令的角色的名称。

passwd — 给角色添加口令。

password — 是向角色添加的口令。口令的长度必须至少为 6 个字符，并且必须遵循标识符的规则。口令不能使用变量。

lock 锁定指定角色。

unlock 解锁指定角色。

all overrides 将以后的设置应用于整个服务器而不是特定的角色。

set 激活它后面的选项。

passwd expiration 指定口令的有效期（以天数表示）。它可以是 0 到 32767 之间的任意值，包括 0 和 32767。

min passwd length 指定特定口令所允许的最小长度。

max failed_logins 指定允许按指定口令登录失败的最多次数。

option_value 指定 **passwd expiration**、**min passwd length** 或 **max failed_logins** 的值。要设置 **all overrides**，将 *option_value* 的值设置为 -1。

示例

1. **alter role intern_role add exclusive membership specialist_role**

   定义 *intern_role* 和 *specialist_role* 为互斥关系。

2. **alter role specialist_role add exclusive membership intern_role**
   alter role intern_role add exclusive activation surgeon_role

   定义角色在成员资格级别和激活级别上互斥。

3. **alter role doctor_role add passwd "physician"**

   为现有角色添加口令。

4. **alter role doctor_role drop passwd**

   删除现有角色的口令。

5. **alter role physician_role lock**

   锁定角色 *physician_role*。

6. **alter role physician_role unlock**

   解锁角色 *physician_role*。

7. **alter role physician_role set max failed_logins 5**

   将允许 *physician_role* 登录失败的最多次数更改为 5 次。

8. **alter role physician_role set min passwd length 5**

   将现有角色 *physician_role* 的最短口令长度设置为五个字符。

9. **alter role "all overrides" set min passwd length -1**

   替换所有角色的最短口令长度。

10. **alter role "all overrides" set max failed_logins -1**

    删除所有角色的最大登录失败次数的替换值。

注释

- **alter role** 命令定义角色之间的互斥性关系，并添加、删除和更改角色的口令。
- 有关变更角色的详细信息，参见*系统管理指南*。
- **all overrides** 参数删除使用 **sp_configure** 和下列任何参数设置的系统替换：
  - **passwd expiration**
  - **max failed_logins**
  - **min passwd length**

  删除角色口令可以删除对口令到期和允许登录失败最多次数的选项的替换。

互斥角色

- 不需要指定互斥性关系的角色或任何特定顺序的角色层次。
- 可使用具有角色层次的互斥性对用户定义角色强加约束。
- 互斥性成员资格比互斥性激活的限制更强。如果定义两个角色在成员资格级别上互斥，它们在激活级别上也隐式互斥。
- 如果定义两个角色在成员资格级别上互斥，那么定义它们在激活级别上的互斥性不会影响对成员资格级别的定义。添加和删除激活级别的互斥性不受成员资格级别的互斥性的影响。
- 授予用户或角色这两个角色后，就不能将它们定义为互斥。在试图定义两个角色在成员资格级别上互斥前，首先要撤消现有被授权者的任何一个授权角色。
- 如果定义两个角色在激活级别上互斥，系统安全员可将两个角色指派给同一用户，但用户不能同时激活两个角色。
- 如果系统安全员定义两个角色在激活级别上互斥，而且用户已经激活了两个角色，或在缺省情况下，已经设置两个角色在登录时激活，Adaptive Server 将使这两个角色互斥，但会发出一条警告消息，指明发生角色冲突的特定用户的名称。用户已经激活的角色不能更改。

更改角色口令

- 要更改角色口令，先删除现有口令，然后添加新口令，如下所示：

```
alter role doctor_role drop passwd
alter role doctor_role add passwd "physician"
```

➤ ***注意***

附加给用户定义角色的口令未到期。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

只有系统安全员才能执行 **alter role**。

参见

命令	**create role**、 **drop role**、 **grant**、 **revoke**、 **set**
函数	**mut_excl_roles**、 **proc_role**、 **role_contain**、 **role_id**、 **role_name**
系统过程	**sp_activeroles**、 **sp_displaylogin**、 **sp_displayroles**、 **sp_modifylogin**

alter table

功能

添加新列；删除、修改现有列；添加、更改或删除约束；对现有表进行分区或取消分区；更改现有表的锁方案；当使用 **alter table** 创建基于索引的参照完整性约束时，指定索引顺序为升序或降序；指定填充页与空白页的比率，以减少存储分段。

语法

```
alter table [database.[owner].]table_name

{add column_name datatype
    [default {constant_expression | user | null}]
    {identity | null | not null}
    [off row | in row]
    [ [constraint constraint_name]
      { { unique | primary key }
          [clustered | nonclustered] [asc | desc]
          [with { { fillfactor = pct
                    | max_rows_per_page = num_rows }
                    , reservepagegap = num_pages }]
          [on segment_name]
        | references [[database.]owner.]ref_table
           [(ref_column)]
        | check (search_condition) ] ... }
     [, next_column]...

| add { [constraint constraint_name]
      { {unique | primary key}
           [clustered | nonclustered]
           (column_name [asc | desc]
                [, column_name [asc | desc]...])
        [with { { fillfactor = pct
                | max_rows_per_page = num_rows}
                , reservepagegap = num_pages}]
        [on segment_name]
   | foreign key (column_name [{, column_name}...])
        references [[database.]owner.]ref_table
            [(ref_column [{, ref_column}...])]
    | check (search_condition)}

| drop {[column_name [, column_name]] |
    [constraint constraint_name]}
```

```
| modify column_name {[datatype] [null] |
    [not null]] [, column_name]

| replace column_name
    default {constant_expression | user | null}

| partition number_of_partitions

| unpartition

| enable | disable trigger

| lock {allpages | datarows | datapages } }

| with exp_row_size = num_bytes
```

关键字和选项

table_name — 是要更改的表的名称。如果该表位于另一数据库中，则指定数据库名，并且在数据库中存在多个使用该名称的表时，指定所有者的名称。*owner* 的缺省值是当前用户，而 *database* 的缺省值是当前数据库。

add — 指定要添加到表中的列或约束的名称。

如果启用了组件集成服务，则不能将 **add** 用于远程服务器。

column_name — 是表中列的名称。如果数据库中已启用 Java，则列可以是 Java-SQL 列。

datatype — 是除了 *bit* 之外的任何系统数据类型，或是除了那些基于 *bit* 的数据类型之外的任何用户定义数据类型。

如果数据库中已启用 Java，数据库中安装的 Java 类的名称属于系统类或用户定义类。有关详细信息，参见 *Adaptive Server Enterprise 中的 Java*。

default — 指定列的缺省值。如果指定了值，并且用户在插入数据时没有提供该列的值，则 Adaptive Server 将插入缺省值。缺省值可以是 *constant_expression*、**user**（插入正在插入数据的用户的名称）或 **null**（插入空值）。

Adaptive Server 生成缺省的名称，其形式为 *tabname_colname_objid*，其中 *tabname* 是表名的前 10 个字符，*colname* 是列名的前 5 个字符，*objid* 是缺省的对象 ID 号。设置缺省值为 **null** 将删除缺省值。

如果启用了组件集成服务，则不能将 **default** 用于远程服务器。

constant_expression — 是用作列的缺省值的常量表达式。它不能包括任何列或其它数据库对象的名称，但可包括内部函数。该缺省值必须与列的数据类型兼容。

user — 指定 Adaptive Server 在用户未提供值时应插入作为缺省的用户名。列的数据类型必须是 *char*(30)、*varchar*(30) 或是 Adaptive Server 隐式转换为 *char* 的类型；然而，如果数据类型不是 *char*(30) 或 *varchar*(30)，可能会发生截断。

null | **not null** — 指定没有缺省值的情况下 Adaptive Server 在数据插入期间的行为。

null 指定添加的列允许有空值。如果用户未提供值，则 Adaptive Server 在插入时指派空值。

not null 指定添加的列不允许有空值。如果不存在缺省值，用户在插入时必须提供非空值。

如果未指定 **null** 或 **not null**，缺省情况下 Adaptive Server 使用 **not null**。然而，可使用 **sp_dboption** 切换该缺省值，以使缺省值与 SQL 标准兼容。如果为新添加的列指定（或隐含）**not null**，则需要缺省子句。缺省值不仅用于新添加列的所有现有行，还适用于将来插入的行。

identity — 表示列具有 IDENTITY 属性。数据库中的每个表可以有一个 *numeric* 类型且标度为零的 IDENTITY 列。IDENTITY 列不可更新且不允许有空值。

IDENTITY 列存储顺序号，例如由 Adaptive Server 自动生成的发票号或职员号。IDENTITY 列的值唯一地标识表的每一行。

如果启用了组件集成服务，则不能将用于 **identity** 远程服务器。

off row | **in row** — 指定 Java-SQL 列是与行分别存储，还是存储到直接在行中分配的存储空间。

in row 列的存储大小不能超过 255 个字节。缺省值是 **off row**。

constraint — 引入完整性约束的名称。

如果启用了组件集成服务，则不能将 **constraint** 用于远程服务器。

constraint_name — 是约束的名称。它必须遵循标识符的规则，且在数据库中是唯一的。如果未指定表级约束的名称，Adaptive Server 生成形式为 *tabname_colname_objectid* 的名称，其中 *tabname* 是表名的前 10 个字符，*colname* 是列名的前 5 个字符，*objectid* 是约束的对象 ID 号。如果未指定唯一约束或主键约束的名称，Adaptive Server 生成格式为 *tabname_colname_tabindid* 的名称，其中 *tabindid* 是表 ID 和索引 ID 的字符串并置。

约束不适用于在添加约束时就已经在表中存在的数据。

unique — 约束指定列中的值，这样任何两行都不会具有相同的非空值。该约束创建了唯一索引，只有删除约束后才能删除这个索引。该选项不能同上面所述的 **null** 选项一同使用。

primary key — 约束指定列中的值，使得任何两行都不会具有相同的值，而且值不为 NULL。该约束创建了唯一索引，只有删除约束后才能删除这个索引。

clustered | **nonclustered** — 指定由 **unique** 或 **primary key** 约束创建的索引是集群或非集群索引。**clustered** 是主键约束的缺省值（除非表中已存在集群索引）； **nonclustered** 是唯一约束的缺省值。每个表只能有一个集群索引。有关详细信息，参见 **create index**。

fillfactor — 指定 Adaptive Server 根据现有数据创建新索引时对每页的填充程度。**fillfactor** 百分比仅在索引已创建后才有意义。随着数据的变化，页不会维持在某个特定的填充程度。

fillfactor 的缺省值是 0； **create index** 语句不包括 **with fillfactor** 时使用该缺省值（除非已经使用 **sp_configure** 更改了该值）。使用 1 到 100 之间的值指定 **fillfactor**。

fillfactor 为 0 时创建页完全填充的集群索引和叶页完全填充的非集群索引。它将在集群索引和非集群索引中的索引 B 树内都保留适当的空间量。很少需要更改 **fillfactor**。

如果设置 **fillfactor** 为 100，则在 Adaptive Server 创建的集群和非集群索引中每一页都百分之百地完全填充。如果 **fillfactor** 为 100，则仅对只读表（不再添加额外的数据的表）有意义。

fillfactor 值小于 100（除了 0 这个特例外）将导致 Adaptive Server 创建的新索引的页没有完全填充。在最终将保存大量数据的表上创建索引时，选择 **fillfactor** 为 10 是适当的，但是如果 **fillfactor** 值很小，将导致每个索引（或索引和数据）占用更多的存储空间。

◆ ***警告！***

用 fillfactor 创建集群索引会影响数据占用的存储空间量，因为 Adaptive Server 在创建集群索引时重新分配数据。

max_rows_per_page — 限制索引中数据页和叶级页的行数。与 **fillfactor** 不同， **max_rows_per_page** 值在用 **sp_chgattribute** 更改前保持不变。

如果没有指定 **max_rows_per_page** 的值， Adaptive Server 在创建索引时使用值 0。为数据页指定 **max_rows_per_page** 时，使用 0 到 256 之间的数值。非集群索引每页的最大行数取决于索引键的大小；如果指定值太高，Adaptive Server 就会返回错误消息。

对于用约束创建的索引，设置 **max_rows_per_page** 为 0 可以创建页完全填充的集群索引和叶页完全填充的非集群索引。设置为 0 将在集群索引和非集群索引中的索引 B 树内都保留适当的空间量。

如果设置 **max_rows_per_page** 为 1，Adaptive Server 创建集群和非集群的叶索引页，每页的每一行都处于叶级。可用它来减少对经常访问的数据的锁争用。

如果 **max_rows_per_page** 值很低，将导致 Adaptive Server 创建的新索引的页未完全填充，从而占用更多存储空间，并造成更多的页拆分。

◆ ***警告！***

用 max_rows_per_page 创建集群索引可能会影响数据占用的存储空间量，因为 Adaptive Server 在创建集群索引时重新分配数据。

on *segment_name* — 指定在已命名段上创建索引。使用 **on** *segment_name* 选项前，必须用 **disk init** 初始化设备，且必须用 **sp_addsegment** 系统过程向数据库添加段。有关数据库中可用段名的列表，请与系统管理员联系或使用 **sp_helpsegment**。

如果指定 **clustered** 并使用 **on** *segment_name* 选项，则整个表迁移到指定的段，因为索引的叶级包含实际数据页。

references — 为参照完整性约束指定列的列表。对一个列约束只能指定一个列值。在将该约束包括在引用其它表的表中时，插入到**引用**表的任何数据必须已经在**被引用**表中存在。

要使用该约束，必须具有对被引用表的 **references** 权限。被引用表上的指定列必须受唯一索引（由 **unique** 约束或 **create index** 语句创建）的约束。如果没有指定列，被引用表中的相应列必须有 **primary key** 约束。而且，引用表列的数据类型必须与被引用表列的数据类型完全匹配。

如果启用了组件集成服务，则不能将 **references** 用于远程服务器。

foreign key — 指定所列出的列是此表的外键，此表的匹配主键是在 **references** 子句中列出的列。

ref_table — 是含有被引用列的表的名称。可引用另一个数据库中的表。约束最多可引用 192 个用户表和内部生成的工作表。使用系统过程 **sp_helpconstraint** 检查表的参照约束。

ref_column — 是被引用表中一个或多个列的名称。

check — 指定 Adaptive Server 对表中所有行强制执行的 *search_condition* 约束。

如果启用了组件集成服务，则不能将 **check** 用于远程服务器。

search_condition — 是对列值定义 **check** 约束的布尔表达式。这些约束包括：

- 由 **in** 引入的常量表达式的列表。
- 由 **like** 引入的可以包含通配符的一组条件。

表达式可包括算术运算符和 Transact-SQL 函数。*search_condition* 不能包含子查询、集合函数、参数或主机变量。

next_column — 包括额外的列定义（用逗号分开），所使用的语法与描述列定义的语法相同。

drop — 指定要从表中删除的列或约束的名称。

如果启用了组件集成服务，则不能将 **drop** 用于为远程服务器。

modify — 指定要更改其数据类型或可为空性的列的名称。

replace — 指定要更改其缺省值的列，用来进行更改的新值由后面的 **default** 子句指定。

如果启用了组件集成服务，则不能将 **replace** 用于远程服务器。

partition *number_of_partitions* — 为表创建多个数据库页链。Adaptive Server 可对每个链的最后一页执行并发的插入操作。*number_of_partitions* 必须是大于或等于 2 的正整数。每个分区需要一个额外的控制页；磁盘空间不足会限制在表中创建的分区的数量。内存不足会限制可访问的已分区的表数。

如果启用了组件集成服务，则不能将 **partition** 用于远程服务器。

unpartition — 通过将后续页链和第一个页链并置创建表的单个页链。

如果启用了组件集成服务，则不能将 **unpartition** 用于远程服务器。

asc | desc — 指定是以升序 **(asc)** 还是降序 **(desc)** 的顺序创建索引。缺省为升序。

reservepagegap = *num_pages* — 指定在为索引（由约束创建）进行页片 I/O 分配操作期间填充页与要保留的空白页的比率。对于每个指定的 *num_pages*，保留一个空白页以用于表的未来扩展。有效值为 0 到 255。缺省值为 0，表示没有保留空白页。

lock datarows | datapages | allpages — 更改用于表的锁方案。

exp_row_size = *num_bytes* — 指定期望的行大小；仅适用于 **datarows** 和 **datapages** 锁方案、具有可变长度行的表，并且仅适用于 **alter table** 执行数据复制的时候。有效值是 0、1 和表长度最大值和最小值之间的任一值。缺省值是 0，表示应用全服务器范围的设置。

示例

1.
```
alter table publishers
add manager_name varchar(40) null
```

   向表中添加列。对表中现有的每一行，Adaptive Server 指派 NULL 列值。

2.
```
alter table sales_daily
add ord_num numeric(5,0) identity
```

   向表中添加 IDENTITY 列。对表中现有的每一行，Adaptive Server 指派唯一的顺序列值。注意，IDENTITY 列的类型是 *numeric* 且标度为零。精度决定可插入列的最大值（10^5 - 1 或 99,999）。

3.
```
alter table authors
add constraint au_identification
primary key (au_id, au_lname, au_fname)
```

   向 *authors* 表中添加主键约束。如果表中已经有主键约束或唯一约束，必须首先删除现有约束（参见示例 5）。

4.
```
alter table authors
add constraint au_identification
primary key (au_id, au_lname, au_fname)
with reservepagegap = 16
```

   在 *authors* 上创建一个索引；索引的 **reservepagegap** 值为 16，即在索引上每 15 个分配页保留 1 个空白页。

5.
```
alter table titles
drop constraint au_identification
```

   删除 *au_identification* 约束。

6.
```
alter table authors
replace phone default null
```

   删除 *authors* 表中 *phone* 列的缺省约束。如果列允许有 NULL 值，则在没有指定列值时插入 NULL。如果列不允许有 NULL 值，则无法进行未指定列值的插入。

7.
```
alter table titleauthor partition 5
```

   为 *titleauthor* 表创建四个新页链。对表分区后，现有数据保留在第一个分区中。而新行插入到所有五个分区中。

```
8. alter table titleauthor unpartition
   alter table titleauthor partition 6
```

并置 *titleauthor* 表的所有页链，然后重新将其分为六个分区。

```
9. alter table titles lock datarows
```

将 *titles* 表的锁方案更改为数据行锁。

```
10.alter table authors
       add author_type varchar(20)
       default “primary_author” not null
```

用 *primary_author* 的缺省值向 *authors* 表添加非空列 *author_type*。

```
11.alter table titles
       drop advance, notes, contract
```

从 *titles* 表中删除 *advance*、*notes* 和 *contract* 列。

```
12.alter table authors
       modify city varchar(30) null
```

将 *authors* 表的 *city* 列修改为具有缺省值 NULL 的 *varchar(30)*。

```
13.alter table stores
       modify stor_name not null
```

将 *stores* 表的 *stor_name* 列修改为 NOT NULL。注意，其数据类型 *varchar(40)* 保持不变。

```
14.alter table titles
       modify type varchar(10)
       lock datarows
```

修改 *titles* 表的 *type* 列，并将 *titles* 表的锁方案从所有页更改为数据行。

```
15.alter table titles
       modify notes varchar(150) not null
       with exp_row_size = 40
```

将 *titles* 表的 *notes* 列从 *varchar(200)* 修改为 *varchar(150)*，将缺省值从 NULL 更改为 NOT NULL，并指定 **exp_row_size** 为 40。

```
16.alter table titles
       add author_type varchar(30) null
       modify city varchar(30)
       drop notes
       add sec_advance money default 1000 not null
       lock datarows
       with exp_row_size = 40
```

添加、修改并删除列，然后在一个查询中添加另一列。变更锁方案并指定新列的 **exp_row_size**。

注释

- 如果使用 **select *** 的存储过程引用已经变更的表，即使使用了 **with recompile** 选项，新列也不会出现结果集中。必须删除该过程并对其重新创建以包含这些新列。

限制

- 不能向现有表添加数据类型为 *bit* 的列。
- 表的列数不能超过 250。每行的最大字节数取决于表的锁方案。在一个 allpage 锁定表中，用户数据的最大字节数是 1960 个字节。对于 DOL 锁定表，每个可变长度列或允许空值的列将扣除 2 个字节。

◆ ***警告！***

不要变更系统表。

- 不能对系统表或已分区的表进行分区。
- 不能在用户定义事务中发出带有 **partition** 或 **unpartition** 子句的 **alter table command**。

获取有关表的信息

- 有关表及其列的信息，使用 **sp_help**。
- 要重命名表，执行系统过程 **sp_rename**（不要重命名系统表）。
- 有关完整性约束（**unique**、**primary key**、**references** 和 **check**）或 **default** 子句的信息，参见本章中的 **create table**。

指定索引中的升序或降序顺序

- 在索引列名后使用 **asc** 和 **desc** 关键字指定索引的排序顺序。按查询的 **order by** 子句中指定的顺序创建索引，将消除查询过程中的排序步骤。有关详细信息，参见 *Performance and Tuning Guide* 中的“Indexing for Performance”。

使用跨数据库参照完整性约束

- 在创建跨数据库约束时，Adaptive Server 将下列信息存储到每一个数据库中的 *sysreferences* 系统表中：

表 6-2：所存储的有关参照完整性约束的信息

存储在 *sysreferences* 中的信息	包含被引用表信息的列	包含引用表信息的列
键列 ID	*refkey1* 至 *refkey16*	*fokey1* 至 *fokey16*
表 ID	*reftabid*	*tableid*
数据库 ID	*pmrydbid*	*frgndbid*
数据库名	*pmrydbname*	*frgndbname*

- 删除引用表或其数据库时，Adaptive Server 从被引用数据库中删除外键信息。
- 因为引用表依赖来自于被引用表的信息，所以 Adaptive Server 不允许：
  - 删除被引用表，
  - 删除包含被引用表的外部数据库，或
  - 用 **sp_renamedb** 重命名任一数据库。

  必须首先用 **alter table** 删除跨数据库约束。
- 每次添加或删除跨数据库约束时，或者是删除含有跨数据库约束的表时，转储**所有**受影响的数据库。

◆ ***警告！***

装载这些数据库的早期转储将引起数据库损坏。

- *sysreferences* 系统表存储外部数据库的 **name** 和 ID 号。如果使用 **load database** 更改数据库名称或将数据库装载到不同的服务器上，Adaptive Server 将无法确保参照完整性。

◆ ***警告！***

为了用不同名称装载数据库或将其移至另一个 Adaptive Server 上，在转储数据库之前，请使用 alter table 删除所有的外部参照完整性约束。

更改缺省值

- 可通过两种方法创建列缺省值：在 **create table** 或 **alter table** 语句中声明缺省的列约束，或者使用 **create default** 语句并用 **sp_bindefault** 将其绑定到列上来创建列缺省值。
- 不能替换用 **sp_bindefault** 绑定到列上的用户定义缺省值。先用 **sp_unbindefault** 对缺省值解除绑定。
- 如果用 **create table** 或 **alter table** 声明缺省列值，不能用 **sp_bindefault** 将缺省值绑定到该列。通过将其变更为 NULL 来删除缺省值，然后绑定用户定义的缺省值。将缺省值更改为 NULL 可对缺省值解除绑定，并将其从 *sysobjects* 表中删除。

设置用于索引的空间管理属性

- **alter table** 语句中的空间管理属性 **fillfactor**、**max_rows_per_page** 和 **reservepagegap** 适用于为 **primary key** 或 **unique** 约束创建的索引。如果约束在 allpage 锁定表上创建一个集群索引，这些空间管理属性将影响表的数据页。
- 使用 **sp_chgattribute to** 更改表或索引的 **max_rows_per_page** 或 **reservepagegap**，或存储 **fillfactor** 的值。
- 在以下情况下将应用索引的空间管理属性：
  - 使用 **alter table** 命令将表锁方案由 allpage 锁更改为 DOL 锁（或相反），从而重新创建了索引。有关详细信息，参见第 30 页上的“更改锁方案”。
  - 作为 **reorg rebuild** 命令的一部分，自动重新建立索引。
- 要查看表的当前有效空间管理属性，使用 **sp_help**。要查看索引的当前有效空间管理属性，使用 **sp_helpindex**。
- 空间管理属性 **fillfactor**、**max_rows_per_page** 和 **reservepagegap** 以下列方式帮助管理表和索引的空间使用：
  - **fillfactor** 在创建索引时在页面上保留额外的空间，但并不一直保持 **fillfactor**。此属性适用于所有锁方案。
  - **max_rows_per_page** 限制数据页或索引页上的行数。其主要用途是提高 allpage 锁定表中的并发性。
  - **reservepagegap** 指定空白页与填充页的比率，用于执行扩展分配的命令中。此属性适用于所有锁方案。

  可以为表和索引存储空间管理属性，以便在 alter table 和 reorg rebuild 命令执行期间应用这些属性。

- 表 6-3 显示了空间管理属性和锁方案的有效组合。如果 **alter table** 命令更改了表，并导致组合不兼容，则存储在系统表中的值仍保留，但在表操作时不会应用这些值。如果表锁方案的更改使属性生效，则应用这些属性。

表 6-3：空间管理属性和锁方案

参数	*allpage*	*数据页*	*数据行*
max_rows_per_page	是	否	否
reservepagegap	是	是	是
fillfactor	是	是	是
exp_row_size	否	是	是

- 表 6-4 显示了空间管理属性的缺省值以及使用缺省值的效果。

表 6-4：空间管理属性的缺省值和效果

参数	缺省值	使用缺省值的效果
max_rows_per_page	0	在页上尽可能装入更多的行，最多可达 255 行
reservepagegap	0	不保留间距
fillfactor	0	完全充满叶页

max_rows_per_page 到 *exp_row_size* 的转换

- 如果表有 **max_rows_per_page** 设置，且表由 allpage 锁转换为 DOL 锁，则在 **alter table...lock** 命令将表复制到新位置之前，**max_rows_per_page** 的值将被转换为 **exp_row_size** 的值。复制期间将强制使用 **exp_row_size**。表 6-5 显示了这些值的转换方式。

表 6-5：将 max_rows_per_page 转换为 exp_row_size

如果 *max_rows_per_page* 被设置为	设置 *exp_row_size* 为
0	通过 **default exp_row_size percent** 设置的百分比值
255	1，即完全充满页
1–254	以下各项中的较小值： • 最大行大小 • 2002 / **max_rows_per_page** 值

使用 *reservepagegap*

- 使用大量空间的命令通过分配页片而不是通过分配单个页来分配新的空间。**reservepagegap** 关键字使这些命令保留空白页，以便将来的页分配在将被拆分的页或有行被转移了的页附近进行。
- 表的 **reservepagegap** 值存储在 *sysindexes* 中，应用于以下情况：表的锁方案由 allpage 锁定更改为 DOL 锁（或相反）。要更改存储的值，应在运行 **alter table** 之前使用系统过程 **sp_chgattribute**。
- 在 allpage 锁定表上，使用 **clustered** 关键字指定的 **reservepagegap** 将覆盖以前使用 **create table** 或 **alter table** 指定的值。

为更有效地执行插入操作而将表分区

- 用 **alter table** 命令的 **partition** 子句对表进行分区可以创建更多的页链，使多个最终页可在任意给定时间用于并发的插入操作。这将减少对页的争用，而且当包含表的段分布于多个物理设备时，分区也会在服务器将数据从缓存刷新到磁盘时减少 I/O 争用，从而更有效地执行插入操作。
- 如果向已分区的表复制数据或从已分区的表中复制出数据，必须配置 Adaptive Server 以便进行并行处理。
- 对表分区时，Adaptive Server 为每个分区（包括第一个分区）分配一个控制页。现有的页链成为第一个分区的一部分。Adaptive Server 为每个后续分区创建首页。由于每个分区有自己的控制页，已分区的表需要的磁盘空间比未分区的表略多一些。
- 空表和包含数据的表都可分区。对表进行分区**不会**移动数据；现有数据保留在第一个分区中原来的存储位置。要得到最佳性能，在插入数据**之前**对表进行分区。
- 不能对系统表或已分区的表进行分区。可对含有 *text* 和 *image* 列的表进行分区；但是分区不影响 Adaptive Server 存储 *text* 和 *image* 列的方法。
- 对表进行分区后，不能对表使用 **truncate table** 命令或 **sp_placeobject** 系统过程。
- 要更改表的分区数，使用 **alter table** 的 **unpartition** 子句并置所有现存的页链，然后再使用 **alter table** 的 **partition** 子句对表重新进行分区。
- 取消对表分区时，重新编译任何相关过程的查询计划。取消分区不会自动重新编译过程。
- 用 **alter table** 命令的 **unpartition** 子句取消对表分区时，Adaptive Server 重新分配所有控制页（包括第一个分区的控制页），并且并置页链。产生的单个页链不含空页，但首页可能会出现例外。取消对表分区**不会**移动数据。

添加 IDENTITY 列

- 向表中添加 IDENTITY 列时，确保列的精度足够大，可以容纳现有的行数。如果行数超过 10 PRECISION - 1，Adaptive Server 将输出错误消息而不添加列。
- 向表中添加 IDENTITY 列时，Adaptive Server：
  - 锁定表，直到生成所有的 IDENTITY 列值。如果表包含大量的行，则该进程占用大量时间。
  - 从值 1 开始，为每个现有行指派唯一的顺序 IDENTITY 列值。
  - 记录每个插入到表中的操作。向含有大量行的表添加 IDENTITY 列之前，使用 **dump transaction** 清除数据库的事务日志。
- 每次向表插入行时，Adaptive Server 都生成一个 IDENTITY 列值，并且该值总比上一个值多 1。同 **alter table** 语句中声明的列缺省值或用 **sp_bindefault** 绑定到列上的缺省值相比，这个值具有更高的优先级。

变更表的模式

- **add**、**drop** 或 **modify** 以及 **lock** 从属子句对于更改现有表的模式很有用。单个语句包含这些从属子句的数量和顺序都不受限制，只要语句中对同一列名的引用不超过一次。
- 如果使用 **select *** 的存储过程引用已经变更的表，即使使用了 **with recompile** 选项，新列也不会出现结果集中。必须删除该过程并对其重新创建以包含这些新列。
- 不能删除表中的所有列。而且，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，就不能再删除剩余的一个列）。要从数据库中删除表，使用 **drop table**。
- 需要数据副本的情况：
  - 要删除列
  - 要添加 NOT NULL 列
  - 用于大多数 **alter table. . . modify** 命令

  使用 **showplan** 确定特定 **alter table** 命令是否需要数据副本。
- 当其它 **alter table** 命令需要数据副本时，可使用其它 **alter table** 命令（**add**、**drop** 或 **modify**）为已修改的表指定更改锁方案。
- **alter table** 执行数据复制时，必须在包含要更改模式的表的数据库中打开 **select into /bulkcopy/pllsort**。
- 当变更已分区的表的模式且更改需要数据副本时，必须配置 Adaptive Server 以便进行并行处理。

- 被修改的表保留现有的空间管理属性（**max_rows_per_page**、**fillfactor** 等等）和表的索引。
- 需要数据副本的 **alter table** 不引发触发器。
- 可使用 **alter table** 来更改由组件集成服务 (CIS) 创建并维护的远程代理表的模式。有关 CIS 的信息，参见 *Component Integration Services User's Guide*。
- 不能在同一语句中执行数据复制并添加表级约束或参照完整性约束。
- 不能在同一语句中执行数据复制并创建集群索引。
- 如果添加了 NOT NULL 列，必须也指定缺省子句。
- 在 allpage 锁定表中始终可以添加、删除或修改列。然而，在 DOL 锁定表中添加、删除或修改列却有限制，如表 6-6 所示：

表 6-6：什么情况下可在 DOL 锁定表中添加、删除或修改列?

索引类型	allpage 锁定的已分区表	allpage 锁定的未分区表	DOL 锁定的已分区表	DOL 锁定的未分区的表
集群	是	是	否	是
非集群	是	是	是	是

如果需要在包含集群索引的 DOL 锁定的已分区表中添加、删除或修改列，可以：

1. 删除集群索引。
2. 变更（DOL 锁定）表。
3. 重新创建集群索引。

- 不能添加 NOT NULL Java 对象作为列。缺省情况下，所有的 Java 列始终具有缺省值 NULL，并作为 *varbinary* 字符串或 *image* 数据类型进行存储。
- 如果修改需要数据副本，则不能修改包含 Java 列的已分区的表。相反，首先取消对表进行分区，运行 **alter table** 命令，然后对该表重新分区。
- 不能从索引或参照完整性约束中删除键列。要删除键列，首先删除索引或参照完整性约束，然后再删除键列。有关详细信息，参见 *Transact-SQL User's Guide*。
- 不能删除包含规则、约束或缺省值的列，或者由规则、约束或缺省引用的列。相反，首先删除规则、约束或缺省值，然后删除列。使用 **sp_helpconstraint** 标识对表的任何约束，并使用 **sp_depends** 标识任何列级依赖性。

- 不能删除系统表的列。而且，不能删除由 Sybase 提供的工具和存储过程创建和使用的用户表的列。
- 如果表为空，一般情况下可将现有列的数据类型修改为任何其它数据类型。如果表不为空，可将数据类型修改为可显式转换为原始数据类型的任何数据类型。
- 可以：
  - 添加新 IDENTITY 列。
  - 删除现有 IDENTITY 列。
  - 修改现有 IDENTITY 的大小。

  有关详细信息，参见 *Transact-SQL User's Guide*。
- 变更表的模式会增加模式计数，并导致访问该表的现有存储过程在下一次执行时重新进行规范化。与数据类型有关的存储过程或视图的更改会因数据类型规范化类型错误而失败。必须更新这些相关对象，使它们引用已修改的表模式。

修改表模式的限制

- 不能从事务中运行 **alter table**。
- 变更表的模式会导致使用 **bcp** 完成的备份无效。这些备份使用的表模式与表的当前模式不再兼容。
- 可添加包含检查约束的 NOT NULL 列，但是，Adaptive Server 不验证对现有数据的约束。
- 如果表具有集群索引而且操作需要数据副本，则不能使用 **alter table . . . add**、**drop** 或 **modify** 命令来更改表的锁方案。不过，可以

  1. 删除集群索引。
  2. 变更表的模式。
  3. 重新创建集群索引。
- 如果表中任何打开的游标处于活动状态，则不能变更表的模式。

修改 *text* 和 *image* 列的限制

- 只能添加接受空值的 *text* 或 *image* 列。

  要添加仅包含非空值的 *text* 或 *image* 列，首先添加仅接受空值的列，然后将其更新为非空值。
- 只能将列从 *text* 数据类型修改为下列数据类型：
  - *char*
  - *varchar*
  - *nchar*
  - *nvarchar*
- 只能将列从 *image* 数据类型修改为 *varbinary* 数据类型，且列只能包括非空数据。
- 只有当表为空时，才可将 *text* 或 *image* 列修改为任何其它数据类型。
- 可添加新的 *text* 或 *image* 列，然后在同一语句中删除现有的 *text* 或 *image* 列。
- 不能将列修改为 *text* 或 *image* 数据类型。

更改锁方案

- **alter table**支持从一个锁方案更改为任何其它锁方案。可以进行以下更改：
  - 从 **allpages** 更改为 **datapages** 或相反
  - 从 **allpages** 更改为 **datarows** 或相反
  - 从 **datapages** 更改为 **datarows** 或相反
- 从 allpage 锁方案更改为 DOL 锁方案（或相反）之前，使用 **sp_dboption** 将数据库选项 **select into/bulkcopy/pllsort** 设置为真，然后，在对任何表进行分区或索引排序需要并行排序时，在数据库上运行 **checkpoint**。
- 在将锁方案从 allpage 锁更改为 DOL 锁（或相反）之后，将禁止使用 **dump transaction** 命令来备份事务日志；必须先执行完整的数据库转储。
- 使用 **alter table...lock** 将表的锁方案从 allpage 锁更改为 DOL 锁（或相反）时，Adaptive Server 将复制表的数据页。在表所驻留的段上必须有足够的空间用于复制全部数据页。在索引所驻留的段上必须有空间用于重建索引。

DOL 锁定表的集群索引在数据页上有一叶级。如果将具有集群索引的表从 allpage 锁变更为 DOL 锁，由此得到的集群索引将要求更多的空间。所需的额外空间取决于索引键的大小。

使用 **sp_spaceused** 确定表当前占用的空间，使用 **sp_helpsegment** 查看可用来存储表的空间。

- 如果将表的锁方案由 allpage 锁更改为数据页锁（或相反），在复制数据行时将空间管理属性应用于表，在重新创建索引时应用于索引。如果从一种 DOL 锁方案更改为另一种，则不会复制数据页，也会不应用空间管理属性。
- 如果对表进行了分区，更改锁方案将以分区对分区的方式执行行复制。复制过程中不会平衡分区上的数据。
- 更改表的锁方案时，**alter table...lock** 命令将获取一个表的排它锁，直到命令完成为止。
- 使用 **alter table...lock** 从数据页锁更改为数据行锁时，该命令不复制数据页或重新创建索引。它只更新系统表。
- 当其他用户在系统上活动时，更改锁方案可能对用户活动有以下影响：
  - 过程高速缓存中访问表的查询计划将在下次运行时被重新编译。
  - 使用表的活动多语句过程将在继续下一步之前重新被编译。
  - 使用表的即席批处理事务将被终止。

◆ ***警告！***

在批量复制操作正在进行时更改表的锁方案可能导致表损坏。进行批量复制操作进行时，首先获取表的信息，并且在读取表信息和开始发送行的时间之间不持有锁，从而导致 alter table...lock 命令在这段时间内启动。

添加 Java-SQL 列

- 如果数据库中已启用 Java，可向表中添加 Java-SQL 列。有关详细信息，参见 *Adaptive Server Enterprise 中的 Java*。
- 新 Java-SQL 列声明的类 (*datatype*) 必须使用 **Serializable** 或 **Externalizable** 界面。
- 向表中添加 Java-SQL 列时，不能指定 Java-SQL 列：
  - 作为外键
  - 在引用子句中

  - 具有 UNIQUE 属性
  - 作为主键
- 如果指定了 **in row**，则存储的值不能超过 255 个字节。
- 如果指定了 **off row**，则：
  - 不能在检查约束中引用该列。
  - 不能在指定 **distinct** 的 **select** 中引用该列。
  - 不能在比较运算符、谓词或 **group by** 子句中指定该列。

标准和一致性

标准	一致性级别	注释
SQL92	Transact-SQL 扩展。	有关数据类型一致性的信息，参见第 1 章 “系统和用户定义的数据类型”。

权限

缺省情况下，表的所有者具有 **alter table** 权限；除向数据库所有者转移外，不能转移该权限，因为它可通过运行 **setuser** 命令来充当表的所有者。系统管理员也可以变更用户表。

参见

命令	**create index**、 **create table**、 **dbcc**、 **drop database**、 **insert**
系统过程	**sp_chgattribute**、 **sp_help**、 **sp_helpartition**、 **sp_rename**

begin...end

功能

包括一系列的 SQL 语句，这样控制流语言（例如 **if...else**）能够影响整个组的性能。

语法

```
begin
   statement block
end
```

关键字和选项

statement block — 是 **begin** 和 **end** 之间包括的一系列语句。

示例

1.
```
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** 条件只能执行一个 SQL 语句。

2.
```
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
else
  print "Deletion successful--no sales for this
      title."
```

   没有 **begin** 和 **end**，**print** 语句无法执行。

注释

- **begin...end** 块可以嵌套在其它 **begin...end** 块内。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都具有 **begin...end** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	if...else

begin transaction

功能

标记用户定义事务的起点。

语法

```
begin tran[saction] [transaction_name]
```

关键字和选项

transaction_name — 是指派给该事务的名称。事务名称必须遵循标识符的规则。仅在最外面的一对嵌套的 **begin transaction** / **commit** 或 **begin transaction** / **rollback** 语句中使用事务名称。

示例

```
1. begin transaction
     insert into publishers (pub_id) values ("9999")
   commit transaction
```

显式开始 **insert** 语句的事务。

注释

- 通过将 SQL 语句和 / 或系统过程包括在短语 **begin transaction** 和 **commit** 内来定义事务。如果设置链式事务模式，Adaptive Server 在执行下列语句之前隐式调用 **begin transaction**：**delete**、**insert**、**open**、**fetch**、**select** 和 **update**。必须用 **commit** 显式结束事务。
- 要取消整个事务或事务的一部分，使用 **rollback** 命令。**rollback** 命令必须在事务中出现；提交事务后就不能再回退该事务。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都具有 **begin transaction** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	commit、 rollback、 save transaction

break

功能

导致从 **while** 循环退出。**break** 通常由 **if** 测试激活。

语法

```
while logical_expression
      statement
   break
      statement
   continue
```

关键字和选项

logical_expression — 是返回 TRUE、FALSE 或 NULL 的表达式（可以是列名、常量、任何通过算术运算符或逐位运算符连接的列名和常量组合、或是子查询）。如果逻辑表达式包含 **select** 语句，用小括号括起 **select** 语句。

示例

```
1. while (select avg(price) from titles) < $30
    begin
     update titles
     set price = price * 2
     select max(price) from titles
     if (select max(price) from titles) > $50
           break
     else
           continue
    end
    begin
     print "Too much for the market to bear"
    end
```

如果平均价格小于 $30，则将价格翻倍。然后，选择最大价格。如果最大价格小于或等于 $50，重新启动 **while** 循环并再次将价格加倍。如果最大价格大于 $50，退出 **while** 循环并输出消息。

注释

- **break** 导致从 **while** 循环中退出。然后执行关键字 **end**（循环结束的标记）后面的语句。
- 如果嵌套了两个或两个以上的 **while** 循环，内部的 **break** 将退出到紧靠最外层的循环中。首先，运行内部循环结束后的所有语句；然后重新启动紧接最外层的循环。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都具有 **break** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	continue、 while

case

功能

支持条件 SQL 表达式；在可以使用值表达式的地方都可以使用。

语法

```
case
    when search_condition then expression
    [when search_condition then expression]...
    [else expression]
end
```

case 和值语法：

```
case expression
    when expression then expression
    [when expression then expression]...
    [else expression]
end
```

关键字和选项

case — 开始 **case** 表达式。

when — 位于搜索条件或要比较的表达式之前。

search_condition — 用来为所选择的结果设置条件。**case** 表达式的搜索条件类似于 **where** 子句中的搜索条件。*Transact-SQL User's Guide* 中详细说明了搜索条件。

then — 位于指定 **case** 结果值的表达式之前。

expression — 可以是列名、常量、函数、子查询，也可以是任何通过算术运算符或逐位运算符连接的列名、常量和函数组合。有关表达式的详细信息，参见第 3 章“表达式、标识符和通配符”中的“表达式”。

示例

1.
```
select au_lname, postalcode,
    case
        when postalcode = "94705"
            then "Berkeley Author"
        when postalcode = "94609"
            then "Oakland Author"
        when postalcode = "94612"
            then "Oakland Author"
        when postalcode = "97330"
            then "Corvallis Author"
    end
from authors
```

从 *authors* 表中选择所有作者，并指定某些作者所居住的城市。

2.
```
select stor_id, discount,
    coalesce (lowqty, highqty)
from discounts
```

返回第一次在 *discounts* 表的 *lowqty* 或 *highqty* 列中出现的非 NULL 值。

3.
```
select stor_id, discount,
    case
        when lowqty is not NULL then lowqty
        else highqty
    end
from discounts
```

这是编写示例 2 的替代方法。

4.
```
select title,
    nullif(type, "UNDECIDED")
from titles
```

从 *titles* 表中选择 *titles* 和 *type*。如果书籍类型是 UNDECIDED，**nullif** 返回值 NULL。

5.
```
select title,
    case
        when type = "UNDECIDED" then NULL
        else type
    end
from titles
```

这是编写示例 4 的替代方法。

注释

- **case** 表达式允许使用 **when...then** 结构来取代 **if** 语句表达搜索条件，从而简化了标准 SQL 表达式。
- SQL 中可以使用表达式的地方都可以使用 **case** 表达式。
- 至少一个 **case** 表达式的结果必须返回非空值。例如：

```
select price,
    coalesce (NULL, NULL, NULL)
from titles
```

产生下面错误消息：

```
All result expressions in a CASE expression must not be NULL.
```
（CASE 表达式中的结果表达式不能全为 NULL）

- 如果查询产生多种数据类型，数据类型层次将决定 **case** 表达式结果的数据类型，如第 1 章“系统和用户定义的数据类型”中的“混合型表达式的数据类型”所示。如果指定了两种 Adaptive Server 不能隐式转换的数据类型（例如，*char* 和 *int*），则查询失败。
- **coalesce** 是 **case** 表达式的缩写形式。示例 3 说明了编写 **coalesce** 语句的替代方法。
- **coalesce** 后面必须带有至少两个表达式。例如：

```
select stor_id, discount,
    coalesce (highqty)
from discounts
```

产生下面错误消息：

```
A single coalesce element is illegal in a COALESCE expression.
```
（单个 coalesce 元素在 COALESCE 表达式中是非法的）

- **nullif** 是 **case** 表达式的缩写形式。示例 5 说明了编写 **nullif** 的替代方法。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都具有 **case** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	select、 if...else、 where 子句

checkpoint

功能

将所有**脏**页（自上次写入以来被更新的页）写入到数据库设备。

语法

```
checkpoint
```

示例

```
1. checkpoint
```

无论系统检查点进度如何，将当前数据库中的脏页写入到数据库设备。

注释

- 仅将 **checkpoint** 用作特殊情况下的预防措施。例如，重新设置数据库选项后，Adaptive Server 将指导您发出 **checkpoint** 命令。
- 每次用系统过程 **sp_dboption** 更改数据库选项时，使用 **checkpoint**。

自动检查点

- **checkpoint** 命令引起的检查点补充每隔一段时间出现的自动检查点，间隔时间由 Adaptive Server 根据可配置的最大可接受恢复时间的值来计算确定。
- **checkpoint** 标识出可确保所有已完成事务都被写入数据库设备的点，从而缩短了自动恢复进程。典型的 **checkpoint** 大约耗时 1 秒，虽然检查点时间会依据在 Adaptive Server 的活动量变化而发生变化。
- Adaptive Server 根据系统活动和系统表 *syscurconfigs* 中的恢复间隔值计算出自动 **checkpoint** 间隔。通过指定系统恢复的最大时间，恢复间隔决定了 **checkpoint** 频率。通过执行系统过程 **sp_configure**，可重新设置该值。
- 如果管家任务可以在服务器的空闲时间内刷新所有已配置缓存中的所有活动缓冲池，它将唤醒检查点任务。检查点任务决定了它是否能够检查数据库。

  作为管家任务的结果发生的检查点称为**自由检查点**。自由检查点不包括将许多脏页写入数据库设备的工作，因为该工作已由管家任务完成。自由检查点可以提高数据库的恢复速度。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，数据库所有者具有 **checkpoint** 权限。但该权限不能转移。

参见

系统过程	**sp_configure**、 **sp_dboption**

close

功能

使游标失效。

语法

```
close cursor_name
```

参数

cursor_name — 是要关闭的游标的名称。

示例

```
1. close authors_crsr
```

关闭名为 *authors_crsr* 的游标。

注释

- **close** 命令实质上删除了游标的结果集。不能为关闭的游标定义在结果集中的位置。
- 如果游标已经关闭或不存在，Adaptive Server 将返回错误消息。

标准和一致性

标准	一致性级别
SQL92	初级一致性

权限

缺省情况下，所有用户都具有 **close** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	deallocate cursor、 declare cursor、 fetch、 open

coalesce

功能

支持条件 SQL 表达式；在可以使用值表达式的地方都可以使用；是 **case** 表达式的替代形式。

语法

coalesce(*expression*, *expression* [, *expression*]...)

关键字和选项

coalesce — 对列出的表达式进行求值并返回第一个非空值。如果所有表达式都为空，**coalesce** 返回空值。

expression — 可以是列名、常量、函数、子查询，也可以是任何通过算术运算符或逐位运算符连接的列名、常量和函数组合。有关表达式的详细信息，参见第 3 章“表达式、标识符和通配符”中的“表达式”。

示例

```
1. select stor_id, discount,
       coalesce (lowqty, highqty)
   from discounts
```

返回第一次在 *discounts* 表的 *lowqty* 或 *highqty* 列中出现的非 NULL 值。

```
2. select stor_id, discount,
       case
           when lowqty is not NULL then lowqty
           else highqty
       end
   from discounts
```

这是编写示例 1 的替代方法。

注释

- **coalesce** 表达式允许将搜索条件表达为简单比较而不是 **when...then** 结构，从而简化了标准 SQL 表达式。
- SQL 中可以使用表达式的地方都可以使用 **coalesce** 表达式。
- **coalesce** 表达式至少有一个结果必须返回非空值。例如：

```
select price,
    coalesce (NULL, NULL, NULL)
from titles
```

产生下面错误消息：

```
All result expressions in a CASE expression must not be NULL.
```
（CASE 表达式中的结果表达式不能全为 NULL）

- 如果查询产生多种数据类型，数据类型层次将决定 **case** 表达式结果的数据类型，如第 1 章“系统和用户定义的数据类型”中的“混合型表达式的数据类型”所示。如果指定了两种 Adaptive Server 不能隐式转换的数据类型（例如，*char* 和 *int*），则查询失败。
- **coalesce** 是 **case** 表达式的缩写形式。示例 2 说明了编写 **coalesce** 语句的替代方法。
- **coalesce** 后面必须带有至少两个表达式。例如：

```
select stor_id, discount,
    coalesce (highqty)
from discounts
```

产生下面错误消息：

```
A single coalesce element is illegal in a COALESCE expression.
```
（单个 coalesce 元素在 COALESCE 表达式中是非法的）

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都具有 **coalesce** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	case、 nullif、 select、 if...else、 where 子句

commit

功能

标记用户定义事务的终点。

语法

```
commit [tran[saction] | work] [transaction_name]
```

关键字和选项

transaction | **tran** | **work** — 是可选的。

transaction_name — 是指派给事务的名称。它必须遵守标识符的命名规则。仅在最外面的一对嵌套的 **begin transaction** / **commit** 或 **begin transaction** / **rollback** 语句中使用事务名称。

示例

```
1. 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

   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

rollback transaction percentchanged

commit transaction
```

在更新两个作者的 *royaltyper* 条目后，插入保存点 *percentchanged*，然后确定书籍价格增加百分之十将如何影响作者的版权收入。通过 **rollback transaction** 命令将事务回退到保存点。

注释

- 通过将 SQL 语句和 / 或系统过程包括在短语 **begin transaction** 和 **commit** 内来定义事务。如果设置链式事务模式，Adaptive Server 在执行下列语句之前隐式调用 **begin transaction**：**delete**、**insert**、**open**、**fetch**、**select** 和 **update**。仍必须用 **commit** 显式包含事务。
- 要取消整个事务或事务的一部分，使用 **rollback** 命令。**rollback** 命令必须包含在事务中。输入 **commit** 后就不能回退事务。
- 如果当前没有活动的事务，**commit** 或 **rollback** 语句对 Adaptive Server 就没有作用。

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	**commit transaction** 和 **commit tran** 形式的语句都是 Transact-SQL 扩展。

权限

缺省情况下，所有用户都具有 **commit** 权限。

参见

命令	**begin transaction**、 **rollback**、 **save transaction**

compute Clause

功能

生成摘要值，这些值在查询结果中显示为附加行。

语法

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

关键字和选项

row_aggregate — 是下列所示之一：

功能	含义
sum	（数值）列中值的总和
avg	（数值）列中值的平均值
min	列中的最小值
max	列中的最大值
count	列中值的个数

column_name — 是列名。必须用小括号将其括起来。只有数值列可使用 **sum** 和 **avg**。

一个 **compute** 子句可将几个集合函数应用于分组列的同一集合（参见示例 2 和 3）。要创建多个组，使用多个 **compute** 子句（参见示例 5）。

by — 计算子群的行集合值。只要 **by** 项的值发生变化，就会生成行集合值。如果使用 **by**，就必须使用 **order by**。

当 **by** 将一个组分为多个子群并在每个分组级别应用函数后，列出多个项。

示例

1.
```
select type, price
from titles
where price > $12
  and type like "%cook"
  order by type, price
compute sum(price) by type
```

```
type       price
---------  ------------
mod_cook          19.99
           sum
           ------------
                  19.99
type       price
---------  ------------
trad_cook         14.99
trad_cook         20.95
           sum
           ------------
                  35.94
(5 rows affected)
```

计算价格在 $12 以上的每种烹调书籍的价格总和。

2.
```
select type, price, advance
from titles
where price > $12
  and type like "%cook"
  order by type, price
compute sum(price), sum(advance) by type
```

```
type       price      advance
---------  ---------  ------------
mod_cook       19.99          0.00
          sum         sum
           ---------  ------------
               19.99          0.00

type       price      advance
---------  ---------  ------------
trad_cook      14.99      8,000.00
trad_cook      20.95      7,000.00
           sum        sum
           ---------  ------------
               35.94     15,000.00
(5 rows affected)
```

计算价格在 $12 以上的每种烹调书籍的价格总和以及预付款总和。

```
3. select type, price, advance
   from titles
   where price > $12
     and type like "%cook"
     order by type, price
   compute sum(price), max(advance) by type

   type      price     advance
   --------- --------- -------------
   mod_cook      19.99         0.00
             sum
             ---------
                 19.99
                       max
                       -------------
                                0.00

   type      price     advance
   --------- --------- -------------
   trad_cook     14.99      8,000.00
   trad_cook     20.95      7,000.00
             sum
             ---------
                 35.94
                        max
                        -------------
                             8,000.00
   (5 rows affected)
```

计算价格在 $12 以上的每种烹调书籍的价格总和以及最大预付款数。

```
4. select type, pub_id, price
   from titles
   where price > $10
     and type = "psychology"
     order by type, pub_id, price
   compute  sum(price) by type, pub_id

   type         pub_id    price
   ------------ --------- -----------
   psychology   0736          10.95
   psychology   0736          19.99
                           sum
                           ---------
                              30.94
```

```
type         pub_id    price
------------ --------- ---------
psychology   0877          21.59
                        sum
                        ---------
                            21.59
(5 rows affected)
```

解析 *type* 和 *pub_id*，并按照类型和出版者 ID 来计算心理学书籍的价格总和。

```
5. select type, pub_id, price
   from titles
   where price > $10
     and 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          10.95
psychology   0736          19.99
                        sum
                        ---------
                            30.94

type         pub_id    price
------------ --------- ---------
psychology   0877          21.59
                        sum
                        ---------
                            21.59
                        sum
                        ---------
                            52.53
(6 rows affected)
```

对价格在 $10 以上的心理学书籍按 *type* 和 *pub_id* 计算价格总和，并计算全部总计数。

```
6. select type, price, advance
   from titles
   where price > $10
     and type like "%cook"
   compute sum(price), sum(advance)
```

```
type       price        advance
---------  -----------  --------------
mod_cook         19.99            0.00
trad_cook        20.95        8,000.00
trad_cook        11.95        4,000.00
trad_cook        14.99        7,000.00
            sum          sum
           -----------  --------------
                 67.88       19,000.00
(5 rows affected)
```

计算价格在 $10 以上的烹调书籍的价格总计数以及预付款总计数。

7.
```
select type, price, price*2
from titles
  where type like "%cook"
compute sum(price), sum(price*2)
```

```
type         price
------------ --------------  ------------
mod_cook              19.99         39.98
mod_cook               2.99          5.98
trad_cook             20.95         41.90
trad_cook             11.95         23.90
trad_cook             14.99         29.98
             sum              sum
             ==============   ============
                     70.87         141.74
```

计算烹调书籍的价格总和以及表达式中使用的价格总和。

注释

- **compute** 子句允许您在一个结果集中查看明细行和摘要行。可计算子群的摘要值，还可计算同一组的多个集合。
- 不带 **by** 的 **compute** 可用于计算总计数，总量等等。如果使用不带 **by** 的 **compute** 关键字，**order by** 是可选的。参见示例 6。
- 如果使用 **compute by**，就必须使用 **order by** 子句。**compute by** 后面列出的列必须与 **order by** 后面列出的列相同或是其子集，它们从左向右的顺序必须一致，以同一表达式开始且不跳过任何表达式。例如，如果 **order by** 子句是：

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

  compute by 子句可以是下列任意（或全部）形式：

  ```
  compute by a, b, c
  compute by a, b
  compute by a
  ```

限制

- 在游标声明中不能使用 **compute** 子句。
- 表达式和列都可计算摘要值。**compute** 子句中出现的任何表达式或列都必须出现在 **select** 列表中。
- **compute** 子句中行集合的参数不允许使用列名的别名，虽然 **select** 列表、**order by** 子句和 **compute** 的 **by** 子句可使用它们。
- 不能将同一语句中的 **select into** 用作 **compute** 子句，因为包括 **compute** 的语句不能生成常规表。

compute 结果作为新的一行或多行出现

- 集合函数通常为表中的所有选定行或每个组生成一个单值，并且这些摘要值显示为新列。例如：

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

```
type
------------  ---------  ----------
mod_cook           22.98  15,000.00
trad_cook          47.89  19,000.00

(2 rows affected)
```

- 使用 **compute** 子句，可以通过一个命令来检索明细行和摘要行。例如：

```
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)
```

- 表 6-7 列出了不同类型 **compute** 子句的输出和分组。

表 6-7：按子句和明细行计算

子句和分组	输出	示例
一个 **compute** 子句，相同函数	一个明细行	1、2、4、6、7
一个 **compute** 子句，不同函数	每个函数类型一个明细行	3
多个 **compute** 子句，相同分组列	每个 **compute** 子句一个明细行；明细行一起输出	结果相同，但一个 compute 子句带有不同函数
多个 **compute** 子句，不同分组列	每个 **compute** 子句一个明细行；明细行根据分组不同而位于不同的位置	5

区分大小写

- 如果服务器安装了不区分大小写的排序顺序，**compute** 就忽略指定列中数据的大小写。例如，以下数据：

```
select * from groupdemo

lname      amount
---------- ------------------
Smith                  10.00
smith                   5.00
SMITH                   7.00
Levi                    9.00
Lévi                   20.00
```

lname 上的 **compute by** 产生下面结果：

```
select lname, amount from groupdemo
order by lname
compute sum(amount) by lname

lname      amount
---------- ------------------------
Levi                          9.00

Compute Result:
------------------------
                     9.00

lname      amount
---------- ------------------------
Lévi                         20.00

Compute Result:
------------------------
                    20.00

lname      amount
---------- ------------------------
smith                         5.00
SMITH                         7.00
Smith                        10.00

Compute Result:
------------------------
                    22.00
```

在不区分大小写和不区分变音的服务器上进行相同的查询将产生以下结果：

```
lname      amount
---------- ------------------------
Levi                          9.00
Lévi                         20.00

Compute Result:
------------------------
                   29.00

lname      amount
---------- ------------------------
smith                         5.00
SMITH                         7.00
Smith                        10.00

Compute Result:
------------------------
                   22.00
```

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

参见

命令	group by 和 having 子句、 select
函数	avg、 count、 max、 min、 sum

connect to...disconnect

（仅适用于组件集成服务）

功能

连接到指定服务器和断开已连接的服务器。

语法

```
connect to server_name
   disconnect
```

关键字和选项

server_name — 是需要直通连接的服务器。

示例

1. connect to SYBASE

建立到名为 SYBASE 的服务器的直通连接。

2. 断开连接

断开已连接的服务器。

注释

- **connect to** 指定需要直通连接的服务器。直通模式能够在远程服务器上执行本地操作。
- *server_name* 必须是 *sysservers* 表中的服务器的名称，该表定义了服务器类和网络名。
- 建立与代表用户的 *server_name* 连接时，组件集成服务使用下列标识符之一：
  - *sysattributes* 中描述的远程登录别名（如果存在的话）
  - 用户名和口令

  任何一种情况下，如果无法连接到指定服务器，Adaptive Server 返回错误消息。
- 有关添加远程服务器的详细信息，参见 **sp_addserver**。
- 完成直通连接后，组件集成服务在接收后续语言文本时会绕过 Transact-SQL 语法分析程序和编译器。它将语句直接传递到指定服务器，并将结果转换为可通过 Open Client 界面识别的形式，然后返回到客户端程序。

- 要关闭由 **connect to** 命令创建的连接，使用 **disconnect** 命令。只有使用 **connect to** 连接后才能使用此命令。
- **disconnect** 命令可以缩写为 **disc**。
- 除非以前已经发出 **connect to** 而且服务器已连接到远程服务器，否则 **disconnect** 命令将返回错误。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

必须由系统管理员显式授予使用 **connect to** 的权限。此命令的语法为：

```
grant connect to user_name
```

在 *master* 数据库中，系统管理员可全局地授予或撤消 *public* 的连接权限。如果系统管理员想要授予或撤消特定用户的 **connect to** 权限，该用户必须是 *master* 数据库的有效用户，而且系统管理员必须先撤消 *public* 的权限，如下所示：

```
use master
go
revoke connect from public
go
sp_adduser fred
go
grant connect to fred
go
```

参见

命令	create existing table、 grant
系统过程	sp_addserver、 sp_autoconnect、 sp_helpserver、 sp_passthru、 sp_remotesql、 sp_serveroption

continue

功能

重新启动 **while** 循环。**continue** 通常由 **if** 测试激活。

语法

```
while boolean_expression
      statement
   break
      statement
   continue
```

示例

```
1. while (select avg(price) from titles) < $30
   begin
     update titles
     set price = price * 2
     select max(price) from titles

     if (select max(price) from titles) > $50
            break
     else
            continue
   end

   begin
   print "Too much for the market to bear"
   end
```

如果平均价格小于 $30，则将价格翻倍。然后，选择最大价格。如果最大价格小于或等于 $50，重新启动 **while** 循环并再次将价格加倍。如果最大价格大于 $50，退出 **while** 循环并输出消息。

注释

- **continue** 重新启动 **while** 循环，跳过 **continue** 后面的任何语句。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都具有 **continue** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	break、 while

create database

功能

创建新的数据库。

语法

```
create database database_name
   [on {default | database_device} [= size]
       [, database_device [= size]]...]
   [log on database_device [= size]
       [, database_device [= size]]...]
   [with {override | default_location = "pathname"}]
   [for {load | proxy_update}]
```

关键字和选项

database_name — 是新的数据库的名称。它必须遵循标识符的规则，且不能是变量。

on — 表示数据库的位置和大小。

default — 表示 **create database** 可以将新的数据库置于任何缺省的数据库设备上 （如 *sysdevices.status* 中所示）。要指定该数据库的大小（不指定位置），请使用以下命令：

```
on default = size
```

要将数据库设备的状态更改为 “缺省”，使用系统过程 **sp_diskdefault**。

database_device — 是数据库所在设备的逻辑名。数据库可以在多个数据库设备的每个设备上占用不同的空间量。用 **disk init** 向 Adaptive Server 添加数据库设备。

size — 是分配给数据库的空间量 （用兆字节表示）。 Adaptive Server 提供的缺省大小是 2MB。

log on — 为数据库日志指定设备的逻辑名。可以在 **log on** 子句中指定多个设备。

with override — 强制 Adaptive Server 接受设备说明，即使这些说明混合了同一设备上的数据和事务日志，由此危及了数据库以分钟计算的恢复能力。如果试图在同一设备上混合日志和数据而不使用该子句， **create database** 命令将失败。如果混合了日志和数据，并使用 **with override**，尽管会受到警告，但是命令仍成功执行。

for load — 调用只可用于装载数据库转储的 **create database** 的最新版本。有关详细信息，参见下面的 “使用 for load 选项”。

with default_location — 指定新表的存储位置。如果还指定了 **for proxy_update** 子句，则将从指定位置自动创建每个远程表或视图的一个代理表。

for proxy_update — 自动从远程位置获取元数据并创建代理表。除非指定 **with default_location**，否则无法使用 **for proxy_update**。

示例

```
1. create database pubs
```

创建名为 *pubs* 的数据库。

```
2. create database pubs
   on default = 4
```

创建名为 *pubs* 的 4MB 大小的数据库。

```
3. create database pubs
   on datadev = 3, moredatadev = 2
```

创建名为 *pubs* 的数据库，其中 3MB 大小在 *datadev segment* 上，2MB 大小在 *moredatadev* 段上。

```
4. create database pubs
   on datadev = 3
       log on logdev = 1
```

创建名为 *pubs* 的数据库，其中 3MB 的数据在 *datadev* 段上，1MB 的日志在 *logdev* 段上。

```
5. create database proxydb
   with default_location
   “UNITEST.pubs.dbo.”
```

创建名为 *proxydb* 的代理数据库，但不自动创建代理表。

```
6. create database proxydb
   on default = 4
   with default_location
   “UNITEST.pubs2.dbo.”
   for proxy_update
```

创建名为 *proxydb* 的代理数据库，并自动创建代理表。

注释

- 从 *master* 数据库使用 **create database**。
- 如果未指定数据库的位置和大小，则缺省的位置就是 *master..sysdevices* 中所指示的任意的缺省数据库设备。缺省大小大于 *model* 数据库的大小或大于 *sysconfigures* 中 **default database size** 参数的大小。

  通过使用 **sp_configure** 来更改 **default database size** 的值然后重新启动 Adaptive Server 可增大缺省大小。**default database size** 参数值至少必须等于 *model* 数据库的大小。如果将 *model* 数据库的大小增大，则也必须增大缺省大小。

  如果 Adaptive Server 无法分配所请求的空间大小，它会尽可能地为每个设备分配接近请求的空间，并输出消息说明已经在每个数据库设备上分配的空间量。数据库的最大大小取决于系统。
- 如果使用以下命令创建代理数据库：

  ```
  create database mydb on my_device
  with default_location = "pathname" for proxy_update
  ```

  设备名的存在可完全避免计算大小，在缺省的数据库大小（*model* 数据库的大小）不足以包含所有代理表的情况下，该命令可能失败。

  要使用 CIS 估计数据库大小，则该命令中不应提供设备名或任何其它选项。

  ```
  create database mydb
  with default_location = "pathname" for proxy_update
  ```

限制

- Adaptive Server 可以管理最多 32,767 个数据库。
- Adaptive Server 每次只能创建一个数据库。如果两个数据库的创建请求发生冲突，用户会收到以下消息：

  ```
  model database in use: cannot create new database
  ```
- 数据库的最大设备片段数是 128。每次用 **create database** 或 **alter database** 在数据库设备上分配空间时，该分配表示一个设备片段，并作为 *sysusages* 中的一行输入。
- 数据库命名段的最大个数是 32。这些段是可用于特定 Adaptive Server 的数据库设备的命名子集。有关段的详细信息，参见*系统管理指南*。

从 *model* 创建新的数据库

- 通过复制 *model* 数据库， Adaptive Server 可创建新的数据库。
- 可以通过添加表、存储过程、用户定义的数据类型和其它对象以及更改数据库选项设置来自定义 *model*。新的数据库将从 *model* 来继承这些对象和设置。
- 要确保可恢复性，**create database** 命令必须清除在复制 *model* 数据库后未经初始化的每一页。该过程可能要占用一些时间，这取决于数据库的大小和系统的速度。

  如果创建数据库是为了向其装载数据库转储，则可以使用 **for load** 选项跳过页清除步骤。这可以使创建数据库的速度极大提高。

确保数据库的可恢复性

- 每次创建新的数据库后，请备份 *master* 数据库。这样做使得 *master* 被损坏后恢复起来更加容易和安全。

➤ ***注意***

如果创建了数据库但无法对 *master* 进行备份，则使用 **disk refit** 就能够恢复更改。

- 使用 **with override** 子句可在单个设备上混合日志和数据段。然而，要获得完全恢复，**log on** 中指定的一个或多个设备应不同于存储数据的物理设备。如果硬盘崩溃，则可从数据库转储和事务日志对数据库进行恢复。

  可在一个用于同时存储事务日志和数据的单个设备上创建小型数据库，但**必须**依靠 **dump database** 命令进行备份。
- 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。作为经验法则，一般将分配给数据库自身的 10-25% 空间分配给日志设备。最好开始时所分配的空间要小一些，原因是分配给事务日志设备的空间无法回收，也无法用于存储数据。

使用 *for load* 选项

使用 **for load** 选项可以从介质故障进行恢复，也可以将数据库从一台计算机移动到另一台计算机 （如果尚未使用 **sp_addsegment** 添加到数据库）。使用 **alter database for load** 可在数据库 （已从该数据库完成待装载的数据库转储）映像文件中创建新的数据库。有关将转储装载到新数据库时重复进行空间分配的说明，参见*系统管理指南*。

- 如果使用 **for load** 选项创建数据库，则装载数据库转储之前在新数据库中只能运行如下命令：
  - **alter database for load**
  - **drop database**
  - **load database**

  在将数据库转储装载到新的数据库以后，还可以在数据库中使用某些 **dbcc** 诊断命令。发出 **online database** 命令后，将不存在命令使用的限制。
- 在 **sp_helpdb** 的输出中使用 **for load** 选项创建的数据库具有 “don't recover” 状态。

获取有关数据库的信息

- 要获取数据库的报告，可执行系统过程 **sp_helpdb**。
- 要获取数据库中使用空间的报告，使用 **sp_spaceused**。

使用 *with default_location* 和 *for proxy_update*

如果无 **for proxy_update** 子句， **with default_location** 子句的行为与存储过程 **sp_defaultloc** 所提供的行为相同 — 建立缺省的存储位置以用于创建新的和现有的表，但在处理 **create database** 命令期间将不自动导入代理表定义。

- 如果指定 **for proxy_update** 时不指定 **default_location**，则报告错误。
- 使用 **for proxy_update** 选项创建代理数据库以后，将调用组件集成服务以便：
  - 提供包含所有代理表需要的数据库大小估计值，代理表表示主服务器的数据库中找到的实际表和视图。此估计值是包含所有代理表和索引所需要的数据库页数。如果未指定大小和数据库设备，则使用该估计值。
  - 创建表示在辅助服务器的数据库中所找到的实际表和视图的所有代理表。
  - 将代理表的所有权限授予 *public*。
  - 将 *guest* 用户添加到代理数据库。
  - 将设置数据库状态来指示该数据库的 “Is_A_Proxy” 状态。该状态包含在 *master.dbo.sysdatabases.status4* 中。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

create database 权限缺省设置为系统管理员，系统管理员可以将该权限传送给主数据库的 *sysusers* 表中所列出的用户。然而， **create database** 权限经常是集中控制的，以便于保持对数据库存储分配的控制。

如果要创建 *sybsecurity* 数据库，您必须是系统安全员。

grant all 命令中不包括 **create database** 权限。

参见

命令	**alter database**、**disk init**、**drop database**、**dump database**、 **load database**、 **online database**
系统过程	**sp_changedbowner**、 **sp_diskdefault**、**sp_helpdb**、 **sp_logdevice**、 **sp_renamedb**、**sp_spaceused**

create default

功能

如果在插入时没有显式提供值，则指定要在列 （或具有用户定义数据类型的所有列）中插入的值。

语法

```
create default [owner.]default_name
   as constant_expression
```

关键字和选项

default_name — 是缺省的名称。它必须遵循标识符的规则，且不能是变量。指定所有者的名称可创建当前数据库中不同用户所拥有的其它同名的缺省名称。 *owner* 的缺省值是当前用户。

constant_expression — 是表达式，它不包括任何列名或其它数据库对象。可以包括不引用数据库对象的内部函数。将字符和日期常量用引号引起，对于二进制常量，请使用 “0x” 前缀。

示例

1. `create default phonedflt as "UNKNOWN"`

   定义缺省值。此时，需要使用 **sp_bindefault** 将该值绑定到适当的列或用户定义的数据类型。

2. `sp_bindefault phonedflt, "authors.phone"`

   只有在 *authors* 表的 *phone* 列中没有条目的情况下，该缺省值才生效。没有条目有别于空值条目。要获取缺省值，请向不包括具有缺省值的列的列列表发出 **insert** 命令。

3. `create default todays_date as getdate()`

   创建缺省值 *todays_date*，该缺省值可将当前日期插入到绑定缺省值的列。

注释

- 使用 **sp_bindefault**，将缺省值绑定到列或用户定义的数据类型 （但不是 Adaptive Server 提供的数据类型）。
- 可以在不解除绑定旧的缺省值的情况下将新的缺省值绑定到数据类型。新的缺省值将替换旧的缺省值，并将其解除绑定。
- 要隐藏缺省值的源文本，请使用 **sp_hidetext**。

限制

- 只可在当前数据库中创建缺省值。
- **create default** 语句无法在单个批处理中与其它语句结合使用。
- 必须先使用 **drop default** 删除缺省值，然后再创建新的同名缺省值，并且在删除缺省值以前，必须使用 **sp_unbindefault** 来解除绑定该缺省值。

数据类型兼容性

- 如果 Adaptive Server 试图插入的缺省值与列的数据类型不兼容，将生成错误消息。例如，如果将字符表达式 （如 “N/A”）绑定到 *integer* 列，则任何未指定该列值的**插入**将失败。
- 如果字符列的缺省值过长， Adaptive Server 将根据 **string_rtruncation** 选项的设置截断该字符串，还可以作为例外处理。有关详细信息，参见 **set** 命令。

获取有关缺省值的信息

- 缺省定义存储在 *syscomments* 中。
- 如果缺省值绑定到列，其对象 ID 将存储在 *syscolumns* 中。如果缺省值绑定到用户定义的数据类型，其对象 ID 将存储在 *systypes* 中。
- 要重命名缺省值，使用 **sp_rename**。
- 对于缺省值文本的报告，使用 **sp_helptext**。

缺省值和规则

- 如果列同时具有缺省值和与其相关联的规则，则缺省值一定不能与规则发生冲突。不能插入与规则冲突的缺省值。 Adaptive Server 每次试图插入这类缺省值时，将生成错误消息。

缺省值和空值

- 如果列不允许空值，并且未创建列的缺省值，当用户试图插入行（但不包括该列的值）时，插入将失败， Adaptive Server 生成错误消息。

表 6-8 说明存在的缺省值和列的 NULL 或 NOT NULL 定义的关系。

表 6-8：空值和列缺省值之间的关系

列空值类型	无条目， 无缺省值	无条目， 缺省值存在	条目为空值， 无缺省值	条目为空值， 缺省值存在
NULL	已插入空值	已插入缺省值	已插入空值	已插入空值
NOT NULL	错误，命令失败	已插入缺省值	错误，命令失败	错误，命令失败

在 *create table* 中指定缺省值

- 除可以使用 **create default** 来定义列缺省值以外，还可以使用 **create table** 语句的 **default** 子句来定义。然而，这些列缺省值是该表专用的；无法将其绑定到其它的表。有关完整性约束的信息，参见 **create table** 和 **alter table**。

标准和一致性

标准	一致性级别	注释
SQL92	Transact-SQL 扩展	使用 **create table** 语句的 **default** 子句可创建符合 SQL92 的缺省值。

权限

缺省情况下，将 **create default** 权限授予数据库所有者，该所有者可以将其移交给其他用户。

参见

命令	**alter table**、 **create rule**、 **create table**、 **drop default**、 **drop rule**
系统过程	**sp_bindefault**、 **sp_help**、 **sp_helptext**、 **sp_rename**、 **sp_unbindefault**

create existing table

（仅对组件集成服务）

功能

创建代理表，然后检索和存储远程表中的元数据，并将数据放置到代理表中。允许将代理表映射到远程位置处的表、视图或过程。

语法

```
create existing table table_name (column_list)
   [ on segment_name ]
   [ [ external {table | procedure} ] at pathname ]
```

关键字和选项

table_name — 指定要为其创建代理表的表的名称。

column_list — 指定存储有关远程表信息的列列表的名称。

on *segment_name* — 指定包含远程表的段。

external — 指定该对象为一个远程对象。

table — 指定远程对象是一个表还是一个视图。缺省为 **external table**。

procedure — 指定远程对象是存储过程。

at *pathname* — 指定远程对象的位置。 *pathname* 采用以下形式：

server_name.dbname.owner.object

其中：

- *server_name* （必需）是包含远程对象的服务器名
- *dbname* （可选）是由包含该对象的远程服务器管理的数据库名
- *owner* （可选）是拥有远程对象的远程服务器用户名
- *object* （必需）是远程表、视图或过程的名称

示例

1.
```
create existing table authors
(
au_id        id,
au_lname     varchar(40)    NOT NULL,
au_fname     varchar(20)    NOT NULL,
phone        char(12),
address      varchar(40)    NULL,
city         varchar(20)    NULL,
state        char(2)        NULL,
zip          char(5)        NULL,
contract     bit
)
```

   创建代理表 *authors*。

2.
```
create existing table syb_columns
(
id          int,
number      smallint,
colid       tinyint,
status      tinyint,
type        tinyint,
length      tinyint,
offset      smallint,
usertype    smallint,
cdefault    int,
domain      int,
name        varchar(30),
printfmt    varchar(255)    NULL,
prec        tinyint         NULL,
scale       tinyint         NULL
)
```

   创建代理表 *syb_columns*。

3.
```
create existing table blurbs
(author_id  id      not null,
copy        text    not null)
at "SERVER_A.db1.joe.blurbs"
```

   为远程服务器 SERVER_A 上的 *blurbs* 表创建名为 *blurbs* 的代理表。

4.
```
create existing table rpc1
(column_1   int,
column_2    int)
external procedure
at "SERVER_A.db1.joe.p1"
```

   为名为 *p1* 的远程过程创建名为 *rpc1* 的代理表。

注释

- **create existing table** 命令不创建新表。但是组件集成服务将检查表映射以确认 *column_list* 中的信息与远程表相匹配，并校验是否存在基础对象，检索和存储有关远程表的元数据。
- 如果主机数据文件或远程服务器对象不存在，该命令将被拒绝，并提供错误消息。
- 如果存在该对象，系统表 *sysobjects*、 *syscolumns* 和 *sysindexes* 将被更新。校验分三步进行：
  - 确定现有对象的性质。对于主机数据文件，这需要确定文件组织和记录格式。对于远程服务器对象，这需要确定该对象是表、视图还是 RPC。
  - 对于远程服务器对象 （不包括 RPC），将获得的表或视图的列特性与 *column_list* 中定义的特性相比较。
  - 将抽取主机数据文件或远程服务器表的索引信息，并用于创建系统表 *sysindexes* 的行。这将用 Adaptive Server 的术语定义索引和键，并启用查询优化程序来考虑表中任何可能存在的索引。
- **on *segment_name*** 子句在本地进行处理，不传递给远程服务器。
- 成功定义现有表后，对表发出 **update statistics** 命令。这将允许查询优化程序在考虑选择索引和连接顺序时做出正确选择。
- 即使远程列定义为 NULL，组件集成服务也允许创建带有 NOT NULL 定义的列的代理表。但它将显示警告通知您这种不匹配的情况。
- 由 **at** 关键字提供的位置信息与由 **sp_addobjectdef** 系统过程提供的信息相同。这些信息将被存储在 *sysattributes* 表中。
- 组件集成服务为远程表中的每个索引在 *systabstats* 目录中插入或更新一条记录。由于更详细的结构化统计信息与远程索引无关，所以在 *systabstats* record*_id*、 *indid* 和 *rowcnt* 中只设置最少数目的列。

数据类型转换

- 使用 **create existing table** 命令时，必须用可识别的 Adaptive Server 数据类型对所有数据类型进行指定。如果远程服务器表驻留在异构服务器类中，则在检索数据时，远程表的数据类型将自动转换成指定的 Adaptive Server 类型。如果不能进行转换，组件集成服务不允许定义表。
- *Component Integration Services User's Guide* 对每个支持的服务器类都有一个小节进行说明，并标识由组件集成服务隐式执行的所有可能的数据类型转换。

服务器类所进行的更改

- 现在所有的服务器类都允许指定的列比远程服务器上表中的列少。
- 现在所有的服务器类都按名称来匹配列。有些服务器类以前是按列 ID 匹配列。
- 现在所有的服务器类都允许任何可与远程表列的数据类型相互转换的数据类型。

远程过程

- 如果代理表是一个过程类型表，则必须提供一个列的表，此列表与远程过程结果集的描述相匹配。**create existing table 不**校验该列表的精确性。
- 不会为过程创建索引。
- 组件集成服务将远程过程的结果集作为虚拟表处理，该表可以排序、可与其它表连接或者可以使用 **insert** 或 **select** 插入另一个表中。但是，过程类型表被认为是只读的，意味着不能对该表发布以下命令：
  - **delete**
  - **update**
  - **insert**
  - **create index**
  - **truncate table**
  - **alter table**
- 列名以下划线 (_) 开始，以指定此列不属于远程过程的结果集。这些列将作为参数列被引用。例如：

```
create existing table rpc1
(
    a       int,
    b       int,
    c       int,
    _p1     int null,
    _p2     int null
)
external procedure
at "SYBASE.sybsystemprocs.dbo.myproc"
```

在此示例中，参数列 *_p1* 和 *_p2* 是输入参数。它们不会在结果集中出现，但可以在查询中引用：

```
select a, b, c from t1
where _p1 = 10 and _p2 = 20
```

CIS 使用名称 *@p1* 和 *@p2*，将搜索自变量作为参数传递给远程过程。

- **create existing table** 语句中的参数列定义必须遵从以下规则：
  - 参数列定义必须允许空值。
  - 参数列不能位于常规结果列之前，它们必须出现在列列表的末尾。
- 如果将参数列包括在 **select** 列表中，并且作为参数传递给远程过程，返回值将由 **where** 子句指定。
- 如果将参数列包括在 **select** 列表中，但没有出现在 **where** 子句中或不能作为参数传递给远程过程，其值将为 NULL。
- 如果 Adaptive Server 查询处理器认为参数列是可搜索自变量，则可将参数列作为参数传递给远程过程。如果没有将参数列包括在任何 **or** 谓词中，参数列将被认为是可搜索自变量。例如，下面查询第二行的 **or** 谓词将使参数列不能用作参数：

```
select a, b, c from t1
where _p1 = 10 or _p2 = 20
```

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

create existing table 权限默认为表所有者拥有，且不能移交。

参见

命令	alter table、 create table、 create proxy_table、 drop index、 insert、 order by Clause、 set、 update

create index

功能

创建表中一个或多个列的索引；为每列按升序或降序创建索引；每个索引最多允许 31 列；创建索引期间保留指定数目的未使用页；允许指定索引分配直方图的梯级数。

语法

```
create [unique] [clustered | nonclustered]
       index index_name
   on [[database.]owner.]table_name
       (column_name [asc | desc]
       [, column_name [asc | desc]]...)
   [with {
       {fillfactor = pct| max_rows_per_page=num_rows},
       reservepagegap = num_pages,
       consumers = x, ignore_dup_key, sorted_data,
       [ignore_dup_row | allow_dup_row],
       , statistics using num_steps values  } ]
   [on segment_name]
```

关键字和选项

unique — 禁止重复的索引值 （也称为 “键值”）。如果已存在数据，在创建索引后系统将检查重复的键值，这种检查也将在每次使用 **insert** 或 **update** 添加数据后进行。如果存在重复的键值 （或者有多行包含空值），命令将失败， Adaptive Server 将输出错误消息，消息中提供重复条目。

◆ ***警告！***

如果表包含任何非空 *text* 或 *image* 列，则 Adaptive Server 不检测重复行。

除非索引通过使用 **ignore_dup_row** 或 **ignore_dup_key** 创建，否则生成重复键值的 **update** 和 **insert** 命令将失败。

组合索引 （其中键值由多个列组成）也可具有唯一性。

缺省情况下是非唯一的。要创建包含重复行的表中的非唯一集群索引，必须指定 **allow_dup_row** 或 **ignore_dup_row**。参见以下 “重复行”。

clustered — 表示当前数据库设备的行的实际顺序与行的索引顺序相同。集群索引的末端 （即**页级**）包含实际的数据页。集群索引检索数据几乎总是比非集群索引快。每个表仅允许有一个集群索引。参见以下 “创建集群索引”。

如果未指定 **clustered**，则假定为 **nonclustered**。

nonclustered — 表示行的实际顺序与其索引顺序不同。非集群索引的页级包含指向数据页上的行的指针。每个表最多可以有 249 个非集群索引。

index_name — 是索引名。表中的索引名必须是唯一的，但数据库中的索引名可以不唯一。

table_name — 是索引列 （一列或多列）所在的表的名称。如果该表位于另一数据库中，则指定数据库名，并且在数据库中存在多个使用该名称的表时，指定所有者的名称。 *owner* 的缺省值是当前用户，而 *database* 的缺省值是当前数据库。

column_name — 是应用索引的列 （一列或多列）。组合索引基于最多 16 个列的组合值。组合索引中使用的所有列的最大长度的总和不能超过 600 个字节。将包括在组合索引中的列 （按其排序顺序）列在 *table_name* 后的小括号内。

asc | desc — 指定以升序还是降序顺序为指定的列创建索引。缺省为升序。

fillfactor — 指定 Adaptive Server 在现有数据上创建新索引时对每页的填充程度。 **fillfactor** 百分比仅在索引已创建后才有意义。随着数据的变化，页不会维持在某个特定的填充程度。

fillfactor 的缺省值是 0 ； 除非已经使用 **sp_configure** 更改了该值，否则在 **create index** 语句不包括 **with fillfactor** 的情况下才使用该缺省值。使用 1 到 100 之间的值指定 **fillfactor**。

fillfactor 为 0 时，将创建页完全填充的集群索引和叶页完全填充的非集群索引。它只在集群索引和非集群索引中的索引 B 树内保留适当的空间量。很少需要更改 **fillfactor**。

如果设置 **fillfactor** 为 100，则在 Adaptive Server 创建的集群和非集群索引中每一页都百分之百地完全填充。 **fillfactor** 为 100 时，仅对只读表 （不再向其添加额外的数据）有意义。

fillfactor 值小于 100（除了 0 这个特例外）将导致 Adaptive Server 创建的新索引的页没有完全填充。在最终将保存大量数据的表上创建索引时，选择 **fillfactor** 为 10 是适当的，但是小的 **fillfactor** 值也将导致每个索引 （或索引和数据）占用更多的存储空间。

◆ ***警告！***

用 fillfactor 创建集群索引会影响数据占用的存储空间量，因为 Adaptive Server 在创建集群索引时重新分配数据。

max_rows_per_page — 限制索引的数据页和叶级页的行数。**max_rows_per_page** 和 **fillfactor** 相互排斥。与 **fillfactor** 不同，**max_rows_per_page** 值在用 **sp_chgattribute** 更改前保持不变。

如果不指定 **max_rows_per_page** 的值， Adaptive Server 创建表时使用值 0。表和集群索引的值介于 0 和 256 之间。非集群索引每页的最大行数取决于索引键的大小。如果指定值太大， Adaptive Server 返回错误消息。

max_rows_per_page 为 0 时，将创建页完全填充的集群索引和叶页完全填充的非集群索引。它将在集群索引和非集群索引中的索引 B 树内都保留适当的空间量。

如果设置 **max_rows_per_page** 为 1，则 Adaptive Server 创建的聚簇和非集群索引在页级每页中都只有一行。使用较小值可减少对常用数据的锁争用。然而，小的 **max_rows_per_page** 值使 Adaptive Server 创建的新索引的页未完全填充，并使用更多的存储空间，还导致更多的页面拆分。

如果启用了组件集成服务，则无法将 **max_rows_per_page** 用于远程服务器。

◆ ***警告！***

用 max_rows_per_page 创建集群索引可能会影响数据占用的存储空间量，因为 Adaptive Server 在创建集群索引时重新分配数据。

with reservepagegap = *num_pages* — 指定扩展 I/O 分配操作期间填充页与要保留的空白页之比。对于每个指定的 *num_pages*，保留一个空白页以用于索引的将来扩展。有效值为 0-255。缺省值为 0。

ignore_dup_key — 取消把重复键放入具有唯一集群 （非集群）索引的表的尝试。 Adaptive Server 取消重复键的 **insert** 或 **update** 尝试，并提供信息性消息。取消后，继续完成包含重复键的事务。

无论 **ignore_dup_key** 设置与否，都无法在包括重复值或多个空值的列中创建唯一索引。如果尝试创建索引， Adaptive Server 将输出错误消息，消息中提供第一个重复值。 Adaptive Server 在列上创建唯一索引之前，必须消除重复值。

ignore_dup_row — 通过从表删除重复行，可以创建包括重复行的表的新的非唯一集群索引，该命令还取消任何会创建重复行的 **insert** 或 **update**，但不会回退整个事务。有关详细信息，参见下面的“重复行”。

allow_dup_row — 可以创建包括重复行的表中的非唯一集群索引，还可以使用 **update** 和 **insert** 语句复制行。有关如何使用这些选项的说明，参见以下 “重复行”。

sorted_data — 如果表中的数据已按排序顺序排列 （例如已使用 **bcp** 将已经排序的数据复制到空表的情况），则可加速集群索引或唯一非集群索引的创建。有关详细信息，参见 “使用 sorted_data 选项可加速排序”。

with statistics using *num_steps* **values** — 指定用于生成直方图 （此图用于优化查询）的梯级数。如果省略该子句：

- 如果前导索引列当前没有存储直方图，则其缺省值为 20，
- 如果前导索引列的直方图已经存在，则将使用当前的梯级数。

如果为 *num_steps* 指定 0，则将重新创建索引，但不会覆盖系统表中有关索引的统计信息。

on *segment_name* — 在已命名段上创建索引。使用 **on** *segment_name* 选项以前，通过 **disk init** 对设备进行初始化，并使用 **sp_addsegment** 系统过程将段添加到数据库。有关数据库中可使用段名的列表，请与系统管理员联系或使用 **sp_helpsegment**。

with consumers — 指定消耗程序进程的个数，这些进程应执行排序操作以创建索引。如果 Adaptive Server 执行排序时可用的工作进程较少，则用于排序索引的消耗程序进程的实际数目可能小于指定数目。

示例

1. ```
   create index au_id_ind
   on authors (au_id)
   ```

   创建 *authors* 表的 *au_id* 列的索引 （名为 *au_id_ind*）。

2. ```
   create unique clustered index au_id_ind
   on authors(au_id)
   ```

   创建 *authors* 表的 *au_id* 列的唯一集群索引 （名为 *au_id_ind*）。

3. ```
   create index ind1
   on titleauthor (au_id, title_id)
   ```

   创建 *titleauthor* 表的 *au_id* 和 *title_id* 列的索引 （名为 *ind1*）。
   ```

4.
```
create nonclustered index zip_ind
on authors(postalcode)
with fillfactor = 25, consumers = 4
```

   创建 *authors* 表的 *postalcode* 列的非集群索引（名为 *zip_ind*），填充每个索引页的四分之一，并将排序限制为由 4 个消耗程序进程进行。

5.
```
create index pub_dates_ix
on titles (pub_id asc, pubdate desc)
```

   创建一个 *pub_id* 列为升序、 *pubdate* 列为降序的索引。

6.
```
create index title_id_ix
    on titles (title_id)
    with reservepagegap = 40,
    statistics using 50 values
```

   创建 *title_id* 上的索引，使用 50 个直方图梯级数用于优化统计，并在索引中每 40 页保留 1 个空白页。

注释

- 如果将要更改索引中键值分布的数据添加到表，请定期运行 **update statistics**。查询优化程序使用 **update statistics** 创建的信息来选择运行表中查询的最佳计划。
- 如果创建非集群索引时表中包含数据， Adaptive Server 将在新的索引上运行 **update statistics**。如果创建集群索引时表中包含数据， Adaptive Server 将在所有表的索引上运行 **update statistics**。
- 对定期用于连接的所有列进行索引。
- 如果启用组件集成服务，将重新构造 **create index** 命令，并将其直接传递给与该表相关联的 Adaptive Server。

限制

- 无法创建数据类型为 *bit*、 *text* 或 *image* 的列的索引。
- 表的非集群索引最多可以达到 249 个。
- 表最多只能有一个集群索引。
- 可以为索引键指定最多 31 个列 （原来为 16 个）。字节总数最大为 600。
- 可以创建临时表的索引。取消临时表时，该索引也随之被取消。
- 只要您是表的所有者，就可以在其它数据库创建该表的索引。
- 无法创建视图的索引。
- 运行 **dump database** 时， **create index** 的运行速度较慢。

- 如果以下条件为真，可以创建已分区的表中的集群索引或者对具有集群索引的表进行分区：
  - 启用 **select into/bulkcopy/pllsort** 数据库选项；
  - 配置 Adaptive Server 可进行并行处理，以及
  - 可用的工作进程的个数比分区个数多一个。

  有关已分区的表中的集群索引的详细信息，参见 *Performance and Tuning Guide* 中的第 13 章 “Parallel Sorting”。
- 具有集群索引的 DOL 锁定的表上允许的最大索引数是 249。一个表可以有 1 个集群索引和 248 个非集群索引。

有效创建索引

- 索引可加速数据检索，但会减慢数据更新的速度。为了达到更佳的性能，当各段位于不同的物理设备时，请在一个段上创建表，而在其它段上创建其非集群索引。
- 如果将表分区并对服务器进行并行配置， Adaptive Server 可以创建并行索引。它还可以使用排序缓冲来减少排序期间所需的 I/O 量。有关详细信息，参见 *Performance and Tuning Guide* 中的第 13 章 “Parallel Sorting”。
- 创建任何非集群索引以前，先创建集群索引，原因是非集群索引将在创建集群索引后自动重建。
- 对 DOL 锁定的表使用并行排序时，工作进程数必须配置为等于或大于分区数 （即使对空表也如此）。也必须启用数据库选项 **select into/bulkcopy/pllsort**。

创建集群索引

- 表 “跟随”其集群索引。如果创建了表，然后使用 **create clustered index** 的 **on** *segment_name* 扩展，则表将迁移到创建索引的段。

  如果在特定段上创建了表，然后创建集群索引而不指定段，则 Adaptive Server 将把表移动到它在其中创建了集群索引的缺省段。

  因为文本和图像数据存储在不同的页链中，所以使用 **on** *segment_name* 创建集群索引不会移动文本和图像列。
- 要创建集群索引， Adaptive Server 将复制现有数据；索引完成后，服务器将删除原始数据。创建集群索引前，使用 **sp_spaceused** 以确保数据库可用空间的大小至少为表大小的 120%。
- 经常创建表的主键 （唯一标识行的一列或多列）的集群索引。主键可以使用系统过程 **sp_primarykey** 记录在数据库中 （用于前端程序和系统过程 **sp_depends**）。

- 要允许集群索引中存在重复行，指定 **allow_dup_row**。

指定索引中的升序或降序顺序

- 在索引列名后使用 **asc** 和 **desc** 关键字可指定索引键的排序顺序。按查询的 **order by** 子句中指定的顺序创建索引，将消除查询过程中的排序步骤。有关详细信息，参见 *Performance and Tuning Guide* 中的第 13 章 “Indexing for Performance”。

索引的空间要求

- 一次按一个扩展 （即八个页）的增量来为表和索引分配空间。每当一个扩展被填满后，将分配另一个扩展。（使用系统过程 **sp_spaceused** 可显示索引所分配和使用的空间量。）
- 在某些情况下，使用 **sorted_data** 选项允许 Adaptive Server 跳过对表 6-11 中说明的数据行所进行的复制操作。在某些情况下，只需要足够的额外空间供索引结构本身使用即可。根据键的大小，这通常大约需要表大小的 20%。

重复行

- 创建非唯一的非集群索引时， **ignore_dup_row** 和 **allow_dup_row** 是不相关的选项。因为 Adaptive Server 在每个非集群索引中内部附加一个唯一的行标识号，所以不必担心出现重复行，也不必担心出现相同的数据值。
- **ignore_dup_row** 和 **allow_dup_row** 相互排斥。
- 非唯一集群索引允许重复键，但不允许重复行，除非指定 **allow_dup_row**。
- **allow_dup_row** 可以创建包括复制行的表中的非唯一集群索引。如果表具有一个未用 **allow_dup_row** 选项创建的非唯一集群索引，则无法使用 **insert** 或 **update** 命令创建新的重复行。

  如果表中存在具有唯一性的索引，则唯一性的要求优先于 **allow_dup_row** 选项。如果表中存在列的唯一索引，将无法使用 **allow_dup_row** 来创建索引。
- **ignore_dup_row** 选项也用于非唯一的集群索引。 **ignore_dup_row** 选项从批处理数据中消除重复。 **ignore_dup_row** 取消任何会创建重复行的 **insert** 或 **update**，但不回退整个事务。

- 表 6-9 说明 **allow_dup_row** 和 **ignore_dup_row** 如何影响尝试创建表（包括有重复行）中非唯一集群索引，以及如果影响尝试将重复行输入表格。

表 6-9：非唯一集群索引的重复行选项

选项设置	创建具有重复行的表中的索引	将重复行插入具有索引的表中
无选项设置	**create index** 失败。	**insert** 失败。
allow_dup_row 设置	**create index** 完成。	**insert** 完成。
ignore_dup_row 设置	创建索引，但删除重复行；显示错误消息。	插入所有的行，但不包括重复行；显示错误消息。

表 6-10 显示可用于不同类型索引的索引选项：

表 6-10：索引选项

索引类型	选项
集群	**ignore_dup_row** | **allow_dup_row**
唯一集群	**ignore_dup_key**
非集群	无
唯一非集群	**ignore_dup_key**、**ignore_dup_row**

使用唯一约束替代索引

- 作为 **create index** 的替代方法，可以通过使用 **create table** 或 **alter table** 语句指定唯一约束来隐式地创建唯一索引。唯一约束创建表中列的集群或非集群唯一索引。这些**隐式**索引按约束命名，但它们遵循使用 **create index** 创建索引的相同规则。
- 使用 **drop index** 语句无法删除支持唯一约束的索引。通过 **alter table** 语句删除约束或删除表以后，将删除索引。有关唯一约束的详细信息，参见 **create table**。

使用 *sorted_data* 选项可加速排序

- **sorted_data** 选项通过跳过排序步骤以及在某些情况下消除向新页复制数据行的需要，从而可减少创建索引所需要的时间。对大型表操作时速度的提高尤其明显，对 1G 以上的表操作时，速度将成倍提高。

  如果指定 **sorted_data**，但数据未按排序顺序排列， Adaptive Server 将显示错误消息，该命令将失败。

  如果行没有重复键，则可以成功创建非唯一非集群索引。如果行具有重复键， Adaptive Server 显示错误消息，该命令将失败。

- **sorted_data** 对创建集群索引的效果取决于表是否被分区以及 **create index** 命令中是否使用其它选项。对于未分区的表，使用某些选项就需要复制数据，对于已分区的表，使用某些选项则需要排序并复制数据，而其它一些选项只有使用以下选项之一时才要求复制数据：
  - 使用 **ignore_dup_row** 选项。
  - 使用 **fillfactor** 选项。
  - 使用 **on** *segmentname* 子句可指定段，所指定的段不同于表数据所在的段。
  - 使用 **max_rows_per_page** 子句可指定值，该值不同于与表相关联的值。
- 表 6-11 显示对于已分区的表和未分区的表，何时需要排序以及何时复制表。

表 6-11：使用 sorted_data 选项来创建集群索引

选项	已分区的表	未分区的表
未指定选项	并行排序；复制数据，在分区上均匀分配；创建索引树。	并行或非并行排序；复制数据，创建索引树。
仅有 **with sorted_data** 或 **with sorted_data on** *same_segment*	只创建索引树。不执行排序或复制数据。不并行运行。	只创建索引树。不执行排序或复制数据。不并行运行。
with sorted_data 和 **ignore_dup_row** 或 **fillfactor** 或 **on** *other_segment* 或 **max_rows_per_page**	并行排序；复制数据，在分区上均匀分配；创建索引树。	复制数据并创建索引树。不执行排序。不并行运行。

指定直方图的梯级数

- 使用 **with statistics** 子句可指定前导索引列的直方图的梯级数。在查询优化期间使用直方图以确定与列搜索参数匹配的行数。
- 要重新创建索引而不更新 *sysstatistics* 中的列值，使用 0 梯级数。这将避免覆盖使用 **optdiag** 更改的统计信息。

空间管理属性

- **fillfactor**、 **max_rows_per_page** 和 **reservepagegap** 按以下不同方式帮助管理索引页上的空间：
  - **fillfactor** 适用于所有的锁定方案的索引。对于 allpage 锁定的表上的集群索引，此参数将影响表的数据页。在所有其它索引上，它将影响索引的叶级。
  - **max_rows_per_page** 仅适用于 allpage 锁定的表上的索引页。
  - **reservepagegap** 适用于所有的锁定方案的表及索引。
- **reservepagegap** 在以下情况下影响索引中的空间使用：
  - 创建索引时
  - 在索引上执行 **reorg** 命令时
  - 创建集群索引后重新创建非集群索引时
- 当在 **create clustered index** 命令中指定 **reservepagegap** 值时，此值将应用于：
  - allpage 锁定的表中的数据页和索引页
  - DOL 锁定的表中的索引页
- *num_pages* 值指定索引叶级上填充页与空白页之比，这样在要求新的空间时，索引可以在已存在页附近分配空间。例如，值为 10 的 **reservepagegap** 将为每 9 个已使用的页保留一个空白页。
- 在 allpage 锁定的表上，使用 **create clustered index** 指定的 **reservepagegap** 将覆盖以前使用 **create table** 或 **alter table** 指定的任何值。
- 可以使用 **sp_chgattribute** 更改索引的空间管理属性。使用 **sp_chgattribute** 更改属性不会立即影响表中索引的存储。未来的大范围分配 （如运行 **reorg rebuild**），将使用 **sp_chgattribute** 的值。
- 由 **sp_chgattribute** 设置的 **fillfactor** 值将被存储在 *sysindexes* 的 *fill_factor* 列中。因使用 **alter table...lock** 命令或 **reorg rebuild** 命令而重新创建索引时，将应用此 **fillfactor**。

索引选项和锁定模式

- 表 6-12 显示了支持的 allpage 锁定和 DOL 锁定的表的索引选项。在 DOL 锁定的表上，在 **create index** 期间将强制使用 **ignore_dup_row** 和 **allow_dup_row** 选项，但在 **insert** 和 **update** 操作期间不强制使用这些选项。 DOL 锁定的表始终允许插入重复行。

表 6-12：对锁定方案支持的 create index 选项

索引类型	allpage 锁定的表	DOL 锁定的表	
		创建索引期间	插入期间
集群	**allow_dup_row** **ignore_dup_row**	**allow_dup_row** **ignore_dup_row**	**allow_dup_row**
唯一集群索引	**ignore_dup_key**	**ignore_dup_key**	**ignore_dup_key**
非集群索引	无	无	无
唯一非集群索引	**ignore_dup_key**	**ignore_dup_key**	**ignore_dup_key**

表 6-13 显示了试图向具有集群索引的表插入重复行所使用的命令的行为，以及何时删除并重新创建集群索引。

表 6-13：重复行选项的强制和错误

选项	allpage 锁定的表	DOL 锁定的表
未指定选项	插入失败，并给出错误消息 2615。重新创建索引成功。	插入成功。重新创建索引失败，提供错误消息 1508。
allow_dup_row	插入和重新创建索引成功。	插入和重新创建索引成功。
ignore_dup_row	插入失败并提供“Duplicate row was ignored”消息。重新创建索引成功。	插入成功。重新创建索引可以删除重复行。

在 DOL 锁定的表上使用 *sorted_data* 选项

- **create index** 的 **sorted_data** 选项只能紧接在向空表的批量复制操作之后使用。只要对该表的数据修改导致了额外的页分配，就不能使用 **sorted_data** 选项。
- 为空间管理属性指定不同的值可能覆盖 **sorted_data** 的禁止排序功能。

获取有关表和索引的信息

- 在 *sysindexes* 中，每个索引（包括组合索引）由一行表示。
- 有关通过索引检索的数据的顺序信息，以及 Adaptive Server 安装的排序顺序效果，参见 **order by** 子句。
- 有关表的索引的信息，执行 **sp_helpindex** 系统过程。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

create index 权限默认为表的所有者拥有，且不能移交。

参见

命令	**alter table**、 **create table**、 **drop index**、 **insert**、 **order by Clause**、 **set**、 **update**
系统过程	**sp_addsegment**、 **sp_chgattribute**、 **sp_helpindex**、 **sp_helpsegment**、 **sp_spaceused**
实用程序	**optdiag**

create plan

功能

创建抽象计划。

语法

```
create plan query plan
       [into group_name]
       [and set @new_id}
```

关键字和选项

query — 是包含查询的 SQL 文本的文字字符串、参数或局部变量。

plan — 是包含抽象计划表达式的文字字符串、参数或局部变量。

into *group_name* — 指定抽象计划组的名称。

and set @*new_id* — 返回变量中抽象计划的 ID 号。

示例

1. ```
   create plan "select * from titles where price >
   $20" "(t_scan titles)"
   ```

   创建指定查询的抽象计划。

2. ```
   declare @id int
   create plan "select au_fname, au_lname from authors
   where au_id = '724-08-9931' "
   "(i_scan au_id_ix authors)"
   into dev_plans
   and set @id
   select @id
   ```

   在 *dev_plans* 组中创建查询的抽象计划，并返回变量 *@id* 中的计划 ID。

注释

- **create plan** 将抽象计划保存在用 **into** 指定的组中。如果未指定组名，将把该计划保存在当前活动的计划组中。
- 不检查 **create plan** 所指定的查询和抽象计划的 SQL 语法是否有效，也不检查计划的抽象计划语法是否有效。另外，不检查是否与 SQL 文本兼容。应立即检查所有用 **create plan** 创建的计划的正确性，方法是运行 **create plan** 语句中指定的查询。
- 如果组中其它计划具有相同的 SQL 文本，则必须使用 **set plan replace on** 来启用 **replace** 模式。否则， **create plan** 命令将失败。
- 必须先声明 *@new_id*，然后才能将其用于 **and set** 子句。
- 使用 **into** 指定的抽象计划组必须已经存在。

标准和一致性

create plan 是 Transact-SQL 扩展。

权限

缺省情况下所有用户都具有 **create plan** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	**set plan**
系统过程	**sp_add_qpgroup**、 **sp_find_qplan**、 **sp_help_qpgroup**、 **sp_set_qplan**

create procedure

功能

创建可使用一个或多个用户提供的参数的存储过程或扩展存储过程 (ESP)。

语法

```
create procedure [owner.]procedure_name[;number]
   [[(]@parameter_name
       datatype [(length) | (precision [, scale])]
       [= default][output]
   [, @parameter_name
       datatype [(length) | (precision [, scale])]
       [= default][output]]...[)]]
   [with recompile]
   as {SQL_statements | external name dll_name}
```

关键字和选项

procedure_name — 是过程名。它必须遵循标识符的规则，且不能是变量。指定所有者的名称可创建当前数据库中不同用户所拥有的其它同名的过程。 *owner* 的缺省值是当前用户。

;number — 是可选整数，用于对同名过程进行分组，以便可以使用一个 **drop procedure** 语句将其一并删除。在同一应用程序中使用的过程经常以此方法分组。例如，如果用于应用程序 （名为 **orders**） 的过程被命名为 *orderproc;1*、 *orderproc;2* 等，则以下语句：

```
drop proc orderproc
```

删除整个组。

一旦过程被分组，组中的个别过程就不能被删除。例如，语句：

```
drop procedure orderproc;2
```

不允许使用。

如果正在运行**已评估的配置**下的 Adaptive Server，将无法对过程进行分组。已评估配置的要求不允许对过程进行分组，因此每个存储过程都有一个唯一的对象标识符，可以单独删除。如果不允许过程分组，系统安全员必须使用系统过程 **sp_configure** 重新设置配置参数 **allow procedure grouping**。有关已评估的配置的详细信息，参见*系统管理指南*。

parameter_name — 是过程的参数名。执行过程时提供每个参数的值。（**create procedure** 语句中的参数名是可选的，过程无需使用参数。）

参数名前面必须带有 @ 符号，且必须符合标识符的规则。包括 @ 符号的参数名可以最多具有 30 个字符。参数是局部的过程：相同的参数名可以用于其它过程。

如果参数值包含非字母数字字符，则必须将其用引号引起。这包括数据库名或所有者名所限定的对象名，因为它们包括实心句号。如果字符参数的值以数字字符开头，也必须将其用引号引起。

datatype [(*length*) | (*precision* [, *scale*])] — 是参数的数据类型。有关数据类型的详细信息，参见第 1 章 “系统和用户定义的数据类型”。存储过程参数的数据类型不能是 *text* 或 *image* 数据类型，也不能是基础类型为 *text* 或 *image* 的用户定义的数据类型。

char、 *varchar*、 *nchar*、 *nvarchar*、 *binary* 和 *varbinary* 数据类型应用小括号将 *length* 括起。如果省略此长度， Adaptive Server 将截断参数值的字符数为 1。

float 数据类型需要小括号中有一个二进制 *precision*。如果省略此精度， Adaptive Server 将使用适合您的平台的缺省精度。

numeric 和 *decimal* 数据类型要求 *precision* 和 *scale*，这两个参数用小括号括起来并用逗号分开。如果省略精度和标度， Adaptive Server 使用的缺省精度为 18，标度为 0。

default — 定义过程参数的缺省值。如果已定义缺省值，则用户无需提供参数值即可执行过程。缺省值必须是常量。如果过程使用的参数名带有关键字 **like**，它可以包括通配符 %、 _、 [ ] 和 [^]（参见示例 2）。

缺省值可以为 NULL。过程定义可以指定在参数值为 NULL 的情况下执行某些操作 （参见示例 3）。

output — 指示参数是返回参数。可以将它的值返回给调用此过程的 **execute** 命令。使用返回参数可将信息返回调用过程 （参见示例 5）。

要通过多层嵌套的过程返回参数值，每个过程必须包括带有参数名的 **output** 选项，并包括调用最高级别过程的 **execute** 命令。

output 关键字可以缩写为 **out**。

with recompile — 表示 Adaptive Server 不会为该过程保存计划；每次执行过程时将创建新计划。如果预计列过程的执行将不合常规（即需要新计划时），则使用此可选子句。 **with recompile** 子句不影响执行扩展存储过程。

SQL_statements — 指定过程所进行的操作。可以包括任意数目和种类的 SQL 语句，但 **create view**、 **create default**、 **create rule**、 **create procedure**、 **create trigger** 和 **use** 除外。

create procedure SQL 语句经常包括控制流语言，包括以下一项或多项：**declare**、**if...else**、**while**、**break**、**continue**、**begin...end**、**goto** 标签、**return**、 **waitfor**、 **/*** *comment* ***/**。它们还可以引用为过程定义的参数。

只要进行适当地限定， SQL 语句可以引用其它数据库的对象。

- **external name** — 创建扩展存储过程。如果使用 **as external name** 语法，则无法将 *number* 参数用于 **as external name**。

dll_name — 指定动态链接库 (DLL) 或共享库的名称，它们包含执行扩展存储过程的函数。可以指定 *dll_name* 不带扩展名，或带有特定于平台的扩展名，如 Windows NT 的 *.dll* 或 Sun Solaris 的 *.so*。如果指定扩展名，请将整个 *dll_name* 用引号引起。

示例

1.
```
create procedure showind @tabname varchar(30)
as
  select sysobjects.name, sysindexes.name, indid
  from sysindexes, sysobjects
  where sysobjects.name = @tabname
  and sysobjects.id = sysindexes.id
```

假设存在一个表名，该过程 *showind* 将显示该表名称及其任意列的任何索引的名称和标识号。

以下是执行 *showind* 的可接受的语法形式：

```
execute showind titles

execute showind @tabname = "titles"
```

或者，如果这是批处理或文件中第一个语句，则：

```
showind titles
```

2.
```
create procedure
showsysind @table varchar(30) = "sys%"
as
  select sysobjects.name, sysindexes.name, indid
  from sysindexes, sysobjects
  where sysobjects.name like @table
  and sysobjects.id = sysindexes.id
```

如果用户没有提供参数，该过程将显示有关系统表的信息。

3.
```
create procedure
showindnew @table varchar(30) = null
as
if @table is null
   print "Please give a table name"
  else
   select sysobjects.name, sysindexes.name, indid
   from sysindexes, sysobjects
   where sysobjects.name = @table
   and sysobjects.id = sysindexes.id
```

   该过程指定参数为 NULL 时 （即用户未提供参数的情况）要进行的操作。

4.
```
create procedure mathtutor @mult1 int, @mult2 int,
  @result int output
as
select @result = @mult1 * @mult2
```

   该过程将两个整数参数相乘并在 **output** 参数 *@result* 中返回结果。如果传递给该过程 3 个整数，则在执行时， **select** 语句将执行乘法运算并指派值，但不输出返回参数：

```
mathtutor 5, 6, 32

(return status 0)
```

5.
```
declare @guess int
select @guess = 32
exec mathtutor 5, 6, @result = @guess output

(1 row affected)
(return status = 0)

Return parameters:

@result
-----------
         30
```

   在此示例中，过程和 **execute** 语句都包括带有参数名的 **output**，从而该过程可以将值返回给调用方。 **execute** 语句中的输出参数 *@result* 以及任何后续参数**必须**以下列形式传递：

   @parameter = value

   - 不管返回参数值更改与否，始终要报告该值。
   - 因为 *@result* 就是传递给 *mathtutor* 的参数的名称，所以无需在批调用中对其进行声明。

- 虽然 *@result* 的更改值返回给 **execute** 语句中指派的变量中的调用方 （此例中为 *@guess*），但它将在其本身的标题 (*@result*) 下显示。

```
6. 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 "Right-o"
   else
       print "Wrong, wrong, wrong!"

   (1 row affected)
   (1 row affected)
   (return status = 0)

   Return parameters:

   @result
   -----------
            30

   Your_answer Right_answer
   ----------- ------------
            32           30

   (1 row affected)
   Wrong, wrong, wrong!
```

返回参数可用于批处理或调用过程中的其它 SQL 语句。此示例显示调用过程期间当 **execute** 语句将 *@guess* 的值存储在其它变量名 (*@store*) 中后，如何在条件子句中使用该值。如果返回参数用于属于 SQL 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。

```
7. create procedure xp_echo @in varchar(255),
       @out varchar(255) output
   as external name "sqlsrvdll.dll"
```

创建一个名为 *xp_echo* 的扩展存储过程 （该过程使用输入参数 *@in*），使其响应输出参数 *@out*。该过程的代码包含在名为 **xp_echo** 的函数中，该函数被编译并链接到名为 *sqlsrvdll.dll* 的 DLL。

注释

- 创建过程后，可以运行该过程，方法是发出带有该过程名和任意参数的 **execute** 命令。如果过程是批处理中的第一个语句，则可以提供其名称而无需关键字 **execute**。
- 使用 **sp_hidetext** 可以隐藏存储在 *syscomments* 中的过程源文本。
- 如果成功执行存储的批处理过程， Adaptive Server 将把 *@@error* 全局变量设置为 0。

限制

- 存储过程最多可以拥有 255 个参数。
- 过程中的局部与全局变量的最大数仅受可用内存的限制。
- 存储过程中文本的最大容量为 16MB。
- **create procedure** 语句不能在单个批处理中与其它语句结合使用。
- 只可以在当前数据库中创建存储过程，但该过程可以引用其它数据库的对象。任何在过程中引用的对象必须在创建过程时已存在。可以在过程内创建对象，然后引用该对象，但条件是引用对象前先创建该对象。

  无法在过程中使用 **alter table** 添加列继而在过程内引用该列。
- 如果在 **create procedure** 语句中使用 **select ***，该过程不会选出任何已添加到表的新列 （即使使用 **with recompile** 选项来**执行**）。您必须对该过程执行 **drop** 并重新创建它。
- 在存储过程内，无法先创建对象 （包括临时表），然后删除它，再创建同名的新对象。Adaptive Server 是在执行过程时创建存储过程中定义的对象，而不在编译该过程时创建。

◆ ***警告！***

对数据库的某些更改 （如删除和重新创建索引）可以导致对象 ID 的更改。对象 ID 更改后，存储过程将自动重新编译，其大小可能有所增加。请保留此增加的空间。

扩展存储过程

- 如果使用 **as** *external name* 语法， **create procedure** 将注册扩展存储过程 (ESP)。扩展存储过程执行过程语言函数而不是 Transact-SQL 命令。
- 在 Windows NT 上，ESP 函数不调用 C 运行期信号例程。这会导致 XP Server 失败，因为 Open Server™ 不支持 Windows NT 上的信号处理。
- 要支持多线程， ESP 函数应使用 Open Server **srv_yield** 函数，该函数挂起 XP Server 线程并对其重新安排，以允许执行其它相同或更高优先级的线程。
- DLL 搜索机制取决于平台。以下是 Windows NT 上 DLL 文件名搜索的顺序：

  a. 从其装载应用程序的目录

  b. 当前目录

  c. 系统目录 (SYSTEM32)

  d. PATH 环境变量中所列目录

  如果 DLL 不在上述三个目录，则设置 PATH 以包括 DLL 所在的目录。

  在 UNIX 平台上，搜索方法因特定平台而异。如果无法查找到 DLL 或共享库，将搜索 *$SYBASE/lib*。

  不支持绝对路径名。

系统过程

- 系统管理员可以在 *sybsystemprocs* 数据库中创建新的系统过程。系统过程名开始的字符必须是 “sp_”。通过指定过程名，可从任何数据库执行这些过程；而无需使用 *sybsystemprocs* 数据库名称对其进行限定。有关创建系统过程的详细信息，参见*系统管理指南*。
- 系统过程结果可能会根据它们的执行环境的不同而有所不同。例如，系统过程 *sp_foo* （执行 **db_name()** 系统函数）将返回执行此过程的数据库的名称。当从 *pubs2* 数据库执行时，它会返回 “pubs2”。

```
use pubs2
sp_foo
------------------------------
pubs2
```

当从 *sybsystemprocs* 执行时，它会返回 “sybsystemprocs”。

```
use sybsystemprocs
sp_foo

------------------------------
sybsystemprocs
```

过程返回状态

- 存储过程可以返回名为 **return status** 的整数值。返回状态指示过程已成功执行，或指定发生的错误类型。
- 执行存储过程后，将自动返回相应的状态代码。 Adaptive Server 在当前返回以下状态代码：

代码	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

从 -15 至 -99 的代码留作将来使用。

- 用户可以使用 **return** 语句生成用户定义的返回状态。状态可以为 0 至 -99 以外的任意整数。下面示例中，如果书籍具有有效合同，则返回 “1”，如果为其它任何情况则返回 “2”：

```
create proc checkcontract @titleid tid
as
if (select contract from titles where
        title_id = @titleid) = 1
    return 1
else
    return 2

checkcontract @titleid = "BU1111"

(return status = 1)
```

```
checkcontract @titleid = "MC3026"
(return status = 2)
```

- 如果在执行时发生了多个错误，将返回具有最大绝对值的状态代码。用户定义的返回值优先于系统定义的值。

对象标识符

- 要更改存储过程的名称，使用 **sp_rename**。
- 要更改扩展存储过程的名称，先删除该过程，然后重命名并重新编译支持函数，再重新创建该过程。
- 如果过程引用的表名、列名或视图名是无效的标识符，则必须在 **create procedure** 命令前执行 **set quoted_identifier on**，并将这样的每个名称用双引号引起。执行该过程时，**quoted_identifier** 选项**不**需要“打开”。
- 如果重命名任何该过程所引用的对象，必须删除该过程并重新创建。
- 如果其他用户要使用存储过程，则在存储过程内，必须使用对象所有者的名称限定用于 **create table** 和 **dbcc** 命令的对象名。例如，如果用户 “mary”拥有表 *marytab*，她的表名用于上述这些命令，则要使其他用户能够执行存储过程，应该在存储过程内对其表名进行限定。这是因为在运行过程时将对对象名进行解析。如果其他用户试图执行该过程， Adaptive Server 将查找用户“mary”所拥有的名为 *marytab* 的表，而不是查找由执行该存储过程的用户所拥有的名为 *marytab* 表。

  存储过程内用于其它语句 （如 **select** 或 **insert**）的对象名无需限定，原因是这些名称将在编译过程时进行解析。

临时表和过程

- 如果在当前会话中创建了临时表，则可以创建过程来引用该临时表。当退出过程后，在该过程内创建的临时表将消失。有关详细信息，参见 *Transact-SQL User's Guide*。
- 系统过程 （如 **sp_help**）使用临时表，但条件是必须从 *tempdb* 使用。

设置过程中的选项

- 可以在存储过程内使用 **set** 命令。在执行过程期间，多数 **set** 选项保持有效，然后可回复至原来的设置。

  然而，如果使用的 **set** 选项 （如 **identity_insert**）要求用户为对象所有者，则不是对象所有者的用户将无法执行该存储过程。

获取有关过程的信息

- 要获得过程所引用的对象的报告，使用 **sp_depends**。
- 要显示存储在 *syscomments* 中的 **create procedure** 语句的文本，请将过程名用作参数，使用系统过程 **sp_helptext**。如果使用 **sp_helptext**，必须使用驻留有该过程的数据库。要显示系统过程的文本，从 *sybsystemprocs* 数据库执行 **sp_helptext**。
- 要查看系统扩展存储过程及其支持的 DLL 的列表，从 *sybsystemprocs* 数据库使用 **sp_helpextendedproc**。

嵌套过程

- 当一个存储过程调用另一存储过程时就发生过程嵌套。
- 如果执行一个调用其它过程的过程，被调用过程可以访问调用过程所创建的对象。
- 嵌套级别在被调用的过程开始执行时会递增，而在其执行结束时递减。如果超过最大嵌套级别 16，将导致事务失败。
- 可以通过名称或变量名 （从而替代实际的过程名）来调用其它过程。
- 当前的嵌套级别存储在 *@@nestlevel* 全局变量中。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

create procedure 权限缺省情况下为数据库所有者拥有，其可将此权限移交给其他用户。

必须使用 **grant** 命令明确授予使用过程的权限，可以使用 **revoke** 命令撤消此权限。

对象的权限：过程创建时间

创建过程时， Adaptive Server 不对过程所引用的对象 （如表和视图）进行权限检查。因此，即使不具有访问其对象的权限，也可以成功创建该过程。所有权限的检查将在用户执行该过程时进行。

对象的权限：过程执行时间

执行过程时，对对象的权限检查取决于是否过程和所有引用的对象为同一用户所拥有。

- 如果过程的对象为其他用户所拥有，则调用者必须被授予直接访问对象的权限。例如，如果过程要从用户无法访问的表执行查询，此过程的执行将失败。
- 但如果过程及其对象为同一用户所拥有，就可应用特殊规则。调用者虽然不能直接访问过程对象，但是将自动具有“隐式的权限”来访问它们。无需授予用户直接的访问表和视图的权限，可以使用存储过程提供给他们限制的访问权限。这样存储过程可以成为安全性机制。例如，过程的调用者可能只能访问表中的某些行和列。

  系统管理指南 中将对隐式权限的规则进行详细说明。

参见

命令	**begin...end**、 **break**、 **continue**、 **declare**、 **drop procedure**、 **execute**、 **goto 标签**、 **grant**、 **if...else**、 **return**、 **select**、 **waitfor**、 **while**
系统过程	**sp_addextendedproc**、 **sp_helpextendedproc**、 **sp_helptext**、 **sp_hidetext**、 **sp_rename**

create proxy_table

（仅对组件集成服务）

功能

创建代理表而无需指定列表。组件集成服务使用从远程表获得的元数据来派生列的列表。

语法

```
create proxy_table table_name
   [ external table ]
   at pathname
```

关键字和选项

table_name — 指定要由后续语句使用的本地代理表名。 *table_name* 采用以下形式：

dbname.owner.object

其中 *dbname* 和 *owner* 是可选的，分别表示本地数据库及所有者名。如果没有指定 *dbname*，则在当前数据库中创建表；如果没有指定 *owner*，则表为当前用户拥有。如果指定了 *dbname* 或 *owner* 之一，则必须将整个 *table_name* 放在引号中。如果只有 *dbname*，则 *owner* 需要有占位符。

external table — 指定对象是一个远程表或视图。该子句是可选的，缺省为 **external table**。

at *pathname* — 指定远程对象的位置。 *pathname* 采用以下形式：

server_name.dbname.owner.object

其中：

- *server_name* （必需）是包含远程对象的服务器名
- *dbname* （可选）是由包含该对象的远程服务器管理的数据库名
- *owner* （可选）是拥有远程对象的远程服务器用户名
- *object* （必需）是远程表或视图的名称

示例

1.
```
create proxy_table t1
at "SERVER_A.db1.joe.t1"
```

创建一个映射到远程表 *t1*、名为 *t1* 的代理表。 CIS 从远程表获取列列表。

注释

- **create proxy_table** 是 **create existing table** 命令的变体。可以使用 **create proxy_table** 来创建代理表，但 （与 **create existing table** 不同）不指定列的列表。 CIS 使用从远程表获取的元数据来得到列的列表。
- 由 **at** 关键字提供的位置信息与由 **sp_addobjectdef** 系统过程提供的信息相同。这些信息将被存储在 *sysattributes* 表中。
- 如果远程服务器对象不存在，该命令将被拒绝，并给出错误消息。
- 如果对象存在，则更新本地系统表。每一列都被使用。并获得表或视图的列及其特性。
- CIS 自动将列的数据类型转换为一种 Adaptive Server 数据类型。如果不能进行转换，则 **create proxy_table** 命令不允许定义表。
- 将抽取远程服务器表的索引信息，并用于创建系统表 *sysindexes* 的行。这将用 Adaptive Server 的术语定义索引和键，并启用查询优化程序来考虑表中任何可能存在的索引。
- 定义代理表后，对表发出 **update statistics** 命令。这将允许查询优化程序在考虑连接顺序时做出正确选择。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

create proxy_table 权限缺省情况下为表的所有者拥有，且不能移交。

参见

命令	create existing table、 create table

create role

功能

创建用户定义的角色；创建时指定口令有效期、最短口令长度和所允许的指定角色的最大失败登录次数。

语法

```
create role role_name [ with passwd "password" ]
   [, "passwd expiration" | "min passwd length" |
   "max failed_logins" ] option_value ] ]
```

关键字和选项

role_name — 是新角色的名称。该角色名称对于服务器来说必须是唯一的，并且必须遵循标识符的规则。它不能是变量。

with passwd — 用户为激活角色而必须输入的附加口令。

password — 是附加到角色的口令。口令的长度必须至少为 6 个字符，并且必须遵循标识符的规则。口令不能使用变量。

role_name 是新角色的名称。该角色名称对于服务器来说必须是唯一的，并且必须遵循标识符的规则。它不能是变量。

with passwd 用户为激活角色而必须输入的附加口令。

password 是附加到角色的口令。口令的长度必须至少为 6 个字符，并且必须遵循标识符的规则。口令不能使用变量。

passwd expiration 指定口令的有效期 （以天数表示）。它可以是 0 到 32767 之间的任意值，包括 0 和 32767。

min passwd length 指定指定的登录所需的口令的最短长度。

max failed_logins 指定允许的指定登录的登录失败尝试次数。

option_value 指定 **passwd expiration**、 **min passwd length** 或 **max failed_logins** 的值。

示例

1. **create role doctor_role**

   创建名为 doctor_role 的角色。

2. **create role doctor_role with passwd "physician"**

   创建名为 doctor_role 的角色，口令为 “physician”。

3. **create role intern_role, with passwd "temp244", passwd expiration 7**

   设置 *intern_role* 的口令有效期。

4. **create role intern_role with passwd “temp244” max failed_logins 20**

   设置允许的 *intern_role* 的最大失败登录次数。

5. **create role intern_role with passwd "temp244", min passwd length 0**

   设置 *intern_role* 的最短口令长度。

注释

- **create role** 命令创建具有所指定的特权、权限和限制的角色。有关如何使用 **create role** 的详细信息，参见*系统管理指南*。

  有关监控和限制访问对象的信息，参见 **set role** 命令。

- 从 *master* 数据库使用 **create role**。
- 使用 **with passwd** *password* 子句可在创建时将口令附加给角色。如果将口令附加给角色，则被授予该角色的用户必须指定该口令以激活角色。

  有关在创建后将口令添加到角色的信息，参见 **alter role** 命令。

➤ ***注意***

附加给用户定义角色的口令未到期。

- 角色名称对于服务器来说必须是唯一的。
- 角色名称一定不能是用户名称。可以创建与用户同名的角色，但在授予特权时，Adaptive Server 解析命名冲突的方式是将特权授予用户而不是角色。

  有关命名冲突的详细信息，参见 **grant role** 命令。

限制

- 可以创建的每个服务器会话的角色最多可为 1024 个。然而有 32 个角色要保留作为 Sybase 系统角色，如 **sa_role** 和 **sso_role**。因此，每个服务器会话可以创建最多 992 个用户定义的角色。
- 如果创建的角色附加有口令，则用户缺省登录时无法激活该角色。如果授予该角色的用户需要在缺省登录时激活该角色，请不要创建带有附加口令的角色。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

只有系统安全员才能使用 **create role**。

create role 权限不包括在 **grant all** 命令中。

参见

命令	**alter role**、 **drop role**、 **grant**、 **revoke**、 **set**
系统过程	**sp_activeroles**、 **sp_displaylogin**、 **sp_displayroles**、**sp_helprotect**、**sp_modifylogin**

create rule

功能

为特定列或具有用户定义数据类型的任意列指定可接受值的域。

语法

```
create rule [owner.]rule_name
   as condition_expression
```

关键字和选项

rule_name — 是新规则的名称。它必须遵循标识符的规则，且不能是变量。指定所有者的名称可创建当前数据库中不同用户所拥有的其它同名的规则。 *owner* 的缺省值是当前用户。

condition_expression — 指定定义该规则的条件。可以是 **where** 子句中任何有效的表达式，而且可以包括算术运算符、关系运算符、**in**、**like** 和 **between** 等。然而，它无法引用列或其它数据库对象。**可以**包括不引用数据库对象的内部函数。

condition_expression 使用一个参数。该参数具有一个前缀 @ 符号，它引用通过 **update** 或 **insert** 命令所输入的值。编写规则时可以使用任何名称或符号来表示值，但是第一个字符必须是 @ 符号。将字符和日期常量用引号引起，二进制常量前面需带有“0x”。

示例

1.
```
create rule limit
as @advance < $1000
```

   创建名为 *limit* 的规则，该规则将 *advance* 的值限制在 $1000 以内。

2.
```
create rule pubid_rule
as @pub_id in ('1389', '0736', '0877')
```

   创建名为 *pubid_rule* 的规则，该规则将 *pub_id* 的值限制为 1389、0736 或 0877。

3.
```
create rule picture
as @value like '_-%[0-9]'
```

   创建名为 *picture* 的规则，该规则将 *value* 的值限制为总以指定字符开头。

注释

- 要隐藏规则文本，请使用 **sp_hidetext**。
- 要重命名规则，使用 **sp_rename**。

限制

- 只可在当前数据库中创建规则。
- 规则不适用于创建规则时已存在于数据库中的数据。
- **create rule** 语句无法在单个批处理中与其它语句结合使用。
- 无法将规则绑定到 Adaptive Server 提供的数据类型或者 *text*、*image* 或 *timestamp* 类型的列。
- 创建新的同名规则以前必须删除规则，删除规则前必须将规则解除绑定。使用：

  sp_unbindrule *objname* [, futureonly]

绑定规则

- 使用系统过程 **sp_bindrule** 可将规则绑定到列或用户定义的数据类型。此命令的语法为：

  sp_bindrule *rulename*, *objname* [, futureonly]

- 当将值插入 （或更新）该类型的列时，绑定到用户定义的数据类型的规则将被激活。规则**不**测试插入到该类型变量的值。
- 该规则必须与列的数据类型兼容。例如，不能将

  @value like A%

  用作精确数值的列或近似数值的列的规则。如果规则与其所绑定的列不兼容， Adaptive Server 将在试图插入值时 （而不是在绑定规则时）生成错误消息。
- 可以将规则绑定到列或数据类型而无需解除绑定现有的规则。

- 不管哪个规则最先被绑定，绑定到列的规则总是优先于绑定到用户定义的数据类型的规则。当将规则绑定到已有规则存在的列和用户定义的数据类型时，其优先顺序如以下图表所示：

表 6-14：规则绑定优先顺序

新规则的绑定位置	绑定到用户定义的数据类型的旧规则	绑定到列的旧规则
用户定义的数据类型	新规则替换旧规则	没有变化
列	新规则替换旧规则	新规则替换旧规则

规则和空值

- 规则不替换列定义。如果将规则绑定到允许空值的列，则可向列中插入 NULL，无论是显式还是隐式，即使规则文本中不包括 NULL 也是可以的。例如，如果创建一个指定 “@val in (1,2,3)” 或 “@amount > 10000” 的规则，并且将该规则绑定到允许空值的表的列，则仍可以将 NULL 插入该列。列的定义将替换规则。

获取有关规则的信息

- 要获取有关规则的报告，请使用 **sp_help**。
- 要显示存储在 *syscomments* 系统表中的规则文本，将规则名作为参数执行系统过程 **sp_helptext**。
- 将规则绑定到特定列或用户定义的数据类型后，其 ID 将存储在 *syscolumns* 或 *systypes* 系统表中。

缺省值和规则

- 如果列具有相关联的缺省值和规则，缺省值必须包括在该规则所定义的域中。无法插入与规则相冲突的缺省值。 Adaptive Server 每次试图插入这类缺省值时，将生成错误消息。

使用完整性约束替代规则

- 还可以使用 **create table** 语句的 **check** 完整性约束来定义规则。然而这些约束特定于该表；无法将其绑定到其它的表。有关完整性约束的信息，参见 **create table** 和 **alter table**。

标准和一致性

标准	一致性级别	注释
SQL92	Transact-SQL 扩展	要创建使用符合 SQL92 语法的规则，使用 **create table** 语句的 **check** 子句。

权限

create rule 权限缺省设置为数据库所有者拥有，其可将此权限移交给其他用户。

参见

命令	**alter table**、 **create default**、 **create table**、 **drop default**、 **drop rule**
系统过程	**sp_bindrule**、 **sp_help**、 **sp_helptext**、 **sp_hidetext**、 **sp_rename**、 **sp_unbindrule**

create schema

功能

为数据库用户创建新的表、视图和权限集合。

语法

```
create schema authorization authorization_name
   create_oject_statement
       [ create_object_statement ... ]
   [ permission_statement ... ]
```

关键字和选项

authorization_name — 必须是数据库中当前用户的名称。

create_object_statement — 是 **create table** 或 **create view** 语句。

permission_statement — 是 **grant** 或 **revoke** 命令。

示例

```
1. create schema authorization pogo
       create table newtitles (
           title_id tid not null,
           title varchar(30) not null)

       create table newauthors (
           au_id id not null,
           au_lname varchar(40) not null,
           au_fname varchar(20) not null)

       create table newtitleauthors (
           au_id id not null,
           title_id tid not null)

       create view tit_auth_view
       as
           select au_lname, au_fname
               from newtitles, newauthors,
                   newtitleauthors
           where
           newtitleauthors.au_id = newauthors.au_id
           and
           newtitleauthors.title_id =
                newtitles.title_id

       grant select on tit_auth_view to public
       revoke select on tit_auth_view from churchy
```

创建 *newtitles*、 *newauthors* 和 *newtitleauthors* 表、 *tit_auth_view* 视图和相应的权限。

注释

- 只可在当前数据库中创建模式。
- *authorization_name* 也称为**模式授权标识符**，它必须是当前用户的名称。
- 该用户必须具有正确的命令权限 （**create table** 和 / 或 **create view**）。如果用户创建其他数据库用户拥有的表的视图，则将在用户尝试通过视图访问数据时 （而非创建该视图时）检查视图的权限。
- **create schema** 命令通过以下方式终止：
  - 常规命令终止符 （**isql** 中的缺省值是 “go”）。
  - **create table**、 **create view**、 **grant** 或 **revoke** 以外的任何语句。
- 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** 将向系统表添加有关表、视图和权限的信息。使用适当的删除命令 （**drop table** 或 **drop view**）可删除 **create schema** 所创建的对象。模式中授予的或删除的权限可由模式创建语句以外的标准 **grant** 和 **revoke** 命令进行更改。

标准和一致性

标准	一致性级别
SQL92	初级一致性

权限

create schema 可由数据库的任何用户执行。用户必须具有权限 （即 **create table** 和 / 或 **create view** 权限）以创建模式中指定的对象。

参见

命令	create table、 create view、 grant、 revoke
实用程序命令	isql

create table

功能

创建新表和可选的完整性约束；为所创建的表指定锁方案；在创建依赖于索引的参照完整性约束时指定升序或降序的索引顺序；指定预期的行大小以减少行转移；指定为每个填充页保留的空页比率；允许您将表映射到远程位置的表、视图或过程。

语法

```
create table [database.[owner].]table_name
(column_name datatype
    [default {constant_expression | user | null}]
    {[{identity | null | not null}]
        [off row | in row]
    | [[constraint constraint_name]
        {{unique | primary key}
       [clustered | nonclustered] [asc | desc]
       [with { { fillfactor = pct
                   | max_rows_per_page = num_rows }
                   , reservepagegap = num_pages }]
        [on segment_name]
        | references [[database.]owner.]ref_table
            [(ref_column)]
        | check (search_condition)}]}...
    | [constraint constraint_name]
    {{unique | primary key}
        [clustered | nonclustered]
        (column_name [asc | desc]
            [{, column_name [asc | desc]}...])
       [with { {fillfactor = pct
             | max_rows_per_page = num_rows },
            reservepagegap = num_pages } ]
         [on segment_name]
    |foreign key (column_name [{, column_name}...])
        references [[database.]owner.]ref_table
            [(ref_column [{, ref_column}...])]
     | check (search_condition) ... }
[{, {next_column | next_constraint}}...])
[lock {datarows | datapages | allpages }]
[with { max_rows_per_page = num_rows ,
        exp_row_size = num_bytes ,
        reservepagegap = num_pages } ]
[on segment_name]
[ [ external table ] at pathname ]
```

关键字和选项

table_name — 是新表的显式名称。如果该表位于其它数据库，则指定数据库名；如果数据库中有多个表使用该名称，则指定所有者的名称。 *owner* 的缺省值是当前用户，而 *database* 的缺省值是当前数据库。

表名不能使用变量。表名在数据库中必须唯一，且对所有者也必须唯一。如果设置 **set quoted_identifier on**，就可以将分隔标识符用作表名。否则，它必须符合标识符规则。有关有效表名的详细信息，参见第 3 章 “表达式、标识符和通配符”中的 “标识符”。

如果在表名前加井号 (#) 或 “tempdb”前缀，则可创建临时表。有关详细信息，参见第 3 章 “表达式、标识符和通配符”中的 “以 # 开头的表 （临时表）”。

您可以在其它数据库中创建表，条件是您在 *sysusers* 表中列出，并具有该数据库的 **create table** 权限。例如，要在数据库 *otherdb* 中创建名为 *newtable* 的表：

```
create table otherdb..newtable
```

或：

```
create table otherdb.yourname.newtable
```

column_name — 是表中列的名称。它在表中必须唯一。如果设置 **set quoted_identifier on**，就可以将分隔标识符用作该列的名称。否则，它必须符合标识符规则。有关有效列名的详细信息，参见第 3 章 “表达式、标识符和通配符”中的 “标识符”。

datatype — 是列的数据类型。可以是系统或用户定义的数据类型。某些数据类型要求在括号中指定长度 *n*：

```
datatype(n)
```

另一些则要求指定精度 *p* 和小数位数 *s*：

```
datatype(p,s)
```

有关详细信息，参见 “数据类型”。

如果数据库中已启用 Java， *datatype* 可以是已安装在数据库中的 Java 类 （系统类或用户定义类）的名称。有关详细信息，参见 *Adaptive Server Enterprise 中的 Java*。

default — 指定列的缺省值。如果指定了缺省值，而用户在插入数据时没有为该列提供值，则 Adaptive Server 将插入该缺省值。缺省值可以是常量表达式。**user**，用以插入执行插入操作的用户的名称，或 **null**，用以插入空值。 Adaptive Server 生成缺省值的名称，其形式为 *tabname_colname_objid*，其中 *tabname* 是表名的前 10 个字符， *colname* 是列名的前 5 个字符， *objid* 是缺省值的对象 ID 号。为具有 IDENTITY 属性的列声明的缺省值对列值没有影响。

constant_expression — 是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部函数。该缺省值必须与列的数据类型兼容，否则当 Adaptive Server 尝试插入缺省值时，将生成数据类型转换错误。

user | null — 指定当用户未提供值时， Adaptive Server 应插入用户名或空值作为缺省值。对于 **user**，列的数据类型必须是 *char(30)* 或者 *varchar(30*)。对于 **null**，列必须允许空值。

identity — 表示列具有 IDENTITY 属性。数据库中的每个表都可具有一个类型为 *numeric* 且小数位数为 0 的 IDENTITY 列。IDENTITY 列不可更新，且不允许空值。

IDENTITY 列用于存储顺序号，例如由 Adaptive Server 自动生成的发票号或职员号。 IDENTITY 列的值唯一地标识表的每一行。

null | **not null** — 指定在没有缺省值的情况下，Adaptive Server 在数据插入期间的行为。

null 指定在用户没有提供值的情况下， Adaptive Server 将分配一个空值。

not null 指定在没有缺省值的情况下，用户必须提供一个非空值。

如果未指定 **null** 或 **not null**， Adaptive Server 将使用缺省值 **not null**。然而，可使用 **sp_dboption** 切换该缺省值，以使缺省值与 SQL 标准兼容。

off row | **in row** — 指定是将 Java-SQL 列存储在行外 (**off row**)，还是将其存储在行中直接分配的存储空间中 (**in row**)。

一个 in-row 列的存储空间不得超过 255 字节。缺省值是 **off row**。

有关详细信息，参见 *Adaptive Server Enterprise 中的 Java*。

constraint — 引入完整性约束的名称。

constraint_name — 是约束的名称。它必须符合标识符规则，并且在数据库中必须唯一。如果未指定参照或检查约束的名称，Adaptive Server 将生成形式为 *tabname_colname_objectid* 的名称，其中 *tabname* 是表名的前 10 个字符， *colname* 是列名的前 5 个字符， *objectid* 是约束的对象 ID 号。如果未指定唯一约束或主键约束的名称， Adaptive Server 将生成格式为 *tabname_colname_tabindid* 的名称，其中 *tabindid* 是表 ID 和索引 ID 的字符串并置。

unique — 约束指定列中的值，以使任何两行都不会具有相同的值。此约束将创建一个唯一索引。只有在使用 **alter table** 删除此约束之后，才能删除该索引。

primary key —约束指定列中的值，以使任何两行都不会具有相同的值，并且值不为 NULL。此约束将创建一个唯一索引。只有在使用 **alter table** 删除此约束之后，才能删除该索引。

clustered | **nonclustered** — 指定 **unique** 或 **primary key** 约束所创建的索引是集群索引还是非集群索引。 **clustered** 是主键约束的缺省值； **nonclustered** 是唯一约束的缺省值。每个表只能有一个集群索引。有关详细信息，参见 **create index**。

asc | desc — 指定为约束创建的索引对于每一列将是升序还是降序。缺省值为升序。

fillfactor — 指定 Adaptive Server 在创建现有数据的新索引时，对每一页的填充程度。 **fillfactor** 百分比仅在创建索引后才有意义。随着数据的变化，页不会维持在某个特定的填充程度。

fillfactor 的缺省值是 0 ； 当 **create index** 语句不包括 **with fillfactor** 时，将使用该缺省值 （除非已经使用 **sp_configure** 更改该值）。当指定 **fillfactor** 时，应使用 1 到 100 之间的值。

如果 **fillfactor** 为 0，创建集群索引时将完全填充页，而创建非集群索引时将完全填充叶页。这样，集群索引和非集群索引的索引 B 树内都将保留适当的空间量。 **fillfactor** 很少需要更改。

如果将 **fillfactor** 设置为 100，那么 Adaptive Server 在创建集群和非集群索引时，都将百分之百地完全填充每一页。 100 的 **fillfactor** 仅对只读表 （将不再添加其它数据的表）有意义。

如果 **fillfactor** 值小于 100 （除了 0 这一特例），将使 Adaptive Server 在创建新索引时不将页完全填充。在最终将保存大量数据的表上创建索引时， **fillfactor** 为 10 是适当的选择，但是较小的 **fillfactor** 值会使每个索引 （或索引和数据）占用更多的存储空间。

如果启用了组件集成服务，则不能将 **fillfactor** 用于远程服务器。

◆ ***警告！***

用 fillfactor 创建集群索引会影响数据占用的存储空间量，因为 Adaptive Server 会在创建集群索引时重新分配数据。

max_rows_per_page — 限制索引中数据页和叶级页的行数。与 **fillfactor** 不同， **max_rows_per_page** 的值在插入或删除数据时仍保持不变。

如果未指定 **max_rows_per_page** 的值， Adaptive Server 将在创建表时使用值 0。表和集群索引的值介于 0 和 256 之间。非集群索引的每页最大行数取决于索引键的大小。如果指定的值太大， Adaptive Server 将返回错误消息。

如果 **max_rows_per_page** 为 0，创建集群索引时将完全填充页，而创建非集群索引时将完全填充叶页。这样，集群索引和非集群索引的索引 B 树内都将保留适当的空间量。

如果将较小的值用于 **max_rows_per_page**，可减少对常用数据的锁争用。但是，较小值也会使 Adaptive Server 在创建新索引时不完全填充页，因而将占用更多的存储空间并导致更多的页面拆分。

如果启用了组件集成服务，则不能将 **max_rows_per_page** 用于远程服务器。

on *segment_name* — 指定索引将在已命名的段上创建。使用 **on** *segment_name* 选项之前，必须用 **disk init** 将设备初始化，并且必须用 **sp_addsegment** 系统过程在数据库中添加指定段。如需数据库中可用段名的列表，请与系统管理员联系或使用 **sp_helpsegment**。

如果指定 **clustered** 并使用 **on** *segment_name* 选项，整个表将迁移到指定段，因为索引的叶级包含实际的数据页。

references — 为参照完整性约束指定列的列表。对一个列约束只能指定一个列值。在将该约束包括在引用其它表的表中时，插入到**引用**表的任何数据必须已经存在于**被引用**表中。

要使用该约束，必须具有被引用表的 **references** 权限。被引用表上的指定列必须受唯一索引 （由 **unique** 约束或 **create index** 语句创建）的约束。如果未指定列，被引用表中的相应列必须有 **primary key** 约束。此外，引用表列的数据类型必须与被引用表列的数据类型相匹配。

foreign key — 指定所列出的列是表的外键，该表的目标键是列在随后 **references** 子句中的列。仅允许对表级约束使用外键语法，而不允许对列级约束使用。

ref_table — 是包括被引用列的表的名称。您可以引用其它数据库中的表。约束最多可引用 192 个用户表和内部生成的工作表。

ref_column — 是被引用表中一个或多个列的名称。

check — 指定 Adaptive Server 对表中所有行强制执行的 *search_condition* 约束。您可以将 **check** 约束指定为表或列约束；**create table** 允许一个列定义中包含多个 **check** 约束。

search_condition — 是对列值的 **check** 约束。这些约束包括：

- 用 **in** 引入的一系列常量表达式。
- 用 **like** 引入的一组条件，可以包含通配符。

列和表级检查约束可引用表中的任何列。

表达式可包括算术运算符和函数。 *search_condition* 不能包含子查询、集合函数、主变量或参数。

next_column | *next_constraint* — 表示可使用为列定义或表约束定义所述的相同语法来包含更多的列定义或表约束 （用逗号分隔）。

lock datarows | datapages | allpages — 指定表要使用的锁方案。缺省值为配置参数 **lock scheme** 的全服务器范围的设置。

exp_row_size = *num_bytes* — 指定期望行大小；仅应用于 **datarows** 和 **datapages** 锁方案，且仅用于具有可变长度行的表。有效值是 0、1 和表长度最大值和最小值之间的任一值。缺省值是 0，表示应用全服务器范围的设置。

reservepagegap = *num_pages* — 指定扩展 I/O 分配操作期间填充页与要保留的空白页之比。对于每个指定的 *num_pages*，保留一个空白页以用于表的未来扩展。有效值为 0-255。缺省值为 0。

external table — 指定对象是一个远程表或视图。缺省值为 **external table**，这种指定是可选的。

at *pathname* — 指定远程对象的位置。 *pathname* 采用以下形式：

server_name.dbname.owner.object;aux1.aux2

其中：

- *server_name* （必需）是包含远程对象的服务器的名称。
- *dbname* （可选）是该对象所在远程服务器管理的数据库的名称。
- *owner* （可选）是拥有远程对象的远程服务器用户的名称。
- *object* （必需）是远程表或视图的名称。

- *aux1.aux2*（可选）是一个字符串，在执行 **create table** 或 **create index** 命令期间，该字符串将传递给远程服务器。仅当服务器是 *db2* 类时才使用该字符串。*aux1* 是放置表的 DB2 数据库，*aux2* 是放置表的 DB2 表空间。

on *segment_name* — 指定要在其上放置表的段的名称。当使用 **on** *segment_name* 时，逻辑设备必须已用 **create database** 或 **alter database** 分配给数据库，且段必须已用 **sp_addsegment** 命令在数据库中创建。如需数据库中可用段名的列表，请与系统管理员联系或使用 **sp_helpsegment**。

示例

1.
```
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)
```

   创建 *titles* 表。

2.
```
create table "compute"
("max" int, "min" int, "total score" int)
```

   创建 *compute* 表。表名和列名 *max* 和 *min* 放在双引号中，因为它们是保留字。 *total score* 列名放在双引号中，因为它包含嵌入的空白。创建表之前，必须设置 **set quoted_identifier on**。

3.
```
create table sales
(stor_id         char(4)      not null,
ord_num          varchar(20)  not null,
date             datetime     not null,
unique clustered (stor_id, ord_num))
```

   用一步完成 *sales* 表和集群索引的创建，并带有唯一约束。（在 *pubs2* 数据库安装脚本中，有单独的 **create table** 和 **create index** 语句。）

4.
```
create table salesdetail
(stor_id    char(4)                 not null,
ord_num     varchar(20)             not null,
title_id     tid                    not null
             references titles(title_id),
qty          smallint default 0     not null,
discount    float                   not null,

constraint salesdet_constr
    foreign key (stor_id, ord_num)
    references sales(stor_id, ord_num))
```

   创建表 *salesdetail*，它带有两个参照完整性约束和一个缺省值。有一个名为 *salesdet_constr* 的表级参照完整性约束，且列 *title_id* 上有一个列级参照完整性约束，它没有指定的名称。两个约束都指定在被引用表中有唯一索引的列 （*titles* 和 *sales*）。 *qty* 列的 **default** 子句指定 0 作为其缺省值。

5.
```
create table publishers
(pub_id char(4) not null
   check (pub_id in ("1389", "0736", "0877",
       "1622", "1756")
    or pub_id like "99[0-9][0-9]"),
pub_name     varchar(40)  null,
city     varchar(20)  null,
state     char(2)  null)
```

   创建表 *publishers*，其 *pub_id* 列上有检查约束。此列级约束可替代 *pubs2* 数据库中的 *pub_idrule*：

```
create rule pub_idrule
as @pub_id in ("1389", "0736", "0877",
    "1622", "1756")
or @pub_id like "99[0-9][0-9]"
```

6.
```
create table sales_daily
(stor_id       char(4)     not null,  ord_num
numeric(10,0)  identity,  ord_amt       money
null)
```

   将 *ord_num* 列指定为 *sales_daily* 表的 IDENTITY 列。当您第一次将行插入表中时， Adaptive Server 会为 IDENTITY 列分配值 1。随后每次插入时，列值都增加 1。

```
7. create table new_titles (
        title_id    tid,
        title       varchar(80) not null,
        type        char(12) ,
        pub_id      char(4) null,
        price       money null,
        advance     money null,
        total_sales int null,
        notes       varchar(200) null,
        pubdate     datetime,
        contract    bit    )
   lock datapages
   with exp_row_size = 200
```

为 *new_titles* 表指定 **datapages** 锁方案，并指定期望行大小为 200。

```
8. create table new_publishers (
   pub_id          char(4) not null,
   pub_name        varchar(40) null,
   city            varchar(20) null,
   state           char(2) null )
   lock datarows
   with reservepagegap = 16
```

指定 **datarows** 锁方案，并将 **reservepagegap** 值设置为 16，这样扩展 I/O 操作将为每 15 个填充页保留 1 个空白页。

```
9. create table sales_south
   (stor_id         char(4)      not null,
   ord_num          varchar(20)  not null,
   date             datetime     not null,
   unique clustered (stor_id asc, ord_num desc))
```

创建唯一集群索引所支持的约束； *stor_id* 的索引顺序为升序，*ord_num* 的索引顺序为降序。

```
10.create table t1
   (a  int,
    b  char(10))
   at "SERVER_A.db1.joe.t1"
```

在远程服务器 SERVER_A 上创建名为 *t1* 的表，并创建名为 *t1* 的代理表，该代理表映射到远程表上。

```
11.create table employees
   (name varchar(30),
   home_addr Address,
   mailing_addr Address2Line)
```

创建名为 *employees* 的表。 *name* 的类型为 *varchar*， *home_addr* 是类型为 **Address** 的 Java-SQL 列，而 *mailing_addr* 则是类型为 **Address2Line** 的 Java-SQL 列。**Address** 和 **Address2Line** 都是安装在数据库中的 Java 类。

注释

- **create table** 创建一个表和可选的完整性约束。除非您在 **create table** 语句中另外指定了数据库，此表将在当前打开的数据库中创建。您可在其它数据库中创建表或索引，条件是您在 *sysusers* 表中列出，并且具有数据库的 **create table** 权限。
- 每次为表和索引分配空间时，将按一个扩展 （即 8 页）的增量来分配。每当一个扩展被填满后，将分配另一个扩展。要查看为表分配的空间量和表所使用的空间量，可使用 **sp_spaceused**。
- 当从组件集成服务中使用 **create table** 命令时，如果有一列定义为 *char*(*n*) NULL，组件集成服务就会将列创建为远程服务器上的 *varchar*(*n*) 列。

限制

- 每个数据库中最多可以有 20 亿个表，每个表可以有 250 个用户定义的列。每个表的行数仅受可用存储空间的限制。
- 每行的最大字节数取决于表的锁方案。在一个 allpage 锁定表中，用户数据的最大字节数是 1960 字节。对于 DOL 锁定表，每个可变长度列或允许空值的列将扣除 2 个字节。
- 在创建表时，如果 *varchar*、 *nvarchar* 或 *varbinary* 列的总定义宽度超过了允许的最大行大小，则会出现警告消息，但表仍会创建。如果向这种行中插入的字节数超过了最大值，或在执行 **update** 命令时使总的行大小超过了最大长度， Adaptive Server 将生成错误消息，而且命令将失败。

➤ ***注意***

当一个 **if...else** 块或 **while** 循环内出现 **create table** 命令时， Adaptive Server 将在确定条件是否为真之前创建该表的模式。如果该表已经存在，就可能会导致错误。确保数据库中没有同名的表。

列定义

- 当您从用户定义的数据类型创建一个列时：
  - 不能更改长度、精度或小数位数。
  - 可以使用 NULL 类型来创建一个 NOT NULL 列，但不能创建 IDENTITY 列。
  - 可以使用 NOT NULL 类型来创建 NULL 列或 IDENTITY 列。
  - 可以使用 IDENTITY 类型来创建 NOT NULL 列，但该列将继承 IDENTITY 属性。不能用 IDENTITY 类型来创建 NULL 列。
- 只有具有可变长度数据类型的列才能存储空值。当您创建具有固定长度数据类型的 NULL 列时， Adaptive Server 会自动将其转换为相应的可变长度数据类型。Adaptive Server 不会向用户告知数据类型的更改。

  下表列出了固定长度数据类型及其转换到的可变长度数据类型。某些可变长度数据类型 （如 *moneyn*）是保留的类型，它们不能用来创建列、变量或参数：

表 6-15：用于存储空值的可变长度数据类型

原始的固定长度数据类型	转换为
char	*varchar*
nchar	*nvarchar*
binary	*varbinary*
datetime	*datetimn*
float	*floatn*
int、 *smallint* 和 *tinyint*	*intn*
decimal	*decimaln*
numeric	*numericn*
money 和 *smallmoney*	*moneyn*

- 可通过两种方法创建列缺省值：在 **create table** 或 **alter table** 语句中将缺省值声明为列约束，或者使用 **create default** 语句创建缺省值并使用 **sp_bindefault** 将缺省值绑定到列。
- 要获得有关表及其列的报告，可执行系统过程 **sp_help**。

临时表

- 临时表存储在临时数据库 *tempdb* 中。
- 临时表名的前 13 个字符在每个会话中必须是唯一的。这样的表仅可由当前的 Adaptive Server 会话访问。它们按照名称及系统提供的数字后缀存储在 *tempdb..objects* 中，并且在当前会话结束时消失，或在显式删除后消失。
- 用 “tempdb..” 前缀创建的临时表可以在 Adaptive Server 用户会话中共享。除非被所有者删除或 Adaptive Server 重新启动，否则它们将一直存在。只有当您要在用户和会话间共享表时，才应使用 “tempdb..” 前缀在存储过程内创建临时表。为了避免对临时表的无意共享，可在存储过程内创建或删除临时表时使用 # 前缀。
- Adaptive Server 会话期间，临时表可在多个用户之间共享。但是，通常无法识别特定的用户会话，因为临时表是用 “guest” 用户 ID 2 来创建的。如果一个以上的用户运行了创建临时表的进程，每个用户都是 “guest” 用户，其 *uid* 值就都是相同的。因此，无从知道临时表中的哪个用户会话属于哪个特定用户。 SA 有可能使用 **sp_addlogin** 将用户添加到临时表中，在这种情况下，每个用户会话在临时表中都有各自的 *uid* ；但这种情况不太可能出现。
- 您可以将规则、缺省值和索引与临时表相关联，但是不能在临时表上创建视图，或使触发器与其相关联。
- 创建临时表时，只有在用户定义的数据类型已列在 *tempdb..systypes* 的情况下，才可使用该数据类型。如果要仅为当前会话将用户定义数据类型添加到 *tempdb* 中，则应在使用 *tempdb* 时执行 **sp_addtype**。要永久性地添加数据类型，可在使用 *model* 数据库时执行 **sp_addtype**，然后重新启动 Adaptive Server，以将 *model* 复制到 *tempdb* 中。

使用索引

- 表将 “跟随” 它的集群索引。如果在一个段中创建表，然后在另一个段上创建其集群索引，该表就会迁移到创建索引的段上。
- 通过在一个段上创建表，在另一个段上 （这两个段处于不同的物理设备上）创建非集群索引，可以加快插入、更新和选择的速度。有关详细信息，参见 *Performance and Tuning Guide*。

重命名表或其列

- 使用 **sp_rename** 来重命名表或列。
- 在重命名表或其列后，可使用 **sp_depends** 来确定哪些过程、触发器和视图依赖于该表，并重定义这些对象。

◆ ***警告！***

如果不重新定义这些依赖对象，那么当 Adaptive Server 重新编译这些对象后，它们将不再有效。

指定索引中的升序或降序

- 在索引列名后使用 **asc** 和 **desc** 关键字可为索引指定排序顺序。如果在创建索引时使列的顺序与查询的 **order by** 子句所指定的顺序相同，就可以消除查询处理过程中的排序步骤。

定义完整性约束

- **create table** 语句有助于通过一系列 SQL 标准所定义的完整性约束来控制数据库的完整性。这些完整性约束子句会限制用户可插入到表中的数据。您还可使用缺省值、规则、索引和触发器来强制实现数据库的完整性。

  完整性约束的优点表现在：在创建表的过程中一步完成完整性控件的定义，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，并且综合性也较低。

- 在多个列上运行的约束必须声明为表级约束；只在一列上运行的约束必须声明为列级约束。这种区别是语法上的区别：列级约束要放在列名和数据类型之后，分隔逗号之前 （参见示例 5）。而表级约束应作为以逗号分隔的各个子句来输入 （参见示例 4）。Adaptive Server 以相同的方式处理表级和列级约束，两种约束的效率相同。

- 您可以在表级或列级创建下列类型的约束：

  - **unique** 约束要求一个表中的任何两行在指定列都不能有相同的值。此外， **primary key** 约束要求列中无空值。
  - 参照完整性 **(references)** 约束要求特定列中插入或更新的数据在指定表和列中有与之相匹配的数据。
  - **check** 约束限制插入到列中的数据值。

  您还可以通过以下方法来强制数据的完整性：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）。

- 可以使用系统过程 **sp_primarykey**、**sp_foreignkey** 和 **sp_commonkey** 来保存系统表中的信息，这些信息有助于理清数据库中表的关系。这些系统过程不强制键关系，也不重复 **create table** 语句中 **primary key** 和 **foreign key** 关键字的功能。如需有关已经定义的键的报告，可使用 **sp_helpkey**。如需有关常用连接的报告，可执行 **sp_helpjoins**。
- Transact-SQL 提供了一些强制完整性的机制。除了可作为 **create table** 的一部分声明的约束之外，您还可以创建规则、缺省值、索引和触发器。下表总结了完整性约束，并说明了强制完整性的其它方法：

表 6-16：强制完整性的方法

在 *create table* 中	**其它方法**
unique 约束	**create unique index** （对允许空值的列）
primary key 约束	**create unique index** （对不允许空值的列）
references 约束	**create trigger**
check 约束 （表级）	**create trigger**
check 约束 （列级）	**create trigger** 或 **create rule** 以及 **sp_bindrule**
default 子句	**create default** 和 **sp_bindefault**

选择哪种方法取决于您的要求。例如，触发器为参照完整性 （例如对其它列或对象的引用）提供的处理方式比在 **create table** 中声明的方法更为复杂。此外，在 **create table** 语句中定义的约束是针对于该表的；与规则和缺省值不同，您不可以将它们绑定到其它表，并且只可使用 **alter table** 来对它们进行删除或更改。即使在同一表上，约束也不能包含子查询或集合函数。

- **create table** 命令可包含许多约束，但有以下限制：
  - **unique** 约束的数目受限于表可具有的索引数。
  - 一个表只能有一个 **primary key** 约束。
  - 表中的每一列只能有一个 **default** 子句，但可以对同一列定义不同的约束。

  例如：

```
create table discount_titles
(title_id   varchar(6)   default "PS7777" not null
        unique clustered
        references titles(title_id)
        check (title_id like "PS%"),
             new_price   money)
```

用每个完整性约束定义新表 *discount_titles* 的列 *title_id*。

- 您可以创建错误消息，并将它们绑定到参照完整性和 **check** 约束。用 **sp_addmessage** 创建消息，并将它们用 **sp_bindmsg** 绑定到约束。有关详细信息，参见 **sp_addmessage** 和 **sp_bindmsg**。
- Adaptive Server 在强制参照约束前先对检查约束求值，并在强制了所有的完整性约束后对触发器进行求值。如果任何约束失败，Adaptive Server 将取消数据修改语句；任何相关的触发器都不会执行。但是，违反约束**并不会**回退当前的事务。
- 在被引用表中，不能更新与引用表中的值匹配的列值或删除与引用表中的值匹配的行。应先从引用表中执行更新或删除，然后再尝试从被引用表中执行更新或删除。
- 在删除被引用表之前，必须先删除引用表；否则将违反约束。
- 如需为表定义的约束的详细信息，可使用 **sp_helpconstraint**。

唯一约束和主键约束

- 可以在表级或列级声明 **unique** 约束。**unique** 约束要求指定列中的所有值都是唯一的。表中的任何两行在指定列中都不能具有相同的值。
- **primary key** 约束是 **unique** 约束的一种限制性更强的形式。带有 **primary key** 约束的列不能包含空值。

➤ ***注意***

create table 语句的 unique 和 primary key 约束将创建索引，以定义列的唯一特性或主键特性。sp_primarykey、sp_foreignkey 和 sp_commonkey 定义列间的逻辑关系。这些关系必须使用索引和触发器来强制执行。

- 表级 **unique** 或 **primary** 键约束在 **create table** 语句中显示为独立的项目，它们必须包括所创建表中的一个或多个列的名称。
- **unique** 或 **primary key** 约束在指定列上创建唯一索引。示例 3 中的 **unique** 约束创建唯一集群索引，如同以下语句：

```
create unique clustered index salesind
    on sales (stor_id, ord_num)
```

唯一的区别是索引名称，您通过命名约束可将其设置为 *salesind*。

- **unique** 约束在 SQL 标准的定义规定，列定义不能允许空值。缺省情况下，如果在列定义中省略了 **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。

- 缺省情况下， **unique** 约束创建唯一非集群索引； **primary key** 约束创建唯一集群索引。一个表中只能有一个集群索引，所以只能指定一个 **unique clustered** 或 **primary key clustered** 约束。
- **create table** 的 **unique** 和 **primary key** 约束为 **create index** 语句提供了一种简化的替代方式。但是，它们具有以下限制：
  - 不能创建非唯一的索引。
  - 不能使用 **create index** 提供的所有选项。
  - 必须使用 **alter table drop constraint** 来删除这些索引。

参照完整性约束

- 参照完整性约束要求，插入到 **referencing** 表中用来定义约束的数据必须在 **referenced** 表中有与之匹配的值。要满足参照完整性约束，必须符合下列条件之一：
  - 引用表中受约束列的数据包含空值。
  - 引用表中受约束列的数据与被引用表中相应列中的数据值相匹配。

  以 *pubs2* 数据库为例，插入到 *salesdetail* 表中的行 （用于记录书籍销售情况）必须在 *titles* 表中具有有效的 *title_id*。 *salesdetail* 是引用表，而 *titles* 表则是被引用表。目前， *pubs2* 使用触发器来强制此参照完整性。但是， *salesdetail* 表可以包含此列定义和参照完整性约束来完成相同的任务：

  ```
  title_id tid
      references titles(title_id)
  ```
- 一个查询允许的最大表引用数为 192。使用系统过程 **sp_helpconstraint** 可检查表的参照性约束。
- 表可以包含对自身的参照完整性约束。例如， *pubs3* 中的 *store_employees* 表列出了雇员及其经理，该表在 *emp_id* 和 *mgr_id* 列之间具有以下自身引用：

  ```
  emp_id id primary key,
  mgr_id id null
         references store_employees(emp_id),
  ```

  此约束确保所有经理同时是雇员，并且为所有雇员分配了有效的经理。
- 只有在删除引用表或删除参照完整性约束之后，才能删除被引用的表 （除非它仅包括对自身的参照完整性约束）。
- Adaptive Server 不对临时表强制参照完整性约束。
- 要创建引用其它用户表的表，必须具有被引用表的 **references** 权限。有关分配 **references** 权限的信息，参见 **grant** 命令。

- 表级参照完整性约束在 **create table** 语句中作为独立的项目出现。它们必须包含 **foreign key** 子句，以及一个或多个列名的列表。

  只有在通过 **primary key** 约束将被引用表中的列指定为主键时，**references** 子句中的列名才是可选的。

  被引用的列必须受该被引用表中唯一索引的约束。您可以使用 **unique** 约束或 **create index** 语句来创建该唯一索引。

- 引用表列的数据类型必须与被引用表列的数据类型相匹配。例如，引用表 (*test_type*) 中 *col1* 的数据类型与被引用表 (*publishers*) 中 *pub_id* 的数据类型相匹配：

  ```
  create table test_type
  (col1 char(4) not null
      references publishers(pub_id),
  col2 varchar(20) not null)
  ```

- 当定义参照完整性约束时，被引用表必须已经存在。对于相互引用的表，可使用 **create schema** 语句来同时定义两个表。或者，也可创建一个无约束的表，然后在随后使用 **alter table** 添加约束。有关详细信息，参见 **create schema** 或 **alter table**。

- **create table** 参照完整性约束为强制数据完整性提供了一种简单的方法。与触发器不同，它们**不能**：

  - 通过数据库中的相关表中进行级联更改
  - 通过引用其它列或数据库对象施加复合限制
  - 执行 “what-if” 分析

  当数据修改违反约束时，参照完整性约束将不回退事务。利用触发器，您可以根据处理参照完整性的方式来选择是回退事务还是继续事务。

➤ ***注意***

Adaptive Server 在检查触发器之前将检查参照完整性约束，使违反约束的数据修改语句不会引发触发器。

使用跨数据库的参照完整性约束

- 在创建跨数据库约束时， Adaptive Server 将把以下信息存储到每个数据库的 *sysreferences* 系统表中：

表 6-17：为参照完整性约束存储的信息

存储在 *sysreferences* 中的信息	包含被引用表信息的列	包含引用表信息的列
键列 ID	*refkey1* 至 *refkey16*	*fokey1* 至 *fokey16*
表 ID	*reftabid*	*tableid*
数据库 ID	*pmrydbid*	*frgndbid*
数据库名	*pmrydbname*	*frgndbname*

- 您可以删除引用表或其数据库，而不会出现任何问题。 Adaptive Server 会自动从被引用数据库中删除外键信息。
- 因为引用表依赖被引用表中的信息，所以 Adaptive Server 不允许：
  - 删除被引用表，
  - 删除包含被引用表的外部数据库，或
  - 用 **sp_renamedb** 重命名任一数据库。

  必须用 **alter table** 删除跨数据库约束，才能执行这些操作。
- 每次添加或删除跨数据库约束时，或者是每次删除包含跨数据库约束的表时，都应转储**两个**受影响的数据库。

◆ ***警告！***

如果装载包含跨数据库约束的数据库早期转储，可能会使数据库遭到损坏。

- *sysreferences* 系统表存储外部数据库的**名称**和 ID 号。如果使用 **load database** 更改数据库名称或在其它服务器上装载该数据库，Adaptive Server 将无法确保参照完整性。

◆ ***警告！***

如果要用其它名称装载数据库或将其移至另一个 Adaptive Server 上，那么在转储数据库之前，应使用 alter table 删除所有外部参照完整性约束。

check 约束

- **check** 约束限制用户可在表列中插入的值。 **check** 约束指定 *search_condition*，所有非空值都必须通过此条件，才能插入到表中。 *search_condition* 可包括：
  - 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
  - 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 内部函数。 *search_condition* 不能包含子查询、集合函数、主变量或参数。 Adaptive Server 不对临时表强制执行 **check** 约束。
- 如果 **check** 约束是列级 **check** 约束，该约束只能引用它定义所在的列，而不能引用表中的其它列。表级 **check** 约束可以引用表中的任何一列。
- **create table** 允许列定义中有多个 **check** 约束。
- **check** 完整性约束为使用规则和触发器提供了另一种方法。它们针对于创建时所在的表，且不能绑定到其它表中的列或绑定到用户定义的数据类型。
- **check** 约束不替换列定义。如果对允许空值的列声明了 **check** 约束，即使 NULL 值并未包含在 *search_condition* 中，也可以在列中显式或隐式地插入 NULL 值。例如，如果在允许空值的表列中创建了 **check** 约束，它规定 “pub_id in (“1389”, “0736”, “0877”, “1622”, “1756”)”，或 “@amount > 10000”，您仍可在该列中插入空值。列定义将替换 **check** 约束。

IDENTITY 列

- 第一次向表中插入行时， Adaptive Server 将把值 1 分配给 IDENTITY 列。每个新行都将获得一个比上一个值大 1 的列值。该值的优先级高于任何用 **create table** 语句为列声明的缺省值，也高于任何用 **sp_bindefault** 系统过程绑定到该列的缺省值。可插入到 IDENTITY 列的最大值为 10 PRECISION - 1。
- 在 IDENTITY 列中插入值后，可以为列指定源值或恢复被误删的行。对基表使用 **set identity_insert** *table_name* **on** 之后，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。只要未在 IDENTITY 列上创建唯一索引， Adaptive Server 就不会验证值的唯一性。您可以插入任何正整数。
- 可以使用 **syb_identity** 关键字来引用 IDENTITY 列，并在必要时用表名代替实际列名来进行限定。

- 系统管理员可使用 **auto identity** 数据库选项使 10 位数的 IDENTITY 列自动包含在新表中。要在数据库中打开这一功能，可使用：

  ```
  sp_dboption database_name, "auto identity", "true"
  ```

  用户每次在数据库中创建表而不指定 **primary** 键、 **unique** 约束或 IDENTITY 列时， Adaptive Server 都会自动定义一个 IDENTITY 列。当您用 **select** * 语句检索列时，SYB_IDENTITY_COL 列是不可见的。必须在选择列表中显式地包含列名。

- 当服务器发生故障时，会使 IDENTITY 列的值出现间隔。间隔的最大大小取决于 **identity burning set factor** 配置参数的设置。造成间隔的原因还包括事务回退、删除行，或手工将数据插入 IDENTITY 列。

指定锁方案

- 要为表指定锁方案，可使用关键字 **lock** 和以下锁方案之一：
  - **allpages** 锁，它锁定受查询影响的数据页和索引
  - **datapages** 锁，它仅锁定数据页
  - **datarows** 锁，它仅锁定数据行

  如果没有指定锁方案，将使用服务器的缺省锁方案。全服务器范围的缺省值用配置参数 **lock scheme** 来设置。

- 可使用 **alter table** 命令更改表的锁方案。

空间管理属性

- 空间管理属性 **fillfactor**、 **max_rows_per_page**、 **exp_row_size** 和 **reservepagegap** 按以下方式管理表空间的使用：
  - **fillfactor** 在创建索引时在页上保留额外的空间，但并不一直保持 **fillfactor**。
  - **max_rows_per_page** 限制数据页或索引页的行数。由于减少行数可减少锁争用，因此该属性的主要用途是提高 allpage 锁定表中的并发性。如果指定一个 **max_rows_per_page** 值并指定 **datapages** 或 **datarows** 锁，将显示一条警告消息。表将被创建，该值将存储在 *sysindexes* 中，但只有在随后将锁方案更改为 **allpages** 时，才会应用该值。

  - **exp_row_size** 指定数据行的期望大小。它只应用于数据行，而不应用于索引，并且只应用于具有可变长度列的 DOL 锁定表。此选项用于减少 DOL 锁定表中转移的行数。它主要用于此类表：行在首次插入时含有空列或列较短，但在以后的更新中行大小增加。**exp_row_size** 保留数据页上的空间，以便于行增加到指定的大小。如果在创建 allpage 锁定表时指定 **exp_row_size**，则会输出警告消息。表将被创建，该值将被存储在 *sysindexes* 中，但只有在随后将锁方案改为 **datapages** 或 **datarows** 后，才会应用该值。
  - **reservepagegap** 指定空白页与填充页的比率，以应用于执行扩展分配的命令。它可应用于所有锁方案中的数据页和索引页。

- 表 6-18 显示了空间管理属性和锁方案的有效组合。如果 **create table** 命令包含不兼容的组合，则显示一条警告消息并创建该表。该值将被存储在系统表中，但不会应用。如果表锁定方案的更改使属性生效，则应用这些属性。

表 6-18：空间管理属性和锁方案

属性	allpages	datapages	datarows
max_rows_per_page	是	否	否
exp_row_size	否	是	是
reservepagegap	是	是	是
fillfactor	是	是	是

- 表 6-19 显示了空间管理属性的缺省值以及使用缺省值的效果。

表 6-19：空间管理属性的缺省值和效果

属性	缺省值	使用缺省值的效果
max_rows_per_page	0	在页上尽可能装入更多的行，最多可达 255 行
exp_row_size	0	使用全服务器范围的缺省值，该缺省值由配置参数 **default exp_row_size percent** 设置
reservepagegap	0	扩展分配期间不保留空白页
fillfactor	0	完全填充叶页，在索引页上保留空间

使用 *exp_row_size*

- 如果应用程序将短行插入 DOL 锁定表中，并且这些行在以后的更新中长度增加，则使用 **exp_row_size** 来减少 DOL 锁定表的行被转移到新位置的次数。

使用 *reservepagegap*

- 使用大量空间的命令通过分配扩展而不是通过分配单个页来分配新的空间。**reservepagegap** 关键字使这些命令保留空白页，这样可使随后的页分配在要被拆分的页或要转移行的页附近进行。表 6-20 显示了何时将应用 **reservepagegap**。

表 6-20：何时应用 reservepagegap

命令	应用于数据页	应用于索引页
快速 **bcp**	是	如果索引存在，则不使用快速 **bcp**。
慢速 **bcp**	仅用于堆表，而不用于具有集群索引的表	不执行扩展分配
select into	是	目标表上不存在索引
create index 或 **alter table...constraint**	是，用于集群索引	是
reorg rebuild	是	是
alter table...lock （用于从 allpage 锁更改为 DOL 锁，或相反）	是	是

- 表的 **reservepagegap** 值将存储在 *sysindexes* 中，在表上执行以上任一种操作时都将应用该值。要更改此已存储的值，可使用系统过程 **sp_chgattribute**。
- **reservepagegap** 不应用于工作表或工作表上的排序。

使用 *at*

- 用 **at** 关键字提供的位置信息与用 **sp_addobjectdef** 系统过程提供的信息相同。这些信息将存储在 *sysattributes* 表中。

Java-SQL 列

- 如果数据库中已启用 Java，就可以创建带有 Java-SQL 列的表。有关详细信息，参见 *Adaptive Server Enterprise 中的 Java*。
- Java-SQL 列的声明类 (*datatype*) 必须实现 **Serializable** 或 **Externalizable** 接口。
- 在创建表时，不能如此指定 Java-SQL 列：
  - 指定为外键
  - 在引用子句中指定
  - 指定为具有 UNIQUE 属性
  - 指定为主键
- 如果指定了 **in row**，存储的值就不能超过 255 字节。
- 如果指定了 **off row**，那么：
  - 不能在检查约束中引用该列。
  - 不能在指定 **distinct** 的 **select** 中引用该列。
  - 不能在比较运算符、谓词或 **group by** 子句中指定该列。

获取有关表的信息

- **sp_help** 显示有关表的信息，它可列出分配给指定表及其索引的所有特性 （如缓存绑定），提供特性的类、名称、整数值、字符值以及备注。
- **sp_depends** 显示数据库中依赖于表的视图、触发器和过程的信息。
- **sp_helpindex** 报告在表上创建的索引的信息。

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	以下是 Transact-SQL 扩展： • 使用数据库名来限定表名或列名 • IDENTITY 列 • **not null** 列缺省值 • **asc** 和 **desc** 选项 • **reservepagegap** 选项 • **lock** 子句 • **on** *segment_name* 子句 有关数据类型一致性的信息，参见“系统和用户定义的数据类型”。

权限

缺省情况下，数据库所有者拥有 **create table** 权限，他可将此权限移交给其他用户。所有用户都可以创建临时表。

参见

命令	**alter table**、 **create existing table**、 **create index**、 **create rule**、 **create schema**、 **create view**、 **drop index**、 **drop rule**、 **drop table**
系统过程	**sp_addmessage**、 **sp_addsegment**、 **sp_addtype**、 **sp_bindmsg**、 **sp_chgattribute**、 **sp_commonkey**、 **sp_depends**、 **sp_foreignkey**、 **sp_help**、 **sp_helpjoins**、 **sp_helpsegment**、 **sp_primarykey**、 **sp_rename**、 **sp_spaceused**

create trigger

功能

创建触发器，即一种经常用来强制完整性约束的存储过程。当用户试图在指定表上使用指定的数据修改语句时，触发器就会自动执行。

语法

```
create trigger [owner.]trigger_name
   on [owner.]table_name
   for {insert , update , delete}
   as SQL_statements
```

或者，使用 **if update** 子句：

```
create trigger [owner.]trigger_name
   on [owner.]table_name
   for {insert , update}
   as
       [if update (column_name)
           [{and | or} update (column_name)]...]
           SQL_statements
       [if update (column_name)
           [{and | or} update (column_name)]...
           SQL_statements]...
```

关键字和选项

trigger_name — 是触发器的名称。它必须符合标识符规则，并且在数据库中必须唯一。指定所有者的名称，以创建由当前数据库中其它用户所拥有的另一同名触发器。 *owner* 的缺省值是当前用户。如果使用所有者名称来限定触发器，则必须以相同的方式来显式限定表名。

触发器名不可使用变量。

table_name — 是创建触发器时所在表的名称。如果数据库中有多个同名表，则应指定所有者的名称。 *owner* 的缺省值是当前用户。

insert、 **update**、 **delete** — 可进行任意组合。 **delete** 不能用于 **if update** 子句。

SQL_statements — 指定触发器条件和触发器动作。触发器条件确定所尝试的 **insert**、 **update** 或 **delete** 操作是否会导致触发器动作的执行。 SQL 语句通常包含一个子查询，子查询之前带有关键字 **if**。在下面的示例 2 中，关键字 **if** 之后的子查询就是触发器条件。

当用户尝试执行 **insert**、 **update** 或 **delete** 操作时，触发器动作将生效。如果指定了多个触发器动作，它们将用 **begin** 和 **end** 来分组。

如需触发器定义中不允许的语句列表，参见 “触发器和事务”。有关可包含在触发器定义中的 *deleted* 和 *inserted* 逻辑表的信息，参见 “deleted 和 inserted 逻辑表”。

if update — 用于测试指定的列是否包含在 **update** 语句的 **set** 列表中，或者它是否受 **insert** 操作的影响。这样，指定的触发器动作就可以同指定列的更新相关联 （参见示例 3）。可以指定一个以上的列，并且可以在一个 **create trigger** 语句中指定多个 **if update** 语句（参见示例 5）。

示例

1.
```
create trigger reminder
on titles
for insert, update as
print "Don’t forget to print a report for
accounting."
```

当有人试图在 *titles* 表中添加数据或更改数据时，输出一条消息。

2.
```
create trigger t1
on titleauthor
for insert as
if (select count(*)
     from titles, inserted
     where titles.title_id = inserted.title_id) = 0
begin
print "Please put the book’s title_id in the
        titles table first."
rollback transaction
end
```

如果在 *titles* 表中没有相应的 *title_id*，则禁止在 *titleauthor* 中插入新行。

3.
```
create trigger t2
on publishers
for update as
if update (pub_id) and @@rowcount = 1
begin
    update titles
    set titles.pub_id = inserted.pub_id
    from titles, deleted, inserted
    where deleted.pub_id = titles.pub_id
end
```

   如果 *publishers* 表的 *pub_id* 列已更改，则在 *titles* 表中进行相应的更改。

4.
```
create trigger t3
on titleauthor
for delete as
begin
    delete titles
    from titles, deleted
    where deleted.title_id = titles.title_id
    delete titleauthor
    from titleauthor, deleted
    where deleted.title_id = titleauthor.title_id
    print "All references to this title have been
    deleted from titles and titleauthor."
end
```

   如果从 *titleauthor* 中删除了任何行，则从 *titles* 表中删除标题。如果该书是由多名作者撰写的，还将删除它在 *titleauthor* 中的其它引用。

5.
```
create trigger stopupdatetrig
on titles
for update
as
if update (title_id)
  and datename(dw, getdate())
  in ("Saturday", "Sunday")
  begin
    rollback transaction
    print "We don't allow changes to"
    print "primary keys on the weekend!"
  end
```

```
if update (price) or update (advance)
  if (select count(*) from inserted
    where (inserted.price * inserted.total_sales)
    < inserted.advance) > 0
    begin
      rollback transaction
      print "We don't allow changes to price or"
      print "advance for a title until its total"
      print "revenue exceeds its latest advance."
    end
```

禁止在周末对主键进行更新。如果某个标题的总收入没有超过其预付金额，则禁止对该标题价格或预付款的更新。

注释

- 每个数据修改语句只引发一次触发器。包含 **while** 循环的复杂查询可以多次重复 **update** 或 **insert**，每次都会引发触发器。

触发器和参照完整性

- 触发器通常用于强制执行**参照完整性**（表或视图的主键和外键间关系的完整性规则），提供级联删除，以及提供级联更新（分别参见示例 2、3 和 4）。
- 只有在数据修改语句已完成，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，触发器才引发。触发器和引发它的语句被当作单个事务，可从触发器中回退。如果检测到服务器错误，整个事务都将回退。
- 您还可以用 **create table** 语句所定义的约束来强制参照完整性，这是使用 **create trigger** 的替代方法。有关完整性约束的信息，参见 **create table** 和 **alter table**。

deleted 和 *inserted* 逻辑表

- *deleted* 和 *inserted* 都是逻辑（概念上的）表。这些表的结构类似于已定义触发器的表（即对其尝试用户操作的表），它们保留将被用户操作更改的行的旧值或新值。
- *deleted* 和 *inserted* 表可以用触发器来进行检查，以确定是否或如何执行触发器动作，但表本身不能用触发器动作来进行变更。
- *deleted* 表与 **delete** 和 **update** 一起使用；*inserted* 表则与 **insert** 和 **update** 一起使用。（更新是先删除再插入的操作：它首先影响 *deleted* 表，然后影响 *inserted* 表。）

触发器限制

- 只能在当前数据库中创建触发器。如果使用所有者名称来限定触发器，则必须以相同的方式来显式限定表名。触发器可以引用当前数据库之外的对象。
- 一个触发器不能应用于多个表。但是，可以在同一个 **create trigger** 语句中为多项用户操作 （如 **insert** 和 **update**）定义相同的触发器动作。一个表最多可以有 3 个触发器：**insert**、 **update** 和 **delete** 各一个。
- 对于相同的操作 （**insert**、 **update** 或 **delete**），表中或列中的每个新触发器将覆盖先前的触发器。覆盖之前不会发出警告。
- 不能在临时表上创建触发器。
- 不能在视图上创建触发器。
- 不能在系统表上创建触发器。
- 如果触发器从已插入或已删除表中的 *text* 或 *image* 列进行选择，则不能使用这样的触发器。
- 建议不要在触发器中包含向用户返回结果的 **select** 语句，因为对这些返回结果的特殊处理必须写入每个允许修改触发器表的应用程序中。
- 如果触发器引用的表名、列名或视图名是无效的标识符，则必须在执行 **create trigger** 命令前设置 **set quoted_identifier on**，并将每个这样的名称用双引号引起来。触发器引发时， **quoted_identifier** 选项**不**需要设置为 “on”。

获取有关触发器的信息

- 触发器的执行计划存储在 *sysprocedures* 中。
- 每个触发器都分配有一个标识号，它作为新行存储在 *sysobjects* 中。同时，在 *deltrig* 列中有它所应用的表的对象 ID，并且还在它所应用的表的 *sysobjects* 行的 *deltrig*、 *instrig* 和 *updtrig* 列中有相应的条目。
- 要显示存储在 *syscomments* 中的触发器文本，可使用 **sp_helptext**。

  如果系统安全员已经用 **sp_configure** 系统过程将 **allow select on syscomments.text column** 参数复位 （这是在评估后的配置中运行 Adaptive Server 所必需的），那么您必须是触发器的创建者或系统管理员，才能通过 **sp_helptext** 来查看触发器的文本。
- 要获取有关触发器的报告，可使用 **sp_help**。
- 要获取有关触发器所引用表和视图的报告，可使用 **sp_depends**。

触发器和性能

- 就性能而言，触发器的开销通常是很小的。运行触发器所需的时间大多花费在引用内存或数据库设备中的其它表。
- *deleted* 和 *inserted* 表是逻辑表，所以经常被总处于内存中的触发器（而不是在数据库设备上的触发器）所引用。触发器所引用的其它表的位置将决定操作所需的时间。

在触发器内设置参数

- 可以在触发器内使用 **set** 命令。在触发器执行期间，您所调用的 **set** 选项将一直有效，然后会还原到它的先前设置。尤其是，**self_recursion** 选项可以在触发器内使用，这样，触发器自身所进行的数据修改可以使触发器再次引发。

删除触发器

- 如果重命名了触发器所引用的任何对象，则必须删除该触发器，然后将其重新创建。您可用 **sp_rename** 来重命名触发器。
- 在删除表时，与之相关的任何触发器也将被删除。

不会引发触发器的操作

- **truncate table** 命令不会被 **delete** 触发器捕获。虽然 **truncate table** 语句实际上类似于不带 **where** 子句的 **delete**（它删除所有行），但由于不记录对数据行的更改，因此不会引发触发器。

  由于 **truncate table** 命令的权限在缺省情况下属于表所有者，并且不可移交，所以只有表所有者才需要考虑是否会在无意中用 **truncate table** 语句绕过 **delete** 触发器。
- **writetext** 命令无论是记录还是不记录，都不会引发触发器。

触发器和事务

- 当定义触发器后，它指定的、对所应用的表执行的操作以及触发器本身始终是事务的隐含部分。触发器通常用来在检测到错误时回退整个事务，也可用来回退特定数据修改的效果。
  - 当触发器包含 **rollback transaction** 命令时，回退操作将中止整个批处理，批处理中的所有后继语句都不会被执行。
  - 当触发器包含 **rollback trigger** 时，回退操作只影响引发触发器的数据修改。**rollback trigger** 命令可包含 **raiserror** 语句。批处理中随后的语句将被执行。
- 由于触发器作为事务的一部分来执行，触发器中不允许使用以下语句和系统过程：

  - 所有 **create** 命令，包括 **create database**、 **create table**、 **create index**、 **create procedure**、**create default**、**create rule**、**create trigger** 以及 **create view**
  - 所有 **drop** 命令
  - **alter table** 和 **alter database**
  - **truncate table**
  - **grant** 和 **revoke**
  - **update statistics**
  - **sp_configure**
  - **load database** 和 **load transaction**
  - **disk init**、**disk mirror**、**disk refit**、**disk reinit**、**disk remirror**、**disk unmirror**
  - **select into**
- 如果所需结果 （如摘要值）取决于数据修改所影响的行数，可使用 *@@rowcount* 来测试多行修改 （基于 **select** 语句的 **insert**、 **delete** 或 **update**），然后采取适当的措施。所有不返回行的 Transact-SQL 语句 （如 **if** 语句）都将 *@@rowcount* 设置为 0，因此 *@@rowcount* 测试应该在触发器的初始阶段进行。

更新和插入触发器

- 在 **insert** 或 **update** 命令执行时， Adaptive Server 会同时在触发器表和 *inserted* 表中添加行。 *inserted* 表中的行始终是触发器表中的一行或多行的副本。
- **update** 或 **insert** 触发器可使用 **if update** 命令来确定 **update** 或 **insert** 是否更改了特定列。只要为列分配了选择列表中的值或 **values** 子句中的值， **if update**(*column_name*) 对于 **insert** 语句就为真。显式的 NULL 或缺省值为列赋值，从而激活触发器。但隐式的 NULL 则不激活触发器。

  例如，如果您创建下面的表和触发器：

```
create table junk
(aaa int null,
bbb int not null)

create trigger trigtest on junk
for insert as
if update (aaa)
    print "aaa updated"
if update (bbb)
    print "bbb updated"
```

在任一列或两个列中插入值时，将同时引发 *aaa* 列和 *bbb* 列的触发器：

```
insert junk (aaa, bbb)
values (1, 2)
```

```
aaa updated
bbb updated
```

在 *aaa* 列中插入显式的空值时，也将引发触发器：

```
insert junk
values (NULL, 2)
```

```
aaa updated
bbb updated
```

如果列 *aaa* 具有缺省值，触发器也会引发。

不过，如果列 *aaa* 没有缺省值并且没有显式插入值， Adaptive Server 将生成隐式的 NULL，而触发器不会引发：

```
insert junk (bbb)
values(2)
```

```
bbb updated
```

if update 从不会对 **delete** 语句为真。

嵌套触发器和触发器递归

- 缺省情况下， Adaptive Server 允许使用嵌套的触发器。要禁止触发器嵌套，可使用 **sp_configure** 将 **allow nested triggers** 选项设置为 0 （即关闭），如下所示：

  ```
  sp_configure "allow nested triggers", 0
  ```

- 触发器可嵌套 16 层。如果触发器更改了包含另一个触发器的表，第二个触发器将引发，随即便可以调用第三个触发器，依此类推。如果链中的任何触发器引发了无限循环，则将超出嵌套层数，触发器将中止，同时回退包含该触发器查询的事务。

➤ ***注意***

由于触发器放置在事务中，如果在一组嵌套触发器的任一层出现故障，都将取消整个事务：所有数据修改都将回退。为触发器提供消息以及其它错误处理和调试帮助，从而确定故障的起因。

- 全局变量 *@@nestlevel* 包含当前执行的嵌套层次。存储过程或触发器每次调用其它存储过程或触发器时，都会增加嵌套层次。如果超过了最大值 16，事务将中止。
- 如果触发器调用的存储过程所执行的操作会使触发器再次引发，那么在只有启用嵌套触发器后，触发器才会被再次激活。除非触发器内有限制递归数的条件，否则将导致嵌套层次溢出。

  例如，如果更新触发器调用的存储过程将执行更新，那么在 **allow nested triggers** 处于关闭状态时，触发器和存储过程将执行一次。如果 **allow nested triggers** 处于打开状态，且更新次数不受触发器或过程中条件的限制，该过程或触发器循环就将继续进行，直至超过 16 层的最大嵌套值。
- 缺省情况下，无论 **allow nested triggers** 配置参数如何设置，触发器都不会因响应对触发器内同一表的第二次数据修改而调用其自身。作为在触发器内修改数据的结果， **set** 选项 **self_recursion** 将使触发器能够再次引发。例如，如果表中一列的更新触发器导致对另一列的更新，那么当 **self_recursion** 被禁用时，该更新触发器仅引发一次，但如果 **self_recursion** 被设置为打开状态，该触发器就将引发 16 次。要进行自递归，还必须启用 **allow nested triggers** 配置参数。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

只有系统安全员才能授予或撤消创建触发器的权限。

缺省情况下，将向用户授予发出 **create trigger** 命令的权限。当撤消用户创建触发器的权限时，会将撤消行增加到该用户的 *sysprotects* 表中。要向该用户授予发出 **create trigger** 命令的权限，必须发出两个 **grant** 命令。第一个命令从 *sysprotects* 中删除撤消行；第二个命令插入授权行。

如果撤消创建触发器的权限，用户就不能创建触发器，即便是在自己的表上也不行。撤消用户创建触发器的权限只影响从中发出 revoke 命令的数据库。

对象的权限：触发器创建时间

当您创建触发器时， Adaptive Server 不对触发器所引用的对象 （如表和视图）进行权限检查。因此，即使不具有访问其对象的权限，也可以成功创建触发器。当触发器引发时，将进行所有权限检查。

对象的权限：触发器执行时间

当触发器执行时，其对象的权限检查取决于触发器及其对象是否属于同一用户。

- 如果触发器及其对象不属于同一用户，已引发触发器的用户必须已被授予直接访问对象的权限。例如，如果触发器要从用户无法访问的表中进行选择，此触发器的执行将失败。同时，引发触发器的数据修改将被回退。
- 如果触发器及其对象属于同一用户，将应用特殊规则。用户将自动具有访问触发器的对象的隐式权限，即使该用户不能直接访问这些对象。有关隐式权限规则的详细讨论，参见*系统管理指南*。

参见

命令	**alter table**、**create procedure**、**create table**、**drop trigger**、 **rollback trigger**、 **set**
系统过程	**sp_commonkey**、 **sp_configure**、 **sp_depends**、**sp_foreignkey**、 **sp_help**、 **sp_helptext**、**sp_primarykey**、 **sp_rename**、 **sp_spaceused**

create view

功能

创建视图，这是查看一个或多个表中数据的替代方法。

语法

```
create view [owner.]view_name
   [(column_name [, column_name]...)]
   as select [distinct] select_statement
   [with check option]
```

关键字和选项

view_name — 是视图的名称。该名称不能包括数据库名。如果您已设置 **set quoted_identifier on**，就可以使用分隔的标识符。否则，视图名不能是变量，并且必须符合标识符规则。有关有效视图名的详细信息，参见第 3 章 “表达式、标识符和通配符”中的 “标识符”。指定所有者的名称，以创建其它用户在当前数据库中拥有的其它同名视图。 *owner* 的缺省值是当前用户。

column_name — 指定可用作视图中列标题的名称。如果您已经设置 **set quoted_identifier on**，就可以使用分隔的标识符。否则，列名必须符合标识符规则。有关有效列名的详细信息，参见第 3 章 “表达式、标识符和通配符”中的 “标识符”。

提供列名始终是合法的，但只有在下列情况下，列名才是必需的：

- 当列派生于算术表达式、函数、字符串并置或常量时
- 当两个或更多列具有相同的名称时 （通常是由于连接）
- 当您要为视图中的列指定的名称与其派生列的名称不同时 （参见示例 3）。

列名还可在 **select** 语句中进行分配 （参见示例 4）。如果未指定列名，视图列将获得与 **select** 语句中的列相同的名称。

select — 开始定义视图的 **select** 语句。

distinct — 指定视图不能包含重复的行。

select_statement — 完成定义视图的 **select** 语句。它可使用多个表和其它视图。

with check option — 表示所有数据修改语句都将对照视图选择标准来进行验证。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。

示例

1.
```
create view titles_view
as select title, type, price, pubdate
from titles
```

   创建从基表 *titles* 的 *title*、 *type*、 *price* 和 *pubdate* 列派生的视图。

2.
```
create view "new view" ("column 1", "column 2")
as select col1, col2 from "old view"
```

   从 “old view” 中创建 “new view” 视图。两列都在新视图重新命名。所有包含嵌入空白的视图名和列名都用双引号引起来。在创建视图之前，必须使用 **set quoted_identifier on**。

3.
```
create view accounts (title, advance, amt_due)
as select title, advance, price * total_sales
     from titles
where price > $5
```

   创建一个视图，它包含价格低于 $5.00 的书籍的标题、预付款和应付款。

4.
```
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
```

   创建派生于两个基表 *authors* 和 *publishers* 的视图。该视图包括其居住城市有出版者的作家的姓名及城市。

5.
```
create view cities2
as select authorname = au_lname,
acity = authors.city, publishername = pub_name,
pcity = publishers.city
from authors, publishers
where authors.city = publishers.city
```

   用示例 3 中的定义创建视图，但其列标题在 **select** 语句中提供。

6.
```
create view author_codes
as select distinct au_id
from titleauthor
```

   创建视图 *author_codes*，它派生于列出唯一作者标识代码的 *titleauthor*。

7.
```
create view price_list (price)
as select distinct price
from titles
```

   创建视图 *price_list*，它派生于列出唯一书籍价格的 *title*。

```
8. create view stores_cal
   as select * from stores
   where state = "CA"
   with check option
```

创建 *stores* 表的视图，该表不包括加州之外书店的信息。**with check option** 子句用视图的选择标准来验证每个已插入或已更新的行。*state* 值不是 “CA” 的行将被拒绝。

```
9. create view stores_cal30
   as select * from stores_cal
   where payterms = "Net 30"
```

创建视图 *stores_cal30*，它派生于 *stores_cal*。新视图从 *stores_cal* 中继承检查选项。所有通过 *stores_cal30* 插入或更新的行都必须具有 *state* 值 “CA”。由于 *stores_cal30* 没有 **with check option** 子句，因此可以通过其 *payterms* 值不是 “Net 30” 的 *stores_cal30* 来插入或更新行。

```
10.create view stores_cal30_check
   as select * from stores_cal
   where payterms = "Net 30"
   with check option
```

创建视图 *stores_cal30_check*，它派生于 *stores_cal*。新视图从 *stores_cal* 继承检查选项。它还具有自己的 **with check option** 子句。对于每个通过 *stores_cal30_check* 插入或更新的行，都对照 *stores_cal* 和 *stores_cal30_check* 的选择标准来进行验证。*state* 值不是 “CA” 的行或 *payterms* 值不是 “Net 30” 的行将被拒绝。

注释

- 通过授予视图 （而不是其基础表）的权限，可以将视图用作安全性机制。
- 可以用 **sp_rename** 来重命名视图。
- 当通过视图进行查询时， Adaptive Server 会检查并确保在语句中任何地方引用的所有数据库对象都存在，它们在语句的环境中是有效的，并且数据更新命令不会违反数据完整性规则。如果上述任何检查失败，都会生成错误消息。如果检查成功， **create view** 将把视图 “转换” 为对基础表的操作。
- 有关视图的详细信息，参见 *Transact-SQL User's Guide*。

对视图的限制

- 只能在当前数据库中创建视图。
- 视图所引用的列数不能超过 250 个。
- 不能在临时表上创建视图。

- 不能在视图上创建触发器或建立索引。
- 不能对视图的 *text* 或 *image* 列使用 **readtext** 或 **writetext**。
- 不能在定义视图的 **select** 语句中包含 **order by** 或 **compute** 子句、 **into** 关键字，或 **union** 运算符。
- **create view** 语句不能在单个批处理中与其它 SQL 语句结合使用。

◆ ***警告！***

当一个 if...else 块或 while 循环内出现 create view 命令时， Adaptive Server 将在确定条件是否为真之前创建该视图的模式。如果该视图已经存在，就可能导致错误。确保数据库中没有同名的视图。

视图分辨率

- 如果您通过添加或删除列变更了视图基础表的结构，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。星号速记符将在首次视图创建时得到解释和扩展。
- 如果视图依赖于已删除的表 （或视图），那么当有人试图使用该视图时， Adaptive Server 将生成错误消息。如果用相同的名称和模式创建新的表 （或视图），以替代删除的表 （或视图），视图将再次变为可用。
- 您无需重新定义依赖于某一视图的其它视图即可重新定义此视图，除非重定义过程使 Adaptive Server 不能转换相关视图。

通过视图修改数据

- **delete** 语句不得用于多表视图。
- 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 不能为基础表或视图中的 **not null** 列提供值。）
- 不能通过包含已计算列的视图来插入行。
- **insert** 语句不得用于通过 **distinct** 或 **with check option** 创建的连接视图。
- **update** 语句可以用于通过 **with check option** 创建的连接视图。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。
- 不能更新或插入到用 **distinct** 子句定义的视图中。

- 数据更新语句不能更改计算视图中的任何列，也不能更改包含集合的视图。

IDENTITY 列和视图

- 不能用 *column_name* = **identity**(*precision*) 语法将新的 IDENTITY 列添加到视图中。
- 要在 IDENTITY 列中插入显式值，表所有者、数据库所有者或系统管理员必须为该列的基表设置 **set identity_insert** *table_name* **on**，而不是通过要进行插入的视图来完成。

group by 子句和视图

- 当出于安全性考虑而创建视图时，应谨慎使用集合函数和 **group by** 子句。 Transact-SQL 扩展允许指定不出现在 **group by** 子句中的列。如果您指定了不在 **group by** 子句中的列， Adaptive Server 将为该列返回详细的数据行。例如，以下查询：

  ```
  select title_id, type, sum(total_sales)
  from titles
  group by type
  ```

  为每一类型 （18 行）返回一行，这比预期的多。而以下查询：

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

  为每种类型 （6 行）返回一行。

  如需有关 **group by** 的详细信息，参见 “group by 和 having 子句”。

distinct 子句和视图

- **distinct** 子句将视图定义为不包含重复行的数据库对象。如果某行的所有列值都与另一行的相同列值相匹配，则该行就定义为另一行的重复行。空值则被当作其它空值的重复。

  如果查询视图的列子集，可能会得到形式上的重复行。如果您选择了列子集，其中的一些列包含相同的值，那么结果看起来就象是包含了重复的行。但是，视图中的基础行仍是唯一的。 Adaptive Server 第一次访问视图时 （在它进行任何投影和选择之前），它将把 **distinct** 要求应用于该视图的定义，从而使视图中的所有行彼此区分。

作为集合函数或 **group by** 子句的一部分，可以在视图定义的 **select** 语句中多次指定 **distinct**，以消除重复的行。例如：

```
select distinct count(distinct title_id), price
from titles
```

- **distinct** 的范围只适用于该视图；它不包括任何从 **distinct** 视图中派生的新视图。

with check option 子句和视图

- 如果视图是用 **with check option** 创建的，通过该视图插入或更新的每一行都必须符合该视图的选择标准。
- 如果视图是用 **with check option** 创建的，所有从“基”视图派生的视图都必须满足其检查选项。通过派生视图插入或更新的每一行都必须可以通过基视图来查看。

获取有关视图的信息

- 要获取有关视图所依赖的表或视图的报告，以及有关依赖于视图的对象的报告，可执行系统过程 **sp_depends**。
- 要显示视图的文本（它存储在 *syscomments* 中），可通过以视图名作为参数来执行系统过程 **sp_helptext**。

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	在 **select** 列表中使用多个 **distinct** 关键字和使用“*column_heading* = *column_name*”都属于 Transact-SQL 扩展。

权限

缺省情况下，数据库所有者拥有 **create view** 权限，他可将此权限移交给其他用户。

对象的权限：视图创建时间

当您创建视图时， Adaptive Server 不对视图所引用的对象 （如表和视图）进行权限检查。因此，即使您不具有访问视图对象的权限，仍可以成功地创建视图。所有权限检查将在用户调用视图时进行。

对象的权限：视图执行时间

当调用视图时，对象的权限检查取决于视图和所有被引用的对象是否属于同一用户。

- 如果视图及其对象不属于同一用户，调用者必须已授予直接访问对象的权限。例如，如果视图要从调用者无法访问的表中执行选择，此选择语句将失败。
- 如果视图及其对象属于同一用户，则应用特殊规则。调用者会自动具有访问视图对象的隐式权限，即使调用者不能直接访问这些对象。无需向用户授予直接访问表的权限，即可为他们提供有限制的视图访问权限。这样，视图就可用作安全性机制。例如，视图的调用者可能只能访问表中的某些行和列。有关隐式权限规则的详细讨论，参见*系统管理指南*。

参见

命令	**create schema**、 **drop view**、 **update**
系统过程	**sp_depends**、**sp_help**、**sp_helptext**、**sp_rename**

dbcc

功能

数据库一致性检查程序 (**dbcc**) 检查数据库的逻辑和物理一致性，并提供统计、计划和修复功能。

语法

```
dbcc checkalloc [(database_name [, fix | nofix])]

dbcc checkcatalog [(database_name)]

dbcc checkdb [(database_name [, skip_ncindex])]

dbcc checkstorage [(database_name)]

dbcc checktable({table_name|table_id}[, skip_ncindex])

dbcc checkverify [(database_name)]

dbcc complete_xact (xid, {"commit" | "rollback"})

dbcc forget_xact (xid)

dbcc dbrepair (database_name, dropdb)

dbcc engine( {offline , [enginenum] | "online" })

dbcc fix_text ({table_name | table_id})

dbcc indexalloc ({table_name | table_id}, index_id
      [, {full | optimized | fast | null}
      [, fix | nofix]])

dbcc rebuild_text (table [, column
   [, text_page_number]])

dbcc reindex ({table_name | table_id})

dbcc tablealloc ({table_name | table_id}
      [, {full | optimized | fast | null}
      [, fix | nofix]])|

dbcc { traceon | traceoff } (flag [, flag ... ])

dbcc tune ( { ascinserts, {0 | 1 } , tablename |
              cleanup, {0 | 1 } |
              cpuaffinity, start_cpu {, on| off } |
              des_greedyalloc, dbid, object_name,
                  " { on|off }" |
              deviochar vdevno, "batch_size" |
              doneinproc { 0 | 1 } |
              maxwritedes, writes_per_batch } )
```

关键字和选项

checkalloc — 检查指定的数据库，以确保正确分配所有页并使用所有已分配的页。如果没有给定数据库名称，**checkalloc** 将检查当前的数据库。它总是使用 **optimized** 报告选项 （参见 **tablealloc**）。

checkalloc 报告已分配和使用的空间量。

database_name — 是要检查的数据库的名称。如果没有给定数据库名称，**dbcc** 将使用当前的数据库。

fix | **nofix** — 确定 **dbcc** 是否修复已发现的分配错误。**checkalloc** 的缺省模式是 **nofix**。要使用 **fix** 选项，必须使数据库处于单用户模式下。

有关 Adaptive Server 中页分配的讨论，参见*系统管理指南*。

checkcatalog — 检查系统表中以及系统表之间的一致性。例如，它可以确保 *syscolumns* 中的每一类型在 *systypes* 中都有一个匹配条目，确保 *sysobjects* 中的每个表和视图在 *syscolumns* 中都至少有一列，并确保 *syslogs* 中的最后一个检查点有效。**checkcatalog** 还报告任何已定义的段。如果没有给定数据库名称，**checkcatalog** 将检查当前的数据库。

checkdb — 运行与 **checktable** 相同的检查，但它是对指定数据库中的每个表 （包括 *syslogs*）运行检查。如果没有给定数据库名称，**checkdb** 将检查当前的数据库。

skip_ncindex — 使 **dbcc checktable** 或 **dbcc checkdb** 跳过对用户表上非集群索引的检查。缺省情况下，将检查所有索引。

checkstorage — 检查指定数据库的分配、OAM 页条目、页一致性、具有文本值的列、文本值列的分配以及文本列链。每个 **dbcc checkstorage** 操作的结果都存储在 *dbccdb* 中。有关使用 **dbcc checkstorage** 以及从 *dbccdb* 创建、维护和生成报告的详细信息，参见*系统管理指南*。

checktable — 检查指定的表，以确保索引和数据页的链接正确、索引的排序正确、所有指针都一致、每页上的数据信息都合理并且页偏移也合理。如果日志段位于它自己的设备上，那么对 *syslogs* 表运行 **dbcc checktable** 时，将报告所用的日志和可用空间。例如：

```
Checking syslogs
The total number of data pages in this table is 1.
*** NOTICE:Space used on the log segment is 0.20 Mbytes, 0.13%.
*** NOTICE:Space free on the log segment is 153,4 Mbytes, 99,87%.
DBCC execution completed.  If dbcc printed error messages, see your
System Administrator.
```

如果日志段不在它自己的设备上，将显示以下消息：

```
*** NOTICE:  Notification of log space used/free cannot be
reported because the log segment is not on its own device.
```

table_name | *table_id* — 是要检查的表的名称或对象 ID。

checkverify — 校验最近一次对指定数据库运行 **dbcc checkstorage** 的结果。有关使用 **dbcc checkverify** 的详细信息，参见*系统管理指南*。

complete_xact — 通过提交或回退其工作来尝试完成事务。 Adaptive Server 将在 *master.dbo.systransactions* 表中保留有关所有尝试完成事务的信息。这样，外部事务协调器就会对事务的完成情况有所了解。

◆ ***警告！***

当尝试完成处于就绪状态的事务时，可能会在整个分布式事务中造成不一致的结果。系统管理员尝试提交或回退事务的决定可能会与协调 Adaptive Server 或协议的决定相矛盾。

forget_xact — 把尝试完成事务的提交状态从 *master.dbo.systransactions* 中删除。当系统管理员不希望协调服务知道事务已尝试完成时，或者当无法使用外部协调器来清除 *systransactions* 中的提交状态时，就可以使用 **forget_xact**。

◆ ***警告！***

由于应该允许外部事务协调器监测尝试完成事务，所以不要在常规 DTP 环境中使用 *dbcc forget_xact*。符合 X/Open XA 的事务管理器和 Adaptive Server 事务协调服务将自动清除 *systransactions* 中的提交状态。

xid — 是来自 *systransactions.xactname* 列的事务名称。也可以使用 **sp_transactions** 来确定有效的 *xid* 值。

dbrepair (*database_name*, **dropdb**) — 删除已损坏的数据库。 **drop database** 命令不能用于已损坏的数据库。

当发出该 **dbcc** 语句时，用户 （包括发出该语句的用户）不能在删除的同时使用所删除的数据库。

engine — 使 Adaptive Server 引擎脱机或联机。如果没有指定 *enginenum*， **dbcc engine (offline)** 将使编号最大的引擎脱机。有关详细信息，参见*系统管理指南*中的第 16 章 “管理多处理器服务器”。

fix_text — 当 Adaptive Server 的字符集从任何字符集更改为新的多字节字符集后，将 *text* 值升级。

当更改为多字节字符集后， *text* 数据的内部管理将变得更加复杂。由于 *text* 的值可以大到占据多页，所以 Adaptive Server 必须能够处理跨越页边界的字符。为此，服务器需要有关每个 *text* 页的附加信息。系统管理员或表所有者必须对每个包含 *text* 数据的表运行 **dbcc fix_text**，以计算所需的新值。有关详细信息，参见 *系统管理指南*。

indexalloc — 检查指定的索引，以确保正确分配所有页并使用所有已分配的页。这是 **checkalloc** 的较小版本，它提供相同的对单个索引的完整性检查。

与 **tablealloc** 相同， **indexalloc** 生成以下三种类型的报告：**full**、**optimized** 和 **fast**。如果没有指定类型或使用 **null**， Adaptive Server 将使用 **optimized**。 **fix|nofix** 选项对 **indexalloc** 的作用与对 **tablealloc** 的作用相同。

➤ ***注意***

只有在包含报告类型值 （full、 optimized、 fast 或 null）的情况下，才能指定 fix 或 nofix。

table_name | *table_id, index_id* — 是表名或表的对象 ID （*sysobjects* 中的 *id* 列）以及索引在 *sysindexes* 中的 *indid*。

full — 报告所有类型的分配错误。

optimized — 根据索引的对象分配映射 (OAM) 页中列出的分配页来生成报告。它不报告也无法修复 OAM 页中未列出的分配页上的未引用扩展。 **optimized** 选项是缺省选项。

fast — 不生成分配报告，但生成页的例外报告，这些页已被引用但没有在扩展中分配 （2521 级错误）。

fix | **nofix** — 确定 **indexalloc** 是否修复在表中发现的分配错误。对于除了系统表索引以外的所有索引，缺省值都是 **fix** ；对于系统表索引，缺省值是 **nofix**。要将 **fix** 选项用于系统表，必须首先使数据库处于单用户模式。

只有在包含报告类型值 （**full**、 **optimized**、 **fast** 或 **null**）的情况下，才能指定 **fix** 或 **nofix**。

rebuild_text — 为 *text* 或 *image* 数据重建或创建内部 Adaptive Server 12.x 数据结构。该数据结构使 Adaptive Server 能够在数据查询过程中执行随机访问和异步预取。

reindex — 通过运行 **dbcc checktable** 的快速版本来检查用户表索引的完整性。它可以用于表名或表的对象 ID（*sysobjects* 中的 *id* 列）。**reindex** 将在发现第一个与索引相关的错误时输出一条消息，然后删除并重新创建可疑索引。当 Adaptive Server 的排序顺序被更改且 Adaptive Server 将索引标记为 “suspect” 之后，系统管理员或表所有者必须运行 **dbcc reindex**。

当 **dbcc** 发现损坏的索引时，它将删除并重新创建相应的索引。如果表的索引已经是正确的，或者表没有索引，**dbcc reindex** 将不重建索引，但会输出信息性消息。

如果表被怀疑为包含损坏的数据，**dbcc reindex** 将中止。当发生这种情况时，将显示错误消息，指导用户运行 **dbcc checktable**。**dbcc reindex** 不允许对系统表执行重新索引。当 Adaptive Server 因排序顺序更改而重新启动之后，如有必要，将检查并重建系统索引，这是自动恢复的一部分。

tablealloc — 检查指定的表，以确保正确分配所有页并使用所有已分配的页。这是 **checkalloc** 的较小版本，它提供相同的对单个表的完整性检查。它可以用于表名或表的对象 ID（*sysobjects* 中的 *id* 列）。有关 **tablealloc** 输出的示例，参见*系统管理指南*。

使用 **tablealloc** 可以生成三种类型的报告：**full**、**optimized** 和 **fast**。如果没有指定类型或使用 **null**，Adaptive Server 将使用 **optimized**。

full — 在表级别上等同于 **checkalloc**；它报告所有类型的分配错误。

optimized — 根据表的对象分配映射 (OAM) 页中列出的分配页来生成报告。它不报告也无法修复 OAM 页中未列出的分配页上的未引用扩展。**optimized** 选项是缺省选项。

fast — 不生成分配报告，但生成页的例外报告，这些页已被引用但没有在扩展中分配（2521 级错误）。

fix | **nofix** — 确定 **tablealloc** 是否修复在表中发现的分配错误。对于除系统表外的所有表，缺省值都是 **fix**；对于系统表，缺省值是 **nofix**。要将 **fix** 选项用于系统表，必须首先使数据库处于单用户模式。

只有在包含报告类型值（**full**、**optimized**、**fast** 或 **null**）的情况下，才能指定 **fix** 或 **nofix**。

traceon | traceoff — 在查询优化过程（*flag* 的值为 302、310 和 317）中切换诊断的输出。值 3604 和 3605 分别将跟踪输出的发送目标切换到用户会话和错误日志。有关详细信息，参见 *Performance and Tuning Guide* 中的 “Tuning with dbcc traceon”。

tune — 启用或禁用特殊性能状况的调优标志。有关各个选项的详细信息，参见 *Performance and Tuning Guide*。

示例

1. **dbcc checkalloc(pubs2)**

   检查 *pubs2* 中的页分配错误。

2. **dbcc checkstorage(pubs2)**

   检查 *pubs2* 的数据库一致性并将信息存放在 *dbccdb* 数据库中。

3. **dbcc tablealloc(publishers, null, nofix)**

   Adaptive Server 返回该表分配的优化报告，但没有修复任何分配错误。

4. **dbcc checktable(salesdetail)**

```
Checking salesdetail
The total number of pages in partition 1 is 3.
The total number of pages in partition 2 is 1.
The total number of pages in partition 3 is 1.
The total number of pages in partition 4 is 1.
The total number of data pages in this table is 10.
Table has 116 data rows.
DBCC execution completed. If DBCC printed error messages, contact
a user with System Administrator (SA) role.
```

5. **dbcc indexalloc ("pubs..titleauthor", 2, full)**

   Adaptive Server 返回索引分配 （该索引在 *titleauthor* 表上的 *indid* 为 2）的完全报告并修复任何分配错误。

6. **dbcc rebuild_text (blurbs)**

   为 *blurbs* 表中的所有 *text* 和 *image* 列重建或创建内部 Adaptive Server 12.x 数据结构。

7. **dbcc reindex(titles)**

```
One or more indexes are corrupt. They will be
     rebuilt.
```

   dbcc reindex 已经发现 *titles* 表中的一个或多个损坏的索引。

8. **dbcc fix_text(blurbs)**

   在字符集更改后升级 *blurbs* 的文本值。

9. **dbcc complete_xact (distributedxact1, "rollback")**

   尝试中止事务 “Distributedxact1”。

10. **dbcc forget_xact (distributedxact1)**

    将事务 “Distributedxact1” 的信息从 *master.dbo.systransactions* 中删除。

注释

- **dbcc** （即数据库一致性检查程序）可以在数据库处于活动状态时运行，但 **dbrepair(*database_name*, dropdb)** 选项和带有 **fix** 选项的 **dbcc checkalloc** 则除外。
- **dbcc** 在检查数据库时会将数据库锁定。有关在使用 **dbcc** 时尽量减少性能问题的信息，参见 *系统管理指南* 中的 **dbcc** 讨论。
- 要通过用户名或数据库名来限定表名或索引名，可以用单引号或双引号将所限定的名称引起来。例如：

  ```
  dbcc tablealloc("pubs2.pogo.testtable")
  ```

- **dbcc reindex** 不能在用户定义的事务内运行。
- **dbcc fix_text** 可以生成大量的日志记录，这些日志记录可能会充满事务日志。按照 **dbcc fix_text** 的设计，更新将在一系列小事务中完成。这样，日志空间一旦出现故障，只会丢失一小部分工作。如果日志空间不足，可以使用表来清除日志并重新启动 **dbcc fix_text**，该表应该与原 **dbcc fix_text** 失败时正在升级的表相同。
- 如果在更改为多字节字符集后，试图对 *text* 值使用 **select**、 **readtext** 或 **writetext**，而没有运行 **dbcc fix_text**，那么命令将会失败，错误消息将指导您对该表运行 **dbcc fix_text**。不过，更改字符集之后，可以在不运行 **dbcc fix_text** 的情况下删除 *text* 行。
- **dbcc** 输出将作为消息或错误发送，而不是作为结果行发送。客户端程序和脚本应检查相应的出错处理程序。
- 如果表已分区， **dbcc checktable** 将返回有关每个分区的信息。
- 已升级到 Adaptive Server 版本 12.x 的 *text* 和 *image* 数据**不会**自动升级到新的存储格式。要提高查询性能并实现该数据的预取，可以对已升级的 *text* 和 *image* 列使用 **rebuild_text** 关键字。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

只有表所有者才能使用 **checktable**、 **fix_text**、 **rebuild_text** 或 **reindex** 关键字来执行 **dbcc**。只有数据库所有者才能使用 **checkstorage**、 **checkdb**、 **checkcatalog**、 **checkalloc**、 **indexalloc** 和 **tablealloc** 关键字。只有系统管理员才能使用 **dbrepair**、 **complete_xact** 和 **forget_xact** 关键字。只有系统管理员才能使用 **dbcc traceon** 和 **dbcc traceoff** 命令。只有系统管理员才能使用 **dbcc engine**。

参见

命令	**drop database**
系统过程	**sp_configure、 sp_helpdb**

deallocate cursor

功能

使游标不可访问并释放已提交到该游标的所有内存。

语法

```
deallocate cursor cursor_name
```

参数

cursor_name — 是要释放的游标的名称。

示例

```
1. deallocate cursor authors_crsr
```

释放名为 “Authors_crsr”的游标。

注释

- 如果游标不存在， Adaptive Server 将返回错误消息。
- 要将游标的名称用作其它 **declare cursor** 语句的一部分，必须先释放该游标。
- 当在存储过程或触发器中指定 **deallocate cursor** 时，它对内存资源的使用没有影响。
- 无论游标是打开的还是关闭的，都可以使用 **deallocate** 来释放游标。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都具有 **deallocate cursor** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	close、 declare cursor

declare

功能

声明批处理或过程的局部变量名称和类型。

语法

变量声明：

```
declare @variable_name datatype
   [, @variable_name datatype]...
```

变量赋值：

```
select @variable = {expression | select_statement}
   [, @variable = {expression | select_statement} ...]
   [from table_list]
   [where search_conditions]
   [group by group_by_list]
   [having search_conditions]
   [order by order_by_list]
   [compute function_list [by by_list]]
```

关键字和选项

@variable_name — 必须以 @ 开始并且必须符合标识符规则。

datatype — 既可以是系统数据类型，也可以是用户定义的数据类型。

示例

1.
```
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"
```

```
you got one
you got two
```

根据变量中的值声明两个变量并输出字符串。

2.
```
declare @veryhigh money
select @veryhigh = max(price)
     from titles
if @veryhigh > $20
  print "Ouch!"
```

如果 *titles* 表中的最高书价大于 $20.00，则输出 “Ouch!”。

注释

- 用 **select** 语句将值赋给局部变量。
- 一个过程中的最大参数数量为 255。局部或全局变量的最大数量仅受可用内存的限制。 @ 符号表示变量名。
- 局部变量经常用作 **while** 循环或 **if...else** 块的计数器。在存储过程中，局部变量被声明在执行存储过程时供该过程以自动、非交互的方式使用。局部变量必须在将其声明的批处理或过程中使用。
- 为局部变量赋值的 **select** 语句通常返回单个值。如果要返回多个值，最后一个值将赋给局部变量。为变量赋值的 **select** 语句不能用于检索同一语句中的数据。
- **print** 和 **raiserror** 命令可以将局部变量当作参数。
- 在 **select** 语句中，用户不能直接创建全局变量，也不能直接更新全局变量的值。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都具有 **declare** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	**print**、 **raiserror**、 **select**、 **while**

declare cursor

功能

定义游标。

语法

```
declare cursor_name cursor
   for select_statement
   [for {read only | update [of column_name_list]}]
```

参数

cursor_name — 是所定义的游标的名称。

select_statement — 是定义游标结果集的查询。有关详细信息，参见 **select**。

for read only — 指定不能更新的游标结果集。

for update — 指定游标结果集可以更新。

of *column_name_list* — 是定义为可更新的游标结果集 （由 *select_statement* 指定）中列的列表。 Adaptive Server 还允许用户包含这样的列：它们未在游标 *select_statement* 的列列表中指定（并排除在结果集之外），但属于在 *select_statement* 中指定的表。

示例

1. ```
   declare authors_crsr cursor
   for select au_id, au_lname, au_fname
   from authors
   where state != 'CA'
   ```

   为 *authors_crsr* 游标定义结果集，其中包含 *authors* 表中所有居住在 California 之外的作者。

2. ```
   declare titles_crsr cursor
   for select title, title_id from titles
   where title_id like "BU%"
   for read only
   ```

   为 *titles_crsr* 游标定义只读结果集，其中包含 *titles* 表中的商业类书籍。

```
3. declare pubs_crsr cursor
   for select pub_name, city, state
   from publishers
   for update of city, state
```

为 *pubs_crsr* 游标定义可更新的结果集，其中包含 *publishers* 表中的所有行。它定义每个出版者的地址 （*city* 和 *state* 列），以便于更新。

注释

游标的限制

- **declare cursor** 语句必须出现在该游标的任何 **open** 语句之前。
- 不能使用 **declare cursor** 将其它语句包含在同一 Transact-SQL 批处理中。
- *cursor_name* 必须是有效的 Adaptive Server 标识符。

游标 *select* 语句

- *select_statement* 可以使用 Transact-SQL **select** 语句的完整语法和语义，但受到以下限制：
  - *select_statement* 必须包含 **from** 子句。
  - *select_statement* 不能包含 **compute**、 **for browse** 或 **into** 子句。
  - *select_statement* 可以包含 **holdlock** 关键字。
- *select_statement* 可以包含对 Transact-SQL 参数名或 Transact-SQL 局部变量的引用 （对于除语言之外的所有游标类型）。这些名称必须引用在包含 **declare cursor** 语句的过程、触发器或语句批处理中定义的 Transact-SQL 参数和局部变量。

  在游标打开之前， **declare cursor** 语句中引用的参数和局部变量不必包含有效值。
- *select_statement* 可以包含在触发器中使用的，对 *inserted* 和 *deleted* 临时表的引用。

范围

- 游标的存在取决于它的**范围**。范围是指使用游标的环境，例如在用户会话内，在存储过程内或在触发器内。

  在用户会话内，游标只在用户结束会话之前存在。对于其它用户启动的任何其它会话，游标都不存在。当用户注销之后，Adaptive Server 将释放在该会话中创建的游标。

  如果 **declare cursor** 语句是存储过程或触发器的一部分，在其中创建的游标将应用于存储过程或触发器范围以及启动该存储过程或触发器的范围。在触发器内对 *inserted* 或 *deleted* 表声明的游标对于所有嵌套的存储过程或触发器都是不可访问的。但是，在触发器内对 *inserted* 或 *deleted* 表声明的游标在该触发器范围内**是**可以访问的。在完成存储过程或触发器之后， Adaptive Server 将释放在其中创建的游标。

  图 6-1 说明了游标在范围间的运行方式。

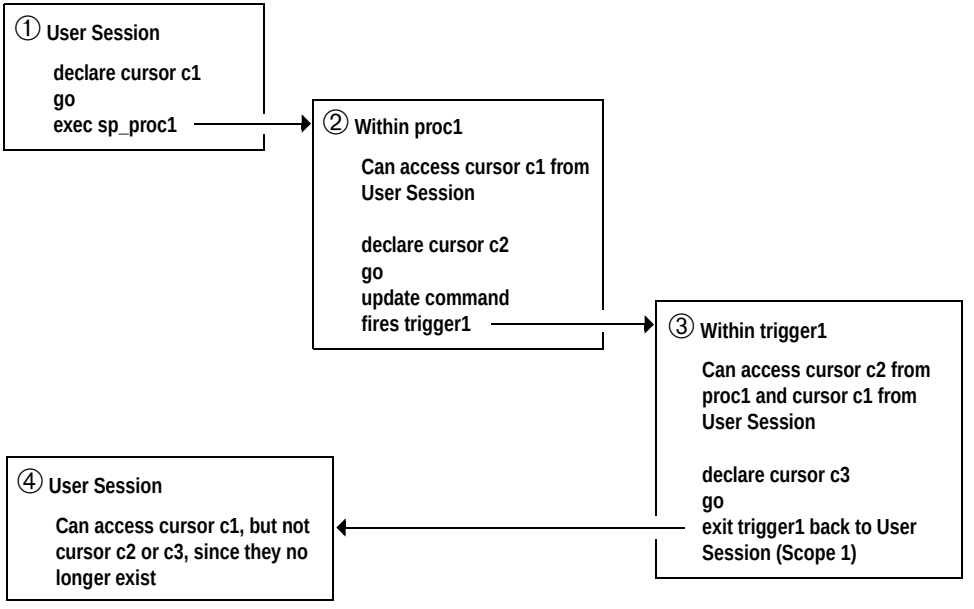


图 6-1：游标在范围内的运行方式

- 在给定范围内，游标名必须唯一。 Adaptive Server 只在运行期才检查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，以下存储过程之所以可行，是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
```

结果集

- 游标结果集中的行可能并不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通过创建内部表来排列游标结果集中行的顺序。 Adaptive Server 不锁定基表中对应于内部表行的那些行，这使其它客户端可以更新这些基表行。在这种情况下，从游标结果集返回给客户端的行可能与基表行不同步。
- 当通过游标的 **fetch** 操作返回行时，将生成该游标的结果集。这意味着处理游标 **select** 查询的方式与处理常规 **select** 查询的方式相同。此进程 （称为**游标扫描**）缩短了操作周期，并且使应用程序不需要的行不再被读取。

  游标扫描存在一项限制：它们只能使用表的唯一索引。但是，如果游标结果集所引用的任何基表都没有被与游标处于相同锁定空间的其它进程更新，那么这一限制就是不必要的。 Adaptive Server 允许在无唯一索引的表上声明游标。但是，只要尝试更新相同锁定空间中的这些表，就会关闭表中的所有游标。

可更新的游标

- 用 **declare cursor** 定义游标之后，Adaptive Server 将确定游标的状态是**可更新**还是**只读**。如果游标是可更新的，就可以更新或删除游标结果集中的行。如果游标是只读的，则不能更改结果集。
- 使用 **for update** 或 **for read only** 子句可以显式地将游标定义为可更新或只读。如果游标的 *select_statement* 包含以下结构之一，则不能将该游标定义为可更新：
  - **distinct** 选项
  - **group by** 子句
  - 集合函数

- 子查询
- **union** 运算符
- **at isolation read uncommitted** 子句

如果省略 **for update** 或 **read only** 子句， Adaptive Server 将检查该游标是否可以更新。

如果声明一个语言类型或服务器类型的游标，而该游标将一个 **order by** 子句作为其 *select_statement* 语句的一部分，那么 Adaptive Server 也会将该游标定义为只读。对于客户端类型和执行类型的游标， Adaptive Server 将以不同的方式处理更新操作，从而消除了这一限制。

- 如果没有用 **for update** 子句指定 *column_name_list*，查询中的所有指定列都将是可更新的。在扫描基表时， Adaptive Server 将尽量为可更新游标使用唯一的索引。对于游标， Adaptive Server 会认为包含 IDENTITY 列的索引是唯一索引，即使并未如此声明。

  如果没有指定 **for update** 子句， Adaptive Server 将选择任意的唯一索引。不过当指定表列没有唯一索引时，它也可以使用其它索引或表扫描。但是，如果指定了 **for update** 子句， Adaptive Server 必须使用为一个或多个列定义的唯一索引来扫描基表。如果不存在唯一索引，将返回一个错误。

- 大多数情况下， **for update** 子句的 *column_name_list* 中应该只包括要更新的列。如果表只有一个唯一索引，则无需在 **for update** 的 *column_name_list* 中包括该表的列；在执行游标扫描时， Adaptive Server 将找到该列。如果该表有多个唯一索引，则应在 **for update** *column_name_list* 中包括该表的列，以便于 Adaptive Server 在执行游标扫描时迅速找到该列。

  这样， Adaptive Server 就可以在游标扫描中使用该唯一索引，以避免出现被称为 **Halloween 问题**的更新异常。要避免出现 Halloween 问题，另一种方法是使用 **unique auto_identity index** 数据库选项来创建表。有关详细信息，参见*系统管理指南*。

  如果客户端所更新的游标结果集行的列定义了从基表返回行的顺序，就会发生这样问题。例如，如果 Adaptive Server 使用某个索引访问基表，并且该索引键被客户端更新，那么被更新的索引行可以在索引中移动，并且可以再次被游标读取。之所以这样，是因为可更新的游标仅以逻辑方式创建游标结果集。该游标结果集实际上是游标所派生的基表。

- 如果指定 **read only** 选项，则不能使用 **delete** 或 **update** 语句来更新游标结果集。

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	**for update** 和 **for read only** 选项是 Transact-SQL 扩展。

权限

缺省情况下，所有用户都具有 **declare cursor** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	open

delete

功能

从表中删除行； **readpast** 选项使 **delete** 命令可以跳过锁定的行而不被阻塞。

语法

```
delete [from]
   [[database.]owner.]{view_name|table_name}
   [where search_conditions]
   [plan "abstract plan"]

delete [[database.]owner.]{table_name | view_name}
   [from [[database.]owner.]{view_name [readpast]|
       table_name [readpast]
           [(index {index_name | table_name }
           [ prefetch size ][lru|mru])]}
    [, [[database.]owner.]{view_name [readpast]|
       table_name [readpast]
           [(index {index_name | table_name }
           [ prefetch size ][lru|mru])]} ...]
   [where search_conditions] ]
   [plan "abstract plan"]

delete [from]
   [[database.]owner.]{table_name|view_name}
   where current of cursor_name
```

关键字和选项

from — （在 **delete** 之后）是可选关键字，用于与其它版本的 SQL 兼容。

view_name | *table_name* — 是要从中删除行的视图或表的名称。当视图或表位于其它数据库时，指定该数据库的名称；当数据库中存在具有该名称的多个视图或表时，指定所有者的名称。 *owner* 的缺省值是当前用户，而 *database* 的缺省值是当前数据库。

where — 是标准的 **where** 子句。有关详细信息，参见 “where 子句”。

from —（在 *table_name* 或 *view_name* 之后）用于在指定要删除的行时指定用于 **where** 子句的多个表或视图。该 **from** 子句使您可以根据存储在其它表中的数据来删除某个表中的行，从而为您提供了嵌入式 **select** 语句的强大功能。

readpast — 指定 **delete** 命令跳过所有持有不兼容锁的页或行，而无需等待锁或超时。对于由数据页锁定的表，该命令将跳过所有持有不兼容锁的页上的行；对于由数据行锁定的表，它将跳过所有持有不兼容锁的行。

index *index_name* — 指定用于访问 *table_name* 的索引。当从视图中进行删除时，不能使用此选项。

prefetch *size* — 为配置了大型 I/O 的缓存所绑定的表指定 I/O 大小（以千字节为单位）。有效的大小值为 2、 4、 8 和 16。当从视图中进行删除时，不能使用此选项。 **sp_helpcache** 显示对象要绑定到的缓存或缺省缓存的有效大小。

当启用组件集成服务后，将不能为远程服务器使用 **prefetch** 关键字。

lru | mru — 指定用于表的缓冲区替换策略。使用 **lru** 可强制优化程序将表读入 MRU/LRU （最近使用最多 / 最近使用最少）链上的缓存。使用 **mru** 可以从缓存中放弃缓冲区，并将其替换为该表的下一个缓冲区。当从视图中进行删除时，不能使用此选项。

plan *"abstract plan"*— 指定用于优化查询的抽象计划。它可以是用抽象计划语言指定的完整计划或部分计划。有关详细信息，参见 *Performance and Tuning Guide* 中的第 22 章 “Creating and Using Abstract Plans”。

where current of *cursor_name* — 使 Adaptive Server 删除 *cursor_name* 的当前游标位置所指定的表或视图中的行。

示例

1.
```
delete authors
```

   删除 *authors* 表中的所有行。

2.
```
delete from authors
where au_lname = "McBadden"
```

   删除 *authors* 表中的一行或多行。

3.
```
delete titles
from titles, authors, titleauthor
where authors.au_lname = 'Bennet'
  and authors.au_id = titleauthor.au_id
  and titleauthor.title_id = titles.title_id
```

   从 *titles* 表中删除作者为 Bennet 的书籍的行。（*pubs2* 数据库包括一个触发器 (*deltitle*)，用于防止删除在 *sales* 表中记录的标题；要使本示例生效，应删除此触发器。）

4. **delete titles where current of title_crsr**

   删除 *titles* 表中当前由游标 *title_crsr* 指定的行。

5. **delete authors**
   where syb_identity = 4

   确定哪一行的 IDENTITY 列的值为 4，并将该行从 *authors* 表中删除。务必要使用 **syb_identity** 关键字，而不要使用 IDENTITY 列的实际名称。

6. **delete from authors from authors readpast**
   where state = "CA"

   删除 *authors* 中的行，跳过所有锁定的行。

7. **delete stores from stores readpast, authors**
   where stores.city = authors.city

   删除 *stores* 中的行，跳过所有锁定的行。如果 *authors* 中的任何行被锁定，查询就会在这些行处阻塞，直到这些锁被释放。

注释

- **delete** 删除指定表中的行。
- 在 **delete** 语句中最多可以引用 15 个表。

限制

- 虽然可以将 **update** 或 **insert** 用于多表视图 （其 **from** 子句指定了多个表），但不能将 **delete** 用于多表视图。如果通过多表视图删除行，将会更改该视图的多个表，这是不允许的。而 **insert** 和 **update** 语句只影响该视图的一个基表，所以是允许的。
- 如果在 **delete** 中对同一表作出两种不同的指定， Adaptive Server 就会将其当作两个表。例如，在 *pubs2* 中发出的以下 **delete** 命令将 *discounts* 指定为两个表 （*discounts* 和 *pubs2..discounts*）：

  ```
  delete discounts
  from pubs2..discounts, pubs2..stores
  where pubs2..discounts.stor_id =
      pubs2..stores.stor_id
  ```

  在此情况下，连接不包括 *discounts*，因此 **where** 条件对于每一行都为真； Adaptive Server 删除 *discounts* 中的所有行 （这并不是预期的结果）。要避免这一问题，可在整个语句中对表使用相同的指定。
- 如果从某个表中删除被其它表通过参照约束引用的行， Adaptive Server 将在允许删除之前检查所有引用表。如果您试图删除的行包含正被某一个引用表用作外键的主键，则不允许此删除。

删除表中的所有行

- 如果不使用 **where** 子句，则将删除在 **delete** [**from**] 后指定的表中的**所有**行。虽然该表已不包含数据，但它会继续存在，直到您发出 **drop table** 命令。
- 未指定行的 **truncate table** 命令与 **delete** 具有相同的功能，但是 **truncate table** 的速度更快。**delete** 每次删除一行并记录这些事务。**truncate table** 则删除整个数据页并且不记录行。

  delete 和 **truncate table** 都能回收数据及其关联索引所占用的空间。
- 不能对已分区的表使用 **truncate table** 命令。要删除已分区表中的所有行，可使用不带 **where** 子句的 **delete** 命令，或在发出 **truncate table** 命令之前取消对表的分区。

delete 和事务

- 在链式事务模式中，如果当前没有活动的事务，每个 **delete** 语句就会隐式地开始一个新事务。使用 **commit** 完成所有删除，或使用 **rollback** 撤消更改。例如：

```
delete from sales where date < '01/01/89'
if exists (select stor_id
    from stores
    where stor_id not in
    (select stor_id from sales))
        rollback transaction
else
        commit transaction
```

  此批处理开始一个事务 （使用链式事务模式）并从 *sales* 表中删除日期早于 Jan. 1, 1989 （1989 年 1 月 1 日）的行。如果它删除与某个商店关联的所有销售条目，则将所有更改回退到 *sales* 并结束该事务。否则，它将提交删除并结束该事务。有关链式模式的详细信息，参见 *Transact-SQL User's Guide*。

删除触发器

- 您可以定义一个触发器，使其在对指定表发出 **delete** 命令时执行指定的动作。

使用 *delete where current of*

- 将子句 **where current of** 用于游标。在使用子句 **where current of** 删除行之前，必须首先使用 **declare cursor** 定义游标，并使用 **open** 语句打开该游标。将游标定位于要用一个或多个 **fetch** 语句来删除的行。此游标的名称不能是 Transact-SQL 参数或局部变量。此游标必须是可更新的游标，否则 Adaptive Server 将返回错误。对游标结果集的任何删除操作也会影响派生游标行的基表行。使用游标时，每次只能删除一行。
- 如果游标的 **select** 语句包含连接子句，那么即使该游标被认为是可更新的，也不能删除游标结果集中的行。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。
- 从游标结果集中删除行之后，游标将定位在游标结果集中的下一行之前。要访问下一行，必须发出 **fetch** 命令。如果已删除的行是游标结果集中的最后一行，游标则定位在结果集的最后一行之后。下面说明了打开的游标在受到 **delete** 影响时的位置和行为：
  - 如果客户端删除某一行（使用其它游标或常规的 **delete**）并且该行表示同一客户端所拥有的其它打开游标的当前游标位置，那么每个受影响的游标的位置将被隐式设置为在下一个可用行之前。但是，一个客户端不能删除表示另一个客户端游标的当前游标位置的行。
  - 如果某个客户端删除的行表示通过连接操作定义并由同一客户端所拥有的另一游标的当前游标位置， Adaptive Server 将接受 **delete** 语句。但是，它将隐式关闭连接操作所定义的游标。

使用 *readpast*

- **readpast** 选项允许 DOL 锁定表上的 **delete** 命令继续进行，而不会被其它任务持有的不兼容锁阻塞。
  - 在数据行锁定表上， **readpast** 将跳过由其它任务持有共享、更新或排它锁的所有行。
  - 在数据页锁定表上， **readpast** 将跳过由其它任务持有共享、更新或排它锁的所有页。
- 如果有排它表锁，指定 **readpast** 的命令就会阻塞。
- 如果为 allpage 锁定表指定了 **readpast** 选项，则将忽略 **readpast** 选项。一旦发现不兼容锁，命令就会阻塞。
- 如果会话范围隔离级别为 3，则忽略 **readpast** 选项。命令在级别 3 执行。命令将在有不兼容锁的任何行或页上阻塞。

- 如果会话的事务隔离级别为 0，则使用 **readpast** 的 **delete** 命令将不发出警告消息。对于数据页锁定表，带有 **readpast** 的 **delete** 命令将修改所有页上未被不兼容锁锁定的所有行。对于数据行锁定表，它将影响没有被不兼容锁锁定的所有行。
- 如果 **delete** 命令应用到具有两个或多个文本列的行，并且任何文本列上有不兼容锁， **readpast** 锁将跳过此行。

使用 *index*、 *prefetch* 或 *lru* | *mru*

- **index**、 **prefetch** 和 **lru | mru** 选项将替换 Adaptive Server 优化程序所作的选择。应小心使用这些选项，并经常通过 **set statistics io on** 检查它们对性能的影响。有关使用这些选项的详细信息，参见 *Performance and Tuning Guide*。

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	在 **from** 子句中使用多个表以及用数据库名限定表名的功能是 Transact-SQL 扩展。 **readpast** 是 Transact-SQL 扩展

权限

缺省情况下，表或视图的所有者具有 **delete** 权限，此所有者可以将其移交给其他用户。

如果将 **set ansi_permissions** 设置为 **on**，那么除了 **delete** 语句所需的常规权限外，还必须对 **where** 子句中出现的所有列具有 **select** 权限。缺省情况下， **ansi_permissions** 设置为 **off**。

参见

命令	**create trigger**、 **drop table**、 **drop trigger**、 **truncate table**、 **where 子句**

delete statistics

功能

删除 *sysstatistics* 系统表中的统计信息。

语法

```
delete [shared] statistics table_name [(column_name [,
   column_name]...)]
```

参数

shared — 删除 *master* 数据库 *sysstatistics* 中的模拟统计信息。

table_name — 删除表中所有列的统计信息。

column_name — 删除指定列的统计信息。

示例

1. `delete statistics titles`

   删除 *titles* 表中所有列的密度、选择性和直方图。

2. `delete statistics titles(pub_id)`

   删除 *titles* 表中 *pub_id* 列的密度、选择性和直方图。

3. `delete statistics titles(pub_id, pubdate)`

   删除 *pub_id* 和 *pubdate* 的密度、选择性和直方图，而不影响单列 *pub_id* 或单列 *pubdate* 的统计信息。

注释

- **delete statistics** 删除 *sysstatistics* 表中指定列或表的统计信息。这不会影响 *systabstats* 表中的统计信息。
- 发出 **drop table** 命令时，将删除 *sysstatistics* 中相应的行。使用 **drop index** 命令时，不会删除 *sysstatistics* 中相应的行。这使得查询优化程序继续使用索引统计信息，而不会为保留表中索引进行开销。

◆ ***警告！***

密度、选择性和直方图对于好的查询优化是必要的。使用 delete statistics 命令来删除优化程序不用的统计信息。如果不注意删除了查询优化所需要的统计信息，则在表、索引或列上运行 update statistics。

- 使用 **optdiag** 实用程序命令装载模拟统计信息时，将使 *master..sysstatistics* 表中增加少量的行。如果不再使用模拟统计信息，则可使用 **delete shared statistics** 命令删除 *master..sysstatistics* 中的这些信息。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

只有表所有者或系统管理员才能使用 **delete statistics**。

参见

命令	**create index**、 **update**
实用程序	**optdiag**

disk init

功能

使物理设备或文件可以被 Adaptive Server 所使用。

语法

```
disk init
   name = "device_name" ,
   physname = "physicalname" ,
   vdevno = virtual_device_number ,
   size = number_of_blocks
   [, vstart = virtual_address ,
   cntrltype = controller_number ]
   [, contiguous]
```

关键字和选项

name — 是数据库设备或文件的名称。该名称必须遵循标识符的规则，并且必须用单引号或双引号引起来。它可以用在 **create database** 和 **alter database** 命令中。

physname — 是数据库设备的完整说明。该名称必须用单引号或双引号引起来。

vdevno — 是虚设备号。在同 Adaptive Server 相关联的数据库设备中，它必须是唯一的。设备号 0 是专为主设备保留的。有效的设备号在 1 到 255 之间，但是最大的设备号必须比 Adaptive Server 所配置的数据库设备号小 1。例如，对于缺省配置为 10 个设备的 Adaptive Server 而言，可以使用的设备号是 1-9。要查看 Adaptive Server 上可以使用的最大设备号，请运行 **sp_configure** 并检查 **number of devices** 值。

要确定虚设备号，请查看 **sp_helpdevice** 报告的 *device_number* 列，并使用下一个未使用的整数。

size — 是数据库设备的大小 （以 2K 的块为单位）。

如果打算将新设备用于新数据库的创建，那么最小的 **size** 就是 *model* 数据库的大小，为 1024 个 2K 块 (2MB)。如果您正在初始化一个日志设备， **size** 可以小到 512 个 2K 块 (1MB)。最大值则取决于系统。

➤ ***注意***

如果物理设备上的 2K 块的数目小于 **size** 和 **vstart** 的和， **disk init** 命令就会失败。

vstart — 是起始的虚地址或起始的偏移量 （以 2K 块为单位）。**vstart** 的值应该为 0 （缺省值），除非您在 AIX 操作系统上运行了 Logical Volume Manager，在这种情况下 *vstart* 应该为 2。

除非 Sybase 技术支持部门特别指明，否则不要指定 **vstart**。

cntrltype — 指定了磁盘控制器。其缺省值为 0。除非 Sybase 技术支持部门特别指明，否则不要重新设置 **cntrltype**。

contiguous — （仅 OpenVMS）强制进行邻接数据库文件的创建。该选项仅在初始化**文件**时才有意义；在初始化**外部设备**时它没有影响。如果使用了 **contiguous** 选项，系统就会创建一个邻接文件，或者该命令失败并给出错误消息。如果没有使用 **contiguous** 选项，系统仍会试图创建一个邻接文件。如果系统未能连续地创建该文件，那么它就会创建一个不强制为邻接的文件。在这两种情况下，系统都会显示一条消息，指出所创建文件的类型。

示例

```
1. disk init
     name = "user_disk",
     physname = "/dev/rxy1a",
     vdevno = 2, size = 5120
```

在 UNIX 系统上初始化 5MB 的磁盘。

```
2. disk init
    name = "user_disk",
    physname = "disk$rose_1:[dbs]user.dbs",
    vdevno = 2, size = 5120,
    contiguous
```

在 OpenVMS 系统上初始化 5MB 的磁盘空间，并强制数据库文件的连续创建。

注释

- 主设备由安装程序进行初始化；无须用 **disk init** 来初始化该设备。
- 要成功地完成磁盘初始化，“sybase”用户必须对要初始化的设备拥有适当的操作系统权限。
- 请对每一个新的数据库设备使用 **disk init**。每次发出 **disk init** 命令时，就会将一行添加到 *master..sysdevices*。新的数据库设备不会自动成为缺省数据库存储池的一部分。请使用系统过程 **sp_diskdefault** 将缺省状态分配给数据库设备。
- 在 OpenVMS 系统中，使用逻辑名来引用 **physname**，这样做比使用硬编码的路径名提供了更大的灵活性。例如，如果您将逻辑名“userdisk”定义为：

  ```
  disk$rose_1:[dbs]user.dbs
  ```

  您可以将以上例 2 中的 **physname** 更改为示例 2 “userdisk”。如要重组磁盘或移动 “user.dbs”，只须将该逻辑名重定义为新的路径。

  Adaptive Server 所使用的任一个逻辑名必须是：

  - 系统逻辑名，或
  - 进程逻辑名。该名称在 runserver 文件中定义，用于 Adaptive Server。
- 在每一次使用 **disk init** 之后，请用 **dump database** 或 **dump transaction** 命令来备份 *master* 数据库。这样做使得 *master* 被损坏后恢复起来更加容易和安全。（如果用 **disk init** 添加了一个设备，且未能备份 *master*，您可以通过使用 **disk reinit** 来恢复此更改，然后停止并重新启动 Adaptive Server。）
- 用 **create database** 或 **alter database** 命令的 **on *device_name*** 子句将用户数据库分配给数据库设备。
- 将数据库的事务日志 （系统表 *syslogs*）放到与存储该数据库的其余部分不同的设备上去，其首选的方法是使用 **log on** 扩展来执行 **create database**。另一种方法是，您可以在创建数据库时至少命名两个设备，然后执行 **sp_logdevice**。您也可以使用 **alter database** 将数据库扩展到第二个设备上去，然后运行 **sp_logdevice**。 **log on** 扩展会立即将整个日志移到不同的设备。在事务活动使此次迁移完成之前， **sp_logdevice** 方法会在原来的数据库设备上保留部分系统日志。
- 要得到系统中所有 Adaptive Server 设备 （数据库设备和转储设备）的报告，请执行 **sp_helpdevice** 系统过程。

- 用系统过程 **sp_dropdevice** 删除数据库设备。必须首先删除该设备上的所有已存在的数据库。

  删除了数据库设备之后，只要给它起一个不同的物理名称和虚设备号，您就可以创建一个拥有同样名称的新设备 （利用 **disk init**）。如果要使用同样的物理名称和虚拟设备号，必须重新启动 Adaptive Server。
- 如果由于 **size** 值对于数据库设备而言过大而使 **disk init** 失败的话，请在再次执行 **disk init** 之前，使用不同的虚设备号或重新启动 Adaptive Server。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

disk init 权限缺省值为系统管理员，且不能移交。您必须利用 *master* 数据库来使用 **disk init**。

参见

命令	**alter database**、**create database**、**disk refit**、**disk reinit**、**dump database**、**dump transaction**、**load database**、 **load transaction**
系统过程	**sp_diskdefault**、 **sp_dropdevice**、 **sp_helpdevice**、 **sp_logdevice**

disk mirror

功能

创建在主设备失败时，可以立即接管的软件镜像。

语法

```
disk mirror
   name = "device_name" ,
   mirror = "physicalname"
   [ , writes = { serial | noserial }]
   [ ,contiguous ] (OpenVMS only)
```

关键字和选项

name — 是要创建镜像的数据库设备的名称。这记录在 *sysdevices* 表中的 *name* 列。该名称必须用单引号或双引号引起来。

mirror — 是要作为辅助设备的数据库镜像设备的完整路径名。它必须用单引号或双引号引起来。如果辅助设备是一个文件，*physicalname* 应该是路径的说明，它可以清楚地标识出 Adaptive Server 要创建的文件。它不能是一个已存在的文件。

writes — 允许您选择是否对设备强制进行串行写。在缺省情况下，**(serial)**，对主数据库设备的写操作一定会在对辅助设备的写操作开始之前结束。若主设备和辅助设备位于不同的物理设备上，串行写可确保在发生电源故障时至少有一个磁盘不受影响。

contiguous — （仅 OpenVMS）只有当镜像是一个文件而不是一个外部设备时，才有意义。对于将用作辅助设备的文件，该选项强制文件要连续地创建。如果使用了 **contiguous** 选项，系统就会创建一个邻接文件，或者该命令失败并给出错误消息。如果没有使用 **contiguous** 选项，系统仍会试图创建一个邻接文件。如果系统未能连续地创建该文件，那么它就会创建一个不强制为邻接的文件。在这两种情况下，系统都会显示一条消息，指出所创建文件的类型。对于 OpenVMS 用户而言， **contiguous** 选项也可以同 **disk init** 一起使用。

示例

```
1. disk mirror
     name = "user_disk",
     mirror = "/server/data/mirror.dat"
```

在文件 *mirror.dat* 上，为数据库设备 *user_disk* 创建软件镜像。

注释

- 磁盘镜像创建了用户数据库设备、主数据库设备或用于用户数据库事务日志的数据库设备的软件镜像。如果数据库设备失败，其镜像会立即替代它。

  磁盘镜像不会干扰数据库中正在进行的活动。在不关闭 SQL Server 的条件下，您就可以对数据库设备创建镜像或取消镜像。

- 在每一次使用 **disk mirror** 之后，请用 **dump database** 命令来备份 *master* 数据库。这样做使得 *master* 被损坏后恢复起来更加容易和安全。

- 向已镜像的设备读取或写入失败后， Adaptive Server 会对这一坏设备进行撤消镜像并显示错误消息。 Adaptive Server 继续在没有镜像的状态下运行。系统管理员必须使用 **disk remirror** 命令来重新启动镜像。

- 您可以对主设备、存储数据的设备和存储事务日志的设备进行镜像。但是，您不能对转储设备进行镜像。

- 镜像是对设备进行的，而不是对数据库进行的。

- 设备及其镜像构成了一个逻辑设备。 Adaptive Server 将镜像设备的物理名存储到 *sysdevices* 表的 *mirrorname* 列中。它不需要 *sysdevices* 中的一个单独的条目，并且不应用 **disk init** 来初始化。

- 要保持异步 I/O 的使用，则总要将能够执行异步 I/O 的设备镜像到其它能够执行异步 I/O 的设备。在绝大多数情况下，这意味着将原始设备镜像到原始设备并且将操作系统文件镜像到操作系统文件。

  如果操作系统不能对文件执行异步 I/O，则将原始设备镜像到某个常规文件将会产生错误消息。将常规文件镜像到原始设备可以使用，但不能使用异步 I/O。

- 镜像所有缺省的数据库设备，以便在 **create** 或 **alter database** 命令影响缺省列表中的数据库设备时仍可以得到保护。

- 要实现更大程度上的保护，请镜像用于事务日志的数据库设备。

- 应始终将用户数据库放在单独的数据库设备上。要将数据库事务日志 （即系统表 *syslogs*）和数据库的其余部分分别放置在两个不同的设备上，可在创建数据库时指定数据设备和日志设备。或者，您也可以使用 **alter database** 将数据库扩展到第二个设备上去，然后运行 **sp_logdevice**。

- 如果为 *master* 数据库镜像了数据库设备，在用 **dataserver** 实用程序重新启动 Adaptive Server 时，您可以将 **-r** 选项和镜像名用于 UNIX，或者将 **mastermirror** 选项用于 OpenVMS。将这添加到用于该服务器的 *RUN_servername* 文件，从而使 **startserver** 实用程序知道。例如：

  ```
  dataserver -dmaster.dat -rmirror.dat
  ```

  启动名为 *master.dat* 的主设备及其镜像 *mirror.dat*。有关详细信息，参见针对于所用平台的*实用程序*手册中的 **dataserver** 和 **startserver**。
- 如果镜像了有未分配空间 （用于额外的 **create database** 和 **alter database** 语句的空间，用以分配部分设备）的数据库设备， **disk mirror** 在这些分配完成时就开始对其进行镜像，而不是在发出 **disk mirror** 命令时才开始。
- 要得到系统中所有 Adaptive Server 设备的报告 （用户数据库设备及其镜像以及转储设备），执行 **sp_helpdevice** 系统过程。
- 有关 OpenVMS 环境中的磁盘镜像的详细信息，参见针对于所用平台的配置文档。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

disk mirror 权限缺省值为系统管理员，且不能移交。您必须利用 *master* 数据库来使用 **disk mirror**。

参见

命令	**alter database**、**create database**、**disk init**、**disk refit**、**disk reinit**、**disk remirror**、**disk unmirror**、**dump database**、**dump transaction**、**load database**、**load transaction**
系统过程	**sp_diskdefault**、**sp_helpdevice**、**sp_logdevice**
实用程序	**dataserver**、**startserver**

disk refit

功能

由 *sysdevices* 中所包含的信息，重建 *master* 数据库的 *sysusages* 和 *sysdatabases* 系统表。

语法

```
disk refit
```

示例

```
1. disk refit
```

注释

- 在 **disk refit** 重建系统表之后， Adaptive Server 将自动关闭。
- 在 **disk reinit** 之后使用 **disk refit**，以此作为恢复主数据库过程的一部分。详细信息，参见*系统管理指南*。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

disk refit 权限缺省值为系统管理员，且不能移交。您必须在 *master* 数据库中使用 **disk redit**。

参见

命令	disk init、 disk reinit
系统过程	sp_addumpdevice、 sp_helpdevice

disk reinit

功能

重建 *master* 数据库的 *sysdevices* 系统表。使用 **disk reinit**，以此作为恢复 *master* 数据库过程的一部分。

语法

```
disk reinit
   name = "device_name" ,
   physname = "physicalname" ,
   vdevno = virtual_device_number ,
   size = number_of_blocks
   [, vstart = virtual_address ,
   cntrltype = controller_number]
```

关键字和选项

name — 是数据库设备的名称。该名称必须遵循标识符的规则，并且必须用单引号或双引号引起来。它可以用在 **create database** 和 **alter database** 命令中。

phyname — 是数据库设备的名称。该物理名必须用单引号或双引号引起来。

vdevno — 是虚拟设备号。在由 Adaptive Server 使用的设备中，它必须是唯一的。设备号 0 是专为 *master* 数据库设备保留的。合法的设备号在 1 到 255 之间，但是它不能比您的系统所配置的数据库设备数大。缺省值为 50 个设备。

size — 是数据库设备的大小 （以 2K 的块为单位）。最小的可用大小是 1024 个 2K 块 (2MB)。

vstart — 是起始的虚拟地址 （即起始偏移量），以 2K 块为单位。**vstart** 的值应该为 0 （缺省值），除非您在 AIX 操作系统上运行了 Logical Volume Manager，在这种情况下 *vstart* 应该为 2。

除非 Sybase 技术支持部门特别指明，否则不要指定 **vstart**。

cntrltype — 指定磁盘控制器。其缺省值为 0。除非 Sybase 技术支持部门特别指明，否则不要重新设置它。

示例

```
1. disk reinit
     name = "user_disk",
     physname = "/server/data/userdata.dat",
     vdevno = 2, size = 5120
```

注释

- 如果主数据库已被损坏，或在 *master* 的最后一次转储后又添加了设备， **disk reinit** 可确保 *master..sysdevices* 的正确性。
- **disk reinit** 与 **disk init** 类似，但是它不初始化数据库设备。
- 有关恢复 *master* 数据库的完整信息，参见*系统管理指南*。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

disk reinit 权限缺省值为系统管理员，且不能移交。您必须在 *master* 数据库中使用 **disk reinit**。

参见

命令	alter database、 create database、 dbcc、 disk init、 disk refit
系统过程	sp_addumpdevice、 sp_helpdevice

disk remirror

功能

在磁盘镜像因所镜像设备的故障而停止，或是临时被 **disk unmirror** 命令禁用之后，重新启动磁盘镜像。

语法

```
disk remirror
   name = "device_name"
```

关键字和选项

name — 是要重镜像的数据库设备的名称。它记录在 *sysdevices* 表中的 *name* 列。该名称必须用单引号或双引号引起来。

示例

```
1. disk remirror
     name = "user_disk"
```

重新开始数据库设备 *user_disk* 上的软件镜像。

注释

- 磁盘镜像将创建用户数据库设备、主数据库设备或用于用户数据库事务日志的数据库设备的软件镜像。如果数据库设备失败，其镜像会立即替代它。

  在镜像临时因所镜像设备的故障而停止，或是临时用 **disk unmirror** 命令的 **mode = retain** 选项而禁用之后，请使用 **disk remirror** 命令来重建镜像。 **disk remirror** 命令将所保留的磁盘上的数据复制到镜像。

- 每次使用完 **disk remirror** 之后，用 **dump database** 命令来备份 *master* 数据库是非常重要的。这样做使得 *master* 被损坏后恢复起来更加容易和安全。

- 如果用 **mode = remove** 选项永远禁用了镜像，您必须在使用 **disk remirror** 命令之前删除包含该镜像的操作系统文件。

- 被镜像的是数据库设备，而不是数据库。

- 您可以镜像、重镜像、挂起镜像数据库设备而无需关闭 Adaptive Server。磁盘镜像不会干扰数据库中正在进行的活动。

- 向已镜像的设备读取或写入失败后， Adaptive Server 会取消这一坏设备的镜像并显示错误消息。 Adaptive Server 继续在没有镜像的状态下运行。系统管理员必须使用 **disk remirror** 命令来重新启动镜像。

- 除了镜像用户数据库设备之外，应始终将用户数据库事务日志放在不同的数据库设备上。用于事务日志的数据库设备也可以被镜像，从而获得更好的保护。要将数据库事务日志 （即系统表 *syslogs*）和数据库的其余部分分别放置在不同的设备上，可在创建数据库时指定数据库设备和记录设备。或者，对第二个设备使用 **alter database**，然后运行 **sp_logdevice**。
- 如果为 *master* 数据库镜像了数据库设备，在用 **dataserver** 实用程序重新启动 Adaptive Server 时，对于 UNIX 平台您可以使用 **-r** 选项和镜像名，对于 OpenVMS 平台可以使用 **mastermirror** 选项。将该选项添加到用于该服务器的 *RUN_servername* 文件，从而使 **startserver** 实用程序知道。例如：

  ```
  dataserver -dmaster.dat -rmirror.dat
  ```

  启动名为 *master.dat* 的主设备及其镜像 *mirror.dat*。有关详细信息，参见所使用平台的*实用程序*手册中的 **dataserver** 和 **startserver**。
- 要得到有关系统中所有 Adaptive Server 设备的报告 （用户数据库设备及其镜像以及转储设备），执行 **sp_helpdevice** 系统过程。
- 有关 OpenVMS 环境中的磁盘镜像的详细信息，参见针对于所用平台的配置文档。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

disk remirror 权限缺省值为系统管理员，且不能移交。使用 **disk remirror** 时必须使用 *master* 数据库。

参见

命令	**alter database**、**create database**、**disk init**、**disk mirror**、**disk refit**、**disk reinit**、**disk unmirror**、**dump database**、**dump transaction**、**load database**、**load transaction**
系统过程	**sp_diskdefault**、**sp_helpdevice**、**sp_logdevice**
实用程序	**dataserver**、**startserver**

disk unmirror

功能

挂起由 **disk mirror** 命令启动的磁盘镜像，以允许硬件维护或硬件设备的更改。

语法

```
disk unmirror
   name = "device_name"
   [, side = { "primary" | secondary }]
   [, mode = { retain | remove }]
```

关键字和选项

name — 是要取消镜像的数据库设备的名称。该名称必须用单引号或双引号引起来。

side — 指定是否禁用 **primary** 设备或 **secondary** 设备 （镜像）。缺省情况下，辅助设备是未镜像的。

mode — 确定取消镜像是临时的 **(retain)** 还是永久的 **(remove)**。缺省情况下，取消镜像是临时的。

当您稍后要在同样的配置中重镜像数据库设备时，请指定 **retain**。该选项模仿了主设备失败时所发生的情况：

- I/O 仅指向**没有**对其进行取消镜像操作的设备
- *sysdevices* 的 *status* 列指示镜像已经失效

remove 消除了所有对镜像设备的 *sysdevices* 引用：

- *status* 列指示镜像功能被忽略
- 如果主设备处于失效状态，则 *phyname* 列由 *mirrorname* 列中辅助设备的名称替换
- *mirrorname* 列设置为 NULL

示例

1. **disk unmirror**
   name = "user_disk"

   挂起数据库设备 *user_disk* 的软件镜像。

2. **disk unmirror name = "user_disk", side = secondary**

   挂起辅助数据库设备 *user_disk* 的软件镜像。

3. **disk unmirror name = "user_disk", mode = remove**

   挂起数据库设备 *user_disk* 的软件镜像，并取消对镜像设备的所有设备引用。

注释

- 磁盘镜像将创建用户数据库设备、主数据库设备或用于用户数据库事务日志的数据库设备的软件镜像。如果数据库设备失败，其镜像会立即替代它。

  disk unmirror 永久或临时地禁用原始的数据库设备或其镜像，从而使该设备对 Adaptive Server 而言不再能够进行读或写。它不会从操作系统中删除相关联的文件。

- 取消磁盘镜像将变更 *master* 数据库中的 *sysdevices* 表。每次使用完 **disk unmirror** 之后，用 **dump database** 命令来备份 *master* 数据库是非常重要的。这样做使得 *master* 被损坏后恢复起来更加容易和安全。

- 您可以在数据库设备正在使用时对其取消镜像。

- 当 **dump database**、**load database** 或 **load transaction** 正在进行时，不能对任一数据库设备进行取消镜像。 Adaptive Server 会显示一条消息询问是否中止转储或装载，或者是否将 **disk unmirror** 命令延迟到转储或装载完成之后。

- 当 **dump transaction** 正在进行时，您不能对数据库记录设备取消镜像。 Adaptive Server 会显示一条消息询问是否中止转储，或将 **disk unmirror** 延迟到转储完成之后。

➤ ***注意***

在取消日志记录设备的镜像时， dump transaction with truncate_only 和 dump transaction with no_log 不会受任何影响。

- 应该镜像所有缺省的数据库设备，以便在 **create** 或 **alter database** 命令影响缺省列表中的数据库设备时仍可以得到保护。
- 向已镜像的设备读取或写入失败后，Adaptive Server 会自动对这一坏设备取消镜像并显示错误消息。Adaptive Server 继续在未镜像的状态下运行。系统管理员必须用 **disk remirror** 命令重新启动镜像。
- 要得到有关系统中所有 Adaptive Server 设备的报告 （用户数据库设备及其镜像以及转储设备），执行 **sp_helpdevice** 系统过程。
- 有关 OpenVMS 环境中的磁盘镜像的详细信息，参见针对于所用平台的配置文档。
- 在用 **disk unmirror** 命令的 **mode = retain** 选项临时停止了镜像之后，请使用 **disk remirror** 来重建镜像。如果用 **mode = remove** 选项永久地禁用了镜像，您必须在使用 **disk remirror** 命令之前删除包含该镜像的操作系统文件。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

disk unmirror 权限缺省值为系统管理员，且不能移交。使用 **disk unmirror** 时必须使用 *master* 数据库。

参见

命令	**alter database**、**create database**、**disk init**、**disk mirror**、 **disk refit**、 **disk reinit**、 **disk remirror**、 **dump database**、 **dump transaction**、 **load database**、 **load transaction**
系统过程	**sp_diskdefault**、 **sp_helpdevice**、 **sp_logdevice**
实用程序	**dataserver**、 **startserver**

drop database

功能

从 Adaptive Server 中删除一个或多个数据库。

语法

```
drop database database_name [, database_name]...
```

关键字和选项

database_name — 是要删除的数据库的名称。请使用 **sp_helpdb** 来获得数据库列表。

示例

```
1. drop database publishing
2. drop database publishing, newpubs
```

所删除的数据库 （及其内容）将会消失。

注释

- 删除数据库将删除该数据库及其所有的对象，释放其存储分配，并从 *master* 数据库中的 *sysdatabases* 和 *sysusages* 系统表中清除其条目。
- **drop database** 将属于所删除数据库的可疑页从 *master..sysattributes* 中清除。

限制

- 删除数据库时必须正在使用 *master* 数据库。
- 不能删除正在使用 （打开来以便任何用户进行读写）的数据库。
- 不能使用 **drop database** 来删除被另一数据库中的表引用的数据库。执行下列查询，以确定哪些表和外部数据库对当前数据库的主键表有外键约束：

```
select object_name(tableid), db_name(frgndbname)
from sysreferences
where frgndbname is not null
```

使用 **alter table** 来删除这些跨数据库的约束，然后重新发出 **drop database** 命令。

- 不能使用 **drop database** 来删除损坏的数据库。使用 **dbcc dbrepair** 命令：

  dbcc dbrepair (*database_name*, dropdb)

- 如果启用了审计，就不能删除 *sybsecurity* 数据库。当审计被禁用时，只有系统安全员才可以删除 *sybsecurity*。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

除了只能由系统安全员删除的 *sybsecurity* 数据库以外，只有数据库所有者可以执行 **drop database**。

参见

命令	alter database、 create database、 dbcc、 use
系统过程	sp_changedbowner、 sp_helpdb、 sp_renamedb、 sp_spaceused

drop default

功能

删除用户定义的缺省值。

语法

```
drop default [owner.]default_name
   [, [owner.]default_name]...
```

关键字和选项

default_name — 是已存在的缺省值名。执行 **sp_help** 来获得已存在的缺省值列表。指定所有者的名称，从而删除由当前数据库中的不同用户所拥有的同样名称的缺省值。*owner* 的缺省值是当前用户。

示例

```
1. drop default datedefault
```

从数据库中删除用户定义的缺省 *datedefault*。

注释

- 不能删除当前已绑定到列或用户定义的数据类型的缺省值。在删除它之前，请使用系统过程 **sp_unbindefault** 来解除绑定缺省值。
- 在没有解除绑定其当前缺省值的情况下，您就可以将新的缺省值绑定到列或用户定义的数据类型。新的缺省值将替换旧的缺省值。
- 当为 NULL 列删除缺省值时， NULL 会成为该列的缺省值。当为 NOT NULL 列删除缺省值时，如果在输入数据时用户没有明确地为该列输入数值，就会出现一条错误消息。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

drop default 权限缺省值为缺省的所有者，且不能移交。

参见

命令	**create default**
系统过程	**sp_help**、 **sp_helptext**、 **sp_unbindefault**

drop index

功能

从当前数据库中的表删除索引。

语法

```
drop index table_name.index_name
   [, table_name.index_name]...
```

关键字和选项

table_name — 是编制索引的列所在的表。该表必须在当前数据库中。

index_name — 是要删除的索引。在 Transact-SQL 中，尽管在表中索引名必须是唯一的，但是在数据库中则不需要是唯一的。

示例

```
1. drop index authors.au_id_ind
```

authors 表中的索引 *au_id_ind* 将不再存在。

注释

- 一旦发出了 **drop index** 命令，您就可以重新获得以前被该索引所占用的全部空间。该空间可以用于任何数据库对象。
- 不能对系统表使用 **drop index**。
- **drop index** 不能删除那些支持唯一约束的索引。要删除这样的索引，请通过 **alter table** 来删除约束，或者删除该表。有关唯一约束索引的详细信息，参见 **create table**。
- 不能删除那些当前正被任何开放游标所使用的索引。有关哪些游标是开放的以及它们使用了什么索引的信息，请使用 **sp_cursorinfo**。
- 要获得有关表上存在了什么索引的信息，请使用：

  ```
  sp_helpindex objname
  ```

  其中 *objname* 是表名。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

drop index 权限缺省为索引的所有者所拥有，且不能移交。

参见

命令	create index
系统过程	sp_cursorinfo、 sp_helpindex、 sp_spaceused

drop procedure

功能

删除过程。

语法

```
drop proc[edure] [owner.]procedure_name
   [, [owner.]procedure_name] ...
```

关键字和选项

procedure_name — 是要删除的过程的名称。指定所有者的名称，从而删除由当前数据库中的不同用户所拥有的同样名称的过程。*owner* 的缺省值是当前用户。

示例

1. `drop procedure showind`

   删除存储过程 **showind**。

2. `drop procedure xp_echo`

   取消注册扩展的存储过程 *xp_echo*。

注释

- **drop procedure** 删除用户定义的存储过程、系统过程和扩展的存储过程 (ESP)。
- 每当用户或程序执行该过程时， Adaptive Server 将检查它的存在性。
- 过程组 （拥有相同的名称，但却具有不同 *number* 后缀的多个过程）可以用单个 **drop procedure** 语句删除。例如，如果用于名为 **orders** 应用程序的过程被命名为 *orderproc;1*、 *orderproc;2* 等等，则下列语句：

  ```
  drop proc orderproc
  ```

  删除整个组。一旦过程被分组，组中的个别过程就不能被删除。例如，语句：

  ```
  drop procedure orderproc;2
  ```

  不允许使用。

  不能将扩展的存储过程作为过程组删除。

- 系统过程 **sp_helptext** 将显示存储在 *syscomments* 中的过程文本。
- 系统过程 **sp_helpextendedproc** 显示 ESP 及其相应的 DLL。
- 删除 ESP 将通过由系统表中删除该过程来对其进行取消注册。这不会影响底层 DLL。
- **drop procedure** 从当前的数据库中仅删除由用户创建的过程。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

drop procedure 权限缺省为过程的所有者所拥有，且不能移交。

参见

命令	create procedure
系统过程	sp_depends、 sp_dropextendedproc、 sp_helpextendedproc、 sp_helptext、 sp_rename

drop role

功能

删除用户定义的角色。

语法

```
drop role role_name [with override]
```

关键字和选项

role_name — 是要删除的角色的名称。

with override — 替换对删除角色的任何限制。在使用 **with override** 选项时，您无须检查是否在每一个数据库中都删除了角色权限，就可以删除任一角色。

示例

1. **drop role doctor_role**

   只有当所有数据库中的全部权限都被撤消后，才能删除所指定的角色。在删除角色之前，系统管理员或对象所有者必须撤消在每一个数据库中所授予的权限，否则该命令就会失败。

2. **drop role doctor_role with override**

   删除指定的角色，并从所有数据库中删除权限信息以及对该角色的任何其它引用。

注释

- 删除角色前不必先删除成员资格。删除一个角色将自动删除此角色中的任何用户的成员资格，无论是否使用了 **with override** 选项。
- 从 *master* 数据库中使用 **drop role**。

限制

- 不能使用 **drop role** 来删除系统角色。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

只有系统安全员才能使用 **drop role**。

drop role 权限不包括在 **grant all** 命令中。

参见

命令	**alter role**、 **create role**、 **grant**、 **revoke**、 **set**
系统过程	**sp_activeroles**、 **sp_displaylogin**、 **sp_displayroles**、 **sp_helprotect**、 **sp_modifylogin**

drop rule

功能

删除用户定义的规则。

语法

```
drop rule [owner.]rule_name [, [owner.]rule_name]...
```

示例

```
1. drop rule pubid_rule
```

在当前数据库中删除规则 *pubid_rule*。

关键字和选项

rule_name — 是要删除的规则的名称。指定所有者的名称，从而删除由当前数据库中的不同用户所拥有的、同样名称的规则。 *owner* 的缺省值是当前用户。

注释

- 在删除规则之前，必须使用系统过程 **sp_unbindrule** 对其进行解除绑定。如果该规则没有被解除绑定，就会出现错误消息，而且 **drop rule** 命令会失败。
- 您可以将新的规则绑定到列或用户定义的数据类型，而无需解除绑定其当前规则。新的规则将替换旧的规则。
- 在删除规则之后， Adaptive Server 会在没有约束的情况下，将新的数据输入到先前由该规则所管理的列中。不管怎样，已存在的数据都不会受到影响。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

drop rule 权限缺省为规则的所有者所拥有，且不能移交。

参见

命令	**create rule**
系统过程	**sp_bindrule**、 **sp_help**、 **sp_helptext**、 **sp_unbindrule**

drop table

功能

从数据库中删除表定义以及其所有的数据、索引、触发器和权限。

语法

```
drop table [[database.]owner.]table_name
   [, [[database.]owner.]table_name ]...
```

关键字和选项

table_name — 是要删除的表的名称。如果该表位于另一数据库中，请指定数据库名；如果在这一数据库中有该名称的表多于一个，请指定所有者的名称。 *owner* 的缺省值是当前用户，而 *database* 的缺省值是当前数据库。

示例

```
1. drop table roysched
```

从当前数据库中删除表 *roysched* 及其数据和索引。

注释

- 在使用 **drop table** 时，表的任何规则或缺省值都会失去其绑定，而且与之相关联的任何触发器都会自动被删除。如果重新创建一个表，您必须重新绑定适当的规则和缺省值，并重新创建每一个触发器。
- 在删除表时受影响的系统表是 *sysobjects*、 *syscolumns*、 *sysindexes*、 *sysprotects* 和 *syscomments*。
- 如果启用了组件集成服务，而且要删除的表是用 **create existing table** 创建的，那么该表就不能从远程服务器删除。相反， Adaptive Server 会从系统表中删除对表的引用。

限制

- 不能对系统表使用 **drop table** 命令。
- 一旦对表进行了划分，就不能删除它。必须使用 **alter table** 命令的 **unpartition** 子句后，才能发出 **drop table** 命令。

- 只要您是表的所有者，就可以删除任一数据库中的表。例如，要删除数据库 *otherdb* 中名为 *newtable* 的表：

  ```
  drop table otherdb..newtable
  ```

  或：

  ```
  drop table otherdb.yourname.newtable
  ```

- 如果使用 **delete** 来删除表中的所有行或使用 **truncate table** 命令，在您 **drop** 它之前该表仍然存在。

删除具有跨数据库的参照完整性约束的表

- 在创建跨数据库约束时， Adaptive Server 将下列信息存储到每一个数据库中的 *sysreferences* 系统表中：

表 6-21：所存储的有关参照完整性约束的信息

存储在 *sysreferences* 中的信息	拥有所引用表信息的列	拥有引用表信息的列
键列 ID	*refkey1* 至 *refkey16*	*fokey1* 至 *fokey16*
表 ID	*reftabid*	*tableid*
数据库名	*pmrydbname*	*frgndbname*

- 因为引用表依赖来自于被引用表的信息，所以 Adaptive Server 不允许：
  - 删除被引用表，
  - 删除包含它的外部数据库，或
  - 用 **sp_renamedb** 重命名任一数据库。

  请使用 **sp_helpconstraint** 系统过程来确定哪些表引用了您要删除的表。在重新发出 **drop table** 命令之前，请使用 **alter table** 来删除约束。
- 您可以删除参考表或其数据库，不会有任何问题。 Adaptive Server 自动从所引用的数据库中删除外键信息。
- 在每次添加或对跨数据库约束时，或者是删除含有跨数据库约束的表时，请转储**所有**受影响的数据库。

◆ ***警告！***

装载这些数据库的早期转储将引起数据库损坏。有关用跨数据库的参照完整性约束来装载数据库的详细信息，参见*系统管理指南*。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

drop table 权限缺省为表的所有者所拥有，且不能移交。

参见

命令	alter table、 create table、 delete、 truncate table
系统过程	sp_depends、 sp_help、 sp_spaceused

drop trigger

功能

删除触发器。

语法

```
drop trigger [owner.]trigger_name
   [, [owner.]trigger_name]...
```

关键字和选项

trigger_name — 是要删除的触发器的名称。指定所有者的名称，从而删除由当前数据库中的不同用户所拥有的、同样名称的触发器。*owner* 的缺省值是当前用户。

示例

```
1. drop trigger trigger1
```

从当前数据库中删除 *trigger1*。

注释

- **drop trigger** 删除当前数据库中的触发器。
- 您无须从表中显式地删除某一触发器来为同一操作 （**insert**、**update** 或 **delete**）创建新的触发器。在表或列中，用于同一操作的每一个新触发器将覆盖先前的触发器。
- 当表被删除时， Adaptive Server 将自动删除任何与其相关联的触发器。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

drop trigger 权限缺省为触发器的所有者所拥有，且不能移交。

参见

命令	create trigger
系统过程	sp_depends、 sp_help、 sp_helptext

drop view

功能

从当前数据库中删除一个或多个视图。

语法

```
drop view [owner.]view_name [, [owner.]view_name]...
```

关键字和选项

view_name — 是要删除的视图的名称。指定所有者的姓名，从而删除由当前数据库中的不同用户所拥有的、同样名称的视图。 *owner* 的缺省值是当前用户。

示例

```
1. drop view new_price
```

从当前数据库中删除视图 *new_price*。

注释

- 在使用 **drop view** 时，视图的定义和有关它的其它信息（包括权限）都会从系统表 *sysobjects*、 *syscolumns*、 *syscomments*、 *sysdepends*、 *sysprocedures* 和 *sysprotects* 中删除。
- 当每一次该视图被引用 （例如，由另一个视图或存储过程引用）时，都会检查视图的存在性。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

drop view 权限缺省为视图的所有者所拥有，且不能移交。

参见

命令	create view
系统过程	sp_depends、 sp_help、 sp_helptext

dump database

功能

以 **load database** 可读取的形式备份整个数据库，包括事务日志。转储和装载均通过 Backup Server 执行。

语法

```
dump database database_name
   to stripe_device [at backup_server_name]
      [density = density_value,
       blocksize = number_bytes,
       capacity = number_kilobytes,
       dumpvolume = volume_name,
       file = file_name]
   [stripe on stripe_device [at backup_server_name]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]]
   [[stripe on stripe_device [at backup_server_name]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]]...]
   [with {
        density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name,
        [dismount | nodismount],
        [nounload | unload],
        retaindays = number_days,
        [noinit | init],
        notify = {client | operator_console}
        } ]
```

关键字和选项

database_name — 是数据库的名称，所复制的数据来自该数据库。可指定数据库名为文字、局部变量或存储过程参数。

to *stripe_device* — 是向其复制数据的设备。有关指定转储设备时使用何种形式的信息，参见本部分的 “指定转储设备”。

at *backup_server_name* — 是 Backup Server 的名称。当转储到缺省 Backup Server 时，不要指定该参数。只有通过网络转储到远程 Backup Server 时才指定该参数。该选项最多可指定 32 个远程 Backup Server。在网络上转储时，要指定运行在转储设备所附机器上的远程 Backup Server 的*网络名*。对于使用接口文件的平台， *backup_server_name* 必须出现在接口文件中。

density = *density_value* — 替换磁带设备的缺省密度。**只有在 OpenVMS 系统上对卷重新初始化时，才使用该选项**。有效的密度为 **800**、**1600**、**6250**、**6666**、**10000** 和 **38000**。不是所有的值对每个磁带驱动器都有效；对磁带驱动器要使用正确的密度。

blocksize = *number_bytes* — 替换转储设备的缺省块大小。块大小必须至少为一个数据库页 （大多数系统为 2048 字节），且必须为数据库页大小的整数倍。在 OpenVMS 系统上，块大小不能超过 55,296 字节。在某些转储设备上增加块大小可提高转储性能。要达到最佳性能，将块大小指定为 2 的次幂，例如， 65536、131072 或 262166。

capacity = *number_kilobytes* — 是设备往单个磁带卷上可写入的最大数据量。该容量至少为五个数据库页，并且小于设备的推荐容量。

计算容量的一般规则是使用设备厂商指定最大容量的 70%，留出 30%用于诸如记录间隔和磁带标记等方面。最大容量是驱动器上设备的容量，而不是驱动器本身的容量。此规则在多数情况下适用，但可能由于各供应商和设备的开销存在差异而不能全部适用。

对于不能可靠检测到磁带结尾标记的 UNIX 平台，必须指明可转储到磁带的千字节数。对指定为物理路径名的转储设备，**必须**提供 **capacity**。如果指定转储设备为逻辑设备名，除非另外指定其容量， Backup Server 将使用存储在 *sysdevices* 系统表中的 *size* 参数。

dumpvolume = *volume_name* — 确定指派给卷的名称。 *volume_name* 的最大长度是 6 个字符。 Backup Server 在覆盖现有转储、转储到崭新的磁带中或转储到一个内容不可识别的磁带中时，将 *volume_name* 写入 ANSI 磁带标签中。 **load database** 是检查标签的命令，如果装载了错误卷则产生错误消息。

◆ ***警告!***

确保每个磁带卷的标签与创建时相同，使操作员能装载正确的磁带。

stripe on *stripe_device* — 是额外的转储设备。最多可使用 32 个设备，包括在 **to** *stripe_device* 子句中命名的设备。 Backup Server 将数据库拆分成大约相等的几部分，并将每部分保存到一个不同的设备。在所有的设备上同时进行转储，不但可减少每个转储所需的时间，而且在转储过程中要求更少的卷变更。有关如何指定转储设备的信息，参见 “指定转储设备”。

dismount | **nodismount** — **在支持逻辑拆卸的平台 （如 OpenVMS）上**，确定是否卸下磁带。缺省情况下，在完成转储时，将卸下所有用于转储的磁带。使用 **nodismount** 可不卸下磁带，供再次进行转储或装载时使用。

nounload | **unload** — 确定完成转储后是否回绕磁带。缺省情况下，不回绕磁带，这样可用同一磁带卷继续进行转储。对要添加到多转储卷的最后一个转储文件指定 **unload**。这样，转储结束后，磁带将回绕并卸载。

retaindays = *number_days* — **在 UNIX 系统上**向磁盘转储时，指定 Backup Server 防止转储被覆盖的天数。如果想在到期之前覆盖转储， Backup Server 会在覆盖未到期卷之前请求确认。**该选项只在向磁盘转储时有意义。对于磁带转储则无意义。**

对于可以立即覆盖的转储， *number_days* 必须是正整数或 0。如果不指定 **retaindays** 值， Backup Server 将使用 **sp_configure** 设置的 **tape retention in days** 值。

noinit | **init** — 确定是否向现有转储文件添加转储或重新初始化 （覆盖）磁带卷。缺省情况下， Adaptive Server 在最后一个磁带结尾标记后面添加转储，允许向同一卷转储其它数据库。在多卷转储中，新的转储只能添加到最后一卷上。对于向磁带转储的第一个数据库，使用 **init** 来覆盖磁带的内容。

当您希望 Backup Server 在磁带配置文件中存储或更新磁带设备特征时，使用 **init**。有关详细信息，参见 *系统管理指南*。

file = *file_name* — 是转储文件名。该名称不能超过 17 个字符，而且必须遵守文件名的操作系统约定。有关详细信息，参见 “转储文件”。

notify = {**client** | **operator_console**} — 替换缺省消息的目标。

在提供操作员终端功能的操作系统上（如 OpenVMS），卷更改消息总是发送到运行 Backup Server 的机器的操作员终端。使用 **client** 将其它 Backup Server 消息传送到启动 **dump database** 的终端会话。

在不提供操作员终端功能的操作系统上（如 UNIX），消息发送到启动 **dump database** 的客户端。使用 **operator_console** 将消息传送到正在运行 Backup Server 的终端。

示例

1. 对于 OpenVMS：

```
dump database pubs2
  to "MTA0:"
```

对于 UNIX：

```
dump database pubs2
  to "/dev/nrmt0"
```

向磁带设备转储数据库 *pubs2*。如果磁带有 ANSI 磁带标签，该命令将转储添加到磁带上已有的文件上，因为未指定 **init** 选项。

2. 对于 OpenVMS：

```
dump database pubs2
        to "MTA0:" at REMOTE_BKP_SERVER
    stripe on "MTA1:"at REMOTE_BKP_SERVER
    stripe on "MTA2:"at REMOTE_BKP_SERVER
```

对于 UNIX：

```
dump database pubs2
    to "/dev/rmt4" at REMOTE_BKP_SERVER
    stripe on "/dev/nrmt5" at REMOTE_BKP_SERVER
    stripe on "/dev/nrmt0" at REMOTE_BKP_SERVER
with retaindays = 14
```

使用 REMOTE_BKP_SERVER Backup Server 转储 *pubs2* 数据库。该命令命名三个转储设备，所以 Backup Server 对每个设备的转储约为数据库的三分之一。该命令向磁带上的现有文件添加转储。在 UNIX 系统上，**retaindays** 选项指定 14 天内不能覆盖磁带。（OpenVMS 系统不使用 **retaindays** 选项，而总是创建文件的新版本。）

3. 对于 OpenVMS：

```
dump database pubs2
    to "MTA0:"
    with init
```

对于 UNIX：

```
dump database pubs2
    to "/dev/nrmt0"
    with init
```

init 选项初始化磁带卷，覆盖任何现有文件。

4. 对于 OpenVMS：

```
dump database pubs2
  to "MTA0:"
    with unload
```

对于 UNIX：

```
dump database pubs2
    to "/dev/nrmt0"
    with unload
```

完成转储时回绕转储卷。

5. 对于 OpenVMS：

```
dump database pubs2
  to "MTA0:"
    with notify = client
```

对于 UNIX：

```
dump database pubs2
    to "/dev/nrmt0"
      with notify = client
```

notify 子句将请求卷变更的 Backup Server 消息发送到启动转储请求的客户端，而不是将它们发送到缺省位置，即 Backup Server 机器的主控台。

注释

- 表 6-22 描述了用于备份数据库的命令和系统过程：

表 6-22：用于备份数据库和日志的命令

操作	使用此命令
对整个数据库进行例行转储，包括事务日志。	**dump database**
对事务日志进行例行转储，然后截断不活动的部分。	**dump transaction**
数据库设备发生故障后转储事务日志。	**dump transaction with no_truncate**
截断日志而不备份，然后复制整个数据库。	**dump transaction with truncate_only** **dump database**
因为日志空间不足导致通常方法失败后，截断日志，然后复制整个数据库。	**dump transaction with no_log** **dump database**
对 Backup Server 的卷更改消息作出反应。	**sp_volchanged**

***dump database* 限制**

- 不能从 11.x Adaptive Server 向 10.x Backup Server 转储。
- Sybase 转储和非 Sybase 数据 （例如， UNIX 档案）不能在同一磁带上保存。
- 如果数据库有跨数据库的参照完整性约束， *sysreferences* 系统表存储外部数据库的**名称** — 而不是 ID 号。如果使用 **load database** 更改数据库名称或在不同的服务器上装载数据库， Adaptive Server 将无法确保参照完整性。

◆ ***警告！***

为了用不同名称装载数据库或将其移至另一个 Adaptive Server 上，在转储数据库之前，请使用 alter table 删除所有的外部参照完整性约束。

- 在用户定义事务中不能使用 **dump database**。
- 如果在一个正在进行 **dump transaction** 的数据库上发出 **dump database** 命令， **dump database** 将休眠，直到完成事务转储。
- 使用 1/4 英寸盒式磁带时，每个磁带只能转储一个数据库或事务日志。
- 不能转储带有脱机页的数据库。要强制脱机页联机，使用 **sp_forceonline_db** 或 **sp_forceonline_page**。

安排转储

- Adaptive Server 数据库转储是**动态的** — 可在数据库活动时进行。然而，数据库转储会略微减慢系统的速度，所以最好在数据库更新负载不重时运行 **dump database**。
- **定期、经常地备份 *master* 数据库**。除了定期备份外，在发出每个 **create database**、 **alter database** 和 **disk init** 命令后，转储 *master*。
- 每次更改数据库时，备份 *model* 数据库。
- 创建数据库后，立即使用 **dump database** 复制整个数据库。只有运行了 **dump database** 后，才能在新的数据库上运行 **dump transaction**。
- 在每次添加或删除跨数据库约束时，或者是删除含有跨数据库约束的表时，请转储**所有**受影响的数据库。

◆ ***警告!***

装载这些数据库的早期转储将引起数据库损坏。

- 安排定期备份用户的数据库和事务日志。
- 使用阈值使备份过程自动化。为了利用 Adaptive Server 的最后机会阈值，可在创建用户数据库时将日志段放在与数据段分离的设备上。有关阈值的详细信息，参见*系统管理指南*。

转储系统数据库

- *master*、*model* 和 *sybsystemprocs* 数据库的事务日志不单独占用段。使用 **dump transaction with truncate_only** 来清除日志，然后使用 **dump database** 备份数据库。
- 当出现影响 *master* 数据库的故障时，执行恢复过程时需要使用 *master* 数据库的备份。有关备份和恢复 *master* 数据库的逐步说明，参见*系统管理指南*。
- 如果使用活动介质备份，整个 *master* 数据库必须装在单个卷中，除非有另一个 Adaptive Server 可对卷更改消息作出反应。

指定转储设备

- 可以将转储设备以文字、局部变量或参数形式指定给一个存储过程。
- 不能向空设备 （UNIX 上的 */dev/null*； OpenVMS 上任何以“NL”开头的设备名）转储。
- 磁带和磁盘设备支持多分条转储。在一个设备上进行多个转储则只能用于磁带设备。

- 可指定本地转储设备为：
  - *sysdevices* 系统表中的一个逻辑设备名
  - 绝对路径名
  - 相对路径名

  Backup Server 使用 Adaptive Server 的当前工作目录解析相对路径名。
- 在网络上转储时，必须指定转储设备的绝对路径名。路径名必须在 Backup Server 运行的机器上有效。如果名称包括任何非字母、数字或下划线 (_) 的字符，都必须用引号将它引起来。
- 转储设备的所有权和权限问题会干扰 **dump** 命令的使用。**sp_addumpdevice** 过程往系统表添加设备，但并不代表可以向该设备转储或创建作为转储设备的文件。
- 只要每个转储所用的转储设备不同，您可以同时运行多个转储（或装载）。
- 如果设备文件已经存在， Backup Server 将其覆盖；而不是将其截断。例如，假设向一个设备文件转储数据库后，设备文件的大小为 10MB。如果该设备的下一个数据库转储小于 10MB，则设备文件的大小仍为 10MB。

确定磁带设备特征

- 如果发出不带 **init** 限定符的 **dump** 命令，且 Backup Server 不能确定设备类型，则 **dump** 命令失败。有关详细信息，参见*系统管理指南*。

Backup Server

- 必须在与 Adaptive Server 相同的计算机上运行 Backup Server。（在 OpenVMS 系统上， Backup Server 可在与 Adaptive Server 相同的集群中运行，只要所有的数据库设备对二者都可见。）Backup Server 必须在 *master..sysservers* 表中列出。该条目在安装或升级时创建，不能删除。
- 如果备份设备位于另一台计算机上，需要在网络上转储，则远程计算机上必须也装有 Backup Server。

转储文件

- 用 **init** 选项转储数据库会覆盖磁带或磁盘上任何现有的文件。
- Backup Server 向 **with notify** 子句指定的位置发送转储文件名。存储备份磁带前，操作员应用数据库名、文件名、日期和其它有关信息对其进行标记。装载未标识标签的磁带时，使用 **with headeronly** 和 **with listonly** 选项来确定内容。

文件名和档案名

- 转储文件的名称标识了所转储的数据库和转储的时间。然而，下面语法中：

  file = *file_name*

  file_name 的含义不同，取决于向磁盘还是向 UNIX 磁带转储。

  向磁盘转储，磁盘文件的路径名就是它的文件名。

  向 UNIX 磁带转储，路径名不是文件名。文件交换的 ANSI 标准格式包含 HDR1 标签中的文件名域。对于遵循 ANSI 规范的磁带，标签中的该域标识了文件名。 ANSI 规范仅适用于磁带的标签；不适用于磁盘文件。

  这会产生两个问题：

  - UNIX 不遵循关于磁带文件名的 ANSI 协议。 UNIX 认为磁带的数据未被标识。虽然可将其分成多个文件，但是这些文件没有名称。
  - 在 Backup Server 中， ANSI 磁带标签用于存储有关档案的信息，否定了 ANSI 的含义。因此，磁盘文件也有 ANSI 标签，因为那里存储有档案名。

  文件名的含义随所执行转储的类型不同而变化。例如，下列语法中：

```
dump database database_name to 'filename' with file='filename'
```

  - 第一个 *filename* 指输入的显示文件的路径名。
  - 第二个 *filename* 实际上是档案名，该名称存储在档案的 HDR1 标签中，用户可用 **dump** 或 **load** 命令的 **file=*filename*** 参数来指定档案名。

  指定档案名后，数据库装载过程中服务器使用该名称来定位所选的档案。

  如果未指定档案名，服务器装载它遇到的第一个档案。

  两种情况下， file ='archivename' 确定存储在 HDR1 标签中的名称，后续 LOAD 也将使用该名称来验证是否正在查看正确的数据。

如果未指定名称， DUMP 将编造一个名称而 LOAD 将使用其遇到的第一个名称。

to '*filename*' 子句中 *filename* 的含义根据转储为磁盘转储还是磁带转储而变化：

- 如果转储到磁带，则 'filename' 是磁带设备的名称；
- 如果转储到磁盘，则 'filename' 是磁盘文件的名称。

如果进行磁盘转储且 'filename' 不是完整路径，则需要利用服务器的当前工作目录来修改 'filename'。

- 如果转储到磁带且未指定文件名，则 Backup Server 通过并置下列内容来创建一个缺省文件名：
  - 数据库名的最后 7 个字符
  - 两位的年份数字
  - 每年的第几天 （以 3 位数表示， 1-366）
  - 转储文件创建时的十六进制编码时间

  例如，文件 *cations980590E100* 包含 1998 年第 59 天创建的 *publications* 数据库的副本：

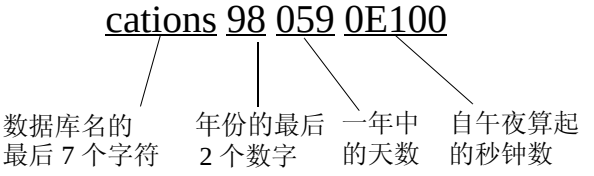


图 6-2：数据库转储到磁带的文件命名协议

卷名

- 转储卷依照 ANSI 磁带标签标准进行标记。标签包括逻辑卷数和分条集内的设备位置。
- 装载时，Backup Server 使用磁带标签来验证卷是否按正确的顺序装入。由此可从少于转储时使用的设备进行装载。

➤ ***注意***

在网络上转储并装载时，对每一个操作必须指定相同数量的分条设备。

更改转储卷

- 在 OpenVMS 系统中，操作系统在检测到卷尾或指定的驱动器脱机时，请求更改卷。装入另一个卷后，操作员使用 **REPLY** 命令答复这些消息。
- 在 UNIX 系统中， Backup Server 在磁带容量满后请求更改卷。装入另一个卷后，操作员通过在任何可与 Backup Server 通信的 Adaptive Server 上执行 **sp_volchanged** 系统过程来通知 Backup Server。
- 如果 Backup Server 检测到当前装入的卷有问题，它通过向客户端或其操作员主控台发送消息来请求改变卷。操作员用 **sp_volchanged** 系统过程对这些消息作出反应。

添加或覆盖卷

- 缺省情况下 **(noinit)**， Backup Server 将连续转储写入同一磁带卷，充分利用高容量的磁带介质。数据添加到最后一个磁带结尾标记后面。在多卷转储中，新的转储只能添加到最后一卷上。向磁带写入数据前， Backup Server 会验证，确保第一个文件尚未到期。如果磁带包含非 Sybase 数据， Backup Server 会拒绝它，以避免可能对重要信息造成破坏。
- 使用 **init** 选项重新初始化卷。如果指定 **init**， Backup Server 将覆盖任何现有内容，即使磁带包含非 Sybase 数据、第一个文件未到期或者磁带具有 ANSI 访问限制。
- 图 6-3 说明了如何使用下列命令向单个卷转储三个数据库：
  - **init** 为首次转储初始化磁带
  - **noinit** （缺省）添加后续转储
  - **unload** 最后的转储后回绕和卸载磁带。

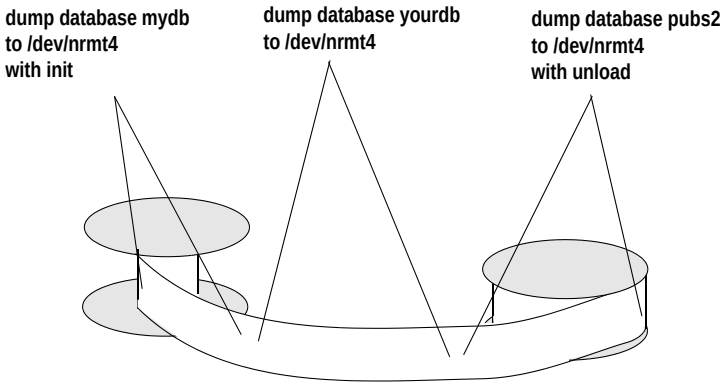


图 6-3：将几个数据库转储到同一卷

转储数据库，其设备已镜像

- **dump database** 开始时， Adaptive Server 向 Backup Server 传递所有数据库和日志设备的主设备名。如果主设备未镜像， Adaptive Server 则传递辅助设备的名称。如果在 Backup Server 完成数据移交前任何已命名设备发生故障， Adaptive Server 中止转储。
- **dump database** 正在进行时，如果用户试图对任何已命名数据库设备解除镜像， Adaptive Server 显示一条消息。执行 **disk unmirror** 命令的用户可中止转储或延迟 **disk unmirror** 到转储完成后。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

只有系统管理员、数据库所有者和具有操作员角色的用户才能执行 **dump database**。

参见

命令	**dump transaction**、 **load database**、 **load transaction**
系统过程	**sp_addthreshold**、 **sp_addumpdevice**、 **sp_dropdevice**、 **sp_dropthreshold**、 **sp_helpdevice**、 **sp_helpdb**、 **sp_helpthreshold**、 **sp_logdevice**、 **sp_spaceused**、 **sp_volchanged**

dump transaction

功能

复制事务日志并删除不活动的部分。

语法

进行例行日志转储：

```
dump tran[saction] database_name
   to stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]
   [stripe on stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]]
   [[stripe on stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name] ]...]
   [with {
       density = density_value,
       blocksize = number_bytes,
       capacity = number_kilobytes,
       dumpvolume = volume_name,
       file = file_name,
       [dismount | nodismount],
       [nounload | unload],
       retaindays = number_days,
       [noinit | init],
       notify = {client | operator_console},
       standby_access }]
```

截断日志而不形成备份副本：

```
dump tran[saction] database_name
   with truncate_only
```

截断已填充至其容量的日志。**仅作为最后一种解决方法：**

```
dump tran[saction] database_name
   with no_log
```

数据库设备发生故障后备份日志：

```
dump tran[saction] database_name
   to stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]
   [stripe on stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name]]
   [[stripe on stripe_device [ at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        capacity = number_kilobytes,
        dumpvolume = volume_name,
        file = file_name] ]...]
   [with {
       density = density_value,
       blocksize = number_bytes,
       capacity = number_kilobytes,
       dumpvolume = volume_name,
       file = file_name,
       [dismount | nodismount],
       [nounload | unload],
       retaindays = number_days,
       [noinit | init],
       no_truncate,
       notify = {client | operator_console}}]
```

关键字和选项

database_name — 是数据库的名称，所复制的数据来自该数据库。名称可以是文字、局部变量或存储过程的参数。

truncate_only — 删除日志的不活动部分，**而不形成备份副本**。数据库的日志段没有放在与数据段不同的设备上时使用，不要指定转储设备或 Backup Server 名称。

no_log — 删除日志的不活动部分，**不形成备份副本，也不在事务日志中记录过程**。只在用尽了日志空间且不能运行通常的 **dump transaction** 命令时，使用 **no_log**。将 **no_log** 作为最后一种解决方法，并仅在 **dump transaction with truncate_only** 失败后使用此命令一次。有关其他信息，参见*系统管理指南*。

to *stripe_device* — 是向其转储数据的设备。有关指定转储设备时使用何种形式的信息，参见 “指定转储设备”。

at *backup_server_name* — Backup Server 的名称。当转储到缺省 Backup Server 时，不要指定该参数。只有通过网络转储到远程 Backup Server 时才指定该参数。该选项最多可指定 32 个不同的远程 Backup Server。在网络上转储时，指定远程 Backup Server 的*网络名*，该服务器在附加了转储设备的机器上运行。对于使用接口文件的平台， *backup_server_name* 必须出现在接口文件中。

density = *density_value* — 替换磁带设备的缺省密度。**只有在 OpenVMS 系统上对卷重新初始化时，才使用该选项**。有效的密度为 **800**、**1600**、**6250**、**6666**、**10000** 和 **38000**。不是所有的值对每个磁带驱动器都有效；对磁带驱动器要使用正确的密度。

blocksize = *number_bytes* — 替换转储设备的缺省块大小。（**可能的话，使用缺省块大小**；这是系统最好的块大小。）块大小必须至少为一个数据库页 （大多数系统为 2048 字节），且必须为数据库页大小的整数倍。在 OpenVMS 系统上，块大小不能超过 55,296 字节。

capacity = *number_kilobytes* — 是设备往单个磁带卷上可写入的最大数据量。该容量至少为五个数据库页，并且略小于设备的推荐容量。

计算容量的一般规则是使用设备厂商指定最大容量的 70%，留出 30%用于诸如记录间隔和磁带标记等方面。此规则在多数情况下适用，但可能由于各供应商和设备的开销存在差异而不能全部适用。

OpenVMS 系统一直写入，直至达到物理磁带结尾标记，此时系统发送一个卷更改请求。

对于不能可靠检测到磁带结尾标记的 UNIX 平台，必须指明可转储到磁带的千字节数。对指定为物理路径名的转储设备，**必须**提供 **capacity**。如果指定转储设备为逻辑设备名，除非另外指定其容量， Backup Server 将使用存储在 *sysdevices* 系统表中的 *size* 参数。

dumpvolume = *volume_name* — 确定指派给卷的名称。 *volume_name* 的最大长度是 6 个字符。 Backup Server 在覆盖现有转储、转储到崭新的磁带中或转储到一个内容不可识别的磁带中时，将 *volume_name* 写入 ANSI 磁带标签中。 **load transaction** 是检查标签的命令，如果装载了错误卷则产生错误消息。

stripe on *stripe_device* — 是额外的转储设备。最多可使用 32 个设备，包括在 **to** *stripe_device* 子句中命名的设备。 Backup Server 将日志拆分成大约相等的几部分，并将每部分保存到一个不同的设备。在所有的设备上同时进行转储，可减少需要的时间和卷更改的数量。有关如何指定转储设备的信息，参见 “指定转储设备”。

dismount | **nodismount** — **在支持逻辑拆卸的平台 （如 OpenVMS）上**，确定是否卸下磁带。缺省情况下，在完成转储时，将卸下所有用于转储的磁带。使用 **nodismount** 可不卸下磁带，供再次进行转储或装载时使用。

nounload | **unload** — 确定完成转储后是否回绕磁带。缺省情况下，不回绕磁带，这样可用同一磁带卷继续进行转储。对要添加到多转储卷的最后一个转储文件指定 **unload**。这样，转储结束后，磁带将回绕并卸载。

retaindays = *number_days* — **在 UNIX 系统上**，指定 Backup Server 防止转储被覆盖的天数。**该选项对磁盘、 1/4 英寸盒带以及单文件介质有意义。在多文件介质上，该选项对除了第一卷以外的所有卷有意义**。如果想在到期之前覆盖转储， Backup Server 会在覆盖未到期卷之前请求确认。

对于要立即覆盖的转储，*number_days* 必须是正整数或 0。如果不指定 **retaindays** 值， Backup Server 将使用 **sp_configure** 设置的全服务器范围的 **tape retention in days** 值。

noinit | **init** — 确定是否向现有转储文件添加转储或重新初始化 （覆盖）磁带卷。缺省情况下， Adaptive Server 在最后一个磁带结尾标记后面添加转储，允许向同一卷转储其它数据库。在多卷转储中，新的转储只能添加到最后一卷上。对于向磁带转储的第一个数据库，使用 **init** 来覆盖磁带的内容。

当您希望 Backup Server 在磁带配置文件中存储或升级磁带设备特征时，使用 **init**。有关详细信息，参见 *系统管理指南*。

file = *file_name* — 是转储文件的名称。该名称不能超过 17 个字符，而且必须遵守文件名的操作系统约定。如果不指定文件名，Backup Server 将创建缺省文件名。有关详细信息，参见 “转储文件”。

no_truncate — 使用指向 *master* 数据库中事务日志的指针转储事务日志，**即使无法访问包含数据库数据段的磁盘**仍可进行转储。当事务日志驻留在未破坏的设备上，而且 *master* 数据库和用户数据库驻留在不同的物理设备上时，**with no_truncate** 选项可提供以分钟计算的日志恢复水准。

notify = {**client** | **operator_console**} — 替换缺省消息的目标。

- 在提供操作员终端功能的操作系统上（如 OpenVMS），卷更改消息总是发送到运行 Backup Server 的计算机的操作员终端上。使用 **client** 将其它 Backup Server 消息传送到启动 **dump database** 的终端会话。
- 在不提供操作员终端功能的操作系统上（如 UNIX），消息发送到启动 **dump database** 的客户端。使用 **operator_console** 将消息传送到正在运行 Backup Server 的终端。

with standby_access — 指定只能转储已完成的事务。转储持续进行，一直到所能找到的以下最远位置：在该点上事务刚好完成且没有其它活动的事务。

示例

1. 对于 **UNIX**：

```
dump transaction pubs2
  to "/dev/nrmt0"
```

对于 **OpenVMS**：

```
dump database pubs2
  to "MTA0:"
```

向磁带转储事务日志，将其添加到磁带的文件中，因为未指定 **init** 选项。

2. 对于 **UNIX**：

```
dump transaction mydb
    to "/dev/nrmt4" at REMOTE_BKP_SERVER
    stripe on "/dev/nrmt5" at REMOTE_BKP_SERVER
with init, retaindays = 14
```

对于 **OpenVMS**：

```
dump transaction mydb
    to "MTA0:"at REMOTE_BKP_SERVER
    stripe on "MTA1:"at REMOTE_BKP_SERVER
with init
```

转储 *mydb* 数据库的事务日志，使用 Backup Server REMOTE_BKP_SERVER。Backup Server 向两种设备中的每一种都转储大约一半的日志。**init** 选项覆盖磁带上的任何现有文件。在 UNIX 系统上，**retaindays** 选项指定 14 天内不能覆盖磁带。

（OpenVMS 系统不使用 **retaindays**；它们总是创建转储文件的新版本。）

3. **dump tran inventory_db to dev1 with standby_access**

   将已完成的事务从 *inventory_db* 事务日志文件转储到设备 *dev1*。

注释

- 表 6-23 描述了用于备份数据库和日志的命令和系统过程：

表 6-23：用于备份数据库和日志的命令

使用此命令	操作
dump database	对整个数据库进行例行转储，包括事务日志。
dump transaction	对事务日志进行例行转储，然后截断不活动的部分。
dump transaction with no_truncate	数据库设备发生故障后转储事务日志。
dump transaction with truncate_only then **dump database**	截断日志而不备份。 复制整个数据库。
dump transaction with no_log then **dump database**	常用方法因日志空间不足而失败后，将截断日志 复制整个数据库。
sp_volchanged	对 Backup Server 的卷更改消息作出反应。

限制

- 不能向空设备（UNIX 上的 */dev/null*；OpenVMS 上任何以“NL”开头的设备名）转储。
- 事务中不能使用 **dump transaction** 命令。
- 使用 1/4 英寸盒式磁带时，每个磁带只能转储一个数据库或事务日志。
- 启用 **trunc log on chkpt** 数据库选项或在启用 **select into/bulk copy/pllsort** 后用 **select into**、快速批量复制操作、缺省未记录的 **writetext** 操作或一个并行排序对数据库进行最低程度的记录更改时，不能发出转储事务日志命令。而应使用 **dump database**。

◆ ***警告！***

不要用 *delete*、*update* 或 *insert* 命令修改日志表 *syslogs*。

- 如果数据库没有与数据段位于不同设备上的日志段，则不能使用 **dump transaction** 复制日志并截断日志。
- 如果用户或阈值过程对正在进行 **dump database** 或另一个 **dump transaction** 的数据库上发出 **dump transaction** 命令，第二个命令将休眠直到第一个命令完成。
- 要恢复数据库，使用 **load database** 装载最近的数据库转储；然后使用 **load transaction** **按其形成的先后顺序**装载每一个后续事务日志转储。
- 在每次添加或删除跨数据库约束时，或者是删除含有跨数据库约束的表时，要转储**所有**受影响的数据库。

◆ ***警告！***

装载这些数据库的早期转储将引起数据库损坏。

- 不能从 11.x Adaptive Server 向 10.x Backup Server 转储。
- Sybase 转储和非 Sybase 数据 （例如， UNIX 档案）不能在同一磁带上保存。
- 如果数据库有脱机页，不能使用 **with no_log** 或 **with truncate_only** 转储事务。

设备故障后复制日志：*with no_truncate*

- 设备出故障后，使用 **dump transaction with no_truncate** 复制日志，而不会截断日志。只有当日志在单独的段上并且 *master* 数据库可访问时，才能使用该选项。
- **dump transaction with no_truncate** 创建的备份是日志的最新转储。恢复数据库时，最后装载该转储。

没有单独日志段的数据库：*with truncate_only*

- 当数据库没有与数据段位于不同设备上的日志段时，使用 **dump transaction with truncate_only** 从日志中删除已提交事务，而不形成备份副本。

◆ ***警告！***

dump transaction with truncate_only 不提供恢复数据库的方法。尽早运行 dump database 以确保数据库可以恢复。

- 在 *master*、 *model* 和 *sybsystemprocs* 数据库上使用 **with truncate_only**（这些数据库没有与数据段位于不同设备上的日志段）。
- 对于在同一设备上存储事务日志和数据的较小数据库，也可以使用该选项。
- 关键的用户数据库应具有与数据段位于不同设备上的日志段。使用 **create database** 的 **log on** 子句创建带有单独日志段的数据库，或使用 **alter database** 和 **sp_logdevice** 向不同的设备移交日志。

仅转储完成的事务：*with standby_access*

- 使用 **with standby_access** 选项转储事务日志，以将其装入作为数据库热备份服务器的服务器中。
- 使用 **with standby_access** 转储事务日志时，转储继续进行到日志中的最远点，在该点，所有的早期事务都已完成，且没有记录属于打开的事务。
- 如果要依次装入两个或多个事务日志并且想要数据库在装载间隔中联机，则必须使用 **dump tran[saction]...with standby_access**。
- 装载完使用 **with standby_access** 选项进行的转储后，使用带有 **for standby_access** 选项的 **online database** 命令以使数据库可被访问。

◆ ***警告！***

如果事务日志含有打开的事务并且没有使用 *with standby_access* 选项来转储它，则版本 11.9.2 不允许装载此日志、使数据库联机以及装载后续事务转储。如果计划装载一系列事务转储，那么只有装载完一个原来使用 with standby_access 选项转储的事务或装载完整个系列事务后，才可以使数据库联机。

所有其它方法失败时：*with no_log*

- 仅在由日志空间不足导致的转储事务日志的通常方法 （**dump transaction** 或 **dump transaction with truncate_only**）失败后，使用 **dump transaction with no_log** 作为最后一种解决方法。
- **dump transaction...with no_log** 截断日志，而不记录转储事务的事件。因为它不复制数据，只需要数据库名称。
- 每次使用 **dump transaction...with no_log** 都会被认为是一个错误，并被记录在 Adaptive Server 的错误日志中。

◆ ***警告！***

dump transaction with no_log 不提供恢复数据库的方法。尽早运行 dump database 以确保数据库可以恢复。

- 如果创建数据库时日志段所在的设备与数据段不同，又编写了一个足够频繁地转储事务日志的最后机会阈值过程，并为日志和数据库分配了足够的空间，则应当不会需要使用此选项。必须使用 **with no_log** 时，增加转储的频率和日志空间的数量。

安排转储

- 事务日志转储是**动态的** — 可在数据库活动时进行。事务日志转储会略微减慢系统的速度，所以最好在数据库更新负载不重时运行。
- 创建数据库后，立即使用 **dump database** 复制整个数据库。只有运行了 **dump database** 后，才能在新的数据库上运行 **dump transaction**。
- 安排定期备份用户的数据库和事务日志。
- **dump transaction** 与 **dump database** 相比，使用更小的存储空间和更少的时间。一般来说，事务日志转储比数据库转储更频繁。

用阈值使 *dump transaction* 自动化

- 使用阈值使备份过程自动化。为了利用 Adaptive Server 的最后机会阈值，可在创建用户数据库时将日志段放在与数据段分离的设备上。
- 当日志段上的空间降至最后机会阈值以下， Adaptive Server 执行最后机会阈值过程。在最后机会阈值过程中包括 **dump transaction** 命令有助于防止用尽日志空间。有关详细信息，参见 **sp_thresholdaction**。
- 可使用 **sp_addthreshold** 向监控日志空间添加第二个阈值。有关阈值的详细信息，参见*系统管理指南*。

指定转储设备

- 可以将转储设备以文字、局部变量或参数形式指定给一个存储过程。
- 可指定本地转储设备为：
  - *sysdevices* 系统表中的一个逻辑设备名
  - 绝对路径名
  - 相对路径名

  Backup Server 使用 Adaptive Server 的当前工作目录解析相对路径名。
- 磁带和磁盘设备支持多分条转储。在一个设备上进行多个转储则只能用于磁带设备。
- 在网络上转储时，指定转储设备的绝对路径名。路径名必须在 Backup Server 运行的机器上有效。如果名称包括任何非字母、数字或下划线 (_) 的字符，用引号将它引起来。
- 转储设备的所有权和权限问题会干扰 **dump** 命令的使用。**sp_addumpdevice** 过程往系统表添加设备，但并不代表可以向该设备转储或创建作为转储设备的文件。
- 可同时运行多个转储 （或装载），只要它们使用不同的转储设备。

确定磁带设备特征

- 如果发出不带 **init** 限定符的 **dump transaction** 命令，而且 Backup Server 不能确定设备类型， **dump transaction** 命令失败。有关详细信息，参见*系统管理指南*。

Backup Server

- 必须在同一计算机上运行 Backup Server 和 Adaptive Server。（在 OpenVMS 系统上， Backup Server 可在与 Adaptive Server 相同的集群中运行，只要所有的数据库设备对二者都可见。）Backup Server 必须在 *master..sysservers* 表中列出。该条目在安装或升级时创建，且不能删除。
- 如果备份设备位于另一台机器以至于要在网络上转储，远程机器上必须也有 Backup Server。

转储文件

- 用 **init** 选项转储日志会覆盖磁带或磁盘上任何现有的文件。
- 转储文件名称标识出转储的数据库和转储时间。如果未指定文件名， Backup Server 通过并置下列内容来创建一个缺省文件名：
  - 数据库名的最后 7 个字符
  - 两位的年份数字
  - 每年的第几天 （以 3 位数表示， 1– 366）
  - 转储文件创建时的十六进制编码时间

  例如，文件 *cations930590E100* 包含 1993 年第 59 天创建的 *publications* 数据库的副本：

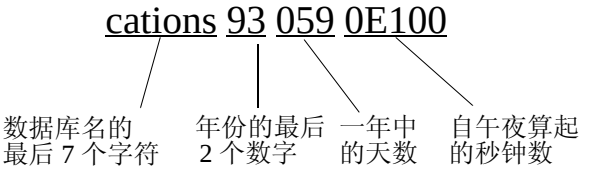


图 6-4：事务日志转储的文件命名协议

- Backup Server 向 **with notify** 子句指定的位置发送转储文件名。存储备份磁带前，操作员应用数据库名、文件名、日期和其它有关信息对其进行标记。装载未标识标签的磁带时，使用 **with headeronly** 和 **with listonly** 选项来确定内容。

卷名

- 转储卷依照 ANSI 磁带标签标准而标记。标签包括逻辑卷数和分条集内的设备位置。
- 装载时， Backup Server 使用磁带标签来验证卷是否按正确的顺序装入。由此可从少于转储时使用的设备进行装载。

➤ ***注意***

在网络上转储并装载时，对每一个操作必须指定相同的分条设备数。

更改转储卷

- 在 OpenVMS 系统中，操作系统在检测到卷尾或指定的驱动器脱机时，请求更改卷。装入另一个卷后，操作员使用 **REPLY** 命令答复这些消息。
- 在 UNIX 系统中， Backup Server 在磁带容量满后请求改变卷。装入另一个卷后，操作员通过在任何可与 Backup Server 通信的 Adaptive Server 上执行 **sp_volchanged** 系统过程来通知 Backup Server。
- 如果 Backup Server 检测到当前装入的卷有问题 （例如，如果装入了错误的卷），它通过向客户端或其操作员主控台发送消息来请求卷更改。操作员用 **sp_volchanged** 系统过程对这些消息作出反应。

添加/覆盖卷

- 缺省情况下 (**noinit**)， Backup Server 将连续转储写入同一磁带卷，充分利用高容量的磁带介质。数据添加到最后一个磁带结尾标记后面。在多卷转储中，新的转储只能添加到最后一卷上。向磁带写入数据前， Backup Server 会验证，确保第一个文件尚未到期。如果磁带包含非 Sybase 数据， Backup Server 会拒绝它，以避免可能对重要信息造成破坏。
- 使用 **init** 选项重新初始化卷。如果指定 **init**， Backup Server 将覆盖任何现有内容，即使磁带包含非 Sybase 数据、第一个文件未到期或者磁带具有 ANSI 访问限制。
- 图 6-5 说明了如何向单个卷转储三个事务日志。使用：
  - **init** 为首次转储初始化磁带
  - **noinit** （缺省）添加后续转储
  - **unload** 最后的转储后回绕和卸载磁带。

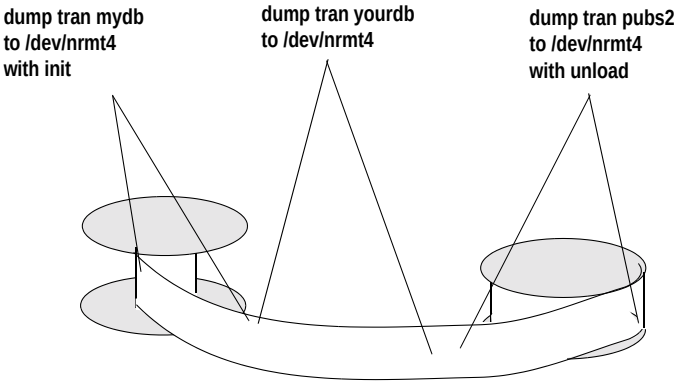


图 6-5：向单个卷转储三个事务日志

转储存储在已镜像设备中的日志

- **dump transaction** 开始时， Adaptive Server 向 Backup Server 传递每个逻辑日志设备的主设备名。如果主设备未镜像， Adaptive Server 则传递辅助设备的名称。如果在 Backup Server 完成数据移交前已命名设备发生故障， Adaptive Server 中止转储。
- 如果当 **dump transaction** 正在进行时试图对已命名的日志设备解除镜像， Adaptive Server 会显示一条消息。执行 **disk unmirror** 命令的用户可中止转储或延迟 **disk unmirror** 到转储完成后。
- **dump transaction with truncate_only** 和 **dump transaction with no_log** 不使用 Backup Server。当日志设备因设备故障或 **disk unmirror** 命令未被镜像时，这些命令不受影响。
- **dump transaction** 仅复制日志段。当仅数据设备因设备故障或 **disk unmirror** 命令未被镜像时，它不受影响。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

只有系统管理员、获得操作员角色的用户和数据库所有者才能执行 **dump transaction**。

参见

命令	**dump database**、 **load database**、 **load transaction**、 **online database**
系统过程	**sp_addumpdevice**、 **sp_dboption**、 **sp_dropdevice**、 **sp_helpdevice**、 **sp_logdevice**、 **sp_volchanged**

execute

功能

运行过程或动态执行 Transact-SQL 命令

语法

```
[exec[ute]] [@return_status = ]
   [[[server.]database.]owner.]procedure_name[;number]
       [[@parameter_name =] value |
           [@parameter_name =] @variable [output]
       [,[@parameter_name =] value |
           [@parameter_name =] @variable [output]...]]
   [with recompile]
```

或

```
exec[ute] ("string" | char_variable
   [+ "string" | char_variable]...)
```

关键字和选项

execute | exec — 用于执行存储过程或扩展存储过程 (ESP)。只有当存储过程调用**不是**批处理中的第一个语句时，它才是必需的。

@return_status — 是一个可选的整数变量，它存储了存储过程的返回状态。在执行语句中使用该变量之前，必须在批处理或存储过程中将其声明。

server — 是远程服务器的名称。要在另一 Adaptive Server 上执行过程，需要拥有使用该服务器以及在指定数据库中执行过程的权限。如果指定了服务器名而没有指定数据库名， Adaptive Server 就将在缺省数据库中搜索过程。

database — 是数据库的名称。如果该过程位于其它数据库，就需指定数据库名。 *database* 的缺省值是当前数据库。只要您是另一数据库的所有者或拥有在该数据库中执行过程的权限，就可以在该数据库中执行过程。

owner — 是过程所有者的名称。如果数据库中有多个具有该名称的过程，就需指定所有者的名称。 *owner* 的缺省值是当前用户。只有当数据库所有者 （“dbo”）拥有该过程或者是您自己拥有它时，所有者的名称才是可选的。

procedure_name — 是用 **create procedure** 语句定义的过程的名称。

;number — 是可选整数，它用于将同名过程进行分组，以便可以使用一个 **drop procedure** 语句将其一并删除。在同一应用程序中使用的过程通常以此方法分组。例如，如果与名为 **orders** 的应用程序一起使用的过程被命名为 *orderproc;1*、 *orderproc;2* 等，则语句：

```
drop proc orderproc
```

将删除整个组。过程一旦被分组，就不能单独删除组中的过程。例如，不能执行该语句：

```
drop procedure orderproc;2
```

parameter_name — 是过程的参数名，它在 **create procedure** 语句中定义。参数名前面必须带有 @ 符号。

如果使用 “*@parameter_name = value*”的形式，参数名和常量就不必按照 **create procedure** 语句中定义的顺序来提供。但是，如果将该形式用于任何参数，则必须将它用于所有的后续参数。

value — 是过程的参数值或自变量值。如果不使用 “*@parameter_name = value*”形式，则必须按照 **create procedure** 语句中定义的顺序来提供参数值。

@variable — 是用来存储返回参数的变量的名称。

output — 表示存储过程将返回一个返回参数。存储过程中的匹配参数必须也用 **output** 关键字来创建。

output 关键字可缩写为 **out**。

with recompile — 强制编译新计划。如果所提供的参数是不规则的，或者数据经过了较大更改，请使用该选项。更改后的计划将用于后续的执行。当执行 ESP 时， Adaptive Server 将忽略该选项。

string — 是文字字符串，它包含部分要执行的 Transact-SQL 命令。通过文字字符串提供的字符数不存在限制。

char_variable — 是提供 Transact-SQL 命令文本的变量的名称。*char_variable* 最多可以提供 255 个字符。

示例

```
1. execute showind titles
```

或：

```
exec showind @tabname = titles
```

或者，如果它是批处理或文件中唯一的语句：

```
showind titles
```

以上三个示例用参数值 *titles* 来执行存储过程 *showind*。

2.
```
declare @retstat int
execute @retstat = GATEWAY.pubs.dbo.checkcontract
"409-56-4008"
```

   在远程服务器 GATEWAY 上执行存储过程 *checkcontract*。将指示成功或失败的返回状态存储到 *@retstat* 中。

3.
```
declare @percent int
select @percent = 10
execute roy_check "BU1032", 1050, @pc = @percent
output
select Percent = @percent
```

   执行存储过程 *roy_check*，并传递三个参数。第三个参数 *@pc* 是一个 **output** 参数。执行完该过程之后，即可在变量 *@percent* 中得到返回值。

4.
```
create procedure
showsysind @table varchar(30) = "sys%"
as
  select sysobjects.name, sysindexes.name, indid
  from sysindexes, sysobjects
  where sysobjects.name like @table
  and sysobjects.id = sysindexes.id
```

   如果用户没有提供参数，该过程将显示有关系统表的信息。

5.
```
declare @input varchar(12)
select @input="Hello World!"
execute xp_echo @in = @input, @out= @result output
```

   执行扩展存储过程 *xp_echo*，并传入“Hello World!”值。扩展存储过程的返回值存储在名为 *result* 的变量中。

6.
```
declare @tablename char(20)
declare @columname char(20)
select @tablename="sysobjects
select @columname="name"
execute ('select ' + @columname + ' from ' +
@tablename + ' where id=3')
```

   最终的 **execute** 命令并置了字符串值和字符变量，以发出 Transact-SQL 命令：

   ```
   select name from sysobjects where id=3
   ```

注释

- 根据执行时所在的数据库，过程结果可能会有所不同。例如，用户定义的系统过程 *sp_foo*（执行 **db_name()** 系统函数）将返回其执行时所在数据库的名称。当在 *pubs2* 数据库中执行时，它会返回“pubs2”。

  ```
  exec pubs2..sp_foo

  ------------------------------
  pubs2

  (1 row affected, return status = 0)
  ```

  当在 *sybsystemprocs* 中执行时，它会返回“sybsystemprocs”。

  ```
  exec sybsystemprocs..sp_foo

  ------------------------------
  sybsystemprocs

  (1 row affected, return status = 0)
  ```

- 提供参数的方法有两种：通过位置或通过使用：

  @parameter_name = value

  如果使用第二种形式，就不必按照 **create procedure** 语句中定义的顺序来提供参数。

  如果您正在使用 **output** 关键字，并且要在批处理或过程中使用附加语句的返回参数，就必须将参数值当作变量来传递。例如：

  parameter_name = @variable_name

  当执行扩展存储过程时，需按照名称或值传递所有参数。在用于 ESP 的单个 **execute** 命令调用中，不能混合使用按值传递的参数和按名称传递的参数。

- 不能将 *text* 和 *image* 列用作存储过程的参数或传递给参数的值。
- 如果某个参数没有在 **create procedure** 语句中定义为返回参数，则不应执行为该参数指定 **output** 的过程。
- 不能使用 **output** 将常量传递给存储过程；返回参数需要一个变量名。在执行该过程之前，必须声明变量的数据类型并对其赋值。返回参数的数据类型不能是 *text* 或 *image*。
- 如果该语句是批处理中的第一个语句，就不必使用关键字 **execute**。批处理是输入文件的一段，它自行终止于一行上的“go”关键字。
- 由于过程的执行计划在其首次运行时即被存储，所以随后运行所需的时间大大少于运行相同独立语句集所需的时间。

- 当一个存储过程调用另一存储过程时，就会发生嵌套。嵌套层次在被调用过程开始执行时递增，而在被调用过程完成执行时递减。超过 16 层的嵌套将使事务失败。当前的嵌套层次存储在 *@@nestlevel* 全局变量中。
- 目前，Adaptive Server 使用返回值 0 和 -1 至 -14 来表示存储过程的执行状态。从 -15 至 -99 的值留作将来使用。如需值列表，参见 **return**。
- 参数不是事务的一部分，所以如果参数在随后回退的事务中作出了更改，它的值就不会还原为先前的值。返回给调用方的值始终是过程返回时的值。
- 如果在 **create procedure** 语句中使用了 **select ***，该过程就不会选取您可能添加到表中的任何新列（即使使用 **with recompile** 选项来执行）。您必须使用 **drop** 删除该过程并将其重新创建。
- 通过远程过程调用来执行的命令不能回退。
- 当 Adaptive Server 执行扩展存储过程时，将忽略 **with recompile** 选项。

动态执行 Transact-SQL

- 如果同 *string* 或 *char_variable* 选项一起使用， **execute** 将并置所提供的字符串和变量，以执行得到的 Transact-SQL 命令。这种形式的 **execute** 命令可用于 SQL 批处理、过程和触发器。
- 不能提供 *string* 和 *char_variable* 选项来执行以下命令：**begin transaction**、**commit**、**declare cursor**、**rollback**、**dump transaction**、**dbcc**、**set**、 **use** 或嵌套的 **execute** 命令。
- 可以使用 **execute()** 来指定 **create view** 命令，但是这只能在 SQL 批处理中进行。 **create view** 不能作为静态命令或 **execute()** 的字符串参数在过程中使用。
- *string* 或 *char_variable* 选项的内容不能引用在 SQL 批处理或过程中声明的局部变量。
- *string* 和 *char_variable* 选项可以通过并置来创建新表。但是，在同一 SQL 批处理或过程内，用 **execute()** 创建的表只对于其它的 **execute()** 命令才是可见的。当 SQL 批处理或过程完成后，动态创建的表将成为持久表，它对其它命令是可见的。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，过程的所有者拥有 **execute** 权限，他可以将此权限移交给其他用户。

对于用 *string* 或 *char_variable* 选项定义的 Transact-SQL 命令，其执行权限将对照执行该命令的用户来进行检查。即使 **execute()** 是在属于其它用户的过程或触发器内定义的，上述规则也同样适用。

参见

命令	**create procedure**、 **drop procedure**、 **return**
系统过程	**sp_addextendedproc**、 **sp_depends**、 **sp_dropextendedproc**、 **sp_helptext**

fetch

功能

从游标结果集中返回一行或一组行。

语法

```
fetch cursor_name [ into fetch_target_list ]
```

参数

cursor_name — 游标的名称

into *fetch_target_list* — 是放置游标结果的参数或局部变量的列表，它们用逗号分隔。参数和变量必须在 **fetch** 之前声明。

示例

```
1. fetch authors_crsr
```

从 *authors_crsr* 游标定义的游标结果集中返回一行信息。

```
2. fetch pubs_crsr into @name, @city, @state
```

从 *pubs_crsr* 游标所定义的游标结果集中，将一行信息返回到变量 *@name*、 *@city* 和 *@state*。

注释

限制

- 在使用 **fetch** 之前，必须声明游标并用 **open** 打开该游标。
- *cursor_name* 不能是 Transact-SQL 参数或局部变量。
- 不能对已经读取的行执行 **fetch** 命令。不能在结果集中回退，但可以关闭并重新打开游标，以重新创建该结果集并从起始处开始。
- 在 *fetch_target_list* 中的变量和由定义该游标的 *select_statement* 所指定的目标列表表达式之间， Adaptive Server 需要有一一对应关系。变量或参数的数据类型必须兼容游标结果集中列的数据类型。
- 当设置链式事务模式后，如果当前没有活动的事务， Adaptive Server 将隐式地用 **fetch** 语句启动事务。但是，只有在设置 **close on endtran** 选项并且在最初打开它的事务结束之后该游标仍然保持打开状态的条件下，才会出现这种情况。这是因为 **open** 语句也会自动启动事务。

游标位置

- 在用 **fetch** 读取完所有行之后，游标指向结果集的最后一行。如果再次执行 **fetch**， Adaptive Server 会通过 *@@sqlstatus* 变量返回警告，指出已经没有数据，并且游标位置已超出结果集的末尾。此时不能再从当前的游标位置执行 **update** 或 **delete**。
- 对于 **fetch into**，当由于 *fetch_target_list* 中的变量数不等于由定义该游标的查询所指定的目标列表表达式的数目而出现错误时，Adaptive Server 将不会向前移动游标位置。但是，如果在变量的数据类型和游标结果集中列的数据类型之间出现了兼容性错误，它仍然会向前移动游标位置。

确定读取的行数

- 一次可以用 **fetch** 读取一个或多个行。使用 **set** 命令的 **cursor rows** 选项可指定要 **fetch** （读取）的行数。

获取有关读取的信息

- *@@sqlstatus* 全局变量保存因执行 **fetch** 语句而产生的状态信息（警告例外）。 *@@sqlstatus* 的值为 0、 1 或 2，如表 6-24 所示。

表 6-24：@@sqlstatus 值

0	表示 **fetch** 语句成功完成。
1	表示 **fetch** 语句导致了错误。
2	表示结果集中不再有数据。如果当前的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，就会出现这一警告。

只有 **fetch** 语句才能设置 *@@sqlstatus*。其它语句对 *@@sqlstatus* 没有影响。

- *@@rowcount* 全局变量保存从游标结果集返回到客户端 （直到最后一个 **fetch**）的行数。换而言之，它表示客户端在任一时刻所见到的总行数。

  一旦从游标结果集读取了所有行， *@@rowcount* 就表示游标结果集中的总行数。每个打开的游标都与一个特定的 *@@rowcount* 变量相关联，该变量将在关闭游标时被删除。在执行 **fetch** 之后检查 *@@rowcount*，以便获得为该 **fetch** 中所指定的游标而读取的行数。

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	使用目标列表中的变量以及读取多行属于 Transact-SQL 扩展。

权限

缺省情况下，所有用户都拥有 **fetch** 权限。

参见

命令	declare cursor、 open、 set

goto 标签

功能

分支到用户定义的标签。

语法

```
label:
   goto label
```

示例

```
1. declare @count smallint
   select @count = 1
   restart:
     print "yes"
   select @count = @count + 1
   while @count <=4
     goto restart
```

显示对 *restart* 标签的使用情况。

注释

- 标签名必须符合标识符规则，并且必须在声明时跟有冒号 (:)。当同 **goto** 一起使用时，它不跟冒号。
- 要避免 **goto** 和标签之间的无穷循环，可以使 **goto** 依赖于 **if** 或 **while** 测试，或者是某一其它条件。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都拥有 **goto** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	if...else、 while

grant

功能

向用户或用户定义的角色分配权限。向用户或者是系统或用户定义的角色分配角色。

语法

要授予访问数据库对象的权限：

```
grant {all [privileges]| permission_list}
   on { table_name [(column_list)]
        | view_name[(column_list)]
        | stored_procedure_name}
   to {public | name_list | role_name}
   [with grant option]
```

要授予执行特定命令的权限：

```
grant {all [privileges] | command_list}
   to {public | name_list | role_name}
```

要将角色授予用户或角色：

```
grant {role role_granted [, role_granted ...]}
   to grantee [, grantee...]
```

关键字和选项

all — 当用来分配访问数据库对象的权限时 （第一种语法格式）， **all** 指定将授予所有适用于指定对象的权限。所有对象所有者都可以使用带对象名的 **grant all** 来授予各自对象的权限。

只有系统管理员或数据库所有者才能分配创建数据库对象的权限（第二种语法格式）。当由系统管理员使用时， **grant all** 将分配所有的 **create** 权限 （**create database**、 **create default**、 **create procedure**、 **create rule**、 **create table** 和 **create view**）。当数据库所有者使用 **grant all** 时， Adaptive Server 授予除 **create database** 外的所有 **create** 权限，并输出信息性消息。

当指定 **all** 时，并不包括执行 **set proxy** 或 **set session authorization** 的权限。

permission_list — 是所授予的对象访问权限的列表。如果要列出多项权限，请用逗号将其隔开。下表说明了对每一类对象所能授予的访问权限：

对象	***permission_list* 可以包括：**
表	**select**、 **insert**、 **delete**、 **update**、 **references**
视图	**select**、 **insert**、 **delete**、 **update**
列	**select**、 **update**、 **references** 列名可以在 *permission_list* 或 *column_list* 中指定 （参见示例 2）。
存储过程	**execute**

command_list — 用户可执行的命令的列表。如果要列出多项命令，请用逗号将其隔开。命令列表可以包括 **create database**、 **create default**、 **create procedure**、 **create rule**、 **create table**、 **create view**、 **set proxy** 和 **set session authorization**。

create database 权限只能由系统管理员授予，并且只能从 *master* 数据库中执行。

只有系统安全员才能向用户授予执行 **set proxy** 或 **set session authorization** 的权限。当授予执行 **set proxy** 或 **set session authorization** 的权限后，被授权者将可以充当服务器中的另一个登录。除了 **set session authorization** 遵循 ANSI92 标准而 **set proxy** 属于 Transact-SQL 扩展之外， **set proxy** 和 **set session authorization** 基本相同。

table_name — 是在其中授予权限的表的名称。该表必须位于当前数据库中。只能为每个 **grant** 语句列出一个对象。

column_list — 是应用权限的列的列表，各列用逗号隔开。如果列已指定，就只能授予 **select**、 **references** 和 **update** 权限。

view_name — 是在其中授予权限的视图的名称。该视图必须位于当前数据库中。只能为每个 **grant** 语句列出一个对象。

stored_procedure_name — 是在其中授予权限的存储过程的名称。该存储过程必须位于当前数据库中。只能为每个 **grant** 语句列出一个对象。

public — 表示所有用户。对于对象访问权限而言， **public** 不包括对象所有者。对于对象创建权限或 **set proxy** 授权而言， **public** 不包括数据库所有者。不能使用 **with grant option** 将权限授予 **(grant)** “public” 或者其它组或角色。

name_list — 是用逗号分隔的用户数据库名和 / 或组名的列表。

with grant option — 允许 *name_list* 中指定的用户将对象访问权限授予其他用户。只能使用 **with grant option** 将权限授予单独的用户，而不能授予 “public” 或者组或角色。

role — 将角色授予用户或者系统或用户定义的角色。

role_granted — 是系统安全员向用户或角色授予的系统 （或用户）定义角色的名称。

grantee — 是向其授予角色的系统角色、用户定义角色或用户的名称。

role_name — 是向其授予权限的系统 （或用户）定义角色的名称。

示例

1. ```
   grant insert, delete
   on titles
   to mary, sales
   ```

   向 Mary 和 “sales” 组授予对 *titles* 表使用 **insert** 和 **delete** 命令的权限。

2. ```
   grant update
   on titles (price, advance)
   to public
   ```

   或：

   ```
   grant update (price, advance)
   on titles
   to public
   ```

   以上是两种向 “public”（包括所有用户）授予 *titles* 表的 *price* 和 *advance* 列的 **update** 权限的方法。

3. ```
   grant set proxy to harry, billy
   ```

   向 Harry 和 Billy 授予通过执行 **set proxy** 或 **set session authorization** 来充当服务器中另一用户的权限。

4. ```
   grant set session authorization to sso_role
   ```

   向拥有 **sso_role** 权限的用户授予通过执行 **set proxy** 或 **set session authorization** 来充当服务器中另一用户的权限。

5. ```
   grant set proxy to vip_role
   ```

   向拥有 **vip_role** 的用户授予充当服务器中另一用户的能力。**vip_role** 必须是系统安全员用 **create role** 命令定义的角色。
   ```

```
6. grant create database, create table
   to mary, john
```

向 Mary 和 John 授予使用 **create database** 和 **create table** 命令的权限。因为正在授予 **create database** 权限，所以此命令只能由系统管理员在 *master* 数据库中执行。 Mary 和 John 的 **create table** 权限仅适用于 *master* 数据库。

```
7. grant all on titles
   to public
```

将 *titles* 表的全部访问权限授予所有用户。

```
8. grant all
   to public
```

将当前数据库中的所有对象创建权限授予所有用户。如果该命令是由系统管理员从 *master* 数据库中执行的，它将包括 **create database** 权限。

```
9. grant update on authors
   to mary
   with grant option
```

给予 Mary 对 *authors* 表使用 **update** 命令并将该权限授予其他人的权限。

```
10.grant select, update on titles(price)
   to bob
   with grant option
```

给予 Bob 对 *titles* 表的 *price* 列使用 **select** 和 **update** 命令并将该权限授予其他人的权限。

```
11.grant execute on new_sproc
   to sso_role
```

向所有系统安全员授予执行 *new_sproc* 存储过程的权限。

```
12.grant references on titles(price)
   to james
```

向 James 授予在引用 *titles* 表 *price* 列的其它表上创建参照完整性约束的权限。

```
13.grant role specialist_role to doctor_role
```

将角色“specialist”连同其所有权限和特权授予角色“doctor”。

```
14.grant role doctor_role to mary
```

将角色“doctor”授予 Mary。

注释

- 在 **grant** 语法中，可以用 **from** 替代 **to**。
- 表 6-25 总结了 Transact-SQL 命令在 Adaptive Server 中的缺省权限。在“在缺省情况下属于”标题下列出的用户是最低级别的用户，他们被自动授予执行命令的权限。如果某一权限是可以移交的，该用户就可使用 **grant** 或 **revoke** 授予或撤消该权限。缺省级别之上的用户既可以被自动授予权限，也可以（在该用户是数据库所有者的情况下）使用 **setuser** 命令来获得权限。

  例如，数据库的所有者不会自动接收其他用户所拥有对象的权限。要获得这样的权限，数据库所有者可以用 **setuser** 命令来获取对象所有者的身份，然后发出适当的 **grant** 或 **revoke** 语句。系统管理员拥有随时访问所有命令和对象的权限。

  Adaptive Server 安装脚本将一组权限分配给缺省组“public”，因此不必为这些权限编写 **grant** 和 **revoke** 语句。

  表 6-25 不包括系统安全员。系统安全员对命令和对象没有任何的特殊权限，他只拥有某些系统过程的特殊权限。

表 6-25：命令和对象权限

语句	在缺省情况下属于					是否可以授予/撤消		
	系统管理员	运算符	数据库所有者	对象所有者	Public	是	否	N/A
alter database			•			(1)		
alter role								•
alter table				•			•	
begin transaction					•			•
checkpoint			•				•	
commit					•			•
create database	•					•		
create default			•			•		
create index				•			•	
create procedure			•			•		
create role								•

(1) 用数据库所有权移交
(2) public 可以创建临时表，而不需要任何权限
(3) 如果是视图，权限在缺省情况下属于视图所有者
(4) 在缺省情况下属于存储过程所有者
(5) 用 **select** 权限移交
(6) 用 **update** 权限移交
“否”表示该命令的使用不受任何限制
“N/A”表示该命令的使用始终受到限制

表 6-25：命令和对象权限 （续）

语句	在缺省情况下属于					是否可以授予/撤消		
	系统管理员	运算符	数据库所有者	对象所有者	Public	是	否	N/A
create rule			•			•		
create table			•		(2)	•(2)		
create trigger					•	•		
create view			•			•		
dbcc	根据选项而有所不同。参见本手册中的 **dbcc**。						•	
delete				•(3)		•		
disk init	•						•	
disk mirror	•							
disk refit	•							
disk reinit	•							
disk remirror	•							
disk unmirror	•						•	
drop （任何对象）				•			•	
dump database		•	•				•	
dump transaction		•	•				•	
execute				•(4)		•		
grant （对象）				•		•		
grant （命令）			•			•		
insert				•(3)		•		
kill	•						•	
load database		•	•				•	
load transaction		•	•				•	
print					•			•
raiserror					•			•
readtext				•		(5)		

(1) 用数据库所有权移交
(2) public 可以创建临时表，而不需要任何权限
(3) 如果是视图，权限在缺省情况下属于视图所有者
(4) 在缺省情况下属于存储过程所有者
(5) 用 **select** 权限移交
(6) 用 **update** 权限移交
“否”表示该命令的使用不受任何限制
“N/A”表示该命令的使用始终受到限制

表 6-25：命令和对象权限 （续）

语句	在缺省情况下属于					是否可以授予/撤消		
	系统管理员	运算符	数据库所有者	对象所有者	Public	是	否	N/A
revoke （对象）				•			•	
revoke （命令）			•				•	
rollback					•			•
save transaction					•			•
select				•(3)		•		
set					•			•
setuser			•				•	
shutdown	•						•	
truncate table				•			•	
update				•(3)		•		
update all statistics				•			•	
update partition statistics				•			•	
update statistics				•			•	
writetext				•		(6)		

(1) 用数据库所有权移交
(2) public 可以创建临时表，而不需要任何权限
(3) 如果是视图，权限在缺省情况下属于视图所有者
(4) 在缺省情况下属于存储过程所有者
(5) 用 **select** 权限移交
(6) 用 **update** 权限移交
“否”表示该命令的使用不受任何限制
“N / A”表示该命令的使用始终受到限制

- 只能授予当前数据库中对象的权限。
- 如果要创建的表包含对引用其它用户表的参照完整性约束，您必须在创建之前被授予被引用表的 **references** 权限 （参见示例 10）。该表还必须包含被引用列的唯一约束或唯一索引。有关参照完整性约束的详细信息，参见 **create table**。
- **grant** 和 **revoke** 命令区别先后顺序。在出现冲突时生效的命令是最近发布的那一条。
- 即使用户对存储过程或视图所引用的对象没有权限，仍可以向他们授予该过程或视图的权限。有关详细信息，参见*系统管理指南*。

- 无论为 **declare cursor** 语句中引用的基表或基视图定义了什么权限，Adaptive Server 都会向所有用户授予声明游标的权限。由于游标没有被定义为 Adaptive Server 对象 （例如表），因此不能对游标应用任何权限。当用户打开游标时， Adaptive Server 将确定该用户是否对定义此游标结果集的对象拥有 **select** 权限。每次打开游标时，它都会执行权限检查。

  如果用户拥有访问该游标所定义的对象的权限， Adaptive Server 将打开该游标并允许用户通过该游标 **fetch** （读取）行数据。Adaptive Server 不会为每一个 **fetch** 应用权限检查。但是，如果用户通过该游标执行 **delete** 或 **update**，以删除和更新游标结果集所引用对象的数据，则将对删除和更新操作进行常规的权限检查。

- 对于每个接受权限的用户、组或角色， **grant** 语句都将在 *sysprotects* 系统表中添加一行。如果您随后用 **revoke** 撤消用户或组的权限， Adaptive Server 将从 *sysprotects* 中删除该行。如果仅撤消所选组成员的权限，而没有将权限从所授予的整个组中撤消， Adaptive Server 就将保持这一初始行，同时为该撤消操作添加新的一行。

- 如果用户作为组成员而继承了某种权限，并且又向该用户显式授予了同样的权限，就不会有任何行添加到 *sysprotects* 中。例如，如果 “public” 被授予 *authors* 表中 *phone* 列的 **select** 权限，然后，向 John （“public” 的成员）授予 *authors* 表中所有列的 **select** 权限。那么，作为向 John 授权 **(grant)** 的结果而添加到 *sysprotects* 的行将包含对 *authors* 表中所有列的引用，但 *phone* 列除外，因为 John 已经拥有该列的权限。

- 缺省情况下，用户将被授予 **create trigger** 命令的权限。当撤消用户创建触发器的权限时，将有一个撤消行添加到该用户的 *sysprotects* 表中。要向该用户授予发出 **create trigger** 命令的权限，必须发出两个 **grant** 命令。第一个命令从 *sysprotects* 中删除撤消行；第二个命令插入授权行。如果撤消创建触发器的权限，用户就不能创建触发器，即便是在自己的表上也不行。撤消用户创建触发器的权限只影响从中发出 revoke 命令的数据库。

- 可以用以下系统过程获得有关权限的信息：

  - **sp_helprotect** 报告数据库对象或用户的权限信息。
  - **sp_column_privileges** 报告表或视图中一列或多列的权限信息。
  - **sp_table_privileges** 报告表或视图中所有列的权限信息。
  - **sp_activeroles** 显示 Adaptive Server 当前登录会话的所有活动角色。
  - **sp_displayroles** 显示授权给另一角色的所有角色，或者以表格形式显示完整的角色层次树。

grant all （对象创建权限）

- 当只使用用户名或组名（没有对象名）时，**grant all** 将分配以下权限：**create database**、**create default**、**create procedure**、**create rule**、**create table** 和 **create view**。**create database** 权限只能由系统管理员授予，并且只能在 *master* 数据库中执行。
- 在没有对象名的情况下，只有数据库所有者和系统管理员才能使用 **grant all** 语法将 **create** 命令权限授予用户或组。当数据库所有者使用 **grant all** 命令时，将输出一条信息性消息，指出只有系统管理员才能授予 **create database** 权限。上述的其它所有权限都会被授予。
- 所有对象所有者都可以使用带对象名的 **grant all** 来授予各自对象的权限。当与表名或视图名以及用户名或组名一起使用时，**grant all** 将启用该表的 **delete**、**insert**、**select** 和 **update** 权限。

grant with grant option 规则

- 不能使用 **with grant option** 将权限授予 “public”或者组或角色。
- 授予权限时，系统管理员被视作对象所有者。如果系统管理员授予另一用户的对象的权限，所有者名将作为授权者显示在 *sysprotects* 和 **sp_helprotect** 输出中。
- 每个 **grant** 的信息都将保存在系统表 *sysprotects* 中，下列情况例外：
  - 如果特定权限被同一授权者多次授予某一用户，Adaptive Server 将显示信息性消息。只保留第一个 **grant**。
  - 如果两个 **grant** 语句除了其中一个是用 **with grant option** 来授予之外完全一样，则将保留 **grant with grant option**。
  - 如果两个 **grant** 语句将特定表的相同权限授予某一特定用户，但在 grant 语句中指定的列不相同，Adaptive Server 会将这两个语句当作一个语句。例如，以下 **grant** 语句是等同的：

```
grant select on titles(price, contract) to keiko
grant select on titles(advance) to keiko

grant select on titles(price, contract, advance)
to keiko
```

授予代理和会话授权

- 授予执行 **set proxy** 或 **set session authorization** 的权限后，被授权者就可以充当 Adaptive Server 中的另一个登录。**set proxy** 和 **set session authorization** 是相同的，但有一项例外：**set session authorization** 遵循 SQL 标准，而 **set proxy** 属于 Transact-SQL 扩展。
- 要授予 **set proxy** 或 **set session authorization** 权限，您必须是系统安全员，并且必须在 *master* 数据库中进行。
- 在 **grant set proxy** 命令中指定的名称必须是数据库中的有效用户；也就是说，该名称必须在数据库的 *sysusers* 表中。
- **grant all** 不包括 **set proxy** 或 **set session authorization** 权限。

向角色授予权限

- 可以使用 **grant** 命令将权限授予所有已授予指定角色的用户。该角色既可以是系统角色（如 **sso_role** 或 **sa_role**），也可以是用户定义的角色。对于用户定义的角色，系统安全员必须用 **create role** 命令来创建该角色。

  然而，**grant execute** 权限不会禁止向没有指定角色的各个用户分别授予执行存储过程的权限。如果要确保只有系统安全员才能被授予执行存储过程的权限，可在该存储过程内使用 **proc_role** 系统函数。此命令检查调用用户是否有执行过程的正确权限。有关详细信息，参见 **proc_role**。
- 向角色授予的权限将替换向用户或组授予的权限。例如，假定 John 被授予系统安全员的角色，而 **sso_role** 被授予对 *sales* 表的权限。如果撤消了 John 对 *sales* 的个人权限，他仍然可以访问 *sales*，因为他的角色权限替换了他的个人权限。

用户和用户组

- 用户组使您可以用单个语句对多个用户授予 (**grant**) 或撤消 (**revoke**) 权限。每一个用户都可以只是一个组的成员，并且始终是“public”的成员。
- 数据库所有者或系统管理员可以用 **sp_adduser** 添加新的用户，并用 **sp_addgroup** 创建组。要允许具有 Adaptive Server 登录的用户以有限的特权来使用数据库，可以用 **sp_adduser** 添加“guest”用户，并为“guest”分配有限的特权。具有登录所有的用户都可以作为“guest”来访问数据库。
- 要删除用户，可使用 **sp_dropuser**。要删除组，可使用 **sp_dropgroup**。

  要将新用户添加到“public”之外的组中，可使用 **sp_adduser**。要更改已建立的用户组，可使用 **sp_changegroup**。

  要显示某一组的成员，可使用 **sp_helpgroup**。

- 当执行 **sp_changegroup** 来更改组成员时，它会通过执行以下语句来清除内存中的保护缓存：

  ```
  grant all to null
  ```

  这样，该缓存可以用 *sysprotects* 表的更新信息来进行刷新。如果需要直接修改 *sysprotects*，请与 Sybase 技术支持部门联系。

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	向组授予权限以及授予 **set proxy** 属于 Transact-SQL 扩展。授予 **set session authorization**（功能与 **set proxy** 相同）遵循 ANSI 标准。

权限

数据库对象访问权限

缺省情况下，对象所有者拥有数据库对象的 **grant** 权限。对象所有者可以向其他用户授予他自己数据库对象的权限。

命令执行权限

只有系统管理员才能授予 **create database** 权限，并且只能从 *master* 数据库中执行。只有系统安全员才能授予 **create trigger** 权限。

代理和会话授权权限

只有系统安全员才能授予 **set proxy** 或 **set session authorization**，并且只能从 *master* 数据库中执行。

角色权限

只能从 *master* 数据库中授予角色。只有系统安全员才能将 **sso_role**、**oper_role** 或用户定义的角色授予某一用户或角色。只有系统管理员才能将 **sa_role** 授予某一用户或角色。只有同时拥有 **sa_role** 和 **sso_role** 权限的用户才能授予包括 **sa_role** 的角色。

参见

分类存储过程	**sp_column_privileges**
命令	**revoke、 setuser、 set**
函数	**proc_role**
系统过程	**sp_addgroup、 sp_adduser、 sp_changedbowner、 sp_changegroup、 sp_dropgroup、 sp_dropuser、 sp_helpgroup、 sp_helprotect、 sp_helpuser、 sp_role**

group by 和 having 子句

功能

用于在 **select** 语句中将表分组，并且只返回符合 **having** 子句中条件的组。

语法

```
Start of select statement
[group by [all] aggregate_free_expression
   [, aggregate_free_expression]...]
[having search_conditions]
End of select statement
```

关键字和选项

group by — 指定要将表划分到的组，而且如果在选择列表中包含了集合函数，它还会查找每一组的摘要值。这些摘要值在结果中显示为列，每组一列。可以在 **having** 子句中引用这些摘要列。

在 **group by**（该表达式通常是列名）之前，可以在选择列表中使用 **avg**、**count**、**max**、**min** 和 **sum** 集合函数。有关详细信息，参见第 2 章 “Transact-SQL 函数”中的 “集合函数”。

可以按列的任意组合来将表分组 — 也就是说，组和组之间可以相互嵌套，如示例 2 所示。

all — 是一种 Transact-SQL 扩展，用于包括结果中的所有组，甚至包括 **where** 子句所排除的那些组。例如：

```
select type, avg(price)
from titles
where advance > 7000
group by all type
```

```
type
-----------------  ----------
UNDECIDED                NULL
business                 2.99
mod_cook                 2.99
popular_comp            20.00
psychology               NULL
trad_cook               14.99

(6 rows affected)
```

集合列中的 “NULL”指出被 **where** 子句排除的组。**having** 子句否定 **all** 的含义。

aggregate_free_expression — 是不包含集合的表达式。除了允许按列名分组外， Transact-SQL 扩展还允许按无集合的表达式来分组。

不能按照列标题或别名进行分组。以下示例是正确的：

```
select Price=avg(price), Pay=avg(advance),
Total=price * $1.15
from titles
group by price * $1.15
```

having — 为 **group by** 子句设置条件，类似于 **where** 为 **select** 子句设置条件的方式。

having 搜索条件可以包括集合表达式；除此之外， **having** 搜索条件与 **where** 搜索条件完全相同。以下是有集合的 **having** 子句的示例：

```
select pub_id, total = sum(total_sales)
from titles
where total_sales is not null
group by pub_id
having count(*)>5
```

Adaptive Server 优化查询时，将评估 **where** 和 **having** 子句中的搜索条件，然后确定哪些条件是可用来选择最佳索引和查询计划的搜索参数 (SARG)。对于查询中的每个表，最多可使用 128 个搜索参数来优化查询。不过，所有搜索条件都将用来限定行。有关搜索参数的详细信息，参见 *Performance and Tuning Guide*。

示例

1.
```
select type, avg(advance), sum(total_sales)
from titles
group by type
```

计算每种书籍的平均预付款以及销售额的总和。

2.
```
select type, pub_id, avg(advance), sum(total_sales)
from titles
group by type, pub_id
```

先按类型将结果分组，然后按每一类型中的 *pub_id* 将结果分组。

3.
```
select type, avg(price)
from titles
group by type
having type like 'p%'
```

计算所有组的结果，但仅显示类型以 “p”开始的组。

4.
```
select pub_id, sum(advance), avg(price)
from titles
group by pub_id
having sum(advance) > $15000
and avg(price) < $10
and pub_id > "0700"
```

   计算所有组的结果，但仅显示符合 **having** 子句中多项条件的组的结果。

5.
```
select p.pub_id, sum(t.total_sales)
from publishers p, titles t
where p.pub_id = t.pub_id
group by p.pub_id
```

   在连接 *titles* 表和 *publishers* 表之后，计算每一组 （出版者）的总销售额。

6.
```
select title_id, advance, price
from titles
where advance > 1000
having price > avg(price)
```

   显示这样的标题：有预付款高于 $1000 并且有价格高于所有标题的平均价格。

注释

- 在 **group by** 后可以使用列名或任何表达式 （列标题或别名除外）。可以使用 **group by** 来计算结果，或者显示不在选择列表中的列或表达式 （Transact-SQL 扩展在 “group by 和 having 的 Transact-SQL 扩展”中进行说明）。
- **group by** 子句中允许的最大列数或表达式数是 16。
- **group by** 子句指定的所有列的最大长度之和不能超过 256 字节。
- **group by** 列中的 Null 将值放置在单个组中。
- 不能在 **group by** 和 **having** 子句中指定 *text* 或 *image* 列。
- 在可更新游标的 **select** 语句中，不能使用 **group by** 子句。
- 集合函数只能在选择列表或 **having** 子句中使用。它们不能用于 **where** 或 **group by** 子句。

  集合函数有两种类型。应用于**表中所有限定行** （每一个函数为整个表产生单个值）的集合称为**标量集合**。不带 **group by** 子句的选择列表中的集合函数将应用于整个表；它是标量集合的一个示例。

应用于**指定列或表达式中一组行**（每一个函数为每一组都产生了一个值）的集合称为**矢量集合**。对于任一种集合类型，集合操作的结果都显示为 **having** 子句可引用的新列。

可以将矢量集合嵌套在标量集合内。有关详细信息，参见第 2 章“Transact-SQL 函数”中的“集合函数”。

带有集合的 *group by* 和 *having* 查询如何工作

- **where** 子句排除不满足其搜索条件的行，其功能对于已分组或未分组的查询都相同。
- 对于 **group by** 表达式中的每个唯一值，**group by** 子句都会将剩余的行集中到一组中。如果省略 **group by**，将为整个表创建单个组。
- 在选择列表中指定的集合函数会计算每一组的摘要值。对于标量集合，该表只有一个值。矢量集合会计算不同组的值。
- **having** 子句从结果中排除那些不满足其搜索条件的组。即使 **having** 子句仅测试行，**group by** 子句的存在会使其看起来象是对组进行操作：
  - 当查询包括 **group by** 时，**having** 将排除结果组的行。因此，**having** 看起来象是对组进行操作。
  - 当查询不包括 **group by** 时，**having** 将从（单组）表中排除结果行。因此，**having** 看起来象是对行进行操作（结果类似于 **where** 子句的结果）。

标准的 *group by* 和 *having* 查询

- “示例”部分中的所有 **group by** 和 **having** 查询都遵循 SQL 标准。它规定，使用 **group by**、**having** 和矢量集合函数的查询将为每一组都生成一行和一个摘要值，其原则如下：
  - 选择列表中的列必须同时位于 **group by** 表达式中，或者它们必须是集合函数的参数。
  - **group by** 表达式只包含选择列表中的列名。但是，如果列仅用作选择列表中集合函数的参数，则不符合条件。
  - **having** 表达式中的列必须是单值的集合参数，举例来说，它们必须在选择列表或 **group by** 子句中。带有选择列表集合和 **having** 子句的查询**必须**具有 **group by** 子句。如果对无选择列表集合的查询省略了 **group by**，所有未被 **where** 子句排除的行都将被当作一组（参见示例 6）。

  在未分组的查询中，“**where** 排除行”的原则似乎是非常直接的。而在已分组的查询中，该原则扩展为“**where** 在 **group by** 之前将行排除在外，而 **having** 则从结果显示中将行排除在外”。

- SQL 标准允许连接了两个或多个表的查询使用 **group by** 和 **having**，如果它们也遵循上述指导方针的话。当指定连接或其它复杂的查询时，如果您尚未充分理解 Transact-SQL 扩展对这两个子句的作用，则应使用 group by 和 having 的标准说法（如 “group by 和 having 的 Transact-SQL 扩展”所述）。

  为了避免在使用扩展时出现的问题， Adaptive Server 为 **set** 命令提供了 **fipsflagger** 选项，它可以对查询中 Transact-SQL 扩展的每一次出现发出非致命警告。有关详细信息，参见 **set**。

group by 和 *having* 的 Transact-SQL 扩展

- 通过允许参照那些没有用于创建组或摘要计算的列和表达式，对标准 SQL 的 Transact-SQL 能够使数据显示更加灵活：
  - 含有集合的选择列表可以包含那些不是集合函数的参数，并且没有包含在 **group by** 子句中的**扩展**列。由于显示了附加的行，扩展列将影响最终结果的显示。
  - **group by** 子句可以包含不在选择列表中的列或表达式。
  - **group by all** 子句会显示所有的组，即使是那些被 **where** 子句排除在外的计算。参见 “关键字和选项”节中的 **all** 关键字的示例。
  - **having** 子句可以包含不在选择列表中以及不在 **group by** 子句中的列或表达式。

  当 Transact-SQL 扩展添加行和列来显示时，或者如果省略了 **group by**，查询结果将很难解释。下列示例可以帮助您理解 Transact-SQL 扩展是如何影响查询结果的。

- 下列示例阐明了使用标准的 **group by** 和 **having** 子句的查询和使用了 Transact-SQL 扩展的查询之间的差别：

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

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

   (6 rows affected)
```

标准分组查询的示例。

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

```
type         price
------------ ---------------- ----------------
business                19.99            13.73
business                11.95            13.73
business                 2.99            13.73
business                19.99            13.73
mod_cook                19.99            11.49
mod_cook                 2.99            11.49
UNDECIDED                NULL             NULL
popular_comp            22.95            21.48
popular_comp            20.00            21.48
popular_comp             NULL            21.48
psychology              21.59            13.50
psychology              10.95            13.50
psychology               7.00            13.50
psychology              19.99            13.50
psychology               7.99            13.50
trad_cook               20.95            15.96
trad_cook               11.95            15.96
trad_cook               14.99            15.96

(18 rows affected)
```

Transact-SQL 扩展列， *price*（在选择表而不是集合中，并且不在 **group by** 子句中），使得所有受限定的行在每一被限定的组中显示，即使标准的 **group by** 子句为每组产生了单个行。 **group by** 仍然影响矢量集合，该集合将计算在每组中的各行上所显示的每组的平均价格 （对于示例 1 而言，它们是相同的计算值）。

```
3. select type, avg(price)
   from titles
   where price > 10,00
   group by type
```

```
type         price
------------ ---------------- ----------------
business                19.99            17.31
business                11.95            17.31
business                 2.99            17.31
business                19.99            17.31
mod_cook                19.99            19.99
mod_cook                 2.99            19.99
popular_comp            22.95            21.48
popular_comp            20.00            21.48
popular_comp             NULL            21.48
```

```
psychology              21.59           17.51
psychology              10.95           17.51
psychology               7.00           17.51
psychology              19.99           17.51
psychology               7.99           17.51
trad_cook               20.95           15.96
trad_cook               11.95           15.96
trad_cook               14.99           15.96

(17 rows affected)
```

处理 Transact-SQL 扩展列的方法可以使其看起来好像是查询忽略了 **where** 子句。此查询仅使用那些满足 **where** 子句的列来计算平均价格，但是它也显示了不匹配 **where** 子句的行。

Adaptive Server 首先利用 **where** 子句，建立了仅包含类型和集合值的工作表。这一工作表被连接到分组列 *type* 中的 *titles* 表的后面，从而在结果中包括了 *price* 列，但是在该连接中**没有**使用 **where** 子句。

titles 中唯一没有在结果中出现的行，是具有 *type* =“UNDECIDED” 和空价格值的独立行，也就是在工作表中没有其结果的行。如果您还要从所显示的价格低于 $10.00 的结果中消除这些行，就必须添加重复 **where** 子句的 **having** 子句，如示例 4 所示。

```
4. select type, price, avg(price)
   from titles
   where price > 10,00
   group by type
   having price > 10.00

   type         price
   ------------ ----------------  ----------------
   business                19.99           17.31
   business                11.95           17.31
   business                19.99           17.31
   mod_cook                19.99           19.99
   popular_comp            22.95           21.48
   popular_comp            20.00           21.48
   psychology              21.59           17.51
   psychology              10.95           17.51
   psychology              19.99           17.51
   trad_cook               20.95           15.96
   trad_cook               11.95           15.96
   trad_cook               14.99           15.96

   (12 rows affected)
```

如果在 **having** 子句中指定了额外的的条件，如集合，请务必还要包括在 **where** 子句中所指定的所有条件。 Adaptive Server 看起来像是忽略了从 **having** 子句中遗失的所有 **where** 子句条件。

5.
```
select p.pub_id, t.type, sum(t.total_sales)
from publishers p, titles t
where p.pub_id = t.pub_id
group by p.pub_id, t.type
```

```
pub_id  type
------  ------------  --------
0736    business         18722
0736    psychology        9564
0877    UNDECIDED         NULL
0877    mod_cook         24278
0877    psychology         375
0877    trad_cook        19566
1389    business         12066
1389    popular_comp     12875

(8 rows affected)
```

这是标准分组查询的示例，该查询使用了两个表之间的连接。它按照 *pub_id*，然后按照出版者 ID 中的 *type* 进行分组，从而为每一行计算矢量集合。看起来可能只需像下面所述的那样，为 *pub_id* 和 *type* 指定 **group by** 来产生结果并添加扩展行：

```
select p.pub_id, p.pub_name, t.type,
    sum(t.total_sales)
from publishers p, titles t
where p.pub_id = t.pub_id
group by p.pub_id, t.type
```

但是，上述查询的结果与本示例中的第一个查询的结果有很大的不同。在连接两个表从而在工作表中确定了矢量集合之后，Adaptive Server 将工作表连接到用于最终结果的扩展列的表 (*publishers*)。来自于不同表中的每一个扩展行都会激活一个额外的连接。

正如您能看到的那样，在连接了表的查询中使用扩展列扩展很容易产生难以理解的结果。在大多数情况下，应该在查询中使用标准的 **group by** 来连接表。

```
6. select p.pub_id, sum(t.total_sales)
   from publishers p, titles t
   where p.pub_id = t.pub_id
   group by p.pub_id, t.type
```

```
pub_id
------  --------
0736       18722
0736        9564
0877        NULL
0877       24278
0877         375
0877       19566
1389       12066
1389       12875

 (8 rows affected)
```

本示例使用了 **group by** 的 Transact-SQL 扩展，从而包括了没有在选择列表中的列。 *pub_id* 和 *type* 列都是用来对矢量集合的结果进行分组的。但是，最终结果不包括每一个出版者中的类型。在这种情况下，您可能只想知道对每一个出版者而言，出售了多少不同的标题类型。

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

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

 (3 rows affected)
```

本示例组合了两种 Transact-SQL 扩展的作用。首先，它省略了 **group by** 子句，而在选择表中包括了集合。其次，它包括了扩展列。通过省略 **group by** 子句：

- 该表变成了单一的组。标量集合有三个受限定的行。
- 因为它没有出现在 **group by** 子句中，所以 *pub_id* 变成了 Transact-SQL 扩展列。没有出现任何 **having** 子句，所以组中的所有行都适于显示。

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

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

    (3 rows affected)
```

where 子句将 *pub_id* 为 1000 或以上的出版者排除在单一组之外，所以标量集合有两个受限定的行。扩展列 *pub_id* 显示了 *publishers* 表中所有受限定的行。

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

   pub_id
   -------------- ---------
   0736                   3
   0877                   3
    (2 rows affected)
```

本示例阐明了没有使用 **group by** 子句的 **having** 子句的作用。

- 该表被视为单一的组。没有任何 **where** 子句把行排除在外，所以组（表）中的所有行都能够被计算在内。
- 单组表中的行按照 **having** 子句进行了测试。
- 这些组合的作用将显示两个被限定的行。

```
10.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)
```

本示例使用了对 **having** 的扩展，它允许包含不在选择列表中以及不在 **group by** 子句中的列或表达式。它确定了每一标题类型的平均价格，但是它将那些总销售量没有超过 $10,000 的类型排除在外，即使 **sum** 集合没有出现在结果中。

group by 和 *having* 以及排序顺序

- 如果服务器拥有不区分大小写的排序顺序，**group by** 忽略分组列中的大小写。例如，假设不区分大小写的服务器上的数据按照 *lname* 进行分组：

```
select lname, amount
from groupdemo
```

```
lname      amount
---------- ------------------
Smith                  10.00
smith                   5.00
SMITH                   7.00
Levi                    9.00
Lévi                   20.00
```

产生了这些结果：

```
select lname, sum(amount)
from groupdemo
group by lname
```

```
lname
---------- ------------------
Levi                    9.00
Lévi                   20.00
Smith                  22.00
```

在不区分大小写和不区分变音的服务器上进行相同的查询将产生以下结果：

```
lname
 ---------- ------------------
 Levi                    29.00
 Smith                   22.00
```

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	对于不在 **group by** 列表中并且没有集合函数的 **select** 列表而言，其中的列的使用是 Transact-SQL 扩展。 **all** 关键字的使用是 Transact-SQL 扩展。

参见

命令	**compute Clause**、 **declare**、 **select**、 **where 子句**
函数	“集合函数”

if...else

功能

对执行 SQL 语句施加条件。

语法

```
if logical_expression [plan "abstract plan"]
   statements
[else
   [if logical_expression] [plan "abstract plan"]
   statement]
```

关键字和选项

logical_expression — 是返回 TRUE、 FALSE 或 NULL 的表达式 （可以是列名、常量、任何由算术运算符或逐位运算符连接的列名和常量组合、也可以是子查询）。如果表达式包含 select 语句，则必须将该 select 语句用小括号括起来。

plan *"abstract plan"* — 指定用于优化查询的抽象计划。它可以是用抽象计划语言指定的完整计划或部分计划。只能为可优化的 SQL 语句 （即访问表的 **select** 查询）指定计划。

statements — 是单个 SQL 语句或由 **begin** 和 **end** 分隔的语句块。

plan *"abstract plan"* — 指定用于优化查询的抽象计划。它可以是用抽象计划语言指定的完整计划或部分计划。只能为 **if** 子句中可优化的表达式 （即访问表的查询）指定计划。有关详细信息，参见 *Performance and Tuning Guide* 中的第 22 章 “Creating and Using Abstract Plans”。

示例

```
1. if 3 > 2
    print "yes"

2. if exists (select postalcode from authors
    where postalcode = "94705")
    print "Berkeley author"
```

```
3. if (select max(id) from sysobjects) < 100
      print "No user-created objects in this database"
   else
    begin
      print "These are the user-created objects"
      select name, type, id
      from sysobjects
      where id > 100
    end
```

if...else 条件测试数据库中是否存在用户创建的对象（其 ID 号都大于 100）。如果存在用户表， **else** 子句将输出一条消息，并选择用户表的名称、类型和 ID 号。

```
4. if (select total_sales
        from titles
       where title_id = "PC9999") > 100
   select "true"
   else
   select "false"
```

由于在 *titles* 表中， PC9999 总销售额的值是 NULL，该查询将返回 FALSE。当查询的 **if** 部分返回 FALSE 或 NULL 时，将执行其 **else** 部分。有关真值和逻辑表达式的详细信息，参见第 3 章 “表达式、标识符和通配符”中的 “表达式”。

注释

- 当满足条件 （逻辑表达式返回 TRUE）时，将执行 **if** 关键字及其条件后的语句。可选的 **else** 关键字引入一个替代 SQL 语句。当不满足 **if** 条件 （逻辑表达式返回 FALSE 时）时，就会执行该语句。
- **if** 或 **else** 条件只影响单个 SQL 语句的执行，除非使多个语句组成关键字 **begin** 和 **end** 之间的一个块 （参见示例 3）。

  语句子句可以是 **execute** 存储过程命令或任何其它的合法 SQL 语句或语句块。
- 如果将 **select** 语句用作布尔表达式的一部分，它必须返回单个值。
- **if...else** 结构可用于存储过程 （通常用来测试某一参数是否存在）或 *ad hoc* 查询 （参见示例 1 和 2）。
- **if** 测试可嵌套在其它 **if** 中，也可嵌套在 **else** 之后。根据在每个 **if...else** 结构中包括的任何 **select** 语句 （或其它语言结构）的复杂性，可以嵌套的 **if** 测试的最大数量会有所不同。

➤ *注意*

当在一个 **if...else** 块中发生 **create table** 或 **create view** 命令时， Adaptive Server 将在确定条件是否为真之前创建该表或视图的模式。如果该表或视图已经存在，这就会导致错误。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都具有 **if...else** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	**begin...end**、**create procedure**

insert

功能

在表或视图中添加新行。

语法

```
insert [into]
   [database.[owner.]]{table_name|view_name}
   [(column_list)]
   {values (expression [, expression]...)
       |select_statement [plan "abstract plan"] }
```

关键字和选项

into — 是可选的关键字。

table_name | *view_name* — 是要删除行的表或视图的名称。当表或视图位于其它数据库时，指定该数据库的名称；当数据库中存在具有该名称的多个表或视图时，指定所有者的名称。 *owner* 的缺省值是当前用户，而 *database* 的缺省值是当前数据库。

column_list — 是将添加数据的一个或多个列的列表。该列表需用小括号括起来。各列可以按任何顺序列出，但添加的数据 （无论是在 **values** 子句还是在 **select** 子句中）必须具有相同的顺序。如果某列具有 IDENTITY 属性，就可以用 **syb_identity** 关键字替代实际的列名。

当表中的某些列 （但不是所有列）将接收数据时，列列表就是必需的。如果没有提供列列表， Adaptive Server 将假定 **insert** 会影响接收表中的所有列 （按 **create table** 顺序）。

有关详细信息，参见 “列列表”。

values — 是引入表达式列表的关键字。

expression — 为指定列指定常量表达式、变量、参数或空值。字符和日期时间常量应该用单引号或双引号引起来。

不能将子查询用作 *expression*。

值列表必须用小括号括起来，并且必须匹配显式或隐式的列列表。有关数据输入规则的详细信息，参见 “数据类型”。

select_statement — 是标准的 **select** 语句，用于检索要插入的值。

plan *"abstract plan"* — 指定用于优化查询的抽象计划。它可以是用抽象计划语言指定的完整计划或部分计划。只能为 **insert...select** 语句指定计划。有关详细信息，参见 *Performance and Tuning Guide* 中的第 22 章 “Creating and Using Abstract Plans”。

示例

```
1. insert titles
   values("BU2222", "Faster!", "business", "1389",
       null, null, null, "ok", "06/17/87", 0)

2. insert titles
   (title_id, title, type, pub_id, notes, pubdate,
       contract)
   values ('BU1237', 'Get Going!', 'business',
        '1389', 'great', '06/18/86', 1)

3. insert newauthors
     select *
     from authors
     where city = "San Francisco"

4. insert test
     select *
     from test
     where city = "San Francisco"
```

注释

- 只应将 **insert** 用来添加新行。要在已插入的行中修改列值，应使用 **update**。

列列表

- 列列表用于确定值的输入顺序。例如，假设有一个名为 *newpublishers* 的表，它的结构和内容与 *pubs2* 中的 *publishers* 表完全相同。在以下示例中， *newpublishers* 表的列列表中的列匹配 *publishers* 表的选择列表中的列。

  ```
  insert newpublishers (pub_id, pub_name)
  select pub_id, pub_name
      from publishers
      where pub_name="New Age Data"
  ```

  “New Age Data”的 *pub_id* 和 *pub_name* 存储在 *newpublishers* 的 *pub_id* 和 *pub_name* 列中。

在下一个示例中， *newpublishers* 表的列列表中列的顺序不匹配 *publishers* 表的选择列表中列的顺序。

```
insert newpublishers (pub_id, pub_name)
    select pub_name, pub_id
    from publishers
    where pub_name="New Age Data"
```

结果是，“New Age Data”的 *pub_id* 存储在 *newpublishers* 表的 *pub_name* 列中，而 “New Age Data”的 *pub_name* 存储在 *newpublishers* 表的 *pub_id* 列中。

- 只要列允许空值，就可以从列列表和值列表中省略这些列以及它们的值 （参见示例 2）。

验证列值

- **insert** 与 **ignore_dup_key**、 **ignore_dup_row** 和 **allow_dup_row** 选项 （用 **create index** 命令设置）相互作用。（有关详细信息，参见 **create index**。）
- 规则或 **check** 约束能够限制可在列中输入的合法值的域。规则用 **create rule** 命令创建，并与系统过程 **sp_bindrule** 绑定。 **check** 约束用 **create table** 语句声明。
- 如果没有显式输入值，缺省值就可以提供值。缺省值用 **create default** 命令创建，并与系统过程 **sp_bindefault** 绑定。或者，它们也可以用 **create table** 语句声明。
- 如果 **insert** 语句违反域规则或完整性规则 （参见 **create rule** 和 **create trigger**）或属于错误的数据类型 （参见 **create table** 和 “系统和用户定义的数据类型”），那么该语句将失败，并且 Adaptive Server 会显示错误消息。

处理空白

- 如果在变量字符类型或 *text* 列中插入一个空字符串 (“”)，就会插入单个空格。 *char* 列将被填充到所定义的长度。
- 在 *varchar* 列中插入的数据中，所有尾随空格都会被删除，除非字符串只包含空格。只包含空格的字符串将截断为单个空格。如果字符串的长度大于 *char*、 *nchar*、 *varchar* 或 *nvarchar* 列的指定长度，则将隐式截断这些字符串，除非 **string_rtruncation** 选项被设置为 **on**。

在 *text* 和 *image* 列中插入

- 如果用 **insert** 将 NULL 插入 *text* 或 *image* 列，既不会创建有效的文本指针，也不会给每个值预分配 2K （而在其它情况下会这样）。而使用 **update** 则可以获取该列的有效文本指针。

插入触发器

- 可以定义一个触发器，使其在对指定表发出 **insert** 命令时执行指定的动作。

在启用组件集成服务后使用 *insert*

- 可以将 **insert** 当作语言事件或参数化的动态语句发送给远程服务器。

插入其它表中的所选行

- 可以从一个表中选择行，然后用一个语句将它们插入同一表中（参见示例 4）。
- 要使用 **select** 将数据从某些域为空值的表中插入不允许空值的表中，必须为初始表的所有 NULL 条目提供替代值。例如，要将数据插入不允许空值的 *advances* 表中，就必须用 0 替代 NULL 域：

  ```
  insert advances
  select pub_id, isnull(advance, 0) from titles
  ```

  如果不使用 **isnull** 函数，该命令就会将带有非空值的所有行插入 *advances* 表，这样将对在 *titles* 表的 *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指定 NOT NULL 的列。

  尽管两个表可以在结构上相同，但它们在某些域是否允许空值方面仍会有所区别。可以使用 **sp_help** 来查看表中各列的空值类型。

事务和 *insert*

- 当设置链式事务模式后，如果当前没有活动的事务， Adaptive Server 将隐式地用 **insert** 语句启动事务。要完成任何插入，必须提交该事务或回退更改。例如：

  ```
  insert stores (stor_id, stor_name, city, state)
      values ('9999', 'Books-R-Us', 'Fremont', 'AZ')
  if exists (select t1.city, t2.city
      from stores t1, stores t2
      where t1.city = t2.city
      and t1.state = t2.state
      and t1.stor_id < t2.stor_id)
          rollback transaction
  else
          commit transaction
  ```

在链式事务模式下，以上批语句启动一个事务并将一个新行插入 *stores* 表中。如果它所插入的行包含与该表中其它商店相同的城市和州信息，则将回退对 *stores* 作出的更改并结束该事务。否则，它将提交插入的数据并结束该事务。有关链式事务模式的详细信息，参见 *Transact-SQL User's Guide*。

在 IDENTITY 列中插入值

- 在表中插入行时，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。如果表中只有 IDENTITY 一列，则应通过如下语句省略列列表并将值列表留为空：

  ```
  insert id_table values()
  ```

- 第一次在某个表中插入行时， Adaptive Server 将把值 1 分配给 IDENTITY 列。每个新行都会获得一个比上一行大 1 的值。该值预先于任何用 **create table** 或 **alter table** 语句为该列声明的缺省值，也预先于任何用 **sp_bindefault** 系统过程绑定到该列的缺省值。

  如果服务器发生故障，则会使 IDENTITY 列的值出现间隔。间隔的最大大小取决于 **identity burning set factor** 配置参数的设置。造成间隔的原因还可能是将数据手工插入 IDENTITY 列，删除行以及事务回退。

- 为列的基表设置 **identity_insert** *table_name* **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。用户一次只能为数据库中的一个表设置 **identity_insert** *table_name* **on**。当 **identity_insert** 设置为 **on** 时，每个 **insert** 语句必须包括一个列列表，并且必须指定 IDENTITY 列的显式值。

  在 IDENTITY 列中插入值后，就可以指定该列的源值或恢复被误删的行。只要未在 IDENTITY 列上创建唯一索引， Adaptive Server 就不会验证值的唯一性，因此可插入任意正整数。

  要在 IDENTITY 列中插入显式值，表所有者、数据库所有者或系统管理员必须为该列的基表 （而不是为用来插入显式值的视图）设置 **identity_insert** *table_name* **on**。

- 可在 IDENTITY 列中插入的最大值是 10 PRECISION -1。当 IDENTITY 列达到该最大值后，所有附加的 **insert** 语句将立即返回错误消息并中止当前的事务。

  一旦出现这种情况，可使用 **create table** 语句创建一个与旧表相同的新表，但新表中的 IDENTITY 列应该具有更大的精度。创建新表之后，使用 **insert** 语句或 **bcp** 实用程序将数据从旧表复制到新表中。

- 使用 *@@identity* 全局变量可检索在 IDENTITY 列中插入的最后一个值。如果最后一个 **insert** 或 **select into** 语句已影响到不带 IDENTITY 列的表， *@@identity* 将返回值 0。
- 选择到结果表中的 IDENTITY 列在继承 IDENTITY 属性时遵守以下规则：
  - 如果多次选择 IDENTITY 列，它将在新表中定义为 NOT NULL。该 IDENTITY 列不继承 IDENTITY 属性。
  - 如果将 IDENTITY 列当作表达式的一部分来选择，结果列将不继承 IDENTITY 属性。当该表达式中的任何列允许空值时，它将创建为 NULL ；否则，将创建为 NOT NULL。
  - 如果 select 语句包含 **group by** 子句或集合函数，结果列将不继承 IDENTITY 属性。包含 IDENTITY 列集合的列将创建为 NULL ；其它列创建为 NOT NULL。
  - 用 union 或 join 选择到表中的 IDENTITY 列不继承 IDENTITY 属性。如果表中包含 IDENTITY 列和 NULL 列的联合，新列将创建为 NULL ；否则，将定义为 NOT NULL。

通过视图插入数据

- 如果视图是用 **with check option** 来创建的，那么通过该视图插入的每一行都必须符合该视图的选择标准。

  例如， *stores_cal* 视图包括 *stores* 表中 *state* 值为 “CA”的所有行。

  ```
  create view stores_cal
  as select * from stores
  where state = "CA"
  with check option
  ```

  with check option 子句对照该视图的选择标准检查每个 **insert** 语句。 *state* 值不是 “CA”的行将被拒绝。

- 如果视图是用 **with check option** 来创建的，从**基**视图派生的所有视图就必须符合基视图的选择标准。通过派生视图插入的每一新行都必须能通过基视图查看。

  以 *stores_cal* 所派生的视图 *stores_cal30* 为例，这一新视图包括位于 California 且付款条件为 “Net 30”的商店的有关信息。

  ```
  create view stores_cal30
  as select * from stores_cal
  where payterms = "Net 30"
  ```

  由于 *stores_cal* 是用 **with check option** 创建的，通过 *stores_cal30* 插入或更新的所有行都必须能通过 *stores_cal* 查看。 *state* 值不是 “CA”的任何行都会被拒绝。

请注意，*stores_cal30* 本身并不具有 **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。即使再也无法通过 *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"
```

- 在用 **with check option** 创建的连接视图中不允许使用 **insert** 语句。
- 如果通过连接视图插入或更新行，所有受影响的列都必须属于同一基表。

为更有效地执行插入操作而将表分区

- 不带有集群索引的未分区表只包含单个双重链接的数据库页链，因此在该表中执行每一 insert 命令都将使用该链的最后一页。Adaptive Server 在插入行的同时持有最后一页的排它锁，以阻止其它并发事务在表中插入数据。

  如果用 **alter table** 命令的 **partition** 子句将表分区，就可以创建更多的页链。每个页链都有最后一页，可用于并发的插入操作。这将减少对页的争用，从而更有效地执行插入操作。当一个表分布于多个物理设备时，分区也会在服务器将数据从缓存刷新到磁盘时减少 I/O 争用，从而更有效地执行插入操作。有关为更有效地执行插入操作而将表分区的详细信息，参见 *Performance and Tuning Guide*。

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	以下内容为 Transact-SQL 扩展： • **insert** 语句的 select 部分中的 **union** 运算符 • 用数据库名称限定表名或列名 • 通过包含连接的视图进行插入 **注意**：通过包含 FIPS 标志程序未检测到的连接的视图进行插入。

权限

缺省情况下，表或视图的所有者具有 **insert** 权限，此所有者可以将其移交给其他用户。

对于表中的 IDENTITY 列， **insert** 权限仅属于表所有者、数据库所有者和系统管理员。

参见

命令	**alter table**、**create default**、**create index**、**create rule**、 **create table**、 **create trigger**、 **dbcc**、 **delete**、 **select**、 **update**
数据类型	“系统和用户定义的数据类型”
系统过程	**sp_bindefault**、 **sp_bindrule**、 **sp_help**、 **sp_helpartition**、 **sp_unbindefault**、 **sp_unbindrule**
实用程序	**bcp**

kill

功能

注销进程。

语法

kill *spid*

关键字和选项

spid — 是要注销的进程的标识号。*spid* 必须是常量，它不能作为参数传递给存储过程，也不能用作局部变量。使用 **sp_who** 可查看进程列表以及其它信息。

示例

1. kill 1378

注释

- 要获取当前进程的报告，可执行系统过程 **sp_who**。下面是一个典型的报告：

```
fid spid  status    loginame origname hostname blk  dbname
      cmd
--- ----- --------   -------- -------- -------- --- ------
      --------------
  0    1  recv sleep  bird     bird     jazzy    0   master
      AWAITING COMMAND
  0    2  sleeping    NULL     NULL              0   master
      NETWORK HANDLER
  0    3  sleeping    NULL     NULL              0   master
      MIRROR HANDLER
  0    4  sleeping    NULL     NULL              0   master
      AUDIT PROCESS
  0    5  sleeping    NULL     NULL              0   master
      CHECKPOINT SLEEP
  0    6  recv sleep  rose     rose     petal    0   master
      AWAITING COMMAND
  0    7  running     robert   sa       helos    0   master
      select
  0    8  send sleep  daisy    daisy    chain    0   pubs2
      select
  0    9  alarm sleep lily     lily     pond     0   master
      WAITFOR
  0   10  lock sleep  viola    viola    cello    7   pubs2
      select
```

spid 列包含在 Transact-SQL **kill** 命令中使用的进程标识号。 *blk* 列包含阻塞进程 （如果有）的进程 ID。阻塞进程当前占有其它进程所需的资源，它可能拥有排它锁。在本例中，进程 10 （对表执行 **select**）被进程 7 （依次对同一表执行 **begin transaction** 和 **insert**）阻塞。

- *status* 列报告命令的状态。以下显示 **sp_who** 的状态值和效果。

表 6-26：sp_who 报告的状态值

状态		*kill* 命令的效果
recv sleep	等待网络读取	立即。
send sleep	等待网络发送	立即。
alarm sleep	等待报警，例如：**waitfor delay "10:00"**	立即。
lock sleep	等待获取锁	立即。
sleeping	等待磁盘 I/O 或其它资源可能表示进程正在运行，但正在执行大量的磁盘 I/O	通常在 “唤醒” 时立即注销。少数处于 sleep 状态的进程不会唤醒，它们需要 Adaptive Server 重新启动才能清除该状态。
runnable	在可运行进程队列中	立即。
running	活跃地运行在某一个服务器引擎上	立即。
infected	Adaptive Server 服务器已检测到严重的错误情况；极其少见	不应使用 **kill** 命令。要清除进程，可能需要重新启动 Adaptive Server。
background	由 Adaptive Server （而不是用户进程）运行的进程，例如阈值过程	立即；使用 **kill** 时必须极其小心。在注销背景进程之前，最好仔细检查 *sysprocesses*。
log suspend	到达日志的最后机会阈值时挂起的进程	立即。

- 要获取对当前锁及持有这些锁的进程 *spid* 的报告，可使用 **sp_lock**。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，系统管理员具有 **kill** 权限，并且不能移交。

参见

命令	**shutdown**
系统过程	**sp_lock**、 **sp_who**

load database

功能

装载用户数据库的备份副本，包括用 **dump database** 创建的事务日志。

语法

```
load database database_name
   from stripe_device [at backup_server_name]
       [density = density_value,
       blocksize = number_bytes,
       dumpvolume = volume_name,
       file = file_name]
   [stripe on stripe_device [at backup_server_name ]
            [density = density_value,
        blocksize = number_bytes,
        dumpvolume = volume_name,
        file = file_name]
   [[stripe on stripe_device [at backup_server_name ]
       [density = density_value,
        blocksize = number_bytes,
        dumpvolume = volume_name,
        file = file_name]]...]
   [with {
       density = density_value,
       blocksize = number_bytes,
       dumpvolume = volume_name,
       file = file_name,
       [dismount | nodismount],
       [nounload | unload],
       listonly [= full],
       headeronly,
       notify = {client | operator_console}
       }]]
```

关键字和选项

database_name — 是将接收备份副本的数据库的名称。它可以是用 **for load** 选项创建的数据库，或是现有的数据库。向现有数据库装载已转储数据将覆盖所有现存数据。接收数据库必须至少等于已转储数据库的大小。可指定数据库名为文字、局部变量或存储过程参数。

from *stripe_device* — 是用来装载数据的设备。有关指定转储设备时要使用的形式的信息，参见 “指定转储设备”。有关支持转储设备的列表，参见 Adaptive Server 安装和配置指南。

at *backup_server_name* — 是在附加了转储设备的计算机上运行的远程 Backup Server 的名称。对于使用接口文件的平台，*backup_server_name* 必须包含在接口文件中。

density = *density_value* — 可忽略。

blocksize = *number_bytes* — 替换转储设备的缺省块大小。**在 OpenVMS 系统上不要指定块大小。**在 UNIX 系统上指定块大小时，它应该与用于转储的块大小相同。

dumpvolume = *volume_name* — 是 ANSI 磁带标签的卷名域。打开磁带时，**load database** 检查该标签。如果装载了错误的卷，就会生成错误消息。

stripe on *stripe_device* — 是额外的转储设备。最多可使用 32 个设备，包括用 **to** *stripe_device* 子句命名的设备。 Backup Server 从所有设备同时装载数据，从而减少了所需要的时间和卷更改的数量。有关如何指定转储设备的信息，参见“指定转储设备”。

dismount | **nodismount** — **在支持逻辑拆卸的平台（如 OpenVMS）上**，确定是否卸下磁带。缺省情况下，在完成装载时，将卸下所有用于装载的磁带。使用 **nodismount** 保持磁带可用于额外的装载或转储。

nounload | **unload** — 确定完成装载后是否回绕磁带。缺省情况下，磁带不回绕，并允许从同一磁带卷进行额外的装载。指定 **unload** 用于要从多转储卷装载的最后一个转储文件。完成装载后，它将回绕并卸载磁带。

file = *file_name* — 是磁带卷上特定数据库转储的名称。如果在进行转储时没有记录转储文件名，则使用 **listonly** 显示所有转储文件的信息。

listonly [**= full**] — 显示磁带卷上所有转储文件的信息，但是**不装载数据库**。**listonly** 标识数据库和设备、转储的日期和时间，以及覆盖转储的日期和时间。**listonly = full** 提供转储的其它信息。这两个报告都按 ANSI 磁带标签进行排序。

列出卷上的所有文件后， Backup Server 发出卷更改请求。操作员可装入其它磁带卷或终止所有转储设备的列示操作。

在当前的实施操作中， **listonly** 选项替换 **headeronly** 选项。

◆ ***警告！***

不要在 1/4 英寸盒式磁带上使用 load database with listonly。

headeronly — 显示单个转储文件的标题信息，但是**不装载数据库**。
headeronly 显示磁带上第一个文件的信息，除非使用 **file =** *file_name* 选项指定了另一个文件名。转储标题指明：

- 转储类型 （数据库或事务日志）
- 数据库 ID
- 文件名
- 转储日期
- 字符集
- 排序顺序
- 页数
- 下一个对象 ID

notify = {**client** | **operator_console**} — 替换缺省消息的目标。

- 在提供操作员终端功能的操作系统上 （如 OpenVMS），卷更改消息总是发送到运行 Backup Server 的计算机的操作员终端上。使用 **client** 将其它 Backup Server 消息传送到启动 **dump database** 的终端会话。
- 在不提供操作员终端功能的操作系统上 （如 UNIX），消息发送到启动 **dump database** 的客户端。使用 **operator_console** 将消息传送到正在运行 Backup Server 的终端。

示例

1. 对于 **UNIX**：

```
load database pubs2
  from "/dev/nrmt0"
```

对于 **OpenVMS**：

```
load database pubs2
  from "MTA0:"
```

从磁带设备重装数据库 *pubs2*。

2. 对于 **UNIX**：

```
load database pubs2
    from "/dev/nrmt4" at REMOTE_BKP_SERVER
    stripe on "/dev/nrmt5" at REMOTE_BKP_SERVER
      stripe on "/dev/nrmt0" at REMOTE_BKP_SERVER
```

对于 **OpenVMS**：

```
load database pubs2
    from "MTA0:"at REMOTE_BKP_SERVER
    stripe on "MTA1:"at REMOTE_BKP_SERVER
    stripe on "MTA2:"at REMOTE_BKP_SERVER
```

使用 Backup Server REMOTE_BKP_SERVER 装载 *pubs2* 数据库。该命令命名三个设备。

注释

- **listonly** 和 **headeronly** 选项显示转储文件的信息但不装载文件。
- 转储和装载均通过 Backup Server 执行。
- 表 6-27 描述了用于从备份恢复数据库的命令和系统过程：

表 6-27：用于从转储恢复数据库的命令

使用此命令	操作
create database for load	为装载转储而创建数据库。
load database	从转储恢复数据库。
load transaction	将最近事务应用于已恢复的数据库。
online database	在完成常规装载序列后，或将数据库升级到 Adaptive Server 的当前版本之后，将数据库变为公用。
load { database | transaction } with {headeronly | listonly}	标识磁带上的转储文件。
sp_volchanged	响应 Backup Server 的卷更改消息。

***load database* 限制**

- 不能装载在不同平台上进行的转储。
- 不能装载在低于版本 10.0 的服务器上生成的转储。
- 如果数据库有跨数据库的参照完整性约束，则 *sysreferences* 系统表存储外部数据库的**名称** — 而不是 ID 号。如果使用 **load database** 更改数据库名称或在不同的服务器上装载数据库， Adaptive Server 将无法确保参照完整性。
- 在每次添加或删除跨数据库约束时，或者是删除含有跨数据库约束的表时，请转储**所有**受影响的数据库。

◆ ***警告！***

装载这些数据库的早期转储会导致数据库损坏。为了用不同名称装载数据库或将其移至另一个 Adaptive Server 上，在转储数据库之前，请使用 alter table 删除所有的外部参照完整性约束。

- **load database** 将从 *master..sysattributes* 中清除与所装载数据库有关的可疑页条目。
- **load database** 覆盖数据库中的任何现有数据。
- 装载数据库转储后，在数据库可使用之前两个进程还需要额外的时间：
  - 如果用 **create database** 的 **for load** 选项创建数据库，在装载完成后数据库中所有未使用的页必须为零。所需时间取决于未使用页的数量。
  - 数据库转储所包含的事务日志中的所有事务必须被回退或前进。所需时间取决于日志中事务的数量和类型。
- 接收的数据库必须等于或大于要装载的数据库。如果接收的数据库太小， Adaptive Server 将显示错误消息并给出所需要的大小。
- 不能从空设备 （UNIX 上的 */dev/null* ； OpenVMS 上任何以“NL”开头的设备名）进行装载。
- 在用户定义事务中不能使用 **load database** 命令。

装载时封锁用户

- 不能使用正在装载的数据库。 **load database** 命令将数据库的状态设为 “脱机”。所有人都不能使用 “脱机”状态的数据库。“脱机”状态防止用户在装载序列期间访问并更改数据库。
- 在发出 **online database** 命令前，用 **load database** 装载的数据库保持不可访问状态。

升级数据库和事务日志转储

- 将用户数据库转储从版本 10.0 或更高版本的服务器恢复并升级到 Adaptive Server 的当前版本：
  1. 装载最近的数据库转储。
  2. **按顺序**装载上一个数据库转储之后的所有事务日志转储。

     Adaptive Server 检查每个转储的时间戳，确保按正确的序列装载到正确的数据库上。
  3. 发布 **online database** 命令升级数据库并使数据库可为公用。
  4. 升级后立即转储刚升级的数据库，创建与 Adaptive Server 的当前版本相一致的转储。

指定转储设备

- 可以将转储设备以文字、局部变量或参数形式指定给一个存储过程。
- 可指定本地设备为：
  - *sysdevices* 系统表中的一个逻辑设备名
  - 绝对路径名
  - 相对路径名

  Backup Server 使用 Adaptive Server 的当前工作目录解析相对路径名。
- 在网络上装载时，指定转储设备的绝对路径名。路径名必须在 Backup Server 运行的计算机上有效。如果名称包括非字母、数字或下划线 (_) 的字符，则用引号将整个名称引起来。
- 转储设备的所有权和权限问题会干扰装载命令的使用。
- 可同时运行多个装载 （或转储），只要每一个装载使用不同的物理设备。

Backup Server

- 必须在与 Adaptive Server 相同的计算机上运行 Backup Server。（在 OpenVMS 系统上， Backup Server 可在与 Adaptive Server 相同的集群中运行，只要所有的数据库设备对这两个服务器都可见。） Backup Server 必须在 *master..sysservers* 表中列出。该条目在安装或升级时创建，且不能删除。
- 如果备份设备位于另一台计算机上，以至于要通过网络进行装载，则远程机器上必须也装有 Backup Server。

卷名

- 转储卷依照 ANSI 磁带标签标准进行标记。标签包括逻辑卷数和分条集内的设备位置。
- 装载时， Backup Server 使用磁带标签来验证卷是否按正确的顺序装入。由此可从少于转储时使用的设备进行装载。

➤ ***注意***

在网络上转储并装载时，对每一个操作必须指定相同数量的分条设备。

更改转储卷

- 如果 Backup Server 检测到当前装入的卷有问题，它通过向客户机或其操作员主控台发送消息来请求卷更改。装入另一个卷后，操作员通过在任何可与 Backup Server 通信的 Adaptive Server 上执行 **sp_volchanged** 系统过程来通知 Backup Server。
- 在 OpenVMS 系统上，当指定设备脱机时，操作系统请求卷更改。装入另一个卷后，操作员使用 REPLY 命令答复卷更改消息。

恢复系统数据库

- 有关从转储恢复系统数据库的逐步说明，参见*系统管理指南*。

磁盘镜像

- 开始装载时， Adaptive Server 向 Backup Server 传递每个逻辑数据库和日志设备的主设备名。如果主设备未镜像， Adaptive Server 则忽略辅助设备的名称。如果在 Backup Server 完成数据传送前任何已命名设备发生故障， Adaptive Server 则中止装载。
- 如果在 **load database** 正在使用时试图对任何已命名的设备解除镜像， Adaptive Server 会显示一条消息。执行 **disk unmirror** 命令的用户可中止装载或推迟 **disk unmirror** 到装载完成之后。
- Backup Server 向主设备装载数据，然后 **load database** 向辅助设备复制数据。如果任何数据库设备已镜像， **load database** 完成的时间就更长。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

只有系统管理员、数据库所有者或具有操作员角色的用户才可以执行 **load database**。

参见

命令	**dbcc**、**dump database**、**dump transaction**、**load transaction**、 **online database**
系统过程	**sp_helpdevice**、 **sp_volchanged**、 **sp_helpdb**

load transaction

功能

装载用 **dump transaction** 命令创建的事务日志的备份副本。

语法

```
load tran[saction] database_name
   from stripe_device [at backup_server_name]
       [density = density_value,
        blocksize = number_bytes,
        dumpvolume = volume_name,
        file = file_name]
   [stripe on stripe_device [at backup_server_name]
       [density = density_value,
        blocksize = number_bytes,
        dumpvolume = volume_name,
        file = file_name]
   [[stripe on stripe_device [at backup_server_name]
       [density = density_value,
        blocksize = number_bytes,
        dumpvolume = volume_name,
        file = file_name]]...]
   [with {
       density = density_value,
       blocksize = number_bytes,
       dumpvolume = volume_name,
       file = file_name,
       [dismount | nodismount],
       [nounload | unload],
       listonly [= full],
       headeronly,
       notify = {client | operator_console}
       until_time = datetime}]]
```

关键字和选项

database_name — 是数据库的名称，该数据库将从事务日志的转储备份副本接收数据。接收数据库的日志段必须至少等于已转储数据库的日志段的大小。可指定数据库名为文字、局部变量或存储过程参数。

from *stripe_device* — 是转储设备的名称，可从该设备装载事务日志。有关指定转储设备时要使用的形式的信息，参见“指定转储设备”。有关支持转储设备的列表，参见 Adaptive Server 安装和配置指南。

at *backup_server_name* — 是在附加了转储设备的计算机上运行的远程 Backup Server 的名称。对于使用接口文件的平台，*backup_server_name* 必须包含在接口文件中。

density = *density_value* — 替换磁带设备的缺省密度。**忽略该选项。**

blocksize = *number_bytes* — 替换转储设备的缺省块大小。在 OpenVMS 系统上不要指定块大小。在 UNIX 系统上指定块大小时，它应该与用于转储的块大小相同。

dumpvolume = *volume_name* — 是 ANSI 磁带标签的卷名域。打开磁带时，**load transaction** 检查该标签，如果装载了错误的卷，就会生成错误消息。

stripe on *stripe_device* — 是额外的转储设备。最多可使用 32 个设备，包括用 **to** *stripe_device* 子句命名的设备。 Backup Server 从所有设备同时装载数据，从而减少了所需要的时间和卷更改的数量。有关如何指定转储设备的信息，参见 “指定转储设备”。

dismount | **nodismount** — **在支持逻辑拆卸的平台 （如 OpenVMS） 上**，确定是否卸下磁带。缺省情况下，在完成装载时，将卸下所有用于装载的磁带。使用 **nodismount** 保持磁带可用于额外的装载或转储。

nounload | **unload** — 确定完成装载后是否回绕磁带。缺省情况下，磁带不回绕，并允许从同一磁带卷进行额外的装载。指定 **unload** 用于要从多转储卷装载的最后一个转储文件。完成装载后，它将回绕并卸载磁带。

file = *file_name* — 是磁带卷上特定数据库转储的名称。如果转储时没有记录转储文件名，则使用 **listonly** 显示所有转储文件的信息。

listonly [**= full**] — 显示磁带卷上所有转储文件的信息，但是**不装载事务日志**。 **listonly** 标识数据库和设备、转储的日期和时间，以及覆盖转储的日期和时间。 **listonly = full** 提供转储的其它信息。这两个报告都按 ANSI 磁带标签进行排序。

列出卷上的所有文件后， Backup Server 发出卷更改请求。操作员可装入其它磁带卷或终止所有转储设备的列表操作。

在当前的实施操作中， **listonly** 选项替换 **headeronly** 选项。

◆ ***警告！***

不要在 1/4 英寸盒式磁带上使用 load transaction with listonly 。

headeronly — 显示单个转储文件的标题信息，但是**不装载数据库**。
headeronly 显示磁带上第一个文件的信息，除非使用 **file =** *file_name* 选项指定了另一个文件名。转储标题指明：

- 转储类型 （数据库或事务日志）
- 数据库 ID
- 文件名
- 转储日期
- 字符集
- 排序顺序
- 页数
- 下一个对象 ID
- 日志中的检查点位置
- 最早的 **begin transaction** 记录的位置
- 新旧序列日期

notify = {**client** | **operator_console**} — 替换缺省消息的目标。

- 在提供操作员终端功能的操作系统上 （如 OpenVMS），卷更改消息总是发送到运行 Backup Server 的计算机的操作员终端上。使用 **client** 将其它 Backup Server 消息传送到启动 **dump database** 的终端会话。
- 在不提供操作员终端功能的操作系统 （如 UNIX）上，消息发送到启动 **dump database** 的客户端。使用 **operator_console** 将消息传送到正在运行 Backup Server 的终端。

until_time — 装载截止于事务日志中指定时间的事务日志。只有指定时间之前提交的事务被保存到数据库中。

示例

1. 对于 **UNIX** ：

```
load transaction pubs2
  from "/dev/nrmt0"
```

对于 **OpenVMS** ：

```
load transaction pubs2
  from "MTA0:"
```

装载数据库 *pubs2* 磁带的事务日志。

2. 对于 **UNIX**：

```
load transaction pubs2
      from "/dev/nrmt4" at REMOTE_BKP_SERVER
    stripe on "/dev/nrmt5" at REMOTE_BKP_SERVER
    stripe on "/dev/nrmt0" at REMOTE_BKP_SERVER
```

对于 **OpenVMS**：

```
load transaction pubs2
  from "MTA0:"at REMOTE_BKP_SERVER
    stripe on "MTA1:"at REMOTE_BKP_SERVER
    stripe on "MTA2:"at REMOTE_BKP_SERVER
```

使用 Backup Server REMOTE_BKP_SERVER 装载 *mydb* 数据库的事务日志。

3.
```
load transaction pubs2
from "/dev/ntmt0"
with until_time = "mar 20, 1997 10:51:43:866am"
```

装载 *pubs2* 的事务日志，截止于 1997 年 3 月 20 日上午 10:51:43:866。

注释

- **listonly** 和 **headeronly** 选项显示转储文件的信息但不装载文件。
- 转储和装载均通过 Backup Server 执行。
- 表 6-28 描述了用于从备份恢复数据库的命令和系统过程：

表 6-28：用于恢复数据库的命令

使用此命令	操作
create database for load	为装载转储而创建数据库。
load database	从转储恢复数据库。
load transaction	将最近事务应用于已恢复的数据库。
online database	在完成常规装载序列后，或将数据库升级到 Adaptive Server 的当前版本之后，将数据库变为公用。
load { database | transaction } with {headeronly | listonly}	标识磁带上的转储文件。
sp_volchanged	响应 Backup Server 的卷更改消息。

load transaction 限制

- 不能装载在不同平台上进行的转储。
- 不能装载在低于版本 10.0 的服务器上生成的转储。
- 数据库和事务日志必须处于同一版本级。
- 按年代顺序装载事务日志。
- 不能从空设备 （UNIX 上的 */dev/null*； OpenVMS 上任何以“NL”开头的设备名）装载。
- 不能在执行升级的 **online database** 命令后使用 **load transaction**。按以下序列升级数据库**是错误的**：**load database**、 **online database**、 **load transaction**。升级数据库的正确序列是 **load database**、 **load transaction**、 **online database**。
- 如果不用升级或更改版本，可在 **online database** 后使用 **load transaction**。
- 在用户定义事务中不能使用 **load transaction** 命令。

恢复数据库

- 要恢复数据库：
  - 装载最近的数据库转储
  - **按顺序**装载上一个数据库转储之后的所有事务日志转储
  - 发布 **online database** 命令使数据库可为公用。
- 在每次添加或删除跨数据库约束时，或者是删除含有跨数据库约束的表时，转储**所有**受影响的数据库。

◆ ***警告！***

装载这些数据库的早期转储会导致数据库损坏。

- 有关备份和恢复 Adaptive Server 数据库的详细信息，参见*系统管理指南*。

将数据库恢复到指定时间

- 对可装载或转储的大部分数据库都可使用 **until_time** 选项。它不适用于数据和日志在同一设备上的数据库，比如 *master*。而且，对于从上一个 **dump database** 起含有被截断日志的任何数据库 （比如 *tempdb*），也不能使用该命令。

- 出于下列原因， **until_time** 选项很有用：
  - 它使数据库与特定时间保持一致。例如，在决策支持系统 (DSS) 数据库和联机事务处理 (OLTP) 数据库的环境中，系统管理员可将 DSS 数据库退回到更早的指定时间，从而比较早期版本和当前版本之间的数据。
  - 如果用户不小心破坏了数据，比如删除了一个重要的表，可使用 **until_time** 选项将数据库往前推至数据被破坏之前的时间，从而撤回错误的命令。
- 要有效地在数据被破坏后使用 **until_time** 选项，必须知道错误发生的确切时间。可在错误发生后立即执行 **select getdate()** 命令来查找时间。在使用毫秒确定更精确的时间时，请使用 **convert** 功能，例如：

  ```
  select convert(char(26), getdate(), 109)

  --------------------------
  Feb 26 1997 12:45:59:650PM
  ```

- 使用 **until_time** 装载事务日志后， Adaptive Server 重新启动数据库日志序列。这意味着再次转储数据库之前，使用了带 **until_time** 的 **load transaction** 命令后，不能装载后续事务日志。在可以转储另一个事务日志之前，需要转储数据库。
- 只有指定时间之前提交的事务被保存到数据库中。而某些情况下，在 **until_time** 说明后立刻提交的事务也应用于数据库数据。这种情况发生在几个事务同时提交的时候。事务的顺序未按时间序列写入事务日志。在这种情况下，时间序列之外的事务将反映在已经恢复的数据中。该时间应该小于一秒。
- 有关恢复到指定时间的数据库的详细信息，参见*系统管理指南*。

装载时封锁用户

- 不能使用正在装载的数据库。与 **load database** 不同， **load transaction** 命令不更改数据库的脱机 / 联机状态。 **load transaction** 保留发现数据库时的状态。 **load database** 命令将数据库设置为 “脱机”。所有人都不能使用 “脱机”状态的数据库。“脱机”状态防止用户在装载序列期间访问并更改数据库。
- 在发出 **online database** 命令前，用 **load database** 装载的数据库保持不可访问状态。

升级数据库和事务日志转储

- 将用户数据库转储从版本 10.0 或更高版本的服务器恢复并升级到 Adaptive Server 的当前版本：

  1. 装载最近的数据库转储。
  2. **按顺序**装载上一个数据库转储之后产生的所有事务日志转储。
  3. 使用 **online database** 进行升级。
  4. 升级后立即转储刚升级的数据库，创建与 Adaptive Server 的当前版本相一致的转储。

指定转储设备

- 可以将转储设备以文字、局部变量或参数形式指定给一个存储过程。
- 从本地设备装载时，可指定转储设备为：
  - 绝对路径名
  - 相对路径名
  - *sysdevices* 系统表中的一个逻辑设备名

  Backup Server 使用 Adaptive Server 的当前工作目录解析相对路径名。
- 在网络上装载时，必须指定转储设备的绝对路径名。（不能使用相对路径名或 *sysdevices* 系统表中的逻辑设备名。）路径名必须在 Backup Server 运行的计算机上有效。如果名称包括任何非字母、数字或下划线 (_) 的字符，都必须用引号将它引起来。
- 转储设备的所有权和权限问题会干扰装载命令的使用。**sp_addumpdevice** 过程向系统表添加设备，但不保证能从该设备装载或创建文件作为转储设备。
- 可同时运行多个装载 （或转储），只要每一个装载使用不同的物理设备。

Backup Server

- 必须在同一计算机上运行 Backup Server 和 Adaptive Server。（在 OpenVMS 系统上， Backup Server 可与 Adaptive Server 在同一集群中运行，只要所有的数据库设备对这两种服务器都可见。） Backup Server 必须在 *master..sysservers* 表中列出。该条目在安装或升级时创建，且不能删除。
- 如果备份设备位于另一台机器，以至于要通过网络进行装载，则远程机器上必须也装有 Backup Server。

卷名

- 转储卷依照 ANSI 磁带标签标准进行标记。标签包括逻辑卷数和分条集内的设备位置。
- 装载时， Backup Server 使用磁带标签来验证卷是否按正确的顺序装入。由此可从少于转储时使用的设备进行装载。

➤ ***注意***

在网络上转储并装载时，对每一个操作必须指定相同数量的分条设备。

更改转储卷

- 如果 Backup Server 检测到当前装入的卷有问题，它通过向客户端或其操作员主控台发送消息来请求卷更改。装入另一个卷后，操作员通过在任何可与 Backup Server 通信的 Adaptive Server 上执行 **sp_volchanged** 来通知 Backup Server。
- 在 OpenVMS 系统上，当指定设备脱机时，操作系统请求卷更改。装入另一个卷后，操作员使用 **REPLY** 命令答复卷更改消息。

恢复系统数据库

- 有关从转储恢复系统数据库的逐步说明，参见*系统管理指南*。

磁盘镜像

- 开始装载时， Adaptive Server 向 Backup Server 传递每个逻辑数据库设备和每个逻辑日志设备的主设备名。如果主设备未镜像， Adaptive Server 则忽略辅助设备的名称。如果在 Backup Server 完成数据传送前任何已命名设备发生故障，则 Adaptive Server 中止装载。
- 如果在 **load transaction** 正在使用时试图对任何已命名的设备解除镜像， Adaptive Server 会显示一条消息。执行 **disk unmirror** 命令的用户可中止装载或推迟 **disk unmirror** 到装载完成之后。
- Backup Server 将数据装载到主设备，然后 **load transaction** 将其复制到辅助设备。如果任何数据库设备已镜像， **load transaction** 完成的时间就更长些。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，数据库所有者和操作员具有 **load transaction** 权限。它不能移交。

参见

命令	**disk unmirror**、 **dump database**、 **dump transaction**、 **load database**、 **online database**
系统过程	**sp_dboption**、 **sp_helpdb**、 **sp_helpdevice**、 **sp_volchanged**

lock table

功能

在事务中显式地锁定表。

语法

```
lock table table_name in {share | exclusive } mode
      [ wait [ numsecs ] | nowait ]
```

关键字和选项

table_name — 指定将被锁定的表的名称。

share | exclusive — 指定将应用到表的锁的类型 （共享或排它）。

wait *numsecs* — 指定如果不能立即获取锁情况下需等待的秒数。如果省略 *numsecs*，则指定 **lock table** 命令等待，直到授予锁。

nowait — 如果不能立即获取锁则让命令失败。

示例

1.
```
begin transaction
lock table titles in share mode
```

   尝试在 *titles* 表上应用共享表锁。如果已使用 **set lock wait** 命令设置会话级等待，则锁定表命令将使用这个等待时间，否则将使用服务器级等待时间。

2.
```
begin transaction
lock table authors in exclusive mode wait 5
```

   尝试在 *authors* 表上应用排它表锁。如果在 5 秒钟之内不能获取锁，命令将返回一条信息性消息。事务内的后续命令将继续进行，就好象没有 **lock table** 命令一样。

3.
```
create procedure bigbatch
as
begin transaction
lock table titles in share mode wait 5
if @@error = 12207
begin
      /*
      ** Allow SA to run without the table lock
      ** Other users get an error message
      */
       if (proc_role("sa_role") = 0)
       begin
```

```
        print "You cannot run this procedure at
           this time, please try again later"
        rollback transaction
        return 100
        end
else
        begin
        print "Couldn't obtain table lock,
           proceeding with default locking."
        end
end
/* more SQL here */
commit transaction
```

如果 5 秒钟之内没有获得表锁，过程将检查用户的角色。如果执行过程的用户具有 **sa_role**，过程将显示一条劝告性消息并且在没有表锁的情况下继续运行。如果用户不具有 **sa_role**，事务将回退。

注释

- 只能在事务中使用 **lock table** 命令。表锁在事务执行期间内存在。
- **lock table** 的作用效果取决于命令中指定的等待时间选项或者会话或服务器级上活动的等待选项。
- 如果没有指定 **wait** 和 **nowait** 选项， **lock table** 命令将使用会话级或服务器级等待时间。如果已使用 **set lock wait** 命令设置会话级等待，则使用会话级等待时间，否则，使用服务器级等待时间。
- 如果不能在限定时间 （如果有）内获取表锁， **lock table** 命令将返回消息 12207。事务不会回退。事务内的后续命令将继续进行，就好象没有 **lock table** 命令一样。
- 不能对系统表或临时表使用 **lock table**。
- 可以在同一事务中使用多个 **lock table** 命令。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

为使用 **lock table in share mode**，必须对表具有 **select** 访问权限。为使用 **lock table in exclusive mode**，必须对表具有 **delete**、 **insert** 或 **update** 访问权限。

参见

命令	**set**

nullif

功能

支持条件 SQL 表达式；在可以使用值表达式的地方都可以使用；是 **case** 表达式的替代形式。

语法

nullif(*expression*, *expression*)

关键字和选项

nullif — 比较两个表达式的值。如果第一个表达式等于第二个表达式，**nullif** 返回 NULL。如果第一个表达式不等于第二个表达式，**nullif** 返回第一个表达式。

expression — 可以是列名、常量、函数、子查询、也可以是任何由算术运算符或逐位运算符连接的列名、常量和函数组合。有关表达式的详细信息，参见第 3 章 “表达式、标识符和通配符”中的“表达式”。

示例

1.
```
select title,
    nullif(type, "UNDECIDED")
from titles
```

从 *titles* 表中选择 *titles* 和 *type*。如果书籍类型是 UNDECIDED，**nullif** 返回值 NULL。

2.
```
select title,
    case
        when type = "UNDECIDED" then NULL
        else type
    end
from titles
```

这是编写示例 1 的替代方法。

注释

- **nullif** 表达式可以替代 **case** 表达式。
- **nullif** 表达式允许使用简单比较来表达搜索条件而不是使用 **when...then** 结构，从而简化了标准 SQL 表达式。
- SQL 中可以使用表达式的地方都可以使用 **nullif** 表达式。
- 至少一个 **case** 表达式的结果必须返回非空值。例如：

```
select price,
    coalesce (NULL, NULL, NULL)
from titles
```

产生下面错误消息：

```
All result expressions in a CASE expression must not be NULL.
(CASE 表达式中的结果表达式不能全为 NULL)
```

- 如果查询产生多种数据类型，数据类型层次将决定 **case** 表达式结果的数据类型，如第 1 章 “系统和用户定义的数据类型”中的“混合型表达式的数据类型”所示。如果指定了两种 Adaptive Server 不能隐式转换的数据类型 （例如， *char* 和 *int*），查询将失败。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都具有 **nullif** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	case、 coalesce、 select、 if...else、 where 子句

online database

功能

在完成常规装载序列后将数据库标记为可公用；如果需要，将装载的数据库升级到 Adaptive Server 的当前版本；在装载了带 **standby_access** 选项转储的事务日志后使数据库联机。

语法

```
online database database_name [for standby_access]
```

参数

database_name — 指定将要使之联机的数据库名。

for standby_access — 假定数据库不包含打开的事务，从而使数据库联机。

示例

1. `online database pubs2`

   在完成装载序列后，使 *pubs2* 数据库可公用。

2. `online database inventory_db for standby_access`

   使数据库 *inventory_db* 联机。使用通过 **dump tran...with standby_access** 转储的事务日志装载完 *inventory_db* 后，使用此命令。

注释

- 在常规数据库或事务日志装载序列后， **online database** 命令将使数据库联机，供常规使用。
- 当发出 **load database** 命令时，将把数据库的状态设置为 “offline”（脱机）。脱机状态是在 *sysdatabases* 系统表中设置的，该设置将保留，直到执行完 **online database** 命令。
- 在装载完所有事务日志之前，**不要**发出 **online database** 命令。命令序列为：
  - **load database**
  - **load transaction** （可能会有多个 **load transaction**）
  - **online database**
- 如果对当前联机的数据库执行 **online database**，将不会进行处理并且不会生成错误消息。

- **online database...for standby_access** 只能与用 **dump transaction...with standby_access** 转储的事务日志一起使用。在装载没有使用 **dump transaction...with standby access** 转储的事务日志之后，如果使用 **online database...for standby_access**， **online database** 命令将生成错误消息并失败。
- 可以使用 **sp_helpdb** 查出数据库是处于当前联机状态、联机用于备用存取状态，还是处于脱机状态。

升级数据库

- 如果需要， **online database** 可以启动已装载数据库和事务日志转储的升级程序，以使数据库与当前版本的 Adaptive Server 兼容。升级完成后，数据库就可以公用了。如果在处理过程中发生错误，数据库将保持脱机状态。
- 只有在数据库或事务日志装载序列后才要求 **online database**。而对于新安装和升级，则不要求。当把 Adaptive Server 升级到新版本时，所有与该服务器相关的数据库都将自动进行升级。
- **online database** 只升级版本 10.0 或更高版本的用户数据库。
- 在用 **online database** 升级数据库后，需要转储刚升级的数据库，以创建与当前版本的 Adaptive Server 一致的转储。必须首先转储已升级的数据库，才能发出 **dump transaction** 命令。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

只有系统管理员、数据库所有者或具有操作员角色的用户才可以执行 **online database**。

参见

命令	**dump database**、 **dump transaction**、 **load database**、 **load transaction**
系统过程	**sp_helpdb**

open

功能

打开要进行处理的游标。

语法

open ***cursor_name***

参数

cursor_name — 是要打开的游标的名称。

示例

```
1. open authors_crsr
```

打开名为 *authors_crsr* 的游标。

注释

- **open** 可以打开一个游标。游标用于分别修改或删除各行。必须首先打开一个游标以便使用 **fetch**、**update** 和 **delete** 语句。有关游标的详细信息，参见 *Transact-SQL User's Guide*。
- 如果游标已经打开或还没有用 **declare cursor** 语句创建游标，Adaptive Server 将返回错误消息。
- 打开游标将导致 Adaptive Server 对定义游标的 **select** 语句求值（在 **declare cursor** 语句中指定），并使游标结果集可供处理。
- 在第一次打开游标时，其位置在游标结果集第一行的前面。
- 当设置链式事务模式后，如果当前没有活动的事务， Adaptive Server 将隐式地用 **open** 语句启动事务。

权限

缺省情况下，所有用户都具有 **open** 权限。

标准和一致性

标准	一致性级别
SQL92	初级一致性

参见

命令	close、 declare cursor、 fetch

order by Clause

功能

按排序顺序在指定列中返回查询结果。

语法

```
[Start of select statement]
[order by {[table_name.| view_name.]column_name
       | select_list_number | expression} [asc | desc]
   [,{[table_name.| view_name.] column_name
         select_list_number|expression} [asc
            |desc]]...]
[End of select statement]
```

关键字和选项

order by — 按列对结果进行排序。

asc — 按升序顺序对结果进行排序。如果没有指定是 **asc** 还是 **desc**，将假定是 **asc**。

desc — 按降序顺序对结果进行排序。

示例

```
1. select title, type, price
   from titles
   where price > $19.99
   order by title
```

```
title
        type         price
------------------------------------------------------------
        ------------ -------------------------
But Is It User Friendly?
         popular_comp                  22.95
Computer Phobic and Non-Phobic Individuals: Behavior Variations
          psychology                    21.59
Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean
         trad_cook                     20.95
Secrets of Silicon Valley
         popular_comp                  20.00
```

选择价格高于 $19.99 的标题并按字母顺序列出。

2.
```
select type, price, advance
from titles
order by type desc
compute avg(price), avg(advance) by type
```

   按类型降序字母顺序列出 *titles* 表中的书籍，并计算每种类型的平均价格和预付款。

3.
```
select title_id, advance/total_sales
from titles
order by advance/total_sales
```

```
 title_id
 -------- ------------------------
 MC3026                       NULL
 PC9999                       NULL
 MC2222                       0.00
 TC4203                       0.26
 PS3333                       0.49
 BU2075                       0.54
 MC3021                       0.67
 PC1035                       0.80
 PS2091                       1.11
 PS7777                       1.20
 BU1032                       1.22
 BU7832                       1.22
 BU1111                       1.29
 PC8888                       1.95
 TC7777                       1.95
 PS1372                      18.67
 TC3218                      18.67
 PS2106                      54.05
```

   列出 *titles* 表中的标题 ID，用预付款除以总销售额，并按照从最小计算值到最大计算值的顺序排序。

4.
```
select title as BookName, type as Type
from titles
order by Type
```

   按类型顺序列出书籍标题和类型，重命名输出中的列。

注释

- **order by** 将按排序顺序在指定列中返回查询结果。 **order by** 是 **select** 命令的一部分。
- 在 Transact-SQL 中，可以使用 **order by** 对没有显示在选择列表中的项进行排序。排序的标准可以是列标题、列名、表达式、别名（如果在选择列表中进行了指定）或表示项在选择列表中的位置的编号 (*select_list_number*)。
- 如果按 *select_list_number* 排序，则 *order by* 子句引用的列必须包含在 **select** 列表中，并且 **select** 列表不能是 * （星号）。
- 使用 **order by** 可以按有意义的顺序显示查询结果。如果没有 **order by** 子句，将无法控制 Adaptive Server 返回结果的顺序。

限制

- **order by** 子句中允许的最大列数是 31。
- **order by** 子句指定的所有列的最大长度的和不能超过 2014 个字节。
- **order by** 不能用在 *text* 或 *image* 数据类型列上。
- 子查询和视图定义中不能包含 **order by** 子句 （或 **compute** 子句或关键字 **into**）。相反地，不能在 **order by** 列表中使用子查询。
- 如果服务器类型或语言类型游标在其 **select** 语句中包含 **order by** 子句，将无法更新其结果集。有关适用于可更新游标的限制的详细信息，参见 *Transact-SQL User's Guide*。
- 如果使用 **compute by**，就必须使用 **order by** 子句。列在 **compute by** 后面的表达式必须与列在 **order by** 后面的表达式相同或是其子集，其从左向右的顺序必须相同，必须以同一表达式开始且不跳过任何表达式。例如，如果 **order by** 子句是：

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

  compute by 子句可以是下列任意 （或全部）形式：

  ```
  compute by a, b, c
  compute by a, b
  compute by a
  ```

  可以在没有 **by** 的情况下使用关键字 **compute** 生成总和、总计数等。在这种情况下， **order by** 是可选的。

归类序列

- 利用 **order by**，可以将空值排在所有其它值的前面。
- Adaptive Server 上的排序顺序 （归类序列）决定数据排序的方式。排序顺序的选项有二进制、字典、不区分大小写、具有优先级的不区分大小写以及不区分大小写和变音。可能还提供针对特定国家语言的排序顺序。

表 6-29：排序顺序选择的效果

Adaptive Server 排序顺序	*order by* 结果的效果
二进制排序	根据每个字符在字符集中的数值字节值对所有数据进行排序。二进制顺序将所有大写字母排在小写字母之前。二进制排序顺序是多字节字符集的唯一选项。
字典顺序	将大写字母排在其小写形式之前 （区分大小写）。字典顺序可以识别字母的各种变音形式并将其排在非变音形式之后。
字典顺序，不区分大小写	将数据按字典顺序排序但不区分大小写。大写字母等同于其小写形式并按照 “排序规则”中的说明进行排序。
字典顺序，具有优先级的不区分大小写	将大写字母排在其小写形式前的首选位置。在执行比较 （如在 where 子句中）时，将不区分大小写。
字典顺序，不区分大小写和变音	按字典顺序对数据排序，但不区分大小写；字母的变音形式与其相关的非变音形式等同对待。在排序结果中，变音形式和非变音形式混合在一起。

- 系统过程 **sp_helpsort** 报告安装在 Adaptive Server 上的排序顺序。

排序规则

- 当两行的 Adaptive Server 排序顺序值等同时，将使用以下规则对行进行排序：
  - 比较在 **order by** 子句中指定的列中的值。
  - 如果两行具有等同的列值，将逐字节比较整个行的二进制值。这种比较将在行上以列在内部存储的顺序执行，而不是以列在查询或在原始 **create table** 子句中指定的顺序执行。（总之，数据将按照这样的顺序存储：所有按顺序排列的固定长度列，然后是所有按顺序排列的可变长度列。）
  - 如果行等同，将比较行 ID。

给定下表：

```
create table sortdemo (lname varchar(20),
                       init char(1) not null)
```

和以下数据：

```
lname      init
---------- ----
Smith      B
SMITH      C
smith      A
```

当按照 *lname* 顺序排序时，将得到如下结果：

```
lname      init
---------- ----
smith      A
Smith      B
SMITH      C
```

由于固定长度 *char* 数据 （*init* 列）在内部存储时排在前面，所以 **order by** 根据二进制值 “Asmith”、“BSmith”和 “CSMITH” 对这些行进行排序。

但是，如果 *init* 属于类型 *varchar*，将首先存储 *lname* 列，然后才存储 *init* 列。比较将根据二进制值 “SMITHC”、“SmithB”和 “smithA”进行并按该顺序返回行。

降序扫描

- 在 **order by** 子句中使用关键字 **desc**，可以使查询优化程序选择这样一种策略：无需工作表和排序步骤就可以按降序顺序返回结果。这种优化行为通过跟随每个索引页前一页上的指针，实现按相反顺序扫描索引的页链。

  为了使用这种优化行为， **order by** 子句中的列必须与索引顺序匹配。它们可以是键的子集，但必须是前缀子集，也就是说，它们必须包含第一个键。如果 **order by** 子句中指定的列是索引键的超集，则不能使用降序扫描优化。

  如果查询涉及到连接，只要符合键的前缀子集要求，就可以按降序键顺序扫描所有表。降序扫描优化还可以在连接中的其它表按升序扫描的同时，用于其中的一个或多个表。

- 如果其他用户进程正在向前扫描执行更新或删除，执行降序扫描可能会导致死锁。在页拆分或收缩过程中，也可能会遇到死锁。可以使用系统过程 **sp_sysmon** 在服务器上跟踪死锁，也可以使用配置参数 **print deadlock information** 将死锁信息发送到错误日志。

- 如果应用程序需要按降序返回结果，但降序扫描优化将导致死锁问题，以下是解决该矛盾的一些可能的方法：
  - 对降序扫描使用事务隔离级别 0 扫描。有关隔离级别 0 读取效果的详细信息，参见 *Performance and Tuning Guide*。
  - 用配置参数 **allow backward scans** 禁用降序扫描优化，这样，所有使用 **desc** 的查询都将以升序扫描表并将结果集按降序排列。有关详细信息，参见*系统管理指南*。
  - 将有问题的降序扫描拆分为两个步骤：第一步，将要求的行以升序选择到临时表中；第二步，以降序从临时表中选择。
- 如果由于出现重复的键值，反向扫描使用包含溢出页的集群索引，降序扫描返回的结果集的顺序可能不会与用升序扫描返回的结果集的顺序恰好相反。指定的键值是按顺序返回的，但溢出页上相同键的行的顺序可能会不同。有关溢出页在集群索引中的存储方法的解释，参见 *Performance and Tuning Guide*。

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	当 **union** 运算符被用作 Transact-SQL 扩展时，在选择语句的 **order by** 子句中指定新的列标题。

参见

命令	**compute Clause**、 **declare**、 **group by 和 having 子句**、 **select**、 **where 子句**
系统过程	**sp_configure**、 **sp_helpsort**、 **sp_lock**、 **sp_sysmon**

prepare transaction

功能

由 DB-Library 在两阶段提交应用中用来查看服务器是否已准备好提交事务。

语法

```
prepare tran[saction]
```

注释

- 有关详细信息，参见 *Open Client DB-Library Reference Manual*。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

参见

命令	begin transaction、 commit、 rollback、 save transaction

print

功能

在用户屏幕上输出用户定义的消息。

语法

```
print
   {format_string | @local_variable |
   @@global_variable}
       [, arg_list]
```

关键字和选项

format_string — 既可以是变量，也可以是字符串。 *format_string* 的最大长度是 255 个字节。

格式字符串最多可以包含 20 个唯一占位符，它们的顺序可以是任意的。当消息文本发送到客户端时，将用 *format_string* 后面跟随的参数的格式化内容替换这些占位符。

对这些占位符进行编号，这样当用其它语法结构将格式字符串翻译为某种语言时，就可以对参数重新排序了。参数占位符的显示形式如下：“%*nn*!”— 一个百分比符号 (%) 后面跟着一个 1 到 20 之间的整数，再后面跟着一个感叹号 (!)。该整数表示参数列表中字符串中的参数编号。“%1!”表示原始版本的第一个参数，“%2!”表示第二个参数，依此类推。

用这种方法指示参数的位置可以使翻译正确，即使在目标语言中参数的出现顺序不同。

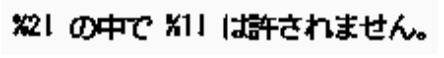
例如，假定以下是一条英文消息：

```
%1! is not allowed in %2!.
```

该消息的德文版本是：

```
%1! ist in %2! nicht zulassig.
```

该消息的日文版本是：



在本示例中，“%1!”在所有三种语言中表示同一参数，“%2!”也是如此。本示例显示了有时在翻译的形式中需要对参数重新排序。

@local_variable — 必须是类型 *char*、 *nchar*、 *varchar* 或 *nvarchar*，而且必须在使用它的批处理或过程中声明。

@@global_variable — 必须是类型 *char* 或 *varchar*，或者可以自动转换为这两种类型，如 *@@version*。当前， *@@version* 是唯一的字符型全局变量。

arg_list — 可以是一系列由逗号分隔开的变量或常量。 *arg_list* 是可选的，除非提供了包含 *"%nn!"* 格式占位符的格式字符串。在这种情况下， *arg_list* 具有的参数数量必须至少与编号最高的占位符相等。参数可以是除 *text* 或 *image* 以外的任何数据类型；在被包含在最终的消息中之前，参数将被转换为字符数据类型。

示例

1. ```
   if exists (select postalcode from authors
   where postalcode = '94705')
   print "Berkeley author"
   ```

   如果 *authors* 表中的任何作者居住在邮政编码为 94705 的地区，则输出 "Berkeley author"。

2. ```
   declare @msg char(50)
   select @msg = "What's up, doc?"
   print @msg
   ```

   ```
   What's up, doc?
   ```

   声明变量、为该变量指派值并输出该值。

3. ```
   declare @tabname varchar(30)
   select @tabname = "titles"

   declare @username varchar(30)
   select @username = "ezekiel"

   print "The table '%1!' is not owned by the user
   '%2!'.", @tabname, @username
   ```

   ```
   The table 'titles' is not owned
   by the user 'ezekiel.'
   ```

   演示消息中变量和占位符的使用方法。
   ```

注释

- 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。
- 如果在格式字符串中使用占位符，请记住：对于字符串中的每个占位符 *n*，占位符 1 直到 *n*-1 也必须在同一字符串内，但这些占位符不必按数值顺序排列。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 也放在该字符串中。如果在格式字符串中省略了一个编号，当执行 **print** 时，将生成错误消息。
- *arg_list* 中必须包含 *format_string* 中每个占位符的参数，否则事务将中止。允许出现参数多于占位符的情况。
- 要使百分比符号成为错误消息的一部分，可以在 *format_string* 中使用两个百分比符号 （“%%”）。如果 *format_string* 中包含不用作占位符的单个百分比符号 （“%”）， Adaptive Server 将返回错误消息。
- 如果参数的求值为 NULL，将把该参数转换为零长度字符串。如果不希望在输出中出现零长度字符串，可以使用 **isnull** 函数。例如，如果 *@arg* 的求值为 null，以下语句：

  ```
  declare @arg varchar(30)
  select @arg = isnull(col1, "nothing") from
  table_a where ...

  print "I think we have %1! here", @arg
  ```

  将输出：

  ```
  I think we have nothing here.
  ```

- 可以将用户定义的消息添加到系统表 *sysusermessages* 中，供任何应用程序使用。使用 **sp_addmessage** 可以将消息添加到 *sysusermessages* 中；使用 **sp_getmessage** 可以检索供 **print** 和 **raiserror** 使用的消息。
- 如果要输出用户定义的错误消息并将错误编号存储在 *@@error* 中，可以使用 **raiserror**，而不要使用 **print**。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都具有 **print** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	**declare**、 **raiserror**
系统过程	**sp_addmessage**、 **sp_getmessage**

quiesce database

功能

挂起并重新开始对指定数据库列表进行的更新。

语法

```
quiesce database tag_name hold dbname [, dbname] ...
```

或

```
quiesce database tag_name release
```

关键字和选项

tag_name — 是将数据库列表指派到 **hold** 或 **release** 的用户定义的名称。 *tag_name* 必须遵循标识符的规则。

database — 是数据库名。

示例

1. `quiesce database report_dbs hold salesdb, ordersdb`

   将对 *salesdb* 和 *ordersdb* 的更新活动挂起。

2. `quiesce database report_dbs release`

   重现开始对标记为 *report_dbs* 的数据库的更新活动。

注释

- 在挂起对其它用户设备的更新**之后**，再挂起对 *master* 数据库的更新。如果 *master* 处于挂起状态，尝试挂起其它用户数据库将生成警告消息。
- 如果 *master* 数据库处于挂起状态，在释放其它用户数据库之前，应该先释放 *master* 来执行更新。如果尝试在 *master* 处于挂起状态下释放用户数据库， Adaptive Server 将显示警告消息。
- **quiesce database** 在与 **hold** 关键字一起使用时，将挂起对指定数据库的所有更新。事务无法更新挂起数据库中的数据，后台任务 （如检查点进程和管家进程）将跳过所有处于挂起状态的数据库。
- **quiesce database** 在与 **release** 关键字一起使用时，将允许恢复对以前处于挂起状态的数据库进行的更新。
- 不需要从同一用户会话执行 **quiesce database hold** 和 **release** 命令。

- 如果 **quiesce database hold** 命令中指定的数据库包含处于就绪状态的分布式或多数据库事务，Adaptive Server 将等待五秒的超时周期，以便使这些事务完成。如果事务在超时周期内没有完成，**quiesce database hold** 将失败。
- 在挂起对这些数据库的更新之前，必须对每个数据库至少执行一次 **dump database** 命令。
- 如果 Adaptive Server 正对在 **quiesce database hold** 中指定的数据库执行 **dump database** 或 **dump transaction** 命令，则只有在 **dump** 命令完成后才能挂起该数据库。
- 如果在对数据库的更新处于挂起状态的同时对该数据库执行 **dump database** 或 **dump transaction** 命令， Adaptive Server 将阻塞这些命令，直到用 **quiesce database release** 释放该数据库为止。
- 在一条 **quiesce database hold** 命令中，最多可以指定八个数据库。如果必须挂起对其它数据库的更新，可以再执行 **quiesce database hold** 命令。

权限

缺省情况下， **quiesce database** 权限属于系统管理员。

参见

命令	**dump database**、 **dump transaction**
系统过程	**sp_helpdb**、 **sp_who**

raiserror

功能

在用户屏幕上输出用户定义的错误消息，并通过设置系统标识来记录已发生错误情况。

语法

```
raiserror error_number
   [{format_string | @local_variable}] [, arg_list]
   [with errordata restricted_select_list]
```

关键字和选项

error_number — 是值大于 17,000 的局部变量或整数。如果 *error_number* 介于 17,000 和 19,999 之间，而 *format_string* 缺失或为空 (“”)， Adaptive Server 将在 *master* 数据库的 *sysmessages* 表中检索错误消息。这些错误消息主要供系统过程使用。

如果 *error_number* 大于或等于 20,000，而 *format_string* 缺失或为空，**raiserror** 将在 *sysusermessages* 表中检索消息文本，而该表所位于的数据库正是启动查询或存储过程的数据库。 Adaptive Server 尝试在 *sysmessages* 或 *sysusermessages* 中检索消息，而检索使用的语言由 *@@langid* 的当前设置定义。

format_string — 是最大长度为 255 个字节的字符串。可以选择在局部变量中声明 *format_string* 并在 **raiserror** 中使用该变量 （参见 *@local_variable*）。

raiserror 可以识别出要输出字符串中的占位符。格式字符串最多可以包含 20 个唯一占位符，它们的顺序可以是任意的。当消息文本发送到客户端时，将用 *format_string* 后面跟随的参数的格式化内容替换这些占位符。

对这些占位符进行编号，这样当用其它语法结构将格式字符串翻译为某种语言时，就可以对参数重新排序了。参数占位符的显示形式如下：“*%nn*!”— 一个百分比符号 (%) 后面跟着一个 1 到 20 之间的整数，再后面跟着一个感叹号 (!)。该整数表示参数列表中字符串中的参数编号。“%1!”表示原始版本的第一个参数，“%2!”表示第二个参数，依此类推。

用这种方法指示参数的位置可以使翻译正确，即使在目标语言中参数的出现顺序与其在源语言中的顺序不同。

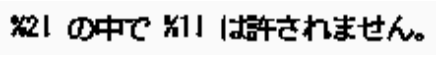
例如，假定以下是一条英文消息：

```
%1! is not allowed in %2!.
```

该消息的德文版本是：

```
%1! ist in %2! nicht zulassig.
```

该消息的日文版本是：



在本示例中，“%1!”在所有三种语言中表示同一参数，而“%2!”也是如此。本示例显示了有时在翻译的形式中需要对参数重新排序。

@local_variable — 是包含 *format_string* 值的局部变量。它必须是类型 *char* 或 *varchar* 并且必须在使用它的批处理或过程内声明。

arg_list — 是一系列由逗号分隔开的变量或常量。 *arg_list* 是可选的，除非提供了包含 “*%nn*!” 格式占位符的格式字符串。参数可以是除 *text* 或 *image* 以外的任何数据类型；在被包含在最终的字符串中之前，参数将被转换为 *char* 数据类型。

如果参数的求值为 NULL， Adaptive Server 将把该参数转换为零长度 *char* 字符串。

with errordata — 为 Client-Library™ 程序提供扩展错误数据。

restricted_select_list — 由下列一项或多项组成：

- “*”，它表示 **create table** 顺序中的所有列。
- 列名的列表，其顺序是您查看它们时所希望的顺序。在选择现有 IDENTITY 列时，可以用 **syb_identity** 关键字 （如有必要，用表名加以限定）替换实际的列名。
- 将新的 IDENTITY 列添加到结果表的说明：

  ***column_name* = identity(*precision*)**

- 缺省列标题 （列名）的替换，其格式为：

  column_heading* = *column_name

  或：

  column_name column_heading

  或：

  column_name* as *column_heading

  所有这些格式的列标题都可以用引号引起来。如果标题不是有效的标识符 （即，标题是保留字、以特殊字符开头或标题中包含空格或标点符号），则必须用引号将其引起来。

- 表达式（列名、常量、函数、任何由算术运算符或逐位运算符连接的列名、常量和函数组合或子查询）。
- 内部函数或集合
- 上述项的任意组合

restricted_select_list 还可以执行变量赋值，其格式是：

```
@variable = expression
    [, @variable = expression ...]
```

restricted_select_list 的限制如下：

- 不能将变量赋值与任何其它 *restricted_select_list* 选项结合使用。
- 不能在 *restricted_select_list* 中使用 **from**、**where** 或其它 **select** 子句。
- 不能使用 “*” 表示 *restricted_select_list* 中的所有列。

详细信息，参见 *Transact-SQL User's Guide*。

示例

1.
```
create procedure showtable_sp @tabname varchar(18)
as
if not exists (select name from sysobjects
    where name = @tabname)
    begin
      raiserror 99999 "Table %1! not found.",
        @tabname
    end
else
    begin
      select sysobjects.name, type, crdate, indid
      from sysindexes, sysobjects
      where sysobjects.name = @tabname
      and sysobjects.id = sysindexes.id
end
```

如果该存储过程示例没有找到随 *@tabname* 参数提供的表，它将返回错误。

2.
```
sp_addmessage 25001,
"There is already a remote user named '%1!'
for remote server '%2!'."

raiserror 25001, jane, myserver
```

该示例将消息添加到 *sysusermessages* 中，然后用 **raiserror** 检测该消息，同时提供替代参数。

```
3. raiserror 20100 "Login must be at least 5
     characters long" with errordata "column" =
       "login", "server" = @@servername
```

该示例使用 **with errordata** 选项将扩展错误数据 *column* 和 *server* 返回到客户端应用，从而指示涉及了哪个列和使用了哪个服务器。

注释

- 用户定义的消息可以即席生成，如示例 1 和 3 所示，也可以将它们添加到系统表 *sysusermessages* 中，以供任何应用程序使用，如示例 2 所示。使用 **sp_addmessage** 可以将消息添加到 *sysusermessages* 中；使用 **sp_getmessage** 可以检索供 **print** 和 **raiserror** 使用的消息。
- 用户定义的错误消息的编号必须大于 20,000。其最大值是 2,147,483,647 (2^{31} -1)。
- 所有用户定义的消息的严重级是 16。该级别表示用户已经犯了非致命错误。
- 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。
- 如果在格式字符串中使用占位符，请记住：对于字符串中的每个占位符 *n*，占位符 *1* 直到 *n-1* 也必须在同一字符串内，但这些占位符不必按数值顺序排列。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 也放在该字符串中。如果在格式字符串中省略了一个编号，当执行 **raiserror** 时，将生成错误消息。
- 如果与 *format_string* 中占位符数量相关的参数数量太少，将显示错误消息并且事务将中止。允许在 *format_string* 中出现参数多于占位符的情况。
- 要使百分比符号成为错误消息的一部分，可以在 *format_string* 中使用两个百分比符号 （“%%”）。如果 *format_string* 中包含不用作占位符的单个百分比符号 （“%”）， Adaptive Server 将返回错误消息。
- 如果参数的求值为 NULL，将把该参数转换为零长度 *char* 字符串。如果不希望在输出中出现零长度字符串，可以使用 **isnull** 函数。
- 在执行 **raiserror** 时，将把错误编号放置在全局变量 *@@error* 中，该全局变量存储系统最近生成的错误编号。
- 如果希望将错误编号存储在 *@@error* 中，可以使用 **raiserror**，而不要使用 **print**。

- 要在 **raiserror** 中包含 *arg_list*，可以在 *error_number* 或 *format_string* 之后、第一个参数之前加一个逗号。要包含扩展错误数据，可以用空格 （不是逗号）将第一个 *extended_value* 与 *error_number*、*format_string* 或 *arg_list* 分隔开。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都具有 **raiserror** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	declare、 print
系统过程	sp_addmessage、 sp_getmessage

readtext

功能

读取 *text* 和 *image* 值，从指定的偏移开始读取并读取指定的字节数或字符数；如果与 **readpast** 一起使用，将跳过有排它锁的行，不等待也不生成消息。

语法

```
readtext [[database.]owner.]table_name.column_name
   text_pointer offset size
   [holdlock | noholdlock] [readpast]
   [using {bytes | chars | characters}]
   [at isolation {
       [ read uncommitted | 0 ] |
       [ read committed | 1 ] |
       [ repeatable read | 2 ]|
       [ serializable | 3 ] } ]
```

关键字和选项

table_name.column_name — 是 *text* 或 *image* 列的名称。必须将表名包含在内。如果该表位于另一数据库中，则指定数据库名；如果该数据库中存在多个与之同名的表，指定所有者的名称。 *owner* 的缺省值是当前用户，而 *database* 的缺省值是当前数据库。

text_pointer — 是一个 *varbinary*(16) 值，其中存储了指向 *text* 或 *image* 数据的指针。使用 **textptr** 函数可以确定该值 （参见示例 1）。 *text* 和 *image* 数据不与其它表列存储在相同的链接页集合中。而是存储在一个独立的链接页集合中。指向实际位置的指针与数据一起存储； **textptr** 返回该指针。

offset — 指定在开始读取 *text* 或 *image* 数据之前，需要跳过的字节数或字符数。

size — 指定要读取数据的字节数或字符数。

holdlock — 导致文本值被锁定为供读取，直到事务结束为止。其他用户可以读取该值，但不能修改该值。

noholdlock — 避免服务器持有执行该语句过程中获取的锁，而不考虑当前有效的事务隔离级别。不能在查询中既指定 **holdlock**，又指定 **noholdlock** 选项。

readpast — 指定 **readtext** 应该自动跳过具有排它锁的行，不等待也不生成消息。

using — 指定 **readtext** 是将 *offset* 和 *size* 参数解释为多个字节 **(bytes)**，还是将其解释为多个 **textptr** 字符 （**chars** 或 **characters** 是同义的）。当用于单字节字符集或用于 *image* 值 （**readtext** 逐字节读取 *image* 值）时，该选项无效。如果没有给出 **using**， **readtext** 将 *size* 和 *offset* 参数解释为字节。

at isolation — 指定查询的隔离级别 （0、 1 或 3）。如果省略该子句，查询将使用其在执行时所位于的会话的隔离级别 （缺省情况下为隔离级别 1）。如果指定 **holdlock** 的查询也指定了 **at isolation read uncommitted**， Adaptive Server 将发出警告并忽略 **at isolation** 子句。对于其它隔离级别， **holdlock** 优先于 **at isolation** 子句。

read uncommitted — 指定查询的隔离级别为 0。可以指定 0，来代替用 **at isolation** 子句指定 **read uncommitted**。

read committed — 指定查询的隔离级别为 1。可以指定 “1”，来代替用 **at isolation** 子句指定 **read committed**。

repeatable read — 指定查询的隔离级别为 2。可以指定 “2”，来代替用 **at isolation** 子句指定 **repeatable read**。

serializable — 指定查询的隔离级别为 3。可以指定 “3”，来代替用 **at isolation** 子句指定 **serializable**。

示例

1.
```
declare @val varbinary(16)
select @val = textptr(copy) from blurbs
where au_id =   "648-92-1872"
readtext blurbs.copy @val 1 5 using chars
```

   在 *copy* 列中选择第二个到第六个字符。

2.
```
declare @val varbinary(16)
select @val = textptr(copy) from blurbs readpast
where au_id = "648-92-1872"
readtext blurbs.copy @val 1 5 readpast using chars
```

注释

- **textptr** 函数将把 16 字节二进制字符串 （文本指针）返回到指定行的 *text* 或 *image* 列，或返回到查询返回的最后一行的 *text* 或 *image* 列 （如果返回了多行）。最好先声明局部变量持有该文本指针，然后将通过 **readtext** 使用该变量。
- 全局变量 *@@textsize* 中的值，是对要返回数据字节数的限制，如果该值小于为 **readtext** 指定的大小，它将替代后者。使用 **set textsize** 可以更改 *@@textsize* 的值。

- 在使用字节数作为偏移和大小时，Adaptive Server 可能会在要返回的 *text* 数据的开头和末尾找到部分字符。如果出现这种情况，并且字符集转换处于打开状态，服务器在将文本返回到客户端之前，将用问号 (?) 替代每个部分字符。
- Adaptive Server 必须确定要发送到客户端的字节数，来响应 **readtext** 命令。当 *offset* 和 *size* 用字节数表示时，确定返回文本中的字节数是很简单的。当偏移和大小用字符数表示时，服务器必须另外执行一步，来计算返回到客户端的字节数。因此，在使用字符数表示 *offset* 和 *size* 时，执行速度可能会较慢。只有在 Adaptive Server 使用多字节字符集时，才会用到 **using characters** 选项：该选项可以确保 **readtext** 不会返回部分字符。
- 不能对视图的 *text* 和 *image* 列使用 **readtext**。
- 如果在更改为多字节字符集后，试图对 *text* 值使用 **readtext**，而没有运行 **dbcc fix_text**，那么命令将会失败，并会出现错误消息，指导您对该表运行 **dbcc fix_text**。

使用 readpast 选项

- **readpast** 选项仅应用于 DOL 锁定表。如果为 allpage 锁定表指定 **readpast**，则将忽略 **readpast**。
- **readpast** 选项与 **holdlock** 选项不兼容。如果在命令中同时指定这两个选项，将产生一个错误并且命令终止。
- 如果 **readtext** 命令指定 **at isolation read uncommitted**，则 **readpast** 选项产生一个警告，但不终止此命令。
- 如果语句隔离级别被设置为 3，**readpast** 选项将产生错误并终止命令。
- 如果会话范围隔离级别为 3，则忽略 **readpast** 选项。
- 如果会话范围隔离级别为 0，**readpast** 选项将产生一个警告，但不终止命令。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

readtext 要求对表有 **select** 权限。移交 **select** 权限时也移交 **readtext** 权限。

参见

命令	**set**、 **writetext**
数据类型	text 和 image 数据类型

reconfigure

功能

reconfigure 命令当前不具有任何作用；加入该命令是为了使现有脚本不经修改即可运行。在以前的版本中，执行系统过程 **sp_configure** 之后需要使用 **reconfigure** 来实施新的配置参数设置。

语法

```
reconfigure
```

注释

➤ ***注意***

如果有包含 **reconfigure** 的脚本，请尽快更改它们。虽然该版本中包含 **reconfigure**，但随后的版本可能不支持该命令。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，系统管理员具有 **reconfigure** 权限，并且不能移交。

参见

系统过程	sp_configure

remove java

功能

从数据库中删除一个或多个 Java-SQL 类、软件包或 JAR。

在数据库中安装了 Java 时使用。有关详细信息，参见 *Adaptive Server Enterprise 中的 Java*。

语法

```
remove java
   class class_name [, class_name]...
   | package package_name [, package_name]...
   | jar jar_name [, jar_name]...[retain classes]
```

参数

class *class_name* — 要从数据库中删除的一个或多个 Java 类的名称。这些类必须安装在当前数据库中。

package *package_name* — 要删除的一个或多个 Java 软件包的名称。这些软件包必须存储在当前数据库中。

jar *jar_name* — SQL 标识符或包含有效 SQL 标识符的字符串值（最长可为 30 个字节）。

每个 *jar_name* 都必须等于当前数据库中的保留 jar 的名称。

retain classes — 指定命名的 JAR 不再保留在数据库中，并且保留类没有关联的 JAR。

注释

- 如果存储过程中包含 **remove java** 语句，当前数据库将是创建过程时的当前数据库，而不是调用过程时的当前数据库。

  如果存储过程中不包含 **remove java** 语句，当前数据库将是执行 **remove** 语句时的当前数据库。
- 如果指定了 **class** 或 **package**，并且任何删除的类都有关联的 JAR，则将出现例外。
- 如果任何存储过程、表或视图中包含对删除类的引用，将其作为列、变量或参数的数据类型，那么将会出现例外。

- 所有删除的类都将
  - 从当前数据库中删除。
  - 从当前连接的 Java 虚拟机 (Java VM) 上卸载。删除的类不会从其它连接的 Java VM 上卸载。
- 如果在执行 **remove java** 的过程中出现了任何例外，将取消所有 **remove java** 操作。

锁

- 在使用 **remove java** 时，将为 *sysxtypes* 加上一个排它表锁。
- 如果指定了 **jar**，将为 *sysjars* 加上一个排它表锁。

权限

只有系统管理员或数据库所有者可以使用 **remove java**。

参见

系统过程	**sp_helpjava**
系统表	*sysjars*、 *sysxtypes*
实用程序	**extractjava**、 **installjava**

reorg

功能

根据使用的选项，回收页面上未使用的空间、删除行转移或将表中的所有行重新写入新页。

语法

```
reorg reclaim_space tablename [indexname]
   [with {resume, time = no_of_minutes}]

reorg forwarded_rows tablename
   [with {resume,time = no_of_minutes}]

reorg compact tablename
   [with {resume, time = no_of_minutes}]

reorg rebuild [ tablename | indexname ]
```

参数和关键字

reclaim_space — 回收由删除和更新操作所留下的未使用空间。对于表中的各个数据页，如果有已提交删除或行缩短更新操作产生的未使用空间，**reorg reclaim_space** 将连续重写当前这些行，使所有未使用空间保留在页尾。如果页中已没有行，则释放该页。

tablename — 指定要重组的表的名称。如果指定了 *indexname*，则仅重组索引。

indexname — 指定将要重组的索引的名称。

with resume — 从前一 **reorg** 命令结束位置点开始重组。当前一 **reorg** 命令指定了时间限制 (**time =** *no_of_minutes*) 时，使用该选项。

with time = *no_of_minutes* — 指定 **reorg** 命令运行的分钟数。

forwarded_rows — 删除行转移。

compact — 组合 **reorg reclaim_space** 和 **reorg forwarded_rows** 的功能，以便一次既可回收空间，又可撤消行转移。

rebuild — 如果指定了表名，则将表中的所有行重写到新的页，以便根据表的集群索引（如果存在）来安排表，并使所有页遵从当前的空间管理设置，没有转移的行且页上的行之间没有间距。如果指定了索引名，**reorg** 将重建索引，并同时使表可以供读取和更新活动访问。

示例

1. `reorg reclaim_space titles`

   回收 **titles** 表中的未使用页空间。

2. `reorg reclaim_space titles titleind`

   回收索引 **titleind** 中的未使用页空间。

3. `reorg compact titles with time = 120`

   对 *titles* 表开始 **reorg compact**。 **reorg** 从表的开头开始，并持续 120 分钟。如果 **reorg** 在此时间限制内完成，它将返回表的开头，并继续进行到指定的时间段完全耗尽为止。

4. `reorg compact titles with resume, time = 30`

   在前一 **reorg compact** 命令停止位置点开始 **reorg compact**，并持续 30 分钟。

注释

- 在 **reorg** 命令中指定的表必须有一种数据行或数据页锁定方案。
- 不能在事务内发出 **reorg** 命令。
- **reorg rebuild** 要求将数据库选项 **select into/bulkcopy/pllsort** 设置为 **true**，并在数据库中运行 **checkpoint**。
- **reorg rebuild** 要求应具有与表及其索引相等的额外磁盘空间。可以通过使用 **sp_spaceused** 查出表当前占有多大的空间。可以使用 **sp_helpsegment** 检查可用空间量。
- 运行 **reorg rebuild** 后，必须在转储事务日志之前转储数据库。
- 详细信息，参见*系统管理指南*。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

必须是系统管理员或对象所有者才能发布 **reorg** 命令。

参见

系统过程	sp_chgattribute

return

功能

从批处理或过程中无条件退出，同时提供可选返回状态。**return** 之后的语句不会被执行。

语法

```
return [integer_expression] [plan "abstract plan"]
```

关键字和选项

integer_expression — 是过程返回的整数值。存储过程可以将整数值返回到调用过程或应用程序。

plan *"abstract plan"* — 指定用于优化查询的抽象计划。它可以是用抽象计划语言指定的完整计划或部分计划。只能为可优化的 SQL 语句（即访问表的查询）指定计划。有关详细信息，参见 *Performance and Tuning Guide* 中的第 22 章 “Creating and Using Abstract Plans”。

示例

1.
```
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
```

如果没有使用参数指定用户名，**return** 命令将导致在将消息发送到用户屏幕后退出过程。如果指定了用户名，将从适当的系统表中检索该用户在当前数据库中创建的规则的名称。

2.
```
print "Begin update batch"
update titles
    set price = price + $3
    where title_id = 'BU2075'
update titles
    set price = price + $3
    where title_id = 'BU1111'
if (select avg(price) from titles
        where title_id like 'BU%') > $15
begin
      print "Batch stopped; average price over $15"
    return
end
update titles
    set price = price + $2
      where title_id = 'BU1032'
```

如果更新导致商业书籍的平均价格超过 $15， **return** 命令将在对 *titles* 进一步执行更新之前终止批处理。

3.
```
create proc checkcontract @param varchar(11)
as
declare @status int
if (select contract from titles where title_id =
@param) = 1
   return 1
else
   return 2
```

该过程将创建两个用户定义的状态代码：如果 *contract* 列中包含 1，则返回值 1；对于任何其它条件 （例如 *contract* 中的值为 0 或 *title_id* 与行不匹配），将返回值 2。

注释

- 返回状态值可以在随后的语句中使用，这些语句位于执行当前过程的批处理或过程中，但必须以下列形式指定：

  execute @*retval* = *procedure_name*

  详细信息，参见 **execute**。

- 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。如果没有提供用户定义的返回值，将使用 Adaptive Server 值。用户定义的返回状态值不能与 Adaptive Server 保留的返回状态值冲突。编号 0 以及 -1 到 -14 当前处于占用状态：

表 6-30：Adaptive Server 错误返回值

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

  保留 -15 到 -99 之间的值供 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。用户定义的返回值总是优先于 Adaptive Server 提供的返回值。
- **return** 命令可以在要从批处理或过程中退出的点上使用。返回是即时且完全的：**return** 之后的语句不会被执行。
- 存储过程不能返回 NULL 返回状态。如果过程尝试返回空值，例如，在 *@status* 为 NULL 的地方使用 **return** *@status*，将生成警告消息，并返回一个 0 到 -14 之间的值。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都具有 **return** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	**begin...end**、 **execute**、 **if...else**、 **while**

revoke

功能

从用户或角色中撤消权限或角色。

语法

要撤消访问数据库对象的权限，可以执行：

```
revoke [grant option for]
   {all [privileges] | permission_list}
   on { table_name [(column_list)]
       | view_name [(column_list)]
       | stored_procedure_name}
   from {public | name_list | role_name}
   [cascade]
```

要撤消创建数据库对象的权限，可以执行 **set proxy** 或执行 **set session authorization**：

```
revoke {all [privileges] | command_list}
   from {public | name_list | role_name}
```

要撤消用户或其它角色的角色，可以执行：

```
revoke role {role_name [, role_name ...]} from
   {grantee [, grantee ...]}
```

关键字和选项

all — 当用于撤消对数据库对象的访问权限时（第一种语法格式），**all** 将撤消适用于指定对象的所有权限。所有对象所有者都可以对对象名使用 **revoke all**，从而撤消对自己对象的权限。

只有系统管理员或数据库所有者可以撤消 **create** 命令的权限（第二种语法格式）。当由系统管理员使用时，**revoke all** 将撤消所有 **create** 权限（**create database**、**create default**、**create procedure**、**create rule**、**create table** 和 **create view**）。当数据库所有者使用 **revoke all** 时，Adaptive Server 将撤消除了 **create database** 之外的所有 **create** 权限，并且输出信息性消息。

all 不适用于 **set proxy** 或 **set session authorization**。

permission_list — 是要撤消的权限的列表。如果要列出多项权限，请用逗号将其隔开。下表对每一类对象可以被授予和撤消的权限进行了说明：

对象	***permission_list* 可以包括：**
表	**select**、 **insert**、 **delete**、 **update**、 **references**
视图	**select**、 **insert**、 **delete**、 **update**
列	**select**、 **update**、 **references** 列名可以在 *permission_list* 或 *column_list* 中指定 （参见示例 2）。
存储过程	**execute**

只有授予权限的用户才能撤消权限。

command_list — 是命令列表。如果要列出多项命令，请用逗号将其隔开。该命令列表可以包含 **create database**、 **create default**、 **create procedure**、**create rule**、**create table**、**create view**、**set proxy** 或 **set session authorization**。只有系统管理员并且只有在 *master* 数据库中才能撤消 **create database** 权限。

set proxy 和 **set session authorization** 是相同的；唯一区别是 **set session authorization** 遵循 SQL 标准，而 **set proxy** 是 Transact-SQL 扩展。撤消执行 **set proxy** 或 **set session authorization** 的权限将撤消成为服务器中其他用户的权限。只有系统安全员并且只有在 *master* 数据库中才能撤消 **set proxy** 或 **set session authorization** 的权限。

table_name — 是要撤消对其权限的表的名称。该表必须位于当前数据库中。只能为每个 **revoke** 语句列出一个对象。

column_list — 是权限所适用的列的列表，中间用逗号分隔。如果指定了列，则只能撤消 **select** 和 **update** 权限。

view_name — 是要撤消对其权限的视图的名称。该视图必须位于当前数据库中。只能为每个 **revoke** 语句列出一个对象。

stored _procedure_name — 是要撤消对其权限的存储过程的名称。该存储过程必须位于当前数据库中。只能为每个 **revoke** 语句列出一个对象。

public — 是所有用户。对于对象访问权限而言，**public** 不包括对象所有者。对于对象创建权限或 **set proxy** 授权而言， **public** 不包括数据库所有者。不能使用 **with grant option** 将权限授予 (**grant**) “public”或者其它组或角色。

name_list — 是用逗号进行分隔的用户名和 / 或组名列表。

role — 是系统或用户定义的角色的名称。使用 **revoke role** 可以撤消授予角色或用户的角色。

role_name — 是系统或用户定义的角色的名称。它使您可以撤消已经被授予特定角色的所有用户的权限。角色名既可以是系统角色，也可以是由系统安全员用 **create role** 创建的用户定义角色。任何一种角色类型都可以通过 **grant role** 命令授予用户。另外，可以使用系统过程 **sp_role** 来授予系统角色。

grantee — 是要撤消其角色的系统角色、用户定义的角色或用户的名称。

grant option for — 会撤消 **with grant option** 权限。因此，在 *name_list* 中指定的用户不能再将指定的权限授予其他用户。如果这些用户已经将权限授予了其他用户，则必须使用 **cascade** 选项撤消那些用户的权限。 *name_list* 中指定的用户保留对象访问权限，但不能再将访问权限授予其他用户。 **grant option for** 仅适用于对象访问权限，而不适用于对象创建权限。

cascade — 将撤消被撤消者已经授予过权限的所有用户指定的对象访问权限。只适用于对象访问权限，而不适用于对象创建权限。（在使用没有 **grant option for** 的 **revoke** 时，被撤销者授予其他用户的权限也将被撤消：级联是自动发生的。）

示例

1. ```
   revoke insert, delete
   on titles
   from mary, sales
   ```

   撤消 Mary 和 “sales” 组对 *titles* 表的 **insert** 和 **delete** 权限。

2. ```
   revoke update
   on titles (price, advance)
   from public
   ```

   或：

   ```
   revoke update (price, advance)
   on titles
   from public
   ```

   撤消 “public” 对 *titles* 表的 *price* 和 *advance* 列的 **update** 权限的两种方法。

3. **revoke create database, create table**
   from mary, john

   撤消 Mary 和 John 使用 **create database** 和 **create table** 命令的权限。由于要撤消 **create database** 权限，该命令必须由系统管理员在 *master* 数据库中执行。只有在 *master* 数据库中才能撤消 Mary 和 John 的 **create table** 权限。

4. **revoke set proxy from harry, billy**

   撤消 Harry 和 Billy 执行 **set proxy** 或 **set session authorization** 来充当服务器中的另一用户的权限。

5. **revoke set session authorization from sso_role**

   撤消具有 **sso_role** 的用户执行 **set proxy** 或 **set session authorization** 的权限。

6. **revoke set proxy from vip_role**

   撤消具有 **vip_role** 的用户充当服务器中另一用户的权限。**vip_role** 必须是系统安全员用 **create role** 命令定义的角色。

7. **revoke all**
   from mary

   撤消 Mary 在当前数据库中的所有对象创建权限。

8. **revoke all**
   on titles
   from mary

   撤消 Mary 对 *titles* 表的所有对象访问权限。

9. **revoke references**
   on titles (price, advance)
   from tom

   或：

   revoke references (price, advance)
   on titles
   from tom

   撤消 Tom 在另一张表上创建参照完整性约束的权限的两种方法，而该表引用的是 *titles* 表中的 *price* 和 *advance* 列。

10. **revoke execute on new_sproc**
    from oper_role

    撤消所有被授予操作员角色的用户执行存储过程 *new_sproc* 的权限。

11. **revoke grant option for**
    insert, update, delete
    on authors
    from john
    cascade

    撤消 John 将对 *authors* 表的 **insert**、**update** 和 **delete** 权限授予其他用户的权限。也将撤消其他用户的已由 John 授予的所有这样的权限。

12. **revoke role doctor_role from specialist_role**

    撤消 specialist_role 的 doctor_role。

13. **revoke role doctor_role, surgeon_role from specialist_role, intern_role, mary, tom**

    撤消 “specialist_role” 和 “intern_role” 以及用户 Mary 和 Tom 的 “doctor_role” 和 “surgeon_role”。

注释

- 有关权限的详细信息，参见 **grant** 命令。
- 只能撤消对当前数据库中的对象的权限。
- 您只能撤消由您授予的权限。
- 不得在用户登录时撤消该用户的角色。
- **grant** 和 **revoke** 命令是区分顺序的。存在冲突时，则最近发出的命令有效。
- 可以用词 **to** 替代 **revoke** 语法中的词 **from**。
- 如果没有在 **revoke** 语句中指定 **grant option for**，将撤消用户的 **with grant option** 权限以及指定的对象访问权限。另外，如果用户已经将指定权限授予了其他用户，所有这些权限也将被撤消。换句话说， **revoke** 是级联的。
- **revoke grant option** 将撤消用户将指定权限授予其他用户的能力，但不撤消该用户的这一权限。如果用户已经将该权限授予其他用户，则必须使用 **cascade** 选项；否则，将收到错误消息，并且 **revoke** 将失败。

  例如，假定您利用如下语句，撤消了用户 Bob 对 *titles* 的 **with grant option** 权限：

  ```
  revoke grant option for select
  on titles
  from bob
  cascade
  ```

  - 如果 Bob 未曾将该权限授予其他用户，该命令将撤消他这样做的能力，但他仍保留了对 *titles* 表的 **select** 权限。
  - 如果 Bob 已经将该权限授予了其他用户，则必须使用 **cascade** 选项。如果不使用该选项，将收到错误消息，并且 **revoke** 将失败。 **cascade** 将撤消已经被 Bob 授予 **select** 权限的用户的这一权限以及这些用户将该权限授予其他用户的能力。

- **grant** 语句将为每一个接受该权限的用户、组或角色在 *sysprotects* 系统表中添加一行。如果您随后 **revoke** （撤消）用户或组的该权限， Adaptive Server 将从 *sysprotects* 中删除该行。如果仅撤消选定组成员的权限，而没有撤消被授予权限的整个组的权限，Adaptive Server 将保留初始行并为该撤消操作添加新的一行。
- 缺省情况下，用户将被授予发出 **create trigger** 命令的权限。撤消用户创建触发器的权限时，会将撤消行增加到该用户的 *sysprotects* 表中。要授权他人发出 **create trigger** 命令，必须发出两个授权命令。第一个命令从 *sysprotects* 中删除撤消行；第二个命令插入授权行。如果撤消创建触发器的权限，则用户不能创建触发器，即便是在自己的表上也不行。撤消用户创建触发器的权限只影响从中发出 revoke 命令的数据库。

撤消 *set proxy* 和 *set session authorization*

- 只有系统安全员并且必须在 *master* 数据库中，才能撤消 **set proxy** 或 **set session authorization** 权限，或撤消角色。
- **set proxy** 和 **set session authorization** 是相同的，但有一个例外：**set session authorization** 遵循 SQL 标准。如果只关心 SQL 标准命令和语法的使用，可以使用 **set session authorization**。
- **revoke all** 不包括 **set proxy** 或 **set session authorization** 权限。

撤消角色、用户和组的权限

- 授予角色的权限将替换授予个人用户或组的权限。因此，如果撤消已经被授予某个角色的用户的权限，而该角色具有相同的权限，用户将保留该权限。例如，假定 John 被授予系统安全员的角色，而 **sso_role** 被授予对 *sales* 表的权限。如果撤消了 John 对 *sales* 的个人权限，他仍然可以访问 *sales*，因为他的角色权限替换了他的个人权限。
- 撤消 “public”或组的特定权限，也将撤消被分别授予该权限的用户的权限。

- 数据库用户组允许您同时 **grant**（授予）多个用户权限或 **revoke**（撤消）他们的权限。用户总是缺省组 “public”的成员，但只能是一个其它组的成员。 Adaptive Server 的安装脚本为 “public”指派了一组权限。

  用系统过程 **sp_addgroup** 创建组，而用 **sp_dropgroup** 删除组。用 **sp_adduser** 向组中添加新用户。用 **sp_changegroup** 更改用户的组成员资格。要显示某一组的成员，可以使用 **sp_helpgroup**。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

数据库对象访问权限

缺省情况下，对象所有者具有数据库对象的 **revoke** 权限。对象所有者可以撤消其他用户对自己的数据库对象的权限。

命令执行权限

只有系统管理员并且只有在 *master* 数据库中，才能撤消 **create database** 权限。只有系统安全员才能撤消 **create trigger** 权限。

代理和会话授权权限

只有系统安全员并且只有在 *master* 数据库中，才能撤消 **set proxy** 或 **set session authorization**。

角色权限

只能在 *master* 数据库中撤消角色。只有系统安全员才能撤消用户或角色的 **sso_role**、 **oper_role** 或用户定义的角色。只有系统管理员才能撤消用户或角色的 **sa_role**。只有具有 **sa_role** 和 **sso_role** 的用户才能撤消包含 **sa_role** 的角色。

参见

命令	**grant**、 **setuser**、 **set**
函数	**proc_role**
系统过程	**sp_activeroles**、 **sp_adduser**、 **sp_changedbowner**、 **sp_changegroup**、 **sp_displaylogin**、 **sp_displayroles**、 **sp_dropgroup**、 **sp_dropuser**、 **sp_helpgroup**、 **sp_helprotect**、 **sp_helpuser**、 **sp_modifylogin**、 **sp_role**

rollback

功能

将用户定义的事务回退到事务中指定的保存点或事务的起始点。

语法

```
rollback {tran[saction] | work}
   [transaction_name | savepoint_name]
```

关键字和选项

transaction | **tran** | **work** — 是可选的。

transaction_name — 是指派给最外层事务的名称。它必须符合标识符的规则。

savepoint_name — 是指派给 **save transaction** 语句中的保存点的名称。该名称必须符合标识符的规则。

示例

```
1. begin transaction
   delete from publishers where pub_id = "9906"
   rollback transaction
```

回退该事务。

注释

- 不带 *transaction_name* 或 *savepoint_name* 的 **rollback transaction** 将用户定义的事务回退到最外层事务的起始处。
- **rollback transaction** *transaction_name* 将用户定义的事务回退到指定事务的起始处。虽然事务可以嵌套，但只能回退最外层的事务。
- **rollback transaction** *savepoint_name* 将用户定义的事务回退到相对应的 **save transaction** *savepoint_name*。

限制

- 如果当前没有活动的事务， **commit** 或 **rollback** 语句就不起作用。
- **rollback** 命令必须包含在事务中。输入 **commit** 后就不能回退事务。

回退整个事务

- 不带保存点名的 **rollback** 将取消整个事务。该事务的所有语句或过程都将被撤消。
- 如果 **rollback** 命令不带 *savepoint_name* 或 *transaction_name*，事务将被回退到这一批中的第一个 **begin transaction**。其中还包括使用链式事务模式，通过隐式 **begin transaction** 启动的事务。

回退到保存点

- 要取消事务的一部分，使用带有 *savepoint_name* 的 **rollback**。保存点是用户通过 **save transaction** 命令保存在事务内的标记集。保存点与 **rollback** 之间的所有语句或过程都将被撤消。

  在事务被回退到保存点后，可以使用 **commit** 继续完成 （执行 **rollback** 之后的任意 SQL 语句），也可以使用不带保存点的 **rollback** 完全取消。事务内的保存点没有数量限制。

触发器和存储过程内的回退

- 在触发器或存储过程中，不带事务名称或保存点名称的 **rollback** 语句将所有语句回退到调用过程或引发触发器的批处理中的第一个显式或隐式 **begin transaction**。
- 如果触发器包含不带保存点名称的 **rollback** 命令，回退就会中止整个批处理。回退后，批处理中的所有语句都不会执行。
- 远程过程调用 (RPC) 是独立于任何包含它的事务来执行的。在标准事务 （即，不使用 Open Client™ DB-Library 两阶段提交的事务）中，由远程服务器通过 RPC 执行的命令不能通过 **rollback** 命令回退，并且执行时不需要依赖 **commit** 。
- 有关使用事务管理语句以及 **rollback** 对存储过程、触发器和批处理的影响的完整信息，参见 *Transact-SQL User's Guide*。

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	**rollback transaction** 和 **rollback tran** 形式的语句以及使用的事务名都是 Transact-SQL 扩展。

权限

rollback 权限缺省设置为 “public”。使用它不需要任何权限。

参见

命令	**begin transaction**、 **commit**、 **create trigger**、 **save transaction**

rollback trigger

功能

回退在触发器中完成的工作 （包括引发触发器的数据修改），并且发出可选的 **raiserror** 语句。

语法

```
rollback trigger
   [with raiserror_statement]
```

关键字和选项

with *raiserror_statement* — 指定 **raiserror** 语句，该语句可以输出用户定义的错误消息并设置系统标志来记录所发生的错误情况。在执行 rollback trigger 时，该语句能够将错误提交到客户端，这样该错误中包含的事务状态就能够反映该回退的情况。有关定义 *raiserror_statement* 的语法和规则的信息，参见 **raiserror** 命令。

示例

```
1. rollback trigger with raiserror 25002
   "title_id does not exist in titles table."
```

回退触发器并发出用户定义的错误消息 25002。

注释

- 执行 **rollback trigger** 时， Adaptive Server 将中止当前正在执行的命令并暂停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，则 Adaptive Server 将把在这些触发器中完成的所有工作回退到（并包含）引起第一个触发器被引发的更新。
- Adaptive Server 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**。然而，如果执行的 **rollback trigger** 语句在触发器外但在事务内，就会产生一个错误，该错误导致 Adaptive Server 回退事务并中止当前的语句批处理。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

rollback trigger 权限缺省设置为 “public”。使用它不需要任何权限。

参见

命令	**create trigger**、 **raiserror**、 **rollback**

save transaction

功能

在事务内设置保存点。

语法

```
save transaction savepoint_name
```

关键字和选项

savepoint_name — 是指派给保存点的名称。它必须符合标识符的规则。

示例

```
1. 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

   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

rollback transaction percentchanged

commit transaction
```

在更新两个作者的 *royaltyper* 条目后，插入保存点 *percentchanged*，然后确定书籍价格提高百分之十会对作者的版权收入造成什么影响。通过 **rollback transaction** 命令，事务将被回退到保存点。

注释

- 有关使用事务语句的完整信息，参见 *Transact-SQL User's Guide*。
- 保存点是事务内用户定义的标记，通过它可以回退事务的各个部分。**rollback** *savepoint_name* 命令回退到指定的保存点；保存点与 **rollback** 之间的所有语句或过程都将被撤消。

  保存点前的语句不会被撤消 — 但也不会被提交。回退到保存点后，事务将继续执行语句。不带保存点的 **rollback** 可以取消整个事务。使用 **commit** 可以继续完成事务。
- 如果嵌套事务，**save transaction** 只在最外层事务中创建保存点。
- 事务内的保存点没有数量限制。
- 如果 **rollback** 命令不带 *savepoint_name* 或 *transaction_name*，则所有语句都将被回退到批处理中的第一个 **begin transaction**，并且整个事务被取消。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

save transaction 权限缺省设置为 “public”。使用它不需要任何权限。

参见

命令	**begin transaction**、 **commit**、 **rollback**

select

功能

从数据库对象中检索行。

语法

select ::=
select [**all** | **distinct**] *select_list*
[*into_clause*]
[*from_clause*]
[*where_clause*]
[*having_clause*]
[*order_by_clause*]
[*compute_clause*]
[*read_only_clause*]
[*isolation_clause*]
[*browse_clause*]
[*plan_clause*]

select_list ::= Defined under “Keywords and Options,” below

into_clause **::=**
into [[*database*.]*owner*.]t*able_name*
[**lock** {**datarows** | **datapages** | **allpages** }]
[**with** *into_option* [*, into_option*] *...*]

into_option ::=
| **max_rows_per_page =** *num_rows*
| **exp_row_size** = *num_bytes*
| **reservepagegap** = *num_pages*
| **identity_gap** = gap

from_clause **::= from** *table_reference* [,*table_reference*]...

table_reference **:=**
table_view_name
| *ANSI_join*

table_view_name **:=**
[[*database*.]*owner*.] {{*table_name* |*view_name*} [as] [*correlation_name*]
[**index** {*index_name* | *table_name* }]
[**parallel** [*degree_of_parallelism*]]
[**prefetch size**][**lru** | **mru**]}]}
[**holdlock** | **noholdlock**] [**readpast**] [**shared**]

ANSI_join **=**
table_reference join_type **join** *table_reference join_conditions*

join_type = inner | *left [outer]* | *right [outer]*

join_conditions **= on** *search_conditions*

where_clause ::= **where** *search_conditions*

group_by_clause ::=
group by [**all**] *aggregate_free_expression*
[, *aggregate_free_expression*]...

having_clause ::= **having** *search_conditions*

order_by_clause ::=
order by sort_clause [, sort_clause]...

sort_clause ::=
{ [[[*database*.]*owner*.]{*table_name*.|*view_name*.}]*column_name*
| *select_list_number*
| *expression* }
[**asc** | **desc**]

compute_clause ::=
compute *row_aggregate*(*column_name*)
[, *row_aggregate*(*column_name*)]...
[**by** *column_name* [, *column_name*]...]

read_only_clause ::=
for {**read only** | **update** [**of** *column_name_list*]}

isolation_clause ::=
at isolation
{ **read uncommitted** | **0** }
| { **read committed** | **1** }
| { **repeatable read** | **2** }
| { **serializable** | **3** }

browse_clause ::= **for browse**

plan_clause ::= **plan** "**abstract plan**"

关键字和选项

all — 包括结果中的所有行。**all** 是缺省选项。

distinct — 只包括结果中的唯一行。**distinct** 必须是选择列表中的第一个词。在浏览模式下，将忽略 **distinct**。

对于关键字 **distinct**，将认为 Null 值都相等：无论遇到多少 NULL，只选择一个。

即使配置为不区分大小写的排序顺序，**distinct** 也会将 “smith” 和 “Smith” 报告为两个不同的行。

select_list — 包括下面的一项或多项：

- “*”，它按照 **create table** 顺序表示所有列。
- 列名的列表，其顺序是您要查看它们的顺序。在选择现有 IDENTITY 列时，可以用 **syb_identity** 关键字 （如有必要，用表名加以限定）替换实际的列名。
- 将新的 IDENTITY 列添加到结果表的说明：

  ```
  column_name = identity(precision)
  ```

- 缺省列标题 （列名）的替换名称，其格式为：

  ```
  column_heading = column_name
  ```

  或：

  ```
  column_name column_heading
  ```

  或：

  ```
  column_name as column_heading
  ```

  所有这些格式的列标题都可以用引号引起来。如果标题不是有效的标识符 （即，标题是保留字、以特殊字符开头或标题中包含空格或标点符号），则必须用引号将其引起来。

- 表达式 （列名、常量、函数、任何由算术运算符或逐位运算符连接的列名、常量和函数组合或子查询）。
- 内部函数或集合
- 上述项的任意组合

  select_list 还可以给变量赋值，其形式如下：

  ```
  @variable = expression
      [, @variable = expression ...]
  ```

  不能将变量赋值与任何其它 *select_list* 选项结合使用。

into — 以在选择列表中指定的列和在 **where** 子句中选择的行为基础，创建新表。参见本节的 “使用 select into”。

lock datarows | datapages | allpages — 指定用 **select into** 命令创建的表将使用的锁定方案。缺省值为配置参数 **lock scheme** 的全服务器范围的设置。

max_rows_per_page — 限定用 **select into** 命令创建的表上的数据页行数。与 **fillfactor** 不同，插入或删除数据时仍维护 **max_rows_per_page** 的值。在 DOL 锁定表上不支持 **max_rows_per_page**。

exp_row_size = *num_bytes* — 指定用 **select into** 命令创建的表的预期行大小。仅对数据行及数据页锁定方案有效，并且仅对具有可变长度行的表有效。有效值为 0、 1 和最小与最大表行长度间的任一值。缺省值为 0，表示使用全服务器范围的缺省设置。

reservepagegap = *num_pages* — 指定填充页与空白页之比，这些空白页将在 **select into** 为存储数据分配扩展时保留。该选项仅对 **select into** 命令有效。对于每个指定的 *num_pages*，保留一个空白页以用于表的未来扩展。有效值为 0-255。缺省值为 0。

readpast — 指定查询应自动跳过具有排它锁的行，而不必等待，也不产生消息。

from — 指定要在 **select** 语句中使用的表和视图。除非选择列表不包含列名 （即只包含常量和算术表达式），否则必须使用该选项：

```
select 5 x, 2 y, "the product is", 5*2 Result
x           y                           Result
----------- ----------- -------------- -----------
          5           2 the product is          10
```

一个查询最多可以引用 16 个表和 12 个工作表 （如集合函数所创建的表）。 16 个表的限制包括：

- 在 **from** 子句中列出的表 （或表的视图）
- 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

view_name, *table_name* — 列出在 **select** 语句中使用的表和视图。当表或视图位于其它数据库时，指定该数据库的名称；当数据库中存在具有该名称的多个表或视图时，指定所有者的名称。 *owner* 的缺省值是当前用户，而 *database* 的缺省值是当前数据库。

如果该列表中有多个表或视图，则应该用逗号分隔其名称。表和视图在关键字 **from** 后的顺序不会影响结果。

您可以在同一语句中查询不同数据库中的表。

为了清晰起见或为了区分表或视图在自连接或子查询中充当的不同角色，可以给表名或视图名提供相关名（别名）。要分配相关名，可依次提供表或视图的名称，一个空格，然后是相关名，如下所示：

```
select pub_name, title_id
    from publishers pu, titles t
    where t.pub_id = pu.pub_id
```

对该表或视图的所有其它引用（例如在 **where** 子句中）必须使用其相关名。相关名不能以数字开头。

index *index_name* — 指定用于访问 *table_name* 的索引。当从视图中进行选择时，不能使用该选项，但可以将它用作 **create view** 语句的 **select** 子句的一部分。

parallel — 在 Adaptive Server 配置为允许并行处理的情况下，指定并行的分区或索引扫描。

degree_of_parallelism — 指定将并行扫描表或索引的工作进程数。如果设置为 1，查询将以串行方式执行。

prefetch *size* — 为配置大型 I/O 的缓存所绑定的表指定 I/O 大小（以千字节为单位）。缺省的大小值为 2、4、8 和 16。当从视图中进行选择时，不能使用该选项，但可以将它用作 **create view** 语句的 **select** 子句的一部分。过程 **sp_helpcache** 显示对象绑定的缓存或缺省缓存的有效大小。

如果启用了组件集成服务，则不能为远程服务器使用 **prefetch**。

lru | mru — 指定用于表的缓冲区替换策略。使用 **lru** 可强制优化程序将表读入 MRU/LRU（最近使用最多/最近使用最少）链上的缓存。使用 **mru** 可以从缓存中放弃缓冲区，并将其替换为该表的下一个缓冲区。当从视图中进行选择时，不能使用该选项，但可以将它用作 **create view** 语句的 **select** 子句的一部分。

holdlock — 通过在事务完成以前占有指定表或视图的共享锁（而不是无论事务是否完成，都在不需要所要求的数据页时释放共享锁），使共享锁更具限制性。

holdlock 选项只应用于为其指定的表或视图，并且只有在它所在语句定义的事务期间才有效。如果设置 **set** 命令的 **transaction isolation level 3** 选项，则将对事务内的每个 **select** 语句隐式应用一个 **holdlock**。在包括 **for browse** 选项的 **select** 语句中，不允许使用关键字 **holdlock**。不能在查询中既指定 **holdlock**，又指定 **noholdlock** 选项。

如果启用了组件集成服务，则不能将 **holdlock** 用于远程服务器。

noholdlock — 使服务器不持有在执行该 **select** 语句时所需的任何锁，而不考虑当前的有效事务隔离级别。不能在查询中既指定 **holdlock**，又指定 **noholdlock** 选项。

shared — 指示 Adaptive Server 对指定表或视图使用共享锁 （而不是更新锁）。这样，其它客户端就可以获取该表或视图的更新锁。当使用 **shared** 关键字时，只能将 **select** 子句当作 **declare cursor** 语句的一部分。例如：

```
declare shared_crsr cursor
for select title, title_id
from titles shared
where title_id like "BU%"
```

在每个表名或视图名后，可以同时使用 **holdlock** 关键字和 **shared**，但 **holdlock** 必须位于 **shared** 之前。

ANSI join — 使用 ANSI 语法的内部或外部连接。 **from** 子句用于指定要连接的表。

inner — 只包括内部和外部表中符合 **on** 子句条件的行。对于不符合 **on** 子句条件的外部表中的行，包含内部连接的查询所得到的结果集中并不包含相应的空值行。

outer — 包括外部表中的所有行，无论这些行是否符合 **on** 子句的条件。如果有一行不符合 **on** 子句的条件，则内部表的值将作为空值存储在连接表中。 ANSI 外部连接中的 **where** 子句将限制查询结果中的行。

left — 左连接将保留在连接语句左侧列出的表引用的所有行。左表引用称为外部表或行保留表。

在以下查询中， *T1* 为外部表， *T2* 为内部表：

```
T1 left join T2
T2 right join T1
```

right — 右连接将保留连接子句右侧的表引用的所有行 （参见上例）。

search_conditions — 用于设置所检索行的条件。搜索条件可以包含列名、表达式、算术运算符、比较运算符、关键字 **not**、**like**、**is null**、**and**、**or**、**between**、**in**、**exists**、**any** 和 **all**，以及子查询、case 表达式或上述项的任何组合。有关详细信息，参见 “where 子句”。

group by — 查找每个组的值。这些值显示为结果中的新列 （而不是新行）。

当 **group by** 用于标准的 SQL 时，选择列表中的每一项必须在组中每一行有一个固定值，或者用于集合函数，为每个组生成单个值。 Transact-SQL 对选择列表中的项目没有这样的限制。此外，

Transact-SQL 还允许按任何表达式 （除了按列别名之外）分组。而使用标准 SQL 时，只能按列分组。

您可以将表 6-31 中列出的集合用于 **group by** （*expression* 几乎始终是列名）：

表 6-31：将集合用于 group by 的结果

集合函数	结果
sum([all | distinct] *expression)*	数字列中值的总和。
avg([all | distinct] *expression)*	数字列中值的平均。
count([all | distinct] *expression)*	列中 （不同）非空值的数目。
count(*)	所选的行数。
max(*expression)*	列中的最大值。
min*(expression)*	列中的最小值。

有关详细信息，参见 “group by 和 having 子句”。

可以按照列的任意一种组合对表进行分组 — 也就是说，组和组之间可以相互嵌套，如示例 2 所示。不能按列标题分组，而必须使用表示项目在选择列表中位置的列名、表达式或数字。

group by all — 在结果中包括所有组，甚至包括其中行不符合搜索条件的组 （有关示例，参见 “group by 和 having 子句”）。

aggregate_free_expression — 是不包含集合的表达式。

having — 为 **group by** 子句设置条件，类似于 **where** 为 **select** 子句设置条件的方式。可包括的条件数不存在限制。

使用 **having** 子句时，可以不带有 **group by** 子句。

如果选择列表中的任何列没有应用集合函数并且不包括在查询的 **group by** 子句中 （在标准 SQL 中为非法）， **having** 和 **where** 的含义就会有所不同。

在这种情况下， **where** 子句限制集合计算中包括的行，但不限制查询返回的行。相反， **having** 子句限制查询返回的行，但不影响集合计算。有关示例，参见 “group by 和 having 子句”。

order by — 按列对结果进行排序。在 Transact-SQL 中，可以将 **order by** 用于不在选择列表中的项目。排序依据可以是表示项目在 **select list** (*select_list_number*) 中位置的列名、列标题 （或别名）、表达式或数字。如果按选择列表号排序， **order by** 子句引用的列就必须包含在选择列表中，并且选择列表不能是 * （星号）。

asc — 按升序对结果进行排序 （缺省）。

desc — 按降序对结果进行排序。

compute — 与行集合 （**sum**、 **avg**、 **min**、 **max** 和 **count**）一起用来生成控制中断摘要值。摘要值显示为查询结果中的附加行，这使您用一个语句即可查看明细行和摘要行。

不能将 **select into** 子句与 **compute** 一起使用。

如果使用 **compute by**，就必须同时使用 **order by** 子句。 **compute by** 后面列出的列必须与 **order by** 后面列出的列相同或是其子集，它们从左向右的顺序必须一致，以同一表达式开始且不跳过任何表达式。

例如，如果 **order by** 子句是：

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

compute by 子句可以是下列任意 （或全部）形式：

```
compute by a, b, c
compute by a, b
compute by a
```

不带 **by** 的关键字 **compute** 可用于生成总数和、总计数等。如果使用不带 **by** 的 **compute**， **order by** 就是可选的。有关详细信息和示例，参见 “compute Clause”。

如果启用了组件集成服务，则不能将 **compute** 用于远程服务器。

for {**read only** | **update**} — 指定游标结果集为只读或可更新。只能在存储过程内，并且只能在该过程为游标定义了查询时使用该选项。在这种情况下， **select** 是该过程中允许的唯一语句。它定义 **for read only** 或 **for update** 选项 （而不是 **declare cursor** 语句）。这种声明游标的方法提供了读取行时的页级锁优势。

如果没有使用存储过程中的 **select** 语句来定义游标， Adaptive Server 将忽略 **for read only | update** 选项。有关如何使用存储过程声明游标的详细信息，参见 Embedded SQL™ 文档。有关只读或可更新游标的信息，参见 *Transact-SQL User's Guide*。

of *column_name_list* — 是用 **for update** 选项定义为可更新的游标结果集中列的列表。

at isolation — 指定查询的隔离级别 （0、 1、 2 或 3）。如果省略该子句，查询将使用它执行时所在会话的隔离级别 （缺省情况下为隔离级别 1）。 **at isolation** 子句仅对于单个查询或仅在 **declare cursor** 语句内才有效。如果将 **at isolation** 用于以下查询， Adaptive Server 将返回语法错误：

- 使用 **into** 子句的查询
- 在子查询内
- **create view** 语句中的查询
- **insert** 语句中的查询
- 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** 后指定 **at isolation** 子句。如果指定 **at isolation read uncommitted** 的查询中指定 **holdlock**、**noholdlock** 或 **shared**， Adaptive Server 将发出警告并忽略 **at isolation** 子句。对于其它隔离级别， **holdlock** 优先于 **at isolation** 子句。有关隔离级别的详细信息，参见 *Transact-SQL User's Guide*。

如果启用了组件集成服务，则不能将 **at isolation** 用于远程服务器。

read uncommitted | 0 — 为查询指定隔离级别 0。

read uncommitted | 1 — 为查询指定隔离级别 1。

repeatable read | 2 — 为查询指定事务隔离级别 2。

serializable | 3 — 为查询指定隔离级别 3。

for browse — 必须附加到发送给 DB-Library 浏览应用程序中 Adaptive Server 的 SQL 语句的末尾。有关详细信息，参见 *Open Client DB-Library Reference Manual*。

plan *"abstract plan"* — 指定用于优化查询的抽象计划。它可以是用抽象计划语言指定的完整计划或部分计划。有关详细信息，参见 *Performance and Tuning Guide* 中的第 22 章 "Creating and Using Abstract Plans"。

示例

```
1. 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
```

从 *publishers* 表中选择所有行和列。

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

从 *publishers* 表的特定列中选择所有行。

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

```
                    Publisher                          pub_id
---------------------- ------------------------------ ------
The publisher's name is New Age Books                  0736
The publisher's name is Binnet & Hardley               0877
The publisher's name is Algodata Infosystems           1389
```

从 *publishers* 表的特定列中选择所有行，同时替换一个列名并在输出中添加一个字符串。

```
4. select type as Type, price as Price
   from titles
```

从 *titles* 表的指定列中选择所有行，同时替换列名。

```
5. select title_id, title, price
   into bus_titles
   lock datarows with reservepagegap = 10
   from titles
   where type = "business"
```

为 **select into** 指定锁方案和保留页间距。

```
6. select title, price
   from titles readpast
       where type = "news"
       and price between $20 and $30
```

仅选择没有被排它锁定的行。如果任何其它用户在限定行上有排它锁，则不会返回该行。

```
7. select pub_id, total = sum (total_sales)
       into #advance_rpt
   from titles
   where advance < $10000
       and total_sales is not null
   group by pub_id
   having count(*) > 1
```

选择特定列和行，同时将结果放入临时表 *#advance_rpt* 中。

```
8. select "Author_name" = au_fname + " " + au_lname
       into #tempnames
       from authors
```

并置两个列名，并将结果放入临时表 *#tempnames* 中。

```
9. select type, price, advance from titles
   order by type desc
   compute avg(price), sum(advance) by type
   compute sum(price), sum(advance)
```

选择特定的列和行，返回从最高类型到最低类型排序的结果，并计算摘要信息。

```
10.select type, price, advance from titles
   compute sum(price), sum(advance)
```

选择特定的列和行，并计算 *price* 列和 *advance* 列的总计值。

```
11.select * into coffeetabletitles from titles
   where price > $20
```

创建 *coffeetabletitles* 表，即只包括 $20 以上的书籍的 *titles* 表的副本。

```
12.select * into newtitles from titles
   where 1 = 0
```

创建 *newtitles* 表，即 *titles* 表的空副本。

```
13.select title_id, title
       from titles (index title_id_ind prefetch 16)
       where title_id like "BU%"
```

提供优化程序提示。

```
14.select sales_east.syb_identity,
   sales_west.syb_identity
   from sales_east, sales_west
```

使用 *syb_identity* 关键字选择 *sales_east* 表和 *sales_west* 表中的 IDENTITY 列。

```
15.select *, row_id = identity(10)
   into newtitles from titles
```

创建 *newtitles* 表，即带有 IDENTITY 列的 *titles* 表的副本。

```
16.select pub_id, pub_name
   from publishers
   at isolation read uncommitted
```

指定查询的事务隔离级别。

```
17.begin tran
   select type, avg(price)
       from titles
       group by type
   at isolation repeatable read
```

使用 **repeatable read** 隔离级别从 *titles* 中进行选择。在事务完成之前，其他用户不能更改受影响行中的值，也不能删除该行。

```
18.select ord_num from salesdetail
       (index salesdetail parallel 3)
```

为查询的并行度提供优化程序提示。

```
19.select au_id, titles.title_id, title, price
   from titleauthor inner join titles
   on titleauthor.title_id = titles.title_id
   and price > 15
```

通过 *titleauthor* 表和 *titles* 表的 *title_id* 列连接这两个表。在结果集中，只有那些包含大于 15 的 *price* 的行。

```
20.select au_fname, au_lname, pub_name
   from authors left join publishers
   on authors.city = publishers.city
```

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。

注释

- 与所有其它语句相同， **select** 语句中的关键字必须按照语法语句中所示的顺序来使用。
- 为了与其它 SQL 实施兼容，可以在 **select** 后使用关键字 **all**。 **all** 是缺省关键字。当在这种情况下使用时， **all** 与 **distinct** 相反。检索到的所有行 （包括重复的行）都会包括在结果中。
- 除在 **create table**、 **create view** 和 **select into** 语句中之外，带引号的列标题可以包含任何字符 （包括空格和 Adaptive Server 关键字）。如果列标题不带引号，则必须符合标识符规则。
- **create table**、 **create view** 和 **select into** 语句中的列标题以及表别名必须符合标识符规则。

- 要使用 **select** 将数据从某些域为空值的表中插入不允许空值的表中，必须为初始表的所有 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 域：

  ```
  insert advances
  select pub_id, isnull(advance, 0) from titles
  ```

  如果不使用 **isnull** 函数，该命令就会将带有非空值的所有行插入 *advances* 表，这样将对在 *titles* 表的 *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指定 NOT NULL 的列。

  尽管两个表可以在结构上相同，但它们在某些域是否允许空值方面仍会有所区别。使用 **sp_help** 可以查看表中各列的空值类型。

- 用 **select** 语句返回的 *text* 或 *image* 数据的缺省长度是 32K。使用 **set textsize** 可以更改该值。当前会话的大小存储在全局变量 *@@textsize* 中。某些客户端软件可能会在登录到 Adaptive Server 中时发出 **set textsize** 命令。

- 远程 Adaptive Server 中的数据可以通过使用远程过程调用来检索。有关详细信息，参见 **create procedure** 和 **execute**。

- 在游标定义中使用的 **select** 语句（通过 **declare cursor**）必须包含 **from** 子句，但不能包含 **compute**、 **for browse** 或 **into** 子句。如果 **select** 语句包含以下任何结构，游标将被当作只读或不可更新：

  - **distinct** 选项
  - **group by** 子句
  - 集合函数
  - **union** 运算符

  如果在存储过程内声明游标时使用了包含 **order by** 子句的 **select** 语句，那么该游标也会被当作只读。如果定义游标的 **select** 语句包含两个或更多个表的连接，那么即使该游标被当作可更新，也不能删除使用该游标的行。有关详细信息，参见 **declare cursor**。

- 如果给变量赋值的 **select** 语句返回多个行，最后返回的值就将赋给该变量。例如：

  ```
  declare @x varchar(40)
  select @x = pub_name from publishers
  print @x

  (3 rows affected)
  Algodata Infosystems
  ```

使用 ANSI 连接语法

- 在用 ANSI 内部和外部连接语法编写查询之前，务必要阅读 *Transact-SQL User's Guide* 第 4 章 "Joins: Retrieving Data From Several Tables"中的 "Outer Joins"。

使用 *select into*

- **select into** 操作包括两步。第一步创建新表，第二步将指定行插入新表。

  由于 **select into** 操作所插入的行不记日志，所以不能在用户定义的事务中发出 **select into** 命令，即使 **ddl in tran** 数据库选项被设置为 **true**。但 **select into** 操作过程中进行的页分配会被记录下来，所以大型的 **select into** 操作可能填充事务日志。

  如果 **select into** 语句在创建新表后失败， Adaptive Server 将**不会**自动删除该表或释放该表的第一个数据页。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。使用 **drop table** 语句删除新表，然后重新发出 **select into** 语句。

- 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。您可以对临时表执行 **select into** 命令 （参见示例 7、 8 和 11）。
- 但不能对已经存在的表执行 **select into** （而应使用 **insert...select** 命令）。有关详细信息，参见 **insert**。
- 任何与基表关联的规则、约束或缺省值都不能转移到新表中。但可以使用 **sp_bindrule** 和 **sp_bindefault** 将规则和缺省值绑定到新表。
- **select into** 不转移基表的 **max_rows_per_page** 值，而创建 **max_rows_per_page** 值为 0 的新表。使用 **sp_chgattribute** 可设置 **max_rows_per_page** 的值。
- 要对永久表执行 **select into**，必须将 **select into/bulkcopy/pllsort** 选项设置为 **true** （通过执行 **sp_dboption**）。由于临时数据库绝不会被恢复，因此在对临时表执行 **select into** 时，不必将 **select into/bulkcopy/pllsort** 选项设置为 **true**。

  在数据库中使用 **select into** 之后，要使用 **dump transaction** 命令，必须先执行完全的数据库转储。 **select into** 操作只对页分配记日志，而不对数据行的更改记日志。因此，无法从事务日志中恢复更改。在这种情况下，如果发出 **dump transaction** 语句，则会产生错误消息，指示您改用 **dump database**。

  缺省情况下，在新创建的数据库中， **select into/bulkcopy/pllsort** 选项被设置为 **false**。要更改缺省条件，可在 *model* 数据库中将该选项设置为 **true**。

- 当执行 **dump database** 时， **select into** 的运行速度较慢。
- 通过在 **where** 子句中提供假条件，可以使用 **select into** 创建无数据的重复表 （参见示例 12）。
- 对于选择列表中所有包含集合函数或任何表达式的列，都必须提供列标题。要在选择列表中使用任何常量、算术或字符表达式、内部函数或并置，受影响的项目就必须具有列标题。列标题必须是有效的标识符，或者用引号引起来 （参见示例 7 和 8）。
- 由于函数允许空值，选择列表中所有包含 **convert** 或 **isnull** 之外任何函数的列也允许空值。
- 不能在用户定义的事务或在同一语句中将 **select into** 用作 **compute** 子句。
- 要将 IDENTITY 列选择到结果表中，应在 **select** 语句的 *column_list* 中包括该列名 （或 **syb_identit** 关键字）。新列遵守以下规则：
  - 如果多次选择 IDENTITY 列，它将在新表中定义为 NOT NULL。该 IDENTITY 列不继承 IDENTITY 属性。
  - 如果将 IDENTITY 列当作表达式的一部分来选择，结果列将不继承 IDENTITY 属性。当该表达式中的任何列允许空值时，它将创建为 NULL ；否则，将创建为 NOT NULL。
  - 如果 **select** 语句包含 **group by** 子句或集合函数，结果列将不继承 IDENTITY 属性。包含 IDENTITY 列集合的列将创建为 NULL ； 其它列则创建为 NOT NULL。
  - 用联合或连接选择到表中的 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，将定义为 NOT NULL。
- 不能使用 **select into** 来创建具有多个 IDENTITY 列的新表。如果 **select** 语句同时包含现有的 IDENTITY 列和形式为 *column_name* = **identity***(precision)* 的新 IDENTITY 列，该语句就会失败。
- 如果启用了组件集成服务，并且 **into** 表位于 Adaptive Server，Adaptive Server 就会使用批量复制例程来将数据复制到新表中。在对远程表执行 **select into** 之前，应将 **select into/bulkcopy** 数据库选项设置为 **true**。
- 有关 Embedded SQL 命令 **select into** *host_var_list* 的详细信息，参见 *Open Client Embedded SQL Reference Manual*。

用 *select...into* 转换目标列的 Null 属性

- 使用 **convert** 命令可以更改（您正将数据选择到的）目标列的可为空性。例如，以下语句将 *titles* 表中的数据选择到名为 *temp_titles* 的目标表中，同时将 *total_sales* 列从 **null** 转换为 **not null**：

```
select title, convert (char(100) not null,
total_sales) into #tempsales
from titles
```

用 *select...into* 指定锁方案

- 与 **select...into** 一起使用的 **lock** 选项使您能够为此命令所创建的表指定锁方案。如果没有指定锁方案，将应用由配置参数 **lock scheme** 设置的缺省锁方案。
- 当使用 **lock** 选项时，也可以指定空间管理属性 **max_rows_per_page**、**exp_row_size** 和 **reservepagegap**。

  使用 **sp_chgattribute** 系统过程，可以更改用 **select into** 创建的表的空间管理属性。

使用 *index*、*prefetch* 和 *lru | mru*

- **index**、**prefetch** 和 **lru | mru** 选项为执行查询指定索引、缓存和 I/O 策略。这些选项将替换 Adaptive Server 优化程序所作的选择。应小心使用这些选项，并经常通过 **set statistics io on** 检查它们对性能的影响。有关使用这些选项的详细信息，参见 *Performance and Tuning Guide*。

使用 *parallel*

- **parallel** 选项可减少 Adaptive Server 优化程序能够用来进行并行处理的工作线程数。 *degree_of_parallelism* 不能大于所配置的 **max parallel degree**。如果指定的值大于所配置的 **max parallel degree**，优化程序将忽略 **parallel** 选项。
- 当多个工作进程合并其结果时， Adaptive Server 返回行的顺序在不同的执行过程之间可能会有所不同。要按一致的顺序从已分区的表中获取行，可使用 **order by** 语句，或通过在查询的 **from** 子句中使用 **parallel 1** 来替换并行查询执行。
- 如果以下任何条件为真，将忽略指定 **parallel** 的 **from** 子句：
  - 为更新或插入使用了 select 语句。
  - 在游标定义中使用了 **from** 子句。
  - 在子查询的任何内部查询块内的 **from** 子句中使用了 **parallel**。
  - select 语句创建了视图。

- 该表是外部连接的内部表。
- 该查询在表上指定 **min** 或 **max** 并指定索引。
- 指定了未分区的集群索引，或者该索引是唯一的 **parallel** 选项。
- 该查询在表上指定 **exists**。
- 配置参数 **max scan parallel degree** 的值为 1，并且查询指定了索引。
- 包括了非集群索引。有关索引范围的详细信息，参见 *Performance and Tuning Guide* 中的第 4 章 “How Indexes Work”。
- 表是系统表或虚拟表。
- 用 OR 策略处理查询。有关 OR 策略的解释，参见 *Performance and Tuning Guide*。
- 查询将向用户返回大量的行。

使用 *readpast*

- **readpast** 选项允许 **select** 命令访问指定的表，而不会被其它任务持有的不兼容锁阻塞。**readpast** 查询只能在 DOL 锁定表上执行。
- 如果为 allpage 锁定表指定了 **readpast** 选项，则将忽略 **readpast** 选项。此命令在为命令或会话指定的隔离级别上运行。如果隔离级别为 0，将执行脏读，此命令返回锁定行中的值并且不会阻塞。如果隔离级别为 1 或 3，那么在必须读取具有不兼容锁的页时，命令将阻塞。
- 会话级隔离级别与表上 **select** 命令中 **readpast** 的相互作用如表 6-32 所示。

表 6-32：会话级隔离级别和 readpast 的效果

会话隔离级别	效果
0， read uncommitted（脏读）	忽略 **readpast**，并向用户返回包含未提交事务的行。输出一条警告消息。
1， read committed	跳过带有不兼容锁的行或页；在被读取的行或页上不持有锁
2， repeatable read	跳过带有不兼容锁的行或页；在语句或事务结束前读取的所有行或页上持有共享锁；在事务完成前由语句读取的所有页上持有锁。
3， serializable	忽略 **readpast**，命令在级别 3 执行。命令将阻塞于任何带有不兼容锁的行或页。

- 如果指定 **readpast** 选项的 **select** 命令还包含以下任何内容，它们将会失败并给出错误消息：
  - 指定 **0** 或 **read uncommitted** 的 **at isolation** 子句
  - 指定 **3** 或 **serializable** 的 **at isolation** 子句
  - 同一表上的 **holdlock** 关键字
- 如果在指定 **readpast** 的 **select** 查询中指定 **at isolation 2** 或 **at isolation repeatable read**，则将在 **readpast** 表上持有共享锁，直到语句或事务完成。
- 如果带有 **readpast** 选项的 **select** 命令遇到带有不兼容锁的文本列，**readpast** 锁将检索行，但返回值为 **null** 的文本列。在这种情况下，包含空值的文本列和因列被锁定而返回的空值之间没有任何区别。

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	以下内容为 Transact-SQL 扩展： • 可创建新表的 **select into** • **lock** 子句 • **compute** 子句 • 全局和局部变量 • **index** 子句、 **prefetch**、 **parallel** 和 **lru	mru** • **holdlock**、 **noholdlock** 和 **shared** 关键字 • *“column_heading = column_name”* • 限定表名和列名 • **for browse** 子句中的 **select** • 在选择列表内使用不在 **group by** 列表中且不具有集合函数的列 • **at isolation repeatable read	2** 选项

权限

缺省情况下，表或视图的所有者拥有 **select** 权限，他们可以将该权限移交给其他用户。

参见

命令	**compute Clause**、 **create index**、 **create trigger**、 **delete**、**group by 和 having 子句**、**insert**、**order by Clause**、**set**、**union 运算符**、**update**、**where 子句**
函数	**avg**、 **count**、 **isnull**、 **max**、 **min**、 **sum**
系统过程	**sp_cachestrategy**、 **sp_chgattribute**、 **sp_dboption**

set

功能

为用户的工作会话过程设置 Adaptive Server 查询处理选项；设置触发器或存储过程内的某些选项；激活或禁用当前会话中的角色；指定查询在中止或返回错误消息之前等待获取锁的时间长度；将事务隔离级别设置为隔离级别 2，可重复读取；指定应使用模拟统计信息将查询优化；设置分布式事务处理选项。

语法

```
set ansinull {on | off}

set ansi_permissions {on | off}

set arithabort [arith_overflow | numeric_truncation]
   {on | off}

set arithignore [arith_overflow] {on | off}

set {chained, close on endtran, nocount, noexec,
   parseonly, procid, self_recursion, showplan,
   sort_resources} {on | off}

set char_convert {off | on [with {error | no_error}] |
   charset [with {error | no_error}]}

set cis_rpc_handling {on | off}

set [clientname client_name | clienthostname host_name
   | clientapplname application_name]

set cursor rows number for cursor_name

set {datefirst number, dateformat format,
   language language}

set fipsflagger {on | off}

set flushmessage {on | off}

set forceplan {on | off}

set identity_insert [database.[owner.]]table_name
   {on | off}

set jtc {on | off}

set lock { wait [ numsecs ] | nowait }

set offsets {select, from, order, compute, table,
   procedure, statement, param, execute} {on | off}

set parallel_degree number

set plan {dump | load } [group_name] {on | off}
```

```
set plan exists check {on | off}
set plan replace {on | off}
set prefetch [on|off]
set process_limit_action {abort | quiet | warning}
set proxy login_name
set quoted_identifier {on | off}
set role {"sa_role" | "sso_role" | "oper_role" |
   role_name [with passwd "password"]} {on | off}
set {rowcount number, textsize number}
set scan_parallel_degree number
set session authorization login_name
set sort_merge {on | off}
set statistics {io, subquerycache, time} {on | off}
set statistics simulate { on | off }
set strict_dtm_enforcement {on | off}
set string_rtruncation {on | off}
set table count number
set textsize {number}
set transaction isolation level {
   [ read uncommitted | 0 ] |
   [ read committed | 1 ] |
   [ repeatable read | 2 ]|
   [ serializable | 3 ] }
set transactional_rpc {on | off}
```

关键字和选项

ansinull — 确定在 SQL 的等于 (=) 或不等于 (!=) 比较或集合函数（也称为 **set 函数**）中， NULL 值操作数的求值是否符合 SQL92 标准。如果使用 **set ansinull on**，每当集合函数从计算中消除空值操作数时， Adaptive Server 都会生成一条警告。此操作不影响 Adaptive Server 在其它类型的 SQL 语句（如 **create table**）中求 NULL 值的方式。

SQL 标准要求，如果一个等于比较的两个操作数中的任一个为 NULL，结果都应是 UNKNOWN。 Transact-SQL 处理 NULL 值的方式则不同。如果一个操作数是列、参数或变量，而另一个操作数是 NULL 常量或值为 NULL 的参数或变量，那么结果为 TRUE 或 FALSE。

ansi_permissions — 确定是否检查了 SQL92 对 **delete** 和 **update** 语句的权限要求。缺省值为 **off**。表 6-33 总结了权限要求：

表 6-33：update 和 delete 所要求的权限

命令	要求权限 (*set ansi_permissions off*)	要求权限 (*set ansi_permissions on*)
update	• 要设置值的列的 **update** 权限	• 要设置值的列的 **update** 权限 • **where** 子句中显示的所有列的 **select** 权限 • **set** 子句右侧所有列的 **select** 权限
delete	• 表的 **delete** 权限	• 表的 **delete** 权限 • **where** 子句中显示的所有列的 **select** 权限

arithabort — 确定 Adaptive Server 在出现算术错误时的行为。两个 **arithabort** 选项 （**arithabort arith_overflow** 和 **arithabort numeric_truncation**）处理不同类型的算术错误。您可以单独设置各个选项，也可以用单个 **set arithabort on** 或 **set arithabort off** 语句同时设置这两个选项。

- **arithabort arith_overflow** 指定在进行显式或隐式数据类型转换时，Adaptive Server 在出现除零错误或精度损失后的行为。这种类型的错误是很严重的。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，并且 Adaptive Server 也不会执行批处理中产生错误的语句之后的任何语句。

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

- **arithabort numeric_truncation** 指定当进行隐式数据类型转换时，Adaptive Server 在精确数值类型导致标度损失后的行为。（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。）缺省设置为 **arithabort numeric_truncation on**，它将中止导致错误的语句，但 Adaptive Server 会继续处理事务或批处理中的其它语句。如果设置 **arithabort numeric_truncation off**，Adaptive Server 就会截断查询结果并继续进行处理。

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。缺省情况下， **arithignore** 选项被设置为 **off**。这使 Adaptive Server 在任何导致数字溢出的查询之后显示警告消息。要让 Adaptive Server 忽略溢出错误，可使用 **set arithignore on**。可以省略可选关键字 **arith_overflow**，而不会导致任何影响。

chained — 在会话开始和事务结束时，在第一个数据检索或数据修改语句前开始一个事务。在链式模式中， Adaptive Server 在以下语句之前隐式执行一个 **begin transaction** 命令：**delete**、 **fetch**、 **insert**、 **open**、 **select** 和 **update**。不能在事务内执行 **set chained**。

char_convert — 启用或禁用 Adaptive Server 和客户端之间的字符集转换。如果客户端使用 Open Client DB-Library 版本 4.6 或更高版本，且客户端和服务器在使用不同的字符集，则将在登录过程中打开转换，并根据客户端正在使用的字符集将转换设置为缺省值。也可以使用 **set char_convert** *charset* 开始服务器字符集和不同的客户端字符集之间的转换。

charset 可以是 *syscharsets* 中 *type* 值小于 2000 的字符集的 ID 或名称。

set char_convert off 关闭转换，使数据在接收和发送时不发生改变。**set char_convert on** 打开已关闭的转换。如果字符集转换未在登录过程中打开或没有用 **set char_convert** 命令打开， **set char_convert on** 将生成错误消息。

如果未包括 **with no_error** 选项，那么当 Adaptive Server 中的字符不能转换为客户端字符集时， Adaptive Server 就不会通知应用程序。错误报告最初在客户端连接 Adaptive Server 时打开：如果不需要错误报告，则必须用 **set char_convert {on | charset} with no_error** 为每个会话关闭错误报告功能。要在会话中重新打开错误报告，可使用 **set char_convert {on | charset} with error**。

不管错误报告是否打开，不能被转换的字节都由 ASCII 问号 (?) 代替。

有关字符集转换中错误处理的详细讨论，参见*系统管理指南*。

cis_rpc_handling — 确定组件集成服务是否在缺省情况下处理外发的远程过程调用 (RPC) 请求。

clientapplname — 为应用程序分配单独的名称。在许多客户端使用同一应用程序名连接到 Adaptive Server 的系统中，它有助于区分不同的客户端。为应用程序分配新的名称之后，该应用程序将显示在 *sysprocesses* 表中的新名称下。

clienthostname — 为主机分配单独的名称。在许多客户端使用同一主机名连接到 Adaptive Server 的系统中，它有助于区分不同的客户端。为主机分配新的名称之后，该主机将显示在 *sysprocesses* 表中的新名称下。

clientname — 为客户端分配单独的名称。在许多客户端使用同一客户端名连接到 Adaptive Server 的系统中，它有助于区分不同的客户端。为用户分配新的名称之后，这些用户将显示在 *sysprocesses* 表中的新名称下。

close on endtran — 使 Adaptive Server 在事务结束时关闭所有在该事务内打开的游标。事务通过使用 **commit** 或 **rollback** 语句来结束。不过，只有在设置此选项的范围（存储过程、触发器等）内声明的游标才会受到影响。有关游标范围的详细信息，参见 *Transact-SQL User's Guide*。

有关已评估的配置的详细信息，参见*系统管理指南*。

cursor rows — 使 Adaptive Server 为客户端应用中的每个游标 **fetch** 请求返回*行数*。*行数*可以是无小数点的数字文字或类型为 *integer* 的局部变量。如果*行数*小于或等于零，该值将设置为 1。无论游标是否打开，都可以为它设置 **cursor rows** 选项。但该选项不影响包含 **into** 子句的 **fetch** 请求。 *cursor_name* 指定要为其设置返回行数的游标。

datefirst — 将每周第一天设置为 1 至 7 的数字。 us_english 语言的缺省值为 1（星期日）。

dateformat — 为输入 *datetime* 或 *smalldatetime* 数据设置日期分量 *month/day/year* 的顺序。有效参数为 *mdy*、 *dmy*、 *ymd*、 *ydm*、 *myd* 和 *dym*。 us_english 语言的缺省值为 *mdy*。

fipsflagger — 确定 Adaptive Server 是否在使用 Transact-SQL 的初级 SQL92 扩展时显示警告消息。缺省情况下， Adaptive Server 不会在您使用不标准的 SQL 时提供提示。此选项不会禁用 SQL 扩展。发出非 ANSI SQL 命令时，处理完成。

flushmessage — 确定 Adaptive Server 何时向用户返回消息。缺省情况下，在生成消息的查询完成之前，或在缓冲区容量添满之前，消息一直存储在缓冲区中。使用 **set flushmessage on** 可以在生成消息时立即将其返回给用户。

forceplan — 使查询优化程序在查询的 **from** 子句中将表的顺序用作查询计划的连接顺序。当优化程序未能选择有效的计划时，通常会使用 **forceplan**。强制错误计划会对 I/O 和性能造成严重的不良影响。有关详细信息，参见 *Performance and Tuning Guide*。

identity_insert — 确定是否允许在表的 IDENTITY 列中进行显式插入。（在任何情况下都不允许更新 IDENTITY 列。）此选项只能用于基表。它不能用于视图，也不能在触发器内设置。

设置 **identity_insert** *table_name* **on** 后，表所有者、数据库所有者或系统管理员可以在 IDENTITY 列中显式地插入值。在 IDENTITY 列中插入值后，就可以指定该列的源值或恢复被误删的行。只要未在 IDENTITY 列上创建唯一索引， Adaptive Server 就不会验证插入值的唯一性，因此可插入任意正整数。

表所有者、数据库所有者或系统管理员可以对表使用 **set identity_insert** *table_name* **on** 命令，以启用在 IDENTITY 列中手工插入值的功能。但是，当 **identity_insert** 为 **on** 时，只有以下用户才能在 IDENTITY 列中实际插入值：

- 表所有者
- 数据库所有者 （如果被表所有者授予该列的显式 **insert** 权限）
- 数据库所有者 （通过使用 **setuser** 命令来充当表所有者）

当设置 **identity_insert** *table_name* **off** 时，将禁止在 IDENTITY 列中显式插入，从而恢复缺省行为。您可以随时在会话中将 **set identity_insert** *table_name* **on** 用于单个数据库表。

ljtc — 切换连接过渡关闭。有关详细信息，参见 *Performance and Tuning Guide*。

language — 是显示系统消息的语言的正式名称。该语言必须已安装在 Adaptive Server 中。缺省值为 us_english。

nocount — 控制语句所影响行数的显示。**set nocount on** 禁用行数显示；**set nocount off** 重新启用行计数。

noexec — 编译但不执行每个查询。 **noexec** 通常与 **showplan** 一起使用。当设置 **noexec on** 后，除非您设置 **noexec off**，否则不执行任何后继的命令 （包括其它 **set** 命令）。

lock wait — 指定命令在中止并返回错误之前等待获取锁的时间长度。

numsecs — 指定命令为获取锁等待的秒数。有效值为 0 到 2147483647 （最大整数值）。

lock nowait — 指定：如果命令不能立即获取锁，命令将返回错误并失败。 **set lock nowait** 等同于 **set lock wait 0**。

offsets — 返回指定关键字在 Transact-SQL 语句中的位置 （相对于查询开头的位置）。关键字列表是用逗号分隔的列表，它可以包含以下任何 Transact-SQL 结构：**select**、**from**、**order**、**compute**、**table**、**procedure**、 **statement**、**param** 和 **execute**。如果不存在任何错误，Adaptive Server 将返回偏移。此选项只用于 Open Client DB-Library。

parallel_degree — 指定在并行执行查询时使用的工作进程数的上限。此数字必须小于或等于每个查询的工作进程数 （用 **max parallel degree** 配置参数设置）。 *@@parallel_degree* 全局变量存储当前设置。

parseonly — 检查每个查询的语法并返回任何错误消息，而不编译或执行查询。不要在存储过程或触发器内使用 **parseonly**。

plan — 引入抽象计划命令。有关详细信息，参见 *Performance and Tuning Guide* 中的第 22 章 “Creating and Using Abstract Plans”。

dump — 对当前连接启用或禁用抽象计划捕获功能。如果未指定 *group_name*，计划将存储在缺省组 *ap_stdout* 中。

load — 对当前连接启用或禁用抽象计划装载功能。如果未指定 *group_name*，计划将从缺省组 *ap_stdout* 中装载。

group_name — 是用来装载或存储计划的抽象计划组的名称。

exists check — 当与 **set plan load** 一起使用时，它在每个用户缓存中存储抽象计划组中多达 20 个查询的散列键。

replace — 启用或禁用在计划捕获模式中替换现有查询计划的功能。缺省情况下，计划替换功能被关闭。

prefetch — 启用或禁用对数据缓存的大型 I/O。

process_limit_action — 指定当可用工作进程数不足时，Adaptive Server 是否执行并行查询。在这种情况下，如果 **process_limit_action** 设置为 **quiet**， Adaptive Server 会自动调整计划，以使用不超过可用进程数的并行度。当可用的工作进程数不足时，如果 **process_limit_action** 设置为 **warning**， Adaptive Server 将在调整计划时发出警告消息；如果 **process_limit_action** 设置为 **abort**， Adaptive Server 将中止查询，并发出一条说明性消息来指出可用的工作进程数。

procid — 在发送存储过程生成的行之前，将该存储过程的 ID 号返回给 Open Client DB-Library/C （而不是用户）。

proxy — 使您可以获取权限、登录名以及 *login_name* 的 *suid* （服务器用户 ID）。对于 *login_name*，应指定 *master..syslogins* 中的有效登录并用引号引起来。要还原到您原来的登录名和 *suid*，可使用 **set proxy** 和您原来的 *login_name*。

有关详细信息，参见 “使用代理”。

quoted_identifier — 确定 Adaptive Server 是否识别分隔标识符。缺省情况下， **quoted_identifier** 为 **off**，所有标识符都必须符合有效标识符的规则。如果使用 **set quoted_identifier on**，通过将标识符用引号引起来，就可以使用以非字母字符开头，并且可以包括在其它情况下不允许的字符或已作为保留字的字符的表名、视图名和列名。当用作系统过程的参数时，分隔标识符不能超过 28 个字节，有可能无法被某些前端产品识别，并且可能产生意外的结果。

当 **quoted_identifier** 为 **on** 时，双引号内的所有字符串都会被当作标识符。字符串和二进制串用单引号引起来。

role — 在当前会话期间打开或关闭指定的角色。当用户登录时，已授予该用户的所有系统角色都将打开。使用 **set role** ***role_name*** **off** 可关闭角色，而使用 **set role** ***role_name*** **on** 可在需要时重新打开该角色。系统角色包括 **sa_role**、**sso_role** 和 **oper_role**。如果您不是当前数据库中的用户并且不存在 “Guest” 用户，就不能设置 **sa_role off**，因为没有您可采用的服务器用户 ID。

role_name — 是系统安全员创建的任何用户定义角色的名称。缺省情况下不打开用户定义的角色。要将用户定义的角色设置为在登录时激活，用户或系统安全员必须使用 **set role on**。

with *passwd* — 指定激活角色所需的口令。如果用户定义的角色具有附加的口令，则必须指定激活该角色的口令。

rowcount — 使 Adaptive Server 在查询影响到指定的行数后停止处理查询（**select**、**insert**、**update** 或 **delete**）。*行数* 可以是无小数点的数字文字或类型为 *integer* 的局部变量。要关闭此选项，可使用：

```
set rowcount 0
```

scan_parallel_degree — 为基于散列的扫描（在未分区表上的并行索引扫描或并行表扫描）指定特定于会话的最大并行度。此数字必须小于或等于 **max scan parallel degree** 配置参数的当前值。*@@scan_parallel_degree* 全局变量存储当前设置。

self_recursion — 确定 Adaptive Server 是否允许触发器再次自行触发（即 **self-recursion**）。缺省情况下，Adaptive Server 不允许触发器中的自递归。只有在当前客户端会话期间才能打开此选项；其影响受限于设置它的触发器的范围。例如，如果设置 **self_recursion on** 的触发器返回或使其它触发器触发，此选项将还原为 **off**。此选项仅在触发器内使用，对用户会话没有任何影响。

session authorization — 除了以下例外，与 **set proxy** 相同：**set session authorization** 遵循 SQL 标准，而 **set proxy** 是 Transact-SQL 扩展。

showplan — 为查询生成处理计划的说明。**showplan** 的结果将在性能诊断中使用。当在存储过程或触发器内使用时，**showplan** 不会输出结果。对于并行查询，**showplan** 输出还包括运行期的调整查询计划（如果适用）。有关详细信息，参见 *Performance and Tuning Guide*。

sort_merge — 允许或禁止在会话中使用排序合并连接。有关详细信息，参见 *Performance and Tuning Guide*。

sort_resources — 为 **create index** 语句生成排序计划的说明。**sort_resources** 的结果可用于确定是将串行还是并行地执行排序操作。当 **sort_resouces** 为 **on** 时， Adaptive Server 将输出排序计划，但不执行 **create index** 语句。有关详细信息，参见 *Performance and Tuning Guide* 中的第 13 章 “Parallel Sorting”。

statistics io — 为在语句中引用的每个表显示以下统计信息：

- 访问表的次数 （扫描计数）
- 逻辑读取的次数 （在内存中访问的页数）
- 和物理读取的次数 （数据库设备访问）

对于每个命令， **statistics io** 都会显示写入的缓冲区数。

如果 Adaptive Server 已配置为强制资源限制， **statistics io** 还会显示 I/O 总成本。有关详细信息，参见 *Performance and Tuning Guide* 中的第 16 章 “Using the set statistics Commands”。

statistics subquerycache — 显示每个子查询的缓存命中数、未命中数以及子查询缓存中的行数。

statistics time — 显示 Adaptive Server 为每个命令进行分析和编译所使用的总时间。对于命令的每一步， **statistics time** 都将显示 Adaptive Server 执行该命令所使用的时间。时间按毫秒和时间周期提供，其确切值取决于计算机。

statistics simulate — 指定优化程序应使用模拟统计信息来优化查询。

strict_dtm_enforcement — 确定服务器是否将事务传播到不支持 Adaptive Server 事务协调服务的服务器。缺省值从 **strict dtm enforcement** 配置参数的值继承。

string_rtruncation — 确定当 **insert** 或 **update** 命令截断 *char* 或 *varchar* 字符串时， Adaptive Server 是否引发 SQLSTATE 例外。如果被截断的字符仅包括空格，则不引发例外。缺省设置为 **off**，它不引发 SQLSTATE 例外，因此字符串将自动截断。

table count — 设置 Adaptive Server 在优化连接时将一次考虑的表个数。所用的缺省值取决于连接中的表个数：

连接的表	一次考虑的表个数
2 – 25	4
26 – 37	3
38 – 50	2

有效值为 0-8。值 0 重新设置缺省行为。大于 8 的值缺省为等于 8。**table count** 可以促进某些连接查询的优化，但也会增加编译成本。

textsize — 指定用 **select** 语句返回 *text* 或 *image* 类型数据的最大大小（以字节表示）。*@@textsize* 全局变量存储当前设置。要将 **textsize** 重新设置为缺省大小 (32K)，可使用以下命令：

```
set textsize 0
```

isql 中的缺省设置为 32K。某些客户端软件设置了其它缺省值。

transaction isolation level — 设置会话的事务隔离级别。当设置此选项后，所有当前或将来的事务都将在该隔离级别上运行。

read uncommitted | 0 — 在隔离级别 0 上的扫描不获取任何锁。因此， 0 级扫描的结果集可能会在扫描过程中发生变化。如果扫描位置因基础表中的更改而丢失，那么必须使用唯一的索引才能重新启动该扫描。如果没有唯一索引，扫描就可能会中止。

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。您可以如下强制 Adaptive Server 选择非唯一索引或表扫描，以替换这一要求：

```
select * from table_name (index table_name)
```

基础表上的活动可能使扫描在完成之前中止。

read committed | 1 — 缺省情况下， Adaptive Server 的事务隔离级别是 **read committed** 或 **1**，即不允许对数据使用共享读取锁。

repeatable read | 2 — 防止非重复的读取。

serializable | 3 — 如果指定隔离级别 3，Adaptive Server 就会将 **holdlock** 应用于事务中的所有 **select** 和 **readtext** 操作，该事务在结束之前将一直持有查询的读取锁。如果同时还设置了链式模式，那么对于所有隐式开始事务的数据检索或修改语句，该隔离级别将继续有效。

transactional_rpc — 控制远程过程调用的处理。如果此选项设置为 **on**，那么当事务待执行时， RPC 就由 Adaptive Server 来协调。如果此选项设置为 **off**，远程过程调用则由 Adaptive Server 节点处理器来处理。缺省值从 **enable xact coordination** 配置参数的值继承。

示例

1. **set showplan, noexec on**
   go
   select * from publishers
   go

   为每个查询返回处理计划的说明，但不执行该查询。

2. **set textsize 100**

   将 **select** 语句返回的 *text* 或 *image* 数据大小上限设置为 100 字节。

3. **set rowcount 4**

   对于每个 **insert**、**update**、**delete** 和 **select** 命令， Adaptive Server 都将在它影响到第四行时停止处理查询。例如：

   ```
   select title_id, price from titles
   title_id  price
   --------  ----------
   BU1032         19.99
   BU1111         11.95
   BU2075          2.99
   BU7832         19.99

   (4 rows affected)
   ```

4. **set char_convert on with error**

   激活字符集转换，将其设置为基于客户端所用字符集的缺省值。当字符不能转换为客户端的字符集时，Adaptive Server 还会通知客户端或应用程序。

5. **set proxy "mary"**

   执行该命令的用户目前在服务器内以登录 “mary”和 Mary 的服务器用户 ID 操作。

6. **set session authorization "mary"**

   示例 5 的另一种表述方式。

7. **set cursor rows 5 for test_cursor**

   为客户端使用 *test_cursor* 请求的每个后继 **fetch** 语句返回五行。

8. **set identity_insert stores_south on**
   go
   insert stores_south (syb_identity)
   values (100)
   go
   set identity_insert stores_south off
   go

将值 100 插入 *stores_south* 表的 IDENTITY 列中，然后禁止在该列中进一步显式插入。注意 **syb_identity** 关键字的用法：Adaptive Server 将用 IDENTITY 列的名称替换该关键字。

9. set transaction isolation level 3

在事务进行过程中，用该事务中的每个 **select** 语句实施读取锁。

10. set role "sa_role" off

使用户的系统管理员角色在当前会话中失效。

11. set fipsflagger on

指示 Adaptive Server 在用户使用 Transact-SQL 扩展时显示警告消息。然后，如果您使用如下非标准 SQL 语句：

```
use pubs2
go
```

Adaptive Server 将显示：

```
SQL statement on line number 1 contains Non-ANSI
text. The error is caused due to the use of use
database.
```

12. set ansinull on

指示 Adaptive Server 按照初级 SQL92 标准评估等于 (=) 和不等于 (!=) 比较以及集合函数的 NULL 值操作数。

如果使用 **set ansinull on**，那么当 Adaptive Server 在一个或多个列或行中发现空值时，集合函数和行集合将引发以下 SQLSTATE 警告：

```
Warning - null value eliminated in set function
```

如果等于或不等于操作数的值为 NULL，比较的结果就是 UNKNOWN。例如，以下查询在 **ansinull** 模式下不返回任何行：

```
select * from titles where price = null
```

如果使用 **set ansinull off**，同一查询将返回 *price* 为 NULL 的行。

13. set string_rtruncation on

使 Adaptive Server 在截断 *char* 或 *nchar* 字符串时生成一个例外。如果 **insert** 或 **update** 语句截断字符串， Adaptive Server 将显示：

```
string data, right truncation
```

```
14.set quoted_identifier on
   go
   create table "!*&strange_table"
        ("emp's_name" char(10),
        age int)
   go
   set quoted_identifier off
   go
```

指示 Adaptive Server 将双引号中的任何字符串当作标识符。当 **quoted_identifier** 为 **on** 时，表名 “ !*&strange_table”和列名 “emp's_name”都是合法的标识符名称。

```
15.set cis_rpc_handling on
```

指定组件集成服务在缺省情况下处理外发的 RPC 请求。

```
16.set transactional_rpc on
```

指定当事务待执行时，由组件集成服务访问方法 （而不是 Adaptive Server 节点处理器）来处理 RPC。

```
17.set role doctor_role on
```

激活 “doctor”角色。此命令由用户用来指定他们要激活的角色。

```
18.set role doctor_role with passwd "physician" on
```

在用户输入口令时激活 “doctor”角色。

```
19.set role doctor_role off
```

使 “doctor”角色失效。

```
20.set scan_parallel_degree 4
```

将未分区表上并行索引扫描和并行表扫描的最大并行度指定为 4。

```
21.set lock wait 5
```

在生成错误消息和失败之前，会话或存储过程中的后续命令等待 5 秒钟以获取锁。

```
22.set lock nowait
```

如果会话或存储过程中的后续命令不能立即获取所请求的锁，则返回错误并失败。

```
23.set lock wait
```

当前会话或存储过程中的后续命令将等待不确定的时间以获取锁。

```
24.set transaction isolation level 2
```

会话中的所有后续查询将在可重复读取事务隔离级别运行。

25.set plan dump dev_plans on

启用将抽象计划捕获到 *dev_plans* 组的功能。

26.set plan load dev_plans on

对当前会话中的查询启用从 *dev_plans* 中装载抽象计划的功能。

```
27.set clientname 'alison'
   set clienthostname 'money1'
   set clientapplname 'webserver2'
```

为该用户分配：

- 客户端名 alison
- 主机名 money1
- 应用程序名 webserver2

注释

- 某些 **set** 选项可以分为一组，例如：
  - **parseonly**、 **noexec**、 **prefetch**、 **showplan**、 **rowcount** 和 **nocount** 控制查询的执行方式。同时将 **parseonly** 和 **noexec** 设置为 on 是毫无意义的。 **rowcount** 的缺省设置是 0 （返回所有行）；其它选项的缺省值是 **off**。
  - **statistics** 选项在每个查询后显示性能统计数据。 **statistics** 选项的缺省设置是 **off**。有关 **noexec**、 **prefetch**、 **showplan** 和 **statistics** 的详细信息，参见 *Performance and Tuning Guide*。
  - **offsets** 和 **procid** 用于在 DB-Library 中解释 Adaptive Server 提供的结果。这些选项的缺省设置为 **on**。
  - **datefirst**、 **dateformat** 和 **language** 影响日期函数、数据顺序和消息显示。当在触发器或存储过程中使用时，这些选项不会还原到它们先前的设置。

    在缺省语言 us_english 中， **datefirst** 为 1 （星期日）， **dateformat** 为 *mdy*，消息以美国英语显示。某些语言的缺省值 （包括 us_english）会产生 Sunday=1、 Monday=2 等，而其它语言缺省值则产生 Monday=1、 Tuesday=2 等。

    set language 意味着 Adaptive Server 应使用它所指定的一周中第一天以及日期格式，但并不替换先前在当前会话中发出的显式 **set datefirst** 或 **set dateformat** 命令。
  - **cursor rows** 和 **close on endtran** 影响 Adaptive Server 处理游标的方式。所有游标的 **cursor rows** 的缺省设置为 1。 **close on endtran** 的缺省设置为 **off**。

- **chained** 和 **transaction isolation level** 使 Adaptive Server 能够以符合 SQL 标准的方式来处理事务。

fipsflagger、**string_rtruncation**、**ansinull**、**ansi_permissions**、**arithabort** 和 **arithignore** 在错误处理和是否符合 SQL 标准这两方面影响 Adaptive Server。

➤ ***注意***

arithabort 和 **arithignore** 选项为版本 10.0 或更高版本进行了重新定义。如果您在应用程序中使用这些选项，请检查它们是否仍产生预期的结果。

- 当启用组件集成服务后，只能使用 **cis_rpc_handling** 和 **transactional_rpc** 选项。
- 如果 Adaptive Server 经过了并行度配置，**parallel_degree** 和 **scan_parallel_degree** 将限制查询的并行度。当使用这些选项时，应提示优化程序限制并行查询，以使工作进程数少于配置参数允许的进程数。如果将这些参数设置为 0，将恢复全服务器范围的配置值。

  如果您指定的数字大于配置参数允许的数字，Adaptive Server 将发出警告消息，并使用配置参数设置的值。

- 如果在触发器或存储过程内使用 **set** 命令，大多数 **set** 选项都将在执行该触发器或过程时还原到它们原来的设置。

  以下选项不会在执行存储过程或触发器时还原到原来的设置。在整个 Adaptive Server 会话中或在您显式将这些选项重新设置之前，它们都会保持不变。

  - **datefirst**
  - **dateformat**
  - **identity_insert**
  - **language**
  - **quoted_identifier**

- 如果指定多个 **set** 选项，第一个语法错误将使所有后继选项被忽略。不过，在出现错误之前指定的选项仍会执行，并且将设置新的选项值。
- 如果为用户分配了客户端名、主机名或应用程序名，这些分配只在当前会话中才有效。当用户下次登录时，必须重新进行这些分配。虽然新名称会出现在 *sysprocesses* 中，当它们并不用于权限检查，并且 **sp_who** 仍会将客户端连接显示为属于初始登录。有关设置用户进程的详细信息，参见*系统管理指南*。

- 除 **showplan** 和 **char_convert** 之外的所有 **set** 选项将立即生效。**showplan** 在以下批处理中生效。下面是两个使用 **set showplan on** 的示例：

```
set showplan on
select * from publishers
go
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)
```

But:

```
set showplan on
go
select * from publishers
go
QUERY PLAN FOR STATEMENT 1 (at line 1).
    STEP 1
        The type of query is SELECT

        FROM TABLE
            publishers
        Nested iteration
        Table Scan
        Ascending Scan.
        Positioning at start of table.

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)
```

角色和 *set* 选项

- 当登录到 Adaptive Server 时，授予您的所有系统定义角色都将自动激活。授予您的用户定义角色不会自动激活。要自动激活授予您的用户定义角色，可使用 **sp_modifylogin**。有关如何使用 **sp_modifylogin** 的详细信息，参见 *Adaptive Server Enterprise 参考手册* 中的 **sp_modifylogin**。使用 **set role** *role_name* **on** 或 **set role** *role_name* **off** 可打开或关闭角色。

  例如，如果已经向您授予系统管理员角色，您就会得到当前数据库中数据库所有者的身份 （和用户 ID）。要获得您的实际用户 ID，可执行以下命令：

  ```
  set role "sa_role" off
  ```

  如果您不是当前数据库中的用户并且不存在 “guest” 用户，则不能设置 **sa_role off**。

- 如果要激活的用户定义角色具有附加的口令，必须指定该口令才能打开该角色。所以，您会输入：

  ```
  set role "role_name" with passwd "password" on
  ```

分布式事务、CIS 和 *set* 选项

- **cis rpc handling** 配置属性和 **set transactional_rpc** 命令的行为因 ASTC 的引入而有所改变。在先前的版本中，启用 **cis rpc handling** 会使**所有** RPC 通过 CIS 的 Client-Library 连接来传递。这样，只要启用 **cis rpc handling**，无论是否明确设置 **transactional_rpc**，其行为都会发生。在 Adaptive Server 12.0 中，这种行为已经改变。如果 **cis rpc handling** 已启用并且 **transactional_rpc** 为 **off**，事务内的 RPC 将通过节点处理器来传递。在事务外执行的 RPC 则通过 CIS 的 Client-Library 连接来发送。

- 当启用 Adaptive Server 分布式事务管理服务后，可以将 RPC 放置在事务内。这些 RPC 称作**事务型 RPC**。所谓事务型 RPC，就是可以在当前事务环境中包括其工作的 RPC。这种远程工作单位可以随本地事务所执行的工作一起提交或回退。

  要使用事务型 RPC，可用 **sp_configure** 启用 CIS 和分布式事务管理，然后发出 **set transactional_rpc** 命令。当 **set transactional_rpc** 为 **on** 且事务待执行时， Adaptive Server （而不是 Adaptive Server 节点处理器）将协调 RPC。

  set transactional_rpc 命令的缺省值为 **off**。 **set cis_rpc_handling** 命令会替换 **set transactional_rpc** 命令。如果设置 **cis_rpc_handling on**，所有外发的 RPC 都将由组件集成服务来处理。

- 有关如何使用 **set transactional_rpc**、 **set cis_rpc_handling** 和 **sp_configure** 的讨论，参见 *Component Integration Services User's Guide*。

使用代理

- 要使用 **set proxy** 或 **set session authorization** 命令，系统安全员必须授予从 *master* 数据库中执行 **set proxy** 或 **set session authorization** 的权限。
- 当用初始的 *login_name* 执行 **set proxy** 或 **set session authorization** 时，将重新建立您原先的身份。
- 不能在事务内执行 **set proxy** 或 **set session authorization**。
- Adaptive Server 只允许一个级别的登录身份更改。因此，当使用 **set proxy** 或 **set session authorization** 更改身份后，必须返回初始身份才能再次将其更改。例如，假定您的登录名是 “ralph”。您想以 “mary” 的身份创建名，以 “joe” 的身份创建视图，然后返回您自己的登录身份。可使用以下语句：

```
set proxy "mary"
    create table mary_sales
    (stor_id  char(4),
    ord_num   varchar(20),
    date      datetime)
grant select on mary_sales to public

set proxy "ralph"

set proxy "joe"
    create view joes_view (publisher, city, state)
    as select stor_id, ord_num, date
    from mary_sales

set proxy "ralph"
```

lock wait

- 缺省情况下，不能立即获取锁的 Adaptive Server 任务将等待，直到不兼容锁被释放，然后继续处理。这等同于未指定 *numsecs* 参数值的 **set lock wait**。
- 使用带有 **lock wait period** 选项的 **sp_configure** 系统过程，就可以设置全服务器范围的锁等待时间。
- 使用 **set lock** 命令在会话级别或存储过程中定义的锁等待时间将替换服务器级别的锁等待时间。
- 如果单独使用 **set lock wait**，且 *numsecs* 没有值，当前会话中的后续命令将等待不确定的时间以获取所请求的锁。
- **sp_sysmon** 过程将报告等待获取锁的任务在等待期间内未能获取锁的次数。

可重复读取事务隔离级别

- 可重复读取隔离级别 （即事务隔离级别 2）在被语句读取的所有页上都持有锁，直到事务结束。
- 如果一个事务读取了表中的行而且第二个事务可以修改这些行，并在第一个事务完成之前提交该修改，则会发生非重复读取。如果第一个事务重新读取这些行，则它们的值将不同，因此最初的读取是不能重复的。可重复的读取在事务期间持有共享锁，并阻塞对锁定行或锁定页上行进行更新的事务。

使用模拟统计信息

- 可以使用 **optdiag** 实用程序的 **simulate** 模式将模拟统计信息装入到数据库。如果已在会话中发出 **set statistics simulate on**，则使用模拟统计信息来优化查询，而不是使用表的实际统计信息。

受 *set* 选项影响的全局变量

- 表 6-34 列出的全局变量包含 **set** 命令所控制的会话选项的信息。

表 6-34：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。
@@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	包含以十六进制表示的会话 **set** 选项。
@parallel_degree	包含当前的最大并行度设置。
@@rowcount	包含最后一个查询所影响的行数。 *@@rowcount* 会被任何不返回行的命令 （例如 **if** 语句）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。 即使 **nocount** 设置为 on， *@@rowcount* 同样会被更新。
@scan_parallel_degree	包含非集群索引扫描的当前最大并行度设置。
@@textsize	包含 **select** 所返回的 *text* 或 *image* 数据的字节数限制。 **isql** 的缺省限制为 32K 字节，该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* 为非链接模式返回 0，为链式模式返回 1。

将 *fipsflagger* 用于数据库中的 Java

- 如果 **fipsflagger** 设置为 on， Adaptive Server 将在使用以下扩展时显示警告消息：
  - **installjava** 实用程序
  - **remove java** 命令
  - 将 Java 类当作数据类型引用的列和变量声明
  - 将 Java-SQL 表达式用于成员引用的语句
- **fipsflagger** 的状态不影响 Java 方法所执行的算术表达式。
- 关于数据库中 Java 的详细信息，参见 *Adaptive Server Enterprise 中的 Java*。

SQL92 一致性

- SQL92 标准规定的行为不同于 Transact-SQL 在先前 Adaptive Server 版本中的行为。缺省情况下，一致的行为将为所有 Embedded-SQL 预编译应用程序启用。其它应用程序如需符合此行为标准，可使用在表 6-35 中列出的 **set** 选项。

表 6-35：为符合初级 SQL92 而设置的选项

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
事务隔离级别	3

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户通常具有 **set** 权限，使用该命令不需要任何特殊的权限。但也有例外，如 **set role**、 **set proxy** 和 **set session authorization**。

要使用 **set role**，系统管理员或系统安全员必须已向您授予相应的角色。如果您因为具有某个角色而获得了访问数据库的权限，那么在使用数据库时就不能关闭该角色。例如，您一般无权使用数据库 *info_plan*，但现在却以系统管理员的角色使用该数据库。如果您在尚未退出 *info_plan* 时设置 **sa_role off**， Adaptive Server 就会返回错误消息。

要使用 **set proxy** 或 **set session authorization**，必须具有系统安全员授予的权限。

参见

命令	**create trigger**、 **fetch**、 **insert**、 **grant**、 **lock table**、 **revoke**
函数	**convert**
实用程序	**isql**、 **optdiag**

setuser

功能

允许数据库所有者充当其它用户。

语法

```
setuser ["user_name"]
```

示例

1.
```
setuser "mary"
go
grant select on authors to joe
setuser
go
```

数据库所有者临时获取 Mary 在数据库中的身份，以便向 Joe 授予 *authors* 表 （由 Mary 拥有）的权限。

注释

- 数据库所有者使用 **setuser** 来获取其它用户的身份，以便使用其它用户的数据库对象，授予权限，创建对象或执行其它操作。
- 当数据库所有者使用 **setuser** 命令时， Adaptive Server 将检查他所充当的用户的权限，而不是该数据库所有者的权限。所充当的用户必须在数据库的 *sysusers* 表中列出。
- **setuser** 只影响本地数据库中的权限。它不影响远程过程调用，也不影响对其它数据库中对象的访问。
- 在发出另一个 **setuser** 命令之前，或者在用 **use** 命令更改当前数据库之前， **setuser** 命令将一直有效。
- 如果在执行 **setuser** 命令时没有指定用户名，则将重新建立数据库所有者的初始身份。
- 系统管理员可以使用 **setuser** 创建将被另一用户拥有的对象。但是，由于系统管理员在权限系统之外操作，他们不能使用 **setuser** 来获取其它用户的权限。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下， **setuser** 权限由数据库所有者拥有，而且不能移交。

参见

命令	grant、 revoke、 use

shutdown

功能

关闭发出该命令的 Adaptive Server、它的本地 Backup Server、或远程 Backup Server。只有系统管理员才能发出该命令。

语法

```
shutdown [srvname] [with {wait | nowait}]
```

关键字和选项

srvname — 是 Backup Server 在 Adaptive Server 的 *sysservers* 系统表中被标识的逻辑名。关闭本地 Adaptive Server 时不需要此参数。

with wait — 是缺省选项。它可正常关闭 Adaptive Server 或 Backup Server。

with nowait — 立即关闭 Adaptive Server 或 Backup Server，而不等待当前执行的语句结束。

➤ ***注意***

使用 **shutdown with nowait** 会导致 IDENTITY 列的值出现间隔。

示例

1. **shutdown**

   关闭发出 **shutdown** 命令的 Adaptive Server。

2. **shutdown with nowait**

   立即关闭 Adaptive Server。

3. **shutdown SYB_BACKUP**

   关闭本地 Backup Server。

4. **shutdown REM_BACKUP**

   关闭远程 Backup Server REM_BACKUP。

注释

- 除非使用 **nowait** 选项，否则 **shutdown** 会通过以下方式正常关闭 Adaptive Server：
  - 禁用登录 （系统管理员除外）
  - 在每个数据库中执行检查点
  - 等待当前正在执行的 SQL 语句或存储过程结束

  不使用 **nowait** 选项而关闭服务器可以减少必须由自动恢复进程完成的工作量。
- 除非使用 **nowait** 选项，否则 **shutdown** *backup_server* 将等待活动的转储和 / 或装载结束。一旦向 Backup Server 发出了 **shutdown** 命令，就不能启动使用该 Backup Server 的任何新的转储或装载。
- 仅在极为特殊的情况下使用 **shutdown with nowait**。在 Adaptive Server 中，在执行 **shutdown with nowait** 前发出 **checkpoint** 命令。
- 使用 **shutdown** 只能暂停本地 Adaptive Server ；不能暂停远程 Adaptive Server。
- 只有在以下情况下可以暂停 Backup Server：
  - *sysservers* 表中列出了该 Backup Server。系统过程 **sp_addserver** 向 *sysservers* 添加条目。
  - 执行该命令的 Adaptive Server 的接口文件中列出了该 Backup Server。
- 使用 **sp_helpserver** 系统过程可确定 Adaptive Server 识别 Backup Server 的名称。指定 Backup Server 的 *name* — 而不是它的 *network_name* — 作为 *srvname* 参数。例如：

```
sp_helpserver
name        network_name   status                                id
----------  -------------  ----------------------------------- --
REM_BACKUP  WHALE_BACKUP   timeouts, no net password encryption 3
SYB_BACKUP  SLUG_BACKUP    timeouts, net password encryption    1
eel         eel
whale       whale                  timeouts, no net password encryption 2
```

要关闭名为 WHALE_BACKUP 的远程 Backup Server，使用以下命令：

```
shutdown REM_BACKUP
```

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，系统管理员具有 **shutdown** 权限，并且不能移交。

参见

命令	alter database
系统过程	sp_addserver、 sp_helpserver

truncate table

功能

删除表中所有的行。

语法

```
truncate table [[database.]owner.]table_name
```

关键字和选项

table_name — 是要截断的表的名称。如果该表位于另一数据库中，则指定数据库名；如果该数据库中存在多个与之同名的表，指定所有者的名称。 *owner* 的缺省值是当前用户，而 *database* 的缺省值是当前数据库。

示例

1. `truncate table authors`

   删除 *authors* 表中的所有数据。

注释

- **truncate table** 删除表中的所有行。该表的结构和所有索引会继续存在，直到发出 **drop table** 命令。绑定到这些列的规则、缺省设置和约束保持绑定，而且触发器继续有效。
- **truncate table** 释放所有索引的分布页；向表添加新行后不要忘记运行 **update statistics**。
- **truncate table** 与不带 **where** 子句的 **delete** 命令等效，但更加快捷。**delete** 每次删除一行，并将所删除的每一行作为一个事务记日志；而 **truncate table** 释放整个数据页，因而产生的日志条目更少。**delete** 和 **truncate table** 都能回收数据及其关联索引所占用的空间。
- 由于所删除的行不记日志， **truncate table** 不能引发触发器。
- 如果其它表中有引用此表的行，就不能使用 **truncate table**。先删除外表的行，或截断外表，然后再截断主表。
- 不能对已分区的表使用 **truncate table** 命令。发出 **truncate table** 命令前，用 **alter table** 命令的 **unpartition** 子句解除表的分区。

  可使用不带 **where** 子句的 **delete** 命令从已分区的表中删除所有行，而无需事先解除分区。此方法通常比使用 **truncate table** 慢，因为它每次只删除一行并对每一个 **delete** 操作记日志。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，表的所有者具有 **truncate table** 权限，且不能移交。要截断系统审计表（*sysaudits_01*、*sysaudits_02*、*sysaudits_03* 等，直到 *sysaudits_08*），您必须是系统安全员。

参见

命令	**create trigger**、**delete**、**drop table**

union 运算符

功能

返回一个包括两个或更多查询的结果的结果集。除非指定了 **all** 关键字，否则将从结果集中删除重复行。

语法

```
select select_list [into clause]
          [from clause] [where clause]
          [group by clause] [having clause]
   [union [all]
       select select_list
          [from clause] [where clause]
          [group by clause] [having clause] ]...
   [order by clause]
   [compute clause]
```

关键字和选项

union — 创建由两个 **select** 语句指定的数据的联合。

all — 包括结果的所有行；不删除重复行。

into — 以在选择列表中指定的列和在 **where** 子句中选择的行为基础，创建新表。联合操作中只有第一个查询可以包含 **into** 子句。

示例

1. ```
   select stor_id, stor_name from sales
   union
   select stor_id, stor_name from sales_east
   ```

   结果集包括 *sales* 表和 *sales_east* 表中 *stor_id* 列和 *stor_name* 列的内容。

2. ```
   select pub_id, pub_name, city into results
   from publishers
   union
   select stor_id, stor_name, city from stores
   union
   select stor_id, stor_name, city from stores_east
   ```

   第一个查询中的 **into** 子句指定：表 *results* 中包含 *publishers*、*stores* 和 *stores_east* 表中指定列的联合的最终结果集。

3.
```
select au_lname, city, state from authors
union
((select stor_name, city, state from sales
union
select stor_name, city, state from sales_east)
union
select pub_name, city, state from publishers)
```

   首先，生成 *sales* 表和 *sales_east* 表中指定列的 **union**。然后，生成上述结果与 *publishers* 的 **union**。最后，生成第二个结果与 *authors* 的 **union**。

注释

- 整个 **union** 查询中可以出现的表的总数是 256。
- **order by** 和 **compute** 子句只能用在 **union** 语句结尾，定义最终结果的顺序或计算摘要值。
- **group by** 和 **having** 子句只能用在各个查询中，而不能用来影响最终结果集。
- 含有 **union** 运算符的 SQL 语句的缺省求值顺序是从左向右。
- 由于 **union** 是二进制操作，所以如果表达式涉及两个以上的查询，就必须添加小括号以便指定求值顺序。
- **union** 语句中的第一个查询可能包含 **into** 子句，该子句创建一个表来保存最终结果集。**into** 语句必须在第一个查询中，否则会接到错误消息 （参见示例 2）。
- **union** 运算符可出现在 **insert...select** 语句中。例如：

  ```
  insert into sales.overall
    select * from sales
    union
    select * from sales_east
  ```

- SQL 语句中的所有选择列表必须具有相同的表达式 （列名、算术表达式、集合函数等等）数。例如，因为第一个选择列表包含的表达式比第二个多，下列语句无效：

  ```
  /* Example of invalid command--shows imbalance */
  /* in select list items */
  select au_id, title_id, au_ord from titleauthor
  union
  select stor_id, date from sales
  ```

- **union** 语句的选择列表中对应列的顺序必须相同，因为 **union** 将按照各个查询给定的顺序对各列一一进行比较。

- **union** 产生的表的列名取自 **union** 语句的**第一个**查询。如果要为结果集定义新的列标题，必须在第一个查询中进行。而且，如果要用新名称引用结果集中的一列 （例如，在 **order by** 语句中），必须在第一个 **select** 语句中用该名称引用该列。例如，以下查询是正确的：

  ```
  select Cities = city from stores
  union
  select city from stores_east
  order by Cities
  ```

- 对作为 **union** 操作的一部分的列的描述则不必相同。表 6-36 列出了结果集中数据类型的规则以及对应的列。

表 6-36：union 操作的结果数据类型

***union* 操作中列的数据类型**	**结果表中对应列的数据类型**
数据类型不兼容 （Adaptive Server 不隐式处理数据转换）	Adaptive Server 返回错误。
两个都是固定长度的字符，长度分别为 L1 和 L2	固定长度的字符，其长度等于 L1 和 L2 中的较大者。
两个都是固定长度的二进制数据类型，长度分别为 L1 和 L2	固定长度的二进制数据类型，其长度等于 L1 和 L2 中的较大者。
其中一个 （或两个）是可变长度的字符	可变长度的字符，其长度等于联合中列的指定长度的最大值。
一个 （或两个）是可变长度的二进制数据类型	可变长度的二进制数据类型，其长度等于联合中列的指定长度的最大值。
两个都是数值数据类型 （例如 *smallint*、 *int*、 *float* 和 *money*）	数据类型等于两列的精度最大值。例如，如果表 A 的一列是 *int* 类型，表 B 的对应列是 *float* 类型，那么结果表中对应列的数据类型是 *float*，因为 *float* 的精度比 *int* 更高。
两列的描述都指定 NOT NULL	指定 NOT NULL。

限制

- 在可更新游标的 **select** 语句中不能使用 **union**。
- **create view** 语句中不能使用 **union** 运算符。
- 子查询中不能使用 **union** 运算符。
- 不能同时使用 **union** 运算符和 **for browse** 子句。
- 不能针对选择 *text* 或 *image* 数据的查询使用 **union** 运算符。

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	以下是 Transact-SQL 扩展： • **insert** 语句的 select 子句中使用 **union** • 如果 **select** 语句中出现 **union** 运算符，在 **select** 语句的 **order by** 子句中指定新的列标题。

参见

命令	**compute Clause**、 **declare**、 **group by 和 having 子句**、 **order by Clause**、 **select**、 **where 子句**
函数	**convert**

update

功能

通过添加数据或者修改现有数据，更改现有行中的数据。

语法

```
update [[database.]owner.]{table_name | view_name}
   set [[[database.]owner.]{table_name.|view_name.}]
       column_name1 =
           {expression1|NULL|(select_statement)} |
       variable_name1 =
           {expression1|NULL|(select_statement)}
       [, column_name2 =
           {expression2|NULL|(select_statement)}]...|
       [, variable_name2 =
           {expression2|NULL|(select_statement)}]...

   [from [[database.]owner.]{view_name [readpast]|
       table_name [readpast]
           [(index {index_name | table_name }
           [ prefetch size ][lru|mru])]}
        [,[[database.]owner.]{view_name [readpast]|
       table_name [readpast]
           [(index {index_name | table_name }
           [ prefetch size ][lru|mru])]}]
   ...]
   [where search_conditions]
   [plan "abstract plan"]

update [[database.]owner.]{table_name | view_name}
   set [[[database.]owner.]{table_name.|view_name.}]
       column_name1 =
          {expression1|NULL|(select_statement)} |
       variable_name1 =
           {expression1|NULL|(select_statement)}
       [, column_name2 =
          {expression2|NULL|(select_statement)}]... |
       [, variable_name2 =
           {expression2|NULL|(select_statement)}]...
   where current of cursor_name
```

关键字和选项

table_name | *view_name* — 是要更新的表或视图的名称。如果表或视图位于另一数据库，指定该数据库的名称；如果该数据库中存在具有该名称的多个表或视图，指定所有者的名称。 *owner* 的缺省值是当前用户，而 *database* 的缺省值是当前数据库。

set — 指定列名或变量名，并赋予新值。该值既可以是表达式，也可以是 NULL。列出多个列名或变量名以及值时，它们之间必须用逗号分隔。

from — 使用其它表或视图中的数据修改要更新的表或视图中的行。

readpast — 使 **update** 命令修改仅数据行锁定表上已解锁的行，或数据页锁定表中已解锁的页上的行。**update...readpast** 自动跳过锁定的行或页，而不会等待锁被释放。

where — 是标准的 **where** 子句 （参见 “where 子句”）。

index *index_name* — 指定用于访问 *table_name* 的索引。更新视图时，不能使用此选项。

prefetch *size* — 为配置了大 I/O 的缓存所绑定的表指定 I/O 大小 （以 KB 为单位）。 *size* 的值是 **2**、 **4**、 **8** 和 **16**。更新视图时不能使用此选项。过程 **sp_helpcache** 显示绑定了某对象的缓存或缺省缓存的有效大小。

如果启用了组件集成服务，则不能为远程服务器使用 **prefetch**。

lru | mru — 指定用于表的缓冲区替换策略。使用 **lru** 可强制优化程序将表读入 MRU/LRU （最近使用最多 / 最近使用最少）链上的缓存中。使用 **mru** 可以从缓存中放弃缓冲区，并将其替换为该表的下一个缓冲区。更新视图时，不能使用此选项。

where current of — 使 Adaptive Server 更新 *cursor_name* 的当前游标位置所指定的表或视图中的行。

index_name — 是要更新的索引的名称。如果未指定索引名，则更新指定表中所有索引的分布统计。

plan *"abstract plan"* — 指定用于优化查询的抽象计划。它可以是用抽象计划语言指定的完整计划或部分计划。有关详细信息，参见 *Performance and Tuning Guide* 中的第 22 章 “Creating and Using Abstract Plans”。

示例

1.
```
update authors
set au_lname = "MacBadden"
where au_lname = "McBadden"
```

   将 *authors* 表中所有的 McBaddens 改为 MacBaddens。

2.
```
update titles
set total_sales = total_sales + qty
from titles, salesdetail, sales
where titles.title_id = salesdetail.title_id
    and salesdetail.stor_id = sales.stor_id
    and salesdetail.ord_num = sales.ord_num
    and sales.date in
       (select max(sales.date) from sales)
```

   修改 *total_sales* 列，反映 *sales* 和 *salesdetail* 表所记录的最近销售情况。在此假定在一个给定日期，对一个给定 title 只记录一组销售情况，而且即时更新。

3.
```
update titles
set price = 24.95
where current of title_crsr
```

   将 *title_crsr* 当前指向的 *titles* 表中书籍的价格更改为 $24.95。

4.
```
update titles
set price = 18.95
where syb_identity = 4
```

   查找 IDENTITY 列等于 4 的行，并将书籍的价格更改为 $18.95。Adaptive Server 将 **syb_identity** 关键字替换为 IDENTITY 列的名称。

5.
```
declare @x money
select @x = 0
update titles
    set total_sales = total_sales + 1,
    @x = price
    where title_id = "BU1032"
```

   使用声明的变量更新 *titles* 表。

6.
```
update salesdetail set discount = 40
    from salesdetail readpast
where title_id like "BU1032"
    and qty > 100
```

   更新其它任务没有锁定的行。

注释

- 使用 **update** 更改已插入行的值。使用 **insert** 添加新行。
- 在一个 **update** 语句中最多可以引用 15 个表。
- **update** 与 **ignore_dup_key**、**ignore_dup_row** 和 **allow_dup_row** 选项（用 **create index** 命令设置）相互作用。（有关详细信息，参见 **create index**。）
- 可以定义一个触发器，使其在对指定表或表的指定列发出 **update** 命令时执行指定的操作。

在 *update* 语句中使用变量

- 可在 **update** 语句的 **set** 子句中对变量赋值，与在 **select** 语句中设置变量类似。
- 在 **update** 语句中使用变量之前，必须使用 **declare** 声明变量，并用 **select** 将其初始化，如示例 5 所示。
- 变量赋值要针对更新中的每个限定行进行。
- 如果 **update** 语句中赋值等式的右侧引用了变量，该变量的当前值将随每行的更新而更改。**当前值**是当前行更新前的变量值。以下示例显示当前值如何随每行的更新而更改。

  假设有以下语句：

```
declare @x int
select @x=0
update table1
    set C1=C1+@x, @x=@x+1
    where column2=xyz
```

  更新开始前 C1 的值是 1。下表显式了每次更新后 @x 变量的当前值如何更改：

行	初始 C1 值	初始 @x 值	计算：C1+@x= 更新后的 C1	更新后的 C1 值	计算：@x+1= 更新后的 @x	更新值
A	1	0	1+0	1	0+1	1
B	1	1	1+1	2	1+1	2
C	2	2	2+2	4	2+1	3
D	4	3	4+3	7	3+1	4

- 如果同一 **update** 语句中给定了多个变量赋值，赋予这些变量的值可能与它们在赋值列表中的顺序有关，但不一定始终如此。要得到最佳结果，不要依赖其放置顺序来确定所赋的值。
- 如果返回了多个行，且对某列变量进行非集合赋值，那么该变量的最终值将只是对最后一行的赋值过程；因此，它不一定有用。
- 为变量赋值的 **update** 语句不需要设置任何限定行的值。
- 如果没有可更新的行，则不对变量赋值。
- **update** 语句的子查询不能引用同一 **update** 语句中已赋值的变量，无论该子查询出现在 **update** 语句的什么位置。
- **update** 语句的 **where** 或 **having** 子句不能引用同一 **update** 语句中已赋值的变量。
- 在由连接驱动的更新中，**update** 语句右侧被赋值的变量使用未更新的表中的列。结果值与为更新选择的连接顺序以及连接表限定的行数有关。
- **update** 语句的回退不影响更新变量，因为所更新变量的值不存储在磁盘上。

将 *update* 用于事务

- 如果设置 **chained transaction mode on**，而当前没有活动的事务，Adaptive Server 将通过 **update** 语句隐式地开始一个事务。要完成更新，必须 **commit** 该事务或 **rollback** 更改。例如：

```
update stores set city = 'Concord'
    where stor_id = '7066'
if exists (select t1.city, t2.city
    from stores t1, stores t2
    where t1.city = t2.city
    and t1.state = t2.state
    and t1.stor_id < t2.stor_id)
        rollback transaction
else
        commit transaction
```

以上批处理开始一个事务（使用链式事务模式）并更新 *stores* 表中的一行。如果它更新的行包含的城市和州信息与该表中的另一商店相同，则将回退对 *stores* 表的更改并结束该事务。否则，它将提交更新并结束该事务。

- Adaptive Server 不会阻止您发出 **update** 语句，在一个给定事务中多次更新单个行。例如，下面两个更新都会影响 *title_id* 为 MC2222 的书籍的价格，因为它的类型 id 是 "mod_cook"。

```
begin transaction
update titles
set price = price + $10
where title_id = "MC2222"
update titles
set price = price * 1.1
where type = "mod_cook"
```

在更新中使用连接

- 在 **update** 的 **from** 子句中执行连接是对用于更新的 ANSI 标准 SQL 语法的 Transact-SQL 扩展。处理 **update** 语句的方法决定了单个语句进行的更新不累积。也就是说，如果 **update** 语句包含连接，而该连接中的另一个表的连接列有多个匹配值，那么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。假设有这样一个连接：

```
update titles set total_sales = total_sales + qty
    from titles t, salesdetail sd
    where t.title_id = sd.title_id
```

对于 *titles* 中每个 *title_id* 和 *salesdetail* 中的一个匹配行，*total_sales* 值仅更新一次。根据查询的连接顺序、表分区情况或者可使用的索引，结果每次都会不同。但每次仅向 *total_sales* 值中添加 *salesdetail* 中的一个值。

如果目的是要返回匹配连接列的值的总和，以下这个使用了子查询的查询将返回正确的结果：

```
update titles set total_sales = total_sales +
    (select isnull(sum(qty),0)
        from salesdetail sd
        where t.title_id = sd.title_id)
    from titles t
```

将 *update* 用于字符数据

- 如果用空字符串 (“”) 更新可变长度的字符数据或 *text* 列，将导致插入一个空格。固定长度字符列将被填充到所定义的长度。
- 所有尾随空格都会从可变长度列数据中删除，除非字符串只包含空格。只包含空格的字符串将被截断为一个空格。如果字符串的长度大于 *char*、 *nchar*、 *varchar* 或 *nvarchar* 列的指定长度，则将自动截断这些字符串，除非将 **string_rtruncation** 设置为 **on**。

- 如果对 *text* 列进行 **update**，将初始化 *text* 列，为其指派有效的文本指针，并分配至少一个 2K 的数据页。

将 *update* 用于游标

- 要更新使用游标的行，用 **declare cursor** 定义该游标，然后打开它。此游标的名称不能是 Transact-SQL 参数或局部变量。此游标必须可更新，否则 Adaptive Server 将返回错误。对游标结果集进行的任何更新操作也会影响派生该游标行的基表行。
- 用 **update...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。如果该 **from** 子句引用多个表或视图（使用连接），可以仅指定被更新的表或视图。

  更新后，游标位置保持不变。只要没有其它 SQL 语句移动该游标的位置，就可以继续更新该游标位置上的行。
- Adaptive Server 允许您更新游标的 *select_statement* 的列列表中没有指定的列，但这些列必须是 *select_statement* 指定的表的一部分。然而，如果用 **for update** 指定要更新的 *column_name_list* 且声明游标，则只能更新那些特定列。

更新 IDENTITY 列

- 不能通过基表或视图更新具有 IDENTITY 属性的列。要确定某列是否定义有 IDENTITY 属性，对该列的基表使用 **sp_help** 系统过程。
- 选入结果表中的 IDENTITY 列在继承 IDENTITY 属性方面遵守以下规则：
  - 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。
  - 如果将 IDENTITY 列作为表达式的一部分来选择，结果列将不继承 IDENTITY 属性。如果该表达式中的任何一列允许空值，它将创建为 NULL；否则，将创建为 NOT NULL。
  - 如果 select 语句包含 group by 子句或集合函数，结果列将不继承 IDENTITY 属性。包含 IDENTITY 列集合的列将创建为 NULL；其它列创建为 NOT NULL。
  - 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，将定义为 NOT NULL。

通过视图更新数据

- 不能 **update** 用 **distinct** 子句定义的视图。
- 如果创建视图时使用了 **with check option**，通过该视图更新的每一行必须能通过该视图查看。例如， *stores_cal* 视图包括 *stores* 表中 *state* 值为 “CA”的所有行。**with check option** 子句对照该视图的选择标准检查每个 **update** 语句：

  ```
  create view stores_cal
  as select * from stores
  where state = "CA"
  with check option
  ```

  如果这样的 **update** 语句将 *state* 更改为不同于 “CA”的值，该语句就会失败。

  ```
  update stores_cal
  set state = "WA"
  where store_id = "7066"
  ```

- 如果创建视图时使用了 **with check option**，从该基视图派生的所有视图都必须符合基视图的选择标准。通过**派生**视图更新的每一行都必须能通过基视图查看。

  以 *stores_cal* 所派生的视图 *stores_cal30* 为例。这一新视图包括位于 California 且付款条件为 “Net 30”的商店的有关信息。

  ```
  create view stores_cal30
  as select * from stores_cal
  where payterms = "Net 30"
  ```

  由于创建 *stores_cal* 时使用了 **with check option**，通过 *stores_cal30* 更新的所有行都必须能通过 *stores_cal* 查看。所有将 *state* 值更改为不同于 “CA”的值的行都会被拒绝。

  请注意， *stores_cal30* 本身并不具有 **with check option** 子句。因此，可通过 *stores_cal30* 用不同于 “Net 30”的 *payterms* 值更新行。例如，以下 **update** 语句将会成功，尽管再也无法通过 *stores_cal30* 查看该行：

  ```
  update stores_cal30
  set payterms = "Net 60"
  where stor_id = "7067"
  ```

- 连接两个或更多表的列的视图不能用来更新行，除非下列两个条件都为真：
  - 该视图没有 **with check option** 子句，而且
  - 所有被更新的列都属于同一基表。

- 包含 **with check option** 子句的连接视图允许 **update** 语句。如果一个表达式包括多个表的列，而其 **where** 子句中出现任何受影响的列，更新都将失败。
- 如果通过连接视图更新行，所有受影响的列都必须属于同一基表。

使用 *index*、*prefetch* 或 *lru* | *mru*

- **index**、**prefetch** 和 **lru | mru** 将替换 Adaptive Server 优化程序所作的选择。使用这些选项应谨慎，并经常通过 **set statistics io on** 查看对性能的影响。有关使用这些选项的详细信息，参见 *Performance and Tuning Guide*。

使用 *readpast*

- **readpast** 选项仅应用于 DOL 锁定表。如果为 allpage 锁定表指定 **readpast**，则将忽略 **readpast**。
- **readpast** 选项与 **holdlock** 选项不兼容。如果在同一 **select** 命令中指定了这两个选项，则产生一个错误，命令终止。
- 如果会话范围隔离级别为 3，将忽略 **readpast** 选项。
- 如果会话的事务隔离级别为 0，使用 **readpast** 的 **update** 命令将不给出警告消息。对于数据页锁定表，这些命令将修改所有页上未被不兼容锁锁定的所有行。对于数据行锁定表，它们影响所有未被不兼容锁锁定的行。
- 如果带有 **readpast** 选项的 **update** 命令应用到两个或多个文本列，并且检查的第一个文本列含有不兼容锁， readpast 锁将跳过此行。如果此列没有不兼容锁，该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，该命令将阻塞直到它可以获取锁并修改此列。
- 有关 **readpast** 锁的详细信息，参见 *Performance and Tuning Guide*。

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	使用 **from** 子句或限定表或列名是由 FIPS 标志程序检测的 Transact-SQL 扩展。通过连接视图或其目标列表包含表达式的视图进行更新，则是运行期之前无法检测且不由 FIPS 标志程序标记的 Transact-SQL 扩展。 使用变量是 Transact-SQL 扩展。 **readpast** 是 Transact-SQL 扩展

权限

缺省情况下，表或视图的所有者具有 **update** 权限，此所有者可以将该权限移交给其他用户。

如果 **set ansi_permissions** 设置为 **on**，则需要在要更新的表上具有 **update** 权限，另外，对出现在 **where** 子句中的所有列和 **set** 子句后的所有列，还必须具有 **select** 权限。缺省情况下， **ansi_permissions** 设置为 **off**。

参见

命令	**alter table**、**create default**、**create index**、**create rule**、 **create trigger**、 **insert**、 **where 子句**
函数	**ptn_data_pgs**
系统过程	**sp_bindefault**、 **sp_bindrule**、 **sp_help**、**sp_helpartition**、 **sp_helpindex**、**sp_unbindefault**、 **sp_unbindrule**

update all statistics

功能

更新给定表的所有统计信息。

语法

```
update all statistics table_name
```

关键字和选项

table_name — 是要更新其统计信息的表的名称。

示例

```
1. update all statistics salesdetail
```

更新 *salesdetail* 表的索引和分区的统计信息。

注释

- **update all statistics** 更新给定表的所有统计信息。 Adaptive Server 保存表中有关页分布的统计信息；在考虑对已分区的表处理查询时是否使用并行扫描、以及处理查询时使用哪些索引时，将用到这些统计信息。查询的优化程度取决于所存储的统计信息的准确程度。
- **update all statistics** 将更新表中所有列的统计信息，并且如果表已分区，还将更新分区的统计信息。
- 如果表未分区， **update all statistics** 仅对表运行 **update statistics**。
- 如果表已分区但没有索引， **update all statistics** 将对该表运行 **update partition statistics**。如果表已分区且有索引，**update all statistics** 将对表运行 **update statistics** 和 **update partition statistics**。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，表的所有者具有 **update all statistics** 权限，且不能移交。

参见

命令	**update statistics**、 **update partition statistics**

update partition statistics

功能

更新有关已分区的表中每个分区内页数的信息。

语法

```
update partition statistics table_name
   [partition_number]
```

关键字和选项

table_nam — 是已分区的表的名称。

partition_number — 是要更新其信息的分区号。如果不指定分区号，**update partition statistics** 将更新指定表中所有分区的数据页数。

注释

- Adaptive Server 保存已分区的表中有关页分布的统计信息，考虑处理查询时是否使用并行扫描时，将用到这些统计信息。查询的优化程度取决于所存储的统计信息的准确程度。如果 Adaptive Server 崩溃，分布信息可能会不正确。

  要查看分布信息是否正确，使用 **data_pgs** 函数确定表的页数，如下所示：

  ```
  select data_pgs(sysindexes.id, doampg)
      from sysindexes
      where sysindexes.id = object_id("table_name")
  ```

  然后，对该表使用 **sp_helpartition** 并在输出的 “ptn_data_pgs” 列中添加该数目。**sp_helpartition** 报告的总页数应略大于 **data_pgs** 返回的数目，因为计算 **sp_helpartition** 的页数时还包括 OAM 页。

  如果分布信息不正确，对表运行 **update partition statistics**。更新分布信息时，**update partition statistics** 将锁定该分区的 OAM 页和控制页。

- 对含有数据的表运行 **update partition statistics**，或为含有数据的表创建索引时，*syspartitions* 的 *controlpage* 列将被更新为指向该分区的控制页。

- **update partition statistics** 更新用于估计表中页数的控制页值。**sp_helpartition** 使用这些统计信息。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，表的所有者具有 **update partition statistics** 权限，并且不能移交。

参见

命令	**alter table**、 **update all statistics**
函数	**ptn_data_pgs**
系统过程	**sp_helpartition**

update statistics

功能

更新指定索引中关于键值分布的信息或指定列中的信息，更新索引中所有列或表中所有列的信息；允许指定直方图的梯级数。

语法

```
update statistics table_name
    [ [index_name] | [( column_list ) ] ]
    [using step values]
    [with consumers = consumers ]

update index statistics table_name [index_name]
    [using step values]
    [with consumers = consumers ]
```

关键字和选项

table_name — 用于 **update statistics** 时， *table_name* 是与索引相关联的表的名称。 *table_name* 是必需的，因为 Transact-SQL 不要求索引名在数据库中唯一。

index_name — 是要更新的索引的名称。如果未指定索引名，则更新指定表中所有索引的分布统计信息。

column_list — 是一个由逗号分隔的各列的列表。

using *step* **values** — 指定直方图的梯级数。对于没有统计信息的列，缺省值是 20。如果在 *sysstatistics* 中已存在列的统计信息，则缺省值是当前的梯级数。

with consumers = *consumers* — 在提供 *column_list* 并且启用并行查询处理的情况下，指定用于排序的消耗程序进程数。

index — 指定索引中所有的列的统计信息都将被更新。

示例

1. update statistics titles (price) using 40 values

产生 *titles* 表 *price* 列的统计信息。

2. update index statistics authors

产生 *authors* 表的所有索引中的所有列的统计信息。

3. update index statistics authors au_names_ix

产生 *authors* 表的 *au_names_ix* 索引中的所有列的统计信息。

注释

- Adaptive Server 保存有关每个索引中键值分布的统计信息，并使用这些统计信息来确定处理查询时要使用的索引。
- 为含有数据的表创建非集群索引时，将对新索引自动运行 **update statistics**。为含有数据的表创建集群索引时，将对所有索引自动运行 **update statistics**。
- 查询的优化程度取决于统计信息的准确程度。如果索引的键值发生重要更改，应对该索引或列重新运行 **update statistics**。如果索引列有大量数据被添加、更改或删除（也就是说，如果怀疑键值的分布已更改），则应使用 **update statistics** 命令。
- 使用带有表名和索引名的 **update statistics** 时，此命令将更新索引前导列的统计信息。如果使用的 **update statistics** 命令仅带有表名，它将更新表中所有索引的前导列的统计信息。
- 使用带有表名和索引名的 **update index statistics** 时，该命令将更新指定索引中所有列的统计信息。如果使用的 **update index statistics** 命令仅带有表名，它将更新表的所有索引中所有列的统计信息。
- 如果指定未建索引的列的名称或索引中非前导列的名称，将为该列产生统计信息，但不创建索引。
- 如果在列的列表中指定多个列，将为第一列产生或更新直方图，并为该列表的所有前缀子集产生或更新密度统计信息。
- 如果使用 **update statistics** 为列或列的列表产生统计信息， **update statistics** 必须扫描表并进行排序。
- 根据设计，**with consumers** 子句用于 RAID 设备上已分区的表。此设备对于 Adaptive Server 来说是单个 I/O 设备，但能产生并行排序所需的高吞吐量。有关详细信息，参见 *Performance and Tuning Guide* 中的第 15 章 “Parallel Sorting”。

- 表 6-37 显示在 **update statistics** 期间执行的扫描类型、获取的锁类型以及何时需要排序。

表 6-37：更新统计信息期间的锁定、扫描和排序

update statistics 指定	执行的扫描和排序	锁定
表名		
allpage 锁表	表扫描，加上每个非集群索引的叶级扫描	级别 1，共享意图表锁、当前页的共享锁
DOL 锁定表	表扫描，加上每个非集群索引和集群索引的叶级扫描（如果存在）	级别 0；脏读
表名和集群索引名		
allpage 锁表	表扫描	级别 1，共享意图表锁、当前页的共享锁
DOL 锁定表	叶级索引扫描	级别 0；脏读
表名和非集群索引名		
allpage 锁表	叶级索引扫描	级别 1，共享意图表锁、当前页的共享锁
DOL 锁定表	叶级索引扫描	级别 0；脏读
表名和列名		
allpage 锁表	表扫描；创建工作表并排序工作表	级别 1，共享意图表锁、当前页的共享锁
DOL 锁定表	表扫描；创建工作表并排序工作表	级别 0；脏读

- **update index statistics** 命令产生一系列更新统计信息操作，这些操作与索引级和列级上的等价命令使用相同的锁定、扫描和排序。例如，如果 *salesdetail* 表在 *salesdetail*（*stor_id*、*ord_num*、*title_id*）上有名为 *sales_det_ix* 的非集群索引，以下命令：

  ```
  update index statistics salesdetail
  ```

  将执行以下 **update statistics** 操作：

  ```
  update statistics salesdetail sales_det_ix
  update statistics salesdetail (ord_num)
  update statistics salesdetail (title_id)
  ```

- **update all statistics** 命令为表中每个索引生成一系列 **update statistics** 操作，然后为所有未建索引的列生成一系列 **update statistics** 操作，最后生成 **update partition statistics** 操作。
- 从更早版本进行升级的过程中，**update statistics** 不对主数据库中的系统表运行。多数系统过程所查询的列上都存在索引，常规使用情况下，不需要对这些表运行 **update statistics**。不过，除了非常规的表之外，允许对所有数据库中的所有系统表运行 **update statistics**。查询时建立在内部结构基础上的表包括 *syscurconfigs*、*sysengines*、*sysgams*、*syslisteners*、*syslocks*、*syslogs*、*syslogshold*、*sysmonitors*、*sysprocesses*、*syssecmechs*、*systestlog* 和 *systransactions*。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，表的所有者具有 **update statistics** 权限，且不能移交。数据库所有者也可以执行此命令，此用户可以通过运行 **setuser** 命令来充当表的所有者。

参见

命令	delete statistics

use

功能

指定要使用的数据库。

语法

use *database_name*

关键字和选项

database_name — 是要打开的数据库的名称。

示例

```
1. use pubs2
   go
```

此时，当前数据库是 *pubs2*。

注释

- 必须先执行 **use** 命令，才能引用数据库中的对象。
- 存储过程或触发器中不能包括 **use**。
- 使用别名，用户可以用另一个名称使用某数据库，从而访问该数据库。使用系统过程 **sp_addalias**。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

如果数据库有 “guest”帐户，所有用户都可以使用该数据库。如果数据库没有 “guest”帐户，要使用该数据库，您必须是数据库的有效用户、拥有该数据库的别名或是系统管理员或系统安全员。

参见

命令	**create database**、 **drop database**
系统过程	**sp_addalias**、 **sp_adduser**、 **sp_modifylogin**

waitfor

功能

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

```
waitfor { delay time | time time | errorexit
   | processexit | mirrorexit }
```

关键字和选项

delay — 指示 Adaptive Server 等待，直到超过指定时间 （最长为 24 小时）。

time — 指示 Adaptive Server 等待到指定时间。

time — 其格式为 *datetime* 数据的一种可接受格式的时间，或字符类型的变量。不能指定日期 — 即不允许指定 *datetime* 值的日期部分。

errorexit — 指示 Adaptive Server 等待，直到内核进程或用户进程异常终止。

processexit — 指示 Adaptive Server 等待，直到内核进程或用户进程因任何原因而终止。

mirrorexit — 指示 Adaptive Server 等待镜像故障。

示例

```
1. begin
       waitfor time "14:20"
       insert chess(next_move)
          values('Q-KR5')
       execute sendmail 'judy'
   end
```

下午 2:20 时， *chess* 表将被我的下一步更新，名为 *sendmail* 的过程将在 Judy 所有的表中插入一行，通知她 *chess* 表中现在存在新的一步。

2.
```
declare @var char(8)
select @var = "00:00:10"
begin
    waitfor delay @var
     print "Ten seconds have passed.  Your time
       is up."
end
```

   10 秒钟后， Adaptive Server 将输出指定消息。

3.
```
begin
   waitfor errorexit
   print "Process exited abnormally!"
end
```

   任何进程异常退出后， Adaptive Server 都将输出指定消息。

注释

- 发出 **waitfor** 命令后，在达到指定的时间或发生指定的事件后，才能使用与 Adaptive Server 的连接。
- 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据。
- 要查找哪些进程已终止，用系统过程 **sp_who** 检查 *sysprocesses* 表。
- 用 **waitfor time** 或 **waitfor delay** 指定的时间可包括小时、分钟和秒。按照第 1 章 “系统和用户定义的数据类型”中的 “日期和时间数据类型”中的说明，使用格式 “hh:mi:ss”。

  例如：

  ```
  waitfor time "16:23"
  ```

  指示 Adaptive Server 等待，直到下午 4:23。语句：

  ```
  waitfor delay "01:30"
  ```

  指示 Adaptive Server 等待一个半小时。
- 系统时间的更改 （例如将时钟调为夏令时）会延迟 **waitfor** 命令。
- 发生镜像故障时，可在 DB-Library 程序中使用 **waitfor mirrorexit** 以通知用户。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都具有 **waitfor** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	begin...end
数据类型	“日期和时间数据类型”
系统过程	sp_who

where 子句

功能

在 **select**、 **insert**、 **update** 或 **delete** 语句中设置搜索条件。

语法

搜索条件紧接在 **select**、**insert**、**update** 或 **delete** 语句的关键字 **where** 之后。如果在一个语句中使用多个搜索条件，用 **and** 或 **or** 连接这些条件。

```
where [not] expression comparison_operator expression

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

where [not] expression is [not] null

where [not]
   expression [not] between expression and expression

where [not]
   expression [not] in ({value_list | subquery})

where [not] exists (subquery)

where [not]
   expression comparison_operator
   {any | all} (subquery)

where [not] column_name join_operator column_name

where [not] logical_expression

where [not] expression {and | or} [not] expression
```

关键字和选项

not — 否定任何逻辑表达式或关键字，例如 **like**、 **null**、 **between**、 **in** 和 **exists**。

expression — 可以是列名、常量、函数、子查询、也可以是任何由算术运算符或逐位运算符连接的列名、常量和函数组合。有关表达式的详细信息，参见第 3 章 “表达式、标识符和通配符”中的“表达式”。

comparison_operator — 是下列某一项：

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

比较 *char*、*nchar*、*varchar* 和 *nvarchar* 数据时，**<** 表示较接近字母表的开头，而 **>** 表示较接近字母表的结尾。

大小写和特殊字符的求值取决于 Adaptive Server 所在计算机的操作系统的归类序列。例如，小写字母优先于大写字母，大写字母优先于数字。

为便于比较，尾随空白将被忽略。例如，“Dirk”与 “Dirk ”相同。

在比较日期时，**<** 表示较早而 **>** 表示较晚。在用于比较运算符时，用引号引起所有字符和日期数据。例如：

```
= "Bennet"
> "94609"
```

有关数据输入规则的详细信息，参见第 1 章 “系统和用户定义的数据类型”中的 “用户定义的数据类型”。

like — 是一个关键字，表示后面的字符串 （用单引号或双引号引起）为匹配模式。**like** 适用于 *char*、 *varchar*、 *nchar*、 *nvarchar* 和 *datetime* 列，但不能用于搜索秒或毫秒。

可将关键字 **like** 和通配符用于 *datetime* 数据以及 *char* 和 *varchar*。如果将 **like** 用于 *datetime* 值， Adaptive Server 就会把日期转换为标准的 *datetime* 格式，然后转换为 *varchar*。由于标准的存储格式不包括秒或毫秒，所以不能用 **like** 和模式来搜索秒或毫秒。

由于 *datetime* 条目可能包含多种日期部分，所以最好使用 **like** 来搜索 *datetime* 值。例如，在名为 *arrival_time* 的列中插入值“9:20”，以下子句：

```
where arrival_time = '9:20'
```

无法找到该值，因为 Adaptive Server 将输入转换为 “Jan 1, 1900 9:20AM”。而以下子句：

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

却可以找到该值。

match_string — 是用引号引起的字符和通配符的字符串。表 6-38 列出这些通配符。

表 6-38：通配符

通配符	含义
%	任何包含 0 个或多个字符的字符串
_	任何单个字符
[ ]	指定范围 ([a-f]) 或集合 ([abcdef]) 内的任何单个字符
[^]	不在指定范围 ([^a-f]) 或集合 ([^abcdef]) 内的任何单个字符

escape — 指定用于搜索通配符实际出现内容的转义字符。

escape_character — 是任何单个字符。有关详细信息，参见第 3 章“表达式、标识符和通配符”中的 “使用 escape 子句”。

is null — 搜索空值。

between — 是表示范围起始点的关键字。使用 **and** 表示范围结束点的值。范围：

```
where @val between x and y
```

是包括端点 x 和 y 的；范围：

```
x and @val < y
```

则不包括 x 和 y。如果指定的第一个值大于第二个值，使用 **between** 进行的查询将不返回任何行。

and — 连接两个条件，而且只有在两个条件都为真的情况下才能返回结果。

如果在一个语句中使用了多个逻辑运算符，通常首先对 **and** 运算符求值。然而，可使用小括号更改执行顺序。

in — 用于选择与值列表中任何一个值相匹配的那些值。比较值可以是常量或列名，列表可以是常量的集合或是一个子查询 （后者更为常见）。（有关对子查询使用 **in** 的信息，参见 *Transact-SQL User's Guide*。）值列表用小括号括起来。

value_list — 是值列表。用单引号或双引号引起字符值，并用逗号将每个值与下一个值分隔开 （参见示例 7）。该列表可以是变量的列表，例如：

```
in (@a, @b, @c)
```

但是不能将包含列表的变量，例如：

```
@a = "'1', '2', '3'"
```

用于值列表。

exists — 与子查询一起使用，测试子查询的某些结果是否存在。（有关详细信息，参见 *Transact-SQL User's Guide*。）

subquery — **select**、 **insert**、 **delete** 或 **update** 语句的 **where** 或 **having** 子句中或子查询中受限制的 **select** 语句 （不允许使用 **order by** 和 **compute** 子句和关键字 **into**）。（有关详细信息，参见 *Transact-SQL User's Guide*。）

any — 用于 **>**、**<** 或 **=** 以及子查询。如果子查询中检索到的所有值都与外层语句的 **where** 或 **having** 子句中的值相匹配，它就会返回结果。（有关详细信息，参见 *Transact-SQL User's Guide*。）

all — 用于 **>** 或 **<** 以及子查询。如果子查询中检索到的所有值都与外层语句的 **where** 或 **having** 子句中的值相匹配，它就会返回结果。（有关详细信息，参见 *Transact-SQL User's Guide*。）

column_name — 是用于比较的列的名称。如果存在任何歧义，用表名或视图名来限定列名。对于具有 IDENTITY 属性的列，可以指定 **syb_identity** 关键字 （如有必要，用表名加以限定），而不是实际列名。

join_operator — 是比较运算符或连接运算符 **=*** 或 ***=** 之一。（有关详细信息，参见 *Transact-SQL User's Guide*。）

logical_expression — 是返回 TRUE 或 FALSE 的表达式。

or — 连接两个条件，如果其中任何一个条件为真，就会返回结果。

如果在一个语句中使用了多个逻辑运算符，通常在 **and** 运算符之后对 **or** 运算符求值。不过，使用小括号可以更改执行顺序。

示例

1. `where advance * $2 > total_sales * price`
2. `where phone not like '415%'`

   查找其电话号码不以 415 开头的所有行。
3. `where au_lname like "[CK]ars[eo]n"`

   查找作者名为 Carson、 Carsen、 Karsen 和 Karson 的行。
4. `where sales_east.syb_identity = 4`

   查找 *sales_east* 表中 IDENTITY 列的值为 4 的那一行。
5. `where advance < $5000 or advance is null`
6. `where (type = "business" or type = "psychology")`
   `and advance > $5500`
7. `where total_sales between 4095 and 12000`
8. `where state in ('CA', 'IN', 'MD')`

   查找状态是列表中的三项之一的那些行。

注释

- **where** 和 **having** 搜索条件一样，只是 **where** 子句中不允许使用集合函数。例如，以下子句是合法的：

  ```
  having avg(price) > 20
  ```

  以下子句不合法：

  ```
  where avg(price) > 20
  ```

  有关使用集合函数的信息，参见第 2 章 “Transact-SQL 函数”；有关示例，参见 “group by 和 having 子句”。
- 连接和子查询是在搜索条件中指定的：有关完整信息，参见 *Transact-SQL User's Guide*。
- 一个 **where** 子句中最多包括 252 个 **and** 和 **or** 条件。
- 在 *char* 或 *varchar* 条目中，有两种方法可以指定实际引号。第一种方法是使用两个 （对）引号。例如，如果用单引号开始某个字符条目，并要在该条目中包括一个单引号，可使用两个单引号：

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

或使用双引号：

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

第二个方法是用相反类型的引号来引起实际的引号。也就是说，用单引号引起包含双引号的条目，反之亦然。示例如下：

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

- 如果输入的字符串超出屏幕宽度，在转到下一行之前输入反斜杠 (\)。
- 在 **where** 子句中将列与常量或变量进行比较时， Adaptive Server 将该常量或变量转换为该列的数据类型，以便优化程序使用索引进行数据检索。例如，将 *float* 表达式与 *int* 列进行比较时，该表达式被转换为 *int*。例如：

  ```
  where int_column = 2
  ```

  选择 *int_column* = 2 的行。
- Adaptive Server 优化查询时，它将计算 **where** 和 **having** 子句中的搜索条件，然后确定哪些条件是可用于选择最佳索引和查询计划的搜索参数 (SARG)。对于查询中的每个表，最多可使用 128 个搜索参数来优化查询。不过，所有搜索条件都用来限定行。有关搜索参数的详细信息，参见 *Performance and Tuning Guide*。
- 标准和一致性

标准	一致性级别
SQL92	初级一致性

参见

命令	**delete**、 **execute**、 **group by 和 having 子句**、 **insert**、 **select**、 **update**
数据类型	日期和时间数据类型
系统过程	**sp_helpjoins**

while

功能

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

```
while logical_expression [plan "abstract plan"]
        statement
```

关键字和选项

logical_expression — 是返回 TRUE、 FALSE 或 NULL 的任何表达式。

plan *"abstract plan"* — 指定用于优化查询的抽象计划。它可以是用抽象计划语言指定的完整计划或部分计划。只能为可优化的 SQL 语句（即访问表的查询）指定计划。有关详细信息，参见 *Performance and Tuning Guide* 中的第 22 章 “Creating and Using Abstract Plans”。

statement — 可以是单个 SQL 语句，但通常是由 **begin** 和 **end** 分隔的 SQL 语句块。

示例

```
1. while (select avg(price) from titles) < $30
   begin
       select title_id, price
           from titles
           where price > $20
       update titles
           set price = price * 2
   end
```

如果平均价格小于 $30，则将 *titles* 表中所有书籍的价格翻一番。如果平均价格仍小于 $30， **while** 循环就不断地使价格翻番。除了确定价格超过 $20 的书目外， **while** 循环中的 **select** 还会指出已完成循环的次数 （Adaptive Server 每返回一个平均结果表示一次循环）。

注释

- 对于在 **while** 循环中执行语句，可以通过 **break** 和 **continue** 命令，在循环内部进行控制。
- 使用 **continue** 命令，可以重新开始 **while** 循环，跳过 **continue** 后的所有语句。使用 **break** 命令，将从 **while** 循环中退出。然后执行关键字 **end** （标记循环结束）后面的所有语句。 **break** 和 **continue** 命令通常由 **if** 测试激活。

  例如：

```
while (select avg(price) from titles) < $30
begin
    update titles
        set price = price * 2
    if (select max(price) from titles) > $50
        break
    else
        if (select avg(price) from titles) > $30
            continue
    print "Average price still under $30"
end

select title_id, price from titles
    where price > $30
```

  只要平均书籍价格小于 $30，该批处理就会一直将 *titles* 表中所有书籍的价格翻番。但是，只要任何书籍的价格超过了 $50， **break** 命令就会停止 **while** 循环。如果平均价格超过 $30， **continue** 命令将阻止执行 **print** 语句。无论 **while** 循环是如何终止的 （正常终止还是通过 **break** 命令终止），最后一个查询都将指出哪些书籍的价格大于 $30。

- 如果嵌套了两个或两个以上的 **while** 循环， **break** 命令将退出到下一个外层循环。先运行最内层循环的所有语句，然后逐一重新开始下一个外层循环。

◆ ***警告！***

如果在一个 while 循环中使用 create table 或 create view 命令， Adaptive Server 将会创建该表或视图的模式，然后确定条件是否为真。如果该表或视图已经存在，就会导致错误。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，所有用户都具有 **while** 权限。也就是说，使用该命令不需要任何权限。

参见

命令	**begin...end**、 **break**、 **continue**、 **goto 标签**

writetext

功能

允许对现有的 *text* 或 *image* 列进行最低程度记录的交互式更新；如果与 **readpast** 选项一并使用，将跳过锁定的行而不会造成阻塞。

语法

```
writetext [[database.]owner.]table_name.column_name
   text_pointer [readpast] [with log] data
```

关键字和选项

table_name.column_name — 是表名和要更新的 *text* 或 *image* 列的名称。如果该表位于另一数据库中，则指定数据库名；如果该数据库中存在多个与之同名的表，指定所有者的名称。 *owner* 的缺省值是当前用户，而 *database* 的缺省值是当前数据库。

text_pointer — 一个 *varbinary(16)* 值，存储指向 *text* 或 *image* 数据的指针。使用 **textptr** 函数可确定该值，如示例 1 所示。 *text* 和 *image* 数据与其它表列不存储在同一组链接页中。它存储在一组单独的链接页中。指向实际位置的指针与数据一起存储； **textptr** 返回该指针。

readpast — 指定此命令应仅修改已解锁的行。如果 **writetext** 命令发现了锁定的行，它将跳过这些行，而不会等待锁被释放。

with log — 对插入的 *text* 或 *image* 数据记日志。使用该选项有助于介质恢复，但是对大块数据记日志将迅速增加事务日志的大小。因此，应确保事务日志位于单独的数据库设备上（有关详细信息，参见 **create database**、 **sp_logdevice** 和*系统管理指南*）。

data — 是要写入 *text* 或 *image* 列的数据。 *text* 数据必须用引号引起。 *image* 数据之前必须有 “0x”。查阅有关您使用的客户端软件的信息，确定该客户软件可容纳的 *text* 或 *image* 数据的最大长度。

示例

1.
```
declare @val varbinary(16)
select @val = textptr(copy) from blurbs
    where au_id = "409-56-7008"
writetext blurbs.copy @val with log "hello world"
```

   该示例将文本指针放置在本地变量 *@val* 中。然后，**writetext** 将文本字符串 “hello world” 放至 *@val* 指向的文本域中。

2.
```
declare @val varbinary(16)
select @val = textptr(copy)
from blurbs readpast
    where au_id = "409-56-7008"
writetext blurbs.copy @val readpast with log "hello
world"
```

注释

- 对于 *text* 和 *image* 数据，可使用 **writetext** 交互插入的文本的最大长度大约是 120KB。
- 缺省情况下，**writetext** 是记日志程度最低的操作；只对页的分配和重新分配记日志，而不对写入数据库时的 *text* 或 *image* 数据记日志。要使用缺省状态 （即最低记日志程度状态）下的 **writetext**，系统管理员必须使用 **sp_dboption** 将 **select into/bulkcopy/pllsort** 设置为 **true**。
- **writetext** 更新现有行中的 *text* 数据。该更新将完全替带所有现有文本。
- **writetext** 操作不会被 **insert** 或 **update** 触发器捕获。
- **writetext** 需要指向 *text* 或 *image* 列的有效文本指针。要具备有效的文本指针， *text* 列必须包含通过 **update** 显式输入的实际数据或空值。

  假定表 *textnull* 包含列 *textid* 和 *x* （其中 *x* 是允许空值的 *text* 列），则该 **update** 将所有的 *text* 值设置为 NULL，并在 *text* 列中指派有效的文本指针：

  ```
  update textnull
  set x = null
  ```

  显式 **insert** 空值不会得到文本指针。

  ```
  insert textnull values (2,null)
  ```

  隐式 **insert** 空值也不会得到文本指针。

  ```
  insert textnull (textid)
  values (2)
  ```

- 对 *text* 列执行 **insert** 和 **update** 是要记录的操作。

- 不能对视图中的 *text* 和 *image* 列使用 **writetext**。
- 如果在更改为多字节字符集后，试图对 *text* 值使用 **writetext**，而没有运行 **dbcc fix_text**，那么命令将会失败，并生成错误消息，指示您对该表运行 **dbcc fix_text**。
- 缺省情况 （不记日志的模式）下，如果正在进行 **dump database**，运行 **writetext** 的速度较慢。
- Client-Library 的函数 **dbwritetext** 和 **dbmoretext** 比 **writetext** 速度更快，且使用的动态内存更少。这些函数最多可插入 2GB 的 *text* 数据。

使用 *readpast* 选项

- **readpast** 选项仅应用于 DOL 锁定表。如果为 allpage 锁定表指定 **readpast**，则将忽略 **readpast**。
- 如果会话范围隔离级别为 3，则忽略 **readpast** 选项。
- 如果会话的事务隔离级别为 0，使用 **readpast** 的 **writetext** 命令将不给出警告消息。如果指定文本列没有被不兼容锁锁定，这些会话隔离级别为 0 的命令将修改文本列。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

缺省情况下，表的所有者具有 **writetext** 权限，此所有者可以将该权限移交给其他用户。

参见

命令	readtext
数据类型	“text 和 image 数据类型”

如需索引，参见第四卷“表格和参考手册索引”。

第四卷“表格和参考手册索引”包括 *Adaptive Server Enterprise 参考手册* 中所有卷的索引条目。