Transact-SQL 用户指南

Sybase® Adaptive Server® Enterprise
12.5

文档 ID：32970-01-1250-01

最后修订日期：2001 年 5 月

Copyright © 1989-2001 by Sybase, Inc. 保留所有权利。

本书适用于 Sybase 数据库管理软件及所有后继版本，除非在新版本或技术注释中另有说明。本文档的信息若有变动，恕不另行通知。本书中所描述的软件按许可协议提供，其使用和复制必须符合协议条款。

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

持有美国许可协议的其它国家的客户可通过上述传真号与客户服务部门联系。其他国际客户请与 Sybase 子公司或当地分销商联系。只在软件的定期发布日提供升级文档。未经 Sybase, Inc. 的事先书面授权，本书的任何部分不能以任何形式、任何手段（电子的、机械的、手工的、光学的或其它）复制、传播或翻译。

Sybase、the Sybase 徽标、ADA Workbench、Adaptable Windowing Environment、Adaptive Component ArchitectureAdaptive Server、Adaptive Server Anywhere、Adaptive Server Enterprise、Adaptive Server Enterprise Monitor、Adaptive Server Enterprise Replication、Adaptive Server Everywhere、Adaptive Server IQ、Adaptive Warehouse、AnswerBase、Anywhere Studio、Application Manager、AppModeler、APT Workbench、APT-Build、APT-Edit、APT-Execute、APT-FORMS、APT-Translator、APT-Library、Backup Server、ClearConnect、Client-Library、Client Services、Data Pipeline、Data Workbench、DataArchitect、Database Analyzer、DataExpress、DataServer、DataWindow、DB-Library、dbQueue、Developers Workbench、Direct Connect Anywhere、DirectConnect、Distribution Director、E-Anywhere、E-Whatever、Embedded SQL、EMS、Enterprise Application Server、Enterprise Application Studio、Enterprise Client/Server、Enterprise Connect、Enterprise Data Studio、Enterprise Manager、Enterprise SQL Server Manager、Enterprise Work Architecture、Enterprise Work Designer、Enterprise Work Modeler、EWA、Gateway Manager、ImpactNow、InfoMaker、Information Anywhere、Information Everywhere、InformationConnect、InternetBuilder、iScript、Jaguar CTS、jConnect for JDBC、KnowledgeBase、MainframeConnect、Maintenance Express、MAP、MDI Access Server、MDI Database Gateway、media.splash、MetaWorks、MySupport、Net-Gateway、Net-Library、NetImpact、ObjectConnect、ObjectCycle、OmniConnect、OmniSQL Access Module、OmniSQL Toolkit、Open Client、Open ClientConnect、Open Client/Server、Open Client/Server Interfaces、Open Gateway、Open Server、Open ServerConnect、Open Solutions、Optima++、PB-Gen、PC APT Execute、PC DB-Net、PC Net Library、Power++、power.stop、PowerAMC、PowerBuilder、PowerBuilder Foundation Class Library、PowerDesigner、PowerDimensions、PowerDynamo、PowerJ、PowerScript、PowerSite、PowerSocket、Powersoft、PowerStage、PowerStudio、PowerTips、Powersoft Portfolio、Powersoft Professional、PowerWare Desktop、PowerWare Enterprise、ProcessAnalyst、Report Workbench、Report-Execute、Replication Agent、Replication Driver、Replication Server、Replication Server Manager、Replication Toolkit、Resource Manager、RW-DisplayLib、RW-Library、S Designor、S-Designor、SDF、Secure SQL Server、Secure SQL Toolset、Security Guardian、SKILS、smart.partners、smart.parts、smart.script、SQL Advantage、SQL Anywhere、SQL Anywhere Studio、SQL Code Checker、SQL Debug、SQL Edit、SQL Edit/TPU、SQL Everywhere、SQL Modeler、SQL Remote、SQL Server、SQL Server Manager、SQL SMART、SQL Toolset、SQL Server/CFT、SQL Server/DBM、SQL Server SNMP SubAgent、SQL Station、SQLJ、STEP、SupportNow、Sybase Central、Sybase Client/Server Interfaces、Sybase Financial Server、Sybase Gateways、Sybase MPP、Sybase SQL Desktop、Sybase SQL Lifecycle、Sybase SQL Workgroup、Sybase User Workbench、SybaseWare、Syber Financial、SyberAssist、SyBooks、System 10、System 11、System XI（徽标）、SystemTools、Tabular Data Stream、Transact-SQL、Translation Toolkit、UNIBOM、Unilib、Uninull、Unisep、Unistring、URK Runtime Kit for UniCode、Viewer、Visual Components、VisualSpeller、VisualWriter、VQL、WarehouseArchitect、Warehouse Control Center、Warehouse Studio、Warehouse WORKS、Watcom、Watcom SQL、Watcom SQL Server、Web Deployment Kit、Web.PB、Web.SQL、WebSights、WebViewer、WorkGroup SQL Server、XA-Library、XA-Server 和 XP Server 是 Sybase, Inc. 的商标。

Unicode 和 Unicode 徽标都是 Unicode, Inc. 的注册商标。

本书中使用的其它公司名和产品名，均可能是相应公司的商标或注册商标。

政府使用、复制或公开本软件受 DFARS 52.227-7013 中的附属条款 (c)(1)(ii)（针对 DOD）和 FAR 52.227-19(a)-(d)（针对民间组织）条款的限制。

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608.

目录

关于本手册 ..... **xvii**

第 1 章 **SQL 构件块** **1**
Adaptive Server 中的 SQL 1
查询、数据修改和命令 2
表、列和行 2
关系操作 3
编译对象 4
命名约定 6
SQL 数据字符 7
SQL 语言字符 7
标识符 8
Adaptive Server 中的表达式 12
算术表达式和字符表达式 13
关系表达式和逻辑表达式 18
Transact-SQL 扩展 19
compute 子句 20
控制流语言 20
存储过程 20
扩展存储过程 21
触发器 21
缺省值和规则 22
错误处理和 set 选项 22
SQL 的其它 Adaptive Server 扩展 23
符合 ANSI 标准 25
美国联邦信息处理标准 (FIPS) 标志程序 25
链式事务和隔离级别 26
标识符 26
SQL 标准样式的注释 26
右截断字符串 27
update 和 delete 语句所要求的权限 27
算术错误 27
同义关键字 28
空值处理 28

Adaptive Server 登录帐号 28
组成员资格 29
角色成员资格 29
获取有关 Adaptive Server 帐号的信息 30
更改口令 30
了解远程登录 31
如何通过 isql 实用程序使用 Transact-SQL 33
缺省数据库 33
通过 isql 使用基于网络的安全服务 34
退出 isql 34
使用 pubs2 和 pubs3 样本数据库 35
样本数据库中有哪些内容？ 35

第 2 章 查询：从表中选择数据 37
什么是查询？ 37
select 语法 38
选择列：select 子句 40
选择所有列：select * 40
选择指定列 41
重新安排列顺序 41
在查询结果中重命名列 42
表达式 42
选择 text 和 image 值 49
选择列表摘要 50
用 distinct 消除重复查询结果 51
指定表：from 子句 52
选择行：where 子句 53
比较运算符 54
范围 （between 和 not between） 56
列表 （in 和 not in） 57
模式匹配 59
匹配字符串：like 60
字符串和引号 65
“未知”值：NULL 65
带有逻辑运算符的连接条件 70

第 3 章 使用集合、分组和排序 73
集合函数的使用 73
集合函数和数据类型 75
count 与 count(*) 76
使用带有 distinct 的集合函数 76
空值和集合函数 77
将查询结果分组：group by 子句 78

group by 语法 80
使用 group by 在查询中引用其它列 82
表达式和 group by 85
使用 group by 嵌套集合 86
空值和 group by 86
where 子句和 group by 87
group by 和 all 89
使用不带 group by 的集合 90
选择数据组：having 子句 91
having、 group by 以及 where 子句交互作用的方式 92
使用不带 group by 的 having 95
排序查询结果：order by 子句 96
order by 和 group by 97
与 select distinct 一起使用的 order by 及 group by 98
汇总数据组：compute 子句 99
行集合和 compute 101
在 compute 后指定多列 103
使用多个 compute 子句 103
将一个集合用在多列上 104
在同一 compute 子句中使用不同集合 105
最终值：不带 by 的 compute 106
组合查询：union 运算符 107
union 查询指南 109
与其它 Transact-SQL 命令一起使用 union 111

第 4 章 连接：从若干表中检索数据 113
连接如何工作 114
连接语法 114
连接和关系模型 115
如何使连接结构化 115
from 子句 116
where 子句 117
如何处理连接 119
等值连接和自然连接 120
使用其它条件的连接 121
未基于等同性的连接 122
自连接和相关名 123
不均等连接 124
不均等连接和子查询 126
连接两个以上的表 127
外部连接 129
内部表和外部表 129
外部连接限制 130
用于外部连接的视图 130

ANSI 内部和外部连接 131
ANSI 外部连接 136
Transact-SQL 外部连接 147
空值如何影响连接 151
确定要连接的表列 152

第 5 章 子查询：在其它查询中使用查询 153
子查询工作方式 153
子查询语法 154
子查询限制 154
使用子查询的示例 155
限定列名 156
带相关名的子查询 157
多重嵌套 158
update、 delete 和 insert 语句中的子查询 159
条件语句中的子查询 160
用子查询替代表达式 160
子查询类型 161
表达式子查询 162
定量判定子查询 164
带 in 的子查询 170
带 not in 的子查询 172
使用包含 NULL 的 not in 的子查询 173
带 exists 的子查询 173
带 not exists 的子查询 176
使用 exists 查找交集与差集 176
使用相关子查询 178
包含 Transact-SQL 外部连接的相关子查询 179
带相关名的相关子查询 180
带比较运算符的相关子查询 180
在 having 子句中的相关子查询 182

第 6 章 使用和创建数据类型 183
Transact-SQL 数据类型的工作原理 183
使用系统提供的数据类型 184
精确数值类型：整数 185
精确数值类型：小数 186
近似数值数据类型 186
货币数据类型 187
日期和时间数据类型 187
字符数据类型 188
二进制数据类型 192
bit 数据类型 193

timestamp 数据类型 194
sysname 数据类型 194
数据类型之间的转换 194
混合型算术和数据类型层次 195
使用 money 数据类型 196
确定精度和标度 197
创建用户定义的数据类型 197
指定长度、精度和标度 198
指定空值类型 199
将规则和缺省值与用户定义的数据类型相关联 199
使用 IDENTITY 属性创建用户定义的数据类型 199
从其它用户定义的数据类型创建 IDENTITY 列 200
删除用户定义的数据类型 200
获取有关数据类型的信息 200

第 7 章 创建数据库和表 203
何为数据库和表？ 203
强制实现数据库的数据完整性 204
数据库中的权限 205
使用和创建数据库 206
选择数据库：use 207
创建用户数据库：create database 207
变更数据库的大小 210
删除数据库 211
创建表 211
创建表示例 212
选择表名 213
create table 语法 213
允许空值 215
使用 IDENTITY 列 217
使用临时表 220
将表创建于不同的数据库中 224
管理表中的标识间隔 224
控制标识间隔的参数 225
identity burning set factor 与 identity_gap 的比较 225
设置特定于表的标识间隔 227
更改特定于表的标识间隔 227
显示特定于表的标识间隔信息 228
由于插入、删除、 identity grab size 和回退原因产生的间隔 230
如果表插入达到 IDENTITY 列的最大值 231
为表定义完整性约束 232
指定表级或列级约束 233
为约束创建错误消息 233
创建检查约束后 234

指定缺省列值 234
指定唯一约束和主键约束 235
指定参照完整性约束 236
指定检查约束 239
设计使用参照完整性的应用程序 240
如何设计和创建表 241
制作设计草图 242
创建用户定义的数据类型 243
选择能接受空值的列 243
定义表 244
从查询结果中创建新表：select into 244
检查错误 247
对 IDENTITY 列使用 select into 248
改变现有表 250
使用 select * 的对象不列出对表的更改 251
对远程表使用 alter table 252
添加列 252
删除列 254
修改列 256
添加、删除和修改 IDENTITY 列 260
数据复制 261
修改锁定方案和表模式 263
变更具有用户定义数据类型的列 263
alter table 生成的错误和警告 264
重命名表和其它对象 267
删除表 268
给用户分配权限 269
获得有关数据库和表的信息 270
获得有关数据库的帮助 270
获得有关数据库对象的帮助 271

第 8 章 添加、更改和删除数据 277
简介 277
权限 278
参照完整性 278
事务 279
使用样本数据库 279
数据类型输入规则 280
char、 nchar、 unichar、 univarchar、 varchar、 nvarchar 和 text 280
datetime 和 smalldatetime 281
binary、 varbinary 和 image 285
money 和 smallmoney 285
float、 real，和 double precision 286

decimal 和 numeric 286
int、 smallint 和 tinyint 287
timestamp 287
添加新数据 287
insert 语法 288
用 values 添加新行 288
插入数据到指定列 289
用 select 添加新行 296
更改现有数据 299
update 语法 300
将 set 子句用于 update 301
将 where 子句用于 update 302
将 from 子句用于 update 303
用连接更新 303
更新 IDENTITY 列 304
改变 text 和 image 数据 304
删除数据 305
将 from 子句用于 delete 306
从 IDENTITY 列删除 307
删除表中的所有行 307
truncate table 语法 308

第 9 章 视图：限制访问数据 309
视图如何工作 309
视图的优点 310
视图示例 312
创建视图 313
create view 语法 314
使用 select 语句和 create view 315
创建视图后 319
使用 with check option 验证视图的选择标准 319
通过视图检索数据 321
视图解析 322
重定义视图 322
重命名视图 323
改变或删除基础对象 324
通过视图修改数据 324
更新视图的限制 325
删除视图 329
使用视图作为安全性机制 329
获取有关视图的信息 330
通过 sp_help 获取有关视图的帮助 330
使用 sp_helptext 显示视图信息 331
使用 sp_depends 列出相关对象 332

列出数据库中的所有视图 332

第 10 章 **使用查询中的内部函数 333**
返回数据库信息的系统函数 333
使用系统函数的示例 337
用于字符串或表达式的字符串函数 338
使用字符串函数的示例 342
并置 346
嵌套字符串函数 347
文本函数用于 text 和 image 数据 348
Readtext 349
集合函数 351
数学函数 352
日期函数 355
数据类型转换函数 360
转换二进制式数据 364
转换错误 366
安全性函数 369

第 11 章 **创建表的索引 371**
索引如何工作 371
比较创建索引的两种方式 372
使用索引指南 372
创建索引 373
create index 的语法 374
索引多个列：组合索引 374
使用 unique 选项 375
非唯一索引中包含 IDENTITY 列 375
升序和降序索排列引列值 376
使用 fillfactor、 max_rows_per_page 和 reservepagegap 377
使用聚簇或非聚簇索引 378
在段上创建聚簇索引 379
在已分区表上创建聚簇索引 379
指定索引选项 380
使用 ignore_dup_key 选项 380
使用 ignore_dup_row 和 allow_dup_row 选项 380
使用 sorted_data 选项 381
使用 on segment_name 选项 382
删除索引 382
确定表上存在哪些索引 383
更新索引的统计信息 384
更新分区统计 385

第 12 章 为数据定义缺省值和规则 387
缺省值和规则如何执行 387
创建缺省值 388
create default 语法 389
绑定缺省值 390
解除绑定缺省值 392
缺省值怎样影响 NULL 值 392
创建缺省值后 393
删除缺省值 393
创建规则 394
create rule 的语法 394
绑定规则 395
规则和 NULL 值 397
定义规则后 397
解除绑定规则 397
删除规则 398
获得有关缺省值和规则的信息 399

第 13 章 使用批处理和控制流语言 401
简介 401
批处理相关的规则 402
使用批处理的示例 404
作为文件提交的批处理 406
使用控制流语言 407
if...else 408
case 表达式 409
begin...end 418
while 和 break...continue 418
declare 和局部变量 421
goto 421
return 422
print 422
raiserror 424
创建 print 和 raiserror 所用的消息 425
waitfor 426
注释 427
局部变量 429
声明局部变量 429
局部变量和 select 语句 430
局部变量和 update 语句 431
局部变量和子查询 431
局部变量和 while 循环以及 if…else 块 432
变量和空值 433
全局变量 434

事务和全局变量 434
受 set 选项影响的全局变量 436
全局变量中的语言和字符集信息 438
监控系统活动的全局变量 438
存储在全局变量中的服务器信息 439
全局变量以及 text 和 image 数据 440

第 14 章 使用存储过程 441
存储过程如何工作 441
创建并使用存储过程的示例 442
存储过程和权限 444
存储过程及性能 444
创建和执行存储过程 445
参数 445
缺省参数 448
使用多个参数 451
过程组 452
在 create procedure 中使用 with recompile 453
在 execute 中使用 with recompile 453
过程中的嵌套过程 453
使用存储过程的临时表 454
在存储过程中设置选项 455
创建存储过程后 456
执行存储过程 456
返回存储过程的信息 457
返回状态 458
检查过程中的角色 460
返回参数 460
与存储过程关联的限制 465
限定过程内的名字 466
重命名存储过程 466
过程引用的重命名对象 466
使用存储过程作为安全机制 467
删除存储过程 467
系统过程 467
执行系统过程 468
系统过程的权限 468
系统过程类型 468
Sybase 提供的其它过程 474
获取有关存储过程的信息 475
用 sp_help 获取报告 475
用 sp_helptext 查看过程的源文本 476
用 sp_depends 标识相关对象 476
用 sp_helprotect 标识权限 477

第 15 章 使用扩展存储过程 479
概述 479
XP Server 480
动态链接库支持 480
Open Server API 481
创建和使用 ESP 的示例 481
ESP 和权限 483
ESP 和性能 483
为 ESP 创建函数 484
用于 ESP 开发的文件 484
Open Server 数据结构 484
Open Server 返回代码 485
简单 ESP 函数概述 485
ESP 函数示例 486
建立 DLL 491
注册 ESP 495
使用 create procedure 495
使用 sp_addextendedproc 496
删除 ESP 496
重命名 ESP 497
执行 ESP 497
系统 ESP 498
获取有关 ESP 的信息 499
ESP 例外和消息 499
手工启动 XP Server 500

第 16 章 触发器：强制实施参照完整性 501
触发器如何工作 501
使用触发器与使用完整性约束的对比 502
创建触发器 503
create trigger 语法 503
使用触发器来维护参照完整性 505
测试触发器测试表的数据修改 506
插入触发器的示例 508
删除触发器的示例 509
更新触发器的示例 511
多行注意事项 516
使用多行的插入触发器示例 516
使用多行的删除触发器示例 517
使用多行的更新触发器示例 518
使用多行的条件插入触发器示例 519
回退触发器 520
嵌套触发器 522
触发器自递归 523

与触发器相关的规则 525
触发器和权限 525
触发器限制 525
隐式和显式空值 526
触发器和性能 527
触发器中的 set 命令 527
重命名和触发器 527
触发器提示 528
禁用触发器 528
删除触发器 529
获取有关触发器的信息 529
sp_help 530
sp_helptext 530
sp_depends 531

第 17 章 游标：逐行访问数据 533
游标的工作方式 533
Adaptive Server 如何处理游标 534
声明游标 537
declare cursor 语法 538
游标类型 539
游标范围 539
游标扫描与游标结果集 541
使游标变为可更新 542
确定可被更新的列 543
打开游标 544
使用游标读取数据行 544
fetch 语法 545
检查游标状态 546
使用每个 fetch 命令获取多个行 546
检查读取的行数 547
使用游标更新和删除行 548
更新游标结果集的行 548
删除游标结果集的行 549
关闭并释放游标 550
一个使用游标的示例 550
在存储过程中使用游标 553
游标和锁定 555
游标锁定选项 556
获取有关游标的信息 556
使用浏览模式而不用游标 557
浏览表 557
浏览模式限制 558
为用于浏览的新表加盖时间戳 558

为现有表加盖时间戳 558
比较 timestamp 值 559
连接游标处理和数据修改 559
可影响游标位置的更新和删除 559
执行 delete 或 update 命令后没有连接的游标定位 560
更新和删除对连接游标的影响 560
连接列缓冲对连接游标的影响 562
建议 565

第 18 章 事务：维护数据一致性和恢复 567
事务工作方式 567
事务和一致性 569
事务和恢复 570
使用事务 570
在事务中允许使用数据定义命令 571
不允许在事务中使用的系统过程 572
开始和提交事务 572
回退和保存事务 573
检查事务的状态 574
嵌套事务 576
事务示例 577
选择事务模式和隔离级别 578
选择事务模式 578
选择隔离级别 580
符合 SQL 标准 587
使用 lock table 命令改善性能 588
使用 lock table 命令的 wait/nowait 选项 589
在存储过程和触发器中使用事务 589
错误和事务回退 591
事务模式和存储过程 594
在事务中使用游标 595
使用事务时应考虑的问题 597
事务备份和恢复 597

第 19 章 锁定命令和选项 599
设置等待锁的时间限制 599
lock table 命令的 wait/nowait 选项 599
设置会话级等待获取锁的时间限制 600
设置全服务器范围的锁等待限制 601
锁等待超时次数的信息 602
队列处理的 readpast 锁定 602
Readpast 语法 602
Readpast 查询期间的不兼容锁 603

allpages 锁表和 readpast 查询 603
带 readpast 的隔离级别 select 查询的作用 603
带有 readpast 和隔离级别的数据修改命令 604
text 和 image 列和 readpast 605
Readpast 锁定示例 605

附录 A **pubs2 数据库 607**
pubs2 数据库中的表 607
publishers 表 607
authors 表 608
titles 表 610
titleauthor 表 617
salesdetail 表 619
sales 表 624
stores 表 625
roysched 表 626
discounts 表 629
blurbs 表 629
au_pix 表 631
pubs2 数据库框图 634

附录 B **pubs3 数据库 635**
pubs3 数据库中的表 635
publishers 表 635
authors 表 636
titles 表 638
titleauthor 表 645
salesdetail 表 647
sales 表 651
stores 表 653
store_employees 表 653
roysched 表 659
discounts 表 661
blurbs 表 662
pubs3 数据库框图 664

索引 665

关于本手册

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

读者

不熟悉 SQL 的 Sybase® Adaptive Server® Enterprise 数据库管理系统的用户可在一开始就将此指南作为一本教科书。Novice SQL 用户应该集中精力于本书的第一部分。第二部分讲述了高级主题。

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

如何使用本手册

此书是 Transact-SQL 的完整指南。它包含一介绍性的章节，给出了 SQL 的概述。其余章节分为两个主要部分：基本概念和高级主题。

第 1 章“SQL 构件块”讲述了 SQL 使用的命名约定和 Transact-SQL 添加的增强功能（也称作扩充）。也包括了如何用 isql 实用程序开始使用 Transact-SQL。Sybase 建议所有用户阅读此章内容。

“第一部分：基本概念 " 包括 2 — 9 章。这些章向您介绍了 SQL 的基本功能。新接触 SQL 的用户应该在阅读第二部分之前熟悉这些章所讲述的概念，有经验的 SQL 用户可以通过浏览这些章节来了解其中所介绍的 Transact-SQL 扩充，并复习一下这部分资料。

“第二部分：高级主题”包括 10 — 18 章。这些章更为详细地讲述 Transact-SQL。此处讲述了大部分 Transact-SQL 扩充内容。熟悉 SQL、但不熟悉 Transact-SQL 的用户，应集中精力于这些章节。

本指南中的示例基于 pubs2 和 pubs3（特别注明处）样本数据库。为了最好地利用 Transact-SQL 用户指南，新用户应该全部浏览这些示例。询问系统管理员如何获得 pubs2 和 pubs3 的原始副本。有关这些数据库的完整讲述，参见附录 A“pubs2 数据库”和附录 B“pubs3 数据库”。

可用 Adaptive Server 独立程序 isql 使用 Transact-SQL。isql 程序是从操作系统直接调用的实用程序。

相关文档

Sybase® Adaptive Server® Enterprise 文档包含以下文档：

- 针对所用平台的发行公告 — 包含未能及时写入本书的最新信息。

  最新版本的发行公告可以从 WWW 上获得。要了解本产品 CD 发行之后增加的重要产品或文档信息，可使用 Sybase Technical Library。

- 针对所用平台的*安装指南* — 讲述所有 Adaptive Server 及相关 Sybase 产品的安装、升级和配置过程。
- 针对所用平台的*配置 Adaptive Server Enterprise* — 介绍为 Adaptive Server 执行特定的配置任务。
- *Adaptive Server Enterprise 的新增功能？* — 阐述 12.5 版 Adaptive Server 中的新功能、为支持这些功能所进行的系统修改以及可能影响现有应用程序的修改。
- *Transact-SQL 用户指南* — 介绍 Transact-SQL，即关系数据库语言的 Sybase 增强版。本手册可用作数据库管理系统初级用户的教科书。本手册还包括 pubs2 和 pubs3 样本数据库的说明。
- *系统管理指南* — 提供有关管理服务器和数据库的详细信息。本手册包含有关管理物理资源、安全性、用户数据库和系统数据库以及指定字符转换、国际语言和排序顺序设置的指导。
- *参考手册* — 包含所有 Transact-SQL 命令、函数、过程和数据类型的详细信息。本手册还包含 Transact-SQL 保留字和系统表定义的列表。
- *性能和调优指南* — 解释如何调优 Adaptive Server，使其达到最佳性能。本手册包含以下方面的信息：影响性能的数据库设计问题、查询优化、如何为大型数据库调优 Adaptive Server、磁盘和高速缓存的问题，以及锁和游标对性能的影响。
- *实用程序指南* — 介绍在操作系统级执行的 Adaptive Server 实用程序，如 isql 和 bcp。
- *快速参考指南* — 在这本袖珍版指南，将提供命令、函数、系统过程、扩展系统过程、数据类型和实用程序的名称和语法的综合列表。同时只提供印刷手册。
- *系统表框图* — 以海报格式说明系统表和它们的实体关系。同时只提供印刷手册。
- *Error Messages and Troubleshooting Guide* — 解释如何分析常见错误消息并阐述用户常见系统问题的解决方法。
- *组件集成服务用户指南* —解释如何使用 Adaptive Server 组件集成服务功能来连接远程 Sybase 和非 Sybase 数据库。

- *Adaptive Server Enterprise 中的 Java* — 阐述在 Adaptive Server 数据库中如何安装 Java 类，如何将它们用作数据类型、函数及存储过程。
- *在高可用性系统中使用 Sybase 故障切换* — 介绍如何在高可用性系统中使用 Sybase 故障切换功能将 Adaptive Server 配置成协同服务器。
- *使用 Adaptive Server 分布式事务管理功能* — 解释如何在分布式事务处理环境中配置、使用 Adaptive Server DTM 功能，如何排除 Adaptive Server DTM 功能的故障。
- *EJB Server 用户指南* — 解释如何使用 EJB Server 在 Adaptive Server 中部署和执行 Enterprise JavaBeans。
- *XA Interface Integration Guide for CICS, Encina, and TUXEDO* — 介绍在 X/Open XA 事务管理器中如何使用 Sybase 的 DTM XA 接口。
- *词汇表* — 定义 Adaptive Server 文档中使用的技术术语。
- *Sybase jConnect for JDBC 程序员手册* — 阐述 jConnect for JDBC 产品并解释如何用它访问存储在关系数据库管理系统中的数据。
- *全文本搜索专业数据存储用户指南* — 阐述如何使用全文本搜索功能和 Verity 来搜索 Adaptive Server Enterprise 数据。
- *Historical Server 用户指南* — 阐述如何使用 Historical Server 获取 SQL Server 和 Adaptive Server 的性能信息。
- *Monitor Server 用户指南* — 阐述如何使用 Monitor Server 从 SQL Server 和 Adaptive Server 获取性能统计信息。
- *Monitor Client Library 程序员指南* — 阐述如何编写能够访问 Adaptive Server 性能数据的 Monitor Client Library 应用程序。

其它信息来源

使用 Sybase Technical Library CD 和 Technical Library Product Manuals Web 站点来了解有关产品的更多信息：

- Technical Library CD 含有产品手册，并随软件提供。使用 DynaText 浏览器（可从 http://www.sybase.com/detail/1,3693,1010661,00.html 上的 Product Manuals 下载）可以方便地获得产品的技术信息。

  关于安装和启动 Technical Library 的指导，参见文档包中的 *Technical Library 安装指南*。

- Technical Library Product Manuals Web 站点是 Technical Library CD 的 HTML 版，可以使用标准的 Web 浏览器访问。除产品手册外，还可以链接到 Technical Documents Web 站点（以前称为 Tech Info Library）、Solved Cases 页面和 Sybase/Powersoft 新闻组。

要访问 Technical Library Product Manuals Web 站点，可访问 http://www.sybase.com/support/manuals/ 上的 Product Manuals。

网上的 Sybase 认证

Sybase Web 站点上的技术文档不断在更新。

❖ **有关产品认证的最新信息**

1 将您的 Web 浏览器指向 http://www.sybase.com/support/techdocs/ 上的 Technical Documents。

2 从左边的导航栏中选择 Products（产品）。

3 从产品列表中选择一个产品名称。

4 选择 Certification Report（认证报告）过滤器，指定时间帧，然后单击 Go（转到）。

5 单击 Certification Report（认证报告）标题显示此报告。

❖ **有关 EBF 和更新的最新信息**

1 将您的 Web 浏览器指向 http://www.sybase.com/support/techdocs/ 上的 Technical Documents。

2 选择 EBFs/Updates（EBF ／更新）。对于现有 Web 帐户，在提示时输入用户名和口令信息，或者建立一个新帐户（免费服务）。

3 指定时间范围并单击 Go（转到）。

4 选择一个产品。

5 单击 EBF/Update（EBF ／更新）标题显示此报告。

❖ **创建 Sybase Web 站点（包括支持页）的个人化视图**

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

1 将您的 Web 浏览器指向 http://www.sybase.com/support/techdocs/ 上的 Technical Documents。

2 单击 MySybase 并创建 MySybase 配置文件。

约定

以下各部分将说明在本手册中使用的约定。

格式化 SQL 语句

SQL 是形式自由的语言。没有规定每一行中的单词数量或者必须折行的地方。然而，为便于阅读，本手册中所有示例和语法语句都有统一格式，这样语句的每个子句都启用一个新行。有多个成分的子句将多占一行，并且多占的行有缩进。

字体和语法约定

表 1 列出了在本手册中出现的语法语句约定：

表 *1*：本手册的字体和语法约定

元素	示例
命令名、命令选项名、实用程序名、实用程序标记和其它关键词用此 font 显示。	select sp_configure
数据库名、数据类型、文件名和路径名都用此 font 显示。	master 数据库
变量或代表填充值的词为*斜体*。	`select` *`column_name`* `from` *`table_name`* `where search_conditions`
小括号表示必须作为命令的一部分输入。	`compute` *`row_aggregate`* `(`*`column_name`*`)`
大括号表示必须至少选择括号中的选项之一。不要输入大括号。	`{cash, check, credit}`
中括号表示可以选择一个或多个括号中的选项，也可不选。不要输入中括号。	`[anchovies]`
竖线表示只可以选择一个所显示的选项。	`{die_on_your_feet | live_on_your_knees | live_on_your_feet}`
逗号表示可以选择任意多个所显示的选项，用逗号将这些选项隔开，使其成为命令的一部分。	`[extra_cheese, avocados, sour_cream]`

省略号 (. . .) 意味着可以将最后一个单元*重复*任意多次。	`buy thing = price [cash | check | credit]`
	`[, thing = price [cash | check | credit]]...`
	必须至少购买一种东西并给出其价格。可以选择一种付款方式：选择方括号中的一项。还可以选购其它东西：购买数量随意。对于要买的每种东西，给出其名称、价格和付款方式（可选）。

- 语法语句（显示命令的语法和所有选项）显示如下：

  sp_dropdevice [*device_name*]

  或者，对于具有多个选项的命令：

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

  在语法语句中，关键字（命令）采用常规字体，而标识符为小写。斜体用于显示用户提供的词。

- 说明 Transact-SQL 命令使用的示例如下输出显示：

```
select * from publishers
```

- 计算机输出的示例如下：

```
0736    New Age Books            Boston       MA
0877    Binnet & Hardley         Washington   DC
1389    Algodata Infosystems    Berkeley      CA
```

大小写

本手册中，大多数示例是小写字体。然而，输入 Transact-SQL 关键词时可以忽略大小写。例如，SELECT、Select 和 select 是相同的。

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

表达式

Adaptive Server 语法语句使用以下类型的表达式：

表 2：在语法语句中使用的表达式类型

用法	定义
expression	可包括常量、文字、函数、列标识符、变量或参数
logical expression	返回 TRUE、FALSE 或 UNKNOWN 的表达式
constant expression	始终返回相同值的表达式，如“5+3”或“ABCDE”
float_expr	任何浮点表达式或隐式转换为浮点值的表达式
integer_expr	任何整数表达式或隐式地转换为整数值的表达式
numeric_expr	任何返回单个值的数字表达式
char_expr	任何返回单个字符型值的表达式
binary_expression	任何返回单个 *binary* 或 *varbinary* 值的表达式

如果需要帮助

每个已购买支持合同的进行安装的 Sybase 产品都有一位或多位指定人员得到授权可与 Sybase 技术支持部门联系。如果使用手册或联机帮助不能解决问题，可让指定人员与 Sybase 技术支持部门联系或与所在区域的 Sybase 子公司联系。

第 1 章

SQL 构件块

本章讨论如下内容：

主题	页
Adaptive Server 中的 SQL	1
命名约定	6
Adaptive Server 中的表达式	12
Transact-SQL 扩展	19
符合 ANSI 标准	25
Adaptive Server 登录帐号	28
如何通过 isql 实用程序使用 Transact-SQL	33
使用 pubs2 和 pubs3 样本数据库	35

Adaptive Server 中的 SQL

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

Transact-SQL 是 Sybase 的 SQL 扩展，它与 IBM SQL 和 SQL 的其它多数商业实施兼容。它提供重要的额外功能和函数，如摘要计算、存储过程（预定义的 SQL 语句）和错误处理。

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

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

查询、数据修改和命令

在 SQL 中，**查询**使用 select 命令请求数据。例如，下面查询请求列出居住在 California 的作者：

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

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

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

其它 SQL 命令（如删除表或添加用户）执行管理操作。例如：

```
drop table authors
```

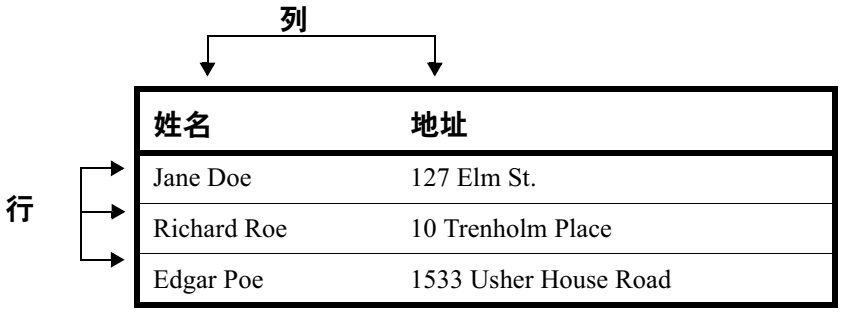
每个命令或 SQL 语句都以**关键字**开始，如 insert，它是要执行的基本操作的名称。很多 SQL 命令也具有一个或多个**关键字短语**或**子句**，它们用于定制命令以满足特定需要。运行查询时，Transact-SQL 显示结果。如果任何数据都不满足查询中所指定的标准，会显示有关此结果的消息。由于数据修改语句和管理语句不检索数据，因此，它们不显示结果。Transact-SQL 提供消息，告知用户是否已执行了数据修改或其它命令。

表、列和行

在关系数据库管理系统中，用户访问和修改存储在表中的数据。SQL 专用于数据库管理的关系模型。

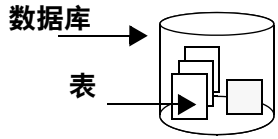
表中的每行或每个记录描述的是一段数据 — 人员、公司、销售量或其它事项。每列或每个字段描述的是数据的一个特性 — 人员姓名或地址、公司名称或总裁、产品销售量。

图 *1-1*：行和列



关系数据库由彼此相关的一组表构成。

图 *1-2*：数据库中的多个表



关系操作

关系系统中的基本查询操作是选择（也称为限制）、投影和连接。这些都被组合在 SQL select 命令中。

选择是表中行的子集。在 select 查询中，可指定限制条件。例如，可能只想查看居住在 California 的所有作者的行。

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

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

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

连接通过比较指定字段中的值将两个或多个表中的行链接。例如，假设有一个包含有关作者信息的表，其中包括列 au_id（作者标识号）和 au_lname（作者姓氏）。第二个表包含有关书籍书名的信息，其中包括列出书籍作者的 ID 号 (au_id) 的列。可连接 authors 表和 titles 表，测试各表的 au_id 列中值是否相等。只要存在匹配，就会创建一个新行（包含两个表中的列），并将其作为连接结果的一部分进行显示。连接通常与投影和选择组合使用，这样可仅显示选定匹配行的所选列。

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

编译对象

Adaptive Server 使用**编译对象**来包含有关各数据库的至关重要的信息，以帮助访问和处理数据。编译对象是需要 sysprocedures 表中条目的任何对象，包括：

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

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

1. 对源文本进行语法分析，捕捉任何语法错误，生成分析树。
2. 使分析树规范化，以创建规范树，它以二叉树格式表示用户语句。这就是编译对象。
3. 将编译对象存储在 sysprocedures 表中。
4. 将源文本存储在 syscomments 表中。

注意：如果要从 Adaptive Server 的 11.5 版或更早版本升级，并已从 syscomments 中删除了源文本，则在升级前必须恢复已删除的部分。要进行此操作，请执行所用平台的安装文档中的步骤。

保存源文本

在 Adaptive Server 的 11.5 版以及在其之前的 SQL Server® 版中，源文本保存在 syscomments 中，这样当用户执行 sp_helptext 就可以得到它。因为这样做的唯一目的就是保存源文本，而用户经常从 syscomments 表中删除此源文本以节省磁盘空间，并从该公共区域删除机密信息。然而，不应删除源文本，因为这样可能导致在将来升级 Adaptive Server 时出现问题。如果已从 syscomments 中删除了源文本，要使用本节中的过程恢复源文本。

恢复源文本

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

- 从备份装载源文本。
- 手工重新创建源文本。
- 重新安装用于创建编译对象的应用程序。

校验和加密源文本

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

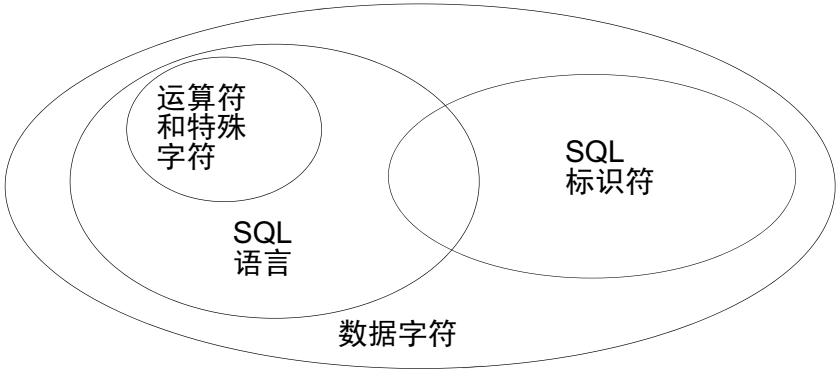
- sp_checkresource — 校验每个编译对象的 syscomments 中是否存在源文本。
- sp_hidetext — 加密 syscomments 表中编译对象的源文本，防止随意查看。
- sp_helptext — 如果 syscomments 中存在源文本，则显示它，否则通知您源文本丢失。
- dbcc checkcatalog — 通知丢失源文本。

命名约定

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

SQL 语句必须符合精确的句法和结构规则，并可包含运算符、常量、SQL 关键字、特殊字符和**标识符**。标识符是服务器中的对象，如数据库名或表名。对于 SQL 语句的某些部分，命名约定有所不同。运算符、常量、SQL 关键字和 Transact-SQL 扩展必须遵循比标识符更严格的命名约定，它们本身不能包含运算符和特殊字符。然而，可按多个许可的规则来命名服务器包含的数据。图 1-3 说明了这些关系。

图 *1-3*：约束 *SQL* 语句的命名约定



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

SQL 数据字符

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

SQL 语言字符

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

表 *1-1*：在 *SQL* 中使用的 *ASCII* 字符

;	（分号）	(	（开括号）	)	（关括号）
,	（逗号）	:	（冒号）	%	（百分号）
-	（减号）	?	（问号）	?	（单引号）
"	（双引号）	+	（加号）	_	（下划线）
*	（星号）	/	（斜杠）		（空格）
<	（小于号运算符）	>	（大于号运算符）	=	（等号运算符）
&	（与符号）	|	（竖线）	^	（脱字符）
[	（左方括号）	]	（右方括号）	\	（反斜杠）

@	（at 符号）	~	（否定符号）	!	（感叹号）
$	（美元符号）	#	（数字符号）	.	（句点）

标识符

数据库对象的命名约定适用于整个 Adaptive Server 软件和文档。不管是否使用多字节字符，标识符可长达 30 个字节。标识符的第一个字符必须是 Adaptive Server 使用的字符集定义中的字母字符。

也可使用 @ 符号或 _（下划线）。@ 符号是**局部变量**标识符的首字符。

临时表名必须以 #（井号）开头（如果它们在 tempdb 之外创建），或其前面有“tempdb..”。tempdb 之外的临时表的表名长度不应超过 13 字节（包括数字符号），因为 Adaptive Server 给它们一个内部的数值后缀。

第一个字符之后，标识符可以包含字母、数字或字符 $、#、@、_、￥（日元）或 £（英磅）。但不能在所命名的对象开头用两个 @@，如“@@myobject”。此命名约定专用于**全局变量**，全局变量是系统定义的变量，由 Adaptive Server 根据具体情况进行更新。

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

```
sp_helpsort
```

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

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

可使用函数 valid_name 确定 Adaptive Server 是否可接受已创建的标识符。例如：

select valid_name ("*string*")

string 是要检查的标识符。如果 *string* 作为标识符无效，则 Adaptive Server 返回 0（零）。如果 *string* 是有效标识符，Adaptive Server 则返回非 0 的数字。如果使用非法字符或如果 *string* 长度超过 30 字节，Adaptive Server 则返回 0。

使用多字节字符集

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

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

一些 8 位的欧洲字符也是如此。例如，字符“Œ,”OE 连字是 Macintosh 字符集的一部分（代码位置 0xCE），但 ISO 8859-1 (iso_1) 字符集中没有该字符。如果在从 Macintosh 字符集转换为 ISO 8859-1 字符集的数据中存在“Œ”则会导致一个转换错误。

如果对象标识符包含无法转换的字符，客户端就无法直接访问该对象。

分隔标识符

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

分隔标识符可以是保留字，可用非字母字符开头，并且也可以包括在其它情况下不允许的字符。分隔标识符不能超过 28 个字节。任何带引号的标识符的第一个字符不能为井号 (#)。（Adaptive Server 11.5 和所有更高版本都有此限制。）

在创建或引用分隔标识符之前，必须执行：

```
set quoted_identifier on
```

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

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

注意：当分隔标识符与系统过程一起使用时，它们不能与 bcp 一起使用，可能不被某些前端产品支持，并能产生意外的结果。

当 quoted_identifier 选项设置为“打开”时，使用单引号而不是双引号。如果用双引号分隔这些字符串，则会使 Adaptive Server 将其当作标识符。例如，要在 1one 的 col1 中插入一个字符串，应使用：

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

而不应使用：

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

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

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

唯一性约定和限定约定

在数据库中，数据库对象的名称不必唯一。但在一张表中，列名与索引名必须唯一，同一数据库中其它对象名必须对其所有者唯一。在 Adaptive Server 中，数据库名必须唯一。

如果试图用表中不唯一的名称创建列，或用在同一数据库中已使用的名称创建另一数据库对象（如表、视图或存储过程），Adaptive Server 就会输出错误消息。

可通过添加限定表或列的其它名称来进行唯一标识。可使用数据库名称、所有者名称、表名或视图名（对于列）来创建唯一 ID。它们中的每个限定符与下一个限定符之间用句号分隔：

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

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

这种命名语法也适用于其它数据库对象。可按类似方式引用任何对象：

```
pubs2.dbo.titleview
dbo.postalcoderule
```

如果 set 命令的 quoted_identifier 选项设置为“打开”，则可以用双引号引起限定对象名称的各部分。对于每个需要引号的限定符，都应单独使用一对引号。例如，使用：

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

而不能使用：

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

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

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

或

```
[owner.]object_name
```

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

```
database..table_name
```

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

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

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



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

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


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

标识远程服务器

可在远程 Adaptive Server 上执行存储过程。存储过程的结果显示在调用该过程的终端上。标识远程服务器和存储过程的语法为：

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

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

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

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

有关将 Adaptive Server 设置成可进行远程访问的信息，参见*系统管理指南*。远程服务器名称（前一示例中的 GATEWAY）必须与本地 Adaptive Server 的 *interfaces* 文件中的服务器名称相匹配。如果 *interfaces* 中的服务器名称使用大写字母，在 RPC 中也必须使用大写字母。

Adaptive Server 中的表达式

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

可在表达式中使用小括号将元素分组。把“*expression*”当作语法语句中的变量提供时，就视其为简单表达式。当只接受逻辑表达式时，才指定“逻辑表达式”。

算术表达式和字符表达式

算术表达式和字符表达式的一般模式为：

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

运算符优先级

运算符的优先级如下，其中 1 是最高级别而 6 是最低级别：

1　一元运算符（单个自变量）- + ~

2　* / %

3　二元运算符（两个自变量）+ - & | ^

4　not

5　and

6　or

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

算术运算符

Adaptive Server 使用以下算术运算符：

表 *1-2*：算术运算符

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

加法、减法、除法和乘法可用于精确数值、近似数值和货币类型列。

不能在 smallmoney、money、float 或 real 列上使用模运算符。模用于计算两个整数相除后的整数余数。例如 21 % 11 = 10，因为 21 除以 11 等于 1 余 10。

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

逐位运算符

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

表 1-3 和表 1-4 汇总了操作数 0 和 1 的结果。如果任一操作数为 NULL，逐位运算符都将返回 NULL：

表 *1-3*：逐位运算真值表

& (and)	1	0
1	1	0
0	0	0
| (or)	1	0
1	1	1
0	1	0
^ (exclusive or)	1	0
1	0	1
0	1	0
~ (not)		
1	0	
0	0	

下个示例使用两个 tinyint 自变量： A = 170（二进制形式为 10101010）和 B = 75（二进制形式为 01001011）。

表 *1-4*：逐位运算示例

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

字符串并置运算符

字符串运算符 + 可并置两个或多个字符或二进制的表达式。例如：

1
```
select Name = (au_lname + ", " + au_fname)
from authors
```

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

2
```
select "abc" + "" + "def"
```

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

当并置非字符、非二进制表达式时，使用 convert：

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

比较运算符

Adaptive Server 使用如下比较运算符：

表 *1-5*：比较运算符

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

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

在比较日期时，“<” 表示较早而 “>” 表示较晚。

给所有使用比较运算符的字符和 datetime 数据加上单引号或双引号：

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

非标准运算符

下列运算符是 Transact-SQL 扩展：

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

比较字符表达式

Adaptive Server 将字符常量表达式视为 varchar。将它们与非 varchar 变量或列数据进行比较时，将在比较中使用数据类型优先级规则（即，将较低优先级的数据类型转换为较高优先级的数据类型）。如果不支持隐式数据类型转换，则必须使用 convert 函数（参见第 361 页上的“支持的转换”）。

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

使用空字符串

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

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

空字符串从不会求值为 NULL。

在字符表达式中包括引号

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

```
’I don’’t understand.’
```

下面是一个包含内部双引号和单引号的示例。单引号不必转义，但双引号必须转义：

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

第二种方法是给引号加上相反类型的引号。也就是说，给包含双引号的条目加上单引号（或反之亦然）。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?"’
```

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

注意：如果 quoted_identifier 选项设置为“打开”，则不要对字符或日期数据使用双引号。必须使用单引号，否则 Adaptive Server 将数据视为标识符。

关系表达式和逻辑表达式

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

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

expression [not] in *expression*

[not] exists *expression*

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

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

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

expression is [not] null

not *logical_expression*

logical_expression {and | or} *logical_expression*

使用 *any*、 *all* 和 *in*

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

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

用 *and* 和 *or* 连接表达式

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

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

表 1-6 显示了逻辑运算的结果，其中包括那些含空值的表达式：

表 *1-6*：逻辑表达式真值表

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

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

Transact-SQL 扩展

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

将大部分 Transact-SQL 改进（也称为扩展）功能汇总如下。*参考手册*中有各命令的 Transact-SQL 扩展。

compute 子句

Transact-SQL compute 子句扩展与行集合函数（sum、max、min、avg 和 count）一起使用来计算摘要值。包含了compute子句的查询在显示结果中带有明细行和摘要行。此报告与多数“数据库管理系统”(DBMS) 用报告发生器生成的报告类似。compute 在结果中将摘要值显示为附加行，而不是作为新列。在第 3 章“使用集合、分组和排序”中对 compute 子句进行了介绍。

控制流语言

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

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

存储过程

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

编写自己存储过程的能力极大地提高了 SQL 数据库语言的功能、效率和灵活性。由于在运行存储过程后保存了执行计划，所以在以后运行存储过程时，要比独立语句快得多。

Adaptive Server 提供的存储过程称为**系统过程**，帮助 Adaptive Server 进行系统管理。第 14 章“使用存储过程”讨论了系统过程并说明如何创建存储过程。在*参考手册*中详细讨论了系统过程。

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

扩展存储过程

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

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

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

有一些由 Adaptive Server 提供的 ESP，称为**系统扩展存储过程**。xp_cmdshell 是其中之一，它可从 Adaptive Server 内部执行操作系统命令。在第 15 章“使用扩展存储过程”中对 ESP 进行了说明。*参考手册*包括有关系统 ESP 的详细信息。

ESP 由称为 XP Server_ 的 Open Server™ 应用程序实现，该应用程序与 Adaptive Server 运行在同一台计算机上。远程执行存储过程称为远程过程调用 (RPC)。Adaptive Server 和 XP Server 通过 RPC 进行通信。XP Server 自动与 Adaptive Server 一起安装。

触发器

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

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

触发器最深可嵌套 16 层，并可调用本地或远程存储过程或其它触发器。

缺省值和规则

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

缺省值是链接到特定列或数据类型的值，如果输入数据时未提供任何值，则由系统插入缺省值。规则是用户定义的、链接到特定列或数据类型的完整性约束，在输入数据时强制执行这些规则。在第 12 章“为数据定义缺省值和规则”中对规则和缺省值进行了讨论。

错误处理和 *set* 选项

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

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

下面各段列出了可用的 set 选项。有关详细信息，参见*参考手册*。

- parseonly、noexec、prefetch、showplan、rowcount、nocount 和 tablecount 控制着查询的执行方式。statistics 选项可在每次查询后显示性能统计信息。flushmessage 用于确定 Adaptive Server 何时将消息返回给用户。有关详细信息，参见*性能和调优指南*。
- arithabort 用于确定在发生算术溢出和算术截断错误时 Adaptive Server 是否中止查询。arithignore 用于确定在查询导致算术溢出时 Adaptive Server 是否输出警告消息。有关详细信息，参见第 27 页上的“算术错误”。
- offsets 和 procid 是在 DB-Library™ 中使用的，解释来自 Adaptive Server 的结果。
- datefirst、dateformat 和 language 影响日期函数、日期顺序和消息显示。
- char_convert 控制 Adaptive Server 与客户端间的字符集转换。

- textsize 控制着用 select 语句返回的 text 或 image 数据的大小。参见第 348 页上的“文本函数用于 text 和 image 数据”。
- cursor rows 和 close on endtran 影响 Adaptive Server 处理游标的方式。参见第 544 页上的“使用游标读取数据行”。
- identity_insert 可允许或禁止那些影响表中 IDENTITY 列的插入。参见第 230 页上的“由于插入、删除、identity grab size 和回退原因产生的间隔”。
- chained 和 transaction isolation level 控制 Adaptive Server 如何处理事务。参见第 578 页上的“选择事务模式和隔离级别”。
- self_recursion 允许 Adaptive Server 处理引起自行触发的触发器。参见第 523 页上的“触发器自递归”。
- ansinull、ansi_permissions 和 fipsflagger 控制 Adaptive Server 是否对非标准 SQL 的使用进行标记。string_rtruncation 控制在截断 char 或 nchar 字符串时 Adaptive Server 是否会发出例外错误。参见第 25 页上的“符合 ANSI 标准”。
- quoted_identifier 控制 Adaptive Server 是否将加双引号的字符串当作标识符。有关详细信息，参见第 9 页上的“分隔标识符”。
- role 控制授予用户的角色。有关角色的信息，参见*系统管理指南*。

SQL 的其它 Adaptive Server 扩展

Transact-SQL 的其它独特的或与众不同的功能包括：

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

- 建立在 Adaptive Server 所提供的数据类型基础上的、用户定义的数据类型。参见第 6 章“使用和创建数据类型”和第 12 章“为数据定义缺省值和规则”。
- 可以使用 insert 将表中数据插入到同一表中。参见第 8 章“添加、更改和删除数据”。
- 可以从一个表中抽取数据，然后用 update 命令将其放到另一表中。参见第 8 章“添加、更改和删除数据”。
- 可以在delete语句中使用连接来删除基于其它表中数据的数据。参见第 8 章“添加、更改和删除数据”。
- 用 truncate table 命令删除指定表中所有行并回收这些空间的快速方式。参见第 8 章“添加、更改和删除数据”。
- IDENTITY 列，它提供由系统生成的、能唯一标识表中各行的值。参见第 8 章“添加、更改和删除数据”。
- 通过视图进行更新和选择。与 SQL 的多数其它版本不同，Transact-SQL 对通过视图检索数据没有任何限制，对通过视图更新数据的限制也较少。参见第 9 章“视图：限制访问数据”。
- 有一些内部函数。参见第 10 章“使用查询中的内部函数”。
- create index 命令有一些选项，可用来调优由索引所确定的性能各个方面，并可控制重复键和重复行的处理。参见第 11 章“创建表的索引”。
- 控制当用户试图在唯一索引中输入重复键或在表中输入重复行时所发生的操作。参见第 11 章“创建表的索引”。
- 与 integer 和 bit 类型的列一起使用的逐位运算符。参见第44页上的“逐位运算符”和第 6 章“使用和创建数据类型”。
- 支持 text 与 image 数据类型。参见第 6 章“使用和创建数据类型”。
- 可访问 Sybase 和非 Sybase 数据库。使用“组件集成服务”，可在远程异构服务器中的表之间实现下列类型的操作：如同本地表一样访问远程表，执行连接，在表之间传送数据，维护参照完整性，使应用程序（如 PowerBuilder®）能够透明地访问异构数据以及使用本机远程服务器的功能。有关详细信息，参见*组件集成服务用户指南*。

符合 ANSI 标准

关系数据库管理系统的标准正在不断发展。这些标准已经或正在被 ISO 和几个国家的标准化组织所采用。SQL86 是这些标准中的第一个标准。该标准已被 SQL89 代替，而 SQL89 又被 SQL92 所代替，SQL92 是当前的标准。SQL92 定义了一致性的三个级别：初级、中间级和完整级别。在美国，国家标准技术研究所 (NIST) 定义了过渡级别，它介于初级和中间级之间。

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

缺省情况下，会为所有 Embedded SQL™ 预编译应用程序启用一致行为。需要与 SQL 标准行为一致的其它应用程序可使用符合初级 SQL92 的表 1-7 中的选项。有关设置这些选项的详细信息，参见*参考手册*中的 set。

表 *1-7*：为符合初级 *SQL92* 而设置的命令标志

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

下列各节介绍了标准行为与缺省的 Transact-SQL 行为之间的区别。

美国联邦信息处理标准 (*FIPS*) 标志程序

为使客户编写出必须符合 SQL92 标准的应用程序，Adaptive Server 提供了 set fipsflagger 选项。打开此选项后，包含有在初级 SQL92 中不允许使用的 Transact-SQL 扩展的所有命令都生成一条信息性消息。此选项不会禁止这些扩展。发出非 ANSI SQL 命令时，处理完成。

链式事务和隔离级别

Adaptive Server 提供符合 SQL 标准的“链式”事务行为作为一个选项。在链式模式中，所有的数据检索命令和修改命令（delete、insert、open、fetch、select 和 update）都隐式开始了一个**事务**。由于这种行为与很多 Transact-SQL 应用程序不兼容，所以 Transact-SQL 类型（或“非链式”）的事务保持缺省值。

可使用 set chained 选项启动链式事务模式。set transaction isolation level 选项用于控制事务隔离级别。有关详细信息，参见第 18 章“事务：维护数据一致性和恢复”。

标识符

为符合初级 SQL92，标识符不能：

- 以井号 (#) 开头
- 超过 18 个字符
- 包含小写字母

分隔标识符

Adaptive Server 支持表名、视图名和列名使用分隔标识符。分隔标识符是双引号中的对象名称。使用它们可避免对象名称的某些限制。

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

SQL 标准样式的注释

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

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

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

右截断字符串

string_rtruncation set 选项无提示地控制字符串截断以便与 SQL 标准兼容。设置此选项为“打开”，以禁止无提示地截断并强制执行 SQL 标准行为。

update 和 *delete* 语句所要求的权限

ansi_permissions set 选项用于确定 delete 和 update 语句所要求的权限。此选项设置为“打开”时，Adaptive Server 会对这些语句使用更严格的 SQL92 权限要求。由于此行为与很多现有应用程序不兼容，所以此选项的缺省设置为“关闭”。

算术错误

arithabort 和 arithignore set 选项按如下所述来符合 SQL92 标准：

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

  如果设置了 arithabort arith_overflow off，Adaptive Server 就会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 用于指定在精确数值类型导致标度损失后的行为。缺省设置为“打开”，它将中止导致错误的语句，但会继续处理事务或批处理中的其它语句。如果设置了 arithabort numeric_truncation off，Adaptive Server 就会截断查询结果并继续进行处理。为符合 SQL92 标准，输入 set arithabort numeric_truncation on。

- arithignore arith_overflow 确定是否在出现除零错误或精度损失后显示消息。缺省设置为“关闭”，在出现这些错误后显示警告消息。如果设置 arithignore arith_overflow on，则会取消发生这些错误后的警告消息。为符合 SQL92 标准，输入 set arithignore off。

同义关键字

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

表 *1-8*：与 *ANSI* 兼容的关键字同义词

当前语法	附加语法
commit tran, commit transaction rollback tran, rollback transaction	commit work rollback work
any	some
grant all	grant all privileges
revoke all	revoke all privileges
max (*expression*)	max ([all | distinct]) *expression*
min (*expression*)	min ([all | distinct]) *expression*
user_name() *built-in function*	user *keyword*

空值处理

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

Adaptive Server 登录帐号

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

- 登录名，它在该服务器上是唯一的。
- 口令。
- 缺省数据库（可选）。如果定义了缺省数据库，用户可在定义的数据库中启动每个 Adaptive Server 会话，而无需发出 use 命令。如果未定义缺省数据库，则要在 master 数据库中启动每个会话。
- 缺省语言（可选）。它指定显示提示和消息所使用的语言。如果未定义语言，则使用安装时设置的 Adaptive Server 缺省语言。

- 全名（可选）。这是用户的全名，对文档记录和身份识别非常有用。

组成员资格

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

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

```
sp_helpuser user_name
```

角色成员资格

在 Adaptive Server 中，系统安全员可定义和创建角色，作为在全服务器范围内一次向多个用户授予或撤消权限的简便方法。例如，文秘人员可能需要能够在几个数据库中的表上进行插入和选择，但他们可能不需要更新这些表。系统安全员可定义一个称为“clerical_user_role”的角色，并将此角色授予每个文秘人员。然后，数据库对象所有者可向“clerical_user_role”授予所需要的权限。

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

要查找有关指派给您的角色的详细信息，可使用：

- sp_displayroles — 找出指派给您的所有角色，无论它们是否处于激活状态。
- sp_activeroles — 找出指派给您的角色哪些处于激活状态。如果指定了 expand_down 参数，Adaptive Server 可显示了当前活动角色中所包含的任何角色。

语法为：

```
sp_displayroles user_name
sp_activeroles expand_down
```

有关角色的详细信息，参见*系统管理指南*。

获取有关 Adaptive Server 帐号的信息

可获取有关自己的 Adaptive Server 登录帐号的信息，方法是使用：

```
sp_displaylogin
```

Adaptive Server 返回以下信息：

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

更改口令

最好定期更改口令。系统安全员可将 Adaptive Server 配置成要求您按预置的一定间隔更改口令。如果您的服务器属于这种情况，当应该更改口令时，Adaptive Server 会通知您。

注意：如果使用了远程服务器，则必须在更改本地服务器口令*之前*更改所访问的所有远程服务器中的口令。有关详细信息，参见第 32 页上的“更改远程服务器上的口令”。

可随时使用 sp_password 更改口令：

```
sp_password old_passwd, new_passwd
```

创建新口令时：

- 其长度至少必须为六字节。
- 可以使用任何可打印字母、数字或符号。

- 口令的最大长度为 30 字节。如果口令超过 30 字节，Adaptive Server 仅使用前 30 个字符。

指定口令时，如果属于以下情况，需要加引号：

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

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

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

返回状态 0 表示口令已被更改。有关 sp_password 的详细信息，参见*参考手册*。

了解远程登录

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

要访问远程服务器：

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

在*系统管理指南*中对这些过程进行了讨论。

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

```
GATEWAY...sp_help
```

或者，要完全限定过程名称，应包括含有过程及其所有者名称的数据库名称：

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

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

- 相互鉴定 — 本地服务器通过检索远程服务器的认证并用安全机制对其进行校验的方式来鉴定远程服务器。通过这种服务，两台服务器的认证都得以确认。
- 通过加密实现消息保密性 — 消息在传送到远程服务器时加密，从远程服务器传来的结果同样如此。
- 消息完整性 — 检查服务器之间的消息是否被篡改。

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

有关基于网络的安全服务的详细信息，参见*系统管理指南*。

更改远程服务器上的口令

在更改本地服务器口令*之前*，必须更改所访问的所有远程服务器上的口令。如果先更改本地服务器的口令，当发出 RPC 在远程服务器上执行 sp_password 时，此命令就会失败，因为本地口令与远程口令不匹配。

在远程服务器上改变口令的语法为：

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

例如：

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

有关在本地服务器上更改口令的信息，参见第 30 页上的“更改口令”。

如何通过 *isql* 实用程序使用 Transact-SQL

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

必须先在 Adaptive Server 中设置帐号或登录名。要使用 isql，应在操作系统提示符后输入类似于下面的命令：

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

其中“horatio”是用户，“tallahassee”是口令，“haze”是要连接的 Adaptive Server 名称。-w 参数表示 isql 输出的宽度为 300 个字符。登录名和口令区分大小写。

如果启动 isql 时不使用登录名，Adaptive Server 则假定登录名与操作系统名称相同。有关为 isql 指定服务器登录名和其它参数的详细信息，参见所用平台的实用程序手册。

注意：切勿使用 isql 的 -P 选项来指定口令，因为其它用户可能会看到该口令。

启动 isql 后，可看到：

```
1>
```

此时可以开始发出 Transact-SQL 命令。

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

缺省数据库

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

可将缺省数据库更改为具有使用权限的任何数据库，或更改为允许访客的任何数据库。具有 Adaptive Server 登录名的任何用户都是访客。要更改缺省数据库，可使用 sp_modifylogin，在*参考手册*中对其进行了说明。

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

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

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

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

通过 *isql* 使用基于网络的安全服务

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

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

有关为使用基于网络的安全性而可以指定的选项的详细信息，参见*系统管理指南*。

退出 *isql*

可随时输入如下命令退出 isql：

```
quit
```

或

```
exit
```

其中的任何一个命令都会使您返回到操作系统提示符处，并不需要终结符“go”。

有关 isql 的详细信息，参见*实用程序指南*。

使用 *pubs2* 和 *pubs3* 样本数据库

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

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

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

样本数据库中有哪些内容?

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

下面是各表的简要说明：

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

- au_pix 包含使用 image 数据类型以二进制形式表示的作者照片。仅在 pubs2 中有 au_pix。
- blurbs 包含使用 text 数据类型对书籍进行的较长说明。

在附录 A“pubs2 数据库”中对 pubs2 数据库进行了说明，在附录 B“pubs3 数据库”中对 pubs3 进行了说明。

第 2 章

查询：从表中选择数据

select 命令使用一个叫做查询的过程从数据库表的行和列中检索存储的数据。查询有三个主要部分： select 子句，from 子句及 where 子句。

本章将对查询进行解释，并提供示例和讨论：

主题	页
什么是查询？	37
选择列：select 子句	40
用 distinct 消除重复查询结果	51
指定表：from 子句	52
选择行：where 子句	53
模式匹配	59

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

什么是查询？

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

有关 select 语句的一个简单示例：

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

select 子句指定要检索的列。from 子句指定要搜索的表。where 子句指定要查看表中的哪一行。例如，下列 select 语句将从 pubs2 数据库的 authors 表中查找居住在 Oakland 的作者的姓名。

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

此查询的结果将以分栏格式显示：

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

(5 rows affected)
```

select 语法

select 语法可能比前一示例更简单或更复杂。一个简单的 select 语句只包含 select 子句；一般情况下都包括 from 子句，在需要从表中检索数据的 select 语句中则必须要包括。所有其它子句（包括 where 子句）都是可选的。select 语句的完整语法包括下列短语和关键字：

select [all | distinct] *select_list*
  [into [[*database*.]*owner*.]*table_name*]
  [from [[*database*.]*owner*.]{*view_name*|*table_name*
    [(index {*index_name* | *table_name* }
      [parallel [*degree_of_parallelism*]]
      [prefetch *size*][lru|mru])]}
    [holdlock | noholdlock] [shared]
  [,[[*database*.]*owner*.]{*view_name*|*table_name*
    [(index {*index_name* | *table_name* }
      [parallel [*degree_of_parallelism*]]
      [prefetch *size*][lru|mru])]}
    [holdlock | noholdlock] [shared]]...]

  [where *search_conditions*]

  [group by [all] *aggregate_free_expression*
    [, *aggregate_free_expression*]...]
  [having *search_conditions*]

  [order by
  {[[[*database*.]*owner*.]{*table_name*.|*view_name*.}]
    column_name | *select_list*_number | *expression*} [asc | desc]
  [,{[[[*database*.]*owner*.]{*table_name*|*view_name*.}]
      column_name | *select_list_number* | *expression*}

```
        [asc | desc]]...]

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

[for {read only | update [of column_name_list]}]

[at isolation {read uncommitted | read committed | serializable}]

[for browse]
```

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

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

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

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

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

- group by、having、order by 和 compute 在第 3 章“使用集合、分组和排序”中进行说明。
- into 在第 7 章“创建数据库和表”中进行说明。
- at isolation 在第 18 章“事务：维护数据一致性和恢复”中进行说明。

holdlock、noholdlock 和 shared 关键字（用来在 Adaptive Server 中处理锁定）和 index 子句在*性能和调优指南*中进行说明。有关 for read only 和 for update 子句的信息，参见*参考手册*中的 declare cursor 命令。

注意：for browse 子句仅用于 DB-Library 应用程序中。有关详细信息，参见 *Open Client DB-Library/C 参考手册*。另请参见第 557 页上的“使用浏览模式而不用游标”。

选择列：select 子句

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

选择所有列：*select **

星号 (*) 从由 from 子句指定的所有表中选择所有列名。如果希望查看表中的所有列，用它可节省键入时间并减少错误的发生。* 将按 create table 的顺序检索数据。

在表中选择所有列的语法为：

```
select *
from table_list
```

下列语句将从 publishers 表中检索所有列并按 create table 的顺序显示它们。下列语句将检索所有行，因为它不包含 where 子句：

```
select *
from publishers
```

结果如下所示：

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

(3 rows affected)
```

如果在 select 关键字后按顺序列出表中的所有列名，将得到同样的结果：

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

在一个查询中也可多次使用“*”号：

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

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

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

因为 select * 将查找表中当前的所有列，对表结构的更改（如添加、删除或重命名列）都将自动修改 select * 的结果。分别列出各列可获得更精确的控制结果。

选择指定列

要选择表中的指定列，可使用：

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

用逗号分隔列名，例如：

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

重新安排列顺序

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

```
select pub_id, pub_name
from publishers
pub_id   pub_name
-----    ---------------
0736     New Age Books
0877     Binnet & Hardley
1389     Algodata Infosystems

(3 rows affected)
select pub_name, pub_id
from publishers
pub_name                      pub_id
---------------------         ------
New Age Books                 0736
Binnet & Hardley              0877
Algodata Infosystems          1389

(3 rows affected)
```

在查询结果中重命名列

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

column_heading = *column_name*

或：

column_name column_heading

或：

column_name as *column_heading*

这将为列提供一个替代名称。例如，在前一个查询中要将 pub_name 替换为“Publisher”，可键入下列任一语句：

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

这些语句的结果如下所示：

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

(3 rows affected)
```

表达式

select 语句也可包括一个或多个**表达式**来操纵检索的数据。

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

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

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

列标题中引用的字符串

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

```
select "Publisher's Name" = pub_name
from publishers
select pub_name "Publisher's Name"
from publishers
Publisher's Name
----------------
New Age Books
Binnet & Hardley
Algodata Infosystems

(3 rows affected)
```

另外，在引用的列标题中还可使用 Transact-SQL 保留字。例如下列查询使用保留字 sum 作为列标题，它也是有效的：

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

引用的列标题的长度不能大于 30 个字节。

注意：在 create table、alter table、select into、create view 语句中将列名用引号括起来之前，必须将 quoted_identifier 设置为打开。

查询结果中的字符串

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

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

下面是一个带字符串的查询：

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

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

(3 rows affected)
```

选择列表中计算的值

可用来自数值列的数据或来自选择列表的数值常量数据进行运算。

逐位运算符

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

表 2-1 显示的是逐位运算符。

表 *2-1*：逐位运算符

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

有关逐位运算符的详细信息，参见*参考手册*。

算术运算符

表 2-2 显示了可用的算术运算符。

表 *2-2*：算术运算符

运算符	运算
+	加法
-	减法
/	除法
*	乘法
%	模

除模运算符外，可在任何数值列（int、smallint、tinyint、numeric、decimal、float 或 money）使用算术运算符。模运算（可用在 money 数据类型之外的所有数值列）可求出两个整数相除后得到的整数余数。例如，21 % 11 = 10，因为 21 除以 11 等于 1，余数为 10。

使用日期函数可对 datetime 列执行某些算术运算操作。有关详细信息，参见第 10 章“使用查询中的内部函数”。可在选择列表中对列名和数值常量的任意组合使用所有这些运算符。例如，要查看 titles 表中计划销售增长为 100%的所有书籍情况，可输入：

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

结果显示如下：

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

(18 rows affected)
```

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

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

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

试着将字符串（如 "Current sales =" 和 "Projected sales are"）添加到 select 语句中。用来生成计算列的列不必出现在选择列表中。例如，在下列子查询中显示的 total_sales 列只是用来将其列值与来自 total_sales * 2 列的值进行比较。如果只查看计算列，键入：

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

如果没有相关常量，算术运算符也可直接用于指定列的数据值。例如：

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

(18 rows affected)
```

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

下面是一个有关连接的示例，此查询将销售的心理学书籍的数量（salesdetail 表中的 qty 列）与书的价格（titles 表中的 price 列）相乘。

```
select salesdetail.title_id, stor_id, qty * price
from titles, salesdetail
where titles.title_id = salesdetail.title_id
and titles.title_id = "PS2106"
title_id         stor_id
---------------- -----------  ------
PS2106           8042         210.00
PS2106           8042         350.00
PS2106           8042         217.00

(3 rows affected)
```

算术运算符优先级

如果表达式中包含多个算术运算符，则先计算乘法、除法和模运算，再计算加法和减法。如果表达式中的所有运算符属于同一级别，则运算顺序为从左向右。括号中的表达式的优先级高于任何其它运算符。

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

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

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

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

```
select title_id,(total_sales * price) - (advance /2)
from titles
title_id
--------     ----------
 BU1032       79,359.05
 BU1111       43,818.20
 BU2075       50,916.28
 BU7832       79,359.05
 MC2222       40,619.68
 MC3021       59,015.54
 MC3026            NULL
 PC1035      198,001.00
```

```
 PC8888       77,900.00
 PC9999            NULL
 PS1372        4,596.25
 PS2091        1,255.25
 PS2106       -2,223.00
 PS3333       80,399.28
 PS7777       24,654.64
 TC3218        4,356.25
 TC4203      178,397.20
 TC7777       57,384.05

(18 rows affected)
```

可以使用括号来改变执行顺序；括号里的计算会首先处理。如果嵌套使用括号，则最内层嵌套的优先级最高。例如，如果使用括号在除法之前强制执行减法运算，则上述示例的含义和结果都会改变：

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

(18 rows affected)
```

选择 *text* 和 *image* 值

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

```
set textsize 25
```

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

注意：如果选择 image 数据，返回值将包括字符“0x”，它表示数据是十六进制的。这两个字符将作为 *@@textsize* 的一部分来计算。

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

```
set textsize 0
```

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

使用 *readtext*

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

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

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

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

有关 readtext 命令的高级论述，参见第 348 页上的“文本函数用于 text 和 image 数据”。

选择列表摘要

select 列表可以包括 *（以 create table 顺序排列的所有列）、以任意顺序排列的列名列表、字符串、列标题及包括算术运算符的表达式。也可使用集合函数，这在第 3 章“使用集合、分组和排序”中进行了讨论。下面是一些 select 列表，它们是用示例数据库 pubs2 中的表进行试验的。

1
```
select titles.*
from titles
```

2
```
select Name = au_fname, Surname = au_lname
from authors
```

3
```
select Sales = total_sales * price,
ToAuthor = advance,
ToPublisher = (total_sales * price) - advance
from titles
```

4
```
select "Social security #", au_id
from authors
```

5
```
select this_year = advance, next_year = advance
   + advance/10, third_year = advance/2,
    "for book title #", title_id
from titles
```

6
```
select "Total income is",
Revenue = price * total_sales,
"for", Book# = title_id
from titles
```

用 *distinct* 消除重复查询结果

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

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

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

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

(25 rows affected)
```

其中包含一些重复列表。使用 distinct 可消除它们。

```
select distinct au_id
from titleauthor
au_id
```

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

(19 rows affected)
```

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

使用 order by 子句时，distinct 可以返回多个值。有关详细信息，参见第 98 页上的“与 select distinct 一起使用的 order by 及 group by”。

指定表：*from* 子句

在涉及表或视图数据的所有 select 语句中都需要有 from 子句。用它来列出包含在列中的所有表和视图（这些列包括在选择列表和 where 子句中）。如果 from 子句包括多个表或者视图，可用逗号分隔它们。

查询中允许的表和视图的最大数量是 16。其中包括在 from 子句中列出的表、由视图定义引用的基表、子查询中引用的所有表、用 into 关键字创建的表，以及作为参照完整性约束的一部分而引用的任何表。

from 语法如下所示：

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

表名的长度可在 1 到 30 个字节之间。可以使用字母、@、# 或作为第一个字符。随后的字符可以是数字、字母或 @、#、$、_、¥ 或 £。临时表的名称必须以“#”（井号）（如果是在 tempdb 外创建的）或“tempdb..”开始。如果在 tempdb 外创建了一个临时表，则此临时表的名称长度必须少于 13 个字节，因为 Adaptive Server 将把一个内部数字后缀附加到其后，以确保该名称是唯一的。有关详细信息，参见第 7 章“创建数据库和表”。

在 from 子句中始终允许表和视图的完整命名语法：

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

但是，仅当名称发生混淆时才有必要使用完整命名语法。

可以赋给表名一个相关名称，以节省键入时间。通过在表名之后提供相关名称，从而在 from 子句中分配相关名，如下所示：

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

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

选择行：*where* 子句

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

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

where 子句中的搜索条件或限定包括：

- 比较运算符（=、<、> 等等）

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

- 范围（between 和 not between）

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

- 列表（in、not in）

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

- 字符匹配（like 和 not like）

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

- 未知值（is null 和 is not null）

```
where advance is null
```

- 搜索条件的组合（and、or）

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

where 关键字还可引入：

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

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

有关搜索条件的详细信息，参见*参考手册*中的“where 子句”部分。

比较运算符

Transact-SQL 使用下列比较运算符：

表 *2-3*：比较运算符

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

运算符	含义
!>	不大于（Transact-SQL 扩展）
!<	不小于（Transact-SQL 扩展）

这些运算符用在下列语法中：

where expression comparison_operator expression

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

出于比较目的，尾随空格将被忽略。例如，“Dirk”与“Dirk ”是相同的。在比较日期时，< 表示较早而 > 表示较晚。用单引号或引号括起所有的 char、nchar、unichar、varchar、nvarchar、univarchar、text 和 datetime 型数据。有关输入 datetime 型数据的详细信息，参见第 8 章“添加、更改和删除数据”。

下面是一些使用比较运算符的 select 语句的示例：

```
select *
from titleauthor
where royaltyper < 50
select authors.au_lname, authors.au_fname
from authors
where au_lname > "McBadden"
select au_id, phone
from authors
where phone != "415 658-9932"
select title_id, newprice = price * $1.15
from  pubs2..titles
where advance > 5000
```

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

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

```
BU1032    business        5,000.00
BU1111    business        5,000.00
BU7832    business        5,000.00
PS2091    psychology      2,275.00
PS3333    psychology      2,000.00
PS7777    psychology      4,000.00

(6 rows affected)
```

范围（*between* 和 *not between*）

使用 between 关键字指定包括的范围。

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

```
select title_id, total_sales
from titles
where total_sales between 4095 and 12000
title_id  total_sales
------    -----------
BU1032           4095
BU7832           4095
PC1035           8780
PC8888           4095
TC7777           4095

(5 rows affected)
```

可用大于（>）和小于（<）操作符指定一个排它范围：

```
select title_id, total_sales
from titles
where total_sales > 4095 and total_sales < 12000
title_id  total_sales
------    -----------
PC1035           8780

(1 row affected)
```

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

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

```
--------     -----------
BU1111              3876
BU2075             18722
MC2222              2032
MC3021             22246
PS1372               375
PS2091              2045
PS2106               111
PS3333              4072
PS7777              3336
TC3218               375
TC4203             15096

(11 rows affected)
```

列表（*in* 和 *not in*）

in 关键字允许选择与值的任一列相匹配的值。表达式可以是常量或列名，值列表可以是一组常量或一个子查询。

例如，要列出居住在 California、Indiana 或 Maryland 的所有作者名称及其所在州名，可使用：

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

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

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

这是来自两个查询的结果：

```
au_lname         state
-----------      -----
White            CA
Green            CA
Carson           CA
OíLeary          CA
Straight         CA
Bennet           CA
Dull             CA
```

```
Gringlesby        CA
Locksley          CA
Yokomoto          CA
DeFrance          IN
Stringer          CA
MacFeather        CA
Karsen            CA
Panteley          MD
Hunter            CA
McBadden          CA

(17 rows affected)
```

in 关键字最重要的用途或许用在嵌套查询中（又称为**子查询**）。有关子查询的完整讨论，参见第 5 章“子查询：在其它查询中使用查询”下列示例给出了一种思想，讲述用嵌套查询和 in 关键字可实现哪些功能。

假设希望知道对合著的书籍收到的总版税低于 50% 的作者的名称。authors 表给出了作者名称，titleauthor 表给出了版税信息。通过使用 in 将两个表放到一起（但并不在同一个 from 子句中将其列出），即可抽取所需信息。下列查询：

- 搜索 titleauthor 表查找所有作者的 au_id，条件是这些作者从任一本书得到的版税少于 50%。
- 从 authors 表中选择 au_id 与 titleauthor 的查询结果相匹配的所有作者名称。结果显示有多个作者满足此分类条件。

```
select au_lname, au_fname
from authors
where au_id in
  (select au_id
   from titleauthor
   where royaltyper <50)
au_lname          au_fname
--------------    ------------
Green             Marjorie
OíLeary           Michael
Gringlesby        Burt
Yokomoto          Akiko
MacFeather        Stearns
Ringer            Anne

(6 rows affected)
```

not in 会将查找与列表中的项目不匹配的作者。下列查询将查找作者名称，他们至少从一本书中得到的版税不少于 50%。

```
select au_lname, au_fname
from authors
where au_id not in
  (select au_id
   from titleauthor
   where royaltyper <50)
au_lname          au_fname
---------------   ------------
White             Johnson
Carson            Cheryl
Straight          Dick
Smith             Meander
Bennet            Abraham
Dull              Ann
Locksley          Chastity
Greene            Morningstar
Blotchet-Halls    Reginald
del Castillo      Innes
DeFrance          Michel
Stringer          Dirk
Karsen            Livia
Panteley          Sylvia
Hunter            Sheryl
McBadden          Heather
Ringer            Albert
Smith             Gabriella

(18 rows affected)
```

模式匹配

可以在 where 子句中使用通配符，从而可以根据共同特征来查找未知字符分组数据。下列各部分利用 SQL 和 Transact-SQL 对模式匹配进行说明。有关模式匹配的更多信息，参见*参考手册*。

匹配字符串：*like*

like 关键字搜索与某一模式相匹配的字符串。like 可用于 char、varcha、nchar、nvarchar、unichar、univarchar binary、varbinary、text 及 datetime 型数据。

like 的语法为：

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

match_string 可包含表 2-4 中的符号：

表 *2-4*：匹配字符串的特殊符号

符号	含义
%	与 0 个或多个字符的任意字符串相匹配。
_	与单个字符相匹配。
[*分类符*]	用括号括起的范围或集合，如 [a–f] 或 [abcdef]。*分类符*可采用两种形式： • *rangespec1-rangespec2*： *rangespec1* 表示字符范围的开始。 – 是一个特殊字符，它表示一个范围。 *rangespec2* 表示字符范围的中止。 • *set*： 可由任意值的离散集合组成，可以按任何顺序排列，如 [a2bR]。范围 [a–f] 和集合 [abcdef] 及 [fcbdae] 将返回同样的一组值。 分类符区分大小写。
[^ *分类符*]	尖号 (^) 位于分类符之前，表示不包括。[^a–f] 指“不在 a–f 的范围内”； [^a2bR] 指“不是 a、2、b 或 R”。

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

- like "Mc%" 将搜索每个以“Mc”开始的名称（如 McBadden）。
- like "%inger" 将搜索每个以“inger”结尾的名称（如 Ringer、Stringer）。
- like "%en%" 将搜索每个包含“en”的名称（如 Bennet、Green、McBadden）。
- like "_heryl" 将搜索每个以“heryl”结尾且长度为六个字母的名称。

- like "[CK]ars[eo]n" 将搜索“Carsen”、“Karsen”、“Carson”和“Karson”（如 Carson）。
- like "[M-Z]inger" 将搜索所有以“inger”结尾且以 M 到 Z 之间任意单个字母开始的名称。
- like "M[^c]%" 将搜索所有以“M”开始且第二个字母不是“c”的名称。

下列查询将在 authors 表中查找区号为 415 的所有电话号码：

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

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

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

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

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

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

最好使用 like 来搜索 datetime 值，因为 datetime 条目可包含日期的各部分。例如，如果将值“9:20”和当前日期插入名为 arrival_time 的列，下列查询将无法找到该值，因为 Adaptive Server 将把此条目转换为“Jan 1 1900 9:20AM”：

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

但是，下列子句可找到值 9:20：

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

使用 *not like*

用于 like 的所有通配符都可用于 not like。例如，要在 authors 表中查找所有区号*不是* 415 的电话号码，可使用下列查询：

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

not like 和 ^ 可能会给出不同的结果

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

例如，下列查询将在以“sys”开头的数据库中查找系统表。

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

如果共有 32 个对象而 like 找到 13 个与此模式匹配的名称，则 not like 将找到 19 个与此模式不匹配的对象。

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

如下所示的模式可能不会产生同样的结果：

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

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

将通配符用作文字字符

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

方括号（Transact-SQL 扩展）

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

表 2-5：使用方括号搜索通配符

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

escape 子句（符合 SQL 标准的）

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

表 2-6：使用 *escape* 子句

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

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

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

指定多个通配符将引发一个 SQLSTATE 错误条件，Adaptive Server 将返回一条错误消息：

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

通配符和方括号的交互作用

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

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

尾随空格和 %

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

在列中使用通配符

可在列中或列名中使用通配符。如果希望在 pubs2 数据库中创建一个名为 special_discounts 的表，为特种买卖制订一个价格方案，可输入：

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

表中应该包含如下数据：

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

下列查询将在 where 子句中的 id_type 中使用通配符：

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

下面是查询结果：

```
title_id  discount    price
-------- ----------- -------------- --------------
BU1032           10          19.99          17.99
BU1111           10          11.95          10.76
BU2075           10           2.99           2.69
BU7832           10          19.99          17.99
PS1372           12          21.59          19.00
PS2091           12          10.95           9.64
PS2106           12           7.00           6.16
PS3333           12          19.99          17.59
PS7777           12           7.99           7.03
MC2222           15          19.99          16.99
MC3021           15           2.99           2.54
MC3026           15           NULL           NULL

(12 rows affected)
```

这将允许复杂的模式匹配而不必构造一系列的 or 子句。

字符串和引号

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

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

“未知”值： NULL

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

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

可使用下列两种方式在允许空值的列中输入 NULL：

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

如果在某个字符列中键入“NULL”时带引号，它将被认为是数据而非空值。

查询结果将显示单词 NULL。例如，titles 的 advance 列允许空值。通过检查该列中的数据，可判断是否一本书*尚未*根据协议支付预付款（在预付款列中 MC2222 行为零），或输入数据时预付款金额为 *not known*（在预付款列中 MC3026 行为 NULL）。

```
select title_id, type, advance
from titles
where pub_id = "0877"
title_id     type             advance
--------     ----------       ---------
MC2222       mod_cook              0.00
MC3021       mod_cook         15,000.00
MC3026       UNDECIDED             NULL
PS1372       psychology        7,000.00
TC3218       trad_cook         7,000.00
TC4203       trad_cook         4,000.00
TC7777       trad_cook         8,000.00

(7 rows affected)
```

测试某一列的值是否为空

在 where、if 和 while 子句中用 is null（如第 13 章“使用批处理和控制流语言”中所述）将列值与 NULL 进行比较，然后根据比较结果对其进行选择或执行某一特定操作。只有返回 TRUE 值的列才被选定或产生指定操作；返回值为 FALSE 或 UNKNOWN 的列则不然。

下列示例只选择 advance 少于 $5000 或为 NULL 的行：

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

Adaptive Server 将以不同的方式处理空值，取决于使用的**操作符**和要比较的值的类型。通常，空值的比较结果是未知的，因为无法确定 NULL 是等于（或不等于）给定值或另一个 NULL。如果 *expression* 是任意求值结果为 NULL 的列、变量或文字（或这些数据类型的组合），则下列情况将返回 TRUE：

- expression is null
- expression = null
- expression = *@x*，其中 *@x* 是包含 NULL 的变量或参数。此例外为用空的缺省参数来编写存储过程提供了方便。

- *expression* != *n*，其中 *n* 是不包含 NULL 的文字，*expression* 的求值结果为 NULL。

当表达式求值结果不为 NULL 时，这些表达式的否定形式将返回 TRUE：

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

如果使用关键字 like 和 not like 而非操作符 = 和 !=，则情况相反。下列比较将返回 TRUE：

- expression not like null

而下列比较将返回 FALSE：

- *expression* like null

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

空列值不与其它空列值相连接。在 where 子句中将空列值与其它空列值进行比较时，无论比较运算符是什么，始终返回 UNKNOWN，且结果中不包括行。例如，以下查询不返回 column1 在两表中都包含 NULL 的结果行（虽然它可能返回其它行）：

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

当与 NULL 连用时，下列操作符将返回结果：

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

如果为了与 SQL 兼容而将 set ansinull 置于“打开”状态，则当 = 和 != 操作符与 NULL 连用时，它们不返回结果。不论 set ansinull 选项是何值，下列操作符在与 NULL 连用时都不会返回值： <, <=, !<, >, >=, !>.

Adaptive Server 可确定某列值为 NULL。因此，下列表达式将被看作 true：

```
column1 =  NULL
```

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

```
where column1 > null
```

没有理由假定两个未知值是一样的。

此逻辑同样适用于在 where 子句中使用两个列名的情况（即连接两个表时）。一条类似于“where column1 = column2”的子句不会返回列中包含空值的行。

也可使用下列模式查找空值或非空值：

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

例如：

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

titles 表中的某些行包含不完整的数据。例如，一本名为 The Psychology of Computer Cooking(title_id = MC3026) 的书已被提议，并且已输入其标题、标题标识号和可能的出版社。但是，由于作者尚未签订合同且有一些细节问题悬而未决，此时在 price、advance、royalty、total_sales 和 notes 列中就会出现空值。因为空值不与任何内容匹配，所以某个查找所有标题标识号且预付款少于 $5000 的查询将不会找到 The Psychology of Computer Cooking。

```
select title_id, advance
from titles
where advance < $5000
title_id  advance
--------  ----------
MC2222         0.00
PS2091     2,275.00
PS3333     2,000.00
PS7777     4,000.00
TC4203     4,000.00

(5 rows affected)
```

下列查询查找预付款少于 $5000 *或*在 advance 列有空值的书籍：

```
select title_id, advance
from titles
where advance < $5000
  or advance is null
title_id  advance
--------  ----------
MC2222          0.00
MC3026          NULL
```

```
PC9999          NULL
PS2091      2,275.00
PS3333      2,000.00
PS7777      4,000.00
TC4203      4,000.00

(7 rows affected)
```

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

FALSE 和 UNKNOWN 之间的区别

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

用一个值替代 NULL

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

isnull(*expression*, *value*)

例如，使用下列语句从 titles 中选择所有行，并在 notes 中用“unknown”值显示所有空值。

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

计算结果为 NULL 的表达式

如果任何操作数为空，则带有算术运算符或逐位运算符的表达式求值结果为 NULL。例如，如果 column1 为 NULL，则下列表达式的计算结果也为 NULL：

```
1 + column1
```

并置字符串和 NULL

如果将字符串与 NULL 并置，则表达式的计算结果为字符串。例如：

```
select "abc" + NULL + "def"
-----
abcdef
```

系统生成的 NULL

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

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

但与系统生成的 NULL 进行同样的比较时将返回 UNKNOWN：

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

为了获得一致的行为，使用 set ansinull on。这时，系统生成的和用户提供的 NULL 都使比较结果返回 UNKNOWN。

带有逻辑运算符的连接条件

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

{where | having} [not]
column_name join_operator column_name

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

and 连接两个或两个以上的条件，并且仅当*全部*条件都为真时才返回结果。例如，下列查询仅查找作者名为 Ringer、姓为 Anne 的行。它不会选择作者姓名为 Albert Ringer 的行。

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

or 也连接两个或两个以上的条件，只要其中*任何一个*条件为真就返回结果。下列查询将在 au_fname 列中搜索包含 Anne 或 Ann 的行。

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

可指定多达 252 个 and 和 or 条件。

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

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

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

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

逻辑运算符的优先级

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

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

```
select title_id, type, advance
from titles
where type = "business" or type = "psychology"
  and advance > 5500
title_id  type        advance
--------  ----------  ----------
BU1032    business     5,000.00
BU1111    business     5,000.00
BU2075    business    10,125.00
BU7832    business     5,000.00
PS1372    psychology   7,000.00
PS2106    psychology   6,000.00

(6 rows affected)
```

要更改查询含义，可加入括号以强制先求 or 的值。下列查询将查找所有预付款大于 $5500 的商业和心理学方面的书籍：

```
select title_id, type, advance
from titles
where (type = "business" or type = "psychology")
  and advance > 5500
title_id  type         advance
--------  ----------   ---------
BU2075    business     10,125.00
PS1372    psychology    7,000.00
PS2106    psychology    6,000.00

(3 rows affected)
```

第 3 章

使用集合、分组和排序

本章介绍 sum、avg、count、count(*)、max 和 min 集合函数，在查询中可用这些函数对检索的数据进行归纳。本章还讨论如何使用 group by、having 以及 order by 子句对数据按类别和子群进行组织。还对两个 Transact-SQL 扩展（compute 子句和 union）运算符进行了讨论。

本章包括以下各节：

主题	页
集合函数的使用	73
将查询结果分组：group by 子句	78
选择数据组：having 子句	91
排序查询结果：order by 子句	96
汇总数据组：compute 子句	99
组合查询：union 运算符	107

如果 Adaptive Server 不区分大小写，可参见*参考手册*中的 group by and having Clauses 以及 compute Clause，其中的示例说明了大小写对这些子句返回的数据所产生的影响。

集合函数的使用

集合函数包括： sum, avg, count, min, max, 和 count(*)。使用**集合函数**可对数据进行计算和归纳。例如，要查看 pubs2 数据库 titles 表中的书销售量，可输入：

```
select sum(total_sales)
from titles
-------------
       97746

(1 row affected)
```

注意，示例中集合列没有列标题。

集合函数将要对其中的值进行运算的列名视为自变量。可将集合函数运用于表中所有的行、由 where 子句指定的表的子集、或表中一组或多组的行。在应用了集合函数的每组行处，Adaptive Server 生成一个值。

集合函数的语法为：

aggregate_function ([all|distinct] *expression*)

Expression 通常为列名。但它也可是常量、函数或任何由算术运算符或逐位运算符连接的列名、常量和函数的组合。在表达式中还可使用 case 表达式或子查询。

例如，如果价格加倍，使用以下语句可计算所有书的平均价：

```
select avg(price * 2)
from titles
-------------
        29.53

(1 row affected)
```

运用集合函数前，可使用带有 sum、avg 和 count 的可选关键字 distinct 消除重复的值。缺省设置为 all，它可对所有行执行运算。

集合函数的语法及其产生的结果如表 3-1 中所示：

表 *3-1*：集合函数的语法和结果

集合函数	结果
sum([all|distinct] *expression*)	表达式中（不同）值的总和
avg([all|distinct] *expression*)	表达式中（不同）值的平均值
count([all|distinct] *expression*)	表达式中（不同）非空值的数目
count(*)	选定的行数
max(*expression*)	表达式中的最大值
min(*expression*)	表达式中的最小值

可如前例所示在选择列表中使用集合函数，或在 having 子句使用集合函数。有关 having 子句的信息，参见第 91 页上的“选择数据组：having 子句”。

不能在 where 子句中使用集合函数，但多数带有选择列表中一个集合函数的 select 语句都包含 where 子句，它限制着集合所运用的行。在本节的前述示例中，每个集合函数都为整个表生成一个摘要值。

如果 select 语句包含 where 子句，但不是 group by 子句（参见第 78 页上的“将查询结果分组：group by 子句”），集合函数就为行子集生成一个值，称为**标量集合**。但是，select 语句也可在其选择列表中包含列（Transact-SQL 扩展），为结果表中每行重复这一个值。

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

```
select "advance and sales", avg(advance),
sum(total_sales)
from titles
where type = "business"
-----------------  -----------------  -----------
advance and sales          6,281.25       30788

(1 row affected)
```

集合函数和数据类型

可对任何类型的列使用集合函数，但有下列例外情况：

- sum 和 avg 只能用于数字型列 — int、smallint、tinyint、decimal、numeric、float 和 money。
- 不能对 bit 数据类型使用 min 和 max。
- 只有 count(*) 可用于 text 和 image 数据类型。

例如，可使用 min（最小值）查找字符类型列中的最小值（字母表最靠前的那个值）：

```
select min(au_lname)
from authors
--------------------------
Bennet

(1 row affected)
```

但是，不能计算文本列内容的平均值：

```
select avg(au_lname)
from authors
--------------------------
Bennet

(1 row affected)
```

```
Msg 257, Level 16, State 1:
Line 1:
Implicit conversion from datatype 'VARCHAR' to 'INT'
is not allowed. Use the CONVERT function to run
thisquery.
```

count 与 *count(*)*

count 查找的是表达式中非空值的数目，而 count(*) 查找的是表中总行数。下面语句得到的是书籍的总数：

```
select count(*)
from titles
------------------
                 18

(1 row affected)
```

count(*) 返回指定表中行数，但未消除重复。它将每行都计数在内，包括那些包含空值的行。

象其它集合函数一样，可将 count(*) 与选择列表中的其它集合以及 where 子句等组合使用：

```
select count(*), avg(price)
from titles
where advance > 1000
---------- ---------
        15     14.42

(1 row affected)
```

使用带有 *distinct* 的集合函数

可选关键字 distinct 只能和 sum、avg 以及 count 一起使用。使用 distinct 时，Adaptive Server 先消除重复值，然后进行计算。

如果使用 distinct，则不能在自变量中包含算术表达式。自变量只能使用列名。distinct 放在列名前的圆括号中。例如，要查找作者所在的不同城市数目，可输入：

```
select count(distinct city)
from authors
-------------
           16

(1 row affected)
```

要精确计算所有商业书籍的平均价格，可省略 distinct。以下语句返回所有商业书籍的平均价格：

```
select avg(price)
from titles
where type = "business"
-------------
        13.73

(1 row affected)
```

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

```
select avg(distinct price)
from titles
where type = "business"
-------------
        11.64

(1 row affected)
```

空值和集合函数

Adaptive Server 不将集合函数运行所在列中的任何空值计入函数中（count(*) 除外，它可包含空值）。如果将 ansinull 设置为 on（打开），则任何空值被忽略时 Adaptive Server 都返回一条错误消息。有关详细信息，参见*参考手册*中的 set 命令。

例如，titles 表中预付款的 count 与标题名的 count 不同，因为在 advance 列中存在空值：

```
select count(advance)
from titles
-------------
           16

(1 row affected)
select count(title)
```

```
from titles
-------------
           18

(1 row affected)
```

如果列中所有值都为空，count 返回 0。如果任何行都不满足 where 子句中指定的条件，count 返回 0。其它函数都返回 NULL。示例如下：

```
select count(distinct title)
from titles
where type = "poetry"
-------------
            0

(1 row affected)
select avg(advance)
from titles
where type = "poetry"
-------------
         NULL

 (1 row affected)
```

将查询结果分组：*group by* 子句

group by 子句将查询的输出分成组。可按一个或多个列名、或按表达式中使用数字数据类型所计算的列的结果进行分组。与集合一起使用时，group by 检索各子群中的计算，并可能返回多行。group by 子句中可使用的最大列数或表达式数是 16。

注意：不能对 text 或 image 数据类型的列进行 group by 操作。

尽管使用 group by 时可不带任何集合，但这样的结构限制了其功能，并可能产生混淆的结果。以下示例是按标题类型对结果进行分组：

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

```
------------      ---------
business           5,000.00
business           5,000.00
business          10,125.00
business           5,000.00
mod_cook               0.00
mod_cook          15,000.00
UNDECIDED              NULL
popular_comp       7,000.00
popular_comp       8,000.00
popular_comp           NULL
psychology         7,000.00
psychology         2,275.00
psychology         6,000.00
psychology         2,000.00
psychology         4,000.00
trad_cook          7,000.00
trad_cook          4,000.00
trad_cook          8,000.00

(18 rows affected)
```

对 advance 列使用集合后，查询返回各组的总和：

```
select type, sum(advance)
from titles
group by type
type
------------ ------------------------
UNDECIDED                       NULL
business                   25,125.00
mod_cook                   15,000.00
popular_comp               15,000.00
psychology                 21,275.00
trad_cook                  19,000.00

(6 rows affected)
```

使用了集合的 group by 子句中的摘要值称为**矢量集合**，它与标量集合相反，后者是在只返回一行时产生的（参见第 73 页上的“集合函数的使用”）。

group by 语法

此处重复了 select 语句的完整语法，以便在上下文中查看 group by 子句：

```
select [all | distinct] select_list
    [into [[database.]owner.]table_name]
    [from [[database.]owner.]{view_name|table_name
        [(index {index_name | table_name }
            [parallel [degree_of_parallelism]]
            [prefetch size ][lru|mru])]}
        [holdlock | noholdlock] [shared]
    [,[[database.]owner.]{view_name|table_name
        [(index {index_name | table_name }
            [parallel [degree_of_parallelism]]
            [prefetch size ][lru|mru])]}
        [holdlock | noholdlock] [shared]]... ]

    [where search_conditions]

    [group by [all] aggregate_free_expression
        [, aggregate_free_expression]... ]
    [having search_conditions]

    [order by
    {[[[database.]owner.]{table_name.|view_name.}]
        column_name | select_list_number | expression} [asc | desc]
    [,{[[[database.]owner.]{table_name|view_name.}]
column_name | select_list_number | expression}
            [asc | desc]]...]

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

    [for {read only | update [of column_name_list]}]

    [at isolation {read uncommitted | read committed | serializable}]

    [for browse]
```

切记，select 语句中子句的顺序非常重要。可忽略所有的可选子句，但使用这些子句时，它们必须按上述顺序出现。

group by 和 SQL 标准

group by 的 SQL 标准比 Sybase 标准更严格。SQL 标准要求：

- 选择列表中的列必须出现在 group by 表达式中，或者它们必须是集合函数的自变量。
- group by 表达式只能包括选择列表中的列名，但不是哪些只作为矢量集合自变量的列名。

Several Transact-SQL 扩展（在以后节中介绍）可放松这些限制，但结果集越复杂，理解起来可能越困难。如果对 fipsflagger 选项进行如下设置，就会出现一个警告消息，说明 Transact-SQL 扩展已被使用：

```
set fipsflagger on
```

有关 fipsflagger 选项的详细信息，参见*参考手册*中的 set 命令。

使用 *group by* 进行嵌套分组

可在 group by 子句中列出多个列以对组进行嵌套。使用 group by 建立集后，集合即被运用。以下语句先按出版者标识号然后按类型进行分组来得到平均价格和书籍销售额总和：

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

(8 rows affected)
```

可在组内嵌套多个组，用 group by 最多可指定 16 个列或表达式。

使用 *group by* 在查询中引用其它列

SQL 标准规定 group by 必须包含选择列表的项目。但是，Transact-SQL 允许在 group by 或选择列表中指定任何有效的列名，无论它们是否运用集合。

通过以下扩展，Sybase 解除了对包含 group by 查询的选择列表中可包括或忽略内容的限制。

- 选择列表中的列*不*仅限于分组列和使用矢量集合的列。
- group by 所指定的列也*不*仅限于选择列表中的非集合列。

矢量集合必须伴随有 group by 子句。SQL 标准要求选择列表中的非集合列必须与 group by 列匹配。但是，上述第一条允许在查询选择列表中指定附加的扩展列。

例如，SQL 的许多版本不允许在选择列表中包括扩展的 title_id 列，但在 Transact-SQL 中却是合法的：

```
select type, title_id, avg(price), avg(advance)
from titles
group by type
type           title_id
------------   --------    -----   -------
business       BU1032      13.73  6,281.25
business       BU1111      13.73  6,281.25
business       BU2075      13.73  6,281.25
business       BU7832      13.73  6,281.25
mod_cook       MC2222      11.49  7,500.00
mod_cook       MC3021      11.49  7,500.00
UNDECIDED      MC3026      NULL       NULL
popular_comp   PC1035      21.48  7,500.00
popular_comp   PC8888      21.48  7,500.00
popular_comp   PC9999      21.48  7,500.00
psychology     PS1372      13.50  4,255.00
psychology     PS2091      13.50  4,255.00
psychology     PS2106      13.50  4,255.00
psychology     PS3333      13.50  4,255.00
psychology     PS7777      13.50  4,255.00
trad_cook      TC3218      15.96  6,333.33
trad_cook      TC4203      15.96  6,333.33
trad_cook      TC7777      15.96  6,333.33

(18 rows affected)
```

上述示例仍然基于 type 列集合了 price 和 advance 列，但其结果也显示了各组中书的 title_id。

上述第二个扩展允许对查询的选择列表中未指定为列的列进行分组。这些列不出现在结果中，但矢量集合仍然计算其摘要值。例如：

```
select state, count(au_id)
from authors
group by state, city
state
-----  --------
CA            2
CA            1
CA            5
CA            5
CA            2
CA            1
CA            1
CA            1
CA            1
IN            1
KS            1
MD            1
MI            1
OR            1
TN            1
UT            2

(16 rows affected)
```

该例尽管不显示各组所包含的城市，却按 state 和 city 对矢量集合结果进行了分组。因此，结果可能易使人产生误解。

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

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

但结果却相差很大。通过不对 state 和 city 列使用 group by，查询记录每个城市的数量，但将该城市各行所对应的内容显示在 authors 中，而不是将其划分到每个城市对应的一个结果行中。

```
state
----- -----------
CA              1
CA              5
CA              2
CA              1
CA              5
KS              1
CA              2
CA              2
CA              1
CA              1
TN              1
OR              1
CA              1
MI              1
IN              1
CA              5
CA              5
CA              5
MD              1
CA              2
CA              1
UT              2
UT              2

(23 rows affected)
```

在包含 where 子句或连接的复杂查询中使用 Transact-SQL 扩展时，结果可能会更加难以理解。要避免这种因使用 group by 带来的容易令人混淆或令人误解的结果，Sybase 建议使用 fipsflagger 选项来识别使用 Transact-SQL 扩展的查询。有关详细信息，参见第 81 页上的“group by 和 SQL 标准”。

有关 group by 的 Transact-SQL 扩展及其如何工作的详细信息，参见*参考手册*。

表达式和 *group by*

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

```
select avg(total_sales), total_sales * price
from titles
group by total_sales * price
---------      -------------
     NULL               NULL
      111             777.00
      375           7,856.25
      375           8,096.25
     2045          22,392.75
     3336          26,654.64
     2032          40,619.68
     3876          46,318.20
    18722          55,978.78
     4095          61,384.05
    22246          66,515.54
     4072          81,399.28
     4095          81,859.05
     4095          81,900.00
    15096         180,397.20
     8780         201,501.00

(16 rows affected)
```

允许使用“total_sales * price”表达式。

尽管仍可在选择列表中使用一个列标题，也不能对列标题（也称为**别名**）进行 group by 操作。以下语句产生会错误消息：

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

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

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

使用 *group by* 嵌套集合

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

```
select avg(price)
from titles
group by type
---------------
 NULL
13.73
11.49
21.48
13.50
15.96

(6 rows affected)
```

在 max 函数中嵌套平均价格可产生一组书（按类型分组）的最高平均价：

```
select max(avg(price))
from titles
group by type
-------------
        21.48

(1 row affected)
```

根据定义，group by 子句适用于最内层的集合 — 本例中为 avg。

空值和 *group by*

如果分组列包含一个空值，则空值所在行在结果中成为自己的组。如果分组列中包含多个空值，则这些空值组成一个组。以下示例使用 group by 和 advance 列，其中包含了一些空值：

```
select advance, avg(price * 2)
from titles
group by advance
advance
------------------  -----------------
             NULL                NULL
             0.00               39.98
          2000.00               39.98
          2275.00               21.90
```

```
          4000.00              19.94
          5000.00              34.62
          6000.00              14.00
          7000.00              43.66
          8000.00              34.99
         10125.00               5.98
         15000.00               5.98

(11 rows affected)
```

如果使用 count(*column_name*) 集合函数，按包含空值的列进行分组时将返回分组行数为零的结果，因为 count(*column_name*) 不计算空值。多数情况下，应使用 count(*)。下例对 titles 表中包含空值的 price 列进行分组和计算，为了进行比较，还显示了 count(*)：

```
select price, count(price), count(*)
from titles
group by price
price
------------- ----- -----
        NULL     0     2
        2.99     2     2
        7.00     1     1
        7.99     1     1
       10.95     1     1
       11.95     2     2
       14.99     1     1
       19.99     4     4
       20.00     1     1
       20.95     1     1
       21.59     1     1
       22.95     1     1

(12 rows affected)
```

where 子句和 *group by*

可在带有 group by 的语句中使用 where 子句。分组之前，不满足 where 子句中条件的行将被排除。下面是一个示例：

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

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

(5 rows affected)
```

只有那些预付款超过 $5000 的行才会包含在用于生成查询结果的组中。

但是，Adaptive Server 处理选择列表中和 where 子句中额外列的方法似乎矛盾。例如：

```
select type, advance, avg(price)
from titles
where advance > 5000
group by type
type           advance
-------------  ---------  --------
business        5,000.00      2.99
business        5,000.00      2.99
business       10,125.00      2.99
business        5,000.00      2.99
mod_cook            0.00      2.99
mod_cook       15,000.00      2.99
popular_comp    7,000.00     21.48
popular_comp    8,000.00     21.48
popular_comp        NULL     21.48
psychology      7,000.00     14.30
psychology      2,275.00     14.30
psychology      6,000.00     14.30
psychology      2,000.00     14.30
psychology      4,000.00     14.30
trad_cook       7,000.00     17.97
trad_cook       4,000.00     17.97
trad_cook       8,000.00     17.97

(17 rows affected)
```

在查看 advance（扩展的）列的结果时，查询似乎忽略了 where 子句。Adaptive Server 仍然只使用那些满足 where 子句的行计算矢量集合，但也显示包含在选择列表中任何扩展列的所有行。要进一步限制这些行在结果中出现，必须使用 having 子句（在本章稍后部分介绍）。

group by 和 *all*

group by 子句中的关键字 all 是一个 Transact-SQL 增强功能。它只在包含 where 子句的 select 语句中使用才有意义。

如果使用 all，即使某些组不包含满足搜索条件的行，查询结果也将包含所有由 group by 子句生成的组。如果不使用 all，包含 group by 的 select 语句将不显示其中任何行都不满足条件的组。

下面是一个示例：

```
select type, avg(advance)
from titles
where advance > 1000 and advance < 10000
group by type
type
------------  ------------------------
business                     5,000.00
popular_comp                 7,500.00
psychology                   4,255.00
trad_cook                    6,333.33

(4 rows affected)
select type, avg(advance)
from titles
where advance > 1000 and advance < 10000
group by all type
type
------------  ------------------------
UNDECIDED                        NULL
business                     5,000.00
mod_cook                         NULL
popular_comp                 7,500.00
psychology                   4,255.00
trad_cook                    6,333.33

(6 rows affected)
```

第一个语句只生成那些预付款在 $1000 和 $10,000 之间的书籍的组。由于现代烹调书籍的预付款不在此范围之内，所以在结果中没有 mod_cook 类型的组。

尽管现代烹调组并不包含任何满足 where 子句中指定条件的行，但第二个语句会生成所有类型的组，包括现代烹调和“UNDECIDED”。对于没有满足条件的行的所有组，Adaptive Server 返回一个 NULL 结果。

使用不带 *group by* 的集合

根据定义，标量集合适用于表中所有的行，对于每个函数，在整个表上只生成一个值。允许包含带矢量集合的扩展列的 Transact-SQL 扩展也允许包含带标量集合的扩展列。例如，请看 publishers 表：

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

它包含三行。以下查询基于表的各行生成一个三行的标量集合：

```
select pub_id, count(pub_id)
from publishers
pub_id
---------- ---------
0736               3
0877               3
1389               3

(3 rows affected)
```

由于 Adaptive Server 将 publishers 作为一个组对待，所以标量集合适用于（单组）表。除标量集合外，结果还为在选择列表中包含的每列显示表的每行。

where 子句对标量集合和矢量集合的作用方式相同。where 子句限制包含在集合摘要值中的列，但对于在选择列表中指定的各扩展列，不影响在结果中出现的行。例如：

```
select pub_id, count(pub_id)
from publishers
where pub_id < "1000"
pub_id
-------------- -----------
0736                    2
0877                    2
1389                    2

(3 rows affected)
```

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

选择数据组：*having* 子句

使用 having 子句可显示或拒绝由 group by 子句所定义的行。having 子句为 group by 子句设置条件的方式和 where 为 select 子句设置条件的方式相同，只是 where 不能包含集合，而 having 却经常包含集合。以下示例是合法的：

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

但下面示例不合法：

```
select title_id
from titles
where avg(price) > $20
```

having 子句可引用出现在选择列表中的任何项。

以下语句是有集合函数的 having 子句的示例。它按类型对 titles 表中的行进行分组，但排除了只包含一本书的组：

```
select type
from titles
group by type
having count(*) > 1
type
----------------
business
mod_cook
popular_comp
psychology
trad_cook

(5 rows affected)
```

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

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

```
(2 rows affected)
```

在 having 子句中包括多个条件时，使用 and、or 或 not 将条件进行组合。例如，要按出版者对 titles 表分组，并且只包含那些预付款总额超过 $15,000、平均书价低于 $18、而且识别号 (pub_id) 大于 0800 的出版者，语句为：

```
select pub_id, sum(advance), avg(price)
from titles
group by pub_id
having sum(advance) > 15000
  and avg(price) < 18
  and pub_id > "0800"
pub_id
------  ---------------- ----------------
0877          41,000.00            15.41

(1 row affected)
```

having、*group by* 以及 *where* 子句交互作用的方式

在查询中包含 having、group by 和 where 子句时，各子句对行影响的顺序决定了最终的结果：

- where 子句排除那些不满足其搜索条件的行。
- 对于 group by 表达式中的每个唯一值，group by 子句都会将剩余的行集中到一组中。
- 在选择列表中指定的集合函数会计算每一组的摘要值。
- having 子句从最终结果中排除那些不满足搜索条件的行。

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

```
select stor_id, title_id, sum(qty)
from salesdetail
where title_id like "PS%"
group by stor_id, title_id
having sum(qty) > 200
stor_id  title_id
-------  --------  -----------
5023     PS1372            375
5023     PS2091          1,845
5023     PS3333          3,437
5023     PS7777          2,206
```

```
6380     PS7777            500
7067     PS3333            345
7067     PS7777            250

(7 rows affected)
```

按以下顺序进行查询：

1 where 子句只识别带有以“PS”开头（心理学书籍）的 title_id 的行，

2 group by 通过公用的 stor_id 和 title_id 聚集行，

3 sum 集合计算每组所销售的书籍的总数，并且

4 having 子句将书总数不超过 200 本的组从最终结果中排除。

所有以前的 having 示例都遵循 SQL 标准，这些标准规定 having 表达式中的列必须具有单个值，而且必须在选择列表或 group by 子句中。但是，having 的 Transact-SQL 扩展允许列或表达式不在选择列表和 group by 子句中。

以下示例确定每一标题类型的平均价格，即使 sum 集合没有出现在结果中，也将那些总销售额未超过 $10,000 的类型排除在外。

```
select type, avg(price)
from titles
group by type
having sum(total_sales) > 10000
type
------------  ----------
business           13.73
mod_cook           11.49
popular_comp       21.48
trad_cook          15.96

(4 rows affected)
```

扩展运行时就好像将列或表达式视为选择列表的一部分而不是显示结果的一部分。如果使用 having 包含了非集合列，但它不是选择列表或 group by 子句的一部分，则此查询生成的结果与本章前面叙述的“扩展”列扩展相似。例如：

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

```
business            13.73
business            13.73
business            13.73
mod_cook            11.49
popular_comp        21.48
popular_comp        21.48
psychology          13.50
trad_cook           15.96
trad_cook           15.96

(9 rows affected)
```

与扩展列不同，total_sales 列不出现在最终结果中，但各类型显示的行数取决于各标题的 total_sales。查询指示三个 business、一个 mod_cook、两个 popular_comp、一个 psychology 以及两个 trad_cook，这些标题的总销售额超过了 $4000。

正如前面所述，Adaptive Server 处理扩展列的方法就好像是查询忽略了最终结果中的 where 子句。要使 where 条件影响扩展列的结果，应在 having 子句中重复这些条件。例如：

```
select type, advance, avg(price)
from titles
where advance > 5000
group by type
having advance > 5000
type            advance
-------------   ---------  --------
business        10,125.00      2.99
mod_cook        15,000.00      2.99
popular_comp     7,000.00     21.48
popular_comp     8,000.00     21.48
psychology       7,000.00     14.30
psychology       6,000.00     14.30
trad_cook        7,000.00     17.97
trad_cook        8,000.00     17.97

(8 rows affected)
```

使用不带 *group by* 的 *having*

带有 having 子句的查询也应有 group by 子句。如果省略 group by，则由 where 子句所排除的所有行都作为一个组返回。

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

下例以下列方式使用 having 子句： 计算平均价格，从结果标题中排除预付款超过 $4,000 的标题，生成低于平均价格的结果：

```
select title_id, advance, price
from titles
where advance < 4000
having price > avg(price)
title_id        advance    price
-------------  ---------  --------
BU1032          5,000.00     19.99
BU7832          5,000.00     19.99
MC2222              0.00     19.99
PC1035          7,000.00     22.95
PC8888          8,000.00     20.00
PS1372          7,000.00     21.59
PS3333          2,000.00     19.99
TC3218          7,000.00     20.95

(8 rows affected)
```

也可使用带 Transact-SQL 扩展的 having 子句，它允许从包含有选择列表中某一集合的查询中省略 group by 子句。这些标量集合函数计算作为单个组的表而不是表中多组的值。

下例中，group by 子句被省略，使集合函数计算整个表的值。having 子句将非匹配行从结果组中排除。

```
select pub_id, count(pub_id)
from publishers
having pub_id < "1000"
pub_id
------ ----------------
0736                  3
0877                  3

(2 rows affected)
```

排序查询结果：*order by* 子句

order by 子句允许按一个或多个（最多为 31 个）列对查询结果排序。每个排序都可是升序 (asc) 或降序 (desc) 的。如果未进行指定，则缺省为 asc。以下查询按 pub_id 排序结果：

```
select pub_id, type, title_id
from titles
order by pub_id
pub_id  type            title_id
------  ------------    --------
0736    business        BU2075
0736    psychology      PS2091
0736    psychology      PS2106
0736    psychology      PS3333
0736    psychology      PS7777
0877    UNDECIDED       MC3026
0877    mod_cook        MC2222
0877    mod_cook        MC3021
0877    psychology      PS1372
0877    trad_cook       TC3218
0877    trad_cook       TC4203
0877    trad_cook       TC7777
1389    business        BU1032
1389    business        BU1111
1389    business        BU7832
1389    popular_comp    PC1035
1389    popular_comp    PC8888
1389    popular_comp    PC9999

(18 rows affected)
```

如果在 order by 子句中命名多个列，Adaptive Server 将对排序进行嵌套。以下语句首先通过 stor_id 降序排序，然后通过 payterms（升序排序，因为未指定 desc），最后通过 country（也是升序）对 stores 表中的行排序。Adaptive Server 首先排序任何组中的空值。

```
select stor_id, payterms, country
from stores
order by stor_id desc, payterms
stor_id payterms     country
------- ------------ ------------
8042    Net 30       USA
7896    Net 60       USA
7131    Net 60       USA
7067    Net 30       USA
7066    Net 30       USA
```

```
6380    Net 60       USA
5023    Net 60       USA

(7 rows affected)
```

可使用选择列表中列的 **position number** 代替列名。列名和选择列表号可混合使用。以下两个语句产生的结果与前一语句相同。

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

多数版本的 SQL 都要求 order by 项出现在选择列表中，但 Transact-SQL 没有此限制。即使该列未出现在选择列表中，也可按 title 对前述查询结果排序。

注意：不能在 text 或 image 列上使用 order by。

Adaptive Server 不允许子查询、集合、变量和常数表达式出现在 order by 列表中。

利用 order by，可以将空值排在其它所有值之前。

order by 子句对混合类型数据的影响取决于安装在 Adaptive Server 上的排序顺序。基本选择为二进制、字典顺序和不区分大小写。系统过程 sp_helpsort 显示服务器的排序顺序。有关排序顺序的详细信息，参见*参考手册*。

order by 和 *group by*

可使用 order by 子句以特定方式对 group by 的结果排序。

将 order by 子句放置于 group by 子句之后。例如，要计算各种类型书籍的平均价格，并按平均价格将结果排序，语句为：

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

```
mod_cook          11.49
psychology        13.50
business          13.73
trad_cook         15.96
popular_comp      21.48

(6 rows affected)
```

与 *select distinct* 一起使用的 *order by* 及 *group by*

如果 order by 或 group by 列不在 select 列表中，带有 order by 或 group by 的 select distinct 查询可返回重复值。例如：

```
select distinct pub_id
from titles
order by type
pub_id
------
0877
0736
1389
0877
1389
0736
0877
0877

(8 rows affected)
```

如果某个查询有 order by 或 group by 子句、而该子句包含了不在 select 列表中的列，Adaptive Server 就会将这些列作为隐藏列添加到正在处理的列中。order by 或 group by 子句中列出的列包含在不同行的测试中。为了符合 ANSI 标准，应在 select 列表中包含 order by 或 group by 列。例如：

```
select distinct pub_id, type
from titles
order by type
pub_id type
------ ------------
0877   UNDECIDED
0736   business
1389   business
0877   mod_cook
1389   popular_comp
```

```
0736   psychology
0877   psychology
0877   trad_cook

(8 rows affected)
```

汇总数据组：*compute* 子句

compute 子句是一个 Transact-SQL 扩展。将其与行集合一起使用可生成报告，显示分组摘要的小计。这些报告通常由报告发生器生成，称为控制中断报告，因为摘要值出现在 compute 子句中所指定的分组控制（“中断”）下的报告中。

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

compute 子句允许使用一个 select 语句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行集合（参见第 101 页上的“行集合和 compute”）。

compute 的常规语法为：

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

能与 compute 一起使用的行集合是 sum、avg、min、max 和 count。sum 和 avg 只能用于数字型列。与 order by 子句不同，不能使用选择列表的列定位号来代替列名。

注意：不能在 compute 子句中使用 text 或 image 列。

以下为两个查询及其结果。第一个使用 group by 和集合。第二个使用 compute 和行集合。注意其结果差别。

```
select type, sum(price), sum(advance)
from titles
group by type
type
------------   -------    ----------
UNDECIDED         NULL         NULL
business         54.92    25,125.00
mod_cook         22.98    15,000.00
popular_comp     42.95    15,000.00
```

```
psychology       67.52    21,275.00
trad_cook        47.89    19,000.00

(6 rows affected)
select type, price, advance
from titles
order by type
compute sum(price), sum(advance) by type
type         price                    advance
------------ ------------------------ --------
UNDECIDED    NULL                     NULL

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

type         price                advance
------------ -------------------- ----------
business     2.99                 10,125.00
business    11.95                  5,000.00
business    19.99                  5,000.00
business    19.99                  5,000.00

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

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

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

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

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

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

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

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

(24 rows affected)
```

每个摘要值均作为一行对待。

行集合和 *compute*

与 compute 一起使用的行集合列在表 3-2 中：

表 *3-2*：集合和 *compute* 语句一起使用的方式

行集合	结果
sum	表达式中值的总和
avg	表达式中的平均值
max	表达式中的最大值
min	表达式中的最小值
count	选定的行数

这些行集合和那些可与 group by 一起使用的集合是相同的，只是没有与 count(*) 等同的行集合函数。要查找由 group by 和 count(*) 生成的摘要信息，可使用不带 by 关键字的 compute 子句。

compute 子句的规则

- Adaptive Server 不允许将 distinct 关键字用于行集合。
- compute 子句中的列必须出现在选择列表中。
- 在同一语句中不能将 select into（参见第 7 章“创建数据库和表”）用作 compute 子句，因为包括 compute 的语句不能生成常规行。
- 如果将 compute 与 by 关键字一起使用，也必须同时使用 order by 子句。在 by 之后列出的列必须与在 order by 后面列出的列相同或是其子集，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，假设 order by 子句为：

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

  compute 子句可以是下列任何一种或全部形式：

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

  compute 子句不能是下列任何形式：

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

  必须在 order by 子句中使用列名或表达式；不能按列标题排序。
- 不带 by 的 compute 关键字可用于生成总和、总计数等。如果使用不带 by 的 compute，order by 就是可选的。在第 106 页上的“最终值： 不带 by 的 compute”中对不带 by 的 compute 关键字进行了讨论。

在 *compute* 后指定多列

在 by 关键字后列出多列会将组分成子群，并对每个分组级运用指定的行集合，从而影响到查询。例如，下面的查询是得到各出版者的心理学书籍的价格总和：

```
select type, pub_id, price
from titles
where type = "psychology"
order by type, pub_id, price
compute sum(price) by type, pub_id
type        pub_id  price
----------- ------- -------------
psychology    0736           7.00
psychology    0736           7.99
psychology    0736          10.95
psychology    0736          19.99

Compute Result:
---------------
          45.93
type        pub_id  price
----------- ------- -------------
psychology    0877          21.59

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

(7 rows affected)
```

使用多个 *compute* 子句

通过包含多个 compute 子句可在同一语句中使用不同的集合。下面的查询与上一个查询类似，只是按出版者对心理学书籍的价格总和进行了加运算：

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

```
psychology    0736           7.00
psychology    0736           7.99
psychology    0736          10.95
psychology    0736          19.99

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

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

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

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

(8 rows affected)
```

将一个集合用在多列上

一个 compute 子句可将同一集合应用于若干列。下面的查询是分别得到各种烹调书籍的价格和预付款总和：

```
select type, price, advance
from titles
where type like "%cook"
order by type
compute sum(price), sum(advance) by type
type      price            advance
--------- ---------------- ---------------
mod_cook              2.99       15,000.00
mod_cook             19.99            0.00

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

type      price            advance
--------- ---------------- ---------------
trad_cook            11.95        4,000.00
```

```
trad_cook            14.99        8,000.00
trad_cook            20.95        7,000.00

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

(7 rows affected)
```

切记，运用了集合的列也必须在选择列表中。

在同一 *compute* 子句中使用不同集合

可在同一 compute 子句中使用不同集合：

```
select type, pub_id, price
from titles
where type like "%cook"
order by type, pub_id
compute sum(price), max(pub_id) by type
type        pub_id  price
----------- ------- --------------
mod_cook    0877             2.99
mod_cook    0877            19.99

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

type        pub_id  price
----------- ------- --------------
trad_cook   0877            11.95
trad_cook   0877            14.99
trad_cook   0877            20.95

Compute Result:
--------------- ----
          47.89 0877

(7 rows affected)
```

最终值： 不带 *by* 的 *compute*

可使用不带 by 的 compute 关键字生成总和、总计数等。

下面语句是得到价格高于 $20 的各种书籍的预付款和价格总和：

```
select type, price, advance
from titles
where price > $20
compute sum(price), sum(advance)
type         price            advance
------------ ----------------  -------------
popular_comp            22.95      7,000.00
psychology              21.59      7,000.00
trad_cook               20.95      7,000.00

Compute Result:
--------------- ---------
          65.49 21,000.00

(4 rows affected)
```

可在同一查询中使用带和不带 by 的 compute。下面的查询是得到按类型计算的价格和预付款总和，然后计算所有类型书籍的价格和预付款总和。

```
select type, price, advance
from titles
where type like "%cook"
order by type
compute sum(price), sum(advance) by type
compute sum(price), sum(advance)
type         price                 advance
-----------  -----------------     ------------
mod_ cook                 2.99       15,000.00
mod_cook                 19.99            0.00

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

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

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

Compute Result:
--------------- ---------
          70.87 34,000.00

(8 rows affected)
```

组合查询：*union* 运算符

union 运算符将两个或多个查询的结果组合成一个结果集。union 的 Transact-SQL 扩展允许：

- 在 insert 语句的 select 子句中使用 union。
- 当 select 语句中有 union 时，在 select 语句的 order by 子句中指定新的列标题。

union 运算符的语法如下：

query1
  [union [all] *queryN*] ...
  [order by *clause]*
  [compute clause]

其中 *query1* 为：

```
select select_list
[into clause]
[from clause]
[where clause]
[group by clause]
[having clause]
```

queryN 为：

```
select select_list
[from clause]
[where clause]
[group by clause]
[having clause]
```

例如，假定有如下所示两个包含数据的表：

表 T1	
a	**b**
char(4)	**int**
abc	1
def	2
ghi	3

表 T2	
a	**b**
char(4)	**int**
ghi	3
jkl	4
mno	5

以下查询是在两个表之间创建 union：

```
create table T1 (a char(4), b int)
insert T1 values ("abc", 1)
insert T1 values ("def", 2)
insert T1 values ("ghi", 3)
create table T2 (a char(4), b int)
insert T2 values ("ghi", 3)
insert T2 values ("jkl", 4)
insert T2 values ("mno", 5)
select * from T1
union
select * from T2
a     b
----  ---------
abc           1
def           2
ghi           3
jkl           4
mno           5

(5 rows affected)
```

缺省情况下，union 运算符从结果集中删除重复的行。使用 all 选项可包含重复行。注意，结果集中的列名与 T1 中的列名相同。可在 Transact-SQL 语句中使用任意数量的 union 运算符。例如：

```
x union y union z
```

缺省情况下，Adaptive Server 从左向右计算含有 union 运算符的语句的值。可使用圆括号指定不同的计算顺序。

例如，以下是两个不同的表达式：

```
x union all (y union z)

(x union all y) union z
```

在第一个表达式中，*y* 与 *z* 并集中的重复值被删除了。而该集合与 *x* 并集中的重复值不被删除。在第二个表达式中，重复值包含在 *x* 与 *y* 的并集中，但它们在随后该集合与 *z* 的并集中被删除； all 不影响该语句的最终结果。

union 查询指南

使用 union 语句时：

- union 语句中的所有选择列表必须具有相同数量的表达式（如列名、算术表达式以及集合函数）。下列语句无效，因为第一个选择列表比第二个长：

```
create table stores_east
(stor_id char(4) not null,
stor_name varchar(40) null,
stor_address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null,
payterms varchar(12) null)


select stor_id, city, state from stores
union
select stor_id, city from stores_east


drop table stores_east
```

- 所有表中的相应列或各个查询中使用的列的任何子集，必须具有相同的数据类型，或两种数据类型之间必须可以进行隐式数据转换，否则应提供显式转换。例如，在 char 数据类型的列与 int 数据类型的列之间就不能进行 union 操作，除非提供了显式转换。但是，在 money 数据类型的列与 int 数据类型的列之间是可以进行 union 操作的。有关在 union 语句中比较数据类型的详细信息，参见*参考手册*中的 union 和“数据类型转换函数”。

- 必须以相同顺序将相应的列放于 union 语句的各个查询中，因为 union 将按照查询给定的顺序对各列逐一比较。例如，假设有以下这样的表：

表 T3		
a	**b**	**c**
int	**char(4)**	**char(4)**
1	abc	jkl
2	def	mno
3	ghi	pqr

表 T4	
a	**b**
char(4)	**int**
abc	1
def	2
ghi	3

以下查询：

```
select a, b from T3
union
select b, a from T4
```

生成：

```
a          b
---------  ---
        1  abc
        2  def
        3  ghi

(3 rows affected)
```

以下查询将导致出现错误消息，因为相应列的数据类型不兼容：

```
select a, b from T3
union
select a, b from T4



drop table T3
drop table T4
```

在 union 语句中组合使用不同（不兼容）的数据类型（如 float 和 int）时，Adaptive Server 将其转换为精度最高的数据类型。

- Adaptive Server 从 union 语句中的*第一个*单独查询来取由 union 生成的表的列名。因此，可在第一个查询中为结果集定义新的列标题。另外，如果要用新名称引用结果集中的一列（例如在 order by 语句中），应在第一个 select 语句中用这种方式进行引用。

  以下是正确的查询：

  ```
  select Cities = city from stores
  union
  select city from authors
  order by Cities
  ```

与其它 Transact-SQL 命令一起使用 *union*

与其它 Transact-SQL 命令一起使用 union 语句时：

- union语句中的第一个查询可以包含一个into子句，该子句可创建一个用于存放最终结果集的表。例如，以下语句创建一个名为 results 的表，它包含 publishers、stores 和 salesdetail 表的并集：

  ```
  use master
  sp_dboption pubs2, "select into", true
  use pubs2
  checkpoint
  select pub_id, pub_name, city into results
  from publishers
  union
  select stor_id, stor_name, city from stores
  union
  select stor_id, title_id, ord_num from
  salesdetail
  ```

  只能在第一个查询中使用 into 子句；如果它出现在其它位置，就会显示错误消息。

- 只能在 union 语句的末尾使用 order by 和 compute 子句，以定义最终结果的顺序或计算摘要值。不能在构成 union 语句的各个查询中使用它们。

- 只能在各个查询中使用 group by 和 having 子句；不能使用它们来影响最终结果集。

- 也可在 insert 语句中使用 union 运算符。例如：

```
create table tour (city varchar(20), state char(2))
```

```
insert into tour
    select city, state from stores
    union
    select city, state from authors
drop table tour
```

- 不能在 create view 语句中使用 union 运算符。
- 不能在 text 和 image 列上使用 union 运算符。
- 不能在涉及 union 运算符的语句中使用 for browse 子句。

第 4 章

连接：从若干表中检索数据

连接操作通过指定各个表的列对两个或多个表（或视图）进行比较，逐行比较这些列中的值，并链接具有匹配值的行。然后，它在新表中显示结果。连接中指定的表可在同一数据库或不同数据库中。

本章讨论如下内容：

主题	页
连接如何工作	114
如何使连接结构化	115
如何处理连接	119
等值连接和自然连接	120
使用其它条件的连接	121
未基于等同性的连接	122
自连接和相关名	123
不均等连接	124
连接两个以上的表	127
外部连接	129
Transact-SQL 外部连接	147
空值如何影响连接	151
确定要连接的表列	152

可声明多个连接作为子查询，它们也可包括两个或多个表。参见第 5 章“子查询：在其它查询中使用查询”。

如果启用了“组件集成服务”，则可执行通过远程服务器的连接。有关详细信息，参见*组件集成服务用户指南*。

连接如何工作

连接两个或多个表时，被比较的列必须具有相似值 — 即，值使用相同或相似数据类型。

有几种类型的连接，如，等值连接、自然连接和外部连接。最常见的是基于等同性的等值连接。下列连接查找居住在同一城市的作者姓名和出版者名称：

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city = publishers.city
au_fname      au_lname      pub_name
--------      --------      --------------------
Cheryl        Carson        Algodata Infosystems
Abraham       Bennet        Algodata Infosystems

(2 rows affected)
```

因为查询得到两个单独的表（publishers 和 authors）中包含的信息，所以需要一个连接来检索所需信息。下列语句使用 city 列作为链接，将 publishers 和 authors 表连接在一起。

```
where authors.city = publishers.city
```

连接语法

可将连接嵌入 select、update、insert、delete 或子查询中。其它连接限制条件和子句可遵循连接条件。连接使用如下语法：

start of select, update, insert, delete, or subquery
from {*table_list* | *view_list*}
where [not]
[*table_name*. | *view_name*.]*column_name join_operator*
[*table_name*. | *view_name*.]*column_name*
[{and | or} [not]
[*table_name*.|*view_name*.]*column_name join_operator*
[*table_name*.|*view_name*.]*column_name*]...

End of select, update, insert, delete, or subquery

连接和关系模型

连接操作是数据库管理的关系模型的特点。除其它任何功能外，连接使关系数据库管理系统区别于其它类型的数据库管理系统。

在通常称为网络和层次系统的结构化数据库管理系统中，数据值间的关系被预定义。建立数据库后，在数据中进行有关非预期关系的查询将变得很困难。

另一方面，在关系数据库管理系统中，数据值间的关系在数据库定义中保持未声明状态。处理数据时，它们成为显式状态 — 当**查询**数据库而不是创建它时。可提出想到的有关存储在数据库中数据的任何问题，而不管该数据库建立时的用途。

根据称为**规范化规则**的优良数据库设计规则，每个表应描述一类实体 — 人员、地点、事件或物品。这就是要比较有关两类或多类实体的信息时需要连接操作的原因。不同表中存储的数据间关系通过使其连接来发现。

此规则的必然结果是：连接操作在将新类型的数据添加到数据库方面获得无限的灵活性。始终可创建包含有关不同类型实体的数据的新表。如果新表某字段上具有的值与现有一个或多个表中某些字段上的值相似，可通过连接将其链接到其它表上。

如何使连接结构化

连接语句如同选择语句一样，都以关键字 select 开始。select 关键字后命名的列是要按其所需顺序包括在查询结果中的列。先前示例指定了包含 authors 表中作者姓名和 publishers 表中出版者名称的列：

```
select au_fname, au_lname, pub_name
from authors, publishers
```

不必通过表名限定列 au_fname、au_lname 和 pub_name，因为有关列属于哪个表很明确。但用于连接比较的 city 列确实需要限定，因为 authors 和 publishers 表中都有此名称的列：

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city = publishers.city
```

虽然每个 city 列都不在结果中输出，但 Adaptive Server 需要表名来执行比较。

要指定将查询涉及表的所有列都包括在结果中，和 select 一起使用星号 (*)。例如，要在前述连接查询中包括 authors 和 publishers 中的所有列，则语句为：

```
select *
from authors, publishers
where authors.city = publishers.city
au_id       au_lname au_fname phone        address
city       state postalcode contract pub_id pub_name
city       state
----------- -------- -------- ------------ ---------------------
---------- ----- ---------- -------- ------ --------------------
---------- -----
238-95-7766 Carson   Cheryl   415 548-7723 589 Darwin Ln.
Berkeley   CA    94705      1        1389   Algodata Infosystems
Berkeley   CA
409-56-7008 Bennet   Abraham  415 658-9932 223 Bateman St
Berkeley   CA    94705      1        1389   Algodata Infosystems
Berkeley   CA

(2 rows affected)
```

显示器显示每个表总计为 2 行 13 列。因为行宽的原因，每行占用显示器上多个水平线。只要使用“*”，结果中的列按该表的 create 语句中的顺序显示。

连接的选择列表和结果不需要包括被连接的两个表中的列。例如，要查找与一个出版者居住在同一城市的作者姓名，则此查询不需要包括 publishers 中的任何列：

```
select au_lname, au_fname
from authors, publishers
where authors.city = publishers.city
```

记住，如同任何 select 语句一样，选择列表中的列名和 from 子句中的表名必须由逗号分隔。

from 子句

使用 from 子句指定要连接的表和视图。这是表示 Adaptive Server 需要连接的子句。可按任何顺序列出表或视图。仅当使用 select * 指定选择列表时，表的顺序影响显示结果。

可在 from 子句中命名两个以上的表或视图。一个查询最多可引用 16 个表。此最大值包括：

- from 子句中列出的基表或视图
- 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
- from 子句中列出的视图所引用的基表
- 用 into 创建的表
- 作为查询结果创建的工作表

如果连接是更新或删除语句的一部分，查询仅可引用 15 个表。

下列示例连接 titles 和 publishers 表中的列，加倍在加利福尼亚出版的所有书的价格：

```
begin tran
update titles
  set price = price * 2
  from titles, publishers
  where titles.pub_id = publishers.pub_id
  and publishers.state = "CA"
rollback tran
```

有关连接包含多于两个表或视图的信息，参见第 127 页上的“连接两个以上的表”。

如第 2 章“查询：从表中选择数据”所述，表名或视图名可由所有者和数据库名称限定，并且为了方便可给出相关名。例如：

```
select au_lname, au_fname
from pubs2.blue.authors, pubs2.blue.publishers
where authors.city = publishers.city
```

可按与表完全相同的方式连接视图，并且只要使用表，就要使用视图。第 9 章“视图：限制访问数据”讨论视图；本章仅在其示例中使用表。

where 子句

使用 where 子句确定结果中包括的行。where 指定在 from 子句中命名的表和视图间的连接。如果有关列属于表或视图存在不确定性，确保限定列名。例如：

```
where authors.city = publishers.city
```

此 where 子句给出要连接的列名（如有必要由表名限定）和连接运算符 --- 通常为等号，有时为“大于号”或“小于号”。有关 where 子句语法的详细信息，参见第 2 章“查询：从表中选择数据”。

注意：如果省略连接的 where 子句，将生成意外结果。没有 where 子句，所讨论的任何连接查询目前将生成 69 行而不是 2 行。第 119 页上的“如何处理连接”将解释发生这种情况的原因。

除链接不同表中的列外，连接语句中的 where 子句还可包括其它条件。也就是说，可在同一个 SQL 语句中包括连接操作和 select 操作。有关示例，参见第 119 页上的“如何处理连接”。

连接运算符

基于等同性匹配列的连接称为**等值连接**。**等值连接**的更精确定义在第 120 页上的“等值连接和自然连接”中给出，并给出不是根据等同性连接的示例。

等值连接使用如下比较运算符：

表 *4-1*：连接运算符

运算符	含义
=	等于
>	大于
>=	大于或等于
<	小于
<=	小于或等于
!=	不等于
!>	小于或等于
!<	大于或等于

使用关系运算符的连接合称为 **theta 连接**。用于**外部连接**的另一组连接运算符也将在本章后面详细讨论。外部连接运算符是 Transact-SQL 扩展，如表 4-2 所示：

表 *4-2*：外部连接运算符

运算符	操作
*=	结果中包括第一个表中的所有行，而不仅是连接列匹配的行。
=*	结果中包括第二个表中的所有行，而不仅是连接列匹配的行。

连接列中的数据类型

被连接的列必须具有相同或兼容的数据类型。比较其数据类型无法隐式转换的列时，使用 convert 函数。虽然被连接的列经常具有相同的名称，但它们不需具有相同名称。

如果与用于连接的数据类型兼容，Adaptive Server 自动将其转换。例如，Adaptive Server 在任何数值类型列 — int、smallint、tinyint、decimal 或 float 和在任何字符类型和日期类型列 — char、varchar、unichar、univarchar、nchar、nvarchar 及 datetime 中转换。有关数据类型转换的详细信息，参见第 10 章“使用查询中的内部函数”和*参考手册*中的“数据类型转换函数”一节。

连接和 *text* 与 *image* 列

对于包含 text 或 image 值的列，无法使用连接。然而，可使用 where 子句比较两个表的文本列的长度。例如：

```
where datalength(textab_1.textcol) >
datalength(textab_2.textcol)
```

如何处理连接

知道如何处理连接有助于理解它们 — 并当声明连接不正确，有时生成意外结果时，可找出其中原因。本节用概念术语描述连接的处理。Adaptive Server 的实际过程更为复杂。

总体地说，处理连接的第一步是形成表的**笛卡儿乘积** — 每个表中行的所有可能组合。在两个表的笛卡儿乘积中，行数等于第一个表中的行数乘第二个表中的行数。

authors 表和 publishers 表的笛卡儿乘积为 69（23 个作者乘以 3 个出版者）。可使用任何查询查看笛卡儿乘积，该查询包括选择列表中多个表的列，from 子句中的多个表，并且没有 where 子句。例如，如果在先前示例使用的连接中，关闭 where 子句，则 Adaptive Server 将 23 个作者与 3 个出版者组合，并返回全部 69 行。

笛卡儿乘积不包含任何特别有用的信息。实际上，它会导致明显的误解，因为它暗示数据库中的每个作者都与该数据库中的每个出版者有关系 — 这根本不正确。

这是必须在连接中包括 where 子句的原因，该子句指定了要匹配的列和匹配基础。它也可能包括其它限制条件。Adaptive Server 形成笛卡儿乘积后，它通过使用 where 子句中的条件消除了不满足连接的行。

例如，先前示例中的 where 子句从结果中消除了作者与出版者不在同一城市的所有行：

```
where authors.city = publishers.city
```

等值连接和自然连接

基于等于 (=) 的连接称为**等值连接**。等值连接比较被连接列中的值是否相等，然后在结果中包括所连接表中的所有列。

下列最初的查询是等值连接的一个示例：

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

在该语句结果中，city 列出现两次。按照定义，等值连接的结果包含两个相同列。因为通常重复相同信息没有意义，可通过重述查询消除这些列之一。此结果称为**自然连接**。

在 city 列上产生 publishers 和 authors 的自然连接的查询是：

```
select publishers.pub_id, publishers.pub_name,
  publishers.state, authors.*
from publishers, authors
where publishers.city = authors.city
```

publishers.city 列未出现在结果中。

自然连接的另一个示例是：

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city = publishers.city
```

可使用多个连接运算符来连接两个以上的表或连接两个以上的列对。这些“连接表达式”通常使用 and 连接，虽然也可以使用 or。

下面是由 and 连接的两个示例。首先按书籍类型排序，列出有关书籍的信息（书籍类型、作者和书名）。多个作者的书籍具有多个列表，每一个与一个作者对应。

```
select type, au_lname, au_fname, title
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
order by type
```

其次查找居住在相同州和城市的作者姓名和出版者名称：

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city = publishers.city
and authors.state = publishers.state
```

使用其它条件的连接

连接查询的 where 子句可包括选择标准以及连接条件。例如，要检索预付款超过 $7500 的所有书籍的书名和出版者，其语句为：

```
select title, pub_name, advance
from titles, publishers
where titles.pub_id = publishers.pub_id
and advance > $7500
title                           pub_name              advance
-----------------------------   --------------------  ---------
You Can Combat Computer Stress!  New Age Books        10,125.00
The Gourmet Microwave            Binnet & Hardley     15,000.00
Secrets of Silicon Valley        Algodata Infosystems  8,000.00
Sushi, Anyone?                   Binnet & Hardley      8,000.00

(4 rows affected)
```

被连接的列（titles 和 publishers 中的 pub_id）不需要显示在选择列表中，因此不会出现在结果中。

可将任意多个选择标准包括在连接语句中。选择标准顺序和连接条件并不重要。

未基于等同性的连接

两列中连接值的条件不必相等。可使用任何其它比较运算符： 不等于 (!=)、大于 (>)、小于 (<)、大于或等于 (>=)、小于或等于 (<=)。Transact-SQL 也提供运算符 !> 和 !<，它们分别等价于 <= 和 >=。

下面大于连接的示例按字母顺序，查找居住在 New Age Books 所在的马萨诸塞州之后各州的 New Age 作者。

```
select pub_name, publishers.state,
   au_lname, au_fname, authors.state
from publishers, authors
where authors.state > publishers.state
and pub_name = "New Age Books"
pub_name         state   au_lname       au_fname         state
-------------    ------  -------------- -----------      -----
New Age Books    MA      Greene         Morningstar      TN
New Age Books    MA      Blotchet-Halls Reginald         OR
New Age Books    MA      del Castillo   Innes            MI
New Age Books    MA      Panteley       Sylvia           MD
New Age Books    MA      Ringer         Anne             UT
New Age Books    MA      Ringer         Albert           UT

(6 rows affected)
```

下列示例根据书籍的总销售额使用 >= 连接和 < 连接查找 roysched 表中正确的 royalty。

```
select t.title_id, t.total_sales, r.royalty
from titles t, roysched r
where t.title_id = r.title_id
and t.total_sales >= r.lorange
and t.total_sales < r.hirange
title_id       total_sales      royalty
--------       -----------      -------
BU1032               4095           10
BU1111               3876           10
BU2075               1872           24
BU7832               4095           10
MC2222               2032           12
MC3021              22246           24
```

```
PC1035                8780           16
PC8888                4095           10
PS1372                 375           10
PS2091                2045           12
PS2106                 111           10
PS3333                4072           10
PS7777                3336           10
TC3218                 375           10
TC4203               15096           14
TC7777                4095           10

(16 rows affected)
```

自连接和相关名

在一个表的同一列中比较值的连接称为**自连接**。要区分出现在表中的这两个角色，使用别名或相关名。

例如，可使用自连接找出在 California Oakland 居住的作者，他们住在相同的邮政编码区域。由于此查询包括 authors 表与其自身的连接，所以 authors 表以两种角色出现。要区分这些角色，可临时在 from 子句中任意给出 authors 表两种不同的相关名 — 如 au1 和 au2。这些相关名限定此查询其余部分的列名。自连接语句如下所示：

```
select au1.au_fname, au1.au_lname,
au2.au_fname, au2.au_lname
from authors au1, authors au2
where au1.city = "Oakland" and au2.city = "Oakland"
and au1.state = "CA" and au2.state = "CA"
and au1.postalcode = au2.postalcode
au_fname     au_lname    au_fname    au_lname
---------    ----------- --------    --------
Marjorie     Green       Marjorie    Green
Dick         Straight    Dick        Straight
Dick         Straight    Dirk        Stringer
Dick         Straight    Livia       Karsen
Dirk         Stringer    Dick        Straight
Dirk         Stringer    Dirk        Stringer
Dirk         Stringer    Livia       Karsen
Stearns      MacFeather  Stearns     MacFeather
Livia        Karsen      Dick        Straight
Livia        Karsen      Dirk        Stringer
```

```
Livia        Karsen      Livia       Karsen

(11 rows affected)
```

在 from 子句中列出别名的顺序与在选择列表中引用它们的顺序相同，如本例所示。根据查询，如果以不同顺序列出它们，则结果可能含糊不清。

要在结果中消除作者匹配自己并且除作者顺序相反外其它都相同的行，可将如下条件加入自连接查询中：

```
select au1.au_fname, au1.au_lname,
au2.au_fname, au2.au_lname
from authors au1, authors au2
where au1.city = "Oakland" and au2.city = "Oakland"
and au1.state = "CA" and au2.state = "CA"
and au1.postalcode = au2.postalcode
and au1.au_id < au2.au_id
au_fname    au_lname     au_fname    au_lname
---------   -----------  ---------   ---------
Dick        Straight     Dirk        Stringer
Dick        Straight     Livia       Karsen
Dirk        Stringer     Livia       Karsen

(3 rows affected)
```

现在它明显指出 Dick Straight、Dirk Stringer 和 Livia Karsen 都具有相同邮政编码。

不均等连接

不均等连接在限制由自连接返回的行方面非常有用。在如下示例中，不均等连接和自连接查找不同价格的两种或多种便宜书籍（少于 $15）的类别。

```
select distinct t1.type, t1.price
from titles t1, titles t2
where t1.price < $15
and t2.price < $15
and t1.type = t2.type
and t1.price != t2.price
type        price
----------  -----
business     2.99
```

```
business    11.95
psychology   7.00
psychology   7.99
psychology  10.95
trad_cook   11.95
trad_cook   14.99

(7 rows affected)
```

表达式“not *column_name* = *column_name*”等于“*column_name* != *column_name*”。

下列示例使用不均等连接，并结合使用自连接。它查找 titleauthor 表中的所有行，该表有两个或多个行具有相同 title_id 但不同的 au_id 编号（即具有多个作者的书籍）。

```
select distinct t1.au_id, t1.title_id
from titleauthor t1, titleauthor t2
where t1.title_id = t2.title_id
and t1.au_id != t2.au_id
order by t1.title_id
au_id           title_id
-----------     --------
213-46-8915     BU1032
409-56-7008     BU1032
267-41-2394     BU1111
724-80-9391     BU1111
722-51-5454     MC3021
899-46-2035     MC3021
427-17-2319     PC8888
846-92-7186     PC8888
724-80-9391     PS1372
756-30-7391     PS1372
899-46-2035     PS2091
998-72-3567     PS2091
267-41-2394     TC7777
472-27-2349     TC7777
672-71-3249     TC7777

(15 rows affected)
```

对于 titles 中的每本书，下列示例查找具有相同类型但不同价格的所有其它书籍：

```
select t1.type, t1.title_id, t1.price, t2.title_id,
t2.price
from titles t1, titles t2
where t1.type = t2.type
```

```
and t1.price != t2.price
```

解释不均等连接的结果时务必要小心。例如，认为使用不均等连接查找居住某城市（无出版者）的作者很方便：

```
select distinct au_lname, authors.city
from publishers, authors
where publishers.city != authors.city
```

然而，此查询却查找（无出版者）居住某城市的作者，这样就包括了全部作者。正确 SQL 语句是如下的子查询：

```
select distinct au_lname, authors.city
from publishers, authors
where authors.city not in
(select city from publishers
 where authors.city = publishers.city)
```

不均等连接和子查询

有时不均等连接查询限制条件不完全，因而由子查询所代替。例如，假定要列出居住某城市（无出版者）的作者姓名。为了表达清晰，限定此查询作者姓氏的首字母为“A”、“B”或“C”。不均等连接查询可以为：

```
select distinct au_lname, authors.city
from publishers, authors
where au_lname like "[ABC]%"
and publishers.city != authors.city
```

结果不是所提问题的答案：

```
au_lname             city
----------------     ------------
Bennet               Berkeley
Carson               Berkeley
Blotchet-Halls       Corvallis

(3 rows affected)
```

系统解释这种形式的 SQL 语句的含义：“查找居住某城市（*有些*出版者不在此城市）的作者姓名”。排除所有作者限定条件，包括居住在 Berkeley 的作者，Berkeley 是出版者 Algodata Infosystems 的所在地。

这种情况下，系统处理连接的方式（在评估其它条件之前首先查找每个符合条件的组合）导致此查询返回意外结果。必须使用子查询获得所需结果。子查询首先消除不合格的行，然后执行剩余限制条件。

下面是正确语句：

```
select distinct au_lname, city
from authors
where au_lname like "[ABC]%"
and city not in
(select city from publishers
where authors.city = publishers.city)
```

现在，下面得出想要的结果：

```
au_lname             city
-------------        ------------
Blotchet-Halls       Corvallis

(1 row affected)
```

有关子查询详细说明，参见第 5 章“子查询：在其它查询中使用查询”。

连接两个以上的表

pubs2 的 titleauthor 表提供一个很好的示例，有助于理解连接两个以上表的情况。要查找特定类型的所有书籍书名及其作者姓名，此查询为：

```
select au_lname, au_fname, title
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
and titles.type = "trad_cook"

au_lname       au_fname     title
-------------- -----------  -----------------------
Panteley       Sylvia       Onions, Leeks, and Garlic: Cooking
                            Secrets of the Mediterranean
Blotchet-Halls Reginald     Fifty Years in Buckingham Palace
                            Kitchens
O'Leary        Michael      Sushi, Anyone?
Gringlesby     Burt         Sushi, Anyone?
```

```
Yokomoto       Akiko        Sushi, Anyone?

      (5 rows affected)
```

注意，from 子句中的表之一 titleauthor 不向结果输出任何列。被连接的任何列 — au_id 和 title_id — 都未出现在结果中。虽然如此，此连接可能仅使用 titleauthor 作为中间表。

在相同语句中也可连接两个以上的列对。例如，下面一个查询，显示 title_id 和它的总销售额及它们所在范围，以及版税：

```
select titles.title_id, total_sales, lorange,
hirange, royalty
from titles, roysched
where titles.title_id = roysched.title_id
and total_sales >= lorange
and total_sales < hirange
title_id    total_sales   lorange hirange  royalty
--------    -----------   ------- -------  -------
 BU1032            4095         0    5000       10
 BU1111            3876         0    4000       10
 BU2075           18722     14001   50000       24
 BU7832            4095         0    5000       10
 MC2222            2032      2001    4000       12
 MC3021            2224     12001   50000       24
 PC1035            8780      4001   10000       16
 PC8888            4095         0    5000       10
 PS1372             375         0   10000       10
 PS2091            2045      1001    5000       12
 PS2106             111         0    2000       10
 PS3333            4072         0    5000       10
 PS7777            3336         0    5000       10
 TC3218             375         0    2000       10
 TC4203           15096      8001   16000       14
 TC7777            4095         0    5000       10

(16 rows affected)
```

在同一语句中有多个连接运算符，连接两个以上的表或连接两个以上的列对时，“连接表达式”几乎总是用 and 连接，如前面示例所示。但是，也可以使用 or 连接。

外部连接

包括所有行而不管是否有匹配行的连接称为**外部连接**。Adaptive Server 支持左外连接和右外连接。例如，下列查询通过 titles 和 titleauthor 表的 title_id 列连接这两个表。

```
select *
from titles, titleauthor
where titles.title_id *= titleauthor.title_id
```

Sybase 支持 Transact-SQL 和 ANSI 外部连接。Transact-SQL 外部连接（如先前示例所示）使用 *= 命令表示左外连接，使用 =* 命令表示右外连接。Transact-SQL 外部连接由 Sybase 创建，作为 Transact-SQL 语言的一部分。有关 Transact-SQL 外部连接的详细信息，参见第 147 页上的“Transact-SQL 外部连接”。

ANSI 外部连接使用关键字 left join 和 right join 分别表示左连接和右连接。Sybase 执行 ANSI 外部连接语法以完全符合 ANSI 标准。有关 ANSI 外部连接的详细信息，参见第 131 页上的“ANSI 内部和外部连接”。下面按 ANSI 外部连接重写先前示例：

```
select *
from titles left join titleauthor
on titles.title_id = titleauthor.title_id
```

内部表和外部表

术语**外部表**和**内部表**说明外部连接中表的位置：

- 在*左连接*中，**外部表**和**内部表**分别是左表和右表。外部表和内部表也分别称为行保留表和空值提供表。
- 在*右连接*中，外部表和内部表分别是右表和左表。

例如，在下面查询中，T1 是外部表，T2 是内部表：

```
T1 left join T2
T2 right join T1
```

或使用 Transact-SQL 语法：

```
T1 *= T2
T2 =* T1
```

外部连接限制

如果表是外部连接的内部成员，则无法参与外部连接子句*和*常规连接子句。下列查询失败，因为 salesdetail 表是外部连接和常规连接子句的一部分：

```
select distinct sales.stor_id, stor_name, title
from sales, stores, titles, salesdetail
where qty > 500
and salesdetail.title_id =* titles.title_id
and sales.stor_id = salesdetail.stor_id
and sales.stor_id = stores.stor_id
Msg 303, Level 16, State 1:
Server 'FUSSY', Line 1:
The table 'salesdetail' is an inner member of an
outer-join clause. This is not allowed if the table
also participates in a regular join clause.
```

如果要知道销售超过 500 册书籍的商店名称，必须使用另一个查询。如果提交具有外部连接的查询，且带有外部连接的内部表中列的限定条件，则可能不会得到预期结果。查询中的限定条件没有限制返回的行数，而是影响包含空值的行。对于不符合限定条件的行，空值显示在那些行的内部表列上。

用于外部连接的视图

如果定义带有外部连接的视图，然后使用外部连接的内部表中列的限定条件查询该视图，则可能不会得到预期结果。查询返回内部表的所有行。不满足限定条件的行在这些行的相应列中显示为 NULL 值。

下列规则确定了可通过连接视图对列进行更新的类型：

- 连接视图中不允许使用 delete 语句。
- 在用 with check option 创建的连接视图中不允许使用 insert 语句。
- 在用 with check option 创建的连接视图中，允许使用 update 语句。如果任何受影响的列出现在 where 子句中，或出现在包含来自多个表的列的表达式中，更新将会失败。
- 如果通过连接视图来插入或更新行，所有受影响的列都必须属于同一基表。

ANSI 内部和外部连接

下面是连接表的 ANSI 语法：

```
left_table [inner | left [outer] | right [outer]]
join right_table
on left_column_name  = right_column_name
```

左表和右表间的连接结果称为**连接表**。连接表在 from 子句中定义。例如：

```
select titles.title_id, title, ord_num, qty
from titles left join salesdetail
on titles.title_id = salesdetail.title_id
title_id title                ord_num                 qty
---------------------------- -------------------- -----
BU1032  The Busy Executive   AX-532-FED-452-2Z7   200
BU1032  The Busy Executive   NF-123-ADS-642-9G3   1000
. . .
TC7777  Sushi, Anyone?       ZD-123-DFG-752-9G8   1500
TC7777  Sushi, Anyone?       XS-135-DER-432-8J2   1090
(118 rows affected)
```

ANSI 连接语法允许编写如下任一项目：

- **内部连接**，连接表仅包括满足 on 子句条件的内部表和外部表的行。有关信息，参见第 133 页上的“ANSI 内部连接”。对于不满足 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任何空值行。
- **外部连接**，连接表包括外部表中的所有行，不论外部表是否满足 on 子句条件。如果某行不满足 on 子句条件，则内部表的值将作为空值存储在连接表中。ANSI 外部连接中的 where 子句将限制查询结果中的行。有关详细信息，参见第 136 页上的“ANSI 外部连接”。

注意：也可使用 ANSI 语法连接视图。

Sybase ANSI 连接语法不支持 using 子句。

ANSI 连接的相关名和列引用规则

下列是专用于 ANSI 连接的相关名和列引用规则。有关相关名的详细信息，参见第 123 页上的“自连接和相关名”。

- 如果表或视图使用引用列或视图的相关名，它必须始终使用同一相关名，而不能使用该表名或视图名。即，不能在查询中用相关名命名表，以后使用其表名。下列示例正确使用相关名 t 指定已指定其 pub_id 列的表：

  ```
  select title, t.pub_id, pub_name
  from titles t left join publishers p
  on t.pub_id = p.pub_id
  ```

  然而，下列示例未使用查询 on 子句中 titles 表 (t.pub_id) 的相关名，而不正确地使用表名，因而后来生成错误消息：

  ```
  select title, t.pub_id, pub_name
  from titles t left join publishers p
  on titles.pub_id = p.pub_id
  Msg 107, Level 15, State 1:
  Server 'server_name', Line 1:
  The column prefix 't' does not match with a table
  name or alias name used in the query. Either the
  table is not specified in the FROM clause or it
  has a correlation name which must be used
  instead.
  ```

- on 子句中指定的限制可引用：
  - 在连接表的引用中指定的列
  - 在 ANSI 连接（如嵌套连接）中包含的连接表中指定的列
  - 在外部查询块中指定的表的子查询中的相关名
- 在 on 子句中指定的条件*不能*引用*包含*另一 ANSI 连接的 ANSI 连接中引入的列（通常当第二连接与第一连接进行连接生成连接表时）。

  下面是非法列引用生成错误的一个示例：

```
select *
from titles left join titleauthor
on titles.title_id=roysched.title_id    /*join #1*/
left join roysched
on titleauthor.title_id=roysched.title_id    /*join #2*/
where titles.title_id != "PS7777"
```

第一个左连接无法引用 roysched.title_id 列，因为此列直到第二连接后才被引入。可正确重写此查询为：

```
select *
from titles
left join (titleauthor
left join roysched
on titleauthor.title_id = roysched.title_id) /*join #1*/
on titles.title_id = roysched.title_id      /*join #2*/
where titles.title_id != "PS7777"
```

另一个示例为：

```
select title, price, titleauthor.au_id, titleauthor.title_id, pub_name,
publishers.city
from roysched, titles
left join titleauthor
on roysched.title_id=titleauthor.title_id
left join authors
on titleauthor.au_id=roysched.au_id, publishers
```

在此查询中，roysched 表或 publishers 表都不是任意左连接的一部分。因此，任何左连接都无法引用任一 roysched 或 publishers 表作为其 on 子句条件的一部分。

ANSI 内部连接

生成仅包括满足限制条件连接表行的结果集的连接称为**内部连接**。连接表中不包括不满足连接限制条件的行。如果要求连接表包括一个表的所有行，而不管它们是否满足限制条件，使用外部连接。有关详细信息，参见第 136 页上的“ANSI 外部连接”。

Adaptive Server 支持使用 Transact-SQL 内部连接和 ANSI 内部连接。使用 Transact-SQL 内部连接的查询用逗号分隔被连接的表，并在 where 子句中列出连接比较和限制条件。例如：

```
select title_id, title, price
from titleauthor, titles
where titleauthor.title_id = titles.title_id
and price > $15
```

有关编写 Transact-SQL 内部连接的详细信息，参见第 115 页上的“如何使连接结构化”。

ANSI 标准 inner joins 语法为：

select *select_list*
　from *table1* inner join *table2*
　on *join_condition*

例如，下面示例使用 inner join，等于上述 Transact SQL 连接：

```
select au_id, titles.title_id, title, price
from titleauthor inner join titles
on titleauthor.title_id = titles.title_id
and price > 15
au_id         title_id  title                       price
----------    --------  ------------------------    -----
213-46-8915   BU1032    The Busy Executiveís Datab  19.99
409-56-7008   BU1032    The Busy Executiveís Datab  19.99
. . .
172-32-1176   PS3333    Prolonged Data Deprivation  19.99
807-91-6654   TC3218    Onions, Leeks, and Garlic:  20.95
(11 rows affected)
```

编写连接的两个方法 ANSI 或 Transact-SQL 等同。例如，下列查询生成的结果集间没有区别：

```
select title_id, pub_name
from titles, publishers
where titles.pub_id = publishers.pub_id
```

和

```
select title_id, pub_name
from titles left join publishers
on titles.pub_id = publishers.pub_id
```

内部连接可以是 update 或 delete 语句的一部分。例如，下列是对于在 California 出版的所有书目价格乘以 1.25 的查询：

```
begin tran
update titles
  set price = price * 1.25
  from titles inner join publishers
  on titles.pub_id = publishers.pub_id
  and publishers.state = "CA"
```

内部连接的连接表

ANSI 连接指定在查询中要连接的表或视图。ANSI 连接中指定的表引用构成连接表。例如，下列查询的连接表包括 title、price、advance 和 royaltyper 列：

```
select title, price, advance, royaltyper
from titles inner join titleauthor
on titles.title_id = titleauthor.title_id
title          price     advance      royaltyper
-----------    -------   ----------   ----------
The Busy...    19.99     5,000.00     40
The Busy...    19.99     5,000.00     60
```

```
. . .
Sushi, A...     14.99     8,000.00     30
Sushi, A...     14.99     8,000.00     40
(25 rows affected)
```

如果连接表用作 ANSI 内部连接中的表引用，则它成为**嵌套**内部连接。ANSI 嵌套内部连接遵循与 ANSI 外部连接相同的规则。

查询在联合的每一侧可引用最多 50 个用户表（或 14 个工作表），包括：

- from 子句中列出的基表或视图
- 对同一表的每个相关引用（自连接）
- 在子查询中引用的表
- 视图或嵌套视图引用的基表
- 用 into 创建的表

ANSI 内部连接的 *on* 子句

ANSI 内部连接的 on 子句指定表或视图连接时所使用的条件。虽然可在表的任何列上连接，但如果这些列编入索引可能会更好地提高性能。通常必须使用限定符（表名或相关名）唯一地标识列及其所属表。例如：

```
from titles t left join titleauthor ta
on t.title_id = ta.title_id
```

此 on 子句消除了未匹配 title_id 的两个表中的行。有关相关名的详细信息，参见第 123 页上的“自连接和相关名”。

此 on 子句通常与 ANSI 连接表进行比较，如下列查询的第三行和第四行所示：

```
select title, price, pub_name
from titles inner join publishers
on titles.pub_id = publishers.pub_id
and total_sales > 300
```

在此 on 子句中指定的连接限制条件删除销售额不大于 300 的连接表中的所有行。on 子句也可指定搜索自变量，如此查询的第四行所示。

无论限制条件放置在 on 子句或 where 子句中，ANSI 内部连接对结果的限制集都类似（除非它们嵌套到外部连接中）。即，下列查询生成相同结果集：

```
select stor_name, stor_address, ord_num, qty
```

```
from salesdetail inner join stores
on salesdetail.stor_id = stores.stor_id
where qty > 3000
```

和

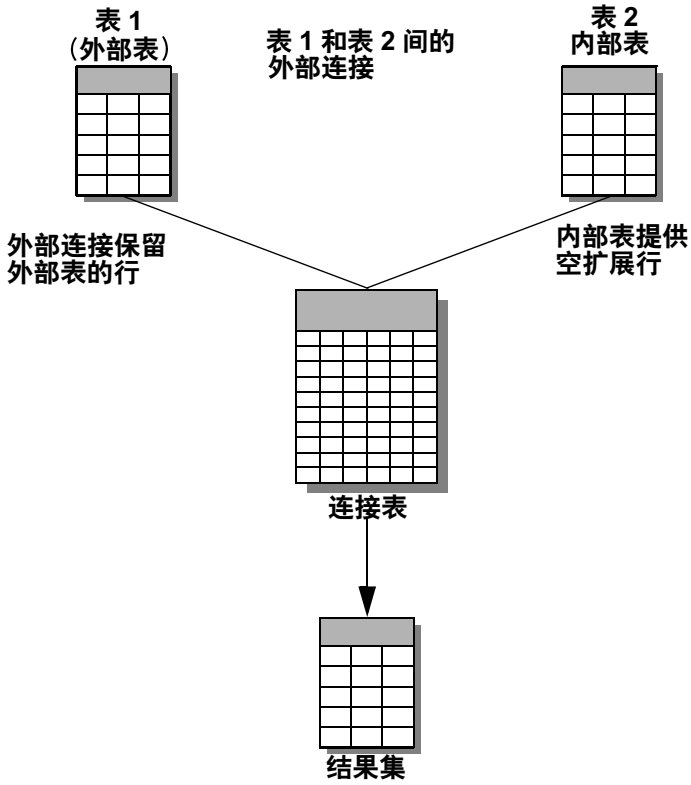
```
select stor_name, stor_address, ord_num, qty
from salesdetail inner join stores
on salesdetail.stor_id = stores.stor_id
and qty > 3000
```

如果限制条件放置在 where 子句中，查询通常更易读；它明确指示用户结果集中包括哪些连接表的行。

ANSI 外部连接

生成包括外部表所有行的连接表的连接（不管 on 子句是否生成匹配行），则称为**外部连接**。内部连接和等值连接生成的结果集仅包括在连接子句中存在匹配值的表中的行。然而，有时不但要包括匹配行，而且要包括在第二个表中没有匹配行的一个表中的行。此类连接为外部连接。在外部连接中，对于外部连接的内部表，具有空值的连接表中包括不满足 on 子句条件的行。内部表也称为空值提供成员。内部表和外部表的角色如图 4-1 所述：

图 *4-1*：外部连接中外部和内部表的角色



Sybase 建议应用程序使用 ANSI 外部连接，因为它们明确地指出是 on 还是 where 子句包含谓词。

本节仅讨论 ANSI 外部连接；有关 Transact-SQL 外部连接的信息，参见第 147 页上的“Transact-SQL 外部连接”。

注意：包含 ANSI 外部连接的查询不能包含 Transact-SQL 外部连接，反之亦然。然而，具有 ANSI 外部连接的查询可引用包含 Transact-SQL 外部连接的视图，反之亦然。

ANSI 外部连接语法为：

```
select select_list
    from table1 {left | right} [outer] join table2
    on predicate
    [join restriction]
```

左连接将保留在连接子句左侧列出的表引用的所有行；右连接将保留在连接子句右侧列出的表引用的所有行。在左连接中，左表引用称为外部表或行保留表。

下列示例确定与出版者居住在同一城市的作者：

```
select au_fname, au_lname, pub_name
from authors left join publishers
on authors.city = publishers.city
au_fname   au_lname    pub_name
---------  ----------  -------------
Johnson    White       NULL
Marjorie   Green       NULL
Cheryl     Carson      Algodata Infosystems
. . .
Abraham    Bennet      Algodata Infosystems
. . .
Anne       Ringer      NULL
Albert     Ringer      NULL
(23 rows affected)
```

结果集包括 authors 表中的所有作者。当作者与他们的出版者不居住同一城市时，pub_name 列中的值就为空。只有 Cheryl Carson 和 Abraham Bennet 与他们的出版者居住在同一城市，这两位作者在 pub_name 列中产生非空值。

可通过在 from 子句中调换表位置，将左外连接重写为右外连接。另外，如果 select 语句指定为“select*”，必须编写所有列名的显式列表，否则结果集中的列可能与重写查询顺序不同。

下面将先前示例重写为右外连接，这样生成与上述左外连接相同的结果集：

```
select au_fname, au_lname, pub_name
from publishers right join authors
on authors.city = publishers.city
```

谓词应放在 *on* 还是放在 *where* 子句?

ANSI 外部连接的结果集取决于在 on 还是 where 子句中放置限制条件。on 子句定义连接表的结果集和此连接表具有空值的行； where 子句定义要在结果集中包括的连接表的行。

在连接条件中使用 on 还是 where 子句，取决于要在结果集中包括的内容。下列示例有助于决定将谓词放置在 on 还是 where 子句中。

外部表上的谓词限制

下列查询在 where 子句中放置了外部表的限制条件。因为限制条件应用于外部连接的结果，所以它删除条件*非*真的所有行：

```
select title, titles.title_id, price, au_id
from titles left join titleauthor
on titles.title_id = titleauthor.title_id
where titles.price > $20.00
title                title_id price      au_id
------------------- -------- ---------- -----------------
But Is It User F... PC1035        22.95 238-95-7766
Computer Phobic ... PS1372        21.59 724-80-9391
Computer Phobic ... PS1372        21.59 756-30-7391
Onions, Leeks, a... TC3218        20.95 807-91-6654
(4 rows affected)
```

有四行满足标准并且结果集中仅包括这些行。

然而，如果将外部表的此限制条件移到 on 子句中，则结果集包括满足 on 子句条件的所有行。不满足条件的外部表的行为空扩展行：

```
select title, titles.title_id, price, au_id
from titles left join titleauthor
on titles.title_id = titleauthor.title_id
and titles.price > $20.00
title                   title_id   price   au_id
--------------------    ---------  ------  ---------------
The Busy Executiveís    BU1032     19.99   NULL
Cooking with Compute    BU1111     11.95   NULL
You Can Combat Compu    BU2075      2.99   NULL
Straight Talk About     BU7832     19.99   NULL
Silicon Valley Gastro   MC2222     19.99   NULL
The Gourmet Microwave   MC3021      2.99   NULL
The Psychology of Com   MC3026      NULL   NULL
But Is It User Friend   PC1035     22.95   238-95-7766
Secrets of Silicon Va   PC8888     20.00   NULL
Net Etiquette           PC9999      NULL   NULL
Computer Phobic and     PS1372     21.59   724-80-9391
Computer Phobic and     PS1372     21.59   756-30-7391
Is Anger the Enemy?     PS2091     10.95   NULL
```

```
Life Without Fear       PS2106      7.00   NULL
Prolonged Data Depri    PS3333     19.99   NULL
Emotional Security:     PS7777      7.99   NULL
Onions, Leeks, and Ga   TC3218     20.95   807-91-6654
Fifty Years in Buckin   TC4203     11.95   NULL
Sushi, Anyone?          TC7777     14.99   NULL
(19 rows affected)
```

将限制条件移动到 on 子句向结果集中增加了 15 个空值行。

通常，如果查询使用外部表的限制条件，并且想要结果集仅删除限制条件为假的行，可能应在 where 子句中放置限制条件，以限制结果集的行。如果外部表谓词在 on 子句上，则它们不用于索引键。

在 on 还是 where 子句中放置外部表的限制条件最终取决于需要查询返回的信息。如果只要结果集仅包括限制条件为真的行，应在 where 子句中放置限制条件。然而，如果需要结果集包括外部表的所有行，而不管它们是否满足限制条件，应在 on 子句中放置限制条件。

内部表的限制

下列查询在 where 子句中包括内部表的限制条件：

```
select title, titles.title_id, titles.price, au_id
from titleauthor left join titles
on titles.title_id = titleauthor.title_id
where titles.price > $20.00
title            title_id   price    au_id
-------------    --------   -----    -----------
But Is It U...   PC1035     22.95    238-95-7766
Computer Ph...   PS1372     21.59    724-80-9391
Computer Ph...   PS1372     21.59    756-30-7391
Onions, Lee...   TC3218     20.95    807-91-6654
(4 rows affected)
```

因为 where 子句的限制条件在连接*后*应用于结果集，所以它删除限制条件*非*真的所有行。换句话说，where 子句对于所有提供的空值为非真并将其删除。将其限制条件放置在 where 子句的连接是一个有效内部连接。

然而，如果将限制条件移动到 on 子句中，则它适用于连接过程中，并在生成连接表时利用它。在这种情况下，结果集包括限制条件为真的内部表中的所有行，以及外部表的所有行（如果它们不满足限制标准则为空扩展行）：

```
select title, titles.title_id, price, au_id
from titleauthor left join titles
on titles.title_id = titleauthor.title_id
and price > $20.00
```

```
title      title_id   price           au_id
---------  ---------  -----------     -----------
NULL       NULL       NULL            172-32-1176
NULL       NULL       NULL            213-46-8915
. . .
Onions,    TC3218     20.95           807-91-6654
. . .
NUL        NULL       NULL            998-72-3567
NULL       NULL       NULL            998-72-3567
(25 rows affected)
```

此结果集包括先前示例未包括的 21 行。

通常情况下，如果查询需要内部表上的限制条件（例如，上述查询中的“and price > $20.00”），可能要将此条件放置到 on 子句中；这样将保留外部表上的行。如果在 where 子句中包括内部表的限制条件，则结果集可能不包括外部表的行。

与在外部表上放置限制标准一样，将内部表的限制条件放置到 on 还是 where 子句上，最终取决于所需的结果集。如果仅对限制条件为真的行感兴趣，并且不包括外部表中行的全集，则将限制条件放置到 where 子句中。然而，如果要求结果集包括外部表的所有行，而不管它们是否满足限制条件，应在 on 子句中放置限制条件。

在内部和外部表中包括的限制

下列查询的 where 子句中的限制条件包括内部表和外部表：

```
select title, titles.title_id, price, price*qty, qty
from salesdetail left join titles
on titles.title_id = salesdetail.title_id
where price*qty > $30000.00
title                title_id   price  qty
-----------------    --------   -----  ---------  -----
Silicon Valley Ga    MC2222     19.99  40,619.68  2032
But Is It User Fr    PC1035     22.95  45,900.00  2000
But Is It User Fr    PC1035     22.95  45,900.00  2000
But Is It User Fr    PC1035     22.95  49,067.10  2138
Secrets of Silico    PC8888     20.00  40,000.00  2000
Prolonged Data De    PS3333     19.99  53,713.13  2687
Fifty Years in Bu    TC4203     11.95  32,265.00  2700
Fifty Years in Bu    TC4203     11.95  41,825.00  3500
(8 rows affected)
```

在 where 子句中放置限制条件消除了结果集中的下列行：

- 限制条件“price*qty>$30000.0”为假的行
- 限制条件“price*qty>$30000.0”为未知的行，因为 price 为空

要保留外部表的不匹配行，将限制条件移动到 on 子句，如下示例所示：

```
select title, titles.title_id, price, price*qty, qty
from salesdetail left join titles
on titles.title_id = salesdetail.title_id
and price*qty > $30000.00
title                title_id   price  qty
-----------------    --------   -----  ---------  -----
NULL                 NULL       NULL   NULL       75
NULL                 NULL       NULL   NULL       75
. . .
Secrets of Silico    PC8888     20.00  40,000.00  2000
. . .
NULL                 NULL       NULL   NULL       300
NULL                 NULL       NULL   NULL       400
(116 rows affected)
```

此查询在结果集中保留 salesdetail 表的所有 116 行，以及不满足限制条件的空扩展行。

放置包括内部和外部表的限制条件的位置取决于所需的结果集。如果只对限制条件为真的行感兴趣，则将限制条件放置在 where 子句中。然而，如果要包括外部表的所有行，而不管它们是否满足限制条件，应在 on 子句中放置限制条件。

嵌套 ANSI 外部连接

嵌套外部连接使用一个外部连接的结果集作为另一个外部连接的表引用。例如：

```
select t.title_id, title, ord_num, sd.stor_id, stor_name
from (salesdetail sd
left join titles t
on sd.title_id = t.title_id)   /*join #1*/
left join stores
on sd.stor_id = stores.stor_id /*join #2*/
title_id  title         ord_num  stor_id  stor_name
--------  ------------  -------  -------  ------------------------
TC3218    Onions, L...  234518   7896     Fricative Bookshop
TC7777    Sushi, An...  234518   7896     Fricative Bookshop
. . .
TC4203    Fifty Yea...  234518   6380     Eric the Read Books
MC3021  The Gourmet...  234518   6380     Eric the Read Books
(116 rows affected)
```

在本示例中，salesdetail 和 titles 表间的连接表在逻辑上首先生成，然后，在 salesdetail.stor_id 等于 stores.stor_id 时，与 stores 表的列进行连接。在语义上，连接中嵌套的每级创建一个连接表，然后用于下一连接。

在上述查询中，因为第一个外部连接成为第二个外部连接的运算符，所以此查询为**左嵌套外连接**。

下列示例显示*右嵌套外连接*：

```
select stor_name, qty, date, sd.ord_num
from salesdetail sd left join (stores /*join #1 */
left join sales on stores.stor_id = sales.stor_id) /*join #2 */
on stores.stor_id = sd.stor_id
where date > "1/1/1990"
stor_name      qty  date                 ord_num
------------   ---- ------------------   --------------------
News & Brews   200  Jun 13 1990 12:00AM  NB-3.142
News & Brews   250  Jun 13 1990 12:00AM  NB-3.142
News & Brews   345  Jun 13 1990 12:00AM  NB-3.142
. . .
Thoreau Read   1005 Mar 21 1991 12:00AM  ZZ-999-ZZZ-999-0A0
Thoreau Read   2500 Mar 21 1991 12:00AM  AB-123-DEF-425-1Z3
Thoreau Read   4000 Mar 21 1991 12:00AM  AB-123-DEF-425-1Z3
```

在本示例中，第二个连接（stores 和 sales 表间）在逻辑上首先生成，然后与 salesdetail 表连接。因为第二个外部连接用作第一个外部连接的表引用，此查询为**右嵌套外连接**。

如果第一个外部连接（“from salesdetail. . .”）的 on 子句失败，它向第二个外部连接的 stores 和 sales 表提供空值。

嵌套外部连接中的小括号

嵌套外部连接生成带有或不带有小括号的相同结果集。如果连接使用小括号构建，具有多个外部连接的较大查询对于用户可更加易读。

在嵌套外部连接中的 on 子句

在嵌套外部连接中的 on 子句的位置确定逻辑上首先处理的连接。从左到右读取，第一个 on 子句是要定义的第一个连接。

在下列示例中，第一个连接（小括号中）中 on 子句的位置表示它是第二个连接的表引用，所以它被*首先*定义，生成与 authors 表连接的表引用：

```
select title, price, au_fname, au_lname
from (titles left join titleauthor
on titles.title_id = titleauthor.title_id)   /*join #1*/
left join authors
on titleauthor.au_id = authors.au_id      /*join #2*/
and titles.price > $15.00
title            price      au_fname      au_lname
---------------  ---------  ------------  -------------
The Busy Exe...  19.99      Marjorie      Green
The Busy Exe...  19.99      Abrahame      Bennet
. . .
Sushi, Anyon...  14.99      Burt          Gringlesby
Sushi, Anyon...  14.99      Akiko         Yokomoto
(26 rows affected)
```

然而，如果 on 子句在其它位置，则连接按不同序列求值，但仍然生成相同结果集（这仅具有解释性的意义；如果连接表在逻辑上按不同顺序生成，则它很可能不会生成相同结果集）：

```
select title, price, au_fname, au_lname
from titles left join
(titleauthor left join authors
on titleauthor.au_id = authors.au_id)        /*join #2*/
on titles.title_id = titleauthor.title_id    /*join #1*/
and au_lname like "Yokomoto"


title                     price        au_fname          au_lname
----------------------    -------      -----------       -----------
The Busy Executiveís      19.99        Marjorie          Green
The Busy Executiveís      19.99        Abraham           Bennet
. . .
Sushi, Anyone?            14.99        Burt              Gringlesby
Sushi, Anyone?            14.99        Akiko             Yokomoto


(26 rows affected)
```

第一个连接（查询的最后一行）的 on 子句位置表示*第二个*左连接是第一个连接的表引用，所以它被首先执行。即，第二个左连接的结果与 titles 表连接。

用连接顺序依赖性转换外部连接

几乎为 Adaptive Server 早期版本编写的所有 Transact-SQL 外部连接，在 12.0 和更高版本的 Adaptive Server 上运行时将生成相同的结果集。然而，有一类外部连接查询，其结果集取决于优化过程中选择的连接顺序。根据查询谓词求值的位置，使用当前版本的 Adaptive Server 发出这些查询时，它们可生成不同的结果集。它们返回的结果集由将谓词分配给连接的 ANSI 规则确定。

谓词引用的所有表被处理后，才能对谓词求值。即在下列查询中，只有处理 titles 表后，才能对谓词“and titles.price > 20”求值：

```
select title, price, au_ord
from titles, titleauthor
where titles.title_id *= titleauthor.title_id
and titles.price > 20
```

早期版本 Adaptive Server 中的谓词根据下列语义求值：

- 如果谓词在外部连接的内部表上被求值，则谓词具有 on 子句的语义。
- 如果使用表（对于所有外部连接都为外部表或不依赖于连接顺序）对谓词求值，则谓词具有 where 子句的语义

注意：在生产环境中运行 Adaptive Server 之前，确保使用跟踪标志 4413 启动它，并且运行认为可能依赖于 12.0 版之前 Adaptive Server 中连接顺序的任何查询。运行依赖于 12.0 版之前 Adaptive Server 中连接顺序的查询时，用跟踪标志 4413 启动的 Adaptive Server 发出类似于下列的消息：

```
Warning: The results of the statement on line %d
are join-order independent.Results may differ on
pre-12.0 releases, where the query is potentially
join-order dependent.
```

确保解决在由 12.0 版本之前 Adaptive Server 生成的连接顺序查询的结果集上应用程序具有的依赖性。

依赖于外部连接的连接顺序何时发生影响?

一般情况下，依赖于查询的连接顺序不会发生任何问题，因为谓词通常仅引用：

- 外部表，它使用 where 子句语义求值
- 内部表，它使用 on 子句语义求值
- 内部表及其所依赖的表

这些表不会生成依赖于外部连接的连接顺序。然而，具有下列任何特性的 Transact-SQL 查询转换为 ANSI 外部连接后，它们可生成不同结果集：

- 谓词包含 or 语句并引用外部连接的内部表和不依赖于此内部表的另一个表
- 内部表特性，与不依赖于内部表连接顺序的表相同谓词中进行引用
- 在子查询中作为相关引用来引用的内部表

下列示例演示了用连接顺序依赖性将 Transact-SQL 查询转换为 ANSI 外部连接查询的问题。

示例：

```
select title, price, authors.au_id, au_lname
from titles, titleauthor, authors
where titles.title_id =* titleauthor.title_id
and titleauthor.au_id = authors.au_id
where (titles.price is null or authors.postalcode = '94001')
```

此查询不依赖于连接顺序，因为外部连接引用 titleauthor 和 titles 表，并且根据如下三个连接顺序使用这些表连接 authors 表：

- authors、titleauthors、titles（作为 on 子句的一部分）
- titleauthors、authors、titles（作为 on 子句的一部分）
- titleauthors、titles、authors（作为 where 子句的一部分）

此查询生成下列消息：

```
Warning: The results of the statement on line 1 are join-order
independent.Results may differ on pre-12.0 releases, where the query is
potentially join-order dependent. Use trace flag 4413 to suppress this warning
message.
```

下面是等同的 ANSI 示例：

```
select title, price, authors.au_id, au_lname
from titles right join
(titleauthor inner join authors
on titleauthor.au_id = authors.au_id)
on titles.title_id = titleauthor.title_id
where (titles.price is null or authors.postalcode = '94001')
```

另一个示例：

```
select title, au_fname, au_lname, titleauthor.au_id, price
from titles, titleauthor, authors
where authors.au_id *= titleauthor.au_id
and titleauthor.au_ord*titles.price > 40
```

此查询依赖于连接顺序，原因与先前示例相同。下面是等同的 ANSI 示例：

```
select title, au_fname, au_lname, titleauthor.au_id, price
from titles,(authors left join titleauthor
on titleauthor.au_id = authors.au_id)
where titleauthor.au_ord*titles.price > 40
```

Transact-SQL 外部连接

Transact-SQL 包括左外连接和右外连接的语法。*左外连接* (*=)，选择满足语句限制条件的第一个表的所有行。如果与连接条件相匹配，第二个表产生值。否则，第二个表产生空值。

例如，下列左外连接列出所有作者并查找其城市中的出版者（如果有）：

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city *= publishers.city
```

右外连接 (=*)，选择满足语句限制条件的第二个表的所有行。如果与连接条件相匹配，第一个表产生值。否则，第一个表产生空值。

注意：在 having 子句中不能包括 Transact-SQL 外部连接。

表是外部连接的内部成员或外部成员。如果连接运算符为 *=，则第二个表为内部表；如果连接运算符为 =*，则第一个表为内部表。可将内部表的某一列比作常量，并将其使用于外部连接中。例如，如果要查找销售超过 4000 册的 title：

```
select qty, title from salesdetail, titles
where qty > 4000
and titles.title_id *= salesdetail.title_id
```

然而，外部连接的内部表也无法编入常规连接子句。

先前示例使用连接来查找与出版者居住在同一城市的作者姓名，返回两个姓名： Abraham Bennet 和 Cheryl Carson。要在结果中包括所有作者，而不管是否与出版者居住在同一城市，则使用外部连接。外部连接的查询和结果如下所示：

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city *= publishers.city
au_fname     au_lname         pub_name
---------    --------------   ---------------
Johnson      White            NULL
Marjorie     Green            NULL
Cheryl       Carson           Algodata Infosystems
Michael      OíLeary          NULL
Dick         Straight         NULL
Meander      Smith            NULL
Abraham      Bennet           Algodata Infosystems
Ann          Dull             NULL
Burt         Gringlesby       NULL
Chastity     Locksley         NULL
Morningstar  Greene           NULL
Reginald     Blotche-Halls    NULL
Akiko        Yokomoto         NULL
Innes        del Castillo     NULL
Michel       DeFrance         NULL
Dirk         Stringer         NULL
Stearns      MacFeather       NULL
Livia        Karsen           NULL
Sylvia       Panteley         NULL
Sheryl       Hunter           NULL
Heather      McBadden         NULL
Anne         Ringer           NULL
Albert       Ringer           NULL

(23 rows affected)
```

比较运算符 *= 使外部连接区别于普通连接。“左”外部连接指示 Adaptive Server 在结果中包括 authors 表的所有行，而无论是否与 publishers 表中的 city 列相匹配。注意，在结果中对于列出的多数作者没有匹配数据，所以这些行在 pub_name 列中包含 NULL。

“右”外连接使用比较运算符 =* 指定，它表示将在结果中包括第二个表的所有行，而不管在第一个表中是否有匹配数据。

在以前所示的外部连接中替换此运算符，得出如下结果：

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city =* publishers.city
au_fname     au_lname    pub_name
---------    ---------   ---------------
NULL         NULL        New Age Books
NULL         NULL        Binnet & Hardley
Cheryl       Carson      Algodata Infosystems
Abraham      Bennet      Algodata Infosystems

(4 rows affected)
```

可通过将其用作常量以进一步限制外部连接。这意味着可精确地放大确实要查看的值，并使用外部连接列出未进行剪切的行。首先查看等值连接，然后将其与外部连接比较。例如，要从任何商店中查找销售量超过 500 册的书目，使用如下查询：

```
select distinct salesdetail.stor_id, title
from titles, salesdetail
where qty > 500
and salesdetail.title_id = titles.title_id
stor_id title
------- ---------------------------------------------
5023    Sushi, Anyone?
5023    Is Anger the Enemy?
5023    The Gourmet Microwave
5023    But Is It User Friendly?
5023    Secrets of Silicon Valley
5023    Straight Talk About Computers
5023    You Can Combat Computer Stress!
5023    Silicon Valley Gastronomic Treats
5023    Emotional Security: A New Algorithm
5023    The Busy Executiveís Database Guide
5023    Fifty Years in Buckingham Palace Kitchens
5023    Prolonged Data Deprivation: Four Case Studies
5023    Cooking with Computers: Surreptitious Balance Sheets
7067    Fifty Years in Buckingham Palace Kitchens

(14 rows affected)
```

另外，要显示任何商店中销售量未超过 500 册的书目，可使用外部连接查询：

```
select distinct salesdetail.stor_id, title
from titles, salesdetail
where qty > 500
and salesdetail.title_id =* titles.title_id
```

```
stor_id    title
-------   ------------------------------------------
NULL    Net Etiquette
NULL    Life Without Fear
5023    Sushi, Anyone?
5023    Is Anger the Enemy?
5023    The Gourmet Microwave
5023    But Is It User Friendly?
5023    Secrets of Silicon Valley
5023    Straight Talk About Computers
NULL    The Psychology of Computer Cooking
5023    You Can Combat Computer Stress!
5023    Silicon Valley Gastronomic Treats
5023    Emotional Security: A New Algorithm
5023    The Busy Executiveís Database Guide
5023    Fifty Years in Buckingham Palace Kitchens
7067    Fifty Years in Buckingham Palace Kitchens
5023    Prolonged Data Deprivation: Four Case Studies
5023    Cooking with Computers: Surreptitious Balance Sheets
NULL    Computer Phobic and Non-Phobic Individuals:
        Behavior Variations
NULL    Onions, Leeks, and Garlic: Cooking Secrets of the
        Mediterranean

 (19 rows affected)
```

可用一个简单子句限制内部表。下列示例列出与出版者居住在同一城市的作者，但不包括作者 Cheryl Carson，他通常作为居住在出版者城市的作者列出：

```
select au_fname, au_lname, pub_name
from authors, publishers
where authors.city =* publishers.city
and authors.au_lname != "Carson"
```

```
au_fname     au_lname    pub_name
---------    ---------   ---------------
NULL         NULL        New Age Books
NULL         NULL        Binnet & Hardley
Abraham      Bennet      Algodata Infosystems
```

```
(3 rows affected)
```

空值如何影响连接

被连接的表或视图中的空值决不能互相匹配。由于 bit 列不允许为空值，所以当内部表中的 bit 列上没有匹配值时，在外部连接中显示值为 0。

NULL 与任何其它值的连接结果为 NULL。因为空值代表未知或不适用的值，Transact-SQL 没有理由认为一个未知值与另一值相匹配。

仅使用一个外部连接，就可检测被连接的一个表中的列是否存在空值。下面是两个表，每个表在参与连接的列中都有 NULL 值。左外连接在第一个表中显示空值。

表 t1

a	b
1	one
NULL	three
4	join4

表 t2

c	d
NULL	two
4	four

下面是左外连接：

```
select *
from t1, t2
where a *= c
 a            b       c            d
 -----------  ------  -----------  ------
           1  one            NULL  NULL
        NULL  three          NULL  NULL
           4  join4             4  four

(3 rows affected)
```

结果很难区分出是数据中的空值还是代表失败连接的空值。在被连接的数据中出现空值时，通常使用常规连接将其从结果中省略。

确定要连接的表列

sp_helpjoins 列出两个表或视图中适于连接的列。此命令的语法为：

```
sp_helpjoins table1, table2
```

例如，下面说明如何使用 sp_helpjoins 查找 titleauthor 和 titles 间适于连接的列：

```
sp_helpjoins titleauthor, titles
first_pair
----------------------------- --------------------
title_id                      title_id
```

sp_helpjoins 显示的列对来自两个来源。首先，sp_helpjoins 检查当前数据库中的 syskeys 表，查看是否已使用 sp_foreignkey 在两个表上定义任何外键，然后，检查是否已用 sp_commonkey 在两个表上定义了任何公用键。如果它未在其中找到任何公用键，此过程应用较少的限制标准来识别可能被适当连接的任何键。它检查具有相同用户数据类型的键，如果仍找不到，它将检查具有相同名称和数据类型的列。

有关系统过程的完整信息，参见*参考手册*。

第 5 章

子查询：在其它查询中使用查询

子查询是一个 select 语句，它嵌套于另一个 select、insert、update 或 delete 语句中，在条件语句或另一个子查询内。

本章讨论如下内容：

主题	页
子查询工作方式	153
子查询类型	161
使用相关子查询	178

也可将子查询表示为连接操作。参见第 4 章“连接：从若干表中检索数据”。

子查询工作方式

子查询又称**内部查询**，它出现于其它 SQL 语句的 where 或 having 子句中，或出现于语句的选择列表中。子查询可用来处理表达为另一个查询的结果的查询请求。包含子查询的语句会依据子查询 select 列表的求值来处理来自某个表的行，它可与外层查询引用相同的表，或者也可以引用其它表。在 Transact-SQL 中，子查询还可用于允许使用表达式的几乎任何地方，条件是子查询返回一个单值。 case 表达式也可包含子查询。

例如，下面的子查询列出 royalty split 值大于 $75 的所有作者的名字：

```
select au_fname, au_lname
from authors
where au_id in
   (select au_id
    from titleauthor
    where royaltyper > 75)
```

包含一个或多个子查询的 select 语句有时也称为**嵌套查询**或**嵌套选择语句**。

您可将连接明确地表示为包含子查询的多个 SQL 语句。 其它的问题也可以只通过子查询来解决。有些人认为子查询非常易于理解。而其它 SQL 用户则尽可能地避免使用子查询。您可以选择自己喜欢的方式。

不返回值的子查询的结果为 NULL。如果子查询返回 NULL，则此查询失败。

子查询语法

子查询的 select 语句务必用小括号括起来。子查询的 select 语法有如下限制：

```
(select [all | distinct] subquery_select_list
    [from [[database.]owner.]{table_name |view_name}
      [({index index_name | prefetch size |[lru|mru]})]}
            [holdlock | noholdlock] [shared]
        [,[[database.]owner.]{table_name |view_name}
      [({index index_name | prefetch size |[lru|mru]})]}
            [holdlock | noholdlock] [shared]]... ]
    [where search_conditions]
    [group by aggregate_free_expression [,
            aggregate_free_expression]... ]
    [having search_conditions])
```

子查询限制

子查询有如下限制：

- 除在 exists 子查询中外（通常以 (*) 代替单个列名），*subquery_select_list* 只能由一个列名组成。不要指定多个列名。如果列名所属的表或视图存在歧义，应用表名或视图名限定列名。
- 子查询可嵌套于外层 select、insert、update 或 delete 语句的 where 或 having 子句中，在另一个子查询或选择列表中。或者，也可编写多个包含子查询的语句作为连接； Adaptive Server 将这种语句作为连接来处理。
- 在 Transact-SQL 中，子查询可用于允许使用表达式的几乎任何地方，条件是子查询返回一个单值。
- 不可在 order by、group by 或 compute by 列表中使用子查询。
- 不可在子查询中包含 for browse 子句或 union。

- 由比较运算符引入的内部子查询中的选择列表只能包含一个表达式或列名，并且子查询必须返回一个单值。在外层语句的 where 子句中指定的列与在子查询选择列表中指定的列必须是连接兼容的。
- 子查询中不允许使用 text 和 image 数据类型。
- 子查询不能在内部处理其结果，也就是说子查询不能包含 order by 子句、compute 子句或 into 关键字。
- 在由 declare cursor 定义的可更新游标的 select 子句中不允许有相关（重复）子查询。
- 嵌套不能超过 16 层。
- 在 union 运算符两边最多可有 16 个子查询。
- 子查询的 where 子句可包含集合函数，但条件是该子查询必须在外层查询的 having 子句中，并且集合值必须为外层查询 from 子句中的某表上的一列。
- 子查询指定的所有列的最大长度之和不能超过 256 字节。

使用子查询的示例

假设要查找与 *Straight Talk About Computers* 价格相同的书。首先，查找 *Straight Talk* 的价格：

```
select price
from titles
where title = "Straight Talk About Computers"
price
-------------
       $19.99

(1 row affected)
```

将第一个查询的结果用于第二个查询中，以查找所有与 *Straight Talk* 价格相同的书：

```
select title, price
from titles
where price = $19.99
title                                          price
-------------------------------------------     -----
The Busy Executiveís Database Guide            19.99
Straight Talk About Computers                  19.99
```

```
Silicon Valley Gastronomic Treats            19.99
Prolonged Data Deprivation: Four Case Studies 19.99

(4 rows affected)
```

使用子查询只需一步即可得到相同的结果：

```
select title, price
from titles
where price =
   (select price
    from titles
    where title = "Straight Talk About Computers")
title                                          price
----------------------------------------       -----
The Busy Executive's Database Guide           19.99
Straight Talk About Computers                 19.99
Silicon Valley Gastronomic Treats             19.99
Prolonged Data Deprivation: Four Case Studies 19.99

(4 rows affected)
```

限定列名

语句中的列名由在同级的 from 子句中引用的表隐式限定。下例中，表名 publishers 隐式限定外层查询 where 子句中的 pub_id 列。在子查询选择列表中对 pub_id 的引用由子查询的 from 子句（即 titles 表）进行限定：

```
select pub_name
from publishers
where pub_id in
   (select pub_id
    from titles
    where type = "business")
```

以下即为隐式假定生效时的查询：

```
select pub_name
from publishers
where publishers.pub_id in
   (select titles.pub_id
    from titles
    where type = "business")
```

显式地规定表名永远都不会错，可通过使用显式限定来替换表名的隐式假定。

带相关名的子查询

如第 4 章“连接：从若干表中检索数据”所述，自连接中需要使用表相关名，因为与其自身连接的表出现出两种不同的角色。也可在嵌套查询（此查询的内部与外部查询引用同一个表）中使用相关名。

例如，可使用以下子查询来查找与 Livia Karsen 居住在同一城市的作者：

```
select au1.au_lname, au1.au_fname, au1.city
from authors au1
where au1.city in
   (select au2.city
    from authors au2
    where au2.au_fname = "Livia"
    and au2.au_lname = "Karsen")
au_lname     au_fname     city
-----------  ---------    -------
Green        Marjorie     Oakland
Straight     Dick         Oakland
Stringer     Dirk         Oakland
MacFeather   Stearns      Oakland
Karsen       Livia        Oakland

(5 rows affected)
```

显式相关名清楚地显示：在子查询中对 authors 的引用与在外层查询中对 authors 的引用并不相同。

如果没有显式相关，则子查询为：

```
select au_lname, au_fname, city
from authors
where city in
   (select city
    from authors
    where au_fname = "Livia"
    and au_lname = "Karsen")
```

或者，可将上述查询及其它语句（其中的子查询与外层查询引用同一个表）表述为自连接：

```
select au1.au_lname, au1.au_fname, au1.city
from authors au1, authors au2
where au1.city = au2.city
and au2.au_lname = "Karsen"
and au2.au_fname = "Livia"
```

重述为连接的子查询返回结果的顺序可能会有所不同；此外，此连接可能需要使用 distinct 关键字来消除重复项。

多重嵌套

子查询可包含一个或多个子查询。在一个语句中最多可嵌套 16 个子查询。

例如：“查找参与编写了至少一本计算机畅销书的作者的名字：”

```
select au_lname, au_fname
from authors
where au_id in
   (select au_id
    from titleauthor
    where title_id in
       (select title_id
        from titles
        where type = "popular_comp") )
au_lname                au_fname
---------------------- ------------
Carson                  Cheryl
Dull                    Ann
Locksley                Chastity
Hunter                  Sheryl

(4 rows affected)
```

最外层查询选择所有的作者名。下一层查询查找作者 ID，最内层查询返回标题 ID 号 PC1035、PC8888 和 PC9999。

也可将此查询表示为一个连接：

```
select au_lname, au_fname
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
and type = "popular_comp"
```

update、*delete* 和 *insert* 语句中的子查询

可在 update、delete 和 insert 语句及 select 语句中嵌套子查询。

注意：运行这些示例查询会改变 pubs2 数据库。要获得样本数据库的原始副本，请向系统管理员寻求帮助。

以下查询将 New Age Books 出版的所有书籍的价格加倍。此语句会更新 titles 表；其子查询引用 publishers 表。

```
update titles
set price = price * 2
where pub_id in
   (select pub_id
    from publishers
    where pub_name = "New Age Books")
```

使用连接的等同 update 语句为：

```
update titles
set price = price * 2
from titles, publishers
where titles.pub_id = publishers.pub_id
and pub_name = "New Age Books"
```

可用以下嵌套 select 语句删除所有的商业书籍销售记录：

```
delete salesdetail
where title_id in
   (select title_id
    from titles
    where type = "business")
```

使用连接的等同 delete 语句为：

```
delete salesdetail
from salesdetail, titles
where salesdetail.title_id = titles.title_id
and type = "business"
```

条件语句中的子查询

在条件语句中可使用子查询。可将上述删除所有商业书籍销售记录的子查询重新编写为下例所示，以在删除这些记录*之前*对它们进行检查：

```
if exists (select title_id
    from titles
    where type = "business")
begin
    delete salesdetail
    where title_id in
      (select title_id
       from titles
       where type = "business")
end
```

用子查询替代表达式

在 Transact-SQL 中，select、update、insert 或 delete 语句中几乎所有可使用表达式的地方都可用子查询来替代。例如，子查询可与外部连接的内部表中的列进行比较。

可将子查询用于 order by 列表中，或者将其用作 insert 语句的 values 列表中的一个表达式。

以下语句显示如何查找由居住于 California 的作者所写并在那里出版的书籍的标题和类型：

```
select title, type
from titles
where title in
   (select title
    from titles, titleauthor, authors
    where titles.title_id = titleauthor.title_id
    and titleauthor.au_id = authors.au_id
    and authors.state = "CA")
and title in
   (select title
    from titles, publishers
    where titles.pub_id = publishers.pub_id
    and publishers.state = "CA")
title                                  type
-----------------------------------    ----------
The Busy Executiveís Database Guide    business
Cooking with Computers:
```

```
     Surreptitious Balance Sheets     business
Straight Talk About Computers         business
But Is It User Friendly?              popular_comp
Secrets of Silicon Valley             popular_comp
Net Etiquette                         popular_comp

(6 rows affected)
```

以下语句选择销量在 5000 册以上的书籍的书名，并列出其价格，及最贵书籍的价格：

```
select title, price,
    (select max(price) from titles)
    from titles
    where total_sales > 5000
title                                 price
-----------------------------------  -----  ------
You Can Combat Computer Stress!       2.99   22.95
The Gourmet Microwave                 2.99   22.95
But Is It User Friendly?             22.95   22.95
Fifty Years in Buckingham Palace
    Kitchens                         11.95   22.95

(4 rows affected)
```

子查询类型

子查询分为两种基本类型：

- **表达式子查询**由无修饰词的比较运算符引入，它必须返回单个值，可用于 SQL 中几乎所有允许使用表达式的地方。
- **定量判定子查询**对由 in 引入的列表、或由经 any 或 all 修饰的比较运算符引入的列表进行操作。定量判定子查询返回 0 或更多的值。此类子查询也用作存在测试，由 exists 引入。

每种类型的子查询或是无关的或是相关的（重复的）。

- **无关子查询**可按独立查询那样进行求值。在概念上，子查询的结果在主语句或外层查询中进行替代。但这并*不是* Adaptive Server 对含有子查询的语句的实际处理方式。无关子查询也可表达为连接，并由 Adaptive Server 将其作为连接来处理。

- **相关子查询**不可作为独立查询进行求值，但可引用外层查询中 from 列表中列出的表中的列。本章最后将详细讨论相关子查询。

表达式子查询

表达式子查询包括：

- select 列表中的子查询（由 in 引入）
- 由比较运算符（=、!=、>、>=、<、<=）连接的 where 或 having 子句中的子查询

表达式子查询采取常规格式：

[以 select、insert、update 或 delete 语句或子查询开始]

```
where expression comparison_operator (subquery)
```

[以 select、insert、update 或 delete 语句或子查询结束]

表达式由子查询或任何由算术运算符或逐位运算符连接的列名、常量和函数的组合组成。

comparison_operator 为以下运算符之一：

运算符	含义
=	等于
>	大于
<	小于
>=	大于或等于
<=	小于或等于
!=	不等于
<>	不等于
!>	不大于
!<	不小于

如果使用外层语句的 where 或 having 子句中的列名，应确保 *subquery_select_list* 中的列名与它是连接兼容的。

由无修饰词的比较运算符（即后面没有 any 或 all）引入的子查询必须求出一个单值。如果此类子查询返回多个值，Adaptive Server 会返回一个错误消息。

例如，假设每个出版者仅位于一个城市中。要查找与 Algodata Infosystems 位于同一个城市的作者的名字，可编写一个含有子查询（由比较运算符 = 引入）的语句：

```
select au_lname, au_fname
from authors
where city =
   (select city
    from publishers
    where pub_name = "Algodata Infosystems")
au_lname       au_fname
-------------- --------------
Carson         Cheryl
Bennet         Abraham

(2 rows affected)
```

使用标量集合函数来确保单值

由无修饰词的比较运算符引入的子查询通常包括标量集合函数，因为它们返回单值。

例如，要查找价格高于当前最低价格的书籍的书名：

```
select title
from titles
where price >
   (select min(price)
    from titles)
title
----------------------------------------------------
The Busy Executiveís Database Guide
Cooking with Computers: Surreptitious Balance
     Sheets
Straight Talk About Computers
Silicon Valley Gastronomic Treats
But Is It User Friendly?
Secrets of Silicon Valley
Computer Phobic and Non-Phobic Individuals:
     Behavior Variations
Is Anger the Enemy?
Life Without Fear
Prolonged Data Deprivation: Four Case Studies
Emotional Security: A New Algorithm
Onions, Leeks, and Garlic: Cooking Secrets of the
     Mediterranean
Fifty Years in Buckingham Palace Kitchens
```

```
Sushi, Anyone?

(14 rows affected)
```

表达式子查询中的 *group by* 和 *having*

因为由无修饰词的比较运算符引入的子查询必须返回一个单值，所以它们不能包括 group by 和 having 子句，除非已知 group by 和 having 子句将返回单值。

例如，以下查询查找价格高于 trad_cook 类别中最低价书的书名：

```
select title
from titles
where price >
   (select min(price)
    from titles
    group by type
    having type = "trad_cook")
```

在表达式子查询中使用 *distinct*

由无修饰词的比较运算符引入的子查询通常包括 distinct 关键字，以确保返回单值。

例如，如果不包含 distinct，以下子查询将由于返回多个值而失败：

```
select pub_name from publishers
    where pub_id =
        (select distinct pub_id
        from titles
        where pub_id  = publishers.pub_id)
```

定量判定子查询

定量判定子查询是由 any、all、in 或 exists 连接的 where 或 having 子句中的子查询，它返回 0 和更高值的列表。any 或 all 子查询运算符修饰比较运算符。

定量判定子查询共分三类：

- any/all子查询。由有修饰词的比较运算符引入的子查询（可能包括 group by 或 having 子句）采取常规格式：

[以 select、insert、update 或 delete 语句或子查询开始]

```
where expression comparison_operator [any | all]
    (subquery)
```

[以 select、insert、update 或 delete 语句或子查询结束]

- in/not in 子查询。由 in 或 not in 引入的子查询采取常规格式：

  [以 select、insert、update 或 delete 语句或子查询开始]

  ```
  where expression [not] in (subquery)
  ```

  [以 select、insert、update 或 delete 语句或子查询结束]

- exists/not exists 子查询。由 exists 或 not exists 引入的子查询是采取常规格式的存在测试：

  [以 select、insert、update 或 delete 语句或子查询开始]

  ```
  where [not] exists (subquery)
  ```

  [以 select、insert、update 或 delete 语句或子查询结束]

尽管 Adaptive Server 允许在定量判定子查询中使用关键字 distinct，但它总是按子查询中不包括 distinct 那样对其进行处理。

带 *any* 和 *all* 的子查询

关键字 all 和 any 修饰引入子查询的比较运算符。

如果在子查询中将 any 和 <、> 或 = 一同使用时，则当在子查询中检索到与外层语句的 where 或 having 子句中的值相匹配的任意值时，将返回结果。

如果在子查询中将 all 和 < 或 > 一同使用，则只有在子查询中检索到与外层语句的 where 或 having 子句中的值相匹配的所有值时，才返回结果。

any 和 all 的语法为：

{where | having} [not]
expression comparison_operator {any |all} (*subquery*)

以 > 比较运算符为例：

- > all表示大于每个值，或大于最大值。例如，> all (1, 2, 3) 表示大于 3。
- > any表示大于至少一个值，或大于最小值。例如，> any (1, 2, 3) 表示大于 1。

如果使用 all 引入子查询，而比较运算符不返回任何值，则整个查询失败。

all 和 any 可以互相转换。例如，您可能会问“哪本书的预付款比 New Age Books 出版的任何一本书的预付款都多？”

可这样解释这个问题，以使 SQL“转换”更清楚：“哪本书的预付款比 New Age Books 付出的最高预付款还多？” 此时需要 all 关键字，而*非* any 关键字：

```
select title
from titles
where advance > all
   (select advance
    from publishers, titles
    where titles.pub_id = publishers.pub_id
    and pub_name = "New Age Books")
title
----------------------------------------
The Gourmet Microwave

(1 row affected)
```

对于每个标题，外层查询从 titles 表中获取标题和预付款值，并将它们与从子查询中返回的 New Age Books 所付预付款数进行比较。外层查询查找列表中的最大值，并确定所考虑的标题是否得到更多的预付款。

> *all* 表示大于所有值

> all 运算符表示：引入子查询的列中的值必须大于子查询为符合外层查询条件的行返回的每个值。

例如，要查找价格高于 mod_cook 类别中最高价书的书名：

```
select title from titles where price > all
    (select price from titles
    where type = "mod_cook")
title
--------------------------------------------------
But Is It User Friendly?
Secrets of Silicon Valley
Computer Phobic and Non-Phobic Individuals:
    Behavior Variations
Onions, Leeks, and Garlic: Cooking Secrets of
    the Mediterranean

(4 rows affected)
```

但是，如果内层查询返回的结果集中包含 NULL，则查询返回 0 行。这是因为 NULL 代表“未知值”，而要判断出比较的值是否大于未知值是不可能的。

例如，要查找价格高于 popular_comp 类别中最高价书的书名：

```
select title from titles where price > all
    (select price from titles
    where type = "popular_comp")
title
----------------------------------------------------

(0 rows affected)
```

未返回行，这是因为子查询发现其中一本书 — *Net Etiquette* — 的价格为空值。

= *all* 表示等于每个值

= all 运算符表示：引入子查询的列中的值必须等于子查询为符合外层查询条件的行返回的值列表中的每个值。

例如，以下查询通过查找邮政编码来找出居住在同一个城市的作者：

```
select au_fname, au_lname, city
from authors
where city = all
     (select city
     from authors
     where postalcode like "946%")
```

> *any* 表示至少大于一个值

> any 表示：引入子查询的列中的值必须至少大于子查询为符合外层查询条件的行返回的多个值中的一个。

下例由经 any 修饰的比较运算符引入。它查找预付款值大于 New Age Books 所付任意预付款数的每本书的标题。

```
select title
from titles
where advance > any
   (select advance
    from titles, publishers
    where titles.pub_id = publishers.pub_id
    and pub_name = "New Age Books")
title
```

```
--------------------------------------------------
The Busy Executive's Database Guide
Cooking with Computers: Surreptitious Balance
    Sheets
You Can Combat Computer Stress!
Straight Talk About Computers
The Gourmet Microwave
But Is It User Friendly?
Secrets of Silicon Valley
Computer Phobic and Non-Phobic Individuals:
    Behavior Variations
Is Anger the Enemy?
Life Without Fear
Emotional Security: A New Algorithm
Onions, Leeks, and Garlic: Cooking Secrets of
    the Mediterranean
Fifty Years in Buckingham Palace Kitchens
Sushi, Anyone?

(14 rows affected)
```

对于外层查询选择的每个标题，内层查询都会查找 New Age Books 所付预付款数的列表。外层查询会查看列表中的所有值，并确定所考虑的标题的预付款是否比这些值中的任意一个多。也就是说，本示例查找预付款大于或等于 New Age Books 所付*最低*预付款数的书籍的标题。

在子查询不返回任何值，则整个查询失败。

= *any* 表示等于一些值

= any 运算符是一个存在性检查；它等同于 in。例如，要查找与任意出版者处于同一城市的作者的名字，可使用 =any 或 in：

```
select au_lname, au_fname
from authors
where city = any
   (select city
    from publishers)
select au_lname, au_fname
from authors
where city in
   (select city
    from publishers)
au_lname       au_fname
-------------- --------------
Carson         Cheryl
```

```
Bennet          Abraham

(2 rows affected)
```

但是，!= any 运算符与 not in 不同。!= any 运算符表示“not = a *or* not = b *or* not = c”；而 not in 表示“not = a *and* not = b *and* not = c”。

例如，要查找居住于无出版者的城市的作者的名字：

```
select au_lname, au_fname
from authors
where city != any
   (select city
    from publishers)
```

结果包括所有 23 个作者。这是因为每个作者居住于无出版者的*某个*城市，且每个作者只居住在一个城市中。

内层查询会找到有出版者的所有城市，然后外层查询为*每个*城市查找不居住于此的作者。

在同一个查询中如果用 not in 替代，则会出现以下情况：

```
select au_lname, au_fname
from authors
where city not in
   (select city
    from publishers)
au_lname                au_fname
--------------          ------------
White                   Johnson
Green                   Marjorie
O'Leary                 Michael
Straight                Dick
Smith                   Meander
Dull                    Ann
Gringlesby              Burt
Locksley                Chastity
Greene                  Morningstar
Blotchet-Halls          Reginald
Yokomoto                Akiko
del Castillo            Innes
DeFrance                Michel
Stringer                Dirk
MacFeather              Stearns
Karsen                  Livia
Panteley                Sylvia
Hunter                  Sheryl
McBadden                Heather
```

```
Ringer                    Anne
Ringer                    Albert

(21 rows affected)
```

这些正是所需的结果。它们包括除 Cheryl Carson 和 Abraham Bennet 之外的所有作者，这两位作者居住于 Berkeley，那里是 Algodata Infosystems 的所在地。

使用 !=all 可得到同样的结果，它等同于 not in：

```
select au_lname, au_fname
from authors
where city != all
   (select city
    from publishers)
```

带 *in* 的子查询

由关键字 in 引入的子查询返回一个 0 和更高值的列表。例如，以下查询会查找出版过商业书籍的出版者的名称：

```
select pub_name
from publishers
where pub_id in
   (select pub_id
    from titles
    where type = "business")
pub_name
----------------------------------------
New Age Books
Algodata Infosystems

(2 rows affected)
```

此语句按两个步骤执行。内层查询返回出版过商业书籍的出版者的标识号，1389 和 0736。然后这些值在外层查询中进行替代，外层查询会在 publishers 表中查找与标识号相匹配的名称。此查询如下：

```
select pub_name
from publishers
where pub_id in ("1389", "0736")
```

使用子查询表达这个查询的另外一种方法是：

```
select pub_name
```

```
from publishers
where "business" in
   (select type
    from titles
    where pub_id = publishers.pub_id)
```

注意，外层查询 where 关键字之后的表达式可以是常量和列名。可以使用其它类型的表达式，例如常量和列名的组合。

上述查询与许多其它子查询一样，也可表达为一个连接查询：

```
select distinct pub_name
from publishers, titles
where publishers.pub_id = titles.pub_id
and type = "business"
```

此查询及表达后的子查询都会查找出版过商业书籍的出版者。尽管可能需要使用 distinct 关键字来消除重复项，但它们都同样正确并产生同样的结果。

但是，对于这个问题或类似问题，使用连接查询而不使用子查询的一个好处是：连接查询在结果中显示来自多个表的列。例如，要在结果中包括商业书籍的标题，需要使用连接：

```
select pub_name, title
from publishers, titles
where publishers.pub_id = titles.pub_id
and type = "business"
pub_name                title
--------------------    ----------------------------------------
Algodata Infosystems    The Busy Executive's Database Guide
Algodata Infosystems    Cooking with Computers: Surreptitious
                        Balance Sheets
New Age Books           You Can Combat Computer Stress!
Algodata Infosystems    Straight Talk About Computers

(4 rows affected)
```

以下是另一个示例，说明语句可表达为子查询或连接查询：“查找所有居住于 California、并从某本书中获得的版税低于 30% 的所有第二作者的名字。” 使用子查询的语句是：

```
select au_lname, au_fname
from authors
where state = "CA"
and au_id in
   (select au_id
    from titleauthor
    where royaltyper < 30
```

```
    and au_ord = 2)
au_lname                 au_fname
------------------------ ------------
MacFeather               Stearns

(1 row affected)
```

外层查询产生一个列表，列出 15 位居住于 California 的作者。然后，对内层查询求值，产生一个列表，列出符合条件的作者的 ID。

在内层查询和外层查询的 where 子句中都可以包含多个条件。

使用连接时，查询表达如下：

```
select au_lname, au_fname
from authors, titleauthor
where state = "CA"
  and authors.au_id = titleauthor.au_id
  and royaltyper < 30
  and au_ord = 2
```

连接总可以表达为子查询。子查询通常可以表达为连接。

带 *not in* 的子查询

由关键字短语 not in 引入的子查询也返回一个 0 和更高值的列表。not in 表示“not = a *and* not = b *and* not = c”。

以下查询查找*尚未*出版过商业书籍的出版者的名称，与第 170 页上的“带 in 的子查询”中的示例相反：

```
select pub_name from publishers
where pub_id not in
   (select pub_id
    from titles
    where type = "business")
pub_name
----------------------------------------
Binnet & Hardley

(1 row affected)
```

除 not in 替代了 in 外，此查询与先前的查询相同。但是，不能将此语句转换为连接，“不等于”连接会查找出版过不属于商业书籍的*某*一书籍的出版者的名称。解释不基于等式的连接含义的难点，将在第 4 章“连接：从若干表中检索数据”中详细讨论。

使用包含 NULL 的 *not in* 的子查询

使用 not in 的子查询会为外层查询中的每一行返回一组值。如果外层查询中的值不在内层查询返回的结果集中，则 not in 等于 TRUE，外层查询将所考虑的记录放在结果中。

但是，如果内层查询返回的结果集中不包含匹配值，但包含 NULL，则 not in 返回 UNKNOWN。这是因为 NULL 代表“未知值”，而要判断出所查找的值是否在包含未知值的集中是不可能的。外层查询会放弃此行。例如：

```
select pub_name
    from publishers
    where $100.00 not in
        (select price
         from titles
         where titles.pub_id = publishers.pub_id)
pub_name
------
New Age Books

(1 row affected)
```

New Age Books 是唯一未出版过价格为 100 美元的书籍的出版者，结果集中不包括 Binnet & Handley 和 Algodata Infosystems，这是因为它们都各自出版了一本价格未定的书籍。

带 *exists* 的子查询

exists 关键字与子查询一起使用可以测试子查询的某些结果是否存在：

{where | having} [not] exists (*subquery*)

也就是说，外层查询的 where 子句会检测子查询返回的行的存在性。实际上，此类子查询并不产生任何数据，而只是返回一个 TRUE 或 FALSE 值。

例如，以下查询查找所有出版商业书籍的出版者的名称：

```
select pub_name
from publishers
where exists
   (select *
    from titles
    where pub_id = publishers.pub_id
    and type = "business")
```

```
pub_name
----------------------------------------
New Age Books
Algodata Infosystems

(2 rows affected)
```

为使此查询的解析概念化，应依次考虑每个出版者的名称。此值是否可使子查询至少返回一个行？ 也就是说，它是否可使存在测试的值为 TRUE？

在前面查询的结果中，第二个出版者的名称为 Algodata Infosystems，其标识号为 1389。titles 表中是否存在 pub_id 为 1389、且 type 为“business”的任何行？ 如果有，“Algodata Infosystems”应为所选值之一。为每个其它的出版者名称重复同样的过程。

由 exists 引入的子查询与其它子查询的不同之处在于：

- 关键字 exists 不优先于列名、常量或其它表达式。
- exists 子查询的值为 TRUE 或 FALSE，而不返回任何值。
- 子查询的选择列表通常包含星号 (*)。不需要指定列名，因为只需测试符合子查询中指定条件的行是否存在。此外，由 exists 引入的子查询的选择列表规则与标准选择列表规则相同。

exists 关键字非常重要，因为通常没有非子查询的表达方法可以替代它。实际上，由 exists 引入的子查询始终为相关子查询（参见第 178 页上的“使用相关子查询”）。

尽管不可将某些由 exists 表达的查询表示为其它任何方式，但却可用 exists 表示使用 in 或由 any 或 all 修饰的比较运算符的所有查询。以下是一些关于使用 exists 的语句及其等同替代方法的示例。

以下两种方法用于查找与出版者处于同一城市的作者：

```
select au_lname, au_fname
from authors
where city = any
   (select city
    from publishers)
select au_lname, au_fname
from authors
where exists
   (select *
    from publishers
    where authors.city = publishers.city)
au_lname          au_fname
```

```
--------------    --------------
Carson            Cheryl
Bennet            Abraham

(2 rows affected)
```

以下两个查询用于查找由位于以字母“B”开头的城市中的出版者所出版的书籍标题：

```
select title
from titles
where exists
   (select *
    from publishers
    where pub_id = titles.pub_id
    and city like "B%")
select title
from titles
where pub_id in
   (select pub_id
    from publishers
    where city like "B%")
title
--------------------------------------------------
You Can Combat Computer Stress!
Is Anger the Enemy?
Life Without Fear
Prolonged Data Deprivation: Four Case Studies
Emotional Security: A New Algorithm
The Busy Executive's Database Guide
Cooking with Computers: Surreptitious Balance
    Sheets
Straight Talk About Computers
But Is It User Friendly?
Secrets of Silicon Valley
Net Etiquette

(11 rows affected)
```

带 *not exists* 的子查询

not exists 与 exists 十分相似，不同之处在于：当子查询不返回行时所处的 where 子句得以满足。

例如，要查找不出版商业书籍的出版者的名称，查询应为：

```
select pub_name
from publishers
where not exists
   (select *
    from titles
    where pub_id = publishers.pub_id
    and type = "business")
pub_name
----------------------------------------
Binnet & Hardley

(1 row affected)
```

此查询查找无销售额的书籍的标题：

```
select title
from titles
where not exists
   (select title_id
    from salesdetail
    where title_id = titles.title_id)
title
------------------------------------------
The Psychology of Computer Cooking
Net Etiquette

(2 rows affected)
```

使用 *exists* 查找交集与差集

可使用由 exists 和 not exists 引入的子查询进行两种集合理论运算：交集和差集。两个集合的交集包含两个初始集合共有的所有元素。而差集则包含仅属于第一个集合的元素。

city 列上 authors 和 publishers 的交集为作者和出版者所同处的城市的集合。

```
select distinct city
from authors
where exists
```

```
  (select *
   from publishers
   where authors.city = publishers.city)
city
--------------------
Berkeley

(1 row affected)
```

city 列上 authors 和 publishers 的差集为有作者居住而没有出版者的城市的集合（即除 Berkeley 外的所有城市）。

```
select distinct city
from authors
where not exists
  (select *
   from publishers
   where authors.city = publishers.city)
city
--------------------
Gary
Covelo
Oakland
Lawrence
San Jose
Ann Arbor
Corvallis
Nashville
Palo Alto
Rockville
Vacaville
Menlo Park
Walnut Creek
San Francisco
Salt Lake City

(15 rows affected)
```

使用相关子查询

可通过执行一次子查询并将结果值替换进外部查询的 where 子句来对前面的多个查询求值；它们是无关子查询。在包含重复子查询（或**相关子查询**）的查询中，子查询的值取决于外部查询。子查询重复执行，即对外部查询选择每一行执行一次。

此例查找获得一本书 100% 版税的作者的名字：

```
select au_lname, au_fname
from authors
where 100 in
   (select royaltyper
    from titleauthor
    where au_id = authors.au_id)
au_lname        au_fname
--------------  ----------
White           Johnson
Green           Marjorie
Carson          Cheryl
Straight        Dick
Locksley        Chastity
Blotchet-Hall   Reginald
del Castillo    Innes
Panteley        Sylvia
Ringer          Albert

(9 rows affected)
```

与前面的大多数子查询不同，对此语句中子查询的解析不能独立于主查询。它需要一个 authors.au_id 值，而此值为一个变量 — 当 Adaptive Server 检查 authors 表的不同行时，它就会变化。

对前面查询的求值方法如下： Transact-SQL 通过替换内部查询中每行的值，来考虑 authors 表中的每一行，以得到将包含于结果中的内容。例如，假设 Transact-SQL 首先检查 Johnson White 的行。然后，authors.au_id 得到值“172-32-1176”，即为 Transact-SQL 为内部查询替换的值：

```
select royaltyper
from titleauthor
where au_id = "172-32-1176"
```

结果为 100，所以外部查询求值为：

```
select au_lname, au_fname
from authors
where 100 in (100)
```

由于 where 条件为真，所以 Johnson White 的行包括在结果中。如果对 Abraham Bennet 行执行同样的过程，会看到此行并未包括在结果中。

包含 Transact-SQL 外部连接的相关子查询

Adaptive Server 12.5 版对包含 Transact-SQL 外部连接的相关子查询的处理不同于较早版本的 Adaptive Server。下面是一个查询示例，它将相关变量用作 Transact-SQL 外部连接的外部成员：

```
select t2.b1, (select t2.b2 from t1 where t2.b1 *= t1.a1) from t2
```

Adaptive Server 的较早版本使用跟踪标志 298 来显示这些查询的错误消息。依据跟踪标志 298 是否打开，以及查询将相关变量用作外部连接的内部成员还是外部成员，Adaptive Server 显示了表 5-1 中描述的行为：

表 *5-1*：在较早版本 *Adaptive Server* 中的行为

查询类型	跟踪标志 298 关闭	跟踪标志 298 打开
相关变量是外部连接的内部成员	不允许： 产生错误消息 11013	没有错误
相关变量是外部连接的外部成员	没有错误	不允许： 产生错误消息 301

以后版本 Adaptive Server 中跟踪标志 298 的行为与之相反。因为，Adaptive Server 12.5 版在预处理阶段将 Transact-SQL 外部连接翻译为 ANSI 外部连接，当允许这样的查询运行时，会有产生不同结果的可能。在 298 跟踪标志打开时，允许运行包含 Transact-SQL 外部连接的相关子查询，这与 Sybase 的历史跟踪标志的用法一致。对于版本 12.5，跟踪标志 298 的行为如下：

表 *5-2*：在 *12.5* 版 *Adaptive Server* 中的行为

查询类型	跟踪标志 298 关闭	跟踪标志 298 打开
相关变量是外部连接的内部成员	不允许： 产生错误消息 11013	不允许： 产生错误消息 11013
相关变量是外部连接的外部成员	不允许：产生错误消息 11055	没有错误

Adaptive Server 已将错误消息 301 改为错误消息 11055，但消息文本保持相同。

带相关名的相关子查询

可使用相关子查询来查找由多个出版者出版的书籍的类型：

```
select distinct t1.type
from titles t1
where t1.type in
   (select t2.type
    from titles t2
    where t1.pub_id != t2.pub_id)
type
--------------------
business
psychology

(2 rows affected)
```

在以下查询中需要相关名，来区分出现 titles 表的两个任务： 此嵌套查询等同于自连接查询：

```
select distinct t1.type
from titles t1, titles t2
where t1.type = t2.type
and t1.pub_id != t2.pub_id
```

带比较运算符的相关子查询

表达式子查询可以是相关子查询。例如，要查找销售量低于该标题平均销售量的心理学书籍：

```
select s1.ord_num, s1.title_id, s1.qty
from salesdetail s1
where title_id like "PS%"
and s1.qty <
   (select avg(s2.qty)
    from salesdetail s2
    where s2.title_id = s1.title_id)
ord_num            title_id     qty
------------------ --------     ---
91-A-7             PS3333        90
91-A-7             PS2106        30
55-V-7             PS2106        31
AX-532-FED-452-2Z7 PS7777       125
BA71224            PS7777       200
NB-3.142           PS2091       200
NB-3.142           PS7777       250
```

```
NB-3.142            PS3333       345
ZD-123-DFG-752-9G8 PS3333       750
91-A-7              PS7777       180
356921              PS3333       200

(11 rows affected)
```

外部查询依次选择 sales 表（或“s1”）中的行。子查询计算外部查询进行选择时考虑的每次销售的平均值。对于 s1 中的每个可能值，Transact-SQL 会计算子查询的值，如果数量小于计算出的平均值，则将所考虑的记录放入结果中。

有时，相关子查询模仿 group by 语句。要查找价格高于同类书平均价格的书的标题，则查询为：

```
select t1.type, t1.title
from titles t1
where t1.price >
   (select avg(t2.price)
    from titles t2
    where t1.type = t2.type)
type         title
---------    ---------------------------------------
business     The Busy Executive's Database Guide
business     Straight Talk About Computers
mod_cook     Silicon Valley Gastronomic Treats
popular_comp But Is It User Friendly?
psychology   Computer Phobic and Non-Phobic
             Individuals: Behavior Variations
psychology   Prolonged Data Deprivation: Four Case
             Studies
trad_cook    Onions, Leeks, and Garlic: Cooking
             Secrets of the Mediterranean

(7 rows affected)
```

对于 t1 中的每个可能值，Transact-SQL 会计算子查询的值，如果该行的价格值高于计算出的平均值，则将该行包含进结果中。不必显式地按类型分组，因为计算平均价格的行由子查询中的 where 子句进行限制。

在 *having* 子句中的相关子查询

定量判定子查询可以是相关子查询。

此例中，外部查询 having 子句中的相关子查询用于查找最高预付款的高于给定组中平均预付款两倍的书的类型：

```
select t1.type
from titles t1
group by t1.type
having max(t1.advance) >= any
   (select 2 * avg(t2.advance)
   from titles t2
   where t1.type = t2.type)
type
----------
mod_cook

(1 row affected)
```

上述子查询对外部子查询定义的每个组求一次值，即对书籍的每个类型求一次值。

第 6 章

使用和创建数据类型

数据类型定义表中每列包含的信息的种类及其存储的方式。定义列时，可使用 Adaptive Server 系统数据类型，或可创建并使用用户定义的数据类型。

本章讨论如下内容：

主题	页
Transact-SQL 数据类型的工作原理	183
使用系统提供的数据类型	184
数据类型之间的转换	194
混合型算术和数据类型层次	195
创建用户定义的数据类型	197
获取有关数据类型的信息	200

Transact-SQL 数据类型的工作原理

在 Transact-SQL 中，数据类型指定表列、存储过程参数以及局部变量的信息类型、大小与存储格式。例如，int（整数）数据类型用于存储正负 2^{31} 范围内的整数，而 tinyint（微整数）数据类型只用于存储 0 到 255 之间的整数。

Adaptive Server 提供几种系统数据类型和两种用户定义的数据类型，timestamp 和 sysname。可使用 sp_addtype 建立基于系统数据类型的用户定义的数据类型。

声明列、局部变量或参数时，必须指定系统数据类型或用户定义的数据类型。下例使用系统数据类型 char、numeric 和 money 定义 create table 语句中的列：

```
create table sales_daily
    (stor_id char(4),
    ord_num numeric(10,0),
    ord_amt money)
```

下一个示例使用 bit 系统数据类型定义 declare 语句中的局部变量：

```
declare @switch bit
```

后面各章将更详细地描述如何使用本章中所述的数据类型声明列、局部变量和参数。使用 sp_help 可确定现有表中的列定义了何种数据类型。

使用系统提供的数据类型

表 6-1 列出了系统提供的用于不同类型信息的数据类型、Adaptive Server 可识别的同义词以及每种数据类型的范围和存储大小。尽管 Adaptive Server 允许系统数据类型以大写或小写方式输入，但它们是以小写字符输出的。大多数由 Adaptive Server 提供的数据类型并不是保留字，且可用于命名其它对象。

表 6-1：*Adaptive Server* 系统数据类型

数据类型种类	同义词	范围	存储的字节数
精确数值：整数			
tinyint		0 到 255（不允许使用负数。）	1
smallint		2^{15} -1 (32,767) 到 -2^{15} (-32,768)	2
int	integer	2^{31} -1 (2,147,483,647) 到 -2^{31} (-2,147,483,648	4
精确数值：小数			
numeric (p, s)		10^{38}-1 到 -10^{38}	2 到 17
decimal (p, s)	dec	10^{38}-1 到 -10^{38}	2 到 17
近似值			
float (precision)		与计算机有关	default precision <16 时为 4，default precision >=16 时为 8
double precision		与计算机有关	8
real		与计算机有关	4
货币			
smallmoney		214,748.3647 到 -214,748.3648	4
money		922.337.203.685.477,5807 到 -922,337,203,685,477.5808	8

数据类型种类	同义词	范围	存储的字节数
日期／时间			
smalldatetime		1900 年 1 月 1 日到 2079 年 6 月 6 日	4
datetime		1753 年 1 月 1 日到 9999 年 12 月 31 日	8
字符			
char(n)	character	页大小	n
varchar(n)	character varying，char varying	页大小	实际的条目长度
unichar	Unicode 字符	页大小	*n* * *@@unicharsize*（*@@unicharsize* 等于 2）
univarchar	Unicode 字符 varying，char varying	页大小	实际字符数 * *@@unicharsize*
nchar(n)	national character，national char	页大小	*n* * *@@ncharsize*
nvarchar(n)	nchar varying，national char varying，national character varying	页大小	*@@ncharsize* * 字符数
text		2 小于或等于 ^{31} -1 (2,147,483,647) 字节	未初始化时为 0： 初始化后是 2K 的倍数
二进制			
binary(n)		小于等于 255 个字节	*n*
varbinary(n)		小于等于 255 个字节	实际条目长度
image		2 小于或等于 ^{31} -1 (2,147,483,647) 字节	未初始化时为 0： 初始化后是 2K 的倍数
位			
bit		0 或 1	1（1 个字节最多容纳 8 bit 列）

精确数值类型： 整数

Adaptive Server 提供了三种数据类型用于存储整数，它们是 tinyint、smallint 和 int。这些类型是精确数值类型，它们在算术运算中保留其精确性。

基于要存储数字的预期大小在各种整数类型中选择。内部存储大小视数据类型的不同会有所变化。

精确数值类型：小数

对于包含小数点的数字，可使用精确数值类型 numeric 和 decimal。将存储在 numeric 和 decimal 列中的数据压缩到保留磁盘空间，并在算术运算后将精确度保留到最低有效位。除以下不同外，numeric 和 decimal 类型在所有方面都完全相同： 只有标度为 0 的 numeric 类型才能用于 IDENTITY 列。

精确数值类型可接受两种可选参数，precision 和 scale，这两个参数加有小括号 , 并以逗号分隔：

datatype [(*precision* [, *scale*])]

Adaptive Server 将每种**精度**和**标度**的组合都定义为一种不同的数据类型。例如，numeric(10,0) 和 numeric(5,0) 是两种不同的数据类型。精度和标度可确定可存储在 decimal 或 numeric 列中的值的范围：

- 精度指定了能够在该列中存储的最大小数位数。它包括小数点左右两侧的所有位数。可将精度指定为 1 至 38 位范围内的一个值，或者使用缺省的 18 位精度。
- 标度指定了能够存储到小数点右侧的最大位数。标度必须小于或等于精度。可将标度指定为 1 至 38 位范围内的一个值，或者使用缺省的 0 位标度。

标度为 0 的精确数值类型显示时没有小数点。不能输入超过列的精度或标度范围的值。

numeric 或 decimal 列的存储大小取决于其精度。1 或 2 位数列的最小存储要求为 2 字节。精度每增加 2 位，存储大小增加 1 字节，最多可增加 17 字节。

近似数值数据类型

数值类型 float、double precision 和 real 用于存储在算术运算中允许舍入的数值数据。

近似数值数据类型将按二进制小数存储的实数的近似表示存储为二进制 traction。只要显示、输出、在两主机间传送或在计算中使用了近似数值，数字都将丢失精度。请注意，isql 只显示小数点后的 6 位有效数字，并舍入其余数字。有关精度和近似数值数据类型的详细信息，参见*参考手册*。

可将近似数值类型用于包含各类数值的数据。除模运算（%）外，它们支持所有集合函数和所有算术运算。

real 和 double precision 类型是在操作系统提供的类型的基础上建立的。float 类型可接受括于小括号内的可选精度。精度为 1-15 的 float 列存储为 real；而具有更高精度的数值则存储为 double precision。这三种类型的范围和存储精度都与所使用的计算机相关。

货币数据类型

货币数据类型 money 和 smallmoney 用于存储货币数据。尽管 Adaptive Server 没有提供将一种货币转换为另一种货币的方法，但可将这些数据类型用于美元和其它十进制货币。可以将 money 和 smallmoney 数据用于除模运算外的所有算术运算以及所有集合函数。

money 和 smallmoney 都精确到货币单位的万分之一，但显示时它们将数值上舍入，只保留两位小数。缺省的输出格式是在每三位后放置一个逗号。

日期和时间数据类型

使用 datetime 和 smalldatetime 数据类型存储从 1753 年 1 月 1 日到 9999 年 12 月 31 日的日期和时间信息。超过此范围的日期必须输入、存储和处理为 char 或 varchar 值。

- datetime列可存放1753年1月1日至9999年12月31日之间的日期。在支持该时间间隔级别的平台上，datetime 值可精确到 1/300 秒。存储大小为 8 字节：4 字节用于存储自 1900 年 1 月 1 日这一基准日期以来的天数，另外 4 字节用于存储当天的时间。
- smalldatetime列可以保存从1900年1月1日到2079年6月6日的日期，其精确度可以精确到分。其存储大小为 4 字节： 其中 2 字节用于存储 1900 年 1 月 1 日后的天数，另 2 字节用于存储自午夜后的分钟数。

将日期和时间信息加以单引号或双引号。可用大写或小写字母输入，并可在数据分量间包含空格。Adaptive Server 可识别多种数据输入格式，详细信息可参见第 8 章“添加、更改和删除数据”。但 Adaptive Server 拒绝接受诸如 0 或 00/00/00 值，不会将它们识别为日期。

日期缺省的显式格式为“Apr 15 1987 10:23PM”。对于其它日期显示格式，可使用 convert 函数进行转换。也可使用内部日期函数对 datetime 值进行某些算术运算，尽管 Adaptive Server 可能会舍入或截断毫秒值。

字符数据类型

可用字符数据类型存储由加以单引号或双引号的字母、数字和符号组成的字符串。可使用 like 关键字搜索这些字符串中的特定字符和处理其内容的内部字符串函数。可使用 convert 函数将由数字组成的字符串转换为精确和近似数值数据类型，然后将其用于算术运算。

char(*n*) 数据类型存储固定长度的字符串，而 varchar(*n*) 数据类型以英语等单字节字符集存储可变长度的字符串。它们对应的国家字符，nchar(*n*) 和 nvarchar(*n*)，以日语等多字节字符集存储固定和可变长度的字符串。unichar 和 univarchar 数据类型存储常量大小的 Unicode 字符。可使用 n 或一个字符的缺省列长度指定最多字符数。对于大小超过页大小的字符串，可使用 text 数据类型。

表 *6-2*：字符数据类型

数据类型	存储
char(n)	固定长度数据，如社会保险号或邮政编码
varchar(n)	名称等长度上可能变化很大的数据
unichar	与 Char 同等的固定长度的 Unicode 数据
univarchar	与 varchar 同等的在长度上可能变化很大的 Unicode 数据
nchar(n)	多字节字符集中的固定长度数据
nvarchar(n)	多字节字符集中的可变长度数据
text	数据页已链接的列表中有多达 2,147,483,647 字节的可输出字符

除非设定 string_rtruncation on，Adaptive Server 将条目截断到指定列的长度而不加警告或出现错误信息。有关详细信息，参见*参考手册*中的 set 命令。空字符串“”或‘’将存储为单个空格而不是 NULL。因此，“abc”+“”+“def”等于“abc def”，而不等于“abcdef”。

固定和可变长度列的行为稍有不同：

- 固定长度列中的数据将被填补空白到列长度。对于 char 和 unichar 数据类型，存储大小为 *n* 字节，(unichar = n*@@unicharsize)；对于 nchar 数据类型，存储大小为平均国家字符长度的 *n* 倍 (*@@ncharsize*)。创建允许空值的 char、unichar 或 nchar 列时，Adaptive Server 将其转换为 varchar、univarchar 或 nvarchar 列，并对这些数据类型使用存储规则。
  但对 char 和 nchar 变量和参数就并非如此。
- 可变长度列中的数据将被去掉尾随空白，其存储大小即为数据的实际长度。对于 varchar 或 univarchar 列，它是字符数；对于 nvarchar 列，它是字符数乘以平均字符长度。可变长度字符数据要求的空间可能比固定长度数据要少，但访问它的速度要稍慢。

Unichar 数据类型

unichar 和 univarchar 数据类型支持 Adaptive Server 中 Unicode 的 UTF-16 编码。这些数据类型不依赖于 char 和 varchar 数据类型，但会镜像它们的行为。

例如，对 char 和 varchar 操作的内部函数也可对 unichar 和 univarchar 进行操作。但 unichar 和 univarchar 只存储 UTF-16 字符，且与 char 和 varchar 不同，它们与缺省字符集 ID 或缺省排序顺序 ID 没有关系。

要使用 unichar 和 univarchar 数据类型，必须将服务器缺省的字符集设定为 UTF-8。

每个 unichar/univarchar 字符要求两字节存储空间。unichar/univarchar 列的声明数是 16 位 Unicode 值的数。下例将创建一个带有 unichar 列的表，该列的 10 个 Unicode 值要求 20 字节的存储空间：

```
create table unitbl (unicol unichar(10))
```

正如 char/varchar 列的长度，unichar/univarchar 列的长度也要受数据页大小的限制。

请注意，Unicode 替代对将使用两个 16 位 Unicode 值的存储空间，也就是四个字节。声明用于存储 Unicode 替代对的列时，要注意这一特点。缺省情况下，Adaptive Server Enterprise 会正确处理替代，小心不将对拆分。处理截断 Unicode 数据的方式与处理 char 和 varchar 数据的方式类似。

在使用包含比较运算符、连接、子查询等 char 表达式的任何地方，都可使用 Unichar 表达式。但要注意，unichar 和 char 的混合型表达式将按 unichar 执行。可参与此类运算的 Unicode 值的数目不能超过最大的 unichar 字符串大小。

*规范化*过程会修改 Unicode 数据，因此在给定的抽象字符序列的数据库中，只有一个表示法。后接组合读音符的字符将被预组合形式所代替。性能可因此而得到显著优化。 缺省情况下，服务器假设所有 Unicode 数据都应规范化。

关系表达式

所有包括至少一个 unichar 或 univarchar 表达式的关系表达式将基于缺省 Unicode 排序顺序。 如果一个表达式是 unichar，而另一个是 varchar（nvarchar、char 或 nchar），则后者将被隐式转换为 unichar。

下列表达式最常在 where 子句中使用，在该子句中它可与逻辑运算符结合使用。

比较 Unicode 字符数据时，“less than”表示更接近缺省 Unicode 缺省排序顺序的开头，而“greater than”表示更接近其结尾。“Equality”表示 Unicode 缺省排序顺序对两值不加区别（尽管它们无需相同）。例如，必须将预先组合字符晔游　韧　谧帜毁后接 U+0302 的组合序列。如果启用了 Unicode 规范化功能（缺省设置），Unicode 数据将被自动规范化，服务器也因此不会看到未规范化的数据。

表 *6-3*：关系表达式

expr1 op_compare [any | all] (subquery)	使用将比较运算符和 expr2 作为子查询的 any 或 all 会隐式调用 min 或 max。例如 "expr1> any expr2" 实际上就是 "expr1> min(expr2)"。
expr1 [not] in (expression list) expr1 [not] in (subquery)	in 运算符检查与 expr2 中的每个元素是否相等，这些元素可以是常量列表或子查询结果。
expr1 [not] between expr2 and expr3	between 运算符用于指定范围。 它实际上是“expr1 >= expr2 和 expr1<= expr3”的简写形式。
expr1 [not] like "match_string" [escape "esc_char"]	like 运算符用于指定匹配的模式。与 Unicode 数据匹配的模式的语义和与规则字符数据匹配的模式的语义相同。如果 expr1 是 unichar 列名，则 "match_string" 可能是 unichar 字符串或 varchar 字符串。在后一种情况下，varchar 和 unichar 间将发生隐式转换

连接运算符

连接运算符出现的方式与比较运算符相同。实际上，连接中可使用任何比较运算符。 至少包括一个 unichar 表达式的表达式将基于缺省的 Unicode 排序顺序。如果一个表达式是 unichar 类型，而另一个表达式是 varchar 类型（nvarchar、char 或 nchar），则后者将隐式转换为 unichar。

union 运算符

union 运算符与 unichar 数据的运算方式和它与 varchar 数据的运算方式非常相似。各个查询中的相应列必须可隐式转换为 unichar，或必须使用显式转换。

子句和修饰符

unichar 和 univarchar 列用于 group by 和 order by 子句时，将根据缺省 Unicode 排序顺序判断等同性。 使用 distinct 修饰符时，也是如此。

text 数据类型

text 数据类型将多达 2,147,483,647 字节的可输出字符存储在各数据页的链接列表中。每页最多可存储 1800 字节的数据。

为节省存储空间，可将 text 列定义为 NULL。用非空值的 insert 或 update 初始化 text 列时，Adaptive Server 会指派一个文本指针，并分配一个完整的 2K 数据页用于容纳值。

不能将 text 数据类型作以下用途：

- 用于存储过程的参数，作为传递给这些参数的值，或用于局部变量
- 用于远程过程调用 (RPC) 的参数
- 在 order by、compute、group by 或 union 子句中
- 用于索引
- 用于子查询或连接
- 用于 where 子句，除非带有关键字 like
- 与 + 并置运算符一起使用
- 用于触发器的 if update 子句

如果使用的是与“组件集成服务“连接的数据库，则在处理 text 数据类型的方式上会有些不同。有关详细信息，参见*组件集成服务用户指南*。

有关 text 数据类型的详细信息，参见第 304 页上的“改变 text 和 image 数据”。

二进制数据类型

二进制数据类型将原始二进制数据，如图片，存储在类似十六进制的符号中。二进制数据以字符“0x”开头，并可包括数字与 A 至 F 的大小写字母的任意组合。binary 和 varbinary 数据中“0x”后的两个数字表示数字的类型：“00”表示正数，“01”表示负数。

如果输入数值不包括“0x”，Adaptive Server 会假定该数值为 ASCII 值并将其转换。

注意：Adaptive Server 对不同平台的二进制类型采取不同的处理方式。对于真十六进制数据，可使用 hextoint 和 inttohex 函数。参见第 10 章“使用查询中的内部函数”。

使用 binary(n) 和 varbinary(n) 数据类型存储达 255 字节长度的数据。每个存储字节可容纳 2 个二进制位。用 *n* 指定列的长度，或使用 1 字节的缺省长度。如果输入了一个长于 *n* 的数值，Adaptive Server 会在不显示警告或出错信息的情况下将该条目截断为指定的长度。

- 对于其中所有条目的长度预期会基本相同的数据，可使用固定长度二进制类型 binary(n)。由于 binary 列中的条目会被用零填充到列长度，因此它们可能会要求占用比 varbinary 列中的条目更大的存储空间，但访问它们的速度会稍快。
- 对于预期长度会变化很大的数据，可使用可变长度二进制类型 varbinary(n)。存储大小是输入的数据值的实际大小，不是列长度。尾随零将被截断。

创建允许空值的 binary 列时，Adaptive Server 会将其转换为 varbinary 列，并使用该数据类型的存储规则。

可用 like 关键字搜索二进制字符串，并用内部字符串函数对其进行运算。由于输入某个特定值的确切格式取决于所使用的硬件，因此*在不同平台上进行涉及二进制数据的计算可能会产生不同的结果。*

image 数据类型

使用 image 数据类型将较大的二进制数据块存储到外部数据页上。image 列可在与表的其它数据存储分离的数据页的链接列表上存储多达 2,147,483,647 字节的数据。

用非空值的 insert 或 update 初始化 image 列时，Adaptive Server 会指派一个文本指针，并分配一个完整的 2K 数据页用于容纳值。每页最多可存储 1800 字节。

为了节省存储空间，可将 image 列定义为 NULL。要添加 image 数据，而又不想在事务日志中记录大块的二进制数据，可使用 writetext。有关 writetext 的详细信息，参见*参考手册*。

不能将 image 数据类型作以下用途：

- 用于存储过程的参数，作为传递给这些参数的值，或用于局部变量
- 用于远程过程调用 (RPC) 的参数
- 在 order by、compute、group by 或 union 子句中
- 用于索引
- 用于子查询或连接
- 用于 where 子句，除非带有关键字 like
- 与 + 并置运算符一起使用
- 用于触发器的 if update 子句

如果使用的是与“组件集成服务“连接的数据库，则在处理 image 数据类型的方式上会有些不同。有关详细信息，参见*组件集成服务用户指南*。

有关 image 数据类型的详细信息，参见第 304 页上的“改变 text 和 image 数据”。

bit 数据类型

将 bit 列用于真 / 假或是 / 否类型的数据。bit 列可存放 0 或 1。可接受 0 或 1 以外的其它整数值，但始终会将这些值解释为 1。存储大小为 1 字节。表中的多个 bit 数据类型被收集到字节中。例如，7-bit 列将占用 1 个字节；而 9-bit 列则要占用 2 个字节。

数据类型为 bit 的列不能为 NULL，且其中不能有索引。syscolumns 系统表中的 status 列用于指示 bit 列的唯一偏移位置。

timestamp 数据类型

对于可在 Open Client™ 的 DB-Library 应用程序中浏览到的表中的列，用户定义的 timestamp 数据类型是必需的。

每次插入或更新包含 timestamp 列的行时，timestamp 列将自动更新。一个表只能有一列 timestamp 数据类型。命名为 timestamp 的列将自动具有系统数据类型 timestamp。它的定义为：

```
varbinary(8) "NULL"
```

因为 timestamp 是用户定义的数据类型，所以不能用它定义其它用户定义的数据类型。必须全部用小写字母将其输入为“timestamp”。

sysname 数据类型

sysname 是一种在系统表中使用的用户定义的数据类型。它的定义为：

```
varchar(30) "NULL"
```

不能使用 sysname 数据类型创建列。但可以用 sysname 的基类型创建用户定义的数据类型。然后才能使用此**用户定义的数据类型**创建列。有关用户定义的数据类型的详细信息，参见第 197 页上的“创建用户定义的数据类型”。

数据类型之间的转换

Adaptive Server 可自动处理许多从一种数据类型到另一种数据类型的转换。这些转换称作隐式转换。可用 convert、inttohex 和 hextoint 函数显式请求其它转换。其它转换，无论是显式转换还是自动转换，都因数据类型间不兼容而无法完成。

例如，为进行比较，Adaptive Server 将把 char 表达式自动转换为 datetime，如果可将它们作为 datetime 值解释。但为方便显示，必须使用 convert 函数将 char 转换为 int。同样，如果想要 Adaptive Server 将其视作字符数据，以便将 like 关键字与其一同使用，则必须对整数数据使用 convert。

convert 函数的语法是：

convert (*datatype*, *expression*, [*style*])

在下例中，为显示以数字 2 开头的所有销售额，convert 使用 char 数据类型显示 total_sales 列：

```
select title, total_sales
from titles
where convert (char(20), total_sales) like "2%"
```

可选的 style 参数用于将 datetime 值转换为 char 或 varchar 数据类型，以获得多种数据显示格式。

有关 convert、inttohex 和 hextoint 函数的详细信息，参见第 10 章“使用查询中的内部函数”。

混合型算术和数据类型层次

对具有不同数据类型的值执行算术运算时，Adaptive Server 必须确定结果的数据类型，在某些情况下，还要确定结果的长度和精度。

每个系统数据类型都有**数据类型层次**，它存储在 systypes 系统表中。用户定义的数据类型继承了它所基于的系统类型的层次。

下列查询按照层次来排列数据库中的数据类型。除了下面显示的信息之外，查询结果还将包括有关数据库中任意用户定义的数据类型的信息：

```
select name, hierarchy
from systypes
order by hierarchy
name                            hierarchy
---------------------------     ---------
floatn                                  1
float                                   2
datetimn                                3
datetime                                4
real                                    5
numericn                                6
numeric                                 7
decimaln                                8
decimal                                 9
moneyn                                 10
money                                  11
smallmoney                             12
smalldatetime                          13
intn                                   14
```

```
int                                     15
smallint                                16
tinyint                                 17
bit                                     18
univarchar                              19
unichar                                 20
reserved                                21
varchar                                 22
sysname                                 22
nvarchar                                22
char                                    23
nchar                                   23
varbinary                               24
timestamp                               24
binary                                  25
text                                    26
image                                   27

(30 rows affected)
```

数据类型层次决定了使用不同数据类型的值所进行的计算的结果。最终所得的值将被指派给最靠近列表顶部的数据类型。

在下例中，sales 表的 qty 同 roysched 表的 royalty 相乘。qty 是 smallint，其层次为 16； royalty 是 int，其层次为 15。因此，结果的数据类型是 int。

smallint(qty) * int(royalty) = int

使用 *money* 数据类型

如果要组合 money 和文字或变量，且需要 money 类型的结果，可使用 money 文字或变量：

```
create table mytable
(moneycol money,)
insert into mytable values ($10.00)
select moneycol * $2.5 from mytable
```

如果要组合货币与列值中的 float 或 numeric 数据类型，可使用 convert 函数：

```
select convert (money, moneycol * percentcol)
    from debits, interest
drop table mytable
```

确定精度和标度

对于 numeric 和 decimal 数据类型，精度和标度的每一种组合都是一个不同的 Adaptive Server 数据类型。如果对两个 numeric 或 decimal 值执行算术运算，其中 n1 精度为 p1，标度为 s1，n2 精度为 p2，标度为 s2，则 Adaptive Server 所确定的结果的精度和标度将如表 6-4 中所示：

表 *6-4*：算术运算后的精度和标度

运算	精度	标度
n1 + n2	max(s1, s2) + max(p1 -s1, p2 - s2) + 1	max(s1, s2)
n1 - n2	max(s1, s2) + max(p1 -s1, p2 - s2) + 1	max(s1, s2)
n1 * n2	s1 + s2 + (p1 - s1) + (p2 - s2) + 1	s1 + s2
n1 / n2	max(s1 + p2 + 1, 6) + p1 - s1 + p2	max(s1 + p2 -s2 + 1, 6)

创建用户定义的数据类型

可使用 SQL 的 Transact-SQL 增强功能命名并设计自己的数据类型，作为对系统数据类型的补充。用户定义的数据类型是根据系统数据类型定义的。可以给经常使用的数据类型定义命名。这使自定义适用于列的数据类型变得更加容易。

注意：要在不止一个数据库中使用用户定义的数据类型，可在 model 数据库中创建该数据类型。该用户定义的数据类型就会对所有新创建的数据库生效。

一旦定义了数据类型，就可以在数据库的任一列中使用。例如，可将 tid 用作几个 pubs2 表中列的数据类型： titles.title_id、titleauthor.title_id、sales.title_id 和 roysched.title_id。

用户定义的数据类型的优点是可将规则和缺省值绑定到它们上面，以供在多个表中使用。关于该主题的详细信息，参见第 12 章“为数据定义缺省值和规则”。

使用 sp_addtype 创建用户数据类型。它将创建的用户数据类型名、创建它的 Adaptive Server 提供的数据类型及可选的 NULL，NOT NULL 或 IDENTITY 规范作为参数。

可使用除 timestamp 外的任何系统数据类型建立用户定义的数据类型。用户定义的数据类型可与其所基于的系统数据类型拥有相同的数据类型层次。与 Adaptive Server 提供的数据类型不同，用户定义的数据类型名区分大小写。

sp_addtype 的语法为：

sp_addtype *datatypename*,
phystype [(length) | (precision [, scale])]
[, "identity" |*nulltype*]

定义 tid 的方法如下：

```
sp_addtype tid, "char(6)", "not null"
```

如果参数包含空白或某种形式的标点符号，或如果参数是 null（例如，identity 或 sp_helpgroup）以外的关键字，则必须为其加以单引号或双引号。本例中，因为有括号，所以需为 char(6) 加引号，但因为有空白，所以需为“not null”加引号。不需要为 tid 加引号。

指定长度、精度和标度

建立基于某种 Adaptive Server 数据类型的用户定义的数据类型时，必须指定额外的参数：

- char、nchar、varchar、nvarchar、binary 以及 varbinary 数据类型要求将长度加以括号。如果未提供长度，Adaptive Server 将假设其为缺省的 1 个字符的长度。
- float数据类型要求将精度加以括号。如果未提供精度，Adaptive Server 将使用所用平台的缺省精度。
- numeric 和 decimal 数据类型要求精度和标度应加以括号，并用逗号分隔。如果未提供精度和标度，Adaptive Server 将使用缺省的精度 18 和标度 0。

在 create table 语句中加入用户定义的数据类型时，不能更改长度、精度或标度规范。

指定空值类型

空值类型决定用户定义数据类型如何处理空值。可创建带有“null”、“NULL”、“nonull”、“NONULL”、“not null”或“NOT NULL”空值类型的用户自定义的数据类型。但 bit 和 IDENTITY 类型不允许空值。

如果省略了空值类型，Adaptive Server 将使用为数据库定义的空值模式（缺省为“NOT NULL”）。为与 SQL 标准兼容，可使用 sp_dboption 将 allow nulls by default 选项设置为真。

在 create table 语句中加入用户定义的数据类型时，可覆盖空值类型。

将规则和缺省值与用户定义的数据类型相关联

创建用户定义的数据类型后，可使用 sp_bindrule 和 sp_bindefault 将规则和缺省值与数据类型相关联。使用 sp_help 输出列有规则、缺省值和其它与数据类型相关联的信息的报告。

在第 12 章“为数据定义缺省值和规则”中对规则和缺省值作了阐述。有关系统过程及其语法的完整信息，参见*参考手册*。

使用 IDENTITY 属性创建用户定义的数据类型

要用 IDENTITY 属性创建用户定义的数据类型，可使用 sp_addtype。新类型必须基于标度为“0”的物理类型 numeric：

```
sp_addtype typename, "numeric (precision, 0)",
"identity"
```

下例用 IDENTITY 属性创建了一个用户定义的数据类型 IdentType：

```
sp_addtype IdentType, "numeric(4,0)", "identity"
```

从 IDENTITY 类型创建列时，可指定 create 或 alter table 语句中的 identity 或 not null，或者两者都不指定。该列自动继承 IDENTITY 属性。

以下是从 IdentType 用户定义的数据类型创建 IDENTITY 列的三种不同方法：

```
create table new_table (id_col IdentType)
drop table new_table
create table new_table (id_col IdentType identity)
drop table new_table
create table new_table (id_col IdentType not null)
drop table new_table
```

注意：如果试图从 IDENTITY 类型创建允许空值的列，则 create table 或 alter table 语句将失效。

从其它用户定义的数据类型创建 IDENTITY 列

可从不具有 IDENTITY 属性的用户定义的数据类型创建 IDENTITY 列。用户定义的类型必须具有标度为“0”的 numeric 物理数据类型，且必须定义为 not null。

删除用户定义的数据类型

要删除用户定义的数据类型，可执行 sp_droptype：

sp_droptype *typename*

注意：不能删除使用中的任何表中的数据类型。

获取有关数据类型的信息

使用 sp_help 显示有关系统数据类型或用户定义的数据类型的属性的信息。报告指示了从中创建数据类型的基本类型，不管它是否允许空值、允许任何绑定到数据类型的规则和缺省值的名称，也不管它是否具有 IDENTITY 属性。

下例显示了有关 money 系统数据类型和 tid 用户定义的数据类型的信息：

```
sp_help money
Type_name  Storage_type Length Prec  Scale
---------- ------------ ------ ----- -----
money      money             8 NULL  NULL
Nulls   Default_name  Rule_name   Identity
-----   ------------  ---------   --------
    1   NULL          NULL            NULL

(return status = 0)
sp_help tid
Type_name  Storage_type Length Prec  Scale
---------- ------------ ------ ----- -----
tid        varchar           6 NULL  NULL
Nulls   Default_name  Rule_name   Identity
-----   ------------  ---------   --------
    0   NULL          NULL               0

(return status = 0)
```

第 7 章

创建数据库和表

本章阐述如何创建数据库和表。

本章包括以下部分：

主题	页
何为数据库和表？	203
使用和创建数据库	206
变更数据库的大小	210
删除数据库	211
创建表	211
管理表中的标识间隔	224
为表定义完整性约束	232
如何设计和创建表	241
从查询结果中创建新表：select into	244
改变现有表	250
删除表	268
给用户分配权限	269
获得有关数据库和表的信息	270

有关管理数据库的信息，参见*系统管理指南*。

何为数据库和表？

数据库在一组数据库对象中存储信息（数据），如互相关联的表。表是拥有含单个数据项的相关列的行集合。创建数据库和表时应定义数据的组织方式。这一过程称为*数据定义*。

Adaptive Server 数据库对象包括：

- 表
- 规则
- 缺省值

- 存储过程
- 触发器
- 视图
- 参照完整性约束
- 检查完整性约束

本章讲述数据库和表的创建、修改和删除，包括完整性约束。

列和**数据类型**用于定义表中包括的数据类型，本章也将就此展开讨论。索引用于说明表中数据的组织方式。它们是 Adaptive Server 所不考虑的数据库对象，并不列在 sysobjects 中。第 11 章“创建表的索引”中将详细讨论索引。

强制实现数据库的数据完整性

数据完整性指数据库中数据的正确性与完全性。为强制实现数据的完整性，可约束或限制用户可在数据库中插入、删除或更新的数据值。例如，pubs2 和 pubs3 数据库中的数据完整性要求：titles 表中的书名必须对应 publishers 表的一个出版者。不能将无有效出版者的书插入 titles，因为它会破坏 pubs2 或 pubs3 的数据完整性。

Transact-SQL 为在数据库中强制实现完整性提供了几种机制，如规则、缺省值、索引和触发器。通过这些机制可以维护以下类型的数据完整性：

- *需求* – 要求表列的每一行都必须包含一个有效值；不允许存在空值。可用 create table 语句限制列的空值。
- *检查* 或 *有效性* — 限制或限定插入到表列中的数据值。可使用触发器或规则来强制实现此类完整性。
- *唯一性* — 不能有两个表行中的一个或多个表列具有相同的非空值。可用索引来强制实现此类完整性。
- *参照* — 插入到表列中的数据必须在其它表列或相同表的其它列中已拥有匹配数据。一个单表最多可拥有 192 个参照。

数据库中数据值的一致性也属于数据完整性（如第 18 章“事务：维护数据一致性和恢复”所述）。

除使用规则、缺省值、索引和触发器外，Transact-SQL 还提供了一系列的**完整性约束**作为 create table 语句的一部分，用以强制实现 SQL 标准规定的数据完整性。这些完整性约束将在本章的以后部分加以阐述。

数据库中的权限

是否可以创建和删除数据库和数据库对象，取决于您的权限或特权。通常，系统管理员或数据库所有者会依据您的工作和所需要的功能为您设置权限。在特定的系统或数据库中，每个用户的这些权限可以不同。

可通过执行如下命令确定您的权限：

```
sp_helprotect user_name
```

user_name 为您的 Adaptive Server 登录名。

为使数据库对象试验尽可能便捷，pubs2 和 pubs3 数据库在其 sysusers 系统表中有一个“**guest**”用户名。创建 pubs2 和 pubs3 的脚本赋予“guest”多种权限。

“guest”机制意味着**登录**到 Adaptive Server（即列在 master..syslogins 中）的任何人都有权访问 pubs2 和 pub3，有权创建和删除对象，如表、索引、缺省值、规则、过程等等。“guest”用户名也允许使用某些存储过程、创建用户定义的数据类型、查询数据库以及修改其中的数据。

要使用 pubs2 或 pubs3 数据库，可发布 use 命令。Adaptive Server 会检查 pubs2..sysusers 或 pubs3..sysusers 中是否列有您的名字。如果没有，则认为您为访客，对您没有任何操作。如果列在 pubs2 或 pubs3 的 sysusers 表中，则 Adaptive Server 承认您的身份，可能给您一些不同于“guest”身份的权限。

注意：本章中的所有示例都假定您被看作“guest”。

大多数用户都可通过先前描述的“guest”机制查看 master 数据库中的系统表。未经 master 数据库中的名称验证的用户允许进入，但被视为“guest”用户。“guest”用户被添加到脚本的 master 数据库中（该脚本在安装时创建了 master 数据库）。

数据库所有者“dbo”可使用 sp_adduser 向任何用户数据库添加“guest”用户。系统管理员可自动成为所用任意数据库的数据库所有者。有关详细信息，参见*系统管理指南*和*参考手册*。

使用和创建数据库

数据库是相关表和其它数据库对象（视图、索引等）的集合。

安装 Adaptive Server 时，它包含以下**系统数据库：**

- master 数据库整体上控制用户数据库和 Adaptive Server 的操作。
- sybsystemprocs 数据库包含系统存储过程。
- sybsystemdb 数据库包含有关分布式事务的信息。
- 临时数据库 tempdb 存储临时对象，包括用名称前缀“tempdb..”创建的临时表。
- model 数据库由 Adaptive Server 使用，作为创建新用户数据库的模板。

此外，系统管理员可安装以下可选数据库：

- pubs2 — 包含代表出版业务的数据的样本数据库。可使用此数据库测试服务器的连接并了解 Transact-SQL。Adaptive Server 文档中的许多示例都要查询 pubs2 数据库。
- pubs3 — pubs2 的一个使用参照完整性示例的版本。pubs3 有一个使用自参照列的表 store_employees。pubs3 的 sales 表中还包括 IDENTITY 列。此外，其主表中的主键使用非聚簇唯一索引，titles 表有一个 numeric 数据库的示例。
- interpubs — 与 pubs2 类似，但包含法文和德文数据。
- jpubs — 与 pubs2 类似，但包含日文数据。安装日文模块后，即可使用。

这些可选数据库为用户数据库。用户的全部数据都存储在用户数据库中。Adaptive Server 通过系统表来管理每一个数据库。master 数据库和其它数据库中的**数据字典**表均被视为系统表。

选择数据库：*use*

使用 use 命令可访问现有数据库。此命令的语法为：

```
use database_name
```

例如，要访问 pubs2 数据库，应输入：

```
use pubs2
```

只有当您是 pubs2 中的已知用户时，此命令才允许您访问 pubs2 数据库。否则，会出现错误消息。

在登录 Adaptive Server 时，系统可能会将用户自动连接到 master 数据库，因此如果要使用另一数据库，应发布 use 命令。您或系统管理员都可使用 sp_modifylogin 命令来更改最初连接到的数据库。但只有系统管理员才可以更改另一用户的缺省数据库。

创建用户数据库：*create database*

如果系统管理员授予您使用 create database 命令的权限，则您可创建新数据库。创建新数据库时，必须正在使用 master 数据库。在许多公司中，由系统管理员创建全部的数据库。数据库的创建者即为其所有者。为您创建数据库的另一用户可通过 sp_changedbowner 转交所有权。

数据库所有者负责赋予用户访问数据库的权限，以及负责授予或撤消某些其它权限。在某些组织中，数据库所有者也有责任维护数据库的定期备份，并在系统出现故障时重新装载数据库。数据库所有者可使用 setuser 命令，来临时获得其它任何用户对数据库的权限。

因为每个数据库都被分配相当大的空间，因此即使它只包含少量数据，您也可能无权使用 create database 命令。

create database 的最简单形式如下：

```
create database database_name
```

要创建称为 newpubs 的新数据库，首先应确认您正在使用 master 数据库而非 pubs2，然后键入命令：

```
use master
create database newpubs
drop database newpubs
use pubs2
```

Adaptive Server 上的数据库名称必须是唯一的，必须遵循第 8 页上的“标识符”中描述的标识符规则。Adaptive Server 最多可管理 32,767 个数据库。一次只能创建一个数据库。任何数据库的最大段数均为 32。

Adaptive Server 创建一个新的数据库，作为 model 数据库的副本，其中包含属于每个用户数据库的系统表。

新数据库的创建记录在 master 数据库表 sysdatabases 和 sysusages 中。

create database 的完整语法为：

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

本章介绍除 with override 外的所有 create database 选项。有关 with override 的详细信息，参见*系统管理指南*。

on 子句

可使用可选 on 子句指定数据库的存储位置及为其分配的空间大小（以兆字节为单位）。如果使用关键字缺省值，则数据库被分配给一个可用的数据库设备，该设备位于 master 数据库表 sysdevices 中指示的缺省数据库设备池中。使用 sp_helpdevice 可查看缺省列表中的设备。

注意：系统管理员可以根据性能统计信息和其它考虑因素进行一些存储分配。创建数据库之前，应向系统管理员核实分配情况。

要为即将存储在此缺省位置处的数据库指定 5MB 大小，可使用 on default = *size*：

```
use master
create database newpubs
on default = 5
drop database newpubs
use pubs2
```

要为数据库指定其它位置，应给出要存储此数据库的数据库设备的逻辑名。一个数据库可以存储于多个数据库设备上，每个设备都有不同的空间量。

本例创建 newpubs 数据库，并在 pubsdata 上为其分配 3MB 空间，在 newdata 上为其分配 2MB 空间：

```
create database newpubs
on pubsdata = 3, newdata = 2
```

如果省略 on 子句和大小，则数据库创建在 sysdevices 指示的缺省数据库设备池中，并占用 2MB 空间。

数据库分配的大小介于 2MB 到 2^{23}MB 之间。

log on 子句

除非要创建非常小且不很重要的数据库，否则应务必对 create database 使用 log on database_device 扩展。这样可将事务日志放到单独的数据库设备上。之所以要将日志放到单独的设备上，是因为以下几种原因：

- 允许使用 dump transaction 命令而不是 dump database 命令，从而节省时间和空间。
- 允许建立固定大小的日志，防止它与其它数据库活动争用空间。

之所以要将日志放到数据表的单独*物理*设备上，是因为以下原因：

- 提高性能。
- 确保在硬盘出现故障时能完全恢复。

下列命令将 newpubs 的日志放置到逻辑设备 pubslog 上，大小为 1MB：

```
create database newpubs
on pubsdata = 3, newdata = 2
log on pubslog = 1
```

注意：使用 log on 扩展时，将数据库事务日志放置到名为“logsegment”的段上。要为现有日志添加更多的空间，应使用 alter database 命令，在某些情况下，使用 sp_extendsegment。有关详细信息，参见*参考手册*或*系统管理指南*。

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，一般为日志分配的空间应为数据库空间的 10-25%。

for load 选项

可选 for load 子句调用只可用于装载数据库转储的 create database 的最新版本。for load 选项用于从介质故障中恢复，或用于将数据库从一台机器移动到另一台机器。有关详细信息，参见*系统管理指南*。

变更数据库的大小

如果数据库的分配存储空间已全部填满，则不能再向其添加新数据或进行更新。当然，现有数据会始终保留。如果为数据库分配的空间过小，则数据库所有者可用 alter database 命令增加空间。缺省情况下，alter database 权限授予给数据库所有者，且不能移交。
要使用 alter database，必须正在使用 master 数据库。

缺省情况下，增加缺省空间池中的 2MB 空间。本语句为缺省数据库设备上的 newpubs 增加 2MB 空间：

```
alter database newpubs
```

完整的 alter database 语法允许将数据库扩展指定的兆字节数（最小 1MB），并允许指定增加存储空间的位置：

alter database *database_name*
    [on {default | *database_device* } [= *size*]
        [, *database_device* [= *size*]]...]
    [log on { default | *database_device* } [= *size*]
        [, *database_device* [= *size*]]...]
    [with override]
    [for load]

alter database 命令中的 on 子句与 create database 中的 on 子句相似。for load 子句与 create database 中的 for load 子句相似，并只能用于使用 for load 子句创建的数据库。

要在数据库设备 pubsdata 上为 newpubs 增加 2MB 空间，在数据库设备 newdata 上为其增加 3MB 空间，应键入：

```
alter database newpubs
on pubsdata = 2, newdata = 3
```

当使用 alter database 在已由数据库使用的设备上分配更多空间时，该设备上所有已有段均可使用增加的空间片段。已映射到现有段的所有对象现在均可扩展到增加的空间内。任何数据库的最大段数均为 32。

当使用 alter database 在未由数据库使用的设备上分配空间时，system 和 default 段被映射到新设备。要更改此段映射，应使用 sp_dropsegment 从设备中删除不想要的段。

注意：使用 sp_extendsegment 会自动取消系统与缺省段的映射。

有关 with override 的信息，参见*系统管理指南*。

删除数据库

drop database 命令用于删除数据库。drop database 可从 Adaptive Server 删除数据库及其所有内容，释放为其分配的所有存储空间，并从 master 数据库删除对它的参照。

此命令的语法为：

drop database *database_name* [, *database_name*]...

不能删除使用中的数据库，即由其他用户打开进行读写操作的数据库。

如前述，可用一个命令删除多个数据库。例如：

```
drop database newpubs, newdb
```

不能用 drop database 删除已损坏的数据库。而应使用 dbcc dbrepair 命令。

创建表

创建表时，应为各列命名并为每个列提供一个数据类型。也可指定特定行能否存在空值，或指定表中各列的完整性约束。每个数据库可有 2,000,000,000 个表。

创建表示例

如果要试验这些示例，应使用先前部分中创建的 newpubs 数据库。否则，这些更改会影响其它数据库，如 pubs2 或 pubs3。

|create|table| 的最简单形式如下：

```
create table table_name
(column_name datatype)
```

例如，要创建带有 *some_name* 列（固定长度 11 字节）的 names 表，应输入：

```
create table names
(some_name char(11))
drop table names
```

最多可以定义 250 列。如果使用 set quoted_identifier on 命令，则表名和列名均可为分隔标识符。否则，它们必须遵循第 8 页上的“标识符”中描述的标识符规则。在同一个表中，列名必须唯一，但同一数据库的不同表中可使用相同的列名。

每个列都必须有数据类型。上示中列名后的单词“char”是指该列的数据类型 – 该列将包含的值的类型。数据类型在第 6 章“使用和创建数据类型”中讨论。

数据类型后小括号中的数字给出可在列中存储的最大字节数。某些数据类型的最大长度可由用户指定。而其它数据类型则使用系统定义的长度。

务必用小括号将列名列表括起，并在每个列定义后以逗号分隔。最后一个列定义后不需要加逗号。

注意：如果缺省值是 create table 语句的一部分，则在该缺省值中不能使用变量。

选择表名

create table 命令可在当前打开的数据库中建立新表。对于每个用户，表名必须是唯一的。

创建临时表有两种途径，一种是在 create table 语句中的表名前加井号 (#)，另一种是指定表名前缀“tempdb..”。有关详细信息，参见第 220 页上的“使用临时表”。

您可使用任何由您创建且未限定名称的表或其它对象。也可使用由数据库所有者创建且未限定名称的对象（只要有相应的权限）。这些规则适用于所有用户，包括系统管理员和数据库所有者。

不同用户可创建相同名称的表。例如，名为“jonah”的用户和名为“sally”的用户可各自创建一个名为 info 的表。对这两个表都拥有权限的用户必须将它们分别限定为 jonah.info 和 sally.info。Sally 必须将对 Jonah 的表的参照限定为 jonah.info，而对自己的表只需使用 info。

create table 语法

create table 的语法为：

```
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,
            identity_gap = num_values } ]
    [on segment_name ]
    [ [ external table ] at pathname ]
```

注意：有关此语法的完整讨论，参见*参考手册*中的 create table。

create table 语句：

- 定义表中的每一列。
- 提供列名和数据类型，指定每个列对空值的处理方式。
- 指定哪列（如果有）有 IDENTITY 属性。
- 定义列级完整性约束和表级完整性约束。每个表定义的每一列和每个表可有多个约束。

例如，pubs2 数据库中 titles 表的 create table 语句为：

```
create table titles
(title_id tid,
title varchar(80) not null,
type char(12),
pub_id char(4) null,
price money null,
advance money null,
royalty int null,
total_sales int null,
notes varchar(200) null,
pubdate datetime,
contract bit not null)
```

以下几节讲述表定义的组成部分： 系统提供的数据类型、用户定义的数据类型、空类型和 IDENTITY 列。

注意：对 create table 的 on segment_name 扩展允许将表放到现有段上。segment_name 指向一个特定的数据库设备或一组数据库设备。在段上创建表之前，请向系统管理员或数据库所有者索取您可以使用的段列表。由于性能原因或其它方面的考虑，某些段可能会被分配给特定的表或索引。

允许空值

可为每一列指定是否允许空值。空值与“零”或“空白”不同。NULL 意味着尚未输入条目，通常暗指“未知值”或“不可用值”。它表示由于某些原因，用户尚未输入任何条目。例如，titles 表的 price 列中的空值不表示该书是免费的，而只表示价格未知，或尚未设置。

如果用户未在用关键字 null 定义的列中输入条目，则 Adaptive Server 提供“NULL”值。用关键字 null 定义的列也接受用户明确输入 NULL 值，而无论其属于何种数据类型。在字符列中输入空值时应小心。如果将“null”一词放在单引号或双引号中，Adaptive Server 会将该条目解释为字符串，而非 NULL 值。

如果在 create table 语句中省略 null 或 not null，则 Adaptive Server 使用为该数据库定义的空值模式（缺省情况下为 NOT NULL）。使用 sp_dboption 将 allow nulls by default 选项设置为 true。

必须在定义为 NOT NULL 的列中输入条目；否则，Adaptive Server 会显示一条错误消息。

将列定义为 NULL，会为未知数据提供一个占位符。例如，在 titles 表中，price、advance、royalty 和 total_sales 均设置为允许 NULL。

但 title_id 和 title 不能这样设置，因为这些列中如果缺少条目则是无意义的，会引起混乱。没有标题的价格毫无意义，但没有价格的标题则只意味着价格尚未确定，或此价格不可用。

在 create table 中，当列中的信息对于其它列的意义至关重要时，应使用关键字 not null。

与空值相关的约束和规则

不能定义一个允许空值的列，接着又用禁止空值的约束或规则覆盖此定义。例如，如果一个列定义指定 NULL，规则指定：

```
@val in (1,2,3)
```

暗指的或明示的 NULL 与规则不发生冲突。此列定义覆盖此规则，即使规则指定：

```
@val is not null
```

有关约束的详细信息，参见第 232 页上的“为表定义完整性约束”。规则在第 12 章“为数据定义缺省值和规则”中详细介绍。

缺省值和空值

对 NULL 和 NOT NULL 列均可使用缺省值（即当不输入条目时，系统自动提供的值）。缺省值与输入条目的处理方式相同。但是，不能为 NOT NULL 列指定 NULL 缺省值。可使用 create table 的 default 约束或使用 create default 将空值指定为缺省值。default 约束将在本章的后面部分介绍； create default 在第 12 章“为数据定义缺省值和规则”介绍。

如果在创建列时指定 NOT NULL，且没有为其创建缺省值，则当用户未能在插入操作期间在该列中输入值时，系统会给出一条错误消息。此外，用户不能对这些列 insert 或 update 为 NULL 值。

表 7-1 说明当用户未指定列值或明确输入一个 NULL 值时，列的缺省值及其空值类型之间相互作用。三个可能结果是空值、缺省值或一条错误消息。

表 *7-1*：列定义和空缺省值

列定义	用户输入	结果
已定义空值和缺省值	不输入值	使用缺省值
	输入 NULL 值	使用 NULL
已定义 Null，未定义缺省值	不输入值	使用 NULL
	输入 NULL 值	使用 NULL
非空，已定义缺省值	不输入值	使用缺省值
	输入 NULL 值	使用 NULL
非空，未定义缺省值	不输入值	错误
	输入 NULL 值	使用 NULL

Null 需要可变长度数据类型

只有具有可变长度数据类型的列才能存储空值。用固定长度的数据类型创建 NULL 列时，Adaptive Server 会将其转换为相应的可变长度数据类型。关于类型改变，Adaptive Server 不作通知。

表 7-2 列出固定长度数据类型及 Adaptive Server 将它们转换到的可变长度数据类型。某些可变长度的数据类型（如 moneyn）是保留的数据类型；不能使用它们来创建列、变量或参数。

表 7-2：固定长度数据类型向可变长度数据类型的转换

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

输入到 char、nchar 和 binary 列中的数据遵循可变长度列的规则，而不是用空格或零填补至列规范的完整长度。

text 和 *image* 列

用 insert 和 NULL 创建的 text 和 image 列未初始化，不包含任何值。它们不占用存储空间，不能用 readtext 或 writetext 存取。

当用 update 将 NULL 值写入 text 或 image 列时，该列被初始化，一个有效的文本指针被插入表中，一 2K 数据页被分配给该列。列初始化后，即可用 readtext 和 writetext 命令进行存取。有关详细信息，参见 Adaptive Server。

使用 IDENTITY 列

IDENTITY 列包含对应每一行的一个值，由 Adaptive Server 自动生成，在表内唯一地标识该行。

每个表只能有一个 IDENTITY 列。 可在创建表时用 create table 或 select into 语句定义 IDENTITY 列，或在以后用 alter table 语句添加。IDENTITY 列不能更新，也不允许有空值。

可通过在 create table 语句中指定关键字 identity 而不是 null 或 not null 来定义 IDENTITY 列。IDENTITY 列的数据类型必须为 numeric，标度为 0。在新表中，可将 IDENTITY 列定义为任何所需精度（1 到 38 位）：

```
create table table_name
    (column_name numeric(precision ,0) identity)
```

最大可能列值为 $10^{precision}$ - 1。例如，此命令创建一个表，其 IDENTITY 列的最大允许值为 10^5 - 1 或 9999：

```
create table sales_daily
    (sale_id numeric(5,0) identity,
    stor_id char(4) not null)
```

使用 auto identity 数据库选项和 size of auto identity 配置参数，可创建自动 IDENTITY 列。要在非唯一性索引中包括 IDENTITY 列，应使用 identity in nonunique index 数据库选项。

注意：缺省情况下，Adaptive Server 以数字 1 开始编写行号，并随着行的增加而连续编号。某些活动（如手工插入、删除或事务回退、服务器关机或故障）会使 IDENTITY 列的值形成间隔。Adaptive Server 提供了几种控制标识间隔的方法，在第 224 页上的“管理表中的标识间隔”中介绍。

创建用户定义数据类型的 IDENTITY 列

可以使用用户定义的数据类型创建 IDENTITY 列。用户定义的数据类型的基础类型必须为 numeric，且标度为 0。如果用户定义的数据类型用 IDENTITY 属性创建，则创建列时，不必重复 identity 关键字。

本例显示一个有 IDENTITY 属性的用户定义数据类型：

```
sp_addtype ident, "numeric(5)", "identity"
```

本示例显示基于 ident 数据类型的 IDENTITY 列：

```
create table sales_monthly
    (sale_id ident, stor_id char(4) not null)
```

如果用户定义的类型创建为 not null，则必须在 create table 语句中指定 identity 关键字。不能从允许空值的用户定义数据类型创建 IDENTITY 列。

引用 IDENTITY 列

创建引用 IDENTITY 列的表列时，应确保任何被引用列与 IDENTITY 列具有相同的数据类型定义。例如，在 pubs3 数据库中，sales 表被定义为以 ord_num 列作为 IDENTITY 列：

```
create table sales
(stor_id char(4) not null
     references stores(stor_id),
ord_num numeric(6,0) identity,
date datetime not null,
unique nonclustered (ord_num))
```

注意，ord_num IDENTITY 列被定义为唯一约束，需要引用 salesdetail 中的 ord_num 列。salesdetail 定义如下：

```
create table salesdetail
(stor_id char(4) not null
     references storesz(stor_id),
ord_num numeric(6,0)
     references salesz(ord_num),
title_id tid not null
     references titles(title_id),
qty smallint not null,
discount float not null)
```

向 sales 中插入一行后，要向 salesdetail 中插入一行，一个简单的方法是使用 *@@identity* 全局变量，将 IDENTITY 列值插入 salesdetail。@@identity 全局变量存储最近生成的 IDENTITY 列值。例如：

```
begin tran
insert sales values ("6380", "04/25/97")
insert salesdetail values ("6380", @@identity,
"TC3218", 50, 50)
commit tran
```

本例属于一个事务，因为这两次插入操作互相依赖、互为基础。例如，如果 sales 插入失败，则 *@@identity* 的值会不同，从而会向 salesdetail 中插入错误行。因为这两次插入同属于一个事务，因此如果一个失败，整个事务都会被拒绝。

有关 IDENTITY 列的详细信息，参见第 294 页上的“用 @@identity 检索 IDENTITY 列值”。有关事务的信息，参见第 18 章“事务：维护数据一致性和恢复”。

用 *syb_identity* 引用 IDENTITY 列

一旦定义了 IDENTITY 列，就不必记住实际的列名。必要时，可在对表的 select、insert、update 或 delete 语句中使用由表名限定的 syb_identity 关键字。

例如，本查询选择 *sale_id* 等于 1 的行：

```
select * from sales_monthly
    where syb_identity = 1
```

自动创建 “隐藏的” IDENTITY 列

系统管理员可使用 auto identity 数据库选项自动将 10 位的 IDENTITY 列包含在新表中。要在数据库中打开此功能，可使用：

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

用户每次创建新表而不指定 primary 键、unique 约束或 IDENTITY 列时，Adaptive Server 都会自动定义一个 IDENTITY 列。使用 select * 检索表中的所有列时，IDENTITY 列是不可见的。必须在选择列表中显式地包含列名 SYB_IDENTITY_COL（全部为大写字母）。如果启用了“组件集成服务”，则代理表的自动 IDENTITY 列称为 OMNI_IDENTITY_COL。

要设置自动 IDENTITY 列的精度，可使用 size of auto identity 配置参数。例如，要将 IDENTITY 列的精度设置为 15，应使用：

```
sp_configure "size of auto identity", 15
```

使用临时表

临时表创建于 tempdb 数据库中。要创建临时表，必须对 tempdb 有 create table 权限。create table 权限缺省授予数据库所有者。

要生成临时表，应在 create table 语句中的表名前使用井号 (#) 或 “tempdb..”。

临时表分为两种：

- 可在 Adaptive Server 会话之间共享的表

  可通过将 tempdb 指定为 create table 语句中表名的一部分，来创建可共享临时表。例如，下列语句创建可在 Adaptive Server 会话之间共享的临时表：

```
create table tempdb..authors
(au_id char(11))
drop table tempdb..authors
```

Adaptive Server 不改变以此方式创建的临时表名。该表会一直存在，直至服务器重启或者表的所有者使用 drop table 将其删除。

- 只能由当前 Adaptive Server 会话或过程访问的表

  可通过在 create table 语句中的表名前指定井号 (#)，来创建非共享临时表。例如：

```
create table #authors
(au_id char (11))
```

  该表会一直存在，直至当前会话或过程结束，或者其所有者用 drop table 将其删除。

如果不在表名前使用井号或“tempdb..”，且当前未使用 tempdb，则表被创建为永久表。永久表会一直保留在数据库中，直至被其所有者明确删除。

此语句创建非共享临时表：

```
create table #myjobs
(task char(30),
start datetime,
stop datetime,
notes varchar(200))
```

可用此表保留今日杂事和任务的一个列表、开始和结束时间的记录、以及可能的任何注释。此表及其数据将在当前工作会话结束时自动删除。临时表不可恢复。

可将规则、缺省值和索引与临时表建立关联，但不能在临时表上创建视图，或将触发器与之建立关联。创建临时表时，只有在用户定义的数据类型存在于 tempdb..systypes 中时，才可使用该数据类型。

要仅为当前会话向 tempdb 添加对象，则应在使用 tempdb 时执行 sp_addtype。要永久性地添加对象，应在 model 中执行 sp_addtype，然后重新启动 Adaptive Server，以将 model 复制到 tempdb 中。

确保临时表名唯一

为确保临时表名对于当前会话是唯一的，Adaptive Server：

- 将表名截断为 13 个字符，包括井号 (#)
- 用下划线 (_) 将较短的名称填补到 13 个字符
- 附加一个17位的数字后缀，它对于Adaptive Server会话是唯一的

下例显示一个创建为 #temptable 并存储为 #temptable___00000050010721973 的表：

```
use pubs2
go
create table #temptable (task char(30))
go
use tempdb
go
select name from sysobjects where name like
    "#temptable%"
go
name
------------------------------
#temptable___00000050010721973

(1 row affected)
```

在存储过程中处理临时表

存储过程可引用在当前会话期间创建的临时表。在存储过程中，不能先创建临时表、将其删除，然后再用相同名称创建新的临时表。

名称以“#”开头的临时表

退出存储过程时，在该存储过程中创建的带有以“#”开头的名称的临时表将会消失。单个过程能够：

- 创建临时表
- 将数据插入表中
- 在表上运行查询
- 调用引用该表的其它过程

因为要创建引用临时表的过程，临时表必须存在，所以要遵循以下步骤：

1 使用 create table 创建临时表。

2 创建访问临时表的过程，但不创建生成该表的过程。

3 删除临时表。

4 创建一个过程，该过程创建表并调用在第 2 步中创建的过程。

名称以 *tempdb..* 开头的表

从存储过程内部，可使用 create table tempdb..tablename 创建不带 # 前缀的表。过程完成后这些表不消失，所以它们可被独立过程引用。可按以上步骤创建这些表。

警告！*只有*当要在用户和会话间共享表时，*才*应使用“tempdb..”前缀从存储过程内创建临时表。创建和删除临时表的存储过程应使用 # 前缀来避免无意的共享。

临时表的一般规则

名称以 # 开头的临时表应遵循如下限制：

- 不能在这些表上创建视图。
- 不能将触发器与这些表建立关联。
- 无法判断出是哪个会话或过程创建了这些表。

这些限制不适用于在 tempdb 中创建的可共享的临时表。

应用于这两类临时表的规则如下：

- 可将规则、缺省值和索引与临时表建立关联。当临时表消失时，在临时表上创建的索引随之消失。
- 系统过程（如 sp_help）可使用临时表，但条件是必须从 tempdb 调用它们。
- 在临时表中不能使用用户定义的数据类型，除非该数据类型存在于 tempdb 中；即，除非自上次 Adaptive Server 重新启动以后，该数据类型已明确创建于 tempdb 中。
- 要对临时表执行 select into 操作，不必将 select into/bulkcopy 选项设置为 on。

将表创建于不同的数据库中

正如 create table 语法所示，通过用其它数据库的名称限定表名，可在当前数据库以外的数据库中创建表。但是，您必须是要创建表的数据库的授权用户，且必须具有其中的 create table 权限。

如果正在使用 pubs2 或 pubs3，且存在另一名为 newpubs 的数据库，则可在 newpubs 中创建一个名为 newtab 的表，方法如下：

```
create table newpubs..newtab (col1 int)
```

不能在当前数据库以外的数据库中创建其它数据库对象，如视图、规则、缺省值、存储过程和触发器。

管理表中的标识间隔

IDENTITY 列包含一个由 Adaptive Server 生成的、表中每一行的唯一 ID 号。由于缺省情况下由服务器生成 ID 号，因此可能偶尔会使 ID 号有很大的间隔。identity_gap 参数可控制特定表中的 ID 号及其可能的间隔。

缺省情况下，Adaptive Server 会在内存中分配一个 ID 号块，而不是将每个 ID 号按需写入磁盘，这样做需要更多的处理时间。服务器会将每个块的最大号写入表的对象分配映射 (OAM) 页。当前分配的号码块已使用或“烧毁”后，此号码用作下一个块的起始点。该块的其它号码保留在内存中，但不保存到磁盘。当号码被分配到内存，随后因为被指派到行中或因为某些反常事件（如系统故障）而被从内存中删除时，这些号码就认为被“烧毁”了。

分配 ID 号码块减少了对表的争用，从而提高了性能。但是，在 ID 号码全部指派完之前，如果服务器出现故障或用 no wait 关机，则未使用的号码被烧毁。当服务器再次运行时，它会以先前写入磁盘的块的最高号码为基础，开始为下一号码块编号。依据出现故障前被指派到行中的已分配号码的多少，ID 号可能会有很大的间隔。

标识间隔也可能由转储和装载活动数据库产生。转储时，数据库对象被保存到 OAM 页中。 如果某对象当前正被使用，则“使用的最大标识值”不在 OAM 页中，因此不被转储。

控制标识间隔的参数

Adaptive Server 提供了控制标识号间隔的参数（如表 7-3 所述）。

表 7-3：控制标识间隔的参数

参数名	范围	使用	说明
identity_gap	特定于表	create table 或 select into	为特定表创建特定大小的 ID 号块。覆盖该表的 identity burning set factor。使用 identity grab size。
identity burning set factor	全服务器范围的	sp_configure	指出要为每个块分配的总可用 ID 号的百分比。使用 identity grab size。如果将表的 identity_gap 设置为 1 或更高值，则 identity burning set factor 对该表无影响。burning set factor 用于 identity_gap 设置为 0 的所有表。 设置 identity burning set factor 时，数字以十进制形式表示，然后乘以 10,000,000 (10^7)，以得到用于 sp_configure 的正确值。例如，要每次释放 15% (.15) 的可能 IDENTITY 列值，指定的值应为 .15 乘以 10^7（或 1,500,000）： sp_configure "identity burning set factor", 1500000
identity grab size	全服务器范围的	sp_configure	为每一进程保留连续的 ID 号码块。使用 identity burning set factor 和 identity_gap。

identity burning set factor 与 *identity_gap* 的比较

使用 identity_gap 参数可控制特定表的标识间隔的大小（如下例所示）。在下例中，已创建了名为 books 的表，列有书店中的所有书籍。我们希望每本书都有唯一的 ID 号，并希望 Adaptive Server 能自动生成 ID 号。

使用 *identity burning set factor* 的示例

为 books 表定义 IDENTITY 列时，我们使用了缺省的数值 (18, 0)，提供总共 999,999,999,999,999,999 个 ID 号。对于 identity burning set factor 配置参数，我们使用缺省设置 5000（999,999,999,999,999,999 的 .05%），即表示 Adaptive Server 分配 500,000,000,000,000 个号码的号码块。

服务器在内存中分配第一批 500,000,000,000,000 个号码，并将该块的最高号码 (500,000,000,000,000) 存储在表的 OAM 页中。当所有号码均被指派到行中或烧毁后，服务器会取来下一个号码块（下一批 500,000,000,000,000 个号码），此块从 500,000,000,000,001 开始，并将 1,000,000,000,000,000 存储为该块的最高号码。

假定，在行号 500,000,000,000,022 后，服务器出现故障。只有号码 1 到 500,000,000,000,022 被用作 books 的 ID 号。号码 500,000,000,000,023 到 1,000,000,000,000,000 被烧毁。当服务器再次运行时，会创建以存储在表的 OAM 页中的最高号加一 (1,000,000,000,000,001) 开始的 ID 号，留下 499,999,999,999,978 个号码间隔。

要减少特定表这样大的标识号间隔，可按“使用 identity_gap 的示例”所述设置标识间隔。

使用 *identity_gap* 的示例

在此情况下，我们用 identity_gap 值 1000 创建 books 表。这样会覆盖能产生 500,000,000,000,000 个 ID 号码的号码块的全服务器范围 identity burning set factor 设置。更改后，服务器会在内存中分配 1000 个 ID 号码的号码块。

服务器分配第一批的 1000 个号码，并将该块的最高号码 (1000) 存储到磁盘中。所有号码均被使用后，服务器会取来下一批 1000 个号码，以 1001 开始，并将 2000 作为最高号码存储。

现在，假定在行号 1002 之后停电了，服务器随之断电。则只有号码 1000 到 1002 被使用，而号码 1003 到 2000 丢失。当服务器再次运行时，会创建以存储在表的 OAM 页中的最高号加一 (2000) 开始的 ID 号，只留下 998 个号码间隔。

通过为表设置 identity_gap 而不使用全服务器范围的 table burning set factor，可极大地减少ID号码的间隔。但是，如果此值设置得过低，会使性能降低。服务器每次都必须将块的最高号码写入磁盘，因此性能受到影响。例如，如果 identity_gap 设置为 1，即表示一次分配一个 ID 号，服务器必须在每次创建行时都写入新号码，这样会在表上出现页锁争用，从而可能会降低性能。必须找到可获得最佳性能、而对您的具体情况来说间隔值又最低的最佳设置。

设置特定于表的标识间隔

使用 create table 或 select into 创建表时，应设置特定于表的标识间隔。

用 *create table* 设置标识间隔

此命令的语法为：

create table *table_name* (*column_name*
datatype(*constant_expression*) identity)
with identity_gap = *value*

例如：

```
create table mytable (IdNum numeric(12,0) identity)
with identity_gap = 10
```

此语句创建一个带 identity 列的名为 mytable 的表。标识间隔设置为 10，表示将在内存中以十个 ID 号的块为单位分配 ID 号。如果服务器出现故障或没有等待就关闭服务器，那么分配给行的最后一个 ID 号与分配给行的下一个 ID 号之间的差距为十个号码。

用 *select into* 设置标识间隔

如果用 select into 语句从具有特定标识间隔设置的表创建表，则新表不继承父表的标识间隔设置。而使用 identity burning set factor 设置。若要为新表指定一个特定的 identity_gap 设置，则应在 select into 语句中指定标识间隔。可以为新表指定一个与父表相同或不同的标识间隔。

例如，要从带标识间隔的现有表 (mytable) 创建新表 (newtable)：

```
select identity into newtable
with identity_gap = 20
from mytable
```

更改特定于表的标识间隔

要更改特定表的标识间隔，应使用 sp_chgattribute：

sp_chgattribute "*table_name*", "identity_gap", *set_number*

其中：

- *table_name* 为要改变标识间隔的表的名称。

- identity_gap 表示要改变标识间隔。
- *set_number* 是标识间隔大小的新值。

例如：

```
sp_chgattribute "mytable", "identity_gap", 20
```

要将 mytable 改为使用 identity burning set factor 设置而不是 identity_gap 设置，应将 identity_gap 设置为 0：

```
sp_chgattribute "mytable", "identity_gap", 0
```

显示特定于表的标识间隔信息

要查看表的 identity_gap 设置，应使用 sp_help。

例如，identity_gap 列（接近输出末尾）的零值表示未设置特定于表的标识间隔。mytable 使用全服务器范围的 identity burning set factor 值。

```
sp_help mytable

Name                            Owner
     Type
------------------------------ --------------------
----------
    ----------------------
mytable                         dbo
     user table

(1 row affected)
Data_located_on_segment        When_created
------------------------------ --------------------
------
default                          Jun  8 1999  5:35PM
Column_name     Type            Length Prec Scale
Nulls Default_name

     Rule_name       Identity
--------------- --------------- ------ ---- ----- -
---- ---------------

    --------------- --------
idnum         numeric           6   12    0    0 NULL
     NULL                   1
Object does not have any indexes.

No defined keys for this object.
```

```
Object is not partitioned.
Lock scheme Allpages
The attribute 'exp_row_size' is not applicable to
tables with
allpages lock scheme.
The attribute 'concurrency_opt_threshold' is not
applicable to
tables with allpages lock scheme.

exp_row_size reservepagegap fillfactor
max_rows_per_page identity_gap
 ------------ -------------- ---------- -----------
------ ------------

         1           0        0             0
0
concurrency_opt_threshold
-------------------------
0
```

如果将 mytable 的 identity_gap 改为 20，则在该表 sp_help 输出的 identity_gap 列会显示 20。此设置会覆盖全服务器范围的 identity burning set factor 值。

```
sp_help mytable

Name                            Owner
     Type
------------------------------ --------------------
----------
    -----------------------
mytable                         dbo
     user table

(1 row affected)
Data_located_on_segment         When_created
------------------------------ --------------------
------
default                          Jun  8 1999  5:35PM
Column_name     Type            Length Prec Scale
Nulls Default_name
     Rule_name       Identity
--------------- --------------- ------ ---- ----- -
---- ---------------
    --------------- --------
idnum         numeric           6   12    0    0 NULL
     NULL                   1
```

```
Object does not have any indexes.
No defined keys for this object.
Object is not partitioned.
Lock scheme Allpages
The attribute 'exp_row_size' is not applicable to
tables with
allpages lock scheme.
The attribute 'concurrency_opt_threshold' is not
applicable to
tables with allpages lock scheme.

exp_row_size reservepagegap fillfactor
max_rows_per_page identity_gap
------------ -------------- ---------- ------------
----- ------------
1                   0        0             0
20
concurrency_opt_threshold
-------------------------
0
```

由于插入、删除、identity grab size 和回退原因产生的间隔

手工插入 IDENTITY 列、删除行、identity grab size 配置参数的值以及事务回退都可使 IDENTITY 列值产生间隔。这些间隔不受 identity burning set factor 配置参数设置的影响。

例如，假设有一带有如下值的 IDENTITY 列：

```
select syb_identity from stores_cal
 id_col
 -------
      1
      2
      3
      4
      5

(5 rows affected)
```

可删除 IDENTITY 列值在 2 和 4 之间的所有行，使列值出现间隔：

```
delete stores_cal
where syb_identity between 2 and 4
select syb_identity from stores_cal
 id_col
```

```
 ------
      1
      5

(2 rows affected)
```

为表设置 identity_insert on 之后，表所有者、数据库所有者或系统管理员可手工插入任何大于 5 的合法数值。例如，如果插入值 55，便会使 IDENTITY 列值产生很大间隔：

```
insert stores_cal
(syb_identity, stor_id, stor_name)
values (55, "5025", "Good Reads")
select syb_identity from stores_cal
id_col
 -------
      1
      5
     55

(3 rows affected)
```

如果 identity_insert 随后被设置为 off，则 Adaptive Server 会在下一次插入时为 IDENTITY 列指派 55 + 1 或 56 的值。如果包含 insert 语句的事务被回退，Adaptive Server 会放弃值 56，并为下一次插入使用值 57。

如果表插入达到 IDENTITY 列的最大值

可插入表中的最大行数取决于 IDENTITY 列的精度设置。如果某表达到该限制，则可用较高精度重新创建一个表，或者如果该表的 IDENTITY 列未用于参照完整性，则可用 bcp 实用工具删除间隔。有关详细信息，参见第 295 页上的“达到 IDENTITY 列的最大值”。

为表定义完整性约束

Transact-SQL 为维护数据库中的数据完整性提供了两种方法：

- 定义规则、缺省值、索引和触发器
- 定义 create table 完整性约束

选择哪种方法取决于您的需求。完整性约束的优点表现在：在创建表的进程中一步完成完整性控件的定义（如 SQL 标准规定），并可简化创建这些完整性控件的过程。但是，与缺省值、规则、索引和触发器相比，完整性约束的范围较为有限，并且综合性也较差。

例如，触发器对参照完整性的处理方式比在 create table 中声明的方法更为复杂。此外，由 create table 语句定义的完整性约束是针对于该表的；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。即使在同一张表上，约束也不能包含子查询或集合函数。

这两种方法不是互斥的。可以把完整性约束与缺省值、规则、索引与触发器结合使用。在应用中可灵活选择最佳方法。本部分介绍 create table 完整性约束。缺省值、规则、索引和触发器将在后面的章节中加以介绍。

可创建以下类型的约束：

- unique 和 primary key 约束要求在一个表中，任意两行的特定列中均不能有相同的值。此外，primary key 约束要求列的任何行中都不能有空值。
- 参照完整性 (references) 约束要求特定列中插入的数据应在指定表和列中有与之相匹配的数据。sp_helpconstraint 用于查找表的被参照表。
- check 约束限制插入到列中的数据值。

也可通过以下方法来强制实现数据完整性：即限制在列中使用空值（null 或 not null 关键字）；为列提供缺省值（default 子句）。有关 null 和 not null 关键字的信息，参见第 215 页上的“允许空值”。

有关为表定义的任何约束的信息，参见第 273 页上的“使用 sp_helpconstraint 查找表的约束信息”。

警告！不要定义或变更系统表的约束定义。

指定表级或列级约束

可在表级或列级声明完整性约束。这种区别是语法上的区别。列级约束应放在列名和数据类型之后，分隔逗号之前。应按单独的逗号分隔子句输入表级约束。Adaptive Server 同等对待表级和列级约束；两种方式同等有效。

但是，必须将操作于多个列的约束声明为表级约束。例如，下列 create table 语句有一个操作于两个列（pub_id 和 pub_name）的 check 约束：

```
create table my_publishers
(pub_id      char(4),
pub_name     varchar(40),
constraint my_chk_constraint
    check (pub_id in ("1389", "0736", "0877")
        or pub_name not like "Bad News Books"))
```

可将仅操作于一个列的约束声明为列级约束，但这并不是必需的。例如，如果上面的 check 约束只使用一个列 (pub_id)，则可将该约束放到该列上：

```
create table my_publishers
(pub_id     char(4) constraint my_chk_constraint
        check (pub_id in ("1389", "0736", "0877")),
pub_name     varchar(40))
```

无论哪种情况下，constraint 关键字和伴随的 *constraint_name* 都是可选的。check 约束在第 239 页上的“指定检查约束”介绍。

注意：不能用检查约束发布 create table，然后在同一批处理或过程中将数据插入到表中。可以将创建和插入语句分放在两个不同的批处理或过程中，或者使用 execute 分别执行操作。

为约束创建错误消息

可创建错误消息并将其绑定到约束，方法为：用 sp_addmessage 创建消息，用 sp_bindmsg 绑定到约束。

例如：

```
sp_addmessage 25001,
     "The publisher ID must be 1389, 0736, or 0877"
sp_bindmsg my_chk_constraint, 25001
insert my_publishers values
```

```
     ("0000", "Reject This Publisher")
Msg 25001, Level 16, State 1:
Server 'snipe'Line 1:
The publisher ID must be 1389, 0736, or 0877
Command has been aborted.
```

要更改约束的消息，绑定一条新的消息即可。新消息会替换旧消息。

使用 sp_unbindmsg 可将消息从约束解除绑定；使用 sp_dropmessage 可删除用户定义的消息。

例如：

```
sp_unbindmsg my_chk_constraint
sp_dropmessage 25001
```

要更改消息文本，又要保留相同的错误号，可解除绑定消息，用 sp_dropmessage 将其删除，然后用 sp_addmessage 再次添加消息，再用 sp_bindmsg 绑定。

创建检查约束后

创建检查约束之后，描述检查约束的**源文本**存储在 syscomments 系统表的 text 列中。*不要从 syscomments 中删除此信息*；否则将来升级 Adaptive Server 时会遇到问题。出于安全考虑，可用 sp_hidetext 在 syscomments 中将文本加密（如*参考手册*所述）。有关详细信息，参见第 4 页上的“编译对象”。

指定缺省列值

定义任何列级完整性约束之前，可用 default 子句为该列指定缺省值。**缺省子句**可为某列指派缺省值，它是 create table 语句的一部分。当用户不输入列的值时，Adaptive Server 会插入该缺省值。

可将下列值用于 default 子句：

- *constant_expression* — 指定要用作列缺省值的常量表达式。它不能包含任何列或其它数据库对象的名称，但可以包括不引用数据库对象的内部函数。该缺省值必须与列的数据类型兼容。
- user — 指定Adaptive Server插入用户名作为缺省值。要使用此缺省值，列的数据类型必须为 char(30) 或 varchar(30)。

- null — 指定 Adaptive Server 插入空值作为缺省值。不能为不允许空值的列定义此缺省值（使用 notnull 关键字）。

例如，此 create table 语句定义两个列缺省值：

```
create table my_titles
(title_id        char(6),
title            varchar(80),
price            money       default null,
total_sales      int         default 0)
```

表中的每个列只能包括一个 default 子句。

使用 default 子句指派缺省值比两步 Transact-SQL 方法要简单。在 Transact-SQL 中，可使用 create default 声明缺省值，然后用 sp_bindefault 将其绑定到该列。

指定唯一约束和主键约束

可声明 unique 或 primary key 约束，来确保一个表中的任意两行的指定列中没有相同的值。这两个约束创建唯一索引，以强制实现此数据完整性。然而，primary key 约束比 unique 约束的限制性更强。带 primary key 约束的列不能包含 NULL 值。通常表的 primary key 约束应与其它表上定义的参照完整性约束结合使用。

SQL 标准的 unique 约束定义规定，列定义不允许使用空值。缺省情况下，如果在列定义中省略 null 或 not null，Adaptive Server 会将列定义为不允许空值（如尚未用 sp_dboption 进行更改）。在 Transact-SQL 中，可以将列定义为在 unique 约束下允许空值，因为用于强制约束的唯一索引允许插入空值。

注意：不能混淆唯一和主键完整性约束与由 sp_primarykey、sp_foreignkey 和 sp_commonkey 定义的信息。唯一和主键约束实际上会创建索引来定义表列的唯一或主键属性。sp_primarykey、sp_foreignkey 和 sp_commonkey 为通过创建索引和触发器强制的表列定义键（在 syskeys 表中）的逻辑关系。

缺省情况下，unique 约束创建唯一的非聚簇索引； primary key 约束创建唯一的聚簇索引。可声明带有任意类型约束的聚簇或非聚簇索引。

例如，下列 create table 语句使用表级 unique 约束，来确保在 stor_id 和 ord_num 列中没有哪两行具有相同的值：

```
create table my_sales
(stor_id      char(4),
ord_num       varchar(20),
date          datetime,
unique clustered (stor_id, ord_num))
```

一个表只能有一个聚簇索引，所以只能指定一个 unique clustered 或 primary key clustered 约束。

强制数据完整性时，可使用 unique 和 primary key 约束来创建唯一索引（包括 with fillfactor、with max_rows_per_page 和 on *segment_name* 选项）。但是，索引可提供附加功能。有关索引及其选项的信息（包括聚簇索引与非聚簇索引之间的差别），参见第 11 章“创建表的索引”。

指定参照完整性约束

参照完整性指用于管理表间关系的方法。创建表时，可定义约束，以确保插入特定表中的数据在另一表中有匹配值。

可在表中定义的参照共有三种类型： 对另一个表的参照、从另一个表的参照和自身参照，即同一个表中的参照。

以下两个表来自 pubs3 数据库，举例说明声明的参照完整性如何工作。第一个表 stores 是“被参照”的表：

```
create table stores
(stor_id      char(4) not null,
stor_name     varchar(40) null,
stor_address  varchar(40) null,
city          varchar(20) null,
state         char(2) null,
country       varchar(12) null,
postalcode    char(10) null,
payterms      varchar(12) null,
unique nonclustered (stor_id))
```

第二个表 store_employees 是“参照”表，因为它包含对 stores 表的参照。它还包含自身参照：

```
create table store_employees
(stor_id      char(4) null
        references stores(stor_id),
emp_id        id not null,
mgr_id        id null
        references store_employees(emp_id),
```

```
emp_lname     varchar(40) not null,
emp_fname     varchar(20) not null,
phone         char(12) null,
address       varchar(40) null,
city          varchar(20) null,
state         char(2) null,
country       varchar(12) null,
postalcode    varchar(10) null,
unique nonclustered (emp_id))
```

store_employees 表中定义的参照强制执行下列限制：

- store_employees 表中指定的任何商店都必须包括在 stores 表中。references 约束规定任何插入到 store_employees 的 stor_id 列中的值必须已存在于 my_stores 的 stor_id 列中，从而强制执行这一限制。
- 所有管理人员都必须有员工标识号。references 约束规定任何插入到 mgr_id 列中的值必须已存在于 emp_id 列中，从而强制执行这一限制。

表级或列级参照完整性约束

可在列级或表级定义参照完整性约束。前例中的参照完整性约束是使用 create table 语句中的 references 关键字在列级定义的。

定义表级参照完整性约束时，应包括 foreign key 子句和一个或多个列名的列表。foreign key 指定：当前表中的列出列为外键，其目标键为以下 references 子句所列出的列。例如：

```
constraint sales_detail_constr
     foreign key (stor_id, ord_num)
     references my_salesdetail(stor_id, ord_num)
```

foreign key 语法只允许用于表级约束，而不允许用于列级约束。有关详细信息，参见第 233 页上的“指定表级或列级约束”。

在列级或表级定义参照完整性约束后，可使用 sp_primarykey、sp_foreignkey 和 sp_commonkey 在 syskeys 系统表中定义键。

表中允许的最大参照数

表中允许的最大参照数为 192。可使用 sp_helpconstraint 来检查表的参照（如第 273 页上的“使用 sp_helpconstraint 查找表的约束信息”所述）。

为交叉引用约束使用 *create schema*

不能创建参照不存在的表的表。要创建两个或多个相互参照的表，应使用 create schema。

模式是特定用户所拥有的对象及与这些对象相关的权限的集合。如果 create schema 语句中的任何语句失败，整个命令将作为一个单元回退，无生效的语句。

create schema 语法为：

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

例如：

```
create schema authorization dbo
    create table list1
        (col_a char(10) primary key,
        col_b char(10) null
        references list2(col_A))
    create table list2
        (col_A char(10) primary key,
        col_B char(10) null
        references list1(col_a))
```

创建参照完整性约束的一般规则

在表中定义参照完整性约束时：

- 确保拥有对被参照表的 references 权限。有关权限的信息，参见*系统管理指南*。
- 确保被参照的列受被参照表中唯一索引的约束。可使用 unique 或 primary key 约束或者 create index 语句来创建唯一索引。例如，stores 表中被参照的列定义如下：

  ```
  stor_id char(4) primary key
  ```

- 确保用在参照定义中的列有匹配的数据类型。例如，在 my_stores 和 store_employees 中的 stor_id 列都是使用 char(4) 数据类型创建的。store_employees 中的 mgr_id 和 emp_id 列都是用 id 数据类型创建的。
- 只有在通过primary key约束将被参照表中的列指定为主键时，才可在 references 子句中省略列名。

- 在被参照表中，不能更新与参照表中的值相匹配的列值或删除与参照表中的值相匹配的行。而应首先从参照表中执行删除或更新，然后再从被参照表中删除它。

  同样，也不能对被参照表使用 truncate table 命令。而应首先截断参照表，然后再截断被参照表。

- 在删除被参照表之前，必须先删除参照表；否则将违反约束。
- sp_helpconstraint 用于查找表的被参照表。

参照完整性约束为强制数据完整性提供了比使用触发器更为简单的方式。但是，触发器为强制表间的参照完整性提供附加能力。有关详细信息，参见第 16 章“触发器：强制实施参照完整性”。

指定检查约束

可声明 check 约束来限制用户可在表列中插入的值。检查约束对于检查有限的具体值域的应用程序非常有用。check 约束指定了值在插入表之前必须通过的 *search_condition*。*search_condition* 可包括：

- 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
- 由 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、集合函数说明或目标说明。

例如，此语句确保只有某些特定的值才能输入 pub_id 列：

```
create table my_new_publishers
(pub_id      char(4)
        check (pub_id in ("1389", "0736", "0877",
             "1622", "1756")
        or pub_id like "99[0-9][0-9]"),
pub_name     varchar(40),
city         varchar(20),
state        char(2))
```

列级检查约束仅可参照定义约束的列；不能参照表中的其它列。表级检查约束可参照表中任意列。create table 允许列定义中有多个检查约束。

因为检查约束不替代列定义，所以，如果列定义允许空值，就不能使用检查约束来禁止空值。如果对允许空值的列声明了检查约束，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，假定对允许空值的表列定义下列检查约束：

```
check (pub_id in ("1389", "0736", "0877", "1622",
"1756"))
```

您仍可将 NULL 插入该列。因为下列表达式始终判断为真，所以列定义替代检查约束：

```
col_name != null
```

设计使用参照完整性的应用程序

设计使用参照完整性功能的应用程序时：

- 不创建不必要的参照约束。表中具有的参照约束越多，需要参照完整性的语句在表上运行得越慢。
- 在表上尽可能少地使用自参照约束。
- 为检查有限的具体值域的应用程序使用 check 约束，而不使用 references 约束。由于没有参照，使用 check 约束消除要求 Adaptive Server 扫描其它表以完成查询的需要。因此，对这样的表运行查询的速度快于对使用参照的表运行查询。

  例如，该表使用 check 约束将作者限制在 California：

  ```
  create table cal_authors
  (au_id id not null,
  au_lname varchar(40) not null,
  au_fname varchar(20) not null,
  phone char(12) null,
  address varchar(40) null,
  city varchar(20) null,
  state char(2) null
       check(state = "CA"),
  country varchar(12) null,
  postalcode char(10) null)
  ```

- 将普通扫描的外键索引绑定到各自的高速缓存中，以优化性能。唯一索引在声明为主键的列上自动创建。当相应的外键被更新或插入时，通常选择这些索引扫描被参照表。
- 保留最小数量的候选键的多行更新。

- 将参照完整性查询放入使用约束检查的过程中。约束检查被编入执行计划；当参照约束改变时，具有该约束的过程在执行时被自动重新编译。
- 如果不能将参照完整性查询内置于过程中，且频繁地在特别批处理中运行参照完整性，那么将系统目录 sysreferences 绑定到其高速缓存上。这样提高了 Adaptive Server 需要重新编译参照完整性查询时的性能。
- 创建了具有参照约束的表之后，使用该表运行查询之前，使用 set showplan, noexec on 对其进行测试。showplan 输出指示运行查询所需的辅助扫描描述符的数量；扫描描述符管理在其上运行查询时表的扫描。如果辅助扫描描述符的数量很大，则重新设计该表使其使用较少扫描描述符，或者增加 number of auxiliary scan descriptors 配置参数的值。

如何设计和创建表

本节给出一个 create table 语句的示例，可用其创建自己的练习用表。如果没有 create table 权限，则咨询系统管理员或您使用的数据库的所有者。

创建一个表通常意味着创建与其相关的索引、缺省值和规则。还经常涉及自定义数据类型、触发器和视图。

当然，可创建表，并在创建索引、缺省值、规则、触发器或视图前输入某些数据及使用该表。这允许您查看最常用的事务，最常输入的数据。

另一方面，同时设计表及其所有组件通常是最有效的方式。这里是您要遵循的步骤的概述。您会发现在实际创建表及其相应对象之前，在纸上草绘计划是最简单的方式。

首先，计划表的设计：

1. 决定在表中需要哪些列以及各列的数据类型、长度、精度与标度。
2. 在定义该表用于何处*之前*，应创建所有新用户定义的数据类型。
3. 决定哪一列应成为 IDENTITY 列（如果有）。
4. 决定哪些列能接受空值，哪些列不能。

5 决定需要在表中的列上增加何种完整性约束或列缺省值（如果有）。这也包括决定何时使用列约束与缺省值代替缺省值、规则、索引与触发器来保证数据完整性。

6 决定是否需要缺省值与规则，如果需要，何处需要何种类型的缺省值与规则。考虑列的 NULL 与 NOT NULL 状态与缺省值及规则之间的关系。

7 决定在何处需要何种索引。在第 11 章“创建表的索引”中，对索引进行了讨论。

现在，创建表及其相关对象：

1 使用 create table 和 create index 创建表及其索引。

2 用 create default 和 create rule 创建需要的缺省值和规则。在第 12 章“为数据定义缺省值和规则”中，对这些命令进行了讨论。

3 用 sp_bindefault 和 sp_bindrule 绑定任何需要的缺省值和规则。如果在 create table 语句中使用了用户定义的数据类型，而该类型已被捆绑了缺省值或规则，则它们将自动生效。在第 14 章“使用存储过程”中，对这些系统过程进行了讨论。

4 使用 create trigger 创建触发器。在第 16 章“触发器：强制实施参照完整性”中，对触发器进行了讨论。

5 使用 create view 创建视图。在第 9 章“视图：限制访问数据”中，对视图进行了讨论。

制作设计草图

称为 friends_etc 的表用于本章和后续章节中，例示创建索引、缺省值、规则、触发器等的方法。其中包含您朋友的名字、地址、电话号码和个人信息。表中没有定义列的缺省值或完整性约束，以便不与这些对象冲突。

如果另一用户已创建了 friends_etc 表，则在计划跟随此示例并创建伴随 friends_etc 的对象之前与系统管理员或数据库所有者核实情况。friends_etc 的所有者需要删除索引、缺省值、规则和触发器，以便在创建这些对象时没有冲突。

表 7-4 显示建议的 friends_etc 表的结构与各列的索引、缺省值与规则。

表 *7-4*：示例表设计

列	数据类型	是否允许 Null?	索引	缺省值	规则
pname	nm	NOT NULL	nmind（组合）		
sname	nm	NOT NULL	nmind（组合）		
address	varchar(30)	NULL			
city	varchar(30)	NOT NULL		citydflt	
state	char(2)	NOT NULL		statedflt	
zip	char(5)	NULL	zipind	zipdflt	ziprule
phone	p#	NULL			phonerule
age	tinyint	NULL			agerule
bday	datetime	NOT NULL		bdflt	
gender	bit	NOT NULL		gndrdflt	
debt	money	NOT NULL		gndrdflt	
notes	varchar(255)	NULL			

创建用户定义的数据类型

前两个列用于个人的名（第一个）和姓。它们定义为 nm 数据类型。在创建表之前，需要创建数据类型。对于 phone 列的 p# 数据类型同样如此：

```
execute sp_addtype nm, "varchar(30)"
execute sp_addtype p#, "char(10)"
```

nm 数据类型允许带有最大长度为 30 字节的可变长度字符条目。p# 数据类型允许带有固定长度为 10 字节的 char 数据类型。

选择能接受空值的列

除被指派了用户定义的数据类型的列外，每一列都有显式的 NULL 或 NOT NULL 条目。不需要在表定义中指定 NOT NULL，因为缺省如此。为了可读，此表被设计成显式地标明 NOT NULL。

缺省 NOT NULL 意味着列要求有条目，如此表中的两个名称列。没有名称，其它的数据就没有意义。另外，gender 列必须为 NOT NULL，因为不能给 bit 列使用 NULL 值。

如果一列指定为 NULL 并绑定了缺省值，则如果输入中没有给定值的话，是该缺省值而不是 NULL 被输入。如果一列被指定为 NULL，而捆绑到该列的规则没有指定 NULL，当没有值输入时，列的定义将替换规则。列可以同时拥有缺省值和规则。在第 12 章 “为数据定义缺省值和规则”中，讨论它们两个之间的关系。

定义表

现在，编写 create table 语句：

```
create table friends_etc
(pname        nm             not null,
sname         nm             not null,
address       varchar(30)    null,
city          varchar(30)    not null,
state         char(2)        not null,
postalcode    char(5)        null,
phone         p#             null,
age           tinyint        null,
bday          datetime       not null,
gender        bit            not null,
debt          money          not null,
notes         varchar(255)   null)
```

现在已为个人名和姓、地址、城市、州 / 省、邮政编码、电话号码、年龄、生日、性别、债务信息和注释定义了列。以后，会为此表创建规则、缺省值、索引、触发器和视图。

从查询结果中创建新表：*select into*

select into 命令允许您根据在 select 语句的选择列表中指定的列以及在 where 子句中选择的行来创建新表。into 子句对创建测试表、作为测试表副本的新表，以及从在表中制作几个较小的表的操作很有用处。仅当 select into/bulkcopy/pllsort 数据库选项被设置为 on 时，才可对永久表使用 select into。系统管理员可用 sp_dboption 打开此选项。使用 sp_helpdb 来查看此选项是否已打开。

这就是当 select into/bulkcopy/pllsort 数据库选项设置为 on 时，sp_helpdb 及其结果：

```
sp_helpdb pubs2

name      db_size owner   dbid    created     status
---------  ------- ------  -----   -----------  ---
---------
pubs       2 MB    sa      5       Jun  5 1997  select
into

/bulkcopy/pllsort

(1 row affected)

device                size         usage
-----------------     ---------    --------------
master                2 MB         data and log

(1 row affected)
```

sp_helpdb output 指示该选项已被设置为 on 还是 off。只有系统管理员或数据库所有者才能设置这些数据库选项。

如果 select into/bulkcopy/pllsort 数据库选项为 on，则可使用 select into 子句来建立新的永久表，而不必使用 create table 语句。可 select into 一个临时表，即使 select into/bulkcopy/pllsort 选项不是为 on。

注意：因为 select into 是最低限度记录的操作，因此在执行 select into 之后使用 dump database 命令来备份数据库，不能执行最低限度记录操作之后转储事务。

不象显示部分表的视图，用 select into 创建的表是分离的、独立的实体。有关详细信息，参见第 9 章“视图：限制访问数据”。

基于在选择列表中指定的列，在 from 子句中命名的表和在 where 子句中选择的行来建立新表。新表的名称在数据库中必须是唯一的，并且必须符合标识符规则。

带有 into 子句的 select 语句允许基于现有定义和数据定义一个表并放入数据，而不必经历通常的数据定义进程。

以下示例显示一个 select into 语句及其结果。使用四列表 publishers 中的两个列，创建了称为 newtable 的表。因为此语句不包括 where 子句，因此 publishers 的所有行（除了两个指定行）中的数据都被复制到 newtable 中。

```
select pub_id, pub_name
into newtable
from publishers
```

```
(3 rows affected)
```

“3 rows affected”指的是插入到 newtable 中的三行。这就是 newtable：

```
select *
from newtable
pub_id  pub_name
------  ------------------------------------
0736    New Age Books
0877    Binnet & Hardley
1389    Algodata Infosystems
(3 rows affected)
```

新表包含 select 语句的结果。成为数据库的一部分，就象其父表一样。

通过在 where 子句中放置一个假条件，可创建一个不带数据的框架 (skeleton) 表。例如：

```
select *
into newtable2
from publishers
where 1=2
(0 rows affected)
select *
from newtable2
pub_id  pub_name                city        state
------    --------------    --------  -----

 (0 rows affected)
```

因为 1 不可能等于 2，所以不会有行插入新表中。

也可使用带有集合功能的 select into 来创建带有摘要数据的表：

```
select type, "Total_amount" = sum(advance)
into #whatspent
from titles
group by type
(6 rows affected)
select * from #whatspent
 type         Total_amount
 ------------ ------------------------
 UNDECIDED                       NULL
 business                   25,125.00
 mod_cook                   15,000.00
 popular_comp               15,000.00
 psychology                 21,275.00
```

```
trad_cook                    19,000.00

(6 rows affected)
```

始终为 select into 结果表中的任意列提供一个名称，该结果表得自一个集合函数或任何其它表达式。以下为示例：

- 算术集合，例如，*amount* * 2
- 并置，例如，lname + fname
- 函数，例如，lower(lname)

下面是使用并置的一个示例：

```
select au_id,
    "Full_Name" = au_fname + ' '+ au_lname
into #g_authortemp
from authors
where au_lname like "G%"
(3 rows affected)
select * from #g_authortemp
 au_id       Full_Name
 ----------- -------------------------
 213-46-8915 Marjorie Green
 472-27-2349 Burt Gringlesby
 527-72-3246 Morningstar Greene

 (3 rows affected)
```

由于函数允许空值，来自 convert() 或 isnull() 之外函数的表中的任意列也允许空值。

检查错误

select into 是个两步骤的操作。第一步创建新表，第二步将指定行插入新表。

因为未记录 select into 操作，所以不能在用户定义的事务中发出这些操作，也不能回退。

如果 select into 语句在创建新表后失败，Adaptive Server 将*不会*自动删除此表或释放此表的第一个数据页。这意味着，发生错误之前在第一页插入的所有行都将保留在该页上。检查 select into 语句后全局变量 *@@error* 的值，以确保没有错误发生。

如果在 select into 操作中发生错误，则使用 drop table 来删除新表，然后重新发出 select into 语句。

对 IDENTITY 列使用 *select into*

本节描述对包含 IDENTITY 列的表使用 select into 命令的特殊规则。

选择一个 IDENTITY 添加到新表

要选择一个现有 IDENTITY 列并添加到新表中，应在 select 语句的 *column_list* 中包括该列名（或 syb_identit 关键字）。

select *column_list*
  into *table_name*
  from *table_name*

以下示例基于来自 stores_cal 表的列，创建一个新表 stores_cal_pay30：

```
select record_id, stor_id, stor_name
into stores_cal_pay30
from stores_cal
where payterms = "Net 30"
```

新列继承 IDENTITY 属性，除非下列任何条件中为真：

- IDENTITY 列被多次选择。
- IDENTITY 列被选择作为表达式的一部分。
- select 语句包含一个 group by 子句、集合函数、union 运算符或连接。

多次选择 IDENTITY 列

一张表不能有多个 IDENTITY 列。如果多次选择某一 IDENTITY 列，它将在新表中定义为 NOT NULL。该 IDENTITY 列将不继承 IDENTITY 属性。

在下述示例中，record_id 列（由名称选择一次又由 syb_identity 关键字选择一次）在 stores_cal_pay60 中被定义为 NOT NULL：

```
select syb_identity, record_id, stor_id, stor_name
into stores_cal_pay60
from stores_cal
where payterms = "Net 60"
```

用 *select into* 命令添加一个新 IDENTITY 列

要在 select into 语句中添加一个新 IDENTITY 列，可在 into 子句之前添加列定义。该定义包括列的精度，但不包括其标度：

select *column_list*
　identity_column_name = identity(*precision*)
　into *table_name*
　from*table_name*

以下示例在 discounts 表中创建一个新表 new_discounts，并添加一个新 IDENTITY 列 id_col：

```
select *, id_col=identity(5)
into new_discounts
from discounts
```

如果 *column_list* 包括一个现有 IDENTITY 列，且您添加了新 IDENTITY 列的描述，则 select into 语句失败。

定义一个必须计算其值的列

IDENTITY 列值由 Adaptive Server 生成。基于 IDENTITY 列生成，且其值必须经计算而不是生成得来的新列，不能继承 IDENTITY 属性。

如果表的 select 语句包括作为表达式的一部分的一个 DENTITY 列，则生成的列值必须计算。如果表达式中的任何列都允许 NULL 值，则新列就被创建为 NULL。否则，为 NOT NULL。

在以下示例中，通过向 record_id 的值增加 1000 来计算的 new_id 列，被创建为 NOT NULL：

```
select new_id = record_id + 1000, stor_name
into new_stores
from stores_cal
```

如果 select 语句包含 group by 子句或集合函数，则列值也被计算。如果 IDENTITY 列是集合函数的自变量，那么结果列被创建为 NULL。否则，为 NOT NULL。

IDENTITY 列通过联合或连接选择到表中

IDENTITY 列的值唯一地标识表的每一行。然而，如果表的 select 语句包含联合或连接，则单独的行可在结果集中多次出现。通过联合或连接选入表中的 IDENTITY 列不保留 IDENTITY 属性。如果表中包含 IDENTITY 列和 NULL 列的联合，新列将定义为 NULL。否则，为 NOT NULL。

有关详细信息，参见第 217 页上的“使用 IDENTITY 列”和第 304 页上的“更新 IDENTITY 列”。另请参见*参考手册*中的 select。

改变现有表

使用 alter table 命令改变现有表的结构。您可以：

- 添加列和约束
- 改变列缺省值
- 添加 null 列或 non-null 列
- 删除列和约束
- 改变锁定方案
- 分区或未分区表
- 转换列数据类型
- 转换现有列的 null 缺省值
- 增加或减小列长度

alter table 包括以下用于修改表的语法：

```
alter table table_name
    [add column_name datatype [identity | null |
        not null] [, column_name datatype [identity
        |null | not null]]]
    [drop column_name [, column_name]
        [modify column_name {[data_type]
        [[null] | [not null]]}
        [, column_name datatype [null | not null]]]
```

其中：

- *table_name* 是要变更的表。
- *datatype* 是变更的列的数据类型。

必须拥有 sa_role 或是对象所有者才可运行 alter table。有关完整的 alter table 语法，参见*参考手册*。

例如，缺省情况下，authors 表的 au_lname 列使用 varchar(50) 数据类型。要将 au_lname 变更为使用 varchar(60)，输入：

```
alter table authors
modify au_lname varchar(60)
```

注意：变量不能用作 alter table 语句中的缺省值。

删除、修改和添加非空列可能执行数据复制，这意味着需要空间和锁定方案。参见第 261 页上的“数据复制”。

修改的表的页链将继承表的当前配置选项（例如，如果 fillfactor 设置为 50%，新页则拥有相同的 fillfactor）。

注意：Adaptive Server 对 alter table 操作进行部分记录（页分配的）。但是，由于 alter table 作为事务执行，因此运行 alter table 后无法转储事务日志；必须转储数据库以确保其可恢复。如果在 alter table 操作期间服务器遇到任何问题，Adaptive Server 将回退该事务。

alter table 在修改表模式时将获取排它表锁。命令完成后，此锁将立即被释放。

alter table 不引发任何触发器。

使用 *select* * 的对象不列出对表的更改

如果数据库拥有的任何对象（存储过程、触发器等）对删除列的表执行 select *，将出现一条错误消息，列出缺少的列。即使使用 with recompile 选项创建对象，上述情况同样会发生。例如，如果从 authors 表删除了 postalcode 列，所有对此表执行 select * 的存储过程都将发出以下错误消息：

```
Msg 207, Level 16, State 4:
Procedure 'columns'Line 2:
Invalid column name 'postalcode'
(return status = -6)
```

如果添加新列并且随后运行包含 select * 的对象，此消息将不出现；在这种情况下，新列不出现在输出中。

必须删除并重新创建引用删除列的任何对象。

对远程表使用 *alter table*

可使用 alter table 修改使用“组件集成服务 (CIS)”的远程表。修改远程表之前，输入下列命令以确保 CIS 运行：

```
sp_configure "enable cis"
```

如果启用了 CIS，此命令的输出为“1”。缺省情况下，安装 Adaptive Server 时即启用了 CIS。

有关启用配置参数的信息，参见*系统管理指南*。有关使用 CIS 的信息，参见*组件集成服务用户指南*。

添加列

要向现有表添加列，可使用：

alter table *table_name*
add *column_name datatype*
[default {*constant_expression*} | user | null}]
{[{identity | null | not null}] | [constraint *constraint_name*
{*constraint_clause*}]
[, *column_name*]

其中：

- *table_name* 是要添加列的表。
- *column_name* 是要添加的列。
- *constant_expression* 是在用户未指定值（此值必须在引号中，除非字符类型为 int、tinyint、smallint、numeric 或 decimal）的情况下，要在行中放置的常量。
- *constraint_name* 是要添加到表的约束。
- *constraint_clause* 是要添加的约束的全文本。可使用单个 alter table 语句添加任何数量的列。

例如，下列内容将向 authors 表添加名为 author_type 的非空列（它包括常量“primary_author”作为缺省值）和名为 au_publisher 的非空列：

```
alter table authors
add author_type varchar(20)
default "primary_author" not null,
au_publisher varchar(40) null
```

添加列将附加列 ID

alter table 将向表中添加一列，其列 ID 比当前最大列 ID 大一。例如，表 7-5 将列出 salesdetail 表的缺省列 ID：

表 7-5：*salesdetail* 表的列 *ID*

列名	stor_id	ord_num	title_id	qty	discount
列 ID	1	2	3	4	5

此命令将 store_name 列附加到列 ID 为 6 的 salesdetail 表的末端：

```
alter table salesdetail
add store_name varchar(40)
default "unknown" not null
```

如果添加其它列，其列 ID 将为 7。

注意：因为添加或删除列时表的列 ID 会更改，因此应用程序不应依赖于列 ID。

添加 NOT NULL 列

可向表添加 NOT NULL 列。这意味着常量表达式和非空值将在添加列时被放置到该列中。这也确保了对所有现有行在创建表时用指定的常量表达式填充新列。

如果用户向 NOT NULL 列输入值失败，Adaptive Server 将发出错误消息。

以下语句将把列 owner 添加到带缺省值“unknown”的 stores 表：

```
alter table stores
add owner_lname varchar(20)
default "unknown" not null
```

添加 NULL 列时，缺省值可以是常量表达式，但仅当添加 NOT NULL 列时（如上示例所示），缺省值才可以是常量值。

添加约束

要向现有列添加约束，可使用：

alter table *table_name*
add constraint *constraint_name* {*constraint_clause*}

其中：

- *table_name* 是要添加约束的表名称。
- *constraint_name* 是要添加的约束。
- *constraint_clause* 是约束强制的规则。

例如，要向不允许预付款超过 10,000 的 titles 表添加约束：

```
alter table titles
add constraint advance_chk
check (advance < 10000)
```

如果用户试图将大于 10,000 的值插入 titles 表，Adaptive Server 将产生与以下类似的错误消息：

```
Msg 548, Level 16, State 1:
Line 1:Check constraint violation occurred,
dbname = 'pubs2'table name= 'titles'
constraint name = 'advance_chk'
Command has been aborted.
```

添加约束不会影响现有数据。同样，如果添加带有缺省值的新列并对该列指定约束，将不会根据约束对缺省值进行校验。

有关删除约束的信息，参见第 255 页上的“删除约束”。

删除列

使用以下语法从现有表删除列：

alter table *table_name*
drop *column_name* [, *column_name*]

其中：

- *table_name* 是包含要删除列的表。
- *column_name* 是要删除的列。

可使用单个 alter table 语句删除任何数量的列。但是，不能删除表中剩余的最后一列（例如，如果从含有五个列的表中删除了四个列，则不能再删除剩余的一个列）。

例如，要从 titles 表删除 advance 和 contract 列：

```
alter table titles
drop advance, contract
```

当 alter table 删除列时，它将重建表的所有索引。

删除列将对列 ID 进行重新编号

当从表中删除列时，alter table 对列 ID 进行重新编号。列 ID 大于删除列编号的将向上移动一个列 ID，填充该删除列留下的间隙。例如，titleauthor 表包含下列列名和列 ID：

表 *7-6*：*titleauthor* 列 *ID*

列名	au_id	title_id	au_ord	royaltyper
列 ID	1	2	3	4

如果从表中删除 au_ord 列：

```
alter table titleauthor drop au_ord
```

titleauthor 现在具有以下列名和列 ID：

表 *7-7*：删除 *au_ord* 后的列 *ID*

列名	au_id	title_id	royaltyper
列 ID	1	2	3

royaltyper 的列 ID 现在为 3。删除 au_ord 后，title_id 和 royaltyper 上的非聚簇索引也进行了重排。同样，不同系统目录中的列 ID 的所有实例均进行了重新编号。

通常不会注意到列 ID 的重新编号。

注意：因为添加或删除列时表的列 ID 被重新编号，因此应用程序不应依赖于列 ID。如果拥有依赖于列 ID 的存储过程或应用程序，必须对它们进行重写，使其可以访问正确的列 ID。

删除约束

要删除约束，可使用：

alter table *table_name*
　drop constraint *constraint_name*

table_name 和 *constraint_name* 与添加约束中所述的具有同样的值。

例如，要删除上面创建的约束：

```
alter table titles
drop constaint advance_chk
```

要查找有关表约束的详细信息，参见第 273 页上的“使用 sp_helpconstraint 查找表的约束信息”。

修改列

要修改现有列，可使用：

alter table table_name
modify column_name *datatype* [null | not null]
[null | not null]
[, column_name ...]

其中：

- *table_name* 是包含要修改列的表。
- *column_name* 是要修改的列。
- *data_type* 是要将列修改成的数据类型。

可使用 alter table 语句修改任何数量的列。

例如，下列命令将把 titles 表中 type 列的数据类型从 char(12) 更改为 varchar(20) 并使其可为空：

```
alter table titles
modify type varchar(20) null
```

警告！拥有的对象（存储过程、触发器等）可能取决于具有特定数据类型的列。修改列之前，确保所有引用该列的对象在修改之后可以顺利运行。使用 sp_depends 来确定表的相关对象。

可以进行转换的数据类型

只有那些能够隐式或显式转换为新的数据类型，或在 Transact-SQL 中有显式转换函数的数据类型才能被转换。有关支持的数据类型转换的列表，参见第 361 页上的“支持的转换”。如果试图进行非法数据类型修改，Adaptive Server 将发出错误消息并中止该操作。

注意：不能将现有列数据类型转换为时间戳数据类型，也不能将使用时间戳数据类型的列修改为其它数据类型。

如果多次发出相同的 alter table 命令，Adaptive Server 将发出一条消息，类似于：：

```
Warning: ALTER TABLE operation did not affect column
'au_lname'.
Msg 13905, Level 16, State 1:
Server 'SYBASE1', Line 1:
Warning: no columns to drop, add or modify. ALTER
TABLE 'authors' was aborted.
```

修改表可能会妨碍以前转储的成功批量复制

修改列的长度或数据类型可能会妨碍成功使用批量复制在表的旧转储中进行复制。旧表模式可能与新表模式不兼容。在对列的长度或数据类型进行修改之前，应进行校验以确保它不会妨碍在表的以前转储中进行复制。

减少列长度可能会截断数据

如果减少列的长度，应确保减少的列长度不会导致截断数据。例如，可使用 alter table 将 titles 表中 title 列的长度从 varchar(80) 减少到 varchar(2)，但数据就没有意义了：

```
select title from titles
title
-----
Bu
Co
Co
Em
Fi
Is
Li
Ne
```

```
On
Pr
Se
Si
St
Su
Th
Th
Th
Yo
```

Adaptive Server 仅当 set string_rtruncation 选项打开时才发出有关截断列数据的错误消息。如果需要截断字符数据，可设置适当的字符串截断选项并修改列以减少其长度。

修改日期列

如果将列从 char 数据类型修改为 datetime 或 smalldatetime，则无法指定输出中月、日和年的出现顺序。也无法指定输出中使用的语言。相反，两项设置均被指定为缺省值。但可使用 set dateformat 或 set language 变更输出以与存储在列中的信息设置相匹配。同样，Adaptive Server 也不支持将列从 smalldatetime 修改为 char 数据类型。有关详细信息，参见*参考手册*。

修改列的 NULL 缺省值

如果仅更改列的 NULL 缺省值，则不需指定列的数据类型。例如，下列命令将 authors 表中 address 列从 NULL 修改为 NOT NULL：

```
alter table authors
modify address not null
```

如果修改了某列并指定数据类型为 NOT NULL，只要所有行均没有 NULL 值该操作即成功。但如果其中任何行有 NULL 值，操作则失败并回退所有不完整的事务。例如，因为 titles 表中 *The Psychology of Computer Cooking* 包含 NULL 值，所以下列语句失败：

```
alter table titles
modify advance numeric(15,5) not null
Attempt to insert NULL value into column 'advance',
table 'pubs2.dbo.titles';
column does not allow nulls. Update fails.
Command has been aborted.
```

要成功运行此命令，可对表进行更新，将修改列的所有 NULL 值改为 NOT NULL，然后重新发出此命令。

修改具有精度或标度的列

在修改列标度之前检查数据长度。

如果 alter table 命令导致列值损失精度（假定从 numeric(10,5) 变为 numeric(5,5)），Adaptive Server 将中止该语句。如果此语句是批处理的一部分，此选项打开时批处理也将中止。

如果 alter table 命令导致列值损失标度（假定从 numeric(10, 5) 变为 numeric(10,3)），行将被截断且不发出任何警告。无论 arithabort numeric_truncation 开关与否，上述情况都会发生。

如果 arithignore arith_overflow 被打开且 alter table 导致数值溢出，Adaptive Server 将发出警告。但如果 arithignore arith_overflow 是关闭的，当 alter table 导致数值溢出时，Adaptive Server 不会发出警告。缺省情况下，安装 Adaptive Server 时，arithignore arith_overflow 是关闭的。

注意：在产品环境中发出可能截断列长度的命令前，确保已查看数据截断规则并完全了解其提示。应首先在一组测试列上执行这些命令。

修改 *text* 和 *image* 列

text 列只能转换为：

- nchar
- varchar

image 列只能转换为：

- varbinary
- binary

不能将 char 或 varchar 数据类型列修改为 text 列。如果要从 text 转换为 char 或 varchar，则列的最大长度为 255 字节。如果未指定列长度，alter table 将使用缺省长度一字节。如果要修改多字节字符 text 或 image 列，但未指定足够包含数据的列长度，Adaptive Server 将截断数据以适合列长度。

添加、删除和修改 IDENTITY 列

本节说明使用 alter table 添加、删除和修改 IDENTITY 列。有关 IDENTITY 列的概述，参见第 217 页上的“使用 IDENTITY 列”。

添加 IDENTITY 列

可以添加仅有缺省值 NOT NULL 的 IDENTITY 列。不能指定新 IDENTITY 列的缺省子句。

要将 IDENTITY 列添加到表，可在 alter table 语句中指定 identity 关键字：

alter table *table_name* add *column_name*
numeric(*precision* ,0) identity not null

以下示例将 IDENTITY 列 record_id 添加到 stores 表：

```
alter table stores
    add record_id numeric(5,0) identity not null
```

将 IDENTITY 列添加到表时，Adaptive Server 从 1 开始为每行指派一个唯一的顺序值。如果表包含大量行，此进程可能会花费大量时间。如果行数超过列所允许的最大值（在此情况下为 10^5 - 1 或 99,999），alter table 语句失败。

删除 IDENTITY 列

可以象删除其它任何列一样删除 IDENTITY 列。例如，要删除在上一节中创建的 IDENTITY 列：

```
alter table stores
drop record_id
```

但是，以下限制适用于删除标识列：

- 如果在数据库中 sp_dboption "identity in nonunique index" 被打开。

  必须首先删除全部索引，然后删除 IDENTITY 列，再重新创建这些索引。

  如果 IDENTITY 列是隐藏的，则必须首先使用 syb_identity 关键字标识该列。参见第 220 页上的“用 syb_identity 引用 IDENTITY 列”。

- 要从 set identity_insert 已打开的表中删除 IDENTITY 列，首先发出 sp_helpdb 以确定 set identity_insert 是否打开。

必须发出以下命令，关闭 set identity_insert 选项：

```
set identity_insert table_name off
```

删除 IDENTITY 列，然后添加新 IDENTITY 列，并通过输入以下命令打开 set identity_insert 选项：

```
set identity_insert table_name on
```

修改 IDENTITY 列

可以修改 IDENTITY 列的大小以增加其范围。如果当前范围太小或由于服务器关闭而使范围用尽时，则有必要进行上述操作。

例如，可通过输入以下语句来增加 record_id 的范围：

```
alter table stores
modify record_id numberic(9,0)
```

通过为目标数据类型指定较小的精度可减小范围。注意，如果表中的 IDENTITY 值相对目标 IDENTITY 列的范围太大，将引起算术转换操作，同时 alter table 将中止该语句。

不能使用需要数据复制的 alter table 命令将非空 IDENTITY 列添加到已分区表。对于已分区表，数据复制并行完成，无法保证唯一的 IDENTITY 值。

数据复制

Adaptive Server 仅在修改表模式前必须将数据从表中临时复制出时，才执行数据复制。如果表具有索引，Adaptive Server 将在数据复制完成时重建索引。

注意：在 alter table 执行数据复制时，包含该表的数据库中的 select into/bulkcopy/pllsort 必须打开。有关此选项的信息，参见*参考手册*。

Adaptive Server 在下列情况下执行数据复制：

- 删除一列。
- 修改某一列的任何一个属性：
  - 数据类型（不包括增加 varchar、varbinary 或 NULL char 或 NULL binary 列的长度）。
  - 从 NULL 到 NOT NULL，或相反。

- 减小长度。减小列的长度时，也许您事先并不知道长度减小的列是否能够容下所有数据。例如，如果将 au_lname 减小到 varchar(30)，它可能包含一个需要 varchar(35) 的名称。减小列的数据长度时，Adaptive Server 首先复制数据以确保列长度的更改成功。

- 减小编号列的长度（例如，从 tinyint 到 int）。当此列中某一行的值为 NOT NULL 时，Adaptive Server 执行数据复制。
- 添加 NOT NULL 列。

下列情况下 alter table 不执行数据复制：

- 更改 varchar 或 varbinary 列的长度。
- 更改用户定义的数据类型 ID，但物理数据类型不变。例如，如果站点有两种数据类型，mychar1 和 mychar2，它们有不同的用户定义数据类型，但物理数据类型相同，则将 mychar1 改为 mychar2 时，不会发生数据复制。
- 将可变长度列的 NULL 缺省值从 NOT NULL 改为 NULL。

要确定 alter table 是否执行数据复制：

1. 将 showplan 设置为打开，报告 Adaptive Server 是否将执行数据复制。
2. 将 noexec 设置为打开，确保不执行任何操作。
3. 如果不需要任何数据复制，则执行 alter table 命令，仅执行目录更新以反映 alter table 命令所引起的改变。

修改 *exp_row_size*

在执行数据复制时，也可更改 exp_row_size，它用于指定每行的空间。仅在被修改的表模式包含可变长度列时，才能更改 exp_row_size，并且只能在被修改表的 sysindexes 中的 maxlen 和 minlen 值指定的范围内进行修改。

如果列中有多个固定长度的列，则仅可将 exp_row_size 改为 0 或 1。如果从表中删除所有可变长度列，必须指定 exp_row_size 为 0 或 1。同样，如果没有为 alter table 命令提供 exp_row_size，则使用旧的 exp_row_size。如果表仅包含固定长度的列且旧的 exp_row_size 与修改模式不兼容，Adaptive Server 将发出错误消息。

不能将 exp_row_size 子句与其它任何 alter table 子子句一同使用（例如，定义约束、更改锁定方案等）。也可使用 sp_chgattribute 来更改 exp_row_size。有关更改用于 alter table 和 sp_chg attribute 的 exp_row_size 的详细信息，参见*参考手册*。

修改锁定方案和表模式

如果 alter table 执行数据复制，则也可同时使用一个更改表的锁定方案的命令。例如，要修改 authors 表的 au_lname 列并将此表的锁定方案从 allpages 锁更改为数据行锁定：

```
alter table authors
modify au_lname varchar(10)
lock datarows
```

但不能使用 alter table 更改具有聚簇索引的表的表模式和锁定方案。如果表具有聚簇索引，则可：

1. 删除索引。
2. 修改表模式并在同一语句中更改锁定方案（如果表模式的更改也包括数据复制）。
3. 重建聚簇索引。

或者，也可执行 alter table 命令更改锁定方案，然后执行另一 alter table 命令更改表模式。

变更具有用户定义数据类型的列

可使用 alter table 添加、删除或修改使用用户定义数据类型的列。

添加具有用户定义数据类型的列

添加具有用户定义数据类型的列的语法，与添加具有系统定义数据类型的列相同。例如，要使用 usertype 数据类型将列添加到 pubs2 的 authors 表：

```
alter table titles
add newcolumn usertype not null
```

用户指定的 NULL 或 NOT NULL 缺省值的优先级高于用户定义数据类型所指定的缺省值。也就是说，如果添加了一列并指定 NOT NULL 为缺省值，则新列的缺省值为 NOT NULL，即使用户定义数据类型指定为 NULL。如果没有指定 NULL 或 NOT NULL，则使用用户定义数据类型指定的缺省值。

添加 not null 的列时，必须提供缺省设置子句，即使已有缺省值绑定到用户定义数据类型。

如果用户定义数据类型指定了 IDENTITY 列属性（精度和标度），则该列被作为 IDENTITY 列添加。

删除具有用户定义数据类型的列

使用与删除具有系统定义数据类型的列相同的方式删除具有用户定义数据类型的列。

修改具有用户定义数据类型的列

修改列使其包含用户定义数据类型的语法与修改列使其包含系统定义数据类型的语法相同。例如，要修改 authors 表的 au_lname，使其使用用户定义的 newtype 数据类型：

```
alter table authors
modify au_lname newtype(60) not null
```

如果没有指定缺省值为 NULL 或 NOT NULL，则列使用用户定义数据类型指定的缺省值。

修改表并不影响绑定到列的任何现行规则或缺省设置。但是，如果指定了新规则或缺省设置，则任何旧规则或绑定到用户定义数据类型的缺省设置都将被删除。如果以前没有任何旧规则或缺省设置绑定到列，则应用任何用户定义的规则和缺省设置。

不能将现有的列改为 IDENTITY 列。只能使用具有 IDENTITY 列属性（精度和标度）的用户定义数据类型修改现有的 IDENTITY 列。

alter table 生成的错误和警告

在运行 alter table 时出现的大多数错误都会告诉您模式结构阻止了所请求命令的执行（例如，如果尝试删除属于索引一部分的列）。在重新发出命令前，必须解决指向与受影响列相关的模式对象的错误或警报。要报告错误情况：

1　将 showplan 设置为打开。

2　将 noexec 设置为打开。

3　执行 alter table 命令。

在更改命令以解决任何报告出的错误后，请将 showplan 和 noexec 设置为关闭，以便 Adaptive Server 能够执行实际操作。

实际运行命令时，alter table 会检测出并报告某些错误（例如，要删除存在参照约束的某列）。所有运行期数据相关的错误（例如，数值溢出错误、字符截断错误等）只能在执行语句时才能识别出来。必须更改命令以适应可用的数据，或修改数据值以便使用语句指定的、必需的目标数据类型。要识别出这些错误，在运行命令时，noexec 必须关闭。

alter table modify 生成的错误和警告

有些错误仅由 alter table modify 命令生成。虽然 alter table modify 用于将列转换为兼容的数据类型，但如果被转换的列具有某些约束，则 alter table 也能发出错误消息。

注意：确保在发出命令前了解修改数据类型的潜在影响。通常，只将 alter table modify 用于可转换数据类型之间的隐式转换。这样可确保 insert 和 update 语句处理期间必需的隐藏转换不会因为数据类型不兼容而失败。

例如，如果将 second_advance 列添加到数据类型为 int 的 titles 表，并为 second_advance 创建聚簇索引，则不能将此列的数据类型修改为 char。这将使 int 值从整数 (1, 2, 3) 转换为字符串 (‘1’, ‘2’, ‘3’)。用已排序数据重建索引后，数据值应按排序顺序排列。但在此例中，数据类型已经从 int 改为 char，且不再按 char 数据类型的排序顺序排列。因此，alter table 命令将在重建索引阶段失败。

如某列属于聚簇索引的索引键列的一部分，则在为其选择新数据类型时，要格外谨慎。alter table modify 必须指定一个在其数据复制阶段结束后，与已修改数据值的排序顺序不发生冲突的目标数据类型。

如果在包含约束的列中将数据类型修改为不兼容的数据类型，则 alter table modify 也将发出警告消息。例如，如果尝试将数据类型 char 修改为数据类型 int，且该列包含一个约束，则 alter table modify 发出以下警告：

```
Warning: a rule or constraint is defined on column
```

```
'new_col' being modified. Verify the validity of
rules and constraints after this ALTER TABLE
operation.
```

此警告说明在约束期望的值与列 new_col 的已修改的数据类型之间可能存在数据类型不一致：

```
Warning: column 'new_col' is referenced by one or
more rules or constraints. Verify the validity of the
rules/constraints after this ALTER TABLE operation.
```

如果尝试将数据类型为 char 的数据插入此表，则操作将失败并显示以下消息：

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

检查约束的定义期望列的数据类型为 int，但要插入数据的数据类型是 char。

而且，不能将数据作为类型 int 插入，因为列现在使用的数据类型为 char。insert 返回以下消息：

```
Msg 257, Level 16, State 1: Line 1: Implicit conver-
sion from datatype 'INT' to 'CHAR' is not allowed.
Use the CONVERT function to run this query.
```

modify 操作非常灵活，但必须谨慎使用。一般情况下，修改为可隐式转换数据类型的操作不会出错。修改为可显式转换数据类型的操作可能导致表模式中的不一致。修改列的数据类型前，请使用 sp_depends 确定所有列级依赖性。

if exists()...alter table 生成的脚本

如果在包括类似如下结构的脚本中定义的表不包含指定的列，则该脚本可能会出错：

```
if exists (select 1 from syscolumns
    where id = object_id("some_table")
        and name = "some_column")
begin
        alter table some_table drop some_column
end
```

在此例中，some_table 中*必须*有 some_column，批处理才能成功。

如果 some_table 中有 some_column，则第一次运行批处理时，alter table 将删除该列。在后续的执行中，批处理将不编译。

Adaptive Server 在预处理此批处理时发现这些错误，这些错误与常规 select 试图访问不存在的列时发现的错误类似。使用需要数据复制的子句修改表模式时，将引起这些错误。如果添加空列并使用上述结构，则 Adaptive Server 不会提示这些错误。

要在修改表模式时避免这样的错误，可将 alter table 包括在 execute immediate 命令中：

```
if exists (select 1 from syscolumns
       where id = object_id("some_table")
         and name = "some_column")
begin
    exec ("alter table some_table drop
        some_column")
end
```

因为 execute immediate 语句只能在 if exists() 函数成功后运行，所以 Adaptive Server 在编译此脚本时不会发现任何错误。

也需要将 execute immediate 结构用于 alter table 的其它用途（例如，更改锁定方案），以及该命令不要求数据复制的任何其它情况。

重命名表和其它对象

要重命名表和其它数据库对象 — 列、约束、数据类型、视图、索引、规则、缺省设置和触发器 — 请使用 sp_rename。

必须拥有对象，才能重命名它。不能更改系统对象或系统数据类型的名称。数据库所有者可更改任何用户对象的名称。另外，要更改名称的对象必须在当前数据库中。

要重命名数据库，请使用 sp_renamedb。参见*参考手册*中的 sp_renamedb。

sp_rename 的语法为：

sp_rename *objname*, *newname*

例如，要将 friends_etc 的名称更改为 infotable：

```
sp_rename friends_etc, infotable
```

要重命名列，请使用：

sp_rename "*table.column*", *newcolumnname*

新列名中必须去掉表名前缀，否则新名将不被接受。

要更改索引的名称，请使用：

sp_rename "*table.index*", *newindexname*

不要将表名包括在新名称中。

要将用户数据类型 tid 的名称更改为 t_id，请使用：

```
exec sp_rename tid, "t_id"
```

重命名相关对象

重命名对象时，也必须更改任何相关过程、触发器或视图的文本以反映新对象名。原始对象名仍出现在查询结果中，直到更改了过程、触发器或视图的名称并对它们进行了编译。最安全的做法是在执行 sp_rename 时更改所有**相关**对象的定义。可使用 sp_depends 获得相关对象的列表。

可使用 defncopy 实用程序将过程、触发器、规则、缺省设置和视图的定义复制到操作系统文件中。编辑此文件以改正对象名，然后使用 defncopy 将这些定义复制回 Adaptive Server 中。有关 defncopy 的详细信息，参见*实用程序指南*。

删除表

使用 drop table 从数据库中删除表。

drop table [[*database*.]*owner*.] *table_name*
[, [[*database*.]*owner*.] *table_name*]...

此命令从数据库中删除指定的表，连同表中的内容以及相关的所有索引与特权。原来绑定到该表的规则和缺省值将不再绑定，其它方面均无变化。

只有表的所有者才能删除它。然而，在用户或前端程序对表进行读写操作时，任何人都不能将删除它。不能对任何系统表使用 drop table 命令，无论是在 master 数据库还是在用户数据库中。

可以删除另一数据库中的表，只要您是表的所有者。

如果使用 delete 删除表中的所有行，或对它使用 truncate table 命令，则该表会始终存在直到使用 drop 将其删除。

drop table 和 truncate table 权限不能移交给其它用户。

给用户分配权限

grant 和 revoke 命令控制 Adaptive Server 命令和对象保护系统。可使用 grant 命令将各种权限分配给用户、组和角色，并使用 revoke 命令撤消这些权限，包括：

- 创建数据库
- 在数据库中创建对象
- 访问表、视图和列
- 执行存储过程

某些命令可由所有用户随时使用，不需要任何权限。其它命令只能由具有某种身份（例如，系统管理员）的用户使用，且这种权限不能移交。

是否可以分配可被授予或撤消的、使用某些命令的权限，由每个用户的身份（如系统管理员、数据库所有者或数据库对象所有者）以及是否授予特定用户将该权限授予其他用户的权限来决定。

数据库所有者并不自动接受其他用户所拥有的对对象的权限。但数据库所有者或系统管理员可通过使用 setuser 命令临时使用对象所有者的身份并编写相关的 grant 或 revoke 语句来获得任何权限。

用 grant 和 revoke 可分配两种权限：**对象访问权限**和**对象创建权限**。

对象访问权限控制访问某些数据库对象的某些命令的使用。例如，必须获得对 authors 表使用 select 命令的权限。对象访问权限由对象所有者授予或撤消。

要授予 Mary 和 Joe 对 titles 表使用 insert 和 delete 命令进行插入和删除操作的对象访问权限：

```
grant insert, delete
on titles
to mary, joe
```

对象创建权限控制创建对象的命令的使用。这些权限只能由系统管理员或数据库所有者授予。

例如，要撤消 Mary 在当前数据库中创建表和规则的权限：

```
revoke create table, create rule
from mary
```

有关使用 grant 和 revoke 的完整信息，参见*系统管理指南*。

系统安全员也可使用角色，简化授予和撤消访问权限的任务。

例如，系统安全员可创建角色，请求对象所有者授权给每一角色，并根据雇员在公司中的职责将这些用户定义的角色授予他们，而不是让对象所有者将每一对象的特权分别授予每一雇员。系统安全员还可以撤消授予雇员的用户定义角色。

有关用户定义角色的完整信息，参见*系统管理指南*。

获得有关数据库和表的信息

Adaptive Server 有几个可用于获得有关数据库、表和其它数据库对象信息的过程和函数。本节描述其中一些。有关完整信息，参见*参考手册*。

获得有关数据库的帮助

sp_helpdb 可报告有关指定数据库或所有 Adaptive Server 数据库的信息。它报告分配给数据库的每个片段的名称、大小和使用情况。

sp_helpdb [*dbname*]

例如，要显示有关 pubs2 的报告：

```
sp_helpdb pubs2

name    db_size   owner  dbid created          status
-----   -------   ------ ---- ----------------   ---
----------
pubs2   2 MB      sa     4    Jun 18 1997       no
options set

(1 row affected)

device              size         usage
-----------------   -----------  --------------
pubsdev             2 MB         data + log

(1 row affected)
```

sp_databases 列出服务器中的所有数据库。例如：

```
sp_databases
database_name     database_size  remarks
----------------- -------------  ------------
master                     5120  NULL
```

```
model                        2048  NULL
pubs2                        2048  NULL
pubs3                        2048  NULL
sybsecurity                  5120  NULL
sybsystemprocs              30720  NULL
tempdb                       2048  NULL

(7 rows affected, return status = 0)
```

要查找数据库的所有者，请使用 sp_helpuser：

```
sp_helpuser dbo
Users_name  ID_in_db  Group_name Login_name
------------- -------- ------------ ------------
dbo            1           public       sa

(return status = 0)
```

使用 db_id() 和 db_name() 来确定当前数据库。例如：

```
select db_name(), db_id()

------------------------------ ------
master                                1
```

获得有关数据库对象的帮助

Adaptive Server 提供系统过程、分类存储过程和内部函数，它们可返回有关数据库对象（如表、列和约束）的有用信息。

对数据库中对象使用 *sp_help*

使用 sp_help 可显示有关指定数据库对象（即在 sysobjects 中列出的任何对象）、指定数据类型（在 systypes 中列出）、或当前数据库中的所有对象和数据类型的信息。

sp_help [*objname*]

下面是针对 publishers 表的输出：

```
Name                          Owner       Type
---------------------------  -----------  ----------
publisher                    dbo          user table

(1 row affected)

Data_located_on_segment            When_created
------------------------------     ----------------
```

```
----
default                          Jul 7 1997 1:43PM

Column_name Type     Length   Prec  Scale
----------- -------  ------   ----- -----
pub_id      char          4    NULL  NULL
pub_name    varchar      40    NULL  NULL
city        varchar      20    NULL  NULL
state       char          2    NULL  NULL

Nulls       Default_name   Rule_name     Identity
-----       -------------  ---------     --------
    0       NULL           pub_idrule           0
    1       NULL           NULL                 0
    1       NULL           NULL                 0
    1       NULL           NULL                 0

index_name index_description   index_keys
index_max_rows_per_page
---------- ------------------  ---------- ---------
--------------
pubind     clustered, unique   pub_id     0
           located on default

(1 row affected)

keytype  object      related_object  related_keys
-------  ----------  --------------  ------------
primary  publishers  -- none --      pub_id, *, *,
*, *, *
foreign  titles      publishers      pub_id, *, *,
*, *, *

(1 row affected)

Object is not partitioned.

(return status = 0)
```

如果执行 sp_help 而没有提供对象名，则报告将显示 sysobjects 中的每个对象，以及其名称、所有者和对象类型。也显示 systypes 中的每个用户定义的数据类型及其名称、存储类型、长度、是否允许空值，以及绑定到它的任何缺省值或规则。该报告也注明是否已经为表或视图定义了任何主键或**外键**列。

sp_help 列出表中的任何索引，包括通过定义唯一或主键约束创建的索引。但不提供关于为表定义的完整性约束的任何信息。有关任何完整性约束的信息，使用 sp_helpconstraint 显示。

使用 *sp_helpconstraint* 查找表的约束信息

sp_helpconstraint 报告有关为表指定的声明参照完整性约束的信息，包括约束名称和缺省设置的定义、唯一或主键约束、参照或检查约束。sp_helpconstraint 也报告与指定表相关联的引用的数目。

其语法为：

sp_helpconstraint [*objname*] [, detail]

objname 是被查询的表的名称。如果不包括表名，则 sp_helpconstraint 显示与当前数据库中每个表相关联的引用的数目。如果包括表名，则 sp_helpconstraint 报告与该表相关联的完整性约束的名称、定义和数目。detail 选项也返回有关约束的用户或错误消息的信息。

例如，假定对 pubs3 中的 store_employees 表运行 sp_helpconstraint。

```
name                          defn
---------------------------   ----------------------
----------
store_empl_stor_i_272004000   store_employees FOR-
EIGN KEY
                              (stor_id) REFERENCES
stores(stor_id)
store_empl_mgr_id_288004057   store_employees FOR-
EIGN KEY
                             (mgr_id) SELF REFERENCES
                              store_employees(emp_id)
store_empl_2560039432         UNIQUE INDEX( emp_id) :
                               NONCLUSTERED, FOREIGN
REFERENCE

(3 rows affected)

Total Number of Referential Constraints: 2
Details:
-- Number of references made by this table: 2
-- Number of references to this table: 1
-- Number of self references to this table: 1

Formula for Calculation:
Total Number of Referential Constraints
= Number of references made by this table
+ Number of references made to this table
- Number of self references within this table
```

要找出与当前数据库中任何表相关联的参照约束的最大数目，请不指定表名而运行 sp_helpconstraint，例如：

```
sp_helpconstraint

id          name
Num_referential_constraints
----------- ------------------------ --------------
-------------
   80003316 titles
4
   16003088 authors
3
  176003658 stores
3
  256003943 salesde-
tail                                        3
  208003772 sales
2
  336004228 titleau-
thor                                        2
  896006223 store_employees
2
   48003202 publish-
ers                                         1
  128003487 roy-
sched                                           1
  400004456 dis-
counts                                          1
  448004627 au_pix
1
  496004798 blurbs
1

(11 rows affected)
```

此报告中，在 pubs3 数据库中表 titles 具有最大数目的参照约束。

查找表占用的空间

使用 sp_spaceused 找出表占用的空间：

sp_spaceused [*objname*]

sp_spaceused 计算并显示表或聚簇或非聚簇索引使用的行和数据页的数量。

要显示 titles 表使用空间的报告：

```
sp_spaceused titles
name    rows   reserved  data  index_size  unused
```

```
------- -----  --------- ----- ---------   ------
titles       18        48 KB   6 KB     4 KB     38 KB

 (0 rows affected)
```

如果没有将对象名包括在内，则 sp_spaceused 显示所有数据库对象使用空间的汇总。

列出表、列和数据类型

分类存储过程用于在表格形式的系统表中检索信息。可为某些参数提供通配符。

按下列格式使用 sp_tables 时，它列出数据库中所有的用户表：

sp_tables @table_type = 'TABLE'

sp_columns 返回数据库中一个或多个表的任一或所有列的数据类型。可使用通配符获得多个表或列的信息。

例如，下列命令返回所有名称中含“sales”的表中的、包括字符串“id”的所有列的信息：

```
sp_columns "%sales%", null, null, "%id%"

table_qualifier table_owner
       table_name     column_name
       data_type type_name  precision  length
scale radix  nullable
       remarks

ss_data_type colid
---------------  -----------
      ----------       -----------
      ---------  --------- ---------  ------  -----
-----  --------
      -------

------------ -----
pubs2              dbo
       sales           stor_id
      1          char       4          4       NULL
NULL    0
NULL

47            1
pubs2            dbo
       salesdetail     stor_id
```

```
        1         char        4           4       NULL
NULL    0
NULL

4             1
pubs          dbo
        salesdetail     title_id
      12     varchar     6         6      NULL  NULL 0
    NULL

39            3

(3 rows affected, return status = 0)
```

查找对象名和 ID

使用 object_id() 和 object_name() 来确定对象的 ID 和名称。例如：

```
select object_id("titles")
----------
  208003772
```

对象名和 ID 存储在 sysobjects 系统表中。

第 8 章

添加、更改和删除数据

在创建数据库、表和索引后，可以将数据置于表中并处理它 — 根据需要添加、更改和删除数据。

本章涉及以下主题：

主题	页
简介	277
数据类型输入规则	280
添加新数据	287
更改现有数据	299
改变 text 和 image 数据	304
删除数据	305
删除表中的所有行	307

简介

用于添加、更改或删除数据的命令称为**数据修改语句**。这些命令有：

- insert — 在表中添加新行。
- update — 更改表中已有的行。
- writetext — 添加或更改 text 和 image 数据，而不用对系统事务日志进行冗长的写操作。
- delete — 从表中删除指定的行。
- truncate table — 从表中删除所有的行。

有关这些命令的详细信息，参见*参考手册*。

也可使用批量复制实用程序 bcp，通过从文件中传送数据，将数据添加到表中。有关您平台的详细信息，参见*实用程序指南*。

可使用 insert、update 或 delete 修改每个语句中一个表的数据。Transact-SQL 对这些命令的增强作用表现在，能根据其它表（甚至其它数据库）中的数据进行修改。

数据修改命令也能执行于视图上，但是有一些限制。参见第 9 章“视图：限制访问数据”。

权限

数据修改命令不必每个人都可用。数据库所有者和数据库对象所有者能使用 grant 和 revoke 命令，指定能够修改数据的用户。

可将使用数据修改命令的任意组合的权限或特权，授予单个用户、组或公众。这些权限在*系统管理指南*讨论。

参照完整性

insert、update、delete、writetext，和 truncate table 允许更改数据而不更改其它表中相关数据，但是，可能导致不一致。

例如，如果更改 authors 表中“Sylvia Panteley”的 au_id 条目，则必须同样更改 titleauthor 表中的条目，以及数据库中所有其它有包含该值的列的表。否则，就查不到诸如 Panteley 女士的书名之类的信息，因为不能连接到她的 au_id 列。

在数据库中，保持数据修改在所有表中的一致称为**参照完整性**。管理它的一个方法是为表定义参照完整性约束。另一个方法是创建称为触发器的特殊过程，当对特定的表或列给出 insert、update 和 delete 命令时，该触发器生效（truncate table 命令不会被触发器或参照完整性约束捕获）。参见第 16 章“触发器：强制实施参照完整性”和第 7 章“创建数据库和表”。

要从参照完整性表中删除数据，首先更改被引用的表，然后更改引用表。

事务

受每条数据修改语句影响的每行的新旧状态，被写入事务日志。这意味着，如果通过发出 begin transaction 命令开始事务，而后意识有错误发生并回退事务，则数据库恢复到以前的状态。

注意：不能回退在远程 Adaptive Server 上，通过远程过程调用（RPC）所作的更改。

当 select/into bulkcopy 数据库选项设定为 false 时，writetext 是一个例外。

writetext 的缺省操作模式并*不*记录事务日志。这可避免事务日志被极长的数据块填满，text 和 image 字段可能包含这些数据块。要记录此命令所作更改的日志，必须使用 writetext 命令的 with log 选项。

第 18 章“事务：维护数据一致性和恢复”中对事务进行了更完整的讨论。

使用样本数据库

如果按照本章中的示例操作，Sybase 建议从 pubs2 或 pubs3 数据库的原始副本开始，并在完成时使其返回该状态。请寻求系统管理员的帮助，获得任一这些数据库的原始副本。

为了防止所作的任何更改变成永久性更改，可以通过将所有输入的语句涵括在一个事务中，然后在完成本章后终止该事务。例如，键入以下语句启动事务：

```
begin tran modify_pubs2
```

此事务命名为 modify_pubs2。可以通过键入以下语句，在任何时候取消事务并将数据库返回其在事务开始前的状态：

```
rollback tran modify_pubs2
```

数据类型输入规则

Adaptive Server 提供的几种数据类型对数据的输入和搜索有特殊规则。有关数据类型的详细信息，参见第 7 章“创建数据库和表”。

char、*nchar*、*unichar*、*univarchar*、*varchar*、*nvarchar* 和 *text*

所有 character、text 和 datetime 数据，在首次输入和进行搜索时，必须用单引号或双引号引起来。如果 set 命令的 quoted_identifier 选项设定为 on，则使用单引号。如果使用双引号，Adaptive Server 视文本为标识符。

如果输入字符串的长度大于 char、nchar、unichar、univarchar、varchar 或 nvarchar 列的指定长度，则将截断这些字符串。设定 string_rtruncation 选项为 on，可在出现这种情况时收到警告消息。

有两种方法可以指定字符输入中的文字引号：

- 使用两个引号。例如，如果在某个字符条目开头用了一个单引号，并要将一个单引号作为该条目的一部分，则可使用两个单引号：‘I don’‘t understand.’ 对于双引号：“He said, ““It’s not really confusing.”””
- 用将被引的材料括在相反类型的引号中。也就是说，用单引号括起包含双引号的条目，或相反。例如：‘George said, “There must be a better way.” ’

如果输入的字符串超出屏幕宽度，则在转到下一行之前输入反斜杠 (\)。

使用 like 关键字和在第 2 章“查询：从表中选择数据”中说明的通配符，搜索 character、text 和 datetime 数据。

关于插入 text 数据以及在 character 数据中尾随空白的详细信息，参见*参考手册*。

datetime 和 *smalldatetime*

datetime 数据的显示和输入格式提供了范围广泛的数据输出格式，并识别多种输入格式。显示和输入格式是分别控制的。缺省的输出格式提供类似“Apr 15 1997 10:23PM”的输出。convert 命令提供显示秒和毫秒的选项，以及用其它日期分量顺序显示日期的选项。有关显示日期值的详细信息，参见第 10 章“使用查询中的内部函数”。

Adaptive Server 识别范围广泛的日期数据输入格式。始终忽略大小写，并且空格可在日期分量间的任何地方出现。当输入 datetime 和 smalldatetime 值时，始终用单引号或双引号将它们括起来。如果 quoted_identifier 选项设置为 on，则使用单引号；如果使用双引号，Adaptive Server 视文本为标识符。

Adaptive Server 分别识别数据的两种日期和时间选项，因此时间可以位于日期之前或之后。可以忽略任何一部分，在这种情况下，Adaptive Server 使用缺省值。缺省的日期和时间为 January 1, 1900, 12:00:00:000AM.

对于 datetime，可使用的最早日期为 1753 年 1 月 1 日；最迟日期为 9999 年 12 月 31 日。对于 smalldatetime，可使用的最早日期为 1900 年 1 月 1 日；最迟日期为 2079 年 6 月 6 日。比这些日期早或迟的日期必须输入、存储并处理为 char，或 unichar ；或 varchar 或 univarchar 值。Adaptive Server 拒绝在这些范围内不能识别为日期的所有值。

输入时间

各时间成分的顺序对于数据的时间部分非常重要。首先输入小时；然后分；然后秒；然后毫秒；最后 AM（或 am）或 PM (pm)。12AM 是午夜，12PM 是正午。要被识别为时间，时间值必须包含一个冒号，或者是 AM 或 PM 标记。smalldatetime 仅精确到分钟。

毫秒前可以带一个冒号或句点。如果前面带冒号，数字就表示千分之多少秒。如果前面带句号，单个数字位表示十分之多少秒，两个数字位表示百分之多少秒，三个数字位表示千分之多少秒。

例如，“12:30:20:1”表示 12:30 过二十又千分之一秒；“12:30:20.1”表示 12:30 过二十又十分之一秒。

可接受的时间数据格式为：

```
14:30
14:30[:20:999]
14:30[:20.9]
4am
4 PM
[0]4[:30:20:500]AM
```

输入日期

当日期以有分隔符的数字字符串输入时，set dateformat 命令指定日期分量的顺序（月、日和年）。根据指定语言缺省的日期格式，set language 也能影响日期的格式。缺省语言为美国英语，缺省日期格式为 *mdy*。有关详细信息，参见*参考手册*中的 set 命令。

注意：dateformat 只影响以带分隔符的数字输入的日期，例如“4/15/90”或“20.05.88”。它不影响月份以字母格式提供的日期，例如“April 15, 1990”，或没有分隔符的日期，例如“19890415”。

日期格式

Adaptive Server 识别三种基本的日期格式，如下所述。每种格式必须用引号引起来，并且能在前面或后面跟上时间规范，如第 281 页上的“输入时间”中所述。

- 月份按字母格式输入。
  - 用字母指定日期的有效格式为：

    ```
    Apr[il] [15][,] 1997
    Apr[il] 15[,] [19]97
    Apr[il] 1997 [15]
    [15] Apr[il][,] 1997
    15 Apr[il][,] [19]97
    15 [19]97 apr[il]
    [15] 1997 apr[il]
    1997 APR[IL] [15]
    1997 [15] APR[IL]
    ```

  - 如当前语言规范所规定，月份可以是三个字符的缩写，也可以是完整的月名称。
  - 逗号是可选的。

- 忽略大小写。
- 如果只指定年的最后两个数字，则小于 50 的值解释为“20yy”，大于或等于 50 的值解释为“19yy”。
- 只有在日被忽略，或需要的不是缺省世纪时，才输入世纪。
- 如果日缺少，Adaptive Server 将该月的第一天作为缺省值。
- 以字母格式指定月时，dateformat 设置被忽略（参见*参考手册*中的 set 命令）。

- 月份在带有斜杠 (/)、连字符 (-)、或句点 (.) 分隔符的字符串中以数字格式输入。
  - 必须指定月、日、年。
  - 字符串必须是如下格式：

    <num> <sep> <num> <sep> <num> [<time spec>]

    或：

    [<time spec>] <num> <sep> <num> <sep> <num>

  - 对日期分量值的解释取决于 dateformat 的设置。如果顺序与设置不匹配，则值将会因为超出了范围而不被解释为日期，或值将被误解。例如，“12/10/08”可以解释为六个不同日期，这取决于 dateformat 设置。有关详细信息，参见*参考手册*中的 set 命令。
  - 要按 *mdy* dateformat 格式输入“1997 年 4 月 15 日”，可以使用以下这些格式：

    ```
    [0]4/15/[19]97
    [0]4-15-[19]97
    [0]4.15.[19]97
    ```

  - 用“/”作为分隔符的其它输入顺序如下所示，也可以使用连字符或句点：

    ```
    15/[0]4/[19]97 (dmy)
    1997/[0]4/15 (ymd)
    1997/15/[0]4 (ydm)
    [0]4/[19]97/15 (myd)
    15/[19]97/[0]4 (dym)
    ```

- 以 4 位、6 位或 8 位无分隔符的字符串给出日期，或以空字符串给出日期，或仅给出时间值而没有日期值。
  - 对于此输入格式，dateformat 总是被忽略。
  - 如果给出四位数字，则该字符串被解释为年，月设定为一月，天为月的第一天。不能忽略世纪。
  - 六位或八位数字字符串总是解释为*ymd*；月和天必须总是两位数字。下列格式是可识别的： [19]960415.
  - 空字符串(“”)或缺失的日期被解释为基准日期，1900年1月1日。例如，象“4:33”这样没有日期的时间值被解释为“1900 年 1 月 1 日，4:33AM”。

当 weekday 或 dw 与 datename 一起使用时，set datefirst 命令指定一周内的天数（星期日、星期一，等等），当与 datepart 一起使用时，指定一个相应的数字。用 set language 更改语言也影响日期的格式，取决于该语言缺省的周的第一天。对于缺省语言 us_english，缺省的 datefirst 设置是 Sunday=1、Monday=2，等等；而对其它语言则为 Monday=1、Tuesday=2 等等。缺省设置可在每个会话期用 set datefirst 更改。有关详细信息，参见*参考手册*中的 set 命令。

搜索日期和时间

可将 like 关键字和通配符用于 datetime 和 smalldatetime 数据，以及 char、unichar、nchar、varchar、univarchar、nvarchar 和 text。当将 like 用于 datetime 或 smalldatetime 值时，Adaptive Server 首先将日期转换为标准的 datetime 格式，然后转换为 varchar 或 univarchar。由于标准的显示格式不包括秒或毫秒，所以不能用 like 和匹配模式来搜索秒或毫秒。使用类型转换函数 convert 来搜索秒和毫秒。

由于 datetime 或 smalldatetime 条目可能包含多种日期分量，所以当搜索 datetime 或 smalldatetime 值时，最好使用 like。例如，如果将值“9:20” 插入名为 arrival_time 的列，则以下子句将找不到它，因为 Adaptive Server 将该条目转换为 “Jan 1, 1900 9:20AM”：

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

但是，下列子句将找到它：

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

如果使用了 like，并且月的天小于 10，则必须在月份和日之间插入两个空格，以与 datetime 值的 varchar 转换相匹配。同样，如果小时小于 10，转换结果将在年份和小时之间放置两个空格。子句 like May 2% 在“May”和“2”之间有一个空格，将找出从 5 月 20 日到 5 月 29 日的所有日期，而不是 5 月 2 日。由于 datetime 值仅对于 like 比较才会转换为 varchar，因此无须在其它日期比较中插入额外的空格，只有使用 like 时才需要这样做。

binary、*varbinary* 和 *image*

当输入 binary、varbinary 或 image 数据时，数据必须加前缀“0x”。例如，要输入“FF”，键入“0xFF”。

如果输入的字符串比 binary 或 varbinary 列指定的长度长，输入将被截断而不显示警告信息。

binary 或 varbinary 列的长度为 10，表明 10 个字节，每个字节存储 2 个十六进制数字。

当在 binary 或 varbinary 列上创建缺省值时，在其前面加上“0x”。

关于十六进制值中尾随零的信息，参见*参考手册*。

money 和 *smallmoney*

输入的带有 E 符号的货币值被解释为 float。当一个条目作为 money 或 smallmoney 值存储时，可能会导致拒绝该条目或丧失该条目的某些精度。

输入 money 和 smallmoney 值时可以在前面加上货币符号，也可以不加，例如美元符号 ($)、日元符号 (￥) 或英镑符号 (￡)。如果要输入负值，请在货币符号后加上一个负号。在条目中不要包含逗号。

不能带逗号输入 money 或 smallmoney 值，尽管 money 或 smallmoney 数据的缺省输出格式为每三个数字后放一个逗号。当显示 money 或 smallmoney 值时，它们将舍入到最接近的分。除了模运算外的所有算术运算都可用于 money。

float、*real*，和 *double precision*

将近似数字类型 — float、real 和 double precision — 作为后跟可选指数的**尾数**输入。尾数可包含一个正号或负号以及一个小数点。“e”或“E”后的指数，可包含一个符号但不能有一个小数点。

为了计算近似数字数据，Adaptive Server 将尾数乘以 10 的给定指数次幂。表 8-1 显示了 float、real 和 double precision 数据的示例：

表 *8-1*：计算数值数据

输入数据	尾数	指数	值
10E2	10	2	$10 * 10^{2}$
15.3e1	15.3	1	$15.3 * 10^{1}$
-2.e5	-2	5	$-2 * 10^{5}$
2.2e-1	2.2	-1	$2.2 * 10^{-1}$
+56E+2	56	2	$56 * 10^{2}$

列的二进制精度决定了尾数中所允许的二进制位的最大位数。对于 float 列，可以指定高达 48 位的精度；对于实数和双精度列，其精度取决于计算机。如果值超过了列的二进制精度，Adaptive Server 将该条目标记为一个错误。

decimal 和 *numeric*

精确的数值类型 — dec、decimal 和 numeric — 由可选的正号或负号开始，并可包含小数点。精确数据的数值取决于列的小数 *precision* 和 *scale*，可以用以下语法指定：

datatype [(*precision* [, *scale*])]

Adaptive Server 将每一种精度和标度的组合都作为一种不同的数据类型来对待。例如，numeric (10,0) 和 numeric (5,0) 是两种不同的数据类型。精度和标度确定了可以存储在 decimal 或 numeric 列中值的范围：

- 精度指定了能够在该列中存储的最大小数位数。它包括小数点左右两侧的所有位数。可以将精度范围指定为 1 至 38 位，或者使用缺省的 18 位精度。
- 标度指定了能够存储到小数点右侧的最大位数。标度必须小于或等于精度。可在 0 至 38 位范围之间指定标度，或者使用缺省的 0 位标度。

如果值超过了列的精度或标度，Adaptive Server 将该条目标记为一个错误。以下是一些有效 dec 和 numeric 数据的示例：

表 8-2：数值数据的有效精度和标度

输入数据	数据类型	精度	标度	值
12.345	numeric(5,3)	5	3	12.345
-1234.567	dec(8,4)	8	4	-1234.567

以下输入会导致错误，因为它们超过了列的精度和标度：

表 8-3：数值数据的无效精度和标度

输入数据	数据类型	精度	标度
1234.567	numeric(3,3)	3	3
1234.567	decimal(6)	6	1

int、*smallint* 和 *tinyint*

可将带有 E 符号的数字值插入 int、smallint 和 tinyint 列，如以前章节所述。

timestamp

不能将数据插入 timestamp 列。必须通过键入“NULL”在列中插入显示空值，或通过提供跳过 timestamp 列的列列表使用隐式空值。每次插入或更新后，Adaptive Server 更新 timestamp 值。有关详细信息，参见第 289 页上的“插入数据到指定列”。

添加新数据

可以使用 insert 命令以两种方式将行添加到数据库；用 values 关键字或 select 语句：

- values 关键字指定了新行中部分或全部列的值。使用 values 关键字的 insert 命令的简化语法为：

  insert *table_name*
  　values (*constant1*, *constant2*, ...)

- 可以在insert语句中使用select语句，从一个或多个表（上限为16个表，包括要插入的表）中取值。使用 select 语句的 insert 命令的简化语法为：

  insert *table_name*
  select *column_list*
  from *table_list*
  where *search_conditions*

insert 语法

以下是 insert 命令的完整语法：

insert [into] [*database*.[*owner*.]]{*table_name* |
view_name} [(*column_list*)]
{values (*constant_expression*
[, *constant_expression*]...) | *select_statement*}

注意：当用 insert 添加 text 或 image 值时，所有数据都写入事务日志。可以使用 writetext 命令来添加这些值，而不记录可能包含 text 或 image 值的长数据块。参见第 289 页上的“插入数据到指定列”和第 304 页上的“改变 text 和 image 数据”。

用 *values* 添加新行

此 insert 语句给 publishers 表添加新行，在该行中为每列提供一个值：

```
insert into publishers
values ("1622", "Jardin, Inc.", "Camden", "NJ")
```

请注意，键入数据值的顺序应与初始 create table 语句中列名称的顺序相同，也即，首先是 ID 号，然后是名称，接着是城市，最后是州。values 数据用括号括起来，并且所有的字符数据用单引号或双引号引起。

对添加的每一行，使用单独的 insert 语句。

插入数据到指定列

可通过仅指定某些列及其数据，将数据添加到某行的这些列。不包含在列列表中的所有其它列，必须定义为允许空值。被跳过的列可接受缺省值。如果被跳过的列有绑定缺省值，则使用缺省值。

您可能特别想使用这种形式的 insert 命令，插入一行中除 text 或 image 外的所有值，然后使用 writetext 插入长数据值，以便这些值不存储在事务日志中。也可使用这种形式的命令跳过 timestamp 数据。

例如，只将数据添加入两列，pub_id 和 pub_name，需要如下命令：

```
insert into publishers (pub_id, pub_name)
values ("1756", "The Health Center")
```

列出列名的顺序必须与列出值的顺序一致。下列示例产生与前一个示例相同的结果：

```
insert publishers (pub_name, pub_id)
values("The Health Center", "1756")
```

两个 insert 语句都将“1756”放到标识号列，并将“The Health Center”放到“publisher name”列。由于在 publishers 中的 pub_id 列有唯一索引，因此两条 insert 语句都不能执行；第二次将“1756”值插入 pub_id 的尝试产生错误消息。

下列 select 语句显示添加到 publishers 的行：

```
select *
from publishers
where pub_name = "The Health Center"
pub_id  pub_name             city    state
------- -----------------    ------  -------
1756    The Health Center    NULL    NULL
```

Adaptive Server 将空值输入 city 和 state 列中，因为在 insert 语句中没有给这些列赋值，并且 publisher 表在这些列上允许空值。

限制列数据：规则

可创建一个规则并将其绑定到列或用户定义的数据类型。规则约束能或不能添加的数据类型。

publishers 表的 pub_id 列是一个示例。一个名为 pub_idrule 的规则绑定到该列，它指定可接受的“publisher”标识号。可接受的 ID 为“1389”、“0736”、“0877”、“1622”以及“1756”，或任何以“99”开头的四位数字。如果输入任何其它数字，都会产生错误消息。

得到此类错误消息时，可能需要使用 sp_helptext 查看规则的定义：

```
sp_helptext pub_idrule

---------
        1

(1 row affected)

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

(1 row affected)
```

有关特定规则的更一般信息，请使用 sp_help。或用表名称作为参数，使用 sp_help，查看是否有任何列拥有规则。参见第 12 章“为数据定义缺省值和规则”。

使用 NULL 字符串

只有在 create table 语句中指定了 NULL，且已显式输入 NULL（无引号），或未输入任何数据的情况下，列才包含空值。应避免将字符串“NULL”（带引号）当作字符列的数据输入。这只会导致混乱。而应使用“N/A”或“none”或类似值。如果要显式输入值 NULL，请*不要*使用单引号或双引号。

要将 NULL 显式插入列中：

values({*expression* | null}
  [, {*expression* | null}]...)

以下示例显示了两个等同的 insert 语句。在第一个语句中，用户将 NULL 显式插入 t1 列中。在第二个语句中，Adaptive Server 为 t1 提供了 NULL 值，因为用户没有指定显式列值：

```
create table test
(t1 char(10) null, t2 char(10) not null)
insert test
values (null, "stuff")
insert test (t2)
values ("stuff")
```

NULL 不是空字符串

空字符串（" " 或 ''）总是作为单个空格存储在变量和列数据中。以下并置语句相当于“abc def”，而不是“abcdef”：

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

空字符串从不会求值为 NULL。

将 NULL 插入不允许 NULL 的列中

要使用 select 将数据从某些字段为空值的表中，插入不允许空值的表中，必须为初始表的所有 NULL 条目提供替代值。例如，为了将数据插入不允许空值的 advances 表中，以下示例用“0”替代 NULL 字段：

```
insert advances
select pub_id, isnull(advance, 0) from titles
```

如果不使用 isnull 函数，此命令就会将带有非空值的所有行插入 advances，并对在 titles 的 advance 列中包含 NULL 的所有行产生错误消息。

如果无法对数据进行这种替代，则不能将包含空值的数据插入已有 NOT NULL 规定的列。

添加所有列都没有值的行

当仅给行中某些列指定值时，没有值的列可出现下列四种情况之一：

- 如果列存在缺省值，或输入了用户定义的列数据类型。有关详细信息，参见*参考手册*中的第 12 章“为数据定义缺省值和规则”或 create default。

- 如果创建表时为列指定了 NULL，并且列或数据类型不存在缺省值，则输入 NULL。另请参见*参考手册*中的 create table。
- 如果列具有 IDENTITY 属性，则输入一个唯一、连续的值。
- 如果创建表时没有为列指定 NULL 并且不存在缺省值，则 Adaptive Server 拒绝该行并显示错误消息。

表 8-4 显示了在这些情况下会看到的情况：

表 *8-4*：没有值的列

缺省值存在于列或数据类型	列已定义为 NOT NULL	列已定义为允许 NULL	列是 IDENTITY
是	缺省值	缺省值	下一个顺序值
否	错误消息	NULL	下一个顺序值

可以使用 sp_help 获得关于指定表、缺省值或系统表 sysobjects 中所列其它任何对象的报告。要查看缺省值的定义，请使用 sp_helptext。

将列的值更改为 NULL

要将列值设定为 NULL，请使用 update 语句：

set *column_name* = {*expression* | null}
[, *column_name* = {*expression* | null}]...

例如，要查找所有 title_id 为 TC3218 的行，并用 NULL 替换 advance：

```
update titles
set advance = null
where title_id = "TC3218"
```

Adaptive Server 生成的 IDENTITY 列的值

当将行插入有 IDENTITY 列的表中时，Adaptive Server 自动生成列值。不要在列列表中包括 IDENTITY 列的名称，也不要在值列表中包括 IDENTITY 列的值。

此 insert 语句将新行添加到 sales_daily 表。请注意，列列表不包含 IDENTITY 列，row_id：

```
insert sales_daily (stor_id)
values ("7896")
```

下列语句显示了添加到 sales_daily 中的列。Adaptive Server 自动为 row_id 生成下一个顺序值，2：

```
select * from sales_daily
where stor_id = "7896"
sale_id      stor_id
-------      -------
      1      7896

(1 row affected)
```

将数据显式插入 IDENTITY 列

有时，可能想要将特定值插入 IDENTITY 列，而不接受服务器生成的值。例如，可能想要插入表的第一行的 IDENTITY 值为 101，而不是 1，或者可能需要重新插入误删除的行。

表的所有者可将值显式插入 IDENTITY 列中。如果数据库所有者和系统管理员获得了表所有者授予的明确权限，或通过 setuser 命令作为表所有者操作，他们能将值显式插入 IDENTITY 列。

插入数据前，设定表的 identity_insert 选项为“on”。在一次会话中，一次仅能对数据库中的一个表将 set identity_insert 设置为“on”。

以下示例指定 IDENTITY 列的源值为 101：

```
set identity_insert sales_daily on
insert sales_daily (syb_identity, stor_id)
values (101, "1349")
```

insert 语句列出了每列，其中包含指定了值的 IDENTITY 列。当 identity_insert 选项设置为“on”时，表的每条 insert 语句都必须指定显式列列表。值列表必须指定 IDENTITY 列值，因为 IDENTITY 列不允许空值。

将 identity_insert 设置为“off”后，能自动插入 IDENTITY 列值，不必与以前一样指定 IDENTITY 列。后续插入根据在将 identity_insert 设置为“on”后指定的显式值，使用 IDENTITY 值。例如，如果为 IDENTITY 列指定了 101，则后续的插入应为 102、103，等等。

注意：Adaptive Server 不强制插入值的唯一性。可在列的声明精度允许的范围内指定任何正整数。为确保只接受唯一列值，在插入任何行前，在 IDENTITY 列上创建唯一索引。

用 @@*identity* 检索 IDENTITY 列值

使用 @@*identity* 全局变量检索插入 IDENTITY 列的最后一个值。每次 insert、select into 或 bcp 语句尝试将行插入表中时，@@*identity* 的值发生改变。如果 insert、select into 或 bcp 语句失败，或如果包含它的事务回退，则 @@*identity* 不还原到它的先前值。如果语句影响没有 IDENTITY 列的表，则 @@*identity* 被设置为“0”。

- 如果语句插入多行，则 @@*identity* 显示最后插入 IDENTITY 列中的值。

存储过程或触发器内的 @@*identity* 值，不影响存储过程或触发器外的值。例如：

```
select @@identity
----------------------------------------
                                     101
create procedure reset_id as
    set identity_insert sales_daily on
    insert into sales_daily (syb_identity, stor_id)
         values (102, "1349")
    select @@identity
select @@identity
execute reset_id
----------------------------------------
                                     102
select @@identity
----------------------------------------
                                     101
```

保留 IDENTITY 列值块

identity grab size 配置参数允许每个 Adaptive Server 进程保留一个 IDENTITY 列值块，用于插入具有 IDENTITY 列的表。此配置参数减少了在插入隐式标识值时，Adaptive Server 引擎必须持有内部同步结构的次数。例如，将保留值的数量设置为 20：

```
sp_configure "identity grab size", 20
```

当用户执行插入包含 IDENTITY 列的表的操作时，Adaptive Server 为该用户保留一个 20 个 IDENTITY 列值的块。因此，在当前会话期间，用户插入的下一 20 行将具有顺序的 IDENTITY 列值。如果第一个用户在进行插入时，第二个用户将行插入同一表中，则 Adaptive Server 将为第二个用户保留下一个 20 个 IDENTITY 列值的块。

例如，假定已创建下列包含 IDENTITY 列的表，并且 identity grab size 设定为“10”：

```
create table my_titles
(title_id   numeric(5,0)    identity,
title       varchar(30)     not null)
```

用户 1 将这些行插入 my_titles 表中：

```
insert my_titles (title)
values ("The Trauma of the Inner Child")
insert my_titles (title)
values ("A Farewell to Angst")
insert my_titles (title)
values ("Life Without Anger")
```

Adaptive Server 允许用户 1 有一个含 10 个顺序 IDENTITY 值的块，例如，title_id 编号 1–10。

当用户 1 将行插入 my_titles 时，用户 2 开始将行插入 my_titles。Adaptive Server 授予用户 2 保留 IDENTITY 值的下一可用块，也就是说，值 11–20。

如果用户 1 只输入三个“title”，然后注销 Adaptive Server，那么剩余的七个保留的 IDENTITY 值将丢失。结果是表的 IDENTITY 值出现一个间隔。要避免将 identity grab size 设置太高，因为这能导致在 IDENTITY 列编号中出现间隔。

达到 IDENTITY 列的最大值

能插入 IDENTITY 列的最大值是 10 precision - 1。如果不指定 IDENTITY 列的精度，Adaptive Server 使用 numeric 列的缺省精度（18 位）。

一旦 IDENTITY 列达到最大值，插入语句将返回错误消息并终止当前事务。发生这种情况时，使用下列方法之一来解决问题。

创建具有更大精度的新表

如果表包含用于参照完整性的 IDENTITY 列，则需要保留 IDENTITY 列值的当前编号。

1. 使用 create table 创建新表，该表除了 IDENTITY 列精度值更大外，与旧表相同。
2. 使用 insert into 将数据从旧表复制到新表中。

用 bcp 对表的 IDENTITY 列重新编号

如果表不包含用于参照完整性的 IDENTITY 列，并且如果在编号序列中有间隔，可以对 IDENTITY 列进行重新编号以消除间隔，以允许更多的插入空间。

要按顺序对 IDENTITY 列值重新编号并删除间隔，请使用 bcp 实用程序：

1 在操作系统命令行，使用 bcp 将数据复制出来。例如：

```
bcp pubs2..mytitles out mytitles_file -N -c
```

-N 指示 bcp 不将 IDENTITY 列值从表中复制到主机文件。-c 指示 bcp 使用字符模式。

2 在 Adaptive Server 中，创建一个与旧表相同的新表。

3 从操作系统命令行，使用 bcp 将数据复制进新表：

```
bcp pubs2..mynewtitles in mytitles_file -N -c
```

-N 指示 bcp 当从主机文件装载数据时，让 Adaptive Server 分配 IDENTITY 列值。-c 指示 bcp 使用字符模式。

4 在 Adaptive Server 中，删除旧表，并使用 sp_rename 将新表的名称更改为旧表的名称。

如果 IDENTITY 列是连接的主键，可能需要更新其它表中的外键。

缺省情况下，当将数据批量复制到有 IDENTITY 列的表时，bcp 为每行分配一个临时的 IDENTITY 列值，0。在将每行插入表时，服务器从下一可用的值开始，为其分配一个唯一的、连续的 IDENTITY 列值。要为每行输入显式 IDENTITY 列值，请指定 -E (UNIX) 或 /identity (OpenVMS) 标志。有关影响 IDENTITY 列的 bcp 选项的详细信息，请参阅所用平台的*实用程序指南*。

用 *select* 添加新行

要从一个或多个其它表中将值取进一个表，请使用 insert 语句中的 select 子句。select 子句能将值插入某行的某些或所有列。

当要从已经存在的表中取出一些值时，插入值到某些列就很方便。然后，可以使用 update 将值添加到其它列。

在将值插入表的某些但不是全部列前，要确保存在缺省值或不插入值的列已经指定为 NULL。否则，Adaptive Server 返回错误消息。

当从一个表向另一个表插入行时，两个表必须具有兼容的结构 — 也就是说，匹配列的数据类型必须相同或者是 Adaptive Server 可在两者间自动转换的数据类型。

注意：如果要插入的任何数据是空值，则不能将数据从允许空值的表插入不允许空值的表。

如果列在其 create table 语句中有相同的顺序，则不必指定每个表的列名称。假定有一个名为 newauthors 的表，它含有一些“author”信息行，它们具有与 authors 中信息相同的格式。要将 authors 中所有行添加到 newauthors：

```
insert authors
select *
from newauthors
```

要根据另一表中的数据将行插入某表，在各自的 create table 语句中不必按相同的顺序列出两个表中的列。可以使用 insert 或 select 语句排序列，以使它们相互匹配。

例如，假定 authors 表的 create table 语句包含顺序如 au_id、au_fname、au_lname 和 address 的列，而 newauthors 包含列 au_id、address、au_lname 和 au_fname。必须在 insert 语句中使列顺序匹配。可以采用下列两种方法之一：

```
insert authors (au_id, address, au_lname, au_fname)
select * from newauthors
```

或

```
insert authors
select au_id, au_fname, au_lname, address
    from newauthors
```

如果两个表中列的顺序不能匹配，则 Adaptive Server 不能完成 insert 操作或不能正确完成操作，将数据放入错误的列。例如，可能在 au_lname 列中得到“address”数据。

计算列

可以在 insert 语句内的 select 语句中使用计算列。例如，假定名为 tmp 的表包含一些 titles 表的新行，其中包含一些过期的数据 — price 数字需要加倍。增加“prices”并将 tmp 行插入 titles 的语句如下所示：

```
insert titles
select title_id, title, type, pub_id, price*2,
    advance, total_sales, notes, pubdate, contract
from tmp
```

对列执行计算时，不能使用 select * 语法。每列都必须在选择列表中单独命名。

插入数据到某些列

可以使用 select 语句将数据添加到行的某些列，但不是所有列，如同 values 子句一样。只需在 insert 子句中，指定数据要添加到的列。

例如，authors 表中的某些“author”没有“title”，因此在 titleauthor 表中没有条目。要将它们的 au_id 编号从 authors 表中取出来，并插入到 titleauthor 表中作为占位符，请执行以下语句：

```
insert titleauthor (au_id)
select au_id
    from authors
    where au_id not in
    (select au_id from titleauthor)
```

此语句是不合法的，因为 title_id 列需要一个值。不允许空值，并且没有指定缺省值。可以通过使用一个常量为 titles_id 输入伪值“xx1111”，如下所示：

```
insert titleauthor (au_id, title_id)
select au_id, "xx1111"
    from authors
    where au_id not in
    (select au_id from titleauthor)
```

titleauthor 表现在包含四个新行，这些新行在 au_id 列有条目，在 title_id 列有伪条目，其它两列为空值。

从相同表中插入数据

能基于相同表中的其它数据将数据插入表中。实质上，这意味着复制一行的全部或部分。

例如，能将新行插入 publishers 表中，该表基于同一表中已经存在行的值。确保遵照 pub_id 列上的规则：

```
insert publishers
select "9999", "test", city, state

    from publishers
    where pub_name = "New Age Books"

(1 row affected)
select * from publishers

pub_id  pub_name              city        state
------- --------------------- -------     ------
0736    New Age Books         Boston      MA
0877    Binnet & Hardley      Washington  DC
1389    Algodata Infosystems  Berkeley    CA
9999    test                  Boston      MA

(4 rows affected)
```

该示例插入两个常量（“9999”和“test”）以及满足查询的行的 city 和 state 列的值。

更改现有数据

使用 update 命令更改表中的单行、行组或所有行。如同在所有数据修改语句中，一次只可更改一个表中的数据。

update 指定了要更改的一行或多行以及新数据。新数据可以是指定的常量或表达式，或者是从其它表中拖出的数据。

如果 update 语句违反完整性约束，则不发生更新，并产生错误消息。例如，如果它影响了表的 IDENTITY 列，或某个要添加值的数据类型错误，或违反了为所涉及的列或数据类型之一定义的规则，则取消更新。

Adaptive Server 允许多次执行更新单个行的 update 命令。当然，update 语句的处理方法决定了单个语句进行的更新不累积。也就是说，如果 update 语句两次修改同一行，则第二次更新将不以第一次更新后的新值为基础，而是以初始值为基础。结果是不可预知的，因为它们取决于处理的顺序。

有关对更新视图的限制，参见第 9 章“视图：限制访问数据”。

注意：update 命令将被记录。如果改变大块的 text 或 image 数据，请尝试使用 writetext 命令，它不会被记录下来。同样，每个 update 语句大约限制为 125K。参见第 304 页上的“改变 text 和 image 数据”中对 writetext 的讨论。

---

update 语法

用表达式更新指定行的简化 update 语法为：

update *table_name*
  set *column_name* = *expression*
  where *search_conditions*

例如，如果 Reginald Blotchet-Halls 决定更改姓名，在 authors 表中改变其行的方式如下：

```
update authors
set au_lname = "Health", au_fname = "Goodbody"
where au_lname = "Blotchet-Halls"
```

以下语句根据另一表中的数据更新一个表：

update *table_name*
  set *column_name* = *expression*
    from *table_name*
    where *search_conditions*

可以在 update 语句中设定变量：

update *table_name*
  set *variable_name* = *expression*
    where *search_conditions*

update 的完整语法为：

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 = {*expression1* | null |
(*select_statement*)}
[from [[*database*.]*owner*.]{*table_name* | *view_name*}
[, [[*database*.]*owner*.]{*table_name* |
view_name}]]...
[where *search_conditions*]

将 *set* 子句用于 *update*

set 子句指定了列和改变的值。where 子句确定要更新哪一行或哪些行。如果没有 where 子句，则*所有*行的指定列都将更新为 set 子句中给定的值。

注意：在尝试本节中的示例前，应确保知道如何重新安装 pubs2 数据库。有关安装 pubs2 数据库的指导，请参见所用平台的安装和配置指南。

例如，如果 publishers 表中所有的出版社都将其总部移到 Georgia 州 Atlanta 市，以下说明如何更新该表：

```
update publishers
set city = "Atlanta", state = "GA"
```

同样，可以将所有“publisher”的名称改为 NULL：

```
update publishers
set pub_name = null
```

也可以在更新中使用计算列值。要将 titles 表中所有“price”加倍，使用：

```
update titles
set price = price * 2
```

由于没有 where 子句，对“price”的改变适用于表中所有行。

在 *set* 子句中对变量赋值

可在 update 语句的 set 子句中对变量赋值，也可以同样方式，在 select 语句中对它们赋值。将变量用于 update 减少了锁争用，并减少了额外的 select 语句与 update 一起使用时可出现的 CPU 消耗。

以下示例使用声明变量更新 titles 表：

```
declare @price money
select @price = 0
update titles
    set total_sales = total_sales + 1,
    @price = price
    where title_id = "BU1032"
select @price, total_sales
    from titles
    where title_id = "BU1032"
                          total_sales
 ------------------------ -----------
                   19.99        4096

(1 row affected)
```

有关在 update 语句中给变量赋值的详细信息，参见*参考手册*。有关声明变量的详细信息，参见第 429 页上的“局部变量”。

将 *where* 子句用于 *update*

where 子句指定哪些行要更新。例如（这件事不太可能），“Northern California”改名为“Pacifica”（简称 PC），并且“Oakland”人投票将他们的城市改名为“Big Bad Bay City”，下面说明如何更新 authors 表中所有地址过时的前“Oakland”居民。

```
update authors
set state = "PC", city = "Big Bad Bay City"
where state = "CA" and city = "Oakland"
```

需要编写另外语句来改变“Northern California”州其它城市居民的州名。

将 *from* 子句用于 *update*

使用 from 子句将数据从一个或多个表取进要更新的表。

例如，在本章的前面，给出了一个示例，其中为没有“title”的“author”将一些新行插入 titleauthor 表，填充 au_id 列，并对其它列使用伪值和空值。这些“author”之一的“Dirk Stringer”写了一本书，*The Psychology of Computer Cooking*，在 titles 表中给这本书分配一个标识号。在 titleauthor 表中，可以通过为他添加一个“title”标识号来修改他的行：

```
update titleauthor
set title_id = titles.title_id
from titleauthor, titles, authors
    where titles.title =
    "The Psychology of Computer Cooking"
    and authors.au_id = titleauthor.au_id
    and au_lname = "Stringer"
```

注意，没有使用 au_id 的 update 将改变 titleauthor 表中的所有 title_ids，因此它们与 *The Psychology of Computer Cooking* 的标识号相同。如果两个表在结构上相同，只是一个表有 NULL 字段和一些空值，而另一个表有 NOT NULL 字段，则不可能用 select 将数据从 NULL 表插入到 NOT NULL 表。也就是说，不允许 NULL 的字段不能通过从允许 NULL 的字段中进行选择而更新（如果有任何数据为 NULL）。

作为 update 语句中的 from 子句的替代，还可以使用符合 ANSI 标准的子查询。

用连接更新

以下示例连接 *titles* 和 *publishers* 表的列，并将所有在“California”出版的书的“price”加倍：

```
update titles
  set price = price * 2
  from titles, publishers
  where titles.pub_id = publishers.pub_id
  and publishers.state = "CA"
```

尽管连接通常能处理多达 16 个表，但是如果连接是更新或删除语句的一部分，则查询只能引用 15 个表。

更新 IDENTITY 列

必要时，可以使用用表名限定的 syb_identity 关键字更新 IDENTITY 列。例如，此 update 语句查找 IDENTITY 列等于 1 的所有行，并将存储的名称改为“Barney 担”：

```
update stores_cal
set stor_name = "Barney's"
where syb_identity = 1
```

改变 *text* 和 *image* 数据

当不想在数据库事务日志中保存长文本值时，使用 writetext 改变 text 或 image 值。不要使用 update 命令，它也可用于 text 或 image 列，因为 update 命令始终都会被记录。缺省模式下，writetext 命令不被记录。

注意：要在缺省状态（不记日志状态）下使用 writetext，系统管理员必须使用 sp_dboption 将 select into/bulkcopy/pllsort 设置为 on。这将允许插入不记入日志的数据。使用 writetext 后，有必要转储数据库。对数据库进行不记入日志的更改后，不能使用 dump transaction。

---

writetext 命令完全覆盖它所影响列中的所有数据。列必须已经包含有效的文本指针。

可以使用 textvalid() 函数查找有效指针：

```
select textvalid("blurbs.copy", textptr(copy))
from blurbs
```

有两种创建文本指针的方法：

- 将实际数据 insert 到 text 或 image 列
- 用数据或 NULLupdate 列

“已初始化”的 text 列使用 2K 存储空间，即使是存储两个字。当用 insert 将显式或隐式空值放入 text 列中时，Adaptive Server 通过不初始化文本列来节省空间。下列代码片段插入一个有空文本指针的值，检查是否存在一个文本指针，然后更新 blurbs 表。在文本中嵌入说明性注释：

```
/* Insert a value with a text pointer. This could
** be done in a separate batch session. */
insert blurbs (au_id) values ("267-41-2394")
/* Check for a valid pointer in an existing row.
** Use textvalid in a conditional clause; if no
** valid text pointer exists, update ëopyíto nul
** to initialize the pointer. */
if (select textvalid("blurbs.copy", textptr(copy))
    from blurbs
    where au_id = "267-41-2394") = 0
begin
    update blurbs
       set copy = NULL
       where au_id = "267-41-2394"
end
/*
** use writetext to insert the text into the
** column. The next statements put the text
** into the local variable @val, then writetext
** places the new text string into the row
** pointed to by @val. */
declare @val varbinary(16)
select @val = textptr(copy)
    from blurbs
    where au_id = "267-41-2394"
writetext blurbs.copy @val
    "This book is a must for true data junkies."
```

有关在本示例中使用的批处理文件和控制流语言的详细信息，参见第 13 章“使用批处理和控制流语言”。

删除数据

delete 适用于单行或多行操作。

delete 的简化语法为：

delete *table_name*
　where *column_name* = *expression*

以下是一个完整的语法语句，它说明能基于指定的表达式也能基于其它表的数据删除行：

delete [from]
　[[*database*.]*owner*.]{*view_name*|*table_name*}

```
    [where search_conditions]

delete [[database.]owner.]{table_name | view_name}
    [from [[database.]owner.]{view_name|table_name
        [(index {index_name | table_name }
            [ prefetch size ][lru|mru])]}
        [, [[database.]owner.]{view_name|table_name
        (index {index_name | table_name }
            [ prefetch size ][lru|mru])]} ]...]
    [where search_conditions]

delete [from]
    [[database.]owner.]{table_name|view_name}
    where current of cursor_name
```

where 子句指定哪些行要删除。delete 语句中没有给定 where 子句时，删除表中*所有*行。

将 *from* 子句用于 *delete*

紧跟在 delete 关键字后可选的 from 的目的是与其它 SQL 版本兼容。delete 语句中第二位的 from 子句是一项特殊的 Transact-SQL 功能，它允许从一个或多个表中选择数据，并从第一个命名的表中删除相应的数据。在 from 子句中选定的行指定 delete 的条件。

假设一个复杂公司业务的结果是另一出版者获得“Big Bad Bay”市（前“Oakland”）的所有作者及其书籍。需要将所有这些书从 titles 表中立即删除，但是不知道它们的“title”或标识号。仅有的信息是“author”的姓名和地址。

可以通过查找 authors 表中以“Big Bad Bay City”作为“town”的行的“author”标识号，并使用这些标识号查找这些书在 titleauthor 表中的“title”标识号来删除 titles 中的行。也即，查找 titles 表中要删除的行需要一个三向连接。

这三个表都包含在 delete 语句的 from 子句中。然而，只有 titles 表中满足 where 子句的行才被删除。必须使用单独的删除语句来删除非 titles 表中的相应行。

以下是所需的语句：

```
delete titles
from authors, titles, titleauthor
where titles.title_id = titleauthor.title_id
and authors.au_id = titleauthor.au_id
and city = "Big Bad Bay City"
```

pubs2 数据库中的 deltitle 触发器禁止实际执行此删除，因为它不允许删除任何在 sales 表中有销售记录的“title”。

从 IDENTITY 列删除

对于包含 IDENTITY 列的表，可以在 delete 语句中使用 syb_identity 关键字。例如，以下语句删除 row_id 等于 1 的行：

```
delete sales_monthly
where syb_identity = 1
```

根据 IDENTITY 列删除相应行后，可能需要消除表的 IDENTITY 列的编号顺序间隔。参见第 296 页上的“用 bcp 对表的 IDENTITY 列重新编号”。

删除表中的所有行

使用 truncate table 删除表中所有行。它几乎总是比不带条件的 delete 语句快，因为 delete 记录每次更改，而 truncate table 只记录整个数据页的重新分配。truncate table 立即释放表数据和索引占用的所有空间。然后自由空间能被任何对象使用。所有索引的分布页也重新分配。给表添加新行后，运行 update statistics。

与 delete 一样，用 truncate table 腾空的表，连同其索引和其它相关联对象，都保留在数据库中，除非输入 drop table 命令。

如果另一表有通过参照完整性约束引用它的行，则不能使用 truncate table。先删除外表的行，或截断外表，然后再截断主表。参见第 238 页上的“创建参照完整性约束的一般规则”。

truncate table 语法

truncate table 的语法是：

truncate table [[*database*.]*owner*.]*table_name*

例如，要删除 sales 中所有数据，键入：

```
truncate table sales
```

与 drop table 相似，缺省情况下，允许表的所有者使用 truncate table，并且不能移交其他用户。

truncate table 命令不会被 delete 触发器捕获。参见第 16 章“触发器：强制实施参照完整性”。

第 9 章

视图：限制访问数据

视图是一个名为 select 的语句，该语句作为一个对象存储在数据库中。 通过它可以在一个或多个表中查看行或列的子集。 可以通过调用它在其它 Transact-SQL 语句中的名称来使用视图。 在特定的数据库中，可以使用视图来关注、简化和自定义每个用户查看表的效果。 视图只允许用户访问他们所需要的数据，这也提供了一种安全性机制。

本章讨论如下内容：

主题	页
视图如何工作	309
创建视图	313
通过视图检索数据	321
通过视图修改数据	324
删除视图	329
使用视图作为安全性机制	329
获取有关视图的信息	330

视图如何工作

视图是在查看表中数据时可采用的一种方法。

例如，假定您正在进行一项关于 Utah 州的项目。 可以创建一个视图，它仅列出居住在 Utah 州的作者：

```
create view authors_ut
as select * from authors
where state = "UT"
```

要显示 authors_ut 视图，应输入：

```
select * from authors_ut
```

当向 authors 表中添加或从中删除居住在 Utah 州的作者时，视图 authors_ut 就会反映出更新后的 authors 表。

视图来自一张或多张真实的表，表的数据物理存储在数据库中。视图所基于的表叫做基表或基础表。视图也可以从另一视图派生得到。

视图的定义以其所基于的基表的方式存储在数据库中。没有单独的数据与这种存储的定义相关联。查看的数据被存储在基础表中。

视图看起来与数据库中其它表很类似。可以用显示和操作其它表的方法来显示并操作视图。 在通过视图进行查询方面没有任何限制，在修改方面的限制也比较少。 在本章稍后部分对例外情况进行了说明。

当通过视图修改数据时，实际上是在更改底层基表上的数据。反之，底层基表数据的改变也会自动地在其派生的视图中反映出来。

视图的优点

可以使用视图来关注、简化和自定义每个用户查看数据库的效果；视图还提供了一种便于使用的安全措施。 更改数据库的结构后视图也非常有用，但用户都愿意使用他们已习惯的数据库结构。

可以使用视图来：

- 关注他们感兴趣的数据和所承担的任务。 用户不感兴趣的数据可以不显示在视图中。
- 将常用的连接、投影和选择定义为视图，这样在每次操作该数据时，用户就不必指定所有的条件和限制。
- 为不同的用户显示不同的数据，即使他们同时使用同一数据。这一优点对不同兴趣、不同技术水平不同的用户共享数据库尤为重要。

安全性

通过视图，用户只能查询和修改所看到的数据。数据库的其余部分既不可见也不可访问。

使用 grant 和 revoke 命令，可将每个用户对数据库的访问限制在指定的数据库对象（包括视图）上。 如果视图、所有表以及派生此视图的其它视图属于同一用户，此用户可以赋予其它用户使用此视图的权限，而禁止他们使用该视图的基础表和基础视图。这是一个简单而有效的安全机制。 有关 grant 和 revoke 命令的详细信息，参见*系统管理指南*。

通过定义不同的视图并有选择性地授予用户权限，可以把他们限制到不同的数据子集上。例如：

- 可以把访问限制到基表中行的子集，即与值相关的子集。例如，可以定义只包含商业与心理学图书的视图，以便使其它类型的图书信息对某些用户来说是隐藏的。
- 可以把访问限制到基表中列的子集，即与值相关的子集。 例如，可以定义包含 titles 表中所有行但不包含 royalty 和 advance 列的视图。
- 可以把访问限制到由基表中部分行与列组成的子集。
- 可以把访问限制到能连接多个基表的行。 例如，可以定义一个连接 titles、authors 和 titleauthor 的视图以显示作者名和他们的书名。此视图将隐藏作者的个人信息以及图书的价格信息。
- 可以把访问限制到基表的统计信息。 例如，通过 category_price 视图，用户只能访问各类图书的平均价格。
- 可以把访问限制到另一个视图的子集或视图与基表的组合上。例如，通过 hiprice_computer 视图，用户可以访问满足 hiprice 视图定义的计算机图书的书名与价格。

要创建视图，用户必须从数据库所有者获得 create view 权限，并且必须对视图定义中引用的表或视图拥有适当的权限。

如果视图引用不同数据库中的对象，视图的用户必须是各个数据库的有效用户或访客。

如果其他用户在您拥有的某个对象上创建了视图，则必须要了解谁能通过什么视图看到什么数据。例如： 数据库所有者已经授予“harold”create view 权限，名叫“maude”的用户授予“harold”在她拥有的表中进行 select 的权限。有了这些权限，“harold”能够从“maude”拥有的表中选择所有的行和列来创建视图。 如果“maude”撤消了“harold”从她的表中进行 select 的权限，“harold”仍能通过他已建的视图查看她的数据。

逻辑数据独立性

如果需要对真实表的结构进行更改，视图可以使用户不受这种更改的影响。

例如，假定通过使用 select into 将 titles 表拆分为如下两个新基表、然后删除 titles 表的方法来调整数据库的结构：

```
titletext (title_id, title, type, notes)
titlenumbers (title_id, pub_id, price, advance,
royalty, total_sales, pub_date)
```

通过连接两个新表的 title_id 列，可以“再生”旧的 titles 表。 可以创建一个视图来连接这两个新表。 甚至可以将其命名为 titles。

对于以前引用基表 titles 的任何查询或存储过程，现在则引用视图 titles。 对于用户来说，select 操作与以前完全相同。 只在新视图中进行检索的用户甚至不需要知道数据库的结构已经改变。

然而，视图只提供一部分逻辑独立。 由于某些限制，在新的 titles 上不能执行某些数据修改语句。

视图示例

第一个示例是从 titles 表中派生的视图。 假定您只对价格高于 $15 并且预付款超过 $5000 的图书感兴趣。 这个简单的 select 语句将查找符合这些条件的行：

```
select *
from titles
where price > $15
  and advance > $5000
```

现在假定要对这组数据进行大量的检索和更新操作。 当然可以将在先前查询中所显示的条件与要发出的命令进行组合。 然而，为方便起见，可以创建一个只显示感兴趣的记录的视图：

```
create view hiprice
as select *
from titles
where price > $15
  and advance > $5000
```

Adaptive Server 接收到此命令后，实际并不执行关键字 as 后面的 select 语句。 而是将 select 语句（视图 hiprice 的定义）存储在系统表 syscomments 中。 还在 sysobjects 和 syscolumns 中为视图里的每列建立条目。

现在，当显示 hiprice 或对其进行操作时，Adaptive Server 将语句与存储的 hiprice 的定义相组合。 例如，可象更改其它表一样更改 hiprice 中的所有价格：

```
update hiprice
    set price = price * 2
```

Adaptive Server 将在系统表中查找视图定义，并将 update 命令转换为语句：

```
update titles
set price = price * 2
where price > $15
  and advance > $5000
```

也就是说，Adaptive Server 从视图定义中得知要更新的数据在 titles 中。 它还知道只应在符合条件（指在视图定义中给出的 price 和 advance 列上的条件和更新语句中的条件）的行中增加价格。

发出更新 hiprice 的命令后，可以在视图或 titles 表中查看结果。 反之，如果已创建了视图并发出了第二条更新语句（此语句直接对基表进行操作），则通过此视图也能看到更改后的价格。

如果用不同的行都符合视图条件的这种方式来更新视图的基础表，就会影响此视图。 例如，假定要想将 *You Can Combat Computer Stress* 一书的价格增加到 $25.95。 由于此书符合视图定义语句中的限定条件，因此会将其视为视图的一部分。

如果通过添加列来更改视图基础表的结构，新列将*不会*出现在用 select * 子句定义的视图中，除非删除并重新定义该视图。 这是因为原视图定义中的星号只考虑原有的列。

创建视图

对于每个用户来说，视图名称在已经存在的表和视图中必须唯一。 如果设置了 set quoted_identifier on，就可以在视图名称上使用分隔标识符。 否则，视图名称必须符合第 8 页上的“标识符”中的标识符规则。

可以在其它视图和引用视图的过程上创建视图。 可以在视图上定义主键、外键和公用键。 但不能使规则、缺省值或触发器与视图相关联，也不能在其上面建立索引。 不能在临时表上创建临时视图或视图。

create view 语法

create view 的完整语法为：

```
create view [owner .]view_name
    [(column_name  [, column_name ]...)]
    as select [distinct] select_statement
    [with check option]
```

如第 312 页上的“视图示例”中的 create view 示例所述，不需要在视图定义语句的 create 子句中指定任何列名称。 Adaptive Server 为视图的列提供了与 select 语句选择列表中所引用的列相同的名称和数据类型。 可以用星号 (*) 指定选择列表，如示例所示，或者它可以是基表中全部或部分列名的列表。

要建立不含重复行的视图，可使用 select 语句的 distinct 关键字来确保视图中的每一行都唯一。 但是，不能更新 distinct 视图。

随时都可以指定列名称。 但在下述任何一种情况下，*必须*在 create 子句中为视图的*每一*列指定列名称：

- 视图中的列是从算术表达式、集合、内部函数或常量派生的。
- 两个或多个视图列拥有相同的名称。 此现象经常出现，因为视图的定义包含连接，而被连接的列具有相同的名称。
- 要为视图中的列指定一个与派生该列的列不同的名称。

  也可以在 select 语句中对列重命名。 无论是否对视图列进行重命名，它都继承它所派生自的列的数据类型。

下面是一个视图定义语句，它能够使视图中的列名称不同于它在基础表中的名称。

```
create view pub_view1 (Publisher, City, State)
as select pub_name, city, state
from publishers
```

下面是用另一种方法创建相同的视图，但是在 select 语句中对列进行重命名：

```
create view pub_view2
as select Publisher = pub_name,
City = city, State = state
from publishers
```

下一节中的视图定义语句示例将说明将列名称包含在 create 子句中的其余规则。

注意：在视图定义不能使用局部变量。

使用 *select* 语句和 *create view*

create view 语句中的 select 语句用于定义视图。 在所创建的视图的 select 语句中，必须拥有对任何引用的对象进行 select 操作的权限。

可以通过任何复杂的 select 语句来用多个表和其它视图创建视图。

视图定义中有一些对 select 语句的限制：

- 不能包含 order by 或 compute 子句。
- 不能包含 into 关键字。
- 不能引用临时表。

带有投影的视图定义

要创建一个包含 titles 表中所有行、但只含列的一个子集的视图，可输入：

```
create view titles_view
as select title, type, price, pubdate
from titles
```

请注意，create view 子句中不包含任何列名称。 视图 titles_view 继承了在选择列表中给定的列名称。

带有计算列的视图定义

下面是一个视图定义语句，用由 price、royalty 和 total_sales 列生成的计算列创建视图：

```
create view accounts (title, advance, amt_due)
as select titles.title_id, advance,
(price * royalty /100) * total_sales
from titles, roysched
where price > $15
and advance > $5000
and titles.title_id = roysched.title_id
```

```
and total_sales between lorange and hirange
```

因为通过将 price、royalty 和 total_sales 相乘得到的列没有可继承的名称，所以必须将列的列表放在 create 子句中。 计算列命名为 amt_due。 计算列在 create 子句中的位置必须与计算时所用的表达式在 select 子句中的位置相同。

带有集合或内部函数的视图定义

包含集合或内部函数的视图定义必须将列名称放在 create 子句中。例如：

```
create view categories1 (category, average_price)
as select type, avg(price)
from titles
group by type
```

当出于安全性考虑而创建视图时，应谨慎使用集合函数和 group by 子句。 对于可以包含在带有 group by 的 select 中的列，由于 Transact-SQL 扩展不对其进行限制，因此可能会导致视图所返回的信息多于所要求的信息。例如：

```
create view categories2 (category, average_price)
as select type, avg(price)
from titles
where type = "business"
```

上述示例中，您可能想用视图将结果限定为“business”类别，但视图也显示了其它类别的信息。 有关 group by 和此 group by Transact-SQL 扩展的详细信息，参见第 78 页上的“将查询结果分组：group by 子句”。

带有连接的视图定义

可以创建由多个基表派生的视图。 下面是由 authors 和 publishers 表派生视图的示例。 此视图包含与出版者居住在同一城市的作者的名称和所在的城市，以及每个出版者的姓名和所在城市。

```
create view cities (authorname, acity,
publishername, pcity)
as select au_lname, authors.city, pub_name,
publishers.city
from authors, publishers
where authors.city = publishers.city
```

用于外部连接的视图

如果定义带有外部连接的视图，然后使用外部连接的内部表中列的限定条件查询该视图，则可能不会得到预期结果。 查询返回内部表的所有行。 不满足限定条件的行在这些行的相应列中显示为 NULL 值。

下列规则确定了可通过连接视图对列进行更新的类型：

- 连接视图中不允许使用 delete 语句。
- 在用 with check option 创建的连接视图中不允许使用 insert 语句。
- 在用 with check option 创建的连接视图中，允许使用 update 语句。如果任何受影响的列出现在 where 子句中，或出现在包含来自多个表的列的表达式中，更新将会失败。
- 如果通过连接视图来插入或更新行，所有受影响的列都必须属于同一基表。

从其它视图派生的视图

可以根据其它视图来定义视图，如下例所示：

```
create view hiprice_computer
as select title, price
from hiprice
where type = "popular_comp"
```

distinct 视图

可以确保视图中的行的唯一性，如下例所示：

```
create view author_codes
as select distinct au_id
from titleauthor
```

如果某行的所有列值都与另一行的相同列值相匹配，则该行就是另一行的重复行。 两个空值被认为是相同的。

Adaptive Server 第一次访问视图时，在进行投影或选择之前，会将 distinct 要求应用到视图定义中。 视图的外观和操作与数据库表类似。 如果选择 distinct 视图的投影（即只选择视图的某些列但选择其所有行），则得到的结果似乎重复。 但视图自身内的每行仍然唯一。 例如，假定创建了一个 distinct 视图 myview，它有三个列，即 a、b 和 c，其中包含下列值：

a	b	c
1	1	2
1	2	3
1	1	0

输入此查询后：

```
select a, b from myview
```

结果如下：

```
a    b
---  ---
1    1
1    2
1    1

 (3 rows affected)
```

第一行与第三行似乎重复。 但基础视图行仍然唯一。

包含 IDENTITY 列的视图

可以通过列出列名或视图的 select 语句中的 syb_identity 关键字来定义包含 IDENTITY 列的视图。例如：

```
create view sales_view
as select syb_identity, stor_id
from sales_daily
```

但是，不能使用 *identity_column_name* = identity(*precision*) 语法向视图添加新的 IDENTITY 列。

可以使用 syb_identity 关键字选择视图中的 IDENTITY 列，除非视图：

- 多次选择 IDENTITY 列
- 由 IDENTITY 列计算新列
- 包含一个集合函数

- 连接多个表的列
- 将 IDENTITY 列作为表达式的一部分

在下述情况下，Adaptive Server 不将该列视为视图的 IDENTITY 列。 在视图上执行 sp_help 时，此列显示的“Identity”值为 0。

在下面示例中，row_id 列不被认为是 store_discounts 视图的 IDENTITY 列，因为 store_discounts 从两个表连接列：

```
create view store_discounts
as
select stor_name, discount
from stores, new_discounts
where stores.stor_id = new_discounts.stor_id
```

定义视图时，基础列保留 IDENTITY 属性。 通过视图更新行时，不能为 IDENTITY 列指定新值。 通过视图插入行时，Adaptive Server 为 IDENTITY 列生成新的顺序值。 为列的基表设置 identity_insert on 后，只有表所有者、数据库所有者或系统管理员才能显式地在 IDENTITY 列中插入值。

创建视图后

创建视图后，描述视图的**源文本**存储在 syscomments 系统表的 text 列中。 *切勿从 syscomments* 中删除此信息。 而应使用在*参考手册*中说明的 sp_hidetext 对 syscomments 中的文本加密。有关详细信息，参见第 4 页上的“编译对象”。

使用 *with check option* 验证视图的选择标准

通常，Adaptive Server 不对视图的 insert 和 update 语句进行检查来确定被影响的行是否在此视图的范围内。 语句可以将行插入到底层基表中，但不能插入到视图中，语句还可以更改现存的行使其不再符合视图的选择标准。

使用 with check option 子句创建视图时，通过该视图使用的每个 insert 和 update 都符合视图的选择标准而有效。 所有通过视图插入或更新的行都必须可以通过该视图来查看，否则语句失败。

下面是用 with check option 创建的视图 stores_ca 的示例。 此视图包含位于 California 的商店的信息，但不包含其它州的商店的信息。 此视图是通过选择 stores 表中 state 值为“CA”的所有行来创建的：

```
create view stores_ca
as select * from stores
where state = "CA"
with check option
```

通过 stores_ca 插入行时，Adaptive Server 校验新行是否在此视图的范围内。 下面的 insert 语句失败，因为新行的 state 值为“NY”而不是“CA”：

```
insert stores_ca
values ("7100", "Castle Books", "351 West 24 St.",
"New York", "NY", "USA", "10011", "Net 30")
```

通过 stores_cal 更新行时，Adaptive Server 将校验此更新操作是否会导致此行从视图中消失。 下面的 update 语句失败，因为它要将 state 值从“CA”更改为“MA”。 更新后，通过视图就不能再看到此行。

```
update stores_ca
set state = "MA"
where stor_id = "7066"
```

从其它视图派生的视图

如果视图是用 with check option 创建的，则所有从“基”视图派生的视图都必须满足其检查选项要求。 通过派生视图插入的每一行都必须能通过基视图查看。 通过派生视图更新的每一行都必须能通过基视图查看。

以从 stores_cal 派生的视图 stores_cal30 为例。此新视图包含位于 California 且付款方式为“Net 30”的商店的有关信息。

```
create view stores_cal30
as select * from stores_ca
where payterms = "Net 30"
```

由于 stores_cal 是用 with check option 创建的，所以通过 stores_cal30 插入或更新的所有行都必须能够通过 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"
```

通过视图检索数据

通过视图检索数据时，Adaptive Server 会进行检查，以确保在语句中任何位置所引用的所有数据库对象都存在，并且它们在语句的环境中均有效。 如果检查通过，Adaptive Server 就会将此语句与存储的视图定义相组合，然后转换为视图基础表上的一个查询。 这一过程被称为**视图解析**。

以下面的视图定义语句及其查询为例：

```
create view hiprice
as select *
from titles
where price > $15
and advance > $5000
select title, type
from hiprice
where type = "popular_comp"
```

在内部，Adaptive Server 将 hiprice 查询与其定义组合，并将查询转换为：

```
select title, type
from titles
where price > $15
and advance > $5000
and type = "popular_comp"
```

通常，就象真实的表一样可以用任何方法查询视图。 对于视图，可以按任何组合来使用连接、group by 子句、子查询和其它查询方法。 但应注意，如果视图是用外部连接或集合函数定义的，则在查询此视图时可能会得到意想不到的结果。参见第 317 页上的“从其它视图派生的视图”。

注意：可以对视图中的 text 和 image 列使用 select。 但在视图中不能使用 readtext 和 writetext。

视图解析

定义视图时，Adaptive Server 将校验 from 子句中的所有表或视图是否存在。 通过视图查询时也会进行类似的检查。

从定义视图到在语句中使用视图这段时间内，情况可能会发生变化。 例如，视图定义的 from 子句中的一个或多个表或视图可能已被删除。 或者视图定义的 select 子句中的一个或多个列可能已被重命名。

要完全解析视图，Adaptive Server 将校验：

- 派生视图的所有表、视图和列仍然存在。
- 视图列所依赖的每列的数据类型尚未被更改成不兼容的类型。
- 如果语句为 update、insert 或 delete，则并不违反修改视图的限制。 这些将在第 324 页上的“通过视图修改数据”中进行讨论。

如果上述任何检查失败，Adaptive Server 都会发出错误消息。

重定义视图

Adaptive Server 允许重定义视图，而无需重定义依赖于它的其它视图，除非重定义过程使 Adaptive Server 不能转换相关视图。

例如，authors 表和三个可能的视图显示如下。 每个后继视图都是通过其前面的视图来定义的： view2 是从 view1 创建的，view3 是从 view2 创建的。 因此，view2 依赖于 view1，view3 依赖于前面两个视图。

每个视图的名称后面都跟随创建它所用的 select 语句。

view1：

```
create view view1
as select au_lname, phone
from authors
where postalcode like "94%"
```

view2：

```
create view view2
as select au_lname, phone
from view1
where au_lname like "[M-Z]%"
```

view3：

```
create view view3
as select au_lname, phone
from view2
where au_lname = "MacFeather"
```

这些视图所基于的 authors 表由以下列组成： au_id、au_lname、au_fname、phone、address、city、state 和 postalcode。

可以删除 view2 并用另一个视图代替，同样命名为 view2，此视图在选择标准方面略有不同，如：

```
create view view2
as select au_lname, phone
from view3
where au_lname like "[M-P]"
```

依赖于 view2 的 view3 仍然有效并且不需要重新定义。 如果使用的查询引用 view2 或引用 view3，则视图解析照常发生。

如果重定义 view2 使其不能派生 view3，则 view3 就会无效。 例如，如果 view2 的另一个新版本只包含一个列 au_lname，而不包含 view3 所要求的那两列，则不能再使用 view3，因为它不能从其所依赖的对象上派生 phone 列。

但是，view3 仍存在，可以通过删除 view2 然后重新创建带有 au_lname 和 phone 列的 view2 来重新使用 view3。

简而言之，只要相关视图的 select 列表有效，就可更改中间视图的定义而不会影响相关视图。 如果违反了此规则，引用了无效视图的查询就会输出错误消息。

重命名视图

可以使用 sp_rename 来重命名视图：

sp_rename *objname* , *newname*

例如，要将 titleview 重命名为 bookview，可输入：

```
sp_rename titleview, bookview
```

重命名视图时应遵循下述约定：

- 确保新名称符合第 8 页上的“标识符”中所述的标识符规则。

- 只能更改自己拥有的视图的名称。 数据库所有者可以更改任何用户的视图的名称。
- 确保此视图在当前数据库中。

改变或删除基础对象

可以更改视图的基础对象的名称。 例如，如果视图引用了名为 new_sales 的表，并将此表重命名为 old_sales，则视图将使用重命名后的表。

但是，如果视图所引用的表已被删除，当有人试图使用该视图时，Adaptive Server 就会输出错误消息。 如果创建了新的表或视图来替代已被删除的表或视图，则此视图将再次变为可用。

如果用 select * 子句定义视图，然后用添加列的方法来改变其基础表的结构，就不会显示新列。 这是因为首次创建视图时，星号速记符得到了解释和扩展。 要查看新列，可删除此视图然后重新创建。

通过视图修改数据

虽然 Adaptive Server 对通过视图检索数据没有限制，并且与 SQL 其它版本相比，Transact-SQL 对通过视图修改数据的限制也较少，但是各种数据修改操作都应遵循下述规则：

- 不允许进行要对视图中的计算列或内部函数进行引用的 update、insert 或 delete 操作。
- 不允许进行要对包含集合或行集合的视图进行引用的 update、insert 或 delete 操作。
- 不允许进行要对 distinct 视图进行引用的 insert、delete 和 update 操作。
- 除非基础表或视图中的所有 NOT NULL 列都包含在用来插入新行的视图中，否则不允许使用 insert 语句。 Adaptive Server 无法为基础对象中的 NOT NULL 列提供值。
- 如果视图中有 with check option 子句，则通过该视图（或通过派生的视图）插入或更新的所有行都必须符合该视图的选择标准。
- delete 语句不得用于多表视图。

- insert 语句不得用于通过 with check option 子句创建的多表视图。
- update 语句可用于使用了 with check option 的多表视图。如果任何受影响的列出现在 where 子句中，或出现在包含来自多个表的列的表达式中，更新将会失败。
- insert 和 update 语句不得用于多表 distinct 视图。
- update 语句不能为 IDENTITY 列指定值。 为列的基表设置 identity_insert on 后，表所有者、数据库所有者或系统管理员可以在 IDENTITY 列中对显式值进行 insert 操作。
- 如果通过多表视图来插入或更新行，所有受影响的列都必须属于同一基表。
- writetext 不得用于视图中的 text 和 image 列。

在视图中执行 update、insert 或 delete 操作时，Adaptive Server 要进行检查，以确保没有违反上述限制以及没有违反数据完整性规则的情况。

更新视图的限制

更新视图的限制适用于下述情况：

- 视图定义中的计算列
- 视图定义中的 group by 或 compute
- 基础对象中的空值
- 用 with check option 创建的视图
- 多表视图
- 带有 IDENTITY 列的视图

视图定义中的计算列

此限制适用于从计算列或内部函数派生的视图的列。 例如，视图 accounts 中的 amt_due 列是计算列。

```
create view accounts (title_id, advance, amt_due)
as select titles.title_id, advance,
(price * royalty/100) * total_sales
from titles, roysched
where price > $15
  and advance > $5000
```

```
and titles.title_id = roysched.title_id
and total_sales between lorange and hirange
```

可通过 accounts 查看的行如下：

```
select * from accounts
title_id     advance     amt_due
--------     --------    ---------
PC1035       7,000.00    32,240.16
PC8888       8,000.00     8,190.00
PS1372       7,000.00       809.63
TC3218       7,000.00       785.63

 (4 rows affected)
```

不允许对 amt_due 列进行 updates 和 inserts 操作，因为无法从 amt_due 列中的输入值推知价格、版税和年销售额的基值。因为没有需要删除的基值，所以 delete 操作毫无意义。

视图定义中的 *group by* 或 *compute*

此限制适用于包含集合值的视图（即其定义中包含 group by 或 compute 子句的视图）中的所有列。 下面是一个用 group by 子句定义的视图和通过它可查看到的行：

```
create view categories (category, average_price)
as select type, avg(price)
from titles
group by type
select * from categories
category         average_price
-------------    -------------
UNDECIDED                 NULL
business                 13.73
mod_cook                 11.49
popular_comp             21.48
psychology               13.50
trad_cook                15.96

(6 rows affected)
```

在视图 categories 中对行进行 insert 操作毫无意义。 被插入的行会属于哪个基础行组？ 因为无法从输入的值得知应该如何更改基价，所以不允许更新 average_price 列。

基础对象中的 NULL 值

当表或派生此视图的视图中包含一些 NOT NULL 列时，此限制适用于 insert 语句。

例如，假定视图下面的表的列中不允许有空值。 通常，在通过视图对新行进行 insert 操作时，所有未包含在此视图内的基表中的列的值都为空值。 如果其中的一列或多列不允许有空值，则不允许通过此视图进行插入操作。

例如，在此视图中：

```
create view business_titles
as select title_id, price, total_sales
from titles
where type = "business"
```

基础表 titles 的 title 列中不允许有空值，所以不允许通过 business_view 使用 insert 语句。 虽然视图中甚至不存在 title 列，但其禁止空值的限制使得在此视图中的任何插入操作都是非法的。

同样，如果 title_id 列具有唯一的索引，那些会重复基础表中任何值的更新或插入操作都受到拒绝，即使条目不重复视图中的任何值也是如此。

用 *with check option* 创建的视图

此限制用于确定通过视图的检查选项可进行修改的类型。 如果视图具有 with check option 子句，则通过该视图插入或更新的每一行必须能通过该视图查看。 无论是直接地还是通过另一个派生视图间接地插入或更新视图都是如此。

多表视图

对于连接多表中列的视图，此限制用于确定通过它们可以进行修改的类型。 Adaptive Server 禁止在多表视图上使用 delete 语句，但是允许使用其它系统所不允许的 update 和 insert 语句。

下述情况下，可以对多表视图进行 insert 或更新操作：

- 该视图没有 with check option 子句。
- 所有被插入或更新的列都属于同一基表。

以下面视图为例，该视图包含了 titles 和 publishers 中的列，但不没有 with check option 子句：

```
create view multitable_view
as select title, type, titles.pub_id, state
from titles, publishers
where titles.pub_id = publishers.pub_id
```

单个 insert 或更新语句*或者*可以指定 titles 中的列的值*或者*可以指定 publishers 中的列的值：

```
update multitable_view
set type = "user_friendly"
where type = "popular_comp"
```

然而此语句因影响了 titles 和 publishers 中的列而失败：

```
update multitable_view
set type = "cooking_trad",
state = "WA"
where type = "trad_cook"
```

带有 IDENTITY 列的视图

此限制用于确定对包含 IDENTITY 列的视图可进行修改的类型。根据定义，IDENTITY 列是不可更新的。 通过视图所进行的更新不能指定 IDENTITY 列的值。

将 IDENTITY 列的插入操作限制为：

- 表所有者
- 表所有者授权的数据库所有者或系统管理员
- 通过使用 setuser 命令充当表所有者的数据库所有者或系统管理员。

要通过视图来启用这类插入，可对此列的基表使用 set identity_insert on。 对于通过其进行插入的视图，仅使用 set identity_insert on 是不够的。

删除视图

要从数据库中删除视图，可使用 drop view：

drop view [*owner*.]*view_name* [, [*owner*.]*view_name*]...

可见，一次可删除多个视图。 只有视图的所有者（或数据库所有者）可以删除视图。

发出 drop view 后，就会从 sysprocedures、sysobjects、syscolumns、syscomments、sysprotects 和 sysdepends 中删除有关视图的信息。对该视图拥有的所有权限也将被删除。

如果视图所依赖的表或另一个视图已被删除，当有人试图使用该视图时，Adaptive Server 将返回错误消息。 如果创建新的表或视图来代替已被删除的表或视图，并且其名称与被删除的表或视图的名称相同，只要视图定义中所引用的列存在，此视图就会再次变为可用。

使用视图作为安全性机制

无论视图基础表的权限如何，必须明确授予或撤消访问视图中数据子集的权限。授权访问视图而非基础表的用户无法看到视图中没有的基础表数据。

例如，可能想禁止某些用户访问 titles 表中有关销售额和销售量的列。 可以创建 titles 表中不包含这些列的一个视图，然后将该视图的权限授予所有用户，而将访问表的权限只授予销售部门。例如：

```
revoke all on titles to public
grant all on bookview to public
grant all on titles to sales
```

有关如何对权限进行 grant 或 revoke 操作的信息，参见*系统管理指南*。

获取有关视图的信息

系统过程、类别存储过程和 Adaptive Server 内部函数从系统表中提供视图的信息。

有关系统过程的完整信息，参见*参考手册*。

通过 *sp_help* 获取有关视图的帮助

可以使用 sp_help 来获取关于视图的报告。例如：

```
sp_help hiprice
Name           Owner   type
---------      ------  -----
hiprice        dbo     view

(1 row affected)
Data_located_on_segment        When_created
------------------------------ --------------------
not applicable                 July 7 1997 11:57AM

Column_name     Type     Length   Precision   Scale
-----------     -------  ------   ---------   -----
title_id        tid         6         NULL    NULL
title           varchar    80         NULL    NULL
type            char       12         NULL    NULL
pub_id          char        4         NULL    NULL
price           money       8         NULL    NULL
advance         money       8         NULL    NULL
total_sales     int         4         NULL    NULL
notes           varchar   200         NULL    NULL
pubdate         datetime    8         NULL    NULL
contract        bit         1         NULL    NULL
```

```
Nulls      Default_name Rule_name    Identity
------     ------------ ---------    --------
0          NULL             NULL            0
0          NULL             NULL            0
0          NULL             NULL            0
1          NULL             NULL            0
1          NULL             NULL            0
1          NULL             NULL            0
1          NULL             NULL            0
1          NULL             NULL            0
0          NULL             NULL            0
0          NULL             NULL            0

No defined keys for this object.

(return status = 0)
```

使用 *sp_helptext* 显示视图信息

系统安全员必须在已评估的配置中重新设置 allow select on syscomments.text column 配置参数。（有关详细信息，参见“词汇表”中的**已评估的配置**。）这种情况下，只有视图的创建者或系统管理员才能通过 sp_helptext 来查看视图的文本。

要显示 create view 语句的文本，可执行 sp_helptext：

```
sp_helptext hiprice
----------
        1

(1 row affected)

text
---------------------------------------------
create view hiprice
as select *
from titles
where price > $15 and advance > $5000

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

如果视图的源文本已用 sp_hidetext 加密，Adaptive Server 会显示一条消息，提示用户文本已被隐藏。有关隐藏源文本的详细信息，参见*参考手册*中的 sp_hidetext。

使用 *sp_depends* 列出相关对象

sp_depends 可列出视图或表在当前数据库中引用的所有对象，以及所有引用该视图或表的对象。

```
sp_depends titles
Things inside the current database that reference
the object.
object                type
-------------         ---------------------------
dbo.history_proc      stored procedure
dbo.title_proc        stored procedure
dbo.titleid_proc      stored procedure
dbo.deltitle          trigger
dbo.totalsales_trig   trigger
dbo.accounts          view
dbo.bookview          view
dbo.categories        view
dbo.hiprice           view
dbo.multitable_view   view
dbo.titleview         view

(return status = 0)
```

列出数据库中的所有视图

以下面格式使用 sp_tables 时，可列出数据库中的所有视图：

```
sp_tables @table_type = "'VIEW'"
```

查找对象名称和 ID

系统的 object_id() 和 object_name() 功能用来标识视图的 ID 和名称。例如：

```
select object_id("titleview")
----------
 480004741
```

对象的名称和 ID 存储在 sysobjects 表中。

第 10 章

使用查询中的内部函数

内部函数是对从数据库返回信息的 SQL 的 Transact-SQL 扩展。

本章讨论如下内容：

主题	页
返回数据库信息的系统函数	333
用于字符串或表达式的字符串函数	338
文本函数用于 text 和 image 数据	348
集合函数	351
数学函数	352
日期函数	355
数据类型转换函数	360
安全性函数	369

有关内部函数的参考资料，参见*参考手册*。

返回数据库信息的系统函数

系统函数返回数据库中特定的信息。 许多系统函数提供查询系统表的快捷方式。

系统函数的一般语法是：

select *function_name*(*argument*[*s*])

可以在 select 列表、where 子句以及允许使用表达式的任何地方使用系统函数。

例如，要找到以“harold”类型登录的同事的用户标识号，键入：

```
select user_id("harold")
```

假定“harold”的用户 ID 为 13，则结果为：

```
-------------
           13

(1 row affected)
```

通常，函数名将指示返回信息的类型。

系统函数 user_name 采用一个 ID 号作为其自变量并返回用户名：

```
select user_name(13)
---------
harold

(1 row affected)
```

要找到当前用户名（即您的名称），则省略该自变量：

```
select user_name()
---------
dbo

(1 row affected)
```

Adaptive Server 对用户 ID 的处理如下所示：

- 系统管理员在使用的数据库中通过假定**服务器用户 ID** 为 1 可以成为数据库所有者。
- 服务器用户 ID 1 总是给予“guest”用户。
- 在数据库中，数据库所有者的 user_name 始终是“dbo”；其用户 ID 为 1。
- 在数据库中，“guest”用户 ID 始终是 2。

表 10-1 列出每个系统函数的名称、采用的自变量以及返回的结果。

表 *10-1*：系统函数、自变量及结果

函数	自变量	结果
col_name	(*object_id*, *column_id* [, *database_id*])	返回列名。
col_length	(*object_name*, *column_name*)	返回已定义的列长度。 使用 datalength 来查看实际的数据大小。
curunreservedpgs	(*dbid*, *lstart*, *unreservedpgs*)	返回磁盘区段中的可用页数。如果数据库已打开，将从内存中取值；如果未使用数据库，则从 sysusages 的 unreservedpgs 列中取值。

函数	自变量	结果
data_pgs	([*dbid*,] *object_id* , {*doampg* | *ioampg* })	返回表（*doampg*）或者索引（*ioampg*）所使用的页数。 结果不包括用于内部结构的页。 在对 sysindexes 表运行查询时可以使用这一函数。
datalength	(*expression*)	返回字节中表达式的长度。*expression* 通常是一个列名。如果 *expression* 是字符串常量，则必须用引号引起来。
db_id	([*database_name*])	返回数据库 ID 号。*database_name* 必须是字符表达式；如果它是常量表达式，则必须用引号引起来。 如果未提供 *database_name*，db_id 将返回当前数据库的 ID 号。
db_name	([*database_id*])	返回数据库的名称。*database_id* 必须是一个数字表达式。如果未提供 *database_id*，db_name 将返回当前数据库的名称。
host_id	( )	返回客户端进程（不是 Adaptive Server 进程）的主机进程 ID。
host_name	( )	返回当前客户端进程（不是 Adaptive Server 进程）的主计算机名。
index_col	(*object_name*, *index_id* , *key_* #[, *user_id*])	返回索引列的名称；如果 *object_name* 不是表名或视图名，则返回 NULL。
isnull	(*expression1*, *expression2*)	当 *expression1* 的求值为 NULL 时，用 *expression2* 中指定的值进行替换。该表达式的数据类型必须隐式转换，或使用 convert 函数进行转换。
lct_admin	({{ "lastchance" | "logfull" | "unsuspend"} , *database_id* } | "reserve", *log_pages* })	管理日志段的最后机会阈值。 lastchance 在指定数据库中创建最后机会阈值。 logfull 如果在指定的数据库中已超过最后机会阈值，则返回 1，否则返回 0。 unsuspend 将唤醒数据库中挂起的任务，如果已超过该阈值，则禁用最后机会阈值。 reserve 将返回成功转储指定大小的事务日志所需的可用日志页数。
object_id	(*object_name*)	返回对象 ID。
object_name	(*object_id* [, *database_id*])	返回对象的名称。
reserved_pgs	(*object_id*, {*doampg*|*ioampg*})	返回分配给表或索引的页数，包括用于内部结构的报告页。
rowcnt	(*doampg*)	返回表的行数（估计值）。

函数	自变量	结果
suser_id	([*server_user_name*])	返回服务器用户在 syslogins 中的 ID 号。 如果未提供 *server_user_name*，它将返回当前用户的服务器 ID。
suser_name	([*server_user_id*])	返回服务器用户的名称。 服务器用户的 ID 存储在 syslogins 中。 如果未提供 *server_user_id*，它将返回当前用户的名称。
tsequal	(timestamp, timestamp2)	比较 timestamp 值，以防止更新被选择用于浏览并已修改的行。timestamp 是浏览行的时间戳； timestamp2 是存储行的时间戳。 允许不调用 DB-Library 即可使用浏览模式。
used_pgs	(*object_id*, *doampg*, *ioampg*)	返回表及其聚簇索引所用的总页数。
用户		返回用户的名称。
user_id	([*user_name*])	返回用户的 ID 号。 sysusers 报告用户在当前数据库中的 ID 号。 如果未提供 *user_name*，它将返回当前用户的 ID。
user_name	([*user_id*])	基于当前数据库中用户 ID，返回数据库中的用户名。 如果未提供 *user_id*，它将返回当前用户的名称。
valid_name	(*character_expression*)	如果 *character_expression* 不是有效标识符（非法字符或者长度超过 30 字节），则返回 0，如果是有效标识符，就返回非 0 数字。
valid_user	(*server_user_id*)	当指定 ID 至少在该 Adaptive Server 上的一个数据库中是有效用户或别名时则返回 1。 必须设置 sa_role 或者 sso_role 角色才能以自己之外的 *server_user_id* 使用该函数。

当系统函数的自变量为可选时，就会采用当前的数据库、主计算机、服务器用户或数据库用户。 使用 user 例外时，即使自变量为 NULL，内部函数也始终使用小括号。

使用系统函数的示例

本节中的示例使用这些系统函数：

- col_length
- datalength
- isnull
- user_name

col_length

该查询查找 titles 表（表包括“x=”，因此结果有一个列标题）中 title 列的长度：

```
select x = col_length("titles", "title")
 x
--------
      80

 (1 row affected)
```

datalength

col_length 查找定义的列长度，与其相反，datalength 报告每行中存储的数据实际长度（以字节表示）。因为 varchar、nvarchar、varbinary、text 和 image 数据类型可以存储可变长度（但不存储尾随空白），所以对于这些数据类型可以使用该函数。任何 NULL 数据的 datalength 都将返回 NULL。当声明 char 值可为 NULL 时，Adaptive Server 在内部将其存储为 varchar。所有其它数据类型将报告其已定义长度。 下面是查找 publishers 表中 pub_name 列的长度示例：

```
select Length = datalength(pub_name), pub_name
from publishers
Length  pub_name
------  ------------------------
13      New Age Books
16      Binnet & Hardley
20      Algodata Infosystems

(3 rows affected)
```

isnull

该查询查找所有标题的平均价格，同时将 price 中所有 NULL 条目的值替换为“$10.00”：

```
select avg(isnull(price,$10.00))
from titles
------------
       14.24

(1 row affected)
```

user_name

该查询查找 sysusers 中的行（其中名称等于对用户 ID 1 应用系统函数 user_name 的结果）：

```
select name
from sysusers
where name = user_name(1)
name
------------------------
dbo

(1 row affected)
```

用于字符串或表达式的字符串函数

字符串函数用于字符串或表达式的多种操作。 少数字符串函数可用于二进制数据及字符数据。 也可以并置二进制数据、字符串或表达式。

字符串内部函数返回在字符数据运算中通常所需的值。字符串函数名不是关键字。

字符串函数通常采用的语法形式为：

select *function_name*(*arguments*)

可以并置二进制或字符串或表达式，如下所示：

select (*expression* + *expression* [+ *expression*]...)

并置非字符表达式和非二进制表达式时，必须使用 convert 函数：

```
select "The price is " + convert(varchar(12),price)
from titles
```

大多数字符串函数只能用于 char、nchar、unichar、varchar、univarchar 和 nvarchar 数据类型以及隐式转换成 char、unichar 或者 varchar、univarchar 的数据类型。 少数字符串函数也可用于 binary 和 varbinary 数据。patindex 也可用于 text、char、nchar、varchar 和 nvarchar 列。

可以并置 binary 和 varbinary 列以及 char、unichar、nchar、varchar、univarchar 和 nvarchar 列。 如果用 char、nchar、nvarchar 和 varchar 并置 unichar 和 univarchar，结果是 unichar 或 univarchar。 但是，*不能* 并置 text 或 image 列。

可以嵌套字符串函数并且在允许使用表达式的任何地方使用它们。 如果将常量用于字符串函数，则应将其加以单引号或双引号。

每种函数还接受可隐式地转换为指定类型的自变量。 例如，接受近似数值表达式的函数还可接受整数表达式。 Adaptive Server 自动将自变量转换成所需类型。

表 10-2 列出了字符串函数中使用的自变量。 如果函数使用相同类型的多个表达式，自变量将编为 *char_expr1*、*char_expr2*，等等。

表 *10-2*：字符串函数中使用的自变量

自变量类型	可替换为
char_expr	类型为 char、unichar、varchar、univarchar、nchar 或 nvarchar 的字符型列名、变量或常量表达式。 解释中将指明接受 text 列名的函数。常量表达式必须加以引号。
uchar_expr	类型为 unichar 或 univarchar 的字符型列名、变量或常量表达式。 解释中将指明接受 text 列名的函数。常量表达式必须加以引号。
expression	二进制或字符类型的列名、变量或常量表达式。 可以是 char、varchar、nchar、nvarchar、unichar 或 univarchar 数据，就 *char_expr* 而言，再加上 binary 或 varbinary。
pattern	char、nchar、varchar 或 nvarchar 数据类型的字符表达式可能包括 Adaptive Server 支持的任何模式匹配的通配符。
approx_numeric	任何近似数值类型（*float*、*real* 或 *double precision*）的列名、变量或常量表达式。
integer_expr	任何整数（例如 tinyint、smallint 或 int）、列名、变量或常量表达式。 应用它们时，要注意它们的最大大小范围。
start	一个 *integer_expr*。
length	一个 *integer_expr*。

表 10-3 列出了函数名、自变量及结果。

表 *10-3*：字符串函数、自变量及结果

函数	自变量	结果
ASCII	(*char_expr*)	返回表达式中第一个字符的 ASCII 码。
char	(*integer_expr*)	将单字节 integer 值转换成 character 值。char 通常作为 ascii 的倒数使用。*integer_expr* 必须在 0 到 255 之间。返回 char 数据类型。如果所得值是一个多字节字符的第一个字节，那么该字符可能尚未定义。
charindex	(*expression1*, *expression2*)	在 *expression2* 中搜索首次出现的 *expression1* 并返回表示其起始位置的整数。 如果没找到 *expression1*，就返回 0。 如果 *expression1* 包含通配符，charindex 就会将它们当作文字。
char_length	(*char_expr*)	返回一个整数，表示字符表达式或 text 值中的字符数。 对于表列中的可变长度数据，char_length 将返回字符数。 对于固定长度数据，将返回列的指定长度。 对于多字节字符集，表达式中字符数通常小于字节数；使用 datalength 系统函数可确定字节数。
difference	(*char_expr1*, *char_expr2*)	返回一个整数用来表示两个 soundex 值之间的差值。 参见下面的 soundex。
lower	(*char_expr*)	转换大写到小写。 返回字符值。
ltrim	(*char_expr*)	删除字符表达式中的前导空白。 而且只删除 SQL 特殊字符说明中等于空格字符的值。
patindex	("%*pattern*%", *char_expr* [using {bytes | chars | characters}])	返回一个整数，表示特殊字符表达式中首次出现的 *pattern* 的起始位置；如果没找到 *pattern*，则返回 0。 缺省情况下，patindex 返回的偏移量以字符表示。 要返回以字节表示的偏移量（多字节字符串），则指定 using bytes。 通配符%必须位于 *pattern* 之前和之后（搜索第一个或最后一个字符的情况除外）。 有关可用于 *pattern* 的通配符说明，参见第 43 页上的“查询结果中的字符串”。patindex 可用于 text 数据。
replicate	(*char_expr*, *integer_expr*)	返回与 *char_expr* 具有相同数据类型的字符串。该字符串是通过将同一表达式重复特定次数后得到的；如果产生的字符串长度将超过 255 个字节，则实际重复的次数将降到使产生的字符串刚好为 255 个字节空间所能容纳为止。
reverse	(*expression*)	返回字符或二进制表达式的逆序字符串；如果 *expression* 是“abcd”，它将返回“dcba”；如果 *expression* 是 0x12345000，则返回 0x00503412。
right	(*expression, integer_expr*)	返回字符或二进制表达式中从右端开始计算指定字符数的部分。 返回值的数据类型与字符表达式的数据类型相同。
rtrim	(*char_expr*)	删除尾随空白。 而且只删除 SQL 特殊字符定义中等于空格字符的值。

函数	自变量	结果
soundex	(*char_expr*)	对于由有效单字节或双字节罗马字母的邻接序列所组成的字符串，它将返回四个字符的 soundex 代码。
space	(*integer_expr*)	返回由指定数量的单字节空格所组成的字符串。
str	(*approx_numeric* [, *length* [, decimal]])	返回表示浮点数的字符。*length* 设置要返回的字符数（包括小数点、小数点左右两侧的所有位数及空白）； decimal 设置要返回的小数位数。 *length* 和 decimal 是可选自变量。 如果给出这两个变量，它们就必须是非负数。 缺省 *length* 是 10；缺省 decimal 是 0。str 将舍入数字的小数部分，使结果不超出指定的 *length*。
stuff	(*char_expr1*, *start*, *length*, *char_expr2*)	在 *start* 的 *char_expr1* 中删除 *length*，然后在 *start* 处将 *char_expr2* 插入到 *char_expr1* 中。 要在不插入其它字符的情况下删除字符，*char_expr2* 应该是 NULL，而不是表示单个空格的“ ”。
substring	(*expression*, *start*, *length*)	返回字符或二进制字符串的一部分。*start* 指定子串开始的字符位置。*length* 指定子串中的字符名称。
to_unichar	(*integer_expr*)	返回具有整数表达式值的 unichar 表达式。 如果整数表达式在 0xD800 和 0xDFFF 之间，将会引发 *sqlstate* 例外、输出错误并中止操作。 如果整数表达式在 0 和 0xFFFF 之间，将返回单个 Unicode 值。 如果整数表达式在 0x10000 和 0x10FFFF 之间，将返回替换对代理对。
upper	(*char_expr*)	转换小写到大写。 返回一个字符值。
uhighsurr	((*uchar_expr*, *start*)	在 *start* 位置，如果 Unicode 值是替换对代理对的上半部分（应首先出现在对中），则返回 1。 否则返回 0。该函数允许替代处理时写入显式代码。 特别是如果子串从 uhighsurr() 为真的 Unicode 字符开始时，将至少提取两个 Unicode 值的子串，因为 substr() 不能只提取 1 个 Unicode 值的子串。 substr() 不提取半个替换对代理对。
ulowsurr	(*uchar_expr*, *start*)	*start* 位置的 Unicode 值是替换对代理对的下半部（应出现在对中的第二部分）时，返回 1。 否则，返回 0。该函数允许在 substr()、stuff() 和 right() 所执行的调整周围显式地编码。 特别是如果子串的末端位于 ulowsurr() 为真的 Unicode 值上，应提取少于 1（或多 1）个字符的子串时，最适合使用该函数。因为 substr() 不会提取包含不匹配替换对代理对的子串。
uscalar	(*uchar_expr*)	返回表达式第一个 Unicode 字符的 Unicode 标量值。 如果第一个字符*不是*替换对代理对半部高阶，则该值的范围在 0 和 0xFFFF 之间。 如果第一个字符*是*替换对代理对半部的高阶部分，则第二个值必须是半部的低阶部分，并且返回值的范围在 0x10000 和 0x10FFFF 之间。 如果在包含不匹配替换对代理对半部的 uchar_expr 上调用该函数，将引发 SQLSTATE 例外、显示错误并中止操作。

使用字符串函数的示例

本节中的示例使用这些系统函数：

- charindex, patindex
- str
- stuff
- soundex, difference
- substring

charindex, patindex

charindex 和 patindex 函数返回指定模式的起始位置。 它们都使用两个自变量，但是稍有不同，因为 patindex 可以使用通配符，但 charindex 不可以。charindex 仅可用于 char、nchar、unichar、univarchar、varchar、nvarchar、binary 和 varbinary 列； patindex 可用于 char、nchar、unichar、univarchar、varchar、nvarchar 和 text 列。

两个函数都使用两个自变量。 第一个自变量是任意的位置模式。 除非将该模式视为列的首字符（省略前面的%）或者最后字符（省略后面的%），否则使用 patindex 时，必须在模式之前和之后加上百分比符号。 对于 charindex，该模式不可以包括通配符。 第二个自变量是字符表达式，通常是 Adaptive Server 搜索指定的模式中的列名。

titles 表中 notes 列的某行中，为了查找“Wonderful”模式的开始位置，可以使用这两个函数，输入：

```
select charindex("wonderful", notes),
    patindex("%wonderful%", notes)
from titles
where title_id = "TC3218"
------------- -------------
           46            46

(1 row affected)
```

如果不限制搜索行，该查询将返回表的所有行，并将不包含该模式的行报告为零值。 下面的示例中，patindex 将查找以“sys”开始并且第四个字符是“a”、“b”、“c”、或“d”的 sysobjects 的所有行：

```
select name
from sysobjects
where patindex("sys[a-d]%", name) > 0
```

```
name
------------------------------
sysalternates
sysattributes
syscolumns
syscomments
sysconstraints
sysdepends

(6 rows affected)
```

str

该 str 函数将数字转换成字符，用可选自变量指定数字（包括符号、小数点和小数点左右两侧的所有位数）的长度和小数点后的位数。

设置长度和小数自变量为 str 正数。 缺省长度为 10。缺省小数为 0。该长度必须足以容纳小数点和数字符号。结果的小数部分将舍入到指定的长度之内。但是，如果数字的整数部分超出了该长度，str 将返回一行具有指定长度的星号。

例如：

```
select str(123.456, 2, 4)

--
**

(1 row affected)
```

短 *approx_numeric* 在指定长度中右对齐，而长 *approx_numeric* 则被截断到指定的小数位数。

stuff

stuff 函数将一个字符串插入到其它字符串中。 在起始位置，它删除 *expr1* 中字符的指定长度。 然后，在起始位置它将 *expr2* 字符串插入到 *expr1* 字符串中。 如果起始位置或长度为负，则返回 NULL 字符串。

如果起始位置超出 *expr1* 的长度，也将返回 NULL 字符串。 如果要删除的长度超出 *expr1* 的长度，将删除 *expr1* 中的全部字符。例如：

```
select stuff("abc", 2, 3, "xyz")
----
axyz

(1 row affected)
```

要使用 stuff 来删除某个字符，应该用 NULL（而不是空引号）来替换 *expr2*。 使用“ ”指定的用空字符将会将其替换为空格。

```
select stuff("abcdef", 2, 3, null)
---
aef

(1 row affected)
select stuff("abcdef", 2, 3, "")
----
a ef

(1 row affected)
```

soundex, *difference*

soundex 函数可将字母字符串转换为四位代码以便于比较。 在比较中，它忽略的元音。 非字母字符将终止 soundex 求值。 该函数总是返回一些值。 这两个名称有相同的 soundex 代码：

```
select soundex ("smith"), soundex ("smythe")
----- -----
S530  S530
```

difference 函数将比较两个字符串的 soundex 值，并评估二者之间的相似性，最后返回从 0 到 4 的值。最佳匹配值是 4。例如：

```
select difference("smithers", "smothers")
---------
        4

(1 row affected)
select difference("smothers", "brothers")
 ---------
         2

(1 row affected)
```

substring

下面的示例使用 substring 函数。 它显示每个作者的姓和名，如“Bennet A”。

```
select au_lname, substring(au_fname, 1, 1)
from authors
```

substring 返回字符或二进制字符串的一部分。

substring 总是带有三个自变量。 第一个自变量是字符或二进制字符串、列名或包括列名的字符串值表达式 第二个自变量指定子串应该开始的位置。 第三个自变量指定字符数和返回的字符串数中的长度。

substring 的语法如下所示：

substring(*expression*, *start*, *length*)

例如，下面是显示了如何指定字符串常量“abcdef”的第二个、第三个及第四个字符：

```
select x = substring("abcdef", 2, 3)
x
---------
bcd

(1 row affected)
```

以下示例显示如何从二进制字段中提取较低的四位数，其中的每个位置都表示两个二进制位：

```
select substring(xactid,5,2) from syslogs
------
0x0000
0x0001
0x0001
0x0001
0x0001
0x0001
0x0001
0x0001
0x0001
0x0001
0x0001

(11 rows affected)
```

其它字符串函数的示例

剩余大部分的字符串函数都很容易使用和理解。

表 *10-4*：字符串函数示例

语句	结果
select right("abcde", 3)	cde
select right("abcde", 6)	abcde
select right(0x12345000, 3)	0x345000
select right(0x12345000, 6)	0x12345000
select upper("torso")	TORSO
select ascii("ABC")	65

并置

可以并置二进制或字符串表达式 — 组合两个或更多的字符或二进制字符串、字符或二进制数据或它们的组合 — 用 + 字符串并置运算符。

并置字符串时，每个字符表达式必须用单引号或双引号引起来。

并置语法为：

select (*expression* + *expression* [+ *expression*]...)

下面是组合两个字符串的方法：

```
select ("abc" + "def")
-------
abcdef

(1 row affected)
```

这个查询在列标题 *Moniker* 下显示 California 作者姓名，按照先姓后名的顺序，并在姓后加一个逗号和空格：

```
select Moniker = (au_lname + ", " + au_fname)
from authors
where state = "CA"
Moniker
-----------------------------------------------
White, Johnson
Green, Marjorie
Carson, Cheryl
OíLeary, Michael
Straight, Dick
```

```
Bennet, Abraham
Dull, Ann
Gringlesby, Burt
Locksley, Chastity
Yokomoto, Akiko
Stringer, Dirk
MacFeather, Stearns
Karsen, Livia
Hunter, Sheryl
McBadden, Heather

(15 rows affected)
```

为了并置数字或 datetime 数据类型，必须使用 convert 函数：

```
select "The due date is " + convert(varchar(30),
    pubdate)
from titles
where title_id = "BU1032"
--------------------------------------
The due date is Jun 12 1986 12:00AM

(1 row affected)
```

并置和空字符串

Adaptive Server 对空字符串（“”或‘’）作为单个空格求值。该语句为：

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

生成：

```
abc def
```

嵌套字符串函数

可以嵌套字符串函数。 例如，为了显示每个作者的姓和名的首字母，可在姓后加一个逗号并在名后加一个句号，键入：

```
select (au_lname + "," + " " + substring(au_fname,
1, 1) + ".")
from authors
where city = "Oakland"
```

```
-------------------------------------------
Green, M.
Straight, D.
Stringer, D.
MacFeather, S.
Karsen, L.

(5 rows affected)
```

要显示标价 $20 以上的书籍的 pub_id 和每个 title_id 的前两个字符，键入：

```
select substring(pub_id + title_id, 1, 6)
from titles
where price > $20
--------------
1389PC
0877PS
0877TC

(3 rows affected)
```

文本函数用于 *text* 和 *image* 数据

文本内部函数用于 text 和 image 数据操作。表 10-5 列出了文本函数名、自变量及结果：

表 *10-5*：用于文本和图像数据的内部文本函数

函数	自变量	结果
patindex	("%*pattern*%", *char_expr* [using {bytes | chars | characters}])	返回一个整数值，表示特殊字符表达式中首次出现的 *pattern* 的起始位置；如果没找到 *pattern*，则返回 0。 缺省情况下，patindex 返回的偏移以字符表示；要返回以字节表示的偏移（多字节字符串），请指定 using bytes。 通配符 % 必须位于 *pattern* 之前和之后（搜索第一个或最后一个字符的情况除外）。 有关在 *pattern* 中使用通配符的说明，参见第 43 页上的 “查询结果中的字符串”。
textptr	(*text_columname*)	返回文本指针值（16 字节的 varbinary 值）。

函数	自变量	结果
textvalid	("*table_name..col_name*", *textpointer*)	检查给定的文本指针是否有效。 注意：text 或 image 列的标识符必须包含表名。 指针有效时返回 1，无效时返回 0。

datalength 也可用于 text 列。 有关 datalength 的信息，参见第 333 页上的“返回数据库信息的系统函数”。

set textsize 命令以字节为单位指定使用 select 语句返回的 text 或 image 数据的限制。 例如，该命令将用 select 语句返回的 text 或 image 数据大小上限设置为 100 字节。

```
set textsize 100
```

当前设置存储在 *@@textsize* 全局变量中。 缺省设置由客户程序控制。 要重新设置缺省值，发出：

```
set textsize 0
```

也可以使用 *@@textcolid*、*@@textdbid*、*@@textobjid*、*@@textptr* 和 *@@textsize* 全局变量来处理 text 和 image 数据。

Readtext

如果您只想检索一列数据的选定部分，readtext 命令提供了一个用来检索 text 和 image 值的方法。readtext 需要表和列的名字、文本指针、列中的起始偏移量以及要检索的字符数和字节数。

readtext 的完整语法为：

```
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 ] } ]
```

holdlock 标志将锁定文本值以供读取，直到事务结束为止。 其他用户可以读取该值，但不能修改该值。 在第 18 章“事务：维护数据一致性和恢复”中对 at isolation 从句进行了描述。

如果正在使用多字节字符集，using 选项允许您选择是否需要 readtext 将偏移和大小解释为字节或字符。 chars 和 characters 都可以指定字符。 当用于单字节字符集或用于 image 值（readtext 只能在逐字节读取的基础上读取 image 值）时，该选项无效。 如果没有给出 using 选项，readtext 将返回似乎是指定字节的值。

Adaptive Server 必须确定要发送到客户端的字节数，来响应 readtext 命令。 当偏移和大小用字节数表示时，要确定返回文本中的字节数很简单。 当偏移和大小用字符数表示时，Adaptive Server 必须计算返回到客户端的字节数。 因此，在使用字符数表示偏移和大小时，执行速度可能会较慢。 using characters 很有用（当 Adaptive Server 使用多字节字符集时）。 该选项可以确保 readtext 不会返回部分字符。

在使用字节数作为偏移时，Adaptive Server 可能会在要返回的 text 数据的开头或末尾找到部分字符。 如果是这样，服务器将文本返回到客户端之前，将用问号替代每个部分字符。

不能对视图中 text 和 image 列使用 readtext。

使用文本函数的示例

该示例使用 textptr 函数来定位 text 列和 copy，该列与 blurbs 表中的 title_idBU7832 相关联。 文本指针（16 字节二进制字符串）放在局部变量 *@val* 中并作为 readtext 命令的参数提供。readtext 将从第二个字节开始返回 5 个字节（偏移为 1）。

```
declare @val binary(16)
select @val = textptr(copy) from blurbs
where au_id = "486-29-1786"
readtext blurbs.copy @val 1 5
```

textptr 将返回 16 字节的 varbinary 字符串。 Sybase 建议将该字符串放在局部变量中（如上例所示），并通过引用来使用它。

使用前面的 declare 示例中 textptr 的另一种方法是 *@@textptr* 全局变量：

```
readtext texttest.blurb @@textptr 1 5
```

当前 Adaptive Server 进程可以设置 *@@textptr* 值为从最后一个 insert 或 update 到任一 *text* 或 *image* 字段的值。 其它进程的插入和更新并不影响当前进程。

使用 convert 函数，支持从 text 到 char、nchar、unichar、varchar、univarchar 或 nvarchar 的显式转换及从 image 到 varbinary 或 binary 的显式转换，但 text 或 image 数据被截断以适应页大小。 不支持将 text 或 image 隐式或显式地转换成上述之外的数据类型。

集合函数

集合函数生成的摘要值在查询结果中显示为新列。表 10-6 列出了集合函数、其自变量以及它们返回的结果。

表 *10-6*：集合函数

函数	自变量	结果
avg	(all | distinct) *expression*	返回所有（不同）值的数字平均值
count	(all | distinct) *expression*	返回（不同）非空值数量或行数
max	(*expression*)	返回表达式的最大值
min	(*expression*)	返回列中的最小值
sum	(all | distinct) *expression*	返回值的总和

示例如下：

```
select avg(advance), sum(total_sales)
from titles
where type = "business"
------------------------ -----------
                  6281.25      30,788

(1 row affected)
select count(distinct city) from authors
-----------
         16

(1 row affected)
select discount from salesdetail
compute max(discount)
discount
--------------------
           40.000000
           ...
           46.700000

Compute Result:
--------------------
           62.200000
```

```
(117 rows affect)
select min(au_lname) from authors
---------------------
Bennet

(1 row affected)
```

数学函数

数学内部函数返回数学数据运算中通常所需的值。

数学函数采用一般格式：

function_name(*arguments*)

下表列出用于内部数学函数的自变量类型：

表 *10-7*：数学函数中使用的自变量

自变量类型	可替换为
approx_numeric	任何近似数值（float、real 或 double precision）的列名、变量或常量表达式或上述组合
integer	任意整数（tinyint、smallint 或 int）列名、变量、常量表达式或上述数据类型的组合
numeric	任意精确数值（numeric、dec、decimal、tinyint、smallint 或 int），近似数值（float、real 或 double precision），或 money 列、变量、常量表达式，或上述数据类型的组合
power	任意精确数值、近似数值或 money 列、变量或常量表达式，或上述数据类型的组合

每种函数还可接受隐式地转换为指定类型的自变量。 例如，接受近似数值类型的函数还可接受整数类型。 Adaptive Server 将自变量转换成所需类型。

如果函数使用相同类型的多个表达式，表达式将编号（例如，*approx_numeric1*、*approx_numeric2*）。

表 10-8 列出了数学函数、其自变量和返回的结果：

表 *10-8*：数学函数

函数	自变量	结果
abs	(numeric)	返回给定表达式的绝对值。结果与数字表达式属于同一类型并且具有相同的精度和标度。
acos	(*approx_numeric*)	返回已指定余弦值的角（以弧度表示）。
asin	(*approx_numeric*)	返回已指定正弦值的角（以弧度表示）。

函数	自变量	结果
atan	(*approx_numeric*)	返回已指定正切值的角（以弧度表示）。
atn2	(*approx_numeric1*, *approx_numeric2*)	返回已指定正切值是 (*approx_numeric1*/*approx_numeric2*) 的角（以弧度表示）。
ceiling	(numeric)	返回大于或等于指定值的最小整数。其结果与数字表达式的类型相同。 对于 numeric 和 decimal 表达式，其结果的精度与该表达式的精度相同，且标度为零。
cos	(*approx_numeric*)	返回指定角的三角余弦（以弧度表示）。
cot	(*approx_numeric*)	返回指定角的三角余弦（以弧度表示）。
degrees	(numeric)	将弧度转换为度。其结果与数字表达式的类型相同。 对于 numeric 和 decimal 表达式，其结果的内部精度为 77，标度与该表达式的标度相同。 当使用 money 数据类型时，如果在内部转换为 float，则会导致精度损失。
exp	(*approx_numeric*)	返回指定值的指数值。
floor	(numeric)	返回小于或等于指定值的最大整数。其结果与数字表达式的类型相同。 对于 numeric 或 decimal 类型的表达式，其结果的精度与表达式的精度相同，标度为 0。
log	(*approx_numeric*)	返回指定值的自然对数。
log10	(*approx_numeric*)	返回指定值的以 10 为底的对数。
pi	()	返回常数值 3.1415926535897931。
power	(numeric, power)	返回 numeric 的 power 幂的值。其结果与 numeric 具有相同的类型。 对于 numeric 或 decimal 类型的表达式，其结果的精度是 77，标度与该表达式的标度相同。
radians	(*numeric_expr*)	将度转换为弧度。其结果与 numeric 具有相同的类型。 对于 numeric 或 decimal 类型的表达式，其结果的内部精度是 77，标度与 numeric 表达式的标度相同。 当使用 money 数据类型时，如果在内部转换为 float，则会导致精度损失。
rand	([*integer*])	使用可选整数作为源值，返回 0 和 1 之间的随机浮动值。
round	(numeric, *integer*)	舍入 numeric 从而使其具有 *integer* 有效数字。 正 *integer* 确定小数点右边的有效位数；负 *integer* 确定小数点左边的有效位数。 结果与数值表达式的类型相同，对于 numeric 和 decimal 表达式，其结果的内部精度等于第一个参数的精度加 1，其标度等于 numeric 的标度。
sign	(numeric)	返回 numeric 的符号： 正 (+1)、零 (0) 或负 (-1)。 结果与 numeric 表达式属于同一类型并且具有相同的精度和标度。
sin	(*approx_numeric*)	返回指定角的三角正弦（以弧度表示）。
sqrt	(*approx_numeric*)	返回指定值的平方根。 值必须是正数或零。
tan	(*approx_numeric*)	返回指定角的三角正切（以弧度表示）。

使用数学函数的示例

数学内部函数对数值数据进行运算。 某些函数需要整数数据和其它近似数值数据。 一些函数对精确数值、近似数值、money 和 float 类型进行运算。 内部运算 float 类型数据的精度缺省为六位小数。

为处理数学函数的域错误或范围错误，提供了错误陷阱。 用户可以通过使用 set arithabort 和 arithignore 选项来确定如何处理域错误。 有关这些选项的详细信息，参见第 366 页上的“转换错误”。

表 10-9 显示了一些简单的数学函数示例：

表 *10-9*：数学函数的示例

语句	结果
select floor(123) select floor(123.45) select floor(1.2345E2) select floor(-123.45) select floor(-1.2345E2) select floor($123.45)	123 123.000000 123.000000 -124.000000 -124.000000 123.00
select ceiling(123.45) select ceiling(-123.45) select ceiling(1.2345E2) select ceiling(-1.2345E2) select ceiling($123.45)	124.000000 -123.000000 124.000000 -123.000000 124.00
select round(123.4545, 2) select round(123.45, -2) select round(1.2345E2, 2) select round(1.2345E2, -2)	123.4500 100.00 123.450000 100.000000

round(*numeric*、*integer*) 函数总返回值。 如果 *integer* 是负数和超过 *numeric* 中的有效位数，Adaptive Server 只舍入最重要位。 例如，返回值 100.00：

```
select round(55.55, -3)
```

小数点右边零的个数与第一个自变量精度加 1 的标度相等。

日期函数

日期内部函数执行数学运算并显示有关 datetime 和 smalldatetime 值的信息。

Adaptive Server 在内部用 datetime 数据类型以两个 4 字节整数形式存储值。 前 4 个字节存储基准日期（以 1900 年 1 月 1 日为基准日期）以前或以后的天数。基准日期是系统参考日期。datetime 值不允许早于 1753 年 1 月 1 日。 另外四个内部 datetime 字节表示存储的时间精度达 1/300 秒。

smalldatetime 数据类型用低于 datetime 的精度存储日期和时间。smalldatetime 值作为两个 2 字节整数存储。 前 2 个字节存储 1900 年 1 月 1 日以后的天数。后 2 个字节存储午夜后的分钟数。 从 1900 年 1 月 1 日到 2079 年 6 月 6 日的日期，其精确度可以精确到分。

缺省显示日期格式如下所示：

```
Apr 15 1997 10:23PM
```

有关改变 datetime 或 smalldatetime 显示格式的信息，参见第 362 页上的“使用常规用途的转换函数：转换”。 当输入 datetime 或 smalldatetime 值时，用单引号或双引号将它们括起来。Adaptive Server 可能会舍入或截断毫秒值。

Adaptive Server 识别范围广泛的 datetime 数据条目格式。 有关 datetime 和 smalldatetime 值的详细信息，参见第 7 章“创建数据库和表”和第 8 章“添加、更改和删除数据”。

表 10-10 列出日期函数及其产生的结果：

表 *10-10*：日期函数

函数	自变量	结果
getdate	()	当前的系统日期和时间。
datename	(*datepart*, *date*)	datetime 或 smalldatetime 值作为 ASCII 串的一部分
datepart	(*datepart*, *date*)	datetime 或 smalldatetime 值（例如，月份）作为整数的一部分
datediff	(*datepart*, *date*, *date*)	在第二个和第一个日期间的时间量，转换为指定日期部分（如月、日、小时）
dateadd	(*datepart*, *number*, *date*)	由添加日期部分到其它日期所产生的日期

datename、datepart、datediff 和 dateadd 函数将 **date part**（年、月、日等）作为自变量。 该 datename 函数在适当情况下产生 ASCII 值（如为一周中的某天）。

datepart 返回符合 ISO 标准 8601 的数字，此标准定义了一周中的第一天和一年中的第一周。 根据 datepart 函数包含的是 calweekofyear 值、calyearofweek 值还是 caldayofweek, 值，对同一时间单元返回的日期可能不同。 例如，如果将 Adaptive Server 配置为以美国英语为缺省语言：

```
datepart(cyr, "1/1/1989")
```

返回 1988，而：

```
datepart(yy, "1/1/1989)
```

返回 1989。

出现这种不一致是由于该 ISO 标准将一年的第一周定义为包含周四*并且*以周一开始的第一周。

如果服务器以美国英语为缺省语言，则一周的第一天是周日，一年的第一周是包含 1 月 4 日的那一周。

表 10-11 列出每个日期分量、其缩写（如果有）和该日期分量的可能整数值。

表 *10-11*：日期分量

日期分量	缩写	值
year	yy	1753–9999
quarter	qq	1–4
month	mm	1–12
week	wk	1–54
day	dd	1–31
dayofyear	dy	1–366
weekday	dw	1–7（星期日 – 星期六）
hour	hh	0–23
minute	mi	0–59
second	ss	0–59
millisecond	ms	0–999

```
select datename (mm, "1997/06/16")

-----------
June

(1 row affected)

select datediff (yy, "1984", "1997")

-----------
```

```
        13

(1 row affected)
select dateadd (dd, 16, "1997/06/16")
------------------------------
           Jul  2 1997 12:00AM

(1 row affected)
```

注意，语言设置将影响 weekday 的值。

以下是一些周日期分量的示例：

```
select datepart(cwk,"1997/01/31")
-----------
          5

(1 row affected)
select datepart(cyr,"1997/01/15")
-----------
       1997

(1 row affected)
select datepart(cdw,"1997/01/24")
-----------
          5

(1 row affected)
```

表 10-12 列出了周数日期分量、其缩写和值

表 *10-12*：周数日期分量

日期分量	缩写	值
calweekofyear	cwk	1–52
calyearofweek	cyr	1753–9999
caldayofweek	cdw	1–7（在美国英语中 1 是星期一）

获取当前日期：*getdate*

getdate 函数以 Adaptive Server 内部格式产生 datetime 值的当前日期和时间。getdate 采用 NULL 自变量 ()。

要查找当前的系统日期和时间，输入：

```
select getdate()
--------------------------
Aug 19 1997 12:45PM

(1 row affected)
```

在设计报告中可使用 getdate，以便每次产生报告时输出当前日期和时间。getdate 对记录帐号上事务的发生时间也是很有用。

要使用毫秒显示日期，请使用 convert 函数。例如：

```
select convert(char(26), getdate(), 109)
--------------------------
Aug 19 1997 12:45:59:650PM

(1 row affected)
```

有关详细信息，参见第 368 页上的“更改日期显示格式”。

查找编号或名称的日期分量

datepart 和 datename 函数产生 datetime 或 smalldatetime 值（年、季度、日、小时等）作为整数或 ASCII 字符串的指定部分。 由于 smalldatetime 仅对于分钟是精确的，所以当在这些函数中使用 smalldatetime 值时，秒和毫秒总是为 0。

以下示例假设日期为 8 月 12 日：

```
select datepart(month, getdate())
--------------
             8

(1 row affected)
select datename(month, getdate())
-------------
August

(1 row affected)
```

计算间隔或增量日期

datediff 函数计算指定的两个日期的第一个和第二个之间日期分量的时间，换句话说就是找到两个日期间的间隔。 结果是带符号的整数值（以日期分量表示），它等于 *date2 - date1*。

本查询使用 1990 年 11 月 30 日并查找在 pubdate 和 1990 年 11 月 30 日之间经历的天数：

```
select pubdate, newdate = datediff(day, pubdate,
    "Nov 30 1990")
from titles
```

对于在有 1990 年 10 月 21 日的 pubdate 的 titles 表中的行，先前查询产生的结果是 40，即在 10 月 21 日和 9 月 30 日之间的天数。要计算月间隔，查询为：

```
select pubdate, interval = datediff(month, pubdate,
    "Nov 30 1990")
from titles
```

本查询在 1990 年 10 月的 pubdate 中产生行值为 1，在 1990 年 6 月 pubdate 中产生行值为 5。当 datediff 函数中的第一天迟于第二天时，返回值是负数。 由于 titles 中的两个行具有使用 getdate 函数作为缺省值指定的 pubdate 值，所以 pubs 数据库被创建并在两个先前查询中返回负值 (-65)。

如果两个日期自变量值中一个或都是 smalldatetime 值，它们将在内部被转换成 datetime 值以用于计算。 为了计算差值，smalldatetime 值中的秒和毫秒会被自动设置为 0。

添加日期间隔：*dateadd*

dateadd 函数添加间隔（指定为整数）到指定的日期。 例如，如果 titles 表中所有书的出版日期延期 3 天，您也可用本语句获得新的出版日期：

```
select dateadd(day, 3, pubdate)
from titles
-------------------
Jun 15 1986 12:00AM
Jun 12 1988 12:00AM
Jul  3 1985 12:00AM
Jun 25 1987 12:00AM
Jun 12 1989 12:00AM
Jun 21 1985 12:00AM
Jul  6 1997  1:43PM
Jul  3 1986 12:00AM
```

```
Jun 15 1987 12:00AM
Jul  6 1997  1:43PM
Oct 24 1990 12:00AM
Jun 18 1989 12:00AM
Oct  8 1990 12:00AM
Jun 15 1988 12:00AM
Jun 15 1988 12:00AM
Oct 24 1990 12:00AM
Jun 15 1985 12:00AM
Jun 15 1987 12:00AM

(18 rows affected)
```

如果日期自变量是 smalldatetime 值，结果也会是 smalldatetime 值。可使用 dateadd 向 smalldatetime 添加秒或毫秒，但只有在 dateadd 返回的结果日期改变了至少一分钟时，这才有意义。

数据类型转换函数

数据类型转换将一种数据类型改变为另一种数据类型并重新格式化日期和时间信息。 Adaptive Server 提供三种数据类型转换函数：convert、inttohex 和 hextoint。

Adaptive Server 自动执行某些数据类型转换。这些转换被称为**隐式转换**。 例如，如果比较 char 表达式和 datetime 表达式，或比较 smallint 表达式和 int 表达式，或比较具有不同长度的 char 表达式，Adaptive Server 就会自动将一种数据类型转换为另一种数据类型。

必须使用一种内部数据类型转换函数显式地请求其它数据类型转换。例如，在并置数字表达式之前，必须将其转换为字符表达式。

对于某些数据类型，Adaptive Server 不允许隐式或显式地将其转换为其它某些数据类型。 例如，不能将 smallint 数据转换为 datetime，或将 datetime 数据转换为 smallint。进行不支持的转换将会导致错误消息。

支持的转换

表 10-13 总结了 Adaptive Server 支持的数据类型转换。

表 *10-13*：显式、隐式和不支持的数据类型转换

原数据类型：　目标数据类型：	tinyint	smallint	int	decimal	numeric	real	float	[n]char	[n]varchar	unichar	univarchar	text	smallmoney	money	bit	smalldatetime	datetime	binary	varbinary	image
tinyint	–	I	I	I	I	I	I	E	E	E	E	U	I	I	I	U	U	I	I	U
smallint	I	–	I	I	I	I	I	E	E	E	E	U	I	I	I	U	U	I	I	U
int	I	I	–	I	I	I	I	E	E	E	E	U	I	I	I	U	U	I	I	U
decimal	I	I	I	I/E	I/E	I	I	E	E	E	E	U	I	I	I	U	U	I	I	U
numeric	I	I	I	I/E	I/E	I	I	E	E	E	E	U	I	I	I	U	U	I	I	U
real	I	I	I	I	I	–	I	E	E	E	E	U	I	I	I	U	U	I	I	U
float	I	I	I	I	I	I	–	E	E	E	E	U	I	I	I	U	U	I	I	U
[n]char	E	E	E	E	E	E	E	I	I	I	I	I	E	E	E	I	I	I	I	I
[n]varchar	E	E	E	E	E	E	E	I	I	I	I	I	E	E	E	I	I	I	I	I
unichar	E	E	E	E	E	E	E	I	I	–	I	I	E	E	E	E	E	I	I	I
univarchar	E	E	E	E	E	E	E	I	I	I	–	I	E	E	E	E	E	I	I	I
text	U	U	U	U	U	U	U	E	E	E	E	U	U	U	U	U	U	U	U	U
smallmoney	I	I	I	I	I	I	I	I	I	E	E	U	–	I	I	U	U	I	I	U
money	I	I	I	I	I	I	I	I	I	E	E	U	I	–	I	U	U	I	I	U
bit	I	I	I	I	I	I	I	I	I	E	E	U	I	I	–	U	U	I	I	U
smalldatetime	U	U	U	U	U	U	U	I	I	E	E	U	U	U	U	–	I	I	I	U
datetime	U	U	U	U	U	U	U	I	I	E	E	U	U	U	U	I	–	I	I	U
binary	I	I	I	I	I	I	I	I	I	I	I	U	I	I	I	I	I	–	I	I
varbinary	I	I	I	I	I	I	I	I	I	I		U	I	I	I	I	I	I	–	I
image	U	U	U	U	U	U	U	U	U	E	E	U	U	U	U	U	U	E	E	U

键：

E　可进行显式数据类型转换。

I　既可以隐式地进行转换，也可以使用显式数据类型转换函数来转换。

I/E 需要可使用显式数据类型转换函数进行转换，前提是有精度或标度损失且启用了 arithabort numeric_truncation；否则，允许只可进行隐式转换。

U　不支持的转换。

–　数据类型向其自身转换。这些转换虽然可以进行，但毫无意义。

使用常规用途的转换函数：转换

常规转换函数 convert，在广泛的数据类型中转换和指定日期和时间信息的新显示格式。此命令的语法为：

convert(*datatype, expression* [, *style*])

这有一个在选择列表中使用 convert 的示例：

```
select title, convert(char(5), total_sales)
from titles
where type = "trad_cook"
title
------------------------------------     -----
Onions, Leeks, and Garlic: Cooking
    Secrets of the Mediterranean           375
Fifty Years in Buckingham Palace
    Kitchens                             15096
Sushi, Anyone?                            4095

 (3 rows affected)
```

在如下示例中，total_sales 列（一个 int 列）转换成 char(5) 列，以便可与 like 关键字一起使用：

```
select title, total_sales
from titles
where convert(char(5), total_sales) like "15%"
    and type = "trad_cook"
title
---------------------------------        -----
Fifty Years in Buckingham Palace
    Kitchens                             15096

(1 row affected)
```

某些数据类型需要使用长度或精度和标度。 如果不指定长度，Adaptive Server 为字符和二进制数据使用缺省长度 30。 如果不指定精度或标度，Adaptive Server 将分别使用缺省袋值 18 和 0。

转换规则

以下各节说明转换不同类型信息时 Adaptive Server 遵循的规则。

将字符数据转换为非字符类型

可以将字符数据转换为非字符类型（例如货币、日期 / 时间、精确数值或近似数值类型），但前提是它必须全部由对于该新类型有效的字符组成。前导空白将被忽略。 然而，如果将包含一个或多个空白的 char 表达式转换为 datetime 表达式，Adaptive Server 就会将空白转换为 Sybase 缺省 datetime 值“Jan 1, 1900”。

当数据包含无法接受的字符时，将产生语法错误。 以下字符类型将导致语法错误：

- 整数数据中的逗号或小数点
- 货币数据中的逗号
- 精确 / 近似数值数据或位流数据中的字母
- 日期 / 时间数据中的误拼月份名

将一种字符类型转换为另一种字符类型

当从多字节字符集向单字节字符集转换时，如果字符无等值的单字节字符，则将被转换为空白。

text 列可以显式地转换为 char、nchar、unichar、varchar、univarchar 或 nvarchar。 但受到数据类型最大长度、页面大小的限制。如果未指定长度，转换后的值将具有缺省的长度（30 个字节）。

将数字转换为字符类型

可将精确和近似数值数据转换为字符类型。如果新类型太短，无法容纳整个字符串，则将产生空间不足的错误。 例如，以下转换试图将 5 个字符的字符串转换为单字符类型：

```
select convert(char(1), 12.34)
```

因为 char 数据类型受一个字符限制，而数值 12.34 需要 5 个字符才能成功转换，所以转换失败。

货币类型转换的舍入

money 和 smallmoney 类型在小数点的右侧存储 4 个数字位，但为了便于显示，它们将上舍入到最接近的百分位 (.01)。 当数据转换为 money 类型时，它将上舍入到第四个小数位。

如果可能，从 money 类型进行转换的数据将遵循相同的舍入规则。如果新类型是小数位少于三的精确数值，数据就会舍入到新类型的标度。例如，当 $4.50 转换为整数后，将得到 5：

```
select convert(int, $4.50)
-----------
          5

(1 row affected)
```

Adaptive Server 假定转换为 money 或 smallmoney 的数据应该是完整货币单位（如美元），而不是辅币单位（如美分）。 例如，整数值 5 将转换为 us_english 语言中的 5 美元（而不是 5 美分）的等值货币。

转换日期和时间信息

可以将可识别为日期的数据转换为 datetime 或 smalldatetime。月份名有误将导致语法错误。超出数据类型可接受范围的日期将导致算术溢出错误。

当 datetime 值转换为 smalldatetime 时，它们将上舍入到最接近的分。

在数字类型之间转换

数据可以从一种数字类型转换为另一种数字类型。如果新类型是精确数值，且其精度或标度不足以容纳数据，就会发生错误。 使用 arithabort 和 arithignore 选项可确定如何处理这些错误。

注意：arithabort 和 arithignore 选项在 SQL Server 版本 10.0 中重新进行了定义。 如果您在应用程序中使用这些选项，请检查它们是否能正确实现预期功能。

转换二进制式数据

Adaptive Server binary 和 varbinary 数据是用于特定平台的数据；您所使用的硬件类型确定如何存储和解释数据。 一些平台认为前缀“0x”后第一个字节是最重要的；其它平台认为第一个字节是不重要的。

convert 函数将 Sybase 二进制数据作为一个字符串对待，而不是数值信息。当将二进制表达式转换为整数或整数表达式时，convert 不考虑字节顺序对二进制值的重要性。 因此，在不同平台上可能会得到不同的转换结果。

在将二进制字符串转换为整数时， convert 将删去其前缀“0x”。如果字符串包含奇数个数字，Adaptive Server 将插入前导零。 当整数类型数据太长，convert 将截断它。 如果数据太短，convert 将添加前导零使其长度相等，然后在其右侧添加零。

假定您要将字符串 0x00000100 转换为整数。 在一些平台上，该字符串代表数字 1 ；其它平台上代表 256。convert 返回 1 还是 256，取决于执行函数的平台。

转换十六进制数据

使用 hextoint 和 inttohex 函数，对于跨平台转换结果是可靠的。

hextoint 接受包括数字和大小写字母 A–F、有或没有前缀“0x”的文字或变量。 以下是所有 hextoint 的有效用法：

```
select hextoint("0x00000100FFFFF")
select hextoint("0x00000100")
select hextoint("100")
```

hextoint 删去前缀“0x”的字符串。 如果数据超过 8 位，则 hextoint 截断它。 如果数据少于 8 位，hextoint 从右侧调整并且用零填充。于是 hextoint 返回独立于平台的等价整数。 执行 hextoint 函数的任何平台，以上表达式都返回相同值 256。

inttohex 函数接受整数数据并返回没有前缀“0x”的 8 字符**十六进制字符串**。无论使用何种平台，inttohex 总返回相同的结果。

将 *image* 数据转换为 *binary* 或 *varbinary*

使用 convert 将 image 列转换为 binary 或 varbinary。这种转换受到 binary 数据类型最大长度（255 个字节）的限制。如果未指定长度，转换后的值将具有缺省的长度（30 个字符）。

在二进制和数字或十进制类型之间转换

在 binary 和 varbinary 数据字符串中，“0x”之后的前两位表示 binary 类型：“00”表示正数，“01”表示负数。 将 binary 或 varbinary 类型转换为 numeric 或 decimal 时，务必要在“00x”位后指定“00”或“01”值；否则转换将失败。

例如，以下是将 binary 数据转换为 numeric 的方法：

```
select convert(numeric
(38,
18),0x000000000000000006b14bd1e6eea000000000000000000000
```

```
000000000000)
----------------------
123.456000000000000000
```

下例将同一 numeric 数据转换回 binary：

```
select convert(binary,convert(numeric(38, 18),
123.456))
-------------------------------------------------------
--------
0x00000000000000006b14bd1e6eea00000000000000000000000000
00000000
```

转换错误

本节说明数据类型转换期间可能发生的错误类型。

算术溢出和除零错误

当 Adaptive Server 试图将数字值除以零时，就会发生除零错误。当新类型的小数位数太少，无法容纳结果时，就会发生算术溢出错误。会发生这些错误的转换过程包括：

- 显式或隐式地转换为具有较低精度或标度的精确类型
- 显式或隐式地转换已超出货币或日期/时间类型可接受范围的数据
- 使用 hextoint 转换长于 4 个字节的字符串

无论是在隐式转换还是在显式转换中发生算术溢出和除零错误，都是很严重的问题。 使用 arithabort arith_overflow 选项可确定 Adaptive Server 处理这些错误的方法。缺省设置为 arithabort arith_overflow on，它将回退发生错误的整个事务。 如果设置 arithabort arith_overflow off，Adaptive Server 将中止导致错误的语句，但会继续处理事务或批处理中的其它语句。可以使用 *@@error* 全局变量来检查语句结果。

使用 arithignore arith_overflow 选项可确定 Adaptive Server 是否在发生这些错误后显示警告消息。缺省设置为 off，它将在发生除零错误或精度损失时显示警告消息。如果设置 arithignore arith_overflow on，则会取消发生这些错误后的警告消息。可省略可选关键字 arith_overflow 而不会有任何影响。

标度错误

当显式转换导致标度损失时，将截断结果而不发出任何警告。 例如，如果将 float、numeric 或 decimal 类型显式地转换为 integer，Adaptive Server 就会认为您要使结果成为整数，从而会截断小数点右侧的所有数字。

当隐式转换为 numeric 或 decimal 类型时，标度损失将导致标度错误。使用 arithabort numeric_truncation 选项可确定这种错误的严重程度。缺省设置为 arithabort numeric_truncation on，它将中止导致错误的语句，但会继续处理事务或批处理中的其它语句。 如果设置了 arithabort numeric_truncation off，Adaptive Server 就会截断查询结果并继续进行处理。

域错误

当 convert() 函数的自变量超出其定义范围时，该函数就会产生域错误。 这种错误应极少发生。

二进制和整数类型之间的转换

binary 和 varbinary 类型存储十六进制式数据，这种数据由前缀“0x”以及随后的数字和字母字符串组成。不同的平台会以不同的方式来解释这种字符串。 例如，字符串 0x0000100 在认为字节 0 最重要的机器上代表 65,536 而在认为字节 0 不重要的机器上代表 256。

转换函数和隐式转换

二进制类型可以用 convert 函数显式地或隐式地转换为整数类型。数据的前缀“0x”字符串被删除，然后用零填充（如果对于新类型太短）或截断（如果对于新类型太长）。

convert 和隐式数据类型转换在不同的平台上会以不同的方式对二进制数据求值。 因此，在不同平台上可能会得到不同的结果。 使用 hextoint 函数可在任何平台上将十六进制字符串转换为整数，而使用 inttohex 函数可在任何平台上将整数转换为十六进制值。

十六进制函数

使用 hextoint 函数可在任何平台上进行数据转换，将十六进制数据转换为整数。hextoint 接受加以引号的有效十六进制字符串（带有或不带有“0x”前缀均可），也接受字符类型列或变量的名。

hextoint 返回十六进制字符串的等值整数。该函数在任何平台上都始终返回给定十六进制字符串的相同等值整数。

inttohex 函数

使用 inttohex 函数可进行与平台无关的整数到十六进制字符串的转换。inttohex 接受任何求值为整数的表达式。该函数对指定表达式在任何平台上始终返回同一等值十六进制数。

将图像列转换为二进制类型

可使用 convert 函数将 image 列转换为 binary 或 varbinary。这种转换受到 binary 数据类型最大长度（255 个字节）的限制。如果未指定长度，转换后的值将具有缺省的长度（30 个字符）。

将其它类型转换为 bit 类型

精确和近似 numeric 类型可以隐式地转换为 bit 类型。字符类型需要使用显式的 convert 函数进行转换。

所转换的表达式必须仅包括数字位、小数点、货币符号以及加号或减号。其它字符的存在会导致语法错误。

0 的等值 bit 为 0。任何其它数字的等值 bit 为 1。

更改日期显示格式

convert 的 *style* 参数提供了将 datetime 或 smalldatetime 数据转换为 char 或 varchar 的多种日期显示格式。 作为 *style* 参数提供的数值自变量确定数据显示的显示方式。 年可用 2 或 4 位显示。 添加 100 到 *style* 值可获得 4 位的年，包括世纪 (*yyyy*)。

表 10-14 显示了 *style* 的可能值及可用的多种日期格式。 当使用 *style* 和 smalldatetime 时，包括秒或毫秒的样式将在那些位置显示零。

表 *10-14*：用样式参数转换日期格式

不含世纪 (yy)	含世纪 (yyyy)	标准	输出
-	0 或 100	缺省值	*mon dd yyyy hh:mi*AM（或 PM）
1	101	美式英语	*mm/dd/yy*
2	2	SQL 标准	*yy.mm.dd*
3	103	英文 / 法文	*dd/mm/yy*
4	104	德文	*dd.mm.yy*
5	105		*dd-mm-yy*
6	106		*dd mon yy*
7	107		*mon dd, yy*
8	108		*hh:mm:ss*
-	9 或 109	缺省值 + 毫秒	*mon dd yyyy hh:mm:sss* AM（或 PM）
10	110	USA	*mm-dd-yy*
11	111	日文	*yy/mm/dd*
12	112	ISO	*yymmdd*

缺省值样式 0 或 100 和 9 或 109 总是返回世纪 (*yyyy*)。

这是一个使用 convert 的 *style* 参数的示例：

```
select convert(char(12), getdate(), 3)
```

将当前日期转换为样式 3*dd/mm/yy*。

安全性函数

安全性函数返回有关安全服务和用户定义角色的信息。表 10-15 列出每个安全函数的名称、使用的自变量和返回结果。

表 *10-15*：安全性函数

函数	自变量	结果
is_sec_service_on	(*security_service_nm*)	确定是否启用了某个安全服务。 如果已启用服务则返回 1。 否则，返回 0。
mut_excl_roles	("*role_1*" , "*role_2*" [, "membership" |"activation"])	返回有关两个角色之间排它性等级的信息。
proc_role	("*role_name*")	如果正调用的用户没操作或激活指定的角色，返回 0；正调用的用户激活指定的角色，返回 1；用户直接或间接操作指定的角色但没激活角色，返回 2。
role_contain	(["*role1* ", "*role2* "])	如果第一个指定角色由第二个角色包含，返回 1。

函数	自变量	结果
role_id	("*role_name*")	返回指定角色名称的角色 ID。
role_name	(*role_id*)	返回指定角色 ID 的角色名称。
show_role	( )	如果有（sa_role、sso_role、oper_role、replication_role 或 *role_name*），该函数就会返回当前登录的活动角色。 如果登录没有任何角色，将返回 NULL。
show_sec_services	( )	返回当前会话启用的可用安全服务列表

有关安全性函数和用户定义角色的详细信息，参见*系统管理指南*。

第 11 章

创建表的索引

利用**索引**可以指定列的值为基础，提供对表中数据的快速访问。一张表可以有多个索引。索引对访问表中数据的用户是透明的；Adaptive Server 可自动确定何时使用为表创建的索引。

本章讨论如下内容：

主题	页
索引如何工作	371
创建索引	373
使用聚簇或非聚簇索引	378
指定索引选项	380
删除索引	382
确定表上存在哪些索引	383
更新索引的统计信息	384

有关如何设计索引以提高性能的信息，参见*性能和调优指南*。

索引如何工作

索引可帮助 Adaptive Server 定位数据。通过指出表列数据在磁盘上的位置，索引可加速数据检索过程。例如，假设需要使用 stores 表中的存储标识号运行常用的查询。为防止 Adaptive Server 不得不搜索表中的每一行（如果 stores 表包含数百万行，这样做将会非常耗时），可创建一个名为 stor_id_ind 的索引，如下所示：

```
create index stor_id_ind
on stores (stor_id)
```

下次查询 stores 中的 stor_id 列时，stor_id_ind 索引自动生效。换句话说，索引对用户是透明的。SQL 不包含在查询中引用索引的语法。只能从表中创建或删除索引； Adaptive Server 将决定是否对提交给表的每个查询都使用索引。如果表中数据发生变化，Adaptive Server 可以更改表的索引以反映这些变化。同样，这些改变对用户来说是透明的，Adaptive Server 将自动处理这些任务。

Adaptive Server 支持下列类型的索引：

- **组合索引** — 这些索引涉及多个列。在有两个或多个列因逻辑关系而适于作为一个整体来搜索时，应使用此类型的索引。
- **唯一索引** — 这类索引不允许在指定的列中任意两行有相同的值。Adaptive Server 在索引建立（如果数据已经存在）以及每次加入数据时检查重复值。
- **聚簇索引**或**非聚簇索引** — 聚簇索引强制 Adaptive Server 不断地对表中的行进行排序和重排序，以使其物理顺序与逻辑（或索引）顺序保持一致。每张表只能有一个聚簇索引。非聚簇索引不要求行的物理顺序与其索引顺序保持一致。在访问数据时，每个非聚簇索引都能提供不同的排序方式。

本章后面部分将详细说明这些类型的索引。

比较创建索引的两种方式

可使用 create index 语句（在本章中说明）创建索引，也可使用 create table 命令的 unique 或 primary key 完整性约束来创建。但是，在下列方式中完整性约束受到限制：

- 不能创建非唯一索引。
- 不能使用 create index 命令提供的选项来定制索引的工作方式。
- 只能使用 alter table 语句将这些索引作为一个约束删除。

如果应用程序需要这些功能，应使用 create index 创建这些索引。另外，unique 或 primary key 完整性约束提供了相对简单的方法来定义表的索引。有关 unique 和 primary key 约束的信息，参见第 7 章“创建数据库和表”。

使用索引指南

索引可加速数据的检索。向列添加索引通常会对查询是快速响应还是长时间等待有重要影响。

但是，建立索引要花费时间并占用存储空间。例如，当聚簇索引重建时，非聚簇索引也将自动重新创建。

此外，插入、删除或更新索引列中的数据比未索引列中的数据所花费的时间要长。但是，索引在检索性能方面的提高程度通常会超过这种开销。

何时使用索引

使用下列常规原则：

- 如果打算在 IDENTITY 列中手工插入值，应创建一个唯一索引以保证不会插入一个已有的值。
- 对于经常按排序顺序访问的列（即由 order by 子句指定的列），可能需要对其建立索引，以便 Adaptive Server 充分利用索引顺序。
- 对经常用于连接的列应始终进行索引，因为如果列是按排序顺序存储的，系统就能更快地执行连接。
- 存储表的主键的列通常进行聚簇索引，尤其是当它经常与其它表中的列连接时更应如此。注意，每个表只能有一个聚簇索引。
- 对经常按范围搜索的列来说最好也进行聚簇索引。在搜索范围内找到了具有第一个值的行后，即可确保具有后面值的列是紧接着存储的。聚簇索引对查找单个值来说并没有太大优势。

何时不应使用索引

在某些情况下，索引不会发挥作用：

- 对在查询中很少或从不被引用的列建立索引没有什么益处，因为系统很少根据这些列的值进行搜索。
- 只有两三个值的列，如男性与女性、是与否，对这些列建立索引毫无用处。

如果系统必须在未建索引的列上进行搜索，则搜索将逐行进行。这种扫描方式所花时间将与表中的行数成正比。

创建索引

create index 命令最简单的形式是：

create index *index_name*
on *table_name* (*column_name*)

要创建关于 authors 表中 au_id 列的索引，其命令为：

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

索引名称必须符合标识符的规则。列和表名称将指定要创建索引的列和包含该列的表。

对于 bit、text 或 image 类型的列，不能创建索引。

只有表的所有者才能 create 或 drop 索引。表的所有者可以随时 create 或 drop 索引，无论表中是否有数据。通过限定表名，可以在其它数据库的表中创建索引。

create index 的语法

create index 命令的完整语法是：

```
create [unique] [clustered | nonclustered]
        index index_name
    on [[database.]owner.]table_name (column_name
        [, column_name]...)
    [with {{fillfactor | max_rows_per_page} = x,
        ignore_dup_key, sorted_data,
        [ignore_dup_row | allow_dup_row]}]
    [on segment_name] [with consumers = x]
```

以下部分解释此命令的各选项。

注意：create index 的扩展 on *segment _name* 允许在段上放置索引，该段指向特定的数据库设备或数据库设备的集合。在段上创建索引之前，要得到有关可能用到的段列表信息，可与系统管理员或数据库所有者联系。由于性能原因或出于其它考虑，可能已将某些段分配给了特定的表或索引。

索引多个列：组合索引

要在所有指定列中创建关于组合值的组合索引，必须指定两个或更多列名。

当两列或多列适合作为一个单元搜索时，可使用组合索引，例如，authors 表中作者的名和姓。在表名后的小括号内，按排序优先级顺序列出组合索引要包括的列，如下所示：

```
create index auth_name_ind
on authors(au_fname, au_lname)
```

组合索引中列的顺序不必与 create table 语句中列的顺序相同。例如，在上述索引创建语句中，au_lname 和 au_fname 的顺序可能相反。

在 Adaptive Server 11.9.2 或更新版本中，在单个组合索引中最多可指定 31 列。组合索引中的所有列必须来自同一个表。组合索引值允许的最大长度为 600 个字节。即组成组合索引的各列长度之和不能超过 600 个字节。

使用 *unique* 选项

唯一索引不允许两列有相同的索引值（包括 NULL）。在创建索引时，如果数据已经存在，系统将检查重复值，以后每次使用 insert 或 update 增加或修改数据时也会进行检查。

只有在数据本身具有唯一性时指定唯一索引才有意义。例如，不应根据 last_name 列建立唯一索引，因为即使在只有几百行的表中也会有多个 "Smith" 或 "Wong" 存在。

而在存储社会保险号的列上建立唯一索引却非常合适。数据具有唯一性 — 每人都有一个不同的社会保险号。另外，唯一索引还作为完整性检查的标准。例如，重复的社会保险号意味着数据输入时出错或政府部门出错。

如果试图在包含重复值的数据上创建唯一索引，该命令将被中止，Adaptive Server 显示出错信息并给出第一个重复的值。不能在多个行中都包含空值的列上创建唯一索引，因为在进行索引时它们被作为重复值处理。

如果试图更改包含唯一索引的数据，其结果取决于是否使用了 ignore_dup_key 选项。有关详细信息，参见第 380 页上的“使用 ignore_dup_key 选项”。

建立组合索引可使用 unique 关键字。

非唯一索引中包含 IDENTITY 列

identity in nonunique index 数据库选项自动在表的索引键中包含一个 IDENTITY 列，以使在表上创建的所有索引都唯一。此选项使逻辑上非唯一的索引在内部唯一，并使其可用于处理可更新游标和隔离级别 0 的读取。

要启用 identity in nonunique indexes，输入：

```
sp_dboption pubs2, "identity in nonunique index",
true
```

表必须已经具有 IDENTITY 列，identity in nonunique index 选项才能工作 , 这可以利用 create table 语句或在创建表之前通过将 auto identity 数据库选项设置为 true 来实现。

如果计划在具有非唯一索引的表上使用游标和隔离级别 0 读取，可使用 identity in nonunique index。唯一索引可确保下一次在游标处执行 fetch 时，游标定位于正确的行。

例如，将 identity in nonunique index 和 auto identity 设置为 true 之后，假设创建下列没有索引的表：

```
create table title_prices
(title varchar(80)  not null,
price  money       null)
```

sp_help 显示表包含一个 IDENTITY 列，即 SYB_IDENTITY_COL，它由 auto identity 数据库选项自动创建。如果创建一个关于 title 列的索引，可使用 sp_helpindex 验证该索引是否自动包括 IDENTITY 列。

升序和降序索排列引列值

可使用 asc（升序）和 desc（降序）关键字对索引的每一列指派一个排序顺序。缺省情况下，排序顺序是升序。

按查询的 order by 子句中指定的顺序创建索引，将消除查询过程中对列的排序过程。下列示例将创建一个关于 Orders 表的索引。该索引有两列，第一列是 customer_ID（按升序排列），第二列是 date（按降序排列），以使最新定单排在最前面：

```
create index nonclustered cust_order_date
    on Orders
    (customer_ID asc,
    date desc)
```

使用 *fillfactor*、*max_rows_per_page* 和 *reservepagegap*

fillfactor、max_rows_per_page 和 reservepagegap 是空间管理属性，它们应用于表和索引并影响向物理页填充数据的方式。有关为索引设置空间管理属性的详细讨论，参见*参考手册*中的 create index。表 11-1 总结了有关索引的空间管理属性的信息。

表 *11-1*：索引的空间管理属性摘要

属性	说明	使用	注释
fillfactor	指定创建索引时某一页可以填充的空间百分比。如果 fillfactor 低于 100%，将为插入留出空间，而不立即进行页拆分。 优点在于： • 开始时页拆分较少。 • 减少对页的争用，因为页很多但页上的行较少。	仅适用于 DOL 锁表上的聚簇索引	仅当对现有数据的表创建索引时，才使用 fillfactor 百分比。它不适用于页以及表创建后的插入操作。 如果未指定 fillfactor，则使用系统范围的缺省 fillfactor 值。最初，它设置为 100%，但可使用 sp_configure 对其进行更改。
max_rows_per_page	指定每页允许的最大行数。 优点在于： • 通过限制每页的行数并增加页数可减少对页的争用。	仅适用于 allpages 锁表。 此属性可以设置的最大值为 256。	从创建索引开始，max_rows_per_page 始终适用。如果未指定，缺省值是页上将要容纳的最大行数。
reservepagegap	确定分配页片时空闲页的数目。例如，reservepagegap 值为 16 是指分配页片时，在 2 个页片中共有 16 页，其中 1 页留作空页。 优点在于： • 可减少行的前移并降低维护活动的频率，如运行 reorg rebuild 和重建索引。	适用于所有锁定方案中的页。	如果未指定 reservepagegap，则分配页片时不留空页。

下列语句将索引的 fillfactor 设置为 65% 并将每一分配页片的 reservepagegap 设置为一个空页：

```
create index postalcode_ind2
    on authors (postalcode)
    with fillfactor = 10, reservepagegap = 8
```

使用聚簇或非聚簇索引

利用聚簇索引，Adaptive Server 可根据具体情况对行进行排序，从而使其物理顺序与逻辑（索引）顺序相同。聚簇索引的底级或**叶级**包含表的实际数据页数。应在创建任何非聚簇索引之前创建聚簇索引，因为在聚簇索引创建之后非聚簇索引会自动重建。

每个表只能有一个聚簇索引。它常根据**主键**（唯一标识行的一列或多列）来创建。

从逻辑上讲，数据库的设计将确定一个主键。利用 create table 或 alter table 语句可指定 primary key 约束，从而创建索引并将主键属性附加到表列上。使用 sp_helpconstraint 可显示有关约束信息。

同样，使用 sp_primarykey、sp_foreignkey 和 sp_commonkey 可显式定义主键、外键和公用键（经常被连接的键对）。但是，这些过程并不强制键关系。

使用 sp_helpkey 可显示有关定义的键的信息，使用 sp_helpjoins 可显示有关适于连接的列的信息。

有关主键和外键的定义，参见第 16 章“触发器：强制实施参照完整性”有关系统过程的完整信息，参见*参考手册*。

对于非聚簇索引，行的物理顺序与其索引顺序不同。非聚簇索引的页级包含指向数据页上的行的指针。更精确地说，每个叶页包含一个索引值以及指向该值所在行的指针。换句话说，非聚簇索引在索引结构与数据本身之间有一个额外的级别。

表中最多有 249 种非聚簇索引，每个都能提供对数据的不同访问顺序。

通过聚簇索引查找数据几乎总是比通过非聚簇索引查找数据要快。另外，聚簇索引更适于检索多个具有连续键值的行的情况 — 即需要经常在列上检索值的范围。找到第一个**键值**所在的行之后，后继的索引值所在的行在物理上将一定相互紧挨，因而不需要进一步查找。

如果不使用关键字 clustered 和 nonclustered，Adaptive Server 将创建非聚簇索引。

以下是 titles 表中 title_id 列的 titleidind 索引的创建过程。如果希望尝试此命令，必须先删除索引：

```
drop index titles.titleidind
```

然后创建聚簇索引：

```
create clustered index titleidind
on titles(title_id)
```

如果需要经常对表 friends_etc（在第 7 章“创建数据库和表”中创建）中的人员按邮政编码排序，可创建一个关于 postalcode 列的非聚簇索引：

```
create nonclustered index postalcodeind
on friends_etc(postalcode)
```

此处唯一索引毫无意义，因为某些联系人很可能具有相同的邮政编码。聚簇索引也不适用，因为邮政编码不是主键。

friends_etc 中的聚簇索引应是一个关于人员的名和姓列的组合索引，例如：

```
create clustered index nmind
on friends_etc(pname, sname)
```

在段上创建聚簇索引

create index 命令允许在指定段创建索引。因为聚簇索引的叶级和数据页在定义上相同，创建 clustered 索引并使用 on segment_name 扩展可将表从创建它的设备移动到已命名段。

在段上创建表或索引之前，请与系统管理员或数据库所有者联系；出于性能方面的考虑，某些段可能被保留。

在已分区表上创建聚簇索引

如果以下所有条件均为真，则可在已分区表上创建聚簇索引。

- 服务器被配置为并行运行，
- select into/bulkcopy/pllsort 数据库选项设置为 true，并且
- 可用的工作线程数与分区数同样多。

但是，为了加速恢复过程，在创建了聚簇索引之后，要转储数据库。

有关已分区表及关于它们创建聚簇索引的详细信息，参见*系统管理指南*。在已分区表上创建聚簇索引之前，可与系统管理员或数据库所有者联系。

指定索引选项

当使用 insert 或 update 创建重复键或重复行时，索引选项 ignore_dup_key、ignore_dup_row 和 allow_dup_row 控制着所发生的情况。表 11-2 显示根据索引类型要使用的选项。

表 *11-2*：索引选项

索引类型	选项
聚簇	ignore_dup_row | allow_dup_row
唯一聚簇索引	ignore_dup_key
非聚簇索引	无
唯一非聚簇索引	ignore_dup_key
唯一非聚簇索引	ignore_dup_row

使用 *ignore_dup_key* 选项

如果试图向具有唯一索引的列中插入一个重复值，则该命令被取消。通过对唯一索引包括 ignore_dup_key 选项，可避免此情况出现。

唯一索引可以是聚簇的或非聚簇的。在开始数据输入时，任何试图插入重复键的操作都被取消，并显示错误信息。取消后，所有活动的事务都将继续执行，就象 update 或 insert 从未发生过一样。非重复键可以正常插入。

无论 ignore_dup_key 设置与否，都不能在已包含重复值的列上创建唯一索引。如果试图这样做，Adaptive Server 会显示出错信息及重复值列表。在列上创建唯一索引之前，必须消除重复值。

以下示例使用 ignore_dup_key 选项：

```
create unique clustered index phone_ind
on friends_etc(phone)
with ignore_dup_key
```

使用 *ignore_dup_row* 和 *allow_dup_row* 选项

ignore_dup_row 和 allow_dup_row 是用于创建非唯一聚簇索引的选项。创建非唯一的非聚簇索引时，这些选项是不相关的。因为 Adaptive Server 非聚簇索引在内部附加了一个唯一的行标识号，所以不会出现重复行 — 也不会出现相同的数据值。

ignore_dup_row 和 allow_dup_row 相互排斥。

非唯一聚簇索引允许重复键，但不允许重复行，除非指定 allow_dup_row。

设置 allow_dup_row 后，可在包含重复行的表上创建一个新的非唯一聚簇索引，随后即可 insert 或 update 重复行。

如果表中存在具有唯一性的索引，则唯一性的要求（最严格的要求）优先于 allow_dup_row 选项。因而，allow_dup_row 只适用于具有非唯一索引的表。如果表中任一列存在唯一聚簇索引，则不能使用此选项。

ignore_dup_row 选项将从批处理数据中消除重复。当输入重复行时，Adaptive Server 将忽略该行并取消特定的 insert 或 update，同时给出错误消息。取消后，当前活动的任何事务将继续执行，就象插入或更新从未发生过一样。非重复行将被正常插入。

ignore_dup_row 只适用于具有非唯一索引的表。如果表中任何一列存在唯一索引，则不能使用此关键字。

表 11-3 说明 allow_dup_row 和 ignore_dup_row 如何影响创建表（包括有重复行）中非唯一聚簇索引及向表中输入重复行的尝试。

表 *11-3*：索引中的重复行选项

选项	有重复	输入重复
无选项设置	create index 命令失败。	命令失败。
allow_dup_row 设置	命令完成。	命令完成。
ignore_dup_row 设置	创建索引，但删除重复行；错误消息。	不插入 / 更新重复；错误消息；事务完成。

使用 *sorted_data* 选项

如果表中的数据已被排序，则 create index 的 sorted_data 选项将加速索引的创建，例如，当使用 bcp 命令将已排序的数据复制到一个空表时。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，将显示错误消息，该命令被中止。

此选项只提高聚簇索引或唯一非聚簇索引的速度。在创建非唯一非聚簇索引时，除非没有任何一行有重复键才会成功。如果有若干行的重复键，系统会显示出错信息，命令被中止。

即使指定了 sorted_data，某些其它 create index 选项也要求排序。参见*参考手册*中有关 create index 的描述。

使用 *on segment_name* 选项

on segment_name 子句指定将要创建索引的数据库的段名称。非聚簇索引可在不同的段上创建，但数据页却不可以。例如：

```
create index titleind
on titles(title)
on seg1
```

创建聚簇索引时，如果使用 segment_name，则包含该索引的表将移动到用户指定的段。在段上创建表或索引之前，请与系统管理员或数据库所有者联系；出于性能方面的考虑，某些段可能被保留。

删除索引

drop index 命令用来删除数据库中的索引。此命令的语法为：

drop index *table_name.index_name*
[, *table_name.index_name*]...

使用此命令时，Adaptive Server 将从数据库中删除指定的索引并回收其存储空间。

只有索引的所有者可删除它。drop index 权限不能转移给其他用户。drop index 命令不能在 master 数据库或用户数据库的任何系统表上使用。

当多数或全部查询都用不到索引时，可能希望删除它。

要删除 friends_etc 表中的 phone_ind 索引，其命令为：

```
drop index friends_etc.phone_ind
```

确定表上存在哪些索引

要查看表上存在的索引，可使用 sp_helpindex。以下是关于 friends_etc 表的报告：

```
sp_helpindex friends_etc
index_name     index_description                index_keys
                                                 index_max_rows_per_page
------------- ----------------------------    -------------
                                                 ------------------------
nmind       clustered located on default    pname, sname
                                                0
postalcodeind nonclustered located on default postalcode
                                               0

(2 rows affected, return status = 0)
```

sp_help 将在其报告的最后运行 sp_helpindex。

sp_statistics 返回表的索引列表。例如：

```
sp_statistics friends_etc
table_qualifier               table_owner
         table_name                       non_unique
         index_qualifier                  index_name
      type  seq_in_index column_name
collation
         cardinality pages
------------------------------- ------------------
--------------
       ------------------------------- ----------
        ------------------------------- ----------
---------------------
        ------ ------------ -----------------------
--------- ---------
        ----------- -----------
pubs2                           dbo
        friends_etc                            NULL
        NULL                             NULL
           0        NULL NULL
NULL
                   0           1
pubs2                             dbo
         friends_etc                              1
         friends_etc                      nmind
           1          1 pname                      A
                   0           1
pubs2                               dbo
```

```
                                    friends_etc                                1
                                    friends_etc                       nmind
                                       1          2 sname                     A
                                               0            1
                              pubs2                              dbo
                                    friends_etc                                1
                                   friends_etc                    postalcodeind
                                     3         1  postalcode                   A
                                          NULL       NULL

                              (4 rows affected, return status = 0)
```

此外，如果在表名后接着输入“1”，则 sp_spaceused 将报告表及其索引所用的空间数量。例如：

```
sp_spaceused friends_etc, 1
index_name            size       reserved   unused
-------------------- ---------- ---------- ----------
nmind                 2 KB       32 KB      28 KB
postalcodeind         2 KB       16 KB      14 KB

name         rowtotal   reserved     data    index_size      unused
------------ ---------- ------------ ------- --------------- --------
friends_etc  1          48 KB        2 KB    4 KB            42 KB

(return status = 0)
```

更新索引的统计信息

在 Adaptive Server 处理查询时，update statistics 命令通过保持索引中键值分布为最新，可帮助 Adaptive Server 作出关于使用何种索引的最优决策。在索引列中添加、更改或删除大量数据后，可使用 update statistics。

启用了组件集成服务后，update statistics 可生成远程表的准确分布统计信息。有关详细信息，参见*组件集成服务用户指南*。

缺省情况下，表的所有者具有 update statistics 权限，并且不能移交。此命令的语法为：

update statistics *table_name* [*index_name*]

如果不指定索引名，则该命令更新指定表中所有索引的分布统计信息。如果给定索引名，将仅对该索引更新统计信息。

通过使用 sp_helpindex 可查找索引名。以下是关于如何列出 authors 表的索引的示例：

```
sp_helpindex authors
index_name index_description  index_keys  index_max_rows_per_page
---------- -----------------  ----------  -----------------------
auidind    clustered, unique  au_id       0
aunmind    nonclustered       au_lname,   0
                              au_fname

(2 rows affected, return status = 0)
```

要更新所有索引的统计信息，键入：

```
update statistics authors
```

要仅更新 au_id 列上的索引的统计信息，键入：

```
update statistics authors auidind
```

因为 Transact-SQL 不要求索引名在数据库中唯一，所以必须给出与索引相关联的表名。对现有数据创建索引时，Adaptive Server 将自动运行 update statistics。

更新分区统计

类似于非分区的表的 update statistics 命令，在 Adaptive Server 处理查询时，update partition statistics 命令通过保持分区中页的数目信息为最新，可帮助 Adaptive Server 作出最优决策。在已分区表中添加、更改或删除大量数据后，可使用 update partition statistics。

缺省情况下，表的所有者具有 update partiton statistics 权限，并且不能移交。此命令的语法为：

update partition statistics *table_name*

例如，假设 authors 表按如下方式分区：

```
alter table authors partition 3
```

即可运行 sp_helpartition 查看分区的分配方式：

```
sp_helpartition authors
partitionid firstpage   controlpage ptn_data_pages
----------- ----------- ----------- --------------
          1         553         554              1
          2         817         816              1
          3        1009        1008              1
```

```
(3 rows affected, return status = 0)
```

然后，如下所示更新 authors 的统计信息：

```
update partition statistics authors
```

对 authors 使用 sp_helpartition 以显示下列更新：

```
partitionid firstpage   controlpage ptn_data_pages
----------- ----------- ----------- --------------
          1         553         554             10
          2         817         816              1
          3        1009        1008              1

(3 rows affected, return status = 0)
```

删除和重新创建聚簇索引可自动重新分配分区内的数据并更新分区统计信息。

第 12 章 为数据定义缺省值和规则

缺省值是当用户没有为列明确输入值时，Adaptive Server 插入该列的值。 在数据库管理中，**规则**指定在特定列或给定用户定义数据类型的任意列中允许或不允许输入什么。 可以使用缺省值和规则来维护数据库中的数据完整性。

本章讨论如下内容：

主题	页
缺省值和规则如何执行	387
创建缺省值	388
删除缺省值	393
创建规则	394
删除规则	398
获得有关缺省值和规则的信息	399

缺省值和规则如何执行

可以为表中的列或用户定义数据类型定义一个值，当用户无明确输入值是该值会自动插入。 例如，可以创建一个其值为“???”或“fill in later”（以后填充）的缺省值。 也可以为表中的该列或数据类型定义一个规则来限制用户可输入值的类型。

在关系数据库管理系统中，每个数据元素必须包含一些值，即使该值为空。 如在第 7 章“创建数据库和表”中所讨论，有些列不接受空值。 对于这些列，必须输入其它值，可以是用户明确输入的值，也可以是由 Adaptive Server 输入的缺省值。

规则强制实现数据完整性的方式不受列的数据类型的影响。 规则可与一个特定列或几个特定列或用户定义的数据类型相关联。

用户每次输入值时，Adaptive Server 根据绑定在特定列上的最新规则检查该值。不检查在规则创建并绑定以前输入的值。

作为缺省值和规则的替代，可以使用 create table 语句的 default 子句和 check 完整性约束来完成一些相同的任务。 然而，它们是针对该表的，并且不能绑定到其它表的列或用户定义的数据类型上。 关于完整性约束的详细信息，参见第 7 章“创建数据库和表”。

创建缺省值

在数据输入表之前或之后，可随时创建或删除缺省值。 通常，要创建缺省值：

1 使用 create default 定义缺省值。

2 用 sp_bindefault 将缺省值绑定到相应表中的列或用户定义的数据类型。

3 通过插入数据测试绑定缺省值。

可用 drop default 删除缺省值，并用 sp_unbinddefault 撤消它们之间的关联。

创建并绑定缺省值时：

- 确保列对于缺省值足够大。 例如，一个 char(2) 列不能持有 17 个字节的字符串，如“Nobody knows yet”。
- 在对用户定义数据类型设置缺省值，并对该类型的单个列设置不同的缺省值时应小心。 如果先绑定数据类型缺省值再绑定列缺省值，列缺省值将只在该命名列代替用户定义数据类型的缺省值。用户定义数据类型的缺省值被绑定到所有其它具有此数据类型的列上。

  但是，一旦把另一个缺省值绑定到因其类型而有了缺省值的列，则该列将不再受其数据类型所绑定的缺省值的影响。 该问题将在第 390 页上的“绑定缺省值”中详细讨论。
- 注意缺省值与规则不要发生冲突。 要确保缺省值是规则所允许的；否则，缺省值可能被规则消除。

  例如，如果规则允许的输入值在 1 到 100 之间，而缺省值设为 0，则规则拒绝该缺省输入值。 改变缺省值或者改变规则。

create default 语法

create default 的语法是：

create default [*owner.]default_name*
as constant_expression

缺省值名必须遵循关于标识符的规则。 只可在当前数据库中创建缺省值。

在一个数据库中，缺省值名对每一个用户必须是唯一的。 例如，不可以创建两个名为 phonedflt 的缺省值。 然而，作为“guest”，即使 dbo.phonedflt 已经存在，仍可创建 phonedflt，因为所有者名将每一个区分开了。

另一个示例： 假设，要创建一个为“Oakland”的缺省值，它可用于 friends_etc 的 city 列，也可能用于其它列或用户数据类型。 要创建该缺省值，输入：

```
create default citydflt
as "Oakland"
```

继续该例，您可使用与将输入到人员表中的人相匹配的任何城市名。

将字符和日期常量用引号引起来；货币、整型和浮点常量不需要这样。 二进制数据必须以“0x”开头，货币数据必须以美元符号 ($) 开头。 缺省值必须与列的数据类型兼容。 例如，不可以用“none”作为数值型列的缺省值，但是可以用 0。

如果在创建列时指定 NOT NULL 并且没有将一个缺省值与之相连，则无论何时有人未给该列输入值，Adaptive Server 都将给出出错信息。

通常，在创建表时输入缺省值。 然而，对于输入在一列或多列中有相同值的许多行的会话，在开始之前创建一个是适合于该会话的缺省值会带来很大方便。

注意：不能用声明缺省值执行 create table，然后在同一批处理或过程的表中插入数据。 可以将创建和插入语句分放在两个不同的批处理或过程中，或者使用 execute 分别执行操作。

绑定缺省值

创建缺省值后，用 sp_bindefault 将缺省值绑定到列或用户定义的数据类型上。 例如，如果创建以下缺省值：

```
create default advancedflt as "UNKNOWN"
```

现在，用 sp_bindefault 将缺省值绑定到适当的列或用户定义的数据类型。

```
sp_bindefault advancedflt, "titles.advance"
```

只有当用户不向 titles 表的 advance 列添加条目时，缺省值才有效。 不输入条目与输入空值是不同的。 一个缺省值可与一个特定列、许多列，或数据库中具有特定用户定义数据类型的所有列相关联。

注意：要获取缺省值，必须对列列表（不含具有缺省值的列）执行 insert 或 update 命令。

以下限制适用于缺省值：

- 缺省值仅适用于新行。 它并不追溯性地改变已存在的行。 仅当无条目输入时缺省值才生效。 如果给列赋予任何值，包括 NULL，缺省值都是无效的。
- 不能将缺省值绑定到系统数据类型。
- 不能绑定缺省值到 timestamp 列，因为 Adaptive Server 为 timestamp 列生成值。
- 不能将缺省值绑定到系统表。
- 可以将缺省值绑定到 IDENTITY 列或有 IDENTITY 属性的用户定义数据类型，但 Adaptive Server 将忽略这样的缺省值。 当没有为 IDENTITY 列指定值而将一行插入表中时，Adaptive Server 分配一个比上一次分配的值大 1 的值。
- 如果列上已存在缺省值，那么在绑定新的缺省值之前必须将其删除。使用 sp_unbindefault 可删除用 sp_bindefault 创建的缺省值。 使用 alter table 删除用 create table 创建的缺省值。

要将 citydflt 绑定到 friends_etc 中的 city 列，键入：

```
sp_bindefault citydflt, "friends_etc.city"
```

注意表名和列名在引号中，因为嵌入式的标点符号（逗号）。

如果为数据库中每张表的所有 city 列创建一个特殊的数据类型，并将 citydflt 绑定到该数据类型，则“Oakland”仅在 city 名适合时才出现。 例如，如果用户数据类型称为 citytype，则将 citydflt 绑定到它的方法为：

```
sp_bindefault citydflt, citytype
```

为防止现有列或特定用户数据类型继承新的缺省值，当将缺省值绑定到用户数据类型时使用 futureonly 参数。 然而，将缺省值绑定到列时不要用 futureonly。 以下是创建和绑定一个新缺省值“Berkeley”到数据类型 citytype 以仅供表的新列使用的方法：

```
create default newcitydflt as "Berkeley"
sp_bindefault newcitydflt, citytype, futureonly
```

对任何使用 citytype 的现有表列，“Oakland”继续作为缺省值出现。

如果表中的大多数人住在相同邮编地区内，可以创建一缺省值来保存数据输入时间。 此处即是一例，其绑定缺省值适用于 Oakland 的一个区域：

```
create default zipdflt as "94609"
sp_bindefault zipdflt, "friends_etc.postalcode"
```

以下是 sp_bindefault 的完整语法：

sp_bindefault *defname, objname* [, futureonly]

defname 是用 create default 创建的缺省值的名字。*objname* 是缺省值绑定的表、列或用户定义的数据类型名。 如果参数的形式不是 *table.column*，它将被认为是用户定义的数据类型。

指定的用户数据类型的所有列与指定缺省值相关联，除非使用可选参数 futureonly，该参数使此用户数据类型的现有列不能继承缺省值。

注意：同一批处理过程中不能将缺省值绑定到列上并使用它。sp_bindefault 不能与调用缺省值的 insert 语句的在同一批处理中。

解除绑定缺省值

解除绑定缺省值意味着断开其与特定列或用户定义数据类型的关联。 解除绑定的缺省值仍保存在数据库中且将来可以使用。 使用 sp_unbindefault 删除缺省值与列或数据类型之间的绑定。

以下是从 friends_etc 表的 city 列解除绑定当前缺省值的方法：

```
execute sp_unbindefault, "friends_etc.city"
```

要从用户定义的数据类型 citytype 上解除绑定缺省值，使用：

```
sp_unbindefault citytype
```

sp_unbindefault 的完整语法是：

sp_unbindefault *objname* [, futureonly]

如果给定的 *objname* 参数的形式不是 *table.column*，Adaptive Server 假设它为用户定义的数据类型。 当从用户定义数据类型中解除缺省值绑定时，该缺省值就从此类型的所有列中解除绑定，除非使用可选参数 futureonly，该操作使此类型的现有列不失去与缺省值的绑定。

缺省值怎样影响 NULL 值

如果在创建列时指定 NOT NULL 并且没有为其创建缺省值，则无论何时有人插入一行并且没有给该列输入值，Adaptive Server 都将给出出错信息。

删除了 NULL 列的缺省值后，每次增加一行而没有给该列输入值时，Adaptive Server 都将在该位置上插入 NULL。 删除了 NOT NULL 列的缺省值后，当增加一行而没有给该列输入值时，将会产生出错信息。

表 12-1 说明缺省值的存在和 NULL 或 NOT NULL 列定义的关系。

表 *12-1*：列定义和空缺省值

列定义	用户输入	结果
已定义空和缺省值	没有值 NULL 值	使用缺省值 使用 NULL
已定义 Null，未定义缺省值	没有值 NULL 值	使用 NULL 使用 NULL
非空，已定义缺省值	没有值 NULL 值	使用缺省值 使用 NULL
非空，未定义缺省值	没有值 NULL 值	错误 使用 NULL

创建缺省值后

创建缺省值后，描述缺省值的**源文本**存储在 syscomments 系统表的 text 列上。 *请勿删除此信息*；这样或许会使 Adaptive Server 的将来版本出现问题。 而应使用*参考手册*中说明的 sp_hidetext 对 syscomments 中的文本加密。有关详细信息，请参见第 4 页上的“编译对象”。

删除缺省值

要从整个数据库中删除缺省值，请使用 drop default 命令。 在删除缺省值前，解除其与所有行和用户数据类型的绑定。（参见第 392 页上的“解除绑定缺省值”。） 如果试图删除仍旧绑定的缺省值，Adaptive Server 会显示出错信息且 drop default 命令会失败。

以下是删除 citydflt 的方法。 首先，解除绑定：

```
sp_unbindefault citytype
```

接下来，可以删除 citydft：

```
drop default citydflt
```

drop default 的完整语法是：

drop default [*owner*.]*default_name*
  [, [*owner*.]*default_name*] ...

缺省值只能由其所有者删除。 有关解除绑定的详细信息，参见*参考手册*中的 sp_unbindefault 和 alter table。

创建规则

利用规则可以指定用户能输入及不能输入什么到特定列或有用户定义数据类型的任意列。 通常，要创建规则：

1 使用 create rule 创建规则。

2 使用 sp_bindrule 将规则绑定到一列或用户定义的数据类型。

3 通过插入数据测试绑定规则。 仅通过使用 insert 或 update 命令进行测试，便可发现许多在创建和绑定规则时产生的错误。

通过使用 sp_unbindrule 或将新规则绑定到列或数据类型，可解除规则到该列或数据类型的绑定。

create rule 的语法

create default 的语法是：

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

规则名必须遵循标识符的规则。 只可在当前数据库中创建规则。

在数据库中，规则名对每一个用户必须是唯一的。 例如，一个用户不可以创建两个名为 socsecrule 的规则。 然而，两个不同的用户都可以创建名为 socsecrule 的规则，因为所有者名将它们区分开了。

以下是创建允许五个不同 pub_id 号和一个伪值（99 后跟任意两个数字）的规则的方法：

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

as 子句包括规则自变量名、前缀“@”，和规则自身的定义。 自变量指受 update 或 insert 语句影响的列值。

在前面的例子中，自变量是 @pub_id，这是一个便于记忆的名字，因为此规则将绑定到 pub_id 列上。 可为自变量起*任何*名字，但第一个字符必须是“@”。 使用规则要绑定的列或数据类型的名称可帮助记忆其用途。

规则定义可包括任何在 where 子句中合法的表达式，并可包括算术运算符、比较运算符、like、in、between 等。 然而，规则定义不能直接引用任何列或其它数据库对象。*可以*包括不引用数据库对象的内部函数。

下例创建了一个强制输入值与特定的“picture”保持一致的规则。这种情况下，列中的每一个输入值都必须以“415”开头，后跟字符多于 7 个。

```
create rule phonerule
as @phone like "415_______"
```

要保证所输入的朋友年龄在 1 到 120 之间，但不可以是 17，请试：

```
create rule agerule
as @age between 1 and 120 and @age ! = 17
```

绑定规则

创建规则后，用 sp_bindrule 将规则链接到列或用户定义的数据类型。

以下是 sp_bindrule 的完整语法：

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

rulename 是用 create rule 创建的规则的名称。 *objname* 是要将规则绑定到的表和列或用户定义数据类型的名称。 如果参数的形式不是 *table.column*，它将被认为是用户数据类型。

仅当将规则绑定到用户定义的数据类型时，使用可选的 futureonly 参数。 指定的用户定义数据类型的所有列与指定规则相关联，除非指定 futureonly，此参数使该用户数据类型的现有列不继承规则。 如果与给定用户定义的数据类型相关联的规则已被改变，Adaptive Server 为该用户定义的数据类型的现有列维护此改变的规则。

以下限制适用于规则：

- 不能将规则绑定到 text、image，或 timestamp 数据类型列上。
- Adaptive Server 不允许规则在系统表上。

绑定到列的规则

使用带规则名、表名和列名（在引号中）的 sp_bindrule 将规则绑定到列。 以下是将 pub_idrule 绑定到 publishers.pub_id 的方法：

```
sp_bindrule pub_idrule, "publishers.pub_id"
```

以下是一个确保所有输入的邮编都以 946 作为头 3 个字符的规则：

```
create rule postalcoderule946
as @postalcode like "946[0-9][0-9]"
```

如下所示将它绑定到 friends_etc 中的 postalcode 列：

```
sp_bindrule postalcoderule946,
"friends_etc.postalcode"
```

同一批处理过程中不能将规则绑定到列上并使用它。sp_bindrule 不能与调用规则的 insert 语句的在同一批处理中。

绑定到用户定义数据类型的规则

不可以将规则绑定到系统数据类型上，但可以将其绑定到用户定义数据类型上。 要将 phonerule 绑定到称为 p# 的用户定义数据类型上，输入：

```
sp_bindrule phonerule, "p#"
```

规则的优先顺序

绑定到列的规则总是比绑定到用户数据类型的规则具有更高的优先级。 将规则绑定到列后，将替换绑定到该列的用户数据类型的规则。但是，将规则绑定到数据类型，则不会替换绑定到该用户数据类型的列的规则。

只有在试图向用户定义数据类型的数据库列插入值或更新它时，绑定到用户定义数据类型的规则才被激活。 由于规则不测试变量，因此不要将值赋予将被绑定到同一数据类型的规则拒绝的用户定义数据类型变量。

当将规则绑定到已有规则存在的列和用户定义数据类型时，其优先顺序如表 12-2 所示：

表 *12-2*：规则的优先顺序

新规则绑定到	旧规则绑定到	
	用户数据类型	*列*
用户数据类型	替换旧规则	没有变化
列	替换旧规则	替换旧规则

在某些列上输入需要特殊临时约束的数据时，可创建一个新规则来帮助检测这些数据。 例如，假设要向 friends_etc 表的 debt 列添加数据。 已知今天要记录的所有 debt 都在 $5 和 $200 之间。为避免偶然打出一个超出此限制的数值，可创建如下规则：

```
create rule debtrule
as @debt = $0.00 or @debt between $5.00 and $200.00
```

@debt 规则定义允许输入 $0.00 以保留原来为此列定义的缺省值。

如下所示将 debtrule 绑定到 debt 列：

```
sp_bindrule debtrule, "friends_etc.debt"
```

规则和 NULL 值

不能定义一个允许空的列，接着又用禁止空值的规则覆盖此定义。例如，如果列定义指定 NULL 而规则指定如下，则隐式或显式的 NULL 并不违背规则：

```
@val in (1,2,3)
```

列定义覆盖规则，即使规则指定：

```
@val is not null
```

定义规则后

定义规则后，描述规则的**源文本**存储在 syscomments 系统表的 text 列中。 *请勿删除此信息*；这样或许会使 Adaptive Server 的将来版本出现问题。 而应使用*参考手册*中说明的 sp_hidetext 对 syscomments 中的文本加密。有关其详细信息，请参见第 4 页上的“编译对象”。

解除绑定规则

解除绑定规则即断开其与特定列或用户定义数据类型的关联。 解除绑定的规则定义仍保存在数据库中且将来可以使用。

有两种解除绑定规则的方法：

- 使用 sp_unbindrule 解除规则和列或用户定义数据类型之间的绑定。
- 使用 sp_bindrule 将新规则绑定到该列或数据类型上。 旧规则将自动解除绑定。

以下是从 friends_etc.debt 上解除 debtrule（或任何其它当前绑定规则）的方法：

```
sp_unbindrule "friends_etc.debt"
```

规则仍在数据库中，但与 friends_etc.debt 无关联。

要从用户定义的数据类型 p# 上解除绑定规则，使用：

```
sp_unbindrule "p#"
```

sp_unbindrule 的完整语法是：

sp_unbindrule *objname* [, futureonly]

如果使用的 *objname* 参数的形式不是“*table.column*”，则 Adaptive Server 假设它为用户定义的数据类型。 当从用户定义的数据类型解除规则时，该规则从此类型的所有列上解除绑定，除非：

- 给出可选的 futureonly 参数，它使该数据类型的现有列不会失去与规则的绑定，或
- 该用户定义数据类型的列上的规则已被改变，因此其当前值与绑定的规则不同。

删除规则

要从整个数据库中删除规则，请使用 drop rule 命令。 在删除规则前，解除其与所有行和用户数据类型的绑定。 如果试图删除仍旧绑定的规则，Adaptive Server 会显示出错信息且 drop rule 会失败。 当然，要绑定新规则，不必先解除绑定接着再删除规则。 只需在其位置上绑定一个新规则。

要在绑定 phonerule 后删除它：

```
drop rule phonerule
```

drop rule 的完整语法是：

drop rule [*owner*.]*rule_name*
[, [*owner*.]*rule_name*] ...

删除规则后，新数据输入先前由该规则控制的列时，将没有这些约束。不管怎样，已存在的数据都不会受到影响。

规则只能由其所有者删除。

获得有关缺省值和规则的信息

系统过程 sp_help 与表名一起使用时，会显示绑定到列的规则和缺省值。 以下示例显示有关 pubs2 数据库中 authors 表的信息，包括规则和缺省值：

```
sp_help authors
```

sp_help 也报告绑定到用户定义数据类型的规则。 要检测规则是否绑定到用户定义数据类型 p#，使用：

```
sp_help "p#"
```

sp_helptext 报告规则或缺省值的定义（create 语句）。

如果缺省值或规则的源文本已使用 sp_hidetext 加密，则 Adaptive Server 显示一条提示用户文本被隐藏的消息。 有关隐藏源文本的详细信息，参见*参考手册*中的 sp_hidetext。

如果系统安全员已经用 sp_configure 重设 allow select on syscomments.text column 参数（这是在已评估的配置中运行 Adaptive Server 所必需的），那么您必须是缺省值或规则的创建者或系统管理员，才能通过 sp_helptext 来查看缺省值或规则的文本。 有关详细信息，参见 *Adaptive Server 词汇表*中的**已评估的配置**。

第 13 章

使用批处理和控制流语言

Transact-SQL 允许交互地或在操作系统文件中将一系列语句组合为批处理。也可以使用 Transact-SQL 提供的控制流结构来连接与编程类似的语句。

变量是一个被赋值的实体。在使用此变量的批处理或存储过程的执行期间，此值可以变化。Adaptive Server 有两种类型的变量：**局部的**和**全局的**。局部变量是用户定义的，而全局变量是由系统提供并预定义的。

本章讨论如下内容：

主题	页
简介	401
批处理相关的规则	402
使用控制流语言	407
局部变量	429
全局变量	434

简介

为了说明这点，*Transact-SQL 用户指南*中的每个示例都包括单个语句。每次只向 Adaptive Server 提交一个语句，交互地输入语句并接收结果。

Adaptive Server 也可以交互地或通过文件将多个语句作为批处理提交。批处理或批处理文件是被一同提交的 Transact-SQL 语句，并作为一组依次执行。批处理用批结束符号来终止。对于 isql 实用程序，批结束符号是独占一行的“go”字。有关 isql 的详细信息，参见所用平台的实用程序手册。

以下是一个包含两个 Transact-SQL 语句的批处理示例：

```
select count(*) from titles
select count(*) from authors
go
```

从技术角度上说，单个 Transact-SQL 语句可以构成一个批处理，但是通常认为批处理包含多个语句。通常将语句的批处理编写为操作系统文件，然后才提交给 isql。

Transact-SQL 提供称为控制流语言的特殊关键字，允许用户控制语句执行流。控制流语言可用于单个语句、批处理、存储过程或触发器。

不使用控制流语言，独立的 Transact-SQL 语句按其出现的顺序依序执行。在第 5 章“子查询：在其它查询中使用查询”中论述的相关子查询为例外。控制流语言使用类似于程序的结构使语句相互连接和关联。

控制流语言，例如用于命令的条件执行的 if...else 和用于重复执行的 while，可以精确地控制 Transact-SQL 语句的操作。Transact-SQL 控制流语言将标准 SQL 转换为更高级的编程语言。

批处理相关的规则

有一些规则约束哪些 Transact-SQL 语句可组合到单个批处理中。这些批处理规则如下：

- 引用数据库对象之前，对该数据库发出 use 语句。例如：

```
use master
go
select count(*)
from sysdatabases
go
```

- *不能*将以下数据库命令与其它语句组合在批处理中：
  - create procedure
  - create rule
  - create default
  - create trigger

- *可以*将以下数据库命令与其它 Transact-SQL 语句组合到批处理中：
  - create database（除非在单个批处理中不能创建数据库并创建或访问此新数据库中的对象）
  - create table
  - create index
  - create view
- 不能在同一批处理中将规则和缺省值绑定到列中，然后使用它们。sp_bindrule 和 sp_bindefault 不能与调用该规则或缺省值的 insert 语句在同一个批处理中。
- 不能在同一批处理中 drop 对象，然后又引用或重新创建它。
- 如果表已经存在，不能在批处理中重新创建它，即使批处理中包括对表存在性的测试。

  Adaptive Server 先编译批处理，然后执行它。编译期间，Adaptive Server 不对批处理所引用的对象（如表和视图）进行权限检查。Adaptive Server 只在执行批处理时才检查权限。当 Adaptive Server 访问的不是当前数据库时情况例外。这种情况下，Adaptive Server 在编译时显示一条错误消息，不执行批处理中任何语句。

  假定批处理中包含以下这些语句：

  ```
  select * from taba
  select * from tabb
  select * from tabc
  select * from tabd
  ```

  如果具有除了第三条语句 (select * from tabc) 外所有语句的必要的权限，则 Adaptive Server 返回第三条语句的错误消息和其它语句的结果。

使用批处理的示例

本节示例说明使用 isql 实用程序格式的批处理，它有明显的批结束符号 — 独占一行的“go”字。以下是一个包含两个 select 语句的批处理：

```
select count(*) from titles
select count(*) from authors
go
-------------
          18

(1 row affected)
-------------
          23

(1 row affected)
```

可以在同一批处理中创建表然后引用它。以下批处理创建表，向其插入一行并从中选择所有列：

```
create table test
  (column1 char(10), column2 int)
insert test
  values ("hello", 598)
select * from test
go

(1 row affected)
column1  column2
-------  -------
hello        598

(1 row affected)
```

只要随后语句所引用的对象仍在开始时打开的数据库中，就可以将 use 语句与其它语句组合使用。以下批处理从 master 数据库中选择一个表，然后打开 pubs2 数据库。批处理从将 master 标记为当前数据库开始；然后，pubs2 成为当前数据库。

```
use master
go
select count(*) from sysdatabases
use pubs2
go
-------------
            6
```

```
(1 row affected)
```

只要在同一批处理中不引用或重新创建删除的对象，就可以将 drop 语句与其它语句组合使用。以下示例将 drop 语句与 select 语句组合使用：

```
drop table test
select count(*) from titles
go
------------
         18

(1 row affected)
```

只要批处理中有一个语法错误，将不执行其中任何语句。例如，以下批处理在最后的语句中有键入错误，其结果为：

```
select count(*) from titles
select count(*) from authors
slect count(*) from publishers
go
Msg 156, Level 15, State 1:
Line 3:
Incorrect syntax near the keyword 'count'
```

和批处理规则冲突的批处理也生成错误消息。以下是一些非法批处理的示例：

```
create table test
    (column1 char(10), column2 int)
insert test
    values ("hello", 598)
select * from test
create procedure testproc as
    select column1 from test
go
Msg 111, Level 15, State 7:
Line 6:
CREATE PROCEDURE must be the first command in a
query batch.

create default phonedflt as "UNKNOWN"
sp_bindefault phonedflt, "authors.phone"
go



Msg 102, Level 15, State 1:
Procedure 'phonedflt'Line 2:
```

```
Incorrect syntax near 'sp_bindefault'
```

如果已经在 use 语句所指定的数据库中，则将执行下一个批处理。但是，如果试图从其它数据库（例如 master）中执行上述批处理，则将给出一条错误消息。

```
use pubs2
select * from titles
go
Msg 208, Level 16, State 1:
Server 'hq', Line 2:
titles not found. Specify owner.objectname or use
sp_help to check whether the object exists (sp_help
may produce lots of output)

drop table test
create table test
(column1 char(10), column2 int)
go



Msg 2714, Level 16, State 1:
Server 'hq', Line 2:
There is already an object named 'test'in the
database.
```

作为文件提交的批处理

可以通过操作系统文件向 isql 提交一个或多个 Transact-SQL 语句的批处理。文件可以包括多个批处理，也即，多个语句集合，每个语句集合以“go”结束。

例如，操作系统文件可能包含以下三个批处理：

```
use pubs2
go
select count(*) from titles
select count(*) from authors
go
create table hello
   (column1 char(10), column2 int)
insert hello
   values ("hello", 598)
select * from hello
go
```

以下是将此文件提交给 isql 实用程序的结果：

```
-------------
           18

(1 row affected)
-------------
           23

(1 row affected)
column1      column2
---------    ---------
hello              598

(1 row affected)
```

有关在特定环境下运行存储在文件中的批处理的信息，参见*实用程序指南*中的 isql。

使用控制流语言

在批处理和存储过程中使用控制流语言的交互式语句。表 13-1 列出了控制流和相关关键字以及它们的功能。

表 *13-1*：控制流和相关关键字

关键字	功能
if	定义条件执行。
…else	定义当 if 条件为假时的替代执行。
case	使用 when…then 语句代替 if…else 定义条件表达式。
begin	语句块的起点。
…end	语句块的终点。
while	条件为真时重复执行语句。
break	从下一个最外层的 while 循环结束处退出。
…continue	重新启动 while 循环。
declare	声明局部变量。
goto label	跳转到语句块中的 label: 位置。
return	无条件退出。
waitfor	设置命令执行的延迟。
print	在用户屏幕上输出用户定义消息或局部变量。

关键字	功能
raiserror	在用户屏幕上输出用户定义消息或局部变量，并在全局变量 *@@error* 中设置系统标记。
/* *comment* */ 或 --*comment*	在 Transact-SQL 语句中任意位置插入注释。

if...else

关键字 if（无论有没有协同的 else）引入确定是否执行下一语句的条件。如果条件满足，也就是说条件返回 TRUE，则执行 Transact-SQL 语句。

else 关键字引入在 if 条件返回 FALSE 时，替代执行的 Transact-SQL 语句。

if 和 else 的语法为：

```
if
        boolean_expression
        statement
    [else
        [if boolean_expression]
        statement ]
```

布尔表达式返回 TRUE 或 FALSE。它可以包括列名、常量、使用算术运算符或逐位运算符连接的列名和常量的任意组合、或子查询（只要子查询返回单个值）。如果布尔表达式包含 select 语句，则 select 语句必须用小括号括起来，并且必须返回单个值。

以下示例只使用了 if：

```
if exists (select postalcode from authors
          where postalcode = "94705")
print "Berkeley author"
```

如果 authors 表中的一个或多个邮政编码的值为“94705”，则输出消息“Berkeley author”。因为使用了关键字 exists，所以此例中的 select 语句返回单个值，TRUE 或 FALSE。exists 关键字在此的功能与其在子查询中的功能相同。参见第 5 章“子查询：在其它查询中使用查询”。

以下示例使用了 if 和 else，用于测试是否存在 ID 号大于 50 的用户定义对象。如果存在这些用户对象，则 else 子句选择其名称、类型和 ID 号。

```
if (select max(id) from sysobjects) < 50
   print "There are no user-created objects in this
database."
else
   select name, type, id from sysobjects
   where id > 50 and type = "U"
(0 rows affected)

 name          type   id
------------   ----   ---------
 authors       U       16003088
 publishers    U       48003202
 roysched      U       80003316
 sales         U      112003430
 salesdetail   U      144003544
 titleauthor   U      176003658
 titles        U      208003772
 stores        U      240003886
 discounts     U      272004000
 au_pix        U      304004114
 blurbs        U      336004228
 friends_etc   U      704005539
 test          U      912006280
 hello         U     1056006793

 (14 rows affected)
```

if...else 结构经常用在存储过程中，测试某些参数的存在性。

if 测试可嵌套在其它 if 测试中，在另一 if 中或在 else 后。if 测试中的表达式只能返回单个值。同样，每个 if...else 结构可以有一个 if 语句和一个 else 语句。要包括多个选择语句，请使用 begin...end 关键字。根据每个 if...else 结构所包括的 select 语句（或其它语言结构）的复杂性，可嵌套 if 测试的最大数目也不同。

case 表达式

case 表达式简化了许多条件 Transact-SQL 结构。无需使用一系列 if 语句，case 表达式可以使用一系列条件，条件满足时将返回适当值。case 表达式符合 ANSI SQL92。

使用 case 表达式，可以：

- 简化查询并编写更有效的代码

- 在数据库（如 int）和应用程序（如 char）所使用的格式之间转换数据
- 返回列列表中的第一个非空值
- 比较两个值，不匹配时返回第一个值，或匹配时返回 NULL 值
- 编写避免除零的查询

case 表达式包括关键字 case、when、then、coalesce 和 nullif。coalesce 和 nullif 是 case 表达式的缩写形式。有关 case 表达式语法的详细信息，参见*参考手册*。

在替代表示中使用 *case* 表达式

使用 case 表达式表示的数据对用户更有意义。例如，pubs2 数据库在 titles 表的 contract 列存储 1 或 0 来指示书籍交易的状态。但是，应用程序代码或用户交互中，可能愿意使用字“Contract”或“No Contract”来指示书籍的状态。要从 titles 表中选择输出类型，使用替代表示：

```
select title, "Contract Status" =
case
        when contract = 1 then "Contract"
        when contract = 0 then "No Contract"
end
from titles
```

```
title                                   Contract Status
-----                                   ---------------
The Busy Executiveís Database Guide     Contract
Cooking with Computers: Surreptitio     Contract
You Can Combat Computer Stress!         Contract
. . .
The Psychology of Computer Cooking      No Contract
. . .
Fifty Years in Buckingham Palace        Contract
Sushi, Anyone?                          Contract

 (18 rows affected)
```

case 和除零

case 表达式可以编写可避免除零（称为意外排除）的查询。例如，如果试图用每本书的 total_sales 列去除 advance 列，当查询试图用 title_idMC2222 的 total_sales (2032) 去除 advance (0.00) 时，查询结果除零：

```
select title_id, total_sales, advance,
total_sales/advance from titles
title_id   total_sales          advance
-------    -----------         ---------     ------
BU1032     4095                  5,000.00     0.82
BU1111     3876                  5,000.00     0.78
BU2075     18722                10,125.00     1.85
BU7832     4095                  5,000.00     0.82

Divide by zero occurred.
```

可以使用 case 表达式，通过在方程中不允许数字 0 来避免除零。在此示例中，查询碰到 0 时，返回预定义的值，而不是执行此除法：

```
select title_id, total_sales, advance, "Cost Per Book" =
case
          when advance != 0
          then convert(char, total_sales/advance)
          else "No Books Sold"
end
from titles
title_id       total_sales     advance          Cost Per Book
--------       -----------     ----------       -------------
BU1032                4095       5,000.00                0.82
BU1111                3876       5,000.00                0.78
BU2075               18722      10,125.00                1.85
BU7832                4095       5,000.00                0.82
MC2222                2032           0.00       No Books Sold
MC3021               22246      15,000.00                1.48
MC3026                NULL           NULL       No Books Sold
. . .
TC3218                 375       7,000.00                0.05
TC4203               15096       4,000.00                3.77
TC7777                4095       8,000.00                0.51

 (18 rows affected)
```

title_idMC2222 的除零操作不再会阻止查询的运行。同样，MC3021 的空值也不会阻止查询的运行。

case 表达式结果

确定 case 表达式的数据类型的规则与确定 union 操作中列的数据类型的规则相同。case 表达式有一系列替代结果表达式（如下示例中的 *R1*, *R2*, ..., *Rn*），它们是用 then 和 else 子句来指定的。例如：

```
case
        when search_condition1 then R1
        when search_condition2 then R2
        ...
        else Rn
    end
```

结果表达式 *R1*, *R2*, ..., *Rn* 的数据类型用于确定整个 case 的数据类型。union 列的数据类型用于指定 *n* 表，并将 *R1*, *R2*, ..., *Rn* 作为第 *i* 列，其确定数据类型的规则与确定 case 表达式的数据类型的规则相同。确定 case 的数据类型的方式与使用如下查询进行确定相同：

```
select...R1...from ...
union
select...R2...from...
union...
...
select...Rn...from...
```

并不是所有的数据类型都是兼容的，如果指定两个不兼容的数据类型（例如，*char* 和 *int*），则 Transact-SQL 查询将会失败。有关 union 数据类型规则的详细信息，参见*参考手册*。

case 表达式要求至少返回一个非空值

case 表达式的结果必须至少返回一个非空值。以下查询：

```
select price,
case
        when title_id like "%" then NULL
        when pub_id like "%" then NULL
end
from titles
```

返回错误消息：

```
All result expressions in a CASE expression must not
be NULL
```

case

使用 case 表达式可以测试确定结果集的条件。

语法为：

```
case
        when search_condition1 then result1
        when search_condition2 then result2
        . . .
        when search_conditionn then resultn
        else resultx
    end
```

其中，*search_condition* 是逻辑表达式，*result* 是表达式。

如果 *search_condition1* 为真，则 case 的值为 *result1*；如果 *search_condition1* 为假，则检查 *search_condition2*。如果 *search_condition2* 为真，则 case 的值为 *result2*，依此类推。如果搜索条件都为假，则 case 的值为 *resultx*。else 子句是可选的。如果没有使用它，则缺省为 else NULL。end 指示 case 表达式的结束。

每个书店的每本书的总销售额都保存在 salesdetail 表中。如果希望显示一系列范围的书籍销售额，则使用以下查询可以跟踪每个书店中每本书的销售额：

- 销售量小于 1000 的书籍（低销售额的书籍）
- 销售量在 1000 与 3000 之间的书籍（中等销售额的书籍）
- 销售量大于 3000 的书籍（高销售额的书籍）

可以编写以下查询：

```
select stor_id, title_id, qty, "Book Sales Catagory" =
     case
           when qty < 1000
              then "Low Sales Book"
           when qty >= 1000 and qty <= 3000
              then "Medium Sales Book"
           when qty > 3000
           then "High Sales Book"
     end
from salesdetail
group by title_id
stor_id     title_id      qty      Book Sales Catagory
-------     --------      ----     -------------------
5023        BU1032        200      Low Sales Book
5023        BU1032       1000      Low Sales Book
7131        BU1032        200      Low Sales Book
. . .
```

```
7896        TC7777         75      Low Sales Book
7131        TC7777         80      Low Sales Book
5023        TC7777       1000      Low Sales Book
7066        TC7777        350      Low Sales Book
5023        TC7777       1500      Medium Sales Book
5023        TC7777       1090      Medium Sales Book

(116 rows affected)
```

以下示例根据 author 的版税率 (*royaltyer*) 从 titleauthor 表中选择 titles，并且为每个 title 分配一个高的、中等的或低的版税：

```
select title, royaltyper, "Royalty Category" =
   case
      when (select avg(royaltyper) from titleauthor tta
        where t.title_id = tta.title_id) > 60 then "High Royalty"
      when (select avg(royaltyper) from titleauthor tta
        where t.title_id = tta.title_id) between 41 and 59
      then "Medium Royalty"
      else "Low Royalty"
   end
from titles t, titleauthor ta
where ta.title_id = t.title_id
order by title
title                                  royaltyper   royalty Category
-------                                ----------   ----------------
But Is It User Friendly?               100          High Royalty
Computer Phobic and Non-Phobic Ind     25           Medium Royalty
Computer Phobic and Non-Phobic Ind     75           Medium Royalty
Cooking with Computers: Surreptiti     40           Medium Royalty
Cooking with Computers: Surreptiti     60           Medium Royalty
Emotional Security: A New Algorith     100          High Royalty
. . .
Sushi, Anyone?                         40           Low Royalty
The Busy Executiveís Database Guide    40           Medium Royalty
The Busy Executiveís Database Guide    60           Medium Royalty
The Gourmet Microwave                  75           Medium Royalty
You Can Combat Computer Stress!        100          High Royalty

 (25 rows affected)
```

case 和值比较

这种形式的 case 用于值比较。它只允许进行两个值的等同性检查；而不允许其它的比较。

此命令的语法为：

```
case valueT
        when value1 then result1
        when value2 then result2
        . . .
        when valuen then resultn
        else resultx
    end
```

其中，*value* 和 *result* 是表达式。

如果 *valueT* 等于 *value1*，则 case 的值为 *result1*。如果 *valueT* 不等于 value1，则 *valueT* 与 *value2* 相比较。如果 *valueT* 等于 *value2*，则 case 的值为 *result2*，依此类推。 如果 *valueT* 的值不等于 *value1* 到 *valuen*，则 case 的值为 *resultx*。

至少有一个结果必须是非空值。所有的结果表达式都必须是兼容的。同样，所有的 values 也必须都是兼容的。

上述语法等价于：

```
case
        when valueT = value1 then result1
        when valueT = value2 then result2
        . . .
        when valueT = valuen then resultn
        else resultx
    end
```

这与用于 case 和搜索条件的格式相同（有关此语法的详细信息，参见第 413 页上的“case”）。

以下示例从 titles 表中选择 title 和 pub_id，并基于 pub_id 指定每本书的出版商：

```
select title, pub_id, "Publisher" =
   case pub_id
     when "0736" then "New Age Books"
     when "0877" then "Binnet & Hardley"
     when "1389" then "Algodata Infosystems"
     else "Other Publisher"
end
from titles
order by pub_id
title                           pub_id       Publisher
```

```
-----                           ------       -------------
Life Without Fear                 0736       New Age Books
Is Anger the Enemy?               0736       New Age Books
You Can Combat Computer           0736       New Age Books
. . .
Straight Talk About Computers     1389       Algodata Infosystems
The Busy Executiveís Database     1389       Algodata Infosystems
Cooking with Computers: Surre     1389       Algodata Infosystems

 (18 rows affected)
```

这等价于以下使用 case 和搜索条件语法的查询：

```
select title, pub_id, "Publisher" =
case
     when pub_id = "0736" then "New Age Books"
     when pub_id = "0877" then "Binnet & Hardley"
    when pub_id = "1389" then "Algodata Infosystems"
     else "Other Publisher"
end
from titles
order by pub_id
```

coalesce

coalesce 检查一系列值 (*value1*, *value2*, ..., *valuen*) 并返回第一个非空值。coalesce 的语法为：

coalesce(*value1*, *value2*, ..., *valuen*)

其中，*value1*, *value2*, ..., *valuen* 是表达式。如果 *value1* 为非空值，则 coalesce 的值为 *value1*；如果 *value1* 为空值，则检查 *value2*，依此类推。继续检查，直到找到非空值为止。第一个非空值成为 coalesce 的值。

使用 coalesce 时，Adaptive Server 在内部将其翻译为以下格式：

```
case
    when value1 is not NULL then value1
    when value2 is not NULL then value2
    . . .
    when valuen-1 is not NULL then valuen-1
    else valuen
end
```

valuen-1 指最后一个值前的值，在最后的值（*valuen*）的前面。

以下示例使用 coalesce 来确定书店订购的书是低数量（大于 100 但小于 1000）还是高数量（大于 1000）：

```
select stor_id, discount, "Quantity" =
    coalesce(lowqty, highqty)
from discounts
stor_id discount    Quantity
------- --------   ---------   ----
NULL               10.500000   NULL
NULL                6.700000    100
NULL               10.000000   1001
8042                5.000000   NULL

 (4 rows affected)
```

nullif

nullif 比较两个值；如果其相等，则 nullif 返回空值。如果两个值不相等，则 nullif 返回其中第一个的值。这有利于查找以编码形式存储的任何遗失的、未知的或不适用的信息。例如，以前一些作为 -1 存储的未知值。使用 nullif 可以用空值替代 -1 值，并获取 Transact-SQL 所定义的空值行为。此命令的语法为：

nullif(*value1*, *value2*)

如果 *value1* 等于 *value2*，则 nullif 返回 NULL。如果 *value1* 不等于 *value2*，则 nullif 返回 *value1*。*value1* 和 *value2* 是表达式，并且其数据类型必须是可比的

使用 nullif 时，Adaptive Server 在内部将其翻译为以下格式：

```
case
      when value1 = value2 then NULL
      else value1
   end
```

例如，titles 表使用值“UNDECIDED”来表示还没有确定类别的书。以下查询在 titles 表上执行书籍类型的搜索；任何类型为“UNDECIDED”的书籍都作为类型 NULL 返回（以下输出出于显示目的进行了重新格式化）：

```
select title, "type"=
   nullif(type, "UNDECIDED")
from titles
title                                    type
-----                                    --------
The Busy Executiveís Database Guide      business
Cooking with Computers: Surreptiti       business
You Can Combat Computer Stress!          business
. . .
The Psychology of Computer Cooking       NULL
```

```
Fifty Years in Buckingham Palace K     trad_cook
Sushi, Anyone?                         trad_cook

 (18 rows affected)
```

注意，*The Psychology of Computing* 作为类型“UNDECIDED”存储在表中，但是查询中将其返回为类型 NULL。

begin...end

begin 和 end 关键字将一系列语句涵括在其中，以便类似于 if...else 的控制流结构将其作为一个单元来处理。涵括在 begin 和 end 中的一系列语句称为一个**语句块**。

begin...end 的语法为：

begin
    statement block
end

以下是一个示例：

```
if (select avg(price) from titles) < $15
begin
 update titles
    set price = price * 2

   select title, price
from titles
   where price > $28
end
```

没有 begin 和 end，if 条件只应用第一个 Transact-SQL 语句。第二个语句的执行独立于第一个语句。

begin...end 块可以嵌套在其它 begin...end 块中。

while 和 break...continue

while 设置重复执行语句或语句块的条件。只要指定条件为真，语句就会重复执行。

此命令的语法为：

while *boolean_expression*
  statement

此例中，只要平均价格小于 $30，就重复执行 select 和 update 语句：

```
while (select avg(price) from titles) < $30
begin
select title_id, price
from titles
   where price > $20
 update titles
    set price = price * 2
end
(0 rows affected)

title_id    price
------      -------
PC1035      22.95
PS1372      21.59
TC3218      20.95

 (3 rows affected)
 (18 rows affected)
 (0 rows affected)
title_id    price
------      -------
BU1032      39.98
BU1111      23.90
BU7832      39.98
MC2222      39.98
PC1035      45.90
PC8888      40.00
PS1372      43.18
PS2091      21.90
PS3333      39.98
TC3218      41.90
TC4203      23.90
TC7777      29.98

 (12 rows affected)
 (18 rows affected)
 (0 rows affected)
```

break 和 continue 控制 while 循环中语句的执行。break 导致从 while 循环中退出。执行在以 end 关键字作为循环结束标记之后的所有语句。continue 导致 while 循环重新启动，并跳过此循环内 continue 之后的所有语句。break 和 continue 经常由 if 测试激活。

break...continue 的语法为：

```
while boolean expression
   begin
      statement
      [statement]...
      break
      [statement]...
      continue
      [statement]...
   end
```

以下示例中使用 while、break、continue 和 if 来撤消上述示例中所导致的涨价。只要平均价格大于 $20，将其价格减半。然后选择最高价格。如果其小于 $40，则退出 while 循环；否则，就继续循环。continue 仅在平均价格大于 $20 时才允许执行 print。while 循环结束之后，输出一条消息和最高价格的书籍列表。

```
while (select avg(price) from titles) > $20
begin
 update titles
        set price = price / 2
    if (select max(price) from titles) < $40
        break
else
        if (select avg(price) from titles) < $20
            continue
    print "Average price still over $20"
end

select title_id, price from titles
    where price > $20

print "Not Too Expensive"
(18 rows affected)
 (0 rows affected)
 (0 rows affected)
Average price still over $20
 (0 rows affected)
 (18 rows affected)
 (0 rows affected)

title_id    price
--------    -------
PC1035        22.95
PS1372        21.59
TC3218        20.95
```

```
 (3 rows affected)
Not Too Expensive
```

如果嵌套有两个或多个 while 循环，则 break 退出到下一外层循环。首先，执行循环内部 end 之后的所有语句。然后，重新启动外部循环。

declare 和局部变量

局部变量用 declare 关键字来声明、命名和指定类型，并用 select 语句为其赋初始值。必须在同一个批处理或过程中声明、赋值并使用它们。

有关详细信息，参见第 429 页上的“局部变量”。

goto

goto 关键字导致到用户定义标签的无条件的分支。goto 和标签可用于存储过程和批处理。标签名必须符合标识符规则，并且必须在首次给出时跟有冒号 (:)。当与 goto 一起使用时，它不后接冒号。

此命令语法为：

label:
　goto *label*

以下示例中使用 goto 和标签、while 循环和一个作为计数器的局部变量：

```
declare @count smallint
select @count = 1
restart:
print "yes"
select @count = @count + 1
while @count <=4
   goto restart
```

为了避免 goto 和标签之间的无穷循环，通常使 goto 依赖于 while 或 if 测试，或者其它一些条件。

return

return 关键字无条件地退出批处理或过程。它可用在批处理或过程中的任何位置。用于存储过程时，return 可以接收可选的自变量以向调用者返回一个状态。return 之后的语句不会执行。

此命令的语法为：

return [*int_expression*]

以下的存储过程示例使用了 return 和 if...else 以及 begin...end：

```
create procedure findrules @nm varchar(30) = null as
if @nm is null
begin
  print "You must give a user name"
  return
end
else
begin
   select sysobjects.name, sysobjects.id,
sysobjects.uid
   from sysobjects, master..syslogins
   where master..syslogins.name = @nm
   and sysobjects.uid = master..syslogins.suid
   and sysobjects.type = "R"
end
```

如果调用 findrules 时没有指定用户名参数，则 return 关键字将导致退出此过程（将消息发送给用户屏幕之后）。如果指定了用户名，则从相应的系统表中检索用户所拥有的规则名称。

return 类似于在 while 循环中使用的 break 关键字。

第 14 章“使用存储过程”中有使用返回值的示例。

print

上述示例中用到的 print 关键字，在用户屏幕上显示用户定义消息或局部变量的内容。必须在使用局部变量的批处理或过程中声明它。消息可以长达 255 个字节。

此命令的语法为：

print {*format_string* | *@local_variable* |
@@global_variable} [,*arg_list*]

例如：

```
if exists (select postalcode from authors
   where postalcode = "94705")
print "Berkeley author"
```

以下是使用 print 来显示局部变量内容的方法：

```
declare @msg char(50)
select @msg = "Whatís up, doc?"
print @msg
```

print 可识别要输出字符串中的占位符。格式字符串最多可包含 20 个任意顺序的唯一占位符。将消息文本发送到客户端时，使用 *format_string* 后接自变量的格式化内容替换这些占位符。

为了在用其它语法结构将格式字符串翻译为某种语言时，可以对自变量重新排序，要对占位符进行编号。自变量所用占位符的显示形式如下： *%nn!*。它由百分比符号 (%) 和后接的 1 到 20 之间的整数以及后接的感叹号 (!) 组成。整数表示原始语言中占位符在字符串中的位置。"%1!" 是原始版本中的第一个自变量，"%2!" 是第二个自变量，依此类推。用这种方法指示自变量的位置可以使翻译正确，即使自变量出现在目标语言中的顺序与其在原始语言中的顺序不同。

例如，假定以下是一条英文消息：

```
%1! is not allowed in %2!.
```

此消息的德文版本是：

```
%1! ist in %2! nicht zul%ssig.
```

此消息的日文版本是：

%2! の中で %1! は許されません。

此示例中，“%1!”在所有三种语言中都表示同一个自变量，“%2!”在所有三种语言中也表示同一个自变量。此示例显示了翻译格式中有时需要对自变量重新排序。

虽然占位符不必按数值顺序使用，但在格式字符串中使用占位符时不能跳过占位符编号。例如，不允许将占位符 1 和 3 放在一个格式字符串中，却不把占位符 2 也放在此字符串中。

可选的 *arg_list* 可以是一系列变量或常量。自变量可以是除 text 或 image 以外的任何数据类型；在出现在最终的消息之前，它将被转换为 char 数据类型。如果没有提供任何自变量列表，则格式字符串必须是要输出的没有任何占位符的消息。

替换后，*format_string* 加上所有自变量的最大输出字符串长度是 512 个字节。

raiserror

raiserror 在用户屏幕上显示用户定义错误或局部变量消息，并设置系统标记来记录已出现过错误。使用 print 时必须在使用其的批处理或过程中声明局部变量。消息可以长达 255 个字节。

以下是 raiserror 的语法：

```
raiserror error_number
    [{format_string | @local_variable}] [, arg_list]
    [extended_value  = extended_value  [{,
    extended_value  = extended_value }...]]
```

error_number 放置在全局变量 *@@error* 中，它存储由 Adaptive Server 最近生成的错误编码。用户定义错误消息的错误编码必须大于 17,000。如果 *error_number* 在 17,000 和 19,999 之间，并且 *format_string* 缺失或空白 (“”)，则 Adaptive Server 从 master 数据库的 sysmessages 表中检索错误消息文本。这些错误消息主要供系统过程使用。

format_string 的长度限制在 255 个字节以内； *format_string* 的最大输出长度加上所有自变量限制在 512 个字节以内。raiserror 消息所用的局部变量必须是 char 或 varchar。*format_string* 或变量是可选的；如果不包括它们，则 Adaptive Server 使用缺省语言的 sysusermessages 中相应 *error_number* 的消息。使用 print 可以替代由 *format_string* 中 *arg_list* 所定义的变量或常量。

作为选项，可以定义 Open Client 应用程序所用的扩展错误数据（当在 raiserror 中包括 *extended_values* 时）。有关扩展错误数据的详细信息，参见 Open Client 文档或*参考手册*中的 raiserror。

如果希望将错误编号存储在 *@@error* 中，请使用 raiserror，而不要使用 print。以下是如何在过程 findrules 中使用 raiserror 的示例：

```
raiserror 99999 "You must give a user name"
```

所有用户定义错误消息的严重级是 16，表示用户已经犯了非致命错误。

创建 *print* 和 *raiserror* 所用的消息

可用 sp_getmessage 从 sysusermessages 中调用 print 或 raiserror 所用的消息。使用 sp_addmessage 创建一组消息。

以下示例使用 sp_addmessage、sp_getmessage 和 print 用英文和德文在 sysusermessages 建立消息、检索它并用于用户定义存储过程，然后输出这些消息。

```
/*
** Install messages
** First, the English (langid = NULL)
*/
set language us_english
go
sp_addmessage 25001,
  "There is already a remote user named '%1!' for
remote server '%2!'."
go
/* Then German*/
sp_addmessage 25001,
        "Remotebenutzername '%1!' existiert bereits
auf dem Remoteserver '%2!'.","german"
go
go
create procedure test_proc @remotename varchar(30),
            @remoteserver varchar(30)
as
        declare @msg varchar(255)
        declare @arg1 varchar(40)
        /*
        ** check to make sure that there is not
        ** a @remotename for the @remoteserver.
        */
        if exists (select *
            from master.dbo.sysremotelogins l,
                master.dbo.sysservers s
            where l.remoteserverid = s.srvid
                and s.srvname = @remoteserver
                and l.remoteusername = @remotename)
        begin
            exec sp_getmessage 25001, @msg output
           select @arg1=isnull(@remotename, "null")
            print @msg, @arg1, @remoteserver
            return (1)
        end
return(0)
go
```

如第 233 页上的“为约束创建错误消息”中所述，也可以给用户定义消息绑定约束。

要删除用户定义消息，使用 sp_dropmessage。要更改消息，首先使用 sp_dropmessage 将其删除，然后使用 sp_addmessage 再次添加它。

waitfor

waitfor 关键字指定一天中的特定时间、时间间隔或者执行语句块、存储过程或事务的时间。

此命令的语法为：

waitfor {delay "*time*" | time "*time*" | errorexit | processexit | mirrorexit}

其中，delay *time* 指示 Adaptive Server 要等到指定的时间段已经过去。time *time* 指示 Adaptive Server 要等到指定的时间，它使用 datetime 的有效数据格式之一。

但是，不能指定日期 — 不允许指定 datetime 值的日期部分。用 waitfor time 或 waitfor delay 指定的时间可包括小时、分钟和秒 — 最多为 24 小时。使用格式“hh:mm:ss”。

例如，以下命令指示 Adaptive Server 等到 4:23 p.m.：

```
waitfor time "16:23"
```

以下命令指示 Adaptive Server 等待 1 小时 30 分钟：

```
waitfor delay "01:30"
```

有关 time 值所用格式的概述，参见第 281 页上的“输入时间”。

errorexit 指示 Adaptive Server 要等待到进程异常终止。 processexit 等到进程因任何原因终止。mirrorexit 等待到读取或写入镜像设备操作失败。

可将 waitfor errorexit 用于注销异常终止进程的过程，从而释放系统资源，避免被受影响的进程占用。要找到受影响的进程，使用 sp_who 来检查 sysprocesses 表。

以下示例指示 Adaptive Server 等到 2:20 p.m.。然后用下一步更新 chess 表并执行存储过程 sendmessage，在 Judy 的一个表中插入一条消息，通知她在 chess 表中存在下一步。

```
begin
waitfor time "14:20"
```

```
insert chess(next_move)
values("Q-KR5")
execute sendmessage "Judy"
end
```

要在 10 秒钟之后而不是等到 2:20 向 Judy 发送消息，替换上述示例中的 waitfor 语句：

```
waitfor delay "0:00:10"
```

发出 waitfor 命令后，在达到指定的时间或发生指定的事件前，不能使用与 Adaptive Server 的连接。

注释

使用注释符号使注释附加到语句、批处理和存储过程。不会执行注释。

可以单独一行或在命令行结束处插入注释。可以使用两种注释样式：“斜线 — 星号”样式：

/* text of comment */

和“双连字符”样式：

-- text of comment

对于多行注释，不能使用双连字符样式。

注释不限制最大长度。

斜线 - 星号样式注释

/* 样式注释是 Transact-SQL 扩展。只要每条注释以“/*”开始并以“*/”结束，多行注释是可接受的。“/*”和“*/”之间的所有内容作为注释的一部分来处理。/* 形式允许嵌套。

多行注释惯用的样式是第一行以“/*”开始，后续各行以“**”开始。注释通常以“*/”结束：

```
select * from titles
/* A comment here might explain the rules
** associated with using an asterisk as
** shorthand in the select list.*/
where price > $5
```

以下过程包括两个注释：

```
/* this procedure finds rules by user name*/
create procedure findmyrule @nm varchar(30) = null
as
if @nm is null
begin
  print "You must give a user name"
return
  print "I have returned"
/* this statement follows return,
** so wonít be executed */
end
else      /* print the rule names and IDs, and
          the user ID */
  select sysobjects.name, sysobjects.id,
     sysobjects.uid
  from sysobjects, master..syslogins
  where  master..syslogins.name = @nm
  and sysobjects.uid = master..syslogins.suid
  and sysobjects.type = "R"
```

双连字符样式注释

此注释样式以两个连续的连字符和随后的一个空格 (--) 开始并以换行符终止。因此，多行注释是不可能的。

Adaptive Server 不会将字符串文本中或 /* 样式注释中的两个连续连字符解释为注释开始的标记。

要表示包含两个连续减号的表达式（binary 后接 unary）, 则在两个连字符之间放置一个空格或一个开始小括号。

示例如下：

```
-- this procedure finds rules by user name
create procedure findmyrule @nm varchar(30) = null
as
if @nm is null
begin
  print "You must give a user name"
return
  print "I have returned"
-- each line of a multiple-line comment
-- must be marked separately.
end
else        -- print the rule names and IDs, and
```

```
      -- the user ID
select sysobjects.name, sysobjects.id,
   sysobjects.uid
from sysobjects, master..syslogins
where  master..syslogins.name = @nm
and sysobjects.uid = master..syslogins.suid
and sysobjects.type = "R"
```

局部变量

局部变量经常用作批处理或存储过程中 while 循环或 if...else 块的计数器。在存储过程中使用局部变量时，它们被声明为，在其执行过程中由该过程自动并且非交互地使用。几乎可以在任何 Transact-SQL 语法指示可以使用表达式的位置使用变量，例如 *char_expr*、*integer_expression*、*numeric_expr* 或 *float_expr*。

声明局部变量

要 declare 局部变量的名称和数据类型，使用：

declare @*variable_name datatype*
  [, @*variable_name datatype*]...

变量名前面必须带有 @ 符号，且必须符合标识符的规则。指定 text、image 或 sysname 外的用户定义数据类型或系统提供的数据类型。

从记忆和性能的角度看，这样编写更有效：

```
declare @a int, @b char(20), @c float
```

而不是编写为：

```
declare @a int
declare @b char(20)
declare @c float
```

局部变量和 *select* 语句

声明变量时，它的值为 NULL。使用 select 语句为局部变量值赋。语法如下：

select @*variable_name* = { *expression* |
(*select_statement*) } [, @*variable* =
{ *expression* | (*select_statement*) } ...]
[from *clause*] [where *clause*] [group by *clause*]
[having *clause*] [order by *clause*] [compute *clause*]

使用 declare 语句时，这样编写更有效：

```
select @a = 1, @b = 2, @c = 3
```

而不是编写为：

```
select @a = 1
select @b = 2
select @c = 3
```

不要使用单个 select 语句为某个变量和另一基于此变量值的变量同时赋值。这样做可能产生不可预料的结果。例如，如下两个查询都要查找 *@c2* 的值。第一个查询产生值 NULL，而第二个查询产生正确答案为 0.033333：

```
/* this is wrong*/
declare @c1 float, @c2 float
select @c1 = 1000/1000, @c2 = @c1/30
select @c1, @c2

/* do it this way */
declare @c1 float, @c2 float
select @c1 = 1000/1000
select @c2 = @c1/30
select @c1 , @c2
```

不能使用 select 语句给变量赋值，同时为用户返回数据。如下示例中的第一个 select 语句给局部变量 *@veryhigh* 赋以最高价格；需要第二个 select 语句显示值：

```
declare @veryhigh money
select @veryhigh = max(price)
from titles
select @veryhigh
```

如果为变量赋值的 select 语句返回多个值，则最后返回的值将赋给此变量。以下查询给变量赋以“select advance from titles”最后返回的值。

```
declare @m money
```

```
select @m = advance from titles
select @m
(18 rows affected)
------------------------
                8,000.00

(1 row affected)
```

赋值语句指示有多少行受 select 语句的影响（返回）。

如果给变量赋值的 select 语句没有返回任何值，则该语句没有更改变量的值。

局部变量可以用作 print 或 raiserror 的自变量。

如果在 update 语句中使用变量，参见第 301 页上的“将 set 子句用于 update”。

局部变量和 *update* 语句

可以直接在 update 语句中给变量赋值。无需用 select 语句为变量赋值。声明变量时，它的值为 NULL。语法如下：

update [[*database*.]*owner*.]{*table_name* | *view_name*}
set [[[*database*.]*owner*.]{*table_name*. | *view_name*.}]
column_name1 =
{*expression1* | null | (*select_statement*)} |
variable_name1 = {*expression1* | null}
[, *column_name2* = {*expression2* | null |
(*select_statement*)}]... |
[, *variable_name2* = {*expression2* | null}]...
[from [[*database*.]*owner*.]{*table_name* | *view_name*}
[, [[*database*.]*owner*.]{*table_name* |
view_name}]]...
[where *search_conditions*]

局部变量和子查询

为局部变量赋值的子查询*必须*仅返回一个值。示例如下：

```
declare @veryhigh money
select @veryhigh = max(price)
from titles
if @veryhigh > $20
   print "Ouch!"
declare @one varchar(18), @two varchar(18)
```

```
select @one = "this is one", @two = "this is two"
if @one = "this is one"
   print "you got one"
if @two = "this is two"
   print "you got two"
else print "nope"
declare @tcount int, @pcount int
select @tcount = (select count(*) from titles),
   @pcount = (select count(*) from publishers)
select @tcount, @pcount
```

局部变量和 *while* 循环以及 *if...else* 块

以下示例在 while 循环的计数器中使用局部变量，以与 if 语句中的 where 子句匹配，并设置和重新设置 select 语句中的值：

```
/* Determine if a given au_id has a row in au_pix*/
/* Turn off result counting */
set nocount on
/* declare the variables */
declare @c int,
        @min_id varchar(30)
/*First, count the rows*/
select @c = count(*) from authors
/* Initialize @min_id to "" */
select @min_id = ""
/* while loop executes once for each authors row */
while @c > 0
begin
        /*Find the smallest au_id*/
        select @min_id = min(au_id)
            from authors
            where au_id > @min_id
        /*Is there a match in au_pix?*/
        if exists (select au_id
            from au_pix
            where au_id = @min_id)
            begin
            print "A Match! %1!", @min_id
        end
      select @c = @c -1  /*decrement the counter */
end
```

变量和空值

声明局部变量时给其赋值 NULL，也可用 select 语句为其赋以空值。NULL 的特殊含义要求空值变量和其它空值之间的比较遵循特殊的规则。

表 13-2 显示了在空值列和空值表达式之间使用不同的比较运算符的比较结果。表达式可以是变量、文字或变量和文字以及算术运算符的组合。

表 *13-2*：比较空值

比较的类型	使用 = 运算符	使用 <、>、<=、!=、!<、!> 或 <> 运算符
column_value 与 *column_value* 相比	FALSE	FALSE
column_value 与 *expression* 相比	TRUE	FALSE
expression 与 *column_value* 相比	TRUE	FALSE
expression 与 *expression* 相比	TRUE	FALSE

例如，如下测试：

```
declare @v int, @i int
if @v = @i select "null = null, true"
if @v > @i select "null > null, true"
```

显示了只有第一个比较返回 TRUE：

```
-----------------
null = null, true

(1 row affected)
```

此示例返回 advance 为 NULL 的 titles 表中的所有行：

```
declare @m money
select title_id, advance
from titles
where advance = @m
title_id advance
-------- ----------------
MC3026              NULL
PC9999              NULL

(2 rows affected)
```

全局变量

全局变量是系统支持的预定义变量。与局部变量不同的是它们在名称前有两个 @ 符号 — 例如，*@@error*。两个 @ 符号是定义全局变量所用标识符的一部分。

用户不能在 select 语句中，直接创建全局变量，也不能直接更新全局变量的值。如果用户声明一个与某**全局变量**同名的局部变量，则此变量作为局部变量处理。

事务和全局变量

一些全局变量提供用于事务的信息。

用 *@@error* 检查错误

@@error 全局变量通常用于检查当前用户会话中最近执行的批处理的错误状态。如果最后的事务成功，则 *@@error* 包含 0；否则 *@@error* 包含系统生成的最后一个错误编号。类似 if @@error != 0 后接 return 的语句，会导致在出错时退出。

每个 Transact-SQL 语句，包括 print 语句和 if 测试，重新设置 *@@error* 以便状态检查必须立即跟踪存在问题的批处理。

@@sqlstatus 全局变量对 *@@error* 输出没有影响。有关详细信息，参见第 435 页上的“检查最后的 fetch 的状态”。

用 *@@identity* 检查 IDENTITY 值

@@identity 包含当前用户会话中最后插入 IDENTITY 列的值。每次 insert、select into 或 bcp 要向表中插入一行时，设置 *@@identity* 的值。insert、select into 或 bcp 语句的失败或者事务回退，不影响 *@@identity* 的值。*@@identity* 保留最后插入 IDENTITY 列的值，即使向 IDENTITY 列插入该值的语句提交失败。

如果语句插入多行，则 *@@identity* 反映插入的最后一行的 IDENTITY 值。如果受影响的表不包含 IDENTITY 列，则 *@@identity* 被设置为 0。

用 @@*trancount* 检查事务嵌套级

@@trancount 包含当前用户会话中事务的嵌套级。批处理中每个 begin transaction 将事务计数加 1。在链式事务模式中查询 *@@trancount* 时，因为查询自动开始一个事务，所以其值不会为 0。

用 @@*transtate* 检查事务状态

@@transtate 包含当前用户会话执行某个语句之后事务的当前状态。但是，与 *@@error* 不同，*@@transtate* 不会被每个批处理清除。*@@transtate* 可能包含如表 13-3 中所述的值：

表 *13-3*：@@*transtate* 值

值	含义
0	事务正在进行： 正在处理显式或隐式事务；前一语句已成功执行。
1	事务已经成功： 完成事务并提交其更改。
2	语句被中止： 前一语句已被中止；对事务无影响。
3	事务被中止： 事务被中止并回退任何更改。

@@transtate 只因执行错误而更改。语法和编译错误不会影响 *@@transtate* 的值。有关显示事务的 *@@transtate* 的示例，参见第 574 页上的“检查事务的状态”。

用 @@*nestlevel* 检查嵌套级

@@nestlevel 包含用户会话中当前执行的嵌套级，初始值为 0。每当存储过程或触发器调用另一存储过程或触发器，将嵌套级加 1。如果超过了最大值 16，事务中止。

检查最后的 *fetch* 的状态

@@sqlstatus 包含当前用户会话中最后的 fetch 语句导致的状态信息。*@@sqlstatus* 可能包含以下值：

表 13-4：@@sqlstatus 值

值	含义
0	fetch 语句成功完成。
1	fetch 语句以出错结束。
2	结果集中不再有数据。当前游标位置在结果集中的最后一行并且客户端对该游标提交了 fetch 语句时，就会出现这一警告。

@@sqlstatus 对 *@@error* 输出没有影响。例如，以下批处理使 fetch *@@error* 语句出错，*@@sqlstatus* 被设置为 1。当然，*@@error* 反映错误消息的编号，而不是 *@@sqlstatus* 输出：

```
declare csr1 cursor
for select * from sysmessages
for read only

open csr1

begin
    declare @xyz varchar(255)
    fetch csr1 into @xyz
    select error = @@error
    select sqlstatus = @@sqlstatus
end
Msg 553, Level 16, State 1:
Line 3:
The number of parameters/variables in the FETCH INTO
clause does not match the number of columns in cursor
'csr1'result set.
```

此时，将 *@@error* 全局变量被设置为 553，即最后生成错误的编号。*@@sqlstatus* 设置为 1。

受 *set* 选项影响的全局变量

set 选项可以自定义结果的显示、显示处理的统计信息并提供其它调试 Transact-SQL 程序所用的诊断帮助。

表 13-5 列出了包含由 set 命令控制的会话选项有关信息的全局变量。

表 *13-5*：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的 set 选项。
@@parallel_degree	包含当前的最大并行度设置。
@@rowcount	包含最后的查询所影响的行数。任何不返回行的命令（例如 if、update 或 delete语句）将*@@rowcount*设置为0。对于游标，*@@rowcount*表示从游标结果集返回到客户端（直到最后一个 fetch 命令）的累积行数。
@@scan_parallel_degree	包含非聚簇索引扫描的当前最大并行度设置。
@@textsize	包含 select 所返回的 text 或 image 数据的字节数的限制。isql 的缺省限制为 32K 字节；缺省值取决于客户端软件。它可以用 set textsize 为某个会话进行更改。
@@tranchained	包含 Transact-SQL 程序的当前事务模式。*@@tranchained* 为非链接模式则返回 0，为链式模式则返回 1。

@@options 全局变量包含会话的 set 选项的十六进制表示。表 13-6 列出了 *@@options* 使用的 set 选项和值。

表 *13-6*：设置 *@@options* 的选项和值

数字值	十六进制值	*set* 选项
4	0x04	showplan
5	0x05	noexec
6	0x06	arithignore
8	0x08	arithabort
13	0x0D	control
14	0x0E	offsets
15	0x0F	statistics io 和 statistics time
16	0x10	parseonly
18	0x12	procid
20	0x14	rowcount
23	0x17	nocount

有关会话选项的详细信息，参见*参考手册*中的 set 命令。

全局变量中的语言和字符集信息

表 13-7 列出了包含语言和字符集信息的全局变量。有关字符集转换的详细信息，参见*系统管理指南*。

表 *13-7*：用于语言和字符集所用的全局变量

全局变量	说明
@@char_convert	如果字符集转换无效，则包含 0。如果字符集转换有效，则包含 1。
@@client_csid	包含客户端的字符集 ID。如果客户端字符集从未初始化，则设置为 -1；否则，将包含 syscharsets 中最近使用的客户端字符集的 *id*。
@@client_csname	包含客户端字符集名。如果客户端字符集从未初始化，则设置为 NULL；否则，将包含最近使用的字符集的名称。
@@langid	定义当前使用语言的本地语言 ID，如 syslanguages.langid 中所指定。
@@language	定义当前使用的语言的名称，如 syslanguages.name 中所指定。
@@maxcharlen	包含缺省字符集中多字节字符的最大长度（以字节计）。
@@ncharsize	包含国家字符的平均长度（以字节计）。
@@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。

监控系统活动的全局变量

许多全局变量报告自 Adaptive Server 最近一次启动以来，系统发生的活动。sp_monitor 显示一些全局变量的当前值。

表 13-8 以 sp_monitor 返回的顺序列出了监控系统活动的全局变量。有关 sp_monitor 的详细信息，参见*参考手册*。

表 *13-8*：监控系统活动的全局变量

全局变量	说明
@@connections	包含登录或试图登录的次数。
@@cpu_busy	包含自 Adaptive Server 上次启动起，CPU 执行 Adaptive Server 工作所花费的时间（以时钟周期计）。
@@idle	包含 Adaptive Server 自上次启动起空闲的时间（以时钟周期计）。
@@io_busy	包含 Adaptive Server 执行输入和输出操作所花费的时间（以时钟周期计）。
@@packet_errors	包含 Adaptive Server 发送和接收包时所出现的错误数。
@@pack_received	包含 Adaptive Server 自上次启动起所读取的输入包的数目。
@@pack_sent	包含 Adaptive Server 自上次启动起所写入的输出包的数目。

全局变量	说明
@@total_errors	包含 Adaptive Server 在进行读取操作时出现的错误数。
@@total_read	包含 Adaptive Server 自上次启动起所读取的磁盘的数目。
@@total_write	包含 Adaptive Server 自上次启动起所写入的磁盘的数目。

存储在全局变量中的服务器信息

表 13-9 列出了包含 Adaptive Server 的杂类信息的全局变量。

表 *13-9*：包含 *Adaptive Server* 信息的全局变量

全局变量	说明
@@cis_version	包含组件集成服务的日期和版本（如果已经安装）。
@@guestuserid	guest 用户 ID。其值为 2。
@@invaliduserid	无效用户 ID。其值为 -1
@@max_connections	包含在当前计算机环境下可同时与 Adaptive Server 连接的最大数目。使用 number of user connections 配置参数可以将 Adaptive Server 的连接数量配置为少于或等于 *@@max_connection* 的任意值。
@@mingroupid	最小的组用户 ID。其最小值为 16384。
@@minsuid	最小服务器用户 ID。其最小值为 -32768.
@@minuserid	最小用户 ID。其最小值为 -32768。
@@maxgroupid	最大组用户 ID。其最大值为 1048576
@@maxuserid	最大用户 ID。其最大值为 2147483647
@@probesuid	probe 用户 ID。其值为 2。
@@procid	包含当前执行过程的存储过程 ID。
@@servername	包含此 Adaptive Server 的名称。使用 sp_addserver 定义服务器名，并重新启动 Adaptive Server。
@@spid	包含当前进程的服务器进程 ID 号。
@@thresh_hysteresis	包含激活阈值所要求自由空间的减小量。此量也称作停滞值（以 2K 数据库页计）。它确定阈值在数据库段中可放置的接近程度。
@@timeticks	包含每个时钟周期中的微秒数。每个时钟周期的时间量与机器有关。

全局变量	说明
@@version	包含 Adaptive Server 当前版本的日期。

全局变量以及 *text* 和 *image* 数据

text 和 image 的值可能相当大。如果选择列表中包括 text 和 image 值，返回数据的长度限制取决于 *@@textsize* 全局变量的设置，它包含 select 返回的 text 或 image 数据的字节数限制。isql 的缺省限制为 32K 字节；缺省值取决于客户端软件。使用 set textsize 更改会话所用的值。

表 13-10 列出了包含文本指针信息的全局变量。注意，这些全局变量是基于会话的。有关文本指针的详细信息，参见第 304 页上的“改变 text 和 image 数据”。

表 *13-10*：存储在全局变量中的文本指针信息

全局变量	说明
@@textcolid	包含 *@@textptr* 所引用列的列 ID。
@@textdbid	包含数据库的数据库 ID，该数据库含有其列被 *@@textptr* 引用的对象。
@@textobjid	包含对象的对象 ID，该对象含有被 *@@textptr* 引用的列。
@@textpt	包含进程最后插入或更新的 text 或 image 列的文本指针。（不要将此变量与 textptr 函数混淆。）
@@textts	包含 *@@textptr* 所引用列的文本时间戳。

第 14 章

使用存储过程

存储过程是 SQL 语句或控制流语言的已命名集合。可以为常用功能创建存储过程以提高性能。Adaptive Server 也提供执行管理任务和更新系统表的**系统过程**。

本章讨论如下内容：

主题	页
存储过程如何工作	441
创建和执行存储过程	445
返回存储过程的信息	457
与存储过程关联的限制	465
重命名存储过程	466
使用存储过程作为安全机制	467
删除存储过程	467
系统过程	467
获取有关存储过程的信息	475

也可创建扩展的存储过程并用它从 Adaptive Server 中调用过程语言函数。参见第 15 章“使用扩展存储过程”。

存储过程如何工作

运行存储过程时，Adaptive Server 会准备一个执行计划，使该过程能快速执行。存储过程可以：

- 带参数
- 调用其它过程
- 把状态值返回给调用过程或批处理，以指明成功或失败，以及失败的原因
- 把参数的值返回给调用过程或批处理
- 在远程 Adaptive Servers 上执行

编写存储过程的能力大大提高了 SQL 的功能、效率和灵活性。编译后的过程显著提高了 SQL 语句的性能与批处理能力。另外，如果您的服务器和远程服务器都设置为允许远程登录，则可以执行其它 Adaptive Servers 上的存储过程。可以在本地 Adaptive Server 上编写触发器，当本地发生删除、更新或插入事件时，这些触发器执行远程服务器上的过程。

存储过程与普通的 SQL 语句和批 SQL 语句不同，它是预编译的。第一次运行过程时，Adaptive Server 查询处理器对它进行分析，准备执行计划并最终将其保存到**系统表**中。随后，过程将按照存储在系统表中的计划执行。由于大多数查询处理工作已被执行，存储过程的执行几乎是瞬时的。

Adaptive Server 提供多种存储过程，作为方便用户使用的工具。存储在 sybsystemprocs 数据库中的名字以“sp_”开头的过程被称为**系统过程**，因为它们插入、更新、删除并报告系统表中的数据。

创建并使用存储过程的示例

创建简单的、没有特殊功能（如带参数）的存储过程的语法是：

create procedure *procedure_name*
  as *SQL_statements*

存储过程是数据库对象，其名字必须遵循标识符的规则。

除 create 语句外，可以包括任意数目和种类的 SQL 语句。参见第 465 页上的“与存储过程关联的限制”。过程可以象一条列出数据库中所有用户名字的语句那样简单：

```
create procedure namelist
as select name from sysusers
```

要执行存储过程，请使用关键字 execute 和存储过程名，或者只要存储过程由其自己提交给 Adaptive Server 或者它是批处理中的第一条语句，仅用过程名就可以执行它。可以用下述任意方法执行 namelist：

```
namelist
execute namelist
exec namelist
```

要在远程 Adaptive Server 上执行存储过程，必须给出服务器的名称。远程过程调用的完整语法是：

execute *server_name*.[*database_name*].[*owner*].*procedure_name*

以下示例执行 GATEWAY 服务器上的 pubs2 数据库中的 namelist 过程：

```
execute gateway.pubs2..namelist
gateway.pubs2.dbo.namelist
exec gateway...namelist
```

仅当 pubs2 是缺省数据库时，最后的示例才生效。有关在 Adaptive Server 上设置远程过程调用的信息，参见*系统管理指南*。

仅当存储过程位于你的**缺省数据库**中时，数据库名才是可选的。仅当数据库所有者 (“dbo”) 拥有该过程或者是您自己拥有它时，所有者的名称才是可选的。您必须拥有执行该过程的**权限**。

一个过程可包含多条语句。

```
create procedure showall as
select count(*) from sysusers
select count(*) from sysobjects
select count(*) from syscolumns
```

执行过程时，每条命令的结果按相关语句在过程中的先后顺序显示。

```
showall
------------
           5

(1 row affected)

------------
          88

(1 row affected)

------------
         349

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

create procedure 命令执行成功后，过程名存储在 sysobjects 中，其**源文本**存储在 syscomments 中。

可以用 sp_helptext 显示过程的源文本：

```
sp_helptext showall
# Lines of Text
---------------
              1
```

```
(1 row affected)

text
----------------------------------------
create procedure showall as
select count(*) from sysusers
select count(*) from sysobjects
select count(*) from syscolumns

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

存储过程和权限

存储过程可用作一种安全机制，因为用户可以被授予执行存储过程的权限，即使该用户不拥有它所引用的表或视图的权限或执行具体命令的权限。有关详细信息，参见*系统管理指南*。

可保护存储过程的源文本免受未经授权的访问，方法为施加限制，使过程创建者和系统管理员才具有对 syscomments 表中 text 列的 select 权限。这种权限限制要求在**已评估的配置**中运行 Adaptive Server。要实施这种限制，系统安全员必须用 sp_configure 重新设置 allow select on syscomments.text column 参数。有关其详细信息，请参见*系统管理指南*。

保护存储过程源文本免受访问的另一种方法是用 sp_hidetext 隐藏源文本。有关信息，参见*参考手册*中的 sp_hidetext。

存储过程及性能

数据库更改时，可以优化用于访问其表的初始查询计划，方法为用 sp_recompile 重新编译它们。这样就不必查找、删除和重新创建每个存储过程及触发器。下面的示例表示访问表 titles 的所有存储过程和触发器在下次执行时都要重新编译。

```
sp_recompile titles
```

有关 sp_recompile 的详细信息，参见*参考手册*。

创建和执行存储过程

create procedure 的完整语法是：

```
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}
```

只能在当前数据库中创建过程。

执行 create procedure 的权限缺省情况下将授予数据库所有者，该所有者可将此权限移交给其他用户。

以下是 execute 的完整的语法语句：

```
[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]
```

参数

参数是存储过程的自变量。在 create procedure 语句中可选择性地声明一个或多个参数。在执行过程时，用户必须提供在 create procedure 语句中指定的每个参数的值。

参数名前面必须带有 @ 符号，且必须符合标识符的规则，参见第 8 页上的“标识符”。参数名局限于创建它们的过程中；同一参数名可用于其它过程。把含标点符号的任何参数值（如用数据库名或所有者名限定的对象名）用单引号或双引号引起来。参数名（包括 @ 符号）最长为 30 个字节。

必须给定参数的系统数据类型（text 或 image 除外）或用户定义的数据类型，和（如果数据类型需要）长度或精度和标度（在小括号中）。

以下是 pubs2 数据库的一个存储过程。给出作者的姓和名后，该过程将显示出该作者所写任何书的书名和出版者的名称。

```
create proc au_info @lastname varchar(40),
  @firstname varchar(20) as
select au_lname, au_fname, title, pub_name
from authors, titles, publishers, titleauthor
where au_fname = @firstname
and au_lname = @lastname
and authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
and titles.pub_id = publishers.pub_id
```

现在，执行 au_info：

```
au_info Ringer, Anne
au_lname au_fname title                 pub_name
-------- -------- --------------------- ----------
Ringer   Anne     The Gourmet Microwave Binnet & Hardley
Ringer   Anne     Is Anger the Enemy?   New Age Books
                          (2 rows affected, return status = 0)
```

下列存储过程查询系统表。给定一个表名作为参数，则过程将显示出该表名、索引名和索引 ID。

```
create proc showind @table varchar(30) as
select table_name = sysobjects.name,
index_name = sysindexes.name, index_id = indid
from sysindexes, sysobjects
where sysobjects.name = @table
and sysobjects.id = sysindexes.id
```

添加列标题（如 table_name）以增强结果的可读性。以下是执行此存储过程的可接受的语法形式：

```
execute showind titles
exec showind titles
execute showind @table = titles
execute GATEWAY.pubs2.dbo.showind titles
showind titles
```

只要语句是批处理中的唯一语句或第一条语句，最后一种语法形式（没有 exec 或 execute）就可接受。

以下是给定 *titles* 作为参数时，在 pubs2 数据库中执行 showind 的结果：

```
table_name  index_name  index_id
----------  ----------  ----------
titles      titleidind      0
titles      titleind        2

(2 rows affected, return status = 0)
```

如果以“*@parameter* = *value*”的形式提供参数，则可以以任意顺序提供它们。否则，必须以参数的 create procedure 语句的顺序提供它们。如果以“*@parameter* = *value*”的形式提供值，就以这种方式提供所有的后续参数。

此过程将显示 salesdetail 表的 qty 列的数据类型。

```
create procedure showtype @tabname varchar(18),
@colname varchar(18) as
  select syscolumns.name, syscolumns.length,
  systypes.name
  from syscolumns, systypes, sysobjects
  where sysobjects.id = syscolumns.id
  and @tabname = sysobjects.name
  and @colname = syscolumns.name
  and syscolumns.type = systypes.type
```

如果 *@tabname* 和 *@colname* 按名字指定，则执行该过程时可从 create procedure 语句中以不同的顺序给定：

```
exec showtype
@colname = qty , @tabname = salesdetail
```

在任何使用值表达式的存储过程中都可使用 case 表达式。下列示例检查 titles 表中所有书的销售记录：

```
create proc booksales @titleid tid
as
select title, total_sales,
case
when total_sales != null then "Books sold"
when total_sales = null then "Book sales not available"
end
from titles
where @titleid = title_id
```

例如：

```
   booksales MC2222
title                                  total_sales
------------------------               -----------
Silicon Valley Gastronomic Treats    2032 Books sold

(1 row affected)
```

缺省参数

可以在 create procedure 语句中为参数指定缺省值。该值可以是任意常量，如果用户没有指定，它将作为该过程的参数。

以下是一个显示所有作者名字的过程，他们所写的书由以参数形式给定的出版者出版。如果没有提供出版者名称，该过程将显示由 Algodata Infosystems 出版的书的作者。

```
create proc pub_info
  @pubname varchar(40) = "Algodata Infosystems" as
select au_lname, au_fname, pub_name
from authors a, publishers p, titles t, titleauthor
ta
where @pubname = p.pub_name
and a.au_id = ta.au_id
and t.title_id = ta.title_id
and t.pub_id = p.pub_id
```

注意：如果缺省值是含有嵌入空白或标点符号的字符串，则它必须用单引号或双引号引起来。

执行 pub_info 时，可以给出任意出版者的名称作为参数值。如果没有提供任何参数，Adaptive Server 将使用缺省值 Algodata Infosystems。

```
exec pub_info

au_lname     au_fname    pub_name
--------------  ------------  --------------------
Green           Marjorie      Algodata Infosystems
Bennet          Abraham       Algodata Infosystems
O'Leary         Michael       Algodata Infosystems
MacFeather      Stearns       Algodata Infosystems
Straight        Dick          Algodata Infosystems
Carson          Cheryl        Algodata Infosystems
Dull            Ann           Algodata Infosystems
Hunter          Sheryl        Algodata Infosystems
Locksley        Chastity      Algodata Infosystems

(9 rows affected, return status = 0)
```

此过程 showind2 将“titles”指派为 *@table* 参数的缺省值：

```
create proc showind2
@table varchar(30) = titles as
select table_name = sysobjects.name,
    index_name = sysindexes.name, index_id = indid
from sysindexes, sysobjects
where sysobjects.name = @table
and sysobjects.id = sysindexes.id
```

列标题（如 table_name）阐明了结果显示。以下是由 showind2 显示的 authors 表的内容：

```
showind2 authors
table_name  index_name      index_id
----------- -------------   ---------
authors     auidind                 1
authors     aunmind                 2

(2 rows affected, return status = 0)
```

如果用户没有提供任何值，Adaptive Server 将使用缺省值 *titles*。

```
showind2
table_name  index_name  index_id
----------- ----------- ---------
titles      titleidind        1
titles      titleind          2

(2 rows affected, return status =0)
```

如果需要参数却没有提供，且 create procedure 语句没有提供缺省值，则 Adaptive Server 将显示错误信息，列出过程所期望的参数。

如果用户执行存储过程且指定的参数超过该过程所期望的参数数目，Adaptive Server 将忽略多余参数。

NULL 作为缺省参数

可以在 create procedure 语句中声明空值为个别参数的缺省值：

create procedure *procedure_name*
  @param datatype [= null]
  [, *@param datatype* [= null]]...

这种情况下，如果用户没有提供参数，Adaptive Server 执行存储过程时不显示错误信息。

过程定义可以指定在用户未提供参数时要进行的操作，通过检查参数值是否为空。下面是一个示例：

```
create procedure showind3
@table varchar(30) = null as
if @table is null
    print "Please give a table name."
else
   select table_name = sysobjects.name,
     index_name = sysindexes.name,
     index_id = indid
   from sysindexes, sysobjects
   where sysobjects.name = @table
   and sysobjects.id = sysindexes.id
```

如果用户未给定参数，Adaptive Server 将把来自过程的信息输出到屏幕上。

有关将缺省值设置为空的其它示例，请用 sp_helptext 检查系统过程的源文本。

缺省参数中的通配符

如果过程使用带有关键字 like 的参数，则缺省值可包括通配符 %、_、[ ] 和 [^]。

例如，如果用户未提供参数，则可以修改 showind 以显示有关系统表的信息，如下：

```
create procedure showind4
@table varchar(30) = "sys%" as
select table_name = sysobjects.name,
    index_name = sysindexes.name,
    index_id = indid
```

```
from sysindexes, sysobjects
where sysobjects.name like @table
and sysobjects.id = sysindexes.id
```

使用多个参数

以下是 au_info 的变量，在两个参数中，该变量均使用带通配符的缺省值：

```
create proc au_info2
  @lastname varchar(30) = "D%",
  @firstname varchar(18) = "%" as
select au_lname, au_fname, title, pub_name
from authors, titles, publishers, titleauthor
where au_fname like @firstname
and au_lname like @lastname
and authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
and titles.pub_id = publishers.pub_id
```

如果执行不带参数的 au_info2，则将显示所有姓氏以“D”开头的作者：

```
au_info2
au_lname au_fname title                        pub_name
-------- -------  -------------------------    -------------
Dull     Ann      Secrets of Silicon Valley    Algodata Infosystems
DeFrance Michel    The Gourmet Microwave        Binnet & Hardley

(2 rows affected)
```

如果参数有缺省值，则这些参数在执行时可被忽略，并从最后的参数开始执行。不能跳过任何参数，除非为其提供的缺省值是 NULL。

注意：如果以 *@parameter* = *value* 的形式提供参数值，则可以按任意顺序提供参数值。也可省略已经为其提供缺省值的参数。如果以 *@parameter* = *value* 的形式提供了一个值，则以这种方式提供所有后续参数。

作为两个参数的缺省值都已定义后略去第二个参数的示例，可以找到姓氏为“Ringer”的所有作者的书及出版者，如下：

```
au_info2 Ringer
au_lname   au_fname    title                     Pub_name
--------   --------    ----------------------    ------------
Ringer     Anne        The Gourmet Microwave     Binnet & Hardley
Ringer     Anne        Is Anger the Enemy?       New Age Books
Ringer     Albert      Is Anger the Enemy?       New Age Books
Ringer     Albert      Life Without Fear         New Age Books

(4 rows affected)
```

如果用户执行存储过程且指定的参数数目超过过程所期望的数目，Adaptive Server 将忽略多余的参数。

请记住 SQL 是形式自由的语言。没有规定每一行中的单词数量或者必须折行的地方。如果在执行系统过程后执行命令，Adaptive Server 将尝试执行系统过程，然后再执行命令。例如，如果执行：

```
sp_help checkpoint
```

Adaptive Server 将返回 sp_help 的输出，然后运行 checkpoint 命令。如果对过程参数使用分隔标识符，会产生意外结果。

过程组

在 create procedure 和 execute 语句中，过程名后可选的分号和整数可以将同名的过程组成一组，以便能用一条 drop procedure 语句一并删除它们。

同一应用程序中使用的过程经常用此方法编组。例如，可以创建一系列过程，名字为 orders;1、orders;2，等等。下面的语句将删除整个组：

```
drop proc orders
```

用向其名字后附加分号和数字的方法将过程分组后，它们就不能被单个删除。例如，以下语句是不允许的：

```
drop proc orders;2
```

要在已评估的配置中运行 Adaptive Server，必须禁止过程分组。此禁止确保每个存储过程都拥有唯一的对象标识符并可被单个删除。如果不允许过程分组，系统安全员必须重新设置配置参数 allow procedure grouping。有关信息，参见*系统管理指南*。

在 *create procedure* 中使用 *with recompile*

在 create procedure 语句中，可选子句 with recompile 正在 SQL 语句之前。它指示 Adaptive Server 不为该过程保存计划。每次执行过程时都要创建一个新计划。

如果没有 with recompile，则 Adaptive Server 存储其创建的执行计划。通常，此执行计划满足要求。

然而，如果提供给后续执行的数据或参数值发生变化，可能导致 Adaptive Server 创建一个与过程初次执行时创建的计划不同的执行计划。这种情况下，Adaptive Server 需要一个新的执行计划。

当您认为需要一个新计划时，在 create procedure 语句中使用 with recompile。有关详细信息，参见*参考手册*。

在 *execute* 中使用 *with recompile*

在 execute 语句中，可选子句 with recompile 在所有参数之后。它指示 Adaptive Server 编译一个用于后续执行的新计划。

执行过程时，如果您的数据已经改变很多，或者提供的参数是不规则的 — 即如果您有理由相信与过程存储在一起的计划可能不是其次此执行的最优计划，请使用 with recompile。

注意：如果在 create procedure 语句中使用 select*，则过程不会选取任何已添加到表的新列（即使使用 with recompile 选项来 execute）。您必须删除该过程并重新创建它。

过程中的嵌套过程

当一个存储过程或触发器调用另一存储过程或触发器时，就会发生嵌套。嵌套级别在被调用的过程或触发器开始执行时会递增，而在其执行结束时递减。超过 16 层的嵌套将导致过程失败。当前嵌套级别存储在 *@@nestlevel* 全局变量中。

可用过程名或变量名替换真正的过程名来调用其它过程。例如：

```
create procedure test1 @proc_name varchar(30)
    as exec @proc_name
```

使用存储过程的临时表

可创建并使用存储过程中的临时表，但临时表只在创建它的存储过程的持续期间内存在。过程完成后，Adaptive Server 将自动删除临时表。单个过程能够：

- 创建临时表
- 插入、更新或删除数据
- 在临时表上运行查询
- 调用引用临时表的其它过程

因为要创建引用临时表的过程，临时表必须存在，所以要遵循以下步骤：

1 使用 create table 语句或 select into 语句创建临时表。例如：

```
create table #tempstores
    (stor_id char(4), amount money)
```

2 创建访问临时表的过程（而非创建它的过程）。

```
create procedure inv_amounts as
    select stor_id, "Total Due" = sum(amount)
    from #tempstores
    group by stor_id
```

3 删除临时表：

```
drop table #tempstores
```

4 创建一个过程，该过程创建表并调用在步骤 2 中创建的过程：

```
create procedure inv_proc as
create table #tempstores
(stor_id char(4), amount money)
```

运行 inv_proc 过程时，它创建该表，但它只在该过程的执行期间存在。试着将值插入 #tempstores 表或运行 inv_amounts 过程：

```
insert #tempstores
select stor_id, sum(qty*(100-
discount)/100*price)
from salesdetail, titles
where salesdetail.title_id = titles.title_id
group by stor_id, salesdetail.title_id
exec inv_amounts
```

因为 #tempstores 表不再存在，所以不能。

也可在存储过程中使用 create table tempdb..*tablename*... 创建不带 # 前缀的临时表。过程完成后这些表不消失，所以它们可被独立过程引用。按以上步骤创建这些表。

在存储过程中设置选项

可以在存储过程内使用 set 命令选项。在过程的执行期间 set 选项始终有效，在过程结束时大多数选项返回到先前的设置。只有 dateformat、datefirst 和 language 选项不返回到先前的设置。

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。

存储过程的自变量

存储过程的自变量最多有 2048 个。

表达式、变量和存储过程自变量的长度

无论页有多大，传递到存储过程的表达式、变量和自变量的最大大小为 16384(16K) 字节。它可以是字符，也可以是二进制数据。无须使用 writetext 命令就可以将最大大小的变量、文字插入到文本列。

如果 Adaptive Server 已升级且先前版本使用较小的最大长度：

Adaptive Server 的早期版本中，存储过程的表达式、变量和自变量的最大大小为 255 字节。

因为页的最大大小增加，所以为使用此旧最大长度的 Adaptive Server 早期版本所写的脚本或存储过程现在可能会返回较大的字符串值。

因为值变大，所以 Adaptive Server 可能要截断字符串，或者如果字符串保存在另一变量中或被插入到列或字符串中，字符串可能溢出。

如果要修改现有表的列以增加字符列的长度，则必须修改使用这些列的数据的所有存储过程以反映新的长度。

```
select datalength(replicate("x", 500)),
    datalength("abcdefgh....255 byte long string.."
+
    "xxyyzz ... another 255 byte long string")
----------- -----------
255 255
```

创建存储过程后

创建存储过程后，描述该过程的**源文本**保存在 syscomments 系统表的 text 列中。*切勿从 syscomments 中删除此信息*；这样做会为 Adaptive Server 将来的升级带来一些问题。而应使用*参考手册*中说明的 sp_hidetext 对 syscomments 中的文本加密。有关详细信息，请参见第 4 页上的“编译对象”。

执行存储过程

存储过程可以延时执行，也可以远程执行。

延时执行过程

waitfor 命令可以在指定的时间延迟存储过程的执行或者延迟一个指定的时间段。

例如，半小时后执行过程 testproc：

```
begin
    waitfor delay "0:30:00"
    exec testproc
end
```

执行 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 的该连接。

远程执行过程

可以在本地 Adaptive Server 上执行远程 Adaptive Server 的过程。两个服务器都正确配置后，只要把服务器名作为标识符的一部分就可以执行远程 Adaptive Server 上的任何过程。例如，执行服务器 GATEWAY 上的过程 remoteproc：

```
exec gateway.remotedb.dbo.remoteproc
```

有关如何为远程执行过程而配置本地和远程 Adaptive Servers 的信息，参见*系统管理指南*。可以将批处理或含有远程过程的 execute 语句中的一个或多个值作为参数传递给远程过程。来自远程 Adaptive Server 的结果显示在本地终端上。

过程的返回状态可用于捕获和传输有关过程执行状态的信息消息。有关详细信息，请参见第 458 页上的“返回状态”。

警告！ 如果不启用组件集成服务，Adaptive Server 将不把远程过程调用 (RPC) 作为事务的一部分。因此，如果把 RPC 作为事务的一部分来执行然后回退该事务，则 Adaptive Server 不回退 RPC 导致的任何改变。如果启用了组件集成服务，则采用 set transactional rpc 和 set cis rpc handling 来使用事务的 RPC。有关 transactional rpc 和 cis rpc handling 选项的详细信息，参见*参考手册*中的 set 命令。

返回存储过程的信息

存储过程可返回下列类型的信息：

- 返回状态 — 表示存储过程是否成功完成。
- proc role 函数 — 检查过程是否被拥有 sa_role、sso_role 或 ss_oper 特权的用户执行。
- 返回参数 — 将参数值报告给调用者，然后调用者可以使用条件语句来检查返回的值。

返回状态和返回参数可以使存储过程模块化。由多个存储过程使用的一组 SQL 语句可以作为一个过程来创建，该过程将其执行状态或参数值返回给调用过程。例如，许多 Adaptive Server 系统过程执行同一个过程，以此来校验某些参数是否为有效的标识符。

远程过程调用（在远程 Adaptive Server 上运行的存储过程）也返回两种信息。如果执行语句的语法包含服务器、数据库、所有者名字和过程名，则下面的所有示例都可以远程执行。

返回状态

存储过程将报告一个**返回状态**，该返回状态指示它们是否成功完成，如果没有，将报告失败原因。当调用过程时，这个值可以保存在变量中，并在以后的 Transact-SQL 语句中使用。Adaptive Server 定义的用于表示失败的返回状态值的范围是 -1 到 -99 ；用户可以定义此范围外的自己的返回状态值

以下是一个批处理的例子，它使用 execute 语句的形式返回状态：

```
declare @status int
execute @status = byroyalty 50
select @status
```

byroyalty 过程的执行状态存储在变量 *@status* 中。“50”是根据 titleauthor 表中的 royaltyper 列提供的参数。该示例用 select 语句将值输出；以后的示例将在条件子句中使用该返回值。

保留的返回状态值

Adaptive Server 保留 0 来指示返回成功，而用 -1 到 -99 之间的负数指示失败的各种原因。目前，0 和从 -1 到 -14 的数已用于版本 12，如表 14-1 中所示：

表 *14-1*：保留的返回状态值

值	含义
0	过程执行时没有发生错误
-1	缺失对象
-2	数据类型错误
-3	进程被选作死锁牺牲品
-4	权限错误
-5	语法错误
-6	杂类用户错误
-7	资源错误，如空间不足
-8	非致命内部问题
-9	达到系统限制
-10	致命内部不一致性

值	含义
-11	致命内部不一致性
-12	表或索引损坏
-13	数据库损坏
-14	硬件错误

值 -15 至 -99 留作 Adaptive Server 将来使用。

如果在执行时发生了多个错误，将返回绝对值最高的状态。

用户生成的返回值

可通过向 return 语句添加参数的方法，在存储过程中生成自己的返回值。可以使用 0 到 -99 范围外的任意整数。下面示例中，如果书籍具有有效合同，则返回“1”，如果为其它任意情况则返回“2”：

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

例如：

```
checkcontract MC2222
(return status = 1)
```

下面的存储过程调用 checkcontract，并使用条件子句检查返回状态：

```
create proc get_au_stat @titleid tid
as
declare @retvalue int
execute @retvalue = checkcontract @titleid
if (@retvalue = 1)
   print "Contract is valid."
else
    print "There is not a valid contract."
```

以下是使用具有有效合同的书的 title_id 执行 get_au_stat 时产生的结果：

```
get_au_stat MC2222
Contract is valid
```

检查过程中的角色

如果存储过程执行系统管理任务或与安全有关的任务，则可能需要保证只有已被授予特定角色的用户才可执行它。过程执行时，proc_role 函数允许检查角色。如果该用户为指定角色则返回 1。角色名为 sa_role、sso_role，和 oper_role。

以下是在存储过程 test_proc 中使用 proc_role 的示例，要求调用者是系统管理员：

```
create proc test_proc
as
if (proc_role("sa_role") = 0)
begin
    print "You do not have the right role."
    return -1
end
else
    print "You have SA role."
    return 0
```

例如：

```
test_proc
You have SA role.
```

返回参数

存储过程可以把信息返回给调用者的另一种方法是*返回参数*。调用者可以使用条件语句检查返回的值。

当 create procedure 语句和 execute 语句都包含带有参数名字的 output 选项时，过程将值返回给调用者。调用者可以是 SQL 批处理或者是其它存储过程。返回值可用于批处理或调用过程中的其它语句。如果返回参数用于属于批处理的 execute 语句，则在执行批处理中的后续语句前，将输出带有标题的返回值。

此存储过程执行两个整数的乘法运算（第三个整数 *@result* 被定义为 output 参数）：

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

要使用 mathtutor 计算乘法问题，必须声明 *@result* 变量并将其包含在 execute 语句中。将 output 关键字添加到 execute 语句，显示返回参数的值。

```
declare @result int
exec mathtutor 5, 6, @result output
(return status = 0)

Return parameters:

-----------
         30
```

如果想通过提供三个整数来猜测答案并执行此过程，则可能无法看到相乘结果。过程中的 select 语句指派值但不输出：

```
mathtutor 5, 6, 32
(return status = 0)
```

output 参数的值必须作为变量而不是常量传递。此示例声明了变量 *@guess* 以存储传递到 mathtutor 的值，以在 *@result* 中使用。Adaptive Server 输出返回参数：

```
declare @guess int
select @guess = 32
exec mathtutor 5, 6,
@result = @guess output
(1 row affected)
(return status = 0)

Return parameters:

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

不管返回参数值更改与否，返回参数的值始终要报告。注意：

- 上例中，output 参数 *@result 必须*以“*@parameter* = *@variable*”的形式传递。如果它不是传递的参数最后一个参数，则后续参数必须以“*@parameter* = *value*”的形式传递。
- *@result* 不必已在调用批处理中声明；它是要传递到 mathtutor 的参数的名称。
- 虽然 *@result* 的更改值在 execute 语句中指派的变量（此例为 *@guess*）中返回给调用者 , 但它将在其本身的标题（*@result*）下显示。

要在 execute 语句后的条件子句中使用 *@guess* 的初始值，过程调用时必须把它用另一个变量名存储。以下示例通过在存储过程执行期间用 *@store* 持有变量值，并在条件子句中使用 *@guess* 的“新”返回值，说明了上述最后两项：

```
declare @guess int
declare @store int
select @guess = 32
select @store = @guess
execute mathtutor 5, 6,
@result = @guess output
select Your_answer = @store,
Right_answer = @guess
if @guess = @store
    print "Bingo!"
else
    print "Wrong, wrong, wrong!"
(1 row affected)
(1 row affected)
(return status = 0)

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

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

Wrong, wrong, wrong!
```

此存储过程检查并决定新的图书销售额是否导致作者的版税比率发生变化（*@pc* 参数定义为 output 参数）：

```
create proc roy_check @title tid, @newsales int,
        @pc int output
as
declare @newtotal int
select @newtotal = (select titles.total_sales +
@newsales
from titles where title_id = @title)
select @pc = royalty from  roysched
   where @newtotal  >= roysched.lorange and
          @newtotal < roysched.hirange
   and roysched.title_id = @title
```

将值指派给变量 *percent* 后，以下 SQL 批处理调用 roy_check。执行批处理时，返回的参数输出到下面的语句之前：

```
declare @percent int
select @percent = 10
execute roy_check "BU1032", 1050, @pc = @percent
output
select Percent = @percent
go
(1 row affected)
(return status = 0)

@pc
-----------
         12
Percent
-----------
         12

(1 row affected)
```

下列存储过程调用 roy_check 并在条件子句中使用 *percent* 的返回值：

```
create proc newsales @title tid, @newsales int
as
declare @percent int
declare @stor_pc int
select @percent = (select royalty from roysched,
titles
        where roysched.title_id = @title
        and total_sales >= roysched.lorange
        and total_sales < roysched.hirange
        and roysched.title_id = titles.title_id)
select @stor_pc = @percent
execute roy_check @title, @newsales, @pc = @percent
  output
if
  @stor_pc != @percent
begin
  print "Royalty is changed."
  select Percent = @percent
end
else
  print "Royalty is the same."
```

如果使用与早期批处理中相同的参数执行此存储过程，则：

```
execute newsales "BU1032", 1050
Royalty is changed
Percent
-----------
         12

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

在上两个调用 roy_check 的示例中，*@pc* 是传递给 roy_check 的参数，而 *@percent* 是含有输出的变量。newsales 执行 roy_check 时，*@percent* 中的返回值可能会根据传递的其它参数发生变化。要比较 *percent* 的返回值和 *@pc* 的初始值，必须在另一变量中存储初始值。上例将值保存在 *stor_pc* 中。

传递参数值

按以下格式传递参数值：

@parameter = *@variable*

不能传递常量；要“接收”返回值，必须有一变量名。参数可以是除 text 或 image 之外的任意 Adaptive Server 数据类型。

注意：如果存储过程要求多个参数，则在 execute 语句中最后传递返回值参数，或以 *@parameter* = *value* 形式传递所有的后续参数。

---

output 关键字

一个存储过程可返回多个值；每个值都必须在存储过程和调用语句中被定义为 output 变量。output 关键字可以缩写为 out。

exec myproc @a = @myvara out, @b = @myvarb out

如果执行过程时指定 output，而在存储过程中没有用 output 定义参数，则将得到错误信息。调用一个包含返回值说明而不要求用 output 返回值的过程是允许的。然而，将得不到返回值。存储过程的编写者控制用户可访问的信息，而用户控制他们的变量。

与存储过程关联的限制

以下是一些创建存储过程的附加限制：

- 不能将 create procedure 语句与同一批处理中的其它语句合并。
- create procedure定义本身能包含任何数目与类型的SQL语句，除 use 和以下 create 语句外：
  - create view
  - create default
  - create rule
  - create trigger
  - create procedure
- 可以在过程内创建其它数据库对象。可以引用本过程中创建的对象，只要它是在引用之前创建的。对象的 create 语句必须位于过程中语句的实际顺序的最前面。
- 在存储过程内，不能在创建某个对象并将其删除之后又以相同的名字创建新的对象。
- Adaptive Server是在执行而非编译存储过程时创建在该过程中定义的对象的。
- 如果执行一个调用其它过程的过程，被调用过程可以访问第一个过程创建的对象。
- 可以在过程中引用临时表。
- 如果在过程中创建带 # 前缀的临时表，则该临时表只用于该过程 — 退出过程时它将消失。用 create table tempdb..*tablename* 创建的临时表不会消失，除非已将它们显式删除。
- 一个存储过程最多有 255 个参数。
- 过程中局部与全局变量的最大数仅受可用内存的限制。

限定过程内的名字

在存储过程中，如果其他用户要使用该存储过程，则同 create table 和 dbcc 一起使用的对象名必须用其所有者名来**限定**。在存储过程中同其它语句（如 select 和 insert）一起使用的对象名无需限定，因为在过程编译时该名称就已被解析。

例如，如果用户“mary”（拥有表 marytab）想要其他用户执行使用该表的过程，则当该表与 select 或 insert 一起使用时，应当限定该表的名字。这是因为对象名在过程运行时被解析。如果没有限定 marytab，且用户“john”试图执行该过程，Adaptive Server 将查找 John 拥有的 marytab 表。

重命名存储过程

用 sp_rename 重命名存储过程。
此命令的语法为：

sp_rename *objname*, *newname*

例如，要将 showall 重新命名为 countall：

```
sp_rename showall, countall
```

新名必须遵循标识符的规则。只能修改您所拥有的存储过程的名字。数据库所有者能修改任意用户的存储过程的名称。该存储过程必须位于当前数据库中。

过程引用的重命名对象

如果重命名了过程所引用的任何对象，则必须删除该过程，然后重新创建它。所引用表或视图的名字已被修改的存储过程可能在一段时间内工作很好。事实上，它只工作到 Adaptive Server 编译它为止。进行重新编译的原因有多种，并且不会通知用户。

用 sp_depends 获取被过程引用的对象的报告。

使用存储过程作为安全机制

可以把存储过程作为安全机制来使用，以控制对表中信息的访问和执行数据修改的能力。例如，可以拒绝其他用户对您拥有的表使用 select 命令，并创建只允许这些用户查看某几行或某几列的存储过程。也可使用存储过程来限制 update、delete 或 insert 语句。

拥有存储过程的用户必须同时拥有过程中使用的表或视图。如果没有授予系统管理员对其他用户表的权限，即使是系统管理员也不能创建在这些表上执行操作的存储过程。

有关授予或撤消存储过程或其它数据库对象权限的信息，参见*系统管理指南*。

删除存储过程

用 drop procedure 删除存储过程。此命令的语法为：

drop proc[edure] [*owner*.]*procedure_name*
[, [*owner*.]*procedure_name*] ...

如果其它存储过程调用已被删除的存储过程，则 Adaptive Server 将显示错误信息。然而，如果定义了一个同名的新过程以代替被删除的过程，则引用初始过程的其它过程可以成功调用它。

过程组（即拥有相同名称，但却具有不同数字后缀的多个过程）可以用单个 drop procedure 语句删除。过程分组后，就不能单独删除组中的过程。

系统过程

系统过程是：

- 从系统表检索信息的快捷方式
- 执行数据库管理和包括更新系统表在内的其它任务的机制

大多数情况下，系统表*仅*通过存储过程更新。通过修改配置变量，系统管理员可以直接更新系统表。有关详细信息，参见*系统管理指南*。

系统过程名以“sp_”开头。它们由 sybsystemprocs 数据库的 installmaster 脚本在 Adaptive Server 安装过程中创建。

执行系统过程

可从任意数据库运行系统过程。如果系统过程是在从 sybsystemprocs 数据库之外的数据库执行，则对系统表的任何引用都要映射到开始运行过程的数据库。例如，如果 pubs2 的数据库所有者在 pubs2 中运行 sp_adduser，则新用户将添加到 pubs2..sysusers。要在特定数据库上运行系统过程，须用 use 命令打开数据库并执行该过程，或用数据库名限定过程名。

如果系统过程的参数是对象名，且对象名由数据库名或所有者名限定，则整个名称必须用单引号或双引号引起来。

系统过程的权限

既然系统过程位于 sybsystemprocs 数据库中，其权限也设在那里。某些系统过程只能由数据库所有者来运行。这些过程确保执行过程的用户是他们在其中执行该过程的数据库的所有者。

其它系统过程可由任何已获取其 execute 权限的用户执行，但此权限必须在 sybsystemprocs 数据库中授予。这种情况有两种后果：

- 用户要么具有在所有数据库中执行某系统过程的权限，要么在任何数据库中都不具有这一权限。
- 用户数据库的所有者不能从自己的数据库内部直接控制该系统过程的权限。

系统过程类型

系统过程可按功能（如审计、安全管理、数据定义等等）进行分组。以下部分列出了系统过程的类型。

有关这些系统过程的完整信息，参见*参考手册*。

用于审计的系统过程

这些系统过程用于：

- 控制审计设置
- 创建并管理持有审计队列的表

此类过程有：

sp_addauditrecord	sp_addauditable	sp_audit	sp_displayaudit

有关审计的详细信息，参见*安全性管理指南*。

用于安全性管理的系统过程

这些系统过程用于：

- 在 Adaptive Server 上添加、删除并报告登录名
- 添加、删除并报告数据库中的用户、组、角色和别名
- 修改口令和缺省数据库
- 添加、删除和报告可以访问当前 Adaptive Server 的远程服务器
- 从远程服务器添加可以访问当前 Adaptive Server 的用户名

该类过程有：

sp_activeroles	sp_changegroup	sp_droplogin	sp_locklogin
sp_addalias	sp_displaylogin	sp_dropuser	sp_modifylogin
sp_addgroup	sp_displayroles	sp_helpgroup	sp_password
sp_addlogin	sp_dropalias	sp_helprotect	sp_role
sp_adduser	sp_dropgroup	sp_helpuser	

Windows NT 还可使用其它系统过程。该类过程有：

sp_grantlogin	sp_logininfo	sp_revokelogin
sp_loginconfig	sp_processmail	

用于远程服务器的系统过程

这些系统过程用于：

- 添加、删除和报告可以访问当前 Adaptive Server 的远程服务器
- 从远程服务器添加可以访问当前 Adaptive Server 的用户名

该类过程有：

sp_addremotelogin	sp_dropserver	sp_remoteoption
sp_addserver	sp_helpremotelogin	sp_serveroption
sp_dropremotelogin	sp_helpserver	

如果安装了组件集成服务，则还可以使用其它系统过程。该类过程有：

sp_addexternlogin	sp_dropexternlogin	sp_passthru
sp_addobjectdef	sp_dropobjectdef	sp_remotesql
sp_autoconnect	sp_helpexternlogin	
sp_defaultloc	sp_helpobjectdef	

用于管理数据库的系统过程

这些系统过程用于：

- 改变数据库的所有者
- 显示和修改数据库选项

该类过程有：

sp_changedbowner	sp_dbremap	sp_renamedb
sp_dboption	sp_helpdb	

用于数据定义和数据库对象的系统过程

这些系统过程用于：

- 绑定和解除绑定规则和缺省值
- 添加、删除和报告主键、外键及公用键
- 添加、删除和报告用户定义的数据类型
- 重新定义数据库对象和用户定义的数据类型
- 重新优化存储过程和触发器
- 将对象绑定到数据高速缓存或将其解除绑定
- 报告数据库对象、用户定义的数据类型以及数据库对象、数据库、索引及表和索引使用空间之间的依赖性
- 获取有关命令语法的帮助

该类过程有：

sp_addextendedproc	sp_dropkey	sp_helptext
sp_addtype	sp_droptype	sp_hidetext
sp_bindcache	sp_estspace	sp_poolconfig
sp_bindefault	sp_foreignkey	sp_primarykey
sp_bindrule	sp_freedll	sp_procxmode
sp_cacheconfig	sp_help	sp_recompile
sp_cachestrategy	sp_helpartition	sp_rename
sp_chgattribute	sp_helpcache	sp_spaceused
sp_checksource	sp_helpconstraint	sp_unbindcache
sp_commonkey	sp_helpextendedproc	sp_unbindcache_all
sp_cursorinfo	sp_helpindex	sp_unbindefault
sp_depends	sp_helpjoins	sp_unbindrule
sp_dropextendedproc	sp_helpkey	

用于用户定义的消息的系统过程

这些系统过程用于：

- 将用户定义的消息添加到用户数据库的 sysusermessages 表中
- 从 sysusermessages 删除用户定义的消息
- 从 master 数据库的 sysusermessages 或 sysmessages 中检索消息，用于 print 和 raiserror 语句

该类过程有：

sp_addmessage	sp_bindmsg	sp_getmessage
sp_altermessage	sp_dropmessage	sp_unbindmsg

语言有关的系统过程

这些系统过程用于：

- 管理字符设置
- 添加和删除替代语言

该类过程有：

sp_addlanguage	sp_helplanguage	sp_setlangalias
sp_checknames	sp_helpsort	
sp_droplanguage	sp_indsuspect	

用于设备管理的系统过程

这些系统过程用于：

- 添加、删除和报告数据库及转储设备
- 添加、删除和扩展数据库段

该类过程有：

sp_addsegment	sp_dropsegment	sp_helpsegment
sp_addumpdevice	sp_extendsegment	sp_logdevice
sp_diskdefault	sp_helpdevice	sp_logiosize
sp_dropdevice	sp_helplog	sp_placeobject

用于备份和恢复的系统过程

这些系统过程用于：

- 创建和管理阈值
- 显示和设置恢复故障隔离模式
- 与 Backup Server™ 通信

该类过程有：

sp_addthreshold	sp_helpthreshold	sp_setsuspect_granularity
sp_dropthreshold	sp_listsuspect_db	sp_setsuspect_threshold
sp_forceonline_db	sp_listsuspect_page	sp_thresholdaction
sp_forceonline_page	sp_modifythreshold	sp_volchanged

用于配置与调优的系统过程

这些系统过程用于：

- 修改和报告配置参数
- 报告系统使用情况
- 报告性能行为
- 监控 Adaptive Server 活动

该类过程有：

sp_clearstats	sp_monitor
sp_configure	sp_monitorconfig
sp_countmetadata	sp_reportstats

sp_displaylevel	sp_showplan
sp_helpconfig	sp_sysmon

用于系统管理的系统过程

这些系统过程用于：

- 报告锁和当前运行进程的用户
- 变更表或数据库的锁升级阈值
- 创建和报告资源限制及时间范围
- 创建和报告执行类及引擎组
- 计划 dbccdb 数据库的布局

该类过程有：

sp_addengine	sp_drop_resource_limit	sp_setpsexe
sp_addexeclass	sp_drop_time_range	sp_showcontrolinfo
sp_add_resource_limit	sp_familylock	sp_showexeclass
sp_add_time_range	sp_help_resource_limit	sp_showpsexe
sp_bindexeclass	sp_lock	sp_unbindexeclass
sp_clearpsexe	sp_modify_resource_limit	sp_setpsexe
sp_dropexeclass	sp_modify_time_range	sp_who
sp_dropengine	sp_setpglockpromote	
sp_dropglockpromote	sp_plan_dbccdb	

有关完成这些管理任务的系统过程的详细信息，参见*系统管理指南*。

用于升级的系统过程

这些系统过程用于：

- 查找保留字
- 重新映射对象
- 检查对象的查询处理模式

该类过程有：

sp_checkreswords	sp_remap	sp_procqmode

Sybase 提供的其它过程

Sybase 提供分类存储过程、系统扩展存储过程（系统 ESP）和 dbcc 过程。

分类存储过程

分类存储过程是在表格形式的系统表中检索信息的系统过程。分类存储过程有：

sp_column_privileges	sp_pkeys	sp_stored_procedures
sp_columns	sp_server_info	sp_table_privileges
sp_databases	sp_special_columns	sp_tables
sp_datatype_info	sp_sproc_columns	
sp_fkeys	sp_statistics	

系统扩展存储过程

扩展存储过程 (ESP) 从 Adaptive Server 中调用过程语言函数。系统扩展存储过程是在安装时通过 installmaster 创建的，它位于 sybsystemprocs 数据库中并由系统管理员所拥有。它们可在任意数据库中运行且名字以“xp_”开头。系统扩展存储过程有：

xp_cmdshell	xp_findnextmsg	xp_sendmai
xp_deletemail	xp_logevent	xp_startmai
xp_enumgroups	xp_readmail	xp_stopmai

有关创建和使用 ESP 的详细信息，参见第 15 章“使用扩展存储过程”。

dbcc 过程

由 *installdbccdb* 创建的 dbcc 过程是存储过程，用来生成有关 dbcc checkstorage 产生的信息的报告。这些过程驻留于 dbccdb 数据库或替代数据库 dbccalt 中。dbcc 过程有：

sp_dbcc_alterws	sp_dbcc_differentialreport	sp_dbcc_statisticsreport
sp_dbcc_configreport	sp_dbcc_evaluatedb	sp_dbcc_summaryreport
sp_dbcc_createws	sp_dbcc_faultreport	sp_dbcc_updateconfig
sp_dbcc_deletedb	sp_dbcc_fullreport	
sp_dbcc_deletehistory	sp_dbcc_runcheck	

获取有关存储过程的信息

多个系统过程在系统表中提供关于存储过程的信息。

在第 467 页上的“系统过程”中对系统过程作了简短讨论。有关系统过程的完整信息，参见*参考手册*。

用 *sp_help* 获取报告

可以使用 sp_help 来获取关于存储过程的报告。例如，可以获取存储过程 byroyalty 的信息，该存储过程是 pubs2 数据库的一部分，如下：

```
sp_help byroyalty
Name        Owner    type
--------    ------   ----------------
byroyalty   dbo      stored procedure

Data_located_on_segment        When_created
----------------------------    --------------------
not applicable                   Jul 10, 1997 4:57PM

Parameter_name Type    Length   Prec  Scale  Param_order
-------------- ------  ------   ----  -----  -----------
@percentage    int          4   NULL   NULL            1

(return status = 0)
```

在使用 sybsystemprocs 数据库时执行 sp_help，可以获取关于系统过程的帮助信息。

用 *sp_helptext* 查看过程的源文本

要显示 create procedure 语句的源文本，请执行 sp_helptext：

```
sp_helptext byroyalty
# Lines of Text
---------------
              1

(1 row affected)

text
--------------------------------------------------
create procedure byroyalty @percentage int
as
select au_id from titleauthor
where titleauthor.royaltyper = @percentage

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

使用 sybsystemprocs 数据库时执行 sp_helptext 可查看系统过程的源文本。

如果存储过程的源文本已使用 sp_hidetext 加密，Adaptive Server 显示一条提示用户文本被隐藏的消息。有关隐藏源文本的详细信息，参见*参考手册*中的 sp_hidetext。

用 *sp_depends* 标识相关对象

sp_depends 列出引用指定对象的所有存储过程或它所依赖的所有过程。

例如，以下命令列出了用户定义的存储过程 byroyalty 引用的所有对象：

```
sp_depends byroyalty
Things the object references in the current
database.
object           type       updated      selected
---------------- ----------- ---------   --------
dbo.titleauthor user table  no      no
```

```
(return status = 0)
```

下列语句使用 sp_depends 列出引用表 titleauthor 的所有对象：

```
sp_depends titleauthor
Things inside the current database that reference
the object.

object          type
--------------  ------------------
dbo.byroyalty   stored procedure
dbo.titleview   view

(return status = 0)
```

如果重命名了过程所引用的任何对象，则必须删除并重新创建该过程。

用 *sp_helprotect* 标识权限

sp_helprotect 报告对存储过程（或其它任何数据库对象）的权限。例如：

```
sp_helprotect byroyalty
grantor    grantee    type  action    object    column  grantable
---------  ---------  ----  --------- -------   -----   ---------
dbo        public     Grant Execute   byroyalty All     FALSE

(return status = 0)
```

第 15 章

使用扩展存储过程

扩展存储过程 (ESP) 提供了一种机制，可以从 Adaptive Server 内部调用外部过程语言函数。调用 ESP 所使用的语法与调用存储过程时使用的相同。不同之处在于 ESP 执行过程语言代码，而不是 Transact-SQL 语句。

本章讨论如下内容：

主题	页
概述	479
为 ESP 创建函数	484
注册 ESP	495
执行 ESP	497
系统 ESP	498
获取有关 ESP 的信息	499
ESP 例外和消息	499
手工启动 XP Server	500

概述

扩展存储过程提供了一种从 Adaptive Server 内部动态加载和执行外部过程语言函数的方法。每个 ESP 都与一个相应的函数相关联，当从 Adaptive Serve 调用 ESP 时将执行该函数。

ESP 允许 Adaptive Server 在 Adaptive Server 外执行任务，以响应在 Adaptive Server 内部发生的事件。例如，可创建一个销售证券的 ESP 函数（在 Adaptive Server 外执行的任务）。调用此函数以响应当证券价格达到某一特定值时引发的触发器。或者，可以创建一个发送 e-mail 通知或在网络范围内广播的 ESP 函数，以响应在相关数据库系统内部发生的事件。

对 ESP 来说，“过程语言”是一种程序语言，它能调用 C 语言函数并能使用 C 语言数据类型。

当某个函数作为 ESP 注册到数据库后，可以象调用存储过程一样从 isql、触发器、其它存储过程或客户端应用程序调用它。

ESP 可以：

- 接受输入参数
- 返回一个状态值，指示成功或失败以及失败的原因。
- 返回输出参数值
- 返回结果集

Adaptive Server 提供了一些系统 ESP。例如系统 ESP，xp_cmdshell，它从 Adaptive Server 内部执行操作系统命令。也可使用 Open Server 应用程序编程接口（API）的子集编写自己的 ESP。

XP Server

扩展存储过程由名为 XP Server 的 Open Server 应用程序实现，该应用程序与 Adaptive Server 运行在同一台机器上。Adaptive Server 和 XP Server 通过远程过程调用 (RPC) 进行通信。在独立的进程中运行 ESP 可防止 Adaptive Server 因 ESP 代码错误而导致结果错误。使用 ESP 而不使用 RPC 的好处在于，ESP 在 Adaptive Server 中的运行方式与存储过程的运行方式相同，不需要使用 Open Server 即可运行 ESP。

安装 Adaptive Server 时自动安装 XP Server。

Adaptive Server 要执行 ESP 必须运行 XP Server。Adaptive Server 在 ESP 首次被调用时启动 XP Server，当 Adaptive Server 自身退出时关闭 XP Server。

在 Windows NT 上，如果 start mail session 配置参数设置为 1，XP Server 将在 Adaptive Server 启动时自动启动。

动态链接库支持

包含 ESP 代码的过程函数编译并链接到动态链接库 (DLL)，它们被装载到 XP Server 的内存中以响应 ESP 的执行要求。动态库始终保留在内存中，直到出现下列情况之一：

- XP Server 退出
- sp_freedll 系统过程被调用
- 使用 sp_configure 设置 esp unload dll 配置参数。

Open Server API

Adaptive Server 使用 Open Server API，它允许用户运行 Adaptive Server 提供的系统 ESP。用户也可用 Open Server API 实现自己的 ESP。

表 15-1 列出了 ESP 开发所要求的 Open Server 例程。有关这些例程的完整文档，参见 *Open Server Server-Library/C 参考手册*。

表 *15-1*：用于 *ESP* 支持的 *Open Server* 例程

函数	目的
srv_bind	描述和绑定参数的程序自变量。
srv_descfmt	描述参数
srv_numparams	根据 ESP 客户端请求返回参数个数。
srv_senddone	发送结果完成消息。
srv_sendinfo	发送消息。
srv_sendstatus	发送状态值
srv_xferdata	发送及接收参数或数据。
srv_yield	挂起当前线程的执行并允许其它线程执行。

创建和使用 ESP 的示例

当 ESP 函数编写、编译完成并链接到 DLL 后，可使用 create procedure 命令的 as external name 子句为函数创建一个 ESP：

create procedure *procedure_name* [*parameter_list*]
  as external name *dll_name*

procedure_name 是 ESP 的名称，它必须与其在 DLL 中的实施函数名相同。ESP 是数据库对象，其名称必须遵守标识符规则。

dll_name 是 DLL 的名称，实施函数存储在其中。

下列语句将创建一个名为 getmsgs 的 ESP，它在 *msgs.dll* 中。getmsgsESP 不接收参数。本示例用于 Windows NT Adaptive Server:

```
create procedure getmsgs
as external name "msgs.dll"
```

在 Solaris Adaptive Server 上，其语句是：

```
create procedure getmsgs
as external name "msgs.so"
```

这反映了 Solaris 的命名约定。

下列语句将创建一个名为 getonemsg *的 ESP*，它也在 *msgs.dll* 中。getmsgsESP 将消息号作为唯一参数。

```
create procedure getonemsg @msg int
as external name "msgs.dll"
```

在表 15-2 中总结了针对特定平台的 DLL 扩展命名约定。

表 *15-2*：*DLL* 扩展命名约定

平台	DLL 扩展
Digital UNIX	*.so*
HP 9000/800 HP-UX	*.sl*
Sun Solaris	*.so*
Windows NT	*.dll*

Adaptive Server 创建了一个 ESP 后，将过程名称存储在 sysobjects 系统表中，将对象类型“XP”和包含 ESP 函数的 DLL 名称存储在系统表 syscomments 中的 text 列。

执行 ESP 与执行用户定义的存储过程或系统过程一样。可使用关键字 execute 和存储过程的名称，或仅给出过程名称，只要它是自己提交到 Adaptive Server 的或是批处理中的第一条语句即可。例如，可以下列任一方式执行 getmsgs：

```
getmsgs
execute getmsgs
exec getmsgs
```

可以下列任一方式执行 getonemsg：

```
getonemsg 20
getonemsg @msg=20
execute getonemsg 20
execute getonemsg @msg=20
exec getonemsg 20
exec getonemsg @msg=20
```

ESP 和权限

在 ESP 中，您可象在通常的存储过程中一样授予和撤销权限。

除了通常的 Adaptive Server 安全性以外，可使用 xp_cmdshell context 配置参数来限制 xp_cmdshell 具有系统管理员权限的用户的执行权限。使用此配置参数可防止普通用户使用 xp_cmdshell 来执行操作系统命令，普通用户将无权直接从命令行执行。 xp_cmdshell 配置参数的行为是平台相关的。

缺省情况下，用户必须使用 sa_role 来执行 xp_cmdshell。 要授权其他用户使用 xp_cmdshell，可使用 grant 命令。可用 revoke 撤销权限。grant 或 revoke 权限都适用，而与 xp_cmdshell 环境设为 0 或 1 无关。

ESP 和性能

由于 Adaptive Server 和 XP Server 驻留在同一台机器上，当执行某一函数消耗大量资源时它们将影响彼此间的性能。

可使用 sp_configure 设置两个参数，esp execution priority 和 esp unload dll，通过设置 ESP 执行的优先级和释放 XP Server 内存来控制 XP Server 对 Adaptive Server 产生的影响。

设置优先级

使用 esp execution priority 将 XP Server 线程的优先级设置高一些，以便 Open Server 调度程序在运行队列中的其它线程之前运行它，或设置低一些，以便调度程序仅在没有其它线程运行时才运行 XP Server。esp execution priority 的缺省值是 8，可将其设置为 0 到 15 之间的任意值。有关调度 Open Server 线程的详细信息，参见 *Open Server Server-Library/C 参考手册*中关于多线程编程的讨论。

释放内存

当 ESP 装载 DLL 的请求结束后，通过将其从 XP Server 的内存中卸载，可使 XP Server 使用的内存降到最低。为此可设置 esp unload dll，以使ESP执行结束后DLL自动卸载。如果未设置esp unload dll，可利用 sp_freedll 显式释放 DLL。

不能卸载支持系统 ESP 的 DLL。

为 ESP 创建函数

对实现 ESP 的函数内容没有限制。唯一的要求是要以过程化编程语言编写，该语言应能够：

- 调用 C 语言函数
- 使用 C 语言数据类型
- 与 Open Server API 相链接

通过使用 Open Client API，ESP 函数可向 Adaptive Server（可以是最初调用它的，也可以是其它 Adaptive Server）发送请求。

但有一个例外，ESP 函数不应调用 Windows NT 上的 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。

用于 ESP 开发的文件

ESP 开发所需的头文件在 *$SYBASE/OCS-12_0/include* 中。为了能在源文件中找到这些文件，应在源代码应包含以下内容：

- *ospublic.h*
- *oserror.h*

Open Server 库在 *$SYBASE/lib* 中。

示例程序的源代码（如第 486 页上的“ESP 函数示例”所示）在 *$SYBASE/$SYBASE_ASE/sample/esp* 中。

Open Server 数据结构

有三种数据结构对编写 ESP 函数非常有用：

- SRV_PROC
- CS_SERVERMSG
- CS_DATAFMT

SRV_PROC

所有的 ESP 函数都被编码为能够接收单个参数及指向 SRV_PROC 结构的指针。SRV_PROC 结构在函数及其调用过程之间传递信息。ESP 开发人员无法直接操纵此结构。

ESP 函数将 SRV_PROC 指针传递到 Open Server 例程，该例程获得参数类型和数据并返回输出参数、状态代码和结果集。

CS_SERVERMSG

利用 *srv_sendinfo* 例程，Open Server 使用 CS_SERVERMSG 结构向客户端发送错误消息。有关 CS_SERVERMSG 的信息，参见 *Open Server Server-Library/C 参考手册*。

CS_DATAFMT

Open Server 使用 CS_DATAFMT 结构来描述数据值和程序自变量。

Open Server 返回代码

Open Server 函数返回 CS_RETCODE 类型的代码。ESP 函数最常见的 CS_RETCODE 值有：

- CS_SUCCEED
- CS_FAIL

简单 ESP 函数概述

考虑到与 Open Server API 的相互作用，一个 ESP 函数应有以下基本结构：

1. 获得参数个数。
2. 获得输入 / 输出参数值并将其绑定到局部自变量。
3. 使用输入参数值执行处理并将结果存储到局部自变量中。
4. 用相应的值初始化所有输出参数，将其与局部自变量绑定并传送到客户端。
5. 使用 srv_sendinfo() 将返回行发送到客户端。

6 使用 srv_sendinfo() 将状态发送到客户端。

7 使用 srv_senddone() 通知客户端处理已完成。

8 如果有错误情况，则使用 srv_sendinfo() 将错误消息发送到客户端。

有关 Open Server 例程的文档，参见 *Open Server Server-Library/C 参考手册*。

多线程

由于 Open Server 目前是非抢先式的，所以所有运行在同一服务器上的 ESP 必须互相屈从，使用 Open Server srv_yield() 例程将其 XP Server 线程挂起并允许其它线程执行。

有关详细信息，参见 *Open Server Server-Library/C 参考手册*中的多线程编程章节。

ESP 函数示例

xp_echo.c 有一个 ESP，它接收用户提供的输入参数并将其回应给 ESP 客户端，它是调用 ESP 的进程。此示例包括 xp_message 函数（该函数发送消息和状态）以及 xp_echo 函数（该函数处理输入参数并执行回应）。可使用此示例作为建立自己的 ESP 函数的模板。源代码位于 *$SYBASE/$SYBASE_ASE/sample/esp* 中。

```
/*
** xp_echo.c
**
**      Description:
**              The following sample program is generic in
**              nature. It echoes an input string which is
**              passed as the first parameter to the xp_echo
**              ESP. This string is retrieved into a buffer
**              and then sent back (echoed) to the ESP client.
*/
#include <string.h>
#include <stdlib.h>
#include <malloc.h>
/* Required Open Server include files.*/
#include <ospublic.h>
#include <oserror.h>
/*
** Constant defining the length of the buffer that receives the
```

```
** input string. All of the Adaptive Server parameters related
** to ESP may not exceed 255 char long.
*/
#define ECHO_BUF_LEN    255
/*
** Function:
**      xp_message
**      Purpose: Sends information, status and completion of the
**      command to the server.
** Input:
**      SRV_PROC *
**      char *  a message string.
** Output:
**      void
*/
void xp_message
(
        SRV_PROC *srvproc,  /* Pointer to Open Server thread
                             control structure */
        char     *message_string /* Input message string */
)
{
        /*
        ** Declare a variable that will contain information
        ** about the message being sent to the SQL client.
        */
        CS_SERVERMSG    *errmsgp;
        /*
        ** Allocate memory for this variable.
        */
        errmsgp = (CS_SERVERMSG *) malloc((CS_INT
           sizeof(CS_SERVERMSG));
        if (errmsgp == NULL)
                return;
        /*
        ** clean the structure */
        */
        memset(errmsgp,(CS_INT)0,(CS_INT) sizeof(CS_SERVERMSG));
        /*
        ** Put you rnumber in as the message number.
        */
        errmsgp->msgnumber = 25000;
        errmsgp->state = 0;
        /*
        **The message is fatal.
        */
```

```
        errmsgp->severity = SRV_FATAL_SERVER;
        /*
        ** Letís copy the string over.
        */
        errmsgp->textlen = strlen(message_string);
        if (errmsgp->textlen >= CS_MAX_MSG )
                return;
        strncpy(errmsgp->text, message_string, errmsgp->textlen);
        errmsgp->status = (CS_FIRST_CHUNK | CS_LAST_CHUNK);
        srv_sendinfo(srvproc, errmsgp, CS_TRAN_UNDEFINED);
        /* Send the status to the client. */
        srv_sendstatus(srvproc, 1);
        /*
        ** A SRV_DONE_MORE instead of a SRV_DONE_FINAL must
        ** complete the result set of an Extended Stored
        ** Procedure.
        */
        srv_senddone(srvproc, SRV_DONE_MORE, 0, 0);
        free(errmsgp);
}

/*
** Function: xp_echo
** Purpose:
**      Given an input string, this string is echoed as an output
**      string to the corresponding SQL (ESP) client.
** Input:
**      SRV_PROC *
** Output
**      SUCCESS or FAILURE
*/
CS_RETCODE xp_echo
(
        SRV_PROC        *srvproc
)
{
        CS_INT          paramnum;  /* number of parameters */
        CS_CHAR         echo_str_buf[ECHO_BUF_LEN + 1];
                               /* buffer to hold input string */
        CS_RETCODE      result = CS_SUCCEED;
        CS_DATAFMT      paramfmt; /* input/output param format */
        CS_INT          len;      /* Length of input param */
        CS_SMALLINT     outlen;
        /*
        ** Get number of input parameters.*/
        */
        srv_numparams(srvproc, &paramnum);
```

```
/*
** Only one parameter is expected.*/
*/
if (paramnum != 1)
{
        /*
        ** Send a usage error message.*/
        */
        xp_message(srvproc, "Invalid number of
         parameters");
        result = CS_FAIL;
}
else
{
        /*
        ** Perform initializations.
        */
        outlen = CS_GOODDATA;
        memset(&paramfmt, (CS_INT)0,
          (CS_INT)sizeof(CS_DATAFMT));
        /*
        ** We are receiving data through an ESP as the
        ** first parameter. So describe this expected
        ** parameter.
        */
        if ((result == CS_SUCCEED) &&
          srv_descfmt(srvproc, CS_GET
          SRV_RPCDATA, 1, &paramfmt) != CS_SUCCEED)
        {
                result = CS_FAIL;
        }
        /*
        ** Describe and bind the buffer to receive the
        ** parameter.
        */
        if ((result == CS_SUCCEED) &&
          (srv_bind(srvproc, CS_GET, SRV_RPCDATA,
           1, &paramfmt,(CS_BYTE *) echo_str_buf,
           &len, &outlen) != CS_SUCCEED))
        {
                result = CS_FAIL;
        }
        /* Receive the expected data.*/
        if ((result == CS_SUCCEED) &&
          srv_xferdata(srvproc,CS_GET,SRV_RPCDATA)
          != CS_SUCCEED)
```

```
{
        result = CS_FAIL;
}
/*
** Now we have the input info and are ready to
** send the output info.
*/
if (result == CS_SUCCEED)
{
        /*
        ** Perform initialization.
        */
        if (len == 0)
                outlen = CS_NULLDATA;
        else
                outlen = CS_GOODDATA;
        memset(&paramfmt, (CS_INT)0,
          (CS_INT)sizeof(CS_DATAFMT));
        strcpy(paramfmt.name, "xp_echo");
        paramfmt.namelen = CS_NULLTERM;
        paramfmt.datatype = CS_CHAR_TYPE;
        paramfmt.format = CS_FMT_NULLTERM;
        paramfmt.maxlength = ECHO_BUF_LEN;
        paramfmt.locale = (CS_LOCALE *) NULL;
        paramfmt.status |= CS_CANBENULL;
        /*
        ** Describe the data being sent.
        */
        if ((result == CS_SUCCEED) &&
                srv_descfmt(srvproc, CS_SET,
                  SRV_ROWDATA, 1, &paramfmt)
                  != CS_SUCCEED)
        {
                result = CS_FAIL;
        }
        /*
        ** Describe and bind the buffer that
        ** contains the data to be sent.
        */
        if ((result == CS_SUCCEED) &&
                (srv_bind(srvproc, CS_SET,
                SRV_ROWDATA, 1,
                &paramfmt, (CS_BYTE *)
                echo_str_buf, &len, &outlen)
                != CS_SUCCEED))
        {
```

```
                        result = CS_FAIL;
                }
                /*
                ** Send the actual data.
                */
                if ((result == CS_SUCCEED) &&
                        srv_xferdata(srvproc, CS_SET,
                        SRV_ROWDATA)!= CS_SUCCEED)
                {
                        result = CS_FAIL;
                }
        }
        /*
        ** Indicate to the ESP client how the
        ** transaction was performed.
        */
        if (result == CS_FAIL)
                srv_sendstatus(srvproc, 1);
        else
                srv_sendstatus(srvproc, 0);
        /*
        ** Send a count of the number of rows sent to
        ** the client.
        */
        srv_senddone(srvproc,(SRV_DONE_COUNT |
          SRV_DONE_MORE), 0, 1);
    }
    return result;
}
```

建立 DLL

可以在服务器平台上使用能生成所需 DLL 的任何编译器。

有关使用 Open Server API 编译和链接的一般信息，参见 *Open Client/Server 补充说明*。

DLL 的搜索顺序

Windows NT 以下列顺序搜索 DLL：

1 调用应用程序的目录

2 当前目录

3 系统目录 (SYSTEM32)

4 PATH 环境变量中所列目录

UNIX 按照在 LD_LIBRARY_PATH 环境变量（对 Solaris 和 Digital UNIX）或 SH_LIBRARY_PATH（对 HP）中所列的目录顺序搜索库。

如果 XP Server 在搜索路径中未能找到 ESP 库，它将尝试从 Windows NT 的 *$SYBASE/DLL* 或其它平台的 *$SYBASE/lib* 中装载它。

不支持 DLL 的绝对路径名。

样本 makefile(UNIX)

下列 makefile，*make.unix*，用来为 UNIX 平台的 xp_echo 程序创建动态链接共享库。它将生成一个文件，在 Solaris 和 Digital UNIX 上名为 *examples.so*，在 HP 上名为 *examples.sl*。源代码位于 *$SYBASE/$SYBASE_ASE/sample/esp* 中，您可对其进行修改以供自己使用。

要使用 makefile 建立一个示例库，请输入：

```
make /f make.unix
```

```
#
# This makefile creates a shared library.  It needs the open
# server header
# files usually installed in $SYBASE/include directory.
# This make file can be used for generating the template ESPs.
# It references the following macros:
#
# PROGRAM is the name of the shared library you may want to create.
PROGRAM        = example.so
BINARY        = $(PROGRAM)
EXAMPLEDLL    = $(PROGRAM)

# Include path where ospublic.h etc reside. You may have them in
# the standard places like /usr/lib etc.

INCLUDEPATH    = $(SYBASE)/include
```

```
# Place where the shared library will be generated.
DLLDIR        = .

RM        = /usr/bin/rm
ECHO        = echo
MODE        = normal

# Directory where the source code is kept.
SRCDIR        = .

# Where the objects will be generated.
OBJECTDIR    = .

OBJS        = xp_echo.o

CFLAGS        = -I$(INCLUDEPATH)
LDFLAGS        =  $(GLDFLAGS) -Bdynamic

DLLLDFLAGS    =  -dy -G

#=================================================================
$(EXAMPLEDLL) : $(OBJS)
    -@$(RM) -f $(DLLDIR)/$(EXAMPLEDLL)
    -@$(ECHO) "Loading $(EXAMPLEDLL)"
    -@$(ECHO) " "
    -@$(ECHO) "    MODE:           $(MODE)"
    -@$(ECHO) "    OBJS:           $(OBJS)"
    -@$(ECHO) "    DEBUGOBJS:      $(DEBUGOBJS)"
    -@$(ECHO) " "

    cd $(OBJECTDIR); \
    ld -o $(DLLDIR)/$(EXAMPLEDLL) $(DEBUGOBJS) $(DLLLDFLAGS) $(OBJS)
    -@$(ECHO) "$(EXAMPLEDLL) done"
    exit 0

#=================================================================
$(OBJS) : $(SRCDIR)/xp_echo.c
    cd $(SRCDIR); \
    $(CC) $(CFLAGS) -o $(OBJECTDIR)/$(OBJS) -c xp_echo.c
```

样本 makefile(Windows NT)

下列 makefile，*xp_echo.mak*，用于为 Windows NT 上的 xp_echo 程序创建 DLL，使用的语言是 Microsoft Visual C++。它将生成一个名为 *xp_echo.dll* 的文件。Makefile 的源代码位于 *$SYBASE\%SYBASE_ASE%\sample\esp* 中，您可对其进行修改以供自己所用。

要使用 makefile 建立一个示例 DLL，请输入：

```
nmake -f xp_echo.mak
# Nmake macros for building Windows 32-Bit apps
!include <ntwin32.mak>
all: xp_echo.dll

# Update the object file if necessary
xp_echo.obj: xp_echo.c
    $(cc) $(cflags) $(cvars) $(cdebug) xp_echo.c

# Update the xp_echo.dll if necessary
xp_echo.dll: xp_echo.obj
    $(link) $(linkdebug) -def:xp_echo.def -out:xp_echo.dll \
    xp_echo.obj shell32.lib libsrv.lib
```

样本定义文件

下列文件，*xp_echo.def*，必须与 *xp_echo.mak* 在相同的目录下。它列出了在 EXPORTS 部分将被用作 ESP 函数的所有函数。

```
LIBRARY   examples

CODE      PRELOAD MOVEABLE DISCARDABLE
DATA      PRELOAD SINGLE

EXPORTS
        xp_echo  .   1
```

注册 ESP

创建了一个 ESP 函数并将其链接到 DLL 后，可在数据库中将其作为 ESP 注册。这使得用户能够将该函数作为 ESP 来执行。

使用下列任一方法都可注册 ESP：

- Transact-SQL 命令 create procedure
- 系统过程 sp_addextendedproc

使用 *create procedure*

创建 ESP 的 create procedure 语法与建议的 ANSI SQL3 语法一致：

```
create procedure [owner.]procedure_name
    [[(]@parameter_name datatype [= default] [output]
    [, @parameter_name datatype [= default]
    [output]]...[)]] [with recompile]
    as external name dll_name
```

procedure_name 是 ESP 的名称，与其在数据库中的名称一样。它必须与其支持的函数具有同样的名称。

参数声明同存储过程的声明一样，如第 14 章“使用存储过程”中所述。

Adaptive Server 将忽略 with recompile 子句（如果它包括在 create procedure 命令中用于创建 ESP）。

dll_name 是包含 ESP 支持函数的库名。

dll_name 可被指定为一个无扩展名的名称（例如 *msgs*）或带有针对具体平台的扩展名，如在 Windows NT 上为 *msgs.dll*，在 Solarison 上为 *msgs.so*。

在任一情况下，针对具体平台的扩展名被假定为是库的实际文件名的一部分。

由于 create procedure 是在指定的数据库中注册 ESP，所以在调用命令前应指定注册 ESP 的数据库。如果尚未工作在目标数据库中，可通过 isql 利用 use *database* 命令指定数据库。

下列语句将注册一个 ESP，该 ESP 由 xp_echo 例程支持（在第 486 页上的“ESP 函数示例”中的示例中进行了描述），并假设该函数在一个名为 *examples.dll* 的 DLL 中编译。该 ESP 在 pubs2 数据库中注册。

```
use pubs2
create procedure xp_echo @in varchar(255)
as external name "examples.dll"
```

有关详细信息，参见*参考手册*中的 create procedure。

使用 *sp_addextendedproc*

sp_addextendedproc 与 Microsoft SQL Server 使用的语法一致。可使用它替代 create procedure。此命令的语法为：

sp_addextendedproc *esp_name*, *dll_name*

esp_name 是 ESP 的名称。它必须与支持 ESP 的函数名称匹配。

dll_name 是包含 ESP 支持函数的 DLL 名称。

sp_addextendedproc 必须由具有 sa_role 权限的用户在 master 数据库中执行。 因此，sp_addextendedproc 始终在 master 数据库中注册 ESP，而不象 create procedure 那样在当前数据库中注册 ESP。与 create procedure 不同，sp_addextendedproc 不允许在 Adaptive Server 中检查参数或给定参数缺省值。

下列语句将在 master 数据库中注册一个 ESP，该 ESP 由 xp_echo 例程支持（在第 486 页上的 “ESP 函数示例”的示例中进行了描述），并假设该函数在一个名为 *examples.dll* 的 DLL 中编译：

```
use master
sp_addextendedproc "xp_echo", "examples.dll"
```

有关 sp_addextendedproc 命令的详细描述，参见*参考手册*。

删除 ESP

可使用 drop procedure 或 sp_dropextendedproc 中的任一个从数据库中删除 ESP。

drop procedure 的语法与存储过程语法相同：

drop procedure [*owner*.]*procedure_name*

下列命令将删除 xp_echo：

```
drop procedure xp_echo
```

sp_dropextendedproc 的语法是：

sp_dropextendedproc *esp_name*

例如：

```
sp_dropextendedproc xp_echo
```

这两种方法都通过删除系统表 sysobjects 和 syscomments 中对 ESP 的引用从而将 ESP 从数据库中删除。它们不会影响底层 DLL。

重命名 ESP

因为 ESP 名称被绑定到其函数的名称，所以不能象重命名存储过程那样使用 sp_rename 重命名 ESP。更改 ESP 的名称：

1. 用 drop procedure 或 sp_dropextendedproc 删除 ESP。
2. 重命名并编译支持函数。
3. 利用 create procedure 或 sp_addextendedproc 方法以新名称重建 ESP。

执行 ESP

执行 ESP 使用与执行通常的存储过程相同的 execute 命令。有关执行存储过程的语法和其它详细信息，参见第 445 页上的“创建和执行存储过程”。

也可以远程执行 ESP。有关执行远程存储过程的详细信息，参见第 457 页上的“远程执行过程”。

因为执行任何 ESP 都涉及 Adaptive Server 和 XP Server 之间的远程过程调用，所有在同一个 execute 命令中不能组合按值传递的参数和按名称传递的参数。所有参数必须全部按名称传递，或全部按值传递。这是执行扩展存储过程和常规存储过程的唯一有区别之处。

ESP 可返回：

- 指明成功或失败以及失败原因的状态值
- 输出参数的值
- 结果集

ESP 函数将利用 srv_sendstatus Open Server 例程报告返回状态值。从 srv_sendstatus 返回的状态值是因应用程序而定的。但零状态表明请求正常完成。

ESP 函数使用 srv_descfmt、srv_bind 和 srv_xferdata Open Server 例程返回输出参数的值和结果集。有关从 ESP 函数传递参数的详细信息，参见第 486 页上的“ESP 函数示例”和 *Open Server Server-Library/C 参考手册*。从 Adaptive Server 端，ESP 的返回值作为常规存储过程来处理。

系统 ESP

除 xp_cmdshell 外，还有多个支持 Windows NT 功能的系统 ESP，如 Adaptive Server 与 Windows NT 事件日志或邮件系统的集成。

所有系统 ESP 的名称都以 "xp_" 开头。在 Adaptive Server 安装过程中，在 sybsystemprocs 数据库中创建了它们。由于系统 ESP 装载在 sybsystemprocs 数据库中，其权限也在该处设置。但您可从任意数据库中运行系统 ESP。系统 ESP 有：

- xp_cmdshell
- xp_deletemail
- xp_enumgroups
- xp_findnextmsg
- xp_logevent
- xp_readmail
- xp_sendmail
- xp_startmail
- xp_stopmail

有关系统 ESP 的文档，参见*参考手册*。除此以外，*为 Windows NT 配置 Adaptive Server* 还详细讨论了一些针对 NT 的功能，如 MAPI 和 NT 事件日志的集成（使用针对 NT 的系统 ESP 实现的）。

获取有关 ESP 的信息

使用 sp_helpextendedproc 可获取有关在当前数据库中注册的 ESP 的信息。

在没有参数的情况下，sp_helpextendedproc 显示数据库中的所有 ESP 及包含其相关函数的 DLL 名称。使用 ESP 作为参数，它仅为指定的 ESP 提供此信息。

```
sp_helpextendedproc getmsgs
ESP Name    DLL
--------    ---------
getmsgs     msgs.dll
```

由于系统 ESP 在 sybsystemprocs 数据库中，所以必须使用 sybsystemprocs 数据库才能显示系统 ESP 和 DLL 的名称：

```
use sybsystemprocs
sp_helpextendedproc
ESP Name      DLL
--------      ---------
xp_freedll    sybyesp
xp_cmdshell   sybyesp
```

如果某个 ESP 的源文本使用 sp_hidetext 加密，Adaptive Server 将显示一条消息，警告您文本被隐藏。有关隐藏源文本的详细信息，参见*参考手册*中的 sp_hidetext。

ESP 例外和消息

Adaptive Server 处理来自 XP Server 的所有消息和例外。除了将标准 ESP 消息发送给客户端外，Adaptive Server 还在日志文件中记录它们。来自用户定义 ESP 的用户定义消息不做记录，也不发往客户端。

ESP 相关消息可由 XP Server、创建或使用 ESP 的系统过程或系统 ESP 生成。有关 ESP 相关消息的列表，参见 *Troubleshooting and Error Messages Guide*。

用户定义 ESP 的函数可使用 srv_sendinfo Open Server 例程生成消息。参见第 486 页上的“ESP 函数示例”中的样本 xp_message 函数。

手工启动 XP Server

通常用户没有必要手工启动 XP Server，因为 Adaptive Server 会在收到第一个 ESP 会话请求时启动 XP Server。但是，如果创建并调试自己的 ESP，则会发现有必要使用 xpserver 实用程序从命令行手工启动 XP Server。有关 xpserver 的语法，参见所用平台的实用程序手册。

第 16 章

触发器：强制实施参照完整性

本章讨论触发器，包括：

主题	页
触发器如何工作	501
创建触发器	503
使用触发器来维护参照完整性	505
多行注意事项	516
回退触发器	520
嵌套触发器	522
与触发器相关的规则	525
禁用触发器	528
删除触发器	529
获取有关触发器的信息	529

触发器如何工作

触发器是一种在表中插入、删除或更新数据时起作用的存储过程。触发器可用来执行一些自动操作，如对相关表进行级联更改、实施列限制、比较数据修改的结果和保持数据库数据的参照完整性。

触发器可*自动操作*。只要数据发生改变它们就会生效 — 无论是操作人员的输入还是应用程序进行的操作。触发器专门用于数据修改的一个或多个操作，如 update、insert 和 delete，并对每个 SQL 语句执行一次操作。

例如，要防止用户将 publishers 表中的出版公司删除，可按以下方法使用触发器：

```
create trigger del_pub
on publishers
for delete
as
begin
    rollback transaction
```

```
        print "You cannot delete any publishers!"
    end
```

下次试图从 publishers 表中删除行时，del_pub 触发器会取消此删除操作，回退事务并输出该结果消息。

只有在数据修改语句完成操作，并且 Adaptive Server 完成对所有数据类型、规则或完整性约束冲突检查之后，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 Adaptive Server 检测到服务器错误，则回退整个事务。

触发器在以下情况下特别有用：

- 触发器能对数据库中的相关表进行“级联”更改。例如，titles 表中 title_id 列上的删除触发器可以删除其它表中的匹配行，使用 title_id 列作为唯一的键来定位 titleauthor 和 roysched 中的行。
- 触发器可以不接受或回退违反了参照完整性的更改，取消试图修改数据的事务。当试图插入与其主键不匹配的外键时，这种触发器将生效。例如，可以在 titleauthor 上创建一个插入触发器，如果新的 titleauthor.title_id 值在 titles.title_id 中没有匹配值，它会将插入回退。
- 触发器能实施比通过规则定义的限制更为复杂的限制。与规则不同，触发器能引用列或数据库对象。例如，触发器能够回退试图增加图书价格超过预付款 1% 的更新操作。
- 触发器可以执行简单的“what if”分析。例如，触发器能比较数据修改前后的表状态，并根据比较结果采取相应操作。

使用触发器与使用完整性约束的对比

作为使用触发器的一种替代方法，可使用 create table 语句中的参照完整性约束来在数据库表中强制实施参照完整性。然而，参照完整性约束*不能*:

- 对数据库中的相关表进行级联更改
- 通过引用其它列或数据库对象实现复合限制
- 执行“what if”分析

另外，强制实施数据完整性不能使参照完整性约束回退当前的事务。使用触发器则可以根据处理参照完整性的方式回退或继续事务。有关事务的信息，参见第 18 章“事务：维护数据一致性和恢复”。

如果应用程序需要执行上述任务之一，则使用触发器。否则，使用参照完整性约束来强制实现数据完整性。Adaptive Server 在检查触发器之前将检查参照完整性约束，使违反约束的数据修改语句也不会引发触发器。有关参照完整性约束的详细信息，参见第 7 章“创建数据库和表”。

创建触发器

触发器是一种数据库对象。创建触发器时，应指定用于“引发”或激活触发器的表和数据修改命令。然后指定触发器要进行的操作。

例如，每次在 titles 表中试图插入、删除或更新数据时，触发器都输出一条消息：

```
create trigger t1
on titles
for insert, update, delete
as
print "Now modify the titleauthor table the same
way."
```

注意：除了名为 deltitle 的触发器，本章所讨论的其它触发器未包括在与 Adaptive Server 随附的 pubs2 数据库中。应用本章所述示例时，可使用 create trigger 语句来创建每个触发器示例。对于表中或列中的相同操作 — insert、update 或 delete — 每个新触发器将在不发出警告的前提下覆盖先前的触发器，且自动删除旧触发器。

create trigger 语法

create trigger 的完整语法为：

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 ]...
```

create 子句将创建并命名触发器。触发器名称必须符合标识符规则。

on 子句给出了用来激活触发器的表的名称。此表有时称为**触发器表**。

虽然触发器可以引用其它数据库中的对象，但它是在当前数据库中创建的。限定触发器名称的所有者名称必须与表中的名称相同。只有表所有者才可在表中创建触发器。如果表所有者是在 create trigger 子句或 on 子句中与表名称一起给出的，也必须在其它子句中指定该所有者。

for 子句指定了触发器表中用于激活触发器的数据修改命令。在前面的示例中，对 titles 所进行的 insert、update 或 delete 操作都可输出此消息。

SQL 语句可指定**触发器状态**和**触发器动作**。触发器状态指定了附加的标准，以决定 insert、delete 或 update 是否会触发触发器动作。可在 if 子句中用 begin 和 end 对多个触发器动作进行分组。

if update 子句可对指定列中的插入或更新进行测试。对于更新操作，即使更新没有改变列的值，如果列名包含在 update 语句的 set 子句中，则 if update 子句为真。不要将 if update 子句与 delete 一起使用。可以指定多列，并且在 create trigger 语句中可以使用多个 if update 子句。由于在 on 子句中指定了表名，所以在使用 if update 时不要在列名之前使用表名。

触发器中不允许有的 SQL 语句

由于触发器将作为事务的一部分被执行，所以在触发器中不允许使用以下语句：

- 所有 create 命令，包括 create database、create table、create index、create procedure、create default、create rule、create trigger 和 create view
- 所有 drop 命令
- alter table 和 alter database

- truncate table
- grant 和 revoke
- update statistics
- reconfigure
- load database 和 load transaction
- disk init、disk mirror、disk refit、disk reinit、disk remirror、disk unmirror
- select into

使用触发器来维护参照完整性

触发器用于维护参照完整性，这可确保数据库中重要的数据（如给定数据的唯一标识符）精确，并在更改数据库后仍可使用。参照完整性是通过主键与外键的使用来实现的。

主键是一个列或列组合，列的值可对行进行唯一标识。值不能为空并且必须有唯一索引。带有主键的表可以与其它表的外键相关联。主键表可以看作是具有**主 — 明细关系**的**主表**。在数据库中可以有多个这样的主 - 明细组。

可以使用 sp_primarykey 来标识主键。该命令使用 sp_helpjoins 显示该键，并把它添加到 syskeys 表中。

例如，title_id 列是 titles 的主键。它唯一地标识了 titles 中的书籍，并与 titleauthor、salesdetail 和 roysched 中的 title_id 相关联。titles 表对于 titleauthor、salesdetail 和 roysched 来说是主表。

第 634 页上的“pubs2 数据库框图”说明了 pubs2 表是如何关联的。第 664 页上的“pubs3 数据库框图”为 pubs3 数据库提供了相同信息。

外键是一个列或列组合，它们的值与主键相匹配。外键不必是唯一的。它经常与主键有着多对一的关系。外键值应是主键值的拷贝。这意味着如果主键中某个值不存在，则在外键中也不应存在此值。外键的值可以为空；如果组合外键的任一部分值为空，则整个外键的值一定为空。带有外键的表通常称作主表的**明细**表或**相关**表。

可以使用 sp_foreignkey 在数据库中标识外键。该命令使用 sp_helpjoins 显示该键，并使用那些引用了 syskeys 表的其它过程。

titleauthor、salesdetail 和 roysched 中的 title_id 列都是外键；这些表是明细表。

多数情况下，由于单个表的最大引用数为 200，所以可使用第 236 页上的“指定参照完整性约束”中所述的参照约束在表与表之间强制实施参照完整性。如果表超过此限制或需要特殊的参照完整性，应使用参照完整性触发器。

参照完整性触发器可保证外键的值与主键的值相匹配。当数据修改影响到键列时，触发器使用名为**触发器测试表**的临时工作表来比较新列值与相关的键。在编写触发器时，是基于临时存储在触发器测试表中的数据进行比较的。

测试触发器测试表的数据修改

Adaptive Server 在触发器语句中使用两种特殊的表： deleted 表和 inserted 表。它们是触发器测试时使用的临时表。编写触发器时，使用这些表来测试数据修改的影响并设置触发器的动作条件。不能直接改变触发器测试表中的数据，但可以在 select 语句中使用这些表来检测 insert、update 或 delete 的影响。

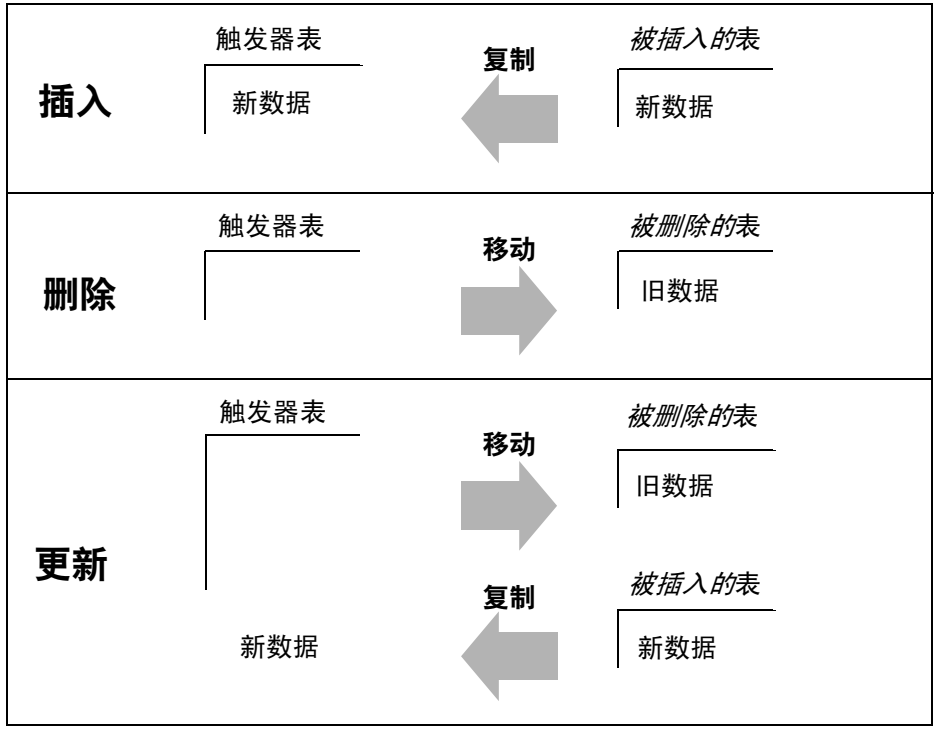
- deleted 表可存储在执行 delete 和 update 语句时受到影响的行的副本。在执行 delete 或 update 语句时，从触发器表中删除的行被传送到 deleted 表中。通常情况下，deleted 表与触发器表没有共用的行。
- inserted 表可存储在执行 insert 和 update 语句时受到影响的行的副本。在执行 insert 或 update 时，新行被同时添加到 inserted 表与触发器表中。inserted 表中的行是触发器表中新行的副本。

  下述触发器片段使用 inserted 表来测试 titles 表中 title_id 列的改变：

```
if (select count(*)
    from titles, inserted
   where titles.title_id = inserted.title_id) !=
    @@rowcount
```

update 实际上是先删除再插入；旧行先被复制到 deleted 表中；然后新行被复制到触发器表和 inserted 表中。下图说明了在执行 insert、delete 和 update 时触发器测试表的状态：

图 *16-1*：在执行插入、删除和更新操作时的触发器测试表



在设置触发器条件时，使用适合数据修改的触发器测试表。在测试 insert 时引用 deleted，或测试 delete 时引用 inserted 并不会导致错误；这种情况下，这些触发器测试表将不包含任何行。

注意：每次查询时，特定触发器只被触发一次。如果触发器动作取决于数据修改所影响的行数，应对多行数据修改进行测试（如 *@@rowcount* 测试），并采取适当措施。

下述触发器示例包含了必要的多行数据修改。用于存储受最近数据修改操作所“影响的行数”的 *@@rowcount* 变量可测试多行 insert、delete 或 update 命令。如果在触发器中其它任何 select 语句先于对 *@@rowcount* 的测试，则使用局部变量来存储该值，以便以后进行检查。所有不返回值的 Transact-SQL 语句将 *@@rowcount* 重新设置为 0。

插入触发器的示例

插入新的外键行时，确保外键与主键匹配。触发器应检查插入行（使用 inserted 表）与主键表中行之间的连接，然后回退与主键表中的键不匹配的外键插入。

下面的触发器对 inserted 表中的 title_id 值与 titles 表中的值进行了比较。假设您输入一个外键且没有插入空值。如果连接失败，事务将被回退。

```
create trigger forinsertrig1
on salesdetail
for insert
as
if (select count(*)
    from titles, inserted
    where titles.title_id = inserted.title_id) !=
    @@rowcount
/* Cancel the insert and print a message.*/
  begin
    rollback transaction
    print "No, the title_id does not exist in
    titles."
  end
/* Otherwise, allow it. */
else
  print "Added! All title_idís exist in titles."
```

@@rowcount 是指向 salesdetail 表添加的行数。也是向 inserted 表添加的行数。触发器连接 titles 和 inserted，以确定所有添加到 salesdetail 的 title_id 是否存在于 titles 表中。如果由 select count(*) 查询得出的连接的行数与 *@@rowcount* 不同，则表明有一个或多个不正确的插入，此事务将被取消。

对于插入被回退或被接受的情况，触发器分别输出不同的消息。要测试第一个条件，可尝试使用 insert 语句：

```
insert salesdetail
values ("7066", "234517", "TC9999", 70, 45)
```

要测试第二种条件，则输入：

```
insert salesdetail
values ("7896", "234518", "TC3218", 75, 80)
```

删除触发器的示例

如果删除了主键行，也应删除相关表中的相应外键行。通过确保主行被删除时明细行也被删除，这样保证了参照完整性。如果不删除相关表中的相应行，数据库中可能会有一些明细行不能被检索或识别。要正确删除相关外键行，可使用触发器来执行级联 delete 命令。

级联删除的示例

对 titles 执行 delete 语句时，就会从 titles 表中删除一行或多行，并添加到 deleted 表中。触发器能检查相关表 — titleauthor、salesdetail 和 roysched — 以查看其中是否有这样的行，即行中含有的 title_id 与从 titles 表中删除的行的 title_id（现存储在 deleted 表中）相匹配。如果触发器找到这样的行，会将其删除。

```
create trigger delcascadetrig
on titles
for delete
as
delete titleauthor
from titleauthor, deleted
where titleauthor.title_id = deleted.title_id
/* Remove titleauthor rows that match deleted
** (titles) rows.*/

delete salesdetail
from salesdetail, deleted
where salesdetail.title_id = deleted.title_id
/* Remove salesdetail rows that match deleted
** (titles) rows.*/
delete roysched
from roysched, deleted
where roysched.title_id = deleted.title_id
/* Remove roysched rows that match deleted
** (titles) rows.*/
```

限制删除的示例

实际上，可能因为历史的需要（激活不连续标题时查看上面有多少销售记录）或因为明细行上的事务尚未完成而想要保留某些明细行。一个编写良好的触发器应考虑这些情况。

防止主键删除

如果在 salesdetail 表中有关于某一主键的明细行，随 pubs2 提供的 deltitle 触发器可以防止删除该主键。此触发器保留了从 salesdetail 表中检索行的功能：

```
create trigger deltitle
on titles
for delete
as
if (select count(*)
    from deleted, salesdetail
    where salesdetail.title_id =
    deleted.title_id) > 0
  begin
    rollback transaction
    print "You cannot delete a title with sales."
  end
```

在此触发器中，通过将从 titles 表中删除的行与 salesdetail 表连接来对其进行测试。如果找到连接，事务将被取消。

同样，如果 titles 主表在 titleauthor 中有相关子项，则下面的限制删除可防止删除发生。它不计算 deleted 和 titleauthor 中的行数，而是检查 title_id 是否被删除。这种方法运行起来效率更高，这是由性能因素决定的，因为它检查的是特定行是否存在而不是浏览整个表计算所有行数。

记录发生的错误

下个示例将 raiserror 用在了错误消息 35003 上。raiserror 设置一个系统标志来记录发生的错误。尝试此例前，将错误消息 35003 添加到 sysusermessages 系统表中：

```
sp_addmessage 35003, "restrict_dtrig - delete
failed: row exists in titleauthor for this
title_id."
```

触发器是：

```
create trigger restrict_dtrig
on titles
for delete as
if exists (select * from titleauthor, deleted where
          titleauthor.title_id = deleted.title_id)
   begin
           rollback transaction
           raiserror 35003
```

```
            return
    end
```

要测试此触发器，可使用 delete 语句：

```
delete titles
where title_id = "PS2091"
```

更新触发器的示例

下个示例是从主表 titles 向相关表 titleauthor 和 roysched 进行级联更新。

```
create trigger cascade_utrig
on titles
for update as
if update(title_id)
begin
     update titleauthor
           set title_id = inserted.title_id
           from titleauthor, deleted, inserted
           where deleted.title_id =
titleauthor.title_id
     update roysched
           set title_id = inserted.title_id
           from roysched, deleted, inserted
          where deleted.title_id = roysched.title_id
     update salesdetail
           set title_id = inserted.title_id
           from salesdetail, deleted, inserted
           where deleted.title_id =
salesdetail.title_id
end
```

要测试此触发器，假定把 *Secrets of Silicon Valley* 一书从 popular_comp 重新归类为心理学书籍。下述查询在 titleauthor、roysched 和 titles 中将 title_id PC8888 更新为 PS8888。

```
update titles
set title_id = "PS8888"
where title_id = "PC8888"
```

限制更新触发器

主键是它的行和其它表中外键行的唯一标识符。通常，不应更新主键。更新主键时应特别慎重。在这种情况下，除非能满足指定的条件，否则要将更新操作回退来保证参照完整性。

Sybase 建议禁止对主键进行任何编辑改变。例如，可通过取消对该列的所有权限来实现这种禁止。但如果只想在某些条件下禁止更新，应使用触发器。

使用日期功能的限制更新触发器

下面的触发器可防止在周末对 titles.title_id 更新。stopupdatetrig 中的 if update 子句允许您关注特定的列 titles.title_id。如果对该列数据进行修改，会引起触发器运行。而对其它列的改变则不会。当触发器检测到一个与触发器条件冲突的更新时，将取消更新并打印出错信息。如果想测试这个触发器，可以用当前的日期代替“Saturday”或“Sunday”。

```
create trigger stopupdatetrig
on titles
for update
as
/* If an attempt is made to change titles.title_id
** on Saturday or Sunday, cancel the update. */
if update (title_id)
    and datename(dw, getdate())
    in ("Saturday", "Sunday")
  begin
    rollback transaction
    print "We do not allow changes to "
    print "primary keys on the weekend."
  end
```

具有多个动作的限制更新触发器

可以使用 if update 对多列指定多个触发器动作。下面的示例对 stopupdatetrig 进行了修改，以包含对 titles.price 或 titles.advance 的更新进行限制的额外触发器动作。除在周末禁止对主键进行更新外，还禁止更新 title 的 price 或 advance，除非该 title 的总收入超过了其 advance 的总量。可以使用相同的触发器名，因为在重新创建触发器时修改后的触发器会替代旧的触发器。

```
create trigger stopupdatetrig
on titles
for update
as
```

```
if update (title_id)
  and datename(dw, getdate())
  in ("Saturday", "Sunday")
  begin
    rollback transaction
    print "We do not allow changes to"
    print "primary keys on the weekend!"
  end
if update (price) or update (advance)
  if exists (select * from inserted
    where (inserted.price * inserted.total_sales)
    < inserted.advance)
    begin
      rollback transaction
      print "We do not allow changes to price or"
      print "advance for a title until its total"
      print "revenue exceeds its latest advance."
    end
```

下个示例是在 titles 上创建的，在出现下述情况之一时，可防止 update 操作：

- 用户试图更改 titles 中的 title_id 主键值
- 在 publishers 中未找到 pub_id 相关键
- 目标列不存在或为空

运用此示例之前，确保 sysusermessages 中存在下述错误消息：

```
sp_addmessage 35004, "titles_utrg - Update Failed: update of primary keys %1!
is not allowed."
sp_addmessage 35005, "titles_utrg - Update Failed: %1! not found in authors."
```

触发器为：

```
create trigger title_utrg
on titles
for update as
begin
     declare @num_updated int,
             @col1_var varchar(20),
             @col2_var varchar(20)
/* Determine how many rows were updated. */
select @num_updated = @@rowcount
    if @num_updated = 0
    return
/* Ensure that title_id in titles is not changed. */
if update(title_id)
```

```
    begin
       rollback transaction
     select @col1_var = title_id from inserted
     raiserror 35004 , @col1_var
     return
   end
 /* Make sure dependencies to the publishers table are accounted for. */
 if update(pub_id)
   begin
     if (select count(*) from inserted, publishers
         where inserted.pub_id = publishers.pub_id
         and inserted.pub_id is not null) != @num_updated
     begin
         rollback transaction
         select @col1_var = pub_id from inserted
         raiserror 35005, @col1_var
         return
     end
   end
/* If the column is null, raise error 24004 and rollback the
** trigger. If the column is not null, update the roysched table
** restricting the update. */
   if update(price)
     begin
         if exists (select count(*) from inserted
         where price = null)
     begin
         rollback trigger with
         raiserror 24004 "Update failed : Price cannot be null. "
     end
     else
     begin
         update roysched
         set lorange = 0,
         hirange = price * 1000
         from inserted
         where roysched.title_id =  inserted.title_id
     end
    end
end
```

要测试第一个错误消息 35004，输入：

```
update titles
set title_id = "BU7777"
where title_id = "BU2075"
```

要测试第二个错误消息 35005：

```
update titles
set pub_id = "7777"
where pub_id = "0877"
```

要测试第三个错误消息 24004：

```
update titles
set price = 10.00
where title_id = "PC8888"
```

此查询失败，因为 titles 中的 price 列为空。如果不为空，它就会更新 title 为 PC8888 的 price，并对 roysched 表进行必要的重新计算。错误 24004 不在 sysusermessages 中，但在此例中有效。它演示了代码中的“rollback trigger with raiserror”部分。

更新外键

外键自身的更改或更新可能会导致错误。外键只是主键的副本。切勿将二者设计为独立的。如果要保留外键的更新功能，应创建一个触发器，它通过检查 master 表的更新（如果与主键不匹配，将它们回退）来保证完整性。

在下例中，触发器对失败的两种可能原因进行测试： title_id 不在 salesdetail 表中或不在 titles 表中。

此例使用嵌套的 if...else 语句。当 update 语句中的 where 子句与 salesdetail 中的值不匹配时，第一个 if 语句为真，即 inserted 表不包含任何行，且 select 返回空值。如果通过了此测试，下一个 if 语句确定 inserted 表中的新行是否与 titles 表中的任何 title_id 连接。如果不连接任何行，则回退事务，输出错误消息。如果连接成功，则输出不同的消息。

```
create trigger forupdatetrig
on salesdetail
for update
as
declare @row int
/* Save value of rowcount. */
select @row = @@rowcount
if update (title_id)
  begin
    if (select distinct inserted.title_id
        from inserted) is null
      begin
        rollback transaction
```

```
        print "No, the old title_id must be in"
        print "salesdetail."
      end
    else
      if (select count(*)
          from titles, inserted
          where titles.title_id =
          inserted.title_id) != @row
        begin
          rollback transaction
          print "No, the new title_id is not in"
          print "titles."
        end
      else
        print "salesdetail table updated"
end
```

多行注意事项

当触发器的功能重新计算摘要值或提供进行中的计数时，多行注意事项尤为重要。

用于保持摘要值的触发器应包含 group by 子句或包含可执行隐式分组的子查询。当执行插入、更新或删除操作的行数超过一时，就会创建摘要值。由于 group by 子句增加了额外开销，因此编写下面示例是为了测试 *@@rowcount* 是否等于 1，等于 1 表示触发器表中只有一行受到影响。如果 *@@rowcount* = 1，触发器动作即可生效，无需 group by 子句。

使用多行的插入触发器示例

每次添加新 salesdetail 行时，下面的插入触发器将更新 titles 表中的 total_sales 列。只要在记录销售量时向 salesdetail 表添加了行，此触发器就会起作用。它更新 titles 表中的 total_sales 列，使 total_sales 等于其以前值加上添加到 salesdetail.qty 的值。这保证了向 salesdetail.qty 中插入后的总数总是最新。

```
create trigger intrig
on salesdetail
for insert as
    /* check value of @@rowcount */
```

```
if @@rowcount = 1
    update titles
      set total_sales = total_sales + qty
      from inserted
      where titles.title_id = inserted.title_id
else
    /* when @@rowcount is greater than 1,
       use a group by clause */
    update titles
      set total_sales =
        total_sales + (select sum(qty)
      from inserted
      group by inserted.title_id
      having titles.title_id = inserted.title_id)
```

使用多行的删除触发器示例

下个示例为删除触发器，每次删除 salesdetail 行时，它更新 titles 表中的 total_sales 列。

```
create trigger deltrig
on salesdetail
for delete
as
    /* check value of @@rowcount */
if @@rowcount = 1
    update titles
      set total_sales = total_sales - qty
      from deleted
      where titles.title_id = deleted.title_id
else
    /* when rowcount is greater than 1,
       use a group by clause */
    update titles
      set total_sales =
        total_sales - (select sum(qty)
      from deleted
      group by deleted.title_id
      having titles.title_id = deleted.title_id)
```

只要从 salesdetail 表中删除了行，触发器就会起作用。它更新 titles 表中的 total_sales 列，使 total_sales 等于其以前值减去从 salesdetail.qty 删除的值。

使用多行的更新触发器示例

每次更新 salesdetail 行中的 qty 字段时，下面的更新触发器将更新 titles 表中的 total_sales 列。回顾一下更新是先插入再删除的操作。此触发器引用 inserted 和 deleted 触发器测试表。

```
create trigger updtrig
on salesdetail
for update
as
if update (qty)
begin
    /* check value of @@rowcount */
    if @@rowcount = 1
        update titles
          set total_sales = total_sales +
            inserted.qty - deleted.qty
          from inserted, deleted
          where titles.title_id = inserted.title_id
          and inserted.title_id = deleted.title_id
    else
    /* when rowcount is greater than 1,
       use a group by clause */
    begin
        update titles
          set total_sales = total_sales +
            (select sum(qty)
                from inserted
                group by inserted.title_id
                having titles.title_id =
                inserted.title_id)
        update titles
          set total_sales = total_sales -
            (select sum(qty)
                from deleted
                group by deleted.title_id
                having titles.title_id =
                deleted.title_id)
    end
end
```

使用多行的条件插入触发器示例

如果仅因为某些数据修改是不可接受的，则不必回退所有数据修改。在触发器中使用相关子查询能够迫使触发器逐行检查修改的行。有关相关子查询的详细信息，参见第 178 页上的“使用相关子查询”。触发器可以对不同的行采取不同的操作。

下面的触发器示例假定存在一个名为 junesales 的表。其 create 语句为：

```
create table junesales
(stor_id     char(4)      not null,
ord_num      varchar(20)  not null,
title_id     tid          not null,
qty          smallint     not null,
discount     float        not null)
```

应在 junesales 表中插入四行来测试条件触发器。junesales 行中有两行具有与 titles 表中不匹配的 title_ids。

```
insert junesales values ("7066", "BA27619", "PS1372", 75, 40)
insert junesales values ("7066", "BA27619", "BU7832", 100, 40)
insert junesales values ("7067", "NB-1.242", "PSxxxx", 50, 40)
insert junesales values ("7131", "PSyyyy", "PSyyyy", 50, 40)
```

将数据从 junesales 插入到 salesdetail 时，语句如下：

```
insert salesdetail
select * from junesales
```

conditionalinsert 触发器逐行分析插入，并删除 titles 中没有 title_id 的行：

```
create trigger conditionalinsert
on salesdetail
for insert as
if
(select count(*) from titles, inserted
where titles.title_id = inserted.title_id)
    != @@rowcount
begin
  delete salesdetail from salesdetail, inserted
    where salesdetail.title_id = inserted.title_id
    and inserted.title_id not in
    (select title_id from titles)
  print "Only records with matching title_ids
    added."
end
```

触发器删除不需要的行。删除刚插入的行的功能取决于触发器引发后的处理顺序。首先，行被插入到表和 inserted 表中；然后引发触发器。

回退触发器

可以使用 rollback trigger 语句或 rollback transaction 语句（如果触发器引发作为事务的一部分）来回退触发器。但是，rollback trigger 仅回退触发器的效果和引起触发器引发的语句；而 rollback transaction 则回退整个事务。例如：

```
begin tran
insert into publishers (pub_id) values ("9999")
insert into publishers (pub_id) values ("9998")
commit tran
```

如果第二个 insert 语句引起 publishers 上的触发器发出 rollback trigger，则只影响第二个 insert；不会回退第一个 insert。如果此触发器发出的是 rollback transaction，则两个 insert 语句都会作为事务的一部分回退。

rollback trigger 的语法为：

```
rollback trigger
    [with raiserror_statement]
```

有关 rollback transaction 的语法，参见第 18 章“事务：维护数据一致性和恢复”。

raiserror_statement 是一个语句，它输出用户指定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，这样该错误包含的事务状态就能够反映出回退的情况。例如：

```
rollback trigger with raiserror 25002
    "title_id does not exist in titles table."
```

有关 raiserror 的详细信息，参见第 13 章“使用批处理和控制流语言”。

下面的插入触发器的示例执行与第 508 页上的“插入触发器的示例”中所述的 forinsertrig1 触发器相似的任务。然而，当回退插入而不是事务时，此触发器使用的是 rollback trigger 而不是 rollback transaction 来显示错误。

```
create trigger forinsertrig2
on salesdetail
for insert
as
if (select count(*) from titles, inserted
    where titles.title_id = inserted.title_id) !=
    @@rowcount
   rollback trigger with raiserror 25003
     "Trigger rollback: salesdetail row not added
     because a title_id does not exist in titles."
```

执行 rollback trigger 时，Adaptive Server 将中止当前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 将回退在这些触发器中完成的所有工作（最多还包含引起第一个触发器引发的那个更新操作）。

如果是通过批处理来执行包含 rollback transaction 语句的触发器，就会中止整个批处理。下述示例中，如果 insert 语句引发了包含 rollback transaction （如 forinsertrig1）的触发器，由于整个批处理将被中止，所以 delete 语句不会被执行：

```
insert salesdetail values ("7777", "JR123",
    "PS9999", 75, 40)
delete salesdetail where stor_id = "7067"
```

如果包含 rollback transaction 语句的触发器由**用户定义的事务**引发，rollback transaction 将回退整个批处理。在下面示例中，如果 insert 语句引发了包含 rollback transaction 的触发器，还会回退 update 语句：

```
begin tran
update stores set payterms = "Net 30"
    where stor_id = "8042"
insert salesdetail values ("7777", "JR123",
    "PS9999", 75, 40)
commit tran
```

有关用户定义的事务的信息，参见第 18 章“事务：维护数据一致性和恢复”。

Adaptive Server 忽略在触发器外执行的 rollback trigger，并且不发出与该语句关联的 raiserror。但在触发器外、事务内执行的 rollback trigger 会产生一个错误，该错误导致 Adaptive Server 回退此事务，并中止当前的语句批处理。

嵌套触发器

触发器可嵌套 16 层。当前的嵌套级别存储在 *@@nestlevel* 全局变量中。嵌套功能是在安装时启用的。系统管理员可以用 allow nested triggers 配置参数来打开和关闭触发器嵌套功能。

如果启用了嵌套的触发器，当表上有另一个触发器时，改变表的那个触发器就会引发第二个触发器，第二个触发器又引发第三个触发器，依次类推。如果链中的任何触发器引发了无限循环，就会超出此嵌套级别，触发器将中止。可以使用嵌套的触发器来执行有用的管家功能，如存储受前一个触发器影响的行的备份副本。

例如，可以在 titleauthor 上创建一个触发器来保存被 delcascadetrig 触发器删除的 titleauthor 行的备份副本。delcascadetrig 触发器生效后，如果从 titles 中删除 title_id “PS2091”，也会从 titleauthor 中删除对应的行。要保存数据，可以在 titleauthor 上创建一个 delete 触发器，将删除的数据保存在另一个表 del_save 中：

```
create trigger savedel
on titleauthor
for delete
as
insert del_save
select * from deleted
```

Sybase 建议按与顺序有关的序列使用嵌套触发器。使用单独触发器对数据进行级联修改，如在前面第 509 页上的“级联删除的示例”中的 delcascadetrig 示所述。

注意：将触发器放置在事务中时，如果在一组嵌套触发器的任一层出现故障，都会取消事务并回退所有数据修改。在触发器中使用 print 或 raiserror 语句来确定出现故障之处。

任一嵌套级别处的触发器中的 rollback transaction 将回退每个触发器的作用并取消整个事务。rollback trigger 只影响嵌套触发器和引起初始触发器引发的数据修改语句。

触发器自递归

缺省情况下，触发器不递归调用自身。即更新触发器不调用自身来响应在此触发器中对同一表的二次更新。如果表中一列上的更新触发器导致对另一列的更新，则此更新触发器只引发一次。但是可以打开 set 命令的 self_recursion 选项来允许触发器递归调用自身。还必须启用 allow nested triggers 配置变量才能实现自递归。

self_recursion 设置仅在当前客户端会话的持续时间内有效。如果此选项被设置为触发器的一部分，它的作用就要受到对它进行设置的触发器的范围的限制。如果设置了 self_recursion on 的触发器返回或引起其它触发器引发，则此选项将还原为 off。触发器打开 self_recursion 选项后，如果其自身的动作导致其再次引发，就能进行重复循环，但不能超过 16 层嵌套的限制。

例如，假定 pubs2 中有下面的 new_budget 表：

```
select * from new_budget
unit            parent_unit     budget
--------------- --------------- -------
one_department  one_division    10
one_division    company_wide    100
company_wide    NULL            1000

 (3 rows affected)
```

可创建一个触发器，每当更改 budget 列时，它就递归更新 new_budget，方法如下：

```
create trigger budget_change
on new_budget
for update as
if exists (select * from inserted
            where parent_unit is not null)
begin
    set self_recursion on
    update new_budget
    set new_budget.budget = new_budget.budget +
        inserted.budget - deleted.budget
    from inserted, deleted, new_budget
    where new_budget.unit = inserted.parent_unit
        and new_budget.unit = deleted.parent_unit
end
```

如果通过将 one_department 单位的 budget 增加 3 来更新 new_budget.budget，Adaptive Server 将执行如下操作（假定嵌套触发器已启用）：

1 将 one_department 从 10 增加到 13，可引发 budget_change 触发器。

2 触发器将 one_department（此例中为 one_division）父项的 budget 值从 100 更新为 103，这将再次引发触发器。

3 触发器将 one_division（此例中为 company_wide）父项从 1000 更新为 1003，这将第三次引发触发器。

4 触发器试图更新 company_wide 的父项，但由于不存在任何父项（值为“NULL”），因此不会发生最后一个 update 且不会引发触发器，自递归结束。可以通过查询 new_budget 来查看最终结果，方法如下：

```
select * from new_budget
unit            parent_unit     budget
--------------- --------------- -------
one_department  one_division    13
one_division    company_wide    103
company_wide    NULL            1003

(3 rows affected)
```

还可以通过其它方式递归执行触发器。触发器调用一个所执行的操作会使触发器再次引发的存储过程（只有在启用嵌套触发器后，触发器才会被再次激活）。除非触发器内有限制递归次数的条件，否则将导致嵌套级别溢出。

例如，如果更新触发器调用一个执行更新操作的存储过程，在将 nested triggers 设置为 off 时，则只执行一次触发器和存储过程。如果将 nested triggers 设置为 on，且触发器或过程中的某一条件要求更新的次数超过了 16，则循环将继续进行，直至触发器或过程超过 16 层这个最大嵌套值。

与触发器相关的规则

除了要预见多行数据修改、触发器回退和触发器嵌套的影响，编写触发器时，还需考虑下列因素。

触发器和权限

触发器是在某一特定表上定义的。只有表的所有者才有在表上进行 create trigger 和 drop trigger 操作的权限。这些权限不能转让给他人。

Adaptive Server 能接受对无权执行的操作采取行动的触发器定义。有了这种触发器，就可中止对触发器表进行修改的任何行为，因为不正确的权限会引起触发器引发并导致修改失败。事务将被取消。必须调整权限或删除触发器。

例如，Jose 拥有 salesdetail 并在上面创建了触发器。更新 salesdetail.qty 时，触发器也应更新 titles.total_sales。而 Mary 是 titles 的所有者，且没有授予 Jose 有关 titles 的权限。当 Jose 试图更新 salesdetail 时，Adaptive Server 检测到该触发器并发现 Jose 没有 titles 的权限，于是将更新事务回退。Jose 必须从 Mary 处获得 titles.total_sales 的更新权限，或者删除 salesdetail 上的触发器。

触发器限制

Adaptive Server 对触发器实行如下限制：

- 一张表最多可以有三个触发器：一个更新触发器、一个插入触发器与一个删除触发器。
- 每个触发器只能作用于一张表上。但一个触发器可用于用户的所有三个操作： update、insert 和 delete。
- 不能在视图或临时表上创建触发器，尽管触发器可以引用视图或临时表。
- writetext 语句不会激活插入触发器或更新触发器。
- 虽然 truncate table 语句由于删除所有行而在实际上与不含 where 子句的 delete 相似，但它却不能引发触发器，因为未记录各个行的删除。
- 不能在临时对象上创建触发器或建立索引或视图 (@object)

- 不能在系统表上创建触发器。如果尝试在系统表上创建触发器，Adaptive Server 将返回错误消息并取消该触发器。
- 如果触发器要从已插入或已删除的表的 text 列或 image 列中进行选择，则不能使用这样的触发器。
- 如果“组件集成服务”已启用，由于不能对正在插入、更新或删除的行（通过 inserted 和 deleted 表）进行检查，所以触发器在代理表上的使用具有局限性。可在代理表上创建可被调用的触发器。但由于 insert 被传递到远程服务器，所以插入或删除的数据并未被写入到代理表的事务日志中。因而，插入和删除的表（它们实际上是事务日志的视图）并不包含代理表的数据。

隐式和显式空值

只要在选择列表或在 values 子句中为列赋值，对于 insert 语句来说，if update(*column_name*) 子句即为真。*显式*空值或缺省值为列赋值，从而激活触发器。但*隐式*空值则不能如此。

例如，如果创建下面的表：

```
create table junk
(a int null,
 b int not null)
```

然后编写下面的触发器：

```
create trigger junktrig
on junk
for insert
as
if update(a) and update(b)
        print "FIRING"

    /*"if update" is true for both columns.
      The trigger is activated.*/
insert junk (a, b) values (1, 2)

    /*"if update" is true for both columns.
      The trigger is activated.*/
insert junk values (1, 2)

    /*Explicit NULL:
      "if update" is true for both columns.
      The trigger is activated.*/
```

```
insert junk values (NULL, 2)

    /* If default exists on column a,
      "if update" is true for either column.
      The trigger is activated.*/
insert junk (b) values (2)

    /* If no default exists on column a,
      "if update" is not true for column a.
      The trigger is not activated.*/
insert junk (b) values (2)
```

只使用下面的子句可以获得同样的结果：

```
if update(a)
```

要创建禁止插入隐式空值的触发器，可使用：

```
if update(a) or update(b)
```

然后触发器中的 SQL 语句进行检测，以确定 *a* 或 *b* 是否为空。

触发器和性能

就性能而言，触发器的开销通常很小。运行触发器所需的时间大多用于引用内存或数据库设备中的其它表。

deleted 和 inserted 触发器测试表总是在活动内存中。触发器所引用的其它表的位置将决定操作所需的时间。

有关触发器影响性能的详细信息，参见*性能和调优指南*。

触发器中的 *set* 命令

可以在触发器内使用 set 命令。在触发器运行期间，调用的 set 选项一直有效。然后，触发器恢复其原来的设置。

重命名和触发器

要更改触发器所引用的对象的名称，必须先删除此触发器然后重新创建，这样触发器的源文本才能反映所引用的对象的新名称。使用 sp_depends 获取触发器所引用的对象的报告。最安全的操作方法是不对触发器所引用的任何表或视图进行重命名。

触发器提示

创建触发器时，应考虑以下提示：

- 假定 insert 或 update 触发器能调用存储过程，而存储过程反过来更新基表。如果将 nested triggers 配置参数设置为真，触发器将进入无限循环。在执行 insert 或 update 触发器之前，将 sp_configure "nested triggers" 设置为假。
- 执行 drop table 时，任何与此表相关的触发器都将被删除。如果要保留这样的触发器，可在删除此表之前用 sp_rename 更改它们的名称。
- 使用 sp_depends 来查看触发器中所引用的表和视图的报告。
- 使用 sp_renamedb 重命名触发器。
- 每个查询（单个数据修改如 insert 或 update）只将触发器引发一次。如果查询在循环中重复，则触发器的引发次数与查询的重复次数相同。

禁用触发器

bcp 和 insert 将引发在向表中复制数据时所遇到的任何触发器，这将增加复制的时间（例如，bcp 从快速模式切换到慢速模式以便引发触发器）。在复制数据之前，使用 alter table 命令的 disable trigger 选项来禁用与表相关的任何触发器。可以使用 disable trigger 选项来禁用所有与表相关的触发器，或者可指定要禁用的某个触发器。但在复制结束之后，已禁用的任何触发器将不会被引发。如果需要使用这些触发器来对任何数据进行 insert、update 或 delete 操作，则必须对它们进行手工引发。

alter table... disable trigger 使用如下语法：

alter table [*database_name*.[*owner_name*].]*table_name*
{enable | disable } trigger [*trigger_name*]

其中 *table_name* 是要禁用触发器的表的名称，*trigger_name* 是要关闭的触发器的名称。例如，要禁用 pubs2 数据库中的 del_pub 触发器：

```
alter table pubs2
disable trigger del_pubs
```

如果未指定触发器，alter table 将禁用在表中定义的所有触发器。

load database 结束后，使用 alter table... enable trigger 重新启用触发器。要重新启用 del_pub 触发器，应发出：

```
alter table pubs2
enable trigger del_pubs
```

注意：只有表所有者或数据库管理员才能使用禁用触发器的功能。

如果触发器包含任何插入、更新或删除语句，则在禁用该触发器后，这些语句不会运行，这可能会影响表的参照完整性。

删除触发器

可以通过删除触发器或删除与其相关的触发器表来删除触发器。

drop trigger 的语法如下：

drop trigger [*owner*.]*trigger_name*
[, [owner.]*trigger_name*]...

删除表时，Adaptive Server 将删除与其相关的任何触发器。drop trigger 权限缺省为表的所有者，此权限不可转让。

获取有关触发器的信息

作为数据库对象，触发器按名称列在 sysobjects 中。sysobjects 的 type 列用缩写“TR”来标识触发器。此查询可找出数据库中的触发器：

```
select *
from sysobjects
where type = "TR"
```

每个触发器的源文本都存储在 syscomments 中。触发器的执行计划存储在 sysprocedures 中。以下几节所介绍的系统过程提供了系统表中有关触发器的信息。

sp_help

可以使用 sp_help 来获取关于触发器的报告。例如，可以获取有关 deltitle 的信息，方法如下：

```
sp_help deltitle
Name        Owner   Type
----------- ------- -----------
deltitle    dbo     trigger

Data_located_on_segment        When_created
-----------------------  -----------------
not applicable          Jul 10 1997 3:56PM

(return status = 0)
```

还可以使用 sp_help 来报告已禁用或已启用的触发器的状态：

```
1> sp_help trig_insert
2> go
Name Owner
Type
------------------------------------
trig_insert dbo
trigger
(1 row affected)
data_located_on_segment When_created
------------------------------------
not applicable Aug 30 1998 11:40PM
Trigger enabled
(return status = 0)
```

sp_helptext

要显示触发器的源文本，可执行 sp_helptext，方法如下：

```
sp_helptext deltitle
# Lines of Text
---------------
              1
text
----------------------------------------------
create trigger deltitle
on titles
for delete
as
```

```
if (select count(*) from deleted, salesdetail
where salesdetail.title_id = deleted.title_id) >0
begin
      rollback transaction
      print "You can't delete a title with sales."
end
```

如果触发器的源文本已用 sp_hidetext 加密，Adaptive Server 会显示一条消息，提示用户文本已被隐藏。有关隐藏源文本的详细信息，参见*参考手册*中的 sp_hidetext。

如果系统安全员已经用 sp_configure 重设 allow select on syscomments.text column 参数（这是在已评估的配置中运行 Adaptive Server 所必需的操作），那么只有触发器的创建者或系统管理员才能通过 sp_helptext 查看触发器的源文本。（有关详细信息，参见 *Adaptive Server 词汇表*中的**已评估的配置**。）

sp_depends

sp_depends 列出引用了对象的触发器，或列出触发器所影响的所有表或视图。下个示例说明如何使用 sp_depends 来获取 deltitle 触发器引用的所有对象的列表：

```
sp_depends deltitle
Things the object references in the current
database.

object               type       updated  selected
----------------     ----------  ------- --------
dbo.salesdetail      user table  no      no
dbo.titles      user table  no      no

(return status = 0)
```

下个语句列出了引用 salesdetail 表的所有对象：

```
sp_depends salesdetail
Things inside the current database that reference
the object.

object                        type
----------------------------  ----------------
dbo.deltitle                  trigger
dbo.history_proc              stored procedure
dbo.insert_salesdetail_proc   stored procedure
dbo.totalsales_trig           trigger

(return status = 0)
```

第 17 章

游标：逐行访问数据

游标每次访问一行或多行 SQL select 语句的结果。游标用于分别修改或删除各行。

本章讨论如下内容：

主题	页
游标的工作方式	533
声明游标	537
打开游标	544
使用游标读取数据行	544
使用游标更新和删除行	548
关闭并释放游标	550
一个使用游标的示例	550
在存储过程中使用游标	553
游标和锁定	555
获取有关游标的信息	556
使用浏览模式而不用游标	557
连接游标处理和数据修改	559

有关游标影响性能的方式的信息，参见*性能和调优指南*。

游标的工作方式

游标是一种与 select 语句相关联的符号名。它由以下部分组成：

- **游标结果集 —** 执行与游标相关联的查询所得的行的集（表）。
- **游标位置 —** 指向游标结果集中的一行的指针。

游标位置指示游标的当前行。可使用带有命名该游标的子句的 update 或 delete 语句显式地修改或删除该行。

执行一次称为**读取**的操作，可改变当前游标位置。fetch 命令将当前游标位置在游标结果集中向下移动一行或多行。

图 17-1 说明了执行 fetch 命令时游标结果集与游标位置如何工作。本例中，游标定义如下：

```
declare cal_authors_crsr cursor
for select au_id, au_lname, au_fname
from authors
where state = "CA"
for update
```

图 *17-1*：执行读取命令时，游表结果集与游标位置如何工作

作者表中的 cal_authors_crsr 游标结果集

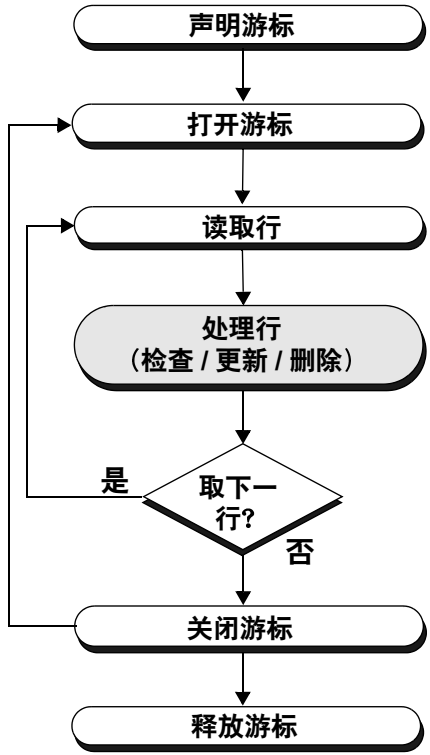
172-32-1176	White	Johnson	◄ 执行读取时的游标位置
213-46-8915	Green	Marjorie	◄ 下一个游标位置
...	...	...	

可将游标看作是 select 语句结果集上的“句柄”。使用它可一次检查并可能一次操作一行。但游标仅支持在查询结果中向前（或顺序）移动。读取几行后，将无法在游标结果集中回溯而再次访问这些行。

Adaptive Server 如何处理游标

图 17-2 显示了使用游标的相关步骤。

图 *17-2*：使用游标的步骤



Adaptive Server 将使用游标访问数据的进程分为以下几步操作：

- 声明游标

  如果在存储过程内声明游标，Adaptive Server 将创建游标结构并编译为该游标定义的查询。它存储但并不执行编译的查询计划。

  例如，下例中的游标声明 business_crsr 将查找 titles 表中所有商业书籍的名称和标识号。它还允许将来通过使用游标更新 price 列：

  ```
  declare business_crsr cursor
  for select title, title_id
  from titles
  where type = "business"
  for update of price
  ```

声明游标时，可使用 for update 子句确保 Adaptive Server 正确地执行定位型更新。

- 打开游标

  打开一个在存储过程外声明的游标后，Adaptive Server 将生成一个优化查询计划并编译该游标。然后，它将完成执行游标中定义的扫描并返回结果集所需的预备操作。例如，游标打开后，Adaptive Server 可能会在表上放置一个共享意向锁或创建一个工作表。

  如果已在存储过程内声明了游标，则该游标已以已编译格式存在。游标打开后，Adaptive Server 只需要完成执行扫描并返回结果集所需的预备操作。

- 从游标中读取

  该 fetch 命令执行已编译的游标，返回满足游标中定义的条件的一行或多行。缺省情况下，fetch 只返回单个行。第一次执行 fetch 命令返回满足游标搜索条件的第一行并存储游标的当前位置。第二次读取使用第一次读取所获的游标位置，返回满足搜索条件的下一行，并存储其当前位置。每次后继读取都将使用上次读取的游标位置查找下一个游标结果。

  使用 set cursor rows 命令可指定执行一次 fetch 返回的行数。参见第 546 页上的“使用每个 fetch 命令获取多个行”。

  在下例中，fetch 命令显示了包含一本商业书籍的 titles 表中第一行中的书名与标识号：

```
fetch business_crsr
title                                  title_id
-----------------------------------    --------
The Busy Executive's Database Guide    BU1032

(1 row affected)
```

  再运行一次 fetch business_crsr 将显示 titles 中下一本商业书籍的书名与标识号。

- 通过游标，使用检查、更新或删除方法处理行

  执行 fetch 后，Adaptive Server 将在当前游标位置更新或删除游标结果集中的数据（以及派生这些数据的相应基表）。此操作为可选操作。

下例中的 update 语句将商业书籍的价格提高了 5%；它只影响 business_crsr 游标当前指向的书籍：

```
update titles
set price = price * .05 + price
where current of business_crsr
```

更新游标行包括更改行中的数据或完全删除行。无法使用游标插入行。通过游标进行的所有更新将影响游标结果集中所包括的相应基表。

- 关闭游标

  Adaptive Server 将关闭游标结果集，删除所有剩余的临时表，并释放为游标结构保留的服务器资源。但它为游标保留了查询计划，以便再次打开游标。例如：

  ```
  close business_crsr
  ```

  关闭游标并将其再次打开后，Adaptive Server 已准备就绪，可重新创建游标结果集。执行 fetch 时，Adaptive Server 将游标定位在第一个有效行之前。这样就可视必要否确定处理游标结果集的次数。可不必经过整个结果集而随时关闭游标。

- 释放游标

  Adaptive Server 从内存转储查询计划并消除游标结构的所有跟踪。例如：

  ```
  deallocate cursor business_crsr
  ```

  使用游标前必须重新声明游标。

声明游标

必须先声明游标，才能使用游标。声明指定定义游标结果集的查询。通过使用关键字 for update 或 for read only 可显式地将游标定义为可更新或只读。如果省略任何一个关键字，Adaptive Server 将根据定义游标结果集的查询类型确定该游标是否可以更新。但 Sybase 建议显式地指定一个或另一个关键字；对更新而言，这样可确保 Adaptive Server 正确地执行定位型更新。不能在只读游标的结果集上使用 update 或 delete 语句。

declare cursor 语法

declare cursor 语句的语法为：

declare *cursor_name* cursor
for *select_statement*
[for {read only | update [of *column_name_list*]}]

cursor_name 是游标的名称。它必须是一个包含不多于 30 个字符的有效 Adaptive Server 标识符，且必须以字母、井号 (#) 或下划线 (_) 开头。

select_statement 是定义游标结果集的查询。有关其选项的信息，参见*参考手册*中的 select。通常，*select_statement* 可使用 Transact-SQL select 语句的完整语法与语义，包括关键字 holdlock。但它不能包含 compute、for browse 或 into 子句。

for read only 选项指定游标结果集不能更新。相反，for update 选项指定游标结果集可以更新。可在 for update 后指定 of *column_name_list*，将 *select_statement* 中的列列表定义为可更新。

declare cursor 语句必须出现在该游标的任何 open 语句之前。在同一 Transact-SQL 批处理中，不能将 declare cursor 与其它语句结合使用，除非是在存储过程中使用游标。

例如，下面的 declare cursor 语句为 authors_crsr 游标定义结果集，其中包含所有居住在 California 以外的作者：

```
declare authors_crsr cursor
for select au_id, au_lname, au_fname
from authors
where state != "CA"
for update
```

select_statement 可包含对 Transact-SQL 参数名或局部变量的引用。但这些名称只能引用在包含 declare cursor 语句的存储过程中定义的参数和局部变量。如果在触发器中使用游标，*select_statement* 也可引用触发器中使用的 inserted 和 deleted 临时表。有关使用 select 语句的信息，参见第 2 章“查询：从表中选择数据”。

游标类型

游标有以下四种类型：

- **客户游标** — 通过 Open Client 调用（或 Embedded SQL）声明。Open Client 跟踪从 Adaptive Server 返回的行，并将其置于缓冲区中供应用程序使用。只有通过 Open Client 调用才能完成对客户游标结果集的更新与删除。
- **执行游标** — 是客户游标的子集，其结果集由存储过程定义。存储过程可使用参数。参数值通过 Open Client 调用来传递。
- **服务器游标** — 在 SQL 中声明。如果在存储过程中使用服务器游标，执行存储过程的客户端不会注意到它们。执行 fetch 返回客户端的结果与执行常规 select 返回的结果相同。
- **语言游标** — 在 SQL 中（但不使用 Open Client）声明。与服务器游标相同，客户端不会注意到该游标，且其执行结果返回客户端的格式与常规 select 的相同。

通过对使用 Open Client 调用或 Embedded-SQL 的应用程序的使用，客户游标是最常用的游标形式。为简化对游标的讨论，本手册只对语言游标和服务器游标举例。有关客户游标或执行游标的示例，参见 Open Client 或 Embedded-SQL 文档。

游标范围

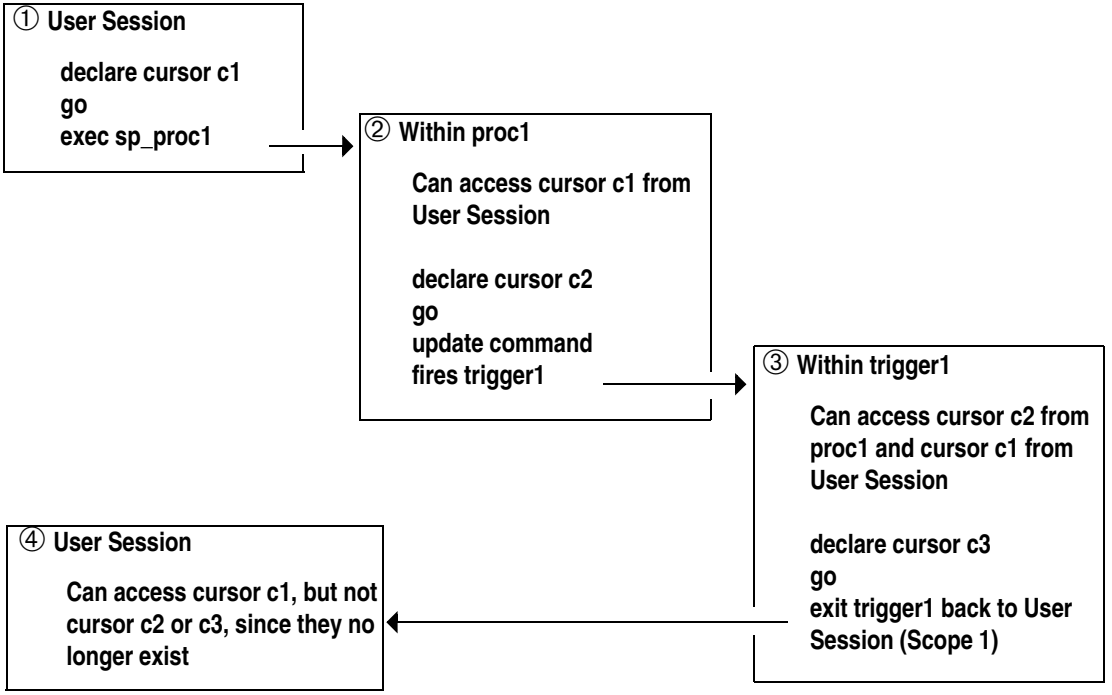
游标的存在取决于其**范围**。范围是指使用游标的环境： 在用户会话、存储过程或触发器内。

在用户会话内，游标只在用户结束会话之前存在。对其它用户启动的任何其它会话而言，游标并不存在。用户注销后，Adaptive Server 将释放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，则在其中创建的游标将应用于该范围以及启动该存储过程或触发器的范围。但所有嵌套的存储过程或触发器将不能访问触发器内的 inserted 或 deleted 表上声明的游标。在该触发器范围内*是*可以访问此类游标的。存储过程或触发器执行完成后，Adaptive Server 将释放在其中创建的游标。

图 17-3 说明了游标在范围间的运行方式。

图 *17-3*：游标在范围内的运行方式



给定范围内的游标名必须唯一。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
```

游标扫描与游标结果集

Adaptive Server 使用什么方法创建游标结果集取决于所使用的查询类型。如果查询不需要工作表，Adaptive Server 将通过使用表的索引键在基表上下移游标来执行 fetch，这与 select 语句的执行方式很类似，区别在于它返回 fetch 指定的行数。执行 fetch 后，Adaptive Server 将游标定位在下一个有效索引键上，直至再次读取或关闭游标。

某些查询使用工作表生成游标结果集。校验一特定游标是否使用工作表，检查 set showplan, no exec on 语句的输出即可。

注意：使用工作表时，用游标的 fetch 语句检索的行不一定能反映实际基表中行的值。例如，用 order by 子句声明的游标通常需要创建工作表才能为游标结果集进行行排序。Adaptive Server 不锁定与工作表中的行相对应的基表中的行，这样一来其他客户端就可更新这些基表行。因此，从游标语句返回客户端的行与基表行不同。有关锁与游标如何协同工作的详细信息，参见第 555 页上的“游标和锁定”。

该游标执行 fetch 操作返回行的同时将生成游标结果集。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**）可缩短周转时间，并使应用程序不需要的行无需再被读取。

Adaptive Server 要求游标扫描必须使用表的唯一索引，特别是对于隔离级别为 0 的读取。如果表有 IDENTITY 列并需要在其上创建非唯一索引，可使用 identity in nonunique index 数据库选项在表的索引键中加入 IDENTITY 列，以使表上创建的所有索引都是唯一索引。此数据库选项使逻辑上非唯一的索引在内部唯一，并允许这些索引用于处理隔离级别 0 读取的可更新游标。

引用无索引表的游标仍然可以使用，条件是这些表都没有被另一个使当前行位置发生移动的进程更新。例如：

```
declare storinfo_crsr cursor
for select stor_id, stor_name, payterms
    from stores
    where state = "CA"
```

上述游标指定的表 stores 不含任何索引。只要尚未在 declare cursor 语句中指定 for update，Adaptive Server 就允许在无唯一索引的表上声明游标。如果 update 不改变行的位置，则至执行下一个 fetch 命令前，游标位置不会改变。

使游标变为可更新

如果游标可更新，就可更新或删除该游标返回的行。如果是只读游标，就只可读取其数据而不能将其更新或删除。缺省情况下，Adaptive Server 在指定游标为只读前会试图确定其是否可更新。

在 declare 语句中使用关键字 read only 或 update 可显式地指定游标是否可更新。将游标指定为可更新能确保 Adaptive Server 正确执行定位型更新。确保被更新的表具有唯一索引。如果没有，Adaptive Server 将拒绝执行 declare cursor 语句。

下例为 pubs_crsr 游标定义一个可更新结果集：

```
declare pubs_crsr cursor
for select pub_name, city, state
from publishers
for update of city, state
```

上例包括 publishers 表中的所有行，但是它只为更新定义了 city 和 state 列。

除非计划通过游标更新或删除行，否则应将游标声明为只读。如果不显式地指定 read only 或 update，如果 select 语句*不*包含以下任何结构时，游标将为隐式可更新：

- distinct 选项
- group by 子句
- 集合函数
- 子查询
- union 运算符
- at isolation read uncommitted 子句

如果游标的 *select_statement* 包含以上结构之一，就不能指定 for update 子句。如果将包括 order by 子句的某些游标类型声明为其 *select_statement* 的一部分，Adaptive Server 也会将游标定义为只读。有关 Adaptive Server 支持的游标类型的信息，参见第 539 页上的“游标类型”。

确定可被更新的列

如果不用 for update 子句指定 *column_name_list*，则查询中指定的所有列都是可更新列。扫描基表时，Adaptive Serve 将试图为可更新游标使用唯一索引。对于游标，Adaptive Server 会将包含 IDENTITY 列的索引视为唯一索引，即使该游标并未如此声明。

可使用 Adaptive Server 更新游标的 *select_statement* 的列列表中没有指定的 *column_name_list* 中的列，但这些列必须是 *select_statement* 中指定的表的一部分。但在使用 for update 指定 *column_name_list* 时，只能更新该列表中的列。

下例中，Adaptive Server 在 publishers 的 pub_id 列上使用唯一索引（即使 newpubs_crsr 的定义中没有包括 pub_id）：

```
declare newpubs_crsr cursor
for select pub_name, city, state
from publishers
for update
```

如果不指定 for update 子句，Adaptive Server 将选择任一唯一索引，尽管如果指定表列没有唯一索引时，它也可使用其它索引或表扫描。但如果指定了 for update 子句，Adaptive Server 必须使用为一个或多个列定义的唯一索引扫描基表。如果不存在唯一索引，Adaptive Server 将返回错误信息。

大多数情况下，应在 for update 子句的 *column_name_list* 中包括要更新的列。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；游标扫描期间，Adaptive Server 会找到该列。如果该表有不止一个唯一索引，则应在 for update *column_name_list* 中包括该表的列，以便 Adaptive Server 可为执行游标扫描迅速找到该列。例如，以下 declare cursor 语句中使用的表在列 c3 上具有唯一索引，因此在 for update 列表中不应包括列：

```
declare mycursor cursor
for select c1, c2, 3
from mytable
for update of c1, c2
```

但如果 mytable 有不止一个唯一索引，例如，在列 c3 和 c4 上有不止一个唯一索引，就需要在 for update 子句中指定唯一索引，如下所示：

```
declare mycursor cursor
for select c1, c2, 3
from mytable
for update of c1, c2, c3
```

允许 Adaptive Server 以这种方式在游标扫描中使用唯一索引有助于防止一种称为 **Halloween 问题**的更新异常。如果客户端通过游标更新列，且该列（即，具有唯一索引的列）定义了行从基表返回的顺序，就会发生 Halloween 问题。例如，如果 Adaptive Server 使用某个索引访问基表，且该索引键被客户端更新，则被更新的索引行可在索引内移动，并可再次被游标读取。该行似乎在结果集中出现过两次： 当索引键被客户端更新且所更新的索引行在结果集中进一步下移时。

避免出现 Halloween 问题的另一种方法是创建将数据库选项 unique auto_identity index 设置为 on 的表。详细信息，参见*系统管理指南*。

打开游标

声明游标之后，要想 fetch、update 或 delete 行，就必须打开游标。打开游标后，Adaptive Server 就可通过对 select 语句求值开始创建游标结果集，该语句定义游标并使其可供处理。open 语句的语法是：

open *cursor_name*

不能打开一个已经打开或者尚未用 declare cursor 语句定义的游标。可重新打开一个关闭的游标，将游标位置重新设置到游标结果集的开头。

使用游标读取数据行

fetch 完成游标结果集并将一行或多行返回到负责从行中抽取列数据的客户端。视游标中定义的查询类型，Adaptive Server 将通过直接扫描表或扫描查询类型生成的工作表创建游标结果集。

fetch 命令将游标定位在游标结果集第一行之前。如果表具有有效索引，则 Adaptive Server 将游标定位在第一个索引键处。

可选择在 fetch 中包含 Transact-SQL 参数或局部变量以存储列值。

fetch 语法

fetch 语句的语法是：

fetch *cursor_name* [into *fetch_target_list*]

生成游标结果集后，Adaptive Serve 将游标位置在游标结果集中下移一行或多行。它检索结果集中的每行数据并存储当前位置，直至 Adaptive Server 到达结果集末端才允许进行额外读取。

例如，声明并打开 authors_crsr 游标后，可 fetch 结果集的第一行，如下所示：

```
fetch authors_crsr
au_id        au_lname            au_fname
-----------  ------------------- ---------------
341-22-1782  Smith               Meander

(1 row affected)
```

每个后继 fetch 将检索游标结果集中的另一行。例如：

```
fetch authors_crsr
au_id       au_lname            au_fname
----------- ------------------- ---------------
527-72-3246 Greene              Morningstar

(1 row affected)
```

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 将通过变量 *@@sqlstatus*（如第 546 页上的“检查游标状态”中所述）返回一个警告表示没有数据。游标位置保持不变。

不能对已经读取的行执行 fetch 命令。关闭并重新打开游标再次生成游标结果集，并开始从头读取数据。

into 子句指定 Adaptive Server 将列数据返回指定的变量中。*fetch_target_list* 必须包括以前声明的 Transact-SQL 参数或局部变量。

例如，声明变量 *@name*、*@city* 和 *@state* 之后，可从 pubs_crsr 游标中读取行，如下所示：

```
fetch pubs_crsr into @name, @city, @state
```

Adaptive Serve 要求 *fetch_target_list* 中的变量和定义该游标的 *select_statement* 所指定的目标列表表达式之间是一一对应关系。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。

检查游标状态

每次读取后 Adaptive Server 返回一个状态值。可通过全局变量 *@@sqlstatus* 访问该值。表 17-1 列出了 *@@sqlstatus* 值及其含义：

表 *17-1*：*@@sqlstatus* 值

值	含义
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。

下例确定当前打开的 authors_crsr 游标的 *@@sqlstatus*：

```
select @@sqlstatus
---------
        0

(1 row affected)
```

只有 fetch 语句才能设置 *@@sqlstatus*。其它语句对 *@@sqlstatus* 没有影响。

使用每个 *fetch* 命令获取多个行

缺省情况下，fetch 每次仅检索一行。可使用 set 命令的 cursor rows 选项更改 fetch 返回的行数。但该选项不会影响包含一个 into 子句的 fetch。

set 的语法是：

set cursor rows *number* for *cursor_name*

number 指定游标的行数。*number* 可以是无小数点的数字文字，或类型为 integer 的局部变量。声明的每个游标的缺省设置为 1。可为任何打开的或关闭的游标 setcursor rows 选项。

例如，可以更改为 authors_crsr 游标读取的行数，如下所示：

```
set cursor rows 3 for authors_crsr
```

设置了游标行数之后，每次对 authors_crsr 的 fetch 将从游标结果集返回三行：

```
fetch authors_crsr
au_id       au_lname            au_fname
----------- ------------------- ---------------
648-92-1872 Blotchet-Halls      Reginald
712-45-1867 del Castillo        Innes
722-51-5424 DeFrance            Michel

(3 rows affected)
```

游标定位在读取的最后一行（此例中为作者 Michel DeFrance）。

一次读取几行对客户端应用程序特别有益。如果读取不止一行，Open Client 或 Embedded SQL 将把发送给客户端应用程序的行放入缓冲区。在客户端上观察到的仍然是逐行访问，但实际上每次 fetch 已较少调用 Adaptive Server，从而提高了性能。

检查读取的行数

使用 *@@rowcount* 全局变量监控直至最后一次读取返回客户端的游标结果集的行数。该变量显示了任何一次读取游标所见到的总行数。

一旦从游标结果集读取了所有行，*@@rowcount* 就表示该结果集中的总行数。每一个打开的游标都与一个特定的 *@@rowcount* 变量相关联。关闭游标后该变量被删除。执行 fetch 后检查 *@@rowcount*，可获知为该 fetch 中所指定的游标而读取的行数。

下例确定当前打开的 authors_crsr 游标的 *@@rowcount*：

```
select @@rowcount
---------
        5

(1 row affected)
```

使用游标更新和删除行

如果游标可更新，使用 update 或 delete 语句更新或删除行。Adaptive Server 通过检查定义游标的 *select_statement* 确定游标是否可更新。使用 declare cursor 语句的 for update 子句，也可显式地将游标定义为可更新。有关详细信息，参见第 542 页上的“使游标变为可更新”。

更新游标结果集的行

可使用 update 语句的 where current of 子句更新当前游标位置处的行。对游标结果集进行的任何更新操作也会影响派生该游标行的基表行。

update...where current of 的语法是：

```
update [[database.]owner.]{table_name | view_name}
    set [[[database.]owner.]{table_name.|view_name.}]
        column_name1  =
            {expression1 |NULL|(select_statement)}
        [, column_name2  =
            {expression2 |NULL|(select_statement)}]...
    where current of cursor_name
```

set 子句指定游标的结果集列名并赋予其新值。当列出不止一个列名和值对时，必须以逗号将其分隔。

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 子句使用连接引用不止一个表或视图，可仅指定实际被更新的表或视图。

例如，可更新 pubs_crsr 游标当前指向的行，如下所示：

```
update publishers
set city = "Pasadena",
    state = "CA"
where current of pubs_crsr
```

更新后，游标位置保持不变。只要另一 SQL 语句没有移动该游标的位置，就可继续更新该游标位置上的行。

Adaptive Server 允许更新游标的 *select_statement* 列列表中没有指定的列，但这些列必须是该语句中指定的表的一部分。但如果使用 for update 指定了 *column_name_list*，就只能更新该列表中的列。

删除游标结果集的行

使用 delete 语句的 where current of 子句可删除当前游标位置处的行。从游标结果集中删除一行时，该行将从基础数据库表中删除。使用游标时，每次只能删除一行。

delete...where current of 的语法是：

delete [from]
[[*database*.]*owner*.]{*table_name*|*view_name*}
where current of *cursor_name*

用 delete...where current of 指定的 *table_name* 或 *view_name* 必须是在定义该游标的 select 语句的第一个 from 子句中指定的表或视图。

例如，可通过输入下面的语句删除 authors_crsr 游标当前指向的行：

```
delete from authors
where current of authors_crsr
```

上例中的关键字 from 是可选项。

注意：不能从包含连接的 select 语句定义的游标中删除行，即使该游标可更新。

从游标中删除一行后，Adaptive Server 将游标定位在游标结果集中被删除行的下面一行之前。要访问下一行，仍然必须使用 fetch 命令。如果删除的行是游标结果集中的最后一行，则 Adaptive Server 将游标定位在结果集最后一行之后。

例如，在上例中删除当前行（即作者 Michel DeFrance）后，可读取游标结果集中的下三位作者（假设 cursor rows 仍设置为 3）：

```
fetch authors_crsr
au_id       au_lname            au_fname
-----------  ------------------- ---------------
807-91-6654  Panteley            Sylvia
899-46-2035  Ringer              Anne
998-72-3567  Ringer              Albert

(3 rows affected)
```

当然，不引用游标也可以从基表中删除行。对基表进行更改后，游标结果集随之更改。

关闭并释放游标

游标结果集使用完毕后，可使用以下命令 close 游标：

close *cursor_name*

关闭游标不会改变其定义。如果重新打开一个游标，Adaptive Server 将使用与以前相同的查询创建一个新的游标结果集。例如：

```
close authors_crsr
open authors_crsr
```

然后就可从其游标结果集的开头开始从 authors_crsr 中读取。任何与该游标相关联的条件（例如由 set cursor rows 定义的读取的行数）仍然有效。

例如：

```
fetch authors_crsr
au_id       au_lname            au_fname
----------- ------------------- ---------------
341-22-1782 Smith               Meander
527-72-3246 Greene              Morningstar
648-92-1872 Blotchet-Halls      Reginald

(3 rows affected)
```

要放弃游标，必须使用以下命令将其释放：

deallocate cursor *cursor_name*

释放游标将释放所有与该游标相关联的资源，包括游标名。释放游标后，才能重新使用游标名。如果释放一个打开的游标，Adaptive Server 会自动将其关闭。终止客户端与服务器的连接也会关闭并释放任何打开的游标。

一个使用游标的示例

以下游标示例使用此查询：

```
select author = au_fname + " " + au_lname, au_id
from authors
```

该查询的结果是：

```
author                      au_id
-------------------------   -----------
Johnson White               172-32-1176
Marjorie Green              213-46-8915
Cheryl Carson               238-95-7766
Michael OíLeary             267-41-2394
Dick Straight               274-80-9391
Meander Smith               341-22-1782
Abraham Bennet              409-56-7008
Ann Dull                    427-17-2319
Burt Gringlesby             472-27-2349
Chastity Locksley           486-29-1786
Morningstar Greene          527-72-3246
Reginald Blotchet Halls     648-92-1872
Akiko Yokomoto              672-71-3249
Innes del Castillo          712-45-1867
Michel DeFrance             722-51-5454
Dirk Stringer               724-08-9931
Stearns MacFeather          724-80-9391
Livia Karsen                756-30-7391
Sylvia Panteley             807-91-6654
Sheryl Hunter               846-92-7186
Heather McBadden            893-72-1158
Anne Ringer                 899-46-2035
Albert Ringer               998-72-3567

(23 rows affected)
```

在以上查询中使用游标：

1　声明游标。

declare cursor 语句使用 select 语句定义游标，如下所示：

```
declare newauthors_crsr cursor for
select author = au_fname + " " + au_lname, au_id
from authors
for update
```

2　打开游标：

```
open newauthors_crsr
```

3 使用游标读取行：

```
fetch newauthors_crsr
author                        au_id
-------------------------     -----------
Johnson White                 172-32-1176

(1 row affected)
```

用 set 命令指定行数可一次读取不止一行：

```
set cursor rows 5 for newauthors_crsr
go
fetch newauthors_crsr
author                        au_id
-------------------------     -----------
Marjorie Green                213-46-8915
Cheryl Carson                 238-95-7766
Michael OíLeary               267-41-2394
Dick Straight                 274-80-9391
Meander Smith                 341-22-1782

(5 rows affected)
```

每个后继 fetch 返回另外 5 行：

```
fetch newauthors_crsr
author                        au_id
-------------------------     -----------
Abraham Bennet                409-56-7008
Ann Dull                      427-17-2319
Burt Gringlesby               472-27-2349
Chastity Locksley             486-29-1786
Morningstar Greene            527-72-3246

(5 rows affected)
```

游标现定位在作者 Morningstar Greene 处，该处是当前 fetch 的最后一行。

4 执行以下更新可更改 Greene 的名：

```
update authors
set au_fname = "Voilet"
where current of newauthors_crsr
```

在执行下一个 fetch 前，游标将停留在 Ms. Greene 的记录位置。

5 游标使用完毕后，可将其关闭：

```
close newauthors_crsr
```

如果再次 open 该游标，Adaptive Server 将返回查询并将游标放置在结果集中的第一行之前。游标仍然设置为执行一次 fetch 返回五行。

6 使用 deallocate 命令取消游标定义：

```
deallocate cursor newauthors_crsr
```

释放游标后，才能重新使用游标名。

在存储过程中使用游标

在存储过程中游标特别有用。使用游标仅用一个查询就能完成一项任务，而不使用游标完成同一项任务将需要用几个查询。但所有游标操作必须在单个过程中执行。存储过程不能 open、fetch 或 close 未在该过程中声明的游标。游标在存储过程范围以外将被取消定义。参见第 539 页上的“游标范围”。

例如，存储过程 au_sales 检查 sales 表查看某个作者的任何书籍的销售状况是否良好。它使用游标检查每一行，然后输出信息。没有游标，需要使用几个 select 语句才能完成相同的任务。在存储过程之外，不能将其它语句和 declare cursor 置于同一批处理中。

```
create procedure au_sales (@author_id id)
as

/* declare local variables used for fetch */
declare @title_id tid
declare @title varchar(80)
declare @ytd_sales int
declare @msg varchar(120)

/* declare the cursor to get each book written
    by given author */
declare author_sales cursor for
select ta.title_id, t.title, t.total_sales
from titleauthor ta, titles t
where ta.title_id = t.title_id
and ta.au_id = @author_id

open author_sales

fetch author_sales
        into @title_id, @title, @ytd_sales
```

```
if (@@sqlstatus = 2)
begin
    print "We do not sell books by this author."
    close author_sales
    return
end

/* if cursor result set is not empty, then process
    each row of information */
while (@@sqlstatus = 0)
begin
    if (@ytd_sales = NULL)
    begin
        select @msg = @title +
           " -- Had no sales this year."
        print @msg
    end
    else if (@ytd_sales < 500)
    begin
        select @msg = @title +
            " -- Had poor sales this year."
        print @msg
    end
    else if (@ytd_sales < 1000)
    begin
        select @msg = @title +
            " -- Had mediocre sales this year."
        print @msg
    end
    else
    begin
        select @msg = @title +
            " -- Had good sales this year."
        print @msg
    end

    fetch author_sales into @title_id, @title,
    @ytd_sales
end
```

例如：

```
au_sales "172-32-1176"
Prolonged Data Deprivation: Four Case Studies -- Had good sales this year.

(return status = 0)
```

有关存储过程的详细信息，参见第 14 章“使用存储过程”。
有关使用游标的存储过程如何影响性能的信息，另请参见*性能和调优指南*。

游标和锁定

*游标锁定方法*与 Adaptive Server 的其它锁定方法类似。通常情况下，读取数据的语句（例如 select 或 readtext）在每个数据页上使用共享锁以避免从未提交的事务中读取已更改的数据。更新语句在其更改的每一页上使用**排它锁**。为减少死锁发生及提高并发性能，Adaptive Server 经常在排它锁前放置一个更新锁，这表示客户端想更改该页上的数据。

对于可更新游标，缺省情况下 Adaptive Server 在扫描用 declare cursor的for update子句引用的表或视图时使用更新锁。如果其中包括 for update 子句，但列表为空，则在 *select_statement* 的 from 子句中引用的所有表与视图缺省情况下将接收更新锁。如果不包括 for update 子句，则引用的表和视图将接收共享锁。如果对某些表更喜欢使用**共享锁**，则可将 shared 关键字添加到每个表名后的 from 子句中，这样就可用共享锁代替更新锁。

注意：游标位置移出数据页后，Adaptive Server 将释放更新锁。因为应用程序为客户端游标将行放入缓冲区中，相应的服务器游标可能会被定位在与客户端游标不同的数据行和页上。在这种情况下，另一个客户端可更新表示第一个客户端当前游标位置的行，即使第一个客户端使用了 for update 选项。

事务中游标获得的任何排它锁将可持有直至该事务结束。使用 holdlock 关键字或 set isolation level 3 选项时，游标获得的共享锁或更新锁也可持有至事务结束。但如果没有设置 close on endtran 选项，则事务结束后游标将仍保持打开状态，且其当前页锁仍然有效。游标读取其它行时还可继续获取锁。

有关 Adaptive Server 中游标锁定的详细信息，参见*性能和调优指南*。

游标锁定选项

这些是定义可更新游标时指定 holdlock 或 shared 选项（在 select 语句中）的结果：

- 如果两个选项都省略，就仅能在当前读取的页上读取数据。其他用户通过游标或其它方式都无法更新当前读取的页。其它用户可在您用于游标的相同表上声名游标，但他们无法在您当前读取的页上获得更新锁。
- 如果指定 shared 选项，就只能在当前读取的页上读取数据。其他用户通过游标或其它方式都无法更新当前读取的页。
- 如果指定 holdlock 选项，就可在读取的所有页（在当前事务中）上或仅在当前读取的页（如果不在事务中）上读取数据。其它用户通过游标或其它方式都无法更新当前读取的或在当前事务中读取的页。其它用户可在用于游标的相同表上声名游标，但他们无法在当前读取的页上或在当前事务中读取的页上获得更新锁。
- 如果两个选项都指定，就可在读取的所有页（在当前事务中）上或仅在当前读取的页（如果不在事务中）上读取数据。其他用户通过游标或其它方式都无法更新当前读取的页。

获取有关游标的信息

使用 sp_cursorinfo 查找有关游标名、其当前状态及其结果列的信息。该示例显示信息 authors_crsr 如下：

```
sp_cursorinfo 0, authors_crsr
Cursor name 'authors_crsr' is declared at nesting
    level '0'.
The cursor id is 327681
The cursor has been successfully opened 1 times
The cursor was compiled at isolation level 1.
The cursor is not open.
The cursor will remain open when a transaction is
    committed or rolled back.
The number of rows returned for each FETCH is 1.
The cursor is updatable.
There are 3 columns returned by this cursor.
The result columns are:
Name = 'au_id', Table = 'authors', Type = ID,
    Length = 11 (updatable)
```

```
Name = 'au_lname', Table = 'authors', Type =
    VARCHAR, Length = 40 (updatable)
Name = 'au_fname', Table = 'authors', Type =
    VARCHAR, Length = 20 (updatable)
```

也可使用全局变量 *@@sqlstatus* 和 *@@rowcount* 检查游标的状态。参见第 546 页上的“检查游标状态”和第 547 页上的“检查读取的行数”。

有关 sp_cursorinfo 的详细信息，参见*参考手册*。

使用浏览模式而不用游标

浏览模式允许在表中每次搜索一行并更新其值。它用于使用 DB-Library 和主机编程语言的前端应用程序中。浏览模式有用处，因为它提供了与 Open Server™ 应用程序和早期 Open Client libraries 的兼容性。但它在最新的 Client-Library™ 应用程序（版本 10.0.x 及更高版本）中使用的效果却无法令人满意，因为游标可提供相同的功能，且可移植性和灵活性更好。另外，因为浏览模式是 Sybase 所特有，所以不适于异构环境。

正常情况下，要按行更改表值时应使用游标更新数据。Client-Library 应用程序可使用 Client-Library 游标实现某些浏览模式功能，例如从表中读取行时更新表。但游标可能会导致所选取的表中发生锁争用。

有关浏览模式的详细信息，参见 Open Client/Server™ 文档中的 dbqual 功能。

浏览表

要浏览前端应用程序中的表，可在发送给 Adaptive Server 的 select 语句末尾附加 for browse 关键字。

例如：

Start of select *statement in an Open Client application*

...

for browse

Completion of the Open Client application routine

如果表中的行带有时间戳，就可在前端应用程序中浏览该表。

浏览模式限制

不能在包含 union 运算符的语句或游标声明中使用 for browse 子句。

在包括 for browse 选项的 select 语句中，不允许使用关键字 holdlock。

浏览模式下，选择语句中的关键字 distinct 将被忽略。

为用于浏览的新表加盖时间戳

在创建用于浏览的新表时，在表定义中加入名为 timestamp 的列。不必指定其数据类型，因为系统将自动为该列分配 timestamp 数据类型。例如：

```
create table newtable(col1 int, timestamp,
    col3 char(7))
```

只要插入或更新某行，Adaptive Server 就将为其加盖时间戳，方法是自动为 timestamp 列分配唯一的 varbinary 值。

为现有表加盖时间戳

要使现有表可用于浏览，应使用 alter table 添加名为 timestamp 的列。例如：

```
alter table oldtable add timestamp
```

在每个现有行中将添加一个包含空值的 timestamp 列。要生成时间戳，可在不指定新列值的情况下更新每个行。

例如：

```
update oldtable
set col1 = col1
```

比较 *timestamp* 值

在前端应用程序中使用浏览模式时，可使用 tsequal 系统函数比较时间戳。例如，以下语句将更新已浏览过的 publishers 中的行。它将该行的浏览版本中的 timestamp 列与其存储版本中的十六进制时间戳相比较。如果这两个时间戳不相等，将接收到错误消息，且该行不会被更新。

```
update publishers
set city = "Springfield"
where pub_id = "0736"
and tsequal(timestamp,0x0001000000002ea8)
```

不要将 where 子句中的 tsequal 函数用作搜索自变量。使用 tsequal 时，where 子句的其余部分应只与单个行唯一匹配。应只在 insert 和 update 语句中使用 tsequal 函数。如果将 timestamp 列用作搜索子句，就应像比较常规 varbinary 列那样比较该列，即 *timestamp1* = *timestamp2*。

连接游标处理和数据修改

本节描述可能会影响使用游标的应用程序的对游标行为的更改，这些更改始于 11.9.2 版。同时还描述有关 DOL 锁表上游标的游标定位和修改规则。

可影响游标位置的更新和删除

以下两种类型的删除和更新可影响游标位置处的行：

- *定位型*删除和更新，使用 delete...where current of 或 update...where current of 改变游标位置所在的行
- *搜索型*删除和更新是指更改游标位置处行中的值，但不包括 where current of 子句的任何 delete 或 update 查询

执行 *delete* 或 *update* 命令后没有连接的游标定位

对于单表查询上的游标，delete 或 update 命令对基表所做的操作取决于修改的类型、基表的锁定方案以及基表的聚簇索引是否会受影响：

- 在游标位置处删除行的定位型 delete 或搜索型 delete 将游标定位到表中下一个限定行。这对于 allpages 锁表和 DOL 锁表都是如此。不允许通过此游标进行随后的定位型 update 或 delete，直到下一个 fetch 完成了将游标定位于下一个限定行。
- 不改变行位置的搜索型或定位型 update 将保持游标的当前位置不变。下一个 fetch 返回下一个限定行。
- 在 allpages 锁表上的搜索型或定位型 update 可以改变行的位置，例如，假如它更新聚簇索引的键列。游标不跟踪行，它仍定位在下一行前的初始位置。不允许进行定位型更新，直到随后的 fetch 返回下一行。如果行移动到搜索顺序中靠后的位置，则更新的行可能再次被游标定位。

  由于 DOL 锁表具有固定的行 ID，因此扩展更新或扩展影响集群键的更新将不会移动该行的位置。游标仍然定位在此行上，且执行下一次 fetch 将返回下一个限定行。

上述游标定位行为与以前版本中相同。

更新和删除对连接游标的影响

如果在包含连接的游标的游标位置处的行发出搜索型或定位型 delete，其结果可以是以下两种之一：

- 搜索型删除会隐式地关闭游标。下一个 fetch 返回错误 582：

  游标‘*cursor_name*’被隐式关闭，因为当前游标位置因执行更新或删除操作而被删除。游标扫描位置无法恢复。This happens for cursors which reference more than one table.（游标引用了不止一个表时会发生这种情况。）

- 定位型删除失败，显式错误 592：

  对游标‘*cursor_name*’执行 `DELETE WHERE CURRENT OF` 失败，因为游标位于连接上。

  游标仍保持打开，定位于同一行。

对游标的连接列发出搜索型或定位型 update 时：

- 对 allpages 锁表上的聚簇索引键的搜索型更新可成功完成，但会隐式地关闭该游标，因此下一次读取会返回错误 582。对聚簇索引键的搜索型更新不会关闭 DOL 锁表上的游标。
- 对于所有锁定方案的定位型更新，以及对于在 allpages 锁表上不改变聚簇索引键的搜索型更新，将会得到成功且不关闭游标。

  游标位置取决于表的类型以及列是否有聚簇索引。有关详细信息，参见第 560 页上的“执行 delete 或 update 命令后没有连接的游标定位”。

  连接列更新时连接列缓冲可影响返回的结果集。有关详细信息，参见第 559 页上的“连接游标处理和数据修改”。

表 17-2 显示了 delete 和 update 命令如何影响连接游标。

在此表中，“保持打开”表示更新未将游标关闭。游标仍然定位在行中，所以仍然可以进行定位型更新，并且下一个 fetch 也将成功。

表 *17-2*：连接游标中的删除和连接列更新的影响

	allpages 锁	DOL 锁
delete 命令		
定位型直接删除	错误 592	错误 592
搜索型直接删除	错误 582	错误 582
搜索型延迟删除	错误 582	错误 582
影响聚簇索引的搜索型删除（延迟）	错误 582	错误 582
update 命令		
定位型直接更新	保持打开	保持打开
搜索型直接更新	保持打开	保持打开
搜索型延迟更新	保持打开	保持打开
影响聚簇索引的搜索型更新（延迟）	错误 582	保持打开

连接列缓冲对连接游标的影响

对于包含连接的查询上游标，连接顺序中外部表的连接列、搜索参数和选择列表值通过第一个 fetch 被放置到缓冲区。如果从连接顺序中的内部表返回多行，后继读取将缓冲区内的值用于外部行。这意味着，通过游标更新连接列和搜索参数的应用程序行为，会随着为查询选择的连接顺序不同而改变。连接列缓冲对 allpages 锁表和 DOL 锁表都有影响。

注意：在 11.5.x 版中，allpages 锁表上的连接游标不会将连接值放入缓冲区。

游标扫描期间列缓冲的影响

如果在会话期间对连接列进行了更新，则视为查询选择的连接顺序不同，执行连接的游标查询可能会产生不同的结果。以下两表说明了连接顺序如何影响游标结果集。

```
select * from dept
dept_id deptloc
------- -----------------
      1 Elm Street
      2 Acacia Drive
      3 Maple Lane
      4 Oak Avenue
select * from employee
dept_id empid
------- ---------
      1 172321176
      1 213468915
      2 341221782
      2 409567008
      2 427172319
      3 472272349
      3 486291786
```

连接列缓冲示例

以下语句将定义一个游标，将其打开，然后读取第一行：

```
declare curs1 cursor for
select d.dept_id, e.empid, d.deptloc
from dept d , employee e
where d.dept_id = e.dept_id
```

```
and d.dept_id = 2
open j_curs
fetch j_curs
dept_id     empid       deptloc
----------- ----------- --------------------
          2   341221782 Acacia Drive
```

如果 dept 被选中作为连接顺序中的外部列，则 dept_id 的值 2 将被放入缓冲区。下面的 update 会改变 dept 中连接列的 dept_id：

```
update dept set dept_id = 12
where current of  j_curs
```

此更新改变存储在表行中的值，但不变更缓冲区中的值，导致游标得不到对基表的更新结果。如果 dept 中的连接列由非定位型更新进行了更新，影响是相同的。

以下 update 可更改 employee 行的 dept_id：

```
update employee set dept_id = 12
where current of j_curs
```

表上的下一次 fetch 将返回结果集中的第二行：

```
fetch j_curs
dept_id     empid       deptloc
----------- ----------- --------------------
          2   409567008 Acacia Drive
```

可以读取和更新 employee 的所有匹配行。

如果此游标的连接顺序选择 employee 作为外部表，且 dept 中的连接列在第一次执行 fetch 之后被更新，则第二次执行 fetch 将找不到匹配行。各步顺序如下：

1. employee.dept_id 的值 2 在第一次执行 fetch 期间将被放入缓冲区。
2. 一个搜索型或定位型更新将 dept.dept_id 的值改为 12。
3. 第二次执行 fetch 时将使用 employee.dept_id 的缓冲值 2，并扫描内部表以查找匹配值；而由于 dept.dept_id 已改为 12，第二次执行 fetch 将不会返回行。

dept_id 值为 2 的 employee 中的两个剩余行不能被游标读取。

搜索自变量缓冲示例

由于将搜索自变量加入了缓冲区，游标结果对游标选择查询上的搜索自变量更新的敏感性也取决于连接顺序。此例使用如第 562 页上的“游标扫描期间列缓冲的影响”所示的 dept 和 employee 表。下面的游标使用 deptloc 列中的搜索参数，在表的 dept_id 列上连接：

```
declare c cursor for
select d.dept_id, empid, deptloc
from dept d , employee e
where d.dept_id = e.dept_id
and deptloc = "Acacia Drive"
```

对表的第一次 fetch 将返回以下行：

```
dept_id     empid       deptloc
--------- ----------- --------------
        2 41221782    Acacia Drive
```

以下定位型更新语句将当前行的 employee.dept_id 值改为 4；需要对位于 Acacia Drive 的部门中的每名员工执行该语句：

```
update employee set dept_id = 4
where current of c
```

以下定位型更新将 dept.deptloc 值从“Acacia Drive”改为“Pine Way”，

```
update dept set deptloc = "Pine Way"
where current of c
```

如果是在第一次 fetch 后发出对 deptloc 的 update，则结果集可能会也可能不会返回需要更改的所有行；结果将取决于为查询选择的连接顺序：

- 如果选择 dept 作为外部表，dept.dept_id 和 dept.deptloc 列、值“2”和“Acacia Drive”将被放入缓冲区。第一个 fetch 后，对 deptloc 的 update 将更新基表，而不更新缓冲值，且后继的 fetch 命令返回其它的结果集行。
- 如果选择 employee 作为外部表，则将只有 employee 表上的 dept_id 限定值被放入缓冲区。对 deptloc 列的 update 将会更改基表，并在下一次执行 fetch，将搜索自变量限定值“Acacia Drive”应用到 deptloc 表时，不会返回任何行，即使表中仍有两个 employee.dept_id 为 2 的行时也是如此。

注意：在 employee 表中的所有行都被读取后，如果发出对 deptloc 的 update，则两种更新都会实现，而这与连接顺序无关。

select 列表缓冲的影响

连接顺序中外部表的选择列表中的各列，在外部表中一行被读取时将被送入缓冲区。如果对选择列表中的一列执行搜索型或定位型更新，并从内部表中读取另一行，外部表的缓冲值将出现在游标结果集行中。

建议

通常情况下，游标扫描正在进行时，应用程序会试图避免因更新连接列或带有搜索子句和其它谓词的列而更改它们的值。

- 还应尽力避免对游标的连接列的更新以及对连接游标的搜索参数的更新，除非已完全确定连接顺序。
- 使用创建工作表的游标查询，然后再对基表使用搜索型更新和删除。当游标选择查询要求一个工作表时，游标一般对底层表的改变不敏感。如果游标包含 order by、distinct、group by 或其它创建工作表的子句，则不能通过定位型更新来更新。
- 如果正在使用游标对两个连接列都进行更新，则考虑使用搜索型更新而不是定位型更新。尽管将来的读取可能返回缓冲值而不是已在数据行中改变的值，但使用搜索型更新能避免可能由选择的连接顺序而导致的 fetch 匹配行失败。allpages 锁表中对聚簇索引键的搜索型更新将会隐式地关闭游标。
- 注意：ANSI SQL 92 初级规范不允许对使用游标的连接列进行更新，因此这种行为要视具体实现而定。对于被设计为可跨不同数据库软件移植的应用程序，必须考虑其各自的实现情况。

第 18 章

事务：维护数据一致性和恢复

事务将一组 Transact-SQL 语句组合以便将它们作为一个单元处理。组中所有的语句都执行或都不执行。

本章讨论如下内容：

主题	页
事务工作方式	567
使用事务	570
选择事务模式和隔离级别	578
使用 lock table 命令改善性能	588
在存储过程和触发器中使用事务	589
在事务中使用游标	595
使用事务时应考虑的问题	597
事务备份和恢复	597

有关事务的详细信息，参见*性能和调优指南*。

事务工作方式

如同管理事务一样，Adaptive Server 自动管理所有数据修改命令，包括单步更改请求。缺省情况下，每个 insert、update 和 delete 语句被视为单个事务。

但是，考虑以下情况： Lee 需要对 authors、 titles 和 titleauthors 表进行一系列的数据检索和修改。当她操作时，Lil 开始更新 titles 表。Lil 的更新可能在 Lee 所执行的工作中导致不一致的结果。要防止发生这种情况，Lee 可以将其语句组合为单个事务，她从 Lil 的表中锁定 Lee 工作中用到的部分。这允许 Lee 基于精确的数据来完成其工作。Lee 完成对表的更新之后，Lil 才能对表进行更新。

可以使用如下命令来创建事务：

- begin transaction — 标记事务块的起点。此命令的语法为：

  ```
  begin {transaction | tran} [transaction_name]
  ```

  transaction_name 是指派给事务的名称。它必须符合标识符的规则。仅在最外面的一对嵌套的 begin/commit 或 begin/rollback 语句中使用事务名称。

- save transaction — 标记事务中的保存点：

  ```
  save {transaction | tran} savepoint_name
  ```

  savepoint_name 是指派给保存点的名称。它必须符合标识符的规则。

- commit — 提交整个事务：

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

- rollback — 将事务回退到保存点或事务的起点：

  ```
  rollback [transaction | tran | work]
  [transaction_name | savepoint_name]
  ```

例如，用户 Lee 开始为 *The Gourmet Microwave* 的两个作者更改 royalty split。因为两次更新之间数据库可能不一致，所以必须将其组合为一个事务，如下示例所述：

```
begin transaction royalty_change

update titleauthor
set royaltyper = 65
from titleauthor, titles
where royaltyper = 75
and titleauthor.title_id = titles.title_id
and title = "The Gourmet Microwave"

update titleauthor
set royaltyper = 35
from titleauthor, titles
where royaltyper = 25
and titleauthor.title_id = titles.title_id
and title = "The Gourmet Microwave"

save transaction percentchanged

/* After updating the royaltyper entries for
** the two authors, insert the savepoint
```

```
** percentchanged, then determine how a 10%
** increase in the bookís price would affect
** the authors?royalty earnings. */

update titles
set price = price * 1.1
where title = "The Gourmet Microwave"

select (price * total_sales) * royaltyper
from titles, titleauthor
where title = "The Gourmet Microwave"
and titles.title_id = titleauthor.title_id

/* The transaction is rolled back to the savepoint
** with the rollback transaction command. */

rollback transaction percentchanged

commit transaction
```

事务允许 Adaptive Server 保证：

- 一致性 — 同时进行的查询和更改请求不能相互冲突，并且用户不能查看或操作部分更改的数据。
- 恢复 — 在系统出现故障时，自动完成数据库恢复。

为了支持符合 SQL 标准的事务，Adaptive Server 允许为事务选择模式和隔离级别。应用程序要求符合 SQL 标准的事务在每个会话的起点设置这些选项。本章稍后描述了事务模式和隔离级别。有关详细信息，参见第 578 页上的“选择事务模式和隔离级别”。

事务和一致性

多用户环境中，Adaptive Server 必须防止相互冲突的同时查询和数据修改请求。因为如果查询处理的数据能够被另一用户的更新所更改，则查询结果可能不明确，所以这一点格外重要。

Adaptive Server 自动为每个事务设置相应的锁定级别。通过在 select 语句中包括 holdlock 关键字可以为基于逐个查询的更多限制共享锁。

事务和恢复

事务既是工作单元又是恢复单元。事实上，Adaptive Server 将单步更改请求作为事务来处理，这表示出现故障时数据库可以完全恢复。

Adaptive Server 恢复时间可用分钟和秒来计算。可以指定最大可接受的恢复时间。

在第 597 页上的“事务备份和恢复”中讨论了与恢复和备份有关的 SQL 命令。

注意：将大量 Transact-SQL 命令组合为一个长期事务可能影响恢复时间。如果提交事务之前 Adaptive Server 失败，由于 Adaptive Server 必须撤消此事务，所以将花费更长的时间恢复。

如果“组件集成服务”使用远程数据库，则处理事务的方式有所不同。有关详细信息，参见*组件集成服务用户指南*。

如果已经购买并安装了 Adaptive Server DTM 功能，则在多个服务器中更新数据的事务也受益于事务一致性。有关详细信息，参见*使用 Adaptive Server 分布式事务功能*。

使用事务

begin transaction 和 commit transaction 命令告诉 Adaptive Server 将任意数量的单独命令作为单个单元进行处理。rollback transaction 撤消事务，并回退到起点或**保存点**。使用 save transaction 在事务中定义一个保存点。

事务使您可以控制事务管理。另外，由于将 SQL 语句组合为单个单元进行处理，每个事务而非每个独立的命令导致每次系统开销，从而提高了性能。

任何用户都可以定义事务。任何事务命令都没有权限要求。

以下各节使用示例讨论了一般事务主题和事务命令。

在事务中允许使用数据定义命令

通过将 ddl in tran 数据库选项设置为真，可以在事务中使用某些数据定义语言命令。如果在某一数据库中 ddl in tran 为真，则在此数据库的事务中可以发出 create table、grant 和 alter table 等命令。如果在 model 数据库中的 ddl in tran 为真，对于在 model 中将 ddl in tran 设置为真之后所创建的所有数据库而言，允许在其事务中使用上述这些命令。要检查 ddl in tran 的当前设置，使用 sp_helpdb。

警告！使用数据定义命令时务必小心。唯一在事务中使用数据定义语言命令须经调整的情况是 create schema 进行对齐。数据定义语言命令持有系统表锁（例如 sysobjects）。如果事务中使用数据定义语言命令，应使事务保持简短。

在事务中避免对 tempdb 使用数据定义语言命令，那样会使系统降低性能直至停止。在 tempdb 中始终将 ddl in tran 设置为假。

要将 ddl in tran 设置为真，输入：

```
sp_dboption database_name,"ddl in tran", true
```

然后在数据库中执行 checkpoint 命令。

第一个参数指定所要设置选项的数据库名。执行 sp_dboption 时，必须使用 master 数据库。任何用户都可以执行不带参数的 sp_dboption 来显示当前选项设置。但是，要设置选项，您必须是系统管理员或数据库所有者。

只有 sp_dboption 的 ddl in tran 选项设置为真，才允许在事务中使用以下命令：

表 *18-1*：允许在事务中使用的 *DDL* 命令

alter table（除 partition 和 unpartition 之外允许的子句）	create default create index create procedure create rule create schema create table create trigger create view	drop default drop index drop procedure drop rule drop table drop trigger drop view	grant revoke

更改 master 数据库或创建临时表的系统过程不能用于事务中。

不能在事务中使用以下命令：

表 *18-2*：不允许在事务中使用的 *DDL* 命令

alter database	disk init	load transaction	select into
alter table...partition	dump database	load database	update statistics
alter table...unpartition	dump transaction	reconfigure	truncate table
create database	drop database		

不允许在事务中使用的系统过程

不能在事务中使用以下系统过程：

- sp_helpdb、sp_helpdevice、sp_helpindex、sp_helpjoins、sp_helpserver、sp_lookup 和 sp_spaceused（因为它们创建了临时表）
- sp_configure
- 更改 master 数据库的系统过程

开始和提交事务

begin transaction 和 commit transaction 命令可以包括任意数量的 SQL 语句和存储过程。这两个语句的语法为：

begin {transaction | tran} [*transaction_name*]

commit {transaction | tran | work} [*transaction_name*]

transaction_name 是指派给事务的名称。它必须符合标识符的规则。

关键字 transaction、tran 和 work（在 commit transaction 中）具有相同含义；可以在使用时相互替换。但是，transaction 和 tran 是 Transact-SQL 扩展中的关键字；而只有 work 是符合 SQL 标准的关键字。

以下是一个简例：

```
begin tran
    statement
    procedure
    statement
commit tran
```

如果事务当前不是活动的，则 commit transaction 不会影响 Adaptive Server。

回退和保存事务

如果提交之前必须取消此事务 — 由于某些故障或用户需要更改 — 那么必须撤消所有已经完成的语句或过程。有关处理期间回退的影响，参见第 592 页的表 18-4。

在 commit transaction 之前，随时可以使用 rollback transaction 命令来取消或回退事务。使用保存点，可以取消整个事务或部分事务。但是，不能取消已经提交的事务。

rollback transaction 的语法是：

rollback {transaction | tran | work}
[*transaction_name* | *savepoint_name*]

保存点是用户放在事务中用于表示事务可回退到此点的标记。提交整个批处理之前，可以通过将不需要的事务部分回退到保存点以实现提交部分批处理。

在事务中放入 save transaction 命令可以插入一个保存点。此命令的语法为：

save {transaction | tran} *savepoint_name*

此保存点名必须符合标识符的规则。

如果 rollback transaction 命令不带 *savepoint_name* 或 *transaction_name*，则事务将回退到此批处理中的第一个 begin transaction。

以下是使用 save transaction 和 rollback transaction 命令的方法：

```
begin tran
    statements                  Group A
    save tran mytran
        statements              Group B
    rollback tran mytran        Rolls back group B
    statements                  Group C
commit tran                      Commits groups A and C
```

直到发出 commit transaction 后，Adaptive Server 将后续语句作为事务的一部分，除非遇到另一个 begin transaction 语句。在此点，Adaptive Server 将后续语句作为新的嵌套事务的一部分。在第 576 页上的“嵌套事务”中，对嵌套事务进行了描述。

如果事务不是当前活动的，则 rollback transaction 或 save transaction 不会影响 Adaptive Server，也不返回错误消息。

也可以在存储过程或触发器中使用 save transaction 创建事务，这样，它们可被回退而不影响批处理或其它过程。例如：

```
create proc myproc as
begin tran
save tran mytran
statements
if ...
    begin
        rollback tran mytran
    /*
    ** Rolls back to savepoint.
    */
        commit tran
    /*
    ** This commit needed; rollback to a savepoint
    ** does not cancel a transaction.
    */
   end

else
commit tran
    /*
    ** Matches begin tran; either commits
    ** transaction (if not nested) or
    ** decrements nesting level.
    */
```

除非正向一个保存点回退，否则只能在最外层的一对 begin/commit 或 begin/rollback 语句中使用事务名。

警告！在嵌套事务语句中使用事务名时，它可能被忽略或导致错误。如果要在其它事务所调用的系统过程或触发器中使用事务，则不要使用其事务名。

检查事务的状态

全局变量 *@@transtate* 跟踪事务的当前状态。Adaptive Server 通过跟踪所有执行语句后的事务更改来确定返回的状态。*@@transtate* 可能包含以下值：

表 *18-3*：@@*transtate* 值

值	含义
0	事务正在进行 事务有效；已成功执行先前语句。
1	事务已经成功。事务完成并提交其更改。
2	语句已被中止。先前语句已被中止；事务无效。
3	事务已被中止。事务已被中止并回退任何更改。

Adaptive Server 不会清除每一语句之后的 @@*transtate*。在事务中，可以使用语句之后的 @@*transtate*（例如 insert）来确定事务是已成功还是被中止，从而确定事务是否有效。以下示例在事务期间和提交事务之后检查 @@*transtate*（在 insert 成功之后）：

```
begin transaction
insert into publishers (pub_id) values ("9999")
(1 row affected)
select @@transtate
----------
         0

(1 row affected)
commit transaction
select @@transtate
----------
         1

(1 row affected)
```

以下示例在 insert 不成功之后（由于规则冲突）和事务回退之后检查 @@*transtate*：

```
begin transaction
insert into publishers (pub_id) values ("7777")
Msg 552, Level 16, State 1:
A column insert or update conflicts with a rule bound
to the column. The command is aborted. The conflict
occured in database 'pubs2', table 'publishers',
rule 'pub_idrule', column 'pub_id'.
select @@transtate
----------
        2

(1 row affected)
rollback transaction
select @@transtate
----------
        3
```

```
(1 row affected)
```

Adaptive Server 不会清除每一语句之后的 *@@transtate*。只有 Adaptive Server 响应事务所采取的操作，它才更改 *@@transtate*。语法和编译错误不会影响 *@@transtate* 的值。

嵌套事务

可以在其它事务中嵌套事务。当嵌套 begin transaction 和 commit transaction 语句时，实际上由最外层对语句开始并提交事务。内部对只跟踪嵌套级。Adaptive Server 直到发出与最外层 begin transaction 相匹配的 commit transaction 时才提交此事务。通常，当包含 begin/commit 语句对的存储过程或触发器相互调用时，才发生这种事务“嵌套”。

@@trancount 全局变量跟踪事务的当前嵌套级别。begin transaction 最初隐式或显式地将 *@@trancount* 设置为 1。每个后续 begin transaction将*@@trancount*递增，而commit transaction将其递减。触发一个触发器也将使 *@@trancount* 递增，并且可用一个可以触发触发器的语句来开始事务。除非 *@@trancount* 为 0，否则不会提交嵌套事务。

例如，以下嵌套语句组直到最后的 commit transaction，才由 Adaptive Server 提交该事务：

```
begin tran
    select @@trancount
    /* @@trancount = 1 */

    begin tran
        select @@trancount
        /* @@trancount = 2 */

        begin tran
            select @@trancount
            /* @@trancount = 3 */
        commit tran

    commit tran

commit tran

select @@trancount
/* @@ trancount = 0 */
```

如嵌套 rollback transaction 语句时，不包括事务名或保存点名，则其将回退到最外层的 begin transaction 语句并取消此事务。

事务示例

此示例显示了指定事务的方法：

```
begin transaction royalty_change
/* A user sets out to change the royalty split */
/* for the two authors of The Gourmet Microwave. */
/* Since the database would be inconsistent */
/* between the two updates, they must be grouped */
/* into a transaction. */


update titleauthor
set royaltyper = 65
from titleauthor, titles
where royaltyper = 75
and titleauthor.title_id = titles.title_id
and title = "The Gourmet Microwave"


update titleauthor
set royaltyper = 35
from titleauthor, titles
where royaltyper = 25
and titleauthor.title_id = titles.title_id
and title = "The Gourmet Microwave"


save transaction percent_changed


/* After updating the royaltyper entries for */
/* the two authors, the user inserts the */
/* savepoint "percent_changed," and then checks */
/* to see how a 10 percent increase in the */
/* price would affect the authors?royalty */
/* earnings. */


update titles
set price = price * 1.1
where title = "The Gourmet Microwave"
```

```
select (price * royalty * total_sales) * royaltyper
from titles, titleauthor, roysched
where title = "The Gourmet Microwave"
and titles.title_id = titleauthor.title_id
and titles.title_id = roysched.title_id


rollback transaction percent_changed


/* The transaction rolls back to the savepoint */
/* with the rollback transaction command. */
/* Without a savepoint, it would roll back to */
/* the begin transaction. */


commit transaction
```

选择事务模式和隔离级别

Adaptive Server 提供以下选项来支持符合 SQL 标准的事务：

- **事务模式**可以设置是否使用隐式 begin transaction 语句开始事务。
- **隔离级别**指定事务期间数据可被其他用户访问的程度。

在符合 SQL 标准的事务所需的每个会话开始设置这些选项。

以下小节将详细描述这些选项。

选择事务模式

Adaptive Server 支持以下事务模式：

- 符合 SQL 标准的模式称为**链式**模式，它在任何数据检索或修改语句之前隐式地开始一个事务。这些语句包括： delete、insert、open、fetch、select 和 update。必须仍然使用 commit transaction 或 rollback transaction 来显式地结束事务。

- 缺省模式称为**非链式**模式或 Transact-SQL 模式，它要求显式地使用 begin transaction 语句和 commit transaction 或 rollback transaction 语句对来完成事务。

可使用 set 命令的 chained 选项设置每种模式。但是，不能在应用程序中混合使用这些事务模式。由于模式不同，存储过程和触发器可以有多种行为，并且所运行的过程要求特殊操作时，可以在一种模式中创建另一模式。

使用链式模式时，SQL 标准要求每个 SQL 数据检索和数据修改语句出现在事务中。开始会话之后或先前事务提交或中止之后，自动从第一个数据检索或数据修改语句开始一个事务。这是链式事务模式。

通过打开 set 语句的 chained 选项可以将当前会话设置为此模式：

```
set chained on
```

但是，不能在事务中执行 set chained 命令。要返回到非链式事务模式，将 chained 选项设置为 off。缺省事务模式为非链式。

在链式事务模式中，在后续数据检索或修改语句之前，Adaptive Server 隐式地执行一个 begin transaction 语句： delete、insert、open、fetch、select 和 update。例如，取决于所使用的模式，以下语句组产生不同的结果：

```
insert into publishers
    values ("9906", null, null, null)
begin transaction
delete from publishers where pub_id = "9906"
rollback transaction
```

在非链式事务模式中，rollback 只影响 delete 语句，因此 publishers 仍然包含所插入的行。在链式模式中，insert 语句隐式地开始一个事务，而且回退可影响所有从事务开始的语句，包括 insert。

所有的应用程序和即席用户查询都应该了解当前的事务模式。所用的事务模式取决于特定的查询或应用程序是否要求符合 SQL 标准。使用链式事务的应用程序（例如，Embedded SQL 预编译程序）应该在每个会话开始设置链式模式。

事务模式和嵌套事务

虽然链式模式使用数据检索或修改语句隐式地开始事务，但是只能通过显式地使用 begin transaction 语句嵌套事务。一旦隐式地开始第一个事务，则进一步的数据检索或修改语句不再开始新的事务，直到第一个事务提交或中止。例如，在以下查询中，第一次 commit transaction 提交链式模式中的所有更改；第二次提交则是不必要的：

```
insert into publishers
    values ("9907", null, null, null)
    insert into publishers
        values ("9908", null, null, null)
    commit transaction
commit transaction
```

注意：在链式模式中，无论数据检索或修改语句是否执行成功，都开始一个事务。甚至不访问任何表的 select 语句也开始一个事务。

查找当前事务模式的状态

可以检查全局变量 *@@tranchained* 来确定 Adaptive Server 当前的事务模式。select *@@tranchained* 在非链式模式中返回 0，而在链式模式中返回 1。

选择隔离级别

SQL92 标准为事务定义了四种隔离级别。每种隔离级别指定在执行并发事务时被禁止的操作种类。较高级别包括由较低级别所施加的限制：

- 级别 0 — 确保事务所写入的数据表示实际数据。它防止其它事务更改已经由未提交的事务所修改的数据（使用 insert、delete、update 等）。在未提交事务提交之前，其它事务不能修改其数据。但是，其它事务仍然可以读取未提交的数据，从而导致**脏读**。
- 级别 1 — 防止脏读。当第一个事务正修改某行，而在此事务提交更改之前，第二个事务读取了此行，便发生脏读。如果第一个事务回退了更改，则第二个事务读取的信息成为无效信息。这是 Adaptive Server 所支持的缺省隔离级别。

- 级别 2 — 防止**非重复读取**。当一个事务读取某行而第二个事务又对该行进行修改时会发生此类读取。如果第二个事务提交其更改，则第一个事务随后会读取到与原读取结果不同的结果。

  Adaptive Server 在 DOL 锁表中支持此级别。在 allpages 锁表中不支持此级别。

- 级别 3 — 确保事务结束之前所读取的数据是有效的，从而防止**幻像行**。Adaptive Server 通过 select 语句中的 holdlock 关键字来支持此级别，用于在指定数据上的读取锁。在某个事务读取了满足某搜索条件的一组行，而第二个事务修改了其中的相关数据（通过 insert、delete、update 等）时，便会发生幻像行。如果第一个事务按相同的搜索条件重复读取，它将获得一组不同的行。

使用 set 命令中的 transaction isolation level 选项可以在会话中设置隔离级别。可以在查询中强制隔离级别，而不是使用 select 语句中的 at isolation 子句。例如：

```
set transaction isolation level 0
```

Adaptive Server 和 SQL92 中的缺省隔离级别

缺省情况下，Adaptive Server 事务隔离级别为 1。而 SQL92 标准要求所有事务的缺省事务隔离级别为级别 3。这可防止脏读、非重复读取和幻像行。要强制此缺省隔离级别，Transact-SQL 提供了 set 语句中的 transaction isolation level 3 选项。此选项指示 Adaptive Server 在事务的所有 select 操作中应用 holdlock。例如：

```
set transaction isolation level 3
```

使用 transaction isolation level 3 的应用程序应该在每个会话开始设置隔离级别。但是，设置 transaction isolation level 3 将导致事务期间 Adaptive Server 始终持有所有的读取锁。如果同时还使用了链式事务模式，那么对于所有隐式开始事务的数据检索或修改语句，此隔离级别将继续有效。这两种情况下，由于较长时间持有更多的锁，从而导致应用程序的并发问题。

要将会话返回到 Adaptive Server 缺省隔离级别：

```
set transaction isolation level 1
```

脏读

如果在每个会话开始设置 transaction isolation level 0，那么访问同一数据时，不受脏读影响的应用程序具有较好的并发性并减少死锁的可能性。查找所有存储于表中储蓄帐号的即时平均余额的应用程序便是一例。因为它只要求当前平均余额的快照，而活动表中会对平均余额进行频繁的更改，所以应用程序应该使用隔离级别 0 来查询此表。其它要求数据一致性的应用程序（例如在表中指定的帐号上存款或取款）应该避免使用隔离级别 0。

以隔离级别 0 执行的扫描不要求为其扫描获取任何读取锁，因此不会阻止其它事务写入同一数据，反之亦然。但是，即使将隔离级别设置为 0，由于修改数据之前必须确保数据正确以维护数据完整性，所以实用程序（如 dbcc）和数据修改语句（如 update）仍然要求扫描的读取锁。

因为以隔离级别 0 执行的扫描不要求任何读取锁，所以隔离级别 0 扫描的结果集可以在扫描过程中发生更改。如果扫描位置因基础表中的更改而丢失，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。可以强制 Adaptive Server 选择非唯一索引或表扫描来替换此要求，如下所示：

select * from *table_name* (index *table_name*)

基础表上的操作可能在完成扫描之前中止。

可重复读取

执行可重复读取的事务在事务期间锁定所有被读取的行或页。当事务中的一个查询读取行后，其它事务不能更新或删除这些行，直到可重复读取事务完成。然而，可重复读取事务不象可串行化事务那样可通过执行域锁定提供幻像保护。其它事务可以插入那些能由可重复读取事务读取的值，并且可以更新行以便它们与可重复读取事务的搜索标准匹配。

执行可重复读取的事务锁定所有事务期间所读取的行或页。当事务中的查询读取行后，其它事务不能更新或删除这些行，直到可重复读取事务完成为止。但是，与可串行化事务不同，可重复读取事务不能通过执行域锁定来提供幻像保护。其它事务可以插入那些能由可重复读取事务读取的值，并且可以更新行以便它们与可重复读取事务的搜索标准相匹配。

注意：事务隔离级别 2 只支持 DOL 锁表。如果在 allpages 锁表使用事务隔离级别 2（可重复读取），则也将强制使用隔离级别 3（可串行化读取）。

要在会话级强制进行可重复读取，使用：

```
set transaction isolation level 2
```

或

```
set transaction isolation level repeatable read
```

要在查询中强制使用事务隔离级别 2，使用：

```
select title_id, price, advance
from titles
at isolation 2
```

或

```
select title_id, price, advance
from titles
at isolation repeatable read
```

只在事务级别上支持事务隔离级别 2。不能使用 select 或 readtext 语句中的 at isolation 子句将查询的隔离级别设置为 2。参见第 584 页上的“更改查询的隔离级别”。

有关事务隔离级别的详细信息，参见*性能和调优指南*。

查找当前隔离级别的状态

全局变量 *@@isolation* 包含 Transact-SQL 会话的当前隔离级别。查询 *@@isolation* 返回活动级别的值（0、1 或 3）。例如：

```
select @@isolation
--------
       1

(1 row affected)
```

有关隔离级别和锁定的详细信息，参见*性能和调优指南*。

更改查询的隔离级别

可通过使用 select 或 readtext 语句中的 at isolation 子句来更改查询的隔离级别。at isolation 子句支持隔离级别 0、1 和 3。而不支持隔离级别 2。at isolation 的 read uncommitted、read committed 和 serializable 选项表示的隔离级别如下表所示：

***at isolation* 选项**	**隔离级别**
read uncommited	0
read committed	1
serializable	3

例如，以下两条语句分别在隔离级别 0 和 3 查询同一个表：

```
select *
from titles
at isolation read uncommitted
select *
from titles
at isolation serializable
```

at isolation 只在单个 select 与 readtext 查询或 declare cursor 语句中才有效。如果使用 at isolation，则 Adaptive Server 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。

SQL-92 标准为 at isolation 和 set transaction isolation level 定义了 read uncommitted、read committed 和 serializable 选项。Transact-SQL 扩展也允许为 at isolation 指定隔离级别 0、1 和 3，但不能指定隔离级别 2。要简化隔离级别的讨论，本手册中的 at isolation 示例没有使用此扩展。

使用 select 语句的 holdlock 关键字也可以强制使用隔离级别 3。但是，不能在指定了 at isolation read uncommitted 的查询中指定 noholdlock 或 shared。（如果在查询中指定 holdlock 和隔离级别 0，则 Adaptive Server 发出警告并忽略 at isolation 子句）。使用不同方法设置隔离级别时，holdlock 关键字优先于 at isolation 子句（隔离级别 0 除外），at isolation 优先于 set transaction isolation level 所定义的会话级。有关隔离级别和锁定的详细信息，参见*性能和调优指南*。

隔离级别优先级

以下内容描述了定义隔离级别的不同方法所用的优先规则：

1. holdlock、noholdlock 和 shared 关键字优先于 at isolation 子句和 set transaction isolation level 选项（隔离级别 0 的情况除外）。例如：

```
/* This query executes at isolation level 3 */
select *
    from titles holdlock
    at isolation read committed
create view authors_nolock
    as select * from authors noholdlock
set transaction isolation level 3
/* This query executes at isolation level 1 */
select * from authors_nolock
```

2. at isolation 子句优先于 set transaction isolation level 选项。例如：

```
set transaction isolation level 2
/* executes at isolation level 0 */
select * from publishers
    at isolation read uncommitted
```

   在同一查询中不能既使用 at isolation 的 read uncommitted 选项，又使用 holdlock、noholdlock 和 shared 关键字。

3. set 命令的 transaction isolation level 0 选项优先于 holdlock、noholdlock 和 shared 关键字。例如：

```
set transaction isolation level 0
/* executes at isolation level 0 */
select *
    from titles holdlock
```

   Adaptive Server 执行上述查询之前发出警告。

游标和隔离级别

Adaptive Server 为游标提供了三种隔离级别：

- 级别 0 — Adaptive Server 在包含表示当前游标位置的行的基表页上不使用锁。游标不为扫描获取任何读取锁，因此不会阻止其它应用程序访问相同数据。但是，此隔离级别上操作的游标是不可更新的，并且它们需要基表上的唯一索引以确保扫描的精确性。
- 级别 1 — Adaptive Server 在包含表示当前游标位置的行的基表页上使用共享锁或更新锁。这些页保持被锁定，直到当前游标位置（由于 fetch 语句的结果）移出这些页或游标被关闭。如果使用索引来搜索基表行，则对相应的索引页也使用共享锁或更新锁。这是 Adaptive Server 中缺省的锁定行为。
- 级别 3 — 在事务中所读取用于代表游标的任何基表页上，Adaptive Server 使用共享锁或更新锁。另外，事务结束之前一直持有锁，而不会在不需要数据页时将其释放。与表或视图上的查询所指定的锁定级别相同，holdlock 关键字也将此锁定级别应用于基表。

隔离级别 2 不支持游标。

除了在隔离级别 3 中使用 holdlock，还可以使用 set transaction isolation level 为会话指定四种隔离级别之一。使用 set transaction isolation level 时，任何打开的游标使用指定的隔离级别，除非将事务隔离级别设置为 2。这种情况下，游标使用隔离级别 3。也可以使用 select 语句的 at isolation 子句为特定的游标指定隔离级别 0、1 或 3。例如：

```
declare commit_crsr cursor
for select *
from titles
at isolation read committed
```

此语句使游标在隔离级别 1 执行操作，并忽略事务或会话的隔离级别。如果声明隔离级别 0 的游标 (read uncommitted)，Adaptive Server 也将其定义为只读游标。不能在 declare cursor 语句中同时指定 for update 子句和 at isolation read uncommitted。

在打开游标（而不是声明游标）时，Adaptive Server 基于以下情况确定游标的隔离级别：

- 如果使用 at isolation 子句声明游标，则游标的隔离级别替换将其打开的事务隔离级别。

- 如果*未*使用 at isolation 声明游标，则游标使用将其打开的事务隔离级别。如果关闭后并重新打开游标，则游标获取当前事务的隔离级别。

Adaptive Server 在声明游标时编译此游标的查询。此编译过程在事务隔离级别 0 与隔离级别 1 或 3 中不同。如果使用事务隔离级别 1 或 3 声明 **language** 或 **client** 游标，则将其在事务隔离级别 0 中打开会导致错误。

例如：

```
set transaction isolation level 1
declare publishers_crsr cursor
    for select *
    from publishers
open publishers_crsr     /* no error */
fetch publishers_crsr
close publishers_crsr
set transaction isolation level 0
open publishers_crsr     /* error */
```

存储过程和隔离级别

Sybase 系统过程始终在隔离级别 1 执行操作，并忽略事务或会话的隔离级别。用户存储过程在执行它的事务隔离级别上进行操作。如果存储过程中更改隔离级别，则新的隔离级别只在执行存储过程期间有效。

触发器和隔离级别

因为触发器使用数据修改语句来触发（如 insert），所以所有触发器在事务隔离级别或隔离级别 1 两者之中较高的级别下执行。因此，如果触发器以事务隔离级别 0 触发，Adaptive Server 执行第一条语句之前将触发器设置为隔离级别 1。

符合 SQL 标准

要获取符合 SQL 标准的事务，必须在每个为随后事务更改模式和隔离级别的应用程序的开始 set chained 和 transaction isolation level 3 选项。如果应用程序使用游标，则也必须设置 close on endtran 选项。这些选项在第 595 页上的“在事务中使用游标”进行了说明。

使用 *lock table* 命令改善性能

lock table 命令允许访问表之前显式地请求事务期间要放置表锁。在即时表锁减少获取大量行锁或页锁的开销并节省整个锁定时间时，这是比较有用的。以下为此情况的示例：

- 在同一事务中可能将表扫描多次，并且每次扫描可能需要获取许多页锁或行锁。
- 扫描会超过表的锁升级阈值，因而会尝试逐步升级到表锁。

  如果未显式地请求表锁，则扫描获取页锁或行锁，直到其达到表的锁升级阈值（参见系统过程 sp_setpglockpromote）并在此点试图获取表锁。

lock table 命令的语法

lock table 命令的语法为：

```
lock table table_name in {share | exclusive } mode
        [ wait [no_of_seconds] | nowait ]
```

wait/nowait 选项允许指定此命令被阻塞时要获取表锁所等待的时间长度（参见第 589 页上的 “使用 lock table 命令的 wait/nowait 选项”）。

使用 lock table 时应考虑以下问题：

- 只能在事务中发出 lock table。
- 不能在系统表中使用 lock table。
- 可以首先在 share 模式中使用 lock table 将表锁定，然后将其锁升级到 exclusive 模式。
- 同一事务中可使用独立的 lock table 命令锁定多个表。
- 一旦获取表锁，使用 lock table 锁定的表与使用锁升级而不使用 lock table 命令锁定的表之间没有区别。

使用 *lock table* 命令的 *wait/nowait* 选项

使用 lock table 命令的 wait 选项来指定此命令等待获取表锁的时间长度。例如，以下事务中的命令将获取 titles 表锁的所等待时间设置为 2 秒：

```
lock table titles in share mode wait 2
```

如果获取表锁之前等待时间就结束了，则将与没有使用锁定表命令一样，继续事务并使用行锁定或页锁定。生成以下信息性消息（错误号 12207）：

```
Could not acquire a lock within the specified wait
period.COMMAND level wait...
```

有关事务期间错误处理的示例，参见*参考手册*中的 lock table。

注意：如果使用 lock table...wait 而没有指定 *no_of_seconds*，则此命令将无限期地等待锁。

在会话级和系统级可以设置等待锁的时间限制。使用 lock table 命令设置的等待时间将替换此两个级别的时间限制，如以下几节所述。

nowait 选项等价于等待 0 秒的 wait 选项：lock table 或者立即获取表锁，或者生成上述给定的信息性消息。如果没有获取锁，则将如同没有使用 lock table 命令一样，继续事务。

在存储过程和触发器中使用事务

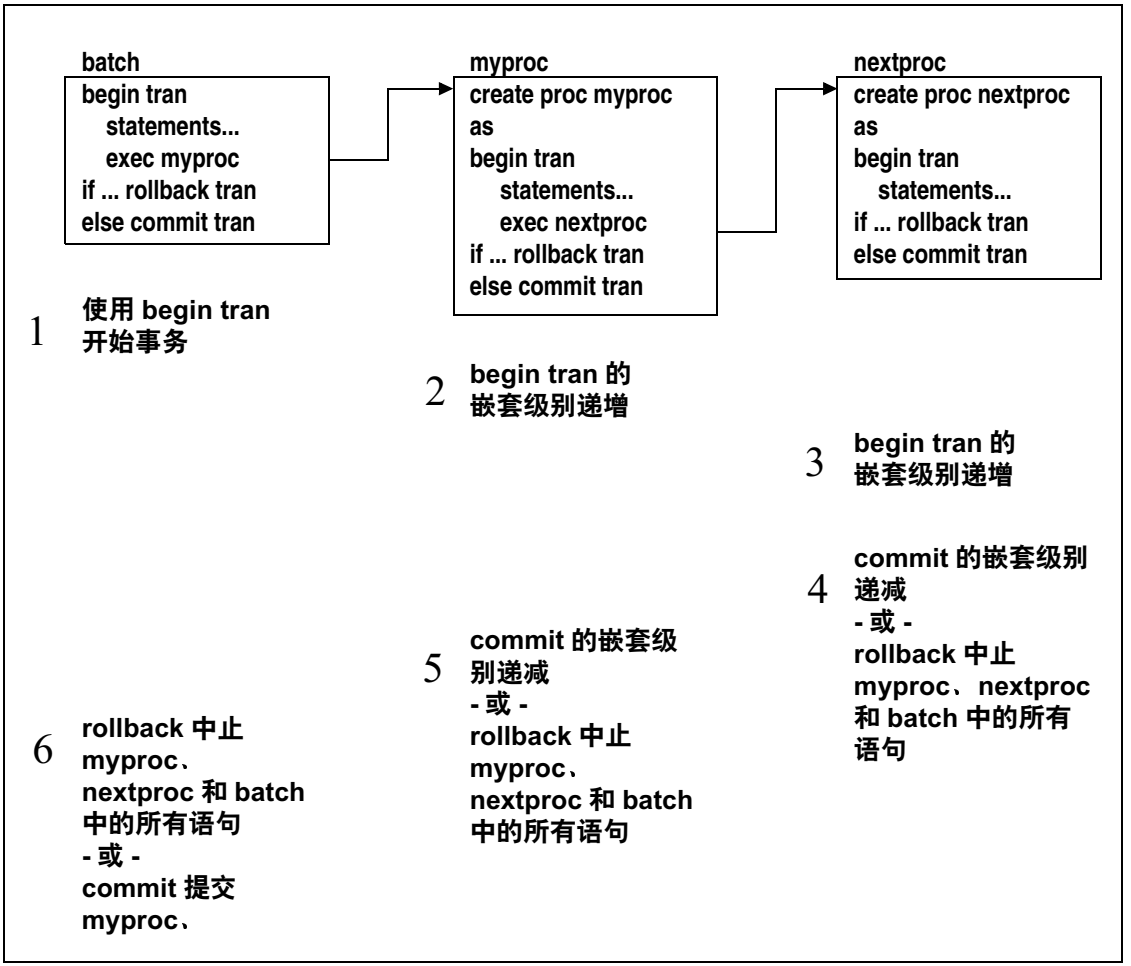
与使用语句批处理一样，可以在存储过程和触发器中使用事务。如果批处理或存储过程中的事务调用另一包含事务的存储过程或触发器，则第二个事务嵌套到第一个事务之中。

使用链式模式时，第一个显式或隐式 begin transaction 开始批处理、存储过程或触发器中的事务。每个后续 begin transaction 的嵌套级递增。每个后续 commit transaction 的嵌套级递减，直到嵌套级到 0 为止。然后 Adaptive Server 提交整个事务。rollback transaction 忽略其扩展存储过程和触发器跨越的嵌套级和嵌套数，中止整个事务，回退到第一个 begin transaction。

在存储过程和触发器中，begin transaction 语句数必须与 commit transaction 语句数相匹配。这也适用于使用链式模式的存储过程。第一个隐式开始事务的语句也必须有一个匹配的 commit transaction 语句。

图 18-1 演示了嵌套事务语句在存储过程中的处理方法：

图 *18-1*：嵌套事务语句



存储过程中的 rollback transaction 语句不影响过程中或最初由过程调用的批处理的随后语句。Adaptive Server 执行存储过程或批处理中后续语句。但是，触发器中的 rollback transaction 语句将中止批处理，因而不执行后续语句。

注意：触发器中的 rollback 语句可以： 1) 回退事务，2) 完成触发器中后续语句，并 3) 中止批处理因而*不*执行批处理中后续语句。

例如，以下的批处理调用存储过程 myproc，它包括了一个 rollback transaction 语句：

```
begin tran
update titles set ...
insert into titles ...
execute myproc
delete titles where ...
```

update 和 insert 语句被回退，事务被中止。Adaptive Server 继续批处理并执行 delete 语句。但是，如果表中有包括 rollback transaction 的 insert 触发器，则中止整个批处理并且不执行 delete。例如：

```
begin tran
update authors set ...
insert into authors ...
delete authors where ...
```

存储过程中不同的事务模式或隔离级别有不同的要求，在第 594 页上的“事务模式和存储过程”中进行了说明。由于调用触发器是作为数据修改语句的一部分，因而不受当前事务模式的影响。

错误和事务回退

影响数据完整性的错误能影响隐式或显式事务的状态：

- 严重级为 19 或更高的错误

  由于错误终止用户与服务器的连接，任何用户事务进行中出现的严重级为 19 或更高的错误将中止事务并回退所有的语句，直到最外层的 begin transaction 为止。Adaptive Server 在会话结束时始终回退所有未提交的事务。

- 影响数据完整性的数据修改命令中的错误（参见表 18-5）：

- 算术溢出和除零错误（使用 set arithabort arith_overflow 命令可以更改事务中的这些影响）
- 权限冲突
- 规则冲突
- 重复键冲突

表 18-4 总结了在几种不同环境中（如事务、存储过程或触发器中），rollback 如何影响 Adaptive Server 的处理：

表 *18-4*：回退如何影响处理

环境	*rollback* 的影响
仅事务	回退从所有事务开始的数据修改。如果事务跨越了多个批处理，则 rollback 将影响所有的批处理。 回退后将继续执行所有的命令。
仅存储过程	无。
事务中的存储过程	回退从所有事务开始的数据修改。如果事务跨越了多个批处理，则 rollback 将影响所有的批处理。 回退后将继续执行所有的命令。 存储过程生成错误消息 266： "Transaction count after EXECUTE indicates that a COMMIT or ROLLBACK TRAN is missing"。
仅触发器	触发器完成，但回退触发器的影响。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。
事务中的触发器	触发器完成，但回退触发器的影响。 回退从所有事务开始的数据修改。如果事务跨越了多个批处理，则 rollback 将影响所有的批处理。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。
嵌套触发器	内部触发器完成，但回退所有触发器的影响。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。
事务中的嵌套触发器	内部触发器完成，但回退所有触发器的影响。 回退从所有事务开始的数据修改。如果事务跨越了多个批处理，则 rollback 将影响所有的批处理。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。

在存储过程和触发器中，begin transaction 语句数必须与 commit 语句数相匹配。包含不成对的 begin/commit 语句的过程或触发器在执行时将生成警告消息。这也适用于使用链式模式的存储过程： 第一个隐式开始事务的语句必须有一个匹配的 commit 语句。

如果有重复键错误和规则冲突，（除非也有 return 语句）则触发器完成，并且执行 print、raiserror 语句和远程过程调用。然后，触发器和剩余的事务都将回退，并且中止剩余的批处理。rollback 语句不能回退在常规 SQL 事务（没有使用 DB-Library 两阶段提交）中执行的远程过程调用。

表 18-5 总结了几种不同环境中由重复键错误或规则冲突所导致的回退如何影响 Adaptive Server 处理：

表 *18-5*：由数据修改错误所导致的回退

环境	事务期间数据修改错误的影响
仅事务	当前命令被中止。不回退以前的命令，并执行后续命令。
存储过程中的事务	同上。
事务中的存储过程	同上。
事务中的存储过程	同上。
仅触发器	触发器完成，但回退触发器的影响。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。
事务中的触发器	触发器完成，但回退触发器的影响。 回退从所有事务开始的数据修改。如果事务跨越了多个批处理，则回退将影响所有的批处理。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。
嵌套触发器	内部触发器完成，但回退所有触发器的影响。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。
事务中的嵌套触发器	内部触发器完成，但回退所有触发器的影响。 回退从所有事务开始的数据修改。如果事务跨越了多个批处理，则回退将影响所有的批处理。 不执行批处理中剩余的任何命令。在下一个批处理恢复处理。
事务中的错误将回退触发器	回退触发器的影响。回退从所有事务开始的数据修改。如果事务跨越了多个批处理，则回退将影响所有的批处理。 触发器继续并导致重复键或规则错误。通常，触发器先回退其影响并继续执行，但是这种情况下不回退触发器的影响。 触发器完成之后，不执行批处理中剩余的任何命令。在下一个批处理恢复处理。
事务中的存储过程	同上。

事务模式和存储过程

使用非链式事务模式编写的存储过程可能与使用链式模式的事务不兼容，反之亦然。例如，以下是一个有效的使用链式事务模式的存储过程：

```
create proc myproc
as
insert into publishers
    values ("9996", null, null, null)
commit work
```

因为 commit 没有相应的 begin，所以使用非链式事务模式的程序调用此过程时将会失败。可能遇到其它问题：

- 使用链式模式开始事务的应用程序可能无法创建长期事务或在整个会话期间持有数据锁，那样会降低 Adaptive Server 的性能。
- 应用程序可能无限期地嵌套事务。依据不同的事务模式，这能够生成不同的结果。

一般来说，使用此事务模式的应用程序应该调用使用其它模式编写的存储过程。例外地，Sybase 系统过程（sp_procxmode 除外）可被使用任何事务模式的会话调用。（有关 sp_proxmode 的详细信息，参见第 595 页上的“设置存储过程的事务模式”。） 如果执行系统过程时没有事务是活动的，则在整个过程期间 Adaptive Server 关闭链式模式。返回之前，Adaptive Server 重新将模式设置为初始设置。

Adaptive Server 在创建过程的会话中使用事务模式（“链式”或“非链式”）来标记所有的过程。这有助于避免与使用某种模式的事务调用使用其它模式的事务相关的问题。标记为“链式”的存储过程不能在使用非链式事务模式的会话中执行，反之亦然。

触发器可在任何事务模式中执行。因为触发器始终被作为数据修改语句的一部分来调用，所以它们或者是链式事务的一部分（如果会话使用链式模式），或者保持其当前事务模式。

警告！使用事务模式时，务必了解每种设置对应用程序的影响。

设置存储过程的事务模式

使用 sp_procxmode 显示或更改存储过程的事务模式。例如，要将存储过程 byroyalty 的事务模式更改为“链式”，输入：

```
sp_procxmode byroyalty, "chained"
```

sp_procxmode "anymode" 使存储过程在链式或非链式事务模式下运行。例如：

```
sp_procxmode byroyalty, "anymode"
```

使用不带任何参数值的 sp_procxmode 来获取当前数据库中所有存储过程的事务模式：

```
sp_procxmode
procedure name                  transaction mode
--------------------------      --------------------
byroyalty                       Any Mode
discount_proc                   Unchained
history_proc                    Unchained
insert_sales_proc               Unchained
insert_salesdetail_proc         Unchained
storeid_proc                    Unchained
storename_proc                  Unchained
title_proc                      Unchained
titleid_proc                    Unchained

(9 rows affected, return status = 0)
```

只能在非链式事务模式中使用 sp_procxmode。

要更改过程的事务模式，必须是系统管理员、数据库所有者或过程所有者。

在事务中使用游标

缺省情况下，Adaptive Server 通过 commit 或 rollback 结束事务时，不更改游标的状态（打开或关闭）。但是，SQL 标准将一个打开的游标与其活动事务关联。提交或回退事务将自动关闭与其关联的任何打开的游标。

为强制进行这种符合 SQL 标准的行为，Adaptive Server 提供了 set 命令的 close on endtran 选项。另外，如果将链式模式设置为 on，Adaptive Server 打开游标时就开始事务，提交或回退事务时就关闭游标。

例如，缺省情况下，以下语句序列生成错误：

```
open test_crsr
commit tran
open test_crsr
```

如果将 close on endtran 或 chained 选项设置为 on，则 commit 之后游标的状态将从打开更改为关闭。这允许重新打开游标。

注意：因为通过游标可以返回客户应用程序缓冲行，并且允许用户在这些缓冲区中滚动，所以中止事务之后客户应用程序不应该反向滚动。因为 close on endtran 选项或链式模式强制执行事务回退（但客户端不知道），所以客户端高速缓存中的行变为无效行。

事务结束之前，事务中的游标一直持有任何所获取的排它锁。使用 holdlock 关键字、at isolation serializable 子句或 set isolation level 3 选项时，这也适用于共享锁。但是，如果不设置 close on endtran 选项，游标在事务结束之前将保持打开，而且其当前页锁保持有效。在读取其它行时，它也可以继续获取锁。

以下规则定义了使用事务游标进行更新的行为：

- 如果在显式事务中发生更新，则更新被作为事务的一部分。如果提交事务，也提交事务所包括的任何更新。如果中止事务，也回退事务所包括的任何更新。但被中止事务以外发生的相同游标进行的更新不受影响。
- 如果游标所作的更新出现在显式（和指定客户端）事务中，则 Adaptive Server 不会在游标关闭时提交它们。只有与此游标相关的事务结束时，才提交或回退待执行更新。
- 事务的提交或中止对不操纵结果行的 SQL 游标语句（例如 declare cursor、open cursor、 close cursor、set cursor rows 和 deallocate cursor）没有任何影响。例如，如果客户端在事务中打开一个游标，并中止了事务，则游标在中止之后保持打开（除非设置了 close on endtran 或使用了链式模式）。

使用事务时应考虑的问题

在应用程序中使用事务时应该考虑以下问题：

- rollback 语句中没有事务名或保存点名，会始终将语句回退到最外层的 begin transaction（显式或隐式）语句并取消此事务。当发出 rollback 时当前没有任何事务，则对语句没有任何影响。

  在触发器或存储过程中，rollback 语句中没有事务名或保存点名，将所有语句回退到最外层的 begin transaction（显式或隐式）。

- rollback 不对用户生成任何消息。如果需要警告，使用 raiserror 或 print 语句。
- 将大量 Transact-SQL 命令组合为一个长期事务可能影响恢复时间。如果长事务期间 Adaptive Server 失败，则由于 Adaptive Server 必须首先撤消整个事务，恢复时间会增加。
- 可以在事务中引用多达 16 个数据库。但是，Adaptive Server 可能引用内部数据库处理事务，所以实际引用的数据库数可以更少。如果打开的数据库超过限制，Adaptive Server 将显示一条错误消息。
- 远程过程调用 (RPC) 可以独立于任何包含它的事务来执行。在标准事务（即，不使用 Open Client DB-Library/C 两阶段提交的事务或 Adaptive Server 分布式管理功能）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 。
- 事务在客户应用程序和服务器之间不能扩展超过一个连接。例如，DB-Library/C 应用程序不能通过多个打开的 DBPROCESS 将 SQL 语句组合为一个事务。

事务备份和恢复

对数据库所作的每次更改，无论是单个 update 语句或一组 SQL 语句执行的结果，都被记录在系统表 syslogs 中。此表称为**事务日志**。

有些更改数据库的命令不能记录，例如 truncate table、批量复制数据到没有索引的表、select into、writetext 和 dump transaction with no_log。

事务日志实时记录 update、insert 或 delete 语句。事务开始时，在日志中记录了一个 begin transaction 事件。每条数据修改语句被接收时，都被记录到日志中。

对数据库本身作出更改之前，日志始终记录所有的数据更改。这种称为前写式日志可确保故障时可以完全恢复数据库。

硬件或媒体问题、系统软件问题、应用软件问题、程序指示的事务取消或用户取消事务都可能导致故障。

一旦出现这些故障，可以从使用 dump 命令的备份中恢复的数据库副本，再现事务日志。

要从故障中恢复，因为部分事务不是精确的更改，所以故障时正在进行但没有提交的事务必须撤消。如果不能保证完成的事务被写入数据库设备，则必须重新执行此事务。

如果 Adaptive Server 发生故障时事务是活动的、长期运行但没有提交的事务，则撤消其更改所要求的时间与事务已经运行的时间相同。不包含与 begin transaction 相匹配的 commit transaction 或 rollback transaction 的事务便是其中一例。这防止了 Adaptive Server 写入任何更改，但增加了恢复时间。

Adaptive Server 动态转储允许正在使用时数据库备份数据库和事务日志。经常备份数据库事务日志。越频繁地备份数据，发生系统故障时丢失的数据就越少。

虽然执行数据库备份的权限可以转交给其他用户，但是每个数据库所有者或具有 ss_oper 角色的用户负责使用 dump 命令来备份数据库和其事务日志。但是，缺省情况下，使用 load 命令的权限属于数据库所有者，而不能被转交。

一旦发出相应的 load 命令，Adaptive Server 将处理所有的恢复进程。Adaptive Server 也控制检查点间隔，检查点保证所有被更改的数据页都写入了数据库设备。必要时，用户可使用 checkpoint 命令强制执行检查点。

有关备份和恢复的详细信息，参见*参考手册*和*系统管理指南*。

第 19 章

锁定命令和选项

Adaptive Server 提供许多用于锁定的命令和选项。本章概述了可用锁定的类型以及如何通过 Transact-SQL 调用它们。

本章包括以下各节：

主题	页
设置等待锁的时间限制	599
队列处理的 readpast 锁定	602

设置等待锁的时间限制

Adaptive Server 允许指定锁等待时间以确定命令将等待多长时间才能获取锁：

- 可以使用 lock table 命令的 wait/nowait 选项指定获取表锁所等待的时间限制。
- 会话期间，可以使用 set lock 命令来指定会话期间发出的所有后续命令的锁等待时间。
- 存储过程中，可以使用 set lock 命令来指定存储过程中发出的所有后续命令的锁等待时间。
- 可以使用 sp_configure 的 lock wait period 选项来设置全服务器范围的锁等待时间。

lock table 命令的 *wait*/*nowait* 选项

事务中，lock table 命令不用等待命令获取足够的行级锁或页级锁以升级到表锁，就允许对表请求一个锁。

lock table 命令包含 wait/nowait 选项，用它可以指定，在其它事务中的操作归还目标表的所有锁前，命令要等待的时间长度。

lock table 命令的语法为：

lock table *table_name* in {share | exclusive } mode
[wait [*no_of_seconds*] | nowait]

例如，以下事务中的命令将获取 titles 表锁的所等待时间设置为 2 秒：

```
lock table titles in share mode wait 2
```

如果在获得表锁之前等待时间就结束了，事务将继续并使用行或页锁定，就如同没有 lock table 一样，并且生成以下信息性消息（错误号 12207）：

```
Could not acquire a lock within the specified wait
period.COMMAND level wait...
```

有关处理事务期间的错误消息的代码示例，参见*参考手册*中的 lock table。

注意：如果使用 lock table...wait 而没有指定 *no_of_seconds*，则此命令将无限期地等待锁。

如以下几节所述，在会话级和系统级可以设置等待锁的时间限制。使用 lock table 命令设置的等待时间会替换上述命令设置的等待时间

nowait 选项等价于等待 0 秒的 wait 选项：lock table 可能立即获取表锁，或生成以上给定的信息性消息。如果没有获取此锁，如同没有 lock table 命令一样，事务将继续进行。

可以在会话级或存储过程中使用 set lock 命令来控制任务等待获取锁的时间长度。

系统管理员可以使用 sp_configure 选项、lock wait period 来设置全服务器范围获取锁的时间限制。

设置会话级等待获取锁的时间限制

可使用 set lock wait 来控制会话或存储过程中命令等待获取锁的时间长度。此命令的语法为：

set lock { wait *no_of_seconds* | nowait }

no_of_seconds 是一个整数。因此，以下示例将会话级等待锁的时间限制设置为 5 秒：

```
set lock wait 5
```

但有一个例外，如果 set lock wait 时间在命令获取锁之前就结束了，则此命令失败，包含此命令的事务回退，并且生成以下错误消息：

```
Msg 12205, Level 17, State 2:
Server 'sagan'Line 1:
Could not acquire a lock within the specified wait
period. SESSION level wait period=300 seconds,
spid=12, lock type=shared page, dbid=9,
objid=2080010441, pageno=92300, rowno=0. Aborting
the transaction.
```

当事务中 lock table 命令设置的等待时间比 set lock wait 设置的时间更长时，出现例外。在这种情况下，事务在超时之前使用 lock table 等待时间，如上一节所述。

set lock nowait 选项与值为 0 的 set lock wait 选项等同。如果除 lock table 以外的命令不能立即获取请求的锁，则此命令失败，回退其事务，并且生成上述的错误消息。

如果全服务器范围的锁等待限制和会话级锁等待限制都被设置，则会话级限制优先。如果没有用 set lock wait 设置会话级等待时间，服务器级等待时间将被使用。

注意：存储过程不使用在服务器级或会话级设置的等待时间。除非使用 set lock wait 命令在存储过程内显式地指定时间限制，否则存储过程中各命令的等待时间是不受限制的。

设置全服务器范围的锁等待限制

系统管理员可以通过配置参数 lock wait period 来配置全服务器范围的锁等待限制。此命令的语法为：

sp_configure "lock wait period" [, no_of_seconds]

如果锁等待时间在命令获取锁之前就到期，除非有替换的 set lock wait 或 lock table 等待时间，否则此命令失败，回退包含其的事务，并且生成以下错误消息：

```
Msg 12205, Level 17, State 2:
Server 'wiz'Line 1:
Could not acquire a lock within the specified wait
period. SERVER level wait period=300 seconds,
spid=12, lock type=shared page, dbid=9,
objid=2080010441, pageno=92300, rowno=0. Aborting
the transaction.
```

通过 set lock wait 或 lock table wait 输入的时间限制将替换服务器级的锁等待时间。因此，例如，如果服务器级等待时间为 5 秒，而会话级等待时间为 10 秒，则 update 命令在命令失败和中止事务前将为获取锁等待 10 秒。

缺省服务器级锁等待时间为“wait forever”（永远等待）。要在设置有限等待时间之后恢复缺省值，可使用 sp_configure 来设置 lock wait period 的值，如下所示：

```
sp_configure "lock wait period", 0, "default"
```

锁等待超时次数的信息

sp_sysmon 报告等待获取锁的任务在指定时间内没有获取锁的次数。有关详细信息，参见*性能和调优指南*。

队列处理的 readpast 锁定

Readpast 锁定是 select 和 readtext 命令以及数据修改命令 update、delete 与 writetext 的可选选项。它指示命令自动地跳过遇到的所有不兼容锁，而不会阻塞、终止或生成消息。它主要用于表中的行组成一个队列的情况。在这种情况下，许多任务可能访问表来处理排队后的行，例如，这些行可以代表排队后的客户或客户订单。给定的任务不处理队列中的指定成员，但可处理任何满足选择标准的队列成员。

Readpast 语法

使用 readpast 锁定的语法是：

```
{ select | update | delete } ...
    from tablename [ holdlock | noholdlock ]
        [ readpast ] [ shared ]
        ...

readtext [[database.]owner.]table_name.column_name
    text_pointer offset size
    [holdlock | noholdlock]  [shared] [readpast]
        ...
```

writetext [[*database*.]*owner*.]*table_name*.*column_name*
text_pointer [readpast] [with log] *data*

Readpast 查询期间的不兼容锁

对于 select 和 readtext 命令，不兼容锁是排它锁。因此，select 和 readpast 命令可以访问持有共享锁或升级锁的任何行或页。

对于 delete、update 和 writetext 命令，任何类型的页锁或行锁都是不兼容的，所以：

- 在数据行锁定表中将跳过任何带有共享更新或排它行锁的行并且
- 在数据页锁定表中将跳过所有带有共享更新或排它锁的页。

如果存在排它表锁，除了在事务隔离级别 0 下执行的 select 命令之外，将阻塞所有指定 readpast 的命令。

allpages 锁表和 readpast 查询

如果对 allpages 锁表指定了 readpast 选项，将忽略 readpast 选项。此命令在为此命令或会话指定的隔离级别下执行：

- 如果隔离级别是0，则执行脏读，命令从锁定行返回值并且不会阻塞。
- 如果隔离级别为 1 或 3，则当必须读取具有不兼容锁的页时，命令将阻塞。

带 readpast 的隔离级别 *select* 查询的作用

Readpast 锁定被设计为在事务隔离级别 1 或 2 中使用。

会话级事务隔离级别和 readpast

对于 DOL 锁表，readpast 在 select 命令中对表的效果如表 19-1 所述。

表 *19-1*：会话级隔离级别和使用 *readpast*

会话隔离级别	效果
0, read uncommitted（脏读）	将忽略 readpast，向用户返回包含未提交事务的行。输出警告消息。
1, read committed	将跳过带有不兼容锁的行或页；在被读取的行或页上不持有锁。
2, repeatable read	将跳过带有不兼容锁的行或页；所读取的所有行或页上都持有共享锁，直到语句或事务结束为止。
3, serializable	将忽略 readpast，命令在级别 3 执行。命令将在任何带有不兼容锁的行或页上阻塞。

查询级的隔离级别和 readpast

如果指定 readpast 的 select 命令还包括以下任何子句，该命令将失败并显示错误消息：

- at isolation 子句，指定 0 或 read uncommitted
- at isolation 子句，指定 3 或 serializable
- 同一表上的 holdlock 关键字

如果已指定 readpast 的 select 查询也指定 at isolation 2 或 at isolation repeatable read，则将在 readpast 表中持有共享锁，直到语句或事务完成为止。

包括 readpast 并指定 at isolation read uncommitted 的 readtext 命令发出警告消息，然后自动在隔离级别 0 下运行。

带有 *readpast* 和隔离级别的数据修改命令

如果会话的事务隔离级别为 0，则使用 readpast 的 delete、update 和 writetext 命令不会发出警告消息。

- 对于数据页锁定表，这些命令将修改所有页上没有被不兼容锁锁定的所有行。
- 对于数据行锁定表，它们影响所有没有被不兼容锁锁定的行。

如果会话的事务隔离级别是 3（可串行化读取），当遇到带有不兼容锁的行或页时，使用 readpast 的 delete、update 和 writetext 命令将自动阻塞。

在事务隔离级别 2（可串行化读取），delete、update 和 writetext 命令：

- 修改没有被不兼容锁锁定的所有页中的所有行。
- 对于数据行锁定表，它们影响所有没有被不兼容锁锁定的行。

text 和 *image* 列和 readpast

如果带有 readpast 选项的 select 命令遇到带有不兼容锁的文本列，则 readpast 锁定将检索行，但返回空值文本列。在这种情况下，由于列被锁定，包含空值的文本列和返回空值文本列之间没有区别。

如果将带有 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，则 readpast 锁定将跳过此行。如果此列没有不兼容锁，此命令将获取一个锁并修改此列。然后，如果行中的任何后续文本列带有不兼容锁，此命令将阻塞，直到它可以获取锁并修改此列为止。

如果所有行中的文本列都带有不兼容锁，则具有 readpast 选项的 delete 命令将跳过此行。

Readpast 锁定示例

以下示例说明了 readpast 锁定。

要跳过对其持有排它锁的所有行：

```
select * from titles readpast
```

要只更新没有被其它会话锁定的行：

```
update titles
    set price = price * 1.1
    from titles readpast
```

要对 titles 表而不是 authors 或 titleauthor 表使用 readpast 锁定：

```
select *
    from titles readpast, authors, titleauthor
    where titles.title_id = titleauthor.title_id
    and authors.au_id = titleauthor.au_id
```

要只删除没有在 stores 表中锁定的行，但允许扫描在 authors 表中阻塞：

```
delete stores from stores readpast, authors
where stores.city = authors.city
```

附 录 A

pubs2 数据库

本附录描述样本数据库 pubs2。 该数据库包含表 publishers、authors、titles、titleauthor、au_pix、salesdetail、sales、stores、discounts、roysched 和 blurbs。

pubs2 数据库也包含主键与外键、规则、缺省值、视图、触发器及存储过程。

第 634 页的图 A-2 显示了 pubs2 数据库的框图。

有关安装 pubs2 的信息，参见适用所用平台的*配置 Adaptive Server Enterprise*。

pubs2 数据库中的表

以下几节详细介绍 pubs2 的每个表。 这些表中，每个列标题指定了列名、数据类型（包括任何用户定义的数据类型）和空或非空状态。 列标题还指定了影响该列的所有缺省值、规则、触发器和索引。

publishers 表

publishers 定义如下：

```
create table publishers
(pub_id char(4) not null,
pub_name varchar(40) not null,
city varchar(20) null,
state char(2) null)
```

其主键为 pub_id：

```
sp_primarykey publishers, pub_id
```

其 pub_idrule 规则定义如下：

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

表 A-1 列出 publishers 的内容：

表 *A-1*：*publishers* 表

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

authors 表

authors 定义如下：

```
create table authors
(au_id id not null,
au_lname varchar(40) not null,
au_fname varchar(20) not null,
phone char(12) not null,
address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null)
```

其主键为 au_id：

```
sp_primarykey authors, au_id
```

au_lname 和 au_fname 列的非聚簇索引定义如下：

```
create nonclustered index aunmind
on authors (au_lname, au_fname)
```

phone 列有如下缺省值：

```
create default phonedflt as "UNKNOWN"
sp_bindefault phonedft, "authors.phone"
```

如下视图使用 authors：

```
create view titleview
as
select title, au_ord, au_lname,
```

```
price, total_sales, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

表 A-2 列出 authors 的内容：

表 *A-2*：*authors* 表

au_id	au_-lname	au_fname	phone	address	city	state	coun-try	post-alcode
172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA	USA	94025
213-46-8915	Green	Marjorie	510 986-7020	309 63rd St. #411	Oakland	CA	USA	94618
238-95-7766	Carson	Cheryl	510 548-7723	589 Darwin Ln.	Berkeley	CA	USA	94705
267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose	CA	USA	95128
274-80-9391	Straight	Dick	510 834-2919	5420 College Av.	Oakland	CA	USA	94609
341-22-1782	Smith	Meander	913 843-0462	10 Mississippi Dr.	Lawrence	KS	USA	66044
409-56-7008	Bennet	Abraham	510 658-9932	6223 Bateman St.	Berkeley	CA	USA	94705
427-17-2319	Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA	USA	94301
472-27-2349	Gringlesby	Burt	707 938-6445	PO Box 792	Covelo	CA	USA	95428
486-29-1786	Locksley	Chastity	415 585-4620	18 Broadway Av.	San Francisco	CA	USA	94130
527-72-3246	Greene	Morningstar	615 297-2723	22 Graybar House Rd.	Nashville	TN	USA	37215
648-92-1872	Blotchet-Halls	Reginald	503 745-6402	55 Hillsdale Bl.	Corvallis	OR	USA	97330
672-71-3249	Yokomoto	Akiko	415 935-4228	3 Silver Ct.	Walnut Creek	CA	USA	94595
712-45-1867	del Castillo	Innes	615 996-8275	2286 Cram Pl. #86	Ann Arbor	MI	USA	48105
722-51-5454	DeFrance	Michel	219 547-9982	3 Balding Pl.	Gary	IN	USA	46403
724-08-9931	Stringer	Dirk	510 843-2991	5420 Telegraph Av.	Oakland	CA	USA	94609

au_id	au_-lname	au_fname	phone	address	city	state	coun-try	post-alcode
724-80-9391	MacFeather	Stearns	510 354-7128	44 Upland Hts.	Oakland	CA	USA	94612
756-30-7391	Karsen	Livia	510 534-9219	5720 McAuley St.	Oakland	CA	USA	94609
807-91-6654	Panteley	Sylvia	301 946-8853	1956 Arlington Pl.	Rockville	MD	USA	20853
846-92-7186	Hunter	Sheryl	415 836-7128	3410 Blonde St.	Palo Alto	CA	USA	94301
893-72-1158	McBadden	Heather	707 448-4982	301 Putnam	Vacaville	CA	USA	95688
899-46-2035	Ringer	Anne	801 826-0752	67 Seventh Av.	Salt Lake City	UT	USA	84152
998-72-3567	Ringer	Albert	801 826-0752	67 Seventh Av.	Salt Lake City	UT	USA	84152

titles 表

titles 定义如下：

```
create table titles
(title_id tid not null,
title varchar(80) not null,
type char(12) not null,
pub_id char(4) null,
price money null,
advance money null,
total_sales int null,
notes varchar(200) null,
pubdate datetime not null,
contract bit not null)
```

其主键为 title_id：

```
sp_primarykey titles, title_id
```

其 pub_id 列为 publishers 表的外键：

```
sp_foreignkey titles, publishers, pub_id
```

title 列的非聚簇索引定义如下：

```
create nonclustered index titleind
on titles (title)
```

其 title_idrule 定义如下：

```
create rule title_idrule
as
@title_id like "BU[0-9][0-9][0-9][0-9]" or
@title_id like "[MT]C[0-9][0-9][0-9][0-9]" or
@title_id like "P[SC][0-9][0-9][0-9][0-9]" or
@title_id like "[A-Z][A-Z]xxxx" or
@title_id like "[A-Z][A-Z]yyyy"
```

type 列有如下缺省值：

```
create default typedflt as "UNDECIDED"
sp_bindefault typedflt, "titles.type"
```

pubdate 列有如下缺省值：

```
create default datedflt as getdate()
sp_bindefault datedflt, "titles.pubdate"
```

titles 使用如下触发器：

```
create trigger deltitle
on titles
for delete
as
if (select count(*) from deleted, salesdetail
where salesdetail.title_id = deleted.title_id) >0
begin
   rollback transaction
   print "You canít delete a title with sales."
end
```

如下视图使用 titles：

```
create view titleview
as
select title, au_ord, au_lname,
price, total_sales, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

表 A-3 列出 titles 的内容：

表 *A-3*：*titles* 表

title-_id	title	type	pub-_id	price	ad-vance	total_-sales	notes	pub-date	con-tract
BU1032	The Busy Executive's Database Guide	business	1389	19.99	5000.00	4095	An overview of available database systems with emphasis on common business appli-cations. Illustrated.	Jun 12, 1986	1
BU1111	Cooking with Computers: Surrepti-tious Balance Sheets	business	1389	11.95	5000.00	3876	Helpful hints on how to use your electronic resources to the best advantage.	Jun 9, 1988	1
BU2075	You Can Combat Computer Stress!	business	0736	2.99	10125.00	18722	The latest medical and psycho-logical techniques for living with the electronic office. Easy-to-understand explana-tions.	Jun 30, 1985	1
BU7832	Straight Talk About Computers	business	1389	19.99	5000.00	4095	Annotated analysis of what computers can do for you: a no-hype guide for the critical user.	Jun 22, 1987	1

title-_id	title	type	pub-_id	price	ad-vance	total_-sales	notes	pub-date	con-tract
MC2222	Silicon Valley Gastro-nomic Treats	mod_-cook	0877	19.99	0.00	2032	Favorite recipes for quick, easy, and elegant meals; tried and tested by people who never have time to eat, let alone cook.	Jun 9, 1989	1
MC3021	The Gourmet Microwave	mod_-cook	0877	2.99	15000.00	22246	Traditional French gourmet recipes adapted for modern microwave cooking.	Jun 18, 1985	1
PC1035	But Is It User Friendly?	popular_comp	1389	22.95	7000.00	8780	A survey of software for the naive user, focusing on the "friendli-ness" of each.	Jun 30, 1986	1
MC3026	The Psychology of Computer Cooking	UNDE-CIDED	0877	NULL	NULL	NULL	NULL	Jul 24, 1991	0

title-_id	title	type	pub-_id	price	ad-vance	total_-sales	notes	pub-date	con-tract
PC8888	Secrets of Silicon Valley	popular_comp	1389	20.00	8000.00	4095	Muckraking reporting by two courageous women on the world's largest computer hardware and software manufacturers.	Jun 12, 1987	1
PC9999	Net Etiquette	popular_comp	1389	NULL	NULL	NULL	A must-read for computer conferencing debutantes!	Jul 24, 1996	0
PS1372	Computer Phobic and Non-Phobic Individuals: Behavior Variations	psychology	0877	21.59	7000.00	375	A must for the specialist, this book examines the difference between those who hate and fear computersand those who think they are swell.	Oct 21,1990	1

title-_id	title	type	pub-_id	price	ad-vance	total_-sales	notes	pub-date	con-tract
PS2091	Is Anger the Enemy?	psycho-logy	0736	10.95	2275.00	2045	Carefully researcheds tudy of the effects of strong emotions on the body. Metabolic charts included.	Jun 15, 1989	1
PS2106	Life Without Fear	psycho-logy	0736	7.00	6000.00	111	New exercise, meditat-ion, and nutritionalte chniquestha t can reduce the shock of daily inter-actions. Popular audience. Sample menus included. Exercise video available separately.	Oct 5, 1990	1
PS3333	Prolonged Data Depriva-tion: Four Case Studies	psycho-logy	0736	19.99	2000.00	4072	What happens when the data runs dry? Searching evaluat-ions of informat-ion-shortage effects on heavy users.	Jun 12, 1988	1

title-_id	title	type	pub-_id	price	ad-vance	total_-sales	notes	pub-date	con-tract
PS7777	Emotional Security: A New Algorithm	psycho-logy	0736	7.99	4000.00	3336	Protecting yourself and your loved ones from undue emotional stress in the modern world. Use of computer and nutritionalai ds empha-sized.	Jun 12, 1988	1
TC3218	Onions, Leeks, and Garlic: Cooking Secrets of the Mediterra-nean	trad_-cook	0877	20.95	7000.00	375	Profusely illustrated in color, this makes a wonderful gift book for a cuisine-oriented friend.	Oct 21, 1990	1
TC4203	Fifty Years in Bucking-ham Palace Kitchens	trad_coo k	0877	11.95	4000.00	15096	More anecdotes from the Queen's favorite cook describing life among English royalty. Recipes, tech-niques, tender vignettes.	Jun 12, 1985	1

title-_id	title	type	pub-_id	price	ad-vance	total_-sales	notes	pub-date	con-tract
TC7777	Sushi, Anyone?	trad_cook	0877	14.99	8000.00	4095	Detailed instruc-tions on improving your position in life by learning how to make authentic Japanese sushi in your spare time. 5-10% increase in number of friends per recipe reported from beta test.	Jun 12, 1987	1

titleauthor 表

titleauthor 定义如下：

```
create table titleauthor
(au_id id not null,
title_id tid not null,
au_ord tinyint null,
royaltyper int null)
```

其主键为 au_id 和 title_id：

```
sp_primarykey titleauthor, au_id, title_id
```

其 title_id 和 au_id 列为 titles 和 authors 的外键：

```
sp_foreignkey titleauthor, titles, title_id
sp_foreignkey titleauthor, authors, au_id
```

au_id 列的非聚簇索引定义如下：

```
create nonclustered index auidind
on titleauthor(au_id)
```

title_id 列的非聚簇索引定义如下：

```
create nonclustered index titleidind
on titleauthor(title_id)
```

如下视图使用 titleauthor：

```
create view titleview
as
select title, au_ord, au_lname,
price, total_sales, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

如下过程使用 titleauthor：

```
create procedure byroyalty @percentage int
as
select au_id from titleauthor
where titleauthor.royaltyper = @percentage
```

表 A-4 列出 titleauthor 的内容：

表 *A-4*：*titleauthor* 表

au_id	title_id	au_ord	royaltyper
172-32-1176	PS3333	1	100
213-46-8915	BU1032	2	40
213-46-8915	BU2075	1	100
238-95-7766	PC1035	1	100
267-41-2394	BU1111	2	40
267-41-2394	TC7777	2	30
274-80-9391	BU7832	1	100
409-56-7008	BU1032	1	60
427-17-2319	PC8888	1	50
472-27-2349	TC7777	3	30
486-29-1786	PC9999	1	100
486-29-1786	PS7777	1	100
648-92-1872	TC4203	1	100
672-71-3249	TC7777	1	40
712-45-1867	MC2222	1	100

au_id	title_id	au_ord	royaltyper
722-51-5454	MC3021	1	75
724-80-9391	BU1111	1	60
724-80-9391	PS1372	2	25
756-30-7391	PS1372	1	75
807-91-6654	TC3218	1	100
846-92-7186	PC8888	2	50
899-46-2035	MC3021	2	25
899-46-2035	PS2091	2	50
998-72-3567	PS2091	1	50
998-72-3567	PS2106	1	100

salesdetail 表

salesdetail 定义如下：

```
create table salesdetail
(stor_id char(4) not null,
ord_num numeric(6,0),
title_id tid not null,
qty smallint not null,
discount float not null)
```

其主键为 stor_id 和 ord_num：

```
sp_primarykey salesdetail, stor_id, ord_num
```

title_id、stor_id 和 ord_num 列为 titles 和 sales 的外键：

```
sp_foreignkey salesdetail, titles, title_id
sp_foreignkey salesdetail, sales, stor_id, ord_num
```

title_id 列的非聚簇索引定义如下：

```
create nonclustered index titleidind
on salesdetail (title_id)
```

stor_id 列的非聚簇索引定义如下：

```
create nonclustered index salesdetailind
on salesdetail (stor_id)
```

其 title_idrule 规则定义如下：

```
create rule title_idrule
as
@title_id like "BU[0-9][0-9][0-9][0-9]" or
```

```
@title_id like "[MT]C[0-9][0-9][0-9][0-9]" or
@title_id like "P[SC][0-9][0-9][0-9][0-9]" or
@title_id like "[A-Z][A-Z]xxxx" or
@title_id like "[A-Z][A-Z]yyyy"
```

salesdetail 使用如下触发器：

```
create trigger totalsales_trig on salesdetail
    for insert, update, delete
as
/* Save processing:  return if there are no rows
affected */
if @@rowcount = 0
    begin
       return
end
/* add all the new values */
/* use isnull:  a null value in the titles table
means
**              "no sales yet" not "sales unknown"
*/
update titles
    set total_sales = isnull(total_sales, 0) +
(select sum(qty)
    from inserted
    where titles.title_id = inserted.title_id)
   where title_id in (select title_id from inserted)
/* remove all values being deleted or updated */
update titles
    set total_sales = isnull(total_sales, 0) -
(select sum(qty)
    from deleted
    where titles.title_id = deleted.title_id)
   where title_id in (select title_id from deleted)
```

表 A-5 列出 salesdetail 的内容：

表 *A-5*：*salesdetail* 表

stor_id	ord_num	title_id	qty	discount
7896	234518	TC3218	75	40.000000
7896	234518	TC7777	75	40.000000
7131	Asoap432	TC3218	50	40.000000
7131	Asoap432	TC7777	80	40.000000
5023	XS-135-DER-432-8J2	TC3218	85	40.000000
8042	91-A-7	PS3333	90	45.000000
8042	91-A-7	TC3218	40	45.000000

stor_id	ord_num	title_id	qty	discount
8042	91-A-7	PS2106	30	45.000000
8042	91-V-7	PS2106	50	45.000000
8042	55-V-7	PS2106	31	45.000000
8042	91-A-7	MC3021	69	45.000000
5023	BS-345-DSE-860-1F2	PC1035	1000	46.700000
5023	AX-532-FED-452-2Z7	BU2075	500	46.700000
5023	AX-532-FED-452-2Z7	BU1032	200	46.700000
5023	AX-532-FED-452-2Z7	BU7832	150	46.700000
5023	AX-532-FED-452-2Z7	PS7777	125	46.700000
5023	NF-123-ADS-642-9G3	TC7777	1000	46.700000
5023	NF-123-ADS-642-9G3	BU1032	1000	46.700000
5023	NF-123-ADS-642-9G3	PC1035	750	46.700000
7131	Fsoap867	BU1032	200	46.700000
7066	BA52498	BU7832	100	46.700000
7066	BA71224	PS7777	200	46.700000
7066	BA71224	PC1035	300	46.700000
7066	BA71224	TC7777	350	46.700000
5023	ZD-123-DFG-752-9G8	PS2091	1000	46.700000
7067	NB-3.142	PS2091	200	46.700000
7067	NB-3.142	PS7777	250	46.700000
7067	NB-3.142	PS3333	345	46.700000
7067	NB-3.142	BU7832	360	46.700000
5023	XS-135-DER-432-8J2	PS2091	845	46.700000
5023	XS-135-DER-432-8J2	PS7777	581	46.700000
5023	ZZ-999-ZZZ-999-0A0	PS1372	375	46.700000
7067	NB-3.142	BU1111	175	46.700000
5023	XS-135-DER-432-8J2	BU7832	885	46.700000
5023	ZD-123-DFG-752-9G8	BU7832	900	46.700000
5023	AX-532-FED-452-2Z7	TC4203	550	46.700000
7131	Fsoap867	TC4203	350	46.700000
7896	234518	TC4203	275	46.700000
7066	BA71224	TC4203	500	46.700000
7067	NB-3.142	TC4203	512	46.700000
7131	Fsoap867	MC3021	400	46.700000
5023	AX-532-FED-452-2Z7	PC8888	105	46.700000
5023	NF-123-ADS-642-9G3	PC8888	300	46.700000
7066	BA71224	PC8888	350	46.700000

stor_id	ord_num	title_id	qty	discount
7067	NB-3.142	PC8888	335	46.700000
7131	Asoap432	BU1111	500	46.700000
7896	234518	BU1111	340	46.700000
5023	AX-532-FED-452-2Z7	BU1111	370	46.700000
5023	ZD-123-DFG-752-9G8	PS3333	750	46.700000
8042	13-J-9	BU7832	300	51.700000
8042	13-E-7	BU2075	150	51.700000
8042	13-E-7	BU1032	300	51.700000
8042	13-E-7	PC1035	400	51.700000
8042	91-A-7	PS7777	180	51.700000
8042	13-J-9	TC4203	250	51.700000
8042	13-E-7	TC4203	226	51.700000
8042	13-E-7	MC3021	400	51.700000
8042	91-V-7	BU1111	390	51.700000
5023	AB-872-DEF-732-2Z1	MC3021	5000	50.000000
5023	NF-123-ADS-642-9G3	PC8888	2000	50.000000
5023	NF-123-ADS-642-9G3	BU2075	2000	50.000000
5023	GH-542-NAD-713-9F9	PC1035	2000	50.000000
5023	ZA-000-ASD-324-4D1	PC1035	2000	50.000000
5023	ZA-000-ASD-324-4D1	PS7777	1500	50.000000
5023	ZD-123-DFG-752-9G8	BU2075	3000	50.000000
5023	ZD-123-DFG-752-9G8	TC7777	1500	50.000000
5023	ZS-645-CAT-415-1B2	BU2075	3000	50.000000
5023	ZS-645-CAT-415-1B2	BU2075	3000	50.000000
5023	XS-135-DER-432-8J2	PS3333	2687	50.000000
5023	XS-135-DER-432-8J2	TC7777	1090	50.000000
5023	XS-135-DER-432-8J2	PC1035	2138	50.000000
5023	ZZ-999-ZZZ-999-0A0	MC2222	2032	50.000000
5023	ZZ-999-ZZZ-999-0A0	BU1111	1001	50.000000
5023	ZA-000-ASD-324-4D1	BU1111	1100	50.000000
5023	NF-123-ADS-642-9G3	BU7832	1400	50.000000
5023	BS-345-DSE-860-1F2	TC4203	2700	50.000000
5023	GH-542-NAD-713-9F9	TC4203	2500	50.000000
5023	NF-123-ADS-642-9G3	TC4203	3500	50.000000
5023	BS-345-DSE-860-1F2	MC3021	4500	50.000000
5023	AX-532-FED-452-2Z7	MC3021	1600	50.000000
5023	NF-123-ADS-642-9G3	MC3021	2550	50.000000

stor_id	ord_num	title_id	qty	discount
5023	ZA-000-ASD-324-4D1	MC3021	3000	50.000000
5023	ZS-645-CAT-415-1B2	MC3021	3200	50.000000
5023	BS-345-DSE-860-1F2	BU2075	2200	50.000000
5023	GH-542-NAD-713-9F9	BU1032	1500	50.000000
5023	ZZ-999-ZZZ-999-0A0	PC8888	1005	50.000000
7896	124152	BU2075	42	50.500000
7896	124152	PC1035	25	50.500000
7131	Asoap132	BU2075	35	50.500000
7067	NB-1.142	PC1035	34	50.500000
7067	NB-1.142	TC4203	53	50.500000
8042	12-F-9	BU2075	30	55.500000
8042	12-F-9	BU1032	94	55.500000
7066	BA27618	BU2075	200	57.200000
7896	124152	TC4203	350	57.200000
7066	BA27618	TC4203	230	57.200000
7066	BA27618	MC3021	200	57.200000
7131	Asoap132	MC3021	137	57.200000
7067	NB-1.142	MC3021	270	57.200000
7067	NB-1.142	BU2075	230	57.200000
7131	Asoap132	BU1032	345	57.200000
7067	NB-1.142	BU1032	136	57.200000
8042	12-F-9	TC4203	300	62.200000
8042	12-F-9	MC3021	270	62.200000
8042	12-F-9	PC1035	133	62.200000
5023	AB-123-DEF-425-1Z3	TC4203	2500	60.500000
5023	AB-123-DEF-425-1Z3	BU2075	4000	60.500000
6380	342157	BU2075	200	57.200000
6380	342157	MC3021	250	57.200000
6380	356921	PS3333	200	46.700000
6380	356921	PS7777	500	46.700000
6380	356921	TC3218	125	46.700000
6380	234518	BU2075	135	46.700000
6380	234518	BU1032	320	46.700000
6380	234518	TC4203	300	46.700000
6380	234518	MC3021	400	46.700000

sales 表

sales 定义如下：

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

其主键为 stor_id 和 ord_num：

```
sp_primarykey sales, stor_id, ord_num
```

stor_id 列为 stores 的外键：

```
sp_foreignkey sales, stores, stor_id
```

表 A-6 列出 sales 的内容：

表 *A-6*：*sales* 表

stor_id	ord_num	date
5023	AB-123-DEF-425-1Z3	Oct 31 1995
5023	AB-872-DEF-732-2Z1	Nov 6 1995
5023	AX-532-FED-452-2Z7	Dec 1 1996
5023	BS-345-DSE-860-1F2	Dec 12 1996
5023	GH-542-NAD-713-9F9	Feb 15 1997
5023	NF-123-ADS-642-9G3	Mar 18 1997
5023	XS-135-DER-432-8J2	Mar 21 1997
5023	ZA-000-ASD-324-4D1	Jul 27 1994
5023	ZD-123-DFG-752-9G8	Mar 21 1997
5023	ZS-645-CAT-415-1B2	Mar 21 1997
5023	ZZ-999-ZZZ-999-0A0	Mar 21 1997
6380	342157	Dec 13 1994
6380	356921	Feb 17 1995
7066	BA27618	Oct 12 1996
7066	BA52498	Oct 27 1995
7066	BA71224	Aug 5 1996
7067	NB-1.142	Jan 2 1997
7067	NB-3.142	Jun 13 1995
7131	Asoap132	Nov 16 1996
7131	Asoap432	Dec 20 1995
7131	Fsoap867	Sep 8 1996
7896	124152	Aug 14 1996
7896	234518	Feb 14 1997

stor_id	ord_num	date
8042	12-F-9	Jul 13 1996
8042	13-E-7	May 23 1995
8042	13-J-9	Jan 13 1997
8042	55-V-7	Feb 20 1997
8042	91-A-7	Mar 20 1997
8042	91-V-7	Mar 20 1997

stores 表

stores 定义如下：

```
create table stores
(stor_id char(4) not null,
stor_name varchar(40) not null,
stor_address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null,
payterms varchar(12) null)
```

其主键为 stor_id：

```
sp_primarykey stores, stor_id
```

表 A-7 列出 stores 的内容：

表 *A-7*：*stores* 表

stor_id	stor_name	stor_address	city	state	country	postal-code	pay-terms
7066	Barnum's	567 Pasadena Ave.	Tustin	CA	USA	92789	Net 30
7067	News & Brews	577 First St.	Los Gatos	CA	USA	96745	Net 30
7131	Doc-U-Mat: Quality Laundry and Books	24-A Avrogado Way	Remulade	WA	USA	98014	Net 60
8042	Bookbeat	679 Carson St.	Portland	OR	USA	89076	Net 30
6380	Eric the Read Books	788 Catamaugus Ave.	Seattle	WA	USA	98056	Net 60
7896	Fricative Bookshop	89 Madison St.	Fremont	CA	USA	90019	Net 60
5023	Thoreau Reading Discount Chain	20435 Walden Expressway	Concord	MA	USA	01776	Net 60

roysched 表

roysched 定义如下：

```
create table roysched
title_id tid not null,
lorange int null,
hirange int null,
royalty int null)
```

其主键为 title_id：

```
sp_primarykey roysched, title_id
```

title_id 列为 titles 的外键：

```
sp_foreignkey roysched, titles, title_id
```

title_id 列的非聚簇索引定义如下：

```
create nonclustered index titleidind
on roysched (title_id)
```

表 A-8 列出 roysched 的内容：

表 *A-8*：*roysched* 表

title_id	lorange	hirange	royalty
BU1032	0	5000	10
BU1032	5001	50000	12
PC1035	0	2000	10
PC1035	2001	3000	12
PC1035	3001	4000	14
PC1035	4001	10000	16
PC1035	10001	50000	18
BU2075	0	1000	10
BU2075	1001	3000	12
BU2075	3001	5000	14
BU2075	5001	7000	16
BU2075	7001	10000	18
BU2075	10001	12000	20
BU2075	12001	14000	22
BU2075	14001	50000	24
PS2091	0	1000	10
PS2091	1001	5000	12
PS2091	5001	10000	14
PS2091	10001	50000	16

title_id	lorange	hirange	royalty
PS2106	0	2000	10
PS2106	2001	5000	12
PS2106	5001	10000	14
PS2106	10001	50000	16
MC3021	0	1000	10
MC3021	1001	2000	12
MC3021	2001	4000	14
MC3021	4001	6000	16
MC3021	6001	8000	18
MC3021	8001	10000	20
MC3021	10001	12000	22
MC3021	12001	50000	24
TC3218	0	2000	10
TC3218	2001	4000	12
TC3218	4001	6000	14
TC3218	6001	8000	16
TC3218	8001	10000	18
TC3218	10001	12000	20
TC3218	12001	14000	22
TC3218	14001	50000	24
PC8888	0	5000	10
PC8888	5001	10000	12
PC8888	10001	15000	14
PC8888	15001	50000	16
PS7777	0	5000	10
PS7777	5001	50000	12
PS3333	0	5000	10
PS3333	5001	10000	12
PS3333	10001	15000	14
PS3333	15001	50000	16
BU1111	0	4000	10
BU1111	4001	8000	12
BU1111	8001	10000	14
BU1111	12001	16000	16
BU1111	16001	20000	18
BU1111	20001	24000	20
BU1111	24001	28000	22

title_id	lorange	hirange	royalty
BU1111	28001	50000	24
MC2222	0	2000	10
MC2222	2001	4000	12
MC2222	4001	8000	14
MC2222	8001	12000	16
MC2222	8001	12000	16
MC2222	12001	20000	18
MC2222	20001	50000	20
TC7777	0	5000	10
TC7777	5001	15000	12
TC7777	15001	50000	14
TC4203	0	2000	10
TC4203	2001	8000	12
TC4203	8001	16000	14
TC4203	16001	24000	16
TC4203	24001	32000	18
TC4203	32001	40000	20
TC4203	40001	50000	22
BU7832	0	5000	10
BU7832	5001	10000	12
BU7832	10001	15000	14
BU7832	15001	20000	16
BU7832	20001	25000	18
BU7832	25001	30000	20
BU7832	30001	35000	22
BU7832	35001	50000	24
PS1372	0	10000	10
PS1372	10001	20000	12
PS1372	20001	30000	14
PS1372	30001	40000	16
PS1372	40001	50000	18

discounts 表

discounts 定义如下：

```
create table discounts
(discounttype varchar(40) not null,
stor_id char(4) null,
lowqty smallint null,
highqty smallint null,
discount float not null)
```

其主键为 discounttype 和 stor_id：

```
sp_primarykey discounts, discounttype, stor_id
```

stor_id 列为 stores 的外键：

```
sp_foreignkey discounts, stores, stor_id
```

表 A-9 列出 discounts 的内容：

表 *A-9*：*discounts* 表

discounttype	stor_id	lowqty	highqty	discount
Initial Customer	8042	NULL	NULL	10.5
Volume Discount	NULL	100	1001	6.7
Huge Volume Discount	NULL	1001	NULL	10
Customer Discount	8042	NULL	NULL	5

blurbs 表

blurbs 定义如下：

```
create table blurbs
(au_id id not null,
copy text null)
```

其主键为 au_id：

```
sp_primarykey blurbs, au_id
```

au_id 列为 authors 的外键：

```
sp_foreignkey blurbs, authors, au_id
```

表 A-10 列出 blurbs 的内容：

表 *A-10*：*blurbs* 表

au_id	copy
486-29-1786	If Chastity Locksley didn't exist, this troubled world would have created her! Not only did she master the mystic secrets of inner strength to conquer adversity when she encountered it in life, but also, after "reinventing herself," as she says, by writing "Emotional Security: A New Algorithm" following the devastating loss of her cat, Old Algorithm, she founded Publish or Perish, the page-by-page, day-by-day, write-yourself-to-wellness encounter workshops franchise empire, the better to share her inspiring discoveries with us all. Her "Net Etiquette," a brilliant social treatise in its own right and a fabulous pun, is the only civilized alternative to the gross etiquette often practiced on the public networks.
648-92-1872	A chef's chef and a raconteur's raconteur, Reginald Blotchet-Halls calls London his second home. "Th' palace kitchen's me first 'ome, act'lly!" Blotchet-Halls' astounding ability to delight our palates with palace delights is matched only by his equal skill in satisfying our perpetual hunger for delicious back-stairs gossip by serving up tidbits and entrees literally fit for a king!
998-72-3567	Albert Ringer was born in a trunk to circus parents, but another kind of circus trunk played a more important role in his life years later. He grew up as an itinerant wrestler and roustabout in the reknowned Ringer Brothers and Betty and Bernie's Circus. Once known in the literary world only as Anne Ringer's wrestling brother, he became a writer while recuperating from a near-fatal injury received during a charity benefit bout with a gorilla. "Slingshotting" himself from the ring ropes, Albert flew over the gorilla's head and would have landed head first on the concrete. He was saved from certain death by Nana, an elephant he had befriended as a child, who caught him in her trunk. Nana held him so tightly that three ribs cracked and he turned blue from lack of oxygen. "I was delirious. I had an out-of-body experience! My whole life passed before me eyes. I promised myself 'If I get through this, I'll use my remaining time to share what I learned out there.' I owe it all to Nana!"
899-46-2035	Anne Ringer ran away from the circus as a child. A university creative writing professor and her family took Anne in and raised her as one of their own. In this warm and television-less setting she learned to appreciate the great classics of literature. The stream of aspiring and accomplished writers that flowed constantly through the house confirmed her repudiation of the circus family she'd been born into: "Barbarians!" The steadily growing recognition of her literary work was, to her, vindication. When her brother's brush with death brought them together after many years, she took advantage of life's crazy chance thing and broke the wall of anger that she had constructed to separate them. Together they wrote, "Is Anger the Enemy?" an even greater blockbuster than her other collaborative work, with Michel DeFrance, "The Gourmet Microwave."

au_id	copy
672-71-3249	They asked me to write about myself and my book, so here goes: I started a restaurant called "de Gustibus" with two of my friends. We named it that because you really can't discuss taste. We're very popular with young business types because we're young business types ourselves. Whenever we tried to go out to eat in a group we always got into these long tiresome negotiations: "I just ate Italian," or "I ate Greek yesterday," or "I NEVER eat anything that's not organic!" Inefficient. Not what business needs today. So, it came to us that we needed a restaurant we could all go to every day and not eat the same thing twice in a row maybe for a year! We thought, "Hey, why make people choose one kind of restaurant over another, when what they really want is a different kind of food?" At de Gustibus you can eat Italian, Chinese, Japanese, Greek, Russian, Tasmanian, Iranian, and on and on all at the same time. You never have to choose. You can even mix and match! We just pooled our recipes, opened the doors, and never looked back. We're a big hit, what can I say? My recipes in "Sushi, Anyone?" are used at de Gustibus. They satisfy crowds for us every day. They will work for you, too. Period!
409-56-7008	Bennet was the classic too-busy executive. After discovering computer databases, he now has the time to run several successful businesses and sit on three major corporate boards. Bennet also donates time to community service organizations. Miraculously, he also finds time to write and market executive-oriented, in-depth computer hardware and software reviews. "I'm hyperkinetic, so being dynamic and fast-moving is a piece of cake. But being organized isn't easy for me or for anyone I know. There's just one word for that: 'databases!' Databases can cure you or kill you. If you get the right one, you can be like me. If you get the wrong one, watch out. Read my book!"

au_pix 表

au_pix 定义如下：

```
create table au_pix
(au_id char(11) not null,
pic image null,
format_type char(11) null,
bytesize int null,
pixwidth_hor char(14) null,
pixwidth_vert char(14) null)
```

其主键为 au_id：

```
sp_primarykey au_pix, au_id
```

au_id 列为 authors 的外键：

```
sp_foreignkey au_pix, authors, au_id
```

表 A-11 列出 au_pix 的内容：

表 *A-11*：*au_pix* 表

au_id	pic	format_type	bytesize	pixwidth_hor	pixwidth_vert
409-56-7008	0x0000...	PICT	30220	626	635
486-29-1786	0x59a6...	Sunraster	27931	647	640
648-92-1872	0x59a6...	Sunraster	36974	647	640
672-71-3249	0x000a...	PICT	13487	654	639
899-46-2035	0x4949...	TIF	52023	648	641
998-72-3567	0x4949...	TIF	52336	653	637

pic 列包含二进制数据，本表中并未显示它们的完整信息。 下页显示了由这一数据表示的图片。 由于 image 数据（六幅图，PICT、TIF 和 Sunraster 文件格式各两幅）相当大，因此*只有*在要使用或检测 image 数据类型时，再运行 installpix2 脚本。 提供 image 数据旨在展示 Sybase 如何存储 image 数据。 Sybase 不提供显示 image 数据的工具： 从数据库抽取此类数据后，必须使用相应的屏幕图形工具来显示图像。

图 *A-1*：来自 *au_pix* 表的作者肖像

Akiko Yokomoto　672-71-3249

Chastity Locksley　486-29-1786

Anne Ringer　899-46-2035

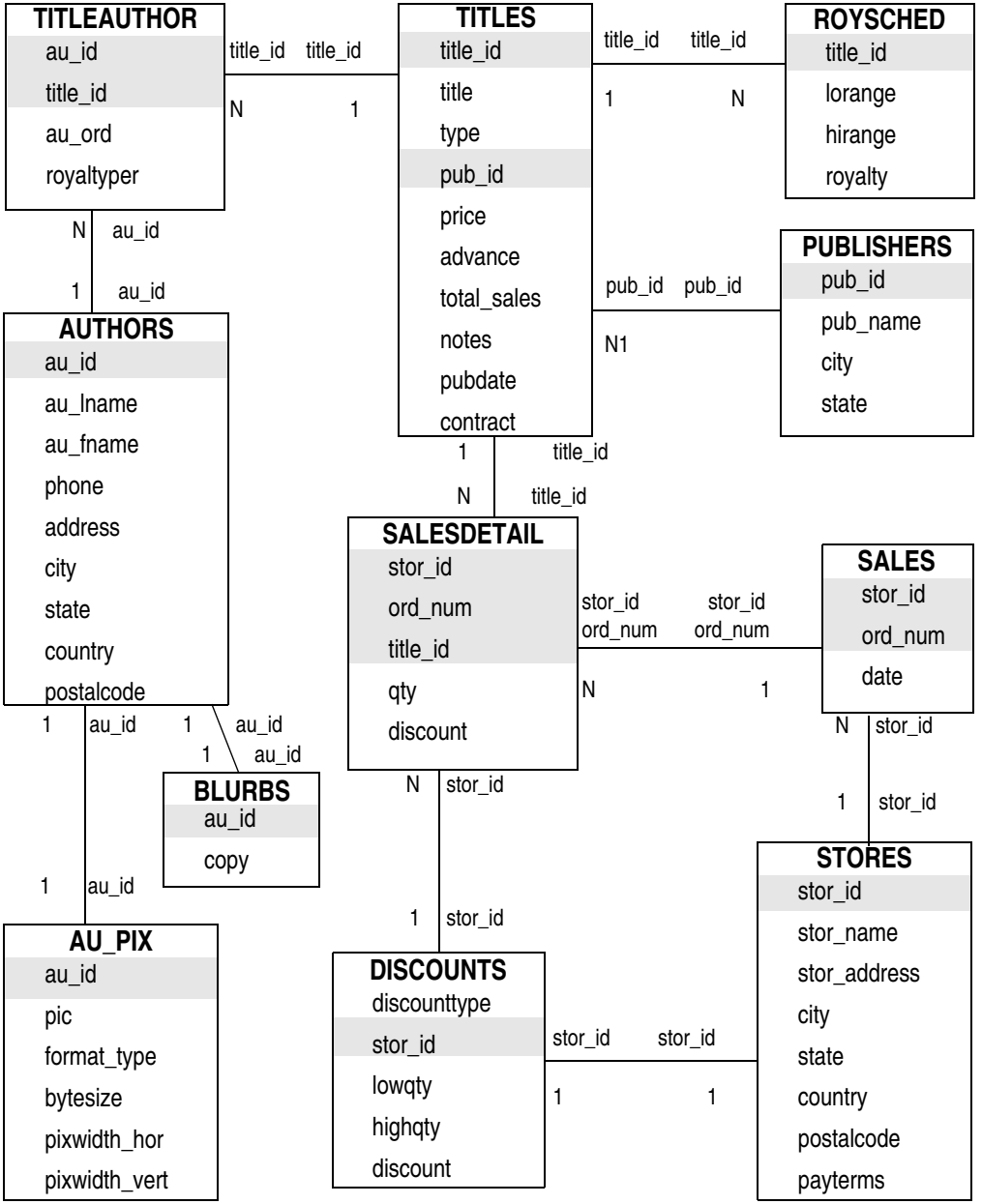
Albert Ringer　998-72-3567

Abraham Bennet　409-56-7008

Reginald Blotchet-Halls　648-92-1872

pubs2 数据库框图

图 *A-2*：*pubs2* 数据库框图



附 录 B

pubs3 数据库

本章描述样本数据库 pubs3。 该数据库包含表 publishers、authors、titles、titleauthor、salesdetail、sales、stores、store_employees、discounts、roysched 和 blurbs。

pubs3 数据库也包含主键与外键、规则、缺省值、视图、触发器及存储过程。

第 664 页的图 B-1 显示了 pubs3 数据库的框图。

有关安装 pubs3 的信息，参见所用平台的配置指南。

pubs3 数据库中的表

以下几节介绍 pubs3 的每个表。 这些表中，每个列标题指定了列名、数据类型（包括任何用户定义的数据类型）、空或非空状态以及如何使用参照完整性。 列标题还指定了影响该列的所有缺省值、规则、触发器和索引。

publishers 表

publishers 定义如下：

```
create table publishers
(pub_id char(4) not null,
pub_name varchar(40) not null,
city varchar(20) null,
state char(2) null,
unique nonclustered (pub_id))
```

其 pub_idrule 规则定义如下：

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

表 B-1 列出 publishers 的内容：

表 *B-1*：*publishers* 表

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

authors 表

authors 定义如下：

```
create table authors
(au_id id not null,
au_lname varchar(40) not null,
au_fname varchar(20) not null,
phone char(12) not null,
address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null,
unique nonclustered (au_id))
```

au_lname 和 au_fname 列的非聚簇索引定义如下：

```
create nonclustered index aunmind
on authors (au_lname, au_fname)
```

phone 列有如下缺省值：

```
create default phonedflt as "UNKNOWN"
sp_bindefault phonedft, "authors.phone"
```

如下视图使用 authors：

```
create view titleview
as
select title, au_ord, au_lname,
price, num_sold, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

表 B-2 列出 authors 的内容：

表 *B-2*：*authors* 表

au_id	au_lname	au_fname	phone	address	city	state	country	postalcode
172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA	USA	94025
213-46-8915	Green	Marjorie	510 986-7020	309 63rd St. #411	Oakland	CA	USA	94618
238-95-7766	Carson	Cheryl	510 548-7723	589 Darwin Ln.	Berkeley	CA	USA	94705
267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose	CA	USA	95128
274-80-9391	Straight	Dick	510 834-2919	5420 College Av.	Oakland	CA	USA	94609
341-22-1782	Smith	Meander	913 843-0462	10 Mississippi Dr.	Lawrence	KS	USA	66044
409-56-7008	Bennet	Abraham	510 658-9932	6223 Bateman St.	Berkeley	CA	USA	94705
427-17-2319	Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA	USA	94301
472-27-2349	Gringlesby	Burt	707 938-6445	PO Box 792	Covelo	CA	USA	95428
486-29-1786	Locksley	Chastity	415 585-4620	18 Broadway Av.	San Francisco	CA	USA	94130
527-72-3246	Greene	Morningstar	615 297-2723	22 Graybar House Rd.	Nashville	TN	USA	37215
648-92-1872	Blotchet-Halls	Reginald	503 745-6402	55 Hillsdale Bl.	Corvallis	OR	USA	97330
672-71-3249	Yokomoto	Akiko	415 935-4228	3 Silver Ct.	Walnut Creek	CA	USA	94595

au_id	au_lname	au_fname	phone	address	city	state	country	postalcode
712-45-1867	del Castillo	Innes	615 996-8275	2286 Cram Pl. #86	Ann Arbor	MI	USA	48105
722-51-5454	DeFrance	Michel	219 547-9982	3 Balding Pl.	Gary	IN	USA	46403
724-08-9931	Stringer	Dirk	510 843-2991	5420 Telegraph Av.	Oakland	CA	USA	94609
724-80-9391	MacFeather	Stearns	510 354-7128	44 Upland Hts.	Oakland	CA	USA	94612
756-30-7391	Karsen	Livia	510 534-9219	5720 McAuley St.	Oakland	CA	USA	94609
807-91-6654	Panteley	Sylvia	301 946-8853	1956 Arlington Pl.	Rockville	MD	USA	20853
846-92-7186	Hunter	Sheryl	415 836-7128	3410 Blonde St.	Palo Alto	CA	USA	94301
893-72-1158	McBadden	Heather	707 448-4982	301 Putnam	Vacaville	CA	USA	95688
899-46-2035	Ringer	Anne	801 826-0752	67 Seventh Av.	Salt Lake City	UT	USA	84152
998-72-3567	Ringer	Albert	801 826-0752	67 Seventh Av.	Salt Lake City	UT	USA	84152

titles 表

titles 定义如下：

```
create table titles
(title_id tid not null,
title varchar(80) not null,
type char(12) not null,
pub_id char(4) null
    references publishers(pub_id),
price money null,
```

```
advance numeric(12,2) null,
num_sold int null,
notes varchar(200) null,
pubdate datetime not null,
contract bit not null,
unique nonclustered (title_id))
```

title 列的非聚簇索引定义如下：

```
create nonclustered index titleind
on titles (title)
```

其 title_idrule 定义如下：

```
create rule title_idrule
as
@title_id like "BU[0-9][0-9][0-9][0-9]" or
@title_id like "[MT]C[0-9][0-9][0-9][0-9]" or
@title_id like "P[SC][0-9][0-9][0-9][0-9]" or
@title_id like "[A-Z][A-Z]xxxx" or
@title_id like "[A-Z][A-Z]yyyy"
```

type 列有如下缺省值：

```
create default typedflt as "UNDECIDED"
sp_bindefault typedflt, "titles.type"
```

pubdate 列有如下缺省值：

```
create default datedflt as getdate()
sp_bindefault datedflt, "titles.pubdate"
```

titles 使用如下触发器：

```
create trigger deltitle
on titles
for delete
as
if (select count(*) from deleted, salesdetail
where salesdetail.title_id = deleted.title_id) >0
begin
   rollback transaction
   print "You can’t delete a title with sales."
end
```

如下视图使用 titles：

```
create view titleview
as
select title, au_ord, au_lname,
price, num_sold, pub_id
```

```
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

表 B-3 列出 titles 的内容：

表 *B-3*：*titles* 表

title_id	title	type	pub-_id	price	ad-vance	num_-sold	notes	pub-date	con-tract
BU1032	The Busy Executive's Database Guide	business	1389	19.99	5000.00	4095	An overview of available database systems with emphasis on common business applica-tions. Illustrated.	Jun 12, 1986	1
BU1111	Cooking with Computers: Surrepti-tious Balance Sheets	business	1389	11.95	5000.00	3876	Helpful hints on how to use your electronic resources to the best advantage.	Jun 9, 1988	1
BU2075	You Can Combat Computer Stress!	business	0736	2.99	10125.00	18722	The latest medical and psycho-logical techniques for living with the electronic office. Easy-to-under-stand explana-tions.	Jun 30, 1985	1

title_id	title	type	pub-_id	price	ad-vance	num_-sold	notes	pub-date	con-tract
BU7832	Straight Talk About Computers	business	1389	19.99	5000.00	4095	Annotated analysis of what computers can do for you: a no-hype guide for the critical user.	Jun 22, 1987	1
MC2222	Silicon Valley Gastro-nomic Treats	mod_-cook	0877	19.99	0.00	2032	Favorite recipes for quick, easy, and elegant meals; tried and tested by people who never have time to eat, let alone cook.	Jun 9, 1989	1
MC3021	The Gourmet Microwave	mod_-cook	0877	2.99	15000.00	22246	Traditional French gourmet recipes adapted for modern microwave cooking.	Jun 18, 1985	1
PC1035	But Is It User Friendly?	popular_comp	1389	22.95	7000.00	8780	A survey of software for the naive user, focusing on the "friendli-ness" of each.	Jun 30, 1986	1
MC3026	The Psychology of Computer Cooking	UNDE-CIDED	0877	NULL	NULL	NULL	NULL	Jul 24, 1991	0

title_id	title	type	pub-_id	price	ad-vance	num_-sold	notes	pub-date	con-tract
PC8888	Secrets of Silicon Valley	popular_comp	1389	20.00	8000.00	4095	Muck-raking reporting by two coura-geous women on the world's largest computer hardware and software manufact-urers.	Jun 12, 1987	1
PC9999	Net Etiquette	popular_comp	1389	NULL	NULL	NULL	A must-read for computer conferenc-ing debu-tantes!	Jul 24, 1996	0
PS1372	Computer Phobic and Non-Phobic Indivi-duals: Behavior Variations	psycho-logy	0877	21.59	7000.00	375	A must for the specialist, this book examines the difference between those who hate and fear computersa nd those who think they are swell.	Oct 21, 1990	1

title_id	title	type	pub-_id	price	ad-vance	num_-sold	notes	pub-date	con-tract
PS2091	Is Anger the Enemy?	psycho-logy	0736	10.95	2275.00	2045	Carefully researched study of the effects of strong emotions on the body. Metabolic charts included.	Jun 15, 1989	1
PS2106	Life Without Fear	psycho-logy	0736	7.00	6000.00	111	New exercise, meditation, and nutritional techniques that can reduce the shock of daily inter-actions. Popular audience. Sample menus included. Exercise video available separately.	Oct 5, 1990	1
PS3333	Prolonged Data Deprivation: Four Case Studies	psycho-logy	0736	19.99	2000.00	4072	What happens when the data runs dry? Searching evaluations of infor-mation-shortage effects on heavy users.	Jun 12, 1988	1

title_id	title	type	pub-_id	price	ad-vance	num_-sold	notes	pub-date	con-tract
PS7777	Emotional Security: A New Algorithm	psycho-logy	0736	7.99	4000.00	3336	Protecting yourself and your loved ones from undue emotional stress in the modern world. Use of computer and nutritional aids empha-sized.	Jun 12, 1988	1
TC3218	Onions, Leeks, and Garlic: Cooking Secrets of the Mediterra-nean	trad_coo k	0877	20.95	7000.00	375	Profusely illustrated in color, this makes a wonderful gift book for a cuisine-oriented friend.	Oct 21, 1990	1
TC4203	Fifty Years in Bucking-ham Palace Kitchens	trad_coo k	0877	11.95	4000.00	15096	More anecdotes from the Queen's favorite cook describing life among English royalty. Recipes, techniques, tender vignettes.	Jun 12, 1985	1

title_id	title	type	pub-_id	price	ad-vance	num_-sold	notes	pub-date	con-tract
TC7777	Sushi, Anyone?	trad_cook	0877	14.99	8000.00	4095	Detailed instructions on improving your position in life by learning how to make authentic Japanese sushi in your spare time. 5-10% increase in number of friends per recipe reported from beta test.	Jun 12, 1987	1

titleauthor 表

titleauthor 定义如下：

```
create table titleauthor
(au_id id not null
    references authors(au_id),
title_id tid not null
    references titles(title_id),
au_ord tinyint null,
royaltyper int null)
```

au_id 列的非聚簇索引定义如下：

```
create nonclustered index auidind
on titleauthor(au_id)
```

title_id 列的非聚簇索引定义如下：

```
create nonclustered index titleidind
on titleauthor(title_id)
```

如下视图使用 titleauthor：

```
create view titleview
as
select title, au_ord, au_lname,
price, num_sold, pub_id
from authors, titles, titleauthor
where authors.au_id = titleauthor.au_id
and titles.title_id = titleauthor.title_id
```

如下过程使用 titleauthor：

```
create procedure byroyalty @percentage int
as
select au_id from titleauthor
where titleauthor.royaltyper = @percentage
```

表 B-4 列出 titleauthor 的内容：

表 *B-4*：*titleauthor* 表

au_id	title_id	au_ord	royaltyper
172-32-1176	PS3333	1	100
213-46-8915	BU1032	2	40
213-46-8915	BU2075	1	100
238-95-7766	PC1035	1	100
267-41-2394	BU1111	2	40
267-41-2394	TC7777	2	30
274-80-9391	BU7832	1	100
409-56-7008	BU1032	1	60
427-17-2319	PC8888	1	50
472-27-2349	TC7777	3	30
486-29-1786	PC9999	1	100
486-29-1786	PS7777	1	100
648-92-1872	TC4203	1	100
672-71-3249	TC7777	1	40
712-45-1867	MC2222	1	100
722-51-5454	MC3021	1	75
724-80-9391	BU1111	1	60
724-80-9391	PS1372	2	25
756-30-7391	PS1372	1	75
807-91-6654	TC3218	1	100
846-92-7186	PC8888	2	50
899-46-2035	MC3021	2	25

au_id	title_id	au_ord	royaltyper
899-46-2035	PS2091	2	50
998-72-3567	PS2091	1	50
998-72-3567	PS2106	1	100

salesdetail 表

salesdetail 定义如下：

```
create table salesdetail
(stor_id char(4) not null
    references sales(stor_id),
ord_num numeric(6,0)
    references sales(ord_num),
title_id tid not null
    references titles(title_id),
qty smallint not null,
discount float not null)
```

title_id 列的非聚簇索引定义如下：

```
create nonclustered index titleidind
on salesdetail (title_id)
```

stor_id 列的非聚簇索引定义如下：

```
create nonclustered index salesdetailind
on salesdetail (stor_id)
```

其 title_idrule 规则定义如下：

```
create rule title_idrule
as
@title_id like "BU[0-9][0-9][0-9][0-9]" or
@title_id like "[MT]C[0-9][0-9][0-9][0-9]" or
@title_id like "P[SC][0-9][0-9][0-9][0-9]" or
@title_id like "[A-Z][A-Z]xxxx" or
@title_id like "[A-Z][A-Z]yyyy"
```

salesdetail 使用如下触发器：

```
create trigger totalsales_trig on salesdetail
    for insert, update, delete
as
/* Save processing:  return if there are no rows
affected */
if @@rowcount = 0
```

```
    begin
       return
end
/* add all the new values */
/* use isnull:  a null value in the titles table
means
**              "no sales yet" not "sales unknown"
*/
update titles
    set num_sold = isnull(num_sold, 0) + (select
sum(qty)
    from inserted
    where titles.title_id = inserted.title_id)
   where title_id in (select title_id from inserted)
/* remove all values being deleted or updated */
update titles
    set num_sold = isnull(num_sold, 0) - (select
sum(qty)
    from deleted
    where titles.title_id = deleted.title_id)
   where title_id in (select title_id from deleted)
```

表 B-5 列出 salesdetail 的内容：

表 *B-5*：*salesdetail* 表

stor_id	ord_num	title_id	qty	discount
7896	100014	TC3218	75	40.000000
7896	100014	TC7777	75	40.000000
7131	100017	TC3218	50	40.000000
7131	100017	TC7777	80	40.000000
5023	100020	TC3218	85	40.000000
8042	100016	PS3333	90	45.000000
8042	100016	TC3218	40	45.000000
8042	100016	PS2106	30	45.000000
8042	100023	PS2106	50	45.000000
8042	100015	PS2106	31	45.000000
8042	100016	MC3021	69	45.000000
5023	100009	PC1035	1000	46.700000
5023	100007	BU2075	500	46.700000
5023	100007	BU1032	200	46.700000
5023	100007	BU7832	150	46.700000
5023	100007	PS7777	125	46.700000
5023	100018	TC7777	1000	46.700000

stor_id	ord_num	title_id	qty	discount
5023	100018	BU1032	1000	46.700000
5023	100018	PC1035	750	46.700000
7131	100004	BU1032	200	46.700000
7066	100012	BU7832	100	46.700000
7066	100021	PS7777	200	46.700000
7066	100021	PC1035	300	46.700000
7066	100021	TC7777	350	46.700000
5023	100025	PS2091	1000	46.700000
7067	100019	PS2091	200	46.700000
7067	100019	PS7777	250	46.700000
7067	100019	PS3333	345	46.700000
7067	100019	BU7832	360	46.700000
5023	100020	PS2091	845	46.700000
5023	100020	PS7777	581	46.700000
5023	100027	PS1372	375	46.700000
7067	100019	BU1111	175	46.700000
5023	100020	BU7832	885	46.700000
5023	100025	BU7832	900	46.700000
5023	100007	TC4203	550	46.700000
7131	100004	TC4203	350	46.700000
7896	100014	TC4203	275	46.700000
7066	100021	TC4203	500	46.700000
7067	100019	TC4203	512	46.700000
7131	100004	MC3021	400	46.700000
5023	100007	PC8888	105	46.700000
5023	100018	PC8888	300	46.700000
7066	100021	PC8888	350	46.700000
7067	100019	PC8888	335	46.700000
7131	100017	BU1111	500	46.700000
7896	100014	BU1111	340	46.700000
5023	100007	BU1111	370	46.700000
5023	100025	PS3333	750	46.700000
8042	100014	BU7832	300	51.700000
8042	100013	BU2075	150	51.700000
8042	100013	BU1032	300	51.700000
8042	100013	PC1035	400	51.700000
8042	100016	PS7777	180	51.700000

stor_id	ord_num	title_id	qty	discount
8042	100014	TC4203	250	51.700000
8042	100013	TC4203	226	51.700000
8042	100013	MC3021	400	51.700000
8042	100023	BU1111	390	51.700000
5023	100003	MC3021	5000	50.000000
5023	100018	PC8888	2000	50.000000
5023	100018	BU2075	2000	50.000000
5023	100010	PC1035	2000	50.000000
5023	100022	PC1035	2000	50.000000
5023	100022	PS7777	1500	50.000000
5023	100025	BU2075	3000	50.000000
5023	100025	TC7777	1500	50.000000
5023	100026	BU2075	3000	50.000000
5023	100026	BU2075	3000	50.000000
5023	100020	PS3333	2687	50.000000
5023	100020	TC7777	1090	50.000000
5023	100020	PC1035	2138	50.000000
5023	100027	MC2222	2032	50.000000
5023	100027	BU1111	1001	50.000000
5023	100022	BU1111	1100	50.000000
5023	100018	BU7832	1400	50.000000
5023	100009	TC4203	2700	50.000000
5023	100010	TC4203	2500	50.000000
5023	100018	TC4203	3500	50.000000
5023	100009	MC3021	4500	50.000000
5023	100007	MC3021	1600	50.000000
5023	100018	MC3021	2550	50.000000
5023	100022	MC3021	3000	50.000000
5023	100026	MC3021	3200	50.000000
5023	100009	BU2075	2200	50.000000
5023	100010	BU1032	1500	50.000000
5023	100027	PC8888	1005	50.000000
7896	100013	BU2075	42	50.500000
7896	100013	PC1035	25	50.500000
7131	100005	BU2075	35	50.500000
7067	100011	PC1035	34	50.500000
7067	100011	TC4203	53	50.500000

stor_id	ord_num	title_id	qty	discount
8042	100015	BU2075	30	55.500000
8042	100015	BU1032	94	55.500000
7066	100001	BU2075	200	57.200000
7896	100013	TC4203	350	57.200000
7066	100001	TC4203	230	57.200000
7066	100001	MC3021	200	57.200000
7131	100005	MC3021	137	57.200000
7067	100011	MC3021	270	57.200000
7067	100011	BU2075	230	57.200000
7131	100005	BU1032	345	57.200000
7067	100011	BU1032	136	57.200000
8042	100015	TC4203	300	62.200000
8042	100015	MC3021	270	62.200000
8042	100015	PC1035	133	62.200000
5023	100002	TC4203	2500	60.500000
5023	100002	BU2075	4000	60.500000
6380	100028	BU2075	200	57.200000
6380	100028	MC3021	250	57.200000
6380	100029	PS3333	200	46.700000
6380	100029	PS7777	500	46.700000
6380	100029	TC3218	125	46.700000
6380	100014	BU2075	135	46.700000
6380	100014	BU1032	320	46.700000
6380	100014	TC4203	300	46.700000
6380	100014	MC3021	400	46.700000

sales 表

sales 定义如下：

```
create table sales
(stor_id char(4) not null
    references stores(stor_id),
ord_num numeric(6,0) identity,
date datetime not null,
unique nonclustered (ord_num))
```

表 B-6 列出 sales 的内容：

表 *B-6*：*sales* 表

stor_id	ord_num	date
5023	100002	Oct 31 1995
5023	100003	Nov 6 1995
5023	100007	Dec 1 1996
5023	100009	Dec 12 1996
5023	100010	Feb 15 1997
5023	100018	Mar 18 1997
5023	100020	Mar 21 1997
5023	100022	Jul 27 1994
5023	100025	Mar 21 1997
5023	100026	Mar 21 1997
5023	100027	Mar 21 1997
6380	100028	Dec 13 1994
6380	100029	Feb 17 1995
7066	100001	Oct 12 1996
7066	100012	Oct 27 1995
7066	100021	Aug 5 1996
7067	100008	Jan 2 1997
7067	100024	Jun 13 1995
7131	100006	Nov 16 1996
7131	100011	Dec 20 1995
7131	100019	Sep 8 1996
7896	100005	Aug 14 1996
7896	100017	Feb 14 1997
8042	100004	Jul 13 1996
8042	100013	May 23 1995
8042	100014	Jan 13 1997
8042	100015	Feb 20 1997
8042	100016	Mar 20 1997
8042	100023	Mar 20 1997

stores 表

stores 定义如下：

```
create table stores
(stor_id char(4) not null,
stor_name varchar(40) not null,
stor_address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode char(10) null,
payterms varchar(12) null,
unique nonclustered (stor_id))
```

表 B-7 列出 stores 的内容：

表 B-7：stores 表

stor_id	stor_name	stor_address	city	state	country	postalcode	payterms
7066	Barnum's	567 Pasadena Ave.	Tustin	CA	USA	92789	Net 30
7067	News & Brews	577 First St.	Los Gatos	CA	USA	96745	Net 30
7131	Doc-U-Mat: Quality Laundry and Books	24-A Avrogado Way	Remulade	WA	USA	98014	Net 60
8042	Bookbeat	679 Carson St.	Portland	OR	USA	89076	Net 30
6380	Eric the Read Books	788 Catamaugus Ave.	Seattle	WA	USA	98056	Net 60
7896	Fricative Bookshop	89 Madison St.	Fremont	CA	USA	90019	Net 60
5023	Thoreau Reading Discount Chain	20435 Walden Expressway	Concord	MA	USA	01776	Net 60

store_employees 表

store_employees 定义如下：

```
create table store_employees
(stor_id char(4) null
    references stores(stor_id),
emp_id id not null,
mgr_id id null
    references store_employees(emp_id),
```

```
emp_lname varchar(40) not null,
emp_fname varchar(20) not null,
phone char(12) null,
address varchar(40) null,
city varchar(20) null,
state char(2) null,
country varchar(12) null,
postalcode varchar(10) null,
unique nonclustered (emp_id))
```

表 B-8 列出 store_employees 的内容：

表 *B-8*：*store_employees* 表

stor-_id	emp-_id	mgr-_id	emp_-lname	emp_-fname	phone	address	city	state	coun-try	postal-code
7066	415-92-7301	415-92-7301	Chichikov	P.	619 336-8978	20 Alfalfa Ave.	Twenty-nine Palms	CA	USA	92277
7066	202-93-0009	415-92-7301	Spindle-shanks	Daisy	619 336-8900	14 Wild Cat Way	Twenty-nine Palms	CA	USA	92277
7066	444-89-2342	415-92-7301	Ratsbane	Ian	619 344-92321	1827 Road-runner Dr.	Twenty-nine Palms	CA	USA	92278
7066	332-78-2382	415-92-7301	Bayless	Chuck	619 777-4356	7 Maynard Dr.	Twenty-nine Palms	CA	USA	92277
7066	898-00-2383	415-92-7301	Pumpion	Tammy	619 222-3434	10089 Lazy Joe Ave.	Twenty-nine Palms	CA	USA	92277
7066	222-90-3737	415-92-7301	Barnum	Arthur	619 333-3232	522 Lupine Ave.	Twenty-nine Palms	CA	USA	92277
7066	939-32-1212	415-92-7301	Ronald	Bill	619 333-4550	69 Coyote Ct.	Twenty-nine Palms	CA	USA	92277
7067	343-38-9494	343-38-9494	St. Augustine	Fiona	408 222-8383	17 Call of the Wild Rd.	Los Gatos	CA	USA	95030
7067	222-90-3483	343-38-9494	Sebastian	Basil	408 334-8928	29 Dharma Rd.	Los Gatos	CA	USA	95030
7067	121-00-3823	343-38-9494	March-main	Albin	408 343-8934	1001 Top of the Hill Ct.	Los Gatos	CA	USA	95030

stor-_id	emp-_id	mgr-_id	emp_-lname	emp_-fname	phone	address	city	state	coun-try	postal-code
7067	232-29-8938	343-38-9494	Barlow	Hilary	408 232-3332	2 Topping Way	Los Gatos	CA	USA	95032
7067	232-89-9393	343-38-9494	Dieuberi	Gustav	408 675-8938	30 Mount Madonna Rd.	Los Gatos	CA	USA	95030
7131	565-67-3920	565-67-3920	Lavator	Marie	206 328-2838	14 Peter Grubb Rd. SE	Renton	WA	USA	98058
7131	848-34-4838	565-67-3920	Sope	Inez	206 438-3434	1238 Index Ave. SE	Renton	WA	USA	98058
7131	232-38-2232	565-67-3920	Tubb	Wally	206 323-2828	180 Moses Lane S	Renton	WA	USA	98058
7131	111-90-3283	565-67-3920	Basin	Chalker	206 323-7777	192 Naches Ave. N	Renton	WA	USA	98055
7131	323-32-4444	565-67-3920	Waters	Johnny	206 323-9020	1 Pelly Ave. N	Renton	WA	USA	98055
7131	238-32-0079	238-32-0079	Scrubb	Billy	206 444-3232	90 Vashion St. NE	Renton	WA	USA	98059
7131	349-00-9393	565-67-3920	Tuwel	Terry	206 323-3233	66 Sunset Blvd. SW	Renton	WA	USA	98055
8042	723-99-9329	723-99-9329	Crook-shank	Flanner y	503 323-9090	SW 222 Laber Ct.	Port-land	OR	USA	97236
8042	343-32-0034	723-99-9329	Sitwell	Daven-port	503 333-9494	14 SE Krause Lane	Port-land	OR	USA	97236
8042	877-12-9393	723-99-9329	Bartholo-mew	Curan	503 333-9332	18 SE Knee St.	Port-land	OR	USA	97266
8042	323-99-0009	723-99-9329	Yaddo	Mary	503 333-4844	NE Klickitat St.	Port-land	OR	USA	97212

stor-_id	emp-_id	mgr-_id	emp_-lname	emp_-fname	phone	address	city	state	coun-try	postal-code
8042	434-94-3203	723-99-9329	Peacock	Mavis	503 494-3413	4978 SW Huns Rd.	Port-land	OR	USA	97223
8042	232-94-5885	723-99-9329	Baldwin	Dorothy	503 333-9494	9 N. Image Canoe Ave.	Port-land	OR	USA	97217
8042	323-09-5872	723-99-9329	Corbit	P.	503 335-3491	89 NW Industry St.	Port-land	OR	USA	97210
8042	323-54-3434	723-99-9329	Pilkinton	Faith	503 333-4531	18 SW Inez St.	Port-land	OR	USA	97224
8042	331-90-8484	723-99-9329	Botteghe	Obscura	503 889-2832	555 SW Keerins Ct.	Port-land	OR	USA	97223
8042	884-32-8282	723-99-9329	Martineau	Tristan	503 323-5954	346 N. Leade Rd.	Port-land	OR	USA	97203
8042	112-90-3829	723-99-9329	Laughton	Karla	503 323-5929	900 N. Magpie St.	Port-land	OR	USA	97202
8042	434-23-9292	723-99-9329	MacLeish	Cochran	503 433-9493	1722 SE Ogden Ct.	Port-land	OR	USA	97202
6380	433-01-3922	433-01-3922	Horgran	Eric	206 323-4983	18 Thud Dr. S.	Seattle	WA	USA	98198
6380	232-30-9999	433-01-3922	Farnsdale	Finwick	206 434-4949	2 Bellevue Ave.	Seattle	WA	USA	98118
6380	333-90-3828	433-01-3922	Wolfe	Wanda	206 323-3333	2 S. Boze St. #15	Seattle	WA	USA	98108
6380	322-09-2122	433-01-3922	Begude	Nina	206 333-9023	1297 W. Cay Way	Seattle	WA	USA	98108
6380	342-78-2832	433-01-3922	Howe	Saphron	206 323-4933	9827 Des Moines Dr.	Seattle	WA	USA	98108

stor-_id	emp-_id	mgr-_id	emp_-lname	emp_-fname	phone	address	city	state	coun-try	postal-code
6380	882-00-3234	433-01-3922	Salidin	Uma	206 390-8328	13 Echo Lake Place	Seattle	WA	USA	98126
6380	321-89-8832	433-01-3922	Marsicano	Evan	206 322-9392	7 Dancer Ave. SWí	Seattle	WA	USA	98126
6380	762-32-5555	433-01-3922	Boulignini	Gus	206 323-9396	308 S. Fidalgo St.	Seattle	WA	USA	98108
7896	433-01-3921	433-01-3921	Malraison	Marcel	206 889-9983	2298 Berry Dr.	Fremont	CA	USA	94555
7896	322-32-8382	433-01-3921	West	Diana	206 898-3233	20-B Knute St.	Fremont	CA	USA	94536
7896	328-74-8748	433-01-3921	Cazalis	Mari	206 326-6785	18 May Dr. #7	Fremont	CA	USA	94555
7896	325-87-6766	433-01-3921	Zorronius	Gaius	206 363-8988	1967 Claude Dr.	Fremont	CA	USA	94555
7896	323-67-5646	433-01-3921	Podding	Marvin	206 438-7858	242 Ram Ct. #123	Fremont	CA	USA	94539
7896	673-03-6728	433-01-3921	Phelan	Micky	206 323-7676	1212 Spinn Ct.	Fremont	CA	USA	94539
7896	326-77-6482	433-01-3921	Bokanow-ski	Peter	206 432-4832	13 Sparto Dr.	Fremont	CA	USA	94539
7896	653-48-2975	433-01-3921	Poodals	F.W.	206 387-6863	739 Sendak Dr.	Fremont	CA	USA	94538
7896	320-37-4682	433-01-3921	Scrubbs	Randy	206 323-3874	18 Washo Dr. #A	Fremont	CA	USA	94539
7896	323-67-4762	433-01-3921	Mullins	M.	206 382-7862	1717 Zircon Ter.	Fremont	CA	USA	94555

stor-_id	emp-_id	mgr-_id	emp_-lname	emp_-fname	phone	address	city	state	coun-try	postal-code
7896	326-76-3863	433-01-3921	Clarac	Sebastian	206 387-6732	5622 Zin St. #78	Fremont	CA	USA	94539
5023	534-92-2910	534-92-2910	Armand	Mary	510 323-5478	1912 Badger Ln.	Con-cord	CA	USA	94521
5023	342-38-2782	534-92-2910	Blanzat	Gomez	510 382-3828	4229 Tiffany Pl.	Con-cord	CA	USA	94518
5023	322-39-3222	534-92-2910	Winate	Patrick	510 322-5873	16 Mahoo Ln.	Con-cord	CA	USA	94521
5023	932-20-4389	534-92-2910	Burke	Aldous	510 433-4829	12 Ponder Dr. #4	Con-cord	CA	USA	94521
5023	322-43-2828	534-92-2910	Rough-head	Jonathan	510 323-4782	2 Athos Ct.	Con-cord	CA	USA	94521
5023	832-32-3289	534-92-2910	Johnson	Alexis	510 323-3824	927 Aramis Ct.	Con-cord	CA	USA	94520
5023	432-32-4444	534-92-2910	Blitzstein	Jack	510 323-3874	2 Porthos Ct.	Con-cord	CA	USA	94519
5023	543-43-8438	534-92-2910	Darby	Andy	510 422-3232	222 Juniper Ct.	Con-cord	CA	USA	94519
5023	423-42-9933	534-92-2910	Askanzi	Robert	510 422-3232	222 Juniper Ct.	Con-cord	CA	USA	94519
5023	838-28-1982	534-92-2910	Zavatta	Virginia	510 423-4932	629 Wind Way	Con-cord	CA	USA	94521
5023	433-43-3838	534-92-2910	Fife	Ben	510 322-8427	782 Sapling Ct.	Con-cord	CA	USA	94519

roysched 表

roysched 定义如下：

```
create table roysched
title_id tid not null
    references titles(title_id),
lorange int null,
hirange int null,
royalty int null)
```

title_id 列的非聚簇索引定义如下：

```
create nonclustered index titleidind
on roysched (title_id)
```

表 B-9 列出 roysched 的内容：

表 *B-9*：*roysched* 表

title_id	lorange	hirange	royalty
BU1032	0	5000	10
BU1032	5001	50000	12
PC1035	0	2000	10
PC1035	2001	3000	12
PC1035	3001	4000	14
PC1035	4001	10000	16
PC1035	10001	50000	18
BU2075	0	1000	10
BU2075	1001	3000	12
BU2075	3001	5000	14
BU2075	5001	7000	16
BU2075	7001	10000	18
BU2075	10001	12000	20
BU2075	12001	14000	22
BU2075	14001	50000	24
PS2091	0	1000	10
PS2091	1001	5000	12
PS2091	5001	10000	14
PS2091	10001	50000	16
PS2106	0	2000	10
PS2106	2001	5000	12
PS2106	5001	10000	14
PS2106	10001	50000	16

title_id	lorange	hirange	royalty
MC3021	0	1000	10
MC3021	1001	2000	12
MC3021	2001	4000	14
MC3021	4001	6000	16
MC3021	6001	8000	18
MC3021	8001	10000	20
MC3021	10001	12000	22
MC3021	12001	50000	24
TC3218	0	2000	10
TC3218	2001	4000	12
TC3218	4001	6000	14
TC3218	6001	8000	16
TC3218	8001	10000	18
TC3218	10001	12000	20
TC3218	12001	14000	22
TC3218	14001	50000	24
PC8888	0	5000	10
PC8888	5001	10000	12
PC8888	10001	15000	14
PC8888	15001	50000	16
PS7777	0	5000	10
PS7777	5001	50000	12
PS3333	0	5000	10
PS3333	5001	10000	12
PS3333	10001	15000	14
PS3333	15001	50000	16
BU1111	0	4000	10
BU1111	4001	8000	12
BU1111	8001	10000	14
BU1111	12001	16000	16
BU1111	16001	20000	18
BU1111	20001	24000	20
BU1111	24001	28000	22
BU1111	28001	50000	24
MC2222	0	2000	10
MC2222	2001	4000	12
MC2222	4001	8000	14

title_id	lorange	hirange	royalty
MC2222	8001	12000	16
MC2222	8001	12000	16
MC2222	12001	20000	18
MC2222	20001	50000	20
TC7777	0	5000	10
TC7777	5001	15000	12
TC7777	15001	50000	14
TC4203	0	2000	10
TC4203	2001	8000	12
TC4203	8001	16000	14
TC4203	16001	24000	16
TC4203	24001	32000	18
TC4203	32001	40000	20
TC4203	40001	50000	22
BU7832	0	5000	10
BU7832	5001	10000	12
BU7832	10001	15000	14
BU7832	15001	20000	16
BU7832	20001	25000	18
BU7832	25001	30000	20
BU7832	30001	35000	22
BU7832	35001	50000	24
PS1372	0	10000	10
PS1372	10001	20000	12
PS1372	20001	30000	14
PS1372	30001	40000	16
PS1372	40001	50000	18

discounts 表

discounts 定义如下：

```
create table discounts
(discounttype varchar(40) not null,
stor_id char(4) null
    references stores(stor_id),
lowqty smallint null,
highqty smallint null,
```

```
discount float not null)
```

表 B-10 列出 discounts 的内容：

表 *B-10*：*discounts* 表

discounttype	stor_id	lowqty	highqty	discount
Initial Customer	8042	NULL	NULL	10.5
Volume Discount	NULL	100	1001	6.7
Huge Volume Discount	NULL	1001	NULL	10
Customer Discount	8042	NULL	NULL	5

blurbs 表

blurbs 定义如下：

```
create table blurbs
(au_id  id not null
    references authors(au_id),
copy text null)
```

表 B-11 列出 blurbs 的内容：

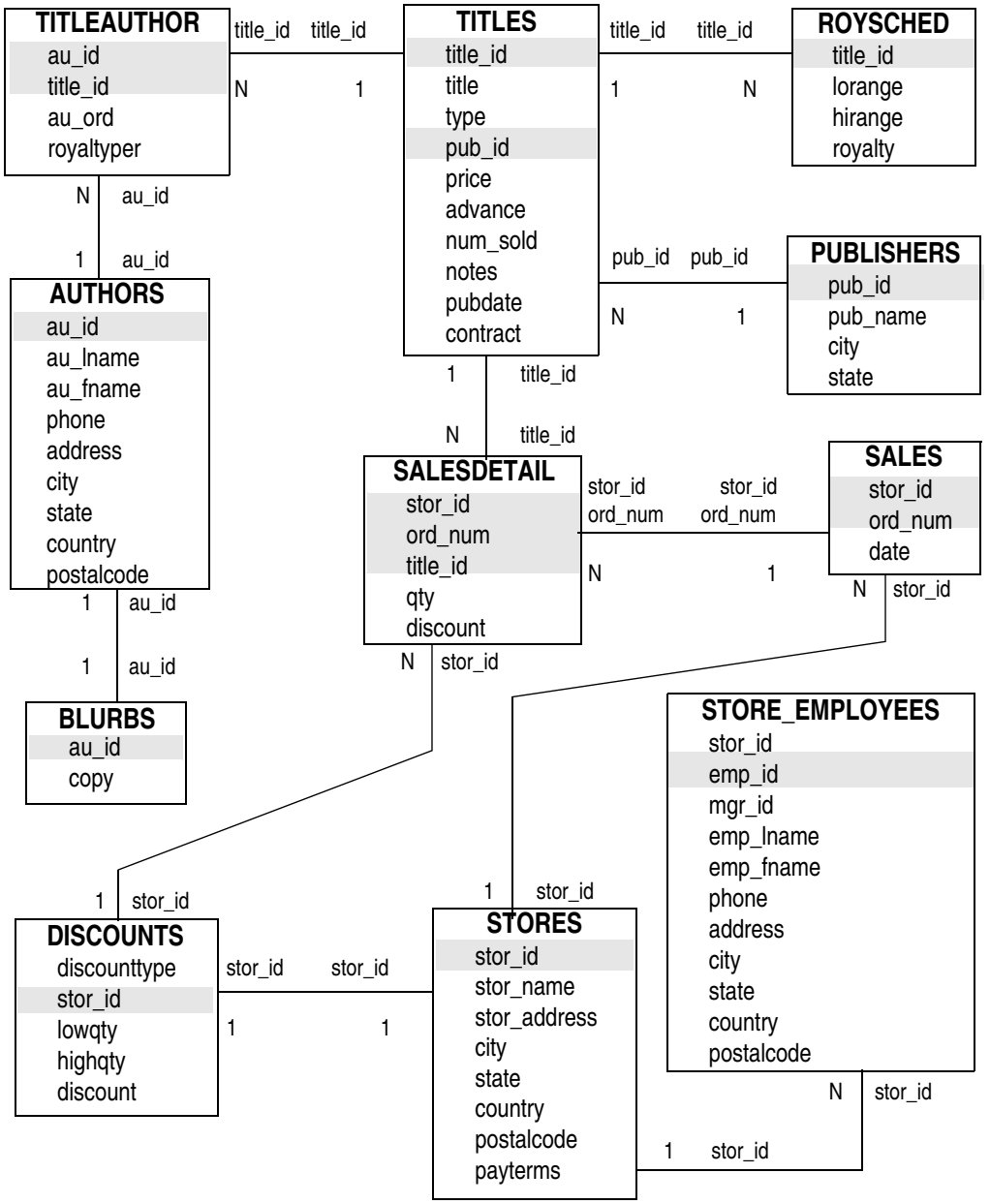
表 *B-11*：*blurbs* 表

au_id	copy
486-29-1786	If Chastity Locksley didnít exist, this troubled world would have created her! Not only did she master the mystic secrets of inner strength to conquer adversity when she encountered it in life, but also, after "reinventing herself," as she says, by writing "Emotional Security: A New Algorithm" following the devastating loss of her cat, Old Algorithm, she founded Publish or Perish, the page-by-page, day-by-day, write-yourself-to-wellness encounter workshops franchise empire, the better to share her inspiring discoveries with us all. Her "Net Etiquette," a brilliant social treatise in its own right and a fabulous pun, is the only civilized alternative to the gross etiquette often practiced on the public networks.
648-92-1872	A chefís chef and a raconteurís raconteur, Reginald Blotchet-Halls calls London his second home. "Thí palace kitchenís me first ëome, actílly!" Blotchet-Hallsí astounding ability to delight our palates with palace delights is matched only by his equal skill in satisfying our perpetual hunger for delicious back-stairs gossip by serving up tidbits and entrees literally fit for a king!

au_id	copy
998-72-3567	Albert Ringer was born in a trunk to circus parents, but another kind of circus trunk played a more important role in his life years later. He grew up as an itinerant wrestler and roustabout in the reknowned Ringer Brothers and Betty and Bernieís Circus. Once known in the literary world only as Anne Ringerís wrestling brother, he became a writer while recuperating from a near-fatal injury received during a charity benefit bout with a gorilla. "Slingshotting" himself from the ring ropes, Albert flew over the gorillaís head and would have landed head first on the concrete. He was saved from certain death by Nana, an elephant he had befriended as a child, who caught him in her trunk. Nana held him so tightly that three ribs cracked and he turned blue from lack of oxygen. "I was delirious. I had an out-of-body experience! My whole life passed before me eyes. I promised myself ëIf I get through this, Iíll use my remaining time to share what I learned out there.í I owe it all to Nana!"
899-46-2035	Anne Ringer ran away from the circus as a child. A university creative writing professor and her family took Anne in and raised her as one of their own. In this warm and television-less setting she learned to appreciate the great classics of literature. The stream of aspiring and accomplished writers that flowed constantly through the house confirmed her repudiation of the circus family sheíd been born into: "Barbarians!" The steadily growing recognition of her literary work was, to her, vindication. When her brotherís brush with death brought them together after many years, she took advantage of lifeís crazy chance thing and broke the wall of anger that she had constructed to separate them. Together they wrote, "Is Anger the Enemy?" an even greater blockbuster than her other collaborative work, with Michel DeFrance, "The Gourmet Microwave."
672-71-3249	They asked me to write about myself and my book, so here goes: I started a restaurant called "de Gustibus" with two of my friends. We named it that because you really canít discuss taste. Weíre very popular with young business types because weíre young business types ourselves. Whenever we tried to go out to eat in a group we always got into these long tiresome negotiations: "I just ate Italian," or "I ate Greek yesterday," or "I NEVER eat anything thatís not organic!" Inefficient. Not what business needs today. So, it came to us that we needed a restaurant we could all go to every day and not eat the same thing twice in a row maybe for a year! We thought, "Hey, why make people choose one kind of restaurant over another, when what they really want is a different kind of food?" At de Gustibus you can eat Italian, Chinese, Japanese, Greek, Russian, Tasmanian, Iranian, and on and on all at the same time. You never have to choose. You can even mix and match! We just pooled our recipes, opened the doors, and never looked back. Weíre a big hit, what can I say? My recipes in "Sushi, Anyone?" are used at de Gustibus. They satisfy crowds for us every day. They will work for you, too. Period!
409-56-7008	Bennet was the classic too-busy executive. After discovering computer databases, he now has the time to run several successful businesses and sit on three major corporate boards. Bennet also donates time to community service organizations. Miraculously, he also finds time to write and market executive-oriented, in-depth computer hardware and software reviews. "Iím hyperkinetic, so being dynamic and fast-moving is a piece of cake. But being organized isnít easy for me or for anyone I know. Thereís just one word for that: ëdatabases!í Databases can cure you or kill you. If you get the right one, you can be like me. If you get the wrong one, watch out. Read my book!"

pubs3 数据库框图

图 *B-1*：*pubs3* 数据库框图



索引

字母

& （与符号）
“与”逐位运算符 14, 44
@ （at 符号）
规则参数和 394
过程参数和 445
局部变量名 429
@@ （at 符号），全局变量名 434
% （百分比符号）
算术运算符 （模）13, 44–48
!> （不大于）比较运算符 16, 55
!= （不等于）比较运算符 16, 54
<> （不等于）比较运算符 16, 54
!< （不小于）比较运算符 16, 55
> （大于）
比较运算符 16, 54
范围指定 56
>= （大于或等于）比较运算符 16, 54
= （等号）
比较运算符 16, 54
=* （等号星号）外部连接运算符 119, 147
, （逗号）
在货币值的缺省输出格式中 187
\ （反斜杠）
字符串延续 18
~ （否定符号）
“非”逐位运算符 14, 44
- （负号）
用于负的货币值 285
算术运算符 44–48
+ （加号）
空值和 70
算术运算符 13, 44–48
字符串并置运算符 15, 346
^ （尖号）
“异或”逐位运算符 14, 44
- （减号）
算术运算符 13
（井号），临时表标识符前缀 213, 220, 222
$ （美元符号）
标识符中 8
在 money 数据类型中 285
￥ （日元符号）
标识符中 8
在 money 数据类型中 285
| （竖线）
“或”逐位运算符 14, 44
-- （双连字符）注释 26, 428
“” （双引号）
比较运算符和 16
表达式中 17
给空字符串加引号 17, 291
给口令加引号 31
将参数值引起来 448
将列标题括起来 43
将值引起来 188, 280
文字说明 17
_ （下划线）
临时表名中 222
() （小括号）
表达式中 12
在内部函数中 336
在匹配列表中 57
在算术语句中 47
使用 **union** 运算符 108
在系统函数中 336
< （小于）
比较日期 54
比较运算符 16
范围查询和 56
<= （小于或等于）比较运算符 16, 54
/ （斜杠）
算术运算符 （除法）13, 44–48
*/ （星号斜线），注释关键字 427
/* （斜线星号），注释关键字 427
* （星号）
select 和 40
表示超长数字 343
乘法运算符 13, 44–48

括住注释的符号对 427
*= （星号等号）外部连接运算符 119, 147
*/ （星号斜线），注释关键字 427
£ （英镑符号）
标识符中 8
在 money 数据类型中 285
“0x” 285
在 *textsize* 中计算 49
@@char_convert 全局变量 437, 438
@@cis_version 全局变量 439
@@client_csid 全局变量 438
@@client_csname 全局变量 438
@@connections 全局变量 438
@@cpu_busy 全局变量 438
@@error 全局变量 434
select into 和 247
@@guestuserid 全局变量 439
@@identity 全局变量 219, 294, 434
@@idle 全局变量 438
@@invaliduserid 全局变量 439
@@io_busy 全局变量 438
@@isolation 全局变量 437, 583
@@langid 全局变量 438
@@language 全局变量 438
@@maxcharlen 全局变量 438
@@max_connections 全局变量 439
@@maxgroupid 全局变量 439
@@maxuserid 全局变量 439
@@mingroupid 全局变量 439
@@minsuid 全局变量 439
@@minuserid 全局变量 439
@@ncharsize 全局变量 438
@@nestlevel 全局变量
嵌套触发器和 522
嵌套过程和 453
@@options 全局变量 437
@@packet_errors 全局变量 438
@@pack_received 全局变量 438
@@pack_sent 全局变量 438
@@parallel_degree 全局变量 437
@@procid 全局变量 439
@@rowcount 全局变量 437
触发器和 507
游标和 547
@@scan_parallel_degree 全局变量 437
@@servername 全局变量 439
@@spid 全局变量 439
@@sqlstatus 全局变量 435
@@textcolid 全局变量 440
@@textdbid 全局变量 440
@@textobjid 全局变量 440
@@textptr 全局变量 440
@@textsize 全局变量 49, 437, 440
@@textts 全局变量 440
@@thresh_hysteresis 全局变量 439
@@timeticks 全局变量 439
@@total_errors 全局变量 439
@@total_read 全局变量 439
@@total_write 全局变量 439
@@tranchained 全局变量 437
@@trancount 全局变量 435, 576
@@transtate 全局变量 435, 574, 576
@@version 全局变量 440
abs 绝对值数学函数 352
acos 数学函数 352
all 关键字
group by 89
select 51–52
union 108
比较运算符和 165, 174
搜索使用 165
子查询包括 18, 166, 174
allow nested triggers 配置参数 522–524
allow_dup_row 选项， **create index** 380–381
alter database 命令 210–211
另请参见 **create database** 命令
alter table
删除约束 255–256
alter table
alter table modify 生成的错误 265–266
arithabort numeric_truncation 打开或关闭 259
execute immediate 语句于 266
错误消息 264–266
关于远程表 252
和 CIS 252
和转储事务日志 251
减少列长度将截断数据 257
将现有转储批量复制到修改的列 257
具有聚簇索引的表 263
命令 250–266
排它表锁 251
删除 IDENTITY 列 260–261

删除 IDENTITY 列的限制 260–261
删除具有用户定义数据类型的列 264
删除列 254–256
删除列，影响列 ID 255
数据复制 261–262
添加非空列 253
添加 IDENTITY 列 260
添加具有用户定义数据类型的列 263
添加列 252–254
添加列，影响列 ID 253
添加约束 254
修改 datetime 列 258
修改带有标度的列 259
修改带有精度的列 259
修改具有用户定义数据类型的列 264
修改列 256–259
修改 NULL 缺省值 258
修改锁定方案 263
修改 *text* 和 *image* 列 259
语法 250
在数据复制期间修改 **exp_row_size** 262
执行 **select *** 的对象 251
执行数据复制时 262
转换数据类型 257
alter table 命令
添加 *timestamp* 列 558
and 关键字
表达式中 19
在连接中 121
在搜索条件中 70–72
ansinull 选项， **set** 28
any 关键字
表达式中 18
搜索使用 165
子查询使用 167–170, 174
arithabort 选项， **set**
arith_overflow 和 27, 366
数学函数和 **arith_overflow** 366
数学函数和 **numeric_truncation** 367
arithignore 选项， **set**
arith_overflow 和 27, 366
ASCII 字符
ascii 字符串函数和 340, 346
ascii 字符串函数 340, 346
asin 数学函数 352
atan 数学函数 353
at 符号 (@)
规则自变量和 394
过程参数和 445
局部变量名 429
atn2 数学函数 353
avg 集合函数 74, 351
作为行集合 101
*另请参见*集合函数
au_pix 表， *pubs2* 数据库 631
authors 表
pubs2 数据库 608–610
pubs3 数据库 636–638
auto identity 数据库选项 220
identity in nonunique indexes 和 376
bcp （批量复制实用程序）
IDENTITY 列和 296
begin transaction 命令 572
begin...end 命令 418
between 关键字 56–57
check 约束，使用 239
binary 数据类型 192
like 和 60
*参见*数据类型
bit 数据类型 193
另请参见 “数据类型”
blurbs 表
pubs2 数据库 629
pubs3 数据库 662
break 命令 419–421
case 表达式 409–418
coalesce 416
when\xc9 then 关键字 413
比较值和 **nullif** 417
避免除零 411
存储过程示例 447
确定数据类型 412
数据表示 410
搜索条件于 413
值比较和 415
ceiling 数学函数 353
chained 选项， **set** 579
@@char_convert 全局变量 437, 438
charindex 字符串函数 340, 342–343
char_length 字符串函数 340
char 数据类型 188–189
表达式中 17

*另请参见*字符数据；数据类型 339
char 字符串函数 340
checkpoint 命令 598
@@cis_version 全局变量 439
@@client_csid 全局变量 438
@@client_csname 全局变量 438
close on endtran 选项， **set** 596
close 命令 550
clustered 约束
  create index 378
  create table 236
coalesce 关键字， **case** 416
col_length 系统函数 334, 337
col_name 系统函数 334
commit 命令 572
compute 子句 99–107
  union 和 111
  不同集合 105
  多列和 103, 104
  分组摘要的小计 99
  使用多个 103
  行集合和 101–105
  游标中不允许 538
  子群和 103
  总和 106–107
@@connections 全局变量 438
continue 命令 419–421
convert 函数 194, 362–369
  并置和 15, 339, 347
  截断值 363
  缺省长度 362
  数据类型 368
  显式转换 351
cos 数学函数 353
cot 数学函数 353
count(*) 集合函数 73–74, 76, 101
  包括空值 77
  带有空值的列上 87
  *另请参见*集合函数
count 集合函数 74, 351
  带有空值的列上 77–78, 87
  作为行集合 101
  *另请参见*集合函数
@@cpu_busy 全局变量 438
create database 命令 207–211
  使用批处理 403
create default 命令 389
  create procedure 用 465
  批处理和 402
create index 命令 373–382
  ignore_dup_key 375
  使用批处理 403
create procedure 命令 444–445, 445–457, 495
  output 关键字 460–464
  with recompile 选项 453
  *另请参见*存储过程；扩展存储过程 (ESPs)
  规则 465
  空值和 450
  使用批处理 402
create rule 命令 394
  create procedure 用 465
  使用批处理 402
create schema 命令 238
create table 命令 212–214
  在不同的数据库中 224
  在存储过程中 466
  空值和 290
  空值类型和 215
  示例 214, 241
  使用批处理 403
  用户定义的数据类型和 198
  约束和 232
  组合索引和 375
create trigger 命令 503–505
  create procedure 用 465
  使用批处理 402
  显示文本 530
create view 命令 313
  create procedure 用 465
  禁止将 **union** 用于 112
  使用批处理 403
  语法 314
CS_DATAFMT 结构 485
CS_SERVERMSG 结构 485
cursor rows 选项， **set** 546
curunreservedpgs 系统函数 334
datalength 系统函数 335, 337, 349
data_pgs 系统函数 335
dateadd 函数 355, 359
datediff 函数 355, 359
dateformat 选项， **set** 282–284
datename 函数 284, 355–358

datepart 函数 284, 355–358
datetime 数据类型 187, 281, 355
  like 和 60
  比较 16
  并置 347
  存储 355
  条目格式 281, 355
  运算 355
  另请参见 “日期”； “*timestamp* 数据类型”
dayofyear 日期分量缩写和值 356
day 日期分量 356
dbcc （数据库一致性检查程序）
  存储过程和 466
db_id 系统函数 335
DB-Library 程序
  set 选项 22
  事务和 593
db_name 系统函数 335
dd
  参见 **day** 日期分量
deallocate cursor 命令 550
decimal 数据类型 286
declare cursor 命令 538
declare 命令 429
default 关键字 234
  create database 208
degrees 数学函数 353
deleted 表 506–507, 538
delete 命令 305–307, 326
  *另请参见*删除
  触发器和 504, 506–507, 509–510
  多表视图和 324
  视图和 324
  游标和 549
  子查询和 159
difference 字符串函数 340, 344
discounts 表
  pubs2 数据库 629
  pubs3 数据库 661
distinct 关键字
  order by 和 98
  select 51–52
  select，空值和 52
  表达式子查询使用 164
  集合函数和 74, 76
  行集合和 102
  游标和 542
DLL （动态链接库）
  *参见*动态链接库
double precision 数据类型 186
drop database 命令 211
drop default 命令 393
drop index 命令 382
drop procedure 命令 452, 467, 496
drop rule 命令 398
drop table 命令 268
drop trigger 命令 529
drop view 命令 329
dw
  参见 **weekday** 日期分量
dy
  请参见 **dayofyear** 日期分量
e 或 E 指数符号
  money 数据类型 285
  近似数值数据类型 286
else 关键字
  参见 **if...else** 条件
end 关键字 413, 418
errorexit 关键字 **waitfor** 426
@@error 全局变量 434
  select into 和 247
ESP
  *参见*扩展存储过程
ESP DLL 的 makefile 492–494
esp execution priority 配置参数 483
esp unload dll 配置参数 480, 483
execute 命令 12, 445
  output 关键字 460–464
  with recompile 选项 453
  用于 ESP 497
exists 关键字 176–177, 408
  搜索条件 173
exp 数学函数 353
FALSE，返回值 173
fetch 命令 545
FIPS 标志程序 25
fipsflagger 选项， **set** 25
float 数据类型 186
  计算 354
  条目格式 286
  另请参见 “数据类型”
floor 数学函数 353

flushmessage 选项， **set** 22
for browse 选项， **select** 112
    游标中不允许 538
for load 选项
    alter database 210
    create database 210
for read only 选项， **declare cursor** 538, 542
for update 选项， **declare cursor** 538, 542
foreign key 约束 237
from 关键字 52
    更新 303
    连接 116
    删除 306
futureonly 选项
    规则 395, 398
    缺省值 391, 392
getdate 日期函数 355–358
go 命令终结符 34, 401
goto 关键字 421
grant 命令 269
group by 子句 78–86, 542
    all 和 89
    having 子句和 91–93
    order by 和 97
    select 80
    where 子句和 87–88
    union 和 111
    不带集合函数 78, 85
    触发器使用 516–518
    多列 85
    集合函数和 78–93, 99
    空值和 86–87
    嵌套 81–86
    相关子查询比较 181
    子查询使用 164
@@guestuserid 全局变量 439
guest 用户 205, 334
Halloween 问题 544
having 子句 91–95
    group by 和 91
    union 和 111
    不带 **group by** 95
    不带集合 91
    逻辑运算符和 92
    与 **where** 子句不同 91
    子查询使用 164, 182
hextoint 函数 194, 368
hh
    参见 **hour** 日期分量
holdlock 关键字 538, 569
    readtext 349
    游标和 556
host_id 系统函数 335
host_name 系统函数 335
hour 日期分量 356
identity grab size 配置参数 294
identity in nonunique index 数据库选项 375, 541
identity 关键字 217
identity_insert 选项， **set** 293
IDENTITY 列 217–220
    @@identity 全局变量 434
    syb_identity 关键字 220
    保留块 294
    创建表 217
    非唯一索引 375
    和参照完整性 219
    间隔，消除 296
    将值插入 293, 328
    删除 260–261
    删除的限制 260–261
    视图和 318–319, 328
    视图中的 **syb_identity** 318
    使用 **select into** 248–250
    数据类型 217
    添加 260
    唯一值用于 293
    显式值用于 293
    选择 220, 248
    用户定义的数据类型和 218
    用 **syb_identity** 关键字更新 304
    用 **syb_identity** 关键字删除 307
    值间隔 230–231
    值中的间隙 296
    重新编号 296
    组件集成服务 220
IDENTITY 列的显式值 293
IDENTITY 列值中的间隔 224–231
@@identity 全局变量 219, 294, 434
@@idle 全局变量 438
ID，用户
    user_id 函数 336
    服务器用户 336

数据库 (**db_id**) 335
if update 子句， **create trigger** 503, 526–527
if...else 条件 408–409, 419
case 表达式相比于 409
ignore_dup_key 选项， **create index** 375, 380
ignore_dup_row 选项， **create index** 380–381
image 数据类型 193
"0x" 前缀 285
union 不允许用于 112
writetext 到 325
另请参见 "数据类型"
被禁止的动作 97
插入 288
触发器和 526
更新 300
空值位于 217
选择 49–350
用空值初始化 217
用 **writetext** 更改 304
在视图中选择 321
子查询使用 155
组件集成服务 193
index_col 系统函数 335
in 关键字 57–59
check 约束，使用 239
表达式中 18
子查询使用 168, 170–172
inserted 表 506–507, 538
insert 命令 287–299, 326
IDENTITY 列和 292
image 数据和 193
text 数据和 191
union 运算符 112
触发器和 504, 506–507, 508
规则和 387
空 / 非空列和 290, 291
视图和 324
使用批处理 403
选择 288
重复数据来自 380, 381
子查询和 159
installmaster 脚本 468
into 关键字
union 111
插入 288
读取 545
选择 244
into 子句， **select**
参见 **select into** 命令
int 数据类型 287
另请参见 "整数数据"，"*smallint* 数据类型"，"*tinyint* 数据类型"
inttohex 函数 194, 368
@@invaliduserid 全局变量 439
@@io_busy 全局变量 438
is null 关键字 69, 291
isnull 系统函数 69, 335, 338
insert 和 291
iso_1 字符集 9
@@isolation 全局变量 437, 583
isql 实用程序命令 xvii, 33–34
go 命令终结符 34, 401
批处理文件用于 406–407
is_sec_service_on 安全函数 369
@@langid 全局变量 438
@@language 全局变量 438
language 选项， **set** 282, 284
lct_admin 系统函数 335
like 关键字 60–65
check 约束，使用 239
搜索日期 284
lock table 命令 588–589, 600
错误消息 12207 和 589, 600
lock wait period 配置参数 601
lock wait 选项， **set** 命令 600
log on 选项
create database 209
log10 数学函数 353
log 数学函数 353
lower 字符串函数 340
ltrim 字符串函数 340
Macintosh 字符集 9
master 数据库 206
guest 用户位于 205
@@maxcharlen 全局变量 438
@@max_connections 全局变量 439
@@maxgroupid 全局变量 439
max 集合函数 74, 351
作为行集合 101
*另请参见*集合函数
@@maxuserid 全局变量 439
mi

参见 **minute** 日期分量
millisecond 日期分量 356
@@mingroupid 全局变量 439
min 集合函数 74, 351
作为行集合 101
*另请参见*集合函数
@@minsuid 全局变量 439
@@minuserid 全局变量 439
minute 日期分量 356
mirrorexit 关键字 426
mm
参见 **month** 日期分量
model 数据库 197, 206
用户定义的数据类型位于 221
money 数据类型 187, 196
另请参见 "*smallmoney* 数据类型"
条目格式 285
month 日期分量 356
ms
参见 **millisecond** 日期分量
"N/A" 使用 "NULL" 或 290
@@ncharsize 全局变量 438
nchar 数据类型 188–189
like 和 60
操作 339–348
输入规则 280
@@nestlevel 全局变量
嵌套触发器和 522
嵌套过程和 453
noholdlock 关键字， **select** 585
nonclustered 约束
create index 378
"none"，使用 "NULL" 或 290
not between 关键字 56
not exists 关键字 176–177
另请参见 **exists** 关键字
not in 关键字 57–59
空值和 173
子查询使用 172–173
not like 关键字 62
not null 关键字 198, 215, 243
*另请参见*空值
not 关键字
另请参见 "逻辑运算符"
搜索条件 55, 71–72
nvarchar 数据类型 189
like 和 60
另请参见 "字符数据"，"数据类型"
操作 339–348
输入规则 280
null 关键字 66, 235
*另请参见*空值
缺省值和 215, 392
用户定义的数据类型 198
nullif 关键字 417
numeric 数据类型 286
object_id 系统函数 335
object_name 系统函数 335
of 选项， **declare cursor** 538
on 关键字
alter database 210
create database 208
create index 374, 379, 382
create table 214
on 子句
内部连接的 （ANSI 语法）135–136
在内部表上 （ANSI 语法）140
在外部连接中 （ANSI 语法）139–142
Open Client 应用程序
set 选项 22
open 命令 544
@@options 全局变量 437
or (|) 逐位运算符 71–72
另请参见 "逻辑运算符"
order by 子句
compute by 和 102
group by 和 97
select 和 96–97
union 和 111
索引和 373
or 关键字
表达式中 19
在连接中 121
output 选项 460–464
@@packet_errors 全局变量 438
@@pack_received 全局变量 438
@@pack_sent 全局变量 438
@@parallel_degree 全局变量 437
patindex 字符串函数 340
text/image 函数 348
示例使用 342–343
数据类型 339

*另请参见*通配符
pi 数学函数 353
power 数学函数 353
print 命令 422–424, 425
processexit 关键字， **waitfor** 426
@@procid 全局变量 439
proc_role 系统函数 460
publishers 表
pubs2 数据库 607
pubs3 数据库 635
pubs2 数据库 35, 607–634
guest 用户位于 205
表名 607
更改数据于 279
结构图 634
框图 634
示例中使用 xvii
pubs3 数据库 35, 635–664
表名 635
qq
参见 **quarter** 日期分量
quarter 日期分量 356
radians 数学函数 353
raiserror 命令 424
rand 数学函数 353
readpast 选项 602–606
隔离级别和 604
readtext 命令 49
隔离级别和 584
和视图 321
readtext 命令 （高级）349–350
real 数据类型
另请参见 “数据类型”； “数值数据”
real 数据类型 186, 286
references 约束 237
replicate 字符串函数 340
reserved_pgs 系统函数 335
return 命令 422, 459
reverse 字符串函数 340
revoke 命令 269
right 字符串函数 340, 346
rollback trigger 命令 520
rollback 命令 573
*另请参见*事务 279, 568
触发器和 520, 597
在存储过程中 597
rowcnt 系统函数 335
@@rowcount 全局变量 437
触发器和 507
游标和 547
round 数学函数 353
roysched 表
pubs2 数据库 626–628
pubs3 数据库 659–661
RPC （远程过程调用）
*参见*远程过程调用
rtrim 字符串函数 340
sales 表
pubs2 数据库 624
pubs3 数据库 651
salesdetail 表
pubs2 数据库 619–623
pubs3 数据库 647–651
save transaction 命令 573
*另请参见*事务 279, 568
@@scan_parallel_degree 全局变量 437
second 日期分量 356
select * 命令 40–41, 116
限制 298–299
新列和限制 453
select distinct 查询， **group by** 和 98
select distinct 查询， **order by** 和 98
select into/bulkcopy/pllsort 数据库选项
select into 和 244
writetext 和 304
select into 命令 244–246
compute 和 102
IDENTITY 列和 249
union 和 111
临时表 223
游标中不允许 538
select 命令 3, 37, 38–72
create view 和 315
if...else 关键字和 408
image 数据 49–350
text 数据 49–350
*另请参见*连接；子查询；视图
保留字在 43
变量赋值和 421–433
变量赋值和多行结果 430
布尔表达式于 408
插入数据用 288, 289, 296–299

使用 **distinct** 51–52
隔离级别和 584
计算 44–48
列标题 42
列顺序在 41
匹配字符串，使用 60–65
和视图 324
数据库对象名和 39
通配符在 60–63
为结果创建表 244–246
显示结果 38, 41–44
显示中的字符串 43
消除重复行用 51–52
选择列 37–38
选择行 37, 53
引号标记在 43
用于浏览 557
字符数据在 65
组合结果 107–112
@@servername 全局变量 439
set 命令
update 中 301
链式事务模式 26
事务隔离级别和 581
在存储过程内 455
字符串截断和 27
setuser 命令 207, 269
shared 关键字
锁定和 585
游标和 556
show_role 系统函数 370
show_sec_services 安全函数 370
sign 数学函数 353
sin 数学函数 353
smalldatetime 数据类型 187
另请参见 *“日期*数据类型”；*“timestamp* 数据类型”
存储 355
日期函数和 359–360
条目格式 281, 355
运算 355–359
转换 368
smallint 数据类型 287
另请参见 *“int* 数据类型”；*“tinyint* 数据类型”
smallmoney 数据类型 187, 196
另请参见 *“money* 数据类型”
条目格式 285
sorted_data 选项， **create index** 381–382
soundex 字符串函数 341, 344
space 字符串函数 341
sp_addextendedproc 系统过程 496
sp_addmessage 系统过程 425
sp_addtype 系统过程 183, 197, 221
sp_bindefault 系统过程 199, 390–391, 392
使用批处理 403
sp_bindrule 系统过程 199, 395–397
使用批处理 403
sp_changedbowner 系统过程 207
sp_commonkey 系统过程 152
sp_dboption 系统过程
事务和 571
sp_depends 系统过程 332, 476–477, 531
sp_displaylogin 系统过程 30
sp_dropextendedproc 系统过程 497
sp_dropsegment 系统过程 211
sp_droptype 系统过程 200
sp_extendsegment 系统过程 211
sp_foreignkey 系统过程 152, 272, 505
sp_freedll 系统过程 480
sp_getmessage 系统过程 425
sp_helpconstraint 系统过程 273
sp_helpdb 系统过程 270
sp_helpdevice 系统过程 208
sp_helpextendedproc 系统过程 499
sp_helpindex 系统过程 383
sp_helpjoins 系统过程 152, 505
sp_helprotect 系统过程 477
sp_helptext 系统过程
触发器 529, 530
规则 399
过程 443, 476
缺省值 399
sp_helpuser 系统过程 29
sp_help 系统过程 270–272
IDENTITY 列和 319
@@spid 全局变量 439
sp_modifylogin 系统过程 207
sp_monitor 系统过程 438
sp_password 系统过程 30
远程服务器和 32
sp_primarykey 系统过程 505
sp_procxmode 系统过程 595

sp_recompile 系统过程 444
sp_rename 系统过程 267–268, 323, 466
sp_spaceused 系统过程 274
sp_unbindefault 系统过程 392
sp_unbindrule 系统过程 397
SQL
    参见 Transact-SQL
SQL 标准 25
    set 选项 23, 25
    事务 587
SQL 的改进 19–24
@@sqlstatus 全局变量 435, 546
sqrt 数学函数 353
SRV_PROC 结构 485
ss
    参见 **second** 日期分量
store_employees 表， *pubs3* 数据库 653–658
stores 表
    pubs2 数据库 625
    pubs3 数据库 653
str 函数的右对齐 343
string_rtruncation 选项， **set** 27
str 字符串函数 341, 343
stuff 字符串函数 341, 343
substring 字符串函数 341
sum 集合函数 74, 351
    作为行集合 101
    *另请参见*集合函数
suser_id 系统函数 336
suser_name 系统函数 336
syb_identity 关键字 220, 318
    IDENTITY 列和 220, 293
sybsystemdb 数据库 206
sybsystemprocs 数据库 468, 498
syscolumns 表 193
syscomments 表 443, 529
sysdatabases 表 208
sysdevices 表 208
syskeys 表 152, 505
syslogs 表 597
sysmessages 表
    raiserror 和 424
sysname 自定义数据类型 194
sysobjects 表 271–272
sysprocedures 表 529
sysprocesses 表 426
systypes 表 195, 271, 272
sysusages 表 208
sysusermessages 表 424, 425
sysusers 表 205
tan 数学函数 353
tempdb 数据库 206
@@textcolid 全局变量 440
@@textdbid 全局变量 440
@@textobjid 全局变量 440
textptr 函数 348, 350
@@textptr 全局变量 440
@@textsize 全局变量 49, 437, 440
textsize 选项， **set** 349
text 数据类型 191
    like 和 60, 61
    where 子句和 54, 61
    union 不允许用于 112
    另请参见 “数据类型”
    被禁止的动作 97
    操作 339, 348–351
    插入 288
    触发器和 526
    返回数据的长度 23
    更新 300
    输入规则 280
    选择 49–350
    用空值初始化 217
    用 **writetext** 更改 304
    在视图中更新 325
    在视图中选择 321
    转换 363
    子查询使用 155
    组件集成服务 192
@@textts 全局变量 440
textvalid 函数 349
theta 连接 118
    *另请参见*连接
@@thresh_hysteresis 全局变量 439
timestamp 数据类型 194
    插入数据和 287
    另请参见 “*日期*数据类型”； “*smalldatetime* 数据类型” 187
    浏览模式和 558
    使用 **tsequal** 函数进行比较 336
    跳过 289
@@timeticks 全局变量 439

time 选项， **waitfor** 426
tinyint 数据类型 287
　另请参见 *“int* 数据类型”； *“smallint* 数据类型”
titleauthor 表
　pubs2 数据库 617–619
　pubs3 数据库 645
titles 表
　pubs2 数据库 610–617
　pubs3 数据库 638–645
@@total_errors 全局变量 439
@@total_read 全局变量 439
@@total_write 全局变量 439
to_unichar 字符串函数 341
@@tranchained 全局变量 437
@@trancount 全局变量 435, 576
Transact-SQL
　改进 19–24
　扩展 23–24
@@transtate 全局变量 435
transtate 全局变量 574–576
TRUE，返回值 173
truncate table 命令 307–308
　参照完整性和 239
　触发器和 525
tsequal 系统函数 336, 559
uhighsurr 字符串函数 341
ulowsurr 字符串函数 341
Unichar 数据 189
unichar 数据类型 188–189
　like 和 60
　操作 339–348
union 运算符 107–112, 542
unique 关键字 375
　重复数据来自 380
univarchar 数据类型
　like 和 60
　操作 339–348
update partition statistics 命令 385
update statistics 命令 384
　组件集成服务 384
update 命令 299–304, 326
　image 数据和 193
　text 数据和 191
　触发器和 506–507, 511–516, 526
　多表视图 325
　规则和 387
　空值和 216, 217, 292
　视图和 130, 317, 324
　游标和 548
　重复数据来自 380, 381
　子查询使用 159
upper 字符串函数 341, 346
uscalar 字符串函数 341
used_pgs 系统函数 336
use 命令 207
　create procedure 用 465
　使用批处理 402
user 关键字
　create table 234
　系统函数 336
user_id 系统函数 336
user_name 系统函数 334, 336, 338
user 系统函数 336
using 选项， **readtext** 350
valid_name 系统函数 336
　检查字符串 8
valid_user 系统函数 336
values 选项， **insert** 287–288
varbinary 数据类型 192
　“0x”前缀 285
　like 和 60
　另请参见 “二进制数据”； “数据类型”
　操作 339–348
varchar 数据类型 189
　like 和 60
　另请参见 “字符数据”； “数据类型”
　表达式中 17
　操作 339–348
　输入规则 280
@@version 全局变量 440
waitfor 命令 426–427
weekday 日期分量 356
week 日期分量 356
when...then 条件
　case 表达式中 413
　在 **case** 表达式中 409
where current of 子句 548, 549
where 子句
　group by 子句和 87–88
　join 和 121
　like 和 61
　text 数据和 54, 61

不允许集合函数 74
更新 302
空值位于 67
框架 (skeleton) 表创建 246
连接和 117–118
搜索条件在 53
与 **having** 比较 91
在外部连接中 （ANSI 语法）139–142
子查询使用 155, 172, 173
while 关键字 418–421
with check option 选项
视图和 319–320
with log 选项， **writetext** 279
with recompile 选项
create procedure 453
执行 453
wk
参见 **week** 日期分量
writetext 命令 304
with log 选项 279
和视图 321, 325
work 关键字 （事务）572
XP Server 21, 480, 500
xp_cmdshell context 配置参数 483
xp_cmdshell 系统扩展存储过程 498
xp_cmdshell context 配置参数和 483
xpserver 实用程序命令 500
year 日期分量 356
yy
参见 **year** 日期分量

A

安全性
*另请参见*权限
存储过程如 444
存储过程为 467
视图和 311, 329
安全性函数 369–370
安装
pubs2 中的 *image* 样本 632

B

百分比符号 (%)
模运算符 13, 45–48
绑定
规则 395–397
缺省值 390–391
帮助报告
另请参见各系统过程
触发器 530
规则 399
列 399
缺省值 399
数据库 270
数据库对象 270–272
数据库设备 208
数据类型 270–272
索引 383
文本，对象 476
系统过程 475
依赖性 476
保存点 573
使用 **save transaction** 进行设置 568
备份
*参见*恢复
本地服务器和远程服务器
*参见*远程服务器
比较运算符 54–56
*另请参见*关系表达式
表达式中 16
符号 16, 54
空值和 66, 433
无修饰词的，子查询中 162–164
相关子查询和 180–182
有修饰词的，子查询中 165
比较值
timestamp 336, 559
表达式中 16
空 67, 433
用于连接 119
变更
*参见*改变
变量
本地 465
比较 433
局部 421–433
全局 434–440

全局，全球 465
声明 429–433
输出值 430
在 **update** 语句中 302
最后一行赋值 430
表 2, 211–224
from 子句中所允许的 52
from 子句中允许的 116
IDENTITY 列 217–218
isnull 系统函数和 291
*另请参见*数据库对象、触发器、视图
创建新的 241–250
访问权限对于 269
改变 250–268
连接 113–152
名称，在连接中 117, 123
命名 8–11, 53, 213
内部 129
删除 268
设计 241
使用空间 274
外部 129
相关 509
相关名 53, 123, 157
行拷入 299
重命名 267–268
子查询中的相关名 180
表达式 55, 74
包括空值 69
并置 338, 339, 346–347
的类型 xxii
定义 12
类型 12
用子查询替代 160
转换数据类型 360–365
表达式子查询 162–164
标点符号
用引号引起来 198
表级约束 233
标量集合 75, 86
表列
*参见*列
标签 421
标题，列 42–43
表行
*参见*行，表
表页
系统函数 335, 336
标识符 6–11
带引号 9
分隔 9
系统函数和 336
表，临时
*参见*临时表
表行
*参见*列；表
别名
表相关名 53
并置 346–347
表达式 339
二进制数据 338
使用 + 运算符 15, 346
使用运算符 +，空值和 70
字符串 339
布尔 （逻辑）表达式 12
select 语句于 408
不均等连接 124–127
与子查询比较 172

C

参考信息
Transact-SQL 函数 333
数据类型 183
参数，过程 445–451
不允许的分隔标识符 9
最大数目 465
参照完整性 236–239, 278
create schema 命令 238
IDENTITY 列 219
触发器 505–506
交叉引用表 238
*另请参见*数据完整性；触发器 21
提示 240
一般规则 238
约束 232, 236
允许的最大参照数 237
层次
*另请参见*优先级
数据类型 195–196
运算符 13

插入
    行 287–299
查询 2, 3
    嵌套子查询 158
    投影 3
    优化 444
查询处理 442
查找
    对象依赖性 268
差集 （用 **exists** 和 **not exists**）176–177
长度
    表达式 （以字节表示）335
    列 334
常量 xxii
    表达式中 17
    在表达式中 42
乘法运算符 (*) 13, 45, 48
重复键错误，用户事务 593
重复行
    索引和 380
    用 **union** 删除 108
重复子查询
    *参见*子查询
重命名
    *另请参见*命名
    表 267–268, 324
    存储过程 466
    视图 323, 324
    数据库对象 267–268
重新定位游标 537, 544
触发器 21, 501–532
    rollback 597
    set 命令 527
    truncate table 命令和 525
    测试表 506–507
    创建 503–505
    存储 529
    递归 523
    对象重命名和 527
    规则和 502
    回退 520
    空值和 526–527
    临时表和 525
    *另请参见*数据库对象；存储过程 441
    命名 504
    嵌套的 522–524
    嵌套的，和 **rollback trigger** 521
    权限和 525, 529
    删除 529
    视图和 313, 314, 525
    事务和 520, 589–595
    在 *image* 列上 526
    在 *text* 列上 526
    系统表和 526, 529–532
    限制 525
    性能和 527
    有关帮助 530
    约束 504
    摘要值和 516–518
    自递归 523
    组件集成服务 526
触发器表 504, 506
    删除 529
除法运算符 (/) 13, 45–48
处理游标 534
创建
    表 241–250
    触发器 503–505
    存储过程 457–465
    规则 394
    临时表 213, 220–221
    缺省值 389, 392
    数据库 207–210
    数据类型 197–199
    索引 373–379
磁盘崩溃
    *参见*恢复
存储过程 20, 441–444
    rollback 597
    with recompile 453
    作为安全性机制 444, 467
    编译 442
    参数 445, 451
    创建 457–465
    存储 443
    定时执行 426
    对象所有者的名字于 466
    返回参数 460–464
    返回状态 457–459
    访问权限对于 269
    分组 452
    隔离级别 594

检查其中的角色 460
结果显示 443
局部变量和 429
控制流语言 407–428
临时表和 222, 465
*另请参见*系统过程；触发器 441
命名 11–12
模式 594
嵌套 453
权限 444, 467
缺省参数 448–451
删除 467
事务和 589–595
信息 475
依赖性 476–477
游标内部 553
源文本 456
重命名 466
错误
convert 函数 363–367
包 438
标度 367
捕获数学 354
触发器和 522
除零 366
返回状态值 458–464
批处理中 405–406
算术溢出 366
在用户定义的事务中 593
域 367
重复键 593
错误处理 22
错误消息
12205 601
12207 589, 600
582 560
592 560
lock table 命令 589, 600
set lock wait 601
编号 424
严重级 424
用户定义的事务和 597
约束和 233

D

大小
列 334
数据库 208, 210–211
大于
*参见*比较运算符
打开游标 536
代码
soundex 341
带引号的标识符 9
当前日期 358
当前数据库
查找号 335
查找名称 335
改变 207
当前用户
suser_id 系统函数 336
suser_name 系统函数 336
user 系统函数 336
登录
帐号信息 30
登录过程 33
等于
*参见*比较运算符
等值连接 118, 120
笛卡儿乘积 119
定位游标 534
定义局部变量 429–433
动态链接库
Windows NT makefile 494
UNIX makefile 492
建立 491–494
搜索顺序 492
样本定义文件 494
用于扩展存储过程 491–494
动态转储 598
逗号 (,)
货币值的缺省输出格式 187
读取游标 536
段，放置对象于 214, 374, 379, 382
对象
*参见*数据库对象；*个别对象名*
对象标识符中的欧洲字符 9
多表视图 130, 317, 327
delete 和 130, 317

insert 和 130, 317
多个 SQL 语句
*参见*批处理
多列索引
*参见*组合索引
多字节字符集
数据类型 188–189
转换 363

E

二进制表达式 xxii
并置 15, 346–347
二进制数据类型 192–193
“0x”前缀 285
并置 338
操作 338–348
转换 367

F

返回参数 460–464
返回状态 457–459
范围查询 56, 373
使用 < 和 > 56
反斜杠 (\)
用于字符串延续 18
非共享临时表 221
非聚簇索引 378–379
完整性约束 235
非空值 216
非链式事务模式 578
非重复读取 581
分布页 307
分隔标识符 9
分支 421
分组
*另请参见*用户定义的事务
同名过程 452
浮点数据 xxii, 286
另请参见 “float 数据类型”；“*real* 数据类型”
符号
money 285
SQL 语句中 xxi
另请参见“通配符”；*此索引的 “符号” 部分*
比较运算符 16, 54
匹配字符串 60
算术运算符 13, 44
逐位运算符 44
服务器
远程登录的权限 31
执行远程过程 31
服务器用户名和 ID
suser_id 函数 336
suser_name 函数 336
服务器游标 539
复制
表，用 **bcp** 296
使用 **select into** 复制表 244
数据，用 **insert...select** 298
行 299

G

改变
*另请参见*更新
表 250–268
对象名 267–268
缺省数据库 33
视图定义 322
数据库大小 210–211
索引名称 267
隔离级别 26, 578, 580
readpast 选项和 604
定义的 580
更改查询的 584
级别 0 读取 375
事务 578–583
游标锁定 586
格式字符串
print 423
更改数据
*参见*数据修改
更新
image 数据类型 300
text 数据类型 300
分区的统计信息 385
*另请参见*更新；数据修改 299
在浏览模式下 336, 557

使用连接操作 303
索引统计信息 384
外键 515–516
游标 536
游标行 548
在浏览模式下防止 336
主键 511–515
公用键
*另请参见*外键、连接、主键
固定长度列
二进制数据类型用于 192
空值位于 216
与可变长度相比 189
字符数据类型用于 188
关闭游标 537
关键字 6–7
短语 2
控制流 407–428
新的 28
管理指令和结果 2
关系表达式 18
*另请参见*比较运算符
关系操作 3
关系模型，连接和 115
关系运算符 118
规范化 115, 190
规则 22, 290, 394
*另请参见*数据库对象
绑定 395–397
标识符 6
测试 388, 394
触发器和 502
创建新的 394
解除绑定 397
空值和 215
列定义冲突 69
命名用户创建的 394
批处理 402
删除用户定义的 398
和视图 313, 314
数据类型和 199
用户事务中的冲突 593
用临时约束 396
优先顺序 396
源文本 397
指定规则用 395
过程
*参见*远程过程调用 20
过程调用，远程 31

H

函数 333–369
image 348–351
text 348–351
安全性 369–370
集合 351–352
日期 355–360
和视图 316
数学 352–354
系统 333–338
转换 360–369
字符串 338–348
后缀名，临时表 222
弧度，转换为度 353
恢复 597–598
备份数据库 245
临时表和 221
日志放置和 209
时间和事务大小 597
事务和 570
样本数据库 35
混合数据类型，算术运算 14, 195–197
货币符号 285
货币值中的负号 (-) 285

J

基表
*参见*表
级别
@@trancount 全局变量 435, 576
嵌套事务 576
事务隔离 578–583
集合函数 73–78, 351–352
compute 子句和 20, 99–106
distinct 关键字和 74, 76–77
group by 子句和 78–93
order by 子句和 97
where 子句，不允许的 74

*另请参见*行集合；*各个函数名*
标量集合 75
在多列上 105
空值和 77–78
嵌套 86
矢量集合 79
视图和 324
数据类型和 75
游标和 542
子查询包括 163
集合理论运算 176–177
级联更改 （触发器）502, 509
记录，表
*参见*行；表
计时
@@error 状态检查 434
计算
*参见*计算列
计算列 44–48, 326
update 301
插入 298
带有空值 45
和视图 316, 325
计算日期 359
基准日期 284, 355
加法运算符 (+) 13, 45–48
加号 (+)
空值和 70
算术运算符 13, 44–48
字符串并置运算符 15, 346
加密
数据 6
检查约束 232, 239
减法运算符 (-) 13, 45–48
减号 (-)
减法运算符 13
监控
系统活动 438
兼容性，数据
create default 和 389
检索
空值 66
检索数据
*参见*查询
键值 378
键，表
*另请参见*公用键、外键、主键
视图和 313
降序 （**desc** 关键字）96
较低级和较高级数据类型
*参见*优先级
交互式 SQL 407–428
交集 （集合运算）176–177
角色
show_role 安全性函数 370
存储过程和 460
角色，用户定义的 269
角，数学函数 352
解除绑定
规则 397
缺省值 392
截断
str 转换和 343
二进制数据类型 285
临时表名 222
字符串 27
结构化查询语言 (SQL) 1
结果
游标结果集 533
进程 （服务器任务）
受影响的 426
近似数值数据类型 186
精度，数据类型
精确数值类型 286
井号 (#) 临时表名前缀 213, 220, 222
局部变量 421–433
屏幕显示 422–424
输出值 430
最后一行赋值 430
聚簇索引 378–379
used_pgs 系统函数和 336
*另请参见*索引
所用的总页数 336
完整性约束 235
句点 (.)
限定符名称的分隔符 10

K

可变长度列
空值位于 216

与固定长度相比 189
可更新游标 542
可共享临时表 220
客户端
游标 539
主计算机名 335
可用页， **curunreservedpgs** 系统函数 334
空白
在比较中 16
空字符串求值为 17
删除 **rtrim** 函数的尾随 340
用 **ltrim** 函数删除前导 340
字符数据类型和 189
空格
like 和 64
在比较时 55
在比较中 64
空格，字符
空字符串 （“”）或 （‘’）作为 17, 291
空间
估计表和索引大小 274
数据库存储 210–211
用于索引页 274
用 **truncate table** 释放 307
空值 65–69, 215
case 表达式和 412, 417–418
create procedure 和 450
distinct 关键字和 52
group by 和 86–87
insert 和 292, 327
比较 67, 433
变量和 430, 431, 433
参数缺省值为 451
插入替代值 291
触发器和 526–527
定义 216
规则和 215, 244
集合函数和 77–78
在计算列中 45
检查约束和 240
空缺省值和 216
连接和 68, 151
内部函数和 336
排序顺序 96, 97
缺省参数为 66, 450
缺省值和 244, 392
使用 **nullif** 比较 417
所允许的数据类型 199
新规则和列定义 69
选择 68–69
约束和 215
在 IDENTITY 列中不允许 217
控制流语言 20, 22, 407–428
控制中断报告 99
空字符串 （“”）或 （‘’）347
不求值为空 188, 291
作为单个空格 17
口令 33
更改 30
框图
pubs2 数据库 634
pubs3 数据库 664
扩展存储过程 21, 479–499
DLL 支持 480
Open Server 支持 481
创建 495, 496
创建函数用于 484–494
函数示例 486
例外 499
命名 481
权限 483
删除 496
释放内存 483
示例 481
消息来自 499
性能影响 483
优先级 483
执行 497
重命名 497
扩展， Transact-SQL 7, 23–24

L

连接 4
from 子句 116
theta 118
where 子句 117–118, 120, 121
比较运算符 122
不等于 122
不均等 124–127
处理 119

等值连接 118, 120
多个表 127–128
更新使用 303
关系模型和 115
关系运算符和 118
结果中的列顺序 115
进程 114
空值和 67, 151
连接表 131
列名 119
内部连接 （ANSI 语法）131, 133–136
视图和 316
事务和 597
索引和 373
外部 118, 129–151
外部连接 （ANSI 语法）136–147
限制 119
相关名和 123
相关名 （ANSI 语法）132–133
选择标准用于 121
选择列表于 115–116
用于视图 130, 317
右连接 129
有助于 152
运算符 118, 122
子查询比较 126, 171–172
自连接 123–124
自连接与子查询的比较 157
自然 120
组件集成服务 113
左连接 129
连接表 131
链式事务模式 26, 578
连字符作为注释 428
列
group by 和 85
IDENTITY 217–220
IDENTITY 值间隔 230–231
insert 语句中的顺序 297
select 语句中的顺序 41
*另请参见*数据库对象； **select** 命令
长度 334
长度定义 217
初始化文本 304
访问权限对于 269
规则 394
规则与定义冲突 217, 396
可变长度 216
空值和检查约束 240
空值和缺省值 216
连接和 115, 119
缺省值 234
缺省值用于 390–391
删除具有用户定义数据类型的列 264
索引多列 374
系统生成 217
修改具有用户定义数据类型的列 264
用 **alter table** 删除 254–256
用 **alter table** 添加 252–254
用 **alter table** 修改 256–259
用 **insert** 添加数据 289, 297–299
在 **insert** 语句中的顺序 289
在子查询中限定名称 156
列表
数据库对象 275
数据类型 （按照类型）195–196
列表，与 **select** 匹配 57–59
列长度
用 **alter table** 减少 257
列出了
dbcc 存储过程 475
列的大小，按数据类型 184
列对
*参见*公用键；连接
*参见*连接；键
列级约束 233
列名
查找 334
在子查询中限定 156
列名称 10
视图中的更改 314
临时表 8, 53
create table 和 213, 220–221
select into 和 223, 244–246
不允许视图 221, 313
触发器和 221, 525
创建 220–221
存储过程和 465
*另请参见*表； *tempdb* 数据库 245
命名 8, 213, 222
零
使用 NULL 或 291

浏览模式 557–559
timestamp 数据类型和 194, 336
游标声明和 558
逻辑表达式 xxii
case 表达式和 409
if...else 408
语法 18
真值表 19
逻辑运算符 70–72
having 子句 92

M

美元符号 ($)
标识符中 8
在 money 数据类型中 285
名称
db_name 函数 335
index_col 和索引 335
object_name 函数 335
suser_name 函数 336
user_name 函数 336
user 系统函数 336
表的别名 53
日期分量 355
事务的 574
主计算机 335
命令 2
另请参见各命令名
用户定义事务中不允许的 571
命令终结符 34
命名
*另请参见*重命名
保存点 573
表 213, 267–268
标签 421
触发器 504
存储过程 11–12
规则 394
过程中的参数 445–446
局部变量 429
列 10
临时表 8, 213, 222
视图 11–12
事务 568, 572
数据库 208
数据库对象 8–11
索引 10
约定 6–11
明细表 505
模式 238
模式匹配 344
charindex 字符串函数 342
difference 字符串函数 344
patindex 字符串函数 342
模运算符 (**%**) 13, 45–48

N

内部表
谓词限制 （ANSI 语法）140–141
内部查询
*参见*子查询
内部函数 333–369
image 348–351
text 348–351
安全性 369–370
集合 351–352
类型转换 360–365
日期 355–360
视图和 316
数学 352–354
系统 333–338
转换 360
字符串 338–348
内部结构
页用于 335
内部连接 133–136
on 子句 （ANSI 语法）135–136
连接表 （ANSI 语法）134
嵌套 （ANSI 语法）135

P

排序顺序
order by 和 96
另请参见 **order by** 子句
比较运算符和 16
批处理 401

go 命令 406
错误于 405–406
规则用于 402–405
局部变量和 421
控制流语言 20, 401–402, 407–428
作为文件提交 406–407
批处理中的 **drop** 命令 403
匹配
行 （*= 或 =*），外部连接 129
偏移位置， **readtext** 命令 350
屏幕消息 422–425

Q

前端应用程序，浏览模式和 557
嵌入连接操作 114
嵌套
begin transaction/commit 语句 576
begin...end 块 418
group by 子句 81–86
if...else 条件 409
while 循环 421
*另请参见*连接
触发器 453, 522–524
存储过程 453
级别 453
集合函数 86
排序 96
矢量集合 86
事务 576, 589
事务的警告 574
注释 427
子查询 158
字符串函数 339, 347
组 81
嵌套查询
*参见*嵌套 153
前写式日志 598
区分大小写 8
在比较表达式中 16
全局变量 434–440, 465
另请参见单个变量名
全名 29
权限 33, 205, 207
create procedure 445
readtext 和列 217
writetext 和列 217
参照完整性 238
触发器和 525, 529
存储过程 444, 467
分配 269
视图 315, 329
数据库对象所有者 308
数据修改 278
系统过程 468–469
缺省设置
存储过程的参数 448–451
货币值的输出格式 187
日期显示格式 188
数据库 33
语言 28, 282, 284
缺省数据库 28
缺省数据库设备 208
缺省语言 28, 282, 284
缺省值 22, 388
insert 语句和 289
*另请参见*数据库对象
绑定 390–391
创建 389, 392
解除绑定 392
空值和 244, 392
列 291
命名 389
删除 393
数据类型 198
数据类型标度 186
数据类型长度 188, 192
数据类型和 199, 390–391
数据类型精度 186

R

日期
like 和 284
另请参见“时间值”
比较 16, 55
存储 355
当前的 358
函数 355–360
计算 359

可接受的范围 281
搜索 65, 284
添加日期分量 359
条目格式 187, 281–285, 355
条目规则 65
显示格式 188, 355
日期分量 282, 355–358
缩写名称和值 355
日期函数 355–360
日文字符集 188
对象标识符和 9
日元符号 （￥）
在 money 数据类型中 285
标识符中 8
日志
*参见*事务日志

S

三角函数 353
删除
另请参见 **delete** 命令；*各* **drop** *命令*
表 268, 324
表中的行 305–307
*参见*删除
触发器 529
对象 403
规则 398
过程 467
缺省值 393
视图 324, 329
数据库 211
索引 382
系统表 268
行 305–307
游标 536
游标行 549
主键 509–510
设备 208
另请参见 sysdevices 表
设计表 241
舍入
datetime 值 188
str 字符串函数和 343
货币值 187
近似数值数据类型 186
转换货币值 363
声明
参数 445–446
局部变量 429–433
游标 535, 537–544
升序， **asc** 关键字 96
释放游标 537, 550
实际值
空 69
时间间隔，执行 426
时间值
like 和 284
另请参见“日期”；“*日期*数据类型”；“*smalldatetime* 数据类型”
存储 355
函数 355–357
搜索 284
条目格式 281–282
显示格式 355
矢量集合 79
嵌套 86
十六进制数字
“0x”前缀用于 49, 285
转换 365
视图 309, 312
distinct 的投影 318
distinct 和 314
from 子句中所允许的 52
from 子句中允许的 116
IDENTITY 列和 328
insert 和 319–320
readtext 和 321
references 332
with check option 130, 317, 319–320, 324, 327
union 和 112
update 和 319–320
*另请参见*数据库对象
安全性和 311
不允许更新 325
查询 321
触发器和 221, 313, 314, 525
创建 312
带有连接 130, 317
访问权限对于 269
规则和 313, 314

函数和 316
集合函数和 324
计算列和 325
键和 313
检索数据 321
解析 321, 322
连接和 113, 316
列名称 314
临时表和 221, 313, 314
命名 10–11
权限 315, 329
缺省值和 313, 314
删除 329
数据类型和 314
数据修改和 324
索引和 313, 314
投影 315
和 **writetext** 321
限制 324–328
相关 322
优点 310
有关帮助 330
源文本 319
重定义 322
重命名 323
重命名列 314
事务 279, 567–598
@@transtate 全局变量 574
不在嵌套中使用的名称 574
触发器和 520, 573
存储过程和 573
存储过程和触发器 589
定时执行 426
符合 SQL 标准 587
隔离级别 26, 578
恢复和 570, 598
命名 572
模式 26, 578
嵌套级别 576, 589
取消 597
锁定 569
性能和 570
游标和 595
允许的数据库数目 597
状态 574
组件集成服务 570
事务隔离级别
readpast 选项和 604
事务日志 597
writetext 和 304
大小 209
在单独的设备上 209
事务中的幻像 581
受影响的进程 426
数据定义 470
数据复制 261–262
alter table 何时执行数据复制 262
数据库 206–211
ID 号， **db_id** 函数 335
use 命令 207
*另请参见*数据库对象
创建用户 207–210
大小 208, 210–211
服务器数目 208
可选的 206
连接和设计 115
名称 208
缺省 28
删除 211
所有权 207
添加用户 205
系统 206
选择 207
用户 206
有关帮助 270
数据库对象 203
alter table 的限制 251
ID 号 (**object_id**) 335
另请参见个别对象名
存储过程和 465, 466
访问权限 269
命名 8–11
删除 403
系统过程和 470
重命名 267–268
数据库对象所有者
存储过程中的名字 466
数据库设备 208
数据库所有者
添加用户 205
转交所有权 207
数据库完整性

*参见*数据完整性；参照完整性
数据类型 183–195
create table 和 212, 215, 375
datetime 值比较 16
money 187
unichar 189
union 107
变更具有用户定义属性的列 263–264
层次 195–196
长度 198
创建 197–200
规则和 199, 395–397
混合，算术运算 14
集合函数和 75
近似值 186
局部变量和 429
连接和 119
临时表和 221
缺省值和 199, 390–391
删除具有用户定义属性的列 264
视图和 314
输入规则 186, 280–287
修改具有用户定义属性的列 264
用 **alter table** 转换 257
摘要 184–185
整数 185
字符 187
数据类型优先级
*参见*优先级
数据类型转换 360–365
case 表达式 412
image 368
标度错误 367
二进制和数值数据 365
货币信息 363
列定义和 217
日期和时间信息 364
舍入 363
十六进制式信息 367
数字信息 363, 364
位信息 368
溢出错误 366
域错误 367
自动 194
字符信息 363
数据类型，自定义
参见“用户定义的数据类型”
数据完整性 21, 204, 394
方法 232
*另请参见*数据修改；参照完整性 2
事务和 591
唯一索引 375
约束 232
数据修改 2, 310
更新 299–304
权限 278
视图和 324
使用 **writetext** 的 *text* 和 *image* 304–305
数据依赖性
*参见*依赖性；数据库对象
数据字典
*参见*系统表
数目 （数量）
rowcnt 报告的行 335
查询中允许的表 52, 116
服务器数据库 208
事务中的数据库 597
输入包，数目 438
数学函数
列出 352–353
示例 354
语法 352
数值数据
并置 347
运算 354
数字
数据库 ID 335
星号 (**) 表示超长的部分 343
数字表达式 xxii
双精度数据类型
条目格式 286
双引号 （“”）
比较运算符和 16
表达式中 17
将参数值引起来 448
将列标题括起来 43
将值引起来 188, 280
对于空字符串 17, 291
文字说明 17
顺序
空值 97
*另请参见*索引；优先级；排序顺序 17

执行表达式中的运算符 13
搜索条件 53
速度 （服务器）
恢复 597
算术表达式 13, 42
不能同 **distinct** 一起使用 76
运算符优先级 47
算术错误 27, 366
算术运算 74
混合模式 195–197
算术运算符 44–47
作为比较运算符 55
表达式中 13
在表达式中 42
优先级 47–48, 71
锁超时
set lock wait 命令 600
锁定
事务和 569
游标和 555
缩写
out 表示 **output** 464
日期分量 355
索引
创建 373–379
在多列 374
非聚簇 378–379
非唯一索引中的 IDENTITY 列 375
分布页 307
检索速度和 372, 373, 378
键值 378
连接和 373
*另请参见*聚簇索引；数据库对象 378
命名 10
删除 382
视图和 313, 314
使用空间 274
搜索 373
完整性约束 235
唯一 375, 380
选项 380–382
叶级 378, 379
关于预排序数据 381–382
指南 372–373
重复值和 380
重命名 267
根据主键 378
主键 373
组合 374
索引的叶级 378, 379
索引页
表的总数和 335
分配 335
系统函数 335

T

特权
*参见*权限
特殊字符 7
填补，数据
空值和 217
临时表名中的下划线 222
添加
timestamp 列 558
表的 IDENTITY 列 260
对于列数据用 **insert** 289
列数据，用 **insert** 297–299
外键 508
用户到数据库 205
用户定义的数据类型 197–198
在表或视图中添加行 287–299
调试辅助程序 22
通配符
缺省参数使用 450
搜索 62
在 **like** 匹配字符串中 60–65
同义词
out 表示 **output** 464
关键字 28
数据类型 184
投影
distinct 视图 318
另请参见 **select** 命令
查询 4
视图 315
图像函数 348
退出
isql 34

W

外部表
    谓词限制 （ANSI 语法）139–140
外部连接 129–151
    ANSI 语法 137–138
    on 子句 （ANSI 语法）139–142
    where 子句的限制条件 （ANSI 语法）141–142
    where 子句 （ANSI 语法）139–142
    *另请参见*连接
    内部表的角色 （ANSI 语法）136
    嵌套外部连接中的 on 子句 （ANSI 语法）143–144
    嵌套外部连接中 **on** 子句的位置（ANSI 语法）143–144
    嵌套外部连接 （ANSI 语法）142–144
    谓词位置 （ANSI 语法）139–142
    限制 130
    用连接顺序依赖性转换外部连接 （ANSI 语法）145–147
    运算符 118, 148–151
    在嵌套外部连接中的小括号 （ANSI 语法）143
    在 Transact-SQL 外部连接中如何对谓词求值 145
外键 505–506
    sp_help 报告 272
    插入 508
    更新 515–516
完整性。*参见* **dbcc** （数据库一致性检查程序）278
谓词
    在外部连接中的位置 （ANSI 语法）139–142
唯一索引 375, 380
唯一约束 232, 235
未知值
    *参见*空值
文本
    用反斜杠 (\) 使行延续 18
文本函数 348–351
文本指针值 304
文件
    批处理 406–407
文字字符说明
    双引号 （“”）17

X

系统表 205, 312
    *另请参见*表；*各表名*
    触发器和 526, 529–532
    删除 268
    系统过程和 467
系统管理员
    数据库所有权 207
    用户 ID 334
系统过程 20, 467–468
    安全性管理 469
    不允许用作参数的分隔标识符 9
    查看文本 476
    用于登录名管理 469
    隔离级别 587
    在临时表上 223
    数据定义 470
    系统管理 473
    用户定义事务中不允许的 572
    重新优化查询用 444
    *另请参见*存储过程；*个别过程名*
系统函数
    示例 333, 337–338
    语法 333, 336
系统扩展存储过程 498
系统数据类型
    参见“数据类型”
下划线 (_)
    临时表名中 222
限定
    表名 213
    存储过程中的对象名 466
    连接中的列名 115
    数据库对象 10
    子查询中的列名 156
显式空值 290
显式事务 580
限制 3
    另请参见 **select** 命令
相关表 509
相关名
    表名 53, 123
    在连接中 （ANSI 语法）132–133
    子查询使用 157, 180
    自连接 123
相关视图 322
相关子查询 178–182
    exists 和 174
    having 子句 182

比较运算符 180
相关名和 180
小括号 ()
另请参见此索引的 “符号” 部分
表达式中 12
在算术语句中 47
消息 422–425
事务和 597
小于
*参见*比较运算符
斜杠 (/)
除法运算符 13, 45
斜线星号 (/*) 注释关键字 427
信息消息 （服务器）
*参见*错误消息；严重级
星号 (*)
select 和 40
超长数字 343
乘法运算符 13, 44–48
在带 **exists** 的子查询中 174
括住注释的符号对 427
行集合 99
compute 和 20, 101
group by 子句和 103
视图和 324
与集合函数相比较 101
性能
变量赋值和 429, 430
触发器和 527
存储过程和 444
日志放置和 209
事务和 570
索引和 373
行 （文本），输入长型数据 18
行，表 2
*另请参见*触发器
复制 299
改变 299–304
删除 305–307
数目 335
添加 287–299
唯一 381
选择 37, 53
摘要 99–101
重复 108, 380–381
修改
表 250
数据库 210
修改数据
参见 数据修改
选择
参见 **select** 命令
选择列表 50, 115–116
union 语句 107, 109
子查询使用 174
循环
break 和 419
continue 和 419
while 418–421

Y

延迟执行 (**waitfor**) 426–427
延续行，字符串 18
严重级，错误
用户定义消息 424
样本数据库。*参见 pubs2* 数据库 35
页数
reserved_pgs 函数 335
used_pgs 函数 336
表和聚簇索引所用的 （总的）336
表或索引所用的 335
分配给表或索引 335
页，数据
data_pgs 系统函数 335
reserved_pgs 系统函数 335
used_pgs 系统函数 336
分配 335
用于内部结构 335
在表或索引中使用 335, 336
以 10 为底的对数函数 353
溢出错误
set arithabort 和 366
数据和时间转换 364
算术 366
依赖性，数据库对象 476–477
依赖性，显示 332
一致性
事务和 569
隐式事务 578
隐式转换 （数据类型的）17, 194, 360

引用 476–477
英镑符号 （£）
在 money 数据类型中 285
标识符中 8
用户
管理 469
添加 205
用户定义的过程 441–477
用户定义的事务
*参见*事务
用户定义的数据类型 197–198
IDENTITY 列和 199, 218
sysname 作为 194
timestamp 作为 194
变更列 263
规则和 197–199, 395–397
临时表和 223
缺省值和 216, 390–391
删除列 264
修改列 264
用户定义的数据类型的 IDENTITY 属性 199
用户定义角色 269
用户 ID 334
valid_user 函数 336
user_id 函数 336
用户名
查找 336
定义的 336
用户数据库 206
游标 533–557
for browse 和 558
Halloween 问题 544
server 539
变量 545
存储过程和 553
错误消息 592 560
错误信息 582 560
打开 544
定位 533
定位型更新和 559
定位型删除和 559
读取 544–547
读取的行数 547
读取多个行 546
范围 539, 539–540
非唯一索引 541
更新行 548
关闭 550
将客户端行放入缓冲区 547
可更新 542
客户端 539
连接列更新 560–565
名称冲突 540
扫描 541
删除行 549
声明 537–544
释放 550
事务和 595–596
搜索型更新和 559
搜索型删除和 559
锁定和 555
唯一索引 541
语言 539
只读 542
执行 539
状态 546
子查询和 155
游标范围 539
游标结果集 533, 541
优先级 17
表达式中的运算符 13
较低级和较高级数据类型 17
语法
Transact-SQL 6–12
语句 2, 7
语句块的定时执行 426
语句块 （**begin...end**）418
语言游标 539
语言，替代
对日期分量的影响 282, 284, 357
阈值
最后机会 335
与 （&）
逐位运算符 14, 44
远程服务器 12, 31, 457, 469
execute 和 442
非 Sybase 24
系统过程影响 469
组件集成服务 24
远程过程调用 12, 31, 457
用户定义的事务 593, 597
语法用于 442

源文本 4
加密 6
源值
rand 函数 353
set identity_insert 和 293
约定
Transact-SQL 6–12
*另请参见*语法
命名 6–12
在手册中使用 xx
月份值
日期分量缩写和 356
约束 204, 232
check 232, 239
default 232
primary key 232, 235
表级 233
参照完整性 232, 236
列级 233
唯一 232, 235
用 **alter table** 删除 255–256
用 **alter table** 添加 254
用规则 396
有关 NULL 值 215
运算符
比较 16, 54–56
关系 118
连接 118
逻辑 70–72
算术 13, 44–47
优先级 13, 47, 71–72
逐位 14–15, 44

Z

在大多数情况下，使用向导是最简单的方法，但是对于“数字”和“货币”字段，您必须手工键入
输入掩码的定义
用户 ID 1 334
在系统函数中
小括号（()）336
脏读 580
*另请参见*隔离级别
摘要行 99–101
摘要值 20, 73
触发器和 516–518
集合函数和 99
占位符
print 消息 423
帐号，服务器
*参见*登录 33
真值表
逻辑表达式 19
整数数据 185
另请参见各个数据类型名
在 SQL 中 xxii
转换 365
整数余数
*参见*模运算符 (**%**)
值
IDENTITY 列 217
空 215
只读游标 538, 542
指数值 353
执行
扩展存储过程 482
执行游标 539
指针
text 或 *image* 列 304
主表 505
主机进程 ID，客户端进程 335
主计算机名 335
主键 272, 505–506
参照完整性和 505, 509
更新 511–515
删除 509–510
索引 373, 378
约束 235
主 — 明细关系 505
注释
ANSI 样式 26
SQL 语句中 427–428
双连字符样式 428
注释文本
*参见*注释
逐位运算的数据二进制表示形式 14
逐位运算符 14–15, 44
转储，数据库 598
转换
大写到小写 340
度到弧度 353

弧度到度 353
较低级到较高级数据类型 17
数据类型 217
小写到大写 341
隐式 17, 194, 360
整数值到字符值 340
整数自变量到二进制数字 14
字符串并置 15
字符集之间 9
转义字符 63
自变量
日期函数 355
数学函数 352
文本函数 348
系统函数 334–336
字符串函数 339
子查询 153
all 关键字和 162, 166, 174
any 关键字和 18, 162, 167, 174
delete 语句使用 159
exists 关键字 173–177
group by 子句 164, 181
having 子句 164, 182
in 关键字和 58, 168, 170–172, 174
insert 语句使用 159
not exists 关键字和 176–177
not in 关键字和 172–173
order by 和 97
select 列表 174
where 子句和 155, 172, 173
update 语句使用 159
*另请参见*连接
比较运算符 162–167, 174
表达式中 18
表达式，替代 160
不均等连接和 126–127
不允许的数据类型 155
处理结果 155
集合函数和 163, 164
空值和 154
类型 161
连接 126
连接比较 171–172
列名 156
嵌套 158
无修饰词的比较运算符和 162–164
限制 154
相关或重复 178–182
相关名 157, 180
相关子查询中的比较运算符 180–182
有修饰词的比较运算符和 165, 174
语法 154–161
重复 178–182
自定义数据类型
参见“用户定义的数据类型”
自动操作
触发器 501
链式事务模式 578
数据类型转换 194, 360
隐藏的 IDENTITY 列 220
字段，数据
*参见*列；表
字符
数目 340
特殊 7
通配 450
通配符 60–65
字符表达式 xxii, 13
字符串 65
并置 15, 338, 339, 346–347
截断 27, 280
空 17, 188, 347
匹配 60
匹配用 **like** 60–65
选择列表使用 44
用反斜杠 (\) 延续 18
指定引号 17
字符串函数 338–348
并置 346–347
测试相似 344
嵌套 339, 347
示例 342–346
字符集 6
iso_1 9
转换错误 9
字符列中的空字符串 290
字符数据 187
另请参见各个字符数据类型名
避免“NULL”290
操作 338–348
输入规则 280
搜索 65

尾随空白 189
转换 363
字符数据类型 188
like 和 60
从多字节向单字节转换 363
输入规则 280
转换数值为 363
字节
@@maxcharlen 限制 438
@@ncharsize 平均长度 438
@@textsixe 限制 437
print 消息限制 422
按 **readtext** 检索 49
表达式的长度 （**datalength** 函数）335
标识符限制 8
分隔标识符限制 9
临时表名的限制 8
十六进制数字和 285
输出字符串限制 424
数据类型存储 184, 212
引用的列限制 43
用于 *text* 和 *image* 数据 440
子查询限制 155
组合索引限制 375
子句 2
自连接 123–124
与子查询比较 157
自然连接 120
总和
compute 106–107
order by 96
*另请参见*集合函数
使用 **compute** 子句 99
最终值 （不带 **by** 的 **compute**）106
组
成员资格 29
数据库用户 29
自由选择访问控制和 29
组合
过程 593
组合索引 374
组件集成服务
image 数据类型 193
text 数据类型 192
触发器 526
和 **update statistics** 命令 384
连接 113
连接到服务器 33
描述的 24
事务和 570
远程表的 **alter table** 命令 252
自动 IDENTITY 列 220
最后机会阈值 335
作者 *blurbs* 表
pubs2 数据库 629
pubs3 数据库 662