Sybase® Adaptive Server™ Enterprise
参考手册

第一卷：构件块

Adaptive Server Enterprise 版本 12
文档 ID：37419-01-1200-01
最后修订日期：1999 年 10 月

主要作者：Enterprise Data Studios Publications

参与作者：Anneli Meyer、 Evelyn Wheeler

文档 ID：37419-01-1200-01

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

文档订购

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

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

升级产品仅在软件的定期发布日提供。

Sybase 商标

有限权利

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

目录

关于本手册

1. 系统和用户定义的数据类型

2. Transact-SQL 函数

3. 表达式、标识符和通配符

4. 保留字

5. SQLSTATE 代码和消息

如需索引，参见第四卷 “表格和参考手册索引”。

表目录

关于本手册

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

第一卷 “*构件块*”介绍 Transact-SQL 的 “构件”：数据类型、内部函数、表达式和标识符、保留字，以及 SQLSTATE 错误。要能够成功地使用 Transact-SQL，您需要首先理解这些构件块的功能，以及它们对 Transact-SQL 语句结果的影响。

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

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

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

读者

*Adaptive Server Enterprise 参考手册*是为各种水平的 Transact-SQL 用户提供的参考工具。

如何使用本手册

本手册包括：

- 第 1 章 “系统和用户定义的数据类型”。该章介绍随 Adaptive Server 提供的系统数据类型和用户定义数据类型，并指出如何使用它们来创建用户定义的数据类型。
- 第 2 章 “Transact-SQL 函数”。该章提供有关 Adaptive Server 集合函数、数据类型转换函数、日期函数、数学函数、行集合函数、字符串函数、系统函数以及文字和图象函数的参考信息。
- 第 3 章 “表达式、标识符和通配符”。该章提供有关如何使用 Transact-SQL 语言的信息。
- 第 4 章 “保留字”。该章提供有关 Transact-SQL 和 SQL92 关键字的信息。

- 第 5 章 “SQLSTATE 代码和消息”。该章包含有关 Adaptive Server 的 SQLSTATE 状态码及相关消息的信息。

Adaptive Server Enterprise 文档

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

- 特定平台的*发行公告* — 包含未能包含在本手册中的最新信息。

  *发行公告*的较新版本有可能在 WWW 上获得。要了解在本产品 CD 发行之后新增的重要产品或文档信息，可使用 Web 上的 SyBooks™。

- 特定平台的 Adaptive Server 安装文档 — 说明所有 Adaptive Server 及其相关的 Sybase 产品的安装、升级和配置过程。
- *Adaptive Server Enterprise 中的新功能* — 说明 Adaptive Server 版本 12 中的新功能、为支持这些功能增加的系统变化和可能影响已有应用程序的变化。
- *Transact-SQL User's Guide* — 描述了 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。
- *Adaptive Server Enterprise 参考手册* — 包含所有 Transact-SQL 的命令、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。
- *Performance and Tuning Guide* — 说明如何调优 Adaptive Server 以达到最佳性能。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、磁盘和缓存的问题，以及锁和游标对性能的影响。
- 所使用平台的*实用程序*手册 — 介绍在操作系统级执行的 Adaptive Server 实用程序，如 **isql** 和 **bcp**。
- *Error Messages* and *Troubleshooting Guide* — 说明如何解决常见错误消息和用户经常遇到的系统问题的解决方法。
- *Component Integration Services User's Guide* — 说明如何使用 Adaptive Server 组件集成服务功能连接远程 Sybase 数据库和非 Sybase 数据库。
- *Adaptive Server Enterprise 中的 Java* — 描述如何将 Java 类安装并用为 Adaptive Server 数据库中的数据类型和用户定义的函数。

- *Using Sybase Failover in a High Availability System* — 指导如何在高可用性系统中使用 Sybase 故障替换将 Adaptive Server 配置为协同服务器。
- *使用 Adaptive Server 分布式事务管理功能* — 说明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 的功能故障。
- *XA Interface Integration Guide for CICS, Encina, and TUXEDO* — 指导如何通过 X/Open XA 事务管理器使用 Sybase 的 DTM XA 接口。
- *Adaptive Server Glossary* — 定义在 Adaptive Server 文档中使用的技术术语。

其它信息来源

使用 Sybase Technical Library CD 和 Technical Library Web 站点来了解产品的更多信息：

- Technical Library CD 含有产品手册和技术文档，随软件提供。Technical Library CD 所包含的 DynaText 浏览器让您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 Web 站点包括 Product Manuals 站点，它是 Technical Library CD 的 HTML 版本，您可以使用标准 Web 浏览器访问。另外，您还可链接到 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 Web 站点，进入 support.sybase.com，单击 “Electronic Support Services” 选项卡，然后选择 “Technical Library” 标题下的链接。

本手册中使用的约定

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开始一个新行。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、实用程序选项和其它关键字为粗体。	**select** **sp_configure**
数据库名、数据类型、文件名和路径名为斜体。	*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*
输入小括号作为命令的一部分。	`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。不要输入中括号。	`[cash | check | credit]`
逗号表示可以选择任意多个所显示的选项。用逗号将您的选择隔开，使其成为命令的一部分。	`cash, check, credit`
竖线 (|) 表示只可选择一个所显示的选项。	`cash | check | credit`

省略号 (...) 表示可以将最后一个单元**重复**任意多次。	`buy thing = price [cash | check | credit]`
	`[, thing = price [cash | check | credit]]...`
	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：选择中括号中的一项。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 命令的示例显示如下：

```
select * from publishers
```

- 计算机输出的示例如下：

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

(3 rows affected)
```

大小写

本手册中，大多数示例是小写字体。然而，输入 Transact-SQL 关键字时可以忽略大小写。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，可改变单字节字符的大小写敏感。更多信息，请参看 *系统管理指南* 中第 19 章 “配置字符集、排序顺序和语言”中的 “改变缺省字符集、排序顺序或语言”。

表达式

Adaptive Server 语法语句使用几种不同的表达式类型。

表 2：在语法语句中使用的表达式类型

用法	定义
expression	可以包括常量、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，他们得到授权可与 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。

1 系统和用户定义的数据类型

本章介绍 Transact-SQL 数据类型。数据类型指定列、存储过程参数和本地变量的类型、大小和存储格式。涉及的主题包括：

数据类型种类

Adaptive Server 提供了几种系统数据类型和用户定义的数据类型 *timestamp* 和 *sysname*。表 1-1 列出了 Adaptive Server 数据类型的种类。每一种类都在本章中的某一节中进行了描述。

表 1-1：数据类型种类

种类	用于
精确数值数据类型	必须确切表示的数值 （整数和带有小数部分的数）
近似数值数据类型	在算术运算中允许舍入的数字数据

表 1-1：数据类型种类 （续）

种类	用于
money 数据类型	货币数据
时间戳数据类型	在 Client-Library™ 应用程序中浏览的表格
日期和时间数据类型	日期和时间信息
字符数据类型	含有字母、数字和符号的字符串
二进制数据类型	十六进制式符号的原始二进制数据，如图象
位数据类型	真 / 假和是 / 否类型的数据
sysname 数据类型	系统表
text 和 image 数据类型	所需存储空间大于 255 字节的可打印字符或十六进制式数据
用户定义的数据类型	定义继承了规则、缺省值、空类型、IDENTITY 属性和基本数据类型的对象

范围和存储大小

表 1-2 列出了系统提供的数据类型及其同义词，并提供了关于有效值的范围和每一数据类型的存储大小的信息。为简单起见，尽管 Adaptive Server 允许将大写或小写字符用于系统数据类型，但数据类型是用小写字符给出的。用户定义的数据类型 （如 *timestamp*）是**区分大小写的**。大多数由 Adaptive Server 提供的数据类型并不是保留字，并可以用于命名其它的对象。

表 1-2：系统数据类型的范围和存储大小

数据类型	同义词	范围	存储的字节数
精确数值数据类型			
tinyint		0 到 255	1
smallint		-2^{15} (-32,768) 到 2^{15} -1 (32,767)	2
int	*integer*	-2^{31} (-2,147,483,648) 到 2^{31} -1 (2,147,483,647)	4
numeric (p, s)		-10^{38} 到 10^{38} -1	2 到 17
decimal (p, s)	*dec*	-10^{38} 到 10^{38} -1	2 到 17

表 1-2：系统数据类型的范围和存储大小 （续）

数据类型	同义词	范围	存储的字节数
近似数值数据类型			
float (precision)		与计算机有关	4 或 8
double precision		与计算机有关	8
real		与计算机有关	4
money 数据类型			
smallmoney		-214,748.3648 到 214,748.3647	4
money		-922,337,203,685,477.5808 到 922,337,203,685,477.5807	8
日期/时间数据类型			
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
字符数据类型			
char(n)	*character*	小于等于 255 个字符	*n*
varchar(n)	*char[acter] varying*	小于等于 255 个字符	实际的条目长度
nchar(n)	*national char[acter]*	小于等于 255 个字符	*n* * @ @ncharsize*
nvarchar(n)	*nchar varying, national char[acter] varying*	小于等于 255 个字符	*n*
二进制数据类型			
binary(n)		小于等于 255 个字节	*n*
varbinary(n)		小于等于 255 个字节	实际的条目长度
位数据类型			
bit		0 或 1	1 （1 字节，最多保存 8 *位*列）
文本和图像数据类型			
text		小于或等于 2^{31} -1 (2,147,483,647) 字节	在初始化之前为 0，然后是 2K 的倍数
image		小于或等于 2^{31} -1 (2,147,483,647) 字节	在初始化之前为 0，然后是 2K 的倍数

声明列、变量或参数的数据类型

您必须为列、本地变量或参数声明数据类型。数据类型可以是系统提供的数据类型中的任意一种，或者是数据库中任意一种用户定义的数据类型。

为表中的列声明数据类型

使用下列语法在 **create table** 或 **alter table** 语句中声明新列的数据类型：

```
create table [[database.]owner.]table_name
   (column_name datatype [identity | not null | null]
       [, column_name datatype [identity | not null |
           null]]...)

alter table [[database.]owner.]table_name
   add column_name datatype [identity | null
       [, column_name datatype [identity | null]...
```

例如：

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

为批处理或过程中的局部变量声明数据类型

使用下列语法为批处理或存储过程中的局部变量声明数据类型：

```
declare @variable_name datatype
   [, @variable_name datatype]...
```

例如：

```
   declare @hope money
```

为存储过程中的参数声明数据类型

使用下列语法为存储过程中的参数声明数据类型：

```
create procedure [owner.]procedure_name [;number]
   [[(]@parameter_name datatype [= default] [output]
       [,@parameter_name datatype [= default]
           [output]]...[)]]
   [with recompile]
   as SQL_statements
```

例如：

```
create procedure auname_sp @auname varchar(40)
as
    select au_lname, title, au_ord
    from authors, titles, titleauthor
    where @auname = au_lname
    and authors.au_id = titleauthor.au_id
    and titles.title_id = titleauthor.title_id
```

确定文字的数据类型

您不能声明文字的数据类型。 Adaptive Server 将所有的字符文字视为 *varchar*。输入的带有 E 符号的数字文字被视为 *float* ； 其它所有数字文字都被视为确切的数字：

- 位于 $2^{31} - 1$ 和 -2^{31} 之间、不带小数点的文字被视为 *integer*。
- 含有小数点或位于整数范围之外的文字被视为 *numeric*。

➤ ***注意***

为了保持向后兼容，请将 E 符号用于应被视为 *float* 的数字文字。

混合型表达式的数据类型

当您对具有不同数据类型的值执行并置或混合算法时， Adaptive Server 必须确定结果的数据类型、长度和精度。

确定数据类型层次

每个系统数据类型都有**数据类型层次**，它保存在 *systypes* 系统表中。用户定义的数据类型继承了其所基于的系统数据类型的层次。

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
varchar                                19
sysname                                19
nvarchar                               19
char                                   20
nchar                                  20
varbinary                              21
timestamp                              21
binary                                 22
text                                   23
image                                  24
(28 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。

在下列示例中，来自于 *sales* 表的 *qty* 同来自于 *roysched* 表的 *royalty* 相乘。 *qty* 是 *smallint*，它的层次为 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

确定精度和标度

对于 *numeric* 和 *decimal* 数据类型而言，精度和标度的每一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：

- *n1* 具有精度 *p1* 和标度 *s1*，并且
- *n2* 具有精度 *p2* 和标度 *s2*

Adaptive Server 所确定的结果的精度和标度如表 1-3 所示：

表 1-3：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)

将一种数据类型转换为另一种数据类型

许多由一种数据类型到另一种数据类型的转换都是由 Adaptive Server 自动处理的。这些转换称作隐式转换。而其它的转换则必须用 **convert**、 **inttohex** 和 **hextoint** 函数来明确执行。有关 Adaptive Server 所支持的数据类型转换的详细信息，参见第 2 章 “Transact-SQL 函数”中的 “数据类型转换函数”。

固定长度的 NULL 列的自动转换

只有具有可变长度数据类型的列才能存储空值。当您创建具有固定长度数据类型的 NULL 列时， Adaptive Server 自动将其转换为相应的可变长度数据类型。 Adaptive Server 不会告知用户数据类型的更改。

表 1-4 列出了所转换为的固定和可变长度数据类型。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 1-4：固定长度数据类型的自动转换

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*

处理溢出和截断错误

arithabort 选项决定了在出现算术错误时，Adaptive Server 如何工作。这两个 **arithabort** 选项 （**arithabort arith_overflow** 和 **arithabort numeric_truncation**）处理不同类型的算术错误。您可以单独地设置每一个选项，或者用单个的 **set arithabort on** 或 **set arithabort off** 语句来设置这两个选项。

- 在显式或隐式数据类型转换的过程中， **arithabort arith_overflow** 指定出现除零错误或精度损失后的行为。这种类型的错误是很严重的。缺省设置为 **arithabort arith_overflow on**，它将回退发生错误的整个事务。如果不包含事务的批处理发生了这种错误，则 **arithabort arith_overflow on** 将不回退批处理中以前的命令，并且 Adaptive Server 也不会执行批处理中产生错误的语句之后的语句。

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

- **arithabort numeric_truncation** 指定了在隐式数据类型转换过程中，精确数值数据类型标度损失后的行为。（当显式转换导致标度损失时，将截断结果而不发出任何警告。）缺省设置为 **arithabort numeric_truncation on**，它将中止导致错误的语句，但会继续处理事务或批处理中的其它语句。如果设置为 **arithabort numeric_truncation off**， Adaptive Server 就会截断查询结果并继续进行处理。

arithignore 选项决定了在出现溢出错误之后， Adaptive Server 是否显示警告消息。缺省情况下， **arithignore** 选项被设置为 **off**。这样就可以使 Adaptive Server 在任何导致数字溢出的查询之后，显示警告消息。如果要忽略溢出错误，请使用 **set arithignore on**。

➤ ***注意***

arithabort 和 **arithignore** 选项为版本 10.0 进行了重新定义。如果您在应用程序中使用这些选项，请检查它们是否仍产生预期的结果。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 提供了 *smallint*、*int*、*numeric*、*decimal*、*float*、*double precision*、*real*、*char* 和 *varchar* SQL92 数据类型。*tinyint*、*binary*、*varbinary*、*image*、*bit*、*datetime*、*smalldatetime*、*money*、*smallmoney*、*nchar*、*nvarchar*、*sysname*、*text*、*timestamp* 和用户定义的数据类型是 Transact-SQL 扩展。

精确数值数据类型

函数

当精确地表示一个数值非常重要时，使用精确数值数据类型。Adaptive Server 为整数和具有小数部分的数值都提供精确数值类型。

整数类型

Adaptive Server 提供了三种精确数值数据类型来存储整数：*int*（或 *integer*）、*smallint* 和 *tinyint*。请基于要存储的数字的预期大小来选择整数类型。内部存储大小因类型而有所不同，如表 1-5 所示：

表 1-5：整数数据类型

数据类型	存储	存储的字节数
int[eger]	位于 -2^{31} 和 $2^{31} - 1$ 之间的整数（-2,147,483,648 和 2,147,483,647 之间）（包括这两个数值）。	4
smallint	位于 -2^{15} 和 $2^{15} - 1$ 之间的整数（-32,768 和 32,767 之间）（包括这两个数值）。	2
tinyint	位于 0 和 255（包括这两个数值）之间的整数。（不允许为负数。）	1

输入整数数据

将整数数据作为没有逗号的数字串来输入。只要小数点右侧的数字全部为 0，整数数据类型就可以包含小数点。*smallint* 和 *integer* 数据类型前面可以有任选的正号或负号。*tinyint* 数据类型前面可以有任选的正号。

表 1-6 显示了数据类型为 *integer* 的列的一些有效的条目，并且指出了 **isql** 显示这些数值的方式：

表 1-6：有效的整数数值

所输入的数值	所显示的数值
2	2
+2	2
-2	-2
2.	2
2.000	2

表 1-7 列出了 *integer* 列的一些无效的条目：

表 1-7：无效的整数值

所输入的数值	错误类型
2,000	不允许有逗号。
2-	负号应该位于数字之前。
3.45	小数点右边的数字是非零数字。

小数数据类型

Adaptive Server 为含有小数点的数字提供了另外两种精确数值数据类型：*numeric* 和 *dec*[*imal*]。将压缩在 *numeric* 和 *decimal* 列中存储的数据以保留磁盘空间，并会在算术运算之后将精确度保存在最低有效位。除了一个方面之外， *numeric* 和 *decimal* 数据类型在各个方面都是一样的：只有标度为 0 的 *numeric* 数据类型能够用于 IDENTITY 列。

指定精度和标度

numeric 和 *decimal* 数据类型接受两个可选的参数， *precision* 和 *scale*，这两个参数用小括号括起来并用逗号分开：

***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。*precision* 和 *scale* 决定了可以在小数或数字列中存储的数值范围：

- 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。您可以将精度指定为 1 位至 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。您可以将标度指定为 0 位至 38 位，或者使用缺省的 0 位标度。

存储大小

numeric 或 *decimal* 列的存储大小取决于其精度。1 或 2 位数字列的最小存储要求是 2 字节。精度每增加 2 位，存储大小约增加 1 字节，最大可以达到 17 字节。

请使用下面的公式来计算 *numeric* 或 *decimal* 列的准确存储大小：

```
ceiling (precision / log 256 ) + 1
```

例如， *numeric*(18,4) 列的存储大小是 9 字节。

输入小数数据

把 *decimal* 和 *numeric* 数据作为一个前面有可选的正号或负号并且包含一个可选的小数点的数字串来输入。如果数值超出了为该列所指定的精度或标度， Adaptive Server 会返回一条错误消息。标度为 0 的精确数值类型显示时没有小数点。

表 1-8 显示了数据类型为 *numeric*(5,3) 的列的一些有效条目，并且指出了 **isql** 显示这些数值的方式：

表 1-8：有效的小数值

所输入的数值	所显示的数值
12.345	12.345
+12.345	12.345
-12.345	-12.345
12.345000	12.345
12.1	12.100
12	12.000

表 1-9 显示了数据类型为 *numeric*(5,3) 的列的一些无效条目：

表 1-9：无效的小数值

所输入的数值	错误类型
1,200	不允许有逗号。
12-	负号应该位于数字之前。
12.345678	小数点右侧的非零位数过多。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 提供了 *smallint*、 *int*、 *numeric* 和 *decimal* SQL92 精确数值数据类型。 *tinyint* 类型是 Transact-SQL 扩展。

近似数值数据类型

功能

使用近似数值类型 *float*、 *double precision* 和 *real* 来用于那些在算术运算过程中能够允许舍入的数字数据。近似数值类型尤其适用于那些涉及很宽范围的数值的数据。除了 **modulo** 之外，它们支持所有的集合函数和所有的算术运算。

理解近似数值数据类型

用来存储浮点数的近似数值数据类型在表示实际的数字时存在固有的微小偏差 — 因此它的名称是 “近似数值”。要使用这些数据类型，您必须理解并接受它们的限制。

任何时候打印或显示一个浮点数，打印的结果同所存储的数字并不完全一样，而且所存储的数字和用户所输入的数字也不完全一样。大多数情况下，所存储的表示法的接近程度已经足够了，并且软件使得打印的输出看起来简直与原始的输入完全相同。但是您必须了解如果打算将浮点数用于计算时，尤其是您要使用近似数值数据类型来执行重复计算时出现的不准确性 — 最终结果的不准确性可能会十分惊人和出乎意料。

造成这种不准确性的原因是浮点数在计算机中是作为二进制小数（即，以被 2 的幂所除的数值形式）来存储的，而我们使用的数字是十进制小数 （10 的幂）。这就意味着只有很少部分的数字可以被准确存储：0.75 (3 / 4) 可以被准确存储，因为它是一个二进制小数 （4 是 2 的幂）； 0.2 (2 / 10) 则不能被准确存储 （10 不是 2 的幂）。

一些数字包含了过多的位数，所以无法精确存储。 *double precision* 被存储为 8 个二进制字节，并能够以适当的精度表示大约 17 位数字。 *real* 被存储为 4 个二进制字节，因而只能够以适当的精度表示大约 6 位数字。

如果您开始使用的数字是接近于正确的，并且使用同样接近于正确的其它数字同其进行计算，最后可能很容易得到一个不大接近于正确数值的结果。如果这些注意事项对应用程序很重要，请考虑使用精确数值数据类型。

范围、精度和存储大小

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* 类型可以接受用小括号括起来的可选的二进制精度。精度为 1-15 的 *float* 列被存储为 *real* ； 而那些精度更高的数字则被存储为 *double precision*。

这三种类型的范围和存储精度都与计算机相关。

表 1-10 显示了每一种近似数值类型的范围和存储大小。请注意， **isql** 仅显示小数点后面的 6 位有效数字，并舍入余下的数字：

表 1-10：近似数值数据类型

数据类型	存储的字节数
float[(default precision)]	*default precision*< 16 时为 4 *default precision* >= 16 时为 8
double precision	8
real	4

输入近似数值数据

将近似数值数据作为一个后跟可选指数的尾数来输入：

- 尾数是一个有符号数或无符号数，它既可以有也可以没有小数点。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。
- 指数以字符 “e”或 “E”起始，并且必须是一个整数。

该条目所表示的值是下列乘积：

```
mantissa * 10EXPONENT
```

例如， 2.4E3 表示 2.4 乘以 10^3，即 2400。

标准和一致性

标准	一致性级别
SQL92	*float*、 *double precision* 和 *real* 数据类型符合初级 SQL92。

money 数据类型

功能

使用 *money* 和 *smallmoney* 数据类型来存储货币数据。您可以将这些类型用于美元和其它十进制货币，但是 Adaptive Server 没有提供将一种货币转换为另一种货币的方法。您可以将 *money* 和 *smallmoney* 数据用于除 **modulo** 外的所有算术运算以及所有集合函数。

精确性

money 和 *smallmoney* 都精确到货币单位的万分之一，但是在显示时它们将上舍入为两位小数。缺省的输出格式是在每三位后放置一个逗号。

范围和存储大小

表 1-11 总结了 money 数据类型的范围和存储要求：

表 1-11：money 数据类型

数据类型	范围	存储的字节数
money	货币值介于 +922,337,203,685,477.5807 和 -922,337,203,685,477.5808 之间	8
smallmoney	货币值介于 +214,748.3647 和 -214,748.3648 之间	4

输入货币值

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。

标准和一致性

标准	一致性级别
SQL92	*money* 和 *smallmoney* 数据类型是 Transact-SQL 扩展。

时间戳数据类型

功能

对于需要在 Client-Library™ 应用程序 （有关详细信息参见 “浏览模式”）中浏览的表格，在其中使用用户定义的 *timestamp* 数据类型。每次修改它的行时， Adaptive Server 会更新 *timestamp* 列。一个表只能有一列 *timestamp* 数据类型。

数据类型定义

timestamp 是一个由 Adaptive Server 提供的、用户定义的数据类型，它被定义为 *varbinary(8)* NULL。它需要 8 字节的存储空间。因为 *timestamp* 是用户定义的数据类型，所以您不能用它来定义其它的用户定义的数据类型。您不能在 **sum** 或 **avg** 集合函数中使用 *timestamp* 数据类型。

与 SQL 标准 *timestamp* 数据类型不同， Transact-SQL *timestamp* 数据类型不保存日期和时间信息，并且不能转换成日期和时间。*timestamp* 保存了类似如下所示的二进制类型数据：

```
timestamp
------------------
0x00010000000000e51
```

创建 *timestamp* 列

如果您创建了一个名为 *timestamp* 的列，却没有指定其数据类型，则 Adaptive Server 会将该列定义为 *timestamp* 数据类型：

```
create table testing
    (c1 int, timestamp, c2 int)
```

您也可以明确地将 *timestamp* 数据类型指派给名为 *timestamp* 的列：

```
create table testing
    (c1 int, timestamp timestamp, c2 int)
```

或者指派给具有另一种名称的列：

```
create table testing
    (c1 int, t_stamp timestamp,c2 int)
```

您可以创建一个名为 *timestamp* 的列并为其指派另一种的数据类型（但这会令其他用户感到混乱，并且不允许在 Open Client™ 中或同 **tsequal** 函数一起使用 **browse** 函数）：

```
create table testing
    (c1 int, timestamp datetime)
```

标准和一致性

标准	一致性级别
SQL92	*timestamp* 数据类型是 Transact-SQL 扩展。

日期和时间数据类型

功能

使用 *datetime* 和 *smalldatetime* 数据类型来存储绝对日期和时间信息。

Adaptive Server 也提供这些数据类型，用来存储二进制类型的信息。

范围和存储要求

表 1-12 总结了 *datetime* 和 *smalldatetime* 数据类型的范围和存储要求：

表 1-12：用于存储日期和时间的 Transact-SQL 数据类型

数据类型	范围	存储的字节数
datetime	1753 年 1 月 1 日至 9999 年 12 月 31 日	8
smalldatetime	1900 年 1 月 1 日至 2079 年 6 月 6 日	4

输入 *datetime* 和 *smalldatetime* 数据

datetime 和 *smalldatetime* 数据类型由一个日期部分以及位于其后面或前面的时间部分组成。（您既可以省略日期，也可以省略时间，或者两者都省略。） *datetime* 和 *smalldatetime* 值都必须用单引号或双引号括起来。

- *datetime* 列可以保存 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* 值可以精确到 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。存储大小为 4 字节：2 字节用于存储自 1900 年 1 月 1 日以来的天数， 2 字节用于存储自午夜以来的分钟数。

输入 *datetime* 或 *smalldatetime* 值的日期部分

日期包括月份、日和年份，并且可以用多种格式输入：

- 您可以将整个日期作为 4 位、 6 位或 8 位的无分隔符的字符串来输入，或是在日期分量之间使用斜杠 (/)、连字符 (-) 或句点 (.) 等分隔符。

- 当把日期作为无分隔符的字符串来输入时，所采用的格式应适用于该字符串长度。请在单个数字位的年份、月份和日前加零。如果输入的日期格式有错误，可能会令人曲解或导致错误发生。
- 当输入含分隔符的日期时，请使用 **set dateformat** 选项来指定所期望的日期分量顺序。如果分隔开的字符串中的第一个日期分量是四位数字， Adaptive Server 会将该字符串解释为 *yyyy-mm-dd* 格式。

- 某些日期格式可以接受 2 位数字的年份 (*yy*)。大于或等于 50 的日期被解释为 19yy ；那些小于 50 的日期则被解释为 20yy。
- 您既可以将月份指定为数字，也可以将其指定为名称。月份名及其缩写是因语言而异的，并且可以用大写、小写或大小写混合的方式输入。
- 如果省略了 *datetime* 或 *smalldatetime* 值的日期部分， Adaptive Server 会使用缺省日期 1900 年 1 月 1 日。

表 1-13 描述了输入 *datetime* 或 *smalldatetime* 值的日期部分时可以接受的格式：

表 1-13：datetime 和 smalldatetime 数据类型的日期格式

日期格式	解释	样本条目	含义
不带分隔符的 4 位数字字符串	被解释为 *yyyy*。日期缺省值为指定年份的 1 月 1 日。	“1947”	1947 年 1 月 1 日
不带分隔符的 6 位数字字符串	被解释为 *yymmdd*。 如果 *yy* < 50，则年份是 20*yy*。 如果 *yy* >= 50，则年份是 19*yy*。	“450128” “520128”	2045 年 1 月 28 日 1952 年 1 月 28 日
不带分隔符的 8 位数字字符串	被解释为 *yyyymmdd*。	“19940415”	1994 年 4 月 15 日
由 2 位数的月份、日和年份（它们被斜杠、连字符、句点或是上述符号的组合隔开）所组成的字符串。	**dateformat** 和 **language set** 选项决定了所期望的日期分量的顺序。对于 us_english 而言，缺省的顺序是 *mdy*。 如果 *yy* < 50，则年份就被解释为 20*yy*。如果 *yy* >= 50，则年份就被解释为 19*yy*。	“4 / 15 / 94” “4.15.94” “4-15-94” “04.15 / 94”	当 **dateformat** 选项被设置为 **mdy** 时，所有这些条目都将被解释为 1994 年 4 月 15 日。
由 2 位数的月份、 2 位数的日和 4 位数的年份 （它们被斜杠、连字符、句号或是上述符号的组合隔开）所组成的字符串。	**dateformat** 和 **language set** 选项决定了所期望的日期分量的顺序。对于 us_english 而言，缺省的顺序是 *mdy*。	“04 / 15.1994”	当 **dateformat** 选项设置为 **mdy** 时，它将被解释为 1994 年 4 月 15 日。

表 1-13：datetime 和 smalldatetime 数据类型的日期格式 （续）

日期格式	解释	样本条目	含义
月份用字符格式 （既可以是完整的月份名称，也可以是其标准的缩写）输入，后跟一个可选的逗号。	如果输入的年份是 4 位数字，则日期分量可以用任何顺序输入。	“April 15, 1994” “1994 15 apr” “1994 April 15” “15 APR 1994”	所有这些条目都被解释为 1994 年 4 月 15 日。
	如果省略了日的输入，则必须指定 4 位数字的年份。日的缺省值为该月的第一天。	“apr 1994”	1994 年 4 月 1 日
	如果年份仅是两位数字 (*yy*)，那么它应该出现在日的后面。如果 *yy* < 50，则年份被解释为 20*yy*。如果 *yy* >= 50，则年份被解释为 19*yy*。	“mar 16 17”	2017 年 3 月 16 日
		“apr 15 94”	1994 年 4 月 15 日
空字符串， ""	日期缺省值为 1900 年 1 月 1 日。	""	1900 年 1 月 1 日

输入 *datetime* 或 *smalldatetime* 值的时间部分

datetime 或 *smalldatetime* 值的时间成分必须按如下所述指定：

```
hours[:minutes[:seconds[:milliseconds]] [AM | PM]
```

- 12AM 表示午夜， 12PM 表示中午。
- 时间值必须包含一个冒号，或者是 AM 或 PM 标记。 AM 或 PM 能够用大写、小写或大小写混合的方式输入。
- 对秒数的说明既可以包括一个以小数点为开头的小数部分，也可以是一个以冒号开头的毫秒数。例如，“12:30:20:1”表示 12:30 过二十秒又一毫秒； “12:30:20.1”表示 12:30 过二十又十分之一秒。
- 如果省略了 *datetime* 或 *smalldatetime* 值的时间部分， Adaptive Server 会使用缺省时间 12:00:00:000AM。

datetime 和 *smalldatetime* 值的显示格式

datetime 和 *smalldatetime* 值的显示格式是 “Mon dd yyyy hh:mmAM”（或 “PM”）。例如，“Apr 15 1988 10:23PM”。要显示秒数和毫秒数，并得到另外的日期样式和日期分量顺序，请使用 **convert** 函数将数据转换为字符串。 Adaptive Server 可能会舍入或截断毫秒值。

表 1-14 列出了 *datetime* 条目及其显示值的一些示例：

表 1-14：datetime 条目示例

条目	所显示的数值
“1947”	Jan 1 1947 12:00AM
“450128 12:30:1PM”	Jan 28 2045 12:30PM
“12:30.1PM 450128”	Jan 28 2045 12:30PM
“14:30.22”	Jan 1 1900 2:30PM
“4am”	Jan 1 1900 4:00AM

查找与某一模式相匹配的 *datetime* 值

使用 **like** 关键字来查找与某一特定模式相匹配的日期。如果使用等于运算符 (=) 来搜索某一特定月份、日和年份的 *datetime* 值，Adaptive Server 仅会为时间正好是 12:00:00:000AM 的日期返回那些数值。

例如，如果将数值 “9:20” 插入名为 *arrival_time* 的列，就会将该条目转变为 “Jan 1 1900 9:20AM”。如果您使用相同运算符查找这一条目，将不会发现它：

```
where arrival_time = "9:20" /* does not match */
```

利用 **like** 运算符，您就可以找到该条目：

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

使用 **like** 时， Adaptive Server 首先将日期转换为 *datetime* 格式，然后再转换为 *varchar*。显示格式包括用当前语言表示的 3 个字符的月份、用于表示日的 2 个字符、用于表示年份的 4 个字符、以及用小时和分钟加上 “AM” 或 “PM” 表示的时间。

用 **like** 搜索时，您不能使用在输入 *datetime* 和 *smalldatetime* 值的日期部分时所能采用的多种输入格式。因为标准的显示格式不包括秒数或毫秒数，所以您不能使用 **like** 和匹配模式来搜索秒数或毫秒数，除非您还使用了 *style* 9 或 109 以及 **convert** 函数。

如果您使用了 **like**，并且当天的月份是位于 1 和 9 之间的一个数值，请在月份和日之间插入两个空格，以此来匹配 *datetime* 值的 *varchar* 转换。同样，如果小时数小于 10，转换结果将在年份和小时之间放置两个空格。该子句是：

```
like May 2%
```

（“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 **like** 比较才会转换为 *varchar*，因此您无须在其它的日期比较中插入额外的空格，只有使用 **like** 时才需要这样做。

处理日期

您可以使用内部日期函数来对 *datetime* 值进行某些算术计算。参见第 2 章 “Transact-SQL 函数”中的 “日期函数”。

标准和一致性

标准	一致性级别
SQL92	*datetime* 和 *smalldatetime* 数据类型是 Transact-SQL 扩展。

字符数据类型

功能

字符数据类型用于存储含有字母、数字和符号的字符串。对于像 us_english 这样的单字节字符集，应使用固定长度的数据类型 *char(n)* 和可变长度的数据类型 *varchar(n)*。对于像日语这样的多字节字符集，应使用固定长度的数据类型 *nchar(n)* 和可变长度的数据类型 *nvarchar(n)*。字符数据类型可以存储最多 255 个字符；对于超过 255 个字符的字符串，应使用 *text* 数据类型（在“text 和 image 数据类型”中说明）。

长度和存储大小

使用 *n* 来为固定长度数据类型 *char(n)* 和 *nchar(n)* 指定字符长度。小于指定长度的条目会用空白填补；大于指定长度的条目将在不发出警告的情况下被截断（除非 **set** 命令的 **string_rtruncation** 选项设置为 **on**）。允许有空值存在的列在内部被转换为可变长度的列。

使用 *n* 来为可变长度数据类型 *varchar(n)* 和 *nvarchar(n)* 指定最大字符长度。可变长度列中的数据会去除掉尾随空白；存储大小是输入数据的实际长度。可变长度变量和参数中的数据将保留全部的尾随空白，但是不会填充到所定义的长度。字符文字被视为可变长度数据类型。

固定长度列将比可变长度列占用更多的存储空间，但是却可以较快地对其进行访问。表 1-15 总结了不同字符数据类型的存储要求：

表 1-15：字符数据类型

数据类型	存储	存储的字节数
char(n)	单字节字符集中的固定长度数据，如社会保险号或邮政编码。	*n*
nchar(n)	多字节字符集中的固定长度数据	*n* * @ *@ncharsize*
varchar(n)	单字节字符集中的可变长度数据，如名称。	输入的实际字符数
nvarchar(n)	多字节字符集中的可变长度数据	实际的字符数 * @ *@ncharsize*

用系统函数确定列长度

使用 **char_length** 字符串函数和 **datalength** 系统函数来确定列长度：

- **char_length** 为可变长度数据类型，返回了去除其尾随空白之后在该列中的字符数。
- **datalength** 为存储在可变长度列中的数据，返回了去除其尾随空白之后的字节数。

当声明 *char* 值可为空时， SQL Server 在内部将其存储为 *varchar*。

输入字符数据

字符串必须用单引号或双引号引起来。如果使用了 **set quoted_identifier on**，请将单引号用于字符串；否则， Adaptive Server 会将其视为标识符。

含有双引号字符的字符串应该用单引号将其引起来。含有单引号字符的字符串应该用双引号将其引起来。例如：

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

您还可以选择其它方法，即对于要在字符串中包括的每一个引号，输入两个引号。例如：

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

如果要使一个字符串延续到屏幕的下一行，请在转到下一行之前输入反斜杠 (\)。

处理空白

下列示例创建了一个名为 *spaces* 的表格，它同时拥有固定长度和可变长度的字符列：

```
create table spaces (cnot char(5) not null,
    cnull char(5) null,
    vnot varchar(5) not null,
        vnull varchar(5) null,
    explanation varchar(25) not null)

insert spaces values ("a", "b", "c", "d",
    "pads char-not-null only")
insert spaces values ("1    ", "2    ", "3    ",
    "4    ", "truncates trailing blanks")
insert spaces values ("    e", "    f", "    g",
     "    h", "leading blanks, no change")
```

```
insert spaces values ("   w ", "   x ", "   y ",
    "   z ", "truncates trailing blanks")
insert spaces values ("", "", "", "",
    "empty string equals space" )

select "[" + cnot + "]",
       "[" + cnull + "]",
       "[" + vnot + "]",
       "[" + vnull + "]",
    explanation from spaces
```

```
                                    explanation
-------  -------  ------- ------- --------------------
[a    ]  [b]      [c]     [d]     pads char-not-null only
[1    ]  [2]      [3]     [4]     truncates trailing blanks
[    e]  [    f]  [    g] [    h] leading blanks, no change
[   w ]  [   x]   [   y]  [   z]  truncates trailing blanks
[     ]  [ ]      [ ]     [ ]     empty string equals space

(5 rows affected)
```

该示例阐明了列数据类型和空类型如何相互作用来确定处理空白区域的方式：

- 只有 *char* **not null** 和 *nchar* **not null** 列可以被填充到列的整个宽度；*char* **null** 列被视为 *varchar*，而 *nchar* **null** 列则被视为 *nvarchar*。
- 前面的空白不受影响。
- 除了 *char* 和 *nchar* **not null** 列之外，尾随空白将被截断。
- 空字符串 ("") 当作单个空格处理。在 *char* 和 *nchar* **not null** 列中，结果是一个空的列长度域。

处理字符数据

您可以使用 **like** 关键字来搜索字符串以找到特定的字符和处理其内容的内部字符串函数。由数字组成的字符串在使用 **convert** 函数将其转换为精确和近似数值数据类型之后，可以用于算术运算。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 提供了 *char* 和 *varchar* SQL92 数据类型。 *nchar* 和 *nvarchar* 数据类型是 Transact-SQL 扩展。

二进制数据类型

函数

使用二进制数据类型 *binary(n)* 和 *varbinary(n)* 可用十六进制式符号存储最多 255 字节的原始二进制数据 （如图象）。

有效的 Binary 和 Varbinary 条目

二进制数据以字符 “0x”起始，并可以包括数字和 A 至 F 的大小写字母的任意组合。

使用 *n* 来以字节为单位指定列的长度，或使用缺省的 1 字节长度。每一字节存储 2 个二进制位。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。

对于其中的所有条目在长度上都应该大体上相等的数据而言，应使用固定长度二进制类型 *binary(n)*。

对于预计在长度上变化很大的数据而言，应使用可变长度二进制类型 *varbinary(n)*。

由于 *binary* 列中的条目会被用零填充以达到列长度 (*n*)，因此它们可能需要占用比 *varbinary* 列中的条目更多的存储空间，但是访问其的速度较快。

将 *image* 数据类型用于多于 255 字节的条目

使用 *image* 数据类型将较大的二进制数据块 （最多为 2,147,483,647 字节）存储到外部数据页上。您不能将 *image* 数据类型用于变量或存储过程中的参数。有关详细信息，参见 “text 和 image 数据类型”一节。

尾随零的处理

所有的 *binary* **not null** 列都会用零来填满列的整个宽度。由于接受空值的列必须被视为可变长度列，因此在所有的 *varbinary* 数据和 *binary* **null** 列中尾随零都会被截断。

下列示例用 *binary* 和 *varbinary* 数据类型的全部四种变化以及 NULL 和 NOT NULL 创建了一个表格。在全部四列中都插入了同样的数据，并根据列的数据类型对数据进行了填充或截断。

```
create table zeros (bnot binary(5) not null,
            bnull binary(5) null,
            vnot varbinary(5) not null,
            vnull varbinary(5) null)

insert zeros values (0x12345000, 0x12345000,
            0x12345000, 0x12345000)
insert zeros values (0x123, 0x123, 0x123, 0x123)

select * from zeros
bnot             bnull        vnot         vnull
------------     ---------    ----------   ---------
0x1234500000     0x123450     0x123450     0x123450
0x0123000000     0x0123       0x0123       0x0123
```

因为每一个存储字节保存了 2 个二进制位，所以 Adaptive Server 希望二进制条目包含跟有偶数个数字的 “0x” 字符。如果 “0x” 之后跟有奇数个数字，则 Adaptive Server 假定您省略了前导的 0 并替您添上 0。

输入值 “0x00” 和 “0x0” 被以 “0x00” 的形式存储到可变长度二进制列 （*binary* **null**、 *image* 和 *varbinary* 列）中。在固定长度二进制 (*binary* **not null**) 列中，该数值会被用 0 填充而达到域的整个长度：

```
insert zeros values (0x0, 0x0,0x0, 0x0)

select * from zeros where bnot = 0x00
bnot           bnull       vnot        vnull
----------     ------      -----       ------------
0x0000000000   0x00        0x00        0x00
```

如果输入数值不包括 “0x”， Adaptive Server 会假定该数值为一个 ASCII 值并对其进行转换。例如：

```
create table sample (col_a binary(8))

insert sample values ('002710000000ae1b')

select * from sample
col_a
------------------
0x3030323731303030
```

平台相关性

输入某个值的确切格式取决于您所使用的平台。因此，涉及二进制数据的计算可能会在不同的计算机上产生不同的结果。

您不能在 **sum** 或 **avg** 集合函数中使用二进制数据类型。

对于十六进制字符串和整数之间的、独立于平台的转换，应使用 **inttohex** 和 **hextoint** 函数，而不是该平台专有的转换函数。有关详细信息，参见第 2 章 “Transact-SQL 函数”中的 “数据类型转换函数”。

标准和一致性

标准	一致性级别
SQL92	*binary* 和 *varbinary* 数据类型是 Transact-SQL 扩展。

位数据类型

功能

bit 数据类型用于包含真/假和是/否类型数据的列。*syscolumns* 系统表中的 *status* 列为 *bit* 数据类型列指示了唯一的偏移位置。

将数据输入 *bit* 列

bit 列保存 0 或 1。也可以接受 0 或 1 之外的其它整数值，但始终将这些数值解释为 1。

存储大小

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。

限制

数据类型为 *bit* 的列不能为 NULL，并且其中不能有索引。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

sysname 数据类型

功能

sysname 是一种用户定义的数据类型，它分布在 Adaptive Server 安装磁带上并在系统表中使用。它的定义为：

```
varchar(30) "not null"
```

使用 *sysname* 数据类型

您不能将一个列、参数或变量声明为 *sysname* 类型。但是，可以用 *sysname* 的基类型来创建用户定义的数据类型。然后，您就可以用这一用户定义的数据类型来定义列、参数和变量了。

标准和一致性

标准	一致性级别
SQL92	所有的用户定义的数据类型 （包括 *sysname*）都是 Transact-SQL 扩展。

text 和 image 数据类型

功能

text 列是可变长度的列，它可以保存最多 2,147,483,647 (2^{31} - 1) 字节的可打印字符。

image 列是可变长度的列，它可以保存最多 2,147,483,647 (2^{31} - 1) 字节的十六进制式数据。

定义 *text* 或 *image* 列

您可以像定义其它任意列一样，使用 **create table** 或 **alter table** 语句来定义 *text* 或 *image* 列。 *text* 和 *image* 数据类型的定义不包括长度。它们允许空值存在。列定义的格式如下：

```
column_name {text | image} [null]
```

例如，对于拥有一个允许有空值的 *text* 列 (*blurb*) 的 *pubs2* 数据库而言，可以用于其中的作者的 *blurbs* 表的 **create table** 语句是：

```
create table blurbs
(au_id id not null,
copy text null)
```

要在拥有一个 *image* 列的 *pubs2* 数据库中创建 *au_pix* 表：

```
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)
```

Adaptive Server 如何存储 *text* 和 *image* 数据

Adaptive Server 将 *text* 和 *image* 数据存储到数据页的一个已链接的列表中，该列表同表的其余部分相分离。每个 *text* 或 *image* 页最多可以存储 1800 字节的数据。不管表中包含了多少 *text* 和 *image* 列，表中的所有 *text* 和 *image* 数据都会被存储到单个页链中。

将额外的页放到另一设备上

您可以用 **sp_placeobject** 将后续的 *text* 和 *image* 数据页放到不同的逻辑设备上。

零填充

小于 255 字节并有奇数个十六进制位的 *image* 值会用前导的零填充（插入的 “0xaaabb”将变为 “0x0aaabb”）。

➤ ***注意***

插入超过 255 字节并且字节数是奇数的 *image* 值是错误的。

分区对数据的存储方式没有影响

您可以使用 **alter table** 命令的 **partition** 选项，对包含 *text* 和 *image* 列的表进行分区。对表进行分区将为表中的其它列创建额外的页链，但是**不会**影响 *text* 和 *image* 列的存储方式。

初始化文本和图像列

在更新它们或插入一个非空值之前， *text* 和 *image* 列不会被初始化。初始化将为每一个非空的 *text* 或 *image* 数据值至少分配一个数据页。它还在表中创建一个指向 *text* 或 *image* 数据位置的指针。

例如，以下语句创建表 *testtext* 并通过插入一个非零值初始化 *blurb* 列。现在该列将具有一个有效的文本指针，并且已经分配有第一个 2K 数据页。

```
create table texttest
(title_id varchar(6), blurb text null, pub_id
char(4))

insert texttest values
("BU7832", "Straight Talk About Computers is an
annotated analysis of what computers can do for
you:a no-hype guide for the critical user.",
"1389")
```

以下语句为 *image* 值创建一个表并对 *image* 列进行初始化：

```
create table imagetest
(image_id varchar(6), imagecol image null,
graphic_id char(4))

insert imagetest values
("94732", 0x0000008300000000000100000000013c,
"1389")
```

➤ ***注意***

记住要将 *text* 值用引号引起来并在 *image* 值前面加上字符 “0x”。

有关用 Client-Library 程序来插入和更新 *text* 和 *image* 数据的信息，参见 *Client-Library/C Reference Manual*。

通过允许空值存在来节省空间

为了节省空的 *text* 或 *image* 列的存储空间，请将其定义为允许空值存在并在使用该列前插入 **(insert)** 空值。插入一个空值不会初始化 *text* 或 *image* 列，因此也就不会创建文本指针或分配 2K 字节的存储空间。例如，以下语句将数值插入上面创建的 *testtext* 表的 *title_id* 和 *pub_id* 列中，但并不初始化 *blurb* 文本列：

```
insert texttest
(title_id, pub_id) values ("BU7832", "1389")
```

当 *text* 或 *image* 行被赋予一个非空值之后，它就会始终包含至少一个数据页。将该值复位为空并不会重新分配其数据页。

从 *sysindexes* 获得信息

每一个含有 *text* 或 *image* 列的表在 *sysindexes* 中都有一个额外的行，该行提供有关这些列的信息。 *sysindexes* 中的 *name* 列使用 “*tablename*” 表单。 *indid* 始终为 255。这些列提供了有关文本存储的信息：

表 1-16：文本和图像数据的存储

列	说明
ioampg	指向文本页链的分配页的指针
first	指向文本数据的第一页的指针
root	指向最后一页的指针
segment	对象驻留的段号

您可以查询 *sysindexes* 表来获得有关这些列的信息。例如，下列查询报告了 *pubs2* 数据库中的 *blurbs* 表所使用的数据页的数目：

```
select name, data_pgs(object_id("blurbs"), ioampg)
from sysindexes
where name = "tblurbs"
name
------------------------------ -----------
tblurbs                                  7
```

使用 *readtext* 和 *writetext*

在您能够使用 **writetext** 来输入 *text* 数据或用 **readtext** 来读取它们之前，必须初始化 *text* 列。有关详细信息，参见 **readtext** 和 **writetext**。

通过使用 **update** 来用 NULL 替代现有的 *text* 和 *image* 数据，将重新声明除第一页 （它仍然可供 **writetext** 在将来使用）之外的所有已分配的数据页。要重新分配该行的所有存储，请使用 **delete** 撤除整行。

确定一列占用了多少空间

sp_spaceused 提供了有关文本数据所使用的空间的信息 （如 *index_size*）：

```
sp_spaceused blurbs
name            rowtotal  reserved  data    index_size unused
--------------- --------  --------- ------- ---------- ------
blurbs          6         32 KB     2 KB    14 KB      16 KB
```

文本和图像列的限制

text 和 *image* 列不能按如下方法使用：

- 用作存储过程的参数或传递给这些参数的值
- 用作局部变量
- 用于 **order by**、 **compute**、 **group by** 和 **union** 子句
- 用于索引
- 用于子查询或连接
- 用于 **where** 子句，除非带有关键字 **like**
- 同 + 并置运算符一起使用
- 用于触发器的 **if update** 子句

查询 *text* 和 *image* 数据

下列的全局变量返回有关 *text* 和 *image* 数据的信息：

表 1-17：文本和图像全局变量

变量	解释
@ @*textptr*	由进程插入或更新的、最后的 *text* 或 *image* 列的文本指针。不要将这一全局变量同 Open Client **textptr()** 函数相混淆。
@ @*textcolid*	由 @ @*textptr* 所引用的列的标识。
@ @*textdbid*	数据库的标识，该数据库含有其列被 @ @*textptr* 所引用的对象。

表 1-17：文本和图像全局变量 （续）

变量	解释
@ @*textobjid*	对象的标识，该对象含有被 @ @*textptr* 引用的列。
@ @*textsize*	**set textsize** 选项的当前值，指定了由 **select** 语句所返回的 *text* 或 *image* 数据的最大长度 （以字节为单位）。它的缺省值为 32K。 @ @*textsize* 的最大大小为 2^{31} - 1 （即 2,147,483,647）。
@ @*textts*	由 @ @*textptr* 所引用的列的文本时间戳。

转换 *text* 和 *image* 数据类型

您可以使用 **convert** 函数，显式地将 *text* 值转换为 *char* 或 *varchar*，将 *image* 值转换为 *binary* 或 *varbinary*，但是字符和二进制数据类型的最大长度要限制为 255 字节。如果未指定长度，转换后的值将具有缺省的长度 （30 个字节）。不支持隐式转换。

text 数据中的模式匹配

使用 **patindex** 函数来搜索 *text*、*varchar* 或 *char* 列中某一指定模式第一次出现的开始位置。通配符%必须位于模式之前和之后 （搜索第一个或最后一个字符的情况除外）。

您也可以使用 **like** 关键字来搜索特定的模式。下列示例由 *blurbs* 表的 *copy* 列中，选择出了每一个包含模式 “Net Etiquette”的 *text* 数据值。

```
select copy from blurb
where copy like "%Net Etiquette%"
```

禁止重复行

指向 *text* 或 *image* 数据的指针唯一地标识了每一行。因此，含有 *text* 或 *image* 数据的表不能包含重复行，除非所有 *text* 和 *image* 数据为 NULL。如果是这种情况，指针不会被初始化。

标准和一致性

标准	一致性级别
SQL92	*text* 和 *image* 数据类型是 Transact-SQL 扩展。

用户定义的数据类型

功能

用户定义的数据类型是建立在系统数据类型和 *sysname* 这一用户定义的数据类型基础上的。在创建了用户定义的数据类型后，您可以使用它来定义列、参数和变量。由用户定义的数据类型而创建的对象将继承这一用户定义的数据类型的规则、缺省值、空类型和 IDENTITY 属性，并继承该用户定义的数据类型所基于的系统数据类型的缺省值和空类型。

在 *model* 数据库中创建常用数据类型

用户定义的数据类型必须在要使用它的每一个数据库中创建。在 *model* 数据库中创建常用的类型是切实可行的。在创建每一个新数据库时，这些类型会被自动添加到其中 （包括用于临时表的 *tempdb*）。

创建用户定义的数据类型

Adaptive Server 允许您基于任一系统数据类型，用 **sp_addtype** 系统过程创建用户定义的数据类型。您不能基于另一种用户定义的数据类型 （如 *pubs2* 数据库中的 *timestamp* 或 *tid* 数据类型）创建用户定义的数据类型。

sysname 数据类型是该规则的一个例外。虽然 *sysname* 是用户定义的数据类型，但是您可以用它来建立用户定义的数据类型。

用户定义的数据类型是数据库对象。其名称是区别大小写的并且必须遵循标识符的规则。

您可以用 **sp_bindrule** 将规则绑定到用户定义的数据类型，并用 **sp_bindefault** 来绑定缺省值。

缺省情况下，基于用户定义的数据类型而建立的对象继承了用户定义的数据类型的空类型或 IDENTITY 属性。您可以替换列定义中的空类型或 IDENTITY 属性。

重命名用户定义的数据类型

使用 **sp_rename** 来重命名用户定义的数据类型。

删除用户定义的数据类型

使用 **sp_droptype** 将用户定义的数据类型从数据库中删除。

➤ ***注意***

不能删除已经在表中使用的数据类型。

获得有关数据类型的帮助

使用 **sp_help** 系统过程来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。也可以用 **sp_help** 显示表中每一列的数据类型、长度、精度和标度。

标准和一致性

标准	一致性级别
SQL92	用户定义的数据类型是 Transact-SQL 扩展。

2 Transact-SQL 函数

本章介绍 Transact-SQL 函数。这些函数用于从数据库中返回信息。在 **select** 列表中，在 **where** 子句中，以及在可以使用表达式的任何地方，都可以使用这些函数。它们通常被用作存储过程或程序的一部分。

函数类型

表 2-1 列出 Transact-SQL 函数的不同类型，并说明各种函数返回的信息类型。

表 2-1：Transact-SQL 函数类型

函数类型	说明
集合函数	生成摘要值，这些值将作为新列或查询结果中的附加行出现。
数据类型转换函数	将表达式从一种数据类型更改为另一种数据类型，并且为数据 / 时间信息指定新的显示格式。
日期函数	计算 *datetime* 值和 *smalldatetime* 值及其日期分量。
数学函数	返回在数学数据运算中通常所需的值。
安全性函数	返回与安全性相关的信息。
字符串函数	对二进制数据、字符串和表达式进行运算。
系统函数	返回数据库中的特殊信息。
文本和图像函数	提供在 *text* 和 *image* 数据运算中通常所需的值。

表 2-2 按字母顺序列出了这些函数。

表 2-2：Transact-SQL 函数列表

函数	类型	返回值
abs	数学	表达式的绝对值。
acos	数学	已指定余弦的角 （以弧度表示）。
ascii	字符串	表达式第一个字符的 ASCII 代码。
asin	数学	已指定正弦的角 （以弧度表示）。
atan	数学	已指定正切的角 （以弧度表示）。
atn2	数学	已指定正弦和余弦的角 （以弧度表示）。

表 2-2：Transact-SQL 函数列表 （续）

函数	类型	返回值
avg	集合	所有 （不同）值的数字平均值。
ceiling	数学	大于或等于指定值的最小整数。
char	字符串	整数的等值字符。
charindex	字符串	返回表示表达式起始位置的整数。
char_length	字符串	表达式中字符的数量。
col_length	系统	已定义的列长度。
col_name	系统	已指定表 ID 和列 ID 的列的名称。
convert	数据类型转换	转换成另一种数据类型或其它 *datetime* 显示格式的指定值。
cos	数学	指定角的余弦 （以弧度表示）。
cot	数学	指定角的余切 （以弧度表示）。
count	集合	（不同）非空值的数量。
curunreservedpgs	系统	指定磁盘区段中的可用页数。
data_pgs	系统	指定表或索引所用的页数。
datalength	系统	指定列或字符串的实际长度 （以字节表示）。
dateadd	日期	向指定日期添加给定数量的年、季度、小时和其它日期分量后所得的日期。
datediff	日期	两个日期之间的差值。
datename	日期	*datetime* 值中指定分量的名称。
datepart	日期	*datetime* 值中指定分量的整数值。
db_id	系统	指定数据库的 ID 号。
db_name	系统	已指定 ID 号的数据库的名称。
degrees	数学	具有指定弧度数的角的大小 （以度表示）。
difference	字符串	两个 **soundex** 值之间的差值。
exp	数学	求常数 e 的指定次幂所得的值。
floor	数学	小于或等于指定值的最大整数。
getdate	日期	当前的系统日期和时间。
hextoint	数据类型转换	与指定的十六进制字符串等值且独立于平台的整数。
host_id	系统	客户端进程的主机进程 ID。
host_name	系统	客户端进程的当前主机名。
index_col	系统	指定表或视图中带索引的列的名称。

表 2-2：Transact-SQL 函数列表 （续）

函数	类型	返回值
inttohex	数据类型转换	与指定整数等值且独立于平台的十六进制值。
isnull	系统	当 *expression1* 求值为 NULL 时，替代在 *expression2* 中指定的值。
is_sec_service_on	安全性	安全服务已启用时为 “1”；未启用时为 “0”。
lct_admin	系统	管理最后机会阈值。
license_enabled	系统	特性许可已启用时为 “1”；未启用时为 “0”。
log	数学	指定数字的自然对数。
log10	数学	指定数字的以 10 为底的对数。
lower	字符串	指定表达式的等值大写表达式。
isnull	字符串	删去前导空白的指定表达式。
max	集合	某一列中的最大值。
min	集合	某一列中的最小值。
mut_excl_roles	系统	两个角色之间的互斥性。
object_id	系统	指定对象的对象 ID。
object_name	系统	已指定对象 ID 的对象的名称。
patindex	字符串、文本和图像	指定模式在第一次出现时的起始位置。
pi	数学	常数值 3.1415926535897936。
power	数学	求指定数字的给定次幂所得的值。
proc_role	系统	用户具有执行过程的正确角色时为 1；用户不具有正确角色时 0。
ptn_data_pgs	系统	某一分区所用的数据页数。
radians	数学	具有指定度数的角的大小 （以弧度表示）。
rand	数学	0 和 1 之间的随机值，使用指定的源值生成。
replicate	字符串	指定表达式重复给定次所组成的字符串。
reserved_pgs	系统	分配给指定表或索引的页数。
reverse	字符串	其字符逆序排列的指定字符串。
right	字符串	字符表达式中从右端到指定个字符的部分。
role_contain	系统	当 *role2* 包含 *role1* 时为 1。
role_id	系统	指定其名称的角色的系统角色 ID。
role_name	系统	指定其系统角色 ID 的角色的名称。
round	数学	指定数字舍入到给定的小数位数后所得的值。

表 2-2：Transact-SQL 函数列表 （续）

函数	类型	返回值
rowcnt	系统	指定表中行数的估计值。
rtrim	字符串	删去尾随空白的指定表达式。
show_role	系统	登录的当前活动角色。
show_sec_services	安全性	用户的当前活动安全服务的列表。
sign	数学	指定值的符号：+1 （正）、 0、 -1 （负）。
sin	数学	指定角的正弦 （以弧度表示）。
soundex	字符串	表示表达式发音方式的 4 字符代码。
space	字符串	由指定个单字节空格组成的字符串。
sqrt	数学	指定数字的平方根。
str	字符串	指定数字的等值字符。
stuff	字符串	从一个字符串中删除指定个字符并将这些字符替换为另一种字符串后所形成的字符串。
substring	字符串	从另一种字符串中抽取指定个字符所形成的字符串。
sum	集合	值的总和。
suser_id	系统	服务器用户在 *syslogins* 系统表中的 ID 号。
suser_name	系统	当前服务器用户的名称或已指定服务器用户 ID 的用户。
tan	数学	指定角的正切 （以弧度表示）。
textptr	文本和图像	指向指定 *text* 列首页的指针。
textvalid	文本和图像	指向指定 *text* 列的指针有效时为 1 ； 无效时为 0。
tsequal	系统	比较 *timestamp* 值，这样，如果某行在被选中进行浏览以来经过了修改，就可防止将其更新。
upper	字符串	指定字符串的等值大写字符串。
used_pgs	系统	指定表及其集群索引所用的页数。
user	系统	当前服务器用户的名称。
user_id	系统	指定用户或当前用户的 ID 号。
user_name	系统	指定用户或当前用户在数据库中的名称。
valid_name	系统	指定字符串不是有效标识符时为 0 ； 该字符串有效时为 0 以外的数字。
valid_user	系统	当指定 ID 在该 Adaptive Server 上的至少一个数据库中是有效用户或别名时为 1。

以下各节将详细介绍这些函数类型。本章的其余部分会按字母顺序提供各种函数的说明。

集合函数

集合函数生成摘要值，这些值在查询结果中显示为新列。集合函数包括：

- **avg**
- **count**
- **max**
- **min**
- **sum**

集合函数可用于 select 列表或 **select** 语句或子查询的 **having** 子句。它们不能用于 **where** 子句。

查询中的每个集合都应有各自工作表。因此，一个查询所使用的集合不得超过在一个查询中允许的最大工作表数 (12)。

当集合函数应用于 *char* 数据类型的值时，它会删去所有尾随空白，将这种值隐式地转换为 *varchar*。

集合与 *group by* 一起使用

集合通常与 **group by** 一起使用。利用 **group by**，可以将表分组。集合为每个组生成单个值。如果不使用 **group by**，那么无论 select 列表的运算对象是一个表中的所有行还是 **where** 子句所定义的行子集，都将生成单个值作为结果。

集合函数和 NULL 值

集合函数用于计算特定列中非空值的摘要值。如果 **ansinull** 选项设置为 **off** （缺省），那么当集合函数遇到空值时就不会发出警告。如果 **ansinull** 设置为 **on**，那么当集合函数遇到空值时，查询将返回以下 SQLSTATE 警告：

```
Warning- null value eliminated in set function
(警告 — 在 set 函数中消除了空值)
```

矢量和标量集合

集合函数可应用于一个表中的所有行，此时会生成单个值，这就是标量集合。集合函数也可应用于在指定列或表达式中具有相同值的所有行 （使用 **group by** 和 / 或 **having** 子句），此时会为每个组生成一个值，这就是矢量集合。集合函数的结果将显示为新列。

可以将矢量集合嵌套在标量集合中。例如：

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

```
type
------------  ------------  ------------
UNDECIDED             NULL         15.23
business             13.73         15.23
mod_cook             11.49         15.23
popular_comp         21.48         15.23
psychology           13.50         15.23
trad_cook            15.96         15.23

(6 rows affected)
```

group by 子句应用于矢量集合（在本例中为 **avg**(*price*)）。标量集合 **avg**(**avg**(*price*)) 是 *titles* 表中按类型计算的平均价格的平均值。

在标准 SQL 中，如果 *select_list* 包括集合，那么所有 *select_list* 列都必须将集合函数应用于这些列，或者所有这些列都必须位于 **group by** 列表中。 Transact-SQL 没有这种限制。

示例 1 将显示一个带有标准限制的 **select** 语句。示例 2 则显示将另一项添加到 select 列表 (*title_id*) 的相同语句。为了说明显示方面的差异，还添加了 **order by**。这些“额外”的列也可以用 **having** 子句来引用。

1.
```
select type, avg(price), avg(advance)
from titles
group by type
```

```
type
------------  ------------  ------------
UNDECIDED             NULL          NULL
business             13.73      6,281.25
mod_cook             11.49      7,500.00
popular_comp         21.48      7,500.00
psychology           13.50      4,255.00
trad_cook            15.96      6,333.33

 (6 rows affected)
```

2.
```
select type, title_id, avg(price), avg(advance)
from titles
group by type
order by type
```

```
type         title_id
-----------  --------    ---------- ---------
UNDECIDED       MC3026      NULL        NULL
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
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
```

在 **group by** 之后可以使用列名或任何其它表达式 （列标题或别名除外）。

group by 列中的空值放置在单个组中。

select 语句中的 **compute** 子句使用行集合来生成摘要值。使用行集合，可以通过一个命令来检索明细行和摘要行。示例 3 说明了这一功能：

```
3. select type, title_id, price, advance
   from titles
   where type = "psychology"
   order by type
   compute sum(price), sum(advance) by type
```

```
type        title_id    price       advance
----------- -------     ----------  ---------
psychology  PS1372          21.59   7,000.00
psychology  PS2091          10.95   2,275.00
psychology  PS2106          7.00    6,000.00
psychology  PS3333          19.99   2,000.00
psychology  PS7777          7.99    4,000.00
                            sum     sum
                            ------- ----------
                            67.52   21,275.00
```

请注意示例 3 和无 **compute** 的示例 （示例 1 和 2）在显示方面的差异。

集合函数不能用于虚拟表 （如 *sysprocesses* 和 *syslocks*）。

如果在游标的 **select** 子句中包括了集合函数，则将无法更新该游标。

作为行集合的集合函数

行集合函数生成摘要值，这些值在查询结果中显示为附加列。

要将集合函数用作行集合，可使用以下语法：

```
Start of select statement

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

其中：

- *column_name* 是列名。必须用小括号将其括起来。只有精确数值、近似数值和货币列可以同 **sum** 及 **avg** 一起使用。

  一个 **compute** 子句可以将同一函数应用于若干列。当使用多个函数时，需使用多个 **compute** 子句。

- **by** 表示将为子群计算行集合值。只要 **by** 项的值发生更改，就会生成行集合值。如果使用 **by**，就必须使用 **order by**。

  当 **by** 将一个组分为多个子群并在每个分组级别应用一个函数后，列出多个项。

使用行集合，可以通过一个命令来检索明细行和摘要行。而集合函数通常为表中的所有选定行或每个组生成一个值，这些摘要值将显示为新列。

以下示例说明了这些差异：

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

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

(2 rows affected)

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
             sum         sum
             ----------  ----------------
                  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
             sum         sum
             ----------  ----------------
                  47.89         19,000.00
(7 rows affected)
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)
```

compute 子句中的列必须出现在 select 列表中。

如果 **ansinull** 选项设置为 **off** （缺省），那么当行集合遇到空值时就不会发出警告。如果 **ansinull** 设置为 **on**，那么当行集合遇到空值时，查询将返回以下 SQLSTATE 警告：

```
Warning- null value eliminated in set function
```
（警告 — 在 set 函数中消除了空值）

由于包含 **compute** 的语句会生成包含摘要结果 （不存储在数据库中）的表，因此不能在同一语句中将 **select into** 用作 **compute** 子句。

数据类型转换函数

数据类型转换函数将表达式从一种数据类型更改为另一种数据类型，并且为数据 / 时间信息指定新的显示格式。数据类型转换函数包括：

- **convert()**
- **inttohex()**
- **hextoint()**

数据类型转换函数可用于 select 列表、 **where** 子句以及允许使用表达式的任何其它地方。

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。

必须使用内部数据类型转换函数之一来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特定的数据类型。例如，不能将 *smallint* 数据转换为 *datetime*，也不能将 *datetime* 数据转换为 *smallint*。进行不支持的转换将会导致错误消息。

表 2-3 将指出各种数据类型转换是否可以隐式或显式地执行，或者是否属于不支持的转换。

表 2-3：显式、隐式和不支持的数据类型转换

原数据类型 \ 目的数据类型	tinyint	smallint	int	decimal	numeric	real	float	char、nchar	varchar、	text	smallmoney	money	bit	smalldatetime	datetime	binary	varbinary	image
tinyint	-	I	I	I	I	I	I	E	E	U	I	I	I	U	U	I	I	U
smallint	I	-	I	I	I	I	I	E	E	U	I	I	I	U	U	I	I	U
int	I	I	-	I	I	I	I	E	E	U	I	I	I	U	U	I	I	U
decimal	I	I	I	I/E	I/E	I	I	E	E	U	I	I	I	U	U	I	I	U
numeric	I	I	I	I/E	I/E	I	I	E	E	U	I	I	I	U	U	I	I	U
real	I	I	I	I	I	-	I	E	E	U	I	I	I	U	U	I	I	U
float	I	I	I	I	I	I	-	E	E	U	I	I	I	U	U	I	I	U
char、 nchar	E	E	E	E	E	E	E	I	I	E	E	E	E	I	I	I	I	E
varchar、 nvarchar	E	E	E	E	E	E	E	I	I	E	E	E	E	I	I	I	I	E
text	U	U	U	U	U	U	U	E	E	U	U	U	U	U	U	U	U	U
smallmoney	I	I	I	I	I	I	I	I	I	U	-	I	I	U	U	I	I	U
money	I	I	I	I	I	I	I	I	I	U	I	-	I	U	U	I	I	U
bit	I	I	I	I	I	I	I	I	I	U	I	I	-	U	U	I	I	U
smalldatetime	U	U	U	U	U	U	U	E	E	U	U	U	U	-	I	I	I	U
datetime	U	U	U	U	U	U	U	E	E	U	U	U	U	I	-	I	I	U
binary	I	I	I	I	I	I	I	I	I	U	I	I	I	I	I	-	I	E
varbinary	I	I	I	I	I	I	I	I	I	U	I	I	I	I	I	I	-	E
image	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	E	E	U

键：

E 要求显式数据类型转换。

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 当存在精度或标度损失并且 arithabort numeric_truncation 为 on 时，要求显式数据转换函数；否则，可以进行隐式转换。

U 不支持的转换。

– 某种数据类型向其自身转换。这种转换虽然可以进行，但毫无意义。

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， SQL Server 就会把空白转换为缺省的 *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。以下导致语法错误的一些字符的示例：

- 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*、 *varchar* 或 *nvarchar*。但受到 *character* 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 个字节）。

将数字转换为字符类型

精确和近似数值数据可以转换为字符类型。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)
Insufficient result space for explicit conversion
of NUMERIC value ’2.34’ to a CHAR field.
```

➤ ***注意***

将 *float* 数据转换为字符类型时，新类型的长度应至少为 25 个字符。

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)。当数据转换为货币类型时，它将上舍入到第四个小数位。

如果可能，从货币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，将生成 5：

```
select convert(int, $4.50)

-----------
          5
```

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 5 美元 （而不是 5 美分）的等值货币。

转换日期/时间信息

可以将可识别为日期的数据转换为 *datetime* 或 *smalldatetime*。不正确的月份名将导致语法错误。超出数据类型可接受范围的数据将导致算术溢出错误。

当 *datetime* 值转换为 *smalldatetime* 时，它们将舍入到最接近的分。

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。如果新类型是精度或标度不足以容纳数据的精确数值，就会发生错误。

例如，如果将浮动值或数字值当作参数提供需要整数的内部函数，就会截断该浮动值或数字值。不过， Adaptive Server 并不隐式地转换具有小数部分的数字，而是返回一个标度错误消息。例如，对具有小数部分的数字， Adaptive Server 返回错误 241，如果没有通过其它数据类型，则返回错误 257。

使用 **arithabort** 和 **arithignore** 选项可确定 Adaptive Server 如何处理因数字转换而导致的错误。

➤ ***注意***

arithabort 和 **arithignore** 选项已经为版本 10.0 或更高版本进行了重新定义。如果您在应用程序中使用这些选项，请检查它们是否仍表现出预期的行为。

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** 转换存储长度应大于 4 个字节的十六进制字符串

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 如何处理这些错误。缺省设置为 **arithabort arith_overflow on**，它将回退发生错误的整个事务。如果不包含事务的批处理发生了这种错误，则 **arithabort arith_overflow on** 将不回退批处理中以前的命令，并且 Adaptive Server 也不会执行批处理中产生错误的语句之后的语句。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。可以省略可选关键字 **arith_overflow**，而不会导致任何影响。

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 **arithabort numeric_truncation off**， Adaptive Server 就会截断查询结果并继续进行处理。

➤ ***注意***

为符合初级的 SQL92，可设置：

- **arithabort arith_overflow off**
- **arithabort numeric_truncation on**
- **arithignore off**

域错误

当 **convert()** 函数的参数超出其定义范围时，该函数就会产生域错误。这种错误极少发生。

二进制和整数类型之间的转换

binary 和 *varbinary* 类型存储十六进制式数据，这种数据由前缀“0x”以及随后的数字和字母字符串组成。

不同的平台会以不同的方式来解释这种字符串。例如，字符串“x0000100”在字节 0 最重要的计算机上表示 65536，而在字节 0 最不重要的计算机表示 256。

二进制类型可以显式地、使用 **convert** 函数或隐式地转换为整数类型。如果数据对于新类型来说太短，就会删去其“0x”前缀并用零填充。如果太长，则会将其截断。

convert 函数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** 函数可独立于平台将十六进制字符串转换为整数，而使用 **inttohex** 函数可独立于平台将整数转换为十六进制值。

在二进制和数字或十进制类型之间转换

在 *binary* 和 *varbinary* 数据字符串中，“0x”之后的前两位表示 *binary* 类型：“00”表示正数，“01”表示负数。将 *binary* 或 *varbinary* 类型转换为 *numeric* 或 *decimal* 时，务必要在“0x”位后指定“00”或“01”值；否则转换将失败。

例如，要将以下 *binary* 数据转换为 *numeric*，可设置：

```
select convert(numeric
(38, 18),0x00000000000000006b14bd1e6eea0000000000000000000000000000000000)

----------
123.456000
```

下例将同一 *numeric* 数据转换回 *binary*：

```
select convert(binary,convert(numeric(38, 18), 123.456))

-------------------------------------------------------------
0x0000000000000000006b14bd1e6eea0000000000000000000000000000000000
```

将图像列转换为二进制类型

可以使用 **convert** 函数来将 *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*

精确和近似数值类型可以隐式地转换为 *bit* 类型。字符类型需要显式的 **convert** 函数。

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。

日期函数

日期函数处理数据类型为 *datetime* 或 *smalldatetime* 的值。

日期函数包括：

- **dateadd**
- **datediff**
- **datename**
- **datepart**
- **getdate**

日期函数可用于 select 列表或查询的 **where** 子句。

datetime 数据类型只能用于 1753 年 1 月 1 日之后的日期。*datetime* 值必须用单引号或双引号引起来。对于更早的日期，应使用 *char*、*nchar*、*varchar* 或 *nvarchar*。Adaptive Server 可识别多种多样的日期格式。有关详细信息，参见 “数据类型转换函数”和第 1 章 “系统和用户定义的数据类型”中的 “日期和时间数据类型”。

如有必要 （例如在比较字符值和 *datetime* 值时）， Adaptive Server 会自动在字符和 *datetime* 值之间进行转换。

日期分量

日期分量、 Adaptive Server 可识别的缩写以及可接受的值包括：

日期分量	缩写	值
year	**yy**	1753 - 9999 （对于 *smalldatetime* 为 2079）
quarter	**qq**	1 - 4
month	**mm**	1 - 12
week	**wk**	1 - 54
day	**dd**	1 - 31
dayofyear	**dy**	1 - 366
weekday	**dw**	1 - 7 （周日 — 周六）
hour	**hh**	0 - 23
minute	**mi**	0 - 59
second	**ss**	0 - 59
millisecond	**ms**	0 - 999

如果输入只有 2 位的年份， <50 为下一世纪 （“25”表示 “2025”）， >=50 为本世纪 （“50”表示 “1950”）。

毫秒前可以带一个冒号或句号。如果带冒号，数字就表示千分之多少秒。如果带句号，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 分二十秒千分之一秒； “12:30:20.1”表示 12:30 分二十秒十分之一秒。 Adaptive Server 可以在添加 *datetime* 数据时舍入或截断毫秒值。

数学函数

数学函数返回在数学数据运算中通常所需的值。数学函数名不是关键字。

每种函数还接受可隐式地转换为指定类型的参数。例如，接受近似数值类型的函数还接受整数类型。 Adaptive Server 自动将参数转换为所需的类型。

数学函数包括：

- **abs**
- **acos**
- **asin**
- **atan**
- **atn2**
- **ceiling**

- **cos**
- **cot**
- **degrees**
- **exp**
- **floor**
- **log**
- **log10**
- **pi**
- **power**
- **radians**
- **rand**
- **round**
- **sign**
- **sin**
- **sqrt**
- **tan**

为处理这些函数的域错误或范围错误，提供了错误陷阱。用户可以通过设置 **(set) arithabort** 和 **arithignore** 选项来确定如何处理域错误：

- **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**，它将回退整个事务或中止发生错误的批处理。如果设置 **arithabort arith_overflow off**， Adaptive Server 将中止导致错误的语句，但继续处理事务或批处理中的其它语句。
- **arithabort numeric_truncation** 指定精确数值类型在隐式数据类型转换过程中导致标度损失后的行为。（当显式转换导致标度损失时，将截断结果而不发出任何警告。）缺省设置为 **arithabort numeric_truncation on**，它将中止导致错误的语句，但会继续处理事务或批处理中的其它语句。如果设置 **arithabort numeric_truncation off**， Adaptive Server 就会截断查询结果并继续进行处理。
- 缺省情况下， **arithignore arith_overflow** 选项设置为 **off**，这使 Adaptive Server 在任何查询导致数字溢出后都显示警告消息。将 **arithignore** 选项设置为 **on** 可忽略溢出错误。

➤ ***注意***

arithabort 和 **arithignore** 选项已经为版本 10.0 或更高版本进行了重新定义。如果您在应用程序中使用这些选项，请检查它们是否仍产生预期的结果。

安全性函数

安全性函数返回与安全性相关的信息。

安全性函数包括：

- **is_sec_service_on**
- **show_sec_services**

字符串函数

字符串函数对二进制数据、字符串和表达式进行运算。字符串函数包括：

- **ascii**
- **char**
- **charindex**
- **char_length**
- **difference**
- **lower**
- **ltrim**
- **patindex**
- **replicate**
- **reverse**
- **right**
- **rtrim**
- **soundex**
- **space**
- **str**
- **stuff**
- **substring**
- **upper**

字符串函数可以嵌套，并且可以用于 select 列表、 **where** 子句或允许使用表达式的任何地方。如果将常量用于字符串函数，则应用单引号或双引号将常量引起来。字符串函数名不是关键字。

每种字符串函数还接受可隐式地转换为指定类型的参数。例如，接受近似数值表达式的函数还接受整数表达式。 Adaptive Server 自动将参数转换为所需的类型。

字符串函数的限制

字符串函数的结果不得超出 255 个字符。

当 **set string_rtruncation** 为 **on** 时，如果 **insert** 或 **update** 截断字符串，用户就会接收到错误。不过，当显示的字符串被截断时， SQL Server 并不报告错误。例如：

```
select replicate("a", 250) + replicate("B", 250)
------------------------------------------------------------------
------------------------------------------------------------------
------------------------------------------------------------------
------------------------------------------------------------
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaBBBBB
```

系统函数

系统函数返回数据库中的特殊信息。系统函数包括：

- **col_length**
- **col_name**
- **curunreservedpgs**
- **data_pgs**
- **datalength**
- **db_id**
- **db_name**
- **host_id**
- **host_name**
- **index_col**
- **isnull**

- **lct_admin**
- **mut_excl_roles**
- **object_id**
- **object_name**
- **proc_role**
- **ptn_data_pgs**
- **reserved_pgs**
- **role_contain**
- **role_id**
- **role_name**
- **rowcnt**
- **show_role**
- **suser_id**
- **suser_name**
- **tsequal**
- **used_pgs**
- **user**
- **user_id**
- **user_name**
- **valid_name**
- **valid_user**

系统函数可用于 **select** 列表、 **where** 子句以及允许使用表达式的任何地方。

如果系统函数的参数是可选参数，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。

文本和图像函数

文本和图像函数对 *text* 和 *image* 数据进行运算。文本和图像函数包括：

- **textptr**
- **textvalid**

文本和图像内部函数名不是关键字。使用 **set textsize** 选项可限制 **select** 语句所检索的 *text* 或 *image* 数据量。

patindex 文本函数可用于 *text* 和 *image* 列，也可用作文本和图像函数。

使用 **datalength** 函数可获取 *text* 和 *image* 列中的数据长度。

text 和 *image* 列不能：

- 用作存储过程的参数
- 用作传递给存储过程的值
- 用作局部变量
- 用于 **order by**、 **compute** 和 **group by** 子句
- 用于索引
- 用于 **where** 子句，除非带有关键字 **like**
- 用于连接
- 用于触发器

abs

功能

返回表达式的绝对值。

语法

abs(*numeric_expression*)

参数

numeric_expression — 是数据类型为精确数值、近似数值、货币 （或任何可隐式转换为这些类型的类型）的列、变量或表达式。

示例

```
1. select abs(-1)

   -----------
             1
```

返回 -1 的绝对值。

注释

- **abs** 是一种数学函数，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。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **abs**。

参见

函数	ceiling、 floor、 round、 sign

acos

功能

返回已指定余弦的角 （以弧度表示）。

语法

acos(*cosine*)

参数

cosine — 是角的余弦，表示为 *float*、 *real*、 *double precision* 类型 （或任何可隐式转换为这些类型的数据类型）的列名、变量或常量。

示例

```
1. select acos(0.52)
   --------------------
              1.023945
```

返回余弦为 0.52 的角。

注释

- **acos** 是一种数学函数，它返回余弦为指定值的角 （以弧度表示）。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **acos**。

参见

函数	**cos**、 **degrees**、 **radians**

ascii

功能

返回表达式中第一个字符的 ASCII 代码。

语法

ascii(*char_expr*)

参数

char_expr — 是类型为 *char*、 *varchar*、 *nchar* 或 *nvarchar* 的字符型列名、变量或常量表达式。

示例

1. **select au_lname, ascii(au_lname) from authors where ascii(au_lname) < 70**

```
au_lname
------------------------------ -----------
Bennet                                  66
Blotchet-Halls                          66
Carson                                  67
DeFrance                                68
Dull                                    68
```

如果 ASCII 代码小于 70，将返回作者的姓和姓中首字母的 ASCII 代码。

注释

- **ascii** 是一种字符串函数，它返回表达式中第一个字符的 ASCII 代码。
- 如果 *char_expr* 是 NULL，将返回 NULL。
- 有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **ascii**。

参见

函数	char

asin

功能

返回已指定正弦的角 （以弧度表示）。

语法

```
asin(sine)
```

参数

sine — 是角的正弦，表示为 *float*、 *real*、 *double precision* 类型 （或任何可隐式转换为这些类型的数据类型）的列名、变量或常量。

示例

```
1. select asin(0.52)

   --------------------
                0.546851
```

注释

- **asin** 是一种数学函数，它返回正弦为指定值的角 （以弧度表示）。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **asin**。

参见

函数	degrees、 radians、 sin

atan

功能

返回已指定正切的角 （以弧度表示）。

语法

```
atan(tangent)
```

参数

tangent — 是角的正切，表示为 *float*、*real*、*double precision* 类型（或任何可隐式转换为这些类型的数据类型）的列名、变量或常量。

示例

```
1. select atan(0.50)

   --------------------
                0.463648
```

注释

- **atan** 是一种数学函数，它返回正切为指定值的角 （以弧度表示）。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **atan**。

参见

函数	atn2、 degrees、 radians、 tan

atn2

功能

返回已指定正弦和余弦的角 （以弧度表示）。

语法

```
atn2(sine, cosine)
```

参数

sine — 是角的正弦，表示为 *float*、 *real*、 *double precision* 类型 （或任何可隐式转换为这些类型的数据类型）的列名、变量或常量。

cosine — 是角的余弦，表示为 *float*、 *real*、 *double precision* 类型 （或任何可隐式转换为这些类型的数据类型）的列名、变量或常量。

示例

```
1. select atn2(.50, .48)
   --------------------
                0.805803
```

注释

- **atn2** 是一种数学函数，它返回已指定正弦和余弦的角 （以弧度表示）。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **atn2**。

参见

函数	atan、 degrees、 radians、 tan

avg

功能

返回所有 （不同）值的数字平均。

语法

avg([all | distinct] *expression*)

参数

all — 对所有值应用 **avg**。 **all** 是缺省参数。

distinct — 在应用 **avg** 之前消除重复值。 **distinct** 是可选参数。

expression — 可以是列名、常量、函数、任何由算术运算符或逐位运算符连接的列名、常量和函数组合，也可以是子查询。与集合使用的表达式通常是列名。有关详细信息，参见第 3 章 “表达式、标识符和通配符”中的 “表达式”。

示例

1.
```
select avg(advance), sum(total_sales)
from titles
where type = "business"

----------------------- -----------
                6,281.25       30788
```

计算平均预付款以及所有商业书籍总销售额的总和。每个这样的集合函数都将为检索到的所有行生成单个摘要值。

2.
```
select type, avg(advance), sum(total_sales)
from titles
group by type

type
 ------------ ----------------------- -----------
 UNDECIDED                      NULL        NULL
 business                   6,281.25       30788
 mod_cook                   7,500.00       24278
 popular_comp               7,500.00       12875
 psychology                 4,255.00        9939
 trad_cook                  6,333.33       19566
```

当用于 **group by** 子句时，集合函数将为每组 （而非整个表）生成单个值。以上语句将生成每种书籍的摘要值。

3.
```
select pub_id, sum(advance), avg(price)
from titles
group by pub_id
having sum(advance) > $25000 and avg(price) > $15
```

按出版者将 *titles* 表分组，并且只包括那些总预付款超过 $25,000 且书籍平均价格高于 $15 的出版者所形成的组。

```
pub_id
------ -------------------- --------------------
0877              41,000.00                15.41
1389              30,000.00                18.98
```

注释

- **avg** 是一种集合函数，它查找一列中所有值的平均。**avg** 只能用于数字（整数、浮点或货币）数据类型。计算平均时将忽略空值。
- 有关集合函数的一般信息，参见第 2-5 页上的“集合函数”。
- 当用 average 计算整数数据的平均值时，即使列的数据类型是 *smallint* 或 *tinyint*， Adaptive Server 也会将结果当作一个 *int* 值。为避免 DB-Library 程序出现溢出错误，应将平均或求和结果的所有变量声明为类型 *int*。
- 不能对二进制数据类型使用 **avg**()。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **avg**。

参见

函数	max、 min

ceiling

功能

返回大于或等于指定值的最小整数。

语法

ceiling(*value*)

参数

value — 是数据类型为精确数值、近似数值、货币 （或任何可隐式转换为这些类型的数字类型）的列、变量或表达式。

示例

```
1. select ceiling(123.45)

   124

2. select ceiling(-123.45)

   -123

3. select ceiling(1.2345E2)

   24.000000

4. select ceiling(-1.2345E2)

   -123.000000

5. select ceiling($123.45)

   124.00

6. select discount, ceiling(discount) from salesdetail
   where title_id = "PS3333"

   discount
    --------------------- ---------------------
               45.000000             45.000000
               46.700000             47.000000
               46.700000             47.000000
               50.000000             50.000000
```

注释

- **ceiling** 是一种数学函数，它返回大于或等于指定值的最小整数。返回值的数据类型与所提供的值相同。

  对于 *numeric* 和 *decimal* 值，结果的精度与所提供的值相同，而标度为零。

- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **ceiling**。

参见

命令	**set**
函数	**abs**、 **floor**、 **round**、 **sign**

char

功能

返回整数的等值字符。

语法

char(*integer_expr*)

参数

integer_expr — 是 0 到 255 之间的任何整数 （*tinyint*、 *smallint* 或 *int*）列名、变量或常量表达式。

示例

```
1. select char(42)

   -
   *

2. select xxx = char(65)

   xxx
   ---
   A
```

注释

- **char** 是一种字符串函数，它将单字节整数值转换为字符值。（**char** 通常作为 **ascii** 的倒数使用。）
- **char** 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。
- 如果 *char_expr* 是 NULL，将返回 NULL。
- 有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。

用 char 重新格式化输出

- 可以使用并置和 **char**() 值来添加 tab 或回车，从而将输出重新格式化。 **char**(10) 转换为回车； **char**(9) 转换为 tab。

例如：

```
/* just a space */
select title_id +  " " + title from titles where title_id = "T67061"
/* a return */
select title_id + char(10) + title from titles where title_id = "T67061"
/* a tab */
select title_id + char(9) + title from titles where title_id = "T67061"
```

```
-----------------------------------------------------------------------
T67061 Programming with Curses
-----------------------------------------------------------------------
T67061

Programming with Curses
-----------------------------------------------------------------------
T67061      Programming with Curses
```

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **char**。

参见

函数	ascii、 str

charindex

功能

返回表示表达式起始位置的整数。

语法

charindex(*expression1*, *expression2*)

参数

expression — 是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* 或 *nvarchar* 数据， *binary* 或 *varbinary*。

示例

```
1. select charindex("wonderful", notes)
   from titles
   where title_id = "TC3218"

   -----------
           46
```

返回字符表达式 “Wonderful”在 *titles* 表 *notes* 列上的起始位置。

注释

- **charindex** 是一种字符串函数，它在 *expression2* 中搜索首次出现的 *expression1* 并返回表示其起始位置的整数。如果没有找到 *expression1*， **charindex** 将返回 0。
- 如果 *expression1* 包含通配符， **charindex** 就会将它们当作文字。
- 如果 *char_expr* 是 NULL，将返回 NULL。
- 有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **charindex**。

参见

函数	**patindex**

char_length

功能

返回表达式中字符的数量。

语法

char_length(*char_expr*)

参数

char_expr — 是类型为 *char*、 *varchar*、 *nchar* 或 *nvarchar* 的字符型列名、变量或常量表达式。

示例

```
1. select char_length(notes) from titles
   where title_id = "PC9999"

    -----------
             39

2. declare @var1 varchar(20), @var2 varchar(20), @char
   char(20)
   select @var1 = "abcd", @var2 = "abcd    ",
       @char = "abcd"
   select char_length(@var1), char_length(@var2),
   char_length(@char)

    ----------- ----------- -----------
              4           8          20
```

注释

- **char_length** 是一种字符串函数，它返回一个整数，以表示字符表达式或文本值中字符的数量。
- 对于可变长度的列和变量，**char_length** 将返回字符的数量（而不是列或变量的已定义长度）。不能删去可变长度变量中包含的显式尾随空白。对于文字和固定长度的字符列和变量，**char_length** 不删去尾随空白的表达式 （参见示例 2）。
- 对于多字节字符集，表达式中字符的数量通常小于字节的数量；使用 **datalength** 可确定字节的数量。
- 如果 *char_expr* 是 NULL， **char_length** 将返回 NULL。
- 有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **char_length**。

参见

函数	datalength

col_length

功能

返回已定义的列长度。

语法

```
col_length(object_name, column_name)
```

参数

object_name — 是数据库对象 （如表、视图、过程、触发器、缺省值或规则）的名称。该名可以是全限定名 （即，可包括数据库名和所有者名）。必须用引号将其引起来。

column_name — 是列名。

示例

```
1. select x = col_length("titles", "title")

    x
    ----
      80
```

查找 *titles* 表中 *title* 列的长度。“x”在结果中提供列标题。

注释

- **col_length** 是一种系统函数，它返回已定义的列长度。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。
- 要查找每行所存储的数据的实际长度，可使用 **datalength**。
- 对于 *text* 和 *image* 列， **col_length** 将返回 16，这是实际文本页的 *binary(16)* 指针的长度。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **col_length**。

参见

函数	datalength

col_name

功能

返回已指定表 ID 和列 ID 的列的名称。

语法

```
col_name(object_id, column_id[, database_id])
```

参数

object_id — 是一种数字表达式，它表示表、视图或其它数据库对象的对象 ID。它们存储在 *sysobjects* 的 *id* 列中。

column_id — 是一种数字表达式，它表示一列的列 ID。它们存储在 *syscolumns* 的 *colid* 列中。

database_id — 是一种数字表达式，它表示数据库的 ID。它们存储在 *sysdatabases* 的 *db_id* 列中。

示例

```
1. select col_name(208003772, 2)
   ------------------------------
   title
```

注释

- **col_name** 是一种系统函数，它返回列的名称。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **col_name**。

参见

函数	db_id、object_id

compare

功能

用于根据替代归类规则来直接比较两个字符串。

语法

```
compare (char_expression1, char_expression2
   [, {collation_name | collation_ID}])
```

参数

char_expression1 是要与 *char_expression2* 比较的字符表达式。

char_expression2 是 *char_expression1* 要与其比较的字符表达式。

char_expression1 和 *char_expression2* 可以是以下类型之一：

- 字符类型 （*char*、 *varchar*、 *nchar* 或 *nvarchar*）
- 字符变量，或
- 用单引号或双引号引起来的常量字符表达式

collation_name 是引起来的字符串或字符变量，用于指定要使用的归类。

collation_ID 是一个整数常量或变量，用于指定要使用的归类。

注释

- **compare** 函数根据所选归类规则返回以下值：
  - 1 — 表示 *char_expression1* 大于 *char_expression2*
  - 0 — 表示 *char_expression1* 等于 *char_expression2*
  - -1 — 表示 *char_expression1* 小于 *char_expression2*
- *char_expression1* 和 *char_expression2* 必须均为服务器缺省字符集中编码的字符。
- *char_expression1* 或 *char_expression2* 均可为空字符串：
  - 如果 *char_expression2* 是空字符串，函数将返回 1。
  - 如果两个字符串都为空，则它们相等，函数返回 0 值。
  - 如果 *char_expression1* 是空字符串，函数返回 -1。

compare 函数并不认为空字符串与仅包含空格的字符串相等，而 Adaptive Server 认为如此。**compare** 使用 **sortkey** 函数生成用于比较的归类键。因此，真正为空的字符串、包含一个空格的字符串或包含两个空格的字符串在比较中是不相等的。

- 如果 *char_expression1* 或 *char_expression2* 为 NULL，则结果为 NULL。
- 如果没有指定 *collation_name* 的值，**compare** 采取二进制归类。
- 如果没有指定 *collation_ID* 的值，**compare** 采取二进制归类。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **compare**。

参见

函数	sortkey

convert

功能

返回已转换为另一种数据类型或其它 *datetime* 显示格式的指定值。

语法

```
convert (datatype [(length) | (precision[, scale])]
   [null | not null], expression [, style])
```

参数

datatype — 是要将表达式转换成的系统数据类型 （例如， *char*(10)、*varbinary*(50) 或 *int*）。不能使用用户定义的数据类型。

如果数据库中已启用 Java， *datatype* 也可能是当前数据库中的 Java-SQL 类。

length — 是一种可选参数，可用于 *char*、*nchar*、*varchar*、*nvarchar*、*binary* 和 *varbinary* 数据类型。如果您不提供长度， Adaptive Server 将把字符类型的数据截断到 30 个字符，把二进制类型的数据截断到 30 字节。字符和二进制数据可以具有的最大长度是 255 个字节。

precision — 是 *numeric* 或 *decimal* 数据类型中有效位的个数。对于*浮点*数据类型，精度是指尾数中有效二进制位的个数。如果您不提供精度， Adaptive Server 将对 *numeric* 和 *decimal* 数据类型使用缺省精度 18。

scale — 是 *numeric* 或 *decimal* 数据类型中小数点右侧的位数。如果您不提供标度， Adaptive Server 将使用缺省标度 0。

null | **not null** — 指定结果表达式的可为空性。如果您不提供 **null** 或 **not null**，转换后的结果将与表达式具有相同的可为空性。

expression — 是从一种数据类型或数据格式转换为另一种数据类型或数据格式的值。

如果数据库中已启用 Java， *expression* 可以是将要转换为 Java-SQL 类的值。

style — 是用于已转换数据的显示格式。如果将 *money* 或 *smallmoney* 数据转换为字符类型，则会采用 *style* 1，在每 3 位数后显示一个逗号。

如果将 *datetime* 或 *smalldatetime* 数据转换为字符类型，将使用表 2-4 中的样式编号来指定显示格式。最左列的值显示 2 位数的年份 (yy)。对于 4 位数的年份 (yyyy)，可添加 100，或使用中间列中的值。

表 2-4：日期/时间信息显示格式

不含世纪 (yy)	含世纪 (yyyy)	输出
无	0 或 100	*mon dd yyyy hh:mi*AM （或 PM）
1	101	*mm/dd/yy*
2	102	*yy.mm.dd*
3	103	*dd/mm/yy*
4	104	*dd.mm.yy*
5	105	*dd-mm-yy*
6	106	*dd mon yy*
7	107	*mon dd, yy*
8	108	*hh:mm:ss*
无	9 或 109	*mon dd yyyy hh:mi:ss:mmm*AM （或 PM）
10	110	*mm-dd-yy*
11	111	*yy/mm/dd*
12	112	*yymmdd*

缺省值 （*style* 0 或 100）和 *style* 9 或 109 总是返回世纪 (yyyy)。如果从 *smalldatetime* 转换为 *char* 或 *varchar*，包括秒或毫秒的样式将在这些位置上显示零。

示例

```
1. select title, convert(char(12), total_sales)
   from titles
2. select title, total_sales
   from titles
   where convert(char(20), total_sales) like "1%"
```

3. **select convert(char(12), getdate(), 3)**

   将当前日期转换为样式 “3” *dd/mm/yy*。

4. **select convert(varchar(12), pubdate, 3) from titles**

   如果值 *pubdate* 为空，则必须使用 *varchar* 而不是 *char*，否则就会出错。

5. **select convert(integer, 0x00000100)**

   返回字符串 “0x00000100”的等值整数。结果会因平台的不同而变化。

6. **select convert (binary, 10)**

   将特定于平台的位模式当作 Sybase 二进制类型返回。

7. **select convert(bit, $1.11)**

   返回 1， $1.11 的等值位字符串。

8. **select title, convert (char(100) not null, total_sales) into #tempsales from titles**

   用数据类型为 *char(100)* 的 *total_sales* 创建 *#tempsales*，且不允许空值。即使将 *titles.total_sales* 定义为允许空值， *#tempsales* 也将使用不允许空值的 *#tempsales.total_sales* 来创建。

注释

- **convert** 是一种数据类型转换函数。为便于显示，它可在多种数据类型之间转换并将日期 / 时间及货币数据重新格式化。
- 有关数据类型转换的详细信息，参见第 2-10 页上的 “数据类型转换函数”。
- 当 **convert()** 的参数超出该函数的定义范围时，该函数就会产生域错误。这种错误应极少发生。
- 使用 **null** 或 **not null** 可指定目标列的可为空性。尤其是，可使用 **select into** 来创建一个新表，然后在源表中更改现有列的数据类型和可为空性 （参见上面的示例 8）。
- 可以使用 **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）。

涉及 Java 类的转换

- 如果数据库中已启用 Java，就可按下列方法使用 **convert** 更改数据类型：
  - 将 Java 对象类型转换为 SQL 数据类型。
  - 将 SQL 数据类型转换为 Java 类型。
  - 如果表达式 （源类）的编译时数据类型是目标类的子类或超类，就会将安装在 Adaptive Server 上的任何 Java-SQL 类转换为安装在 Adaptive Server 上的任何其它 Java-SQL 类。

  转换结果与当前数据库相关联。
- 要查看允许的数据类型映射的列表并详细了解涉及 Java 类的数据类型转换，参见 *Adaptive Server Enterprise 中的 Java*。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **convert**。

参见

数据类型	用户定义的数据类型
函数	**hextoint**、 **inttohex**

COS

功能

返回指定角的余弦。

语法

cos(*angle*)

参数

angle — 是任意近似数值类型 （*float*、 *real* 或 *double precision*）的列名、变量或常量表达式。

示例

```
1. select cos(44)
    0.999843
```

注释

- **cos** 是一种数学函数，它返回指定角 （以弧度表示）的余弦。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **cos**。

参见

函数	**acos**、 **degrees**、 **radians**、 **sin**

cot

功能

返回指定角的余切。

语法

```
cot(angle)
```

参数

angle — 是任意近似数值类型 （*float*、 *real* 或 *double precision*）的列名、变量或常量表达式。

示例

```
1. select cot(90)

   --------------------
              -0.501203
```

注释

- **cos** 是一种数学函数，它返回指定角 （以弧度表示）的余切。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **cot**。

参见

函数	degrees、 radians、 tan

count

功能

返回 （不同）非空值的数量或所选行的数量。

语法

```
count([all | distinct] expression)
```

参数

all — 对所有值应用 **count**。 **all** 是缺省参数。

distinct — 在应用 **count** 之前消除重复值。 **distinct** 是可选参数。

expression — 可以是列名、常量、函数、任何由算术运算符或逐位运算符连接的列名、常量和函数组合，也可以是子查询。与集合使用的表达式通常是列名。有关详细信息，参见第 3 章 “表达式、标识符和通配符”中的 “表达式”。

示例

1.
```
select count(distinct city)
from authors
```

   查找作者居住的不同城市数量。

2.
```
select type
from titles
group by type
having count(*) > 1
```

   列出 *titles* 表中的类型，但是消除只包含一本书或不包含任何书的类型。

注释

- **count** 是一种集合函数，它查找列中非空值的数量。有关集合函数的一般信息，参见第 2-5 页上的 “集合函数”。
- 当指定 **distinct** 后， **count** 将查找唯一非空值的数量。除了 *text* 和 *image* 之外， **count** 可用于所有数据类型。进行计数时将忽略空值。

- 对于空表、仅包含空值的列和仅包含空值的组，**count**(*column_name*) 返回的值为 0。
- **count(*)** 查找行数。**count(*)** 不带任何参数，并且不能同 **distinct** 一起使用。无论是否含有空值，所有行均被计数。
- 当连接多个表时，应在 **select list** 中包括 **count(*)**，以生成连接结果中的行数。如果目标是计算一个表中与标准相匹配的行数，可使用 **count**(*column_name*)。
- **count()** 可在子查询中用来进行存在性检查。例如：

```
select * from tab where 0 <
    (select count(*) from tab2 where ...)
```

然而，因为 **count()** 对所有匹配值进行计数，**exists** 或 **in** 可更快地返回结果。例如：

```
select * from tab where exists
    (select * from tab2 where ...)
```

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **count**。

参见

命令	compute Clause、group by 和 having 子句、select、where 子句

curunreservedpgs

功能

返回指定磁盘区段中的可用页数。

语法

curunreservedpgs(*dbid, lstart,unreservedpgs*)

参数

dbid — 是数据库的 ID。它们存储在 *sysdatabases* 的 *db_id* 列中。

lstart — 是要返回其页的磁盘区段中的一页。

unreservedpgs — 是在 *dbtable* 当前对于所请求的数据库无效时将返回的缺省值。

示例

```
1. select db_name(dbid), d.name,
       curunreservedpgs(dbid, lstart, unreservedpgs)
       from sysusages u, sysdevices d
       where d.low <= u.size + vstart
           and d.high >= u.size + vstart -1
           and d.status &2 = 2
```

```
master           master        184
master           master        832
tempdb           master        464
tempdb           master       1016
tempdb           master        768
model            master        632
sybsystemprocs    master      1024
pubs2             master      248
```

返回数据库名称、设备名和每个设备区段中的未保留页数。

```
2. select curunreservedpgs (dbid, sysusages.lstart, 0)
```

显示从 *sysusages.lstart* 开始的 *dbid* 段上的可用页数。

注释

- **curunreservedpgs** 是一种系统函数，它返回磁盘区段中的可用页数。有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。
- 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。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **curunreservedpgs**。

参见

函数	db_id、 lct_admin

data_pgs

功能

返回指定表或索引所用的页数。

语法

```
data_pgs(object_id,
   {data_oam_pg_id | index_oam_pg_id})
```

参数

object_id — 是表、视图或其它数据库对象的对象 ID。它们存储在 *sysobjects* 的 *id* 列中。

data_oam_pg_id — 是数据 OAM 页的页 ID，它存储在 *sysindexes* 的 *doampg* 列中。

index_oam_pg_id — 是索引 OAM 页的页 ID，它存储在 *sysindexes* 的 *ioampg* 列中。

示例

1.
```
select sysobjects.name,
Pages = data_pgs(sysindexes.id, doampg)
from sysindexes, sysobjects
where sysindexes.id = sysobjects.id
    and sysindexes.id > 100
    and (indid = 1 or indid = 0)
```

   估计用户表（其对象 ID 大于 100）所用的数据页数。 *indid* 为 0 表示没有集群索引的表； *indid* 为 1 表示有集群索引的表。此例不包括非集群索引或文本链。

2.
```
select sysobjects.name,
Pages = data_pgs(sysindexes.id, ioampg)
from sysindexes, sysobjects
where sysindexes.id = sysobjects.id
    and sysindexes.id > 100
     and (indid > 1)
```

   估计用户表（其对象 ID 大于 100）、非集群索引和页链所用的数据页数。

注释

- **data_pgs** 是一种系统函数，它返回表 (*doampg*) 或索引 (*ioampg*) 所用的页数。在对 *sysindexes* 表运行查询时必须使用这一函数。有关系统函数的详细信息，参见第 2-20 页上的 “系统函数”。
- **data_pgs** 仅用于当前数据库中的对象。
- 结果不包括用于内部结构的页。要查看表、集群索引和内部结构的页数报告，可使用 **used_pgs**。

结果的精确性

- 当用于事务日志 (*syslogs*) 时，其结果可能不精确，最多会偏差 16 页。

错误

- 如果出现下列任何情况， **data_pgs** 将返回 0，而不是返回错误：
  - *sysobjects* 中不存在 *object_id*
  - *control_page_id* 不属于 *object_id* 所指定的表
  - *object_id* 为 -1
  - *page_id* 为 -1

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **data_pgs**。

使用的表

sysindexes、 *syspartitions*

参见

函数	object_id、 rowcnt、 used_pgs
系统过程	sp_spaceused

datalength

功能

返回指定列或字符串的实际长度 （以字节表示）。

语法

datalength(*expression*)

参数

expression — 是列名、变量、常量表达式，或任意求值结果为单个值的列名、变量、常量表达式组合。它可以是任何数据类型。*expression* 通常是列名。如果 *expression* 是字符串常量，则必须用引号引起来。

示例

```
1. select Length = datalength(pub_name)
   from publishers

   Length
   -----------
            13
            16
             20
```

查找 *publishers* 表中 *pub_name* 列的长度。

注释

- **datalength** 是一种系统函数，它返回 *expression* 的长度 （以字节表示）。
- **datalength** 查找每行存储的数据的实际长度。**datalength** 对于 *varchar*、 *varbinary*、 *text* 和 *image* 数据类型非常有用，因为这些数据类型可以存储可变长度 （但不存储尾随空白）。如果声明 *char* 值可为空，Adaptive Server 就会在内部将其存储为 *varchar*。对于所有其它数据类型， **datalength** 将报告它们的已定义长度。
- 任何 NULL 数据的 **datalength** 都将返回 NULL。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **datalength**。

参见

函数	**char_length**、 **col_length**

dateadd

功能

返回向指定日期添加给定数量的年、季度、小时和其它日期分量后所得的日期。

语法

dateadd(*date_part*, *integer*, *date*)

参数

date_part — 是日期分量或缩写。有关 Adaptive Server 可识别的日期分量和缩写的列表，参见第 2-17 页上的 “日期分量”。

numeric — 是一种整数表达式。

date — 可以是函数 **getdate**、具有可接受日期格式的字符串、求值结果为有效日期格式的表达式，或者是 *datetime* 列的名称。

示例

```
1. select newpubdate = dateadd(day, 21, pubdate)
   from titles
```

在 *titles* 表中所有书的出版日期推迟了 21 天的情况下，显示新的出版日期。

注释

- **dateadd** 是一种日期函数，它向指定日期添加时间间隔。有关日期函数的详细信息，参见第 2-16 页上的 “日期函数”。
- **dateadd** 带有三个参数：日期分量，数量和日期。结果是等于日期加日期分量数的 *datetime* 值。

  如果数据参数是 *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。

- *datetime* 数据类型只能用于 1753 年 1 月 1 日之后的日期。*datetime* 值必须用单引号或双引号引起来。对于更早的日期，应使用 *char*、 *nchar*、 *varchar* 或 *nvarchar*。 Adaptive Server 可识别多种多样的日期格式。有关详细信息，参见第 1 章 “系统和用户定义的数据类型”中的 “用户定义的数据类型”和第 2 章 “Transact-SQL 函数”中的 “Transact-SQL 函数”中的 “数据类型转换函数”。

  如有必要 （例如在比较字符值和 *datetime* 值时）， Adaptive Server 会自动在字符和 *datetime* 值之间进行转换。

- 在 **dateadd** 中使用日期分量 **weekday** 或 **dw** 是不合逻辑的，将产生虚假结果。而应使用 **day** 或 **dd**。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **dateadd**。

参见

命令	select、 where 子句
数据类型	“日期和时间数据类型”
函数	datediff、 datename、 datepart、 getdate

datediff

功能

返回两个日期之间的差值。

语法

datediff(*datepart*, *date1*, *date2*)

参数

datepart — 是日期分量或缩写。有关 Adaptive Server 可识别的日期分量和缩写的列表，参见第 2-17 页上的 “日期分量”。

date1 — 可以是函数 **getdate**、具有可接受日期格式的字符串、求值结果为有效日期格式的表达式，或者是 *datetime* 列的名称。

date2 — 可以是函数 **getdate**、具有可接受日期格式的字符串、求值结果为有效日期格式的表达式，或者是 *datetime* 或 *smalldatetime* 列的名称。

示例

1.
```
select newdate = datediff(day, pubdate, getdate())
from titles
```

   此查询查找在 *pubdate* 和当前日期 （用 **getdate** 函数获取）之间经历的天数。

注释

- **datediff** 是一种日期函数，它计算两个指定日期间的日期分量数。有关日期函数的详细信息，参见第 2-16 页上的 “日期函数”。
- **datediff** 带有三个参数。第一个是日期分量。第二个和第三个是日期。结果是带符号的整数值 （以日期分量表示），它等于 *date2* - *date1*。
- **datediff** 产生数据类型为 *int* 的结果，如果结果大于 2,147,483,647，则会导致错误。对于毫秒，约为 24 天， 20:31.846 小时。对于秒，是 68 年， 19 天， 3:14:07 小时。
- 当结果不是日期分量的偶数倍时，总会截断 （而不是舍入） **datediff** 结果。例如，如果使用 **hour** 作为日期分量，“4:00AM” 和 “5:50AM” 的差值就是 1。

当使用 **day** 作为日期分量时， **datediff** 将计算两个指定时间之间的午夜数。例如， 1992 年 1 月 1 日 23:00 和 1992 年 1 月 2 日 01:00 之间的差值是 1； 1992 年 1 月 1 日 00:00 和 1992 年 1 月 1 日 23:59 之间的差值是 0。

- **month** 日期分量计算两个日期之间月份第一天的数目。例如， 1 月 25 日与 2 月 2 日之间的差值是 1； 1 月 1 日与 1 月 31 日之间的差值是 0。
- 在 **datediff** 中使用日期分量 **week** 时，将得到两个日期之间（含第二个日期，但不含第一个日期）周日的数目。例如， 1 月 4 日周日与 1 月 11 日周日之间的周数是 1。
- 如果使用 *smalldatetime* 值，为便于计算，将在内部把它们转换为 *datetime* 值。为了计算差值， *smalldatetime* 值中的秒和毫秒会被自动设置为 0。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **datediff**。

参见

数据类型	“日期和时间数据类型”
命令	**select**、 **where 子句**
函数	**dateadd**、 **datename**、 **datepart**、 **getdate**

datename

功能

返回 *datetime* 值中指定分量的名称。

语法

```
datename (datepart, date)
```

参数

datepart — 是日期分量或缩写。有关 Adaptive Server 可识别的日期分量和缩写的列表，参见第 2-17 页上的 “日期分量”。

date — 可以是函数 **getdate**、具有可接受日期格式的字符串、求值结果为有效日期格式的表达式，或者是 *datetime* 或 *smalldatetime* 列的名称。

示例

```
1. select datename(month, getdate())
   November
```

此例假定当前日期是 1998 年 11 月 20 日。

注释

- **datename** 是一种日期函数，它以字符串形式返回 *datetime* 或 *smalldatetime* 值中指定分量的名称 （例如 “June”）。如果结果是数字 （例如 “23”日），该函数仍返回字符串。
- 有关日期函数的详细信息，参见第 2-16 页上的 “日期函数”。
- 如果在 **datename** 中使用日期分量 **weekday** 或 **dw**，将返回星期值（星期日、星期一等）。
- 由于 *smalldatetime* 仅对于分钟是精确的，所以当在 **datename** 中使用 *smalldatetime* 时，秒和毫秒总是为 0。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **datename**。

参见

数据类型	“日期和时间数据类型”
命令	**select**、 **where 子句**
函数	**dateadd**、 **datediff**、 **datepart**、 **getdate**

datepart

功能

返回 *datetime* 值中指定分量的整数值。

语法

datepart(*date_part*, *date*)

参数

date_part — 是日期分量。表 2-5 列出了 **datepart** 可识别的日期分量、缩写以及可接受的值。

表 2-5：日期分量及其值

日期分量	缩写	值
year	yy	1753 – 9999 （对于 *smalldatetime* 为 2079）
quarter	qq	1 – 4
month	mm	1 – 12
week	wk	1 – 54
day	dd	1 – 31
dayofyear	dy	1 – 366
weekday	dw	1 – 7 （周日 — 周六）
hour	hh	0 – 23
minute	mi	0 – 59
second	ss	0 – 59
millisecond	ms	0 – 999
calweekofyear	cwk	1 – 53
calyearofweek	cyr	1753 – 9999
caldayofweek	cdw	1 – 7

如果输入只有 2 位的年份， <50 为下一世纪 （“25”表示 “2025”）， >=50 为本世纪 （“50”表示 “1950”）。

毫秒前可以带一个冒号或句号。如果带冒号，数字就表示千分之多少秒。如果带句号，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。

date — 可以是函数 **getdate**、具有可接受日期格式的字符串、求值结果为有效日期格式的表达式，或者是 *datetime* 或 *smalldatetime* 列的名称。

示例

```
1. select datepart(month, getdate())

   -----------
            11
```

此例假定当前日期是 1995 年 11 月 25 日。

```
2. select datepart(year, pubdate) from titles where
   type = "trad_cook"

    -----------
           1990
           1985
           1987
```

```
3. select datepart(cwk,'1993/01/01')

   -----------
             53
```

```
4. select datepart(cyr,'1993/01/01')

   -----------
           1992
```

```
5. select datepart(cdw,'1993/01/01')

   -----------
              5
```

注释

- **datepart** 是一种日期函数，返回 *datetime* 值中指定部分的整数值。有关日期函数的详细信息，参见第 2-16 页上的 “日期函数”。
- **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 **caldayorweek** 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。

- 如果在 **datepart** 中使用日期分量 **weekday** 或 **dw**，将返回相应的数字。与星期名称相对应的编号取决于 **datefirst** 的设置。某些语言的缺省设置（包括 us_english）会生成 Sunday=1、 Monday=2，依此类推；而其它语言的缺省设置则生成 Monday=1、Tuesday=2，依此类推。可在每次会话时用 **set datefirst** 更改缺省行为。
- **calweekofyear** 可缩写为 **cwk**，它返回一年中周的序数。 **calyearofweek** 可缩写为 **cyr**，它返回一周开始的年份。 **caldayofweek** 可缩写为 **cdw**，它返回一周中日的序数。 **calweekofyear**、 **calyearofweek** 和 **caldayofweek** 不能用作 **dateadd**、 **datediff** 和 **datename** 的日期分量。
- 由于 *smalldatetime* 仅对于分钟是精确的，所以在 **datepart** 中使用 *smalldatetime* 时，秒和毫秒总为 0。
- 语言设置将影响星期日期分量的值。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **datepart**。

参见

数据类型	“日期和时间数据类型”
命令	**select**、 **where 子句**
函数	**dateadd**、 **datediff**、 **datename**、 **getdate**

db_id

功能

返回指定数据库的 ID 号。

语法

```
db_id(database_name)
```

参数

database_name — 是数据库的名称。 *database_name* 必须是字符表达式。如果它是常量表达式，则必须用引号将其引起来。

示例

```
1. select db_id("sybsystemprocs")
   ------
   4
```

注释

- **db_id** 是一种系统函数，它返回数据库的 ID 号。
- 如果未指定 *database_name*， **db_id** 将返回当前数据库的 ID 号。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **db_id**。

参见

函数	db_name、 object_id

db_name

功能

返回已指定 ID 号的数据库的名称。

语法

db_name([*database_id*])

参数

database_id — 是数据库 ID 的数字表达式 （存储在 *sysdatabases.dbid* 中）。

示例

1. **select db_name()**

   返回当前数据库的名称。

2. **select db_name(4)**

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

注释

- **db_name** 是一种系统函数，它返回数据库的名称。
- 如果未提供 *database_id*， **db_name** 将返回当前数据库的名称。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **db_name**。

参见

函数	db_id、 col_name、 object_name

degrees

功能

返回具有指定弧度数的角的大小 （以度表示）。

语法

degrees(*numeric*)

参数

numeric — 是要转换为度的数字 （以弧度表示）。

示例

```
1. select degrees(45)

   -----------
          2578
```

注释

- **degrees** 是一种数学函数，它将弧度转换为度。其结果与数字表达式的类型相同。

  对于数字和小数表达式，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。

- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **degrees**。

参见

函数	radians

difference

功能

返回两个 **soundex** 值之间的差值。

语法

difference(*char_expr1, char_expr2*)

参数

char_expr1 — 是一个类型为 *char*、 *varchar*、 *nchar* 或 *nvarchar* 的字符型列名、变量或常量表达式。

char_expr2 — 是另一个类型为 *char*、 *varchar*、 *nchar* 或 *nvarchar* 的字符型列名、变量或常量表达式。

示例

```
1. select difference("smithers", "smothers")

   ---------
   4
2. select difference("smothers", "brothers")

   ---------
   2
```

注释

- **difference** 是一种字符串函数，它返回表示两个 **soundex** 值之间差值的整数。
- **difference** 函数比较两个字符串并评价二者之间的相似性，最后返回从 0 到 4 的值。最佳匹配值是 4。

  字符串值必须由有效单字节或双字节罗马字母的邻接序列组成。
- 如果 *char_expr1* 或 *char_expr2* 是 NULL，将返回 NULL。
- 有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **difference**。

参见

函数	soundex

exp

功能

返回对常数 e 求指定次幂所得的值。

语法

exp(*approx_numeric*)

参数

approx_numeric — 是任何近似数值类型 （*float*、 *real* 或 *double precision*）的列名、变量或常量表达式。

示例

```
1. select exp(3)
   --------------------
                20.085537
```

注释

- **exp** 是一种数学函数，它将返回指定值的幂值。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **exp**。

参见

函数	log、 log10、 power

floor

功能

返回小于或等于指定值的最大整数。

语法

floor(*numeric*)

参数

numeric — 是任何精确的数值 （*numeric*、 *dec*、 *decimal*、 *tinyint*、 *smallint* 或 *int*），近似数值 （*float*、 *real* 或 *double precision*），或 *money* 列、变量、常数表达式，或上述数据类型的组合形式。

示例

```
1. select floor(123)

   -----------
           123
2. select floor(123.45)

   -------
       123
3. select floor(1.2345E2)

   --------------------
            123.000000
4. select floor(-123.45)

   -------
      -124
5. select floor(-1.2345E2)

   --------------------
           -124.000000
6. select floor($123.45)

   ------------------------
                    123.00
```

注释

- **floor** 是数学函数，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。

  对于数字和小数表达式，其结果的精度与该表达式的精度相同，小数位数是零。

- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **floor**。

参见

函数	**abs**、 **ceiling**、 **round**、 **sign**

getdate

功能

返回当前的系统日期和时间。

语法

```
getdate()
```

参数

无。

示例

```
1. select getdate()
   Nov 25 1995 10:32AM
2. select datepart(month, getdate())
   1
3. select datename(month, getdate())
   November
```

以上示例采用 November 25, 1995, 10:32 a.m 作为当前日期。

注释

- **getdate** 是一种日期函数，它返回当前的系统日期和时间。
- 有关日期函数的详细信息，参见第 2-16 页上的 “日期函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **getdate**。

参见

数据类型	“日期和时间数据类型”
函数	**dateadd**、 **datediff**、 **datename**、 **datepart**

hextoint

功能

返回指定的十六进制字符串的等价整数，其值与平台无关。

语法

hextoint (*hexadecimal_string*)

参数

hexadecimal_string — 是要转换为整数的十六进制值。它必须是字符类型列或变量的名，或用引号引起来的有效十六进制字符串 （带有或不带有 “0x”前缀都可以）。

示例

```
1. select hextoint ("0x00000100")
```

返回十六进制字符串 0x00000100 的等值整数。其结果总是 256，无论它所处的平台如何。

注释

- **hextoint** 是一种数据类型转换函数，它返回独立于平台的十六进制字符串的等值整数。
- 使用 **hextoint** 函数可进行独立于平台的数据转换，将十六进制数据转换为整数。 **hextoint** 接受包含在引号中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。无论执行的平台如何，该函数始终返回给定十六进制字符串的相同整数等价值。
- 有关数据类型转换的详细信息，参见第 2-10 页上的 “数据类型转换函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **hextoint**。

参见

函数	**convert**、 **inttohex**

host_id

功能

返回客户端进程的主机进程 ID。

语法

```
host_id()
```

参数

无。

示例

```
1. select host_id()
   ------------------------------
   24711
```

注释

- **host_id** 是一种系统函数，它返回客户端进程 （而不是服务器进程）的主机进程 ID。
- 有关系统函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **host-id**。

参见

函数	host_name

host_name

功能

返回当前客户端进程的主机计算机名。

语法

```
host_name()
```

参数

无。

示例

```
1. select host_name()
   ------------------------------
   violet
```

注释

- **host_name** 是一种系统函数，它返回当前客户端进程 （而不是服务器进程）的主机计算机名。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **host-name**。

参见

函数	host_id

index_col

功能

返回指定表或视图中带索引的列的名称。

语法

```
index_col (object_name, index_id, key_# [, user_id])
```

参数

object_name — 是表或视图的名称。该名可以是全限定名 （即，可包括数据库和所有者名）。必须用引号将其引起来。

index_id 是 *object_name* 的索引的编号。此号与 *sysindexes.indid* 的值相同。

key_# — 是索引中的键。对于集群索引，此值界于 1 和 *sysindexes.keycnt* 之间；对于非集群索引，此值界于 1 和 *sysindexes.keycnt*+1 之间。

user_id — 是 *object_name* 的所有者。如果不指定 *user_id*，缺省情况下，它是调用者的用户 ID。

示例

1.
```
declare @keycnt integer
select @keycnt = keycnt from sysindexes
     where id = object_id("t4")
     and indid = 1
while @keycnt > 0
 begin
     select index_col("t4", 1, @keycnt)
     select @keycnt = @keycnt - 1
end
```

查找表 *t4* 的集群索引中的键名。

注释

- **index_col** 是一种系统函数，它返回带索引的列的名称。
- 如果 *object_name* 不是表名或视图名， **index_col** 将返回 NULL。
- 有关系统函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **index_col**。

参见

函数	object_id
系统过程	sp_helpindex

index_colorder

功能

返回列的顺序。

语法

```
index_colorder (object_name, index_id, key_#
   [, user_id])
```

参数

object_name — 是表或视图的名称。该名可以是全限定名（即，可包括数据库和所有者名）。必须用引号将其引起来。

index_id 是 *object_name* 的索引的编号。此号与 *sysindexes.indid* 的值相同。

key_# — 是索引中的键。有效值是 1 和索引中键的个数。键数存储在 *sysindexes.keycnt* 中。

user_id — 是 *object_name* 的所有者。如果不指定 *user_id*，缺省情况下，它是调用者的用户 ID。

示例

```
1. select name, index_colorder("sales", indid, 2)
   from sysindexes
   where id = object_id ("sales")
   and indid > 0

   name
   ------------------------ ------------------------
   salesind                 DESC
```

返回 DESC，因为表 *sales* 的索引 *salesind* 是降序排列的。

注释

- **index_colorder** 是系统函数，对升序的列返回 “ASC”，对降序的列返回 “DESC”。
- **index_colorder** 返回 NULL，只要 *object_name* 不是表名，或只要 *key_#* 不是有效的键数。
- 有关系统函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **index_colorder**。

inttohex

功能

返回指定整数的等值十六进制数，与平台无关。

语法

```
inttohex (integer_expression)
```

参数

integer_expression — 是要转换为十六进制字符串的整数值。

示例

```
1. select inttohex (10)

   --------
   0000000A
```

注释

- **inttohex** 是数据类型转换函数，返回整数的等值十六进制字符串（无 “0x” 前缀），与平台无关。
- 使用 **inttohex** 函数可进行整数到十六进制字符串的独立于平台的转换。**inttohex** 接受任何其值为整数的表达式。无论在何种平台上执行，该函数对指定表达式始终返回同一等值十六进制数。
- 有关数据类型转换的详细信息，参见第 2-10 页上的 “数据类型转换函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **inttohex**。

参见

函数	convert、 hextoint

isnull

功能

当 *expression1* 求值为 NULL 时，用 *expression2* 中指定的值进行替换。

语法

```
isnull(expression1, expression2)
```

参数

expression — 是列名、变量、常数表达式，或上述各项的组合，其求值结果为单个值。它可以是任何数据类型。 *expression* 通常是列名。如果 *expression* 是字符串常数，则必须包含在引号中。

示例

1.
```
select isnull(price,0)
from titles
```

返回 *titles* 表中的所有行，并将 *price* 中的所有空*值* 替换为 0。

注释

- **isnull** 是一种系统函数，当 *expression1* 求值为 NULL 时，用 *expression2* 中指定的值替换该值。有关系统函数的一般信息，参见第 2-19 页上的 “字符串函数”。
- 表达式的数据类型必须隐式转换，或使用 **convert** 函数进行转换。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **isnull**。

参见

函数	convert

is_sec_service_on

功能

安全服务处于活动状态时为 “1”；否则为 “0”。

语法

is_sec_service_on(*security_service_nm*)

参数

security_service_nm — 是安全服务的名称。

示例

```
1. select is_sec_service_on("unifiedlogin")
```

注释

- 使用 **is_sec_service_on** 来确定给定的安全服务在会话期间是否处于活动状态。
- 要查找安全服务中的有效名称，请运行以下查询：

  ```
  select * from syssecmechs
  ```

  available_service 列显示受 Adaptive Server 支持的安全服务。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **is_sec_service_on**。

参见

函数	show_sec_services

lct_admin

功能

管理最后机会阈值。

返回最后机会阈值的当前值。

中止已经到达其最后机会阈值的事务日志中的事务。

语法

```
lct_admin({{"lastchance" | "logfull" }, database_id
   |"reserve", {log_pages | 0 }
   | "abort", process-id [, database-id]})
```

参数

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。

database_id — 指定数据库。

reserve — 获取最后机会阈值的当前值或转储指定大小的事务日志所需的日志页数。

log_pages — 是为其确定最后机会阈值的页数。

0 — 返回最后机会阈值的当前值。在带有分离的日志和数据段的数据库中，最后机会阈值的大小不会动态改变。根据事务日志的大小，它有固定值。在带有混合日志和数据段的数据库中，最后机会阈值会动态改变。

abort — 中止事务日志已到达其最后机会阈值的数据库中的事务。只有处于 “日志挂起”模式的事务可以被中止。

process-id — 处于日志挂起模式的进程的 ID (*spid*)。当进程在到达其最后机会阈值 (LCT) 的事务日志中有打开的事务时，它被置于日志挂起模式下。

database-id — 事务日志已到达其 LCT 的数据库的 ID。如果 *process-id* 为 **0**，则在指定数据库中所有打开的事务都将被终止。

示例

1. `select lct_admin("lastchance", 1)`

   此语句将为 *dbid* 是 1 的数据库创建日志段最后机会阈值。它返回新阈值所在的页号。如果已有先前的最后机会阈值，它将被替换。

2. `select lct_admin("logfull", 6)`

   如果已超过 *db_id* 为 6 的数据库的最后机会阈值，则返回 1，否则返回 0。

3. `select lct_admin("reserve", 64)`

   ```
   -----------
            16
   ```

   计算并返回在包含 64 页的日志中成功转储事务日志所需的日志页数。

4. `select lct_admin("reserve",0)`

   返回发出命令的数据库中事务日志的当前最后机会阈值。

5. `select lct_admin("abort", 83)`

   中止属于进程 83 的事务。此进程必须处于日志挂起模式。只终止已经到达其 LCT 的事务日志中的事务。

6. `select lct_admin("abort", 0, 5)`

   中止数据库 ID 为 5 的数据库中所有打开的事务。

   这种格式将唤醒在日志段的最后机会阈值时可能挂起的所有进程。

注释

- **lct_admin** 是一种系统函数，它用于管理日志段的最后机会阈值。有关系统函数的一般信息，参见第 2-19 页上的 “字符串函数”。
- 如果 **lct_admin ("lastchance",** *dbid***)** 返回了零，则日志不在此数据库的单独段上，故而最后机会阈值不存在。
- 每当用单独的日志段来创建数据库时，服务器将创建一个默认的最后机会阈值，默认调用 **sp_thresholdaction**。即使在服务器上根本没有称为 **sp_thresholdaction** 的过程，亦是如此。

  当您的日志超过了最后机会阈值时， Adaptive Server 将挂起活动，然后尝试调用 **sp_thresholdaction**，接着发现它不存在。此时将生成一个错误，随后挂起进程，直到日志可被截断为止。
- 要终止已到达其 LCT 的事务日志中最早打开的事务，输入启动事务的进程的 ID。

- 要终止已到达其 LCT 的事务日志中所有打开的事务，输入 **0** 作为 *process_id*，并在 *database-id* 参数中指定数据库 ID。
- 详细信息，参见*系统管理指南*。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

只有系统管理员可以执行 **lct_admin abort**。所有的用户都可以执行其它 **lct_admin** 选项。

参见

命令	**dump transaction**
函数	**curunreservedpgs**
系统过程	**sp_addthreshold**、 **sp_dropthreshold**、 **sp_helpthreshold**、 **sp_modifythreshold**、 **sp_thresholdaction**

license_enabled

功能

如果启用了一个功能的许可，则返回 1，如果没有启用则返回 0，如果指定的许可名无效，则返回空值。

语法

```
license_enabled("ase_server" | "ase_ha" | "ase_dtm" |
   "ase_java" | "ase_asm")
```

参数

ase_server 为 Adaptive Server 指定许可。

ase_ha 为 Adaptive Server 高可用性功能指定许可。

ase_dtm 为 Adaptive Server 分布式事务管理功能指定许可。

ase_java 为 Adaptive ServerJava 功能指定许可。

ase_asm 为 Adaptive Server 高级安全机制指定许可。

示例

```
1. select license_enabled("ase_dtm")

   --------------
                1
```

表示 Adaptive Server 分布式事务管理功能的许可已启用。

注释

- 有关为 Adaptive Server 功能安装许可密钥的详细信息，参见 *安装指南*。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **license_enabled**。

参见

系统过程	**sp_configure**

log

功能

返回指定数字的自然对数。

语法

log(*approx_numeric*)

参数

approx_numeric — 是任何近似数值类型 （*float*、 *real* 或 *double precision*）列名、变量或常数表达式。

示例

```
1. select log(20)
   --------------------
                2.995732
```

注释

- **log** 是一种数学函数，它将返回指定值的对数值。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **log**。

参见

函数	log10、 power

log10

功能

返回指定数字的以 10 为底的对数。

语法

log10(*approx_numeric*)

参数

approx_numeric — 是任何近似数值类型 （*float*、 *real* 或 *double precision*）的列名、变量或常数表达式。

示例

```
1. select log10(20)

   --------------------
              1.301030
```

注释

- **log10** 是一种数学函数，它将返回指定值以 10 为底的对数值。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **log10**。

参见

函数	log、 power

lower

功能

返回指定表达式的等值小写表达式。

语法

lower(*char_expr*)

参数

char_expr — 是类型为 *char*、 *varchar*、 *nchar* 或 *nvarchar* 的字符型列名、变量或常量表达式。

示例

```
1. select lower(city) from publishers

   --------------------
   boston
   washington
   berkeley
```

注释

- **lower** 是字符串函数，它将大写转换为小写，返回字符值。
- **lower** 是与 **upper** 相反的命令。
- 如果 *char_expr* 是 NULL，返回 NULL。
- 有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **lower**。

参见

函数	upper

ltrim

功能

返回删去前导空白的指定表达式。

语法

```
ltrim(char_expr)
```

参数

char_expr — 是类型为 *char*、 *varchar*、 *nchar* 或 *nvarchar* 的字符型列名、变量或常量表达式。

示例

```
1. select ltrim("    123")
   -------
   123
```

注释

- **ltrim** 是字符串函数，它将删除字符表达式中的前导空白，而且只删除当前字符集中等于空格字符的值。
- 如果 *char_expr* 是 NULL，返回 NULL。
- 有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **ltrim**。

参见

函数	rtrim

max

功能

返回表达式的最大值。

语法

```
max(expression)
```

参数

expression — 可以是一个列名、一个常量、一个函数、或是一个由算术运算符或位运算符连接的列名、常量和函数组合、也可以是一个子查询。

示例

1.
```
select max(discount) from salesdetail

--------------------
            62.200000
```

返回表 *salesdetail* 中 *discount* 列的最大值，作为一个新列。

2.
```
select discount from salesdetail
compute max(discount)
```

返回表 *salesdetail* 中 *discount* 列的最大值，作为一个新行。

注释

- **max** 是一种集合函数，它在列或表达式中查找最大值。有关集合函数的一般信息，参见第 2-5 页上的 “集合函数”。
- **max** 可用于精确或近似数值、字符以及 *datetime* 列。但不可用于 *bit* 列。对于字符列， **max** 查找归类序列中的最大值。 **max** 将会忽略空值。 **max** 隐式地将 *char* 数据类型转换为 *varchar*，同时去掉所有的尾随空白。
- 当集合列有索引时，Adaptive Server 直接到索引的末尾查找最后一行以获得 **max** 值，以下情况除外：
  - *expression* 不是列
  - 该列不是索引的第一列
  - 查询中有另一集合
  - 有 **group by** 或 **where** 子句

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	缺省情况下， **max** 是不一致的；如要获得一致的行为，请使用 **set ansinull on**。

权限

任何用户都可以执行 **max**。

参见

命令	**compute Clause**、 **group by** 和 **having** 子句、**select**、 **where** 子句
函数	**avg**、 **min**

min

功能

返回列中的最小值。

语法

min(*expression*)

参数

expression — 可以是一个列名、一个常量、一个函数、或是一个由算术运算符或位运算符连接的列名、常量和函数组合、也可以是一个子查询。与集合使用的表达式通常是列名。有关详细信息，参见第 3 章 “表达式、标识符和通配符”中的 “表达式”。

示例

```
1. select min(price) from titles
   where type = "psychology"

   ------------------------
                        7.00
```

注释

- **min** 是一种集合函数，它在列或表达式中查找最小值。
- 有关集合函数的一般信息，参见第 2-5 页上的 “集合函数”。
- **min** 可用于数值、字符以及 *datetime* 列，但不可用于 *bit* 列。对于字符列，**min** 在有序序列中查找最小值。**min** 隐式地将 *char* 数据类型转换为 *varchar*，同时去掉所有的尾随空白。**min** 将会忽略所有的空值。**distinct** 不可用，因为它与 **min** 合用时没有意义。
- 当集合列有索引时， Adaptive Server 直接到第一个限定行查找 **min** 值，以下情况除外：
  - *expression* 不是列
  - 该列不是索引的第一列
  - 查询中有另一集合
  - 有 **group by** 子句

标准和一致性

标准	一致性级别	注释
SQL92	初级一致性	缺省情况下， **min** 是不一致的；如要获得一致的行为，请使用 **set ansinull on**。

权限

任何用户都可以执行 **min**。

参见

命令	**compute Clause**、 **group by** 和 **having** 子句、**select**、 **where** 子句
函数	**avg**、 **max**

mut_excl_roles

功能

返回有关两个角色之间互斥性的信息。

语法

```
mut_excl_roles (role1, role2 [membership |
   activation])
```

参数

role1 — 是在互斥性关系中的一个用户定义的角色。

role2 — 是在互斥性关系中的另一个用户定义的角色。

level — 是指定的角色之间相互排斥的级别 （membership 或 activation）。

示例

```
1. alter role admin add exclusive membership
   supervisor
   select
   mut_excl_roles("admin", "supervisor", "membership")
   -----------
             1
```

显示 *admin* 和 *supervisor* 角色是相互排斥的。

注释

- **mut_excl_roles** 是一种系统函数，它返回有关两个角色之间相互排斥关系的信息。如果系统安全性管理员将 *role1* 定义为与 *role2* 相互排斥的角色，或直接由 *role2* 所包含的角色，则 **mut_excl_roles** 返回 1；如果角色之间不是相互排斥的，则 **mut_excl_roles** 返回 0。
- 有关系统函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **mut_excl_roles**。

参见

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

object_id

功能

返回指定对象的对象 ID。

语法

```
object_id(object_name)
```

参数

object_name — 是数据库对象 （如表、视图、过程、触发器、缺省值或规则）的名称。该名可以是全限定名 （即，可包括数据库名和所有者名）。应该用引号将 *object_name* 引起来。

示例

```
1. select object_id("titles")

   -----------
     208003772

2. select object_id("master..sysobjects")

   -----------
             1
```

注释

- **object_id** 是一种系统函数，它返回对象的 ID。对象 ID 存储在 *sysobjects* 的 *id* 列中。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **object_id**。

参见

函数	col_name、 db_id、 object_name
系统过程	sp_help

object_name

功能

返回指定对象 ID 的对象名称。

语法

object_name(*object_id*[, *database_id*])

参数

object_id — 是数据库对象 （如表、视图、过程、触发器、缺省值或规则）的对象 ID。对象 ID 存储在 *sysobjects* 的 *id* 列中。

database_id — 是当对象不在当前数据库中时的数据库 ID。数据库 ID 存储在 *sysdatabase* 的 *db_id* 列中。

示例

```
1. select object_name(208003772)

   ------------------------------
   titles
2. select object_name(1, 1)

   ------------------------------
   sysobjects
```

注释

- **object_name** 是一种系统函数，它返回对象的名称。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **object_name**。

参见

函数	**col_name**、 **db_name**、 **object_id**
系统过程	**sp_help**

patindex

功能

返回指定模式在第一次出现时的起始位置。

语法

```
patindex("%pattern%", char_expr [, using {bytes |
   characters | chars} ] )
```

参数

pattern — 是 *char* 或 *varchar* 数据类型的字符表达式，可用包括 Adaptive Server 支持的任何模式匹配的通配符。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 3 章 “表达式、标识符和通配符”中的 “与通配符匹配的模式”。

char_expr — 是类型为 *char*、 *varchar*、 *nchar* 或 *nvarchar* 的字符型列名、变量或常量表达式。

using — 指定起始位置的格式。

bytes — 返回的偏移以字节表示。

chars 或 **characters** — 返回的偏移以字符表示 （缺省）。

示例

1.
```
select au_id, patindex("%circus%", copy)
from blurbs
```

```
au_id
 ----------- -----------
 486-29-1786           0
 648-92-1872           0
 998-72-3567          38
 899-46-2035          31
 672-71-3249           0
 409-56-7008           0
```

选择作者 ID 和 *copy* 列中词 “circus”的起始字符位置。

2.
```
select au_id, patindex("%circus%", copy,
     using chars)
from blurbs
```

3.
```
select au_id, patindex("%circus%", copy,
     using chars)
from blurbs
```

   与示例 1 相同。

4.
```
select name
from sysobjects
where patindex("sys[a-d]%", name) > 0
```

```
name
------------------------------
sysalternates
sysattributes
syscharsets
syscolumns
syscomments
sysconfigures
sysconstraints
syscurconfigs
sysdatabases
sysdepends
sysdevices
```

   查找 *sysobjects* 中所有以 “sys” 开头且第四个字符是 “a”、“b”、“c” 或 “d” 的行。

注释

- **patindex** 是一种字符串函数，它返回的整数表示指定字符表达式中 *pattern* 首次出现时的起始位置；如果未找到 *pattern*，则返回零。
- **patindex** 可用于所有字符数据，包括 *text* 和 *image* 数据。
- 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**。
- 在 *pattern* 之前和之后加上百分比符号。要将 *pattern* 视为列的首字符，省略前面的%。要将 *pattern* 视为列的最后字符，省略后面的%。
- 如果 *char_expr* 是 NULL，返回 0。
- 有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **patindex**。

参见

函数	**charindex**、 **substring**

pi

功能

返回常数值 3.1415926535897936。

语法

```
pi()
```

参数

无。

示例

```
1. select pi()
   -------------------
             3.141593
```

注释

- **pi** 是一种数学函数，它返回常数值 3.1415926535897931。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **pi**。

参见

函数	degrees、 radians

power

功能

返回求指定数字的给定次幂所得的值。

语法

power(*value*, *power*)

参数

value — 是数字值。

power — 是精确数值、近似数值或货币值。

示例

```
1. select power(2, 3)
   -----------
             8
```

注释

- **power** 是一种数学函数，它返回求 *value* 的 *power* 次幂所得的值。其结果与 *value* 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，标度与该表达式的标度相同。

- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **power**。

参见

函数	exp、 log、 log10

proc_role

功能

返回关于是否已授予用户特定角色的信息。

语法

```
proc_role ("role_name")
```

参数

role_name — 是系统角色或用户定义角色的名称。

示例

1.
```
create procedure sa_check as
if (proc_role("sa_role") > 0)
begin
     return(1)
end
print "You are a System Administrator."
```
   创建一个过程来检查用户是不是系统管理员。

2.
```
select proc_role("sso_role")
```
   检查是否已授予用户系统安全员角色。

3.
```
select proc_role("oper_role")
```
   检查是否已授予用户操作员角色。

注释

- **proc_role** 是一种系统函数，它检查是否已将指定的角色授予给调用用户，以及是否已将该角色激活。
- 如果出现下列任何情况，**proc_role** 返回 0：
  - 未授予用户指定角色
  - 未授予用户包含指定角色的角色
  - 已授予用户指定角色，但未激活该角色
- 如果已授予调用用户指定角色并已激活该角色，**proc_role** 返回 1。
- 如果调用用户拥有一个当前活动角色，其中包含所指定的角色，但未激活该指定角色，**proc_role** 返回 2。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **proc_role**。

参见

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

ptn_data_pgs

功能

返回某一分区所用的数据页数。

语法

```
ptn_data_pgs(object_id, partition_id)
```

参数

object_id — 是表的对象 ID，存储在 *sysobjects*、 *sysindexes* 和 *syspartitions* 的 *id* 列中。

partition_id — 是表的分区号。

示例

```
1. select ptn_data_pgs(object_id("salesdetail"), 1)
   -----------
             5
```

注释

- **ptn_data_pgs** 是一种系统函数，它返回某一已分区表中的数据页数。
- 使用 **object_id** 函数可获取对象 ID，而使用 **sp_helpartiton** 可将分区列在表中。
- **ptn_data_pgs** 返回的数据页可能不精确。要得到最精确的值，可在使用 **ptn_data_pgs** 之前使用 **update partition statistics**、 **dbcc checktable**、**dbcc checkdb** 或 **dbcc checkalloc** 命令。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

只有表的所有者才能执行 **ptn_data_pgs**。

参见

命令	**update partition statistics**
函数	**data_pgs、 object_id**
系统过程	**sp_helpartition**

radians

功能

返回指定度数角的以弧度表示的大小。

语法

radians(*numeric*)

参数

numeric — 是任意精确数值 （*numeric*、 *dec*、 *decimal*、 *tinyint*、 *smallint* 或 *int*），近似数值 （*float*、 *real* 或 *doubleprecision*），或 *money* 列、变量、常数表达式，或上述数据类型的组合形式。

示例

```
1. select radians(2578)

   -----------
            44
```

注释

- **radians** 是一种数学函数，它可将度转换为弧度。其结果与 *numeric* 具有相同的类型。

  对于数字或十进制类型的表达式，其结果的内部精度是 77，标度与该数字表达式的标度相同。

  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。

- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **radians**。

参见

函数	degrees

rand

功能

返回 0 和 1 之间的随机值，该值是使用指定的源值来生成的。

语法

rand([*integer*])

参数

integer — 是任意整数 （*tinyint*、 *smallint* 或 *int*）列的名称、变量、常量表达式或上述数据类型的组合形式。

示例

```
1. select rand()

   --------------------
              0.395740

2. declare @seed int
   select @seed=100
   select rand(@seed)

   --------------------
                0.000783
```

注释

- **rand** 是一种数学函数，它使用可选整数作为源值，返回 0 和 1 之间的随机浮动值。
- **rand** 函数使用 32 位伪随机整数发生器的输出。该整数最多可被 32 位整数除，得出 0.0 到 1.0 之间的双精度数值。服务器启动时将随机产生 **rand** 函数的源值，所以不可能得到同一顺序的随机数字，除非用户先用常量源值对该函数进行了初始化。 **rand** 函数是全局资源。如果有多个用户调用 **rand** 函数，则按同一顺序的伪随机值依次进行。如果需要一系列可重复的随机数字，用户必须确保函数起初的源值相同，而且还要确保当需要重复序列时，没有其他用户调用 **rand**。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **rand**。

参见

数据类型	“近似数值数据类型”

replicate

功能

返回将指定表达式重复特定次数所形成的字符串。

语法

```
replicate (char_expr, integer_expr)
```

参数

char_expr — 是类型为 *char*、 *varchar*、 *nchar* 或 *nvarchar* 的字符型列名、变量或常量表达式。

integer_expr — 是任意整数 （*tinyint*、 *smallint* 或 *int*）列名、变量或常量表达式。

示例

```
1. select replicate("abcd", 3)

   ------------
   abcdabcdabcd
```

注释

- **replicate** 是一种字符串函数，它返回与 *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。
- 如果 *char_expr* 是 NULL，返回单个 NULL。
- 有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **replicate**。

参见

函数	stuff

reserved_pgs

功能

返回分配给指定表或索引的页数。

语法

```
reserved_pgs(object_id, {doampg|ioampg})
```

参数

object_id — 是一个数字表达式，表示表、视图或其它数据库对象的对象 ID。它存储在 *sysobjects* 的 *id* 列中。

doampg | ioampg — 指定表 (**doampg**) 或索引 (**ioampg**)。

示例

```
1. select reserved_pgs(id, doampg)
   from sysindexes where id =
       object_id("syslogs")

   -------------
             534
```

返回 *syslogs* 表的页数。

注释

- **reserved_pgs** 是一种系统函数，它返回分配给表或索引的页数。
- **reserved_pgs** 报告用于内部结构的页。
- **reserved_pgs** 仅用于当前数据库的对象。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **reserved_pgs**。

参见

命令	**update statistics**
函数	**data_pgs、 used_pgs**

reverse

功能

返回其字符逆序排列的指定字符串。

语法

reverse(*expression*)

参数

expression — 是一个字符类型或二进制类型 （即 *char*、 *varchar*、 *nchar*、 *nvarchar*、 *binary* 或 *varbinary* 类型）的列名、变量或常量表达式。

示例

```
1. select reverse("abcd")

   ----
   dcba
2. select reverse(0x12345000)

   ----------
   0x00503412
```

注释

- **reverse** 是一种字符串函数，它返回 *expression* 的逆序字符串。
- 如果 *expression* 是 NULL，返回 NULL。
- 有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **reverse**。

参见

函数	lower、 upper

right

功能

表达式右端具有指定个数字符的部分。

语法

right(*expression*, *integer_expr*)

参数

expression — 是一个字符类型或二进制类型（*char*、 *varchar*、 *nchar*、 *nvarchar*、 *binary* 或 *varbinary* 类型）的列名、变量或常量表达式。

integer_expr — 是任意整数 （*tinyint*、 *smallint* 或 *int*）列名、变量或常量表达式。

示例

```
1. select right("abcde", 3)
    ---
    cde
2. select right("abcde", 2)
    --
    de
3. select right("abcde", 6)
    -----
    abcde
4. select right(0x12345000, 3)
    -------
    0x345000
5. select right(0x12345000, 2)
    ------
    0x5000
6. select right(0x12345000, 6)
    ---------
    0x12345000
```

注释

- **right** 是一种字符串函数，它返回从字符或二进制表达式最右边开始的指定数目的字符。
- 返回值与字符或二进制表达式的数据类型相同。
- 如果 *expression* 是 NULL，返回 NULL。
- 有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **right**。

参见

函数	**rtrim**、 **substring**

role_contain

功能

当 *role2* 包含 *role1* 时返回 1。

语法

```
role_contain("role1", "role2")
```

参数

role1 — 是系统角色或用户定义角色的名称。

role2 — 是另一个系统角色或用户定义角色的名称。

示例

```
1. select role_contain("intern_role", "doctor_role")

   -----------
   1

2. select role_contain("specialist_role",
   "intern_role")

   -----------
   0
```

注释

- **role_contain** 是一种系统函数，当 *role2* 包含 *role1* 时返回 1。
- 有关包含角色和角色层次的详细信息，参见*系统管理指南*的第 6 章“管理 Adaptive Server 登录和数据库用户”中的“定义和改变角色层次”。
- 有关系统函数的详细信息，参见第 2-20 页上的“系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **role_contain**。

参见

函数	**mut_excl_roles**、**proc_role**、**role_id**、 **role_name**
命令	**alter role**
系统过程	**sp_activeroles**、**sp_configure**、**sp_displaylogin**、**sp_displayroles**、 **sp_helprotect**、**sp_modifylogin**、 **sp_role**

role_id

功能

返回指定角色的系统角色 ID。

语法

```
role_id("role_name")
```

参数

role_name — 是系统角色或用户定义角色的名称。角色名称和角色 ID 存储在 *syssrvroles* 系统表中。

示例

1.
```
select role_id("sa_role")

------
0
```

返回 **sa_role** 的系统角色 ID。

2.
```
select role_id("intern_role")

------
6
```

返回 “intern_role” 的系统角色 ID。

注释

- **role_id** 是一种系统函数，它返回系统角色 ID (*srid*)。系统角色 ID 存储在 *syssrvroles* 系统表的 *srid* 列中。
- 如果 *role_name* 在系统中无效， Adaptive Server 返回 NULL。
- 关于角色的详细信息，参见 *系统管理指南* 的第 6 章 “管理 Adaptive Server 登录和数据库用户” 中的 “为用户创建和指派角色”。
- 有关系统函数的详细信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **role_id**。

参见

函数	**mut_excl_roles**、**proc_role**、**role_contain**、**role_name**

role_name

功能

返回指定其系统角色 ID 的角色名称。

语法

```
role_name(role_id)
```

参数

role_id — 是角色的系统角色 ID (*srid*)。角色名称存储在 *syssrvroles* 中。

示例

```
1. select role_name(01)

   ------------------------------
   sso_role
```

注释

- **role_name** 是一种系统函数，它返回角色的名称。
- 关于角色的详细信息，参见*系统管理指南*的第 6 章 “管理 Adaptive Server 登录和数据库用户”中的 “为用户创建和指派角色”。
- 有关系统函数的详细信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **role_name**。

参见

函数	mut_excl_roles、 proc_role、 role_contain、 role_id

round

功能

返回指定数字舍入到给定的小数位数后所得的值。

语法

round(*number*, *decimal_places*)

参数

numeric — 是任意精确数值 （*numeric*、 *dec*、 *decimal*、 *tinyint*、 *smallint* 或 *int*），近似数值 （*float*、 *real* 或 *double precision*），或 *money* 列、变量、常数表达式，或上述数据类型的组合形式。

decimal_places — 是要舍入到的小数位数。

示例

```
1. select round(123.4545, 2)

   ----------
     123.4500
2. select round(123.45, -2)

   ----------
     100.00
3. select round(1.2345E2, 2)

   -----------------
          123.450000
4. select round(1.2345E2, -2)

   -----------------
         100.000000
```

注释

- **round** 是一种数学函数，它通过舍入 *number* 以使其具有 *decimal_places* 个有效位。
- 正 *decimal_places* 决定小数点右边的有效位数；负 *decimal_places* 决定小数点左边的有效位数。
- 结果与 *number* 的类型相同，对于数字和小数表达式，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 *number* 的标度。

- **round** 总是返回一个值。如果 *decimal_places* 为负且超过 *number* 的有效位数， Adaptive Server 返回 0。（它被表示为 0.00 的形式，其中小数点右边 0 的个数等于 *numeric* 的标度。）例如：

  ```
  select round(55.55, -3)
  ```

  返回 0.00。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **round**。

参见

函数	**abs**、 **ceiling**、 **floor**、 **sign**、 **str**

rowcnt

功能

返回指定表中行数的估计值。

语法

```
rowcnt(sysindexes.doampg)
```

参数

sysindexes.doampg — 是 *sysindexes* 中的行数。

示例

```
1. select name, rowcnt(sysindexes.doampg)
        from sysindexes
        where name in
           (select name from sysobjects
            where type = "U")

   name
   ---------------------------- -----------
   roysched                              87
   salesdetail                          116
   stores                                 7
   discounts                              4
   au_pix                                 0
   blurbs                                 6
```

注释

- **rowcnt** 是一种系统函数，它返回表的估计行数。
- 当 Adaptive Server 重新启动并恢复事务时，**rowcnt** 返回的值会出乎意料的变化。运行下列命令之一后，返回值最精确：
  - **dbcc checkalloc**
  - **dbcc checkdb**
  - **dbcc checktable**
  - **update all statistics**
  - **update statistics**
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **rowcnt**。

参见

编目存储过程	**sp_statistics**
命令	**dbcc**、 **update statistics**
函数	**data_pgs**
系统过程	**sp_helpartition**、 **sp_spaceused**

rtrim

功能

返回删去尾随空白的指定表达式。

语法

```
rtrim(char_expr)
```

参数

char_expr — 是类型为 *char*、 *varchar*、 *nchar* 或 *nvarchar* 的字符型列名、变量或常量表达式。

示例

```
1. select rtrim("abcd    ")
   --------
   abcd
```

注释

- **rtrim** 是一种字符串函数，它撤除尾随空白。
- 如果 *char_expr* 是 NULL，返回 NULL。
- 只删除当前字符集中等于空格字符的值。
- 有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **rtrim**。

参见

函数	ltrim

show_role

功能

显示登录的当前活动角色，这些角色由系统定义。

语法

```
show_role()
```

参数

无。

示例

```
1. select show_role()
   sa_role sso_role oper_role replication_role
2. if charindex("sa_role", show_role()) >0
   begin
        print "You have sa_role"
   end
```

注释

- **show_role** 是一种系统函数，只要登录在当前有任何活动的系统定义角色 （**sa_role**、 **sso_role**、 **oper_role** 或 **replication_role**），该函数就会返回这些角色。如果登录没有任何角色， **show_role** 将返回 NULL。
- 当数据库所有者在使用 **setuser** 之后调用 **show_role** 时， **show_role** 将显示数据库所有者的活动角色，而不显示通过 **setuser** 所充当的用户。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **show_role**。

参见

命令	**alter role**、 **create role**、 **drop role**、 **grant**、 **set**、 **revoke**
函数	**proc_role**、 **role_contain**
系统过程	**sp_activeroles**、 **sp_displaylogin**、 **sp_displayroles**、 **sp_helprotect**、 **sp_modifylogin**、 **sp_role**

show_sec_services

功能

列出可供会话使用的安全服务。

语法

```
show_sec_services()
```

参数

无。

示例

```
1. select show_sec_services()
   encryption, replay_detection
```

显示用户的当前会话正在加密数据并执行重放检测检查。

注释

- 使用 **show_sec_services** 可列出在会话过程中处于活动状态的安全服务。
- 如果没有活动的安全服务， **show_sec_services** 将返回 NULL。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **show_sec_services**。

参见

函数	is_sec_service_on

sign

功能

返回指定值的符号：+1 （正）、 0、 -1 （负）。

语法

sign(*numeric*)

参数

numeric — 是任何精确数值 （*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。

示例

```
1. select sign(-123)
   -----------
            -1
2. select sign(0)
   -----------
             0
3. select sign(123)
   -----------
             1
```

注释

- **sign** 是一种数学函数，它返回正 (+1)、零 (0) 或负 (-1)。
- 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。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **sign**。

参见

函数	**abs**、 **ceiling**、 **floor**、 **round**

sin

功能

返回指定角 （以弧度表示）的正弦。

语法

sin(*approx_numeric*)

参数

approx_numeric — 是任何近似数值类型 （*float*、 *real* 或 *double precision*）的列名、变量或常量表达式。

示例

```
1. select sin(45)

   --------------------
              0.850904
```

注释

- **sin** 是一种数学函数，它返回指定角 （以弧度表示）的正弦。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **sin**。

参见

函数	**cos**、 **degrees**、 **radians**

sortkey

功能

生成可用于根据归类行为对结果进行排序的值。该函数允许您采用缺省设置（基于拉丁字符的字典排序顺序行为以及大小写或变音区分行为）之外的字符归类行为。

语法

```
sortkey (char_expression [, {collation_name |
   collation_ID}])
```

参数

char_expression 是以下之一：

- 字符类型（*char*、*varchar*、*nchar* 或 *nvarchar*）
- 字符变量，或
- 用单引号或双引号引起来的常量字符表达式

collation_name 是引起来的字符串或字符变量，用于指定要使用的归类。

collation_ID 是一种整数常量或变量，用于指定要使用的归类。

注释

- **sortkey** 函数生成可用于根据归类行为对结果进行排序的值。该函数允许您采用缺省设置（基于拉丁字符的字典排序顺序行为以及大小写或变音区分行为）之外的字符归类行为。函数返回值是 *varbinary* 数据类型的值，此值包含从 **sortkey** 函数保留的输入字符串的编码归类信息。

  例如，可以将 **sortkey** 返回的值存储在具有源字符串的列中。如果要按期望的顺序检索字符数据，对于包含 **sortkey** 运行结果的列，**select** 语句只需包含 **order by** 子句即可。

  sortkey 函数保证其为给定归类标准返回的值能够用于 *varbinary* 数据类型的二进制比较。
- *char_expression* 必须是由服务器的缺省字符集中编码的字符组成。
- *char_expression* 可以是空字符串。如果是空字符串：
  - **sortkey** 返回长度为零的 *varbinary* 值，且
  - Adaptive Server 将空字符串存储为空白。

  空字符串与数据库列中的 NULL 字符串有不同的归类值。

- 如果 *char_expression* 是 NULL， **sortkey** 返回的值将为 NULL。
- 如果没有指定 *collation_name* 或 *collation_ID* 的值， **sortkey** 将采取二进制归类。

➤ ***注意***

sortkey 最多可为每个输入字符生成 6 个字节的归类信息。因此，使用 **sortkey** 的结果可能超过 *varbinary* 数据类型的 255 字节长度限制。如果发生这种情况，结果将被相应截断。截断将撤除每个输入字符的结果字节，直到结果字符串小于 255 字节为止。如果发生截断，将发布一条警告消息，但包含 **sortkey** 函数的查询或事务会继续进行。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **sortkey**。

参见

函数	compare

soundex

功能

返回表示表达式发音方式的 4 字符代码。

语法

soundex(*char_expr*)

参数

char_expr — 是类型为 *char*、 *varchar*、 *nchar* 或 *nvarchar* 的字符型列名、变量或常量表达式。

示例

```
1. select soundex ("smith"), soundex ("smythe")

   ----- -----
   S530  S530
```

注释

- **soundex** 是一种字符串函数。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 4 个字符的 soundex 代码。
- **soundex** 函数可将字母字符串转换为四位代码，用以查找发音相似的单词或名称。除了作为字符串首字母的元音之外，将忽略所有的元音。
- 如果 *char_expr* 是 NULL，将返回 NULL。
- 有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **soundex**。

参见

函数	difference

space

功能

返回由指定数量的单字节空格所组成的字符串。

语法

space(*integer_expr*)

参数

integer_expr — 是任何整数 （*tinyint*、 *smallint* 或 *int*）列名、变量或常量表达式。

示例

```
1. select "aaa", space(4), "bbb"
   --- ---- ---
   aaa      bbb
```

注释

- **space** 是一种字符串函数，它返回具有指定数量单字节空格的字符串。
- 有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **space**。

参见

函数	isnull、 isnull、 rtrim

sqrt

功能

返回指定数字的平方根。

语法

sqrt(*approx_numeric*)

参数

approx_numeric — 是求值为正数的任何近似数值类型 (*float*、 *real* 或 *double precision*) 的列名、变量或常数表达式。

示例

```
1. select sqrt(4)
    2.000000
```

注释

- **sqrt** 是一种数学函数，它返回指定值的平方根。
- 如果选择负数的平方根， Adaptive Server 将返回以下错误消息：`Domain error occurred.`
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **sqrt**。

参见

函数	power

str

功能

返回指定数字的等值字符。

语法

str(*approx_numeric* [, *length* [, *decimal*]])

参数

approx_numeric — 是任何近似数值类型 （*float*、 *real* 或 *double precision*）的列名、变量或常量表达式。

length — 设置要返回的字符数 （包括小数点、小数点左右两侧的所有数字位以及空格）。缺省值为 10。

decimal — 设置要返回的小数位数。缺省值为 0。

示例

```
1. select str(1234.7, 4)

   ----
   1235
2. select str(-12345, 6)

   ------
   -12345
3. select str(123.45, 5, 2)

   -----
   123.5
```

注释

- **str** 是一种字符串函数，它返回表示浮点数的字符。有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。
- *length* 和 *decimal* 是可选参数。如果给出这两个函数，它们就必须是非负数。**str** 将舍入数字的小数部分，使结果不超出指定的长度。该长度必须足以容纳小数点，并且在为负时必须能够容纳数字的符号。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** 则返回一行具有指定长度的星号。例如：

  ```
  select str(123.456, 2, 4)

  --
  **
  ```

  短 *approx_numeric* 按指定长度右对齐，而长 *approx_numeric* 则被截断到指定的小数位数。
- 如果 *approx_numeric* 是 NULL，将返回 NULL。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **str**。

参见

函数	abs、 ceiling、 floor、 round、 sign

stuff

功能

返回通过以下方法形成的字符串：从一个字符串中删除指定个字符，并将这些字符替换为另一个字符串。

语法

stuff(*char_expr1, start, length, char_expr2*)

参数

char_expr1 — 是一个类型为 *char*、 *varchar*、 *nchar* 或 *nvarchar* 的字符型列名、变量或常量表达式。

start — 指定开始删除字符的字符位置。

length — 指定要删除的字符数。

char_expr2 — 是另一个类型为 *char*、 *varchar*、 *nchar* 或 *nvarchar* 的字符型列名、变量或常量表达式。

示例

```
1. select stuff("abc", 2, 3, "xyz")

   ----
   axyz
2. select stuff("abcdef", 2, 3, null)

   go
   ---
   aef
3. select stuff("abcdef", 2, 3, "")

   ----
   a ef
```

注释

- **stuff** 是字符串函数，它从 *char_expr1* 中的 *start* 处开始删除 *length* 个字符，然后在 *char_expr1* 中的 *start* 处插入 *char_expr2*。有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。
- 如果起始位置或长度为负，将返回 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* 的全部字符 （参见示例 1）。

- 要使用 **stuff** 来删除某个字符，应用 “NULL“ （而不是空引号）来替换 *expr2*。如果使用 “ ‘’ 来指定用空字符，则会将其替换为空格 （参见示例 2 和 3）。
- 如果 *char_expr1* 是 NULL，将返回 NULL。如果 *char_expr1* 是字符串值，而 *char_expr2* 是 NULL，则不用任何字符替换被删除的字符。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **stuff**。

参见

函数	replicate、 substring

substring

功能

返回通过从另一个字符串中抽取指定数目的字符所形成的字符串。

语法

substring(*expression, start, length*)

参数

expression — 是一种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* 或 *nvarchar* 数据、 *binary* 或 *varbinary*。

start — 指定子串开始的字符位置。

length — 指定子串中的字符数。

示例

1. `select au_lname, substring(au_fname, 1, 1)`
   `from authors`

   显示每个作者的姓和名，如 “Bennet A”。

2. `select substring(upper(au_lname), 1, 3)`
   `from authors`

   将作者的名转换为大写形式，然后显示前三个字符。

3. `select substring((pub_id + title_id), 1, 6)`

   将 *pub_id* 和 *title_id* 并置，然后显示所生成字符串的前六个字符。

4. **`select substring(xactid,5,2) from syslogs`**

   从二进制域中抽取较低的四位数，其中的每个位置都表示两个二进制位：

注释

- **substring** 是一种字符串函数，它返回字符或二进制字符串的一部分。有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。
- 如果 **substring** 的任何参数为 NULL，将返回 NULL。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **substring**。

参见

函数	**charindex**、 **patindex**、 **stuff**

sum

功能

返回值的总和。

语法

```
sum([all | distinct] expression)
```

参数

all — 对所有值应用 **sum**。**all** 是缺省参数。

distinct — 在应用 **sum** 之前消除重复值。**distinct** 是可选参数。

expression — 可以是列名、常量、函数、任何由算术运算符或逐位运算符连接的列名、常量和函数组合，也可以是子查询。与集合使用的表达式通常是列名。有关详细信息，参见第 3 章 “表达式、标识符和通配符”中的 “表达式”。

示例

1. ```
   select avg(advance), sum(total_sales)
   from titles
   where type = "business"
   ```

   计算平均预付款以及所有商业书籍总销售额的总和。每个这样的集合函数都将为检索到的所有行生成单个摘要值。

2. ```
   select type, avg(advance), sum(total_sales)
   from titles
   group by type
   ```

   当用于 **group by** 子句时，集合函数将为每组 （而非整个表）生成单个值。以上语句将生成每种书籍的摘要值。

3. ```
   select pub_id, sum(advance), avg(price)
   from titles
   group by pub_id
   having sum(advance) > $25000 and avg(price) > $15
   ```

   按出版者将 *titles* 表分组，并且只包括那些总预付款超过 $25,000 且书籍平均价格高于 $15 的出版者所形成的组。

注释

- **sum** 是一种集合函数，它计算出一列中所有值的和。**sum** 只能用于数字 （整数、浮点或货币）数据类型。计算和时将忽略空值。
- 有关集合函数的一般信息，参见第 2-5 页上的 “集合函数”。
   ```

- 当用 sum 计算整数数据的和时，即使列的数据类型是 *smallint* 或 *tinyint*， Adaptive Server 也会将结果当作一个 *int* 值。为避免 DB-Library 程序出现溢出错误，应将平均或求和结果的所有变量声明为类型 *int*。
- 不能对二进制数据类型使用 **sum**。

标准和一致性

标准	一致性级别	注释
SQL92	Transact-SQL 扩展	在缺省情况下， **sum** 是不一致的。要获得一致的行为，应指定 **set ansinull on**。

权限

任何用户都可以执行 **sum**。

参见

命令	**compute Clause**、 **group by 和 having 子句**、 **select**、 **where 子句**
函数	**count**、 **max**、 **min**

suser_id

功能

返回服务器用户在 *syslogins* 表中的 ID 号。

语法

```
suser_id([server_user_name])
```

参数

server_user_name — 是 Adaptive Server 登录名。

示例

```
1. select suser_id()
   ------
        1
2. select suser_id("margaret")
   ------
        5
```

注释

- **suser_id** 是一种系统函数，它返回服务器用户在 *syslogins* 中的 ID 号。有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。
- 要从 *sysusers* 表中查找特定数据库中的用户 ID，可使用 **user_id** 系统函数。
- 如果未应用 *server_user_name*， **suser_id** 将返回当前用户的服务器 ID。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **suser_id**。

参见

函数	suser_name、 user_id

suser_name

功能

返回当前服务器用户的名称或已指定服务器用户 ID 的用户。

语法

```
suser_name([server_user_id])
```

参数

server_user_id — 是 Adaptive Server 用户 ID。

示例

```
1. select suser_name()
   ------------------------------
   sa
2. select suser_name(4)
   ------------------------------
   margaret
```

注释

- **suser_name** 是一种系统函数，它返回服务器用户的名称。服务器用户 ID 存储在 *syslogins* 中。如果未提供 *server_user_id*，**suser_name** 将返回当前用户的名称。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **suser_name**。

参见

函数	suser_id、 user_name

syb_sendmsg

功能

将消息发送到 UDP （用户数据报协议）端口。

语法

syb_sendmsg *ip_address*, *port_number*, *message*

参数

ip_address — 是运行 UDP 应用程序的计算机的 IP 地址。

port_number — 是 UDP 端口的端口号。

示例

1. **select syb_sendmsg("120.10.20.5", 3456, "Hello")**

   将消息 “Hello”发送到 IP 地址为 120.10.20.5 的端口 3456。

2. ```
   declare @msg varchar(255)
   select @msg = "Message to send"
   select syb_sendmsg (ip_address, portnum, @msg)
   from sendports
   where username = user_name()
   ```

   从用户表中读取 IP 地址和端口号，并使用变量表示要发送的消息。

注释

- Windows NT 不支持 **sp_sendmsg**。
- 系统安全员必须将配置参数 **allow sendmsg** 设置为 1 才能启用 UDP 消息传送功能。
- 对于 **sp_sendmsg** 没有安全性检查。 Sybase 强烈建议您在使用 **sp_sendmsg** 在网络上发送敏感信息时一定要谨慎。如果启用此功能，则表示用户接受使用该功能所导致的任何安全性问题。
- 有关用于创建 UDP 端口的 C 程序的示例，参见 **sp_sendmsg**。
   ```

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **syb_sendmsg**。

参见

系统过程	sp_sendmsg

tan

功能

返回指定角 （以弧度表示）的正切。

语法

tan(*angle*)

参数

angle — 是以弧度表示的角大小，表示为 *float*、 *real*、 *double precision* 类型 （或任何可隐式转换为这些类型的数据类型）的列名、变量或表达式。

示例

```
1. select tan(60)

   --------------------
              0.320040
```

注释

- **tan** 是一种数学函数，它返回指定角 （以弧度表示）的正切。
- 有关数学函数的一般信息，参见第 2-17 页上的 “数学函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **tan**。

参见

函数	atan、 atn2、 degrees、 radians

textptr

功能

返回指向指定 *text* 或 *image* 列首页的 16 字节 *varbinary* 指针。

语法

textptr(*column_name*)

参数

column_name — 是 *text* 列的名称。

示例

1. **declare @val varbinary(16)**
   select @val = textptr(copy) from blurbs
   where au_id = "86-29-1786"
   readtext blurbs.copy @val 1 5

   上例使用 **textptr** 函数来定位 *text* 列：*copy*。该列与作者 *blurbs* 表中的 *au_id* 486-29-1786 相关联。文本指针放在局部变量 *@val* 中并作为 **readtext** 命令的参数提供，该命令将从第二个字节开始返回 5 个字节 （偏移为 1）。

2. **select au_id, textptr(copy) from blurbs**

   从 *texttest* 表中选择 *title_id* 列和 16 字节的 *blurb* 列文本指针。

注释

- **textptr** 是一种文本和图像函数，它返回文本指针值：16 字节的 *varbinary* 值。应检查文本指针，确保它指向文本或图像的首页。
- 如果还没有用非空的 **insert** 或任何 **update** 语句将 *text* 或 *image* 列初始化， **textptr** 就将返回 NULL 指针。用 **textvalid** 来检查文本指针是否存在。如果没有有效的文本指针，将不能使用 **writetext** 或 **readtext**。
- 有关文本和图像函数的一般信息，参见第 2-22 页上的 "文本和图像函数"。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **textptr**。

参见

数据类型	“text 和 image 数据类型”
函数	**textvalid**

textvalid

功能

指向指定 *text* 列的指针有效时返回 1；无效时返回 0。

语法

```
textvalid("table_name.column_name", textpointer)
```

参数

table_name.column_name — 是表及其 *text* 列的名称。

textpointer — 是文本指针的值。

示例

```
1. select textvalid ("texttest.blurb", textptr(blurb))
   from texttest
```

报告 *texttest* 表 *blurb* 列中的每个值是否都有有效的文本指针。

注释

- **textvalid** 是一种文本和图像函数，它检查给定的文本指针是否有效。指针有效时返回 1；无效时返回 0。
- *text* 或 *image* 列的标识符必须包含表名。
- 有关文本和图像函数的一般信息，参见第 2-22 页上的“文本和图像函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **textvalid**。

参见

数据类型	“text 和 image 数据类型”
函数	**textptr**

tsequal

功能

比较 *timestamp* 值，这样，如果某行在被选中进行浏览之后经过了修改，就可防止将其更新。

语法

tsequal(*browsed_row_timestamp*, *stored_row_timestamp*)

参数

browsed_row_timestamp — 是所浏览行的 *timestamp* 列。

stored_row_timestamp — 是所存储行的 *timestamp* 列。

示例

```
1. update publishers
   set city == "Springfield"
   where pub_id = "0736"
   and tsequal(timestamp, 0x0001000000002ea8)
```

从当前版本的 *publishers* 表中检索 *timestamp* 列，并将其与已保存的 *timestamp* 列中的值进行比较。如果这两个 *timestamp* 列中的值相等，将更新该行。如果它们的值不相等，则返回错误消息。

注释

- **tsequal** 是一种系统函数，它比较 *timestamp* 列的值，这样，如果某行在被选中进行浏览之后经过了修改，就可防止将其更新。有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。
- **tsequal** 使您无需调用 DB-Library 中的 **dbqual** 函数，即可使用浏览模式。浏览模式支持在查看数据的同时执行更新。它用于使用 Open Client 和主机编程语言的前端应用程序。如果表中的行带有时间戳，就可以浏览该表。
- 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：

Open Client 应用程序中 **select** *语句的起始位置*

```
...
   for browse
```

***Open Client** 应用程序例程的结束位置*

- **tsequal** 函数不应该用于 select 语句的 **where** 子句，而只能用于 **insert** 和 **update** 语句的 where 子句，因为其中 **where** 子句的其余部分匹配一个唯一的行。

  如果将 *timestamp* 列用作搜索子句，就应该象比较普通 *varbinary* 列那样来比较该列，即 *timestamp1 = timestamp2*。

为用于浏览的新表加盖时间戳

- 在创建用于浏览的新表时，在表定义中添加名为 *timestamp* 的列。系统将自动为该列分配 *timestamp* 数据类型，所以您不必指定其数据类型。例如：

  ```
  create table newtable(col1 int, timestamp,
      col3 char(7))
  ```

  只要插入或更新某行， Adaptive Server 就会自动为 *timestamp* 列分配唯一的 *varbinary* 值，从而为其加盖时间戳。

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
  ```

  将带有 NULL 值的 *timestamp* 列添加到每个现有行中。要生成时间戳，可在不指定新列值的情况下更新每个现有行。例如：

  ```
  update oldtable
  set col1 = col1
  ```

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **tsequal**。

参见

数据类型	“时间戳数据类型”

upper

功能

返回指定字符串的等值大写字符串。

语法

upper(*char_expr*)

参数

char_expr — 是类型为 *char*、 *varchar*、 *nchar* 或 *nvarchar* 的字符型列名、变量或常量表达式。

示例

```
1. select upper("abcd")

   ----
   ABCD
```

注释

- **upper** 是一种字符串函数，它将小写转换为大写并返回字符值。
- 如果 *char_expr* 是 NULL，将返回 NULL。
- 有关字符串函数的一般信息，参见第 2-19 页上的 “字符串函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **upper**。

参见

函数	lower

used_pgs

功能

返回指定表及其集群索引所用的页数或返回非集群索引中的页数。

语法

used_pgs(*object_id, doampg, ioampg*)

参数

object_id — 是表的对象 ID 或索引所属表的对象 ID。

doampg — 是表或集群索引的对象分配映射的页数，它存储在 *sysindexes* 的 *doampg* 列中。

ioampg — 是非集群索引的分配映射页数，它存储在 *sysindexes* 的 *ioampg* 列中。

示例

1. ```
   select name, id, indid, doampg, ioampg
   from sysindexes where id = object_id("titles")
   ```

   ```
   name          id          indid  doampg   ioampg
   ------------- ----------- ------ -------- -------
   titleidind      208003772      1      560     552
   titleind        208003772      2        0     456
   ```

   ```
   select used_pgs(208003772, 560, 552)
   ```

   ```
   -----------
             6
   ```

   返回 *titles* 表的数据和集群索引所用的页数。

2. ```
   select name, id, indid, doampg, ioampg
   from sysindexes where id = object_id("stores")
   ```

   ```
   name          id          indid  doampg   ioampg
   ------------- ----------- ------ -------- -------
   stores          240003886      0     464       0
   ```

   ```
   select used_pgs(240003886, 464, 0)
   ```

   ```
   -----------
             2
   ```

   返回 *stores* 表所用的页数 （该表没有索引）。

注释

- **used_pgs** 是一种系统函数，它返回表及其集群索引所用的总页数或非集群索引中的页数。
- 在以上示例中， *indid* 0 表示表； *indid* 1 表示集群索引； *indid* 2-250 表示非集群索引；而 *indid* 255 表示 *text* 或 *image* 数据。
- **used_pgs** 仅用于当前数据库中的对象。
- 每个表和表中的每个索引都有一个对象分配映射 (OAM)，其信息包括分配给对象的页数和对象所用的页数。当分配或重新分配页时，多数 Adaptive Server 进程都将更新该信息。**sp_spaceused** 系统过程将读取这些值，以快速地进行空间估计。某些 **dbcc** 命令会在执行一致性检查时更新这些值。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **used_pgs**。

参见

函数	data_pgs、 object_id

user

功能

返回当前用户的名称。

语法

```
user
```

参数

无。

示例

```
1. select user
   ------
   dbo
```

注释

- **user** 是一种系统函数，它返回用户的名称。
- 如果 **sa_role** 处于活动状态，您将自动成为所用数据库的数据库所有者。在数据库中，数据库所有者的用户名总是 “dbo”。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	初级一致性

权限

任何用户都可以执行 **user**。

参见

函数	user_name

user_id

功能

返回数据库中指定用户或当前用户的 ID 号。

语法

```
user_id([user_name])
```

参数

user_name — 是用户的名称。

示例

```
1. select user_id()
   ------
        1
2. select user_id("margaret")
   ------
        4
```

注释

- **user_id** 是一种系统函数，它返回用户的 ID 号。有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。
- **user_id** 从 *sysusers* 中报告用户在当前数据库中的 ID 号。如果未提供 *user_name*， **user_id** 将返回当前用户的 ID。要查找服务器用户 ID （它在每个 Adaptive Server 数据库中都相同），可使用 **suser_id**。
- 在数据库中，“guest”的用户 ID 总是 2。
- 在数据库中，数据库所有者的 **user_id** 总是 1。如果 **sa_role** 处于活动状态，您将自动成为所用数据库的数据库所有者。要返回到实际的用户 ID，可在执行 **user_id** 之前使用 **set sa_role off**。如果您不是数据库中的有效用户，那么当您使用 **set sa_role off** 时，Adaptive Server 将返回错误。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

只有系统管理员或系统安全员才能以自己之外的 *user_name* 使用该函数。

参见

命令	setuser
函数	suser_id、 user_name

user_name

功能

返回指定用户或当前用户在数据库中的名称。

语法

```
user_name([user_id])
```

参数

user_id — 是用户的 ID。

示例

```
1. select user_name()

   ------------------------------
   dbo
2. select user_name(4)

   ------------------------------
   margaret
```

注释

- **user_name** 是一种系统函数，它根据用户在当前数据库中的用户 ID 来返回用户的名称。有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。
- 如果未提供 *user_id*，**user_name** 将返回当前用户的名称。
- 如果 **sa_role** 处于活动状态，您将自动成为所用数据库的数据库所有者。在数据库中，数据库所有者的 **user_name** 总是 “dbo”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

只有系统管理员或系统安全员才能以自己之外的 *user_id* 使用该函数。

参见

函数	suser_name、 user_id

valid_name

功能

当指定字符串不是有效标识符时返回 0；当该字符串是有效标识符时返回 0 以外的数字。

语法

valid_name(*character_expression*)

参数

character_expression — 是类型为 *char*、 *varchar*、 *nchar* 或 *nvarchar* 的字符型列名、变量或常量表达式。常量表达式必须用引号引起来。

示例

```
1. create procedure chkname
   @name varchar(30)
   as
        if valid_name(@name) = 0
        print "name not valid"
```

创建标识符有效性的验证过程。

注释

- **valid_name** 是一种系统函数。当 *character_ expression* 不是有效标识符时 （非法字符、长度大于 30 字节或保留字），它返回 0；如果是有效标识符，则返回非 0 数字。
- 无论是使用单字节字符还是使用多字节字符，Adaptive Server 标识符的长度最多可以是 30 个字节。标识符的首字符必须是字母字符 （如当前字符集中的定义），或是下划线 (_) 字符。临时表名以井号 (#) 开头，而局部变量名以 at 符号 (@) 开头，这两种情况不受该规则的限制。对于以井号 (#) 和以 at 符号 (@) 开头的标识符，**valid_name** 将返回 0。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

任何用户都可以执行 **valid_name**。

参见

系统过程	sp_checkreswords

valid_user

功能

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返回 1。

语法

valid_user(*server_user_id*)

参数

server_user_id — 是服务器用户 ID。服务器用户 ID 存储在 *syslogins* 的 *suid* 列中。

示例

```
1. select valid_user(4)
   ---------------
                 1
```

注释

- **valid_user** 是一种系统函数。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，它将返回 1。
- 有关系统函数的一般信息，参见第 2-20 页上的 “系统函数”。

标准和一致性

标准	一致性级别
SQL92	Transact-SQL 扩展

权限

只有系统管理员或系统安全员才能以自己之外的 *server_user_id* 使用该函数。

参见

系统过程	sp_addlogin、 sp_adduser

3 表达式、标识符和通配符

本章介绍 Transact-SQL 表达式、有效标识符和通配符。

表达式

表达式是由运算符分隔开的常量、文字、函数、列标识符和 / 或变量的组合，一个表达式将返回单个值。表达式有几种类型，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，可以在表达式中使用子查询。在表达式中可使用 **case** 表达式。

表 3-1 列出了在 Adaptive Server 语法语句中使用的表达式类型。

表 3-1：在语法语句中使用的表达式类型

用法	定义
expression	可以包括常量、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

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}]...
```

关系表达式和逻辑表达式

逻辑表达式和关系表达式返回 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***

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 （单个参数） - + ~
2. * / %
3. 二元运算符 （两个参数） + - & | ^
4. **not**
5. **and**
6. **or**

当一个表达式中的所有运算符属于同一级别时，执行顺序为从左向右。可使用小括号来更改执行顺序 — 嵌套在最里面的表达式将最先执行。

算术运算符

Adaptive Server 使用以下算术运算符：

表 3-2：算术运算符

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

另外，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。

模运算符不能用于 *smallmoney*、*money*、*float* 或 *real* 列。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的类型。有关详细信息，参见第 1 章 “系统和用户定义的数据类型”。

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 3-3 总结了操作数 0 和 1 的结果。如果任一个操作数为 NULL，逐位运算符将返回 NULL：

表 3-3：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	FALSE
0	0

表 3-4 中的示例使用了两个 *tinyint* 参数， A = 170 （二进制形式为 10101010）和 B = 75 （二进制形式为 01001011）。

表 3-4：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 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 为 1，但二者不同时为 1，则结果列等于 1。
(~A)	10101010 ---------- 01010101	85	所有 1 变为 0，所有 0 变为 1

字符串并置运算符

字符串运算符 **+** 可用于并置两个或更多个字符或二进制表达式。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”。空字符串被解释为在所有 *char*、*varchar*、 *nchar*、 *nvarchar* 和 *text* 并置以及在 *varchar* 插入和赋值语句中的单个空格。

在并置非字符、非二进制表达式时，总是使用 **convert**：

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

与 NULL 并置的字符串求值为字符串的值。这是 SQL 标准的一个例外，该标准规定与 NULL 并置的字符串应求值为 NULL。

比较运算符

Adaptive Server 使用在表 3-5 中列出的比较运算符：

表 3-5：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，**<** 表示接近服务器排序顺序的开头，而 **>** 表示接近服务器排序顺序的结尾。如果排序顺序不区分大小写，则大写字母与小写字母相同。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。所以，“Dirk”与 “Dirk ” 相同。

在比较日期时，**<** 表示较早而 **>** 表示较晚。

用单引号或双引号引起所有使用比较运算符的字符和 *datetime* 数据：

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

非标准运算符

以下运算符为 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 **=***

使用 *any*、 *all* 和 *in*

any 与 **<**、**>** 或 **=** 以及子查询一起使用。当在子查询中检索到的任何值与外部语句的 **where** 或 **having** 子句中的值相匹配时，它就会返回结果。有关详细信息，参见 *Transact-SQL User's Guide*。

all 与 **<** 或 **>** 以及子查询一起使用。当在子查询中检索到的任何值小于 **(<)** 或大于 **(>)** 外部语句的 **where** 或 **having** 子句中的值时，它就会返回结果。有关详细信息，参见 *Transact-SQL User's Guide*。

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**。有关详细信息，参见 "where 子句"。

否定和测试

not 否定关键字或逻辑表达式的含义。

使用后接子查询的 **exists** 可测试是否存在特定的结果。

范围

between 是表示范围开始的关键字； **and** 是表示范围结束的关键字。范围：

```
where column1 between x and y
```

包括 x 和 y。

范围：

```
where column1 > x and column1 < y
```

不包括 x 和 y。

在表达式中使用 Null

使用 **is null** 或 **is not null** 对列查询，这些列已被定义为允许空值。

如果任何操作数为空，则带有逐位运算符或算术运算符的表达式求值结果为 NULL。例如：

```
1 + column1
```

如果 *column1* 为 NULL，该示例求值结果为 NULL。

返回 TRUE 的比较

一般情况下，比较空值所得的结果是 UNKNOWN，这是因为无法确定 NULL 是等于 （或不等于）给定值还是等于 （或不等于）另一个 NULL。然而，如果 *expression* 是任何求值结果为 NULL 的列、变量或文字 （或这些数据类型的组合），在下列情况下将返回 TRUE：

- *expression* **is null**
- *expression* = **null**
- *expression* = *@x*，其中 *@x* 是包含 NULL 的变量或参数。该例外有利于用空缺省参数来编写存储过程。
- *expression* != *n*，其中 *n* 是不包含 NULL 的文字，并且 *expression* 求值结果为 NULL。

当表达式求值结果不为 NULL 时，这些表达式的否定版本将返回 TRUE：

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

注意，这些例外的最右侧是文字空或包含 NULL 的变量或参数。如果比较的最右侧是表达式 （例如 *@nullvar* + 1），则整个表达式求值结果为 NULL。

按照这些规则，空列值不与其它空列值连接。在 **where** 子句中对空列值与其它空列值进行比较时，无论比较运算符是什么，总会为空值返回 UNKNOWN，并且结果中不包括行。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
```

FALSE 和 UNKNOWN 之间的区别

虽然 FALSE 和 UNKNOWN 都不返回值，但它们之间却存在重要的逻辑区别，因为与 false 相反 (“not false”) 的是 true。例如，“1 = 2”求值结果为 false，而与其相反的 “1 != 2”求值结果为 true。但是 “not unknown”仍为 unknown。如果比较中包括空值，则不能通过否定表达式得到行的相反集或相反的真值。

将“NULL”用作字符串

只有在 **create table** 语句中指定 NULL 且显式输入 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”、“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。

比较 NULL 和空字符串

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：

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

求值为“abc def”，而不是“abcdef”。空字符串从不会求值为 NULL。

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。可使用小括号更改执行顺序。

表 3-6 显示了逻辑运算的结果，其中包括那些含空值的表达式：

表 3-6：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

表 3-6：逻辑表达式真值表 （续）

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。有关详细信息，参见第 3-6 页上的 “在表达式中使用 Null”。

在表达式中使用小括号

可以在表达式中使用小括号将元素分组。当把 “表达式”当作语法语句中的变量提供时，将视其为简单表达式。当只接受逻辑表达式时，则指定 “逻辑表达式”。

比较字符表达式

字符常量表达式被当作 *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。

在比较 *char* 表达式和 *varchar* 表达式时，应遵照数据类型优先级：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 (‘’) 被解释为 *varchar* 数据的 **insert** 或赋值语句中的单个空格。在 *varchar*、 *char*、 *nchar*、 *nvarchar* 数据的并置中，空字符串被解释为单个空格；例如：

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

被存储为 “abc def”。空字符串从不会求值为 NULL。

在字符表达式中包括双引号

在 *char* 或 *varchar* 条目中，有两种指定文字引号的方法。第一种方法是使用两个（对）引号。例如，如果用单引号开始某个字符条目并要将单引号包括在该条目中，可使用两个单引号：

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

对于双引号：

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

第二个方法是用相反类型的引号来引起引号。也就是说，用单引号（或相反）引起包含双引号的条目。以下是些示例：

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

使用延续字符

要使一个字符串延续到屏幕上的下一行，可在转入下一行之前输入反斜杠 (\)。

标识符

标识符是数据库对象 （如数据库、表、视图、列、索引、触发器、过程、缺省值、规则和游标）的名称。

Adaptive Server 标识符最长可为 30 个字节 （无论是使用单字节字符还是使用多字节字符）。标识符的首字符必须是字母字符 （如当前字符集中的定义），或是下划线 (_) 字符。

➤ ***注意***

临时表名以井号 (#) 开头，而局部变量名以 at 符号 (@) 开头，这两种情况不受该规则的限制。

后续字符可以包括字母、数字、符号 #、@、_ 和货币符号，如 $（美元）、¥（日元）和 £（英镑）。标识符不能包括特殊字符，如 !、%、 ^、 &、 * 和 .，也不能包括嵌入的空格。

不能将保留字 （如 Transact-SQL 命令）用作标识符。如需保留字的完整列表，参见第 4 章 “保留字”。

以 # 开头的表 （临时表）

名称以井号 (#) 开头的表是临时表。在创建其它类型的对象时，其名称不能以井号开头。

Adaptive Server 对临时表名执行特殊操作，从而在每个会话中对其命以唯一的名称。长临时表名将截断至 13 个字符 （包括井号）；而短名称将用下划线 (_) 填充至 13 个字符。将附加一个 17 位的数字后缀，它对于每个 Adaptive Server 会话都是唯一的。

区分大小写和标识符

标识符和数据是否区分大小写 （大写或小写）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，可以更改单字节字符集的区分大小写设置 （有关详细信息，参见*系统管理指南*）。在实用程序选项中，大小写十分重要。

当 Adaptive Server 安装不区分大小写的排序顺序后，如果已存在名为 MyTable 或 mytable 的表，则不能创建名为 *MYTABLE* 的表。同样，以下命令：

```
select * from MYTABLE
```

将返回名称为 *MYTABLE*、 MyTable 或 mytable （或名称中的任意大小写字母组合）的表中的行。

对象名称的唯一性

数据库中的对象名称不必唯一。但是，一个表中的列名和索引名必须唯一，并且其它对象名称对于一个**数据库**中的每个**所有者**都必须是唯一的。 Adaptive Server 上的数据库名称必须唯一。

使用分隔标识符

分隔标识符是用双引号引起的对象名称。使用分隔标识符可避免对象名称的某些特定限制。表名、视图名和列名可用引号来分隔；而其它对象名称则不行。

分隔标识符可以是保留字，它们可以以非字母字符开头，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。

◆ ***警告！***

分隔标识符可能不会被所有前端应用程序识别，并且不应用作系统过程的参数。

在创建或引用分隔标识符之前，必须执行：

```
set quoted_identifier on
```

每当在语句中使用分隔标识符时，都必须将其放在双引号中。例如：

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

create table "grant"("add" int)
insert "grant"("add") values (3)
```

在启用 **quoted_identifier** 选项时，不要用双引号引起字符或日期字符串，而应使用单引号。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 *1table* 的 *col1* 中插入一个字符串，应使用：

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

而不是：

```
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，在 *col1* 中插入字符 “a’b”，应使用：

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

使用限定的对象名称

通过增加其它限定性名称 — 如数据库名称、所有者名称和列的表名或视图名，可以唯一地标识表或列。每个限定符通过句号与下一个限定符分隔开。例如：

```
database.owner.table_name.column_name

database.owner.view_name.column_name
```

命名约定是：

```
[[database.]owner.]table_name

[[database.]owner.]view_name
```

在对象名称中使用分隔标识符

如果使用 **set quoted_identifier on**，则可以用双引号引起限定对象名称的各个部分。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"
```

而不是：

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

省略所有者名称

只要系统具有足够的信息来标识对象，就可以在名称中省略中间元素并用圆点来表示它们的位置：

database..table_name

database..view_name

在当前数据库中引用您自己的对象

在当前数据库中引用您自己的对象时，不需要使用数据库名称或使用者名称。 *owner* 的缺省值是当前用户，而 *database* 的缺省值是当前数据库。

在引用对象时，如果没有用数据库名称和使用者名称来限定该对象，Adaptive Server 将从您拥有的当前数据库对象之中查找该对象。

引用数据库所有者拥有的对象

如果您省略了所有者名称但却不拥有具备该名称的对象， Adaptive Server 将查找数据库所有者所拥有的具备该名称的对象。只有在您拥有同名对象，却希望使用数据库所有者拥有的对象时，才必须限定数据库所有者拥有的对象。但是，无论您是否拥有同名对象，都必须使用该用户名限定其它用户拥有的对象。

一致地使用限定标识符

在同一语句中限定列名和表名时，务必要对列名和表名使用相同的限定表达式。这些表达式必须求值为字符串且相互匹配；否则，将返回错误。下面的第二个示例不正确，因为其列名的语法样式与表名的语法样式不匹配。

```
1. select demo.mary.publishers.city
   from demo.mary.publishers

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

2. select demo.mary.publishers.city
   from demo..publishers

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

确定标识符是否有效

更改字符集之后或创建表或视图之前，应使用系统函数 **valid_name** 来确定 Adaptive Server 是否可接受该对象名称。语法如下：

```
select valid_name("Object_name")
```

如果 *object_name* 是无效的标识符 （例如，它包含非法字符或长度超过 30 个字节）， Adaptive Server 将返回 0。如果 *object_name* 是有效的标识符， Adaptive Server 将返回一个非零数字。

重命名数据库对象

用 **sp_rename** 重命名用户对象 （包括用户定义的数据类型）。

◆ ***警告！***

将表或列重命名后，务必要重新定义任何依赖于重命名对象的过程、触发器和视图。

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。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、 Greek、 Cyrillic 或 ASCII。

虽然在日文系统中，将 Hankaku Katakana 字符用作标识符是合法的，但最好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。

有些 8 位的欧洲字符集也是如此。例如，字符 “Œ”（OE 连字）存在于 Macintosh 字符集中 （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，客户端将失去对该对象的直接访问权。

与通配符匹配的模式

在 *match_string* 中，通配符表示一个或多个字符，或字符范围。*match_string* 是包含要在表达式中查找的模式的字符串。它可以是常量、变量和列名的任意组合，或是并置的表达式，例如：

```
like @variable + "%".
```

如果匹配字符串是常量，则必须始终用单引号或双引号将其引起来。

使用带有 **like** 关键字的通配符可查找与特定模式匹配的字符和日期字符串。但不能使用 **like** 来搜索秒或毫秒 （参见第 3-20 页上的 “将通配符用于 datetime 数据”）。

在 **where** 和 **having** 中使用通配符来查找与匹配字符串 **like** （或 **not like**）的字符或日期 / 时间信息：

```
{where | having} [not]
   expression [not] like match_string
       [escape "escape_character"]
```

expression 可以是列名、常量或带有字符值的函数的任意组合。

未使用 **like** 的通配符不具有特殊的含义。例如，以下查询查找任何以 “415%” 四个字符开头的电话号码：

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

使用 *not like*

使用 **not like** 可查找与特定模式不匹配的字符串。下面两个查询是等同的：它们都在 *authors* 表中查找所有不以区号 415 开头的电话号码。

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

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

例如，以下查询在数据库中查找其名称以 “sys” 开头的系统表：

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

要查找**不是**系统表的所有对象，可使用：

```
not like "sys%"
```

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

not like 和否定通配符 **[^]** 可能会产生不同的结果 （参见第 3-18 页上的“尖号 (^) 通配符”）。不能总用 **like** 和 **^** 来复制 **not like** 模式。因为 **not like** 查找与整个 **like** 模式不匹配的项，而带有否定通配符的 **like** 一次只对一个字符求值。

例如，模式 **like "[^s][^y][^s]%"** 会产生不同的结果。它得到的结果是 14，而不是 19，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”的名称以及系统表名称。这是因为带否定通配符的匹配字符串将分步求值 （一次对一个字符求值）。如果在求值的任一步匹配失败，则将消除该匹配项。

不区分大小写和变音

如果 Adaptive Server 使用不区分大小写的排序顺序，那么在比较 *expression* 和 *match_string* 时将忽视大小写。例如，以下子句：

```
where col_name like "Sm%"
```

将在不区分大小写的 Adaptive Server 上返回 “Smith”、“smith” 和 “SMITH”。

如果 Adaptive Server 同时还不区分变音，则会将所有变音及其对应的非变音字符 （大写和小写）视为等同。 **sp_helpsort** 系统过程显示被视为等同的字符，并在它们之间显示 “=”。

使用通配符

在匹配字符串中可以使用多种通配符，以下各节将详细讨论这些通配符。表 3-7 对通配符进行了总结：

表 3-7：用于 like 的通配符

符号	含义
%	任何包含 0 个或多个字符的字符串
_	任何单个字符
[ ]	指定范围 ([a-f]) 或集合 ([abcdef]) 内的任何单个字符
[^]	不在指定范围 ([^a-f]) 或集合 ([^abcdef]) 内的任何单个字符

用单引号或双引号引起通配符和匹配字符串 (**like** “[dD]eFr_nce“)。

百分号 (%) 通配符

使用%通配符可表示任何包含 0 个或多个字符的字符串。例如，在 *authors* 表中查找所有以 415 区号开头的电话号码：

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

要查找包含字符 “en”的名称 （Bennet、 Green、 McBadden）：

```
select au_lname
from authors
where au_lname like "%en%"
```

like 子句中 “%”后的尾随空白将被截断成单个尾随空白。例如，后接两个空格的 “%”匹配 “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。

下划线 (_) 通配符

使用 _ 通配符可表示任何单个字符。例如，要查找所有以 “hery1”结尾的六字母姓名 （例如 Chery1）：

```
select au_fname
from authors
where au_fname like "_heryl"
```

中括号 ([ ]) 中的字符

使用中括号可括起字符范围 （如 [a-f]）或字符集合 （如 [a2Br]）。当使用范围时，将返回排序顺序中在 *rangespec1* 和 *rangespec2* 之间（包含二者）的所有值。例如，“[0-z]”匹配 7 位 ASCII 中的 0-9、A-Z 和 a-z （以及几个标点字符）。

要查找以 “inger”结尾并以 M 和 Z 间的任意单个字符开头的姓名：

```
select au_lname
from authors
where au_lname like "[M-Z]inger"
```

要同时查找 “DeFrance”和 “deFrance”：

```
select au_lname
from authors
where au_lname like "[dD]eFrance"
```

尖号 (^) 通配符

尖号是否定通配符。它用于查找与特定模式不匹配的字符串。例如，“[^a-f]”查找不在 a-f 范围内的字符串，“[^a2bR]”查找不是“a”、“2”、“b”或“R”的字符串。

要查找以“M”开头且第二个字母不是“c”的姓名：

```
select au_lname
from authors
where au_lname like "M[^c]%"
```

当使用范围时，将返回排序顺序中在 *rangespec1* 和 *rangespec2* 之间（包含二者）的所有值。例如，“[0-z]”匹配 7 位 ASCII 中的 0-9、A-Z、 a-z 和几个标点字符。

使用多字节通配符

如果在 Adaptive Server 上配置的多字节字符集为通配符 _、 %、 - [、] 和 ^ 定义了等同的双字节字符，则可以在匹配字符串中替代等同的字符。下划线的等同字符表示匹配字符串中的单字节或双字节字符。

将通配符用作文字字符

要在字符串中搜索 %、 _、 [、] 或 ^，必须使用转义字符。当通配符与转义字符结合使用时， Adaptive Server 将直接解释通配符，而不用它来表示其它字符。

Adaptive Server 提供了两种转义字符：

- 中括号 （Transact-SQL 扩展）
- 紧接在 **escape** 子句后的任何单个字符 （符合 SQL 标准）

将中括号用作转义字符

将中括号用作百分号、下划线和左中括号的转义字符。右中括号不需要转义字符；使用其本身即可。如果将连字符用作文字字符，它必须是一组中括号中的首字符。

表 3-8 显示了将中括号用作转义字符的一些示例：

表 3-8：使用中括号来搜索通配符

like 谓词	含义
like "5%"	5 后接有包含 0 个或多个字符的字符串
like "5[%]"	5%
like "_n"	an、 in、 on 等
like "[_]n"	_n
like "[a-cdf]"	a、 b、 c、 d 或 f
like "[-acdf]"	-、 a、 c、 d 或 f
like "[[]"	[
like "]"	]
like "[[]ab]"_	[]ab

使用 *escape* 子句

使用 **escape** 子句可指定转义字符。服务器缺省字符集中的任何单个字符都可用作转义字符。如果试图将多个字符用作一个转义字符，Adaptive Server 就会产生例外。

不要将现有通配符用作转义字符，原因如下：

- 如果将下划线 (_) 或百分号 (%) 指定为转义字符，它将在该 **like** 谓词内失去其特殊含义，而仅充当一个转义字符。
- 如果将左中括号或右中括号 （[或]）指定为转义字符，中括号的 Transact-SQL 含义将在该 **like** 谓词中禁用。
- 如果将连字符 (-) 或尖号 (^) 指定为转义字符，它将失去其特殊含义，而仅充当一个转义字符。

与下划线、百分号和左 / 右中括号等通配符不同，转义字符将在中括号中保留其特殊含义。

转义字符只在自己的 **like** 谓词中才有效，而对于同一语句包含的其它 **like** 谓词则无效。在转义字符之后，只有通配符（_、 %、 [、] 或 [^]）和转义字符本身才有效。转义字符只影响它后面一个字符，而不受影响接下去的字符。

如果字符在模式中两次作为转义字符出现，该字符串就必须包含四个连续的转义字符。如果转义字符没有将模式划分为包含一个或两个字符的区段， Adaptive Server 将返回错误消息。

以下是带 **escape** 子句的 **like** 谓词的示例：

表 3-9：使用 escape 子句

like 谓词	含义
like "5@%" escape "@"	5%
like "*_n" escape "*"	_n
like "%80@%%" escape "@"	字符串包含 80%
like "*_sql%" escape "*"**	字符串包含 _sql*
like "%#####_#%%" escape "#"	字符串包含 ##_%

将通配符用于 *datetime* 数据

如果将 **like** 用于 *datetime* 值， Adaptive Server 就会把日期转换为标准的 *datetime* 格式，然后转换为 *varchar*。由于标准的存储格式不包括秒或毫秒，所以不能用 **like** 和模式来搜索秒或毫秒。

由于 *datetime* 条目可能包含多种日期分量，所以最好使用 **like** 来搜索 *datetime* 值。例如，在名为 *arrival_time* 的列中插入值 “9:20”和当前日期，以下子句：

```
where arrival_time = '9:20'
```

将无法找到值，因为 Adaptive Server 会将该条目转换为 “Jan 1 1900 9:20AM”。但以下子句却可以找到该值：

where arrival_time like '%9:20%'

4 保留字

关键字也称为保留字，它们是具有特殊意义的字词。本章将列出 Transact-SQL 和 SQL92 的关键字。

Transact-SQL 关键字

下表中的字词是 Adaptive Server 所保留的关键字，它们是 SQL 命令语法的一部分。这些字词不能用作数据库对象 （如数据库、表、规则或缺省值）的名称，但可以用作局部变量名和存储过程参数名。

要查找其名称是保留字的现有对象，可使用 **sp_checkreswords**。

A

add、 all、 alter、 and、 any、 arith_overflow、 as、 asc、 at、 authorization、 avg

B

begin、 between、 break、 browse、 bulk、 by

C

cascade、 case、 char_convert、 check、 checkpoint、 close、 clustered、 coalesce、 commit、 compute、 confirm、 connect、 constraint、 continue、 controlrow、 convert、 count、 create、 current、 cursor

D

database、 dbcc、 deallocate、 declare、 default、 delete、 desc、 disk distinct、 double、 drop、 dummy、 dump

E

else、 end、 endtran、 errlvl、 errordata、 errorexit、 escape、 except、 exclusive、 exec、 execute、 exists、 exit、 exp_row_size、 external

F

fetch、 fillfactor、 for、 foreign、 from

G

goto、 grant、 group

H

having、 holdlock

I

identity、identity_gap、identity_insert、identity_start、if、in、index、insert、install、intersect、into、is、isolation

J

jar、join

K

key、kill

L

level、like、lineno、load、lock

M

max、max_rows_per_page、min、mirror、mirrorexit、modify

N

national、noholdlock、nonclustered、not、null、nullif、numeric_truncation

O

of、off、offsets、on、once、online、only、open、option、or、order、over

P

partition、perm、permanent、plan、precision、prepare、primary、print privileges、proc、procedure、processexit、proxy_table、public

Q

quiesce

R

raiserror、read、readpast、readtext、reconfigure、references remove、reorg、replace、replication、reservepagegap、return、revoke、role、rollback、rowcount、rows、rule

S

save、schema、select、set、setuser、shared、shutdown、some、statistics、stripe、sum、syb_identity、syb_restree

T

table、temp、temporary、textsize、to、tran、transaction、trigger、truncate、tsequal

U

union、unique、unpartition、update、use、user、user_option、using

V

values、varying、view

W

waitfor、when、where、while、with、work、writetext

SQL92 关键字

Adaptive Server 引入了初级的 SQL92 功能。 SQL92 的全部实现包括以下各表中作为命令语法列出的字词。标识符升级可能是一个复杂的过程。为了便于完成该过程，我们提供了以下列表。这一信息的公布并不表示 Sybase 承诺将在后继的版本中提供所有这些 SQL92 功能。另外，后继版本还可能包含下表中未列出的关键字。

该列表中的 SQL92 关键字在 Transact-SQL 中不是保留字。

A

absolute、 action、 allocate、 are、 assertion

B

bit、 bit_length、 both

C

cascaded、 case、 cast、 catalog、 char、 char_length、 character、 character_length、 coalesce、 collate、 collation、 column、 connection、 constraints、 corresponding、 cross、 current_date、 current_time、 current_timestamp、 current_user

D

date、 day、 dec、 decimal、 deferrable、 deferred、 describe、 descriptor、 diagnostics、 disconnect、 domain

E

end-exec、 exception、 extract

F

false、 first、 float、 found、 full

G

get、 global、 go

H

hour

I

immediate、indicator、initially、inner、input、insensitive、int、integer、 interval

J

join

L

language、 last、 leading、 left、 local、 lower

M

match、 minute、 module、 month

N

names、 natural、 nchar、 next、 no、 nullif、 numeric

O

octet_length、 outer、 output、 overlaps

P

pad、 partial、 position、 preserve、 prior

R

real、 relative、 restrict、 right

S

scroll、second、section、session_user 、size 、smallint、space、sql、 sqlcode、 sqlerror、 sqlstate、 substring、 system_user

T

then、 time、 timestamp、 timezone_hour、 timezone_minute、trailing、 translate、 translation、 trim、 true

U

unknown、 upper、 usage

V

value、 varchar

W

when、 whenever、 write、 year

Z

zone

潜在的 SQL92 保留字

如果您使用的是 ISO / IEC 9075:1989 标准，还应避免使用下表中列出的字词，因为它们可能在以后成为 SQL92 保留字。

A

after、 alias、 async

B

before、 boolean、 breadth

C

call、 completion、 cycle

D

data、 depth、 dictionary

E

each、 elseif、 equals

G

general

I

ignore

L

leave、 less、 limit、 loop

M

modify

N

new、 none

O

object、 oid、 old、 operation、 operators、 others

P

parameters、 pendant、 preorder、 private、 protected

R

recursive、ref、referencing、resignal、return、returns、routine、row

S

savepoint、 search、 sensitive、 sequence、 signal、 similar、 sqlexception、 structure

T

test、 there、 type

U

under

V

variable、 virtual、 visible

W

wait、 without

5 SQLSTATE 代码和消息

本章介绍 Adaptive Server 的 SQLSTATE 状态代码及其相关的消息。SQLSTATE 代码是符合初级 SQL92 所必需的。它们提供有关以下两种条件的诊断信息：

- **警告** — 需要用户注意但尚未严重到妨碍 SQL 语句成功执行的条件
- **例外** — 使 SQL 语句无法对数据库发生作用的条件

每个 SQLSTATE 代码都由双字符类以及随后的三字符子类组成。类指定有关错误类型的一般信息。子类指定更具体的信息。

SQLSTATE 代码存储在 *sysmessages* 系统表中，一同存储的还有将在检测到这些条件时显示的消息。并非所有 Adaptive Server 错误条件都与 SQLSTATE 代码相关 — 只有 SQL92 所指定的错误条件才与其相关。在某些情况下，会有多个 Adaptive Server 错误条件与一个 SQLSTATE 值相关。

警告

Adaptive Server 当前只检测一个 SQLSTATE 警告条件，如表 5-1 所述：

表 5-1：SQLSTATE 警告

消息	值	说明
Warning - null value eliminated in set function.（警告 — 在 set 函数中消除了空值。）	01003	发生条件是：对具有空值的表达式使用集合函数（**avg**、 **max**、 **min**、 **sum** 或 **count**）。

例外

Adaptive Server 将检测以下类型的例外：

- 基数冲突
- 数据例外
- 完整性约束冲突
- 无效的游标状态

- 语法错误和访问规则冲突
- 事务回退
- **with check option** 冲突

表 5-2 到表 5-8 将对例外条件进行说明。每个例外类都显示在它自己的表中。在每个表中，条件都按照消息文本的字母顺序来进行排序。

基数冲突

发生基数冲突的条件是：应该只返回一行的查询将多个行返回到 Embedded SQL™ 应用程序中。

表 5-2：基数冲突

消息	值	说明
Subquery returned more than 1 value.（子查询返回多个值。） This is illegal when the subquery follows =, !=, <, <=, >, >=. or when the subquery is used as an expression.（当子查询跟在 =、!=、<、<=、>、>= 之后时或当子查询用作表达式时，这种情况是非法的。）	21000	发生条件： • 标量子查询或行子查询返回多个行。 • Embedded SQL 中的 **select into parameter_list** 查询返回多个行。

数据例外

发生数据例外的条件是：

- 输入太长，不适合其数据类型；
- 输入包含非法的转义序列；
- 输入包含其它格式错误。

表 5-3：数据例外

消息	值	说明
Arithmetic overflow occurred.（发生算术溢出。）	22003	发生条件： • 算术运算或 **sum** 函数使精确数值类型丢失精度或标度。 • 截断、舍入或 **sum** 函数使近似数值类型丢失精度或标度。
Data exception - string data right truncated.（数据例外 — 字符串数据被从右侧截断。）	22001	发生条件是：*char* 或 *varchar* 列太短，不能容纳插入或更新的数据，因而必须截断非空字符。

表 5-3：数据例外 （续）

消息	值	说明
Divide by zero occurred.（发生除零错误。）	22012	发生条件是：对数字表达式求值而除数的值为零。
Illegal escape character found.（发现非法的转义字符。） There are fewer bytes than necessary to form a valid character.（没有足够数量的字节来形成有效字符。）	22019	发生条件是：在转义序列不是由单个字符构成时，搜索与给定模式相匹配的字符串。
Invalid pattern string.（无效的模式字符串。） The character following the escape character must be percent sign, underscore, left square bracket, right square bracket, or the escape character.（转义字符后的字符必须是百分号、下划线、左中括号、右中括号或转义字符。）	22025	发生条件是：在搜索与特定模式相匹配的字符串时， • 转义字符后没有紧接百分号、下划线或转义字符本身，或者 • 转义字符将模式分隔为长度不是 1 或 2 个字符的子串。

完整性约束冲突

发生完整性约束冲突的条件是：**insert**、 **update** 或 **delete** 语句违反 **primary key**、 **foreign key**、 **check**、 **unique** 约束或唯一索引。

表 5-4：完整性约束冲突

消息	值	说明
Attempt to insert duplicate key row in object *object_name* with unique index *index_name*（试图在具有唯一索引 *index_name* 的对象 *object_name* 中插入重复的键行。）	23000	发生条件是：在具有唯一约束或索引的表中插入重复的行。
Check constraint violation occurred, dbname = *database_name*, table name = *table_name*, constraint name = *constraint_name* （发生检查约束冲突，dbname = *database_name*、table name = *table_name*、 constraint name = *constraint_name*）	23000	发生条件是：**update** 或 **delete** 违反列的检查约束。
Dependent foreign key constraint violation in a referential integrity constraint.（违反参照完整性约束中的相关外键约束。） dbname = *database_name*, table name = *table_name*, constraint name = *constraint_name*	23000	发生条件是：主键表上的 **update** 或 **delete** 违反外键约束。

表 5-4：完整性约束冲突 （续）

消息	值	说明
Foreign key constraint violation occurred, dbname = *database_name*, table name = *table_name*, constraint name = *constraint_name*（发生外键约束冲突， dbname = *database_name*、 table name = *table_name*、 constraint name = *constraint_name*）	23000	发生条件是：在主键表中无匹配值的情况下对外键表执行 **insert** 或 **update**。

无效的游标状态

发生无效游标状态的条件是：

- **fetch** 使用当前未打开的游标，或者
- **update where current of** 或 **delete where current of** 影响已修改或删除的游标行，或者
- **update where current of** 或 **delete where current of** 影响尚未读取的游标行。

表 5-5：无效的游标状态

消息	值	说明
Attempt to use cursor *cursor_name* which is not open.（试图使用未打开的游标 *cursor_name*。） Use the system stored procedure sp_cursorinfo for more information.（如需详细信息，请使用系统存储过程 sp_cursorinfo。）	24000	发生条件是：试图在从未打开或者已经被 **commit** 语句或隐式或显式 **rollback** 关闭的游标中进行读取。重新打开游标并重新执行 **fetch**。
Cursor *cursor_name* was closed implicitly because the current cursor position was deleted due to an update or a delete.（当前游标位置因执行 update 或 delete 而被删除，因此游标 *cursor_name* 被隐式关闭。） The cursor scan position could not be recovered.（游标扫描位置不能恢复。）This happens for cursors which reference more than one table.（游标在引用多个表时就会发生这种情况。）	24000	发生条件是：多表游标的连接列已被删除或更改。发出另一 **fetch** 命令来重新定位游标。

表 5-5：无效的游标状态 （续）

消息	值	说明
The cursor *cursor_name* had its current scan position deleted because of a DELETE / UPDATE WHERE CURRENT OF or a regular searched DELETE / UPDATE.（游标 *cursor_name* 的当前扫描位置因执行 DELETE / UPDATE WHERE CURRENT OF 或常规搜索的 DELETE / UPDATE 而被删除。） You must do a new FETCH before doing an UPDATE or DELETE WHERE CURRENT OF.（在执行 UPDATE 或 DELETE WHERE CURRENT OF 之前，必须首先执行新的 FETCH。）	24000	发生条件是：用户发出的 **update/delete where current of** 的当前游标位置已被删除或更改。在重新尝试 **update/delete where current of** 之前，需发出另一 **fetch**。
The UPDATE / DELETE WHERE CURRENT OF failed for the cursor *cursor_name* because it is not positioned on a row.（未能对游标 *cursor_name* 执行 UPDATE / DELETE WHERE CURRENT OF，因为它不在行上。）	24000	发生条件是：用户对其发出 **update/delete where current of** 的游标 • 尚未读取行 • 在到达结果集结尾后已读取了一个或多个行

语法错误和访问规则冲突

当 SQL 语句包含未终止的注释、Adaptive Server 不支持的隐式数据类型转换或其它错误的语法时，将导致语法错误。

当用户试图访问不存在的对象或对其没有正确权限的对象时，将导致访问规则冲突。

表 5-6：语法错误和访问规则冲突

消息	值	说明
command permission denied on object *object_name*, database *database_name*, owner *owner_name*.（对象 *object_name*、数据库 *database_name*、所有者 *owner_name* 上的 *command* 权限被拒绝。）	42000	发生条件是：用户试图访问他们没有正确权限的对象。
Implicit conversion from datatype '*datatype*' to '*datatype*' is not allowed.（不允许执行从数据类型 “*datatype*” 到 “*datatype*” 的隐式转换。）Use the CONVERT function to run this query.（使用 CONVERT 函数来运行该查询。）	42000	发生条件是：用户试图将一种数据类型转换为另一种数据类型，但 Adaptive Server 无法隐式执行该转换。
Incorrect syntax near *object_name*.（*object_name* 附近的语法不正确。）	42000	发生条件是：在指定对象附近找到错误的 SQL 语法。

表 5-6：语法错误和访问规则冲突 （续）

消息	值	说明
Insert error: （插入错误：） column name or number of supplied values does not match table definition. （列名或所提供值的数量与表定义不匹配。）	42000	发生条件是：使用了无效的列名或插入了不正确数量的值。
Missing end comment mark “*/”（丢失结束注释符 “*/”）	42000	发生条件是：以 /* 起始分隔符开始的注释没有 */ 结束分隔符。
object_name not found. （未找到 *object_name*。） Specify owner.objectname or use sp_help to check whether the object exists (sp_help may produce lots of output). （指定 owner.objectname 或使用 sp_help 来检查对象是否存在 （sp_help 可能会产生很多输出）。）	42000	发生条件是：用户试图引用他们不拥有的对象。当引用其他用户所拥有的对象时，务必要用其所有者的名称来限定对象名。
The size (*size*) given to the *object_name* exceeds the maximum. （为 *object_name* 指定的大小 (*size*) 超出最大值。） The largest size allowed is *size*. （所允许的最大值为 *size*。）	42000	发生条件： • 表定义中所有列的大小总和超出所允许的行大小最大值。 • 单个列或参数的大小超出其数据类型所允许的最大值。

事务回退

发生事务回退的条件是：将 **transaction isolation level** 设置为 3 时，但 Adaptive Server 无法确保并发事务可以序列化。这种例外通常是由系统问题 （如磁盘崩溃和脱机磁盘）造成的。

表 5-7：事务回退

消息	值	说明
Your server command (process id #*process_id*) was deadlocked with another process and has been chosen as deadlock victim. （您的服务器命令 （进程 id 号 *process_id*）与其它进程发生死锁并且已被选作死锁牺牲品。） Re-run your command. （重新运行您的命令。）	40001	发生条件是：Adaptive Server 发现它无法确保两个或两个以上的并发事务可以序列化。

with check option 冲突

发生此类例外的条件是：无法通过视图查看用该视图插入或更新的数据。

表 5-8：with check option 冲突

消息	值	说明
The attempted insert or update failed because the target view was either created WITH CHECK OPTION or spans another view created WITH CHECK OPTION.（插入或更新尝试失败，因为目标视图通过 WITH CHECK OPTION 创建或跨越其它用 WITH CHECK OPTION 创建的视图。）At least one resultant row from the command would not qualify under the CHECK OPTION constraint.（至少有一个从该命令生成的行不满足 CHECK OPTION 约束。）	44000	发生条件是：用 **with check option** 子句创建视图或它所依赖的任何视图。

如需索引，参见第四卷“表格和参考手册索引”。

第四卷“表格和参考手册索引”包括 *Adaptive Server Enterprise 参考手册* 中所有卷的索引条目。