SYBASE®
An SAP® Company

参考手册：构件块

Adaptive Server® Enterprise

15.7 ESD #2

文档 ID：DC37419-01-1572-01

最后修订日期：2012 年 8 月

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

目录

本书内容

*《Adaptive Server 参考手册》*包含有关 Sybase® Adaptive Server® Enterprise 和 Transact-SQL® 语言的四本指南：

- *《构件块》*介绍了 Transact-SQL 的各个部件：数据类型、内置函数、全局变量、表达式、标识符、保留字和 SQLSTATE 错误。要想成功使用 Transact-SQL，您首先必须了解这些构件块的功能，以及它们对 Transact-SQL 语句执行结果的影响。
- 《命令》提供了有关用于创建语句的各种 Transact-SQL 命令的参考信息。
- *《过程》*提供了有关系统过程、目录存储过程、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。
- *《表》*提供了有关系统表的参考信息。系统表用于存储有关服务器、数据库和用户的信息，以及服务器的其它详细信息。它还提供有关 dbccdb 和 dbccalt 数据库中的表的信息。

约定

以下各部分将说明在这些参考手册指南中使用的约定。

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

表 1 说明本手册中出现的语法语句的约定：

表 1：本手册的字体和语法约定

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

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

键入小括号作为命令的一部分。	compute *row_aggregate* (*column_name*)
双冒号加等号表示语法是用 BNF 符号编写的。请勿输入此符号。表示 “被定义为”。	`::=`
大括号表示必须至少选择括号中的一个选项。不要输入大括号。	`{cash, check, credit}`
中括号表示可以选择括号中的一个或多个选项，也可不选。不要输入中括号。	`[cash | check | credit]`
逗号表示可以选择任意多个显示的选项。可输入逗号作为命令的一部分来分隔选项。	`cash, check, credit`
竖线 (|) 表示只可选择所显示的选项中的一个。	`cash | check | credit`

省略号 (...) 表示可以将最后一个单元*重复*任意次。	`buy thing = price [cash | check | credit]`
	`[, thing = price [cash | check | credit]]...`
	您必须至少购买一件产品，并给出其价格。可以选择一种付款方式：中括号中的选项之一。还可选择购买其它物品：可根据需要购买任意数量的物品。对于要买的每种产品，给出其名称、价格和付款方式 （可选）。

- 语法语句 （显示命令的语法和所有选项）显示如下：

  sp_dropdevice [*device_name*]

  对于具有多个选项的命令：

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

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

- 说明 Transact-SQL 命令用法的示例如下：

  ```
  select * from publishers
  ```

- 计算机输出的示例如下：

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

(3 rows affected)
```

本手册中，大多数示例是小写字体。不过，键入 Transact-SQL 关键字时可以忽略大小写。例如， SELECT、 Select 和 select 之间没有区别。

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

第 1 章

系统数据类型和用户定义的数据类型

本章描述了 Transact-SQL 数据类型，这些数据类型指定了列、存储过程参数以及局部变量的类型、大小和存储格式。

数据类型种类

Adaptive Server 提供了几种系统数据类型以及用户定义的数据类型 timestamp、sysname 和 longsysname。表 1-1 列出了 Adaptive Server 数据类型的类别。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值数据
货币数据类型	货币数据
Timestamp 数据类型	在 Client-Library™ 应用程序中浏览的表
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	原始二进制数据，例如图片，类似十六进制的符号
bit 数据类型	真 / 假和是 / 否类型的数据
sysname 和 longsysname 数据类型	系统表
text、 image 和 unitext 数据类型	可打印字符或十六进制式数据，它们需要的列大小超出了服务器的逻辑页大小所提供的最大列大小。
抽象数据类型	Adaptive Server 通过 Java 类支持抽象数据类型。有关详细信息，请参见 *Adaptive Server Enterprise 中的 Java*。
用户定义的数据类型	定义继承了此表所列数据类型的规则、缺省值、空值类型、 IDENTITY 属性和基本数据类型的对象。如果客户端使用不同字符集， text 会进行字符集转换，而 image 则不会。

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，并提供了有关每种数据类型的有效值范围和存储大小的信息。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输出的。用户定义的数据类型（如 timestamp）*区分大小写*。Adaptive Server 提供的大多数数据类型都不是保留字，并且可以用于命名其它对象。

表 1-2：Adaptive Server 系统数据类型

数据类型种类	同义词	范围	存储的字节数
精确数值：整数			
bigint		介于 2^{63} 到 $-2^{63} - 1$ 之间的整数（从 -9,223,372,036,854,775,808 到 +9,223,372,036,854,775,807，包括这两个值）。	8
int	整数	2^{31} -1 (2,147,483,647) 到 -2^{31} (-2,147,483,648)	4
smallint		2^{15} -1 (32,767) 到 -2^{15} (-32,768)	2

数据类型种类	同义词	范围	存储的字节数
tinyint		0 到 255 （不允许使用负数）	1
unsigned bigint		0 到 18,446,744,073,709,551,615 之间的整数	8
unsigned int		0 到 4,294,967,295 之间的整数	4
unsigned smallint		0 到 65535 之间的整数	2
精确数值：小数			
numeric (p, s)		10^{38} -1 到 -10^{38}	2 到 17
decimal (p, s)	dec	10^{38} -1 到 -10^{38}	2 到 17
近似值			
float (precision)		与计算机有关	缺省精度 < 16 时为 4， 缺省精度 >= 16 时为 8
double precision		与计算机有关	8
real		与计算机有关	4
货币			
smallmoney		214,748.3647 到 -214,748.3648	4
money		922,337,203,685,477.5807 到 -922,337,203,685,477.5808	8
日期/时间			
smalldatetime		1900 年 1 月 1 日至 2079 年 6 月 6 日	4
datetime		1753 年 1 月 1 日至 9999 年 12 月 31 日	8
date		0001 年 1 月 1 日至 9999 年 12 月 31 日	4
time		12:00:00AM 到 11:59:59:990PM	4
bigdatetime		0001 年 1 月 1 日至 9999 年 12 月 31 日， 12:00.000000 AM 至 11:59:59.999999 PM	8
bigtime		12:00:00.000000 AM 至 11:59:59.999999 PM	8
字符			
char(n)	character	pagesize	n
varchar(n)	character varying、char varying	pagesize	实际条目长度
unichar	Unicode 字符	pagesize	*n* * *@@unicharsize*（*@@unicharsize* 等于 2）
univarchar	Unicode 字符 varying、char varying	pagesize	实际字符数 * *@@unicharsize*
nchar(n)	national character、national char	pagesize	*n* * *@@ncharsize*

数据类型种类	同义词	范围	存储的字节数
nvarchar(n)	nchar varying、national char varying、 national character varying	pagesize	*@@ncharsize* * 字符数
text		2^{31} -1 (2,147,483,647) 个字节或更少	未初始化时为 0；初始化后是 2K 的倍数
unitext		1 – 1,073,741,823	未初始化时为 0；初始化后是 2K 的倍数
二进制			
binary(n)		pagesize	*n*
varbinary(n)		pagesize	实际条目长度
image		2^{31} -1 (2,147,483,647) 个字节或更少	未初始化时为 0；初始化后是 2K 的倍数
位			
bit		0 或 1	1（1 个字节最多容纳 8 个 bit 列）

列、变量或参数的数据类型

您必须为列、局部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，也可以是数据库中用户定义的数据类型中的任意一种。

为表中的列声明数据类型

要在 create table 或 alter table 语句中声明新列的数据类型，请使用以下语法：

```
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)
```

还可以在 select into 语句中使用 convert 或 cast 声明新列的数据类型：

```
select convert(double precision, x), cast ( int, y) into
    newtable from oldtable
```

为批处理或过程中的局部变量声明数据类型

要在批处理或存储过程中为局部变量声明数据类型，请使用下面的语法：

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* [;n*umber*]
[[()@*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
```

确定数字文字的数据类型

输入的带有 E 符号的数字文字被视为 float；所有其它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。

注释　为了保持向后兼容，请对应被视为 float 的数字文字使用 E 符号。

确定字符文字的数据类型

在 Adaptive Server 12.5.1 版之前，如果客户端的字符集不同于服务器的字符集，通常会进行转换以便可以在处理 SQL 查询文本前先将其转换为服务器的字符集。如果因为字符无法显示在服务器的字符集中而无法转换它，会拒绝整个查询。从 Adaptive Server 12.5.1 版开始，已经消除了这种字符集 “瓶颈”。

您不能声明字符文字的数据类型。 Adaptive Server 将字符文字视为 varchar，但如果字符文字中包含无法转换为服务器缺省字符集的字符，则除外。这种文字被视为 univarchar。这样，便可以执行这样的查询：使用 “sjis”（日语）客户端在为 “iso_1”配置的服务器中选择 unichar 数据。例如：

```
select * from mytable where unichar_column = '五'
```

由于 （在 “iso_1”中）无法使用 char 数据类型表示该字符文字，因此会将它升级为 unichar 数据类型，并且查询会成功。

混合模式表达式的数据类型

当您对具有不同数据类型的值执行并置或混合模式算术时， 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**，数据类型层次存储在 systypes 系统表中。用户定义的数据类型继承了其所依据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
smalldatet        13
intn              14
uintn             15
bigint            16
ubigint           17
int               18
uint              19
smallint          20
usmallint         21
tinyint           22
bit               23
univarchar        24
unichar           25
unitext           26
sysname           27
varchar           27
nvarchar          27
longsysnam        27
char              28
nchar             28
timestamp         29
varbinary         29
binary            30
text              31
image             32
date              33
time              34
daten             35
timen             36
bigdatetime       37
bigtime           38
bigdatetimen      39
bigtimen          40
xml               41
extended time     99
```

注释　u<int_type> 是一种内部表示形式。无符号类型的正确语法是 unsigned {int | integer | bigint | smallint }

数据类型层次决定了使用不同数据类型的值所进行的计算的结果。结果值的数据类型会被指定为最接近此列表顶部的数据类型或层次值最小的数据类型。

在下面的示例中，来自 sales 表的 *qty* 同来自 roysched 表的 royalty 相乘。qty 是 smallint，它的层次为 20； royalty 是 int，其层次为 18。因此，结果的数据类型是 int：

```
smallint(qty) * int(royalty) = int
```

确定精度和标度

对于 numeric 和 decimal 数据类型，精度和标度的每种组合都是一个不同的 Adaptive Server 数据类型。如果对两个 numeric 或 decimal 值执行算术运算：

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

Adaptive Server 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)

数据类型转换

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。其它转换都必须使用 convert、hextoint、inttohex、hextobigint 和 biginttohex 函数显式执行。有关 Adaptive Server 支持的数据类型转换的详细信息，请参见 *《Transact-SQL 用户指南》*。

固定长度的 NULL 列的自动转换

只有数据类型长度可变的列才能存储空值。创建数据类型长度固定的 NULL 列时， Adaptive Server 会自动将其转换为长度可变的相应数据类型。 Adaptive Server 不会告诉用户数据类型有更改。

表 1-4 列出了长度固定的数据类型及其要转换的可变长度数据类型。某些可变长度的数据类型（如 moneyn）是保留数据类型；不能使用它们来创建列、变量或参数：

表 1-4：固定长度的数据类型的自动转换

原始的固定长度的数据类型	转换为
char	varchar
unichar	univarchar
nchar	nvarchar
binary	varbinary
datetime	datetimn
date	daten
time	timen
float	floatn
bigint、 int、 smallint 和 tinyint	intn
unsigned bigint、 unsigned int 和 unsigned smallint	uintn
decimal	decimaln
numeric	numericn
money 和 smallmoney	moneyn

处理溢出和截断错误

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

- arithabort arith_overflow 指定显式或隐式数据类型转换过程中出现被零除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该错误，则 arithabort arith_overflow on 不会回退批处理中以前的命令，但是 Adaptive Server 不会执行批处理中产生错误的语句后面的任何语句。

  将 arith_overflow 设置为 on 指的是执行时间，而不是指将 Adaptive Server 设置成的规范化级别。

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

- arithabort numeric_truncation 指定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。如果设置了 arithabort numeric_truncation off， Adaptive Server 就会截断查询结果并继续进行处理。

arithignore 选项确定在出现溢出错误之后 Adaptive Server 是否输出警告消息。缺省情况下，arithignore 选项设置为 off。这会使 Adaptive Server 在任何导致数字溢出的查询之后显示警告消息。若要忽略溢出错误，请使用 set arithignore on。

标准和遵从性

表 1-5 列出了 ANSI SQL 标准和 Transact-SQL 数据类型符合 ANSI SQL 的级别。

表 1-5：ANSI SQL 标准和 Transact-SQL 数据类型符合 ANSI SQL 的级别

Transact-SQL 一 ANSI SQL 数据类型	Transact-SQL 扩展 一 用户定义的数据类型
• char • varchar • smallint • int • bigint • decimal • numeric • float • real • date • time • double precision	• binary • varbinary • bit • nchar • datetime • smalldatetime • bigdatetime • bigtime • tinyint • unsigned smallint • unsigned int • unsigned bigint • money • smallmoney • text • unitext • image • nvarchar • unichar • univarchar • sysname • longsysname • timestamp

精确数值数据类型

在必须要精确表示一个值时，请使用精确数值数据类型。Adaptive Server 为整数和小数提供精确数值类型。

整数类型

Adaptive Server 提供了以下用于存储整数的精确数值数据类型：bigint、int（或 integer）、 smallint、 tinyint 以及它们各自对应的无符号形式的数据类型。根据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异，如表 1-6 所示。

表 1-6：整数数据类型

数据类型	存储	存储的字节数
bigint	-2^{63} 到 2^{63} - 1 之间的整数 （从 -9,223,372,036,854,775,808 到 +9,223,372,036,854,775,807，包括这两个值）。	8
int[eger]	-2^{31} 和 2^{31} - 1 （- 2,147,483,648 和 2,147,483,647）之间的整数，包括这两个值。	4
smallint	-2^{15} 和 2^{15} - 1 （- 32,768 和 32,767）之间的整数，包括这两个值。	2
tinyint	0 和 255 （包括这两个数值）之间的整数。（不允许为负数。）	1
unsigned bigint	0 到 18,446,744,073,709,551,615 之间的整数	8
unsigned int	0 到 4,294,967,295 之间的整数	4
unsigned smallint	0 到 65,535 之间的整数	2

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，整数数据就可以包含小数点。smallint、integer 和 bigint 数据类型前面可以加上正号或负号，也可以不加。tinyint 数据类型前面可以有任选的正号。

表 1-7 显示了数据类型为 integer 的列的一些有效条目，并且指出了 isql 显示这些值的方式：

表 1-7：有效的整数值

输入的值	显示的值
2	2
+2	2
-2	-2
2.	2
2.000	2

表 1-8 列出了 integer 列的一些无效条目：

表 1-8：无效的整数值

输入的值	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]。除了下面这种情况以外， numeric 和 decimal 数据类型在其它所有方面都相同。只有 integer 数据类型和标度为 0 的 numeric 数据类型才能用于 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值的范围：

- 精度指定了能够在该列中存储的最大小数位数。它包括小数点左右两侧的*所有*位数。您可以将精度范围指定为 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。对于 1 位或 2 位的列，最小存储要求为 2 个字节。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。

使用下面的公式可计算 numeric 或 decimal 列的准确存储大小：

```
ceiling (precision / log10(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 1-9 显示了数据类型为 numeric(5,3) 的列的一些有效条目，并指出了 isql 显示这些值的方式：

表 1-9：有效的小数值

输入的值	显示的值
12.345	12.345
+12.345	12.345
-12.345	-12.345
12.345000	12.345
12.1	12.100
12	12.000

表 1-10 显示了数据类型为 numeric(5,3) 的列的一些无效条目：

表 1-10：无效的小数值

输入的值	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。

标准和遵从性

Transact-SQL 提供了 smallint、int、bigint、numeric 和 decimal ANSI SQL 精确数值数据类型。unsigned bigint、unsigned int、unsigned smallint 和 tinyint 类型为 Transact-SQL 扩展。

近似数值数据类型

将近似数值类型 float、double precision 和 real 用于允许舍入的数值数据。近似数值类型尤其适用于那些涉及的取值范围很宽的数据。它们支持所有集合函数和所有算术运算。

了解近似数值数据类型

用于存储浮点数的近似数值数据类型在表示实数时存在固有的微小偏差，因此它们被称为 “近似数”。若要使用这些数据类型，您必须了解其相关限制。

当输出或显示浮点数时，输出表示与所存储的数字并非完全相同，而且所存储的数字与用户输入的数字也不完全相同。大多数情况下，所存储的数字表示形式会足够接近，软件也会使显示的输出看起来就像原始输入一样，但是如果您要将浮点数用于计算，尤其是要使用近似数值数据类型来执行重复计算，则必须要了解不准确性，所计算结果的不准确性可能会令人吃惊和出乎意料。

造成这种不准确性的原因是浮点数在计算机中以二进制小数（即除以 2 的幂后得到的表示数字）形式存储，而我们使用的数则是十进制（10 的幂）数。这意味着，只有非常少的数字可以准确存储：0.75 (3/4) 可以准确存储，因为它是二进制小数（4 是 2 的幂）； 0.2 (2/10) 不能准确存储（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正确的，并且使用同样接近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太接近正确数值的结果。如果这些注意事项对应用程序很重要，请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创建的。float 类型接受小括号内的可选二进制精度。精度为 1-15 的 float 列存储为 real ； 那些精度更高的列则存储为 double precision。

这三种类型的范围和存储精度都与使用的计算机相关。

表 1-11 显示了每种近似数值类型的范围和存储大小。isql 只显示小数点后的 6 位有效数字，并舍入其余数字：

表 1-11：近似数值数据类型

数据类型	存储的字节数
float[(default precision)]	对于 default precision < 16，为 4 对于 default precision >= 16，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”开头）必须是整数。

该条目所表示的值是下列乘积：

尾数 `* 10` 指数

例如， 2.4E3 表示 2.4 乘以 10^3 这一值，即 2400。

NaN 和 Inf 值

“NaN”和 “Inf”是特殊值， IEEE754/854 浮点数标准使用它们分别表示“不是一个数字”和“无限”的值。根据 ANSI SQL92 标准，Adaptive Server 12.5 和更高版本不允许在数据库中插入这些值，也不允许生成这些值。在 Adaptive Server 12.5 版之前的版本中， Open Client 客户端（如本机模式 bcp、 JDBC 和 ODBC）有时会强制将这些值填入表中。

如果在数据库中遇到 NaN 或 Inf 值，请与 Sybase 客户支持部门联系，并提供有关如何重现问题的详细信息。

标准和遵从性

符合 ANSI SQL 的级别 float、double precision 和 real 数据类型符合初级标准。

货币数据类型

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位制货币，但是 Adaptive Server 没有提供将一种货币转换为另一种货币的方法。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，但显示时它们将数值向上舍入，只保留两位小数。缺省的输出格式是在每三位后放置一个逗号。

范围和存储大小

表 1-12 总结了 money 数据类型的范围和存储要求：

表 1-12：货币数据类型

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	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。

可以输入带或不带前导货币符号（例如美元符号 ($)、日元符号 (¥) 或英镑符号 (£)）的 money 和 smallmoney 值。若要输入负值，请在货币符号后加上一个负号。在条目中不要包含逗号。

标准和遵从性

ANSI SQL － money 和 smallmoney 数据类型是 Transact-SQL 扩展。

Timestamp 数据类型

对于要在 Client-Library™ 应用程序中（有关详细信息，请参见“浏览模式”）浏览的表，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 都会更新 timestamp 列。一个表只能有一列 timestamp 数据类型。

创建 *timestamp* 列

如果创建一个名为 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)
```

您也可以显式将 timestamp 数据类型指派给名为 timestamp 的列：

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

或者指派给具有其它名称的列：

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

您可以创建一个名为 timestamp 的列，并为其指派其它数据类型（虽然这可能会令其他用户感到困惑，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)
```

日期和时间数据类型

使用 datetime、smalldatetime、bigdatetime、bigtime、date 和 time 可存储绝对日期和时间信息。使用 timestamp 可存储二进制类型信息。

Adaptive Server 使用多种数据类型来存储日期和时间值。

- date
- time
- smalldatetime
- datetime
- bigdatetime
- bigtime

日期的缺省显示格式为 “Apr 15 1987 10:23PM”。 bigdatetime/bigtime 类型的缺省显示格式为“Apr 15 1987 10:23:00.000000PM”。对于其它日期显示样式，可以使用 convert 函数进行转换。虽然 Adaptive Server 可能会舍入或截断毫秒值，但是也可使用内置日期函数对 date 和 time 值执行一些算术运算。

- datetime 列可保存从 1753 年 1 月 1 日到 9999 年 12 月 31 日之间的日期。在支持 1/300 秒精度级别的平台上， datetime 值可精确到该级别。小数秒数的最后一个数字始终是 0、3 或 6。其它数字将舍入为这三个数字之一，即，0 和 1 舍入为 0，2、3 和 4 舍入为 3，5、6、7 和 8 舍入为 6， 9 舍入为 10。存储大小为 8 个字节：4 个字节用于存储自 1900 年 1 月 1 日这一基准日期以来的天数，另外 4 个字节用于存储当天的时间。
- smalldatetime 列可保存从 1900 年 1 月 1 日到 2079 年 6 月 6 日之间的日期，其精确度可以精确到分钟。存储大小为 4 个字节：2 个字节用于存储 1900 年 1 月 1 日以后的天数，另外 2 个字节用于存储自午夜以后的分钟数。
- bigdatetime 列可保存从 0001 年 1 月 1 日到 9999 年 12 月 31 日的日期以及 12:00:00.000000 AM 到 11:59:59.999999 PM 之间的时间。存储大小为 8 个字节。 bigdatetime 的内部表示是一个 64 位整数，其中包含自 0000 年 1 月 1 日以来所逝去的微秒数。
- bigtime 列可保存从 12:00:00.000000 AM 到 11:59:59.999999 PM 之间的时间。存储大小为 8 个字节。 bigtime 的内部表示是一个 64 位整数，其中包含自午夜以来所逝去的微秒数。
- date 列可以保存从 0001 年 1 月 1 日到 9999 年 12 月 31 日之间的日期。存储大小为 4 个字节。
- time 介于 00:00:00:000 和 23:59:59:990 之间。time 值精确到 1/300 秒。小数秒数的最后一个数字始终是 0、3 或 6。其它数字将舍入为这三个数字之一，即，0 和 1 舍入为 0，2、3 和 4 舍入为 3，5、6、7 和 8 舍入为 6，9 舍入为 10。您可以使用军用时间，或使用 12AM 代表中午，使用 12PM 代表午夜。时间值必须包含一个冒号，或者 AM 或 PM 标记。 AM 或 PM 可以大写，也可以小写。

输入日期和时间信息时，始终要用单引号或双引号将时间或日期引起来。

范围和存储要求

表 1-13 总结了 datetime、smalldatetime、bigdatetime、bigtime、date 和 time 数据类型的范围和存储要求：

表 1-13：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
bigdatetime	0001 年 1 月 1 日至 9999 年 12 月 31 日	8
bigtime	12:00:00.000000AM 至 11:59:59.999999PM	8
date	0001 年 1 月 1 日至 9999 年 12 月 31 日	4
time	12:00:00 AM 到 11:59:59:990 PM	4

输入日期和时间数据

datetime、smalldatetime、bigdatetime 和 bigtime 数据类型由日期部分和时间部分组成，时间部分可在日期部分的前面或后面。（您既可以省略日期，也可以省略时间，或者两者都省略。） date 数据类型只含日期，而 time 数据类型只含时间。这些值必须用单引号或双引号引起来。

输入日期

日期包括月、日和年；对于 date、datetime、bigdatetime、bigtime 和 smalldatetime，可以使用各种不同的格式输入日期：

- 您可以将整个日期作为 4 位、6 位或 8 位无分隔符的字符串来输入，也可以在日期分量之间使用斜杠 (/)、连字符 (-) 或句点 (.) 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。请为一位数的年份、月份和日使用前导零。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 选项来确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。
- 某些日期格式接受 2 位数的年份 (*yy*)：
  - 那些小于 50 的数字将被解释为 20*yy*。例如，`01` 为 2001，`32` 为 2032，`49` 为 2049。
  - 那些等于或大于 50 的数字被解释为 19*yy*。例如，`50` 为 1950，`74` 为 1974，`99` 为 1999。
- 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。如果省略了 bigdatetime 的日期部分，将添加缺省值 0001 年 1 月 1 日。

表 1-14 描述了输入 datetime 或 smalldatetime 值的日期部分时可以接受的格式：

表 1-14：日期和时间数据类型的日期格式

日期格式	解释	样本条目	含义
4 不带分隔符的 6 位数字串	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
6 不带分隔符的 6 位数字串	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
8 不带分隔符的 6 位数字串	解释为 *yyyymmdd*。	“19940415”	1994 年 4 月 15 日
由 2 位数的月、日和年 （它们由斜杠、连字符或句点，或是上述符号的组合进行分隔）所组成的字符串	dateformat 和 language set 选项确定所期望的日期分量顺序。对于 us_english 而言，缺省的顺序是 *mdy*。 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*。	“4/15/94” “4.15.94” “4-15-94” “04.15/94”	当 dateformat 选项设置为 mdy 时，所有这些条目均被解释为 1994 年 4 月 15 日。
由 2 位数的月、2 位数的日和 4 位数的年 （它们由斜杠、连字符或句点，或上述符号的组合加以分隔）组成的字符串	dateformat 和 language set 选项确定所期望的日期分量顺序。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，解释为 1994 年 4 月 15 日。
月份用字符格式 （可以是完整的月份名称，也可以是它的标准缩写）输入，后跟一个可选的逗号	如果输入的年份是 4 位数，则可以按任何顺序输入日期分量。	“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。
	如果不输入日，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
	如果年份仅有 2 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” “apr 15 94”	2017 年 3 月 16 日 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

输入时间

datetime、 smalldatetime 或 time 值的时间成分必须按如下方式指定：

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

bigdatetime 或 bigtime 值的时间成分必须按如下方式指定：

```
hours[:minutes[:seconds[.microseconds]] [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 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。
- 对秒数的说明可以是一个小数点，后面跟小数部分；也可以是一个冒号，后面跟毫秒数。例如，“15:30:20:1”表示下午 3:30 过二十秒又一毫秒；“15:30:20.1”表示下午 3:30 过二十又十分之一秒。表示微秒时必须用小数点。
- 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、*smalldatetime* 和 *date* 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”。要显示秒和毫秒，并获取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 1-15 列出了 datetime 条目及其显示值的一些示例：

表 1-15：datetime 和 date 条目的示例

条目	所显示的值
“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 14:30:00AM
“4am”	Jan 1 1900 4:00AM
日期示例	
“1947”	1947 年 1 月 1 日
“450128”	2045 年 1 月 28 日
“520317”	1952 年 3 月 17 日

显示 bigdatetime 和 bigtime 的格式

对于 bigdatetime 和 bigtime，显示值会反映微秒精度。bigdatetime 和 bigtime 的缺省显示格式可容纳此提高的精度。

- hh:mm:ss.zzzzzzAM 或 PM
- hh:mm:ss.zzzzzz

- mon dd yyyy
  hh:mm:ss.zzzzzzAM(PM)
- mon dd yyyy
  hh:mm:ss.zzzzzz
- yyyy-mm-dd
  hh:mm:ss.zzzzzz

时间格式必须按如下方式指定：

hours[:minutes[:seconds[.microseconds]] [*AM* | *PM*]

hours[:minutes[:seconds[number of milliseconds]] [*AM* | *PM*]

使用 12 AM 表示午夜，使用 12 PM 表示正午。bigtime 值必须包含一个冒号，或者 AM 或 PM 标记。 AM 或 PM 能够用大写、小写或大小写混合的方式输入。

对秒数的说明可以包括一个小数点，后面跟小数部分；也可以包括一个冒号，后面跟毫秒数。例如，“12:30:20:1”表示 12:30 过二十秒又一毫秒；“12:30:20.1”表示 12:30 过二十又十分之一秒。

要存储包括微秒的 bigdatetime 或 bigtime 时间值，请使用小数点指定一个字符串文字。“00:00:00.1”表示午夜过十分之一秒；“00:00:00.000001”表示午夜过百万分之一秒。在冒号之后指定秒数部分的任何值仍然表示毫秒数。例如，“00:00:00:5”表示 5 毫秒。

显示 *time* 值的格式

time 值的显示格式是“hh:mm:ss:mmmAM”（或“PM”）；例如，“10:23:40:022PM”。

表 1-16：时间条目的示例

条目	所显示的值
“12:12:00”	12:12PM
“01:23PM”或“01:23:1PM”	13:23:00AM
“02:24:00:001”	2:24AM

查找与某一模式匹配的值

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、日和年份的日期值或时间值， Adaptive Server 只会返回其时间正好为 12:00:00:000AM 的那些值。

例如，如果将值“9:20”插入名为 arrival_time 的列，则以下子句将找不到它，因为 Adaptive Server 将这个输入内容转换为“Jan 1, 1900 9:20AM：”如果您使用等于运算符查找这一条目，将不会找到它：

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

使用 like 运算符可以找到该条目：

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

使用 like 时，Adaptive Server 会首先将日期转换为 datetime 或 date 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、 2 个字符的日、 4 个字符的年份、以及用小时和分钟及 “AM” 或 “PM” 表示的时间。

使用 like 进行搜索时，您不能使用在输入 datetime、 smalldatetime、bigdatetime、bigtime、date 和 time 值的日期部分时可以采用的多种不同的输入格式。除非同时使用 *style* 9 或 109 以及 convert 函数，否则不能用 like 和匹配模式来搜索秒或毫秒。

如果您要使用 like，并且该月份中的该天是 1 和 9 之间的数字，请在月份和日之间插入两个空格，以与 datetime 值的 varchar 转换匹配。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。以下子句 （在 “May” 和 “2” 之间有一个空格）查找从 5 月 20 日到 5 月 29 日的所有日期，而不是 5 月 2 日：

```
like "May 2%"
```

只需为 like 插入额外空格，而不需要为其它日期比较插入，因为只有在进行 like 比较时， datetime 值才会转换为 varchar。

处理日期

您可以使用内置日期函数对 date 和 time 数据类型值执行一些算术计算。请参见 *《Transact-SQL 用户指南》*。

标准和遵从性

符合 ANSI SQL 的级别 datetime 和 smalldatetime 数据类型是 Transact-SQL 扩展。 date 和 time 数据类型符合初级标准。

字符数据类型

对于某种情况，您要使用哪种数据类型，取决于您要存储的数据的类型：

- 使用字符数据类型来存储含有字母、数字和符号的字符串。
- 将 varchar(n) 和 char(n) 用于单字节字符集 （如 us_english）和多字节字符集 （如日文）。
- 使用 unichar(n) 和 univarchar(n) 数据类型存储 Unicode 字符。如果每个字符需要的字节数固定，则它们适用于单字节或多字节字符。

- 将固定长度的数据类型 nchar(n) 和可变长度的数据类型 nvarchar(n) 用于单字节和多字节字符集（如日语）。nchar(n) 与 char(n) 以及 nvarchar(n) 与 varchar(n) 之间的区别在于 nchar(n) 和 nvarchar(n) 都基于每个字符的字节数（基于缺省字符集）的 *n* 倍分配存储空间。char(n) 和 varchar(n) 则分配 *n* 字节的存储空间。
- 字符数据类型最多可以存储相当于一页大小的数据
- 将 text 数据类型（在第 32 页上的“text、 image 和 unitext 数据类型”中进行了介绍）— 或子表中的多行 — 用于比 char 或 varchar 数据类型允许的字符串长的字符串。

unichar、 *univarchar*

在可以使用 char 和 varchar 字符数据类型的任何位置均可以使用 unichar 和 univarchar 数据类型，而且不必更改语法。

在 Adaptive Server 12.5.1 和更高版本中，如果查询包含无法用服务器的字符集表示的字符文字，则会自动将这样的查询升级为 unichar 数据类型，这样，就不必更改数据操作语言 (DML) 语句的语法。其它语法适用于指定字符文字形式的任意字符，但是确定将文字“升级”为 unichar 时只基于表示形式。

使用数据定义语言 (DDL) 语句，需要进行的语法更改最少。例如，在 create table 命令中， Unicode 列的大小是以 16 位的 Unicode 值为单位指定的，而不是以字节为单位，因而可在 char(200) 和 unichar(200) 之间保持相似性。报告列长度的 sp_help 使用相同的单位。乘数 (2) 存储在新的全局变量 *@@unicharsize* 中。

有关 Unicode 的详细信息，请参见 *《系统管理指南》*中的第 8 章“配置字符集、排序顺序和语言”。

长度和存储大小

当在游标的 varchar 列中填充变量时，字符变量会将字符串中的尾随空格去掉。

使用 *n* 为 char 和 varchar 数据类型指定存储字节数。对于 unichar，请使用 *n* 指定 Unicode 字符数（分配的存储空间量为每个字符 2 个字节）。对于 nchar 和 nvarchar，*n* 是字符数（分配的存储空间量是服务器的当前缺省字符集中每个字符的字节数的 *n* 倍）。

如果不使用 *n* 指定长度：

- 对于使用 create table、 alter table 创建的列以及使用 declare 创建的变量，缺省长度为 1 个字节。
- 对于使用 convert 函数创建的值，缺省长度为 30 字节。

小于指定长度的条目会被空白填充；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 NULL 值的固定长度列在内部转换为可变长度的列。

使用 *n* 为可变长度数据类型 varchar(n)、univarchar(n) 和 nvarchar(n) 指定最大长度 （以字符为单位）。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，但是不会被填充以达到所定义的长度。字符文字被视为可变长度数据类型。

固定长度列占用的存储空间往往比可变长度列占用的存储空间多，但是您可以更快地对其进行访问。表 1-17 总结了不同字符数据类型的存储要求：

表 1-17：字符数据类型

数据类型	存储	存储的字节数
char(n)	字符	*n*
unichar(n)	Unicode 字符	*n**@@*unicharsize* （@@*unicharsize* 等于 2）
nchar(n)	国家特有字符	*n* * @@*ncharsize*
varchar(n)	可变长度的字符	输入的实际字符数
univarchar(n)	Unicode character varying	实际字符数 * @@*unicharsize*
nvarchar(n)	可变的国家特有字符	实际字符数 * @@*ncharsize*

用系统函数确定列长度

使用 char_length 字符串函数和 datalength 系统函数确定列长度：

- char_length 返回列中的字符数，去除可变长度数据类型的尾随空白。
- datalength 返回字节数，去除存储在可变长度列中的数据的尾随空白。

当声明 char 值可为 NULL 值时， Adaptive Server 在内部将其存储为 varchar。

如果 min 或 max 集合函数用于 char 列中，则返回的结果为 varchar，因此会去除掉所有尾随空格。

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 set quoted_identifier on，请为字符串使用单引号；否则 Adaptive Server 会将其视为标识符。

含有双引号的字符串应该用单引号引起来。含有单引号字符的字符串应该用双引号引起来。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 (\)。

有关带引号的标识符的详细信息，请参见 *《Transact SQL 用户指南》*中的 “分隔标识符”一节。

输入 Unicode 字符

使用可选语法可以指定任意 Unicode 字符。如果字符文字前面紧接 U& 或 u&（中间没有空格），则分析程序可识别该文字中的转义序列。\xxxx（其中 xxxx 表示 4 个十六进制数字）形式的转义序列由标量值为 xxxx 的 Unicode 字符替换。同样，\+yyyyyy 形式的转义序列由标量值为 yyyyyy 的 Unicode 字符替换。转义序列 \\ 由单个的 \ 替换。例如，下面两个查询是等同的：

```
select * from mytable where unichar_column = '五'
```

```
select * from mytable where unichar_column = U&'\4e94'
```

U& 或 u& 前缀只进行转义识别。选择文字的数据类型时只基于表示形式。因此，下面两个查询示例是等同的：

```
select * from mytable where char_column = 'A'
```

```
select * from mytable where char_column = U&'\0041'
```

在这两种情况下，字符文字的数据类型是 char，因为 “A” 是 ASCII 字符，而 ASCII 是 Sybase 支持的所有服务器字符集的子集。

U& 和 u& 前缀还可以与带双引号的字符文字和带引号的标识符一同使用。但是，带引号的标识符必须能用服务器的字符集表示，这是因为在系统表中是按名称标识所有数据库对象，并且所有这类名称均属于 char 数据类型。

处理空白

下面的示例创建一个名为 spaces 的表，该表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 NULL 值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 列会被填充以达到列的全部宽度；char null 列被视为 varchar，而 nchar null 列被视为 nvarchar。
- 只有 unichar not null 列会被填充以达到列的全部宽度； unichar null 列则被视为 univarchar。
- 前面的空白不受影响。
- 除了 char、 unichar 和 nchar not null 列之外，尾随空白会被截断。
- 空字符串 (“”) 被视为单个空格。在 char、 nchar 和 unichar not null 列中，结果是一个由空格组成的字段，其长度等于列长度。

处理字符数据

您可以使用 like 关键字来搜索字符串以查找特定字符，并使用内置字符串函数来处理其内容。对于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，可以将其用于算术运算。

标准和遵从性

符合 ANSI SQL 的级别 Transact-SQL 提供了 char 和 varchar ANSI SQL 数据类型。 nchar、 nvarchar、 unichar 和 univarchar 数据类型是 Transact-SQL 扩展。

二进制数据类型

使用二进制数据类型 binary(n) 和 varbinary(n) 可用原始二进制符号存储原始二进制数据 （如图片），存储大小最大可达到服务器逻辑页大小的最大列大小。

有效的 *binary* 和 *varbinary* 条目

二进制数据以字符“0x”开头，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* 以字节为单位指定列的长度，或使用 1 字节的缺省长度。每一字节存储 2 个二进制位。如果输入了一个长于 *n* 的值， Adaptive Server 会在不显示警告或错误的情况下将该条目截断为指定的长度。

对于预计其中所有条目在长度上都大致相等的数据，使用固定长度二进制类型 binary(n)。

对于预计长度会变化很大的数据，使用可变长度二进制类型 varbinary(n)。

由于 binary 列中的条目会被零填充以达到列长 (*n*)，因此它们需要的存储空间可能比 varbinary 列中的条目多，但是对其的访问速度较快。

如果不使用 *n* 指定长度：

- 对于使用 create table、 alter table 创建的列以及使用 declare 创建的变量，缺省长度为 1 个字节。
- 对于使用 convert 函数创建的值，缺省长度为 30 字节。

大于最大列大小的条目

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，请参见第 32 页上的“text、image 和 unitext 数据类型”。

尾随零的处理

所有 binary not null 列都会被零填充以达到列的全部宽度。由于接受空值的列必须作为可变长度列处理，因此在所有 varbinary 数据和 binary null 列中，尾随零都会被截断。

下面的示例使用 binary 和 varbinary 数据类型、NULL 和 NOT NULL 的全部四种不同组合创建了一个表。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都含 2 个二进制数字，所以 Adaptive Server 希望二进制条目包含后面跟有偶数个数字的“0x”字符。如果“0x”之后跟有奇数个数字， Adaptive Server 会假定您省略了前导 0 并为您添加它。

输入值“0x00”和“0x0”作为“0x00”存储在可变长度二进制列（binary null、image 和 varbinary 列）中。在固定长度二进制 (binary not null) 列中，该值会被零填充，从而达到字段的全部长度：

```
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 ASCII 值并将其转换。例如：

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

insert sample values ('002710000000ae1b')

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

平台相关性

输入某个特定值所采用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。

您不能在二进制数据类型中使用 sum 或 avg 集合函数。

对于十六进制字符串和整数之间的独立于平台的转换，请使用 inttohex 和 hextoint 函数，而不是该平台专有的转换函数。有关详细信息，请参见 *《Transact-SQL 用户指南》*。

标准和遵从性

符合 ANSI SQL 的级别：binary 和 varbinary 数据类型是 Transact-SQL 扩展。

bit 数据类型

将 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。

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。

存储大小为 1 字节。表中的多个 bit 数据类型被收集到字节中。例如，7 bit 列将占用 1 个字节；而 9 bit 列则要占用 2 个字节。

数据类型为 bit 的列不能为 NULL，并且其中不能有索引。

标准和遵从性

符合 ANSI SQL 的级别 Transact-SQL 扩展。

sysname 和 *longsysname* 数据类型

sysname 和 longsysname 是用户定义的数据类型，它们分布在 Adaptive Server 安装介质上并在系统表中使用。它们的定义为：

- sysname – varchar(30) "not null"
- longsysname – varchar(255) "not null"

您可以将列、参数或变量声明为 sysname 和 longsysname 类型。此外，您还可以使用 sysname 和 longsysname 的基本类型创建用户定义的数据类型，然后使用用户定义的数据类型定义列、参数和变量。

标准和遵从性

符合 ANSI SQL 的级别包括 sysname 和 longsysname 在内的所有用户定义的数据类型都是 Transact-SQL 扩展。

text、 *image* 和 *unitext* 数据类型

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

可变长度的 unitext 数据类型最多可容纳 1,073,741,823 个 Unicode 字符（2,147,483,646 字节）。

image 列是可变长度的列，最多可以容纳 2,147,483,647 (2^{31} - 1) 字节的原始二进制数据。

text 和 image 之间的一个主要区别是：如果您使用的不是 Adaptive Server 的缺省字符集，则 text 会进行字符集转换。 image 则不会进行字符集转换。

使用 create table 或 alter table 语句定义 text、unitext 或 image 列，定义方式和定义任何其它列的方式一样。text、unitext 或 image 数据类型定义不包括长度。 text、 unitext 和 image 列允许空值。它们的列定义格式如下：

column_name {text | image | unitext} [null]

例如，对于作者在 pubs2 数据库中的 blurbs 表，其中拥有允许空值的 text 列 (blurb)， create table 语句是：

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

下面这个示例创建一个允许空值的 unitext 列：

```
create table tb (ut unitext null)
```

在拥有 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 将 text、 unitext 和 image 数据存储到数据页的链接列表中，这些数据页同表的其余部分相分离。每个 text、unitext 或 image 页可存储相当于一个逻辑页大小的数据 （2、 4、 8 或 16K）。无论表中包含多少 text、unitext 和 image 列，表中的所有 text、unitext 和 image 数据都会被存储到单个页链中。

您可以使用 sp_placeobject 将针对 text、unitext 和 image 数据页的后续分配放置在其它逻辑设备上。

具有奇数个十六进制数字的 image 值会被前导零填充（插入的“0xaaabb”将变为 “0x0aaabb”）。

您可以使用 alter table 命令的 partition 选项对包含 text、unitext 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、 unitext 和 image 列的存储方式。

您可以在使用 text 数据类型的任何位置，使用具有相同语义的 unitext。无论 Adaptive Server 缺省字符集是什么，unitext 列都采用 UTF-16 编码存储。

用于存储 *text*、 *unitext* 和 *image* 数据的数据结构

在分配 text、 unitext 或 image 数据时，会在所分配的行中插入一个 16 字节的文本指针。此文本指针的一部分引用 text、unitext 或 image 数据头处的文本页页码。该文本指针就是所谓的第一文本页。

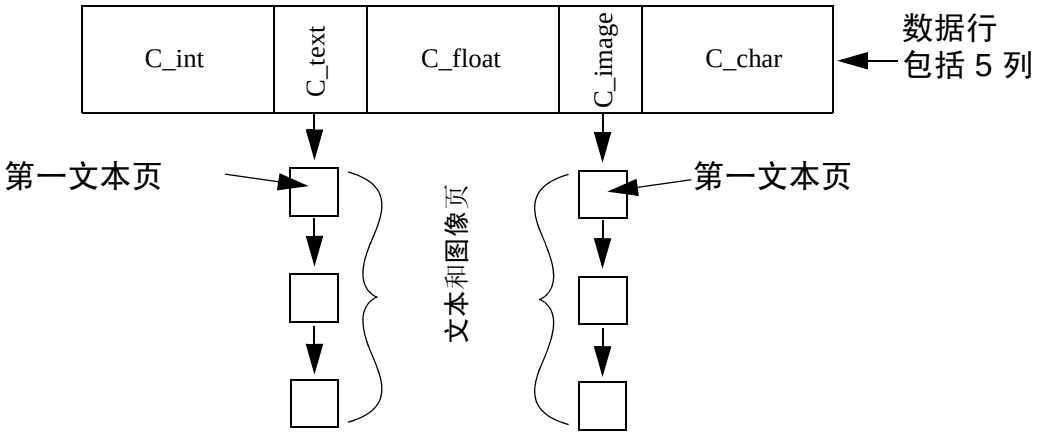
第一文本页包含两部分：

- 文本数据页链，它包含文本和图像数据，是文本页的双链接列表
- 可选文本节点结构，用于访问用户文本数据

为 text、unitext 或 image 数据分配第一文本页后，便永远不再释放它。如果更新现有的 text、unitext 或 image 数据行后产生的文本页少于当前分配给该 text、unitext 或 image 数据的文本页， Adaptive Server 将释放额外的文本页。如果对 text、 unitext 或 image 数据的更新将其值设置为 NULL，则除第一文本页以外，所有页都将被释放。

图 1-1 显示了数据行和文本页之间的关系。

图 1-1：文本指针和数据行之间的关系



在图 1-1 中，列 c_text 和 c_image 是包含该图底部的页的文本和图像列。

初始化 *text*、 *unitext* 和 *image* 列

在更新 text、 unitext 和 image 列或插入非空值之前，这些列不会被初始化。初始化将为每个非空 text、unitext 或 image 数据值至少分配一个数据页。它还在表中创建了一个指向 text、 unitext 或 image 数据位置的指针。

例如，以下语句创建表 testtext 并通过插入一个非空值初始化 blurb 列。现在，该列具有一个有效的文本指针，并且已经分配了第一个文本页。

```
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")
```

注释　注意，text 值两侧要加上引号，image 值前面要加上字符 “0x”。

有关使用 Client-Library 程序插入和更新 text、unitext 和 image 数据的详细信息，请参见 *《Client-Library/C 参考手册》*。

定义 unitext 列

可以使用 create table 或 alter table 语句定义 unitext 列，定义方式与定义其它数据类型的方式相同。不要定义 unitext 列的长度，并且列可以为空。

下面这个示例创建一个允许空值的 unitext 列：

```
create table tb (ut unitext null)
```

缺省的 unicode 排序顺序为 like 子句和 patindex 函数中的模式匹配定义了 unitext 列的排序顺序，这与 Adaptive Server 缺省的排序顺序无关。

通过允许使用 NULL 值来节省空间

为了节省空的 text、unitext 或 image 列占用的存储空间，请将其定义为允许空值并在使用该列前 insert 空值。插入一个空值不会初始化 text、unitext 或 image 列，因此也就不会创建文本指针或分配存储空间。例如，以下语句将值插入上面创建的 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")
```

从 *sysindexes* 获得信息

每一个含有 text、unitext 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 列采用的格式为“tablename”。 indid 始终为 255。这些列提供有关文本存储的信息：

表 1-18：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，下面的查询报告了 pubs2 数据库中 blurbs 表所使用的数据页的数目：

```
select name, data_pages(db_id(), object_id("blurbs"), indid)
    from sysindexes
    where name = "tblurbs"
```

注释 系统表张贴画显示了 sysindexes 和 systabstats 之间的一对一关系。这是正确的，但 text 和 image 列除外，因为其信息未保存在 systabstats 中。

使用 *readtext* 和 *writetext*

在使用 writetext 输入 text 数据或使用 readtext 读取该数据前，必须先初始化 text 列。有关详细信息，请参见 《参考手册：命令》中的 readtext 和 writetext 命令》。

如果使用 update 将现有的 text、unitext 和 image 数据替换为 NULL，可回收除第一页 （它仍然可供 writetext 在将来使用）以外的所有已分配的数据页。要释放该行的所有存储空间，可使用 delete 删除整个行。

对定义用于 unitext 的列使用 readtext 和 writetext 时，存在一些限制。有关详细信息，请参见 《参考手册：命令》中 readtext 和 writetext 下的 “用法”部分*命令*。

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
```

对 *text*、 *image* 和 *unitext* 列的限制

不能在以下情况中使用 text、 image 或 unitext 列：

- order by、 compute、 group by 或 union 子句
- 用于索引
- 用于子查询或连接
- 用于 where 子句，除非带有关键字 like

在触发器中，插入的文本值和删除的文本值都引用新值；不能引用旧值。

选择 *text*、 *unitext* 和 *image* 数据

下列全局变量返回有关 text、 unitext 和 image 数据的信息：

表 1-19：text、 unitext、和 image 全局变量

变量	解释
@@textptr	进程上一次插入或更新的 text、 unitext 或 image 列的文本指针。不要将此全局变量与 textptr 函数混淆。
@@textcolid	*@@textptr* 引用的列的 ID。
@@textdbid	数据库的 ID，该数据库包含其列被 *@@textptr* 引用的对象。
@@textobjid	对象的 ID，该对象包含被 *@@textptr* 引用的列。
@@textsize	set textsize 选项的当前值，该值指定由 select 语句返回的 text、 unitext 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* 引用的列的文本时间戳。

text、unitext 和 image 的值可能非常大。如果 select 列表中包括 text 和 image 值，返回数据的长度限制取决于 *@@textsize* 全局变量的设置，它包含 select 返回的 text 或 image 数据的字节数限制。 isql 的缺省限制为 32K 字节；缺省值取决于客户端软件。使用 set textsize 更改会话所用的值。

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、unichar、varchar 和 univarchar，将 image 值转换为 binary 或 varbinary，但会受到 character 和 binary 数据类型的最大长度的限制，该长度由服务器逻辑页大小的最大列大小决定。如果未指定长度，转换后的值将具有缺省的长度（30 个字节）。不支持隐式转换。

从 unitext 转换或转换成 unitext

可以将任何字符或二进制数据类型隐式转换成 unitext，还可以显式地在其它数据类型和 unitext 之间相互转换。但是，转换结果会受到目标数据类型的最大长度的限制。如果 Unicode 字符边界上的目标缓冲区无法容纳 unitext 值，则会截断数据。如果启用了 enable surrogate processing，则决不会在值代理对的中间截断 unitext 值，这意味着在数据类型转换之后可能会返回更少的字节。例如，如果表 tb 的 unitext 列 ut 中存储了字符串 “U+0041U+0042U+00c2”（U+0041 表示 Unicode 字符 “A”）并且服务器的字符集设置为 UTF-8，则下面的查询会返回值 “AB”，这是因为 U+00C2 被转换为 2 个字节的 UTF-8 0xc382：

```
select convert(char(3), ut) from tb
```

表 1-20：从 unitext 转换和转换成 unitext

转换	数据类型
下面的数据类型可隐式转换*为* unitext	char、varchar、unichar、univarchar、binary、varbinary、text、image
下面的数据类型可*从* unitext 隐式转换	text、 image
下面的数据类型可*从* unitext 显式转换	char、 varchar、 unichar、 univarchar、 binary、 varbinary

alter table modify 命令不支持将 text、image 或 unitext 列作为被修改的列。从 text 迁移至 unitext 列：

- 使用 bcp out -Jutf8 复制 text 列数据
- 创建一个包含 unitext 列的表
- 使用 bcp in -Jutf8 将数据插入新表中

text 数据中的模式匹配

使用 patindex 函数搜索指定模式在 text、 unitext、 varchar、 univarchar、 unichar 或 char 列中第一次出现时的起始位置。通配符 % 必须位于模式之前或之后 （搜索第一个或最后一个字符的情况除外）。

您也可以使用 like 关键字搜索特定模式。下面的示例从 blurbs 表的 copy 列中选择包含模式 “Net Etiquette”的每个 text 数据值。

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

重复行

指向 text、image 和 unitext 数据的指针唯一地标识每一行。因此，对于包含 text、 image 和 unitext 数据的表，如果不存在所有的 text、 image 和 unitext 数据都为 NULL 的行，则该表中不会有重复的行。如果是这种情况，指针不会被初始化。

在存储过程中使用大对象 *text*、 *unitext* 和 *image* 数据类型

Adaptive Server 允许您：

- 为局部变量声明大对象 (LOB) text、 image 或 unitext 数据类型，并将该变量作为输入参数传递给存储过程。
- 准备包括 LOB 参数的 SQL 语句。

当您启用语句高速缓存时，Adaptive Server 会使用 LOB 来高速缓存 SQL 语句。请参见 *《系统管理指南，第 2 卷》*第 3 章 “配置内存”。

某些限制适用于在存储过程中使用 LOB。

- 不支持对复制使用 LOB 参数。
- 不能将 LOB 数据类型用于 execute immediate 和 deferred compilation。

声明 LOB 数据类型

要为局部变量声明 LOB 数据类型，请使用：

```
declare @variable LOB_datatype
```

其中 *LOB_datatype* 是以下数据类型之一：text、 image 和 unitext。

此示例将 *text_variable* 声明为 text 数据类型：

```
declare @text_variable text
```

创建 LOB 参数

要创建 LOB 参数，请使用：

create procedure *proc_name* [*@parameter_name LOB_datatype*
as {*SQL_statement*}

此示例将创建使用 text LOB 数据类型的 new_proc 过程：

```
create procedure new_proc @v1 text
as
select char_length(@v1)
```

使用 LOB 数据类型

示例 1 将 LOB 用作存储过程的输入参数：

1 创建 table_1：

```
create table t1 (a1 int, a2 text)
    insert into t1 values(1, "aaaa")
    insert into t1 values(2, "bbbb")
    insert into t1 values(3, "cccc")
```

2 将 LOB 局部变量用作参数来创建存储过程：

```
create procedure my_procedure @new_var text
    as select @new_var
```

3 声明局部变量并执行存储过程。

```
declare @a text
select @a = a2 from t1 where a1 = 3
exec my_procedure @a
---------------------------
cccc
```

示例 2 在 text 函数中使用 LOB 变量：

```
declare @a text
select @a = "abcdefgh"
select datalength(@a)
--------------
             8
```

示例 3 声明 LOB text 局部变量：

```
declare @a text
select @a = '<doc><item><id>1</id><name>Box</name></item>'
    +'<item><id>2</id><name>Jar</name></item></doc>'
select id from xmltable ('/doc/item' passing @a
    columns id int path 'id', name varchar(20) path 'name')
```

```
as items_table
 id
-----------
          1
          2
```

然后将同样的 LOB 参数传递给存储过程：

```
create proc pr1 @a text
as
select id from xmltable ('/doc/item' passing @a
columns id int path 'id', name varchar(20) path 'name') as items_table
declare @a text
select @a =
'<doc><item><id>1</id><name>Box</name></item>'
+'<item><id>2</id><name>Jar</name></item></doc>'
 id
-----------
          1
          2
```

标准和遵从性

符合 ANSI SQL 的级别：text、 image 和 unitext 数据类型是 Transact-SQL 扩展。

数据类型和加密列

表 1-21 列出了 Adaptive Server 15.0.2 版中加密列的受支持数据类型以及受支持数据类型的加密列在磁盘上的长度。

表 1-21：加密列的数据类型长度

数据类型	输入数据长度	加密列类型	最大加密数据长度（无 init_vector）	实际加密数据长度（无 init_vector）	最大加密数据长度（有 init_vector）	实际加密数据长度（有 init_vector）
date	4	varbinary	17	17	33	33
time	4	varbinary	17	17	33	33
smalldatetime	4	varbinary	17	17	33	33
bigdatetime	8	varbinary	17	17	33	33
bigtime	8	varbinary	17	17	33	33
datetime	8	varbinary	17	17	33	33
smallmoney	4	varbinary	17	17	33	33
money	8	varbinary	17	17	33	33
bit	8	varbinary	17	17	33	33
bigint	8	varbinary	17	1 7	33	33
unsigned bigint	8	varbinary	17	17	33	33
unichar(10)	2 （1 个 unichar 字符）	varbinary	33	17	49	33
unichar(10)	20 （10 个 unichar 字符）	varbinary	33	33	49	49
univarchar(20)	20 （10 个 unichar 字符）	varbinary	49	33	65	49

此版本的 Adaptive Server 不支持 text、 image 和 unitext 数据类型。

用户定义的数据类型

用户定义的数据类型建立在系统数据类型以及用户定义的 sysname 或 longsysname 数据类型基础之上。在创建了用户定义的数据类型后，可以使用它来定义列、参数和变量。根据用户定义的数据类型创建的对象将继承该用户定义的数据类型的规则、缺省值、空值类型和 IDENTITY 属性，并会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 NULL 值类型。

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 数据库中创建常用类型。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）。

Adaptive Server 允许基于任一系统数据类型，使用 sp_addtype 创建用户定义的数据类型。无法基于其它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。

但 sysname 和 longsysname 数据类型不在此例。尽管 sysname 和 longsysname 是用户定义的数据类型，仍然可以使用它们创建用户定义的数据类型。

用户定义的数据类型是数据库对象。它们的名称是区分大小写的，而且必须符合标识符的规则。

您可以使用 sp_bindrule 将规则绑定到用户定义的数据类型并使用 sp_bindefault 绑定缺省值。

缺省情况下，基于用户定义的数据类型建立的对象继承用户定义的数据类型的空值类型或 IDENTITY 属性。您可以覆盖列定义中的空值类型或 IDENTITY 属性。

使用 sp_rename 可重命名用户定义的数据类型。

使用 sp_droptype 可将用户定义的数据类型从数据库中删除。

注释　不能删除已经在表中使用的数据类型。

使用 sp_help 可显示有关系统数据类型或用户定义的数据类型的属性信息。也可以使用 sp_help 来显示表中每一列的数据类型、长度、精度和标度。

标准和遵从性

符合 ANSI SQL 的级别用户定义的数据类型是 Transact-SQL 扩展。

第 2 章

Transact-SQL 函数

本章介绍每个 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。它们往往被用作存储过程或程序的一部分。

有关如何使用这些函数的详细信息，请参见 *《Transact-SQL 用户指南》*中的第 16 章 “在查询中使用 Transact-SQL 函数”。

有关以下 XML 函数的详细信息，请参见 *《XML 服务》*：xmlextract、xmlparse、 xmlrepresentation、 xmltable、 xmltest 和 xmlvalidate。

abs

说明

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

返回 -1 的绝对值：

```
select abs(-1)

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

用法

abs 是一个数学函数，返回给定表达式的绝对值。结果与数值表达式属于同一类型并且具有相同的精度和标度。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 abs。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 ceiling、 floor、 round、 sign

acos

说明　返回具有指定余弦的角 （以弧度表示）。

语法　acos(*cosine*)

参数　*cosine*

是角的余弦，可表示为 float、real、double precision 类型（或任何可隐式转换为这些类型之一的数据类型）的列名、变量或常量。

示例　返回余弦为 0.52 的角：

```
select acos(0.52)

--------------------
            1.023945
```

用法　acos 是一个数学函数，它返回余弦为指定值的角 （以弧度表示）。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 acos。

另请参见　**文档**　*《Transact-SQL 用户指南》*

函数　cos、 degrees、 radians

ascii

说明

返回表达式中第一个字符的 ASCII 代码。

语法

ascii(*char_expr* | *uchar_expr*)

参数

char_expr

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

uchar_expr

类型为 unichar 或 univarchar 的字符型列名、变量或常量表达式。

示例

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：

```
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
```

用法

- ascii 是一个字符串函数，它返回表达式中第一个字符的 ASCII 代码。
- 当字符串函数接受两个字符表达式，但只有一个表达式为 unichar 时，另一个表达式将被 “提升” 并在内部转换为 unichar。这种转换遵循混合模式表达式的现有规则。但是，这种转换可能会导致截断，因为 unichar 数据有时要占用两倍空间。
- 如果 *char_expr* 或 *uchar_expr* 是 NULL，则返回 NULL。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 ascii。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 char、 to_unichar

asehostname

说明　返回 Adaptive Server 运行所在的物理或虚拟主机。

语法　asehostname

参数

无

示例　返回 Adaptive Server 主机名：

```
select asehostname()

-----------------------------------
linuxkernel.sybase.com
```

标准　符合 SQL/92 和 SQL/99

权限　只有具有 sa_role 的用户才可以执行 asehostname。

asin

说明

返回指定正弦的角 （以弧度表示）。

语法

asin(*sine*)

参数

sine

是角的正弦，可表示为 float、real、double precision 类型（或任何可隐式转换为这些类型之一的数据类型）的列名、变量或常量。

示例

```
select asin(0.52)

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

用法

asin 是一个数学函数，它返回其正弦为指定值的角 （以弧度表示）。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 asin。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 degrees、 radians、 sin

atan

说明　返回指定正切的角 （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之一的数据类型）的列名、变量或常量。

示例

```
select atan(0.50)

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

用法　atan 是一个数学函数，它返回其正切为指定值的角 （以弧度表示）。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 atan。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　atn2、 degrees、 radians、 tan

atn2

说明

返回具有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，可表示为 float、real、double precision 类型（或任何可隐式转换为这些类型之一的数据类型）的列名、变量或常量。

cosine

是角的余弦，可表示为 float、real、double precision 类型（或任何可隐式转换为这些类型之一的数据类型）的列名、变量或常量。

示例

```
select atn2(.50, .48)

--------------------
              0.805803
```

用法

atn2 是一个数学函数，它返回已指定正弦和余弦的角 （以弧度表示）。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 atn2。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 atan、 degrees、 radians、 tan

avg

说明

计算所有 （不同）值的数字平均值。

语法

avg([all | distinct] *expression*)

参数

all

将 avg 应用到所有值。 all 为缺省值。

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。使用集合时，表达式通常是列名。有关详细信息，请参见第 339 页上的 “表达式”。

示例

示例 1 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一个汇总值：

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

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

示例 2 当用于 group by 子句时，集合函数将为每个组（而非整个表）生成一个值。此语句将为每类书籍生成汇总值：

```
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
```

示例 3 将 titles 表按出版商分组，并且只包括那些预付款总额超过 $25,000 且书籍平均价格高于 $15 的出版商所形成的组。

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

```
having sum(advance) > $25000 and avg(price) > $15

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

用法

- avg 是一个集合函数，它查找一列中所有值的平均值。avg 只能用于数字 （整数、浮点或货币）数据类型。计算平均值时将忽略空值。
- 在计算 （有符号或无符号） int、smallint、tinyint 数据的平均值时，Adaptive Server 返回 int 值形式的结果。在计算（有符号或无符号） bigint 数据的平均值时，Adaptive Server 返回 bigint 值形式的结果。若要避免 DB-Library 程序出现溢出错误，请适当声明结果使用的变量。
- 不能对二进制数据类型使用 avg。
- 由于仅在数值数据类型上定义平均值，因此使用 avg Unicode 表达式会产生错误。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 avg。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 max、 min

audit_event_name

说明　返回对审计事件的描述。

语法　audit_event_name(*event_id*)

参数　*event_id*
是审计事件的编号。

示例　**示例 1** 查询表创建事件的审计追踪：

```
select * from audit_data where audit_event_name(event) = "Create Table"
```

示例 2 获取当前审计事件值。有关审计值及其描述的完整列表，请参见下面的 “用法”部分。

```
create table #tmp(event_id int, description varchar(255))
go
declare @a int
select @a=1
while (@a<120)
begin
      insert #tmp values (@a, audit_event_name(@a))
      select @a=@a + 1
end
select * from #tmp
go

----------------------------------------
event_id    description
---------   -------------------
      1     Ad hoc Audit Record
      2     Alter Database
      ...
      104   Create Index
      105   Drop Index
```

用法　下面列出了审计事件中每个事件的 ID 和名称：

1 Ad Hoc Audit record
2 Alter Database
3 Alter table
4 BCP In
5 NULL
6 Bind Default
7 Bind Message
8 Bind Rule
9 Create Database
10 Create Table
11 Create Procedure
12 Create Trigger
13 Create Rule
14 Create Default
15 Create Message
16 Create View
17 Access To Database
18 Delete Table
19 Delete View
20 Disk Init
21 Disk Refit
22 Disk Reinit
23 Disk Mirror
24 Disk Unmirror
25 Disk Remirror
26 Drop Database
27 Drop Table
28 Drop Procedure
29 Drop Trigger
30 Drop Rule
31 Drop Default
32 Drop Message
33 Drop View
34 Dump Database
35 Dump Transaction
36 Fatal Error
37 Nonfatal Error
38 Execution Of Stored Procedure
39 Execution Of Trigger
40 Grant Command
41 Insert Table
42 Insert View
43 Load Database
44 Load Transaction
45 Log In
46 Log Out
47 Revoke Command
48 RPC In
49 RPC Out
50 Server Boot
51 Server Shutdown
52 NULL
53 NULL
54 NULL
55 Role Toggling
56 NULL
57 NULL
58 NULL
59 NULL
60 NULL
61 Access To Audit Table
62 Select Table
63 Select View
64 Truncate Table
65 NULL
66 NULL
67 Unbind Default
68 Unbind Rule
69 Unbind Message
70 Update Table
71 Update View
72 NULL
73 Auditing Enabled
74 Auditing Disabled
75 NULL
76 SSO Changed Password
79 NULL
80 Role Check Performed
81 DBCC Command
82 Config
83 Online Database
84 Setuser Command
85 User-defined Function Command
86 Built-in Function
87 Disk Release
88 Set SSA Command
90 Connect Command
91 Reference
92 Command Text
93 JCS Install Command
94 JCS Remove Command
95 Unlock Admin Account
96 Quiesce Database Command
97 Create SQLJ Function
98 Drop SQLJ Function
99 SSL Administration
100 Disk Resize
101 Mount Database
102 Unmount Database
103 Login Command
104 Create Index
105 Drop Index
106 NULL
107 NULL
108 NULL
109 NULL
110 Deploy UDWS
111 Undeploy UDWS
115 Password Administration

注释 注意如果 audit_event_name 返回 NULL，则 Adaptive Server 不记录事件。

标准 符合 ANSI SQL 的级别：Transact-SQL 扩展。

权限 任何用户都可以执行 audit_event_name。

另请参见 **命令 .** select、 sp_audit

authmech

说明

确定指定的已登录服务器进程 ID 使用哪种鉴定机制。

语法

authmech ([*spid*])

示例

示例 1 返回服务器进程 ID 42 的鉴定机制，无论是 KERBEROS、 LDAP 还是任何其它机制：

```
select authmech(42)
```

示例 2 返回当前登录的服务器进程 ID 的鉴定机制：

```
select authmech()
```

或者

```
select authmech(0)
```

示例 3 打印用于每个登录会话的鉴定机制：

```
select suid, authmech(spid)
    from sysprocesses where suid!=0
```

用法

- 此函数从一个可选参数返回 varchar 类型的输出。
- 如果服务器进程 ID 的值为 0，此函数将返回当前客户端会话的服务器进程 ID 使用的鉴定方法。
- 如果未指定任何参数，输出将与服务器进程 ID 的值为 0 的情况相同。
- 可能的返回值包括 `ldap`、 `ase`、 `pam` 和 NULL。

权限

对 authmech 函数的权限检查因您的细化权限设置而异。

细化权限已启用	在启用细化权限的情况下，任何用户都可以执行 authmech 来查询当前个人会话。必须具有 authmech 的 select 权限才能查询其他用户的会话的详细信息。
细化权限已禁用	在禁用细化权限的情况下，任何用户都可以执行 authmech 来查询当前个人会话。您必须是具有 sso_role 的用户或具有 authmech 的 select 权限才能查询其他用户的会话的详细信息。

biginttohex

说明

返回与指定的整数等值且与平台无关的 8 字节十六进制值。

语法

biginttohex (*integer_expression*)

参数

integer_expression

是要转换为十六进制字符串的整数值。

示例

将大整数 -9223372036854775808 转换为十六进制字符串：

```
11> select biginttohex(-9223372036854775808)
2> go
----------------
8000000000000000
```

用法

- biginttohex 是一个数据类型转换函数，用于返回与整数等值且与平台无关的十六进制值 （无 “0x”前缀）。
- 使用 biginttohex 函数可进行与平台无关的整数到十六进制字符串的转换。 biginttohex 接受任何求值为 bigint 的表达式。该函数对指定表达式在任何平台上始终返回同一等值十六进制数。

另请参见

函数 convert、 hextobigint、 hextoint、 inttohex

bintostr

说明　将十六进制数字序列转换为其等价的字母数字字符串或 varbinary 数据。

语法　select bintostr(*sequence of hexadecimal digits*)

参数　*sequence of hexadecimal digits*
是有效的十六进制数字序列，由 [0 – 9]、[a – f] 和 [A – F] 组成，并且带有 “0x” 前缀。

示例　**示例 1** 将十六进制序列 “0x723ad82fe” 转换为同一值的字母数字字符串：

```
11> select bintostr(0x723ad82fe)
2> go
-----------------------------------
0723ad82fe
```

在以下示例中，十六进制数字序列及其等价的字母数字字符串的内存中表示形式为：

十六进制数字 （5 字节）

0	7	2	3	a	d	8	2	f	e

字母数字字符串 （9 字节）

0	7	2	3	a	d	8	2	f	e

该函数从右向左处理十六进制数字。在本示例中，输入数字的位数是奇数。因此，字母数字字符序列带有前缀 “0” 并且会反映在输出中。

示例 2　将名为 *@bin_data* 的局部变量的十六进制数字转换为与 “723ad82fe” 值等价的字母数字字符串：

```
declare @bin_data varchar(30)
select @bin_data = 0x723ad82fe
select bintostr(@bin_data)
go

----------
0723ad82fe
```

用法

- 输入中的任何无效字符均会导致输出为 NULL。
- 输入必须是有效的 varbinary 数据。
- NULL 输入会生成 NULL 输出。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 bintostr。

另请参见　**函数**　strtobin

cache_usage

说明

以表所属高速缓存中所有对象的百分比形式返回高速缓存使用情况。

语法

cache_usage(*table_name*)

参数

table_name

是表的名称。该名称可以是完全限定名（即，可包括数据库和所有者名称）。

示例

示例 1 返回 titles 表使用的高速缓存的百分比：

```
select cache_usage("titles")
----------------------------
                      98.876953
```

示例 2 从 master 数据库返回 authors 表使用的高速缓存的百分比

```
select cache_usage ("pubs2..authors")
-----------------------------------
                                98.876953
```

用法

- cache_usage 以跨所有高速缓存池的百分比形式提供高速缓存使用情况。
- cache_usage 不提供任何有关当前对象所使用的高速缓存大小的信息，也不提供有关索引（如果它们绑定到不同的高速缓存中）的高速缓存使用情况的信息。
- （在集群环境中） cache_usage 提供当前节点中对象绑定到的高速缓存的高速缓存使用情况。

权限

任何用户都可以执行 cache_usage。

case

说明

支持条件 SQL 表达式；可用于任何能够使用值表达式的情况。

语法

case 和 *expression* 语法：

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

case 和 *value* 语法：

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

参数

case

开始 case 表达式。

when

位于搜索条件或需要比较的表达式之前。

search_condition

用来为选择的结果设置条件。case 表达式的搜索条件与 where 子句中的搜索条件相似。搜索条件在 *《Transact-SQL 用户指南》* 中进行了详细讨论。

then

位于指定 case 结果值的表达式之前。

expression 和 *value*

可以是列名、常量、函数、子查询或者任何由算术运算符或逐位运算符连接起来的列名、常量和函数的组合。有关表达式的详细信息，请参见第 339 页上的 “表达式”。

else

是可选的。如未指定，则表示使用 else null。

示例

示例 1 从 authors 表中选择所有作者，并且对于特定作者，指定他们所居住的城市：

```
select au_lname, postalcode,
    case
            when postalcode = "94705"
                then "Berkeley Author"
            when postalcode = "94609"
                then "Oakland Author"
```

```
                when postalcode = "94612"
                    then "Oakland Author"
                when postalcode = "97330"
                    then "Corvallis Author"
            end
    from authors
```

示例 2 返回 discounts 表的 lowqty 或 highqty 列中第一次出现的非空值：

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

还可以使用以下格式以生成相同的结果，因为 coalesce 是 case 表达式的缩写形式：

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

示例 3 从 *titles* 表中选择 *titles* 和 *type*。如果书籍类型为 UNDECIDED，nullif 将返回一个 NULL 值：

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

还可以使用以下格式以生成相同的结果，因为 nullif 是 case 表达式的缩写形式：

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

示例 4 生成错误消息，因为至少一个表达式必须是 null 关键字以外的其它值：

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

```
所有 CASE 表达式的结果不能为 NULL。
```

示例 5 生成错误消息，因为 coalesce 后面至少有两个表达式：

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

在 COALESCE 表达式中使用单个 coalesce 元素是非法的。

示例 6 这个含 *values* 的 case 示例可为员工更新工资信息：

```
update employees
    set salary =
        case dept
         when 'Video' then salary * 1.1
         when 'Music' then salary * 1.2
         else 0
        end
```

示例 7 在 movie_titles 表中， movie_type 列以整数进行编码，而不是以拼写“Horror”、“Comedy”、“Romance”和“Western”所需的 cha(10) 进行编码。但是，会通过使用 case 表达式向应用程序返回一个文本字符串：

```
select title,
    case movie_type
        when 1 then 'Horror'
        when 2 then 'Comedy'
        when 3 then 'Romance'
        when 4 then 'Western'
        else null
    end,
    our_cost
from movie_titles
```

用法

使用：

- case 表达式允许使用 when...then 结构 （而非 if 语句）来表达搜索条件，从而简化了标准 SQL 表达式。
- 比较值时，可将 *value* 与 case 结合使用，其中 *value* 为所需值。如果 *value* 等于 *expression*，则 case 的值为 *result*。如果 value1 不等于 express，则 valuet 与 value2 相比较。如果值等于 value2，则 CASE 的值为 result2。如果 value1 ... valuen 中的任何值都不等于所需的 valuet，则 CASE 的值为 resultx。所有的 resulti 可以是值表达式，也可以是关键字 NULL。所有 valuei 都必须为可比类型，且所有结果都必须具有可比数据类型。 SQL 中可以使用
- 表达式的地方都可以使用 case 表达式的数据类型。
- 如果查询生成了多种数据类型，那么数据类型层次就决定了 case 表达式结果的数据类型。如第 6 页上的“混合模式表达式的数据类型”中所述。如果指定了两种 Adaptive Server 不能隐式转换的数据类型（例如， char 和 int），查询将会失败。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限	case 权限缺省情况下授予所有用户。使用它无需任何权限。
另请参见	**命令** coalesce、 nullif、 if...else、 select、 where clause

cast

说明

将指定的值转换为其它数据类型。

语法

cast (*expression* as *datatype* [(*length* | *precision*[, *scale*])])

参数

expression

是要从一种数据类型或日期格式转换为另一种数据类型或日期格式的值。它包括列、常量、函数以及由算术运算符或逐位运算符或子查询连接在一起的常量和函数的任意组合。

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

当 unichar 用作目标数据类型时，如果未指定长度，则使用缺省长度，即 30 个 Unicode 值。

length

是与 char、 nchar、 unichar、 univarchar、 varchar、 nvarchar、 binary 和 varbinary 数据类型一起使用的可选参数。如果不提供长度， 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。字符和二进制表达式可以具有的最大长度是 64K。

precision

是 numeric 或 decimal 数据类型中的有效位数。对于 float 数据类型，精度是指尾数中有效的二进制位数。如果不提供精度，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。如果不提供标度， Adaptive Server 将使用缺省标度 0。

示例

示例 1 将日期转换为更加易读的 datetime 格式：

```
select cast("01/03/63" as datetime)
go

 --------------------------
        Jan  3 1963 12:00AM

(1 row affected)
```

示例 2 将标题数据库中的 total_sales 列转换为 12 字符列：

```
select title, cast(total_sales as char(12))
```

用法

- cast 对 date 和 time 数据类型使用缺省格式。
- 当 cast 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。

- 无法使用 null/not null 关键字指定结果数据类型是否可为空值。但是，您可使用含空值本身的 cast 得到可为空的结果数据类型。要将值转换为可为空的数据类型，可使用允许使用 null/not null 关键字的 convert() 函数。
- 您可以使用 cast 将 image 列转换为 binary 或 varbinary。这种转换受到 binary 数据类型的最大长度的限制，该长度由服务器逻辑页大小的最大列大小决定。如果未指定长度，转换后的值将具有缺省的长度（30 个字符）。
- 可以将 unichar 表达式用作目标数据类型，也可将它们转换为其它数据类型。 unichar 表达式可在服务器支持的任何其它数据类型间显式或隐式转换。
- 当 unichar 用作目标类型时，如果您未指定长度，则使用缺省长度，即 30 个 Unicode 值。如果目标类型的长度不足以容纳给定的表达式，就会显示出错消息。

隐式转换

如果两种数据类型的主字段不匹配，那么在这两种数据类型之间进行隐式转换可能导致数据截断、插入缺省值或产生错误消息。例如，如果将 datetime 值转换为 date 值，时间部分将被截断，只留下日期部分。如果将 time 值转换为 datetime 值，将在新的 datetime 值中添加缺省的日期部分 Jan 1, 1900。如果将 date 值转换为 datetime 值，则将向 datetime 值中添加缺省的时间部分 00:00:00:000。

```
DATE -> VARCHAR、 CHAR、 BINARY、 VARBINARY、 DATETIME、 SMALLDATETIME
TIME -> VARCHAR、 CHAR、 BINARY、 VARBINARY、 DATETIME、 SMALLDATETIME
VARCHAR、 CHAR、 BINARY、 VARBINARY、 DATETIME、 SMALLDATETIME -> DATE
VARCHAR、 CHAR、 BINARY、 VARBINARY、 DATETIME、 SMALLDATETIME -> TIME
```

显式转换

如果试图将 date 显式转换为 datetime，并且值（如“Jan 1, 1000”）超出 datetime 范围，则不允许转换并出现提示性的错误消息。

```
DATE -> UNICHAR、 UNIVARCHAR
TIME -> UNICHAR、 UNIVARCHAR
UNICHAR、 UNIVARCHAR -> DATE
UNICHAR、 UNIVARCHAR -> TIME
```

涉及 Java 类的转换

- 如果数据库中已启用 Java，则可按下列方法使用 cast 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，则会将安装在 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。

转换的结果与当前的数据库相关联。

标准　　符合 ANSI SQL 的级别符合 ANSI 标准。

权限　　任何用户都可以执行 cast。

ceiling

说明 返回大于或等于指定值的最小整数。

语法 ceiling(*value*)

参数 *value*

是列、变量或表达式，它们的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之一的类型。

示例 **示例 1** 返回值 124：

```
select ceiling(123.45)

124
```

示例 2 返回值 -123：

```
select ceiling(-123.45)

-123
```

示例 3 返回值 24.000000：

```
select ceiling(1.2345E2)

24.000000
```

示例 4 返回值 -123.000000：

```
select ceiling(-1.2345E2)

-123.000000
```

示例 5 返回值 124.00

```
select ceiling($123.45)

124.00
```

示例 6 从 salesdetail 表（其中 title_id 为值“PS3333”）返回“discount”的值：

```
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
```

用法

- ceiling 是一个数学函数，它返回大于或等于指定值的最小整数。返回值与所提供值的数据类型相同。

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

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 ceiling。

另请参见

文档　*《Transact-SQL 用户指南》*

命令　set

函数　abs、 floor、 round、 sign

char

说明

返回整数的等值字符。

语法

char(*integer_expr*)

参数

integer_expr

是任意整数 （tinyint、 smallint 或 int）类型的列名、变量或 0 到 255 之间的常量表达式。

示例

示例 1

```
select char(42)

-
*
```

示例 2

```
select xxx = char(65)

xxx
---
A
```

用法

- char 是一个字符串函数，用于将单字节整数值转换为字符值 （char 通常用作 ascii 的反函数）。
- 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，则返回 NULL。

用 char 重新格式化输出

- 您可以使用并置和 char 值添加制表符或回车，从而将输出重新格式化。 char(10) 转换为回车； char(9) 转换为制表符。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	符合 ANSI SQL 的级别：Transact-SQL 扩展。
权限	任何用户都可以执行 char。
另请参见	**文档**　*《Transact-SQL 用户指南》*
	函数　ascii、 str

char_length

说明

返回表达式中字符的数量。

语法

char_length(*char_expr* | *uchar_expr*)

参数

char_expr

类型为 char、varchar、nchar、text_locator、unitext_locator 或 nvarchar 的字符型列名、变量或常量表达式。

uchar_expr

类型为 unichar 或 univarchar 的字符型列名、变量或常量表达式。

示例

示例 1

```
select char_length(notes) from titles
    where title_id = "PC9999"
 -----------
          39
```

示例 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
```

用法

- char_length 是一个字符串函数，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。
- 对于压缩的大对象 (LOB) 列， char_length 将返回原始明文字符数。
- 对于可变长度的列和变量， char_length 将返回字符数 （而不是定义的列或变量长度）。如果可变长度变量中包含显式尾随空白，则不会将其删除。对于文字和固定长度的字符列和变量，char_length 不删除尾随空白的表达式 （请参见示例 2）。
- 对于 unitext、unichar 和 univarchar 列， char_length 返回 Unicode 值（16 位）的数量，其中一个代理对算作两个 Unicode 值。例如，如果 unitext 列 ut 包含行值 U+0041U+0042U+d800dc00，则将返回以下信息：

  ```
  select char_length(ut) from unitable
  ------------
  4
  ```

- 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。

- 对于 Unicode 表达式，返回表达式中 Unicode 值 （不是字节）字符的数量。将对计数替换为两个 Unicode 值。
- 如果 *char_expr* 或 *uchar_expr* 是 NULL，则 char_length 返回 NULL。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 char_length。

另请参见　**文档**　*《Transact-SQL 用户指南》*

函数　datalength

charindex

说明

返回表示表达式起始位置的整数。

语法

charindex(*expression1*, *expression2* [, *start*])

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、 nvarchar、 unichar、 univarchar、 binary、 text_locator、unitext_locator、 image_locator 或 varbinary。

start

指定后，会导致对 *expression1* 的搜索在 *expression2* 中的给定偏移处开始。如果不给定 *start*，搜索就会在 *expression2* 的开头开始。 *start* 可以是一个表达式，但必须返回整数值。

示例

示例 1 返回字符表达式 “Wonderful” 在 titles 表 notes 列上的起始位置。

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

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

示例 2 此查询成功执行，并返回零行。列 spt_values.name 被定义为 varchar(35)：

```
select name
from spt_values
where charindex( 'NO', name, 1000 ) > 0
```

在比较中，此查询不使用 *start*，返回字符表达式 “Wonderful” 在 titles 表的 notes 列上的起始位置：

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

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

用法

- charindex 是一个字符串函数，它在 *expression2* 中搜索首次出现的 *expression1*，然后返回表示其起始位置的整数。如果未找到 *expression1*，则 charindex 返回 0。
- 如果 *expression1* 包含通配符， charindex 就会将它们当作文字。
- 如果 *expression2* 为 NULL，则返回 0。

- 如果 varchar 表达式是作为一个参数提供的，并且 unichar 表达式是作为另一个参数提供的，则 varchar 表达式就会隐式转换为 unichar（可能会发生截断）。
- 如果 *expression1* 和 *expression2* 只有一个是定位符，则另一表达式的数据类型必须可隐式转换为该定位符所引用的 LOB 的数据类型。
- 当 *expression1* 是定位符时，该定位符所引用的 LOB 的最大长度是 16KB。
- *start* 值对于 varchar、univarchar、text_locator 和 unitext_locator 数据类型会被解释为在开始搜索之前跳过的字符数，对于 binary 和 image_locator 数据类型会被解释为字节数。
- *expression1* 的最大长度是 16,384 字节。
- 如果 varchar 表达式是作为一个参数提供的，并且 unichar 表达式是作为另一个参数提供的，则 varchar 表达式就会隐式地转换为 unichar（可能会发生截断）。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 charindex。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　patindex

coalesce

说明

支持条件 SQL 表达式；可用于任何可以使用值表达式的情况； case 表达式的替代形式。

语法

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

参数

coalesce

求出所列表达式的值并返回第一个非空值。如果所有表达式都为空，则 coalesce 返回 NULL。

expression

可以是列名、常量、函数、子查询或者任何由算术运算符或逐位运算符连接起来的列名、常量和函数的组合。有关表达式的详细信息，请参见第 337 页上的 “表达式”。

示例

示例 1 返回 discounts 表的 lowqty 或 highqty 列中第一次出现的非空值：

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

示例 2 这是编写上一个示例的另一种方法：

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

用法

- coalesce 表达式允许以简单的比较 （而非使用 when...then 结构）来表达搜索条件，从而简化了标准 SQL 表达式。
- SQL 中可使用表达式的地方都可以使用 coalesce 表达式。
- coalesce 表达式的结果中至少必须有一个返回非空值。本示例生成以下错误消息：

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

```
所有 CASE 表达式的结果不能为 NULL。
```

- 如果查询生成了多种数据类型，那么数据类型层次就决定了 case 表达式结果的数据类型，如第 6 页上的“混合模式表达式的数据类型”中所述。如果指定了两种 Adaptive Server 不能隐式转换的数据类型（例如， char 和 int），查询将会失败。
- coalesce 是 case 表达式的缩写形式。例 2 描述了书写 coalesce 语句的另一种方法。

- coalesce 后面必须至少有两个表达式。本示例生成以下错误消息：

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

在 COALESCE 表达式中使用单个 coalesce 元素是非法的。

标准　　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　　任何用户都可以执行 coalesce。

另请参见　　**命令**　case、 nullif、 select、 if...else、 where clause

col_length

说明

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。该名称可以是完全限定名 （即，可包括数据库和所有者名称），并且必须带有引号。

column_name

是列名。

示例

查找 titles 表中 title 列的长度。“x”在结果中提供列标题：

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

 x
----
   80
```

用法

- col_length 是一个系统函数，它返回已定义的列长度。
- 要查找每行所存储的数据的实际长度，可使用 datalength。
- 对于 text、unitext 和 image 列，col_length 将返回 16，这是指向实际文本页的 binary(16) 指针的长度。
- 对于 unichar 列，所定义的长度为定义该列时声明的 Unicode 值的数量 （并非所代表的字节数量）。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 col_length。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 datalength

col_name

说明

返回已指定表 ID 和列 ID 的列的名称，且最长可为 255 个字节。

语法

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

参数

object_id

是一个数值表达式，它是表、视图或其它数据库对象的对象 ID。它们存储在 sysobjects 的 id 列中。

column_id

是一个数值表达式，它表示一列的列 ID。它们存储在 syscolumns 的 colid 列中。

database_id

是一个数值表达式，它表示数据库的 ID。它们存储在 sysdatabase 的 db_id 列中。

示例

```
select col_name(208003772, 2)

------------------------------
title
```

用法

- col_name 是一个系统函数，它返回列的名称。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 col_name。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　db_id、 object_id

compare

说明

您可以根据替代归类规则直接比较两个字符串。

语法

compare ({*char_expression1* | *uchar_expression1*},
{*char_expression2*|*uchar_expression2*}),
[{*collation_name* | *collation_ID*}]

参数

char_expression1 或 *uchar_expression1*
是要与 *char_expression2* 或 *uchar_expression 2* 进行比较的字符表达式。

char_expression2 或 *uchar_expression2*
是要与 *char_expression1* 或 *uchar_expression1* 进行比较的字符表达式。

char_expression1 和 *char_expression2* 可以是：

- 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

uchar_expression1 和 *uchar_expression2* 可以是：

- 字符类型 （unichar 或 univarchar）
- 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
是引号内的字符串或字符变量，用来指定要使用的归类。第 82 页上的表 2-2 显示了有效值。

collation_ID
是一个整数常量或变量，它指定要使用的归类。第 82 页上的表 2-2 显示了有效值。

示例

示例 1 比较 aaa 和 bbb：

```
1> select compare ("aaa","bbb")
2> go

 -----------
          -1
(1 row affected)
```

另外，您也可以使用下面的格式比较 aaa 和 bbb：

```
1> select compare (("aaa"),("bbb"))
2> go
```

```
 -----------
          -1
(1 row affected)
```

示例 2 比较 aaa 和 bbb 并指定二进制排序顺序：

```
1> select compare ("aaa","bbb","binary")
2> go

 -----------
          -1
(1 row affected)
```

另外，您可以使用下面的格式比较 aaa 和 bbb，并在比较中使用归类 ID 而不是归类名称：

```
1> select compare (("aaa"),("bbb"),(50))
2> go

 -----------
          -1
(1 row affected)
```

用法

- compare 函数根据所选归类规则返回以下值：
  - 1 — 表示 *char_expression1* 或 *uchar_expression1* 大于 *char_expression2* 或 *uchar_expression2*。
  - 0 — 表示 *char_expression1* 或 *uchar_expression1* 等于 *char_expression2* 或 *uchar_expression2*。
  - -1 — 表示 *char_expression1* 或 *uchar_expression1* 小于 *char_expression2* 或 *uchar expression2*。
- compare 可为每个输入字符最多生成六个字节的归类信息。因此，使用 compare 的结果可能超过 varbinary 数据类型的长度限制。如果发生这种情况，结果将被截断，以符合限制要求。Adaptive Server 将发出警告消息，但继续运行包含 compare 函数的查询或事务。由于此限制取决于服务器的逻辑页大小，因此截断操作会删除每个输入字符的结果字节，直到最终得到的字符串小于 DOL 锁定表和 APL 表中的以下值：

表 2-1：行和列的最大长度 — APL 和 DOL

锁定方案	页大小	最大行长度	最大列长度
APL 表	2K （2048 字节）	1962	1960 字节
	4K （4096 字节）	4010	4008 字节
	8K （8192 字节）	8106	8104 字节
	16K （16384 字节）	16298	16296 字节

锁定方案	页大小	最大行长度	最大列长度
DOL 表	2K （2048 字节）	1964	1958 字节
	4K （4096 字节）	4012	4006 字节
	8K （8192 字节）	8108	8102 字节
	16K （16384 字节）	16300	16294 字节 （如果表不包含任何可变长度列）
	16K （16384 字节）	16300 （取决于 varlen 的最大起始偏移 = 8191）	81918191-6-2 = 8183 字节 （如果表至少包含一个可变长度列）。 *

* 此大小包含六个字节的行开销和两个字节的行长度字段

- *char_expression1*、 *uchar_expression1* 和 *char_expression2*、*uchar_expression2* 都必须是用服务器缺省字符集进行编码的字符。
- *char_expression1*、 *uchar_expression 1* 或 *char_expression2* 及 *uchar_expression2* 中可以有一个或者两个都可以为空字符串：
  - 如果 *char_expression2* 或 *uchar_expression2* 为空，则函数返回 1。
  - 如果两个字符串都为空，则它们相等，函数返回 0。
  - 如果 *char_expression1* 或 *uchar_expression 1* 为空，函数返回 -1。

  compare 函数不等于空字符串以及只包含空格的字符串。 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*、 *uchar_expression1* ； 或 *char_expression2*、*uchar_expression2* 是 NULL，则结果为 NULL。
- 如果 varchar 表达式是作为一个参数提供的，并且 unichar 表达式是作为另一个参数提供的， varchar 表达式就会隐式地转换为 unichar （可能会发生截断）。
- 如果未指定 *collation_name* 或 *collation_ID* 的值， compare 将假定为二进制归类。
- 表 2-2 列出了 collation_name 和 collation_ID 的有效值。

表 2-2：归类名称和 ID

说明	归类名称	归类 ID
缺省的 Unicode 多语种	缺省	20
泰文字典顺序	thaidict	21
ISO14651 标准	iso14651	22
UTF-16 排序 — 与 UTF-8 二进制排序匹配	utf8bin	24
CP 850 方案 — 没有变音	altnoacc	39

说明	归类名称	归类 ID
CP 850 方案 — 小写优先	altdict	45
CP 850 西欧 — 没有大小写优先级	altnocsp	46
CP 850 斯堪的纳维亚文 — 字典排序	scandict	47
CP 850 斯堪的纳维亚文 — 不区分大小写，具有优先级	scannocp	48
GB 拼音	gbpinyin	不可用
二进制排序	binary	50
Latin-1 英文、法文、德文字典	dict	51
Latin-1 英文、法文、德文，没有大小写	nocase	52
Latin-1 英文、法文、德文，没有大小写优先级	nocasep	53
Latin-1 英文、法文、德文，没有变音	noaccent	54
Latin-1 西班牙文字典	espdict	55
Latin-1 西班牙文，没有大小写	espnocs	56
Latin-1 西班牙文，没有变音	espnoac	57
ISO 8859-5 俄文字典	rusdict	58
ISO 8859-5 俄文，没有大小写	rusnocs	59
ISO 8859-5 古斯拉夫文字典	cyrdict	63
ISO 8859-5 古斯拉夫文，没有大小写	cyrnocs	64
ISO 8859-7 希腊文字典	elldict	65
ISO 8859-2 匈牙利文字典	hundict	69
ISO 8859-2 匈牙利文，没有变音	hunnoac	70
ISO 8859-2 匈牙利文，没有大小写	hunnocs	71
ISO 8859-9 土耳其文字典	turdict	72
ISO 8859-9 土耳其文，没有变音	turknoac	73
ISO 8859-9 土耳其文，没有大小写	turknocs	74
CP932 二进制排序	cp932bin	129
中文拼音排序	dynix	130
GB2312 二进制排序	gb2312bn	137
通用古斯拉夫文字典	cyrdict	140
土耳其文字典	turdict	155
EUCKSC 二进制排序	euckscbn	161
中文拼音排序	gbpinyin	163
俄文字典排序	rusdict	165
SJIS 二进制排序	sjisbin	179
EUCJIS 字典排序	eucjisbn	192
BIG5 二进制排序	big5bin	194
Shift-JIS 二进制顺序	sjisbin	259

标准	符合 ANSI SQL 的级别 Transact-SQL 扩展。
权限	任何用户都可以执行 compare。
另请参见	**函数** sortkey

convert

说明

将指定的值转换成另一数据类型或其它 datetime 显示格式。

语法

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

参数

datatype

系统提供的数据类型（例如，char(10)、unichar (10)、varbinary (50) 或 int），表达式可以转换成这些数据类型。不能使用用户定义的数据类型。

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

length

是与 char、 nchar、 unichar、 univarchar、 varchar、 nvarchar、 binary 和 varbinary 数据类型一起使用的可选参数。如果不提供长度， 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。字符和二进制表达式可以具有的最大长度是 64K。

precision

是 numeric 或 decimal 数据类型中的有效位数。对于 float 数据类型，精度是指尾数中有效的二进制位数。如果不提供精度， 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

是要从一种数据类型或日期格式转换为另一种数据类型或日期格式的值。

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

当 unichar 用作目标数据类型时，如果未指定长度，则使用缺省长度，即 30 个 Unicode 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，则使用 *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，则使用 表 2-3 中的样式编号指定显示格式。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，或使用中间列的值。

在将 date 数据转换为字符类型时，将使用表 2-3 中的样式编号 1 到 7（101 到 107）或 10 到 12 （110 到 112）来指定显示格式。缺省值为 100 （mon dd yyyy hh:miAM （或 PM））。如果 date 数据转换为包含时间部分的样式，该时间部分将显示缺省值 0。

在将 time 数据转换为字符类型时，使用样式编号 8 或 9（108 或 109）来指定显示格式。缺省值为 100（mon dd yyyy hh:miAM（或 PM））。如果 time 数据转换为包含日期部分的样式，将显示缺省日期 Jan 1, 1900。

表 2-3：使用 style 参数的日期格式转换

不含世纪 (yy)	含世纪 (yyyy)	标准	输出
-	0 或 100	Default	*mon dd yyyy hh:mm* AM （或 PM）
1	101	美国	*mm/dd/yy*
2	2	SQL 标准	*yy.mm.dd*
3	103	英语 / 法语	*dd/mm/yy*
4	104	德语	*dd.mm.yy*
5	105		*dd-mm-yy*
6	106		*dd mon yy*
7	107		*mon dd, yy*
8	108		*HH:mm:ss*
-	9 或 109	缺省值 + 毫秒	*mon dd yyyy hh:mm:ss* AM（或 PM）
10	110	美国	*mm-dd-yy*
11	111	日本	*yy/mm/dd*
12	112	ISO	*yymmdd*
13	113		*yy/dd/mm*
14	114		*mm/yy/dd*
14	114		*hh:mi:ss:mmm*AM （或 PM）
15	115		*dd/yy/mm*
-	16 或 116		*mon dd yyyy HH:mm:ss*
17	117		*hh:mmAM*

说明 “mon”表示英文拼写的月份，“mm”表示月份或分钟。“HH”表示 24 小时制时钟值，“hh”表示 12 小时制时钟值。最后一行 （第 23 行）包含文字 “T”以分离格式中的日期部分和时间部分。

不含世纪 (yy)	含世纪 (yyyy)	标准	输出
18	118		*HH:mm*
19			*hh:mm:ss:zzzAM*
20			*hh:mm:ss:zzz*
21			*yy/mm/dd HH:mm:ss*
22			*yy/mm/dd HH:mm* AM （或 PM）
23			*yyyy-mm-ddTHH:mm:ss*

说明 “mon”表示英文拼写的月份，“mm”表示月份或分钟。“HH”表示 24 小时制时钟值，“hh”表示 12 小时制时钟值。最后一行 （第 23 行）包含文字 “T”以分离格式中的日期部分和时间部分。

缺省值 （*style* 0 或 100）和 *style* 9 或 109 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

示例 1

```
select title, convert(char(12), total_sales)
from titles
```

示例 2

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

示例 3 将当前日期转换为样式 “3”， dd/mm/yy：

```
select convert(char(12), getdate(), 3)
```

示例 4 如果值 pubdate 可以为空，则必须使用 varchar 而不是 char，否则可能出错：

```
select convert(varchar(12), pubdate, 3) from titles
```

示例 5 返回字符串 “0x00000100”的等值整数。结果会因平台的不同而变化：

```
select convert(integer, 0x00000100)
```

示例 6 将特定于平台的位模式作为 Sybase 二进制类型返回：

```
select convert (binary, 10)
```

示例 7 返回 1，它是 $1.11 的等值位字符串：

```
select convert(bit, $1.11)
```

示例 8 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，也使用不允许空值的 #tempsales.total_sales 来创建 #tempsales：

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

用法

- convert 是一个数据类型转换函数，它在多种数据类型之间进行转换并将日期 / 时间及货币数据重新格式化，以便于显示。
- 如果它们被压缩， convert 会在将它们转换为其它数据类型之前先解压缩大对象 (LOB) 列。
- convert — 返回已转换为其它数据类型或其它 datetime 显示格式的指定值。当从 unitext 向其它字符和二进制数据类型转换时，结果受到目标数据类型的最大长度的限制。如果未指定长度，转换后的值将具有缺省大小 （30 个字节）。如果正在使用 enabled enable surrogate processing，将返回整个代理对。例如，如果将包含数据 U+0041U+0042U+20acU+0043 （代表 “AB unitext 列转换为 UTF-8 varchar(3) 列，则会返回以下信息：

  ```
  select convert(varchar(3), ut) from untable
  ---
  AB
  ```

- 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 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（参见前面的示例 8）。

  在以下条件下，结果将为未定义的值：

  - 所转换的表达式将是 not null 结果。
  - 表达式的值为空。

  使用下面的 select 语句能够为可预测的结果生成已知的非空值：

```
select convert(int not null isnull(col2, 5)) from table1
```

- 您可以使用 convert 将 image 列转换为 binary 或 varbinary。这种转换受到 binary 数据类型的最大长度的限制，该长度由服务器逻辑页大小的最大列大小决定。如果未指定长度，转换后的值将具有缺省的长度 （30 个字符）。

- 可以将 unichar 表达式用作目标数据类型，也可将它们转换成其它数据类型。 unichar 表达式可在服务器支持的任何其它数据类型间显式或隐式转换
- 当 unichar 用作目标类型时，如果您未指定长度，则使用缺省长度，即 30 个 Unicode 值。如果目标类型的长度不足以容纳给定的表达式，就会显示出错消息。

隐式转换

如果两种数据类型的主字段不匹配，那么在这两种数据类型之间进行隐式转换可能导致数据截断、插入缺省值或产生错误消息。例如，如果将 datetime 值转换为 date 值，时间部分将被截断，只留下日期部分。如果将 time 值转换为 datetime 值，将在新的 datetime 值中添加缺省的日期部分 Jan 1, 1900。如果将 date 值转换为 datetime 值，则将向 datetime 值中添加缺省的时间部分 00:00:00:000。

```
DATE -> VARCHAR、 CHAR、 BINARY、 VARBINARY、 DATETIME、 SMALLDATETIME
TIME -> VARCHAR、 CHAR、 BINARY、 VARBINARY、 DATETIME、 SMALLDATETIME
VARCHAR、 CHAR、 BINARY、 VARBINARY、 DATETIME、 SMALLDATETIME -> DATE
VARCHAR、 CHAR、 BINARY、 VARBINARY、 DATETIME、 SMALLDATETIME -> TIME
```

显式转换

如果试图将 date 显式转换为 datetime，并且值（如“Jan 1, 1000”）超出 datetime 范围，则不允许转换并出现提示性的错误消息。

```
DATE -> UNICHAR、 UNIVARCHAR
TIME -> UNICHAR、 UNIVARCHAR
UNICHAR、 UNIVARCHAR -> DATE
UNICHAR、 UNIVARCHAR -> TIME
```

涉及 Java 类的转换

- 如果数据库中已启用 Java，则可按下列方法使用 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，则会将安装在 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。

  转换的结果与当前的数据库相关联。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 convert。

另请参见

文档 *《Transact-SQL 用户指南》*； *《Adaptive Server Enterprise 中的 Java》*（其中列出了允许的数据类型映射，并提供了涉及 Java 类的数据类型转换的详细信息）。

数据类型 用户定义的数据类型

函数 hextoint、 inttohex

cos

说明　返回指定角 （以弧度表示）的余弦。

语法　cos(*angle*)

参数　*angle*
是任何近似的数值型 （float、 real 或 double precision）列名、变量或常量表达式。

示例

```
select cos(44)

 0.999843
```

用法

- cos 是一个数学函数，它返回指定角 （以弧度表示）的余弦。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 cos。

另请参见　**文档**　*《Transact-SQL 用户指南》*

函数　acos、 degrees、 radians、 sin

cot

说明

返回指定角 （以弧度表示）的余切。

语法

cot(*angle*)

参数

angle

是任何近似的数值型 （float、 real 或 double precision）列名、变量或常量表达式。

示例

```
select cot(90)

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

用法

- cot 是一个数学函数，它返回指定角 （以弧度表示）的余切。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 cot。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 degrees、 radians、 sin

count

说明

以整数形式返回 （不同）非空值的数量或选定的行数。

语法

count([all | distinct] *expression*)

参数

all

将 count 应用到所有值。 all 是缺省值。

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。使用集合时，表达式通常是列名。有关详细信息，请参见第 339 页上的 “表达式”。

示例

示例 1 查找作者居住的不同城市的数量：

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

示例 2 列出 titles 表中的类型，但消除只包含一本书或不包含任何书的类型：

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

用法

- count 是一个集合函数，用于查找列中非空值的数量。
- 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，包括 unichar。进行计数时将忽略空值。
- 对于空表、只包含空值的列，以及只包含空值的组，count(*column_name*) 会返回一个为 0 的值。
- 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**选择列表**中包括 count(*)，以生成连接结果中的行数。如果是为了计算一个表中符合标准的行数，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，所以 exists 或 in 返回结果的速度可能更快。例如：

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

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 count。

另请参见

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

文档 *《Transact-SQL 用户指南》*

count_big

说明

以 bigint 形式返回 （不同）非空值的数量或选定的行数。

语法

count_big([all | distinct] *expression*)

参数

all

将 count_big 应用到所有值。 all 是缺省值。

distinct

在应用 count_big 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。使用集合时，表达式通常是列名。

示例

查找 systypes 中 *name* 出现的次数：

```
1> select count_big(name) from systypes
2> go
--------------------
42
```

用法

- count_big 是一个集合函数，用于查找列中非空值的数量。
- 指定 distinct 后， count_big 将查找唯一非空值的数量。进行计数时将忽略空值。
- 对于空表、只包含空值的列，以及只包含空值的组，count_big(*column_name*) 会返回一个为 0 的值。
- count_big(*) 查找行数。 count_big(*) 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，应在选择列表中包括 count_big(*)，以生成连接结果中的行数。如果是为了计算一个表中符合标准的行数，可使用 count_big(*column_name*)。
- count_big 可在子查询中用来进行存在性检查。例如：

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

  然而，因为 count_big 对所有匹配值进行计数，所以 exists 或 in 返回结果的速度可能更快。例如：

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

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限	任何用户都可以执行 count_big。
另请参见	**命令** compute 子句、 group by 和 having 子句、 select、 where 子句

create_locator

说明

为指定的 LOB 显式创建定位符，然后返回该定位符。

由 create_locator 创建的定位符仅在包含使用了 create_locator 的查询的事务的存续期间内有效。如果未启动任何事务，则该定位符仅在包含 create_locator 的查询执行完毕之前有效。

语法

create_locator (*datatype*, *lob_expression*)

参数

datatype

是 LOB 定位符的数据类型。有效值为：

- text_locator
- unitext_locator
- image_locator

lob_expression

数据类型为 text、 unitext 或 image 的 LOB 值。

示例

示例 1 根据简单的文本表达式创建文本定位符：

```
select create_locator(text_locator, convert (text, "abc"))
```

示例 2 创建一个类型为 text_locator 的局部变量 *@v*，然后使用 *@v* 作为在 my_table 的 textcol 列中存储的 LOB 的句柄来创建一个定位符。

```
declare @v text_locator

    select @v = create_locator(text_locator, textcol) from my_table where
id=10
```

权限

任何用户都可以执行 create_locator。

另请参见

命令 deallocate locator、 truncate lob

Transact-SQL 函数 locator_literal、 locator_valid、 return_lob

current_bigdatetime

说明

返回一个 bigtime 值，该值表示具有微秒级精度的当前时间。当前时间部分的精度受到系统时钟准确性的限制。

语法

current_bigdatetime()

参数

无

示例

示例 1 查找当前 bigdatetime：

```
select current_bigdatetime())
------------------------------
Nov 25 1995 10:32:00.010101AM
```

示例 2 查找当前 bigdatetime：

```
select datepart(us, current_bigdatetime())
------------------------------
010101
```

用法

查找服务器上存在的当前日期。

标准

符合 ANSI SQL 的级别符合初级- 标准。

权限

任何用户都可以执行 current_date。

另请参见

数据类型 日期和时间数据类型

命令 select、 where clause

函数 dateadd、 datediff、 datepart、 datename、 current_bigtime

current_bigtime

说明

返回一个 bigtime 值，该值表示具有微秒级精度的当前时间。当前时间部分的精度受到系统时钟准确性的限制。

语法

current_bigtime()

参数

无

示例

示例 1 查找当前 bigtime：

```
select current_bigtime())
------------------------------
10:32:00.010101AM
```

示例 2 查找当前 bigtime：

```
select datepart(us, current_bigtime())
------------------------------
01010
```

用法

查找服务器上存在的当前日期。

标准

符合 ANSI SQL 的级别符合初级 — 标准。

权限

任何用户都可以执行 current_date。

另请参见

数据类型　日期和时间数据类型

命令　select、 where 子句

函数　dateadd、 datediff、 datepart、 datename、 current_bigdatetime

current_date

说明 返回当前日期。

语法 current_date()

参数 无

示例 **示例 1** 用 datename 标识当前日期：

```
1> select datename(month, current_date())
2> go

------------------------------
August
```

示例 2 用 datepart 标识当前日期：

```
1> select datepart(month, current_date())
2> go

-----------
8

(1 row affected)
```

用法 查找服务器上存在的当前日期。

标准 符合 ANSI SQL 的级别符合初级 — 标准。

权限 任何用户都可以执行 current_date。

另请参见 **数据类型** 日期和时间数据类型

命令 select、 where clause

函数 dateadd、 datename、 datepart、 getdate

current_time

说明　返回当前时间。

语法　current_time()

参数　无

示例

示例 1 查找当前时间：

```
1> select current_time()
2> go

------------------------
            12:29PM

(1 row affected)
```

示例 2 与 datename 一起使用：

```
11> select datename(minute, current_time())
2> go

------------------------------
45

(1 row affected)
```

用法　查找服务器上存在的当前时间

标准　符合 ANSI SQL 的级别符合初级 — 标准。

权限　任何用户都可以执行 current_time。

另请参见

数据类型　日期和时间数据类型

命令　select、 where clause

函数　dateadd、 datename、 datepart、 getdate

curunreservedpgs

说明

显示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 sysdatabase 的 db_id 列中。

lstart

您正在检索数据的磁盘区段的起始逻辑页码。lstart 使用 unsigned int 数据类型。

unreservedpgs

内存数据不可用时 curunreservedpgs 返回的缺省值。unreservedpgs 使用 unsigned int 数据类型。

示例

示例 1 返回数据库名称、设备名和每个设备段中的未保留页数

如果数据库已打开，curunreservedpgs 将从内存中取值。如果未使用数据库，则从 curunreservedpgs 中指定的第三个参数取值。在本示例中，该值取自 sysusages 表中的 unreservedpgs 列。

```
select
(dbid), d.name,
    curunreservedpgs(dbid, lstart, unreservedpgs)
    from sysusages u, sysdevices d
where u.vdevno=d.vdevno
and d.status & 2 = 2
```

```
                            name
 ------------------------------ -------------------------- -----------
master                          master                            1634
tempdb                          master                             423
model                           master                             423
pubs2                           master                              72
sybsystemdb                     master                             399
sybsystemprocs                  master                            6577
sybsyntax                       master                             359

(7 rows affected)
```

示例 2 显示从 sysusages.lstart 开始的 dbid 段上的可用页数：

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

用法

- curunreservedpgs 是一个系统函数，它返回磁盘区段中的可用页数。

- 如果数据库已打开，将从内存中获得由 curunreservedpgs 返回的值。如果未使用数据库，则从 curunreservedpgs 中指定的第三个参数取值。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 curunreservedpgs。

另请参见　**文档**　*《Transact-SQL 用户指南》*

函数　db_id、 lct_admin

data_pages

说明

返回指定表、索引或特定分区所用的页数。结果不包括用于内部结构的页。

此函数可替换 Adaptive Server 15.0 之前版本中的 data_pgs 和 ptn_data_pgs。

语法

data_pages(*dbid*, *object_id* [, *indid* [, *ptnid*]])

参数

dbid
是包含数据页的数据库的数据库 ID。

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。

indid
是目标索引的索引 ID。

ptnid
是目标分区的分区 ID。

示例

示例 1 返回指定数据库中对象 ID 为 31000114 的对象所使用的页数（包括所有索引）：

```
select data_pages(5, 31000114)
```

示例 2 （在集群环境中）返回数据层中的对象所使用的页数，而不管是否存在聚簇索引：

```
select data_pages(5, 31000114, 0)
```

示例 3 （在集群环境中）返回索引层中的对象为聚簇索引所使用的页数。其中不包括数据层所使用的页数：

```
select data_pages(5, 31000114, 1)
```

示例 4 返回特定分区的数据层中对象所使用的页数，在此示例中为 2323242432：

```
select data_pages(5, 31000114, 0, 2323242432)
```

用法

- 以 APL （所有页锁）表为例，如果表中存在聚簇索引，则传递值为以下数字的 *indid*：
  - 0 — 报告数据页。
  - 1 — 报告索引页。

  当 *object_id* 在当前数据库中不存在，或当目标 *indid* 或 *ptnid* 无法找到时，所有错误条件都返回值零。

- data_pages　会放弃尚未在高速缓存中的对象的描述符，而不是消耗资源。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 data_pages。

另请参见

函数　object_id、 row_count

系统过程　sp_spaceused

datachange

说明

用于测量自上次运行 update statistics 以来的数据分布更改量。具体而言，它测量给定对象、分区或列中发生的 inserts、updates 和 deletes 操作的次数，并帮助您确定调用 update statistics 是否有益于查询计划。

语法

datachange(*object_name*, *partition_name*, *column_name*)

参数

object_name

是当前数据库中的对象名称。

partition_name

是数据分区名称。此值可以为空。

column_name

是要请求 datachange 的列的名称。此值可以为空。

示例

示例 1 提供 author_ptn 中 au_id 列的百分比更改：

```
select datachange("authors", "author_ptn", "au_id")
```

示例 2 提供 au_ptn 分区上的 authors 表中的百分比更改。*column_name* 参数的空值表示将检查所有具有直方图统计信息的列并从中获取最大的 datachange 值。

```
select datachange("authors", "au_ptn", null)
```

用法

- datachange 函数需要上述所有三个参数。
- datachange 是衡量 inserts、 deletes 和 updates 的尺度，但不会分别为它们计数。 datachange 将一个 update 计作一个 delete 和一个 insert，因此每个 update 在 datachange 计数器中占用的计数为 2。
- 内置 datachange 返回作为行数的百分比的 datachange 计数，但此百分比基于剩余行数，而不是基于原始行数。例如，如果一个表包含五个行，在删除一行后， datachange 报告的值为 25 %，这是因为当前行计数为 4 且 datachange 计数器为 1.
- datachange 表示为表或分区 （如果已指定一个分区）中的总行数的百分比。百分比值可以大于 100 %，原因是更改对象的次数可以大于表中的行数，特别是在对表的删除或更新次数非常多的情况下。
- datachange 显示的值是内存中的值。此值与磁盘上的值不同，因为当运行 sp_flushstats 或刷新对象描述符时，管理程序将对磁盘上的值进行更新。
- 当为分区表上的全局索引创建直方图时，不会重新设置 datachange 值。
- datachange 会放弃尚未在高速缓存中的对象的描述符，而不是消耗资源。

在以下情况下，将重置 datachange 或将其初始化为零：

- 已添加新列，且已初始化它们的 datachange 值。
- 已添加新分区，且已初始化它们的 datachange 值。
- 已创建、删除或更新数据特定于分区的直方图。发生上述情况后，将为相应的列和分区重新设置直方图的 datachange 值。
- 将为表或分区截断数据，并重新设置数据的 datachange 值
- 作为其它命令的结果，已将表直接或间接地重新分区，并且已为所有表的分区和列重新设置 datachange 值。
- 已取消对表进行分区，且已为表的所有列重新设置 datachange 值。

datachange 有以下限制：

- datachange 统计信息不会在系统 tempdb 或用户定义的 tempdb 中的表、系统表或代理表上保留。
- datachange 更新是非事务性的。如果回退事务，datachange 值不会回退，这些值会变得不准确。
- 如果列级计数器的内存分配失败， Adaptive Server 将跟踪分区级 datachange 值，而非列级值。
- 如果 Adaptive Server 不维护列级 datachange 值，则无论何时重新设置列的 datachange 值，它都将重新设置分区级 datachange 值。

权限　任何用户都可以执行 datachange。

datalength

说明 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，或任何求值结果为单个值的列名、变量、常量表达式的组合。*expression* 可以是任何数据类型，它通常是列名。如果 *expression* 是字符常量，则必须用引号引起来。

示例 查找 publishers 表中 pub_name 列的长度：

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

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

用法

- datalength 是一个系统函数，它返回 *expression* 的长度 （以字节表示）。
- datalength 返回大对象列的非压缩长度 （即使列被压缩）。
- 对于为 Unicode 数据类型定义的列， datalength 返回存储在每行中的数据的实际字节数。例如，如果 unitext 列 ut 包含行值 U+0041U+0042U+d800dc00，则将返回以下信息：

  ```
  select datalength(ut) from unitable
  -------------
  8
  ```

- datalength 查找每一行存储的数据的实际长度。 datalength 适用于 varchar、 univarchar、 varbinary、 text 和 image 数据类型，因为这些数据类型可以存储可变长度 （并且不存储尾随空白） 。如果声明 char 或 unichar 值可为空， Adaptive Server 就会在内部将其存储为 varchar 或 univarchar。对于所有其它数据类型，datalength 将报告它们的已定义长度。
- datalength 接受 text_locator、 unitext_locator 和 image_locator LOB 数据类型。
- 任何 NULL 数据的 datalength 都将返回 NULL。

标准 符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限 任何用户都可以执行 datalength。

另请参见 **函数** char_length、 col_length

dateadd

说明

向指定日期或时间添加时间间隔。

语法

dateadd(*date_part*, *integer*, {*date* | *time* | *bigtime* | *datetime*, | *bigdatetime*})

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，请参见 *《Transact-SQL 用户指南》*。

dateadd

是一个整数表达式。

date expression

是 datetime、smalldatetime、bigdatetime、bigtime、date、time 类型的表达式，或 datetime 格式的字符串。

示例

示例 1 向 bigtime 添加一百万微秒：

```
declare @a bigtime
select @a = "14:20:00.010101"
select dateadd(us, 1000000, @a)
------------------------------
2:20:01.010101PM
```

示例 2 向 bigdatetime 添加 25 小时，并让时间增加：

```
declare @a bigdatetime
select @a = "apr 12, 0001 14:20:00 "
select dateadd(hh, 25, @a)
------------------------------
Apr 13 0001   2:20PM
```

示例 3 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：

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

示例 4 将日期增加一天：

```
declare @a date
select @a = "apr 12, 9999"
select dateadd(dd, 1, @a)
-------------------------
Apr 13 9999
```

示例 5 将时间减少五分钟：

```
select dateadd(mi, -5, convert(time, "14:20:00"))
-------------------------
```

```
2:15PM
```

示例 6 将时间增加一天，并让该时间保持不变：

```
declare @a time
select @a = "14:20:00"
select dateadd(dd, 1, @a)
--------------------------
2:20PM
```

示例 7 尽管与 datetime 值类似，对于每个 date_part 也存在限制，但可以添加较大的值，它将导致各值滚到下一个有效字段：

```
--Add 24 hours to a datetime
select dateadd(hh, 24, "4/1/1979")
--------------------------
Apr  2 1979 12:00AM

--Add 24 hours to a date
select dateadd(hh, 24, "4/1/1979")
-------------------------
Apr  2 1979
```

用法

- dateadd 是一个日期函数，它向指定日期添加时间间隔。有关日期的信息，请参见 *《Transact-SQL 用户指南》*。
- 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。如果最后一个参数是 bigtime，并且日期分量是年、月或日，则结果是原始 bigtime 参数。

  如果数据参数是 smalldatetime 值，结果也会是 smalldatetime 值。可以使用 dateadd 向 smalldatetime 中添加秒或毫秒，但只有在 dateadd 返回的结果日期改变了至少一分钟时，此条件才有意义。
- 如果给定的字符串作为参数代替按时间顺序的值，则服务器会将其解释为 datetime 值（而不考虑其外观精度）。通过设置配置参数 builtin date strings 或设置 builtin_date_strings 选项，可以更改此缺省行为。当设置这些选项时，服务器会将为按时间顺序的 builtin 指定的字符串解释为 bigdatetime。有关详细信息，请参见 *《系统管理指南》*。
- 当为此 builtin 字符串指定微秒的日期分量时，值将始终被解释为 bigdatetime。
- 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。date 数据类型用于 0001 年 1 月 1 日至 9999 年 1 月 1 日的日期。 date 必须用单引号或双引号引起来。char、nchar、varchar 或 nvarchar 用于更早的日期。Adaptive Server 可识别各种各样的日期格式。有关详细信息，请参见第 42 页上的“用户定义的数据类型”和 *《Transact-SQL 用户指南》*。

如有必要（例如将字符值与 datetime 值进行比较时），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。

表 2-4：date_part 识别的缩写

日期分量	缩写	值
Year	yy	1753 – 9999 (datetime 1900 – 2079 (smalldatetime 0001 – 9999 (date
Quarter	qq	1 – 4
Month	mm	1 – 12
Week	wk	1054
Day	dd	1 – 7
dayofyear	dy	1 – 366
Weekday	dw	1 – 7
Hour	hh	0 – 23
Minute	mi	0 – 59
Second	ss	0 – 59
millisecond	ms	0 – 999
microsecond	us	0 – 999999

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 dateadd。

另请参见

数据类型　日期和时间数据类型

命令　select、 where clause

函数　datediff、 datename、 datepart、 getdate

datediff

说明

计算两个指定的日期或时间之间的日期分量数。

语法

datediff(*datepart*, {*date, date* | *time*, *time* | *bigtime*, *bigtime* | *datetime*, *datetime* | *bigdatetime*, *bigdatetime*}])

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，请参见 *《Transact-SQL 用户指南》*。

date expression1

是 datetime、smalldatetime、bigdatetime、bigtime、date、time 类型的表达式，或 datetime 格式的字符串。

date expression2

是 datetime、smalldatetime、bigdatetime、bigtime、date、time 类型的表达式，或 datetime 格式的字符串。

示例

示例 1 返回两个 bigdatetimes 之间的微秒数：

```
declare @a bigdatetime
declare @b bigdatetime
select @a = "apr 1, 1999  00:00:00.000000"
select @b = "apr 2, 1999  00:00:00.000000"
select datediff(us, @a, @b)
--------------------------
86400000000
```

示例 2 返回毫秒返回值的溢出大小：

```
select datediff(ms, convert(bigdatetime, "4/1/1753"),
convert(bigdatetime, "4/1/9999"))
Msg 535, Level 16, State 0:
第 2 行：
Difference of two datetime fields caused overflow at
runtime.
Command has been aborted
```

示例 3 查找在 pubdate 与当前日期（用 getdate 函数获取的）之间经历的天数：

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

示例 4 查找两个时间之间的小时数：

```
declare @a time
declare @b time
select @a = "20:43:22"
```

```
select @b = "10:43:22"
select datediff(hh, @a, @b)
-----------
   -10
```

示例 5 查找两个日期之间的小时数：

```
declare @a date
declare @b date
select @a = "apr 1, 1999"
select @b = "apr 2, 1999"
select datediff(hh, @a, @b)
-----------
     24
```

示例 6 查找两个时间之间的天数：

```
declare @a time
declare @b time
select @a = "20:43:22"
select @b = "10:43:22"
select datediff(dd, @a, @b)
-----------
     0
```

示例 7 返回毫秒返回值的溢出大小：

```
select datediff(ms, convert(date, "4/1/1753"), convert(date, "4/1/9999"))
Msg 535, Level 16, State 0:
µ⁄ 2 --:
Difference of two datetime fields caused overflow at runtime.
Command has been aborted
```

用法

- datediff 带有三个参数。第一个是日期分量。第二个和第三个是按时间顺序的值。对于 date、time、datetime 和 bigdatetime，结果是带符号的整数值（以日期分量表示），它等于 date2 和 date1。

  • 如果第二个或第三个参数是日期，而日期分量是小时、分钟、秒、毫秒或微秒，日期将被视为午夜。

  • 如果第二个或第三个参数是时间，而 datepart 是年、月或日，则返回零。

  • 当结果不是 datepart 的偶数倍时，则会截断（而不是舍入）datediff 结果。

  • 对于较小的 time 单位，存在溢出值，如果超出这些限制，函数将返回溢出错误。

- 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，为 68 年， 19 天， 3:14:07 小时。
- 当结果不是日期分量的偶数倍时，始终会截断（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各月份第一天的数目。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，将看到两个日期之间 （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。
- 如果第二个或第三个参数是日期，而 datepart 是小时、分钟、秒或毫秒，日期将被视为午夜。
- 如果第二个或第三个参数是时间，而 datepart 是年、月或日，则返回 0。
- 当结果不是日期分量的偶数倍时，则会截断 （而不是舍入） datediff 结果。
- 如果给定的字符串作为参数代替按时间顺序的值，则服务器会将其解释为 datetime 值 （而不考虑其外观精度） 。通过设置配置参数 builtin date strings 或设置 builtin_date_strings 选项，可以更改此缺省行为。当设置这些选项时，服务器会将为按时间顺序的 builtin 指定的字符串解释为 bigdatetime。有关详细信息，请参见*系统管理指南*。
- 当为此 builtin 字符串指定微秒的日期分量时，值将始终被解释为 bigdatetime。
- 对于较小的 time 单位，存在溢出值，如果超出这些限制，函数将返回溢出错误：
  - 微秒：大约 3 天
  - 毫秒：大约 24 天

- 秒：大约 68 年
- 分钟：大约 4083 年
- 其它：无溢出限制

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 datediff。

另请参见

数据类型　日期和时间数据类型

命令　select、 where clause

函数　dateadd、 datename、 datepart、 getdate

datename

说明

以字符串形式返回指定 date 或 time 的指定 datepart。

语法

datename(*datepart* {*date* | *time* | *bigtime* | *datetime* | *bigdatetime*})

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，请参见 *《Transact-SQL 用户指南》*。

date_expression

datetime、smalldatetime、bigdatetime、bigtime、time 类型的表达式，或 datetime 格式的字符串。

示例

示例 1 查找 bigdatetime 的月份名称：

```
declare @a bigdatetime
select @a = "apr 12, 0001 00:00:000.010101"
select datename(mm, @a)
------------------------------
April
```

示例 2 假定当前日期是 2000 年 11 月 20 日：

```
select datename(month, getdate())

November
```

示例 3 查找日期的月份名称：

```
declare @a date
select @a = "apr 12, 0001"
select datename(mm, @a)
------------------------------
    April
```

示例 4 查找时间的秒数：

```
declare @a time
select @a = "20:43:22"
select datename(ss, @a)
------------------------------
    22
```

用法

- datename 是一个日期函数，它以字符串的形式返回 datetime 或 smalldatetime 值中指定分量的名称（例如月份 “June”）。如果结果是数字（例如 “23”日），则仍以字符串的形式返回结果。
- 将 date、time、bigdatetime、bigtime、datetime 或 smalldatetime 值作为它的第二个参数

- 如果在 datename 中使用日期分量 weekday 或 dw，它将返回一周中的某一天（星期日、星期一，等等）。
- 由于 smalldatetime 仅对于分钟是精确的，所以在 datename 中使用 smalldatetime 值时，秒和毫秒总为 0。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 datename。

另请参见

数据类型　日期和时间数据类型

命令　select、 where clause

函数　dateadd、 datename、 datepart、 getdate

datepart

说明 返回日期表达式的指定分量的整数值。

语法 datepart(*date_part* {*date* | *time* | *datetime* | *bigtime* | *bigdatetime*})

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
两位数	yy	1753 – 9999（对于 smalldatetime 则为 2079）。如果是 date，则为 0001 到 9999
quarter	qq	1 – 4
month	mm	1 – 12
week	wk	1 – 54
day	dd	1 – 31
dayofyear	dy	1 – 366
weekday	dw	1 - 7 （周日— 周六）
hour	hh	0 – 23
minute	mi	0 – 59
second	ss	0 – 59
millisecond	ms	0 – 999
microsecond	us	0 - 999999
calweekofyear	cwk	1 – 53
calyearofweek	cyr	1753 – 9999（对于 smalldatetime 则为 2079）。如果是 date，则为 0001 到 9999
caldayofweek	cdw	1 – 7

当以两位数字输入年份时 (*yy*)：

- 那些小于 50 的数字将被解释为 20*yy*。例如，`01` 为 2001，`32` 为 2032，`49` 为 2049。
- 那些等于或大于 50 的数字被解释为 19*yy*。例如，`50` 为 1950，`74` 为 1974，`99` 为 1999。

对于 datetime、smalldatetime 和 time 类型，毫秒前可以带一个冒号或句号。如果前面带冒号，数字就表示千分之多少秒。如果前面带句点，单个数字位表示十分之多少秒，两个数字位表示百分之多少秒，三个数字位表示千分之多少秒。例如，“12:30:20:1”表示 12:30 过二十又千分之一秒；“12:30:20.1”表示 12:30 过二十又十分之一秒。

微秒前面必须有小数点且表示小数秒数。

date_expression

是 datetime、smalldatetime、bigdatetime、bigtime、date、time 类型的表达式，或 datetime 格式的字符串。

示例

示例 1 查找 bigdatetime 的微秒：

```
declare @a bigdatetime
select @a = "apr 12, 0001 12:00:00.000001"
select datepart(us, @a)
-----------
000001
```

示例 2 假定当前日期是 1995 年 11 月 25 日：

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

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

示例 3 返回传统烹调书籍的出版年份：

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

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

示例 4

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

-----------
```

```
           53
```

示例 5

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

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

示例 6

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

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

示例 7 查找时间中的小时数：

```
declare @a time
select @a = "20:43:22"
select datepart(hh, @a)
-----------
    20
```

示例 8 如果使用 datename 或 datepart 请求 date 中的小时、分钟或秒部分，则返回 0（零），结果为缺省时间；如果使用 datename 或 datepart 请求 time 中的月份、天或年，则返回缺省日期 Jan 1 1990：

```
-- 查找日期中的小时数
declare @a date
select @a = "apr 12, 0001"
select datepart(hh, @a)
-----------
    0

-- 查找时间的月份
declare @a time
select @a = "20:43:22"
select datename(mm, @a)
------------------------------
January
```

如果指定空值作为 datetime 函数的参数，将返回 NULL。

用法

- 以整数的形式返回指定 date （第二个参数）的第一个参数中指定的 datepart。将 date、time、datetime、bigdatetime、bigtime 或 smalldatetime 值作为它的第二个参数。如果 datepart 是小时、分钟、秒、毫秒或微秒，则结果为 0。

- 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，对同一时间单位返回的日期可能不同。例如，如果将 Adaptive Server 配置为以美国英语作为缺省语言，以下命令将返回 1988：

  ```
  datepart(cyr, "1/1/1989")
  ```

  不过，以下命令将返回 1989：

  ```
  datepart(yy, "1/1/1989)
  ```

  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，将返回相应的编号。与星期名称对应的编号取决于 datefirst 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。有关详细信息，请参见 set 命令的 datefirst 选项。

- 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，它返回一周中每一天的序数。calweekofyear、calyearofweek 和 caldayofweek 不能用作 dateadd， datediff 和 datename 的日期分量。

- 由于 datetime 和 time 仅精确到 1/300 秒，因此当这些数据类型用于 datepart 时，毫秒会舍入到最近的 1/300 秒。

- 由于 smalldatetime 仅对于分钟是精确的，所以在 datepart 中使用 smalldatetime 值时，秒和毫秒总为 0。

- 语言设置将影响星期日期分量的值。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 datepart。

另请参见

数据类型 日期和时间数据类型

命令 select、 where clause

函数 dateadd、 datediff、 datename、 getdate

day

说明

返回指定日期的 datepart 中表示日期的整数。

语法

day(*date_expression*)

参数

date_expression

是 datetime、 smalldatetime、 date 类型的表达式，或 datetime 格式的字符串。

示例

返回整数 02：

```
day("11/02/03")
----------
02
```

用法

day(*date_expression*) 等同于 datepart(dd,*date_expression*)。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 day。

另请参见

数据类型 datetime、 smalldatetime、 date、 time

函数 datepart、 month、 year

db_attr

说明　返回指定数据库的 durability、 dml_logging、 template 设置和压缩级别。

语法　db_attr('*database_name*' | *database_ID* | NULL, 'attribute')

参数

database_name
数据库的名称。

database_ID
数据库的 ID

NULL
如果包括它， db_attr 将报告当前数据库

attribute
为以下值之一：

- help — 显示 db_attr 用法信息。
- durability — 返回给定数据库的持久性：full、 at_shutdown 或 no_recovery。
- dml_logging — 返回指定数据库的数据操作语言 (DML) 日志记录的值：full 或 minimal。
- template — 返回用于指定数据库的模板数据库的名称。如果未使用任何数据库作为模板来创建数据库，则返回 NULL。
- compression — 返回数据库的压缩级别。

示例　**示例 1** 返回 db_attr 的语法：

```
select db_attr(0, "help")
-------------------------------------------------------
Usage:db_attr('dbname' | dbid | NULL, 'attribute')
List of options in attributes table:
         0 : help
         1 : durability
         2 : dml_logging
         3 : template
         4 : compression
```

示例 2 从 sysdatabses 中选择名称、durability 设置、dml_logging 设置和所用的模板：

```
select name = convert(char(20), name),
durability = convert(char(15), db_attr(name,     "durability")),
     dml_logging = convert(char(15), db_attr(dbid,     "dml_logging")),
     template = convert(char(15), db_attr(dbid, "template"))
```

```
from sysdatabases
name              durability         dml_logging                 template
---------------   ---------------    ------------------------    --------
master             full               full                        NULL
model              full               full                        NULL
tempdb             no_recovery        full                        NULL
sybsystemdb        full               full                        NULL
sybsystemprocs     full               full                        NULL
repro              full               full                        NULL
imdb               no_recovery        full                        db1
db                 full               full                        NULL
at_shutdown_db     at_shutdown        full                        NULL
db1                full               full                        NULL
dml                at_shutdown        minimal                     NULL
```

示例 3 对不存在的 DoesNotExist 数据库运行 db_attr：

```
select db_attr("DoesNotExist", "durability")
-----------------------------------------------------
NULL
```

示例 4 对不存在的 ID 为 12345 的数据库运行 db_attr：

```
select db_attr(12345, "durability")
-----------------------------------------------------
NULL
```

示例 5 对不存在的属性运行 db_attr：

```
select db_attr(1, "Cmd Does Not Exist")
-----------------------------------------------------
NULL
```

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 db_attr。

另请参见

函数

db_id

说明　显示指定数据库的 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　返回 sybsystemprocs 的 ID 号：

```
select db_id("sybsystemprocs")

------
4
```

用法

- db_id 是一个系统函数，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。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 db_id。

另请参见　**文档**　*《Transact-SQL 用户指南》*

函数　db_name、 object_id

db_instanceid

说明

（仅限集群环境）返回指定的本地临时数据库的所有者实例 ID。如果指定的数据库是全局临时数据库或非临时数据库，则返回 NULL。

语法

db_instanceid(*database_id*)
db_instanceid(*database_name*)

参数

database_id
数据库的 ID。

database_name
数据库的名称

示例

返回 ID 为 5 的数据库的所有者实例

```
select db_instanceid(5)
```

用法

- 只允许从所有者实例访问本地临时数据库。db_instanceid 确定指定的数据库是否是本地临时数据库以及本地临时数据库的所有者实例。然后，可以连接到所有者实例并访问其本地临时数据库。
- db_instanceid 必须包括参数。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以运行 sdc_intempdbconfig。

db_name

说明　显示具有指定 ID 号的数据库的名称。

语法　db_name([*database_id*])

参数　*database_id*

是数据库 ID 的数值表达式（存储在 sysdatabases.dbid 中）。

示例　**示例 1** 返回当前数据库的名称：

```
select db_name()
```

示例 2 返回数据库 ID 4 的名称：

```
select db_name(4)

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

用法

- db_name 是一个系统函数，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。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 db_name。

另请参见　**文档**　*《Transact-SQL 用户指南》*

函数　col_name、 db_id、 object_name

db_recovery_status

说明

（仅限集群环境）返回指定数据库的恢复状态。如果未包括 *database_ID* 或 *database_name* 的值，则返回当前数据库的恢复状态。

语法

db_recovery_status([*database_ID* | *database_name*])

参数

database_ID

是要请求其恢复状态的数据库的 ID。

database_name

是要请求其恢复状态的数据库的名称。

示例

示例 1 返回当前数据库的恢复状态：

```
select db_recovery_status()
```

示例 2 返回名为 test 的数据库的恢复状态：

```
select db_recovery_status("test")
```

示例 3 返回数据库 ID 为 8 的数据库的恢复状态：

```
select db_recovery_status(8)
```

用法

返回值 0 指示数据库不在节点故障切换恢复中。返回值 1 指示数据库在节点故障切换恢复中。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 db_recovery_status。

degrees

说明　返回具有指定弧度的角的大小 （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

```
select degrees(45)

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

用法

- degrees 是一个数学函数，它将弧度转换为度。其结果与数值表达式的类型相同。

  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。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 degrees。

另请参见　**文档**　*《Transact-SQL 用户指南》*

函数　radians

derived_stat

说明

返回指定的对象和索引的派生统计信息。

语法

derived_stat("*object_name*" | *object_id*,
index_name | *index_id*,
["*partition_name*" | *partition_id*,]
"*statistic*")

参数

object_name
是您所需的对象的名称。如果未指定完全限定对象名， derived_stat 将搜索当前数据库。

object_id
是 *object_name* 的替代参数，并且是您所需对象的对象 ID。*object_id* 必须位于当前数据库中

index_name
是属于您感兴趣的指定对象的索引的名称。

index_id
是 *index_name* 的替代参数，并且是您所需的指定对象的索引 ID。

partition_name
是分区的名称，属于您所需的特定分区。 *partition_name* 是可选参数。当您使用 *partition_name* 或 *partition_id* 时， Adaptive Server 会返回目标分区而不是整个对象的统计信息。

partition_id
是 *partition_name* 的替代参数，是您所需的指定对象的分区 ID。*partition_id* 是可选参数。

"*statistic*"
要返回的派生统计信息。可用统计信息为：

- data page cluster ratio 或 dpcr — 对象 / 索引对的数据页集群比
- index page cluster ratio 或 ipcr — 对象 / 索引对的索引页集群比
- data row cluster ratio 或 drcr — 对象 / 索引对的数据行集群比
- large io efficiency 或 lgio — 对象 / 索引对的大 I/O 效率
- space utilization 或 sput — 对象 / 索引对的空间利用率

示例

示例 1 为 titles 表的 titleidind 索引选择空间利用率：

```
select derived_stat("titles", "titleidind", "space utilization")
```

示例 2 为 titles 表的索引 ID 2 选择数据页集群比。请注意，您可以使用 `"dpcr"` 或 `"data page cluster ratio"`：

```
select derived_stat("titles", 2, "dpcr")
```

示例 3 由于未指定分区 ID 或名称，将报告有关整个对象的统计信息：

```
11> select derived_stat(object_id("t1"), 2, "drcr")
2> go

---------------------------
                   0.576923
```

示例 4 报告分区 tl_928003396 的统计信息：

```
11> select derived_stat(object_id("t1"), 0, "t1_928003306", "drcr")
2> go

 ---------------------------
                   1.000000

(1 row affected)
```

示例 5 使用来自 syspartitions 的数据，为给定表的所有索引选择派生的统计信息：

```
select convert(varchar(30), name) as name, indid,
    convert(decimal(5, 3), derived_stat(id, indid, 'sput')) as 'sput',
    convert(decimal(5, 3), derived_stat(id, indid, 'dpcr')) as 'dpcr',
    convert(decimal(5, 3), derived_stat(id, indid, 'drcr')) as 'drcr',
    convert(decimal(5, 3), derived_stat(id, indid, 'lgio')) as 'lgio'
from syspartitions where id = object_id('titles')
go

 name                           indid   sput     dpcr     drcr     lgio
 ------------------------------ ------ -------- -------- -------- --------
 titleidind_2133579608               1    0.895    1.000    1.000    1.000
 titleind_2133579608                 2    0.000    1.000    0.688    1.000

(2 rows affected)
```

示例 6 为分区表的所有索引和分区选择派生的统计信息。此处，mymsgs_rr4 是循环分区表，它在创建时带有全局索引和局部索引。

```
11> select * into mymsgs_rr4 partition by roundrobin 4 lock datarows
2> from master..sysmessages
2> go

(7597 rows affected)

11> create clustered index mymsgs_rr4_clustind on mymsgs_rr4(error, severity)
2> go
1> create index mymsgs_rr4_ncind1 on mymsgs_rr4(severity)
2> go
1> create index mymsgs_rr4_ncind2 on mymsgs_rr4(langid, dlevel) local index
2> go
```

```
2> update statistics mymsgs_rr4
1>

2> select convert(varchar(10), object_name(id)) as name,
3>      (select convert(varchar(20), i.name) from sysindexes i
4>       where i.id = p.id and i.indid = p.indid),
5> convert(varchar(30), name) as ptnname, indid,
6> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'sput')) as 'sput',
7> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'dpcr')) as 'dpcr',
8> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'drcr')) as 'drcr',
9> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'lgio')) as 'lgio'
10> from syspartitions p
11> where id = object_id('mymsgs_rr4')
 name                          ptnname                      indid sput dpcr drcr lgio
------------------------------ ---------------------------- ----- ---- ---- ---- ----
mymsgs_rr4 mymsgs_rr4           mymsgs_rr4_786098810            0 0.90 1.000 1.00 1.000
mymsgs_rr4 mymsgs_rr4           mymsgs_rr4_802098867            0 0.90 1.000 1.00 1.000
mymsgs_rr4 mymsgs_rr4           mymsgs_rr4_818098924            0 0.89 1.000 1.00 1.000
mymsgs_rr4 mymsgs_rr4           mymsgs_rr4_834098981            0 0.90 1.000 1.00 1.000
mymsgs_rr4 mymsgs_rr4_clustind mymsgs_rr4_clustind_850099038 2 0.83 0.995 1.00 1.000
mymsgs_rr4 mymsgs_rr4_ncind1    mymsgs_rr4_ncind1_882099152    3 0.99 0.445 0.88 1.000
mymsgs_rr4 mymsgs_rr4_ncind2    mymsgs_rr4_ncind2_898099209    4 0.15 1.000 1.00 1.000
mymsgs_rr4 mymsgs_rr4_ncind2    mymsgs_rr4_ncind2_914099266    4 0.88 1.000 1.00 1.000
mymsgs_rr4 mymsgs_rr4_ncind2   mymsgs_rr4_ncind2_930099323   4 0.877 1.000 1.000 1.000
mymsgs_rr4 mymsgs_rr4_ncind2   mymsgs_rr4_ncind2_946099380   4 0.945 0.993 1.000 1.000
```

示例 7 为当前数据库中的全部所有页锁定表选择派生统计信息：

```
22> select convert(varchar(10), object_name(id)) as name
3>     (select convert(varchar(20), i.name) from sysindexes i
4>      where i.id = p.id and i.indid = p.indid),
5> convert(varchar(30), name) as ptnname, indid,
6> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'sput')) as 'sput',
7> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'dpcr')) as 'dpcr',
8> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'drcr')) as 'drcr',
9> convert(decimal(5, 3), derived_stat(id, indid, partitionid, 'lgio')) as 'lgio'
10> from syspartitions p
11> where lockscheme(id) = "allpages"
12>   and (select o.type from sysobjects o where o.id = p.id) = 'U'

 name                          ptnname                       indid sput dpcr drcr lgio
----------- ----------------- ----------------------------- ----- ---- ---- ---- ----
 stores     stores            stores_18096074                0 0.276 1.000 1.000 1.000
 discounts  discounts         discounts_50096188             0 0.075 1.000 1.000 1.000
 au_pix     au_pix            au_pix_82096302                0 0.000 1.000 1.000 1.000
 au_pix     tau_pix           tau_pix_82096302             255 NULL NULL NULL NULL
 blurbs     blurbs            blurbs_114096416               0 0.055 1.000 1.000 1.000
 blurbs     tblurbs           tblurbs_114096416            255 NULL NULL NULL NULL
 t1apl      t1apl             t1apl_1497053338               0 0.095 1.000 1.000 1.000
```

```
t1apl      t1apl             t1apl_1513053395             0 0.082 1.000 1.000 1.000
t1apl      t1apl             t1apl_1529053452             0 0.095 1.000 1.000 1.000
t1apl      t1apl_ncind       t1apl_ncind_1545053509       2 0.149 0.000 1.000 1.000
t1apl      t1apl_ncind_local t1apl_ncind_local_1561053566 3 0.066 0.000 1.000 1.000
t1apl      t1apl_ncind_local t1apl_ncind_local_1577053623 3 0.057 0.000 1.000 1.000
t1apl      t1apl_ncind_local t1apl_ncind_local_1593053680 3 0.066 0.000 1.000 1.000
authors    auidind           auidind_1941578924           1 0.966 0.000 1.000 1.000
authors    aunmind           aunmind_1941578924           2 0.303 0.000 1.000 1.000
publishers pubind            pubind_1973579038            1 0.059 0.000 1.000 1.000
roysched   roysched          roysched_2005579152          0 0.324 1.000 1.000 1.000
roysched   titleidind        titleidind_2005579152        2 0.777 1.000 0.941 1.000
sales      salesind          salesind_2037579266          1 0.444 0.000 1.000 1.000
salesdetai salesdetail       salesdetail_2069579380       0 0.614 1.000 1.000 1.000
salesdetai titleidind        titleidind_2069579380        2 0.518 1.000 0.752 1.000
salesdetai salesdetailind    salesdetailind_2069579380    3 0.794 1.000 0.726 1.000
titleautho taind             taind_2101579494             1 0.397 0.000 1.000 1.000
titleautho auidind           auidind_2101579494           2 0.285 0.000 1.000 1.000
titleautho titleidind        titleidind_2101579494        3 0.223 0.000 1.000 1.000
titles     titleidind        titleidind_2133579608        1 0.895 1.000 1.000 1.000
titles     titleind          titleind_2133579608          2 0.402 1.000 0.688 1.000

(27 rows affected)
```

用法

- derived_stat 返回双精度值。
- derived_stat 返回的值与 optdiag 实用程序提供的值匹配。
- 如果指定对象或索引不存在，则 derived_stat 返回 NULL。
- 在错误消息中指定无效的统计类型结果。
- 使用可选的 *partition_name* 或 *partition_id* 报告有关目标分区的请求的统计信息，否则， derived_stat 报告整个对象的统计信息。
- derived_stat 会放弃尚未在高速缓存中的对象的描述符，而不是消耗资源。
- 如果提供：
  - 四个参数 — derived_stat 将第三个参数用作分区，并返回有关第四个参数的派生统计信息。
  - 三个参数 — derived_stat 假定您不指定分区并返回由第三个参数指定的派生统计信息。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

对 derived_stat 的权限检查因您的细化权限设置而异。

细化权限已启用	在启用细化权限的情况下，您必须是表的所有者或具有 manage database 权限才能执行 derived_stat
细化权限已禁用	在禁用细化权限的情况下，您必须是表所有者或具有 sa_role 的用户才能执行 derived_stat。

另请参见

文档 *《性能和调优指南》*：

- “单个表的访问方法和查询开销”
- “统计表与用 optdiag 显示统计信息”

实用程序 optdiag

difference

说明

返回两个 soundex 值之间的差值。

语法

difference(*expr1*,*expr2*)

参数

expr1

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

expr2

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

示例

示例 1

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

---------
4
```

示例 2

```
select difference("smothers", "brothers")

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

用法

- difference 是一个字符串函数，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 *expr1* 或 *expr2* 是 NULL，则返回 NULL。
- 如果 varchar 表达式是作为一个参数提供的，并且 unichar 表达式是作为另一个参数提供的，varchar 表达式就会隐式转换为 unichar（可能会发生截断）。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 difference。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　soundex

dol_downgrade_check

说明

返回指定数据库中含有宽于 8191 字节的可变长度列的仅数据锁定 (DOL) 表的数量。如果没有宽的可变长度列，则返回 0，您可以放心执行降级。

语法

dol_downgrade_check('*database_name*', *target_version*)

参数

database_name

要检查的数据库的名称或 ID。*database_name* 可以是限定对象名（例如 mydb.dbo.mytable）。

target_version

要降级到的 Adaptive Server 的整数版本 （例如， 15.0.3 是 1503）。

示例

检查 pubs2 数据库中的 DOL 表有无宽的可变长度列，以便可以降级到 15.5 版：

```
select dol_downgrade_check('pubs2', 1550)
```

用法

- 如果目标版本是 15.7 或更高版本，则返回零 （成功），表明无需进行任何工作。
- 如果指定一个限定表，但不指示其所属的数据库，dol_downgrade_check 会检查当前数据库。

exp

说明

计算对常量求指定次幂所得的值。

语法

exp(*approx_numeric*)

参数

approx_numeric

是任何近似的数值型（float、 real 或 double precision）列名、变量或常量表达式。

示例

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

用法

- exp 是一个数学函数，它将返回指定值的幂值。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 exp。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　log、 log10、 power

floor

说明

返回小于或等于指定值的最大整数。

语法

floor(*numeric*)

参数

numeric

是任意精确数值（numeric、dec、decimal、tinyint、smallint、int 或 bigint）、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形式。

示例

示例 1

```
select floor(123)

-----------
        123
```

示例 2

```
select floor(123.45)

-------
    123
```

示例 3

```
select floor(1.2345E2)

--------------------
         123.000000
```

示例 4

```
select floor(-123.45)

-------
   -124
```

示例 5

```
select floor(-1.2345E2)

--------------------
        -124.000000
```

示例 6

```
select floor($123.45)

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

用法

- floor 是一个数学函数，它返回小于或等于指定值的最大整数。其结果与数值表达式的类型相同。

对于数字和小数表达式，其结果的精度与该表达式的精度相同，标度为零。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 floor。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　abs、 ceiling、 round、 sign

get_appcontext

说明

返回指定环境中的属性值。get_appcontext 是应用程序环境功能 (ACF) 提供的内置函数。

语法

get_appcontext ("*context_name*", "*attribute_name*")

参数

context_name

是指定应用程序环境名的行，它保存为 char(30) 数据类型。

attribute_name

是指定应用程序环境属性名的行，它保存为 char(30) 数据类型。

示例

示例 1 显示为 ATTR1 返回的 VALUE1。

```
select get_appcontext("CONTEXT1", "ATTR1")
--------------
VALUE1
```

CONTEXT2 中不存在 ATTR1：

```
select get_appcontext("CONTEXT2", "ATTR1")
```

示例 2 显示没有相应权限的用户尝试获取应用程序环境时所返回的结果。

```
select get_appcontext("CONTEXT1", "ATTR2", "VALUE1")
Select permission denied on built-in get_appcontext, database dbid
-----------
-1
```

用法

- 此函数返回 0 表示成功，返回 -1 表示失败。
- 如果应用程序环境中不存在所需的属性，则 get_appcontext 返回 NULL。
- get_appcontext 将属性保存为 char 数据类型。如果创建将属性值与其它数据类型作比较的访问规则，该规则应将 char 数据转换为相应的数据类型。
- 此函数的所有参数都是必需的。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

对 get_appcontext 的权限检查因您的细化权限设置而异。

细化权限已启用	在启用细化权限的情况下，您必须具有 get_appcontext 的 select 权限才能执行此函数。
细化权限已禁用	在禁用细化权限的情况下，您必须具有 get_appcontext 的 select 权限或是具有 sa_role 的用户才能执行此函数。

另请参见

有关 ACF 的详细信息，请参见 *《系统管理指南》*第 11 章“管理用户权限”中的“行级访问控制”。

函数　get_appcontext、 list_appcontext、 rm_appcontext、 set_appcontext

getdate

说明 返回当前系统日期和时间。

语法 getdate()

参数 无

示例

示例 1 假定当前日期是 1995 年 11 月 25 日上午 10:32：

```
select getdate()
Nov 25 1995 10:32AM
```

示例 2 假定当前日期是 11 月：

```
select datepart(month, getdate())
11
```

示例 3 假定当前日期是 11 月：

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

用法

- getdate 是一个日期函数，它返回当前的系统日期和时间。

标准 符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限 任何用户都可以执行 getdate。

另请参见

数据类型 日期和时间数据类型

文档 *《Transact-SQL 用户指南》*

函数 dateadd、 datediff、 datename、 datepart

get_internal_date

说明 返回 Adaptive Server 维护的内部时钟的当前日期和时间。

语法 get_internal_date

示例

示例 1 系统时钟与 Adaptive Server 内部时钟同步。当前系统日期：2007 年 1 月 20 日凌晨 5:04：

```
select get_internal_date()
Jan 20 2007 5:04AM
```

示例 2 系统时钟与 Adaptive Server 内部时钟不同步。当前系统日期：August 27, 2007, 1:08AM。

```
select get_internal_date()
Aug 27 2007 1:07AM
```

用法

get_internal_date 可能返回与 getdate 不同的值。 getdate 返回系统时钟的值，而 get_internal_date 返回服务器内部时钟的值。

启动时， Adaptive Server 使用操作系统时钟的当前值来初始化其内部时钟，并根据操作系统的定期更新来递增时间。

Adaptive Server 定期将内部时钟与操作系统时钟同步。两者通常最多相差一分钟。

Adaptive Server 使用内部时钟值来维护对象创建日期和事务日志记录的时间戳等。要检索此类值，可使用 get_internal_date 而非 getdate。

权限

任何用户都可以执行 get_internal_date。

另请参见

getdate

getutcdate

说明	返回协调通用时间 (UTC) 格式的日期和时间值。每次插入或选择行时都要计算 getutcdate。
语法	getutcdate()
示例	`insert t1 (c1, c2, c3) select c1, getutcdate(), getdate() from t2)`
另请参见	**函数** biginttohex、 convert

has_role

说明

返回关于是否已授予用户指定角色的信息。

语法

has_role ("*role_name*", *option*)

参数

role_name
是系统角色或用户定义的角色的名称。

option
用于限制返回的信息的范围。当前，支持的唯一一个选项是 1，用于取消审计。

示例

示例 1 创建一个过程以检查用户是否是系统管理员：

```
create procedure sa_check as
if (has_role("sa_role", 0) > 0)
begin
    print "You are a System Administrator."
    return (1)
end
```

示例 2 检查用户是否已被授予系统安全员角色：

```
select has_role("sso_role", 1)
```

示例 3 检查用户是否已被授予操作员角色：

```
select has_role("oper_role", 1)
```

用法

- has_role 的作用与 proc_role 的作用相同。从 Adaptive Server 15.0 版开始，Sybase 支持并建议使用 has_role，而不要使用 proc_role。不过，您不必将所有现在使用的 proc_role 转换为 has_role。
- has_role 是一个系统函数，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。
- 如果出现以下情形，
- has_role 将返回 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，则 has_role 返回 1。
- 如果调用用户当前具有一个活动角色，而该角色又包含指定角色，则 has_role 返回 2。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限	任何用户都可以执行 has_role。
另请参见	**命令** alter role、 create role、 drop role、 grant、 set、 revoke
	文档 *《Transact-SQL 用户指南》*
	函数 mut_excl_roles、 role_contain、 role_id、 role_name、 show_role

hash

说明

生成固定长度的散列值表达式。

语法

hash(*expression*, [*algorithm*])

参数

expression

是要散列化的值。此值可以是列名、变量、常量表达式，或任何求值结果为单个值的列名、变量、常量表达式的组合。它不能为 image、text、 unitext 或行外 Java 数据类型。 Expression 通常为列名。如果 expression 是字符常量，则必须用引号引起来。

algorithm

是用于生成散列值的算法。可以取值 md5 或 sha1、2（表示 md5 二进制）或 3（表示 sha1 二进制）的字符文字 （不是变量或列名）。如果省略此参数，则使用 md5。

算法	结果
hash(*expression*, 'md5')	32 字节的 varchar 字符串。 md5 （消息摘要算法 5）是具有 128 位散列值的 cryptographic 散列函数。
hash(*expression*)	32 字节的 varchar 字符串
hash(*expression*, 'sha1')	40 字节的 varchar 字符串 sha1 （安全散列算法）是具有 160 位散列值的 cryptographic 散列函数。
hash(*expression*, 2)	16 字节的 varbinary 值 （使用 md5 算法）
hash(*expression*, 3)	20 字节的 varbinary 值 （使用 sha1 算法）

示例

此示例演示如何实现封装。存在名为“atable”、包含列 id、sensitive_field 和 tamper seal 的表。

```
update atable set tamper_seal=hash(convert(varchar(30),
id) + sensitive_field+@salt, 'sha1')
```

用法

指定为字符文字时， *algorithm* 不区分大小写 — “md5” 、“Md5”和“MD5”等效。但如果将 *expression* 指定为字符数据类型，则该值区分大小写。“Time”、“TIME”和 “time”将产生不同的散列值。

如果 *algorithm* 是字符文字，结果将为 varchar 字符串。对于 “md5”，这是一个 32 字节的字符串，其中包含散列计算的 128 位结果的十六进制表示形式。对于 “sha1”，这是一个 40 字节的字符串，其中包含散列计算的 160 位结果的十六进制表示形式。

如果 *algorithm* 是一个整数文字，则结果为 varbinary 值。对于 2，这将是包含散列计算的 128 位结果的 16 字节值。对于 3，这将是包含散列计算的 160 位结果的 20 字节值。

注释 在插入到 varbinary 列中时，Adaptive Server 会剪裁掉尾随的空值。

构成 *expression* 的各字节将按照它们在内存中的出现顺序进入散列算法。对于许多数据类型而言，顺序非常重要。例如， 4 字节 INT 值 1 在 MSB-first (big-endian) 平台上的二进制表示为 0x00, 0x00, 0x00, 0x01，在 LSB-first (little-endian) 平台上的二进制表示则为 0x01, 0x00, 0x00, 0x00。由于不同平台上的字节流不同，因此散列值也不同。使用 hashbytes 函数可以获取与平台无关的散列值。

注释 Cryptographic 社区不再将散列算法 MD5 和 SHA1 视为是完全安全的。对于任何此类算法，都应注意在安全性很关键的上下文中使用 MD5 或 SHA1 的风险。

标准 符合 SQL92 和 SQL99

权限 任何用户都可以执行 hash。

另请参见 另请参见 hashbytes 以了解与平台无关的散列值。

hashbytes

说明　生成固定长度的散列值表达式。

语法　hashbytes(*algorithm*, *expression*[, *expression...*] [, using *options*])

参数

expression[, *expression*...]

是要散列化的值。此值可以是列名、变量、常量表达式，或结果为单个值的列名、变量、常量表达式的组合。它不能为 image、text、unitext 或行外 Java 数据类型。

algorithm

是用于生成散列值的算法。可取值“md5”、“sha”、“sha1”或“ptn”的字符文字 （不是变量或列名）。

- Md5 （消息摘要算法 5）— 具有 128 位散列值的 cryptographic 散列算法。hashbytes('md5', *expression*[,...]) 生成一个 16 字节的 varbinary 值。
- Sha-Sha1 （安全散列算法）— 具有 160 位散列值的 cryptographic 散列算法。hashbytes('sha1', *expression*[,...]) 生成一个 20 字节的 varbinary 值。
- Ptn — 具有 32 位散列值的分区散列算法。“ptn”算法忽略 *using* 子句。hashbytes(*'ptn'*, *expression*[,...]) 生成一个 4 字节的 unsigned int 值。
- using — 用于排序字节以实现平台独立性。可选的 using 子句可位于以下 option 字符串之前：
  - lsb — 所有与字节顺序相关的数据都在进行散列化之前规范化为 little-endian 字节顺序。
  - msb — 所有与字节顺序相关的数据都在进行散列化之前规范化为 big-endian 字节顺序。
  - unicode — 字符数据在进行散列化之前规范化为 unicode (UTF-16)。

  注释　UTF - 16 字符串类似于短整型数组。由于它与字节顺序相关，因此 Sybase 建议为了平台独立性，将 lsb 或 msb 与 UNICODE 结合使用。

  - unicode_lsb — unicode 和 lsb 的组合。
  - unicode_msb — unicode 和 msb 的组合。

示例　**示例 1** 封装表的各行以防篡改。此示例假定存在名为“xtable”的用户表以及 col1、 col2、 col3 和 tamper_seal。

```
update xtable set tamper_seal=hashbytes('sha1', col1,
col2, col4, @salt)
--
declare @nparts unsigned int
select @nparts= 5
select hashbytes('ptn', col1, col2, col3) % nparts from
xtable
```

示例 2 演示如何使用 col1、 col2 和 col3 来将行划分为五个分区。

```
alter table xtable partition by hash(col1, col2, col3) 5
```

用法

algorithm 参数不区分大小写； “md5”、“Md5”和 “MD5”都等效。但如果将 *expression* 指定为字符数据类型，则该值区分大小写。“Time”、“TIME”和 “time”将产生不同的散列值。

注释 在插入到 varbinary 列中时，Adaptive Server 会剪裁掉尾随的空值。

如果缺少 using 子句，则构成 *expression* 的字节将按照它们在内存中的出现顺序进入散列算法。对于许多数据类型而言，顺序非常重要。例如， 4 字节 INT 值 1 在最高位在前 (big-endian) 平台上的二进制表示为 0x00, 0x00, 0x00, 0x01，在最低位在前 (little-endian) 平台上的二进制表示则为 0x01, 0x00, 0x00, 0x00。由于不同平台上的字节流不同，因此散列值也不同。

通过 using 子句，构成 *expression* 的字节可以与平台无关的方式进入散列算法。 using 子句也可用来将字符数据转换为 Unicode，以使散列值与服务器的字符配置无关。

注释 Cryptographic 社区不再将散列算法 MD5 和 SHA1 视为是完全安全的。请注意在安全性很关键的上下文中使用 MD5 或 SHA1 的风险。

标准

符合 SQL92 和 SQL99

权限

任何用户都可以执行 hashbyte。

另请参见

另请参见 hash 以了解与平台相关的散列值。

hextobigint

说明　返回十六进制字符串的等值 bigint 值

语法　hextobigint(*hexadecimal_string*)

参数　*hexadecimal_string*

是要转换为大整数的十六进制值；必须是字符型列、变量名或括在引号中的有效十六进制字符串 （带或不带 “0x” 前缀均可）。

示例　以下示例将十六进制字符串 0x7fffffffffffffff 转换为大整数。

```
1> select hextobigint("0x7fffffffffffffff")
2> go
--------------------
9223372036854775807
```

用法

- hextobigint 是一个数据类型转换函数，用于返回与十六进制字符串等值且与平台无关的整数。
- 使用 hextobigint 函数可进行与平台无关的十六进制数据到整数的转换。 hextobigint 接受括在引号中的有效十六进制字符串 （带或不带 “0x” 前缀均可），也接受字符型列名或变量名。

  hextobigint 返回十六进制字符串的等值 bigint。该函数在任何平台上都始终返回给定十六进制字符串的相同等值 bigint。

另请参见　**函数**　biginttohex、 convert、 inttohex、 hextoint

hextoint

说明

返回与十六进制字符串等值且与平台无关的整数。

语法

hextoint(*hexadecimal_string*)

参数

hexadecimal_string

是要转换为整数的十六进制值；必须是字符型列、变量名或括在引号中的有效十六进制字符串 （带或不带 “0x”前缀均可）。

示例

返回十六进制字符串 “0x00000100”的等值整数。不论采用什么执行平台，其结果总是 256：

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

用法

- hextoint 是一个数据类型转换函数，用于返回与十六进制字符串等值且与平台无关的整数。
- 使用 hextoint 函数可进行与平台无关的十六进制数据到整数的转换。hextoint 接受括在引号中的有效十六进制字符串 （带或不带 “0x”前缀均可），也接受字符型列名或变量名。

  hextoint 返回十六进制字符串的等值整数。对于给定的十六进制字符串，在任何平台上执行该函数都始终返回同一个等值整数。
- 有关数据类型转换的详细信息，请参见 *《Transact-SQL 指南》*。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 hextoint。

另请参见

函数 biginttohex、 convert、 inttohex

host_id

说明　返回当前 Adaptive Server 客户端的客户端计算机操作系统进程 ID。

语法　host_id()

参数　无

示例　在此示例中，客户端计算机的名称是“ephemeris”，计算机“ephemeris”上用于 Adaptive Server 客户端进程的进程 ID 是 2309：

```
select host_name(), host_id()
------------------------------ ----------------------
 ephemeris                      2309
```

下面是使用 UNIX ps 命令从计算机“ephemeris”上收集的进程信息，该信息显示此示例中的客户端是 “isql”，其进程 ID 是 2309：

```
2309 pts/2    S  0:00 /work/as125/OCS-12_5/bin/isql
```

用法

- host_id 是一个系统函数，它返回客户端进程 （而不是服务器进程）的主机进程 ID。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 host_id。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　host_name

host_name

说明 显示客户端进程的当前宿主计算机名。

语法 host_name()

参数 无

示例

```
select host_name()

------------------------------
violet
```

用法

- host_name 是一个系统函数，它返回客户端进程（而不是服务器进程）的当前主计算机名。

标准 符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限 任何用户都可以执行 host_name。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 host_id

instance_id

说明

（仅限集群环境）返回命名实例的 ID；如果没有为 *name* 提供值，则返回发出该函数的实例的 ID。

语法

instance_id([*name*])

参数

name

要研究其 ID 的实例的名称。

示例

返回本地实例的 ID：

```
select instance_id()
```

用法

返回名为“myserver1”的实例的 ID：

```
select instance_id(myserver1)
```

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 instance_id。

identity_burn_max

说明

跟踪给定表的 identity burn 最大值。此函数只返回值；它不执行更新。

语法

identity_burn_max(*table_name*)

参数

table_name
是选定表的名称。

示例

```
select identity_burn_max("t1")

t1
------------
51
```

用法

identity_burn_max 跟踪给定表的 identity burn max 值。

权限

对 identity_burn_max 的权限检查因您的细化权限设置而异。

细化权限已启用	在启用细化权限的情况下，您必须是表的所有者或具有 manage database 权限才能执行 identity_burn_max。
细化权限已禁用	在禁用细化权限的情况下，您必须是数据库所有者或表所有者，或者是具有 sa_role 的用户才能执行 identity_burn_max。

index_col

说明

显示指定表或视图中带索引的列的名称，最长可为 255 个字节。

语法

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

参数

object_name

是表名或视图名。该名称可以是完全限定名（即，可包括数据库和所有者名称），并且必须带有引号。

index_id

是 *object_name* 的索引的数目。此数目与 sysindexes.indid 的值相同。

key_#

是索引中的关键字。对于聚簇索引，此值界于 1 和 sysindexes.keycnt 之间；对于非聚簇索引，此值界于 1 和 sysindexes.keycnt+1 之间。

user_id

是 *object_name* 的所有者。如果未指定 *user_id*，缺省情况下，它是调用方的用户 ID。

示例

查找表 t4 的聚簇索引中的键名：

```
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
```

用法

- index_col 是一个系统函数，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 就会返回 NULL。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 index_col。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　object_id

系统过程　sp_helpindex

index_colorder

说明 返回列的顺序。

语法 index_colorder(*object_name*, *index_id*, *key_#*[, *user_id*])

参数

object_name
是表名或视图名。该名称可以是完全限定名（即，可包括数据库和所有者名称），并且必须带有引号。

index_id
是 *object_name* 的索引的数目。此数目与 sysindexes.indid 的值相同。

key_#
是索引中的关键字。有效值是 1 和索引中关键字的数目。关键字的数目存储在 sysindexes.keycnt 中。

user_id
是 *object_name* 的所有者。如果未指定 *user_id*，缺省情况下，它是调用方的用户 ID。

示例 返回 “DESC”，因为表 sales 的索引 salesind 按降序排列：

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

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

用法

- index_colorder 是一个系统函数，对于采用升序的列，它返回“ASC”，对于采用降序的列，它返回 “DESC”。
- 如果 *object_name* 不是表名，或者，如果 *key_#* 不是有效的关键字数目， index_colorder 就会返回 NULL。

标准 符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限 任何用户都可以执行 index_colorder。

另请参见 **文档** *《Transact-SQL 用户指南》*

index_name

说明

当您提供了索引 ID、数据库 ID 和定义了其索引的对象时，返回索引的名称。

语法

index_name(*dbid*, *objid*, *indid*)

参数

dbid
是在其上定义了索引的数据库的 ID。

objid
是在其上定义了索引的表 （在指定数据库中）的 ID。

indid
是需要其名称的索引的 ID。

示例

示例 1

说明此函数的常规用法。

```
select index_name(db_id("testdb"),
    object_id("testdb...tab_apl"),1)
------------
```

示例 2 说明数据库 ID 为 NULL 并且使用当前数据库 ID 时的输出。

```
select index_name(NULL,object_id("testdb..tab_apl"),1)
-------------------------
```

示例 3 显示索引 ID 为 0 并且数据库 ID 和对象 ID 有效时的表名。

```
select index_name(db_id("testdb"),
    object_id("testdb..tab_apl"),1)
----------------------------
```

用法

- 如果在 *dbid* 参数中传递 NULL 值，则 index_name 将使用当前数据库 ID。
- 如果在 *dbid* 参数中传递 NULL 值，则 index_name 返回 NULL。
- 如果索引 ID 为 0，并且为对象 ID 和数据库 ID 传递有效输入，则 index_name 将返回对象名称。

权限

任何用户都可以执行此函数。

另请参见

db_id、 object_id

inttohex

说明

返回与指定的整数等值且与平台无关的十六进制值。

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

```
select inttohex (10)

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

用法

- inttohex 是一个数据类型转换函数，用于返回与整数等值且与平台无关的十六进制值（不带“0x”前缀）。
- 使用 inttohex 函数可进行与平台无关的整数到十六进制字符串的转换。inttohex 接受任何求值结果为整数的表达式。对于给定的表达式，在任何平台上执行该函数都始终返回同一个等值十六进制数。
- 有关数据类型转换的详细信息，请参见 *《Transact-SQL 指南》*。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 inttohex。

另请参见

函数 convert、 hextobigint、 hextoint

isdate

说明　确定输入表达式是否为有效的 datetime 值。

语法　isdate(*character_expression*)

参数　*character_expression*
是字符型变量、常量表达式或列名。

示例　**示例 1** 确定字符串 12/21/2005 是否为有效的 datetime 值：

```
select isdate('12/21/2005')
```

示例 2 确定 sales 表中的 stor_id 和 date 是否为有效的 datetime 值：

```
select isdate(stor_id), isdate(date) from sales
---- ----
0    1
```

store_id 不是有效的 datetime 值，但 date 是。

用法　如果表达式是有效的 datetime 值，则返回 1；否则返回 0。对于 NULL 输入，返回 0。

is_quiesced

说明

指示数据库是否处于 quiesce database 模式。如果数据库已停顿，则 is_quiesced 返回 1，否则返回 0。

语法

is_quiesced(*dbid*)

参数

dbid

是数据库的数据库 ID。

示例

示例 1 使用 test 数据库，该数据库的数据库 ID 为 4 并且已停顿：

```
1>select is_quiesced(4)
2> go

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

(1 row affected)
```

示例 2 运行 quiesce database 后，使用 test 数据库以挂起活动：

```
1> quiesce database tst hold test
2> go
1> select is_quiesced(4)
2> go

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

(1 row affected)
```

示例 3 在使用 quiesce database 恢复活动后，使用 test 数据库：

```
1> quiesce database tst release
2> go
1> select is_quiesced(4)
2> go

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

(1 row affected)
```

示例 4 使用无效数据库 ID 执行带 is_quiesced 的 select 语句：

```
1>select is_quiesced(-5)
2> go

 -----------
       NULL
```

```
(1 row affected)
```

用法

- is_quiesced 没有缺省值。如果执行 is_quiesced 并且不指定数据库，则会出现错误。
- 如果指定了一个不存在的数据库 ID，则 is_quiesced 返回 NULL。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 is_quiesced。

另请参见

命令 quiesce database

is_sec_service_on

说明

确定是否启用了某个特定的安全服务。如果已启用该服务则返回 1 ； 否则返回 0。

语法

is_sec_service_on(*security_service_nm*)

参数

security_service_nm
是安全服务的名称。

示例

```
select is_sec_service_on("unifiedlogin")
```

用法

- 使用 is_sec_service_on 确定在会话期间是否启用了给定的安全服务。
- 若要查找安全服务的有效名称，请执行：

  ```
  select * from syssecmechs
  ```

  结果可能会是这样：

  ```
  sec_mech_name available_service
  ------------- --------------------
   dce           unifiedlogin
   dce           mutualauth
   dce           delegation
   dce           integrity
   dce           confidentiality
   dce           detectreplay
   dce           detectseq
  ```

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

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 is_sec_service_on。

另请参见

函数 show_sec_services

is_singleusermode

说明　如果 Adaptive Server 不在单用户模式下运行，则返回 0。如果 Adaptive Server 在单用户模式下运行，则返回 1。

语法　is_singleusermode()

参数　is_singleusermode 不包括任何参数。

示例　此示例显示在单用户模式下运行的服务器：

```
select is_singleusermode()

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

权限　任何用户都可以运行 is_singleusermode

isnull

说明

当 *expression1* 求值为 NULL 时，替代 *expression2* 中指定的值。

语法

isnull(*expression1*, *expression2*)

参数

expression

是列名、变量、常量表达式，或任何求值结果为单个值的列名、变量、常量表达式的组合。它可以是任何数据类型，包括 unichar。*expression* 通常是列名。如果 *expression* 是字符常量，则必须用引号引起来。

示例

返回 titles 表中的所有行，并将 price 中的空值替换为 0：

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

用法

- isnull 是一个系统函数，当 *expression1* 求值为 NULL 时，替换 *expression2* 中指定的值。有关系统函数的常规信息，请参见 *《Transact-SQL 用户指南》*。
- 表达式的数据类型必须隐式转换，或必须使用 convert 函数进行转换。
- 如果 *expression1* 参数的数据类型为 char，而 *expression2* 为文字参数，则包含 isnull 的 select 语句得出的结果将因是否启用文字自动参数化而有所不同。为避免这种情况，请勿自动参数化 isnull() 中数据类型为 char 的文字。

  使用包含相同表达式设置的 isnull() 的存储过程也可能产生意外行为。如果发生了这种情况，则重新创建对应的自动参数化。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 isnull。

另请参见

文档 *《性能和调优系列：查询处理和抽象计划》*中的“控制文字参数化”，*《系统管理指南：第 1 卷》*，*《Transact-SQL 用户指南》*

函数 convert

isnumeric

说明

确定表达式是否为有效的 numeric 数据类型。

语法

isnumeric (*character_expression*)

参数

character_expression

是字符型变量、常量表达式或列名。

示例

示例 1 确定 authors 表的 postalcode 列中的值是否包含有效的 numeric 数据类型：

```
select isnumeric(postalcode) from authors
```

示例 2 确定值 $100.12345 是否为有效的 numeric 数据类型：

```
select isnumeric("$100.12345")
```

用法

- 如果输入表达式是有效的整数、浮点数、货币或小数类型，则返回 1；如果不是这些类型或者输入为 NULL 值，则返回 0。返回值 1 确保可将表达式转换为以下数值类型之一。
- 可以在输入中包含货币符号。

instance_name

说明

（仅限集群环境）返回提供了其 ID 的 Adaptive Server 的名称；如果没有为 *id* 提供值，则返回发出该函数的 Adaptive Server 的名称。

语法

instance_name([*id*])

参数

id

是要研究其名称的 Adaptive Server 的 ID。

示例

返回 ID 为 12 的实例的名称：

```
select instance_name(12)
```

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 instance_name。

lc_id

说明

（仅限集群环境）返回提供了其名称的逻辑集群的 ID ；如果未提供名称，则返回当前逻辑集群的 ID。

语法

lc_id(*logical_cluster_name*)

参数

logical_cluster_name
是逻辑集群的名称。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 lc_id

lc_name

说明

（仅限集群环境）返回提供了其 ID 的逻辑集群的名称；如果未提供 ID，则返回当前逻辑集群的名称。

语法

lc_name([*logical_cluster_ID*])

参数

logical_cluster_ID
是逻辑集群的 ID。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 lc_name。

lct_admin

说明

管理最后机会阈值 (LCT)。它返回 LCT 的当前值，并中止已经到达其 LCT 的事务日志中的事务。

语法

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

参数

lastchance
在指定数据库中创建 LCT。

logfull
如果指定的数据库中的 LCT 已被超过，则返回 1，否则返回 0。

reserved_for_rollbacks
确定数据库当前为回退保留的页数。

database_id
指定数据库。

reserve
获取 LCT 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 LCT 的页数。

0
返回 LCT 的当前值。在带有独立日志和数据段的数据库中，LCT 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， LCT 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”模式的事务才可以被中止。

logsegment_freepages
描述日志段的可用空间。这是可用空间总值，而不是每个磁盘的可用空间值。

process-id
是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，它将被置于日志挂起模式。

database-id
是事务日志已到达其 LCT 的数据库的 ID。如果 *process-id* 为 0，将终止指定数据库中打开的所有事务。

示例

示例 1 为 dbid 为 1 的数据库创建日志段最后机会阈值。它返回新阈值驻留的页数。如果存在以前的最后机会阈值，将替换以前的阈值：

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

示例 2 如果已超过 dbid 为 6 的数据库的最后机会阈值，则返回 1，否则返回 0：

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

示例 3 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：

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

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

示例 4 返回发出命令的数据库中事务日志的当前最后机会阈值：

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

示例 5 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：

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

示例 6 中止在 dbid 为 5 的数据库中打开的所有事务。这种格式将唤醒可能在日志段的最后机会阈值挂起的所有进程：

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

示例 7 确定 dbid 为 5 的 pubs2 数据库中为回退保留的页数：

```
select lct_admin("reserved_for_rollbacks", 5, 0)
```

示例 8 描述 dbid 为 4 的数据库的可用空间：

```
select lct_admin("logsegment_freepages", 4)
```

用法

- lct_admin 是一个系统函数，用于管理日志段的最后机会阈值。有关系统函数的常规信息，请参见 *《Transact-SQL 用户指南》*。
- 如果 lct_admin("lastchance", *dbid*) 返回了零，则日志不在此数据库的独立段上，因此不存在最后机会阈值。
- 每当创建具有独立日志段的数据库时，服务器将创建一个缺省的最后机会阈值，它缺省调用 sp_thresholdaction。即使在服务器上根本没有称为 sp_thresholdaction 的过程，亦是如此。

  当日志超出最后机会阈值时，Adaptive Server 将挂起活动，然后试图调用 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。

- 若要终止已到达其 LCT 的事务日志中最早打开的事务，请输入启动事务的进程的 ID。
- 若要终止已到达其 LCT 的事务日志中所有打开的事务，请输入 0 作为 *process-id*，并在 *database-id* 参数中指定数据库 ID。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　对 lct_admin 的权限检查因您的细化权限设置而异。

细化权限已启用	在启用细化权限的情况下，您必须具有 manage database 权限才能执行 lct_admin abort。任何用户都可以执行其它 lct_admin 选项。
细化权限已禁用	在禁用细化权限的情况下，您必须是具有 sa_role 的用户才能执行 lct_admin abort。任何用户都可以执行其它 lct_admin 选项。

另请参见

文档　*《系统管理指南》*。

命令　dump transaction

函数　curunreservedpgs

系统过程　sp_thresholdaction

left

说明

返回字符串最左侧指定数目的字符。

语法

left(*character_expression*, *integer_expression*)

参数

character_expression
是从中选择左侧字符的字符串。

integer_expression
是指定返回的字符数的正整数。如果 *integer_expression* 为负，则返回错误。

示例

示例 1 返回每本书的书名中最左边的五个字符。

```
use pubs
select left(title, 5) from titles
order by title_id

-----
The B
Cooki
You C
.....
Sushi

(18 row(s) affected)
```

示例 2 返回字符串 “abcdef”最左边的两个字符：

```
select left("abcdef", 2)
---------
ab
(1 row(s) affected)
```

用法

- *character_expression* 可以是能够隐式转换为 varchar 或 nvarchar 的任何数据类型 （text 或 image 除外）。 *character_expression* 可以常量、变量或列名。您可以使用 convert 显式转换 character_expression。
- left 等同于 substring(*character_expression*, *1*, *integer_expression*)。有关此函数的详细信息，请参见 第 283 页的 substring。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 left。

另请参见

数据类型 varchar、 nvarchar

函数 len、 str_replace、 substring

len

说明　返回指定字符串表达式（不包括尾随空白）的字符数（而不是字节数）。

语法　len(*string_expression*)

参数　*string_expression*
是要求值的字符串表达式。

示例　返回字符

```
select len(notes) from titles
where title_id = "PC9999"
-----------
39
```

用法　此函数等同于 char_length(*string_expression*)。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 len。

另请参见　**数据类型**　char、 nchar、 varchar、 nvarchar

函数　char_length、 left、 str_replace

license_enabled

说明

如果启用了某功能的许可证，则返回 1；如果未启用许可证，则返回 0；如果指定的许可证名称无效，则返回 NULL。

语法

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

参数

ase_server
指定 Adaptive Server 的许可证。

ase_ha
指定 Adaptive Server 高可用性功能的许可证。

ase_dtm
指定 Adaptive Server 分布式事务管理功能的许可证。

ase_java
指定 Adaptive Server 中的 Java 的功能的许可证。

ase_asm
指定 Adaptive Server 高级安全性机制的许可证。

示例

表示已启用 Adaptive Server 分布式事务管理功能的许可证：

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

用法

- 有关安装 Adaptive Server 功能许可证密钥的详细信息，请参见安装指南。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 license_enabled。

另请参见

文档 您的平台的安装指南

系统过程 sp_configure

list_appcontext

说明

列出当前会话中所有环境的全部属性。 list_appcontext 由 ACF 提供。

语法

list_appcontext(["*context_name*"])

参数

context_name

是一个可选参数，它为会话中的所有应用程序环境属性命名。

示例

示例 1 显示具有相应权限的用户尝试列出应用程序环境时所返回的结果：

```
select list_appcontext ([context_name])

Context Name:(CONTEXT1)
Attribute Name:(ATTR1) Value:(VALUE2)
Context Name:(CONTEXT2)
Attribute Name:(ATTR1) Value:(VALUE1)
```

示例 2 显示没有相应权限的用户尝试列出应用程序环境时所返回的结果：

```
select list_appcontext()

Select permission denied on built-in list_appcontext,
database DBID
--------
-1
```

用法

- 此函数返回 0 表示成功。
- 由于内置函数不返回多个结果集，因此客户端应用程序将收到以消息形式返回的 list_appcontext。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展

权限

对 list_appcontext 的权限检查因您的细化权限设置而异。

细化权限已启用	在启用细化权限的情况下，您必须具有 list_appcontext 的 select 权限才能执行此函数。
细化权限已禁用	在禁用细化权限的情况下，您必须具有 list_appcontext 的 select 权限或是具有 sa_role 的用户才能执行此函数。

另请参见

有关 ACF 的详细信息，请参见 *《系统管理指南》* 第 11 章 “管理用户权限”中的 “行级访问控制”。

函数 get_appcontext、 list_appcontext、 rm_appcontext、 set_appcontext

locator_literal

说明

将二进制值标识为定位符文字。

语法

locator_literal(*locator_type*, *literal_locator*)

参数

locator_type

定位符的类型。 text_locator、 image_locator 和 unitext_locator 之一。

literal_locator

是 LOB 定位符的实际二进制值。

示例

以下示例在 my_table 的 imagecol 列中插入一个在内存中存储并由其定位符标识的 image LOB。使用 locator_literal 函数可确保 Adaptive Server 正确地将二进制值解释为 LOB 定位符。

```
insert my_table (imagecol) values
    (locator_literal(image_locator,
0x9067ef450100000000100000004010040080000000)
```

用法

使用 locator_literal 可确保 Adaptive Server 正确地识别实际定位符值，而不会将其误解为 image 或其它二进制值。

权限

任何用户都可以执行 locator_literal。

另请参见

命令 deallocate locator、 truncate lob

Transact-SQL 函数 locator_valid、 return_lob、 create_locator

locator_valid

说明　确定 LOB 定位符是否有效。

语法　locator_valid (*locator_descriptor*)

参数　*locator_descriptor*

LOB 定位符的有效表示：宿主变量、局部变量或定位符的实际二进制值。

示例　验证定位符值 0x9067ef450100000000100000004010040080000000：

```
locator_valid (0x9067ef450100000000100000004010040080000000)
-------------------
                  1
```

用法

- 如果指定的定位符有效，locator_valid 将返回 1，否则返回 0（零）。
- 如果 deallocate lob 命令使其失效，或者在事务终止时，定位符会变得无效。

权限　任何用户都可以执行 locator_valid。

另请参见　**命令**　deallocate locator、 truncate lob

Transact-SQL 函数　locator_literal、 return_lob、 create_locator

lockscheme

说明

以字符串的形式返回指定对象的锁定方案。

语法

lockscheme(*object_name*)

lockscheme(*object_id* [, *db_id*])

参数

object_name

是锁定方案返回的对象的名称。 *object_name* 也可以是完全限定名。

db_id

object_id 指定的数据库的 ID。

object_id

是锁定方案返回的对象的 ID。

示例

示例 1 为当前数据库中的 titles 表选择锁定方案：

```
select lockscheme("titles")
```

示例 2 为数据库 ID 4 （pubs2 数据库）中的 *object_id* 224000798 （本例中为 titles 表）选择锁定方案：

```
select lockscheme(224000798, 4)
```

示例 3 返回 titles 表的锁定方案 （本示例中的 *object_name* 为完全限定名）：

```
select lockscheme(tempdb.ownerjoe.titles)
```

用法

- lockscheme 返回 varchar(11) 并允许为 NULL。
- lockscheme 缺省为当前数据库，前提是：
  - 未提供全限定 *object_name*。
  - 未提供 *db_id*。
  - 为 *db_id* 提供 Null。
- 如果指定对象不是表， lockscheme 将返回字符串 “not a table”。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 lockscheme。

log

说明　计算指定数字的自然对数。

语法　log(*approx_numeric*)

参数　*approx_numeric*
是任何近似的数值型（float、 real 或 double precision）列名、变量或常量表达式。

示例

```
select log(20)

--------------------
              2.995732
```

用法

- log 是一个数学函数，它返回指定值的自然对数。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 log。

另请参见　**文档**　*《Transact-SQL 用户指南》*

函数　log10、 power

log10

说明 计算指定数字的以 10 为底数的对数。

语法 log10(*approx_numeric*)

参数 *approx_numeric*

是任何近似的数值型（float、 real 或 double precision）列名、变量或常量表达式。

示例

```
select log10(20)

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

用法

- log10 是一个数学函数，它返回指定值的以 10 为底数的对数。

标准 符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限 任何用户都可以执行 log10。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 log、 power

lower

说明

将大写字符转换为小写。

语法

lower(*char_expr* | *uchar_expr*)

参数

char_expr

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

uchar_expr

类型为 unichar 或 univarchar 的字符型列名、变量或常量表达式。

示例

```
select lower(city) from publishers

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

用法

- lower 是一个字符串函数，它将大写转换为小写，并返回字符值。
- lower 是 upper 的倒数。
- 如果 *char_expr* 或 *uchar_expr* 是 NULL，则返回 NULL。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 lower。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　upper

lprofile_id

说明

返回指定登录配置文件名称的登录配置文件 ID，或者与当前登录或指定登录名关联的登录配置文件的登录配置文件 ID。

语法

lprofile_id(*name*)

参数

name

（可选）登录配置文件名或登录名。

如果您指定登录配置文件名， lprofile_id 会返回对应的登录配置文件 ID。如果您指定登录名， lprofile_id 会返回关联的 （如果有）登录配置文件 ID。

如果您不指定 *name*， lprofile_id 会返回当前登录名的登录配置文件 ID。

权限

对 lprofile_id 的权限检查因您的细化权限设置而异。

细化权限已启用	在启用细化权限的情况下，任何用户都可以执行 lprofile_id 返回自己配置文件的 ID。您必须具有 manage any login profile 权限才能执行 lprofile_id 和检索其他用户的配置文件 ID。
细化权限已禁用	在禁用细化权限的情况下，任何用户都可以执行 lprofile_id 返回自己配置文件的 ID。您必须是具有 sso_role 的用户才能执行 lprofile_id 和检索其他用户的配置文件 ID。

lprofile_name

说明

返回指定登录配置文件 ID 的登录配置文件名称，或者与当前登录或指定登录 suid 关联的登录配置文件的登录配置文件名称。

语法

lprofile_id(*ID*)

参数

ID

（可选）登录配置文件 ID 或登录 suid。

如果您指定登录配置文件 ID， lprofile_name 会返回对应的登录配置文件名。如果您指定登录 suid， lprofile_name 会返回关联的 （如果有）登录配置文件名。

如果您不指定 *ID*， lprofile_id 会返回当前登录名的登录配置文件名。

权限

对 lprofile_name 的权限检查因您的细化权限设置而异。

细化权限已启用	在启用细化权限的情况下，任何用户都可以执行 lprofile_name 返回自己配置文件的配置文件名。您必须具有 manage any login profile 权限才能执行 lprofile_name 和检索其他用户的配置文件名。
细化权限已禁用	在禁用细化权限的情况下，任何用户都可以执行 lprofile_name 返回自己配置文件的配置文件名。您必须具有 sso_role 才能执行 lprofile_name 和检索其他用户的配置文件名。

ltrim

说明 删去指定表达式的前导空白。

语法 ltrim(*char_expr* | *uchar_expr*)

参数

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

uchar_expr
类型为 unichar 或 univarchar 的字符型列名、变量或常量表达式。

示例

```
select ltrim("    123")

-------
123
```

用法

- ltrim 是一个字符串函数，它删除字符表达式中的前导空白。只删除当前字符集中等于空格字符的值。
- 如果 *char_expr* 或 *uchar_expr* 是 NULL，则返回 NULL。
- 对于 Unicode 表达式，返回与指定表达式等值的小写 Unicode。表达式中无等值小写的字符保持不变。

标准 符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限 任何用户都可以执行 ltrim。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 rtrim

max

说明　返回表达式的最大值。

语法　max(*expression*)

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。

示例　**示例 1** 将表 salesdetail 中 discount 列中的最大值作为一个新列返回：

```
select max(discount) from salesdetail

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

示例 2 将表 salesdetail 中 discount 列中的最大值作为一个新行返回：

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

用法

- max 是一个集合函数，它在列或表达式中查找最大值。有关集合函数的常规信息，请参见 *《Transact-SQL 用户指南》*。
- max 可用于精确和近似数值、字符以及 datetime 列；不可用于 bit 列。对于字符列，max 查找归类序列中的最大值。max 将忽略空值。max 将 char 数据类型隐式转换为 varchar，将 unichar 数据类型转换为 univarchar，同时去除所有尾随空白。
- unichar 数据按缺省的 Unicode 排序顺序归类排列。
- max 在 varbinary 数据中保留尾随零。
- max 从对 binary 数据的查询中返回 varbinary 数据类型。
- 当集合列上有索引时， Adaptive Server 直接到索引的末尾查找最后一行以获得 max 值，除非：
  - *expression* 不是列。
  - 该列不是索引的第一列。
  - 查询中还有另外一个集合。
  - 存在 group by 或 where 子句。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 max。

另请参见　**命令**　compute clause、 group by and having clauses、 select、 where clause

函数　avg、 min

migrate_instance_id

说明	如果在迁移任务环境中发出 migrate_instance_id 命令，会返回作为调用方法迁移源的实例的实例 ID。如果在非迁移任务环境中发出 migrate_instance_id 命令，会返回当前实例的 ID。
语法	migrate_instance_id()
用法	可从登录触发器发出 migrate_instance_id 命令，以确定在迁移任务时应执行触发器中的哪些语句。
标准	符合 ANSI SQL 的级别 Transact-SQL 扩展。
权限	您必须是系统管理员才能发出 migrate_instance_id 命令。

min

说明　返回列中的最小值。

语法　min(*expression*)

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。使用集合时，表达式通常是列名。有关详细信息，请参见第 339 页上的 “表达式”。

示例

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

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

用法

- min 是一个集合函数，它查找列中的最小值。
- min 可用于数值、字符、time 和 datetime 列；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，将 unichar 数据类型转换为 univarchar，同时去除所有尾随空白。min 忽略空值。distinct 不可用，因为它与 min 一起使用时没有意义。
- min 在 varbinary 数据中保留尾随零。
- min 从对 binary 数据的查询中返回 varbinary 数据类型。
- unichar 数据按缺省的 Unicode 排序顺序归类排列。
- 当集合列上有索引时，Adaptive Server 直接到第一个限定行查找 min 值，除非：
  - *expression* 不是列。
  - 该列不是索引的第一列。
  - 查询中还有另外一个集合。
  - 存在一个 group by 子句。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 min。

另请参见

命令　compute clause、 group by and having clauses、 select、 where clause

文档　*《Transact-SQL 用户指南》*

函数　avg、 max

month

说明

返回指定日期的 datepart 中表示月份的整数。

语法

month(*date_expression*)

参数

date_expression

是 datetime、 smalldatetime、 date 类型的表达式，或 datetime 格式的字符串。

示例

返回整数 11：

```
day("11/02/03")
----------
11
```

用法

month(*date_expression*) 等同于 datepart(*mm*, *date_expression*)。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 month。

另请参见

数据类型 datetime、 smalldatetime、 date

函数 datepart、 day、 year

mut_excl_roles

说明

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

是互斥关系中一个用户定义的角色。

role2

是在互斥关系中由用户定义的另一个角色。

level

是指定角色之间相互排斥的级别 （membership 或 activation）。

示例

演示 admin 和 supervisor 角色是互斥的：

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

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

用法

- mut_excl_roles 是一个系统函数，它返回有关两个角色之间互斥性的信息。如果系统安全员将 role1 定义为与 role2 互斥的角色，或直接由 role2 所包含的角色，则 mut_excl_roles 返回 1。如果角色之间不是互斥的，则 mut_excl_roles 返回 0。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展

权限

任何用户都可以执行 mut_excl_roles。

另请参见

命令　alter role、 create role、 drop role、 grant、 set、 revoke

文档　*《Transact-SQL 用户指南》*

函数　proc_role、 role_contain、 role_id、 role_name

系统过程　sp_activeroles、 sp_displayroles、 sp_role

newid

说明

根据提供的参数生成两种不同格式的、人工可读的全局唯一 ID (GUID)。人工可读格式的 GUID 值长度为 32 个字节 （无短划线）或 36 个字节（带短划线）。

语法

newid([*optionflag*])

参数

option flag

- 0 或没有值 — 生成的 GUID 是人工可读的 (varchar)，但不含短划线。此参数是缺省值，它用于将值转换为 varbinary。
- -1 — 生成的 GUID 是人工可读的 (varchar)，并且包含短划线。
- -0x0 — 将 GUID 作为 varbinary 返回。
- newid 的任何其它值都返回 NULL。

示例

示例 1 创建 varchar 列长度为 32 个字节的表，然后在 insert 语句中使用不带参数的 newid：

```
create table t (UUID varchar(32))
go
insert into t values (newid())
insert into t values (newid())
go
select * from t

UUID
--------------------------------
f81d4fae7dec11d0a76500a0c91e6bf6
7cd5b7769df75cefe040800208254639
```

示例 2 生成包含短划线的 GUID：

```
select newid(1)
-------------------------------------
b59462af-a55b-469d-a79f-1d6c3c1e19e3
```

示例 3 创建将不带短划线的 GUID 格式转换为 varbinary(16) 列的缺省值：

```
create table t (UUID_VC varchar(32), UUID
varbinary(16))
go
create default default_guid
as
strtobin(newid())
go
sp_bindefault default_guid, "t.UUID"
go
insert t (UUID_VC) values (newid())
```

```
go
```

示例 4 返回从查询返回的每一行的 varbinary 类型的新 GUID：

```
select newid(0x0) from sysobjects
```

示例 5 将 newid 用于 varbinary 数据类型：

```
sp_addtype binguid, "varbinary(16)"
create default binguid_dflt
as
newid(0x0)
sp_bindefault "binguid_dflt","binguid"
create table T1 (empname char(60), empid int, emp_guid
binguid)
insert T1 (empname, empid) values ("John Doe", 1)
insert T1 (empname, empid( values ("Jane Doe", 2)
```

用法

- newid 根据传递给 newid 的参数为全局唯一 ID (GUID) 生成两个新值。缺省参数生成不带短划线的 GUID。缺省情况下， newid 返回每一过滤行的新值。
- 与其它函数类似， newid 可以用于缺省值、规则和触发器。
- 确保 varchar 列的长度至少为 32 个字节（不带短划线的 GUID 格式）或 36 个字节 （带短划线的 GUID 格式）。如果没有用这些必需的最小长度声明列长度，则会截断列长度。截断将增大出现重复值的可能性。
- 参数 0 等同于缺省值。
- 可以在 strtobin 函数中使用不带短划线的 GUID 格式来将 GUID 值转换为 16 字节的二进制数据。不过，在 strtobin 中使用带短划线的 GUID 格式将导致空值。
- 因为 GUID 是全局唯一的，所以可以在域之间传输 GUID，而不会产生重复值。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 newid。

next_identity

说明

检索下一个 insert 可用的下一个标识值。

语法

next_identity(*table_name*)

参数

table_name

标识正在使用的表。

示例

将 c2 的值更新为 10。下一个可用值是 11。

```
select next_identity ("t1")
t1
--------
11
```

用法

- next_identity 返回要由该任务插入的下一值。某些情况下，当多个用户在同一个表中插入值时，如果其他用户执行中间插入，那么报告要插入的下一个实际值将与真正插入的值不同。
- next_identity 返回 varchar 字符以支持 identity 列的所有精度。如果该表是代理表、非用户表或没有 identity 属性的表，则返回 NULL。

权限

对 next_identity 的权限检查因您的细化权限设置而异。

细化权限已启用	在启用细化权限的情况下，您必须是表所有者，或者是具有表 identity 列 select 权限的用户，或者具有 manage database 权限才能执行 next_identity。
细化权限已禁用	在禁用细化权限的情况下，您必须是数据库所有者或表所有者，或者是具有 sa_role 的用户，或者具有表 identity 列的 select 权限才能执行 next_identity。

nullif

说明

允许为条件值编写 SQL 表达式。所有可以使用值表达式的地方都可以使用 nullif 表达式；可替代 case 表达式。

语法

nullif(*expression*, *expression*)

参数

nullif

比较两个表达式的值。如果第一个表达式等于第二个表达式，则 nullif 返回 NULL。如果第一个表达式不等于第二个表达式，则 nullif 返回第一个表达式。

expression

可以是列名、常量、函数、子查询或者任何由算术运算符或逐位运算符连接起来的列名、常量和函数的组合。有关表达式的详细信息，请参见第 339 页上的 “表达式”。

示例

从 titles 表中选择 titles 和 type。如果书籍类型为 UNDECIDED，nullif 将返回空值：

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

或者，您也可以编写：

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

用法

- nullif 表达式替代 case 表达式。
- nullif 表达式允许以简单的比较（而不是使用 when...then 结构）来表达搜索条件，从而简化了标准 SQL 表达式。
- SQL 中可以使用表达式的地方都可以使用 nullif 表达式。
- case 表达式的结果中至少必须有一个返回非空值。例如，下列语句会导致错误消息：

```
select price, coalesce (NULL, NULL, NULL)
from titles
All result expressions in a CASE expression must not be NULL.
```

- 如果查询生成了多种数据类型，那么数据类型层次就决定了 case 表达式结果的数据类型，如第 6 页上的“混合模式表达式的数据类型”中所述。如果指定了两种 Adaptive Server 不能隐式转换的数据类型（例如， char 和 int），查询将会失败。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 nullif。

另请参见

命令 case、 coalesce、 select、 if...else、 where clause

object_attr

说明

根据会话、表和数据库范围的设置来报告表的当前记录模式。

语法

object_attr(*table_name*, *string*)

参数

table_name

表名。

string

已经过查询的表属性的名称。受支持的字符串值包括：

- dml_logging — 根据显式设置的表或数据库的 DML 记录级别，返回请求的有效对象的 DML 记录级别。
- dml_logging for session — 根据用户运行的 object_attr、表模式和有关多语句事务的规则等，返回当前会话的 DML 记录级别。此参数的返回值可能因用户而异，并且对于同一用户的语句或事务也可能不同。
- compression — 返回所请求的对象的压缩类型。
- help — 输出受支持的字符串参数的列表。

示例

示例 1 要确定可以查询哪些属性，该用户可以运行：

```
select object_attr('sysobjects', 'help')
Usage:object_attr('tabname', 'attribute')

List of options in attributes table:
     0 : help
     1 : dml_logging
     2 : dml_logging for session
     3 : compression
```

dml_logging 报告对象的静态定义的 dml_logging 级别， dml_logging for session 根据特定于数据库的会话设置，报告为对象选择的运行期记录级别。

示例 2 持久性设置为 full 的表的缺省记录模式：

```
select object_attr("pubs2..authors",
         "dml_logging")

Returns:FULL
```

示例 3 如果会话对所有表禁用了记录，则为此用户拥有的表返回的记录模式是 minimal。

```
select object_attr("pubs2..authors",
         "dml_logging")
```

```
Returns:FULL

SET DML_LOGGING MINIMAL
go

select object_attr("pubs2..authors",
        "dml_logging for session")

Returns:MINIMAL
```

示例 4 如果表已更改为显式选择最少记录，则 object_attr 返回值 minimal，即使会话和数据库范围的记录为 FULL 也是如此。

```
create database testdb WITH DML_LOGGING = FULL
go

create table non_logged_table (...)
WITH DML_LOGGING = MINIMAL
go

select object_attr("non_logged_table",
         "dml_logging")
Returns:MINIMAL
```

示例 5 将表的记录从 full 更改为 minimal。在显式创建具有 full 记录的表时，如果您是表所有者或具有 sa_role 的用户，则可以在会话期间将记录重置为 minimal：

1 创建具有 minimal 记录的 testdb 数据库：

```
create database testdb
with dml_logging = minimal
```

2 创建 dml_logging 设置为 full 的表：

```
create table logged_table(...)
with dml_logging = full
```

3 将会话的记录重置为 minimal：

```
set dml_logging minimal
```

4 表的记录为 minimal：

```
select object_attr("logged_table",
        "dml_logging for session")
--------------------
minimal
```

示例 6 如果创建表时没有指定记录模式，则更改会话的记录模式也会更改表的记录模式：

- 创建表 normal_table：

```
create table normal_table
```

- 检查会话记录：

```
select object_attr("normal_table", "dml_logging")
-------------------------
FULL
```

- 将会话记录设置为 minimal：

```
set dml_logging minimal
```

- 将表的记录设置为 minimal：

```
select object_attr("normal_table",
         "dml_logging for session")
-----------------------------
minmimal
```

示例 7 object_attr 返回的记录模式取决于对其运行此函数的表。在此示例中，用户 joe 运行脚本，但 Adaptive Server 返回的记录模式发生了更改。表 joe.own_table 和 mary.other_table 使用 full 记录模式：

```
select object_attr("own_table","dml_logging")
--------------
          FULL
```

当 joe 对 mary.other_table 运行 object_attr 时，此表也会设置为 full：

```
select object_attr("mary.other_table", "dml_logging")
--------------
          FULL
```

如果 joe 将 dml_logging 更改为 minimal，则只会影响他拥有的表的记录模式：

```
set dml_logging minimal
select object_attr("own_table", "dml_logging for
session")
--------------
       MINIMAL
```

其它用户拥有的表将继续以缺省记录模式运行：

```
Select object_attr("mary.other_table", "dml_logging for
session")
--------------
```

```
FULL
```

示例 8 识别记录新 show_exec_info 的运行期选择，并在 SQL 批处理中使用：

1 启用 set showplan：

```
set showplan on
```

2 启用 set 命令：

```
set show_exec_info on
```

3 将 dml_logging 设置为 minimal 并使用 object_attr 检查日志记录：

```
set dml_logging minimal
select object_attr("logged_table", "dml_logging for session")
```

4 从表中删除行：

```
delete logged_table
```

Adaptive Server 使用 show_exec_info 参数在运行期报告表的记录模式。

用法

- 返回类型为 varchar，根据要查询的属性相应地返回属性值（例如 on 或 off）。
- 如果在同一批处理中用于更改表的记录模式的 DML 执行前有 set 语句，则 showplan 输出扩展报告的记录模式在运行期可能会受影响
- 对于未知属性，返回值是值 NULL （不是字符串 “NULL”）。
- help 是一个特殊类型的字符串参数，它将 object_attr 的所有当前受支持的属性输出到会话的输出。这样，可以快速识别 object_attr 支持哪些属性。

object_id

说明　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。该名称可以是完全限定名（即，可包括数据库和所有者名称），并且需要为 *object_name* 加上引号。

示例

示例 1

```
select object_id("titles")

-----------
  208003772
```

示例 2

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

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

用法

- object_id 是一个系统函数，用于返回对象的 ID。对象 ID 存储在 sysobjects 的 id 列中。
- object_id 会放弃尚未在高速缓存中的对象的描述符，而不是消耗资源。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 object_id。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　col_name、 db_id、 object_name

系统过程　sp_help

object_name

说明

返回具有指定对象 ID 的对象的名称；最长可为 255 个字节。

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

示例 1

```
select object_name(208003772)

------------------------------
titles
```

示例 2

```
select object_name(1, 1)

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

用法

object_name 是一个系统函数，它返回对象的名称。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 object_name。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 col_name、 db_id、 object_id

系统过程 sp_help

object_owner_id

说明　返回对象的所有者 ID。

语法　object_owner_id(*object_id*[, *database_id*])

参数

object_id
是要研究的对象的 ID。

database_id
是对象所在的数据库的 ID。

示例　选择 ID 为 1 的数据库（master 数据库）中 ID 为 1 的对象的所有者 ID：

```
select object_owner_id(1,1)
```

权限　任何用户都可以执行 object_owner_id。

pagesize

说明

以字节为单位返回指定对象的页大小。

语法

pagesize(*object_name*[,])

pagesize(*object_id*[,*db_id*[, *index_id*]])

参数

object_name
是此函数返回的页大小的对象的名称。

index_name
表示要返回的页大小的索引的名称。

object_id
是此函数返回的页大小的对象 ID。

db_id
是对象的数据库 ID。

index_id
是要返回的对象的索引 ID。

示例

示例 1 为当前数据库中的 title_id 索引选择页大小。

```
select pagesize("title", "title_id")
```

示例 2 返回 *db_id* 为 2 的数据库（前一个示例缺省为当前数据库）中 *object_id* 为 1234 的对象的数据层的页大小：

```
select pagesize(1234,2, null)
select pagesize(1234.2)
select pagesize(1234)
```

示例 3 全部缺省为当前数据库：

```
select pagesize(1234, null, 2)
select pagesize(1234)
```

示例 4 为 pubs2 数据库 (db_id 为 4) 中的 titles 表 (object_id 为 224000798) 选择页大小：

```
select pagesize(224000798, 4)
```

示例 5 返回驻留在当前数据库中的非聚簇索引的页表 mytable 的页大小：

```
pagesize(object_id('mytable'), NULL, 2)
```

示例 6 返回当前数据库中对象 titles_clustindex 的页大小：

```
select pagesize("titles", "titles_clustindex")
```

用法

- 如果未提供索引名称或 *index_id*（例如 `select pagesize("t1")`），或者使用“null”一词作为参数（例如 `select pagesize("t1", null)`），则 pagesize 缺省为数据层。
- 如果指定的对象不是需要用物理数据存储来存储页的对象（例如，如果提供视图的名称），pagesize 将返回 0。
- 如果指定的对象不存在，pagesize 将返回 NULL。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 pagesize。

partition_id

说明

返回指定数据或索引分区名称的分区 ID。

语法

partition_id(*table_name*, *partition_name*[,*index_name*])

参数

table_name
是表的名称。

partition_name
是表分区或索引分区的分区名称。

index_name
是所需的索引的名称。

示例

示例 1 返回与分区名称 testtable_ptn1 和索引 ID 0（基表）相对应的分区 ID。 testtable 必须存在于当前数据库中：

```
select partition_id("testtable", "testtable_ptn1")
```

示例 2 返回与索引名称 clust_index1 的分区名称 testtable_clust_ptn1 相对应的分区 ID。 testtable 必须存在于当前数据库中：

```
select partition_id("testtable", "testtable_clust_ptn1", "clust_index1")
```

示例 3 除了用户所在的数据库不需要与目标表所在的数据库相同之外，此示例与上例相同：

```
select partition_id("mydb.dbo.testtable", "testtable_clust_ptn1",
    "clust_index1")
```

用法

必须用引号将 *table_name*、 *partition_name* 和 *index_name* 引起来。

另请参见

函数 data_pages、object_id、partition_name、reserved_pages、row_count、used_pages

partition_name

说明

返回新分区的显式名称，partition_name 返回指定数据或索引分区 ID 的分区名称。

语法

partition_name(*indid*, *ptnid*[, *dbid*])

参数

indid

是目标分区的索引 ID。

ptnid

是目标分区的 ID。

dbid

是目标分区的数据库 ID。如果不指定此参数，则认为目标分区位于当前数据库中。

示例

示例 1 返回属于基表 （索引 ID 为 0）的给定分区 ID 的分区名称。由于未指定数据库 ID，已在当前数据库中完成查找：

```
select partition_name(0, 1111111111)
```

示例 2 返回属于 testdb 数据库中的聚簇索引 （已指定索引 ID 为 1）的给定分区 ID 的分区名称。

```
select partition_name(1, 1212121212, db_id("testdb")
```

用法

- 如果此搜索未找到目标分区，则返回 NULL。

另请参见

函数 data_pages、 object_id、 partition_id、 reserved_pages、 row_count

partition_object_id

说明

显示指定分区 ID 和数据库 ID 的对象 ID。

语法

partition_object_id(*partition_id* [, *database_id*])

参数

partition_id
是要检索其对象 ID 的分区的 ID。

database_id
是分区的数据库 ID。

示例

示例 1 显示分区 ID 为 2 的分区的对象 ID：

```
select partition_object_id(2)
```

示例 2 显示分区 ID 为 14 且数据库 ID 为 7 的分区的对象 ID：

```
select partition_object_id(14,7)
```

示例 3 由于为以下函数传递了 NULL 值，因而为数据库 ID 返回 NULL 值：

```
select partition_object_id( 1424005073, NULL)

 -----------
NULL
(1 row affected)
```

用法

- 如果没有包含数据库 ID， partition_object_id 将使用当前数据库 ID。
- 如果使用 NULL 值作为 *partition_id*，partition_object_id 将返回 NULL。
- 如果包含空值作为数据库 ID， partition_object_id 将返回空值。
- 如果提供无效或不存在的 *partition_id* 或 *database_id*，partition_object_id 将返回 NULL。

password_random

说明

生成的伪随机密码可满足 Adaptive Server 上定义的全局口令复杂程度检查。“伪随机”指 Adaptive Server 模拟类似随机编号的编号，因为没有计算机能够生成真正意义上的随机编号。复杂程度检查包括：

- 最小口令长度
- 下列内容的最小数目：
  - 口令中包含的数字
  - 口令中包含的特殊字符
  - 口令中包含的字母字符
  - 口令中包含的大写字符
  - 口令中包含的小写字符

语法

password_random ([*pwdlen*])

参数

pwdlen

指定随机口令长度的整数。如果省略 pwdlen， Adaptive Server 会生成长度由 “最小口令长度”全局选项确定的口令，缺省值为 6。

示例

示例 1 显示服务器中存储的口令复杂程度检查：

```
minimum password length:           10
min digits in password:             2
min alpha in password:              4
min upper char in password:         1
min special char in password:      -1
min lower char in password:         1

select password_random()
----------------------
6pY5l6UT]Q
```

示例 2 显示服务器中存储的口令复杂程度检查：

```
minimum password length:                  15
minimum digits in password:                4
minimum alpha in password:                 4
minimum upper-case characters in password: 1
minimum lower-case characters in password: 2
minimum special characters in password:    4

select password_random(25)
-----------------
S/03iuX[ISi:Y=?8f.[eH%P51
```

示例 3 使用所有姓名以 “A” 开头的员工的随机口令更新 password 列：

```
update employee
set password = password_random()
where name like 'A%'
```

示例 4 生成随机口令，并使用它为用户 “anewman” 创建登录帐户。

```
declare @password varchar(10)
select @password =              password_random(10)
exec sp_addlogin 'jdoe', @password
```

示例 5 如果直接使用生成的随机口令，则用单引号或双引号将其括起来：

```
select @password = password_random(11)
-----------
%k55Mmf/2U2

sp_adlogin 'jdoe','%k55Mmf/2U2'
```

用法

password_random() 生成的口令是伪随机的；要生成真正的随机口令，需使用更强大的随机生成器。

patindex

说明

返回指定模式第一次出现时的起始位置。

语法

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

参数

pattern

是 char 或 varchar 数据类型的字符表达式，可包括 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）。有关通配符的说明，请参见第 360 页上的 “与通配符匹配的模式”。

char_expr

类型为 char、varchar、nchar、nvarchar、text_locator 或 unitext_locator 的字符型列名、变量或常量表达式。

uchar_expr

类型为 unichar 或 univarchar 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

返回偏移量 （以字节表示）。

chars 或 characters

返回以字节表示的偏移量 （缺省值）。

示例

示例 1 选择作者 ID 和 “circus” 一词在 copy 列中的起始字符位置。

```
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
```

示例 2

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

示例 3 查找 sysobjects 中所有以“sys”开头且第四个字符为“a”、“b”、“c”或 “d”的行：

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

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

用法

- patindex 是一个字符串函数，它返回整数，表示指定字符表达式中的 *pattern* 首次出现时的起始位置，如果未找到 *pattern*，则返回 0。
- patindex 可用于所有字符数据，包括 text 和 image 数据。
- 对于 text、unitext 和 image 数据，如果 ciphertext 设置为 1，则不支持 patindex。将会显示错误消息。
- 对于 text、unitext 和 image 数据，如果 ciphertext 设置为 0，则会返回纯文本内的模式的字节或字符索引。
- 对于 unichar、univarchar 和 unitext，patindex 返回以 Unicode 字符表示的偏移量。在进行比较之前，模式字符串将隐式转换为 UTF-16，且此比较基于 default unicode sort order 配置。例如，如果 unitext 列包含行值 U+0041U+0042U+d800U+dc00U+0043，则将返回以下信息：

  ```
  select patindex("%C%", ut) from unitable
  -----------
  4
  ```

- 缺省情况下，patindex 返回偏移量（以字符表示）；若要返回以字节表示 （多字节字符串）的偏移量，请指定 using bytes。
- 在 *pattern* 之前和之后加上百分比符号。若要查找作为前几个字符出现在列中的 *pattern*，请省略前面的 %。若要查找作为最后几个字符出现在列中的 *pattern*，请省略后面的%。
- 如果 *char_expr* 或 *uchar_expr* 是 NULL，则 patindex 返回 0。

- 如果将 varchar 表达式用作一个参数，并将 unichar 表达式用作另一个参数，则 varchar 表达式将会隐式地转换为 unichar（可能会发生截断）。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 patindex。

另请参见　**文档**　*《Transact-SQL 用户指南》*

函数　charindex、 substring

pi

说明	返回常量值 3.1415926535897936。
语法	pi()
参数	无

示例	`select pi()`
	`------------------`
	`3.141593`

用法	pi 是一个数学函数，它返回常量值 3.1415926535897931。
标准	符合 ANSI SQL 的级别 Transact-SQL 扩展。
权限	任何用户都可以执行 pi。
另请参见	**文档** *《Transact-SQL 用户指南》* **函数** degrees、 radians

power

说明　返回求指定数字的给定次幂所得的值。

语法　power(*value*, *power*)

参数

value
是数值。

power
是精确数值、近似数值或货币值。

示例

```
select power(2, 3)

-----------
          8
```

用法

- power 是一个数学函数，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。

  在 numeric 或 decimal 类型的表达式中，此函数返回精度 38 和标度 18。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 power。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　exp、 log、 log10

proc_role

说明

返回有关是否已授予用户指定角色的信息。

注释 Sybase 支持并建议使用 has_role 而不使用 proc_role。不过，您不必将现在使用的 proc_role 转换为 has_role。

语法

proc_role("*role_name*")

参数

role_name

是系统角色或用户定义的角色的名称。

示例

示例 1 创建一个过程以检查用户是否是系统管理员：

```
create procedure sa_check as
if (proc_role("sa_role") > 0)
begin
    print "You are a System Administrator."
    return (1)
end
```

示例 2 检查用户是否已被授予系统安全员角色：

```
select proc_role("sso_role")
```

示例 3 检查用户是否已被授予操作员角色：

```
select proc_role("oper_role")
```

用法

- 将 proc_role 用于以 “sp_” 开始的过程会返回错误。
- proc_role 是一个系统函数，它检查是否已将指定角色授予调用用户，以及是否已将该角色激活。
- 如果出现以下情形，
- proc_role 将返回 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，则 proc_role 返回 1。
- 如果调用用户当前具有活动角色，而该角色又包含指定角色，则 proc_role 将返回 2。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 proc_role。

另请参见

命令　alter role、 create role、 drop role、 grant、 set、 revoke

文档　*《Transact-SQL 用户指南》*

函数　mut_excl_roles、 role_contain、 role_id、 role_name、 show_role

pssinfo

说明

从 Adaptive Server 进程状态结构 (pss) 返回信息。

语法

pssinfo(*spid* | 0, '*pss_field*')

参数

spid

如果输入 0，则使用当前进程。

pss_field

是进程状态结构字段。有效值为：

- dn — 使用 LDAP 鉴定时的区分名。
- extusername — 使用外部鉴定（如 PAM、LDAP）时，extusername 将返回所使用的外部 PAM 或 LDAP 用户名。
- ipaddr — 客户端 IP 地址。
- ipport — 用于与要查询的用户任务相关的客户端连接的客户端 IP 端口号。
- isolation_level — 当前会话的隔离级别。
- tempdb_pages — 使用的 tempdb 页数。

示例

显示 spid 号 14 的端口号

```
select pssinfo(14,'ipport')
----------------
           52039
```

用法

- pssinfo 函数也包含用于显示外部用户名和区分名的选项。
- ipport 输出与 ipaddr 输出结合使用可以唯一标识 Adaptive Server 和客户端之间的网络通信量。

权限

对 pssinfo 函数的权限检查因您的细化权限设置而异。

细化权限已启用	在启用细化权限的情况下，您必须是进程 ID 的所有者或具有 manage server 权限才能执行 pssinfo。
细化权限已禁用	在禁用细化权限的情况下，您必须是进程 ID 的所有者，或是具有 sa_role 或 sso_role 的用户才能执行 pssinfo。

radians

说明　将度转换为弧度。返回指定度数角的大小 （以弧度表示）。

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 double precision）或 money 列、变量、常量表达式，或上述数据类型的组合形式。

示例

```
select radians(2578)

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

用法

- radians 是一个数学函数，它可将度转换为弧度。其结果与 *numeric* 的类型相同。

  若要表达数值数据类型或小数数据类型，此函数返回精度 38 和标度 18。

  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。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 radians。

另请参见　**文档**　*《Transact-SQL 用户指南》*

函数　degrees

rand

说明

使用指定 （可选）整数作为源值，返回 0 和 1 之间的随机浮点值。

语法

rand([*integer*])

参数

integer

是任意整数型 （tinyint、 smallint 或 int）列名、变量、常量表达式或上述数据类型的组合形式。

示例

示例 1

```
select rand()

--------------------
          0.395740
```

示例 2

```
declare @seed int
select @seed=100
select rand(@seed)

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

用法

- rand 是一个数学函数，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 位伪随机整数生成器的输出。该整数最多可除以 32 位整数，得出 0.0 到 1.0 之间的双精度值。服务器启动时将随机产生 rand 函数的源值，因此不可能得到相同的随机编号序列，除非用户先用常量源值对该函数进行初始化。 rand 函数是全局资源。如果有多个用户调用 rand 函数，则按单一伪随机值流依次进行。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可重复序列时，没有其他用户调用 rand。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 rand。

另请参见

数据类型 近似数值数据类型

文档 *《Transact-SQL 用户指南》*

函数 rand2

rand2

说明

返回 0 和 1 之间的随机值，该值由指定的种子值来生成，在 select 列表中使用时将为每个返回的行计算该值。

语法

rand2([*integer*])

参数

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

示例

如果表 t 中有 n 行，则下面的 select 语句将返回 n 个不同的随机值，而不是只返回一个。

```
select rand2() from t
------------
```

用法

- rand2 是一个数学函数，它使用可选整数作为源值，返回 0 和 1 之间的随机浮点值。与 rand 不同，它是在用于 select 列表时为每个返回的行计算的。
- 当前未定义 rand2 在 select 列表以外的位置的行为。
- 有关 32 位伪随机整数生成器的详细信息，请参见 rand 的 “用法” 部分。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 rand。

另请参见

数据类型　近似数值数据类型

文档　*《Transact-SQL 用户指南》*

函数　rand

replicate

说明

返回一个字符串，其中包含重复给定次数的指定表达式，或包含重复多次，直至其大小可占据 16KB 空间的指定表达式 （取其次数少者）。

语法

replicate(*char_expr* | *uchar_expr*, *integer_expr*)

参数

char_expr

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

uchar_expr

类型为 unichar 或 univarchar 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）类型的列名、变量或常量表达式。

示例

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

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

用法

- replicate 是一个字符串函数，它返回与 *char_expr* 或 *uchar_expr* 具有相同数据类型的字符串，该字符串包含重复指定次数的同一表达式，或包含重复多次，直至其大小可占据 16K 空间的同一表达式 （取其次数少者）。
- 如果 *char_expr* 或 *uchar_expr* 是 NULL，则返回一个 NULL。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 replicate。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 stuff

reserve_identity

说明

reserve_identity 允许进程保留标识值块，以供该进程使用。

在进程调用 reserve_identity 以保留值块后，将从此保留池中提取此进程需要的后续标识值。当这些保留的数字用完后或将数据插入不同表中时，将应用现有的标识选项。reserve_identity 可以保留多个标识值块，因此如果对不同表的插入过程被单个进程隔开，将使用表的保留块中的下一个值。

为指定表保留指定大小的标识值块，这些标识值由调用过程专用。返回保留的起始数字，此进程对指定表执行的后续 insert 将使用这些值。当进程终止时，将删除所有未使用的值。

语法

reserve_identity (*table_name*, *number_of_values*)

参数

table_name

是为其进行保留的表的名称。该名称可以是完全限定名；也就是说，它可以包括 *database_name*、 *owner_name* 和 *object_name* （用引号括起）。

number_of_values

是为此进程保留的顺序标识值的数目。此值必须是一个正值，不会使任何保留值超出标识列的数据类型的最大值。

示例

介绍 reserve_identity 的典型使用方案，并假定 table1 包括 col1（数据类型为 int）和 col2 （数据类型为 int 的标识列）。此进程针对 spid 3：

```
select reserve_identity("table1", 5 )
---------------
10
```

为 spid 3 和 4 插入值：

```
Insert table1 values(56) -> spid 3
Insert table1 values(48) -> spid 3
Insert table1 values(96) -> spid 3
Insert table1 values(02) -> spid 4
Insert table1 values(84) -> spid 3
```

从表 table1 中选择：

```
select * from table1

Col1          col2
--------       -----
3              1-> spid 3 reserved 1-5
3              2-> spid 3
3              3-> spid 3
4              6<= spid 4 gets next unreserved value
```

```
3               4<= spid 3 continues with reservation
```

结果集显示， spid 3 保留了标识值 1 – 5， spid 4 接收下一个未保留值，然后 spid 3 保留后续标识值。

用法

- sp_configure 系统过程的“identity reservation size”参数为传递至 *number_of_values* 的某个值指定了全服务器范围的限制。
- 返回值 *start_value* 是保留的标识值块的起始值。调用进程将此值用于到指定表的下一次插入
- reserve_identity 允许进程执行下列操作：
  - 保留标识值，而不发出 insert 语句。
  - 在发出 insert 语句前即知道值已保留
  - 根据需要 “争夺”不同大小的标识值块。
  - 通过只保留所需内容更好地控制“超出间隔”（也就是说，它们不受预设置的服务器争夺大小的限制）
- 自动使用值，而不更改 insert 语法。
- 将在以下情况下返回 NULL 值：
  - 将负值或零指定为块大小。
  - 表不存在。
  - 表不包含标识列。
- 如果对此进程已在其中保留标识值的表发出 reserve_identity，该函数将成功并使用最新的值组。
- 不能使用 reserve_identity 在代理表中保留标识值。如果本地服务器调用的远程过程调用了 reserve_identity，则本地服务器可以对远程表使用 reserve_identity。由于这些保留值存储在远程服务器上，但却在属于本地服务器的会话中，因此对远程表的后续插入将使用这些保留值。
- 如果 identity_gap 小于保留的块大小，保留通过保留指定值块大小（而不是 identity_gap 大小）取得成功。如果进程未使用这些值，则无论 identity_gap 设置如何，都将导致间隔可能达到指定块大小。

权限

必须对表具有 insert 权限才能保留标识值。权限检查不会因细化权限设置而不同。

另请参见

过程 sp_configure

reserved_pages

说明

报告为数据库、对象或索引保留的页数。结果包括用于内部结构的页。

此函数取代了 Adaptive Server 15.0 之前版本中使用的 reserved_pgs 函数。

语法

reserved_pages(*dbid*, *object_id*[, *indid*[, *ptnid*]])

参数

dbid
是目标对象所在的数据库的数据库 ID。

object_id
是表的对象 ID。

indid
是目标索引的索引 ID。

ptnid
是目标分区的分区 ID。

示例

示例 1 返回指定数据库中对象 ID 为 31000114 的对象所保留的页数（包括所有索引）：

```
select reserved_pages(5, 31000114)
```

示例 2 返回数据层中的对象所保留的页数，而不管是否存在聚簇索引：

```
select reserved_pages(5, 31000114, 0)
```

示例 3 返回索引层中的对象为聚簇索引所保留的页数。其中不包括数据层所使用的页数：

```
select reserved_pages(5, 31000114, 1)
```

示例 4 返回特定分区的数据层中的对象所保留的页数，在此示例中为 2323242432：

```
select reserved_pages(5, 31000114, 0, 2323242432)
```

示例 5 使用 reserved_pages，通过以下三种方法之一计算数据库中的空间：

- 使用 case 表达式选择一个适合要检查的索引的值，以便在此数据库的 sysindexes 中选择所有非日志索引。在此查询中：
  - 数据具有值“索引 0”，并在包括语句 `when sysindexes.indid = 0` 或 `sysindexes.indid = 1` 时可用。
  - 大于 1 的 indid 值是索引。因为此查询不将数据空间合计到索引计数中，所以它不包括 indid 为 0 的页计数。
  - 每个对象具有的索引条目的索引为 0 或 1，绝不会同时为这两者。
  - 此查询只对每个表的索引 0 计数一次。

```
select
'data rsvd' = sum( case
          when indid > 1 then 0
          else reserved_pages(db_id(), id, 0)
          end ),
'index rsvd' = sum( case
          when indid = 0 then 0
          else reserved_pages(db_id(), id, indid)
          end )
from sysindexes
where id != 8
data rsvd   index rsvd
----------   -----------
       812           1044
```

- 查询 sysindexes 多次以在所有查询完成后显示结果：

```
declare @data int,
@dbsize int,
@dataused int,
@indices int,
@indused int
select @data = sum( reserved_pages(db_id(), id, 0) ),
     @dataused = sum( used_pages(db_id(), id, 0) )
from sysindexes
where id != 8
and indid <= 1
select @indices = sum( reserved_pages(db_id(), id, indid) ),
     @indused = sum( used_pages(db_id(), id, indid) )
from sysindexes
where id != 8 and indid > 0
select @dbsize   as 'db size',
@data as 'data rsvd'
db size    data rsvd
----------- -----------
       NULL         820
```

- 查询 sysobjects 以获取数据空间信息，查询 sysindexes 以获取索引信息。从 sysobjects 中选择表对象：[S] 系统或 [U] 用户：

```
declare   @data int,
          @dbsize int,
          @dataused int,
          @indices int,
          @indused int
select @data = sum( reserved_pages(db_id(), id, 0) ),
@dataused = sum( used_pages(db_id(), id, 0) )
from sysobjects
```

```
where id != 8
and type in ('S', 'U')
select @indices = sum( reserved_pages(db_id(), id, indid) ),
       @indused = sum( used_pages(db_id(), id, indid) )
from sysindexes
where id != 8
and indid > 0
select   @dbsize as 'db size',
         @data as 'data rsvd',
         @dataused as 'data used',
         @indices as 'index rsvd',
         @indused as 'index used'
db size    data rsvd    data used    index rsvd    index used
---------  -----------  -----------  -----------   ----------
     NULL          812          499          1044          381
```

用法

- 如果所有页锁定表中存在聚簇索引，则传递的索引 ID 为 0 时，将报告保留的数据页；传递的索引 ID 为 1 时，将报告保留的索引页。所有错误情形都将导致返回的值为零。
- reserved_pages 计数所指定的内容；如果提供的是有效的数据库、对象、索引（每个表的数据都是“索引 0”），它将返回此数据库、对象或索引的保留空间。但是，它也可能对数据库、对象或索引进行多次计数。如果让它对具有多个索引的表中的每个索引的数据空间进行计数，应让它对每个索引的数据空间计数一次。如果合计这些结果，应将索引数乘以总数据空间，而不是对象中的总数据页数。
- reserved_pages 会放弃尚未在高速缓存中的对象的描述符，而不是消耗资源。
- 对于 Adaptive Server 15.0 和更高版本，reserved_pages 取代了 reserved_pgs 函数。 reserved_pages 和 reserved_pgs 之间存在差异。
  - 在 Adaptive Server 12.5 及更早版本中，Adaptive Server 将数据和索引的 OAM 页存储在 sysindexes 中。在 Adaptive Server 15.0 及更高版本中，此信息按分区存储在 sysparitions 中。因为此信息的存储位置不同，所以 reserved_pages 和 reserved_pgs 需要不同的参数并具有不同的结果集。
  - reserved_pgs 需要页 ID。如果提供的值没有匹配的 sysindexes 行，则提供的页 ID 为 0（例如，非聚簇索引行的数据 OAM 页）。因为 0 决不是有效的 OAM 页，所以如果提供的页 ID 为 0，则 reserved_pgs 返回 0 ；因为输入值无效，所以 reserved_pgs 无法对任何内容进行计数。

不过，reserved_pages 需要索引 ID，0 是有效的索引 ID（例如，每个表的数据都是 “索引 0”）。因为 reserved_pages 无法从环境中知道不需要它对除 indid 0 或 1 以外的任何索引行的数据空间进行重新计数，所以它会在您每次将 0 作为索引 ID 传递时对数据空间进行计数。因为 reserved_pages 无法从环境中知道不需要它对除 indid 0 或 1 以外的任何索引行的数据空间进行重新计数，所以它会在您每次将 0 作为索引 ID 传递时对数据空间进行计数。

这些差异如下所述：

- 如果提供 0 作为 OAM 页输入的页 ID 值，则 reserved_pgs 不会影响合计；它只返回值 0。
- 如果为 reserved_pages 提供值 0 作为索引 ID，则它对数据空间进行计数。仅在需要对数据进行计数时发出 reserved_pages，否则会影响合计。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 reserved_pgs。

另请参见

命令 update statistics

函数 data_pages、 reserved_pages、 row_count、 used_pages

return_lob

说明　取消引用定位符，并返回该定位符引用的 LOB。

语法　return_lob (*datatype*, *locator_descriptor*)

参数

datatype
是 LOB 的数据类型。有效的数据类型为：

- text
- unitext
- image

locator_descriptor
LOB 定位符的有效表示：宿主变量、局部变量或定位符的实际二进制值。

示例　以下示例取消引用定位符并返回实际定位符值 0x9067ef450100000000100000004010040080000000 0 所引用的 LOB。

```
return_lob (text, locator_literal(text_locator,
0x9067ef450100000000100000004010040080000000))
```

用法　return_lob 会覆盖 set send_locator on 命令，并始终返回 LOB。

权限　任何用户都可以执行 return_lob。

另请参见　**命令**　deallocate locator、 truncate lob

Transact-SQL 函数　locator_literal、 locator_valid、 create_locator

reverse

说明

返回其字符逆序排列的指定字符串。

语法

reverse(*expression* | *uchar_expr*)

参数

expression

类型为 char、varchar、nchar、nvarchar、binary 或 varbinary 的字符型或二进制型列名、变量或常量表达式。

uchar_expr

类型为 unichar 或 univarchar 的字符型或二进制型列名、变量或常量表达式。

示例

示例 1

```
select reverse("abcd")

----
dcba
```

示例 2

```
select reverse(0x12345000)

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

用法

- reverse 是一个字符串函数，它返回 *expression* 的逆序字符串。
- 如果 *expression* 是 NULL，则 reverse 返回 NULL。
- 代理对被视为不可拆分，因此不能逆序。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 reverse。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 lower、 upper

right

说明

返回字符或二进制表达式中从右端起具有指定字符数的部分。返回值的数据类型与字符表达式的数据类型相同。

语法

right(*expression*, *integer_expr*)

参数

expression

类型为 char、varchar、nchar、unichar、nvarchar、univarchar、binary 或 varbinary 的字符型或二进制型列名、变量或常量表达式。

integer_expr

是任意整数（tinyint、smallint 或 int）类型的列名、变量或常量表达式。

示例

示例 1

```
select right("abcde", 3)

 ---
 cde
```

示例 2

```
select right("abcde", 2)

 --
 de
```

示例 3

```
select right("abcde", 6)

 -----
 abcde
```

示例 4

```
select right(0x12345000, 3)

 -------
 0x345000
```

示例 5

```
select right(0x12345000, 2)

 ------
 0x5000
```

示例 6

```
select right(0x12345000, 6)

 ---------
 0x12345000
```

用法

- right 是一个字符串函数，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。
- 如果指定的最右部分始于替换对的第二个代理（下代理），则返回的值从下一个完整字符开始。因此，将少返回一个字符。
- 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，则 right 返回 NULL。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展

权限

任何用户都可以执行 right。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 rtrim、 substring

rm_appcontext

说明

删除特定应用程序环境或所有应用程序环境。rm_appcontext 由 ACF 提供。

语法

rm_appcontext("*context_name*", "*attribute_name*")

参数

context_name
是用于指定应用程序环境名的行。它保存为 char(30) 数据类型。

attribute_name
是用于指定应用程序环境属性名的行。它保存为 char(30) 数据类型。

示例

示例 1 通过指定某些属性或所有属性来删除应用程序环境：

```
select rm_appcontext("CONTEXT1", "*")
----------
0

select rm_appcontext("*", "*")
---------
0

select rm_appcontext("NON_EXISTING_CTX","ATTR")
---------
-1
```

示例 2 显示没有相应权限的用户尝试删除应用程序环境时所返回的结果：

```
select rm_appcontext("CONTEXT1","ATTR2")
----------
-1
```

用法

- 此函数始终返回 0 表示成功。
- 此函数的所有参数都是必需的。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

对 rm_appcontext 的权限检查因您的细化权限设置而异。

细化权限已启用	在启用细化权限的情况下，您必须具有 rm_appcontext 的 select 权限才能执行此函数。
细化权限已禁用	在禁用细化权限的情况下，您必须是具有 sa_role 的用户，或者具有 rm_appcontext 的 select 权限才能执行此函数。

另请参见

有关 ACF 的详细信息，请参见 *《系统管理指南》* 第 11 章 “管理用户权限”中的 “行级访问控制”。

函数 get_appcontext、 list_appcontext、 set_appcontext

role_contain

说明

确定一个指定的角色是否包含在另一个指定的角色中。

语法

role_contain("*role1*", "*role2*")

参数

role1

是系统角色或用户定义的角色的名称。

role2

是其它系统角色或用户定义的角色的名称。

示例

示例 1

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

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

示例 2

```
select role_contain("specialist_role", "intern_role")
-----------
0
```

用法

role_contain 是一个系统函数，如果 *role2* 包含 *role1*，则返回 1。否则，role_contain 返回 0。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 role_contain。

另请参见

文档 有关包含的角色和角色层次的详细信息，请参见 *《系统管理指南》*。有关系统函数，请参见 *《Transact-SQL 用户指南》*。

函数 mut_excl_roles、 proc_role、 role_id、 role_name

命令 alter role

系统过程 sp_activeroles、 sp_displayroles、 sp_role

role_id

说明　返回指定角色名的角色 ID。

语法　role_id("*role_name*")

参数　*role_name*

是系统角色或用户定义的角色的名称。角色名称和角色 ID 都存储在 syssrvroles 系统表中。

示例　**示例 1** 返回 sa_role 的系统角色 ID：

```
select role_id("sa_role")
------
0
```

示例 2 返回 “intern_role” 的系统角色 ID：

```
select role_id("intern_role")
------
6
```

用法

- role_id 是一个系统函数，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* 在系统中是无效角色，则 Adaptive Server 返回 NULL。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 role_id。

另请参见　**文档**　有关以下内容的详细信息：

- 角色 — 请参见 *《系统管理指南》*
- 系统函数 — 请参见 *《Transact-SQL 用户指南》*。

函数　mut_excl_roles、 proc_role、 role_contain、 role_name

role_name

说明

返回指定角色 ID 的角色名。

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

```
select role_name(01)

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

用法

role_name 是一个系统函数，它返回角色名称。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展

权限

任何用户都可以执行 role_name。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 mut_excl_roles、 proc_role、 role_contain、 role_id

round

说明

返回指定数字舍入到指定的小数位数后所得的值。

语法

round(*number*, *decimal_places*)

参数

number

是任意精确数值（numeric、dec、decimal、tinyint、smallint、int 或 bigint）、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形式。

decimal_places

是要舍入到的小数位数。

示例

示例 1

```
select round(123.4545, 2)
----------
  123.4500
```

示例 2

```
select round(123.45, -2)
----------
  100.00
```

示例 3

```
select round(1.2345E2, 2)
-----------------
       123.450000
```

示例 4

```
select round(1.2345E2, -2)
-----------------
      100.000000
```

用法

- round 是一个数学函数，它通过舍入 *number* 来使其具有 *decimal_places* 个有效位数。
- *decimal_places* 的正值确定小数点右边的有效位数；*decimal_places* 的负值确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 始终会返回值。如果 *decimal_places* 为负且超过了为 *number* 指定的有效位数，则 Adaptive Server 返回 0。（它以 0.00 的形式表示，其中小数点右边 0 的个数等于 numeric 的标度。）例如，以下命令返回值 0.00：

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

标准	符合 ANSI SQL 的级别 Transact-SQL 扩展。
权限	任何用户都可以执行 round。
另请参见	**文档** *《Transact-SQL 用户指南》* **函数** abs、 ceiling、 floor、 sign、 str

row_count

说明　返回指定表中行数的估计值。

语法　row_count(*dbid*, *object_id* [,*ptnid*] [, "*option*"])

参数

dbid
目标对象所在的数据库 ID。

object_id
表的对象 ID。

ptnid
所需的分区 ID。

示例

示例 1 返回给定对象中行数的估计值：

```
select row_count(5, 31000114)
```

示例 2 返回对象 ID 为 31000114 的对象的指定分区（分区 ID 为 2323242432）中行数的估计值：

```
select row_count(5, 31000114, 2323242432)
```

用法

- 所有错误情形都将导致返回的值为零。
- row_count 会放弃尚未在高速缓存中的对象的描述符，而不是消耗资源。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 row_count。

另请参见　**函数** reserved_pages、 used_pages

rtrim

说明 删去指定表达式的尾随空白。

语法 rtrim(*char_expr* | *uchar_expr*)

参数

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

uchar_expr
类型为 unichar 或 univarchar 的字符型列名、变量或常量表达式。

示例

```
select rtrim("abcd    ")

--------
abcd
```

用法

- rtrim 是一个字符串函数，它删除尾随空白。
- 对于 Unicode，空白被定义为 Unicode 的值 U+0020。
- 如果 *char_expr* 或 *uchar_expr* 是 NULL，则返回 NULL。
- 只删除当前字符集中等于空格字符的值。

标准 符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限 任何用户都可以执行 rtrim。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 ltrim

sdc_intempdbconfig

说明　（仅限集群环境）如果系统当前处于临时数据库配置模式，则返回 1 ；否则返回 0。

语法

```
sdc_intempdbconfig()
```

示例

```
select sdc_intempdbconfig()
```

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以运行 sdc_intempdbconfig。

set_appcontext

说明

为指定应用程序的属性定义的用户会话设置应用程序环境名称、属性名称和属性值。 set_appcontext 由 ACF 提供。

语法

set_appcontext("*context_name*, "*attribute_name*", "*attribute_value*")

参数

context_name
是用于指定应用程序环境名的行。它保存为 char(30) 数据类型。

attribute_name
是用于指定应用程序环境属性名的行。它保存为 char(30) 数据类型。

attribute_value
是用于指定应用程序属性值的行。它保存为 char(30) 数据类型。

示例

示例 1 创建名为 CONTEXT1 的应用程序环境，其属性 ATTR1 的值为 VALUE1。

```
select set_appcontext("CONTEXT1", "ATTR1", "VALUE1")

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

试图覆盖已创建的现有应用程序环境将导致以下结果：

```
select set_appcontext("CONTEXT1", "ATTR1", "VALUE1")

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

示例 2 显示值中包含数据类型转换的 set_appcontext。

```
declare@numericvarchar varchar(25)
select @numericvar = "20"
select set_appcontext ("CONTEXT1", "ATTR2",
convert(char(20), @numericvar))

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

示例 3 显示没有相应权限的用户尝试设置应用程序环境时所返回的结果。

```
select set_appcontext("CONTEXT1", "ATTR2", "VALUE1")
-------------
-1
```

用法

- set_appcontext 返回 0 表示成功，返回 -1 表示失败。
- 如果设置的值在当前会话中已存在，则 set_appcontext 返回 -1。
- 此函数不能替换现有应用程序环境的值。若要为一个环境指定新值，应先删除该环境，然后使用新值重新创建一个环境。

- set_appcontext 将属性保存为 char 数据类型。如果创建必须将属性值与其它数据类型进行比较的访问规则，该规则应将 char 数据转换为相应的数据类型。
- 此函数的所有参数都是必需的。

标准　符合 ANSI SQL 的级别：Transact-SQL 扩展。

权限　对 set_appcontext 的权限检查因您的细化权限设置而异。

细化权限已启用	在启用细化权限的情况下，您必须具有 set_appcontext 的 select 权限才能执行此函数。
细化权限已禁用	在禁用细化权限的情况下，您必须是具有 sa_role 的用户，或者具有 set_appcontext 的 select 权限才能执行此函数。

另请参见

文档　有关 ACF 的详细信息，请参见 *《系统管理指南》*第 11 章 “管理用户权限”中的 “行级访问控制”。

函数　get_appcontext、 list_appcontext、 rm_appcontext

setdata

说明

覆盖一个大对象 (LOB) 的部分或全部。

语法

setdata(*locator_name*, *offset_value*, *new_value*)

参数

locator_name

是引用要修改的 LOB 值的定位符。

offset_value

是 *locator_name* 在 LOB 内所指向的位置。这是 Adaptive Server 开始写入 *new_value* 内容的位置。*offset_value* 的值对于 text_locator 和 unitext_locator 采用字符形式，对于 image_locator 采用字节形式。LOB 的第一个字符或字节的 *offset_value* 为 1。

new_value

是要用来覆盖旧数据的数据。

示例

以下示例中的最后一个 select 语句返回字符串 “Sybase ABC/IQ/ASA”，而不是原始字符串 “Sybase ASE/IQ/ASA”：

```
declare @v text_locator
  select @v = create_locator
    (text_locator, convert(text, "Sybase ASE/IQ/ASA")
  select setdata(@v, 8, "ABC")
  select return_lob(text, @v)
```

用法

- setdata 就地修改 LOB 值。也就是说，在 LOB 被修改之前，Adaptive Server 不会复制它。
- 如果在跳过 *offset_value* 之后， *new_value* 的长度超过 LOB 的剩余长度， Adaptive Server 会扩展 LOB 以容纳 *new_value* 的整个长度。
- 如果 *new_value* 和 *offset_value* 的总和比 LOB 的长度短， Adaptive Server 不会在 LOB 的结尾处更改或截断数据。
- 如果 *offset_value* 比要更新的 LOB 值长， setdata 会返回 NULL。

权限

任何用户都可以执行 setdata。

另请参见

命令 deallocate locator、 truncate lob

Transact-SQL 函数 locator_valid、 return_lob、 create_locator

show_cached_plan_in_xml

说明

以 XML 格式显示语句高速缓存中的查询的执行查询计划。

show_cached_plan_in_xml 以 XML 格式返回 showplan 实用程序输出的各个部分。

语法

show_cached_plan_in_xml(*statement_id*, *plan_id*, [*level_of_detail*])

参数

statement_id

轻量过程的对象 ID。轻量过程是可由 Adaptive Server 在内部创建和调用的过程。这是 monCachedStatement 的 SSQLID 列，其中包含每个高速缓存语句的唯一标识符。

plan_id

计划的唯一标识符。这是来自 monCachedProcedures 的 PlanID。如果 *plan_id* 的值为零，会显示所指示的 SSQLID 的所有高速缓存计划的 showplan 输出。

level_of_detail

是一个 0 – 6 的值，指示 show_cached_plan_in_xml 返回的细节数量（请参见表 2-6）。*level_of_detail* 确定 show_cached_plan_in_xml 将返回 showplan 的哪些部分。缺省值为 0。

show_cached_plan_in_xml 的输出包括 *plan_id* 和以下部分：

- `parameter` — 包含用于编译查询的参数值以及导致最低性能的参数值。编译参数是用 `<compileParameters>` 和 `</compileParameters>` 标记指示的。最低性能参数值是用 `<execParameters>` 和 `</execParameters>` 标记指示的。对于每个参数，show_cached_plan_in_xml 都会显示：
  - 编号
  - 数据类型
  - 值 — 大于 500 字节的值和插入值语句的值不会显示。对于两个参数集来说，用于存储所有参数的值的总内存均为 2KB。

示例

示例 1 在 XML 中呈现的一个查询计划：

```
select show_cache_plan_in_xml(1328134997,0)
go
-------------------------------------------

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

```
</text>
<ºýªÆ>
    <planId>11</planId>
    <planStatus> available </planStatus>
    <execCount>1371</execCount>
    <maxTime>3</maxTime>
    <avgTime>0</avgTime>
    <compileParameters/>
    <execParameters/>
    <opTree>
        <Emit>
        <VA>1</VA>
        <est>
            <rowCnt>10</rowCnt>
            <lio>0</lio>
            <pio>0</pio>
            <rowSz>22.54878</rowSz>
        </est>
        <act>
            <rowCnt>1</rowCnt>
        </act>
        <arity>1</arity>
            <IndexScan>
                <VA>0</VA>
                <est>
                    <rowCnt>10</rowCnt>
                    <lio>0</lio>
                    <pio>0</pio>
                    <rowSz>22.54878</rowSz>
                </est>
                <act>
                    <rowCnt>1</rowCnt>
                    <lio>3</lio>
                    <pio>0</pio>
                </act>
                <varNo>0</varNo>
                <objName>sysobjects</objName>
                <scanType>IndexScan</scanType>
                <indName>csysobjects</indName>
                <indId>3</indId>
                <scanOrder> ForwardScan </scanOrder>

                <positioning> ByKey </positioning>
                <perKey>
                    <keyCol>id</keyCol>
                    <keyOrder> Ascending </keyOrder>
```

```
                </perKey>
                <indexIOSizeInKB>2</indexIOSizeInKB>
                <indexBufReplStrategy> LRU </indexBufReplStrategy>
                <dataIOSizeInKB>2</dataIOSizeInKB>
                <dataBufReplStrategy> LRU </dataBufReplStrategy>
             </IndexScan>
         </Emit>
    </opTree>
</plan>
```

示例 2 以下示例显示了 Adaptive Server 15.7.1 和更高版本中提供的增强的 <est>、 <act> 和 <scanCoverage> 标记：

```
select show_cached_plan_in_xml(1123220018, 0)
go

<?xml version="1.0" encoding="UTF-8"?>
< query>
   <statementId>1123220018</statementId>
   <Œƒ±æ>
   <![CDATA[
   SQL Text:select distinct c1, c2 from t1, t2 where c1 = d1 PLAN '(
distinct_hashing ( nl_join ( t_scan t2 ) ( i_scan i1t1 t1 ) ) )']]>
   </text>
   <ºýªÆ>
      <planId>6</planId>
      <planStatus> available </planStatus>
      <execCount>1</execCount>
      <maxTime>16</maxTime>
      <avgTime>16</avgTime>
      <compileParameters/>
      <execParameters/>
      <opTree>
         <Emit>
            <VA>4</VA>
            <est>
               <rowCnt>1</rowCnt>
               <lio>0</lio>
               <pio>0</pio>
               <rowSz>10</rowSz>
            </est>
            <arity>1</arity>
            <HashDistinct>
               <VA>3</VA>
               <est>
                  <rowCnt>1</rowCnt>
                  <lio>5</lio>
```

```
      <pio>0</pio>
      <rowSz>10</rowSz>
   </est>
   <arity>1</arity>
   <WorkTable>
      <wtObjName>WorkTable1</wtObjName>
   </WorkTable>
   <NestLoopJoin>
      <VA>2</VA>
      <est>
         <rowCnt>1</rowCnt>
         <lio>0</lio>
         <pio>0</pio>
         <rowSz>10</rowSz>
      </est>
      <arity>2</arity>
      <TableScan>
         <VA>0</VA>
         <est>
            <rowCnt>1</rowCnt>
            <lio>1</lio>
            <pio>0.9999995</pio>
            <rowSz>6</rowSz>
         </est>
         <varNo>0</varNo>
         <objName>t2</objName>
         <scanType>TableScan</scanType>
         <scanOrder> ForwardScan </scanOrder>
         <positioning> StartOfTable </positioning>
         <scanCoverage> NonCovered </scanCoverage>
         <dataIOSizeInKB>16</dataIOSizeInKB>
         <dataBufReplStrategy> LRU </dataBufReplStrategy>
      </TableScan>
      <IndexScan>
         <VA>1</VA>
         <est>
            <rowCnt>1</rowCnt>
            <lio>0</lio>
            <pio>0</pio>
            <rowSz>10</rowSz>
         </est>
         <varNo>1</varNo>


         <objName>t1</objName>
         <scanType>IndexScan</scanType>
         <indName>i1t1</indName>
```

```
                        <indId>1</indId>
                        <scanOrder> ForwardScan </scanOrder>
                        <positioning> ByKey </positioning>
                        <scanCoverage> NonCovered </scanCoverage>
                        <perKey>
                           <keyCol>c1</keyCol>
                           <keyOrder> Ascending </keyOrder>
                        </perKey>
                        <dataIOSizeInKB>16</dataIOSizeInKB>
                        <dataBufReplStrategy> LRU </dataBufReplStrategy>
                     </IndexScan>
                  </NestLoopJoin>
               </HashDistinct>
            </Emit>
            <est>
               <totalLio>6</totalLio>
               <totalPio>0.9999995</totalPio>
            </est>
            <act>
               <totalLio>0</totalLio>
               <totalPio>0</totalPio>
            </act>
         </opTree>
      </plan>
   </query>
```

用法

- 在使用 show_cached_plan_in_xml 之前先启用语句高速缓存。
- 仅将 show_cached_plan_in_xml 用于高速缓存语句。
- 如果计划正在使用中，便不会进行打印。状态为 `available`（可用）的计划会输出计划详细信息。状态为 `in use`（使用中）的计划仅显示进程 ID。
- 下表给出了针对 *level_of_detail* 值而显示的 show_cached_plan_in_xml 部分：

表 2-6：详细程度

level_of_detail	parameter	opTree	execTree
0（缺省值）	X	X	
1	X		
2		X	
3			X
4		X	X
5	X		X
6	X	X	X

权限	对 show_cached_plan_in_xml 的权限检查因您的细化权限设置而异。
细化权限已启用	在启用细化权限的情况下，您必须是具有 mon_role 的用户，或者具有 monitor qp performance 权限才能执行 show_cached_plan_in_xml。
细化权限已禁用	在禁用细化权限的情况下，您必须是具有 mon_role 或 sa_role 的用户才能执行 show_cached_plan_in_xml。

show_cached_text

说明　显示高速缓存语句的 SQL 文本。

语法　show_cached_text(*statement_id*)

参数　*statement_id*

该语句的 ID。派生自 monCachedStatement 的 SSQLID 列。

示例　显示 monCachedStatement 的内容，然后使用 show_cached_text 函数显示 SQL 文本：

```
select InstanceID, SSQLID, Hashkey, UseCount, StmtType
from monCachedStatement
InstanceID  SSQLID     Hashkey      UseCount  StmtType
----------  ----------  ----------  ----------  --------
0           329111220  1108036110           0       2
0           345111277  1663781964           1       1

select show_cached_text(329111220)
-------------------------------------
select id from sysroles
```

用法

- show_cached_text 最多可显示 16K 的 SQL 文本，并会截断超过 16K 的文本。将 show_cached_plan_in_xml 用于超过 16K 的文本。
- show_cached_text 将返回 varchar 数据类型。

show_cached_text_long

说明 显示大于 16K 的高速缓存语句的 SQL 文本。

语法 show_cached_text_long(*statement_id*)

参数 *statement_id*

该语句的 ID。派生自 monCachedStatement 的 SSQLID 列。

示例 从 monCachedStatement 监控表选择 SQL 文本 （为便于阅读，已将结果集缩短）：

```
select show_cached_text_long(SSQLID) as sql_text, StatementSize from
monCachedStatement
sql_text
StatementSize
------------------------------------------------------------------------------
------------------------------------------
SELECT first_column ......
188888
```

用法

- show_cached_text_long 最多显示 2M 的 SQL 文本。
- show_cached_text_long 将返回 text 数据类型。
- 使用 show_cached_text_long 需要您将 set textsize *value* 配置为较大的值。如果您配置的值过小， Adaptive Server 客户端 （如 isql）会将 show_cached_text_long 结果集截断。

show_dynamic_params_in_xml

说明　以 XML 格式返回动态 SQL 查询 （预准备语句）的参数信息。

语法　show_dynamic_params_in_xml(*object_id*)

参数　*object_id*

要调查的动态 SQL 轻量存储过程的 ID。通常返回 *@@plwpid* 全局变量的值。

示例　对于此示例，首先查找对象 ID：

```
select @@plwpid
----------------
707749902
```

然后将该 ID 用作 show_dynamic_params_in_xml 的输入参数：

```
select show_dynamic_params_in_xml(707749902)

<?xml version="1.0" encoding="UTF-8"?>
< query>
    <parameter>
        <number>1</number>
        <type>INT</type>
        <column>tab.col1</column>
    </parameter>
</query>
```

参数	值	定义
number	1	动态参数位于语句的第一个位置
type	INT	表使用 int 数据类型
column	tab.col1	查询使用 tab 表的 col1 列

用法

- show_dynamic_params_in_xml 允许 where 子句、update 的 set 子句以及 insert 的 *values* 列表中有动态参数。
- 对于 where 子句， show_dynamic_params_in_xml 根据涉及含有列的表达式、关系型运算符以及含有参数的表达式的最小子目录树来决定关联。例如：

  ```
  select * from tab where col1 + 1 = ?
  ```

  如果查询没有子目录树， show_dynamic_params_in_xml 会省略 <column> 元素。例如：

  ```
  select * from tab where ?< 1000
  ```

- 对于涉及多个列的表达式，show_dynamic_params_in_xml 会选择它遇到的第一个列：

  ```
  delete tab where col1 + col2 > ?
  ```

- 关联对于 update . . . set 语句是明确的。例如：

  ```
  update tab set col1 = ?
  ```

show_plan

说明

针对指定的服务器进程（目标进程）和 SQL 语句检索查询计划。sp_showplan 多次调用该函数是因为每次调用内置函数只能返回一个值，而 sp_showplan 必须将多个值返回至客户端。

语法

show_plan(*spid*, *batch_id*, *context_id*, *statement_number*)

参数

spid
是任何用户连接的进程 ID。

batch_id
批处理的唯一编号。

context_id
每个过程（或触发器）的唯一编号。

statement_number
批处理中当前语句的编号。

示例

在下面示例中， show_plan 将执行以下操作：

- 验证 sp_showplan 无法验证的参数值。如果用户在不设置参数值的情况下执行了 sp_showplan，将传入 -1。只有 *spid* 值是必需的。
- 如果仅收到一个进程 ID，则 show_plan 将通过 sp_showplan 进行的连续三次调用返回批处理 ID、环境 ID 和语句编号。
- 查找指定 SQL 语句编号的 E_STMT 指针。
- 针对该语句检索目标进程的查询计划。对于并行工作进程，可检索等同的父计划以减少对性能的影响。
- 与目标进程同步查询计划访问。

```
if (@batch_id is NULL)
  begin
    /* Pass -1 for unknown values.*/
    select @return_value = show_plan(@spid, -1, -1, -1)
    if (@return_value < 0)
        return (1)
    else
    select @batch_id = @return_value

    select @return_value = show_plan(@spid, @batch_id, -1, -1)
    if (@return_value < 0)
        return (1)
    else
      select @context_id = @return_value
```

```
    select @return_value = show_plan(@spid, @batch_id, @context_id, -1)
    if (@return_value < 0)
        return (1)
    else
    begin
       select @stmt_num = @return_value
      return (0)
    end
end
```

如示例所示，调用三次 show_plan 以得到 *spid*：

- 第一次返回批处理 ID
- 第二次返回环境 ID
- 第三次显示查询计划，并返回当前语句编号。

用法

对于未能正确执行的语句，您可以通过变更优化程序设置或指定抽象计划来更改计划。

如果将现有 show_plan 参数中的第一个 int 变量指定为“-”，则 show_plan 会将第二个参数视为 SSQLID。

注释 语句高速缓存中的单个条目可以与多个可能不同的 SQL 计划相关联。 show_plan 只显示这些计划中的一个。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

另请参见

过程 sp_showplan

show_role

说明　显示当前登录名的目前处于活动状态的系统定义角色。

语法　show_role()

参数　无

示例

示例 1

```
select show_role()

sa_role sso_role oper_role replication_role
```

示例 2

```
if charindex("sa_role", show_role()) >0
begin
    print "You have sa_role"
end
```

用法

- show_role 是一个系统函数，只要登录名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启用的角色，而不显示通过 setuser 充当的用户。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 show_role。

另请参见

命令　alter role、 create role、 drop role、 grant、 set、 revoke

文档　*《Transact-SQL 用户指南》*

函数　proc_role、 role_contain

系统过程　sp_activeroles、 sp_displayroles、 sp_role

show_sec_services

说明	列出可供会话使用的安全服务。
语法	show_sec_services()
参数	无

示例	演示用户的当前会话正在加密数据并执行重放检测检查：
	`select show_sec_services()`
	`encryption, replay_detection`

用法	• 使用 show_sec_services 可列出在会话过程中处于启用状态的安全服务。 • 如果没有启用安全服务， show_sec_services 就会返回 NULL。
标准	符合 ANSI SQL 的级别 Transact-SQL 扩展。
权限	任何用户都可以执行 show_sec_services。
另请参见	**函数** is_sec_service_on

sign

说明　返回指定值的符号：1 （正）、 0 或 -1 （负）。

语法　sign(*numeric*)

参数　*numeric*

是任意精确数值 （numeric、 dec、 decimal、 tinyint、 smallint、 int 或 bigint）、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形式。

示例

示例 1

```
select sign(-123)
-----------
         -1
```

示例 2

```
select sign(0)
-----------
          0
```

示例 3

```
select sign(123)
-----------
          1
```

用法

- sign 是一个数学函数，它返回正 (1)、零 (0) 或负 (-1)。
- 结果与数值表达式属于同一类型并且具有相同的精度和标度。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 sign。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　abs、 ceiling、 floor、 round

sin

说明

返回指定角 （以弧度表示）的正弦。

语法

sin(*approx_numeric*)

参数

approx_numeric

是任何近似的数值型 （float、 real 或 double precision）列名、变量或常量表达式。

示例

```
select sin(45)
--------------------
            0.850904
```

用法

sin 是一个数学函数，它返回指定角 （以弧度表示）的正弦。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 sin。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 cos、 degrees、 radians

sortkey

说明

生成可用于根据归类行为对结果进行排序的值，该函数允许使用缺省设置（基于拉丁字符的字典排序顺序以及区分大小写或变音）以外的字符归类行为。

语法

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

参数

char_expression

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

uchar_expression

类型为 unichar 或 univarchar 的字符型列名、变量或常量表达式。

collation_name

是加有引号的字符串或字符变量，它指定要使用的归类。第 264 页上的表 2-8 显示了有效值。

collation_ID

是一个整数常量或变量，它指定要使用的归类。第 264 页上的表 2-8 显示了有效值。

示例

示例 1 显示基于欧洲语言字典顺序的排序：

```
select * from cust_table where cust_name like "TI%" order by
    (sortkey(cust_name, "dict")
```

示例 2 显示基于简体中文拼音顺序的排序：

```
select *from cust_table where cust name like "TI%" order by
    (sortkey(cust-name, "gbpinyin")
```

示例 3 显示使用内嵌选项且基于欧洲语言字典顺序的排序：

```
select *from cust_table where cust_name like "TI%" order by cust_french_sort
```

示例 4 显示使用已存在的关键字且基于简体中文拼音顺序的排序：

```
select * from cust_table where cust_name like "TI%" order by
    cust_chinese_sort.
```

用法

- sortkey 是一个系统函数，它生成用于根据归类行为对结果进行排序的值。此函数允许使用缺省设置（基于拉丁字符的字典排序顺序以及大小写或变音区分）以外的字符归类行为。返回值为 varbinary 数据类型的值，该值包含从 sortkey 函数返回的输入字符串的编码归类信息。

例如，可将 sortkey 返回的值存储在具有源字符串的列中。若要按期望的顺序检索字符数据，对于包含 sortkey 运行结果的列，应在 select 语句中包含 order by clause。

sortkey 保证其为给定的一套归类标准返回的值能够用于对 varbinary 数据类型执行的二进制比较。

- sortkey 可为每个输入字符最多生成 6 个字节的归类信息。因此，使用 sortkey 的结果可能超过 varbinary 数据类型的长度限制。如果发生这种情况，结果将被截断，以符合限制要求。由于此限制取决于服务器的逻辑页大小，因此截断操作会删除每个输入字符的结果字节，直到最终得到的字符串小于 DOL 锁定表和 APL 表中的以下值：

表 2-7：行和列的最大长度 — APL 和 DOL 表

锁定方案	页大小	最大行长度	最大列长度
APL 表	2K （2048 字节）	1962	1960 字节
	4K （4096 字节）	4010	4008 字节
	8K （8192 字节）	8106	8104 字节
	16K （16384 字节）	16298	16296 字节
DOL 表	2K （2048 字节）	1964	1958 字节
	4K （4096 字节）	4012	4006 字节
	8K （8192 字节）	8108	8102 字节
	16K （16384 字节）	16300	16294 字节 如果表不包含任何可变长度的列
	16K （16384 字节）	16300 （取决于 varlen 的最大起始偏移 = 8191）	8191-6-2 = 8183 字节 如果表至少包含一个可变长度列。 *

* 此大小包含六个字节的行开销和两个字节的行长度字段。

如果发生此情况， Adaptive Server 将发出警告消息，但包含 sortkey 函数的查询或事务仍将继续运行。

- *char_expression* 或 *uchar_expression* 必须由用服务器的缺省字符集进行编码的字符组成。
- *char_expression* 或 *uchar_expression* 可以是空字符串。如果是空字符串，则 sortkey 返回零长度 varbinary 值，并为空字符串存储一个空白。

  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* 或 *uchar_expression* 是 NULL，则 sortkey 返回 NULL 值。

- 如果 unicode 表达式没有指定的排序顺序，则 Adaptive Server 使用 binary 排序顺序。
- 如果未指定 *collation_name* 或 *collation_ID* 的值，sortkey 将假定为二进制归类。
- 从 sortkey 函数生成的二进制值可能因 Adaptive Server 主要版本的不同而不同（例如，12.0 与 12.5、12.9.2 与 12.0 等都是不同的主要版本）。如果是升级到 Adaptive Server 的当前版本，则应在进行任何二进制比较之前，重新生成键并重新填充影子列。

注释　从版本 12.5 升级到 12.5.0.1 不需要执行该步骤，如果不重新生成键，Adaptive Server 也不会生成任何错误或警告消息。虽然涉及影子列的查询应能正常运行，但比较结果可能与升级之前的服务器比较结果不同。

归类表

有两种归类表可用于进行多语种排序：

1　通过 sortkey 函数创建的一种“内置”归类表。可以使用归类名或归类 ID 指定内置表。

2　另一种是使用 Unilib 库排序函数创建的外部归类表。必须使用归类名指定外部表。这些文件位于 *$SYBASE/collate/unicode* 中。

这两种方法同样有效，区别在于“内置”表与 Adaptive Server 数据库联系紧密，而外部表则不然。如果使用 Adaptive Server 数据库，内置表可提供最佳性能。这两种方法都可以处理英语、欧洲语言和亚洲语言的任意混合。

使用 sortkey 的方法有两种：

1　内嵌 — 这种方法将 sortkey 作为 order by clause 的一部分使用，对改进现有应用程序以及最大程度地减少更改很有用。但是，这种方法会动态地生成排序键，因此，在超过 1000 条记录的大数据集上无法获得最佳性能。

2　预存在键 — 一旦表中添加了要求进行多语种排序的新记录（如新客户名），该方法就会调用 sortkey。必须在数据库，最好是在同一表中对影子列（binary 或 varbinary 类型）进行设置，每列对应所需的一种排序顺序（如法语，中文等）。当查询要求对输出进行排序时，order by clause 将使用其中一个影子列。在这种方法中，由于键都已生成并存储，而且只需根据它们的二进制值就可进行快速比较，因此可提供最佳性能。

您可查看可用的归类规则列表。打印列表的方法为：执行 sp_helpsort，或在 syscharsets 中查询并选择 name、 id 和 description （type 介于 2003 和 2999 之间）。

- 表 2-8 列出了 collation_name 和 collation_ID 的有效值。

表 2-8：归类名称和 ID

说明	归类名称	归类 ID
缺省的 Unicode 多语种	缺省	20
泰文字典顺序	thaidict	21
ISO14651 标准	iso14651	22
UTF-16 排序 — 与 UTF-8 二进制排序匹配	utf8bin	24
CP 850 方案 — 没有变音	altnoacc	39
CP 850 方案 — 小写优先	altdict	45
CP 850 西欧 — 没有大小写优先级	altnocsp	46
CP 850 斯堪的纳维亚文 — 字典排序	scandict	47
CP 850 斯堪的纳维亚文 — 不区分大小写，具有优先级	scannocp	48
GB 拼音	gbpinyin	不可用
二进制排序	binary	50
Latin-1 英文、法文、德文字典	dict	51
Latin-1 英文、法文、德文，没有大小写	nocase	52
Latin-1 英文、法文、德文，没有大小写优先级	nocasep	53
Latin-1 英文、法文、德文，没有变音	noaccent	54
Latin-1 西班牙文字典	espdict	55
Latin-1 西班牙文，没有大小写	espnocs	56
Latin-1 西班牙文，没有变音	espnoac	57
ISO 8859-5 俄文字典	rusdict	58
ISO 8859-5 俄文，没有大小写	rusnocs	59
ISO 8859-5 古斯拉夫文字典	cyrdict	63
ISO 8859-5 古斯拉夫文，没有大小写	cyrnocs	64
ISO 8859-7 希腊文字典	elldict	65
ISO 8859-2 匈牙利文字典	hundict	69
ISO 8859-2 匈牙利文，没有变音	hunnoac	70
ISO 8859-2 匈牙利文，没有大小写	hunnocs	71
ISO 8859-9 土耳其文字典	turdict	72
ISO 8859-9 土耳其文，没有变音	turknoac	73
ISO 8859-9 土耳其文，没有大小写	turknocs	74
CP932 二进制排序	cp932bin	129

说明	归类名称	归类 ID
中文拼音排序	dynix	130
GB2312 二进制排序	gb2312bn	137
通用古斯拉夫文字典	cyrdict	140
土耳其文字典	turdict	155
EUCKSC 二进制排序	euckscbn	161
中文拼音排序	gbpinyin	163
俄文字典排序	rusdict	165
SJIS 二进制排序	sjisbin	179
EUCJIS 字典排序	eucjisbn	192
BIG5 二进制排序	big5bin	194
Shift-JIS 二进制顺序	sjisbin	259

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 sortkey。

另请参见　**函数**　compare

soundex

说明

对于由有效单字节或双字节罗马字母的邻接序列所组成的字符串，它将返回四个字符的 soundex 代码。

语法

soundex(*char_expr* | *uchar_expr*)

参数

char_expr

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

uchar_expr

类型为 unichar 或 univarchar 的字符型列名、变量或常量表达式。

示例

```
select soundex ("smith"), soundex ("smythe")
----- -----
S530  S530
```

用法

- soundex 是一个字符串函数，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，用于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* 或 *uchar_expr* 是 NULL，则返回 NULL。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 soundex。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 difference

space

说明　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*

是任意整数（tinyint、smallint 或 int）类型的列名、变量或常量表达式。

示例

```
select "aaa", space(4), "bbb"

--- ---- ---
aaa      bbb
```

用法　space 是一个字符串函数，它返回具有指定数量的单字节空格的字符串。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 space。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　isnull、 rtrim

spid_instance_id

说明

（仅限集群环境）返回运行指定进程 ID (spid) 的实例 ID。

语法

spid_instance_id(*spid_value*)

参数

spid_value
请求其实例 ID 的 spid 号

示例

返回运行进程 ID 号 27 的实例的 ID：

```
select spid_instance_id(27)
```

用法

- 如果未包括 spid 值，则 spid_instance_id 返回 NULL。
- 如果输入的进程 ID 值无效或不存在，则 spid_instance_id 返回 NULL。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 spid_instance_id。

square

说明　计算表示为 float 的指定值的平方值。

语法　square(*numeric_expression*)

参数　*numeric_expression*
是 float 类型的数值表达式。

示例　**示例 1** 从整数列返回平方值：

```
select square(total_sales)from titles
------------
16769025.00000
15023376.00000
350513284.00000
...
16769025.00000
(18 row(s) affected)
```

示例 2 从货币列返回平方值：

```
select square(price) from titles
-----------
399.600100
142.802500
8.940100
NULL
...
224.700100
(18 row(s) affected)
```

用法　此函数等同于 power(*numeric_expression*,2)，但它返回 float 类型而不返回 int 类型。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 square。

另请参见　**函数**　power

数据类型　exact_numeric、 approximate_numeric、 money、 float

sqrt

说明

计算指定数字的平方根。

语法

sqrt(*approx_numeric*)

参数

approx_numeric

是求值结果为正数的任何近似的数值型（float、real 或 double precision）列名、变量或常量表达式。

示例

```
select sqrt(4)

 2.000000
```

用法

- sqrt 是一个数学函数，它返回指定值的平方根。
- 如果试图选择负数的平方根，Adaptive Server 将返回以下错误消息：

```
Domain error occurred.
```

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 sqrt。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 power

stddev

说明

计算包含数值表达式的样本的标准偏差，数据类型为 double。

注释　stddev 和 stdev 是 stddev_samp 的别名。有关详细信息，请参见 第 275 页的 stddev_samp。

stdev

说明

计算包含数值表达式的样本的标准偏差，数据类型为 double。

注释 stddev 和 stdev 是 stddev_samp 的别名。有关详细信息，请参见 第 275 页的 stddev_samp。

stdevp

说明

计算包含数值表达式的总体的标准偏差，数据类型为 double。

注释　stdevp 是 stddev_pop 的别名。有关详细信息，请参见 第 274 页的 stddev_pop。

stddev_pop

说明

计算包含数值表达式的总体的标准偏差，数据类型为 double。 stdevp 是 stddev_pop 的别名，并使用相同的语法。

语法

stddev_pop ([all | distinct] *expression*)

参数

all

将 stddev_pop 应用于所有值。 all 是缺省值。

distinct

在应用 stddev_pop 之前消除重复值。

expression

该表达式 （通常为列名）用于对一组行计算其基于总体的标准偏差。

示例

下列语句列出了 pubs2 数据库中每种类型书籍预付款的平均和标准偏差。

```
select type, avg(advance) as "avg", stddev_pop(advance)
  as "stddev" from titles group by type order by type
```

用法

计算对每个组行求值的所提供值表达式的总体标准偏差 （如果指定了 distinct，则已删除在复制之后仍保留的各行），其定义为总体方差的平方根。

图 2-1：总体相关的统计集合函数的公式

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

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

σ^2 = 方差

n = 总数

$\mu = x_i$ 的均值

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 stddev_pop。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 stddev_samp、 var_pop、 var_samp

stddev_samp

说明

计算包含数值表达式的样本的标准偏差，数据类型为 double。 stdev 和 stddev 是 stddev_samp 的别名，并使用相同的语法。

语法

stddev_samp ([all | distinct] *expression*)

参数

all
　将 stddev_samp 应用于所有值。 all 为缺省值。

distinct
　在应用 stddev_samp 之前消除重复值。

expression
　是任意数值数据类型 （float、 real 或 double precision）表达式。

示例

下列语句列出了 pubs2 数据库中每种类型书籍预付款的平均和标准偏差。

```
select type, avg(advance) as "avg",
  stddev_samp(advance) as "stddev" from titles
  where total_sales > 2000 group by type order by type
```

用法

计算对每个组行求值的所提供值表达式的样本标准偏差 （如果指定了 distinct，则已删除在复制之后仍保留的各行），其定义为样本方差的平方根。

图 2-2：样本相关的统计集合函数的公式

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

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

s^2 = 方差
n = 样本数
$\bar{x} = x_i$ 的均值

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 stddev_samp。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　stddev_pop、 var_pop、 var_samp

str

说明

返回指定数字的等值字符，并用字符或数值将输出填补到指定长度。

语法

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

参数

approx_numeric

是任何近似的数值型（float、 real 或 double precision）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以及空白）。缺省值为 10。

decimal

设置要返回的小数位数。缺省值为 0。还可以用于通过字符或数值将输出填补到指定长度。

当您将字符或数值指定为文字字符串时，该字符或数值将被用来填充字段。当您指定数字值时，它会设置小数位数。缺省值为 0。如果未设置 *decimal*，会按 *length* 指定的值用空白来填充字段。

示例

示例 1 如果 *decimal* 设置为字符串文字 “0”，则会用 0 将字段填充到 10 个空格的长度。

```
select str(5,10,'0')
----------------
  0000000005
```

示例 2 如果 *decimal* 为数值 5，则小数位数会设置为 5。

```
select str(5,10,5)
----------------
   5.00000
```

示例 3 如果 *decimal* 设置为字符 “_”，则会保留原始值，并用指定的字符将字段填充到 16 个空格的长度。

```
select str(12.34500,16,'_')
----------------
________12.34500
```

示例 4 如果不设置 *decimal*，则会将浮点数设置为零个小数位数，并用空白将字段填充到 16 个空格的长度。

```
select str(12.34500e,16)
----------------
              12
```

示例 5 如果 *decimal* 设置为数值，则会将浮点数处理成 7 个小数位数，并用空白将字段填充到 16 个空格的长度。

```
select str(12.34500e,16,7)
----------------
      12.3450000
```

示例 6 通过以下示例指定前缀字符并将浮点数处理成指定的小数位数：

```
select str(convert(numeric(10.2),12.34500e),16,'-')
----------------
-----------12.35

select str(convert(numeric(10,8),12.34500e),16,'-')
----------------
-----12.34500000
```

用法

- *length* 和 *decimal* 是可选的，但如果使用它们，就必须是正整数。 str 将舍入数字的小数部分，以使结果不超出指定长度。该长度必须足以容纳小数点，如果数字为负，则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 将返回一行具有指定长度的星号。例如：

  ```
  select str(123.456, 2, 4)
  --
  **
  ```

- 如果 *approx_numeric* 为 NULL，则返回 NULL。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 str。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　abs、 ceiling、 floor、 round、 sign

str_replace

说明

将第一个字符串表达式 (*string_expression1*) 中出现的第二个字符串表达式 (*string_expression2*) 的所有实例替换为第三个表达式 (*string_expression3*)。

语法

str_replace("*string_expression1*", "*string_expression2*", "*string_expression3*")

参数

string_expression1

是源字符串或要搜索的字符串表达式，表示为 char、varchar、unichar、univarchar、 varbinary 或 binary 数据类型。

string_expression2

是模式字符串或要在第一个表达式 (*string_expression1*) 中查找的字符串表达式。*string_expression2* 表示为 char、varchar、unichar、univarchar、varbinary 或 binary 数据类型。

string_expression3

是替换字符串表达式，表示为 char、varchar、unichar、univarchar、binary 或 varbinary 数据类型。

示例

示例 1 将字符串 *cdefghi* 中的字符串 *def* 替换为 *yyy*。

```
str_replace("cdefghi","def","yyy")
-------------
cyyyghi
(1 row(s) affected)
```

示例 2 将所有空格替换为 “toyota”。

```
select str_replace("chevy, ford, mercedes",
"","toyota")
----------
chevy,toyotaford,toyotamercedes
(1 row(s) affected)
```

注释 Adaptive Server 将空字符串常量自动转换为包含 1 个空格的字符串，以便将该字符串与空值区分开。

示例 3 返回 “abcghijklm”：

```
select str_replace("abcdefghijklm", "def", NULL)

----------
abcghijklm
(1 row affected)
```

用法

- 如果 *string_expression*（1、2 或 3）为 char 或 varchar，则返回 varchar 数据。

- 如果 *string_expression*（1、2 或 3）为 unichar 或 univarchar，则返回 univarchar 数据。
- 如果 *string_expression*（1、2 或 3）为 binary 或 varbinary，则返回 varbinary 数据。
- 所有参数必须共享同一数据类型。
- 如果以上三个参数中任一参数是 NULL，该函数就会返回空。

  str_replace 在第三个参数中接受 NULL，将其视为尝试用 NULL 替换 *string_expression2*，有效地将 str_replace 转换成“字符串切除”操作。

  例如，下面的命令返回 “abcghijklm”：

  ```
  str_replace("abcdefghijklm", "def", NULL)
  ```

- 结果的长度可能会有所不同，具体取决于编译表达式时对参数值的了解情况。如果所有参数都是带有已知常量值的变量， Adaptive Server 将按以下方式计算结果：

  ```
  result_length = ((s/p)*(r-p)+s)
  where
  s = length of source string
  p = length of pattern string
  r = length of replacement string
  if (r-p) <= 0, result length = s
  ```

- 如果源字符串 (*string_expression1*) 是列， *string_expression2* 和 *string_expression3* 是在编译时已知的常量值，Adaptive Server 将使用上述公式计算结果长度。
- 如果在编译表达式时参数值是未知的，因而 Adaptive Server 无法计算结果长度，那么，使用的结果长度就会是 255，除非打开跟踪标志 244。打开该标志后，结果长度就是 16384。
- result_len 从不超过 16384。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 str_replace。

另请参见　**数据类型**　char、 varchar、 binary、 varbinary、 unichar、 univarchar

函数　length

strtobin

说明

将字母数字字符序列转换为等值的十六进制数字。

语法

select strtobin("*string of valid alphanumeric characters*")

参数

string of valid alphanumeric characters
是有效的字母数字字符串，其中包含 [1 - 9]、[a - f] 和 [A - F]。

示例

示例 1 将字母数字字符串 “723ad82fe”转换为十六进制数字序列：

```
select strtobin("723ad82fe")
go

-----------------------------------
0x0723ad82fe
```

字母数字字符串及其等值的十六进制数字在内存中的表示形式为：

字母数字字符串 （9 字节）

0	7	2	3	a	d	8	2	f	e

十六进制数字 （5 字节）

0	7	2	3	a	d	8	2	f	e

该函数从右向左处理字符。在此示例中，输入的字符数是奇数。因此，十六进制序列带有前缀 “0”并且会反映在输出中。

示例 2 将名为 *@str_data* 的局部变量的字母数字字符串转换为与 “723ad82fe”等值的十六进制数字序列：

```
declare @str_data varchar(30)
select @str_data = "723ad82fe"
select strtobin(@str_data)
go

----------
0x0723ad82fe
```

用法

- 输入中的任何无效字符均会导致输出为 NULL。
- 十六进制数字的输入序列必须带有前缀 “0x”。
- NULL 输入会生成 NULL 输出。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 strtobin。

另请参见

函数 bintostr

stuff

说明

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数量的字符，然后将这些字符替换为另一个字符串。

语法

stuff(*char_expr1* | *uchar_expr1*, *start*, *length*, *char_expr2* | *uchar_expr2*)

参数

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

uchar_expr1
类型为 unichar 或 univarchar 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
另一种类型为 char、varchar、nchar 或 nvarchar 的字符型列名、变量或常量表达式。

uchar_expr2
另一种类型为 unichar 或 univarchar 的字符型列名、变量或常量表达式。

示例

示例 1

```
select stuff("abc", 2, 3, "xyz")

----
axyz
```

示例 2

```
select stuff("abcdef", 2, 3, null)

go
---
aef
```

示例 3

```
select stuff("abcdef", 2, 3, "")

----
a ef
```

用法

- stuff 是一个字符串函数，它从 *char_expr1* 或 *uchar_expr1* 的 *start* 开始删除数量为 *length* 的字符，然后在 *char_expr1* 或 *uchar_expr2* 的 *start* 处插入 *char_expr2* 或 *uchar_expr2* 。有关字符串函数的常规信息，请参见 *《Transact-SQL 用户指南》*。
- 如果起始位置或长度为负，则返回 NULL 字符串。如果起始位置为零或超出 *expr1* 的长度，将返回 NULL 字符串。如果要删除的长度超出 *expr1* 的长度，则会删除 *expr1* 的全部字符 （请参见示例 1）。
- 如果起始位置位于代理对的中央，则 start 将被调整为减小一。如果起始长度位置位于代理对的中央，则 length 将被调整为减小一。
- 若要使用 stuff 删除某个字符，请将 *expr2* 替换为 NULL （而不是空引号）。使用 “ ” 指定空字符时，会将其替换为空格 （请参见示例 2 和 3）。
- 如果 *char_expr1* 或 *uchar_expr1* 为 NULL，则 stuff 返回 NULL。如果 *char_expr1* 或 *uchar_expr1* 是字符串值，而且 *char_expr2* 或 *uchar_expr2* 为 NULL，则 stuff 不将删除的字符替换为任何其它内容。
- 如果将 varchar 表达式用作一个参数，并将 unichar 表达式用作另一个参数，则 varchar 表达式将会隐式地转换为 unichar （可能会发生截断）。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 stuff。

另请参见

函数 replicate、 substring

substring

说明

返回通过从另一字符串中提取指定数量的字符所形成的字符串。

语法

substring(*expression*, *start, length*)

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、nchar、unichar、 varchar、 univarchar 或 nvarchar 数据， binary 或 varbinary。

start

指定位于子串开始处的字符位置。

长度

指定子串中的字符数。

示例

示例 1 显示每个作者的姓和名，如 “Bennet A”：

```
select au_lname, substring(au_fname, 1, 1)
from authors
```

示例 2 将作者的姓转换为大写形式，然后显示前三个字符：

```
select substring(upper(au_lname), 1, 3)
from authors
```

示例 3 将 pub_id 和 title_id 并置，然后显示所生成字符串的前六个字符：

```
select substring((pub_id + title_id), 1, 6)
from titles
```

示例 4 从二进制字段中提取后四位，其中的每个位置都表示两个二进制位：

```
select substring(xactid,5,2)
from syslogs
```

用法

- substring 是一个字符串函数，它返回字符或二进制字符串的一部分。有关字符串函数的常规信息，请参见 *《Transact-SQL 用户指南》*。
- 如果 substring 的第二个参数是 NULL，则结果为 NULL。如果 substring 的第一个或第三个参数是 NULL，则结果为空白。
- 如果从 *uchar_expr1* 开始处算起的起始位置位于代理对的中央， *start* 将被调整为减少一。如果从 *uchar_expr1* 开始处算起的起始长度位置位于代理对的中央， *length* 将被调整为减少一。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 substring。

另请参见

函数 charindex、 patindex、 stuff

sum

说明

返回值的总和。

语法

sum([all | distinct] *expression*)

参数

all

将 sum 应用到所有值。 all 为缺省值。

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。使用集合时，表达式通常是列名。有关详细信息，请参见第 339 页上的 “表达式”。

示例

示例 1 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一个汇总值：

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

示例 2 当用于 group by 子句时，集合函数将为每个组（而非整个表）生成一个值。此语句将为每类书籍生成汇总值：

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

示例 3 将 titles 表按出版商分组，并且只包括那些预付款总额超过 25,000 美元且书籍平均价格高于 15 美元的出版商所形成的组。

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

用法

- sum 是一个集合函数，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。
- 在对整数求和时， Adaptive Server 将结果视为 int 值，即使列的数据类型为 smallint 或 tinyint 也是如此。在对 bigint 数据求和时， Adaptive Server 将结果视为 bigint。如果避免在 DB-Library 程序中出现溢出错误，应对所有表示求平均值或求和结果的变量进行相应的声明。
- 不能将 sum 与二进制数据类型一起使用。
- 此函数只定义了数值类型，若与 Unicode 表达式一起使用会产生错误。

标准	符合 ANSI SQL 的级别 Transact-SQL 扩展。
权限	任何用户都可以执行 sum。
另请参见	**命令** compute clause、 group by and having clauses、 select、 where clause **文档** *《Transact-SQL 用户指南》* **函数** count、 max、 min

suser_id

说明

从 syslogins 表中返回服务器用户的 ID 号。

语法

suser_id([*server_user_name*])

参数

server_user_name

是 Adaptive Server 登录名。

示例

示例 1

```
select suser_id()

------
     1
```

示例 2

```
select suser_id("margaret")

------
     5
```

用法

- suser_id 是一个系统函数，它从 syslogins 中返回服务器用户的 ID 号。有关系统函数的常规信息，请参见 *《Transact-SQL 用户指南》*。
- 若要从 sysusers 表查找特定数据库中的用户 ID，请使用 user_id 系统函数。
- 如果未提供 *server_user_name*，suser_id 就会返回当前用户的服务器 ID。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 suser_id。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 suser_name、 user_id

suser_name

说明

返回当前服务器用户的名称，或已指定服务器 ID 的用户的名称。

语法

suser_name([*server_user_id*])

参数

server_user_id

是 Adaptive Server 用户 ID。

示例

示例 1

```
select suser_name()

------------------------------
sa
```

示例 2

```
select suser_name(4)

------------------------------
margaret
```

用法

suser_name 是一个系统函数，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*，则 suser_name 返回当前用户的名称。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 suser_name。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　suser_id、 user_name

syb_quit

说明 终止连接。

语法 syb_quit()

示例 终止在其中执行该函数且返回错误消息的连接。

```
select syb_quit()

-------------
CT-LIBRARY error:
    ct_results():network packet layer:
internal net library error: 由于断开连接，已终止 Net-
Library 操作
```

用法 可以使用 syb_quit 在 isql 预处理程序命令 exit 导致错误时终止脚本。

权限 任何用户都可以执行 syb_quit。

syb_sendmsg

说明

（仅限于 UNIX）将消息发送到用户数据报协议 (UDP) 端口。

语法

syb_sendmsg *ip_address*, *port_number*, *message*

参数

ip_address
是运行着 UDP 应用程序的计算机的 IP 地址。

port_number
是 UDP 端口的端口号。

message
是要发送的消息。它最长可达 255 个字符。

示例

示例 1 将消息 “Hello” 发送到 IP 地址为 120.10.20.5 的端口 3456：

```
select syb_sendmsg("120.10.20.5", 3456, "Hello")
```

示例 2 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：

```
declare @msg varchar(255)
    select @msg = "Message to send"
    select syb_sendmsg (ip_address, portnum, @msg)
    from sendports
    where username = user_name()
```

用法

- 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。
- 使用 syb_sendmsg 时不执行安全检查。 Sybase 强烈建议您不要使用 syb_sendmsg 通过网络发送敏感信息。用户启用此功能，即表示其接受因使用此功能而导致的任何安全问题。
- 有关创建 UDP 端口的 C 程序示例，请参见 sp_sendmsg。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 syb_sendmsg。

另请参见

系统过程 sp_sendmsg

sys_tempdbid

说明

（仅限集群环境）返回指定实例的有效本地系统临时数据库的 ID。如果未指定 *instance_id*，则返回当前实例的有效本地系统临时数据库的 ID。

语法

sys_tempdbid(*instance_id*)

参数

instance_id

实例的 ID。

示例

返回实例 ID 为 3 的实例的有效本地系统临时数据库 ID。

```
select sys_tempdbid(3)
```

用法

如果未指定实例 ID，则 sys_tempdbid 返回当前实例的有效本地系统临时数据库的 ID。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以运行 sys_tempdbid。

tan

说明　计算指定角 （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之一的数据类型）的列名、变量或表达式。

示例

```
select tan(60)

--------------------
            0.320040
```

用法　tan 是一个数学函数，它返回指定角 （以弧度表示）的正切。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 tan。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　atan、 atn2、 degrees、 radians

tempdb_id

说明

报告给定会话分配到的临时数据库。tempdb_id 函数的输入是服务器进程 ID，其输出是将进程分配到的临时数据库。如果未提供服务器进程，则 tempdb_id 报告分配给当前进程的临时数据库的 dbid。

语法

tempdb_id()

示例

查找分配到给定临时数据库的所有服务器进程：

```
select spid from master..sysprocesses
    where tempdb_id(spid) = db_id("tempdatabase")
```

用法

select tempdb_id 给出与 select @@*tempdbid* 相同的结果。

另请参见

命令 select

textptr

说明

返回一个指针，指向 text、 image 或 unitext 列的第一页。

语法

textptr(*column_name*)

参数

column_name

是 text 列的名称。

示例

示例 1 使用 textptr 函数在作者的 blurbs 表中定位与 au_id 486-29-1786 相关的 text 列 copy。文本指针放在局部变量 *@val* 中并作为参数提供给 readtext 命令，该命令将从第二个字节开始返回 5 个字节（偏移量为 1）：

```
declare @val binary(16)
    select @val = textptr(copy) from blurbs
    where au_id = "486-29-1786"
    readtext blurbs.copy @val 1 5
```

示例 2 从 blurbs 表中选择 title_id 列和 copy 列的 16 字节文本指针：

```
select au_id, textptr(copy) from blurbs
```

用法

- textptr 是一个文本和图像函数，用于返回文本指针值，即 16 字节的 varbinary 值。
- 针对位于仅数据锁定数据行中的行内 LOB 列返回的、由行转发的 textptr 值在转发后保持不变，仍然有效。
- 如果还没有用非空的 insert 或任何 update 语句将 text、 unitext 或 image 列初始化，则 textptr 返回 NULL 指针。用 textvalid 检查文本指针是否存在。不能在无有效文本指针的情况下使用 writetext 或 readtext。

注释 当 varbinary 值存储在表中时，将截断这些值中的尾随 `f`。如果在表中存储文本指针值，可使用 binary 作为列的数据类型。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 textptr。

另请参见

数据类型 text、 image 和 unitext 数据类型

文档 *《Transact-SQL 用户指南》*

函数 textvalid

命令 insert、 update、 readtext、 writetext

textvalid

说明

指向指定 text、 unitext、行内和行外 LOB 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
是文本指针值。

示例

报告 texttest 表的 blurb 列中的每个值是否都有有效的文本指针：

```
select textvalid ("texttest.blurb", textptr(blurb))
from texttest
```

用法

- textvalid 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。
- 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，请参见 *《Transact-SQL 用户指南》*。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 textvalid。

另请参见

数据类型 text、 image 和 unitext 数据类型

文档 *《Transact-SQL 用户指南》*

函数 textptr

to_unichar

说明　返回具有指定整数表达式值的 unichar 表达式。

语法　to_unichar(*integer_expr*)

参数　*integer_expr*

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

用法

- to_unichar 是一个字符串函数，它将 Unicode 整数值转换为 Unicode 字符值。
- 如果 unichar 表达式只引用代理对的一半，就会出现错误消息，且运算将中止。
- 如果 *integer_expr* 是 NULL，则 to_unichar 返回 NULL。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 to_unichar。

另请参见

数据类型　text、 image 和 unitext 数据类型

文档　*《Transact-SQL 用户指南》*

函数　char

tran_dumpable_status

说明 返回一个指明是否允许 dump transaction 的 true/false 值。

语法 tran_dumpable_status("*database_name*")

参数 *database_name*
是目标数据库的名称。

示例 检查 pubs2 数据库是否可以转储：

```
1> select tran_dumpable_status("pubs2")
2> go

 -----------
         106

(1 row affected)
```

在此示例中，无法转储 pubs2。返回码 106 是满足的所有条件 (2、8、32、64) 的和。请参见 “用法” 部分有关返回码的说明。

用法 tran_dumpable_status 可让您不用运行命令就可以确定是否允许对数据库执行转储事务。 tran_dumpable_status 将执行在发出转储事务时 Adaptive Server 所执行的所有检查。

如果 tran_dumpable_status 返回 0，则可以对数据库执行 dump transaction 命令。如果返回任何其它值，则无法执行该命令。非零值有：

- 1 — 指定名称的数据库不存在。
- 2 — 日志没有位于单独的设备上。
- 4 — 日志首页位于仅限数据的磁盘片段区域内。
- 8 — 为数据库设置了 trunc log on chkpt 选项。
- 16 — 在数据库上发生了未记录的写入操作。
- 32 — 仅截断 dump tran 已中断发送到转储设备的任意连续的转储系列。
- 64 — 最近创建或升级了数据库。在执行 dump database 之前，不会转储事务日志。
- 128 — 数据库持久性不允许进行事务转储。
- 256 — 数据库是只读的。 dump transaction 启动了事务，这对于只读数据库来说是不允许的。
- 512 — 数据库已联机以进行备用访问。dump transaction 启动了事务，这对于用于备用访问的数据库来说是不允许的，因为事务会打乱装载序列。

- 1024 — 数据库是存档数据库，它不支持 dump transaction。

标准　　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　　任何用户都可以执行此函数。

另请参见　　**命令**　dump transaction

tsequal

说明

比较 timestamp 的值，以防止对某个在为浏览而被选中以来做过修改的行进行更新。

语法

tsequal(*browsed_row_timestamp*, *stored_row_timestamp*)

参数

browsed_row_timestamp
是被浏览的行的 timestamp 列。

stored_row_timestamp
是所存储行的 timestamp 列。

示例

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。添加 timestamp 列：

```
alter table publishers add timestamp
```

如果这两个 timestamp 列中的值相等，则 tsequal 更新该行。如果它们的值不相等，则 tsequal 返回以下错误消息：

```
update publishers
set city = "Springfield"
where pub_id = "0736"
and tsequal(timestamp, 0x0001000000002ea8)
Msg 532, Level 16, State 2:

Server 'server_name', Line 1:
The timestamp (changed to 0x0001000000002ea8) shows
that the row has been updated by another user.
Command has been aborted.
(0 rows affected)
```

用法

- tsequal 是一个系统函数，它会比较 timestamp 列值，以防止对某个自为浏览而被选中以来做过修改的行进行更新。有关系统函数的常规信息，请参见 *《Transact-SQL 用户指南》*。
- tsequal 允许在不调用 DB-Library 中的 dbqual 函数的情况下使用浏览模式。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，请在发送给 Adaptive Server 的 select 语句末尾附加 for browse 关键字。例如：

```
Start of select statement in an Open Client application
...
   for browse

Completion of the Open Client application routine
```

- 不要在 select 语句的 where clause 中使用 tsequal；只应在 insert 和 update 语句的 where 子句中使用，且 where clause 的其余部分与一个唯一的行匹配。

  如果将 timestamp 列用作搜索子句，则应像比较常规 varbinary 列那样来比较该列，即 timestamp1 = timestamp2。

为用于浏览的新表加盖时间戳

- 在创建用于浏览的新表时，在表定义中加入名为 timestamp 的列。系统将自动为该列指派 timestamp 数据类型，所以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 就为其加盖时间戳，方法是：自动为 timestamp 列指派唯一的 varbinary 值。

为现有表加盖时间戳

- 若要准备现有的表供浏览，可使用 alter table 添加名为 timestamp 的列。例如，若要添加一个 timestamp 列并使列中现有的每一行均为 NULL 值：

```
alter table oldtable add timestamp
```

  若要生成时间戳，请在不指定新列值的情况下更新现有的每一行：

```
update oldtable
set col1 = col1
```

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 tsequal。

另请参见　**数据类型**　Timestamp 数据类型

uhighsurr

说明

如果 start 位置的 Unicode 值是代理对的上半部 （应首先出现在对中），则返回 1。否则返回 0。该函数允许编写用于处理代理的显式代码。

语法

uhighsurr(*uchar_expr*, start)

参数

uchar_expr

类型为 unichar 或 univarchar 的字符型列名、变量或常量表达式。

start

指定要研究的字符位置。

用法

- uhighsurr 是一个字符串函数，允许编写用于处理代理的显式代码。具体而言，如果子字符串从 uhighsurr 为 true 的 Unicode 字符开始，则提取至少包含两个 Unicode 值的子字符串（*substr* 不提取代理对的一半）。
- 如果 *uchar_expr* 是 NULL，则 uhighsurr 返回 NULL。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 uhighsurr。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 ulowsurr

ulowsurr

说明

如果 *start* 处的 Unicode 值是代理对的下半部（应出现在对中的第二部分），则返回 1。否则返回 0。该函数允许在 substr()、stuff() 和 right() 所执行的调整周围显式代码。

语法

ulowsurr(*uchar_expr*, start)

参数

uchar_expr

类型为 unichar 或 univarchar 的字符型列名、变量或常量表达式。

start

指定要研究的字符位置。

用法

- ulowsurr 是一个字符串函数，允许根据 substr、 stuff 或 right 执行的调整编写显式代码。具体而言，如果子字符串以 ulowsurr 为 true 的 Unicode 值结尾，则用户就会知道提取少 1（或多 1）个字符的子字符串。 substr 不会提取包含不匹配代理对的字符串。
- 如果 *uchar_expr* 是 NULL，则 ulowsurr 返回 NULL。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 ulowsurr。

另请参见

文档　*《Transact-SQL 用户指南》*

函数　uhighsurr

upper

说明

将指定的小写字符串转换为等值大写字符串。

语法

upper(*char_expr*)

参数

char_expr

类型为 char、unichar、varchar, nchar、nvarchar 或 univarchar 的字符型列名、变量或常量表达式。

示例

```
select upper("abcd")

----
ABCD
```

用法

- upper 是一个字符串函数，它将小写转换为大写并返回字符值。
- 如果 *char_expr* 或 *uchar_expr* 是 NULL，则 upper 返回 NULL。
- 没有等值大写的字符保持不变。
- 如果创建的 unichar 表达式只包含代理对的一半，则会出现错误消息，且运算将中止。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 upper。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 lower

uscalar

说明　返回表达式第一个 Unicode 字符的 Unicode 标量值。

语法　uscalar(*uchar_expr*)

参数　*uchar_expr*
类型为 unichar 或 univarchar 的字符型列名、变量或常量表达式。

用法

- uscalar 是一个字符串函数，它返回表达式中第一个 Unicode 字符的 Unicode 值。
- 如果 *uchar_expr* 是 NULL，则返回 NULL。
- 如果在包含不匹配代理对一半的 *uchar_expr* 上调用 uscalar，就会出现错误，而且运算会中止。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 uscalar。

另请参见　**文档**　*《Transact-SQL 用户指南》*

函数　ascii

used_pages

说明

报告表、索引或特定分区所用的页数。与 data_pages 不同， used_pages 还包括用于内部结构的页。此函数替换 Adaptive Server 15.0 之前的版本中使用的 used_pgs 函数。

语法

used_pages(*dbid*, *object_id*[, *indid*[, *ptnid*]])

参数

dbid

是目标对象所在的数据库的 ID。

object_id

是要查看其已使用页的表的对象 ID。若要查看索引使用的页，请指定索引所属表的对象 ID。

indid

是所需的索引 ID。

ptnid

是所需的分区 ID。

示例

示例 1 返回指定数据库中对象 ID 为 31000114 的对象所使用的页数（包括所有索引）：

```
select used_pages(5, 31000114)
```

示例 2 返回数据层中的对象所使用的页数，而不管是否存在聚簇索引：

```
select used_pages(5, 31000114, 0)
```

示例 3 返回索引层中对象为索引 ID 为 2 的索引所使用的页数。这不包括数据层使用的页数 （有关例外，请参见 “用法”部分中的第一个符号项）：

```
select used_pages(5, 31000114, 2)
```

示例 4 返回特定分区的数据层中对象所使用的页数，在此示例中为 2323242432：

```
select used_pages(5, 31000114, 0, 2323242432)
```

用法

- 在具有聚簇索引的所有页锁定表中，最后一个参数的值决定返回哪些已用页：
  - used_pages(dbid, objid, 0) — 以索引 ID 形式显式传递 0，只返回数据层使用的页。
  - used_pages(dbid, objid, 1) — 返回索引层和数据层使用的页。

  对于具有聚簇索引的所有页锁定表，若要获取索引层使用的页，请从 used_pages(dbid, objid, 1) 中减去 used_pages(dbid, objid, 0)。

- used_pages 会放弃尚未在高速缓存中的对象的描述符，而不是消耗资源。
- 在具有聚簇索引的所有页锁定表中，对于值 `indid = 0`，used_pages 只传递数据层中的已用页。当传递 `indid=1` 时，将返回数据层和聚簇索引层中的已用页，这与以前的版本一样。
- used_pages 类似于旧的 used_pgs(objid, doampg, ioampg) 函数。
- 任何错误情形都会导致返回值为零。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 used_pgs。

另请参见　**函数**　data_pages、 object_id

user

说明 返回当前用户的名称。

语法 user

参数 无

示例

```
select user

------
dbo
```

用法

- user 是一个系统函数，它返回用户的名称。
- 如果 sa_role 处于启用状态，您将自动成为所使用的任何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始终是 “dbo”。

标准 符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限 任何用户都可以执行 user。

另请参见

文档 *《Transact-SQL 用户指南》*

函数 user_name

user_id

说明

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

示例 1

```
select user_id()

------
     1
```

示例 2

```
select user_id("margaret")

------
     4
```

用法

- user_id 是一个系统函数，它返回用户的 ID 号。有关系统函数的常规信息，请参见 *《Transact-SQL 用户指南》*。
- user_id 报告当前数据库中 sysusers 的编号。如果未提供 *user_name*，user_id 就会返回当前用户的 ID。若要查找服务器用户 ID（它在 Adaptive Server 上的每个数据库中都相同），请使用 suser_id。
- 在数据库中，“guest”用户 ID 始终是 2。
- 在数据库中，数据库所有者的 user_id 始终是 1。如果 sa_role 处于启用状态，您将自动成为所使用的任何数据库的数据库所有者。若要返回到实际用户 ID，请在执行 user_id 之前使用 set sa_role off。如果您不是数据库中的有效用户，Adaptive Server 将在您使用 set sa_role off 时返回错误。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

只有系统管理员或系统安全员才能用自己以外的 user_name 使用该函数。

另请参见

命令 setuser

文档 *《Transact-SQL 用户指南》*

函数 suser_id、user_name

user_name

说明 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 **示例 1**

```
select user_name()

------------------------------
dbo
```

示例 2

```
select user_name(4)

------------------------------
margaret
```

用法

- user_name 是一个系统函数，它根据用户在当前数据库中的用户 ID 返回用户名。
- 如果未提供 *user_id*， user_name 就会返回当前用户的名称。
- 如果 sa_role 处于启用状态，您将自动成为所使用的任何数据库的数据库所有者。在数据库中，数据库所有者的 user_name 始终是 “dbo”。

标准 符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限 只有系统管理员或系统安全员才能用自己以外的 user_id 使用该函数。

另请参见 **文档** *《Transact-SQL 用户指南》*

函数 suser_name、 user_id

valid_name

说明

如果指定字符串不是有效标识符，则返回 0；如果该字符串是有效标识符，则返回非 0 数字，且长度最大可以为 255 个字节。

语法

valid_name(*character_expression*[, *maximum_length*])

参数

character_expression

类型为 char、varchar、nchar 或 nvarchar 的字符型列名、变量或常量表达式。常量表达式必须加以引号。

maximum_length

是一个大于 0 且小于或等于 255 的整数。缺省值为 30。如果标识符长度大于第二个参数，则 valid_name 返回 0；如果标识符长度无效，则返回一个大于 0 的值。

示例

创建一个过程来验证标识符的有效性：

```
create procedure chkname
@name varchar(30)
as
     if valid_name(@name) = 0
     print "name not valid"
```

用法

- valid_name 是一个系统函数。如果 *character_expression* 不是有效标识符（非法字符、长度大于 30 个字节或保留字），则返回 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多字节字符， Adaptive Server 标识符最长可以含 16384 个字节。标识符的首字符必须是字母字符（按当前字符集中的定义），或是下划线 (_) 字符。以井号 (#) 开头的临时表名和以 at 符号 (@) 开头的局部变量名是该规则的例外。对于以井号 (#) 和以 at 符号 (@) 开头的标识符， valid_name 将返回 0。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 valid_name。

另请参见

文档　*《Transact-SQL 用户指南》*

系统过程　sp_checkreswords

valid_user

说明

如果指定的 ID 至少在一个数据库中是有效用户或别名，则返回 1。

语法

valid_user(*server_user_id* [, *database_id*])

参数

server_user_id

服务器用户 ID 存储在 syslogins 的 suid 列中。

database_id

您要确定用户是否对其有效的数据库的 ID。数据库 ID 存储在 sysdatabases 的 dbid 列中。

示例

示例 1 suid 为 4 的用户在至少一个数据库中是有效的用户或别名：

```
select valid_user(4)
---------------
              1
```

示例 2 suid 为 4 的用户在 ID 为 6 的数据库中是有效的用户或别名。

```
select valid_user(4,6)
---------------
              1
```

用法

- 如果指定的 *server_user_id* 在指定的 *database_id* 中是有效的用户或别名，则 valid_user 返回 1。
- 如果您未指定 *database_id*，或者如果它为 0，valid_user 会确定用户是否在至少一个数据库中是有效的用户或别名。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

对 valid_user 的权限检查因您的细化权限设置而异。

细化权限已启用	在启用细化权限的情况下，您必须具有 manage any login 或 manage server 权限才能对自己以外的 server_user_id 执行 valid_user。
细化权限已禁用	在禁用细化权限的情况下，您必须是具有 sa_role 或 sso_role 的用户才能对自己以外的 server_user_id 执行 valid_user。

另请参见

文档 *《Transact-SQL 用户指南》*

系统过程 sp_addlogin、 sp_adduser

var

说明

计算包含数值表达式的样本的统计方差，数据类型为 double，并返回一组数值的方差。

注释　var 和 variance 是 var_samp 的别名。有关详细信息，请参见 第 313 页的 var_samp。

var_pop

说明

计算包含数值表达式的总体的统计方差，数据类型为 double。 varp 是 var_pop 的别名，语法相同。

语法

var_pop ([all | distinct] *expression*)

参数

all

将 var_pop 应用于所有值。 all 是缺省值。

distinct

在应用 var_pop 之前消除重复值。

expression

一个表达式 （通常为列名），用于对一组行计算其基于总体的方差。

示例

列出 pubs2 数据库中每种类型书籍预付款的平均值和方差：

```
select type, avg(advance) as "avg", var_pop(advance)
  as "variance" from titles group by type order by type
```

用法

计算对每个组行求值的所提供值表达式的总体方差 （如果指定了 distinct，则已删除在复制之后仍保留的各行），其定义为值表达式与值表达式均值之差的平方和，然后再除以组中或分区中的行数。

图 2-3：总体相关的统计集合函数的公式

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

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

σ^2 = 方差

n = 总数

$\mu = x_i$ 的均值

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 var_pop。

另请参见

有关集合函数的一般信息，请参见 《Adaptive Server Enterprise 参考手册：*构件块*》。

函数 stddev_pop、 stddev_samp、 var_samp

var_samp

说明

计算包含数值表达式的样本的统计方差，数据类型为 double，并返回一组数值的方差。var 和 variance 是 var_samp 的别名，并使用相同的语法。

语法

var_samp ([all | distinct] *expression*)

参数

all

将 var_samp 应用于所有值。 all 是缺省值。

distinct

在应用 var_samp 之前消除重复值。

expression

是任意数值数据类型 （float、 real 或 double）表达式。

示例

列出 pubs2 数据库中每种类型书籍预付款的平均值和方差：

```
select type, avg(advance) as "avg", var_samp(advance)
  as "variance" from titles where
  total_sales > 2000 group by type order by type
```

用法

var_samp 返回双精度浮点数据类型的结果。如果应用于空集，则结果为 NULL。

图 2-4：样本相关的统计集合函数的公式

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

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

s^2 = 方差

n = 样本数

$\bar{x} = x_i$ 的均值

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以执行 var_samp。

另请参见

有关集合函数的一般信息，请参见 《Adaptive Server Enterprise 参考手册：*构件块*》。

函数　stddev_pop、 stddev_samp、 var_pop

variance

说明

计算包含数值表达式的样本的统计方差，数据类型为 double，并返回一组数值的方差。

注释 var 和 variance 是 var_samp 的别名。有关详细信息，请参见 第 313 页的 var_samp。

varp

说明

计算包含数值表达式的总体的统计方差，数据类型为 double。

注释　varp 是 var_pop 的别名。有关详细信息，请参见 第 312 页的 var_pop。

workload_metric

说明

（仅限集群环境）查询指定实例的当前工作负荷指标，或更新指定实例的指标。

语法

workload_metric(*instance_id* | *instance_name* [, *new_value*])

参数

instance_id
实例的 ID。

instance_name
实例名。

new_value
表示新指标的浮点值。

示例

示例 1 查看当前实例上的用户指标：

```
select workload_metric()
```

示例 2 查看实例 “ase2” 上的用户指标：

```
select workload_metric("ase2")
```

示例 3 将 “ase3” 上的用户指标值设置为 27.54：

```
select workload_metric("ase3", 27.54)
```

用法

- 空值指示当前实例。
- 如果为 *new_value* 指定了值，则指定值将成为当前用户指标。如果没有为 *new_value* 指定值，则返回当前工作负荷指标。
- *new_value* 的值必须等于或大于零。
- 如果为 *new_value* 提供了值， workload_metric 将在运算成功时返回该值。否则， workload_metric 返回 -1。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

您必须具有 sa_role 或 ha_role 才能执行 workload_metric

xa_bqual

说明　返回 ASCII XA 事务 ID 的 bqual 组成部分的二进制对应值。

语法　xa_bqual(*xid*, 0)

参数

xid

是 Adaptive Server 事务的 ID，是从 systransactions 中的 xactname 列或从 sp_transactions 获取的。

0

留作将来使用。

示例　**示例 1** 返回“0x227f06ca80”，该值是将 Adaptive Server 事务 ID“0000000A_IphIT596iC7bF2#AUfkzaM_8DY6OE0”的分支限定符转换为二进制后的结果。 Adaptive Server 事务 ID 最初是使用 sp_transactions 获取的：

```
1sp_transactions

xactkey                          type     coordinator starttime           st
ate      connection dbid  spid  loid  failover    srvname  namelen  xactna
me
------------------------------ -------- ----------- ------------------ --
-------- ---------- ----- ----- ----- ----------- -------  -------  ------
--------------------------------
0x531600000600000017e4885b0700 External XA          Dec 9 2005  5:15PM In
Command  Attached     7   20    877  Resident Tx  NULL         39
0000000A_IphIT596iC7bF2#AUfkzaM_8DY6OE0

1> select xa_bqual("0000000A_IphIT596iC7bF2#AUfkzaM_8DY6OE0", 0)
2> go

...
------------------------------------------------------------------------
0x227f06ca80
```

示例 2 xa_bqual 通常和 xa_gtrid 一起使用。在此示例中，返回 systransactions 中 coordinator 列的值为“3”的所有行的全局事务 ID 和分支限定符：

```
11> select gtrid=xa_gtrid(xactname,0),
    bqual=xa_bqual(xactname,0)
    from systransactions where coordinator = 3
2> go

         gtrid

        bqual

------------------------------------------------------------------------
```

```
0xb1946cdc52464a61cba42fe4e0f5232b

0x227f06ca80
```

用法

如果 Adaptive Server 阻塞了外部事务并且使用 sp_lock 和 sp_transactions 来标识阻塞事务，则可以使用 XA 事务管理器来终止全局事务。但是，当执行 sp_transactions 时，返回的 *xactname* 的值为 ASCII 字符串格式，同时 XA Server 使用未解码的二进制值。因而使用 xa_bqual 可以以 XA 事务管理器能够理解的格式来确定事务名的 bqual 部分。

xa_bqual 返回：

- 此字符串的转换值，该值位于第二个“_”（下划线）之后，第三个“_”或者字符串结尾值 （以位置靠前者为准）之前。
- NULL，如果无法对事务 ID 进行解码，或者事务 ID 为预料之外的格式。

注释 xa_bqual 不对 xid 执行验证检查，而只返回转换后的字符串。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以使用 xa_bqual。

另请参见

函数 xa_gtrid

存储过程 sp_lock、 sp_transactions

xa_gtrid

说明　返回 ASCII XA 事务 ID 的 gtrid 组成部分的二进制对应值。

语法　xa_gtrid(*xactname*, *int*)

参数

xid

是 Adaptive Server 事务的 ID，是从 systransactions 中的 xactname 列或从 sp_transactions 获取的。

0

留作将来使用。

示例　**示例 1** 在这种典型情况下，对于 Adaptive Server 事务 ID “0000000A_IphIT596iC7bF2#AUfkzaM_8DY6OE0”，返回分支限定符的二进制转换值 “0x227f06ca80”和全局事务 ID “0xb1946cdc52464a61cba42fe4e0f5232b”：

```
1> select xa_gtrid("0000000A_IphIT596iC7bF2#AUfkzaM_8DY6OE0", 0)
2> go

       ...
------------------------------------------------------------------------------
          0xb1946cdc52464a61cba42fe4e0f5232b

(1 row affected)
```

示例 2 xa_bqual 通常和 xa_gtrid 一起使用。在此示例中，返回 systransactions 中 coordinator 列的值为“3”的所有行的全局事务 ID 和分支限定符：

```
11> select gtrid=xa_gtrid(xactname,0),
    bqual=xa_bqual(xactname,0)
    from systransactions where coordinator = 3
2> go

         gtrid

        bqual

------------------------------------------------------------------------------
         0xb1946cdc52464a61cba42fe4e0f5232b

         0x227f06ca80
```

用法

如果 Adaptive Server 阻塞了外部事务并且使用 sp_lock 和 sp_transactions 来标识阻塞事务，则可以使用 XA 事务管理器来终止全局事务。但是，当执行 sp_transactions 时，返回的 *xactname* 的值为 ASCII 字符串格式，同时 XA Server 使用未解码的二进制值。因而使用 xa_gtrid 可以以 XA 事务管理器能够理解的格式来确定事务名的 gtrid 部分。

xa_gtrid 返回：

- 此字符串的转换值，该值位于第一个“_”（下划线）之后，第二个“_”或者字符串结尾值 （以位置靠前者为准）之前。
- NULL，如果无法对事务 ID 进行解码，或者事务 ID 为预料之外的格式。

注释 xa_gtrid 不对 xid 执行验证检查，而只返回转换后的值。

标准

符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限

任何用户都可以使用 xa_gtrid。

另请参见

函数 xa_bqual

存储过程 sp_lock、 sp_transactions

xact_connmigrate_check

说明　（仅限集群环境）确定某个连接是否可以处理外部事务。

语法　xact_connmigrate_check("*txn_name*")

参数　*txn_name*
事务 ID。该参数是可选的。

示例　**示例 1** XA 事务 “txn_name” 在实例 “ase1” 上运行。

```
select xact_connmigrate_check("txn_name")
--------
    1
```

示例 2 XA 事务 “txn_name” 在实例 “ase2” 上运行。连接可以迁移。

```
select xact_connmigrate_check("txn_name")
--------
    1
```

示例 3 XA 事务 “txn_name” 在实例 “ase2” 上运行。连接不能迁移。

```
select xact_connmigrate_check("txn_name")
--------
    0
```

用法

- 如果指定了 XID， xact_connmigrate_check 会返回：
  - 1 （如果连接通到运行指定事务的实例，或者连接通到另一个处于可迁移状态的实例）
  - 0 （如果连接或事务 ID 不存在，或者连接通到另一个不处于可迁移状态的实例）
- 如果未指定 XID， xact_connmigrate_check 会返回：
  - 1 （如果连接处于可迁移状态）
  - 0 （如果连接不存在或不处于可迁移状态）

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 xact_connmigrate_check。

另请参见　**函数**　xact_owner_instance

xact_owner_instance

说明 （仅限集群环境）返回正在运行分布式事务的实例 ID。

语法 xact_owner_instance("*txn_name*")

参数 *txn_name*
事务 ID。

示例 **示例 1** XA 事务 “txn_name” 在实例 “ase1” 上运行。

```
select xact_owner_instance(txn_name)
-------
    1
```

示例 2 XA 事务 “txn_name” 不在运行。

```
select xact_owner_instance(txn_name)
-------
  NULL
```

用法

- xact_owner_instance returns:
  - 运行事务的实例的实例 ID，或者
  - Null （如果事务不在运行）

标准 符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限 任何用户都可以执行 xact_owner_instance。

另请参见 **函数** xact_connmigrate_check

xmlextract

说明

将 XML 查询表达式应用到 XML 文档并返回指定的结果。返回的信息中可以带有或不带有 XML 标记。

另请参见

请参见 *XML 服务*以了解 xmlextract 的语法、示例和用法信息，以及所有其它支持在数据库中使用 XML 的 Transact-SQL 函数。

xmlparse

说明

分析作为参数传递的 XML 文档，并返回包含经过分析的文档的 image（缺省值）、 binary 或 varbinary 值。

另请参见

请参见 XML 服务以了解 xmlparse 的语法、示例和用法信息，以及所有其它支持在数据库中使用 XML 的 Transact-SQL 函数。

xmlrepresentation

说明

检查表达式的 image 参数，并返回一个整数值，此整数值指示该参数是否包含经过分析的 XML 数据或其它类型的 image 数据。

另请参见

请参见 XML 服务以了解 xmlrepresentation 的语法、示例和用法信息，以及所有其它支持在数据库中使用 XML 的 Transact-SQL 函数。

xmltable

说明	从 XML 文档中提取数据，并以 SQL 表的形式返回该数据。
另请参见	请参见 XML 服务以了解 xmltable 的语法、示例和用法信息，以及所有其它支持在数据库中使用 XML 的 Transact-SQL 函数。

xmltest

说明

是对 XML 查询表达式求值的 SQL 谓词，它可以引用 XML 文档参数，并将返回布尔结果。 xmltest 类似于 SQL like 谓词。

另请参见

请参见 XML 服务以了解 xmltest 的语法、示例和用法信息，以及所有其它支持在数据库中使用 XML 的 Transact-SQL 函数。

xmlvalidate

说明	验证 XML 文档。
另请参见	请参见 XML 服务以了解 xmlvalidate 的语法、示例和用法信息，以及所有其它支持在数据库中使用 XML 的 Transact-SQL 函数。

year

说明　返回指定日期的 datepart 中表示年份的整数。

语法　year(*date_expression*)

参数　*date_expression*

是 datetime、smalldatetime、date、time 类型的表达式，或 datetime 格式的字符串。

示例　返回整数 03：

```
year("11/02/03")
----------
03
(1 row(s) affected)
```

用法　year(*date_expression*) 等同于 datepart(*yy*, *date_expression*)。

标准　符合 ANSI SQL 的级别 Transact-SQL 扩展。

权限　任何用户都可以执行 year。

另请参见　**数据类型**　datetime、 smalldatetime、 date

函数　datepart、 day、 month

第 3 章

全局变量

Adaptive Server 的全局变量

全局变量是系统 — 定义的变量，这种变量在系统运行时由 Adaptive Server 进行更新。有些全局变量是会话特有的，而有些则是服务器实例特有的。例如， *@@error* 包含系统为给定的用户连接生成的最后一个错误号。

若要指定应用程序环境变量，请参见 get_appcontext 和 set_appcontext。

若要查看任何全局变量的值，请输入：

```
select variable_name
```

例如：

```
select @@char_convert
```

许多全局变量报告自 Adaptive Server 最近一次启动以来，系统发生的活动。 sp_monitor 显示一些全局变量的当前值。

表 3-1 列出了 Adaptive Server 可用的全局变量：

表 3-1：Adaptive Server 的全局变量

全局变量	定义
@@active_instances	返回集群中活动实例的数目。
@@authmech	一个只读变量，表示鉴定用户所用的机制。
@@bootcount	返回 Adaptive Server 安装程序已启动的次数。
@@boottime	返回 Adaptive Server 上次启动的日期和时间。
@@bulkarraysize	返回使用批量复制接口进行传输之前缓冲到本地服务器内存中的行数。它仅用于组件集成服务，以使用 select into 将行传输到远程服务器。请参见 *《组件集成服务用户指南》*。
@@bulkbatchsize	返回使用批量接口通过 select into *proxy_table* 传输到远程服务器的行数。仅用于组件集成服务，以便通过 select into 将行传输到远程服务器。请参见 *《组件集成服务用户指南》*。
@@char_convert	如果字符集转换无效，则返回 0。如果字符集转换有效，则返回 1。
@@cis_rpc_handling	如果 cis rpc handling 关闭，则返回 0。如果 cis rpc handling 打开，则返回 1。请参见 *《组件集成服务用户指南》*。
@@cis_version	返回组件集成服务的日期和版本。

全局变量	定义
@@client_csexpansion	返回从服务器字符集转换为客户端字符集时使用的扩展因子。例如，如果它包含一个值 2，则服务器字符集中的字符在转换为客户端字符集后最多可以占用原来字节数的两倍。
@@client_csid	如果客户端字符集从未初始化，则返回 -1 ；如果客户端字符集已初始化，则通过 syscharsets 返回连接的客户端字符集 ID。
@@client_csname	如果客户端字符集从未初始化，则返回 NULL ；如果客户端字符集已初始化，则返回用于连接的字符集名称。
@@clusterboottime	返回首次启动集群的日期和时间，即使最初启动集群的实例已关闭
@@clustercoordid	返回当前集群事务协调器的实例 ID
@@clustermode	返回字符串：“共享磁盘集群” (shared-disk cluster)
@@clustername	返回集群的名称
@@cmpstate	返回在高可用性环境下 Adaptive Server 的当前模式。不用于非高可用性环境中。
连接	返回用户尝试登录的次数。
@@cpu_busy	返回自 Adaptive Server 上次启动起，CPU 执行 Adaptive Server 工作所花费的时间 （以时钟周期计）。

@@cursor_rows	为可滚动游标专门设计的全局变量。该变量显示游标结果集中的总行数。它返回以下值：
	• -1 — 游标是：
	  • 动态 — 因为动态游标会反映所有的更改，所以符合该游标的条件的行数会不断变化。您永远都无法保证会检索所有符合条件的行。
	  • semi_sensitive 并且是可滚动的，但尚未完全填充滚动工作表 — 检索到该值时，符合该游标的条件的行数是未知的。
	• 0 — 没有打开游标，没有行符合上次打开的游标的条件，或者上次打开的游标已关闭或已释放。
	• *n* — 上次打开的或获取的游标结果集已完全填充。返回的值是游标结果集中的总行数。

@@curloid	返回当前会话的锁所有者的 ID。
@@datefirst	使用 set datefirst *n* 进行设置，其中 *n* 是 1 和 7 之间的一个值。返回 *@@datefirst* 的当前值，该值指示将每个星期的哪一天指定为第一天，其数据类型为 tinyint。 在 Adaptive Server 中，缺省值为星期日 （根据 us_language 缺省值），您可以通过指定 `set datefirst 7` 来进行设置。有关这些设置和值的详细信息，请参见 set 命令的 datefirst 选项。

@@dbts	返回当前数据库的时间戳。
	时间戳列始终按照 big-endian 字节顺序显示值，但在小端平台上， *@@dbts* 以 little-endian 字节顺序显示。要将小端 *@@dbts* 值转换为可与时间戳列比较的大端值，请使用：
	`reverse(substring(@@dbts,1,2)) + 0x0000 +`
	`reverse(substring(@@dbts,5,4))`

全局变量	定义
错误	返回系统最近一次生成的错误号。
@@errorlog	返回指向 Adaptive Server 错误日志所在目录的完整路径，该路径是相对于 *$SYBASE* 目录 （在 NT 上为 *%SYBASE%*）的路径。
@@failedoverconn	如果与主协同服务器的连接发生了故障切换而转到辅助协同服务器上执行，则返回一个大于 0 的值。仅用于高可用性环境，并且因会话而异。
@@fetch_status	返回： • 0 — 获取操作成功 • -1 — 获取操作失败 • -2 — 该值留作以后使用
@@guestuserid	返回 guest 用户的 ID。
@@hacmpservername	返回高可用性设置中的协同服务器的名称。
@@haconnection	如果连接启用了故障切换属性，则返回一个大于 0 的值。这是一个因会话而异的属性。
@@heapmemsize	返回堆内存池的大小，单位为字节。有关堆内存的详细信息，请参见 *《系统管理指南》*。
@@identity	返回最近一次生成的 IDENTITY 列值。
@@idle	返回 Adaptive Server 自上次启动起空闲的时间 （以时钟周期计）。
@@instanceid	返回从中执行它的实例的 ID
@@instancename	返回从中执行它的实例的名称
@@invaliduserid	对于无效的用户 ID 返回 -1。
@@io_busy	返回 Adaptive Server 执行输入和输出操作所花费的时间 （以时钟周期计）。
@@isolation	返回当前 Transact-SQL 程序特定于会话的隔离级别值 （0、 1 或 3）。
@@jsinstanceid	Job Scheduler 正在其中运行或启用后将在其中运行的实例的 ID。
@@kernel_addr	返回包含内核区域的第一个共享内存区域的起始地址。返回结果的格式为：0x *地址指针值*。
@@kernel_size	返回属于第一个共享内存区域的一部分的内核区域的大小。
@@kernelmode	返回为 Adaptive Server 配置的模式 （线程或进程）。
@@langid	返回当前在整个服务器范围内使用的语言的语言 ID，该值在 syslanguages.langid 中指定。
@@language	返回当前使用的语言的名称，该名称在 syslanguages.name 中指定。
@@lastkpgendate	返回按 sp_passwordpolicy 的 “keypair regeneration period” 策略选项所设置的那样生成最后一个密钥对的日期和时间。
@@lastlogindate	*@@lastlogindate* 可供每个用户登录会话使用，包括 datetime 数据类型，其值为当前会话建立之前的登录帐户的 lastlogindate 列。此变量特定于每个登录会话，可供该会话用于确定帐户的上次登录。如果以前未使用过该帐户，或者 “sp_passwordpolicy 'set', enable last login updates” 为 0，则 *@@lastlogindate* 的值为 NULL。

全局变量	定义
@@lock_timeout	使用 set lock wait n 进行设置。返回当前的 *lock_timeout* 设置，单位为毫秒。*@@lock_timeout* 返回值 n。缺省值为无超时。如果会话开始时没有执行任何 set lock wait n，则 *@@lock_timeout* 会返回 -1。
@@lwpid	返回最近运行的轻量过程的对象 ID
@@max_connections	返回在当前计算机环境下可同时与 Adaptive Server 连接的最大数目。您可以使用 number of user connections 配置参数将 Adaptive Server 的连接数配置为任何小于或者等于 *@@max_connections* 的值。
@@max_precision	返回由服务器设置的 decimal 和 numeric 数据类型所使用的精度级别。该值固定为常数 38。
@@maxcharlen	返回 Adaptive Server 的缺省字符集中一个字符的最大长度，单位为字节。
@@maxgroupid	返回最大的组用户 ID。
@@maxpagesize	返回服务器的逻辑页大小。
@@maxspid	返回 spid 的最大有效值。
@@maxsuid	返回最大的服务器用户 ID。
@@maxuserid	返回最大的用户 ID。
@@mempool_addr	返回全局内存池表的地址。返回结果的格式为：0x *地址指针值*。此变量仅供内部使用。
@@min_poolsize	返回命名高速缓存池的最小大小，单位为千字节。该值是根据 DEFAULT_POOL_SIZE 的值（为 256）和 max database page size 的当前值进行计算得出的。
@@mingroupid	返回最小的组用户 ID。
@@minspid	返回 1，即 spid 的最小值。
@@minsuid	返回最小的服务器用户 ID。
@@minuserid	返回最小的用户 ID。
@@monitors_active	减少 sp_sysmon 显示的消息数。
@@ncharsize	返回当前服务器的缺省字符集中一个字符的最大长度，单位为字节。
@@nestlevel	返回当前的嵌套级别。
@@nextkpgendate	返回按 sp_passwordpolicy 的 “keypair regeneration period”策略选项所设置的那样预定生成下一个密钥对的日期和时间。
@@nodeid	返回当前安装的 48 位节点标识符。Adaptive Server 在第一次使用主设备时生成一个 nodeid，并唯一地标识一个 Adaptive Server 安装。
@@optgoal	返回查询优化的当前优化目标设置。
@@optoptions	返回活动选项的位图。
选项	返回以十六进制表示的会话的 set 选项。
@@optlevel	返回当前的优化级别设置。
@@opttimeoutlimit	返回查询优化当前的优化超时限制设置。
@@ospid	（仅限线程化模式）返回服务器的操作系统 ID。
@@pack_received	返回 Adaptive Server 读取的输入包的数目。

全局变量	定义
@@pack_sent	返回 Adaptive Server 写入的输出包的数目。
@@packet_errors	返回 Adaptive Server 读写数据包时检测到的错误数。
@@pagesize	返回服务器的虚拟页大小。
@@parallel_degree	返回当前的最大并行度设置。
@@plwpid	返回最近准备的轻量过程的对象 ID。
@@probesuid	对于 probe 用户 ID 返回值 2。
@@procid	返回当前执行过程的存储过程 ID。
@@quorum_physname	返回仲裁设备的物理路径
@@recovery_state	根据以下这些返回值，指出 Adaptive Server 是否正在进行恢复： • NOT_IN_RECOVERY：Adaptive Server 未处于启动恢复或故障切换恢复状态。恢复已完成，且所有可以联机的数据库都已处于联机状态。 • RECOVERY_TUNING：Adaptive Server 正在进行恢复 （启动或故障切换），且正在调节最佳恢复任务数。 • BOOTIME_RECOVERY：Adaptive Server 正在进行启动恢复，并且已经完成了调节最佳恢复任务数，但不是所有数据库都已恢复。 • FAILOVER_RECOVER：Adaptive Server 正处于某次 HA 故障切换过程中的恢复状态，并且已经完成了调节最佳恢复任务数。所有数据库都还没有联机。
@@remotestate	返回高可用性环境中主协同服务器的当前模式。有关返回的值，请参见 *《在高可用性环境中使用 Sybase 故障切换》*。
@@repartition_degree	返回当前的动态重新分区程度设置。
@@resource_granularity	返回查询优化的最大资源使用情况提示设置。
@@rowcount	返回上次查询所影响的行数。指定游标是仅向前游标还是可滚动游标将影响 *@@rowcount* 的值。 如果游标是缺省的不可滚动游标，则 *@@rowcount* 的值只能向前按一递增，直至读取完结果集中的行。这些行是从基础表中读取到客户端的。*@@ rowcount* 的最大值是结果集中的行数。 在缺省游标中， *@@rowcount* 由不返回或不影响行的任何命令 （如 if 或 set 命令）设置为 0，或者由不影响任何行的 update 或 delete 语句设置为 0。 如果游标是可滚动游标，则 *@@rowcount* 无最大值。每次执行 fetch 时，无论方向如何，该值都会继续增加，并且没有最大值。可滚动游标中的 *@@rowcount* 值反映的是从结果集中读取到客户端的行数，而不是从基础表中读取到客户端的行数。
@@scan_parallel_degree	返回用于非聚簇索引扫描的当前最大并行度设置。
@@servername	返回 Adaptive Server 的名称。
@@setrowcount	返回 set rowcount 的当前值。

全局变量	定义
@@shmem_flags	返回共享内存区域的属性。此变量仅供内部使用。总共有 13 个不同的属性值，分别对应于整数中的 13 位。从低位到高位表示的有效值分别为：MR_SHARED、 MR_SPECIAL、 MR_PRIVATE、 MR_READABLE、MR_WRITABLE、 MR_EXECUTABLE、 MR_HWCOHERENCY、MR_SWCOHERENC、MR_EXACT、MR_BEST、MR_NAIL、MR_PSUEDO、MR_ZERO。
@@spid	返回当前进程的服务器进程 ID。
@@sqlstatus	返回执行 fetch 语句时产生的状态信息 （警告例外）。
@@ssl_ciphersuite	如果 SSL 未用于当前连接，则返回 NULL ；否则，它返回在当前连接的 SSL 握手期间您选择的密码成套程序的名称。
@@stringsize	返回从 toString() 方法返回的字符数据的数量。缺省值为 50，最大值为 2GB。值 0 指定缺省值。有关详细信息，请参见 *《组件集成服务用户指南》*。
@@sys_tempdbid	返回正在执行的实例的有效本地系统临时数据库的数据库 ID
@@system_busy	Adaptive Server 正在运行系统任务的时钟周期数 [1]
@@system_view	返回特定于会话的系统视图设置 （“实例”或 “集群”）
@@tempdbid	返回为会话指定的临时数据库的有效临时数据库 ID (dbid)。
@@textcolid	返回 *@@textptr* 所引用列的列 ID。
@@textdataptnid	返回一个文本分区的分区 ID，该文本分区包含由 *@@textptr* 引用的列。
@@textdbid	返回一个数据库的数据库 ID：该数据库包含的对象具有被 *@@textptr* 引用的列。
@@textobjid	返回一个对象的对象 ID，该对象包含被 *@@textptr* 引用的列。
@@textptnid	返回一个数据分区的分区 ID，该分区包含由 @@textptr 引用的列。
@@textptr	返回某个进程上次插入或更新的 text、 unitext 或 image 列的文本指针 （与 textptr 函数不同）。
@@textptr_parameters	如果 textptr_parameters 配置参数的当前状态为关闭，则返回 0。如果 textptr_parameters 的当前状态为打开，则返回 1。有关详细信息，请参见 *《组件集成服务用户指南》*。
@@textsize	返回 select 返回的 text、unitext 或 image 数据的字节数限制。isql 的缺省限制为 32K 字节；缺省值取决于客户端软件。它可以用 set textsize 为某个会话进行更改。
@@textts	返回 *@@textptr* 所引用列的文本时间戳。
@@thresh_hysteresis	返回激活阈值所要求的可用空间的减小量。此数量也称作停滞值 （以 2K 数据库页为单位）。它确定阈值在数据库段中可放置的接近程度。
@@timeticks	返回每个时钟周期中的微秒数。每个时钟周期的时间量与计算机有关。
@@total_errors	返回 Adaptive Server 读写数据时检测到的错误数。
@@total_read	返回 Adaptive Server 的磁盘读取操作的数目。
@@total_write	返回 Adaptive Server 的磁盘写入操作的数目。
@@tranchained	如果 Transact-SQL 程序的当前事务模式为非链式，则返回 0。如果 Transact-SQL 程序的当前事务模式为链式，则返回 1。
@@trancount	返回当前用户会话中事务的嵌套级别。

全局变量	定义
@@transactional_rpc	如果对远程服务器的 RPC 操作是事务型的，则返回 0。如果对远程服务器的 RPC 操作不是事务型的，则返回 1。请参见 *《参考手册》*中的 enable xact coordination 和 set option transactional_rpc。另外，请参见 *《组件集成服务用户指南》*。
@@transtate	返回当前用户会话执行某个语句之后事务的当前状态。
@@unicharsize	返回 2，它是 unichar 类型的字符的大小。
@@user_busy	Adaptive Server 正在运行用户任务的时钟周期数 [1]
@@version	返回 Adaptive Server 当前版本的日期、版本字符串等信息。
@@version_as_integer	返回 Adaptive Server 当前版本的最近升级版本的版本号，用整数表示。例如，如果运行的是 Adaptive Server 版本 12.5、 12.5.0.3 或 12.5.1，*@@version_as_integer* 将返回 12500。
@@version_number	返回 Adaptive Server 当前版本的完整版本号，用整数表示。

[1] *@@user_busy* + *@@system_busy* 的值应等于 *@@cpu_busy* 的值

在集群环境中使用全局变量

对于 *@@servername*， Cluster Edition 返回集群的名称，而非实例名称。使用 *@@instancename* 可返回实例的名称。

在非集群 Adaptive Server 环境中， *@@identity* 的值对于每条插入的记录是不同的。如果插入的最新记录包含一个具有 IDENTITY 属性的列，则 *@@identity* 将设置为此列的值，否则它将设置为 “0”（无效值）。此变量因会话而异，并且基于此会话期间发生的最后一个插入操作获取其值。

在集群环境中，多个节点对表执行插入，因此不会为 *@@identity* 保留特定于会话的行为。在集群环境中， *@@identity* 的值取决于在当前会话的节点中插入的最后一条记录，而非在集群中插入的最后一条记录。

第 4 章

表达式、标识符和通配符

本章介绍了 Transact-SQL 表达式、有效标识符和通配符。

涉及的主题包括：

表达式

表达式是由运算符分隔的一个或多个常量、文字、函数、列标识符和 / 或变量的组合，它将返回一个值。表达式可有几种类型，包括**算术表达式**、**关系表达式**、**逻辑表达式**（或**布尔表达式**）和**字符串表达式**。在某些 Transact-SQL 子句中，可在表达式中使用子查询。在表达式中可使用 case 表达式。

表 4-1 列出了在 Adaptive Server 语法语句中使用的表达式类型。

表 4-1：在语法语句中使用的表达式类型

使用情况	定义
expression	可包括常量、文字、函数、列标识符、变量或参数
逻辑表达式	返回 TRUE、 FALSE 或 UNKNOWN 的表达式
常量表达式	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值表达式
char_expr	任何返回单个字符型值的表达式
binary_expression	返回单个 binary 或 varbinary 值的表达式

表达式的大小

返回二进制数据或字符数据的表达式最长可为 16384 个字节。但是，Adaptive Server 的早期版本只允许表达式最多包含 255 个字节。如果您已经从 Adaptive Server 的早期版本升级，而且您的存储过程或脚本存储的结果字符串最多可为 255 个字节，则其余部分将被截断。考虑到表达式的长度会有所增加，您可能必须重新编写这些存储过程和脚本。

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

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

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

expression [not] in *expression*

[not] exists *expression*

expression [not] between expression and *expression*

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

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

expression is [not] null

not *logical_expression*

logical_expression {and | or} *logical_expression*

运算符优先级

运算符的优先级如下，其中 1 是最高级别， 6 是最低级别：

1　一元运算符 （单个参数） - + ~

2　* / %

3　二元运算符 （两个参数） + - & | ^

4　非

5　与

6　或

当表达式中的所有运算符属于同一级别时，执行顺序为从左向右。您可以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 4-2：算术运算符

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

加法、减法、除法和乘法可用于精确数值、近似数值和货币类型列。

不能对 smallmoney 或 money 列使用模运算符。模运算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 TSQL 中，模运算的结果与被除数具有相同的符号。例如：

```
1> select -11 % 3, 11 % -3, -11 % -3
2> go
-------------- ----------- ----------- -----------
            -2           2          -2

(1 row affected)
```

在对混合数据类型 （例如 float 和 int）执行算术运算时， Adaptive Server 将按照特定规则确定结果的类型。有关详细信息，请参见第 1 章 “系统数据类型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 true ； 值 0 对应 false。

表 4-3 总结了操作数 0 和 1 的结果。如果任一操作数为 NULL，则逐位运算符将返回 NULL：

表 4-3：逐位运算真值表

& （与）	1	0
1	1	0
0	0	0
| （或）	1	0
1	1	1
0	1	0
^ （异或）	1	0
1	0	1
0	1	0
~ （非）		
1	FALSE	
0	0	

表 4-4 中的示例使用两个 tinyint 参数 A 和 B，其中 A = 170（二进制形式为 10101010）、 B = 75 （二进制形式为 01001011）。

表 4-4：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 都变为 0，所有 0 都变为 1。

字符串并置运算符

可以使用 + 和 || （双竖线）字符串运算符并置两个或更多字符或二元表达式。例如，在列标题 Name 下按照先姓后名的顺序显示作者姓名，姓后加一个逗号；例如 “Bennett, Abraham”：

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

下面这个示例的结果为 "abcdef", "abcdef"：

```
select "abc" + "def", "abc" || "def"
```

下面会返回字符串 “abc def” 。空字符串可诠释为所有 char、 varchar、unichar、nchar、nvarchar 和 text 并置以及 varchar 和 univarchar 插入和赋值语句中的一个空格：

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

在并置非字符、非二元表达式时，应始终使用 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 4-5 中列出的比较运算符：

表 4-5：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
!=	（Transact-SQL 扩展）不等于
!>	（Transact-SQL 扩展）不大于
!<	（Transact-SQL 扩展）不小于

在比较字符数据时，< 表示更接近服务器排序顺序的开头，而 > 表示更接近服务器排序顺序的结尾。如果排序顺序不区分大小写，则大写字母与小写字母等同。可使用 sp_helpsort 查看 Adaptive Server 的排序顺序。为便于比较，尾随空白将被忽略。因此，举例来说，“Dirk”与“Dirk ”是相同的。

比较日期时，< 表示较早而 > 表示较晚。

应给所有使用比较运算符的字符和 datetime 数据加上单引号或双引号：

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

非标准运算符

下列运算符是 Transact-SQL 扩展：

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

使用 *any*、 *all* 和 *in*

any 与 <、> 或 = 及子查询一起使用。如果在子查询中检索到的任何值与外层语句的 where 或 having 子句中的值相匹配，就会返回结果。有关详细信息，请参见 *《Transact-SQL 用户指南》*。

all 与 < 或 > 以及子查询一起使用。当在子查询中检索到的所有值都小于 (<) 或大于 (>) 外层语句的 where 或 having 子句中的值时，它就会返回结果。有关详细信息，请参见 *《Transact-SQL 用户指南》*。

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，请参见 *《参考手册：命令》*中的 where clause。

否定和测试

not 否定关键字或逻辑表达式的含义。

使用后跟子查询的 exists 可测试是否存在特定的结果。

范围

between 是范围开始关键字； and 是范围结束关键字。下面的语句表示介于起始界限之间的范围 （包括界限值）：

```
where column1 between x and y
```

下面的范围不包括界限值：

```
where column1 > x and column1 < y
```

在表达式中使用空值

在对定义为允许空值的列的查询中，可以使用 is null 或 is not null。

在使用逐位运算符或算术运算符的表达式中，如果任何操作数为空值，则该表达式的求值结果为 NULL。例如，如果 *column1* 为 NULL，则以下表达式的求值结果为 NULL：

```
1 + column1
```

返回 TRUE 的比较运算

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。此例外为用空的缺省参数来编写存储过程提供了方便。
- *expression* != *n*，其中 *n* 是不包含 NULL 的文字，并且 *expression* 求值结果为 NULL。

当表达式求值结果不为 NULL 时，这些表达式的否定形式将返回 TRUE：

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

注释 这些例外的最右侧是文字 Null 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），则整个表达式的求值结果为 NULL。

按照这些规则，空列值不与其它空列值连接。如果在 where clause 中将 NULL 列值与其它 NULL 列值进行比较，则无论使用什么比较运算符，总会为 NULL 值返回 UNKNOWN，并且结果中不包括行。例如，以下查询不返回 column1 在两个表中都包含 NULL 的结果行（虽然它可能返回其它行）：

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

FALSE 和 UNKNOWN 之间的区别

虽然 FALSE 和 UNKNOWN 都不返回值，但它们之间却存在较大的逻辑差异，因为 false 的相反值 (“not false”) 是 true。例如，“1 = 2”求值为 false，而它的相反运算 “1 != 2”求值为 true。但 “not unknown”仍然是 unknown。如果在比较运算中包括空值，则不能通过否定表达式来获得行的相对集合或相对真值。

将 “NULL”作为字符串使用

只有在 create table 语句中指定了 NULL，且已显式输入了 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。请改用 “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。

NULL 与空字符串比较

空字符串 （" " 或 '' ）总是作为单个空格存储在变量和列数据中。以下并置语句等同于 “abc def”，而不是 “abcdef”：

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

空字符串的求值结果从不会是 NULL。

连接表达式

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

如果在一个语句中使用了多个逻辑运算符，则先对 and 求值，后对 or 求值。可使用小括号更改执行顺序。

表 4-6 显示了逻辑运算 （包括涉及空值的运算）的结果。

表 4-6：逻辑表达式真值表

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

结果 UNKNOWN 表示一个或多个表达式求值结果为 NULL，并且不能确定运算结果是 TRUE 还是 FALSE。有关详细信息，请参见第 345 页上的 “在表达式中使用空值”。

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“expression”作为语法语句中的变量提供时，就会被视为简单表达式。只有在逻辑表达式可接受时，才指定 “逻辑表达式”。

比较字符表达式

字符常量表达式被视为 varchar。如果将它们与非-varchar 变量或列数据进行比较，将在比较中使用数据类型优先级规则 （即，将优先级较低的数据类型转换为优先级较高的数据类型） 。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，应按照数据类型优先规则：将“较低级”的数据类型转换为 “较高级”的数据类型。为便于比较，将所有 varchar 表达式都转换为 char（即添加尾随空白）。如果将 unichar 表达式与 char（varchar、nchar、nvarchar）表达式比较，则后者将被隐式转换为 unichar。

使用空字符串

空字符串 ("") 或 ('') 被解释为 insert 中或者 varchar 或 univarchar 数据上赋值语句中的单个空格。在 varchar、 char、 nchar、 nvarchar 数据的并置中，空字符串被诠释为单个空格；在以下示例中存储为 “abc def”：

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

空字符串的求值结果从不会是 NULL。

在字符表达式中包括引号

在 char 或 varchar 条目中，有两种方法可以指定文字引号。第一种方法是使用双重引号。例如，如果在某个字符条目开头用了一个单引号，并要将另一个单引号作为该条目的一部分，则可使用两个单引号：

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

使用双引号：

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

第二个方法是用另一种引号将引述内容括起来。也就是说，用单引号括起包含双引号的条目，反之亦然。下面是一些示例：

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

使用延续字符

若要让一个字符串延续到屏幕的下一行，可在转至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。

对于常规标识符，对象名或标识符的长度限制为 255 个字节，对于分隔标识符则为 253 个字节。此限制适用于大多数用户定义的标识符，包括表名、列名、索引名等。由于这些扩展限制，一些系统表（目录）和内置函数也进行了扩展。

对于变量，“@”计为 1 个字节，它允许的名称长度为 254 个字节。

下面列出了受这些限制影响的标识符、系统表和内置函数。

现在，这些标识符的最大长度为 255 字节。

- 表名
- 列名
- 索引名
- 视图名
- 用户定义的数据类型
- 触发器名
- 缺省名称
- 规则名
- 约束名
- 过程名称
- 变量名
- JAR 名称
- LWP 或动态语句的名称
- 函数名
- 时间范围的名称
- 应用程序环境名

无论是使用单字节字符还是多字节字符，大多数用户定义的 Adaptive Server 标识符最长都可以为 255 个字节。其它的最长可以为 30 个字节。有关 255 个字节长的标识符和 30 个字节长的标识符的列表，请参见*《Transact-SQL 用户指南》*。

标识符的首字符必须是字母字符 （按当前字符集中的定义），或是下划线 (_) 字符。

注释　临时表名以井号 (#) 开头，变量名以 at 符 (@) 开头，它们是此规则的例外。

后续字符可以包括字母、数字、符号 #、@、_ 以及货币符号（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。有关保留字的完整列表，请参见第 5 章 “保留字”。

不能将破折号 (—) 用作标识符。

短标识符

这些标识符的最大长度为 30 字节：

- 游标名称
- 服务器名
- 主机名
- 登录名
- 口令
- 主机进程标识
- 应用程序名
- 初始语言名称
- 字符集名
- 用户名
- 组名
- 数据库名称
- 逻辑设备名
- 段名
- 会话名称

- 执行类名
- 引擎名
- 停顿标记名称
- 高速缓存名

以 # 开头的表 （临时表）

名称以井号 (#) 开头的表是临时表。您不能创建其它类型的名称以井号开头的对象。

Adaptive Server 对临时表名执行特殊操作以在每个会话中保持唯一命名。当您创建一个名称长度小于 238 字节的临时表时， tempdb 中的 sysobjects 名称会增加 17 个字节以使该表名唯一。如果表名长度大于 238 字节， sysobjects 中的临时表名会只使用前 238 个字节，然后再增加 17 个字节以使它唯一。

在 Adaptive Server 15.0 版之前，sysobjects 中的临时表名为 30 个字节。如果您使用长度小于 13 字节的表名，则会用下划线 (_) 将该表名补至 13 字节，然后再补充 17 个字节的其它字符使该表名达到 30 字节。

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，请参见 *《系统管理指南》*。在实用程序选项中，大小写十分重要。

如果安装 Adaptive Server 时设置了不区分大小写- 的排序顺序，当已经存在名为 MyTable 或 mytable 的表时，不能创建名为 MYTABLE 的表。同样，以下命令将从 MYTABLE、MyTable、mytable 或使用该名称中大小写字母的任意组合的表中返回行。

```
select * from MYTABLE
```

对象名称的唯一性

数据库中的对象名称不必唯一。但是，表中的列名和索引名必须唯一，并且其它对象名称对于 *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

分隔标识符是加双引号的对象名称。使用分隔标识符可避免对象名称的某些限制。在 Adaptive Server 的早期版本中，只有表名、视图名和列名可用引号来分隔，而其它对象名称则不能。从 Adaptive Server 15.7 开始这已经发生了变化，但启用该功能需要设置一个配置参数。

分隔标识符可以是保留字，可以使用非字母字符开头，并且可以包括在其它情况下不允许使用的字符。分隔标识符不能超过 253 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，必须执行以下命令：

```
set quoted_identifier on
```

每次在语句中使用分隔标识符时，必须将其括在双引号内。例如：

```
create table "1one"(col1 char(3))
create table "include spaces" (col1 int)
create table "grant"("add" int)
insert "grant"("add") values (3)
```

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

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

不要使用：

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

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

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

包括引号的语法

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

```
set quoted_identifier on
create table '1one' (c1 int)
```

但是， object_id()() 需要字符串，因此，必须给表名加上引号才能选择信息：

```
select object_id('1one')
```

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

您可以通过使引号加倍出现，从而在带引号的标识符中包括一个嵌入的双引号：

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

但是，在语句语法要求对象名表示为字符串时，无需使引号加倍出现：

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

启用带引号的标识符

quoted identifier enhancement 配置参数允许 Adaptive Server 将带引号的标识符用于：

- 表
- 视图
- 列名
- 索引名 （Adaptive Server 15.7 版和更高版本）
- 系统过程参数 （Adaptive Server 15.7 版和更高版本）

quoted identifier enhancement 是 enable functionality group 的一部分，其缺省设置取决于 enable functionality group 配置参数的设置。请参见*《系统管理指南，第 1 卷》*。要启用带引号的标识符：

1 将 enable functionality group 或 quoted identifier enhancement 配置参数设置为 1。例如：

```
sp_configure "enable functionality group", 1
```

必须重新启动 Adaptive Server 才能使该更改生效。

2 对当前会话打开 quoted_identifier：

```
set quoted_identifier on
```

启用 quoted identifier enhancement 后，当您在对象定义中包括带引号的标识符时，查询处理器会从对象定义中删除分隔符和尾随空格。例如，Adaptive Server 会认为 `"ident"`、`[ident]` 和 `ident` 是相同的。如果不启用 quoted identifier enhancement，则 `"ident"` 会被认为是与其它两个标识符完全不同的标识符。

在启动启用了 quoted identifier enhancement 的 Adaptive Server 时：

- 如果您在重新启动启用了 enable functionality group 配置参数的 Adaptive Server 之前用带引号的标识符创建了一些对象，则当您启动启用了该参数的服务器之后，这些对象不是自动可访问的；反之亦然。也就是说，Adaptive Server 不会自动重命名所有数据库对象。

  但您可以使用 sp_rename 来手动重命名对象。例如，如果您创建一个名为 `"ident"` 的对象，然后重新启动启用了 enable functionality group 的 Adaptive Server，则可通过发出以下命令来重命名该对象：

  ```
  sp_rename '"ident"', 'ident'
  ```

- Adaptive Server 会将 `[tab.dba.ident]` 和 `"tab.dba.ident"` 视为完全限定名。
- 任何接受对象标识符的 Transact-SQL 语句、函数以及系统或存储过程也会认可分隔标识符。
- valid_name 函数会将符合一般规则的标识符字符串与符合分隔标识符规则的标识符字符串区分开来，并返回一个非零值以指示一个有效的名称。

  例如，由于 `'ident/v1'` 只有作为分隔标识符时才有效，因此 `valid_name('ident/v1')` 会返回 true（零）。不过，由于 `'ident'` 作为分隔标识符或普通标识符时都有效，因此 `valid_name('ident')` 会返回非零值。

- 标识符限制使用 253 个字符（28 个字节）（不启用 quoted identifier enhancement 时的长度为 255 个字符（30 个字节））。分隔标识符的有效长度包括分隔符以及任何嵌入空格或尾随空格。

注释　Sybase 建议您避免使用分隔标识符区域（254 – 255 个字符或 29 – 30 个字节长）无法表示的传统标识符。Adaptive Server 及其子系统有时会通过向标识符中添加分隔符来构造内部 SQL 语句。

- Sybase 不建议您在标识符中使用圆点和分隔符，因为 Adaptive Server 如何解释 varchar 字符串中的双引号与标识符有关。
- 如果标识符与 Adaptive Server 之外的项目有关系，则还存在以下约束：
  - 标识符必须以字母字符开头，后跟字母数字字符或几个特殊字符（$、#、@、_、¥、£）。此外：
    - SQL 变量可将 @ 作为首字符。
    - 临时对象（tempdb 中的对象）可将 # 作为首字符。
  - 不能将保留字用作标识符。请参见第 5 章 “保留字”。

- 分隔标识符不需要符合传统标识符规则，但必须通过对称的中括号或双引号来进行分隔。
- 不能将分隔标识符用于变量或标签。
- 必须启用 set quoted_identifier 才能使用带引号的标识符。启用 set quoted_identifier 后，必须用单引号 （而非双引号）将 varchar 字符串文字引起来。
- 包含标识符的 varchar 字符串文字不能包括分隔符字符。
- 分隔标识符不能以井号 (#) 开头。 Sybase 也不建议：
  - 以 (@) 开头
  - 包含空格
  - 包含圆点字符 (.) 或分隔符字符："、 [或]
- 必须从分隔标识符中去除尾随空格，并且不允许使用零长度标识符。

通过限定对象名来标识表或列

通过增加其它限定性名称 （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。每个限定符与下一个限定符之间用句点分隔。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，则可给限定对象名称的各个部分加上双引号。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"
```

不要使用：

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

省略所有者名称

只要系统获得了标识对象所需的足够信息，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自身的对象

在当前数据库中引用自身的对象时，不需要使用数据库名称或所有者名称。 *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所有者名称来限定该对象，Adaptive Server 会试图从当前数据库内您拥有的对象中查找该对象。

引用数据库所有者拥有的对象

如果您省略所有者名称但不拥有具有该名称的对象， Adaptive Server 将查找数据库所有者拥有的具有该名称的对象。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他用户拥有的对象。

保持使用限定标识符的一致性

在同一语句中限定列名和表名时，一定要对每个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。示例 2 不正确，因为列名的语法样式与表名的语法样式不匹配。

示例 1

```
select demo.mary.publishers.city
from demo.mary.publishers

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

示例 2

```
select demo.mary.publishers.city
from demo..publishers
```

列前缀“`demo.mary.publishers`”与查询中使用的表名或别名名称不匹配。

确定标识符是否有效

更改字符集之后或者创建表或视图之前，使用系统函数 valid_name 来确定 Adaptive Server 是否可以接受该对象名。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

使用 sp_rename 重命名用户对象 （包括用户定义的数据类型）。

警告！ 重命名表或列后，必须重新定义所有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，有多种字符可用于标识符。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，但建议不要在异构系统中使用它们。这些字符不能在 EUC-JIS 和 Shift-JIS 字符集之间转换。

对于一些 8 位的欧洲字符也是如此。例如，OE 连字是 Macintosh 字符集的一部分（码点 0xCE）。ISO 8859-1 (iso_1) 字符集中不存在该字符。如果从 Macintosh 字符集转换为 ISO 8859-1 字符集的数据中存在 OE 连字，则会导致转换错误。

如果对象标识符包含无法转换的字符，客户端就无法直接访问该对象。

like 模式匹配

Adaptive Server 允许您逐个处理 like 模式匹配算法中的中括号。

例如，要在较早版本的 Adaptive Server 中将某个行与 '[XX]' 匹配，需要使用：

```
select * from t1 where f1 like '[[]XX[]]'
```

但也可以使用：

```
select * from t1 where f1 like '[[]XX]'
```

由于需要完全兼容，此功能仅在 Adaptive Server 15.7 及更高版本中可用，具体做法是启用以下命令：

```
sp_configure "enable functionality group", 1
```

如果不启用此功能，中括号的 like 模式匹配行为就和 Adaptive Server 15.7 之前版本中一样。

当您启用此功能时：

- like 模式匹配允许在左中括号（“[”）后面紧跟着右中括号（“]”）来表示它自己，因此模式 “[]]”与字符串 “]”相匹配。
- 初始尖号 (“^”) 颠倒所有字符范围的含义，以使模式 “[^]]”与任何不是 “]”的单个字符串相匹配。
- 在任何其它位置，结束中括号 (“]”) 标记字符范围的结束。

启用此功能时可用的模式有：

模式	匹配项
“[[]”	“[”
“[]]”	“]”
“]”	“]”
“[[]XX]”	“[XX]”
“[[]XX[]]”	“[XX]”

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：它们在 authors 表中查找不以区号 415 开头的所有电话号码。

```
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%"
```

若要查找*不是*系统表的所有对象，请使用：

```
 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 （请参见第 363 页上的 “尖号 (^) 通配符”）。您不能总是用 like 和 ^ 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 一次只可对一个字符求值。

诸如 like "[^s][^y][^s]%" 的模式可能不会产生相同的结果。因为从结果中排除了所有首字母是 “s”*或*第二个字母是 “y”*或*第三个字母是 “s”的名称及系统表名称，所以最终结果可能是 14，而不是 19。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）。如果在计算过程的任一点匹配失败，它将被排除。

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、变量和列名或者并置表达式的任意组合，例如：

```
like @variable + "%".
```

如果匹配字符串是常量，则必须始终为其加上单引号或双引号。

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串和日期字符串。不能使用 like 搜索秒或毫秒。有关详细信息，请参见第 365 页上的 “将通配符用于 datetime 数据”。

在 where 和 having 子句中使用通配符查找匹配字符串 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%"
```

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，则在比较*expression* 和 *match_string* 时将忽略大小写。例如，如果 Adaptive Server 不区分大小写，则以下子句将返回 “Smith”、“smith”和 “SMITH”：

```
where col_name like "Sm%"
```

如果 Adaptive Server 还不区分变音，则会将所有变音字符及与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

可以在匹配字符串中使用多个通配符，以下各节将详细讨论这些通配符。表 4-7 对通配符进行了总结：

表 4-7：与 like 一同使用的通配符

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

给通配符和匹配字符串加上单引号或双引号 (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 表中查找所有区号以 415 开头的电话号码：

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

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，后面跟有两个空格的“%”与“X ”（一个空格）匹配；“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

使用下划线 (_) 通配符表示任何单个字符。例如，查找所有以 “heryl” 结尾的六个字母的姓名 （例如 Cheryl）：

```
select au_fname
from authors
where au_fname like "_heryl"
```

使用中括号 ([ ]) 括起的字符

使用中括号括起字符范围 （如 [a-f]）或字符集 （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”与 7 位 ASCII 形式的 0-9、 A-Z 和 a-z （及若干标点字符）匹配。

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"
```

查找 “DeFrance”和 “deFrance”：

```
select au_lname
from authors
where au_lname like "[dD]eFrance"
```

使用加中括号的标识符创建对象 （例如使用 create table [*table_name*] 或 create dstabase [*dbname*]）时，必须至少包含一个有效字符。

会从对象名中删除加中括号的标识符内的所有尾随空格。例如，执行以下 create table 命令会得到相同的结果：

- create table [tab1<space><space>]
- create table [tab1]
- create table [tab1<space><space><space>]
- create table tab1

此规则适用于可使用加中括号的标识符创建的所有对象。

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。

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]”与 7 位 ASCII 形式的 0-9、 A-Z、 a-z 和几个标点字符匹配。

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一起使用时， Adaptive Server 将按字面意义解释通配符，而不是使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
- 紧跟在 escape 子句后面的任何单个字符 （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 4-8 显示了将中括号与 like 一起用作转义字符的示例。

表 4-8：使用中括号搜索通配符

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

使用 *escape* 子句

使用 escape 子句指定转义字符。可使用服务器缺省字符集中的任意一个字符作为转义字符。如果试图使用多个字符作为转义字符，则 Adaptive Server 会生成例外。

不要使用现有通配符作为转义字符，原因是：

- 如果将下划线 (_) 或百分号 (%) 指定为转义字符，它将在该 like 谓词内失去本身的特殊含义，并且仅充当转义字符。
- 如果将左中括号或右中括号（ [或] ）指定为转义字符，则在该 like 谓词中会禁用括号的 Transact-SQL 含义。
- （如果将连字符- ）或尖号 (^) 指定为转义字符，则它会失去它的特殊含义并只充当转义字符。

不同于通配符 （如下划线、百分号和左括号），转义字符在中括号中会保留它的特殊含义。

转义字符仅在它的 like 谓词中有效，且对同一语句中包含的其它 like 谓词没有影响。在转义字符之后，只有通配符（_、 %、 [、] 或 [^]）及转义字符本身才有效。转义字符只影响它后面的字符，而不会影响后续字符。

如果字符在某一模式中作为转义字符出现两次，则该字符串必须包含四个连续的转义字符。如果转义字符未将模式划分为由一个或两个字符组成的多个部分，则 Adaptive Server 会返回错误消息。表 4-9 显示与 like 一起使用的 escape 子句的示例。

表 4-9：使用 escape 子句

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

将通配符用于 *datetime* 数据

将 like 与 datetime 值一起使用时，Adaptive Server 会将日期转换为标准的 datetime 格式，然后再转换为 varchar。由于标准存储格式不包括秒或毫秒，所以不能用 like 和某一模式来搜索秒或毫秒。

最好使用 like 来搜索 datetime 值，因为 datetime 条目可包含多种日期分量。例如，如果在名为 arrival_time 的列中插入值 “9:20”和当前日期，则子句：

```
where arrival_time = '9:20'
```

将找不到该值，这是因为 Adaptive Server 将该条目转换成了“Jan 1 1900 9:20AM”。但是，下面的子句可以找到它：

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

第 5 章

保留字

关键字也称为保留字，它们是具有特殊意义的字词。本章列出了 Transact-SQL 和 ANSI SQL 的关键字。

涉及的主题包括：

Transact-SQL 保留字

表 5-1 中的字词是 Adaptive Server 保留的关键字（是 SQL 命令语法的一部分）。这些字词不能用作数据库对象（例如，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。

若要查找作为保留字的现有对象名称，请使用 sp_checkreswords 参考手册：过程中的*过程》*。

表 5-1：Transact-SQL 保留字列表

	保留字
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、 compressed、 compute、 confirm、 connect、 constraint、 continue、 controlrow、 convert、 count、 count_big、 create、 current、 cursor
D	database、 dbcc、 deallocate、 declare、 decrypt、 default、 delete、 desc、 deterministic、 disk、 distinct、 drop、 dual_control、 dummy、 dump
E	else、 encrypt、 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

	保留字
I	identity、 identity_gap、 identity_start、 if、 in、 index、 inout、 insensitive、 insert、 install、 intersect、 into、 is、 isolation
J	jar、 join
K	key、 kill
L	level、 like、 lineno、 load、 lob_compression、 lock
M	materialized、 max、 max_rows_per_page、 min、 mirror、 mirrorexit、 modify
N	national、 new、 noholdlock、 nonclustered、 not、 null、 nullif、 numeric_truncation 注释尽管 “new”不是 Transact-SQL 保留字，但它将来可能会成为保留字，因此 Sybase 建议避免使用它 （例如，用它命名数据库对象）。“New”是一个特例 （有关其它保留字的信息，请参见第 369 页上的 “可能的 ANSI SQL 保留字”），因为它出现在 spt_values 表中，同时 sp_checkreswords 将它显示为保留字。
O	of、 off、 offsets、 on、 once、 online、 only、 open、 option、 or、 order、 out、 output、 over
P	partition、 perm、 permanent、 plan、 prepare、 primary、 print、 privileges、 proc、 procedure、 processexit、 proxy_table、 public
Q	quiesce
R	raiserror、 read、 readpast、 readtext、 reconfigure、 references、 release_locks_on_close、 remove、 reorg、 replace、 replication、 reservepagegap、 return、 returns、 revoke、 role、 rollback、 rowcount、 rows、 rule
S	save、 schema、 scroll、 select、 semi_sensitive、 set、 setuser、 shared、 shutdown、 some、 statistics、 stringsize、 stripe、 sum、 syb_identity、 syb_restree、 syb_terminate
T	table、 temp、 temporary、 textsize、 to、 tracefile、 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
X	xmlextract、 xmlparse、 xmltest

ANSI SQL 保留字

Adaptive Server 包含了初级的 ANSI SQL 功能。ANSI SQL 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。因此，为方便起见，我们提供了以下列表。这一信息的公布并不表示 Sybase 承诺将在后续版本中提供所有这些 ANSI SQL 功能。另外，后续版本可能包含此列表中未列出的关键字。

表 5-2 中的字词是 ANSI SQL 关键字，但不是 Transact-SQL 的保留字。

表 5-2：ANSI SQL 保留字列表

	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、 semi_sensitive、 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

可能的 ANSI SQL 保留字

如果您使用的是 ISO/IEC 9075:1989 标准，还应避免使用下面的列表中所示的字词，因为它们以后可能会成为 ANSI SQL 保留字。

表 5-3：可能的 ANSI SQL 保留字列表

	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

第 6 章

SQLSTATE 代码和消息

本章介绍 Adaptive Server 的 SQLSTATE 状态码及其相关消息。

涉及的主题包括：

SQLSTATE 代码是达到初级 ANSI SQL 标准所必需的内容。它们提供有关以下两种情况的诊断信息：

- *警告*：需要用户注意但尚未严重到妨碍 SQL 语句成功执行的情况
- *例外*：使 SQL 语句无法对数据库产生影响的情况

每个 SQLSTATE 代码都由一个双字符类后跟一个三字符子类组成。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，一同存储的还有在检测到这些情况时将显示的消息。并非所有 Adaptive Server 错误情况都与 SQLSTATE 代码相关，只有 ANSI SQL 所指定的那些错误情况才与其相关。在某些情形中，会有多个 Adaptive Server 错误情况与一个 SQLSTATE 值相关。

警告

Adaptive Server 目前检测以下 SQLSTATE 警告情况，表 6-1 对这些情况进行了说明：

表 6-1：SQLSTATE 警告

消息	值	说明
Warning – null value eliminated in set function.（警告 — set 函数中的 NULL 值已消除。）	01003	发生条件是：对具有 NULL 值的表达式使用集合函数（avg、max、min、sum 或 count）。

消息	值	说明
Warning – string data, right truncation （警告 — 字符串数据，右截断）	01004	发生条件是：将字符、 unichar 或二进制数据截断为 255 字节。这些数据可能是： • select 语句的结果，在该语句中客户端不支持 WIDE TABLES 属性。 • 那些不支持 WIDE TABLES 属性的远程 Adaptive Server 或 Open Server 上的 RPC 操作的参数。

例外

Adaptive Server 可以检测到下列类型的例外：

- 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 6-2 到表 6-8 对例外情况进行了说明。每个例外类都显示在它自身的表中。在每个表中，例外情况都按消息文本的字母顺序来排序。

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给嵌入式 SQL™ 应用程序。

表 6-2：基数冲突

消息	值	说明
子查询返回不止一个值。(Subquery returned more than 1 value.) 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。(This is illegal when the subquery follows =, !=, <, <=, >, >=. or when the subquery is used as an expression.)	21000	发生条件： • 标量子查询或行子查询返回不止一行。 • 嵌入式 SQL 中的 select into parameter_list 查询返回不止一行。

数据例外

发生数据例外的条件是：

- 条目就其数据类型而言太长，
- 条目包含非法的转义序列，或
- 条目包含其它格式错误。

表 6-3：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。
数据例外 - 字符串数据从右侧被截断。(Data exception - string data right truncated.)	22001	发生条件是：因 char、 unichar、 univarchar 或 varchar 列太短，不能容纳插入或更新的数据，而必须截断非空字符。
Divide by zero occurred.（发生被零除错误。）	22012	发生条件是：对数值表达式求值而除数的值为零。
发现非法的转义字符。 (Illegal escape character found.) 形成有效字符所需的字节数不够。 (There are fewer bytes than necessary to form a valid character.)	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。
无效的模式字符串。 (Invalid pattern string.) 转义字符后的字符必须是百分号、下划线、左方括号、右方括号或转义字符。 (The character following the escape character must be percent sign, underscore, left square bracket, right square bracket, or the escape character.)	22025	发生条件是：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

完整性约束冲突发生的条件是：insert、 update 或 delete 语句与 primary key、 foreign key、 check 或 unique 约束或唯一索引发生冲突。

表 6-4：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 与列的检查约束发生冲突。
Dependent foreign key constraint violation in a referential integrity constraint.（参照完整性约束中发生相关外键约束冲突。）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 与外键约束发生冲突。
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

无效游标状态的发生条件是：

- 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 6-5：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.（有关详细信息，请使用系统存储过程 sp_cursorinfo。）	24000	发生条件是：试图从从未打开或已被 commit 语句或者隐式（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.（由于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.（游标扫描位置无法恢复。）This happens for cursors which reference more than one table.（游标引用了不止一个表时会发生这种情况。）	24000	发生条件是：多表游标的连接列已被删除或更改。发出另一个 fetch 以重新定位游标。
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，需发出另一个 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：用户对游标发出了 update/delete where current of，而该游标 • 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 6-6：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适当权限的对象。
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.)	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 附近有错误的语法。）	42000	发生条件是：在指定对象附近发现错误的 SQL 语法。
插入错误：(Insert error:) 列名或所提供值的数量与表定义不匹配。(column name or number of supplied values does not match table definition.)	42000	发生条件是：插入时使用了无效的列名或插入的值的个数不正确。
Missing end comment mark '*/'.（缺少结束注释符 '*/'。）	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。）指定 owner.objectname 或使用 sp_help 检查对象是否存在（sp_help 可能会产生很多输出）。(Specify owner.objectname or use sp_help to check whether the object exists (sp_help may produce lots of output).)	42000	发生条件是：用户试图引用他们不具有拥有权的对象。当引用另一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*.（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行大小的最大允许值。 • 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

事务回退的发生条件是：transaction isolation level 设置为 3，但 Adaptive Server 无法保证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）导致的。

表 6-7：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）与另一进程发生死锁并且已被选作死锁牺牲品。）Re-run your command.（请重新运行命令。）	40001	发生条件是：Adaptive Server 检测到它无法保证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 6-8：with check option 冲突

消息	值	说明
插入或更新尝试失败，原因是目标视图是使用 WITH CHECK OPTION 创建的或跨越了另一个用 WITH CHECK OPTION 创建的视图。(The attempted insert or update failed because the target view was either created WITH CHECK OPTION or spans another view created WITH CHECK OPTION.)至少有一个由该命令生成的行不符合 CHECK OPTION 约束。(At least one resultant row from the command would not qualify under the CHECK OPTION constraint.)	44000	发生条件是：视图或该视图所依赖的任何视图是用 with check option 子句创建的。

索引

符号

数字

英文

A

B

C

D

F

G

H

J

K

L

M

N

P

Q

R

S

T

W

X

Y

Z