返回总目录

目　　录

第 五 部 分

参　考　素　材

第 21 章 Transact-SQL 编程语言参考

Transact-SQL 是 Microsoft SQL Server 的编程语言，是结构化查询语言(SQL)的增强版本，SQL 是首先由 IBM 开发的数据库语言。Transact-SQL 可用来从数据库中提取数据，执行 SQL 语言的数据定义(DDL)、数据操作(DML)和数据控制(DCC)等操作。Transact-SQL 将继续朝着 American National Standards Institute(ANSI)标准的方向向前发展，并在许多情况下进一步改进。

Transact-SQL 用下列格式对语法进行分类：

语法格式	说明
UPPERCASE	关键词
斜体字	用户提供的内容
|(竖杠)	选择一个，与单词“or”相对应
[] 方括号	可选的语法
{ } 花括号	必选的语法
[,...n]	向后重复，并以逗号或空格分隔开
<标号> :: =	标记可能有多个长度单位的语法

加号(+)和字符串连接(+)

加号(+)有两种功能，它可以表示加法运算符，把两个数字加到一起或将几天加到日期中。也可使用列名、常数和表达式。

加号的第二种用途是在字符串表达式中用作连接两个或多个字符或二进制字符串的运算符。把字符串放在引号内，就可以使用从非字符到字符或从非二进制到二进制表达式的 CONVERT(转换)或 CAST(加法)函数。

语法

```
expression + expression [+ [...n]]
```

变量

expression	对于此运算符，expression 的数据类型可以是 int(整型)、smallint(短整型)、tinyint(微整型)、numeric(数值)、decimal(小数)、float(实型)、real money(货币型)、smallmoney(小货币型)、datetime(日期时间)以及 smalldatetime(小日期时间)。当进行日期运算时，可提供一个 datetime 或 smalldatetime 表达式；而另一个表达式必须是天数，以便计算日期表达式。对于字符串的连接，将字符串放在引号中，就可以使用从非字符到字符或从非二进制到二进制表达式的 CONVERT(转换)或 CAST(加法)函数。
n	n 表示前面的各项可以重复。

示例

下面这个加号运算符的例子是将两个用数值表达的数加在一起，再将两个用列名表达的数加在一起：

```
SELECT 9＋7
```

和

```
SELECT salary ＋ bonus
FROM mymoneytable
```

在这个例子中，将数值 9 和 7 加在一起，也可以用列名表示数值，并使用 FROM 子句。

减号 (-)

减号是将两数相减的运算符。也可以用减号运算符从日期中减去天数，该运算符可以使用列名、常数和／或表达式。

语法

```
expression-expression [- […n]]
```

变量

expression	对于此运算符，expression 的数据类型可以是 int（整型）、smallint（短整型）、tinyint（微整型）、numeric

（数值）、decimal（小数）、float（实型）、real money（货币型）、smallmoney（小货币型）、datetime（日期时间）以及 smalldatetime（小日期时间）。当进行日期运算时，可提供一个 datetime 或 smalldatetime 表达式；而另一个表达式必须是天数，以便计算日期表达式。

n　　n 表示前面的各项可以重复。

示例

下面是减号运算符的示例。

```
SELECT 4-2
```

本例中，SELECT 语句进行了 4 减 2 的运算。

乘号 (*)

乘号是用于乘法的运算符，它可使用列名、常数、和／或表达式。

语法

```
expression * expression  [* [...n]]
```

变量

expression　　对于此运算符，expression 的数据类型可以是 int

（整型）、smallint（短整型）、tinyint（微整型）、numeric（数值）、decimal（小数）、float（实型）、real money（货币型）以及 smallmoney（小货币型）。

n　　n 表示前面的各项可以重复。

示例

下面是乘法运算符的一个例子：

```
SELECT 7*8
```

本例中，使数字 7 与数字 8 相乘。

除号 (／)

除号是执行除法的运算符，它可使用列名、常数和／或表达式。

语法

```
expression／expression  [／ [… n]]
```

变量

expression　　对于此运算符，expression 的数据类型可以是 int（整型）、smallint（短整型）、tinyint（微整型）、numeric（数值）、decimal（小数）、float（实型）、real

money（货币型）以及 smallmoney（小货币型）。

n　　n 表示前面的各项可以重复。

示例

下面是除法运算符的示例：

```
SELECT 56 ／ 8
```

本例中，数 56 除以 8。

模及通配符（%）

符号“%”用于两种不同的操作。它可以提供一个数除以另一个数后的余数，也可以在用列名、常数和／或表达式查找字符串时用作通配符。用关键词 LIKE 和通配符可查找日期时间、字符和可变字符等数据类型，但不能查询秒。当搜索不同数据部分的日期时间型值时，最好使用 LIKE。如果要搜索“%”，可使用方括号。

语法

百分号(%)可用作模运算符，它返回相除的余数：

```
SELECT dividend % divisor
```

或者作为通配符(%)：

string-expression [NOT] LIKE string_expression

变量

dividend	**模运算(返回相除时的余数)中要相除的数字表达式、整型数、短整型数或微短整型数。**
divisor	**模运算中(返回相除时的余数),要除以 dividend 的数字表达式、整型数、短整型数,或微短整型数。**
string_expression	**括在引号内的字符串和通配符(%)。**
NOT	**NOT 是布尔表达式的否定表示符。**
LIKE	**LIKE 进行模式比较。**

示例

下面是个模运算符返回余数的示例:

SELECT 87 % 9

本例中,数字 87 除以数字 9,返回余数 6。

下面是个通配符(%)的例子:

```
SELECT  *
FROM authors
WHERE name
like  ' Heming% '
```

本例中，模式比较查找字符串中前 6 个字符是 Heming 的名称。

按位与操作(&)

先将整型、短整型或微短整型数据转换成二进制表达式。按位与操作可在二进制表达式上进行操作。

语法

```
expression & expression
```

变量 expression 是转换成二进制的整型、短整型、微短整型数据。

示例

下面是按位与操作的示例：

```
SELECT(integer & integer)
0 0 0 1 0 1 0 1 二进制表示的整数 A
0 1 1 0 1 0 1 1 二进制表示的整数 B
----------------
```

0 0 0 0 0 0 0 1

本例中，当 A 的位是 0，B 的位是 1 时，结果是 0，当 A 的位和 B 的位都为 1 时，结果是 1，A 的位和 B 的位都为 0 时，结果是 0。A 的位为 1，B 的位为 0 时，结果也是 0。

按位或操作(|)

将整型、短整型或微短整型数据转换成二进制表达式。按位或操作可在二进制表达式上进行操作。

语法

expression | expression

变量 expression 是转换成二进制的整型、短整型或微短整型数。

示例

下面是按位或操作的例子：

```
(integer | integer)
1 0 1 0 1 0 1 0          二进制表示的整数 A
1 1 0 0 1 0 1 1          二进制表示的整数 B
-----------------
```

1 1 1 0 1 0 1 1

本例中，当 A 和 B 的位都是 1 时，结果是 1，当 A 的位是 0，B 的位也是 0 时，结果是 0。当 A 的位为 0，B 的位数为 1 时，结果是 1，当 A 的位为 1，B 的位为 0 时，结果是 1。

按位异或操作 (^)

将整型、短整型或微短整型数据转换成二进制表达式，按位异或可在二进制表达式上进行操作。

语法

expression ^ expression

变量 expression 是转换成二进制的整型、短整型、微短整型数据。按位异或返回表达式的数据类型。

示例

下面是按位异或操作的示例：

```
SELECT(A ^ B)
1 0 1 0 1 0 1 0          二进制表示的整数 A
0 1 0 0 1 0 1 1          二进制表示的整数 B
```

```
----------------
1 1 1 0 0 0 0 1
```

本例中，当 A 和 B 的位都是 1 时，结果是 0，当 A 和 B 的位都是 0 时，结果是 0。当 A 的位为 0，B 的位为 1，或 A 的位为 1，B 的位为 0 时，返回值为 1。

按位非操作(～)

将整型、短整型或微短整型数据转换成二进制表达式，按位非可在二进制表达式上进行操作。

语法

SELECT ～ expression

变量 expression 是转换成二进制的整型、短整型或微短整型数。按位非返回表达式的数据类型。由于微短整型、短整型和整型数存储的字节数不同，所以当存储十进制数时，应注意要与原来使用的数据类型一致。

示例

下面是按位非的示例：

```
～ 1 1 0 0 1 1 0 0
```

本例中，返回值为 0 0 1 1 0 0 1 1。所有的 1 变为 0，所有的 0 变为 1。

等于 (=)

Equal 号表示进行比较的两个表达式彼此相等。

语法

```
expression = expression
```

变量 expression 是任意有效的表达式，它可以是变量或列名。只能比较数据类型相同的表达式。

示例

下面是等于比较运算符的例子：

```
SELECT   *
FROM My Table
WHERE column1=column2
```

本例中，表格中的某一列值与另一列值相比较，并选出列值相等的行。

大于 (>)

Great Than 是一个比较运算符，它将两个表达式进行比较，看哪一个更大。

语法

expression > expression

变量 expression 是任意有效的表达式，它可以是变量或列名。只能比较数据类型的表达式。

示例

下面是 Great Than 比较运算符的示例：

```
SELECT *
FROM MyTable
WHERE column1 > column2
```

本例中，表中 column1 列包含的值与 column2 列包含的值相比较，看它是否大于 column2 列的值，如果是，就选择此行。

小于 (<)

Less Than 是个比较运算符，它将两个表达式进行比较，看哪个值小。

语法

expression < expression

变量 expression 是任意有效的表达式，可以是变量或列名，只能比较数据类型相同的表达式。

示例

下面是 Less Than 比较运算符的示例：

```
SELECT  *
FROM MyTable
WHERE column1 < column2
```

本例中，表中 column1 列包含的值与 column2 列值相比较，看它是否小于后者。如果是，则选择此行。

大于或等于 (>=)

Greater Than or Equal To 是一个比较运算符，它将表达式进行比较，看其中一个是否大于或等于另一个。

语法

```
expression >= expression
```

变量 expression 是任意有效的表达式，可以为变量或列名，只能比较数据类型相同的表达式。

示例

下面是 Great Than or Equal To 比较运算符的示例：

```
SELECT  *
FROM MyTable
WHERE column1 >= column2
```

本例中，将表中 column1 列包含的值与 column2 列包含的值进行比较，看它是否大于或等于后者。如果是，就选择此行。

小于或等于 (<=)

Less Than or Equal To 是一个比较运算符，它比较两个表达式，看其中一个表达式是否小于或等于另一个表达式。

语法

```
expression <= expression
```

变量 expression 是任意有效的表达式，它可以是变量或列名。只能比较数据类型相同的表达式。

示例

下面是 Less Then or Equal To 比较运算符的示例：

```
SELECT *
```

```
FROM MyTable
WHERE column1 <= column2
```

本例中，将表格中 column1 列包含的值与 column2 列包含的值相比较，看它是否小于或等于第 2 列包含的值。如果是，则选择此行。

不等于(<>)

Not Equal 号(<>)表示比较的两个表达式彼此不相等。

语法

```
expression <> expression
```

变量 expression 是任意有效的表达式，它可以是变量或列名。只能比较数据类型相同的表达式。

示例

下面是 Not Equal 比较运算符的示例：

```
SELECT *
FROM MyTable
WHERE column1 <> column2
```

本例中，将表中某列的值与另一列的值相比较，选出列值不等的行。

不小于 (!<)

Not Less Than 是一个比较运算符，它用来比较表达式，看某个值是否不小于另一个值。

语法

```
expression !< expression
```

变量 expression 是任意有效的表达式，可以是变量或列名。只能比较数据类型相同的表达式。

示例

下面是 Not Less Than(!<)比较的示例：

```
SELECT  *
FROM MyTable
WHERE column1  !<  column2
```

本例中，将表格中 column1 列包含的值与 column2 列包含的值相比较，看它是否不小于第二列的值。如果是，选择此行。

不等于(!=)

此版本的 Not Equal 号(!=)表示相比较的两个表达式彼此不相等。

语法

```
expression  !=  expression
```

变量 expression 是任意有效的表达式，可以是变量或列名。只能比较数据类型相同的表达式。

示例

下面是 Not Equal 比较运算符的示例：

```
SELECT  *
FROM MyTable
WHERE column1  !=  column2
```

本例中，将表中某一列的值为另一列的值相比较，选出不相等的值。

不大于(!>)

Not Greater Than(!>)是一个比较运算符，它比较两个表达式，看哪一个

更大。

语法

```
expression !> expression
```

变量 expression 是任意有效的表达式，可以是变量或列名，只能比较数据类型相同的表达式。

示例

下面是 Not Greater Than 比较运算符的示例：

```
SELECT   *
FROM MyTable
WHERE column1 !> column2
```

本例中，将表中 column1 列包含的值与 column2 列包含的值相比较，看它是否不大于后者，如果是，则选择此行。

注释 (--)

服务器会忽略注释，但是可以用这个语法将注释放在包含 Transact-SQL 语法的行末，或将注释放在单独的一行中。注释不能包括 GO 语句。

语法

--Whatever your comment is

变量 whatever your comment is可以是任意字符串。

示例

下面是注释的例子：

```
SELECT * --Any character string can be placed on a comment line
FROM MyTable
```

本例中，两条虚线表示注释的开始。SQL Server 将不会检查行中两条虚线后面的部分。

注释(／*...*／)

SQL Server 会忽略注释。可以用此语法将注释放在包含 Transact-SQL 语句的行中间，或将注释放在单独的一行或数行中。注释不能包含 GO 语句。

语法

```
／* Whatever your comment is *／
```

变量 whatever your comment is可以是任意字符串。

示例

下面是注释的示例：

```
／* Any character string can be placed on a comment line
and can span multiple l
ines*／
```

本例中，“／*”表明注释的开始，“*／”表示注释的结束。SQL Server 将不会检查“／*”和“*／”之间的行的内容。

匹配的通配符([ ])

在通配搜寻过程中，方括号([ ])表示 SQL Server 会搜索方括号内的任意字符。方括号内的值也可指定值的范围。

escape 子句提供这样一种机制来通知 SQL 服务器：SQL Server 通常用作通配语法的某些字符串，现在不在通配环境中使用，而是用作字符串中正在搜索的值。换言之，要寻找串中的字符，而不是将该字符用作通配符。

语法

```
string_expression [ NOT ] LIKE string_expression [ ESCAPE 'escape_character' ]
|[NOT] string_expression LIKE string_expression[ESCAPE ' escape_character
```

']

变量

string_expression	string_expression 是字符串和通配符。它可以包括方括号以便在某个范围内搜寻。
NOT	NOT 为布尔表达式的否定指示符。
LIKE	LIKE 进行模式匹配。
ESCAPE 'escape_character'	该变量表示要搜寻串中指定的字符，而不是用作通配句法中的字符。

示例

下面的匹配的通配符示例可搜寻列中某范围内的字母：

```
SELECT *
FROM table2
WHERE column2 LIKE '[C-H]urry'
```

本例中，将选出 column2 列中以字母 C 到 H 开头，以“urry"结尾的任何值。

下面是搜寻列中方括号的例子：

```
CREATE TABLE table1
(column1 char(30))
```

```
INSERT Table1 VALUES ('[Searchforbracket')
INSERT table1 VALUES ('Donotsearchforbracket')
SELECT column1
FROM table1
WHERE column1 LIKE '[[Searchforbracket' ESCAPE '['
```

本例中，ESCAPE 子句和指定的换码字符“[”用于查找列中的字符串“'[Searchforbracket'”。它表示不是将方括号用作通配符，而是寻找列中的方括号。

不匹配的通配符([^])

在通配搜寻过程中，“^”表示 SQL Server 查找不在方括号内的字符。方括号内的值也可指定不在搜寻范围内的数值。

escape 子句提供这样一种机制来通知 SQL 服务器：SQL Server 通常用作通配语法的某些字符串，现在不在通配环境中使用，而是用作字符串中不查找的值。换言之，要寻找不在串中的字符，而不是将该字符用作通配符。

语法

```
string_expression [ NOT ] LIKE string_expression [ ESCAPE 'escape_character' ]
| [ NOT ] string_expression LIKE string_expression [ ESCAPE '
```

escape_character ']

变量

string_expression	**string_expression 是字符串和通配符，以及引号中放置的任意字符串。它可以包括方括号以便在某个范围内搜寻。**
NOT	**NOT 为布尔表达式的否定指示符。**
LIKE	**LIKE 进行模式匹配。**
ESCAPE ' escape_character '	**该变量表示要搜寻串中指定的字符，而不是用作通配句法中的字符。**

示例

下面的不匹配的通配符示例可搜寻列中某范围内的字母：

```
SELECT  *
FROM table2
WHERE column2 NOT LIKE ' [^I-Z] '
```

本例中，将从表中选出 column2 列中不以字母 I 到 Z 开头的任何值。

一个字符匹配的通配符(_)

在通配搜寻过程中，下划线会通知 SQL Server 搜寻后面可以跟随 string_expression 中其余值的单个字符。 escape 子句提供这样一种机制来通知 SQL 服务器：SQL Server 通常用作通配语法的某些字符，现在不在通配环境中使用，而是用作字符串中正在搜索的值。换言之，要寻找串中的字符，而不是将该字符用作通配符。

语法

```
string_expression [ NOT ] LIKE string_expression [ ESCAPE 'escape_character' ]
| [ NOT ] string_expression LIKE string_expression [ ESCAPE 'escape_character' ]
```

变量

string_expression	string_expression 是字符串和通配符，以及放在引号中的任意字符串。它可以包括方括号以便在某个范围内搜寻。
NOT	NOT 为布尔表达式的否定指示符。

LIKE　　LIKE 进行模式匹配。

ESCAPE 'escape_character'　该变量表示要搜寻串中指定的字符，而不是用作通配句法中的字符。

示例

下面的匹配的通配符示例可搜寻后面能接串中其余值的一个字符：

```
SELECT  *
FROM table2
WHERE column2 LIKE '_urry'
```

本例中，将从表中选出后跟字母“urry”的任意字符。

@@connections

此全局变量可保存自 Microsoft SQL Server 最近一次启动以来注册和未成功注册的数目。

语法

```
SELECT @@connections
```

@@cpu_busy

全局变量@@cpu_busy保存了自 Microsoft SQL Server 最近一次启动以来，CPU 执行 SQL Server 命令占用的滴嗒(3.3 千分之一秒)数。

语法

```
SELECT @@ cpu_busy
```

@@cursor_rows

全局变量@@cursor_rows 可保存最近一次启动时游标中的行数。

如果异步占用游标，则返回值(-m)是当前键集内的行数。如果全部占用游标，则返回值(n)为游标中的所有行。如果没有打开游标，返回值为 0(零)。

语法

```
SELECT @@ cursor_rows
```

示例

下面是使用@@cursor_rows 的例子：

```
SELECT @@cursor_rows
```

```
DECLARE cursor1 CURSOR FOR
SELECT customer_name FROM customer
OPEN cursor1
FETCH NEXT FROM cursor1
SELECT @@cursor_rows
CLOSE cursor1
DEALLOCATE cursor1
```

本例中，因为没有打开游标，@@cursor_rows 第一次出现时，返回值为 0。下一次出现@@cursor_rows 时，返回值为-1，因为只有 FETCH NEXT 语句异步占用了一行游标。

@@datefirst

全局变量@@datefirst 包含 SET DATEFIRST 的值，该设置表示 SQL Server 不考虑是每星期的哪一天，其中，星期天返回 1；星期一返回 2；星期二返回 3；星期三返回 4；星期四返回 5；星期五返回 6；星期六返回 7。

语法

```
SELECT @@datefirst
```

@@dbts

全局变量@@dbts保存了当前数据库的当前时间戳。

语法

```
SELECT @@dbts
```

@@error

全局变量@@error 保存了对话过程中执行最后一个 Transact-SQL 语句的错误号。如果执行 Transact-SQL 语句时没有出现错误，@@error 就设置为 0(零)。如果有一个 SQL Server 错误，@@error 就将错误号存储在 sysmessages 系统表中。该变量只将错误号保存到下一个 Transact-SQL 语句开始执行时。只能在执行完一个 Transact-SQL 语句，且还没开始执行下一个 Transact-SQL 语句时查看全局变量@@error。因此，要在语句执行完后立即查看@@error 失误，如果以后在 Transact-SQL 代码中需要@@error 的值，也可以将@@error 存储到一个局部变量中。

语法

```
SELECT @@error
```

示例
下面是将@@error 存储到局部变量中的例子：

```
DECLARE @error_number int
SELECT *
FROM table1
SELECT @error_number = @@error
```

本例中，@@error 中的值存储到局部变量@error_number 中。

@@fetch_status

全局变量@@ fetch_status 可保存游标 FETCH 语句的状态，其中，零(0)表示指读取成功，-1 表示读取失败，-2 表示已读取的行不再有效。
语法

```
SELECT @@fetch_status
```

示例
下面为使用@@fetch_status 的示例：

```
DECLARE cursor1 CURSOR FOR
SELECT customer_name
FROM table1
FETCH NEXT FROM cursor1
WHILE @@fetch_status = 0
BEGIN
FETCH NEXT FROM cursor1
END
CLOSE cursor1
DEALLOCATE cursor1
```

本例中，@@fetch_status 用于控制游标。

@@identity

全局变量@@identity包含最后一次使用的等同值。执行 INSERT 或 SELECT INTO 语句可以改变全局变量@@identity，当将成批拷贝插入到表中时也会改变全局变量@@identity。如果对没有等同列的表格进行操作，对话的@@ identity 就可设置为空。如果操作被回滚，@@identity将不复位，而保持其递增的值。

语法

```
SELECT @@identity
```

示例
下面是使用@@identity的示例：

```
SELECT @@identity
```

本例中，将显示@@identity 的值，在对有全同列的表格进行插入操作后，用此语句可以查看刚才插入到等同列中的值。

@@idle

全程变量@@idle可存储 SQL Server启动后没有使用的 3.33 毫秒数。
示例
下面是使用@@ idle的示例：

```
SELECT @@idle
```

本例中，将显示 SQL Server处于空闲状态的@@idle时间量。

@@io_busy

全局变量@@io_busy可包含 SQL Server启动后执行输入输出操作的 3.33 毫

秒数。

示例

下面是使用@@io_busy的示例：

```
SELECT @@io_busy
```

本例中，@@io-busy 将显示 SQL-Server 最近一次启动以来执行输入输出操作所占用的时间。

@@LANGID

全局变量@@LANGID可返回当前使用的语言的本地语言 ID。

语法

```
@@LANGID
```

@@language

全局变量@@ language可保存 Microsoft SQL Server当前所用语言的

语言名称。

语法

```
SELECT @@language
```

示例

下面是使用@@ language的示例：

```
SELECT @@language
```

本例中，@@language变量将显示 SQL Server 所用的语言名称。

@@lock_timeout

全局变量@@lock_timeout 可保存运行过程中的超时锁定设置，以毫秒为单位。

语法

```
SELECT @@lock_timeout
```

@@max_connections

全局变量@@max_connections 可保存 Microsoft SQL Server 并行联接的最大值，这个最大值是用 sp_configure 运行阶段的值为 SQL Server 配置的。

语法

```
SELECT @@max_connections
```

@@max_precision

全局变量@@max_precision 可保存由小数或数字型数据使用的精度等级，其缺省值为 28。在命令提示行上调用 sqlservr，并使用／p 参数启动 Microsoft Server，就可以改变该变量。

语法

```
SELECT @@max_precision
```

@@nestlevel

全局变量@@nestlevel 可及时存储当前存储过程在某一位置的嵌套级。当一个存储过程调用另一个存储过程时，嵌套级从 0 开始递增，递增的权限值为 32。

语法

```
SELECT @@nestlevel
```

@@options

全局变量@@options 可保存当前对每个用户都有效的 SET 选项信息。用 sp_configure 配置选项“user options”可改变整个设置，并代表@@options，SET 命令可用于改变语言和查询选项。可以用 SET 命令改变选项，然后立刻检查@@options 中的选项值。

语法

```
SELECT @@options
```

@@pack_received

全局变量@@pack_received 可保存输入数据包的数目，此数据包为自Microsoft SQL Server最近一次启动以来接收到的和读取的数据包。

语法

```
SELECT @@ pack_received
```

@@pack_sent

全局变量@@pack_sent 可保存输出数据包的数目，此数据包为自 Microsoft SQL Server最近一次启动以来发送的数据包。

语法

```
SELECT @@pack_sent
```

@@packet_errors

全局变量 packet_errors 可保存 Microsoft SQL Server 最近一次启动数据包出现错误的数目。

语法

SELECT @@packet_errors

@@procid

全局变量@@procid 可保存当前正执行的存储过程的 ID。

语法

SELECT @@procid

@@remserver

全局变量@@remserver 位于注册记录中，它包含了远程服务器的名称。

语法

SELECT @@remserver

@@rowcount

全局变量@@rowcount可保存受Transact-SQL语句影响的行数，这些Transact-SQL语句是仅是为那些影响各行的语句执行的，例如：SELECT语句，INSERT语句，UPDATE语句和DELETE语句。

语法

SELECT @@rowcount

@@servername

全局变量@@servername可保存本地Microsoft SQL Server的名称。

语法

SELECT @@servername

@@servicename

全局变量@@servicename可保存Microsoft SQL Server的注册键MSSQLServer的名称。

语法

```
SELECT @@servicename
```

@@spid

全局变量@@spid可保存当前运行的程序的程序ID。

语法

```
SELECT @@spid
```

@@textsize

全局变量@@textsize可保存SET选项TEXTSIZE的当前值。TEXTSIZE选项

可决定 SELECT 语句能返回的文本或图像数据的最长长度。

语法

```
SELECT @@textsize
```

@@timeticks

全局变量@@timeticks 可保存计算机从属滴嗒的长度，1 毫秒表示 1 滴嗒。操作系统认为 1 滴嗒为 31.25 毫秒，或 1／32 秒。

语法

```
SELECT @@timeticks
```

@@total_errors

全局变量@@total_errors 可保存自 Microsoft SQL Server 最近一次启动以来发现的磁盘读／写错误总数。

语法

```
SELECT @@total_errors
```

@@total_read

全局变量@@total_read可保存自Microsoft SQL Server最近一次启动以来查看到的磁盘读取总数。

语法

```
SELECT @@total_read
```

@@total_write

全局变量@@total_write可保存自Microsoft SQL Server最近一次启动以来查看到的磁盘写入总数。

语法

```
SELECT @@total_write
```

@@trancount

全局变量@@trancount 可保存当前对话过程中为用户打开的活动事务的总数。

语法

```
SELECT @@trancount
```

@@version

全局变量@@version 可保存 Microsoft SQL Server 当前使用的处理器的版本日期、版本和类型。

语法

```
SELECT @@version
```

ABS

ABS 函数返回数值表达式的绝对值。

语法

ABS(numeric_expression)

变量 numeric expression 可以是整型数、短整型数、微短整型数、小数、浮点数、货币、数字、实型数或小货币型数。

ACOS

ACOS 函数是一个数学函数，它返回以弧度表示的角度，即 arc cosine。

语法

ACOS(float_expression)

变量 float_expression 是浮点数数据类型，表示角度的反余弦。

ALL

ALL 可返回 true(真)或 false(假)，当由子查询返回的内容与表达式想相匹配时，就返回真，当不是全部匹配或完全不匹配时，返回假。

语法

```
[NOT]  expression { = | <> | != | > | >= | !> | < | <= | !< }
ALL (subquery)
```

变量

NOT	NOT 为布尔表达式的否定指示符。
expression	expression 可以是为列名，常数，函数、变量、子查询程序、多个常数或多个函数
{= | <> | != | > | >= | !> | < | <= | !<}	这些符号代表运算符。
subquery	子程序中不能有 ORDER BY 语句，COMPUTE 子句和关键词 INTO。

ALTER DATABASE

ALTER DATABASE 可改变数据库大小和文件。

语法

```
ALTER DATABASE database
 { ADD FILE <filespec> [,...n] [TO FILEGROUP filegroup_name]
| ADD LOG FILE <filespec> [,...n]
| REMOVE FILE logical_file_name
| ADD FILEGROUP filegroup_name
| REMOVE FILEGROUP filegroup_name
| MODIFY FILE <filespec>
| MODIFY FILEGROUP filegroup_name filegroup_property
}
<filespec> ::=
(NAME = 'logical_file_name'
 [, FILENAME = 'os_file_name' ]
 [, SIZE =size]
 [, MAXSIZE = { max_size | UNLIMITED }]
 [, FILEGROWTH =growth_increment] )
```

变量

DATABASE	要修改的数据库。
ADD FILE	ADD FILE 添加文件或文件组。
NAME	Name 指文件的逻辑名称。
FILENAME	FILENAME 是操作系统上的文件名。
SIZE	SIZE 是文件开始的大小,MB(缺省值为最小 1MB)表示兆字节，KB 表示千字节。如果没有提供文件的大小，SIZE 变量对数据预置为 3MB，对记录预置为 1MB。
MAXSIZE	MAXSIZE 是文件可以扩展的最大容量，以 MB(缺省值)或 KB 表示。如果没有提供此变量，文件将扩展到磁盘满为止。
UNLIMITED	UNLIMITED 表示让文件扩展至磁盘满为止。
FILEGROWTH	FILEGROWTH 指文件一次增长的量(缺省值为 256KB，最小 64KB)，零(0)表示没有增长。
TO FILEGROUP	TO FILEGROUP 指将文件添加至文件组。
FOR RESTORE	这是一个向后兼容的变量(版本 6.5 使用 FOR LOAD)。由于要恢复数据库，所以无需初始化就可以建立文件。
ADD LOG FILE	ADD LOG FILE 可将记录文件添加到数据库中。
DROP LOG FILE	DROP LOG FILE 可从数据库中撤消记录文件。

CREATE FILEGROUP filegroup_name	CREATE FILEGROUP 可给要建立的文件组命名。要添加或撤消的文件组名称。
REMOVE FILEGROUP	REMOVE FILEGROUP 可指定要撤消的文件组。
MODIFY FILE	MODIFY FILE 可指示 SQL Server 修改文件的 FILENAME，SIZE 和 MAXSIZE。
MODIFY FILEGROUP filegroup_name filegroup_property	MODIFY FILEGROUP 可给出文件组的文件组属性。文件组属性有：READONLY=只读文件组，READWRITE=允许修改，DEFAULT=文件组是建立新表格和新索引的缺省文件组。

示例

下面是 ALTER DATABASE 语句的示例：

```
ALTER DATABASE mydatabase
ADD FILE (NAME = 'mddat2',
FILENAME = 'c:\mssql7\data\mddat2.ndf',
SIZE =10MB,
MAXSIZE=100MB,
FILEGROWTH=2MB)
```

本例中，文件添加到 mydatabase 中，容量为 10MB，最大容量为 100MB，文件以 2MB 的增量递增。

ALTER PROCEDURE

ALTER PROCEDURE 可改变已有的存储过程，但不影响许可与从属对象。如果要保存可选项，就必须用 ALTER PROCEDURE 使用 WITH ENCRYPTION 或 WITH RECOMPILE。

语法

```
ALTER PROC [EDURE] procedure_name [;number]
 [({@parameter_name | parameter } data_type [= default]
 [OUTPUT] )] [,...n]
WITH { RECOMPILE |ENCRYPTION | RECOMPILE ,ENCRYPTION}]
 [ FOR REPLICATION]
AS
 sql_statement [,...n]
```

变量

procedure_name	**procedure_name 是要修改的程序名称**
;number	**number 将程序存储集中在一起，以便能一次撤消它们。**
{ @parameter_name | parameter}	**存储过程的变量或参数。**
data_type	**data_type 是变量或参数的数据类型，不支持图**

	像数据类型。
Default	default是量或参数的缺省值。
OUTPUT	OUTPUT表示变量或参数用来返回一个值。
N	n表示可以列出更多的变量或参数，最多可达1,024个变量或参数。
{RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION}	RECOMPILE表示每次程序执行时都重新编译。ENCRYPTION表示将加密存储CREATE PROCEDURE语句的syscomments系统表格行。
FOR REPLICATION	FOR REPLICATION可筛选只通过复制执行的存储过程和用FOR REPLICATION子句建立的存储程序。它们不能在订阅服务器上执行，也不能和变量WITH RECOMPILE一起使用。
AS	存储过程的命令位于关键词AS的后面。
sql_statement	Transact-SQL语句建立存储过程。
n	n表示多个Transact-SQL语句组成存储过程，并可以列出来。

示例

下面是ALTER PROCEDURE语句的示例：

```
ALTER PROCEDURE MyProcedure
WITH ENCRYPTION
AS
```

```
SELECT col1,col2
FROM MyTable
WHERE col1=5
GO
```

本例中，加密了存储过程。

ALTER TABLE

ALTER TABLE 语句可通过撤消列、添加列、添加表格或列级约束、禁止或重新启动约束或触发器来改变指定表格的定义。

语法

```
ALTER TABLE MyTable
{
 [WITH CHECK |WITH NOCHECK]
{ [ALTER COLUMN column_name
{
 [ new_data_type [ (precision [, scale] )]
 [NULL |NOT NULL ] ]
| [ {ADD |DROP} ROWGUIDCOL ]
}
]
```

```
|ADD
{   [ <column_definition> ]
|column_name AS computed_column_expression
| [ <table_constraint> ]
}(,...n)
|DROP
{   [CONSTRAINT]   constraint
| COLUMN column
} [,...n]
 |   {CHECK |NOCHECK} CONSTRAINT
{ALL | constraint   [,...n] }
| {ENABLE | DISABLE} TRIGGER
{ALL | trigger [,...n] }
}
}
<column_definition> ::= {column_name data_type}
 [ NULL | NOT NULL ]
 [ IDENTITY   [(seed [,increment]   )
 [NOT FOR REPLICATION ]   ]
 [ ROWGUIDCOL ]
 [ <column_constraint>::=
 [ CONSTRAINT constraint_name]
```

```
{   { PRIMARY KEY | UNIQUE }
 [CLUSTERED |NONCLUSTERED]
 [WITH  [FILLFACTOR = fillfactor]
 [[,]  {SORTED_DATA
| SORTED_DATA_REORG} ]]
 [  ON filegroup ]
|  [FOREIGN KEY]
REFERENCES ref_table
 [  ( ref_column) ]
 [NOT FOR REPLICATION]
| DEFAULT constant_expression
|CHECK  [NOT FOR REPLICATION]
(logical_expression)
}
]    [...n]
<table_constraint> ::=  [CONSTRAINT constraint_name]
{  [  { PRIMARY KEY | UNIQUE}
 [  CLUSTERED | NONCLUSTERED ]
{ (column [,...n]  ) }
 [  WITH  [  FILLFACTOR = fillfactor]
 [[,]  { SORTED_DATA
| SORTED_DATA_REORG} ]
```

```
]
 [ON filegroup]
]
| FOREIGN KEY
 [(column [,...n] )]
REFERENCES ref_table  [(ref_column [,...n] )]
 [NOT FOR REPLICATION]
| DEFAULT constant_expression
 [FOR column]
| CHECK  [ NOT FOR REPLICATION]
(search_conditions)
}
```

变量

Table	**Table 是要修改的表格名称。**
WITH CHECK | WITH NOCHECK	**WITH CHECK | WITH NOCHECK 可确定是否检查了新约束的有效性(WITH CHECK 是新约束的缺省值)或是重新启动(WITH NOCHECK 是重新启动约束的缺省值)FOREIGN KEY 或者 CHECK 约束。**
ALTER COLUMN	**ALTER COLUMN 是指修改的列。**
column_name	**column_name 是要修改、增加，或撤消的列**

	名。该变量不能是文本、图像、ntext、时间戳、表格的 ROWGUIDCOL;已计算的列、已复制的列、已索引的列;列名不能用于 CHECK、FOREIGN KEY,UNIQUE 或 PRIMARY KEY 约束中,(可以改变约束中可变长度列的长度);列名不能与缺省值一起使用。
new_data_type	New Data Type 指修改列中的新数据类型。
precision	precision 是指数据类型的精度。
scale	scale 是数据类型的尺度。
[{ADD | DROP} ROWGUIDCOL]	这些变量说明列是从添加到表格中或是从表格中删去。并说明列是具有 ROWGUIDCOL 属性(全局唯一标识符列可以分配给唯一标识符列的全局唯一标识符列)
ADD	ADD 表示添加列的定义、已计算的列定义或表格约束。
AS computed_column_expression	计算列是虚拟列,是新的可选项,现在可以存储和检索计算列,但是不能 INSERT 或 UPDATE 它。不能将计算列用作关键列,也不能用于 PRIMARY KEY,UNIQUE,FOREIGN KEY 或 CHECK 约束。
DROP [CONSTRAINT] constraint | COLUMN column	DROP CONSTRAINT 或 COLUMN 表示将约束或列从表格中删除。

N	N 表示前面的项可重复
CHECK | NOCHECK	CHECK 选项启用约束，而 NOCHECK 选项禁用约束。禁用约束表示进行 INSERT 或 UPDATE 时不能使用约束条件，这个变量只适用于 FOREIGN KEY 和 CHECK 约束。
ALL	ALL 是用 CHECK 或 NOCHECK 选项启用或禁用所有的约束。
{ENABLE | DISABLE} TRIGGER	ENABLE TRIGGER 或 DISABLE TRIGGER 表示触发器仍作为数据库中的一个对象，但是如果禁用它，INSERT、UPDATE 和 DELETE 语句将不能启动触发器。
ALL	ALL 是用 ENABLE TRIGGER 或 DISABLE TRIGGER 选项启用或禁用所有触发器。
trigger	trigger 是要启用或禁用的触发器名。
data_type	Data_type 是新列的 SQL Server 数据类型。如果用唯一标识符数据类型，且表格中已有几行，就可以用 DEFAULT 约束和 NEWID() 函数为已存在的行在新列中生成唯一的标识符值。
NULL | NOT NULL	NULL 或者 NOT NULL 确定列能否存储 Null 值。如果表格中有行，且使用 NOT NULL，则使用 DEFAULT 约束 WITH VALUES，已有行

	的新列将自动包含缺省值。
IDENTITY	IDENTITY 是一个属性，它可用来将递增的整数放在新列中。如果表格中已有行，用 IDENTITY 属性添加一列，为已有行添加的新列将自动为已有的行在每个新列中包含递增的整数。在将来添加新行时，也能执行上述操作。
seed	seed 指用 IDENTITY 属性添加一列时第一行的整数值，它也表明新 IDENTITY 列中的值从哪儿开始。如果不用此关键词，SQL Server 就把值 1 放到该列中。
Increment	如果用 IDENTITY 属性添加一列，当添加新行时，Increment 就是新 IDENTITY 列中要增加的增量。如果不用 Increment，SQL Server 将对最后添加的行中的值加 1。
NOT FOR REPLICATION	如果使用复制，并用 IDENTITY 属性添加一列，可用该变量通知 SQL Server，当 SQL Server 系统复制注册(例如 sqlrepl)将数据插入到表格中时，不要施加 IDENTITY 条件。
CONSTRAINT	CONSTRAINT 选项是指如何建立 PRIMARY KEY、UNIQUE、FOREIGN KEY、CHECK 或 DEFAULT

	约束。CONSTRAINT 用来保持数据的完整性，添加 CONSTRAINT 可以建立索引。
constraint_name	constraint_name 是新建立 CONSTRAINT 的名称。如果没给约束命名，那么 SQL Server 将为其命名。
PRIMARY KEY	PRIMARY KEY 是一个约束(一个表格中只有一个)，它在一列或几列中建立唯一的索引，并将一列或几列指定为表格中主键。
UNIQUE	列值可以是表中唯一的值。但不必指定为主键。UNIQUE 约束可建立列中唯一的索引。
CLUSTERED | NONCLUSTE-RED	如果要建立 PRIMARY KEY(缺省值为 CLUSTERED)或 UNIQUE(缺省值为 NON-CLUSTERED)约束，CLUSTERED 或 NONCLUSTERED 可确定添加何种索引。如果已有成簇索引，就不能使用 CLUSTERED，除非删除了现有的 CLUSTERED 索引，因为一个表格中只能有一个索引。如果已有一个成簇索引，再添加 PRIMARY KEY 约束，该变量的缺省值就是 NONCLUSTERED。
WITH FILLFACTOR =fillfactor	WITH FILLFACTOR 说明每一索引页中有多少数据。可选值是 1 到 100，缺省值为 0，以百分数表示。填充因子越低，页中索引项

	可用的空间就越多，这样，页中就为插入新行留出了更多的空间，也就不需 SQL Server 为新行分配较多的空间。
SORTED_DATA | SORTED_DATA_REORG	SORTED DATA 通知 SQL Server 数据已分类，当添加成簇索引时，SQL Server 就不必对数据分类。SQL Server 将进行检查，并通过检查索引值来确保数据按分类排列。如果检查失败，将返回错误，ALTER table 也失败，另一方面，SORTED_DATA_REORG 通知 SQL Server 数据已经分类，只需在磁盘上重新组织这些数据。当使用 FILLFACTOR OPTION 时使用 SORTED_DATA_REORG 可改变数据在页中的存储方式。用 DBCC SHOWCONTIG 可确认表格是否已经分段，是否需要重新组织。
ON filegroup	ON filegroup 可决定为约束建立的索引将驻留在磁盘的哪一部分。如果不用 ON filegroup，索引将与表格放在同一文件组中。当添加 PRIMARY KEY CLUSTERED 索引或 UNIQUE CLUSTERED 索引时，使用 ON filegroup 会出现下面的情况:由于成簇索引的数据位于成簇索引的底部，整个表格

	将向指定的文件组移动。
FOREIGN KEY... REFERENCES	FOREIGN KEY...REFERENCES 可通过确保列中的每一项位于引用表格的列中，来建立保持引用完整性的约束。
ref_table	ref_table 是 FOREGIGN KEY 约束所引用的表格名称。
ref_column	ref_column 是 FOREIGN KEY 约束所引用的列。
NOT FOR REPLICATION	如果正使用复制，并用 IDENTITY 属性添加一列，用 NOT FOR REPLICATION 通知 SQL Server，当 SQL Server 系统复制注册(例如 sqlrepl)将数据插入表格时不施加 IDENTITY 条件。
DEFAULT	DEFAULT 是一种约束，如果没有指定 INSERT 列中的值，它决定插入列中的值。由于一列中只能有一个值，当列中带有时间戳数据类型、IDENTITY 属性或另一个 DEFAULT 约束时，不能使用 DEFAULT 约束。如果需添加另一个 DEFAULT 约束，必须撤消旧的缺省值。DEFAULT 可以将数值添加至已有行的新列中，这一方面它的功能很强。
VALUES	VALUES 可和 DEFAULT 一起使用，是已有行

	的新值，它还可用于以 ALTER TABLE 添加的新列。
constant_expression	constant_expression 可和 DEFAULT 一起使用，是列的缺省值，它可以是常数、NULL 或系统函数。
CHECK	CHECK 是一种约束，它限制可以插入到列中的值。
logical_expression	logical_expression 可以引用同一行中其它的列，其返回值为 true 或 false。
column [,...n]	column 表示约束的列目录。
FOR column	FOR column 是和 DEFAULT 约束一起使用的列。

示例

下面是个 ALTER TABLE 语句的示例：

```
ALTER TABLE MyTable ADD col1 int null
CONSTRAINT myconstraint UNIQUE
GO
```

本例中，单独的约束和列添加到表格中。

ALTER TRIGGER

Transact-SQL 现在能修改触发器对象，不必为修改触发器对象而从数据库中和 syscomments 系统表中删除它。修改触发器的功能很重要，因为触发器携带了其它参数和许可，一旦撤消对象，参数与许可就将被删除。只修改对象而不是删除它，将保留与对象有关的参数和许可，如果使用 CREETE TRIGGER 语句的 WITH APPEND 选项，那么每个 Update，Insert Delete 可以有多个触发器，这意味着每个 Update、Insert、Delete 可以启动多个触发器。这种 WITH APPEND 功能包含在由 sp_dbcmptlevel 设置的 7.0 级别的兼容中。WITH APPEND 只需要 65 或 60 级别的兼容，因为当兼容级别达到 70 时，WITH APPEND 就是缺省操作。

触发器受其可使用的 Transact-SQL 语句的限制，触发器中不能使用下列 Transact-SQL 语句：

- ALTER DATABASE
- ALTER TABLE
- ALTER TRIGGER
- CREATE DATABASE
- CREATE DEFAULT
- CREATE PROCEDURE
- CREATE RULE
- CREATE TABLE
- LOAD LOG
- RESTORE LOG
- REVOKE
- SELECT INTO(因为该语句创建了表格)
- TRUNCATE TABLE
- ALTER PROCEDURE
- ALTER VIEW

- CREATE TRIGGER
- DENY
- DISK INIT
- DROP DATABASE
- DROP DEFAULT
- DROP PROCEDURE
- DROP RULE
- DROP TRIGGER
- DROP VIEW
- LOAD DATABASE
- CREATE INDEX
- CREATE TABLE
- CREATE VIEW
- DISK RESIZE
- DROP INDEX
- DROP TABLE
- GRANT
- RESTORE DATABASE
- RECONFIGURE
- UPDATE STATISTICS

语法

```
ALTER TRIGGER trigger_name
ON table
 [WITH ENCRYPTION]
{FOR { [,] [INSERT] [,] [UPDATE] [,] [DELETE] }
 [NOT FOR REPLICATION]
AS
sql_statement [...n]
} | {FOR { [,] [INSERT] [,] [UPDATE] }
 [NOT FOR REPLICATION]
AS
```

```
IF UPDATE (column)
 [ {AND | OR} UPDATE (column) [,...n] ]
sql_statement [...n] }
```

变量

TriggerName	**要修改的触发器名。**
TableName	**触发器执行时所在的表格。**
WITH ENCRYPTION	**加密 syscomments 系统表中的代码。**
{ [,] [INSERT] [,] [UPDATE] [,] [DELETE] } | { [,] [INSERT] [,] [UPDATE] }	**指明启动触发器的语句的关键词。**
NOT FOR REPLICATION	**如果与复制有关的 sqlrepl 注册 ID 修改了表格，就不启动触发器。**
AS	**表示下一个语句是 Transact-SQL 语句。**
Transact-SQL statement(s)	**是触发器的 Transact-SQL 语句。**
IF UPDATE (ColumnName)	**根据是否对指定列进行 INSERT 或 UPDATE，来提供 IF 逻辑。**
ColumnName	**检查 INSERT 操作或 UPDATE 操作的列名。**

示例

下面是 ALTER TRIGGER 语句的示例：

```
ALTER TRIGGER mytrigger
ON tablename
FOR INSERT
AS RAISERROR ( ' mytrigger error ' ,1, 2)
```

本例中，修改了触发器 mytrigger。

ALTER VIEW

ALTER VIEW 能修改视图对象，而不必为修改视图对象而将该对象从数据库中或 syscomments 系统表中删除它。修改视图的功能很重要，因为视图携带了其他参数和许可，删除对象时，这些参数和许可也被删除，只修改对象而不是删除它，将保留与对象有关的参数和许可。SELECT 语句中不要引用临时表，也不能包括 ORDER BY 子句、COMPUTE 子句、COMPUTE BY 子句或关键词 INTO。

语法

```
ALTER VIEW view_name
column  [,...n] )]
 [WITH ENCRYPTION]
```

```
AS
select_statement  [WITH CHECK OPTION]
```

变量

view_name	**要修改的视图名**
Column [,…n]	**Column 是列名的名称，用逗号分隔开，是视图的一部分。**
WITH ENCRYPTION	**加密 syscomments 系统表中的对象。**
AS	**表示下一个语句是 Transact-SQL 语句。**
select_statement [WITH CHECK OPTION]	**SELECT 语句是视图的基础，WITH CHECK OPTION 表示 SQL Server 将进行检查，以确保视图的任意数据修改语句都符合视图中 select_statement 包含的边界设置。**

示例

下面是使用 ALTER VIEW 的语句的示例：

```
CREATE VIEW MyView
AS
```

```
SELECT column1,column2
FROM   MyTable
```

本例中，修改了视图，而没有删除并再创建它。

AND

AND 语句可串接两个条件。当两个条件都为真时，它也为真。SQL Server 中 AND 运算符先于其他逻辑运算符进行，如果使用括号，将影响执行逻辑运算符的顺序。

语法

```
[NOT] <predicate>AND  [NOT]  <predicate> [,…n]
```

变量

NOT	NOT 是布尔表达式的否定指示符
Prepredicate [,…]	Predicate 也称为表达式，用逗号分隔开，它返回 TRUE 或 FALSE。

示例

下面是 AND 语句的示例：

```
SELECT *
FROM MyTable
WHERE column1 = column2
AND column3 = column4
```

本例中，会选择出表格中的 Column 1 等于 Column 2,且 Column 3 等于 Column 4 的行。

ANY

使用关键字 ANY，将返回 TRUE 和 FALSE，即使与子查询程序一起使用也是如此(不返回子查询程序的行，只返回 TRUE 或 FALSE)。

当子查询中检索出的数据与表达式中的比较相符时返回 TRUE。

当对子查询程序中所有的行进行比较出现错误时，返回 FALSE，子查询程序没有找到行时也返回 FAUISE。

语法

```
[NOT] expression { = | <> | != | > | >= | !> | > | <= | !< }
{SOME|ANY}(subquery)
```

变量

NOT	NOT 是布尔表达式的否定指示符。

Expression {=|< >|!=|>|>=|!> |<|<=|!<} subquery	表达式可以是列名、常数、函数、变量、子查询程序、几个常数或几个函数。 这些符号表示算术运算符。 子查询程序中不允许出现 ORDER BY 子句、COMPUTE 子句、关键词 INTO。

示例

下面是关键字 ANY 的示例：

```
IF @myvar>= ANY (SELECT au_lname FROM authors)
PRINT ' OK, this value is not less than all the names in the table '
ELSE
PRINT ' Sorry, you cannot use a value less than all the names in the table '
```

本例中，返回 TRUE。

APP_NAME

如果程序设置此函数，执行的对话将返回最大长度的字符型或可变字符型程序名。用此函数可以确定哪个客户应用程序在运行 Microsoft SQL Server。

语法

APP_NAME()

示例
下面是 APP_NAME()函数的示例：

```
SELECT APP_NAME()
```

本例中，将选择出返回“MS SQL Query Analyzer”的 APP_NAME 的内容，因为是在 Query Analyzer 中执行程序而且 Query Analyzer 设置了此函数。

ASCII

ASCII 函数返回的整数表示字符表达式最左边的字符的 ASCII 代码值。
语法

```
ASCII (character_expression)
```

字母变量 character_expression 可以是常量、变量或列。
示例
下面是 ASCII 函数的示例：

```
SELECT ASCII("AB")
```

本例中，返回的“65”是字符 A 的 ASCII 码值，因为不管 A 字符后面放置什么字符，都返回字符表达式中最左边的字符的 ASCII 码值。

ASIN

数学函数 ASIN 返回以弧度表示的角度。该函数通常是指反正弦函数。如果看到错误信息“域有差错”,Microsoft SQL Server 是指角度值不在 ASIN 函数的有效范围内。

语法

```
ASIN(float_expression)
```

变量 float_expression 是浮点型数据。

示例

下面是 ASIN 函数的示例：

```
SELECT ASIN (.345986)
```

本例中，正弦值为 0.345986 的浮点表达式，其返回值 0.35328948592348308 是以弧度表示的角度。

ATAN

数学函数 AIAN 返回以弧度表示的角度，此角度的正切是作为变量传递给该函数的浮点表达式，该函数也称为反正切函数。

语法

```
ATAN (float_expression)
```

变量 float expression 是浮点型数据。

示例

下面是 ATAN 函数的示例：

```
SELECT ATAN(0.345986)
```

本例中，正切值为 0.345986 的浮点表达式，其返回值 0.3330944069420642 1是以弧度表示的角度。

ATN2

数学函数 ATN2 返回角度的弧度值，此角度的正切位于两个作为变量传递给函数的浮点表达式之间，该函数也称为反正切函数。

语法

ATN2(float_expression,float_expression)

变量 float expression 是浮点型数据，且必须给出两个浮点表达式。

示例

下面是 ATN2 函数的示例：

SELECT ATN2 (.34，.35)

本例中，返回值 0.77090642433194834 是以弧度表示的角度，该角度的正切位于两个浮点表达式 0.34 和 0.35 之间。

AVG

AVG 是个集合函数，它计算出数值列的平均值，但忽略给定表达式中的空值。如果使用了 DISTINCT，则对重复的值只取一次进行平均。

语法

AVG([ALL | DISTINCT] expression)

变量

ALL	集合函数适用于所有数值，ALL是缺省值。
DISTINCT	对重复的值只取一次进行平均。
expression	表达式可以是常数、列名、函数、子查询程序、算术运算符、按位运算符和字符串运算符。

示例

下面是AVG集合函数的示例：

```
SELECT AVG (age)
FROM table1
```

本例中，对表 table1 中的“age”列进行平均并返回。如果任意行有 NULL 值，则此行被忽略。

BACKUP

BACKUP语句可备份整个数据库，事务处理记录、文件或文件组。

语法

备份整个数据库的语法：

```
BACKUP DATABASE {database_name | @database_name_var}
TO <backup_device>  [,...n]
 [WITH
```

[BLOCKSIZE = {blocksize | @blocksize_variable}]
[[,] DESCRIPTION = { text | @text_variable}]
[[,] DIFFERENTIAL]
[[,] EXPIREDATE ={ date | @date_var}
| RETAINDAYS = {days|@days_var}]
[[,] FORMAT | NOFORMAT]
[[,] {INIT |NOINIT}]
[[,] MEDIADESCRIPTION ={ text | @text_variable}]
[[,] MEDIANAME ={ media_name | @media_name_variable}]
[[,] [NAME ={ backup_set_name | @backup_set_name_var}]
[[,] {NOSKIP |SKIP}]
[[,] {NOUNLOAD |UNLOAD}]
[[,] [RESTART]
[[,] STATS [= precentage]]
]

语法

备份属于某数据库的文件或文件组的语法：

BACKUP DATABASE {database_name | @database_name_var}
<file_or_filegroup> [,...n]
TO <backup_device> [,...n]
[WITH
[BLOCKSIZE = {blocksize | @blocksize_variable}]

[[,] DESCRIPTION = { text | @text_variable}]
[[,] EXPIREDATE ={ date | @date_var}
| RETAINDAYS = {days|@days_var}]
[[,] FORMAT | NOFORMAT]
[[,] {INIT |NOINIT}]
[[,] MEDIADESCRIPTION ={ text | @text_variable}]
[[,] MEDIANAME ={ media_name | @media_name_variable}]
[[,] NAME ={ backup_set_name | @backup_set_name_var}]
[[,] {NOSKIP |SKIP}]
[[,] {NOUNLOAD |UNLOAD}]
[[,] [RESTART]
[[,] STATS [= precentage]]
]

语法

备份事务处理记录的语法：

```
BACKUP LOG {database_name | @database_name_var}
{
 [WITH
{ NO_LOG | TRUNCATE_ONLY }]
}
|
{
```

```
TO <backup_device>  [,...n]
 [WITH
 [BLOCKSIZE = {blocksize | @blocksize_variable}]
 [[,]  DESCRIPTION = { text | @text_variable}]
 [[,]  EXPIREDATE ={ date | @date_var}
| RETAINDAYS = {days|@days_var}]
 [[,]  FORMAT | NOFORMAT]
 [[,]  {INIT |NOINIT}]
 [[,]  MEDIADESCRIPTION ={ text | @text_variable}]
 [[,]  MEDIANAME ={ media_name | @media_name_variable}]
 [[,]   [NAME ={ backup_set_name | @backup_set_name_var}]
 [[,]  NO_TRUNCATE]
 [[,]  {NOSKIP |SKIP}]
 [[,]  {NOUNLOAD |UNLOAD}]
 [[,]   [RESTART]
 [[,]  STATS  [= precentage]]
]
}
<backup_device>::=
{
{backup_device_name | @ backup_device_name_var}
|
```

```
{DISK | TYPE | PIPE }=
{ ' temp_backup_device '  |@ temp_backup_device_var}
}
<file_or_filegroup>::=
{
FILE ={ logical_file_name | @ logical_file_name_var}
|
FILEGROUP = { logical_filegroup_name | @ logical_filegroup_name_var}
}
```

变量

DATABASE	**如果使用该变量，就命令 Microsoft SQL Server 对数据库进行完整的备份。也可以和文件或文件组列表一起使用该变量，这将限定将数据库备份到指定的文件或文件组中。现在，Microsoft SQL Server 7 在恢复备份文件时，只备份足以建立数据库的记录，并带有完整的事务处理。**
databasename	**此变量是包含在备份程序中的数据库名，可以用变量或常数串的形式传递数据库名称。databasename = @dbvariable 或者 databasename = plaintextname。**

backupfilename	此变量是由 sp_addumpdevice 建立的备份设备的逻辑名称。该名称必须遵从标识符 1～128 字符的命名规则。可以用变量或常数串的形式传递备份文件名： backupfilename=@BFNvariable 或 backupfilename=plaintextname
{DISK | TAPE | PIPE}=PATHFILENAME	此变量允许自由直接备份至文件，而不必使用逻辑备份设备，第一次进行备份时建立新文件，以后再进行备份时可以使用该文件。关于临时备份文件:DISKDISK 备份文件每次都重新建立，要使用完整的路径和文件名。TAPE 对于有效的 Microsoft Tape 格式数据集，使用 WITH FORMAT。要使用完整的路径和文件名。例如：＼＼.＼tape0。PIPE 由客户应用程序使用的指定管道的客户机名称。可以用变量或常量字符串的形式传递临时备份文件名： tempbackupfilename = @TBFvariable 或 tempbackupfilename= plaintextname
NOUNLOAD	此变量可确定备份后磁带不会从磁带驱动器上自动卸载。用这种方式设定此设置，直到备份命令中给出改变此设置的 UNLOAD 指令为止。
UNLOAD	此变量确定备份完成以后，磁带将自动倒回并卸载，这是磁带驱动器的缺省设置。用这种方式设定此设置，直到备份命令中给出改变此设置的

	NOUNLOAD 指令为止。
DESCRIPTION	这是用户定义的文本，用来提供备份信息。可以键入多达 255 个字符说明其内容，也可以是日期或与以后备份的标识符有关的任何其他信息。
BLOCKSIZE	指定块的实际大小，以字节表示。DISK 不用 TAPE 只有在用 FORMAT 覆盖磁带时才使用，否则备份程序可以确定最合适的容量，除非忽略此变量，自己给出磁带的块容量。PIPE 如果没有给出块的大小，Microsoft SQL Server 将使用 65，536。
FORMAT	此变量改写备份文件和介质标题。可以利用格式重新初始化带状的备份集。如果使用了该变量，就不必使用 SKIP 和 INIT 变量，因为 FORMAT 可自动调用它们。对于所有的 FORMAT 命令，该变量还删除数据和标题信息，使用时须注意。FORMAT 也会删除口令信息。
NOFORMAT	NOFORMAT 和 FORMAT 正相反。它不改写所有卷上的介质标题，也不改写备份文件。
MEDIANAME	此变量可以是多达 128 个字符的用户定义文本，此文本是备份介质集的介质名称。如果给出介质名称，媒体名称将试图与备份卷的介质名称相匹配。如果没有给出介质名称，或使用了 SKIP 变量，就不会检查介质名称的匹配情况。如果该变量与

	FORMAT 变量一起使用，MEDIANAME 将作为介质名称，和 NT 备份共用的磁带必须有介质名称。
MEDIADESCRIPTION	此变量可有 255 个字符，是描述介质集的用户定义文本。
INIT	此变量指要覆盖设备上的所有数据，只有 INIT 备份文件在 DISK 或 TAPE 介质上。因为它不覆盖介质标题，所以 INIT 与 FORMAT 不同。它与 FORMAT 的不同之处还在于口令信息防止 SQL Server 写到设备上。用 FORMAT 和口令可覆盖介质。要覆盖用口令保护或加密的介质，可指定 WITH FORMAT 选项。如果介质上的所有备份集还没有到期(详见 EXPIREDATE 和 RETAINDAYS 选项)，或者 BACKUP 语句中给出的备份集名称（如果提供该名称）与备份介质(详见 NAME 选项)上的名称不相符，就不会覆盖备份介质。注意，SQL Server 备份可以和 Windows NT 备份共存于磁带介质上，因为 Microsoft SQL Server 7.0 备份格式符合 Microsoft Tape Format(MFT)，Windows NT 磁带备份也使用这个格式。使用 SKIP 选项可忽略这些检查。
NOINIT	此变量决定备份集应添加到设备上，而不是覆盖最后一次备份。如果不指明 INIT 或者 NOINIT，缺

	省值就是 NOINIT。
EXPIREDATE	使用此变量表示到终止日期备份集才会覆盖。必须对介质上所有的备份集定义这此变量，才能使它有效。用 SKIP 变量可忽略此变量。可以用变量或常数串的形式传递终止日期： expiredate = @Evariable 或 expiredate = "constantstring"
RETAINDAYS	使用此变量表示超过保留天数就将覆盖备份集，必须对介质上所有的备份集定义这此变量，才能使它有效。用 SKIP 变量可忽略此变量。可以用变量或整数的形式传递保留天数，retaindays = @Rvariable 或 retaindays = integer 和 INIT 一起使用此选项以使它有效。EXPIREDATE 或 RETAINDAYS 的缺省值为 sp_configure 配置设置 Media Retention。防止覆盖的设置只由 Microsoft SQL Server 用来防止覆盖其备份。
DIFFERENTIAL	这是一个新的备份性能。此变量指只备份自最近一次全集备份以来修改的数据库部分，此选项可节省备份的时间与空间。现在，当恢复备份文件时，Microsoft SQL Server 只备份足以建立数据库的记录，并带有完整的事务处理。
NAME	此变量遵从 1～128 个字符标识符的规则，可以留有空格，它是备份集的名称。

RESTART	这是 Microsoft SQL Server 7.0 又一个新的可选功能。它允许 Microsoft SQL Server 重新启动中断的备份操作。要执行此重新启动功能,用 RESTART 变量重复 BACKUP 语句，它只适用于 TAPE。
NOSKIP	此变量强制检查介质名称、终止日期和保留天数。如果给出了 NOSKIP 或 SKIP 变量，当创建了备份设备时，它们会忽略任何已定义的设置。
SKIP	如果使用此变量，终止日期和介质名称的检查不用来防止覆盖备份集和备份集中不匹配的介质名称。如果给出了 NOSKIP 或 SKIP 变量，当创建了备份设备时，它们会忽略任何已定义的设置。
STATS	此变量可查看备份程序的进展情况，并将进度以完成的百分比显示到屏幕上或输出文件上。
FILE FILEGROUP	此变量用于下列情况:数据库太大，从性能上讲不推荐进行全集备份整个数据库，它可能是一个文件或多个文件，也可能是一个文件组或多个文件组，备份文件和／或文件组与 checkpoint 选项中 trunc 记录不兼容。如果表格及其索引驻留在不同的文件或文件组上，所有这些文件必须一起备份。
LOG	此变量表示备份的是事务处理记录，而不是数据库。用户将获得自最后一次成功的 LOG 备份以来对数据库的所有变动。

TRUNCATE_ONLYNO_LOG	如果使用这些变量，且正删除事务处理记录，就应备份整个数据库，此变量会删去记录中不活动的部分(不做记录的备份拷贝)，并截取记录。在用 NO_LOG 备份记录后，记录中对数据库的变动不能应用于恢复数据库，这就是为何有必要备份数据库的原因。
NO_TRUNCATE	备份记录，但不截取。

示例

下面是备份整个数据库的示例：

```
BACKUP DATABASE Customer TO customer_db1
```

本例中，将客户数据库备份到转储设备 customer_db1 中。

示例

下面的例子是备份整个数据库和事务处理记录：

```
BACKUP DATABASE Customer TO customer_db1
GO
BACKUP LOG Customer TO customer_log1
GO
```

BEGIN...END

BEGIN...END 语句将 Transact-SQL 语句组合在一起，这样，当使用流程控制语句（如 IF...ELSE）时，整个语句组将按流程控制逻辑运行，而不会在执行了 IF 语句后单个执行一个语句。

语法

```
BEGIN
{sql_statement | statement_block}
END
```

变量{sql_statement | statement_block}是任意系列的 Transact-SQL 语句。

示例

下面是 BEGIN...END 的示例：

```
IF @var1=5
BEGIN
SELECT @var2=6
PRINT 'I am also doing this print if var1=5'
END
```

本例中，如果@varl=5，就执行 SELECT 与 PRINT 语句。

BEGIN DISTRIBUTED TRANSACTION

BEGIN DISTRIBUTED TRANSACTION 语句表明，Microsoft Distributed Transaction Coordinator (MS DTC)开始控制 Transact-SQL 分布式事务处理。

语法

```
BEGIN DISTRIBUTED TRAN [SACTION]    [transaction_name]
```

变量 transaction_name 由用户命名，MS DTC 用它来引用事务处理。

示例

下面是 BEGIN DISTRIBUTED TRANSACTION 语句的示例：

```
BEGIN DISTRIBUTED TRANSACTION
UPDATE mylocaltable
SET col1=' N／A '
EXECUTE remote.db1.dbo.reset_col1  ' N／A '
COMMIT TRAN
```

本例中，利用双向委托，UPDATE 语句更新本地数据库，远程存储过程更新

远程数据库。

BEGIN TRANSACTION

BEGIN TRANSACTION 语句表明开始进行 Transact-SQL 本地事务处理。

语法

```
BEGIN TRAN [SACTION] [transaction_name]
```

变量 transaction_name 是由用户命名的事务处理。

示例

下面是 BEGIN TRANSACTION 语句的示例：

```
BEGIN TRANSACTION
UPDATE mylocaltable
SET col1=' N／A '
EXECUTE reset_col1  ' N／A '
COMMIT TRAN
```

本例中，利用双向委托，由 UPDATE 语句更新本地数据库，远程存储过程更新远程数据库。

BETWEEN

BETWEEN 包括某一范围的两个数值。

语法

[NOT] expression [NOT] BETWEEN expression AND expression

变量

NOT	**NOT 是布尔表达式的否定指示符。**
expression	**表达式可以是常数、列名、函数、子查询程序、算术运算符、按位运算符和串运算符。**
AND	**AND 语句串接两个条件。**

示例

下面是 BETWEEN 语句的示例：

```
SELECT *
FROM tableX
WHERE col1 BETWEEN 1 AND 5
```

本例中，选择出的行，其列值在 1～5 之间，查询中将选择有 1 和 5 的行。

binary

当列中的值有相同的长度时，就可以用固定长度的二进制数据类型存储二进制数据，如果长度不一致，就采用可变长度的数据类型。

语法

binary [(n)]

变量(n)是字符数，它最多可存储 8000 字节。即使列比(n)短，(n)仍是存储所占用的空间总量，如果列不超过 15，将 15 用作(n)，可以避免列比要存储的数据大得太多。如果临时表中的行超过 8000，查询处理器就不可能处理这种情形。

在定义数据时，长度的缺省值(如果没有提供)为 1，如果使用 CONVERT，长度为 30，如果数据太长，在一列中放不下，就截去数据的尾部。

bit

位数据类型存储 1 或 0，通常用来表示 1=是，0=非。如果想在位数据类型中插入非 0 或 1 的内容，就插入 1。

语法

Bit (n)

变量(n)为一整数，表示位列的长度。存储长度为 1 个字节，一个字节包含 8 位。有位数类型的列不能加索引。

BREAK

BREAK 语句用在 Transact-SQL 流程控制语言中以退出 WHILE 循环。

语法

BREAK

示例

下面是 WHILE 循环中用 BREAK 的示例：

```
WHILE ... --某些内容为真
BEGIN
IF...--其他内容为真
BREAK
ELSE
CONTINUE
```

```
END
```

本例中，如果WHILE循环中其它内容为真，就退出循环。

BULK INSERT

BULK INSERT是将数据文件以指定格式存放到数据库表中。
语法

```
BULK INSERT  [[ ' database_name ' .] [ ' owner ' ] .] { ' table_name '
FROM data_file}
 [WITH
(
 [ BATCHSIZE  [=batch_size]]
 [[,]  CHECK_CONSTRAINTS]
 [[,]  CODEPAGE  [ = ACP|OEM |RAW |code_page]]
 [[,]  DATAFILETYPE  [ =
{ ' char '  |' native ' |' widechar ' |' widenative ' }]]
 [[,]  FIELDTERMINATOR  [=' field_terminator ' ]]
 [[,]  FIRSTROW  [=first_row]]
 [[,]  FORMATFILE  [=' format_file_path ' ]]
 [[,]  KEEPIDENTITY]
```

```
[[,] KEEPNULLS]
[[,] LASTROW [ =last_row]]
[[,] maxerrors [ =max_errors]]
[[,] ORDER ({column [ASC | DESC ] } [,...n] )]
[[,] ROWTERMINATOR [ = ' row_terminator ' ]]
[[,] TABLOCK] )]
```

变量

'database_name'	database_name 是放置表格的数据库名称。
'owner'	owner 是表格拥有者的名称。
'table_name'	table_name 是要到成批插入数据的表格。
data_file	data_file 是指数据文件的路径。
BATCHSIZE [=batch_size]	BATCHSIZE 是成批处理的行数。
CHECK_CONSTRAINTS	CHECK_CONSTRAINTS 指成批插入时对表格上的约束进行检查。缺省设置是不检查约束。
CODEPAGE [=ACP | OEM | RAW | code_page]	CODEPAGE 是数据文件中数据的代码页，这些数据是字符、可变字符或文本列，它们的字符值大于 127 或小于 32。RAW 由于不用转换，速度最快，

	其它码页则转换 SQL 成 Server 代码页。
<value>	<value>是代码页数。
DATAFILETYPE { [='char' | 'native' | 'widechar' | 'widenative'}]	DATAFILETYPE 指令 BULK INSERT 用给定类型的缺省值进行成批插入。char 用＼t(制表符)作为缺省的字段分隔符，用＼n(新行字符)作为缺省的行终止符。Native 用数据的本机数据类型作为缺省值进行成批复制。每个字段都没有提示。Widechar 用 Unicode 字符作为缺省值，也没有提示，用 nchar 的缺省值作为缺省存储类型，用＼t(制表符)作为缺省的字段分隔符。用＼n(新行字符)作为缺省的行终止符。Widenative 除字符、可变字符和文本列保存为 Unicode 字符外，其他情况与 native 相同。Widenative 比 Widechar 快，并其用途是用数据文件将数据从一个 SQL Server 传输到另一个 SQL Server。
FIELDTERMINATOR [= 'field_terminator']	field_terminator 是用于字符和宽字符的字符终止符，缺省值为＼t。

FIRSTROW [=first_row]	first_row 是要复制的第一行的数目，缺省值为 1。
FORMATFILE [= ' format_file_path ']	format_file_path 格是响应前一次使用的格式文件的完整路径。
KEEPIDENTITY	KEEPIDENTITY 表示标识列在文件中，它从数据文件中提取标识列的值。
KEEPNULLS	KEEPNULLS 表示是指保持空值，而不使用缺省值。
LASTROW [=last_row]	last_row 是要复制的最后一行的数目，缺省值为 0。它表示最后一行实际上在文件中。
MAXERRORS [=max_errors]	max_errors 是在停止成批复制操作前出现最多的错误数目，缺省值为 10。
ORDER({column [ASC | DESC] } [,...n]	ORDER 说明数据文件中的数据如何分类。缺省状态下，成批复制操作假定数据文件没有排序。如果其顺序与表的成簇索引相同，则运行较快。
n	n 表示指前面的项可重复。
ROWTERMINATOR [= ' row_terminator ']	row_terminator 是用于字符和宽字符的行结束符，缺省值为＼n。
TABLOCK	TABLOCK 指进行成批复制的过程中需

要表格级别的锁定。如果表中没有索引，且使用了 TABLOCK，则多个客户可以插入表格中。缺省值是表中需要行级锁定。

示例

下面是 BULK INSERT Transact-SQL 语句是示例：

```
BULK INSERT Mydb..MyTable
FORM ' a:\myfile.txt'
WITH (FIELDTERMINATOR= ',', ROWTERMINATOR='\n')
```

本例中，myfile 中的数据以及用逗号表示的字段终止符和新行的行终止符，插入到数据库 Mydb 的表 Mytable 中。

CASE

CASE 语句是 Transact-SQL 表达式中使用条件值的工具。

语法

简单 CASE 语句的语法

```
CASE expression
{WHEN expression THEN result} [,...n]
```

[ELSE result]
END

语法

搜索的 CASE 语句的语法：

CASE
WHEN Boolrean_expression THEN result_expression [,...n]
[ELSE else_result_expression]
END

变量

expression	**expression 可以是常数、列名、函数、子查询程序，也可以是算术运算符、按位运算符或串运算符。**
result	**result 是表达式与 WHEN 子句相符时返回的值。**
N	**该变量表示可以多次重复 WHEN 子句。**
Boolean_expression	**布尔表达式表明是否执行 THEN 子句。**
ELSE result	**如果 WHEN 子句失败，就返回 ELSE result。如果没有使用 ELSE，而 WHEN 子句失败，将返回 NULL。**

示例

下面是 CASE 语句的示例：

```
SELECT name = CASE Software
WHEN ' Word Processor' THEN ' Microsoft Word'
WHEN ' Spreadsheet' THEN ' Microsoft Excel'
END
FROM table1
WHERE company = ' Microsoft'
```

本例中，如果在 table1 的 Software 列中找到“Word Processor”或“Spreadsheet”值，就将返回“Microsoft Word”或者“Microsoft Excel”的值。

CAST

CAST 语句和 COWERT 语句相同，它们都用来在 Transact-SQL 中进行数据类型的转换。

语法

```
CAST(expression AS data_type)
```

变量

expression	expression 可以是常数、列名、函数、子查询程序，也可以是算术运算符，按位运算符或串运算符。
data_type	数据类型是列可以转换的任意系统数据类型。

示例

下面是 CAST 语句的示例：

```
CAST(MyDate AS varchar)
```

本例中，MyDate 值转换成可变字符数据类型，而不再是日期时间数据类型。

CEILING

CEILING 函数用于数学环境，它的返回值是大于或等于给定的数值表达式的最短整数。

语法

```
CEILING(numeric_expression)
```

变量 numeric_expression 可以是小数、浮点数、整型数、货币、实数、短整型数、小货币数或微短整型数。

示例

下面是 CEILING 函数的一些示例：

```
SELECT CEILING(1.45)
```

本例中，返回整数 2。

```
SELECT CEILING(-1.45)
```

本例中，返回整数 1。

字符数据类型

列中的值有相同长度时，可用固定长度的字符数据类型存储字母数据。长度不一致时，用可变长度的可变字符数据类型。

语法

```
char [(n)]
```

变量(n)是代表列长度的整数。

CHAR 字符串函数

CHAR 是字符串函数，用于将 ASCII 码整数转换为字符。
语法

CHAR(integer_expression)

变量（integer_expression）是 0～255 间的正整数。
示例
下面是 CHAR 字符串函数的示例：

```
SELECT 'BACKUP DATABASE '+name + ' to' + name +
_dump with init, stats + CHAR(13) + 'GO'
FROM master..sysdatabases
WHERE name NOT LIKE 'tempdb'
```

本例中，以下述语法显示输出结果，并以主从 SQL SELECT 语句的方式建立 Transact-SQL 代码。这表示主 SQL 语句建立以后运行的从 SQL 语句。在运行从属的 Transact-SQL 前需要在编辑过程中删除文本“(4 row(s) affected)”。或者在 SELECT 语句前设置 NOCUNT，就将不打印“(4 row(s) affected)”。可以包括虚线行，因为它被解释为注释。

前面的 SELECT 语句将建立备份 SQL Server 上所有数据库的语法。CHAR(13) 提供换行字符，当需要使用 GO 语句时，GO 语句需要此换行字符为新行的第一个字符。ASCII 码 13 就是换行字符。

```
BACKUP DATABASE master to master_dump with init,statsGO
BACKUP DATABASE model to model_dump with init,statsGO
BACKUP DATABASE msdb to msdb_dump with init,statsGO
BACKUP DATABASE pubs to pubs_dump with init,statsGO
```

CHARINDEX 字符串函数

CHARINDEX 字符串函数在搜索字符串中的模式时，返回模式的开始位置。

语法

```
CHARINDEX('pattern', expression [, start_location])
```

变量

pattern	pattern 是搜索的字母值。
expression	expression 是任意有效的 SQL Server 字符串表达式。
start_location	start_location 是列中开始搜索模式的位置，如果没有提供此可选项，SQL Server

将从第一个位置开始搜索。

示例

下面是 CHARINDEX 字符串函数的示例：

```
SELECT CHARINDEX ('find me in row 4', Column2,10)
FROM MyTable
WHERE Column1='4'
GO
```

本例中，返回的整数是 Column2 中模式“find me in row 4”开始的位置，或是第 10 个字符后面。

CHECKPOINT

当前数据库中的 CHECKPOINT 向 SQL Server 发布命令，将自最后一个检查点以来更新的全部页面写入磁盘。发布 CHECKPOINT 语句检查数据库，其它检查点由 SQL Server 根据 sp_configure 选项的设置自动执行，此选项称为“恢复间隔”集，它决定可接受的最长恢复时间（以分钟为单位）。

语法

```
CHECKPOINT
```

CLOSE

CLOSE 语句关闭打开的游标，保留数据结构，以便能再打开关闭的游标。

语法

```
CLOSE{{ [GLOBAL]  cursor_name} | cursor_variable_name}
```

GLOBAL	GLOBAL 指全局游标。
cursor_name	cursor_name 是要关闭的开启游标名，如果 GLOBAL 是变量，cursor_name 就指的是 GLOBAL 游标，如果没有使用关键词 GLOBAL，指的就是本地游标。
cursor_variable_name	cursor_variable_name 指包含要关掉的开启游标的变量名。

COALESCE

COALESCE 的返回值是按顺序检查每个变量时找到的第一个 NOT NULL 值。SQL Server 将检查第一个变量、列或表达式。如果其为 NULL，就查下一个变量，如果这个又是 NULL，就再查下一个变量，直到检查的变量为 NOT NULL，它就作为返回值。

语法

```
COALESCE(expression [,…n] )
```

变量

expression	表达式可以是常数、列名、函数、子查询程序，也可以是算术运算符，按位运算符和串运算符。
n	n 表示前面的项可以重复。

示例

下面是 COALESCE 函数的示例：

```
SELECT COALESCE (Column1, column2, column3)
FROM MyTable
WHERE Column4 < 9
GO
```

本例中，分析 Column1 中是否有 NOT NULL，如果 Column1 中有 NOT NULL，就返回 Column1 中的值，但如果 Column1 中有 NULL，就分析 Column2 是否有 NOT NULL，如果第二列有值，返回此值，一直往下进行，直到 COALESCE 函数找到某变量中有 NOTNULL 值为止。

COL_LENGTH

COL_LENGTH 返回列的长度，以字节为单位。

语法

```
COL_LENGTH('table', 'column')
```

变量

table	table 是表格名称，可以用 owner.tablename 的形式。
Column	column 是列名。

示例

下面是 COL_LENGTH 函数的示例：

```
SELECT COL_LENGTH('MyTable', 'Column1')
```

本例中，返回表 MyTable 中 Column1 的长度(整数)。

COL_NAME

COL_NAME 在给出函数、表格标识数和列标识数后，返回数据库的列名。

语法

```
COL_NAME(table_id, column_id)
```

变量

table_id	表格的系统 ID。
column_id	列的系统 ID。

COLUMNPROPERTY

COLUMNPROPERTY 返回列的信息或程序变量的信息。

语法

```
COLUMNPROPERTY (id, column,' property ' )
```

变量	
id	id 是表格或程序变量的 ID。
column	column 是列和程序变量的名称。
property	property 指表格或程序变量返回的信息类型。属性的值可以是 'AllowsNull', 'IsIdentity', 'IsIdNot- ForRep1',' IsOutParam',' IsRowGuidCol',' Precision',' Scale', 和 ' UsesAnsiTrim'。

示例

下面是 COLUMNPROPERTY 函数的示例：

```
SELECT COLUMNPROPERTY (
OBJECT_ID ('MyTable'),'Column1','PRECISION')
```

本例中返回值是 MyTable 中 Column1 的精度。

COMMIT TRANS [ACTION]

COMMIT TRANSACTION 或 COMMIT TRANS 是事务处理的结束，此时，从事务处理开始对数据所做的变动将成为数据库的固定内容。

语法

```
COMMIT [TRAN [SACTION] [transaction_name] ]
```

变量 transaction_name 只由程序员用来通知哪个 COMMIT 归属于哪个 BEGIN。

示例

下面是 COMMIT TRANS [ACTION] 语句的示例：

```
BEGIN TRANSACTION
UPDATE Mytable
SET Column1 = 3
WHERE Column2 = 8
GO
COMMIT TRANSACTION
GO
```

本例中，事务处理以 BEGIN TRANSACTION 开始，以 COMMIT TRANSACTION 结束。

COMMIT［WORK］

COMMIT WORK 是事务处理的结束，此时，从事务处理开始对数据所做的变动将成为数据库的固定内容。它和 COMMIT TRANSACTION 完全一样，是为与 ANSI SQL-92 兼容而添加的。

语法

COMMIT ［WORK］］

变量 WORK 是可选项。单词 COMMIT 和 COMMIT WORK 起同样的作用。

示例

下面是 COMMIT［WORK］语句的示例：

```
BEGIN TRANSACTION
UPDATE Mytable
SET Column1 = 3
WHERE Column2 = 8
GO
COMMIT
GO
```

本例中，以 COMMIT 开始和结束事务处理。

CONTAINS

CONTAINS 搜索以字符为基础的数据类型中列的精度或其他方面的匹配，例如单词或词组，单词或词组的前缀，一个单词旁边的另一个单词，由一个单词生成的另一个单词 (例:make 和 made)，以及比一个单词的权值更高的另一个单词。此函数取决于数据库／表格／列能否进行全文搜索。请参阅第 22 章了解全文存储过程。

语法

```
CONTAINS
 ({    column     |*}, ) ' <contains_search_condition> '
)
<contains_search_condition>::=
{
<simple_term>
| <prefix_term>
| <proximity_term>
| <generation_term>
| <isabout_term>
| (<contains_search_condition>)
}
```

```
 [ {
{AND | AND NOT | OR } < contains_search_condition>
}
]    [...   n   ]

<isabout_term>::=
ISABOUT
({
<generation_term>
| <prefix_term>
| <proximity_term>
| <simple_term>
}
 [ WEIGHT (   weight_value)  >
)

< generation_term>::=
FORMSOF (INFLECTIONAL, <simple_term>)

<prefix_term>::=
   word     * | "   phrase     * "
```

```
<proximity_term>::=
{<simple_term> | <prefix_term>} [NEAR() | ~]

<simple_term>::=
   word      |"   phrase    "
```

变量

column	**column 指字符类型列的名称。列必须注册才能进行全文搜索。**
*	*表示表中所有的列都适于搜索(注册了全文搜索)。如果 FROM 子句中有多个表格，就要用表名限定。
<contains_search_condition>	< contains_search_condition>指要在列中搜索的文本。
word	word 单词指中间无空格、无标点的一串字符。
phrase	phrase 指带空格的一个或多个单词。
<isabout_term>	<isabout_term>决定返回的行与有各自加权值的单词或词组列表相匹配。
ISABOUT	ISABOUT 是<isabout_term>的关键词。
WEIGHT (weight_value)	weight_value 定义加权值，其值位于 0.0～

	1.0 之间。<isabout_term>的每一部分都有一个加权值。
AND | AND NOT | OR	AND | AND NOT | OR 是逻辑运算符。
<generation_term>	<generation_term>是当<simple_terms>与原来搜索的单词不同时的单词匹配。
INFLECTIONAL	INFLECTIONAL 表示名词的单数或复数形式，动词的各种时态互相匹配。
<prefix_term>	<prefix_term>指定匹配的单词或词组以指定的文本开头(例如，“MS*”搜索以“MS”开始的任意串)。
<proximity_term>	< proximity_term>定义匹配的单词或词组互相近似。
NEAR() | ～	NEAR() | ～表示 NEAR()或～一侧的串、单词或词组与其另一侧的串、单词或词组相近。
<simple_term>	<simple_term>确定确切匹配单词或词组(词组括在双引号内)。
N	n 指前面的项可重复。

示例

下面是关键词 CONTAINS 的示例：

```
SELECT MyFullTextEnabledColumn
FROM MyTable
WHERE CONTAINS (MyFullTextEnabledColumn,' computer * OR hardware*
' )
```

本例中，MyFullTextEnabledColumn 中，包含所提供的字符串的值显示为结果集。

示例

下面是带有近似语法的 CONTAINS 的示例：

```
SELECT MyFullTextEnabledColumn
FROM MyTable
WHERE CONTAINS (MyFullTextEnabledColumn,' Diana NEAR Princess' )
```

本例中，MyFullTextEnabledColumn 的值中，显示所提供的单词中相互近似的单词。

示例

下面是使用 CONTAINS 后缀语法的示例：

```
SELECT MyFullTextEnabledColumn
FROM MyTable
WHERE CONTAINS (MyFullTextEnabledColumn, ' FORMSOF
(INFLECTIONAL, agree)' )
```

本例中，显示 MyFullTextEnabledColumn 中与所提供的单词形式相符的值(agreeable，agreed，agreeing，agrees，等等)。

示例

下面是使用 CONTAINS 加权语法的示例：

```
SELECT MyFullTextEnabledColumn
FROM MyTable
WHERE CONTAINS (MyFullTextEnabledColumn, ' ISABOUT ( dolphin weight (.8),whale weight (.4),fish weight (.2)) ' )
```

本例中，搜索 MyFullTextEnabledColumn 中包含单词海豚、鲸和鱼并带有相应权重的行。

CONTAINSTABLE

CONTAINSTABLE 返回的表格中，列与单词或词组精确或不太精确的匹配，单词与另一些单词近似或者权重匹配。它像表格名称一样可用于 SELECT 语句的 FROM 子句。它还返回每行的级别值。详细内容可参阅 CONTAINS。

语法

```
CONTAINSTABLE (table, {column | *}, ' <contains_search_condition> ' )
<contains_search_condition>::=
{
```

| <generation_term>
| <prefix_term>
| <proximity_term>
<simple_term>
| < weighted_term >
| (<contains_search_condition>)
}
[
{
{AND | AND NOT | OR } < contains_search_condition>
}
] [...n]
< weighted_term >::=
ISABOUT
({{
<generation_term>
| <prefix_term>
| <proximity_term>
| <simple_term>
}
[WEIGHT (weight_value)]
} [,...n]

)
< generation_term>::=
FORMSOF (INFLECTIONAL, <simple_term> [,...n])
<prefix_term>::=
{ " word * " | "phrase * " }
<proximity_term>::=
{<simple_term> | <prefix_term>}
{{NEAR | ~} {<simple_term> | <prefix_term>}} [...n]

<simple_term> ::=
word | "phrase "

变量

table	**table 是启用全文搜索的表格的可选限定名。**
Column	**column 指要全文搜索的所启用的列名。**
*	***指表格中所有的列都可以全文搜索。**
<contains_search_condition>	**contains_search_condition 指全文搜索所启用列中的文本(不是一个变量)。**

CONTINUE

CONTINUE 语句有时跟 IF 语句一起使用，它可用 CONTINUE 后面没有执行的内容重新启动 WHILE 循环。请参看 WHILE。

语法

```
CONTINUE
```

CONVERT

CONVERT 用于表达式由一种数据类型转换成另一种数据类型的明显式转换，可使用表达式的地方就可使用 CONVERT。

语法

```
CONVERT (data_type [(length)], expression [,style] )
```

变量

data_type	data_type 是表达式要使用的数据类型，但不是用户定义的数据类型。
length	length 是表达式的可选长度。

expression
任意有效的表达式。

style
style 是将日期时间或小日期时间数据转换成字符数据时使用的数据格式结构，它可以是字符、可变字符、nchar、nvarchar 数据类型。如果使用字符串格式将浮点数、实数、货币或小货币型数据转换成字符数据时，数据类型可以是字符、可变字符、nchar 或 nvarchar 数据类型。
日期时间或小日期时间转换成字符的结构(为年份添加 100，用 yyyy 格式表示年，而不用 YY 格式表示年)：
-0 或 100 mon dd yyyy hh:miAM 或 PM
1 或 101 USA mm／dd／yy
2 或 102 ANSI yy.mm.dd
3 或 103 英／法 dd／mm／yy
4 或 104 德 dd.mm.yy
5 或 105 意 dd-mm-yy
6 或 106 -dd mon yy
7 或 107 -mon dd,yy
8 或 108 -hh:mm:ss
-9 或 109 缺省值＋毫秒 mon dd yyyy

hh:mi:ss:mmmAM 或 PM
10 或 110 USA mm-dd-yy
11 或 111 日本 yy／mm／dd
12 或 112 ISO yymmdd
-13 或 113 欧洲 缺省值＋毫秒 dd mon yyyyhh:mm:ss:mmm(24 小时)
14 或 114-hh:mi:ss:mmm(24 小时)
-20 或 120ODBC 规范 yyyy-mm-dd hh:mi:ss(24 小时)
-21 或 121ODBC 规范(用毫秒)
yyyy-mm-dd hh:mi:ss.mmm(24 小时)
浮点数或实数型数据转换成字符的结构如下:
0(缺省值)=6 位或小于 1=8 位
2=16 位
money 或 smallmoney 转换成字符的结构如下:
0=无逗号，2 位小数(例:1234.11)，这是缺省值。
1=逗号和 2 位小数(例:1,234.11)
2=无逗号，小数点右边 4 位(例:1234.1111)

示例
下面是 CONVERT 函数的示例：

```
SELECT ' My Birthday is Today ' + CONVERT(varchar(12),getdate())
```

本例中，日期转换成字符串后，字符串就能与日期串接在一起。用加号还可以串接字符数据类型。

COS

COS 为数学函数，它返回表达式中角度的余弦。
语法

```
COS(float_expression)
```

变量 float_expression 是浮点型数据。

COT

COT 为数学函数，它返回表达式中角度的余切。

语法

COT(float-expression)

变量 float-expression 是浮点型数据。

COUNT

COUNT 是一个集合函数，它返回的整数代表表达式中(通常是一列)值的个数。DISTINCT 和 COUNT 一起使用时，返回值为单一值的个数。

语法

COUNT({ [ALL | DISTINCT] expression] | *})

变量

ALL	**ALL 是缺省值，计数所有的值，但忽略 NULL。**
DISTINCT	**DISTINCT 和 COUNT 一起使用时，返回单一值的个数。**
expression	**表达式可以是常量、列名、函数、子查询程序、算术运算符、按位运算符和串运算符。**
*	*表示所有行。

示例
下面是 COUNT 集合函数的示例：

```
SELECT COUNT(Column1)
FROM Mytable
```

本例中，COUNT 函数将计数 Mytable 的 Column1 中没有空值的行数，如果使用*而不是列名，将返回表中所有行的数目。

CREATE DATABASE

CREAE DATABASE 是建立新数据库及其相关文件的语句。
语法

```
CREATE DATABASE database_name
 [ ON  [PRIMARY]
 [ <filespec>  [,...n]]
 [ ,<filegroup>  [,...n]]
]
 [ LOG ON { <filespec> } ]
 [ FOR LOAD | FOR ATTACH ]
<filespec>::=
```

```
( [NAME = logical_file_name, ]
FILENAME = ' os_file_name'
 [,SIZE =size]
 [, MAXSIZE= { max_size | UNLIMITED} ]
 [,FILEGROWTH=growth_incrernent] ) [,...n]
<filegroup>::=
FILEGROUP filegroup_name <filespec>  [,...n]
```

变量

database_name

用标识符规则建立的新数据库名称。

[ON [PRIMARY] [<filespec> [,…n]] [,<filegroup> [,…n]]

如果使用 ON，就要提供存储数据的磁盘文件名称。ON 后面跟一系列文件，这些文件是主文件组的数据文件，文件中也可以有用户建立的文件组列表。如果没有使用关键词 PRIMARY，列出第一个文件就存储为主文件。

PRIMARY

PRIMARY 表示 filespec 列表包含主文件组中的文件，而主文件组包含文件组中没有定义的系统表和对象。数据库只能有一个 PRIMARY 文件，它命名数据库的起始文件。如果不使用关键

	词 PRIMARY，列出的第一个文件就存储为主文件。
n	n 表示可以多次重复命名文件。
LOG ON	如果使用 LOG ON，就要表示为事务处理记录提供记录文件名。LOG ON 后面跟一系列文件，这些文件将成为数据库的记录文件。如果 CREATE DATABASE 语句中省略了 LOG ON，就将建立一个由 SQL Server 选定名称的记录文件，其容量为指定数据文件容量的 25%。
FOR LOAD	只向后兼容， SQL Server 7.0 不需要该变量，因为 SQL Server 可以在恢复时建立数据库。
FOR ATTACH	FOR ATTACH 表示有现成的操作系统文件集，必须提供定义第一个主文件的 filespec，如果与文件首次建立或者最后附加时的路径不同，还需要为不同路径的文件提供 filespec 列表，除非必须多于 16 个 filespec，否则，可使用 sp_attach_db，而不使用 CREATE DATABASE FOR ATTACH。
NAME	NAME 指要引用的文件的逻辑名称。

logical_file_name	文件使用的逻辑名称。
FILENAME	FILENAME 表示文件的实际或操作系统文件名。
'os_file_name'	在本地服务器上使用的操作系统文件名和路径。
SIZE	指位已命名的文件指定容量。
size	size 是文件的起始容量，MB 或 KB 和数字一起使用表示兆字节或千字节；缺省值为 MB，容量的最小允许值为 1MB。如果不用 SIZE，建立文件时，数据文件的容量为 3MB，记录文件的容量为 1MB。
MAXSIZE	MAXSIZE 表示提供文件可能扩展的最大容量。
max_size	max_size 表示文件能扩展的最大容量，如果没有提供 max_size，文件将扩展到磁盘满为止。当磁盘接近某一容量百分数时，应使用 SQL Server Agent 生成警告。
UNLIMITED	UNLIMITED 指文件可以扩展至磁盘满为止。
FILEGROWTH	FILEGROWTH 表示提供文件扩展时的增

	量，其总值不能超过 MAXSIZE。
growth_increment	growth_increment(缺省值为 256KB，最小值为 64KB)给出扩展增量，该增量化整至最接近 64KB。

示例

下面是 CREATE DATABASE 语句的示例：

```
USE master
GO
CREATE DATABASE mydb
ON
( NAME = 'mydb_dat1',
FILENAME = 'c:\mssql7\data\mydbdat.mdf',
SIZE=100,
MAXSIZE =900,
FILEGROWTH =5 )
LOG ON
( NAME = 'mydb_log1',
FILENAME = 'd:\mssql7\data\mydblog.ldf',
SIZE = 25MB,
MAXSIZE =200MB,
FILEGROWTH =5MB )
GO
```

在本例中，数据文件和记录文件中建立了一个新的数据库。

CREATE DEFAULT

CREATE DEFAULT 语句建立一个需要限制列或用户定义的数据类型的对象(这就是最好使用 CREATE 或 ALTER TABLE 语句中 DEFAULT 子句的原因)。当插入了一行，而又没有提供数值时，利用用户定义的数据类型，其缺省值可以确定插入一列或所有列中的数值。如果重复使用缺省值，就可以使用 CREATE DEFAULT 语句，因为它比编写 CREATE TABLE 语句更为有效。

语法

```
CREATE DEFAULT default
AS constant_expression
```

变量

default	default 是使用标识符规则的缺省名称。
constant_expression	常数表达式是常数值，而不是数据库的对象名称。可以使用任意常数、函数、数学表达式和全局变量。字符与日期应加上单引号。二进制数据加 0x 前缀，货币加前缀美元符号 ($)。

示例

下面是 CREATE DEFAULT 语句的示例：

```
CREATE DEFAULT Phone_Default AS ' No Phone Number'
GO
sp_bindefault Phone_Default, ' Mytable.Telephone_Nbr'
GO
```

本例中，建立了 Phone_Default，当没有给出 Mytable 中 Telenhone_Nbr 列的值，就用 INSERT 语句在该列中插入字符“ No Phone Number”。

CREATE INDEX

CREATE INDEX 语句建立表格的索引，但是用户必须是表格的拥有者，如果所建索引是 PRIMARY KEY,可在 CREATE TABLE 或 ALTER TABLE 语句中使用 PRIMARY KEY 约束。使用数据库.表格拥有者.表名限制表名，并限制数据库和拥有者，可以在另一数据库的表格中建立索引。

语法

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
INDEX index_name ON table (column [,...n] )
 [WITH
  [PAD_INDEX]
```

```
    [[,]   FILLFACTOR = fillfactor]
    [[,]   IGNORE_DUP_KEY]
    [[,]   DROP_EXISTING]
    [[,]   STATISTICS_NORECOMPUTE]
  ]
   [ON filegroup]
```

变量

UNIQUE	**UNIQUE 表示建立唯一的索引，此索引禁止组成该键的列有相同的值，SQL Server 通过检查重复值来保证上述情况。SQL Server 不允许在带有重复列的行上放置索引。即使重复行为空，也必须更新或删除它们才能建立唯一索引。**
CLUSTERED	**CLUSTERED INDEX 表示行的物理顺序与行的索引顺序相同，这些行带有保存数据行的成簇索引的叶页。如果建立了成簇索引，表格中的所有非成簇索引也都重建。这表示第一步应先建立成簇索引，特别是有删除或重建索引的脚本时就更应如此。成簇索引是已分类的表格，所以建立成簇索引，并使用 ON filegroup 子句可将已分类的表格移到在 CREATE INDEX 语句中指定的新文件组上，要保**

	证文件有足够的空间(表格所需空间的 1.2 倍)。
NONCLUSTERED	NONCLUSTERED 索引确定表格的逻辑顺序。与 CLUSTERED 索引不同，逻辑顺序与物理顺序不同，NONCLUSTERED 索引包含有实际物理行的指针，指针可以是目前存储在 NONCLUSTERED 索引而不是过去版本的 Row ID (RID)中的成簇键码。如果表格没有 CLUSTERED 索引，指针就指向由文件 ID、页数和行 ID 组成的行。每个表格最多可以有 249 个 NONCLUSTERED 索引。但是，表格中索引越多，它占用的 INSERT 行就越长。
index_name	index_name 是索引的名称，而且，按照标识符规则，每个表格(不是每个数据库)必须有唯一的索引名称。
table	table 是包含进行索引的列的表格名称。如果提供了数据库和表格拥有者，且有许可，就可以在另一个数据库中建立索引。
column	column 是要建立索引的列名。需列出多个列名，这些列名放在圆括号内，并用逗号分隔开，按建立复合索引的优先分类进行排序。不适用于文本、图像、位和已计算的列，它们也不能用作任何索引列。复合索引最多为 16 列，其最大容量为 900 字节。

n	n 指前面的列可以重复。
PAD_INDEX	PAD_INDEX 指将 FILLFACTOR 应用于内部(非叶)索引页(节点)以及叶页。
FILLFACTOR= fillfactor	FILLFACTOR 决定每一索引页的填满程度，而且只在第一次创建索引时使用，但是页面的实际填满程度会随时间的改变而改变。有效值是从 1～100，如果不提供该值,缺省值为 0。可以用 sp_configure 改变 FILLFACTOR 的缺省值。如果 FILLFACTOR 为 0，索引节点中留出至少两项的空间，其中一项就是索引页面的索引指针。如果 FILLFACTOR 为 100，所有的数据将 100%地压缩在一起，这只适用于只读表格。当然，FILLFACTOR 决定索引和数据(在成簇索引的情况下)在磁盘上占有的空间量。用户定义的 FILLFACTOR 越低，索引和数据页就需要越多的空间存储数据。
IGNORE_DUP_KEY	IGNORE_DUP_KEY 决定重复键和其他行一起插入唯一成簇索引时的结果。SQL Server 将发出警告，但是会插入 INSERT 中非重复键的其他行。忽略重复键的行，且不插入它们，如果不使用 IGNORE_DUP_KEY，将部插入所有的行，不只是重复键行。
SORTED_DATA_REORG	SORTED_DATA_REORG 通知 SQL Server，数据已经

	分类，只需在磁盘上重复组织它们。当使用 FILLFACTOR OPTION，改变数据在页面中的存储方式时，可使用 SORTED_DATA_REORG。用 DBCC SHOWCONTIG 可决定表格是否分段，是否需要重新组织。
DROP EXISTING	使用 DROP EXISTING 将删除并重建存储索引。同时更新所有已有的非成簇索引。该变量比 DROP 和 CREATE 更有效，因为只需进行一次更新，而不必在删除成簇索引后更新一次，在重建成簇索引后再更新一次。
STATISTICS_NORECOMPUTE	使用 STATISTICS_NORECOMPUTE 也可以提高效率，这表示指索引统计数字不会自动重新计算。使用 STATISTICS_NORECOMPUTE 可能带来伤害，因为它不能提供查询优化程序进行分布统计，也就不能指定以后的查询计划。
ON filegroup	ON filegroup 表示放置索引的文件组。如果不使用 ON filegroup，索引将与表格放在相同的文件组中。

示例

下面是 CREATE INDEX 语句的示例：

```
CREATE INDEX Column1_2_ind ON Mytable (Column1,Column2)
```

本例建立了一个复合索引。

CREATE PROCEDURE

CREATE PRECEDURE 语句建立存储过程，此存储过程是 Transact-SQL 语句的编译集，能有选择性地接受变量。

语法

```
CREATE PROC [EDURE]      procedure_name      [;number]
 [
({ [@]      parameter data_type}      [VARYING ]     [=      default     ]
 [OUTPUT] )



]
 [,...n]
 [
WITH {RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION}
]
 [ FOR REPLICATION]
AS
```

sql_statement [...n]

变量	
procedure_name	procedure_name 是新存储过程使用标识符规则的名称，存储过程名对于数据库和拥有者是唯一的。在局部临时程序的名称前加一个英磅符，或在全局临时程序的名称前加两个英磅符，就可以建立临时存储过程，存储过程也可创建为启动存储过程，在 SQL Server 用 sp_makestartup 系统存储过程启动时，该存储过程将自动运行。存储过程名不得超过 128 个字符，其中包括英磅符“#”。
; number	number 是个整数，用于将同名的程序组合在一起，这样便于用一个 DROP PROCEDURE 语句将它们全部删除。名为 samename;1,samename;2 等的存储过程可以用“DROP PROCEDURE samename”一起删除，这将删除整个组，一旦有这样的组，就不能删除单个程序。
[@] parameter	@parameter 指程序的参数或变量。0,1

	或 1 个以上的变量都可以在 CREATE PROCEDURE 语句中声明。如果没有编写缺省值，当运行程序时，由用户提供每个声明变量的值。变量最多可以有 1024 个，每个变量都是存储过程的局部变量。
data_type	data_type 是指变量或参数的数据类型。游标数据类型只能用于输出变量，而且必须同 VARYING 一起使用。
VARYING	VARYING 将返回集定义为输出参数，它用于游标数据类型。
default	default 是指变量或参数的缺省值，它必须是个常数。利用缺省值，不必提供参数或变量的值，就能运行存储过程。如果以后和 LIKE 一起使用变量，就可以使用通配符(%，_，[] 和 [^]，缺省值可以是 NULL。
OUTPUT	OUTPUT 表示参数或变量是返回参数。值返回至调用程序，不适用于文本参数，因为它们不能用作 OUTPUT 参数或变量，但是关键词 OUTPUT 可以保存游标占位符。

n	n 指前面的项可以重复。
{RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION}	RECOMPILE 指每次运行存储过程时都重新编译它。ENCRYPTION 通知 SQL Server 为存储过程加密存储在 syscomments 系统表中的行。
FOR REPLICATION	FOR REPLICATION 表示由该子句特地为复制建立的存储过程不能在预定服务器上运行，它用于建立只能通过复制来执行的筛选存储过程。FOR REPLICATION 不能和 RECOMPILE 选项一起使用。
AS	AS 先于程序执行的语句。
sql_statement	SQL 语句指存储过程中要运行的多个 Transact-SQL 语句。
n	n 指可能有多个 SQL 语句。

示例

下面是 CREATE PROCEDURE 的示例：

```
CREATE PROCEDURE  MyProceduce
AS
SELECT Column1,Column2
```

```
FROM Mytable
WHERE Column3 = 8
GO
```

本例中，建立了一个存储过程。

CREATE RULE

CREATE RULE 建立数据库对象 rule。规则可以连接列或用户定义的数据类型，决定哪些值可以放在列中。或者可以用 CHECK CONSTRAINTS 来查看，因为可以在一列或多个列上定义多个约束。一列只能连接一个规则。

语法

```
CREATE RULE rule
AS condition_expression
```

变量

rule	**rule 是使用标识符规则的规则名称。**
condition_expression	**condition_expression 是在定义规则的 WHERE 子句中有效的任意表达式。规则不能引用任何其它的列或数据库**

对象。列中的新值以加'@'前缀的标识符表示。

示例

下面是 CREATE RULE 的示例：

```
CREATE RULE Color_Rule
AS
@color IN ('Red', 'Blue','Yellow')
GO
sp_bindrule 'Color_Rule', 'Mytable.Column3'
GO
```

本例中，规则创建为数据库对象，并与列相连接。也可将规则与用户定义的数据类型相连接。用 sp_unbindrule 切断与对象的连接。

CREATE SCHEMA

CREATE SCHEMA 建立模式，模式是没有数据的数据库结构，它可用于开发环境或测试环境。

语法

```
CREATE SCHEMA
 [AUTHORIZATION owner]
```

[schema_element [schema_element2 [...schema_elementn]]]

变量

AUTHORIZATION owner	AUTHORIZATION owner 是数据库中有效安全帐户的注册 ID。
schema_element={table_definition | view_definition | grant_statement}	table_definition = CREATE TABLE 语句，它在此模式中建立表格。view_definition=CREATE VIEW 语句，它在此模式中建立视图，grant_statement=GRANT 语句，它在此模式中给用户或用户组授予许可。

示例

下面是 CREATE SCHEMA 语法的示例：

```
CREATE SCHEMA AUTHORIZATION MyAccount
GRANT SELECT on View1 TO public
CREATE VIEW View1 (Column1) AS SELECT Column1 from MyTable
CREATE TABLE MyTable (Column1 int)
```

本例中，为安全帐户所拥有的数据库对象创建 MyAcount 模式。

CREATE TABLE

CREATE TABLE 指在数据库中建立新的表格。

语法

```
CREATE TABLE    table_name
({<column_definition >
|    column_name    AS    computed_column_expression
| <table_constraint>
}  [, ...   n   ]
)
 [ON {   filegroup    ／DEFAULT}]
 [TEXTIMAGE_ON    filegroup   ]
<column_definition> ::= {    column_name data_type   }
 [  NULL | NOT NULL ]
 [  IDENTITY  [(   seed    [,    increment   ]  )
 [NOT FOR REPLICATION]  ]  ]
 [  ROWGUIDCOL ]
 [  <column_constraint> ::=
 [CONSTRAINT    constraint   _   name   ]
{ { PRIMARY KEY ＼ UNIQUE }
 [CLUSTERED | NONCLUSTERED]
```

```
 [WITH [FILLFACTOR = fillfactor ]
 [ON fillegroup ]
| [FOREIGN KEY]
REFERENCESref_table
 [ ( ref _ column ) ]
 [NOT FOR REPLICATION]
|DEFAULT constant_expression
| CHECK [NOT FOR REPLICATION]
( logical_expression )
}
] [ ... n ]
<table_constraint> ::= [CONSTRAINT constraint_name ]
{ [ { PRIMARY KEY |UNIQUE }
 [ CLUSTERED |NONCLUSTERED]
{ ( column [,... n ] ) }
 [ WITH [FILLFACTOR= fillfactor ]

]
 [ON { filegroup ／ DEFAULT}]
]
| FOREIGN KEY
 [( column [,...n] )]
```

```
    REFERENCES ref_table [( ref_column [,...n] )]
     [NOT FOR REPLICATION]
    | CHECK [NOT FOR REPLICATION]
    ( search_conditions )
    }
```

变量

table	**table 是要建立的表格名称。**
column_name	**column_name 是要建立的列名，时间戳数据类型可以省略列名。timestamp 将成为 SQL Server 所使用的列名。**
AS computed_column_expression	**已计算的列是虚拟列，它是个新选项，现在，已计算的列可以存储和检索，但不能 INSERT 或 UPDATE 计算列。计算列不能用作 PRIMARY KEY，UNIQUE、FOREIGN KEY 或 CHECK 约束中的关键列。**
ON filegroup	**ON filegroup 决定为约束建立的表格在磁盘上驻留的位置。如果不用 ON filegroup，表格将放在缺省的文件组中。**
data_type	**data_type 与新列的 SQL Server 数据类型有关，如果使用唯一标记符数据**

	类型，而且表中有现成的行，就可以用 DEFAULT 约束和 NEWID()函数，为现有行中的新列生成唯一标识符值。
NULL | NOT NULL	NULL | NOT NULL 决定列是否可以保存 NULL 值。
IDENTITY	IDENTITY 是用来在新列中放置递增整数的属性。将来添加新列时，该属性也为真。
seed	seed 与用 IDENTITY 属性添加一列时希望在第一行中出现的的整数值有关，它表示新 IDENTITY 列的值开始的位置。如果不用此关键词，SQL Server 将列的种子值定为 1。
increment	如果用 IDENTITY 属性添加一列，Increment 是指在添加新行时，新 IDENTITY 列中的值的增量。如果不使用 increment，SQL Server 对所添加的最后一行中的值增加 1。
NOT FOR REPLICATION	如果使用复制，并用 IDENTITY 属性添加一列，就可以用 NOT FOR REPLICATION 通知 SQL Server，当 SQL Server 系统复制注册(例如 sqlrepl)

	将数据插入表格中时，不施加IDENTITY条件。
ROWGUIDCOL	ROWGUIDCOL 属性是全局唯一标识符列，它只能分配给唯一标识符列，而且每个表格一个。
CONSTRAINT	CONSTRAINT 选项是如何建立 PRIMARY KEY、UNIQUE、FOREIGN KEY、CHECK 或 DEFAULT 约束。CONSTRAINTS 用来保持数据的完整性。添加 CONSTRAINT 可以建立索引。
constraint_name	constraint_name 是指新建立的 CONSTRAINT 名称。如果没有给 CONSTRAINT 命名，SQL Server 将为其命名。
PRIMARY KEY	PRIMARY KEY 是一种约束(每个表格只有一个)，它在列中建立唯一索引，并将列指定为表格的主键。如果计划在引用此表的其它表格中包含 FOREIGN KEY，就需要一个 PRIMARY KEY。
UNIQUE	列值对于表格可以是唯一的，但不指定为主键。UNIQUE 约束在列中建立唯

	一的索引。
CLUSTERED|NONCLUSTERED	要建立一个 PRIMARY KEY(缺省 CLUSTERED)或唯一 UNIQUE(缺省 NONCLUSTERED)约束，CLUSTERED 或者 NONCLUSTERED 决定要添加的索引类型。
WITH FILLFACTOR=fillfactor	WITH FILLFACTOR 描述每个索引页有多少数据。可选值为 1～100，缺省值为 0，以百分数表示。填充因子越低，可用空间就越多。这样就在页面上留下较多的空间以插入新行，而不必强迫 SQL Server 为新行分配较多的空间。
ON filegroup	ON filegroup 确定为约束建立的索引在磁盘上驻留的位置。如果不使用 ON filegroup，索引将和表格放在相同文件组中。
FOREIGN KEY… REFERENCES	FOREIGN KEY… REFERENCES 建立约束并通过保证列中的每一项存在于引用表格的列中来实施引用的完整性。
ref_table	ref_table 是指 FOREIGN KEY 约束是引用的表格名称。

ref_column	ref_column 是指 FOREIGN KEY 约束所引用的列。
NOT FOR REPLICATION	如果使用复制，并用 IDENTITY 属性添加列，就可以用 NOT FOR REPLICATION 通知 SQL Server，当 SQL Server 系统复制注册(例如 sqlrepl)将数据插入到表格中时，不施加 IDENTITY 条件。
DEFAULT	DEFAULT 是一种约束，它决定如果为 INSERT 上的列指定数值，在列中插入哪些值。对带有时间戳的数据类型、IDENTITY 属性或另一个 DEFAULT 约束的列不能使用该约束，因为一列只能有一个值。
constant_expression	constant_expression 和 DEFAULT 一起使用，作为列的缺省值，它可以是常数，NULL 或系统函数。
CHECK	CHECK 是一种约束，它限制可以插入列中的值。
logical_expression	logical_expression 可以引用同一行中其他的列，并返回真或假。
column [,…n]	column 代表约束的列的清单。

示例

下面是 CREATE TABLE 语句的示例：

```
CREATE TABLE Mytable
(
MyTableID int identity,
MyDate datetime,
UsedByName AS USER_NAME()
)
```

本例中，建立了表格，并将 USER_NAME 系统函数作为计算列 UsedByName。

CREATE TRIGGER

CREATE TRIGGER 建立触发器，该触发器定义为一种特殊的存储过程，且该存储过程在用户发出作用于表格的 INSERT、UPDATE 或 DEFAULT 语句时运行。如果表格上有 CONSTRAINTS，它们将在引发触发器前检查约束。如果 CONSTRAINT 的有效性检查失败，将不引燃触发器。现在，对于任意的 INSERT、UPDATE 或 DEFAULT 语句，可以有多个触发器。

语法

```
CREATE TRIGGER trigger_name
```

```
ON table
 [WITH ENCRYPTION]
{
{FOR { [,]  [INSERT]  [,]  [UPDATE  [,]  [DELETE] }
 [WITH APPEND]
 [NOT FOR REPLICATION]
AS
   sql_statement  [,...n]
}
|
{FOR { [,]  [INSERT]  [,]  [UPDATE] }
 [WITH APPEND]
 [NOT FOR REPLICATION]
AS
IF UPDATE (  column  )
 [{AND | OR} UPDATE (  column  ) [,...n]]
   sql_statement     [,...  n  ]
| IF (COLUMNS_UPDATED() {bitwise_operator} updated_bitmask)
{comparison_operator} column_bitmask  [...n]
}
}
```

变量	
trigger_name	trigger_name 是指使用标识符规则的触发器名。在数据库中该变量是唯一的。
table	指触发器运行的表格。
WITH ENCRYPTION	WITH ENCRYPTION 加密触发器的 syscomments 系统表格行。
{ [,] [INSERT] [,] [UPDATE] [,] [DELETE] } | { [,] [INSERT] [,] [UPDATE] }	INSERT、UPDATE AND DELETE 是引发或运行触发器的数据修改语句。可以以任意顺序将它们组合在一起使用，并用逗号分隔开它们。
WITH APPEND	WITH APPEND 指要添加现有类型的另一个触发器(兼容级别低于 70 时，只向后兼容)
NOT FOR REPLICATION	如果复制修改表格，且使用着 NOT FOR REPLICATION，将不运行触发器。
AS	AS 优先于触发器执行的语句。
sql_statement	Transact-SQL 语句建立触发器，但不应用来将数据返回给用户。只供触发器调用进行删除或插入的特殊表格叫概念表格。它们类似于表格，在其上放置触发器并保存变化的行的新旧数

值。可以在触发器中引用它们。

示例
下面是 CREATE TRIGGER 语句的示例：

```
CREATE TRIGGER MyTrigger
ON MyTable
FOR INSERT,UPDATE,DELETE
AS
EXEC master..xp_sendmail ' gaylecof ', ' MyTable has changed.'
GO
```

本例中，建立了表格变化时发送邮件的触发器。

CREATE VIEW

CREATE VIEW 语句建立虚拟的表格，此表格表示数据的逻辑视图，而不直接访问物理表格。CREATE VIEW 不能使用 ORDER BY 子句，COMPUTE 子句，或 COMPUTE BY 子句，或关键词 INTO，也不能引用临时表。

语法

```
CREATE VIEW    view_name [(column  [,...n] )]
```

WITH ENCRYPTION]
AS
select_statement
[WITH CHECK OPTION]

变量

view_name	**view_name 是指使用标识符规则的视图名称。**
column n	**column 指视图中的列名。n 指列可以重复。**
WITH ENCRYPTION	**加密视图的 syscomments 系统表格行。**
AS	**视图的操作。**
select_statement	**定义视图的 Transact SQL-SELECT 语句。**
WITH CHECK OPTION	**WITH CHECK OPTION 保证修改视图数据以后，可通过视图看见数据。**

示例

下面是 CREATE VIEW 语句的示例：

```
CREATE VIEW MyView (Column1,Column2,Column3)
WITH ENCRYPTION
AS
SELECT a.Column1,
  a.Column2,
  b.Column3
FROM MyTable a,
  YourTable b
WHERE a.Column1 = b.Column1
GO
```

CURRENT_TIMESTAMP

CURRENT_TIMESTAMP 可以用作函数，或者在 CREATE TABLE 或 ALTER TABLE 语句中与 DEFAULT 约束一起使用，对保存当前数据和时间的列建立缺省值。

语法

```
CURRENT_TIMESTAMP
```

示例

下面是 CURRENT_TIMESTAMP 函数的示例：

```
DECLARE @myvariable datetime
SET @myvariable = CURRENT_TIMESTAMP
```

本例中，将当前的数据与时间提供给局部变量 Myvariable。

CURRENT_USER

CURRENT_USER 可用作函数，也可以和 CREATE TABLE 或 ALTER TABLE 语句中的 DEFAULT 约束一起使用，为保存当前用户的列建立缺省值。

语法

```
CURRENT_USER
```

示例

下面是 CURRENT_USER 函数的示例：

```
DECLARE @myvarlable sysname
SET @myvariable = CURRENT_USER
```

本例中，将当前游标提供给局部变量 myvariable。

游标数据类型

游标数据类型用于创建游标变量，需要用它作为游标 OUTPUT 变量，它不能用于 CREATE TABLE 语句。只能用 SET @local_variable 语句和 CURSOR_STATUS 函数对游标数据类型进行操作。

CURSOR_STATUS

CURSOR_STATUS 为一标量函数，它允许通知存储过程的调用者，存储过程是否为给定的参数返回游标和结果集。

如果返回值为 1，游标的结果集至少有一行用于感应游标、不感应游标和动态游标；返回集有 0，1 或多行。打开游标，如果返回值为 0，游标的返回集为空，如果返回值为-1，关闭光标。如果返回值为-2，就没有给已声明的游标变量分配游标。如果返回值为-3，指定名称的游标不存在或者还没有给它分配游标。

语法

```
CURSOR_STATUS
(
{
{ 'local',    cursor_name   }
```

```
|{ ' global ' ,    cursor_name    }
|{ ' variable ' ,    cursor_variable}
}
)
```

变量

' local '	**local 是一常数，表示游标是本地游标。**
cursor_name	**cursor_name 是使用标识符规则的游标名称。**
' global '	**global 是一常数，表示游标是全局游标。**
' variable '	**variable 是一常数，表示游标源是一局部变量。**
cursor_variable	**cursor 是用 CURSOR 数据类型定义的游标变量名。**

cursors

使用游标时，最好一次处理结果集的一行。现在，可以用游标数据类型将游标赋给变量或参数，下面的语句可与游标一起使用：DECLARE CURSOR，CLOSZ，DECLARE @local_variable，DEALLOCATE，OPEN，UPDATE，SET，DELETE，FETCH，

CREATE PROCEDURE。下面的全局变量可与游标一起使用：@@cursor_rows，@@fetch_status，CURSOR_STATUS，下列系统存储过程可与游标一起使用：sp_describe_cursor、sp_describe_cursor_columns、sp_describe_cursor_tables、sp_cursor_list。

DATABASEPROPERTY

当需要提供数据库和属性名称时，DATABASEPROPERTY 可返回指定数据库的属性值，该值为 1=True，0=False，NULL(如果输入出错)。

语法

DATABASEPROPERTY('database', 'property')

变量

'database'	database 是指想获得信息的数据库名称。
'property'	property 是返回信息的数据库属性名称，这些属性包括： IsAnsiNullDefault = SQL-92 无效性能 IsAnsiNullsEnabled = 将空值赋给未知数，进行比较 IsAnsiWarningsEnabled = 出现错误时给出警告信息 IsAutoClose = 最后一个用户退出系统后关闭数据库 IsAutoShink = 自动压缩数据库文件

IsBulkCopy=因某些操作登录没发生
IsCloseCursorsOnCommitEnabled = 当提交事务时关闭游标
IsDboOnly = 只有数据库拥有者能访问数据库
IsDetached =用 sp_detach_db 分离数据库
IsEmergencyMode = 应急模式允许使用可疑的数据库
IsFulltextEnabled =允许使用全文数据库
IsInLoad = 恢复数据库
IsInRecorery = 正在恢复数据库
IsInStandBy = 只读数据库，附带恢复记录
IsLocalCursorsDefault =Default 表示声明游标为 LOCAL
IsNotRecovered = 数据库没有恢复
IsNullConcat =NULL 并置数提供 NULL 结果集
IsOffline = 脱机数据库
IsQuotedIdentifiersEnabled = 有效标识符可以使用双引号
IsReadOnly = 只读数据库
IsRecursiveTriggersEnabled = 循环触发器
IsShutDown =数据库的启动问题
IsSingleUser = 只有一个用户可以使用数据库
IsSuspect = 可疑的数据库
IsTrunclog = 检查点截断数据库的事务处理记录
Version =已关闭的数据库的版本号或 NULL

示例

下面是 DATABASEPROPERTY 函数的示例：

```
USE master
SELECT DATABASEPROPERTY('master','IsTrunclog')
```

本例中，返回值为 master 系统数据库中的 IsTrunclog 属性设置，返回值为 1 表示真，0 为假，NULL 表示没找到值。

数据类型

数据类型决定列、存储过程参数和局部变量的数据特征，分为系统提供的数据类型和用户定义的数据类型，系统提供的数据类型在下面几节中列出。

(一) 二进制数据(Binary Data)

二进制数据的数据类型有：

binary	**binary 是固定长度的数据类型，最多可以有 8000 个字节。**
varbinary	**varbinary 是可变长度的数据类型，最多可以有 8000 个字节。**

(二) 字符数据(Character Data)

字符数据的数据类型有：

char	char 指固定长度的数据类型，最多可以有 8000 个 ANST 字符。
varchar	varchar 是可变长度的数据类型，最多可以有 8000 个 ANSI 字符。
nchar	nchar 指固定长度的数据类型，最多可以有 4000 个 Unicode(统一长度)的字符。Unicode 字符表示每字符 2 个字节。UNICODE 包括所有国际通用字符。
nvarchar	nvarchar 是可变长度的数据类型，最多可以有 4000 个 Unicode(统一长度)的字符。Unicode 字符表示每字符 2 个字节。UNICODE 包括所有国际通用字符。

(三) 日期和时间数据(Date and Time Data)

日期和时间数据的类型：

datetime	datetime 数据类型的可能值是从 1753 年 1 月 1 日开始，到 9999 年 12 月 31

	日。
smalldatetime	smalldatetime 数据类型没有 datetime 数据类型精确(datetime 精确至 1／300 秒，smalldatetime 精确至 1 分钟)，它的可能值从 1900 年 1 月 1 日开始，到 2079 年 6 月 6 日。datetime 数据占 8 个字节，smalldatetime 数据占 4 个字节。

(四) 精确数字数据(Exact Numeric Data)

精确值数据类型有：

decimal	decimal 是一种精确值数据类型，其可能值从 $-10^{38}-1$ 开始，到 $10^{38}-1$。
numeric	和 decimal 数据类型相同。
int	int 是只保存整数的数据类型，此整数值从 $-2^{31}-1$(-2，147，483，648)到 $2^{31}-1$(2，147，483，647)。
smallint	smallint 也是一种只存整数的数据类型，此整数值从 -2^{15}(-32，768)开

始到 2^{15}-1(32，667)。

tinyint	tinyint 是一种只存整数的数据类型，此数值从 0 开始至 255。

(五) 近似值数字数据类型(Appxoximate Numeric Data)

近似值的数据类型有：

float	float 是浮点数数据类型。缺省状态下为 15 位精度，正数范围从 2.23E-308 开始至 1.79E＋308，负数范围从 -2.23E-308 开始至 -1.79E＋308。
real	此数据类型可以存储类似于 float 的正负浮点数，实数据类型有 7 位精度，正值范围是 1.18E-38 至 3.40E-38，可以存储 0，负数范围是 -1.18E-38 至 -3.40E＋38。

(六) 货币数据(Monetary Data)

财政货币数据的数据类型有：

money	money 是表示财政货币的数据类型，

smallmoney	其值为-922，337，203，685，477.5707到＋922，337，203，685，477.5807，精确到千分之十货币单位。 smallmoney数据类型可存储的货币值范围是-214，748.3648至＋214，748.3647，精确到为千分之十货币单位。smallmoney值显示时可圆整至2位小数。

(七) 特殊数据(Special data)

特殊数据类型有1位，其值为1或0。SQL Server认为大于0的整数都是1。

(八) 游标(Cursor)

游标数据类型和游标变量一起使用，且只用于游标环境。

(九) 系统名称(Sysname)

Sysname是SQL Server数据类型，它由SQL Server定义为nvarchar(128)型数据，128个Unicode字符,256个字节。SQL Server需要介于系统表格和存储过程之间的共用数据类型，SQL Server将系统名称用作共用数据类型。

(十) 时间戳(timestamp)

时间戳是一种数据类型，每当插入行或更新行时，SQL Server 都会更新时间戳。每个表格只限一个时间戳，且是二进制(8)或可变二进制(8)型数据。

(十一) 唯一标识符(uniqueidentifier)

唯一标识符是 16 个字节的全局唯一全局标识符，也称 GUID。

(十二) 用户自定义的数据类型(user-defined)

user-defined 是建立在 SQL Server 数据类型的基础上，用 sp_addtype 系统存储过程定义的数据类型。

(十三) 文本和图像数据(Text and Image Data)

文字和图像数据类型有：

text	text 是一种数据类型，最多可以保存 2^{31}-1 或 2，147，483，647 个字符，缺省长度为 16，该缺省长度可以改变。
ntext	ntext 为一种数据类型，最多可以保存 2^{30}-1 或 1,073,741,823 个字符，

	其长度可变。
Image	Image 是一种数据类型，最多可以保存 2 31-1 或 2，147，483，647 个字节的二进制数据，其长度可变。

(十四) 统一代码数据(Unicode Data)

Nchar(n前缀是 SQL-92统一代码标准)	Nchar 是统一代码数据，最多可以保存 4000 个固定长度的字符。
nvarchar	nvarchar 是统一代码数据，最多可以保存 4000 个可变长度的字符。
Ntext	Ntext 是统一代码数据，最多可以保存 1，073，741，823 个可变长度的字符。

DATALENGTH

DATALENGTH 是系统函数，它返回任意数据类型表格式的长度。

语法

DATALENGTH(expression)

变量 expression 可以是常数、列名、函数与子查询程序，算术运算符，按位运算符和串运算符。

示例

下面是 DATALENGTH 系统函数的示例：

```
SELECT DATALENGTH(Column1)
FROM MyTable
GO
```

本例中，返回的整数代表 Column1 的长度（以字节为单位），将 DATALENGTH 用于可变字符、可变二进制数、文本、图像、nvarchar 和 ntext 数据类型，可以查看它们的长度。如果该列为空，则返回 NULL。数字数据类型的存储在本版本中有所不同，DATALENGTH 对数值表达式返回 5，9，13 或 17。

DATEADD

DATEADD 返回根据日期时间添加的日期，其单位可以是在日期单位列表中列出的任一单位。

语法

```
DATEADD(datepart, number, date)
```

日期函数

下表适用于提取 datepart 参数的所有日期函数。

Date part(日期单位)	Abbreviation(缩写)	Values(值)
Year	Yy	1753-9999
Quarter	Qq	1-4
Month	Mm	1-12
Day-of-year	Dy	1-366
Day	dd1	-31
Week	Wk	1-53
Weekday	Dw	1-7(Sun.-Sat.)
Hour	Hh	0-23
Minute	Mi	0-59
Second	Ss	0-59
Millisecond	Ms	0-999

变量

datepart	datepart 是用于计算的日期单位。参阅日期函数下面的表格。
number	number 是指添加至日期变量中的日期单位数。
date	date 是 GETDATE 函数、格式化为日期的字符串、或日期时间型的列名、变量或参数。

示例

下面是 DATEADD 函数的示例：

```
SELECT DATEADD (day,7,DateColumn1)
FROM MyTable
GO
```

本例中，将 7 天添加至 DateColumn1 的值中。

DATEDIFF

DATEDIFF 返回两个指定日期的日期单位的差别。

语法

DATEDIFF (datepart， date,date)

变量

datepart	**datepart 是用于计算的日期单位，参阅日期函数下面的表格。**
date	**date 是 GETDATE 函数；格式化为日期的字符串、或日期时间型的列名、变量或参数。**

示例

下面为 DATEDIFF 函数的示例：

```
SELECT DATEDIFF(day,DateColumn1,getdate())
FROM Mytable
```

本例中，返回值为 DateColumn1 中的值和当前日期相差的天数。

DATENAME

DATENAME 返回日期单位的名称。

语法

```
DATENAME(    datepart, date    )
```

变量

datepart	**datepart 是指用于计算的日期单位，参阅日期函数下面的表格。**
Date	**date 日期指 GETDATE 函数，格式化为日期的字符串、或日期时间型的列名。**

示例

下面是 DATENAME 函数的示例：

```
SELECT DATENAME(dw,getdate())
```

本例中，如果今天是星期一，就返回单词“Monday”的字符数据。

DATEPART

DATEPART返回的整数表示日期中的日期单位。

语法

```
DATEPART(    datepart,date    )
```

变量

datepart	datepart是指用于计算的日期单位，参阅日期函数下面的表格。
date	date日期指GETDATE函数，格式化为日期的字符串、或日期时间型的列名。

示例

下面是DATEPART函数的示例：

```
SELECT DATEPART(month getdate())
GO
```

本例中，返回值为一年 12 个月中的月份。

DAY

DAY 显示所提供的日期的整数，即天数部分。

语法

```
DAY(    date    )
```

变量 date 是 samlldatetime 或 datetime 数据类型。

DB_ID

DB_ID 为系统函数，它返回数据库的标识号。

语法

```
DB_ID( [ ' database_name ' ] )
```

变量 database_name 是用户希望得到的数据库标识号的数据库名称，缺省值为当前使用的数据库。

示例

下面是 master 数据库中运行的 DB_ID 函数的示例：

```
SELECT DB_1D('master')
```

本例中，返回的 1 是 master 系统数据库的数据库标识号。

DB_NAME

传递数据库标识号时，DB_NAME 返回数据库名称。

语法

```
DB_NAME( [    database_id    ] )
```

变量 database_id 是要查看的标识号，缺省值为当前使用的数据库。

示例：

下面是 DB_NAME 系统函数的示例：

```
SELECT DB_NAME (1)
```

本例中，返回数据库标识号为 1 的数据库。数据库名称是 master。

DBCC CHECKALLOC

Database Consistency Checker 语句 DBCC CHECKALLOC 检查数据库中所有页的分配使用情况。

语法

```
DBCC CHECKALLOC
(   ' database_name'
 [, NOINDEX |
{ REPAIR_ALLOW_DATA_LOSS
 | REPAIR_FAST
 | REPAIR_REBUILD
}
]
)
 [WITH {All_ERRORMSGS | NO_INFOMSGS}]
```

变量

' database_name '	**database_name 是语句所作用的数据库名称，在当前数据库中执行此命令时不需要**

	database_name。
NOINDEX	NOINDEX 指用户定义的表格的非成簇索引不包含在数据库的检查之中。
REPAIR_ALLOW_DATA_LOSS	REPAIR_ALLOW_DATA_LOSS修理 REPAIR _REBUILD所修复的内容；它也修复分配错误和结构行或页面错误，并删除可能导致丢失数据的坏文本对象。
REPAIR_FAST	REPAIR_FAST 进行少量而快速的修复操作，例如修复少量的索引。它不会丢失数据。
REPAIR_REBUILD	REPAIR_REBUILD 进行 REPAIR_FAST 的修复工作，更多的时间是在修理索引，它不会丢失数据。
WITH NO_INFOMSGS | ALL_ERRORMSGS	WITH NO_INFOMSGS 指报告不包括信息性消息和占用的空间，只报告错误信息。WITH ALL_ERRORMSGS指显示所有的错误信息。

DBCC CHECKCATALOG

DBCC CHECKCATALOG检查系统表的完整性。

语法

```
DBCC CHECKCATALOG [( 'database_name' )]
```

[WITH NO_INFOMSGS]

变量

'database_name'	database_name 是语句所作用的数据库名称，在当前数据库中执行此命令时不需要 database_name。
WITH NO_INFOMSGS	WITH NO_INFOMSGS 表示报告不包括信息性消息和占用的空间，只报告错误信息。

DBCC CHECKDB

DBCC CHECKDB 检查数据库中所有内容的完整性。如果执行了 DBCC CHECKDB，就不需要执行 DBCC CHECKALLOC 或 CHECKTABLE。

语法

```
DBCC CHECKDB
('database_name'
 [,NOINDEX|
{ REPAIR_ALLOW_DATA_LOSS
 | REPAIR_FAST
 | REPAIR_REBUILD
}
```

]
)
[WITH {ALL_ERRORMSGS | NO_INFOMSGS}]

变量

'database_name'	database_name 是语句所作用的数据库名称，在当前数据库中执行此命令时不需要 database_name。
NOINDEX	NOINDEX 指用户定义的表格的非成簇索引不包含在数据库的检查之中。
REPAIR_ALLOW_DATA_LOSS	REPAIR_ALLOW_DATA_LOSS 修理 REPAIR_REBUILD 所修复的内容；它也修复分配错误和结构行或页面错误，并删除可能导致丢失数据的坏文本对象。
REPAIR_FAST	REPAIR_FAST 进行少量而快速的修复操作，例如修复少量的索引。它不会丢失数据。
REPAIR_REBUILD	REPAIR_REBUILD 进行 REPAIR_FAST 的修复工作，更多的时间是在修理索引，它不会丢失数据。
WITH NO_INFOMSGS | ALL_ERRORMSGS	WITH NO_INFOMSGS 指报告不包括信息性消息和占用的空间，只报告错误信息。WITH ALL_ERRORMSGS 指显示所有的错误信息。

DBCC CHECKFILEGROUP

检查文件组中所有表格的分配情况。

语法

```
DBCC CHECKFILEGROUP
 [( [{ '     filegroup_name' | filegroup_id   }] [,NOINDEX] )]
 [WITH NO_INFOMSGS | ALL_ERRORMSGS]
```

变量

'filegroup_name'	**filegroup_name 指 DBCC 语句作用的文件组名称。**
filegroup_id	**filegroup_id 是 sysfilegroups 系统表中的文件组的 ID。**
NOINDEX	**NOINDEX 指用户定义的非成簇索引不包括在数据库的检查之中。**
WITH NO_INFOMSGS | ALL_ERRORMSGS	**WITH NO_INFOMSGS 指报告不包括信息性消息和占用的空间，只报告错误信息。WITH ALL_ERRORMSGS 显示所有的错误信息。**

DBCC CHECKIDENT

DBCC CHECKIDENT 检查表格的当前等同值，并校正等同值。

语法

```
DBCC CHECKIDENT ('    table_name    '
 [, {NORESEED | {RESEED [,    new_reseed_value    ]}}])
```

变量

' table_name '	**table_name 是指 DBCC 语句作用的表格名称。**
NORESEED	**NORESEED 通知 SQL Server 不校正等同值。**
RESEED	**如果没有提供 new_reseed_value，RESEED 通知 SQL Server 将等同值校正为等同列中的最大值。**
new_reseed_value	**new_reseed_value 指等同列中新的起始数值。**

DBCC CHECKTABLE

DBCC CHECKTABLE 检查并有选择地修补表格中所有页面的完整性。

语法

```
DBCC CHECKTABLE
('    table_name    '
 [,NOINDEX |    index_id    )|
 {REPAIR_ALLOW_DATA_LOSS
  |  REPAIR_FAST
  |  REPAIR_REBUILD
}
]
)
 [WITH NO__INFOMSGS]
```

变量

'table_name'	**table_name 指 DBCC 语句作用的表格名称。**
NOINDEX	**NOINDEX 指用户定义的表格的非成簇索引不包含在数据库的检查之中。**
index_id	**index_id 指 DBCC 语句作用的索引的 ID。**
REPAIR_ALLOW_DATA_LOSS	**REPAIR_ALLOW_DATA_LOSS 修理 REPAIR_REBUILD 所修复的内容；它也修复分配错误和结构行或页面错误，并删除可能导致丢失数据的坏文本对象。**
REPAIR_FAST	**REPAIR_FAST 进行少量而快速的修复操作，例如修复少量的索引。它不会丢失数据。**
REPAIR_REBUILD	**REPAIR_REBUILD 进行 REPAIR_FAST 的修复工**

	作，更多的时间是在修理索引，它不会丢失数据。
WITH NO_INFOMSGS | ALL_ERRORMSGS	WITH NO_INFOMSGS 意味着报表不含通知消息，使用空格。只报告错误消息。WITH ALL_ERRORMSGS 显示所有错误消息。
WITH NO_INFOMSGS	WITH NO_INFOMSGS 指报告不包括信息性消息和占用的空间，只报告错误信息。

DBCC DBREPAIR

DBCC DBREPAIR 只向后兼容，现在使用 DROP DATABASE。

语法

DBCC DBREPAIR (database_name ,DROPDB [,NOINIT])

变量

database_name	database_name 是要删除的数据库名。
DROPDB	DROPDB 指撤消数据库。
NOINIT	NOINIT 指不改变数据库的分配页。

DBCC DBREINDEX

DBCC DBREINDEX 重建表格的索引，并可在不必撤消再重建索引的情况下，保存相关的约束，以减少撤消和重建约束的辅助操作。

语法

```
DBCC DBREINDEX ( [ ' database.owner.table_name' [,index_name
 [,fillfactor]]] )
 [WITH NO_INFOMSGS]
```

变量

'database.owner.table_name'	**database.owner.table_name 是重建索引的表格的全限定名称。**
index_name	**index_name 是要重建的索引名称，使用两个单引号可以通知 SQL Server 重建表格上的全部索引，如果需要指定填充因子，就只需使用两个单引号；否则可以省略最后 2 个变量。**
fillfactor	**fillfactor 表示每个索引叶页面上要存储数据的空间百分比，只有建立了索引**

	才能使用它。如果为 0，则使用初始填充因子。
WITH NO_INFOMSGS	WITH NO_INFOMSGS 表示报告不包括信息性消息和占用的空间，只报告错误信息。

DBCC dllname(FREE)

DBCC dllname(FREE)将扩展存储过程中的动态连接库(DLL)从内存中提取出来。

语法

```
DBCC    dllname    (FREE)
```

变量 dllname 是要从内存中删除的 dll 名称。

DBCC HELP

DBCC HELP 显示提供的 DBCC 语句的语法信息。

语法

```
DBCC HELP  ('dbcc_statement'|@dbcc_statement_var| '?')
```

变量

dbcc_statement| @dbcc_statement_var	DBCC 语句为 DBCC 语句或 DBCC 后面部分的名称(例如，CHECKTABLE)
?	?显示带帮助信息的 DBCC 语句。

DBCC INPUTBUFFER

DBCC INPUTBUFFER 显示从客户机发送至 SQL Server 的最后一个 Transact-SQL 语句。

语法

```
DBCC    INPUTBUFFER(spid)
```

变量 spid 是用于用户连接的系统进程 ID。(通过执行系统存储过程 sp_who 可以选择 spid)

DBCC MEMUSAGE

DBCC MEMUSAGE 显示 8K 页中关于内存使用情况的缓冲器和程序高速缓存的信息。

语法

DBCC MEMUSAGE([' BUFFER '] [,] [' PROCEDURE '])

变量

' BUFFER '	**BUFFER 指显示缓冲器高速缓存信息。**
' PROCEDURE '	**PROCEDURE 指显示程序高速缓存信息。**

DBCC NEWALLOC

DBCC NEWALLOC 检查数据库的每个表格中数据与索引页面的分配；然而，与 DBCC CHECKALLOC 相同，DBCC NEWALLOC 只是为了向后兼容。可以用 DBCC CHECKALLOC 代替它。

语法

DBCC NEWALLOC [(' database_name ' [,NOINDEX])]
[WITH NO_INFOMSGS]

变量

' database_name '	**database_name 是语句所作用的数据库名称，在当前数据库中执行此命令时不需要 database_name。**

NOINDEX	NOINDEX 指用户定义的表格的非成簇索引不包含在数据库的检查之中。
WITH NO_INFOMSGS	WITH NO_INFOMSGS 指报告不包括信息性消息和占用的空间，只报告错误信息。

DBCC OPENTRAN

DBCC OPENTRAN 显示数据库中最旧的活动信息以及分布式和非分布式复制事务处理信息。打开的事务处理可以由 spid 标识并可终止。

语法

```
DBCC OPENTRAN  [('database_name' | database_id)]
 [WITH TABLERESULTS [,NO_INFOMSGS]]
```

变量

'database_name'	database_name 是 DBCC 语句作用的数据库名称。
database_id	database_id 是数据库标识号。
WITH TABLERESULTS	WITH TABLERESULTS 将结果以能放进表格的格式存放。
WITH NO_INFOMSGS	WITH NO_INFOMSGS 指报告不包括信息性消息和占用的空间，只报告错误信息。

DBCC OUTPUTBUFFER

DBCC OUTPUTBUFFER 显示 spid 中输出缓冲区的内容。

语法

```
DBCC OUTPUTBUFFER(  spid  )
```

变量 spid 是用于用户连接的系统过程 ID。

DBCC PERFMON

DBCC PERFMON 显示三种 SQL Server 性能信息，即 IOSTATS，LRUSTATS 和 NETSTATS。

语法

```
DBCC PERFMON
```

DBCC PINTABLE

DBCC PINTABLE为表格定位，即表格页面在内存中保存和修改，当SQL Server需要空间读取新页面上的数据时，不刷新表格。对于常用的小表格，最好限制使用DBCC PINTABLE。

语法

DBCC PINTABLE(database_id,table_id)

变量

database_id	**database_id是定位表格的数据库标识号。**
table_id	**table_id是要定位的表格的标识符。**

DBCC PROCCACHE

DBCC PROCCACHE可由Microsoft SQL Server Performance Monitor用来监控程序高速缓存。

语法

DBCC PROCCACHE

DBCC ROWLOCK

在 Microsoft SQL Server 6.5 中，DBCC ROWLOCK 激活表格中的 Insert Row Locking(IRL)，并且支持它只为向后兼容。在版本 7.0 中，使用 DBCC ROWLOCK 不进行任何操作。Microsoft SQL Server 7.0 有动态行级锁定。

DBCC SHOWCONTIG

DBCC SHOWCONTIG 显示表格中数据和索引的存储碎片，不使用 SORTED_DATA 选项就撤消并重建成簇索引可以减少存储碎片。

语法

DBCC SHOWCONTIG(table_id [,index_id])

变量

table_id	table_id 指 DBCC 语句作用的表格的 OBJECT_ID，如果没有提供 table_id，DBCC 语句将作用于所有的表格。
index_id	index_id 是 DBCC 语句作用的索引 indid，如果不提供 index_id，DBCC 语句将作用于表格中所有的索引。

DBCC SHOW_STATISTICS

DBCC SHOW_STATISTICS 显示表格中索引或列的分布式页面统计信息。分布密度越高，选择性就越低(因而索引的作用就越小)。如果想知道统计信息最后一次更新的时间，可使用系统函数 STATS_DATE。

语法

DBCC SHOW_STATISTICS(table,target)

变量

table	**table 是要查看统计信息的表格名称。**
target	**target 是索引名称或列名称。**

DBCC SHRINKDATABASE

DBCC SHRINKDATABASE 压缩数据库中的数据文件。

语法

DBCC SHRINKDATABASE
(database_name [,target_percent]

[, {NOTRUNCATE | TRUNCATEONLY}]
)

变量

database_name	database_name 指要压缩的数据库名称。
target_percent	target_percent 指压缩后要保留在数据库文件中的自由空间的百分比。
NOTRUNCATE	NOTRUNCATE 表示自由文件空间应保留在数据文件中，缺省是将空间释放到操作系统中。TRUNCATEONLYTRUNCATEONLY 指将数据文件中没有使用的空间释放到操作系统中。不必将行移到未分配的页中，并忽略 Target_percent。

DBCC SHRINKFILE

DBCC SHRINKFILE 压缩数据库的数据文件。

语法

DBCC SHRINKFILE
({file_name | file_id} [,target_size]
[, {EMPTYFILE | NOTRUNCATE | TRUNCATEONLY}]
)

变量

file_name	file_name 是要压缩的数据库文件的逻辑名称。
file_id	file_id 是指 sysfiles 系统表中要压缩的数据库文件的 FILE_ID。
target_size	target_size 是指压缩后数据文件的大小，以兆字节为单位。如果不给出 target_size，将尽量压缩文件。
EMPTYFILE	EMPTYFILE 将数据从一个文件移至同一文件组中的另一个文件中，然后，就可用 ALTER DATABASE 语句将文件删除。
NOTRUNCATE	NOTRUNCATE 表示自由的文件空间应保留在数据库文件中，缺省是将空间释放到操作系统中。
TRUNCATEONTY	TRUNCATEONTY 指将数据库文件中未使用的空间都释放到操作系统中。不将行移至未分配的页面上，并忽略 target_size。

DBCC TEXTALL

DBCC TEXTALL 在带有文本、ntext 或图像列的表格中执行 TEXTALLOC。Microsoft Server 7.0 包括此函数，只是为了向后兼容。可以用 DBCC CHECKDB

代替 DBCC TEXTALL。

语法

```
DBCC TEXTALL ( [ 'database_name' | database_id ] [,'Full' | 'FAST' ]
)
```

DBCC SQLPERF

DBCC SQLPERF 显示所有数据库中使用事务处理记录空间的统计信息。

语法

```
DBCC SQLPERF(LOGSPACE)
```

DBCC TEXTALL

DBCC TEXTALL 在数据库中带有文本、ntext 或图像列的表格中运行 DBCC TEXTALLOC。此函数包含在 Microsoft Server 7.0 中，只是为了向后兼容。可以用 DBCC CHECKTABLE 代替 DBCC TEXTALL。

语法

DBCC TEXTALL

DBCC TEXTALLOC

DBCC TEXTALLOC 检查有文本、next 或图像列的表。在 Microsoft SQL Server 7.0 中包含它是为了向后兼容。利用 DBCC CHECKTABLE 代替它。

语法

DBCC TEXTALLOC ({ table_name | table_id } [,' FULL' | ' FAST'])

DBCC TRACEOFF

DBCC TRACEOFF 从作用的 SQL Server 中删去跟踪标记。

语法

DBCC TRACEOFF (trace# [,… trace#])

变量

trace# **trace#是追踪标志号。**

DBCC TRACEON

DBCC TRACEON 可打开作用于 SQL Server 的跟踪标记。

语法

DBCC TRACEON (trace# [,…trace#])

变量

trace# **trace#是追踪标志号。**

DBCC TRACESTATUS

DBCC TRACESTATUS 显示跟踪标记的状态。如果打开跟踪标记，SQL Server 就返回 1，关闭追踪标记，就返回 0。使用值为 -1 的 trace#查看参阅作用于服务器的所有跟踪标记。

语法

DBCC TRACESTATUS (trace# [，…trace#])

变量

trace#
trace#是追踪标志号。

DBCC UNPINTABLE

DBCC UNPINTABLE 可从缓冲区高速缓存中取消表格页。
语法

DBCC UNPINTABLE (database_id, table_id)

变量

database_id	**database_id 是包含要取消的表格的数据库标识号。**
table_id	**table_id 是要取消的表格的 OBJECT_ID。**

DBCC UPDATEUSAGE

DBCC UPDATEUSAGE 修改 sysindexes 系统表并删除任何可能导致在 sp_spaceused 系统存储程序中出现不准确报表的旧统计资料。

语法

```
DBCC UPDATEUSAGE ( { 'database_name' | 0} [, 'table_name'
[,index_id ]] )
[WITH [NO_INFOMSGS] [[,] COUNT_ROWS]
```

变量

'database_name' |0	database_name 是 DBCC 语句作用的数据库，如果此变量为 0，DBCC 语句就作用于当前数据库。
'table_name'	table_name 是 DBCC 语句作用的表格。
index_id	index_id 是 DBCC 语句作用的索引 indid，缺省为表格中的所有索引。
WITH NO_INFOMSGS	WITH NO_INFOMSGS 指报表不包括信息性消息和占用的空间，只报告错误信息。
WITH COUNT_ROWS	WITH COUNT_ROWS 用表格中的当前行数更新 sysindexes 系统表中的行列，如果表格较大，需要的时间将较长。

示例

下面是 DBCC UPDATEUSAGE 语句的示例。

```
DBCC UPDATEUSAGE(0)
```

GO

本例中，对当前数据库中所有表格与所有索引，sysindexes 表都进行了更新。

DBCC USEROPTIONS

DBCC USEROPTIONS 显示用于当前连接的活动 SET 选项。

语法

DBCC USEROPTIONS

DEALLOCATE

DEALLOCATE 是一个语句，用于通过名称或变量名删除对游标的引用。删除最后一次对游标的引用后，数据结构和锁定就恢复自由。

语法

DEALLOCATE {{[GLOBAL] cursor_name} | cursor_variable_name}

变量

cursor_name	已声明了 cursor_name，它是一个局部变量，除非指定为 GLOBAL，或者不存在使用该名称的局部游标。
cursor_variable_name	cursor_variable_name 是游标变量名。

十进制数据类型

十进制数据类型是精确数值，精确到数的最小单位，另一种精确数值数据类型为数值。两者的值域都是从 $10^{38}-1$ 到 $-10^{38}-1$。

decimal [(p [,s])] and numeric [(p [,s])]

变量

precision	precision（p）指小数点两边存储的小数位数的最大值。
scale	scale 指小数点右边小数位数的最大值，其值为从 0 到任意精度变量的值。

DECLARE @local_variable

变量可用 DECLARE 语句在批注、存储过程或触发器中声明。可用 SET 或 SELECT 语句给变量赋值，所有声明的局部变量都初始化为 NULL。

语法

```
DECLARE
 {
 {@local_variable data_type}
 |  {cursor_variable_name CURSOR}
 } [,...n]
```

变量

@local_variable	@符号放在局部变量名的开头是 SQL Server 识别局部变量的方式。Transact SQL 语句中的变量不能用作数据库对象，只能用作常量。如果不存在使用该名称的局部变量，在程序、触发器或批注中变量就是局部变量。
data_type	data_type 是系统或用户定义的数据类型，但不是图像、文字或 ntext 数据类型。
cursor_variable_name	cursor_variable_name 是游标变量的名称，其规

	范与局部变量相同，但不以@开始。
CURSOR	CURSOR 表示变量为局部游标变量。
N	N 指前面的句法可以重复。
SET|SELECT	SET|SELECT 将表达式中的值赋给变量。
Expression	表达式可以是列名、常量、函数、变量、子查询程序，多个常量和函数，算术运算符、按位运算符和字符串运算符。

DECLARE CURSOR

DECLARE CURSOR 声明 Transaet-SQL 游标的属性。

语法

```
DECLARE cursor_name [INSENSITIVE] [SCROLL] CURSOR
FOR select_statement
 [FOR {READ ONLY | UPDATE [OF column_list ] }]
```

或

```
DECLARE cursor_name CURSOR
 [LOCAL | GLOBAL]
 [FORWARD_ONLY | SCROLL]
```

[STATIC |KEYSET | DYNAMIC]
[READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
FOR select_statement
[FOR { READ ONLY |UPDATE [OF column_list] }]

变量

cursor_name	用标识符规则声明的游标名称。
INSENSITIVE	INSENSITIVE 声明一游标，该游标复制数据，当提取语句从游标中读取行时，不显示对基行的修改。游标不能更新。
SCROLL	SCROLL 表示可以用第一个、最后一个、上一个、下一个、相关和绝对提取命令滚动游标。
select_statement	select_statement 是个 SELECT 语句，它决定游标的结果集。不允许使用 COMPUTE，COMPUTE BY，FOR BROWSE 和 INTO。
READ_ONLY	READ_ONLY 指不允许更新的只读游标。
UPDATE [OFcolumn_list]	UPDATE [OFcolumn_list] 决定游标可更新的列，如果没有清单，所有的列都可以更新。
LOCAL	LOCAL 表示游标的作用域对于它在其中声明的批注、存储过程或触发器而言是局部的。
GLOBAL	GLOBAL 表示游标的作用域对于连接是全局的。
FORWARD ONLY	FORWARD ONLY 指游标能向前滚，从第一行滚动到最

	后一行。
STATIC	STATIC 声明一游标，该游标复制数据，当提取语句从游标中读取行时，不显示对基行的修改。游标不能更新。
KEYSET	KEYSET 指打开游标时，设置行的成员和顺序。当拥有者滚动游标时，可以看到表格的变化。看不到其他用户进行的插入；可是，如果删去一行，全局变量 @@fetch_status 为－2。更新就相当于删除和插入。
DYNAMIC	当滚动游标时，DYNAMIC 游标可以了解其结果集中对行所作的所有数据改动。每次读取时不使用绝对和相对的读取选项，也可以修改行。
SCROLL_LOCKS	SCROLL_LOCKS 指通过游标进行的更新和删除肯定成功。SQL Server 锁定行。
OPTIMISTIC	OPTIMISTIC 表示如果行读入游标后进行过更新，通过游标进行的修改就不会成功。

DEGREES

DEGREES 为所提供的角度的弧度值给出了一个数值表达式（以度数表示）。

语法

DEGREES(numeric_expression)

变量 numeric_expression 是一表达式，它可以使用小数、浮点数、整型数、微短整数、货币、数值、实数、短整型数或小货币数据类型。

DELETE

DELETE 语句从表格中删去行。

语法

```
DELETE [FROM { table_name | view_name }
 [ <from_clause>:: =
FROM
{<table_or_view.
|( select_statement ) [AS] alias
| <table_or_view> CROSS JOIN <table_or_view>
| <table_or_view>
 [{INNER | {LEFT | RIGHT | FULL} [OUTER] }]
 [<join_hint>] JOIN
<table_or_view> ON search_conditions
} [,...n]
]
```

```
 [WHERE {    search_conditions   }|
{CURRENT OF { {  [GLOBAL ]     cursor_name} | cursor_variable_name
  }}]
 [OPTION (<query_hint>  [,...n] )]

<table_or_view>::=
{    table     [[AS]     alias   ]   [<table_hint>]
|    view    [[AS]    alias   ] }
|     OpenRowset( ' provider_name ' , { ' datasource ' ; ' user_id ' ; '
password ' |
   ' provider_string ' },{ [catalog.] [schema.] object_name |  ' query ' })
|OpenQuery(   linked_server,' query '   )}

<table_hint>::=
( [INDEX = {   index_name | index_id} [,...n]   ]   [FASTFIRSTROW]
 [HOLDLOCK  [NOLOCK]   [PAGLOCK]   [READCOMMITTED]
 [READPAST]   [READUNCOMMITTED]   [REPEATABLEREAD]
 [ROWLOCK]   [SERIALIZABLE]   [TABLOCK]   [TABLOCKX]
 [UPDLOCK] )
<join_hint>::=
{LOOP | HASH | MERGE}
```

```
<query_hint>::=
{{HASH | ORDER} GROUP
|{MERGE | HASH | CONCAT} UNION
|FAST     n
| FORCE ORDER
| ROBUST PLAN
}
```

变量

FROM	FROM 指从中删去行的表格或视图。
table_name|view_name	table_name|view_name 是从中删去行的表格或视图
Alias	Alias 是表格或视图的别名。
CROSS JOIN	CROSS JOIN 是两表格的交叉积。
<join_lint>	<join_lint>通知查询优化程序在查询的 FROM 子句中，每个连接都要使用一个 join_hint。
LOOP｜HASH｜MERGE	LOOP｜HASH｜MERGE 决定查询中的连接使用循环、散列或合并。
JOIN	JOIN 表明在 DELETE 操作中使用了 SQL-92 联接。
ON search_conditions	ON search_conditions 是连接的条件。
N	N 指前面的项可以重复。

WHERE	WHERE 决定筛选出要删去的行的条件。
Search_conditions	Search_conditions 是要删去的行的约束条件。
CURRENT OF	CURRENT OF 指从游标的当前位置上 DELETE。
GLOBAL	GLOBAL 表示游标是全局游标。
cursor_name	cursor_name 是要提取的打开的游标名称。
cursor_variable_name	cursor_variable_name 是游标变量的名称。
OpenRowset	OpenRowset 有要访问数据源中的远程数据的所有联接信息。
OpenQuery	OpenQuery 执行通过查询。
<table_hint>	<table_hint>是表格扫描，由优化程序或锁定方式采用的索引。
INDEX=index_name | index_id	要使用的 INDEX 名称。
OPTION(<query_hint>, [,…n])	OPTION 表示伴随优化程序提示。
{HASH | ORDER} GROUP	{HASH | ORDER} GROUP 指查询中 GROUP BY 或 COMPUTE 子句指定的集合要使用散列或排序。
{MERGE | HASH | CONCAT} UNION	{MERGE | HASH | CONCAT} UNION 指 UNION 操作需通过合并、散聚或联接 UNION 集来执行。
FAST	FAST 指查询程序可以快速检索前 n 行，并返回整个结果集。

N	N 指要检索的行号。
FORCE ORDER	FORCE ORDER 指查询优化期间查询语法的联结顺序相同。
ROBUST PLAN	ROBUST PLAN 表示查询计划争取行的最大可能容量。

DENY

DENY 是一个语句，它不仅用于否定安全帐户的许可，还阻止从其他组或角色成员中继承许可。在冲突的情况下，最严格的许可应用该语句，也可和 public 角色和 guest 用户一起使用 DENY。从安全帐户中删除否定许可，必须使用 REVOKE 语句。

语法

```
DENY {ALL | statement [,…n]}
TO security_account [, …n]
```

变量

ALL	ALL 指否定所有的许可。
statement	statement 指许可被否定的语句，DENY 可以否

	定用户的下列语句：CREATE DATABASE，CREATE DEFAULT，CREATE PROCEDURE，CREATE RULE，CREATE TABLE，CREATE VIEW，BACKUP DATABASE，BACKUP TRANSACTION 和 BACKUP TABLE。
TO	TO 指下个词为安全帐户。
security_account	安全帐户可以是 SQL Server 用户、SQL Server 角色、Windows NT 用户或 Windows NT 组。
PRIVILEGES	与 SQL-92 兼容的可选关键词。
permission	permission 指表格或视图的否定语句许可。许可列表如下：SELECT，INSERT，DELETE 和 UPDATE，还有表格的 REFERENCES。用于列的许可列表有：SELECT 和 UPDATE，用于存储过程的许可是 EXECUTE。
n	n 指前面的项可以重复。
column	column 指许可被否定的列名。
Table	Table 指许可被否定的表格。
view	view 指许可被否定的视图。
procedure	procedure 指许可被否定的存储过程。
extended_procedure	extended_procedure 指许可被否定的扩展存储过程名称。
CASCADE	CASCADE 表示否定安全帐户的许可，以及由该安全帐户授予的其他安全帐户的许可。如果不使

用 CASCADE，但授予用户 WITH GRANT OPTION 许可，就可能发生错误。

示例
下面是 DENY 语句的示例：

```
DENY INSERT，UPDATE，DELETE
ON MyTable。
TO SecAcct1,SecAcct2, SecAccts。
```

本例中，否定 MyTable 中安全帐户的许可。

DIFFERENCE

DIFFERENCE 通过 SOUNDEX 函数显示两个字符表达式间的差别。
语法

```
DIFFERENCE(character_expression, character_expression)
```

变量

character_expression 字符表达式可以是字符或可变字符数据类型，

也可以常量、变量或列。

示例
下面是 DIFFERENCE 函数的示例：

```
DIFFERENCE('marry','mary')
```

本例中，差别不可能是 4，而可能是 0。

DROP DATABASE

DROP DATABASE 实际删去数据库。
语法

```
DROP DATABASE    database_name [, …n]
```

变量

database_name	**database_name 指要删除的数据库。**
n	**n 指前面的项可以重复。**

示例

下面是 DROP DATABASE 语句的示例。

```
DROP DATABASE pubs,mydb
```

在本例中，pubs 和 mydb 数据库被删除。

DROP DEFAULT

DROP DEFAULT 是一个语句，它删除用户建立的 DEFAULT 对象，但不删除 DEFAULY CONSTRAINT。要删除 DEFAULT CONSTRAINT，必须使用 ALTER TABLE。

语法

```
DROP DEFAULT    default [, …n]
```

变量

default	default 指要删除的缺省值。
n	n 指前面的项可以重复。

示例

下面是 DROP DEFAULT 语句的示例：

```
DROP DEFAULT usrdefdflt
```

本例中，删去了用户定义的缺省值。

DROP INDEX

DROP INDEX 是一个语句，它删除 INDEX，但不删除 PRIMARY KEY 或 UNIQUE 约束建立的索引。

语法

```
DROP INDEX{   'table.index' } [, …n]
```

变量

table	table 指索引列的位置。
index	index 指要删除的索引。
n	n 指前面的项可以重复。

示例

下面是 DROP INDEX 语句的示例。

```
DROP INDEX 'MyTable.idx1'
```

本例中，删除 MyTable 中的索引 idx1。

DROP PROCEDURE

DROP PROCEDURE 是一个语句，它删除数据库中的存储过程或存储过程组。

语法

```
DROP PROCEDURE    procedure [, ...n]
```

变量

procedure	procedure 是要删除的存储过程。
n	n 指前面的项可以重复。

示例

下面是 DROP PROCEDURE 语句的示例：

```
DROP PROCEDURE MyProcedure
```

本例中，删去了一个存储过程。

DROP RULE

DROP RULE 是一个语句，它删去作为数据库对象的用户定义的规则。

语法

```
DROP RULE    rule [, …n]
```

变量

rule	rule 是要删去的用户定义的规则名。
n	n 指前面的项可以重复。

示例

下面是 DROP RULE 语句的示例。

```
DROP RULE MyRule
```

本例中，删除了一个规则。

DROP STATISTICS

DROP STATISTICS 可删去列的统计数字。

语法

DROP STATISTICS table.column [,...n]

变量

table	table 指表格的名称。
column	column 指列名。
n	n 指前面的项可以重复。

DROP TABLE

DROP TABLE 删去表格及其索引、触发器、约束和许可，但使用该表格的视图必须用 DROP VIEW 来删除。

语法

DROP TABLE table_name

变量 table_name 指要删去的表格。

DROP TRIGGER

DROP TRIGGER 可删除作为数据库对象的触发器。

语法

```
DROP TRIGGER    trigger_name [, …n]
```

变量 trigger_name 指从数据库中删去的触发器的名称。

DROP VIEW

DROP VIEW 删去视图，因为当删除表格时，视图不会自动撤消，所以才使用 DROP VIEW。

语法

```
DROP VIEW    view_name [,…n]
```

变量 view_name 是要作为数据库对象删去的视图。

DUMP

DUMP 只是为了向后兼容，可以用 BACKUP 来代替 DUMP。

ELSE(IF…ELSE)

ELSE(IF…ELSE)是条件语句集。如果条件返回真，则执行语句；如果 IF 条件为假，则执行 ELSE 语句。

语法

```
IF Boolean_expression
{sql_statement | statement_block}
 [ELSE
{sql_statement | statement_block}]
```

变量

Boolean_expression	Boolean_expression 返回真或假。
{sql_statement | statement_block}	Transact-SQL 语句或语句。

示例

下面是 IF(IF…ELSE)条件语句集的示例：

```
IF (SELECT count(Columnl)
FROM MyTable
WHERE Columnl = l ) > 0
BEGIN
SELECT Column2
  FROM MyTable
  END
ELSE
  BEGIN
  SELECT Column3
  FROM MyTable
  END
```

本例中，如果在 MyTable 中进行的查询选出的行数大于 0，就选择 Cloumn2；否则，选择 Column3。

END（BEGIN...END)

BEGIN…END 将一系列 Transact-SQL 语句括起来，像在 IF…ELSE 逻辑中一

样，使这些语句作为程序块来执行。

语法

```
BEGIN
{sql_statement | statement_block}
END
```

变量 sql_statement 或 statement_block 是 Transact-SQL 语句或一组语句。

示例

下面是 BEGIN…END 语句的示例：

```
IF (SELECT count(Column1)
FROM MyTable
WHERE Column1 = 1) > 0
BEGIN
SELECT Column2
FROM MyTable2
END
ELSE
BEGIN
SELECT Column3
FROM MyTable
```

```
END
```

本例中，如果在 MyTable 中进行的查询选出的行数大于 0，就选择 Cloum2；否则，选择 Column3。

EXECUTE

EXECUTE 执行存储过程或扩展存储过程，也可以执行 Transact-SQL 批处理中的字符串。

语法

```
[[EXEC[UTE]]
{
 [   @return_status     = ]
{   procedure_name  [;number]  |@procedure_name_var
}
   [[@parameter=] {value|@variable   [OUTPUT] | [DEFAULT]]  [,...
   n   ]
 [WITH RECOMPILE]
```

执行字符串的语法

```
EXEC[UTE]  ({@string_variable|  'tsql_string'  |expression}  [+...n] )
```

变量

@return_status	@return_status 可保存存储过程的返回状态，而且必须声明该变量。
procedure_name	procedure_name 指要执行的存储过程。
number	number 将存储过程组合在一起，以便于一次删除它们。
@procedure_name_var	@procedure_name_var 是保存存储过程名称的本地变量。
@parameter	@parameter 指存储过程的参数。
Value	Value 是放在参数变量中的值。
@variable	@variable 是保存参数的局部变量。
OUTPUT	OUTPUT 指存储过程将返回参数。
DEFAULT	DEFAULT 指参数的缺省值。
n	N 表示前面的项可以重复。
WITH RECOMPILE	WITH RECOMPILE 使查询处理器为存储过程开发新查询计划。
@string_variable	@string_variable 是本地串变量的名称，它保存联接在一起的 Transact-SQL 语句。
' tsql_string '	' tsql_string ' 是括在引号中的 Transact-SQL 语句或几个语句串。
Expression	表达式可以是常量、列名、函数、子查询程序或

算术运算符、按位运算符和字符串运算符。

[NOT] EXISTS

EXISTS 或 NOT EXISTS 通过返回真或假检测子查询程序返回的行是否存在。

语法

[NOT] EXISTS (subquery)

变量

NOT	NOT 为布尔表达式的否定指示符
Subquery	subquery 是 SELECT 语句。此语句中不能使用 ORDER BY、COMPUTE 或 INTO。

示例

下面是 NOT EXISTS 检测的示例。

```
SELECT DISTINCT name
FROM MyTable
WHERE NOT EXISTS
```

```
(SELECT *
FROM MyTable2
WHERE MyTable_ID = MyTable.MyTable_ID)
```

本例中，如果第二个表格中存在 ID，就返回表的准确名称。

EXP

EXP 返回浮点表达式的指数值。
语法

```
EXP（float_expresison）
```

变量 float_expresison 是浮点型数据。

FETCH

FETCH 返回 Transact-SQL 游标中的一行。
语法

```
FETCH
```

```
[[NEXT | PRIOR | FIRST | LAST
| ABSOLUTE { n | @nvar } | RELATIVE { n | @nvar} ]
FROM] { { [GLOBAL] cursor_name } | cursor_variable_name
  }
[INTO @variable_name [,...n ]]
```

变量

NEXT	NEXT 检索当前行下面的一行，它是 FETCH 的缺省选项。
PRIOR	PRIOR 检索当前行的前一行。
FIRST	FIRST 检索游标中的第一行。
LAST	LAST 检索游标中的最后一行。
ABSOLUTE{n | @nvar}	如果 n 或@nvar 为正，ABSOLUTE 使这一行成为当前行中游标前面的第 n 行，并对其进行检索。如果 n 或@nvar 为负，将检索游标末端前的第 n 行。
RELATIVE{n | @nvar}	如果 n 或@nvar 为正，RELATIVE 检索当前行后的第 n 行，如果 n 或@nvar 为负，RELATIVE 返回当前行前的第 n 行。
GLOBAL	GLOBAL 指全局游标。
cursor_name	cursor_name 是要从中提取数据的游标名称。

cursor_variable_name	cursor_variable_name 是游标变量的名称。
INTO@variable-name […n]	INTO 将数据放在局部变量中。

FILE_ID

FILE_ID 返回当前数据库中逻辑文件名称的文件标识号。
语法

```
FILE_ID( 'File_name' )
```

变量 file_name 是 sysfiles 系统表中文件的逻辑名称。

FILE_NAME

FILE_NAME 返回当前数据库中某个文件标识号代表的文件名称。
语法

```
FILE_NAME( file_id )
```

变量 file_id 是 sysfiles 系统表中的文件标识符列。

FILEGROUP_ID

FILEGROUP_ID 返回当前数据库中某文件组名称的文件组标识号。

语法

```
FILEGROUP_ID( 'filegroup_name' )
```

变量 filegroup_name 是文件组名称，且与 sysfilegroups 系统表中的文件组名称列相符。

FILEGROUP_NAME

FILEGROUP_NAME 返回当前数据库中某个文件组标识号对应的文件组名。

语法

```
FILEGROUP_NAME( filegroup_id )
```

变量 filegroup_id 是 sysfilegroups 系统表中的组标识符列。

FILEGROUPPROPERTY

当将文件组和属性名称作为变量提供时，FILEGROUPPROPERTY 返回文件组的属性值，1=真，0=假，NULL=属性值没有找到，属性名称可以是 IsReadOnly，IsDefault 和 IsUserDefinedFG。

语法

```
FILEGROUPPROPERTY( 'filegroup_name', 'property' )
```

变量

'filegroup_name'	filegroup_name 是文件组的名称。
'property'	property 是文件组属性的名称，IsReadOnly，IsDefault，和 IsUserDefinedFG。

FILEPROPERTY

FILEPROPERTY 返回文件的属性值，1=真，0=假，NULL=属性值没有找到，属性名可以是 IsLogFile，SpaceUsed，IsReadOnly 和 IsPrimaryFile。

语法

FILEPROPERTY('file_name' , 'property')

变量

'file_name'	file_name 是文件的名称。
'property'	property 是文件属性的名称 :IslogFile , SpaceUsed, IsReadOnly 和 IsPrimaryFile。

浮点数和实数

参阅 DataTypes。

FLOOR

FLOOR 返回小于或等于数字表达式变量的整数值。

语法

FLOOR(numeric_expression)

变量 numeric_expression 可以是小数、浮点数、整型数、货币值、实数、短整型数、小货币数或微短整型表达式。

示例

下面是 FLOOR 函数的示例：

```
SELECT FLOOR(2.3);FLOOR(-2.3)
GO
```

本例中，FLOOR(2.3)返回值为 2，FLOOR(-2.3)返回值为 -3。

FORMATMESSAGE

FORMATMESSAGE 格式化 sysmessages 系统表的系统信息中的信息，并返回已格式化的信息。

语法

```
FORMATMESSAGE(msg_number, param_value, [,…n] )
```

变量

msg_number	**msg_number 是信息的标识符，此变量作为一行保存在 master 系统数据库的 sysmessages 系统表格中。**
param_value	**param_value 是表示代替信息中的变量的正值(最大为 20)。**

FREETEXT

FREETEXT 搜索字符全文启动列中意思相符的列，在匹配搜索前，先把 'freetext-string' 分成加权搜索项。

语法

```
FREETEXT({column | *}, ' freetext_string ' )
```

变量

column	**column 指启动全文搜索的字符列的名称。**
*****	***指搜索所有的全文启动列，查找意义上与 freetext_string 变量匹配的列。**
freetext_string	**freetext_string 表示要在列中搜索的意义接近的字符。**

示例

下面为使用 FREETEXT 的示例：

```
SELECT MyColumn
FROM MyTable
WHERE FREETEXT (MyFullTextEnabledColumn, ' planets in the universe ' )
GO
```

本例中，在 MyFullTextEnabledColumn 中搜寻意思与变量 'planets in the universe' 相近的项，此搜索不如使用另一种全文搜索工具 CONTAINS 语法精确。

FREETEXTTABLE

FREETEXTTABLE 将结果集提供为表格行，在该表格中，字符列值在意义上与 'freetext_string' 变量中字符一致的每一行都附带有 RANK。该语法允许在使用 FREETEXT 的 SELECT 查询的 FROM 子句中使用 FREETEXTTABLE。

语法

```
FREETEXTTABLE(  table, {column | *}, 'freetext_string'  )
```

变量

table	table 是全文启用表格的名称。
Column	column 是搜寻的全文启用列的名称。
*	* 表示搜索所有的全文启用列，查找与 freetext_string 匹配的列。
freetext_string	freetext_string 表示要在列中搜索的意义一致的字符。

FROM

FROM 子句放在表格、视图、派生表，以及在 DELETE、INSERT、SELECT 和 UPDATE 语句中使用的连接表格的前面。

语法

```
FROM
{<table_or_view> | ( select_statement ) [AS] alias |<table_or_view>
 [{CROSS | INNER | {FULL | LEFT | RIGHT } [OUTER] [<join_hint>]
JOIN
<table_or_view> ON search_conditions ]
| CONTAINSTABLE ( table,{ column |*},
' <contains_search_condition> ' )
| FREETEXTTABLE
( table,{column | * },' freetext_string ' )} [,...n]

<table_or_view>::=
{ table [[AS] alias ] [WITH(<table_hint> [... m ] )]
| view [[AS] alias ]
| OPENROWSER( ' provider_name ' ,
{ ' datasource ' ;' user_id ' ;' password ' | ' provider_string ' },
```

```
   { [catalog.] [schema.] object_name |  'query' })
|OPENQUERY(   linked_server, 'query' )}

<table_hint>::=
( [INDEX =
{
   index_name
  | index_id]
}  [,...  n  ]
 [FASTFIRSTROW] [HOLDLOCK] [NOLOCK] [PAGLOCK]
 [READCOMMITTED] [READPAST] [READUNCOMMITTED]
 [REPEATABLEREAD] [ROWLOCK] [SERIALIZABLE] [TABLOCK]
 [TABLOCKX] [UPDLOCK]
)
<join_hint>::=
{
HASH
| LOOP
| MERGE
}
```

变量

<table_or_view>	<table_or_view>指 FROM 子句中引用的表格或视图。
alias	可以给每个表格或视图命名一个别名，即表格或视图的简写名称。两个表格中很可能有 2 个同名的列，这种情形下用别名以全限定的方式引用表格中的列就较方便。
<table_hint>	table_hint 决定所使用的索引，表格扫描或锁定策略。对新优化程序的提示请参阅第 6 章。
n	n 表示前面的项可以重复。
OPENROWSET	OPENROWSET 定义从数据源中访问远程数据的联接信息。请参阅本章后面的“ OPENROWSET ”。
OPENQUERY	OPENQUERY 执行通过查询程序。请参阅本章后面的“ OPENQUERY ”。
select_statement	此语句负责从数据库中返回行。
CROSS	CROSS 表示两表格的交叉积。
FULL [OUTER]	如果在 SELECT 语句中提供了 FULL [OUTER]，即使两个表格没有任何匹配，也要从两表格中接收行，不匹配行的列以 NULL 来显示。
INNER	INNER 联接是缺省值，它表示只返回匹配的行。
LEFT [OUTER]	LEFT [OUTER] 联接表示返回匹配的行，对于不匹

	配的行，左边表格中的行包括在结果集中，另一个表格中的行设置为 NULL。
RIGHT [OUTER]	RIGHT [OUTER] 联接表示返回匹配的行，对于不匹配的行，右边表格中的行包括在结果集中，另一个表格中的行设置为 NULL。
JOIN	JOIN 表示将进行联接操作。
<join_hint>	为查询优化程序提供 join_hint，说明在 FROM 子句中如何利用每个联接中的联接提示进行联系操作。
search_conditions	search_conditions 用 WHERE 子句的 Transact-SQL 语法列出对返回的行的限制。
n	n 表示前面的项可以重复。
CONTAINSTABLE	当搜索字符全文启用列，查找某个单词或短语的精确匹配或相近匹配，几个单词与其他单词的相近匹配，以及加权匹配时，CONTAINSTABLE 将返回一个表格。
FREETEXTTABLE	FREETEXTTABLE 返回在意义上与字符全文启用列相匹配的表格。

示例

下面是 ANSI FULL OUTER JOIN 的示例：

```
SELECT Column1, Column2
```

```
FROM MyTable a FULL OUTER JOIN MyTable2 b
ON a.My_Id = b.My_Id
WHERE MyTable.My_OtherId IN (987, 346, 985)
```

本例中，形成了全外部连接。

FULLTEXTCATALOGPROPERTY

FULLTEXTCATALOGPROPERTY显示与全文目录属性有关的信息。

语法

```
FULLTEXTCATALOGPROPERTY('catalog_name', 'property')
```

变量

'catalog_name'	**catalog_name是全文目录的名称。**
'property'	**全文目录的属性有:PopulateStatus 0=Idle（空闲），1=Population in progress(在进程中装载)，2=Paused(暂停)，3=Throttled(调节)，4=Recovering（恢复），5=Shutdown（关闭），6=Incremental population in progress(在进程中递增总值）7=更新索引。**

ItemCount 全文目录项的计数。
IndexSize 全文索引的容量，以 MB 为单位。
UniqueKeyCount 全文索引中专用键的计数。
logSize 最后一次装载的全文索引的容量，以 MB 表示。
PopulateCompletionAge 计算完成最后一次装载的时间时，可减去完成的日期和 01／01／1990 00：00：00。

FULLTEXTSERVICEPROPERTY

FULLTEXTSERVICEPROPERTY 显示与全文服务属性有关的信息。

语法

```
FULLTEXTSERVICEPROPERTY('property')
```

变量

'property'	全文服务的属性有:Resourceusage，1=背景，5=专用。ConnectTimeOut，乘 4 得到 Microsoft Search Service 的秒数，启动全文索引的装载。

IsFulltextInstalled，1=安装 Microsoft Search Service，0=没有安装 Microsoft Search Service

GETANSINULL

GETANSINULL 是一个函数，如果数据库使用无效性的 ANSI 缺省值，则该函数返回 1。如果无效性不是 ANSI 缺省值，就返回 0，在 CREATE TABLE 或 ALTER TABLE 语句中指定 NULL 就可以忽略它。设置缺省操作还可以使用 sp_dboption 'ANSI NULL DEFAULT'，{ture | false}，SET ANSI_NULL_DFLT_ON {ON | OFF}或者 SET ANSI_NULL_DEFLT_OFF {ON | OFF}。

语法

GETANSINULL(['database'])

变量 database 是数据库名称。

GETDATE

GETDATE 是一函数，它返回系统日期和时间。

语法

GETDATE()

GO

GO 是执行批量 Transact-SQL 语句的命令。
语法
GO

GOTO

GOTO 是将程序控制流传送给标签的命令。
语法

GOTO label

变量 label 是在 Transact-SQL 中 GOTO 语句传送程序控制的点。

GRANT

参阅第 5 章。

GROUP BY

参阅本章后面的“SELECT”。

GROUPING

在 CUBE 或 ROLLUP 子句插入行时，GROUPING 为包含值 1 的输出建立附加列，如果不是由 CUBE 或 ROLLUP 语句插入行，而是 CUBE 或 ROLLUP 和 GROUP BY 子句一起使用，则附加列将包含 0。

语法

GROUPING(column_name)

变量 column_name 表示检查 CUBE 或 ROLLUP 提示符的 GROUP 子句中的附加列。

HAVING

HAVING 是与 SELECT...GROUP BY 语句一起使用的分组搜索条件（通常使用集合函数）。它与 WHERE 语句很相似。

语法

HAVING < search_condition >

变量 search_condition 是必须满足的分组条件。

HOST_ID

HOST_ID 函数返回工作站的标识号。
语法

HOST_ID（）

HOST_NAME

HOST_NAME 函数返回工作站的名称。
语法

HOST_NAME（）

IDENT_INCR

IDENT_INCR 是一个函数。当用等同列创建表格或视图时，此函数返回所指定的表格或视图的相等的增量值。

语法

```
IDENT_INCR ( 'table_or_view' )
```

变量 table_or_view 是要查看其相等增量值的表格或视图。

IDENT_SEED

IDENT_SEED 是一个函数。当用等同列创建表格或视图时，此函数返回所指定的表格或视图的初值（即 seed)。

语法

```
IDENT_SEED ( 'table_or_view' )
```

变量 table_or_view 是要查看其相等初值的表格或视图。

IDENTITY

等同列提供某列中唯一的、自动生成的值。
语法

IDENTITY（data_type [, seed , increment]）

示例
下面是等同列的示例：

```
CREATE TABLE MyTable (My_Id int IDENTITY(1,1), MyDescr varchar(30))
```

本例中，将 My_Id 列创建为等同列。

IF...ELSE

参见本章后面的 ELSE。

IN

IN 子句查看某值是否在其它值的列表中。

语法

```
Expression IN
(subquery | expression  [,...n] )
```

变量

expression	**expression 指任意合法的 SQL 表达式。**
Subquery	**subquery 的结果集必须具有一列，表达式的数据类型必须一致。**

示例

下面是 IN 语句的示例：

```
SELECT MyColumn,MyId
FROM MyTable
WHERE MyId IN
(SELECT MyId
FROM MyOtherTable
WHERE MyOtherColumn LIKE (%this%))
```

在本例中，当在 MyOther 表中查找到 MyId 时，就返回 MyColumn 和 MyId。

INDEXPROPERTY

INDEXPROPERTY 对于在表中的索引返回索引的属性值。索引的属性有：IndexDepth，IsClustered，IsUnique，IndexFillFactor，IsPadIndex，IsFullTextKey，其中 1=真（true），0=假（false），NULL=没找到属性值。

语法

```
INDEXPROPERTY（ table_ID，index'，property' ）
```

变量

table_ID	table_ID 是表的 OBJECT_ID。
'index'	index 是索引名。
'property'	property 是索引的属性名。

INDEX_COL

INDEX_COL 是一个函数，它将返回已索引的列名。

语法

INDEX_COL（'table'， index_id，key_id）

下列变量的意义为：

'table'	table是表的名字。
index_id	index_id是索引的 indid。
key_id	key_id是键索引的主要列数（1到 16)。

INSERT

INSERT语句可在表或视图中插入一行或几行。

语法

```
INSERT [INTO] {<table_or_view>}
{
{
 [( column_list )] VALUES
(
  {
DEFAULT
|   constant_expression
```

```
} [,...  n  ]
)
|  select_statement
|  execute_statement
}
|DEFAULT VALUES
}
<table_or_view > ::=
{
   table  [[AS]  alias]  [WITH(<table_hint>  [ ...  m  ] )]
|  view  [[AS]  alias  ]
|OPENROWSET('  provider_name  ',
{
   'datasource';'user_id';'password'
|  'provider_string'
},
{
   [catalog.] [schema.] object_name
|  'query'
})
|OPENQUERY(  linked_server,'query'  )
}
```

```
<table_hint>::=
( [INDEX =
{
   index_name
|   index_id]
} [,... n ]
 [HOLDLOCK] [PAGLOCK] [READCOMMITTED]
 [READUNCOMMITTED] [REPEATABLEREAD]
 [ROWLOCK] [SERIALIZABLE] [TABLOCK] [TABLOCKX]
)
```

变量

INTO	**INTO 是表或视图前面的关键字。**
Column_list	**Column_list 是列的清单。**
VALUES	**VALUES 是在列的清单中按一一对应赋给各列的数值。**
DEFAULT	**DEFAULT 表示要放在列中的列的缺省值。该变量不能与等同列一起使用。**
constant_expression	**constant_expression 可以是表达式，常数，或变量。**
n	**n 表示前面的值可以重复。**

select_statement	SELECT 语句用于查询要放在表中的数据。
execute_statement	exectute_statement 是任意合法的 EXECUTE 语句。这些语句返回的结果集与要插入的结果集的列相匹配。
DEFAULT VALUES	DEFULT VALUES 表示新行将包含每一列的缺省值。
<table_or_view>	table_or_view 是表或视图的标题名。
table	table 是要放入数据的表名。
alias	alias 是赋给表或视图的缩写名。
view	view 是视图名。
OPENROWSET	OPENROWSET 包含了使用数据源中的远程数据的连接定义。
OPENQUERY	OPENQUERY 运行通过查询。请参阅本章后面的“OPENQUERY”。
<table_hint>	table_hint 通知 SQL Server 查询优化程序如何执行查询。请参阅第 6 章的“新优化程序提示”。
INDEX=index_name |index_id	如果给出了索引优化程序提示，该索引可用来处理查询。

示例

下面是 INSERT 语句的示例：

```
INSERT MyTable
(Column1, Column2, Column3)
VALUES
(1, DEFAULT, ' Value for column 3 ' )
```

本例中，值 1 是 column2 的缺省值，column3 的值可作为一行插入到 MyTable 中。

int,smallint 和 tinyint

请参阅数据类型。

IS_MEMBER

如果当前用户不是组或角色的成员，IS_MEMBER 函数就返回 0，如果当前用户是组或角色的成员，IS_MEMBER 函数就返回 1，如果不存在组或角色，IS_MEMBER 函数就返回 NULL。

语法

```
IS_MEMBER ( {' group' | ' role' } )
```

变量

'group'	group 是 Windows NT 组的名称，由 domain \ group 引用。
'role'	role 是 SQL Server 角色的名称，而不是服务器角色的名称。对于服务器角色，使用 IS_SRVROLEMEMBER。

IS_SRVROLEMEMBER

如果当前用户不是服务器角色的成员，IS_ SRVROLEMEMBER 函数就返回 0，如果当前用户是服务器角色的成员，IS_ SRVROLEMEMBER 函数就返回 1，如果不存在服务器角色，IS_ SRVROLEMEMBER 函数就返回 NULL。

语法

IS_SRVROLEMEMBER ('role' [,'login'])

变量

'role'	role 是服务器角色 sysadmin，dbcreator，diskadmin，processadmin，serveradmin，setupadmin，securityadmin 的名称。

'login'	login 是要检查的注册名。如果没有提供该变量，就使用当前用户的注册帐户。

ISDATE

ISDATE 函数检查变量或 varchar 列，当表达式包含合法的日期时，它返回 1，当表达式不包含日期时，它返回 0。

语法

```
ISDATE（expression）
```

变量 expression 是变量或 varchar 列。

IS [NOT] NULL

IS [NOT] NULL 检查表达式是否为 NULL。

语法

```
expression IS [NOT] NULL
```

变量

expression	expression 是任一表达式。
NOT	NOT 是布尔表达式的负运算符。

ISNULL

ISNULL 函数可用 NULL 替代非空数据。

语法

ISNULL（expression，substitution_value）

变量

expression	expression 是要检查空值的内容。
substitution_value	如果表达式为 NULL，就返回 substitution_value。

ISNUMERIC

如果表达式是整数、浮点数、货币或小数，ISNUMERIC 函数就返回 1，如果表达式不是一个数值，ISNUMERIC 函数就返回 0。

语法

ISNUMERIC（ expression ）

变量必须是一个字符串。

KILL

KILL 命令结束一个过程。
语法

KILL { spid } [WITH {ABORT | COMMIT}]

变量 spid 是要终止的过程 ID。

LEFT

LEFT 函数返回字符表达式从左边开始的给定的字节数。如果该整数是一个负数，就返回 NULL。
语法

LEFT（character_experession，integer_expression）

变量

character_experession	**character_experession 可以是变量、常数或表中的列。**
integer_expression	**integer_expression 可以是变量、常数或表中的列，表示要返回的字符数。**

LEN

LEN 函数提供了字符串表达式中字符的长度。
语法

LEN (string_ expression)

变量 string_expression 可以是变量、常数或列，也可以是返回字符数据类型的表达式（类似于合并字符串函数，它返回一个字符串）。

LIKE

LIKE 子句和字符串一起使用，以确定与通配符相匹配的模式。

语法

match_expression [NOT] LIKE pattern

变量

match_expression	**match_expression 是任何合法的字符 SQL 表达式。**
Pattern	**pattern 是要在变量 match_expression 中查找的内容。在模式中可以使用通配符。** **%表示查找任意长度的内容。** **_表示查找任一字符。** **[] 包含字符的范围或列表。** **[^] 要排除的字符。**

示例

下面是 LIKE 子句的示例：

SELECT *

```
FROM Mytable
WHERE Column1 LIKE "%MS%"
```

本例中，返回 MyTable 表的 Column1 列中在任意位置上有连续字符 MS 的所有行。

LOAD

LOAD 只为了向后兼容。可用 RESTORE 命令代替它。参阅本章后面的“RESTORE”。

LOG

当给出浮点表达式时，LOG 函数返回浮点表达式的自然对数。

语法

```
LOG（float_expression）
```

变量 float_expression 是浮点表达式。

LOG10

当给出浮点表达式时，LOG10 函数返回浮点表达式的底数为 10 的对数。
语法

LOG10（float_expression）

变量 float_expression 是浮点表达式。

LOWER

LOWER 函数将大写字符转换为小写字符。
语法

LOWER（character_expression）

变量是字符表达式。

LTRIM

LTRIM 删除字符表达式中的前导空格。
语法

LTRIM （character_expression）

变量是字符表达式。

MAX

MAX 返回表达式中的最大值，但它不能与位列一起使用。
语法

MAX ([ALL | DISTINCT] expression)

变量

ALL	**ALL 是缺省值。**
DISTINCT	**DISTINCT 表示唯一值。在这里使用该变量是与 ANSI 兼容的，但没有意义。**

Expression	expression 可以是常数、列或函数，也可以包含算术运算符、按位运算符和字符串运算符。

示例
下面是 MAX 函数的示例：

```
SELECT MAX(Column1)
FROM MyTable
```

本例中，返回表 MyTable 的 Column1 列的最大值。

MIN

MIN 返回表达式中的最小值，但它不能与位列一起使用。
语法

```
MIN( [ALL | DISTINCT]      expression    )
```

变量

ALL	ALL 是缺省值。
DISTINCT	DISTINCT 表示唯一值。在这里使用该变量是与 ANSI

兼容的，但没有意义。

Expression　　expression 可以是常数、列或函数，也可以包含算术运算符、按位运算符和字符串运算符。

示例

下面是 MIN 函数的示例：

```
SELECT MIN(Column1)
FROM MyTable
```

本例中，返回表 MyTable 的 Column1 列的最小值。

money 和 samllmoney

请参阅本章前面的“ Data Types”。

MONTH

请参阅本章前面的“日期函数”。MONTH（date）与 DATEPART(mm，date)相同。

NCHAR

NCHAR 返回整数 0-65535 的标准 Unicode 字符。
语法

NCHAR（integer_expression）

变量是整数 0-65535。

nchar 和 nvarchar

请参阅本章前面的“ Data Types”。

NEWID

NEWID 函数返回 uniqueidentifier 值。
语法

NEWID（）

NOT

NOT 是布尔表达式的负运算符。
语法

[NOT] boolean_expression

变量是合法的 SQL 布尔表达式。

ntext，text 和 image

请参阅本章前面的“ Data Types”。

NULLIF

如果一对表达式相等，NULLIF 函数就返回 NULL。如果它们不相等，NULLIF 函数就返回第一个表达式的值。
语法

NULLIF（expression ，expression）

变量 expression 可以是常数、列或函数，也可以包含算术运算符、按位运算符、和字符串运算符。

numeric

请参阅本章前面的“ Data Types”。

OBJECT_ID

当给当前数据库中的对象名时，OBJECT_ID 将返回对象标识号。
语法

OBJECT_ID（' object'）

变量 object_id 是对象名。

OBJECT_NAME

当给处当前数据库中的对象名时，OBJECT_NAME 函数返回对象名。
语法

OBJECT_NAME（ object_id ）

变量 object_id 是对象的标识号。

OBJECTPROPERTY

OBJECTPROPERTY 函数生成数据库对象的大量信息，它返回下述属性值：
CnstIsClustKey,CnstIsColumn, CnstIsDisabled, CnstIsNonclustKey,
CnstIsNotRepl, ExecIsAnsiNullsOn , ExecIsDeleteTrigger ,
ExecIsInsertTrigger , ExecIsQuotedIdentOn , ExecIsStartup ,
ExecIsTriggerDisabled, ExecIsUpdateTrigger, IsCheckCnst, IsConstraint,
IsDefault, IsDefaultCnst, IsExecuted, IsExtendedProc, IsForeignKey,
IsMSShipped , IsPrimaryKey , IsProcedure , IsReplProc , IsRule ,
IsSystemTable, IsTable, IsTrigger, IsUniqueCnst, IsUserTable, IsView,
OwnerId , TableDeleteTrigger , TableDeleteTriggerCount ,
TableFullTextKeyColumn , TableFullTextCatalogId ,
TableHasActiveFulltextIndex, TablehasCheckCnst, TableHasClustIndex,
TableHasDefaultCnst , TableHasDeleteTrigger , TableHasForeignKey ,
TableHasForeignRef , TableHasIdentity , TableHasIndex ,
TableHasInsertTrigger , TableHasNonclustIndex , TableHasPrimaryKey ,
TableHasRowGuidCol, TableHasTextImage, TableHasTimestamp,
TableHasUniqueCnst , TableHasUpdateTrigger , TableInsertTrigger ,

TableInsertTriggerCount , TableIsPinned , TableUpdateTrigger , TableUpdateTriggerCount , TriggerDeleteOrder , TriggerInsertOrder , TriggerUpdateOrder.

语法

```
OBJECTPROPERTY（id, ' property ' ）
```

变量

id	是对象的 ID。
' property '	是有 id 的对象的返回信息。Property 可以是上述值中的任一个。

OPEN

OPEN 语句与 Transact-SQL 服务器游标一起使用，以打开该服务器游标。

语法

```
OPEN {{ [GLOBAL] cursor_name} | cursor_variable_name}
```

变量

GLOBAL	GLOBAL 表示如果局部游标与全局游标同名，就使用全局游标。
cursor_name	cursor_name 是游标名称。如果局部游标与全局游标同名，就使用全局游标。如果指定了可选关键字 GLOBAL，就使用局部游标。
cursor_variable_name	cursor_variable_name 是游标变量的名称。

OPENQUERY

OPENQUERY 函数用于查询的 FROM 子句中，以替代表名，它也可以是 INSERT、UPDATE 或 DELETE 语句的目标表（如果能使用 OLE DB 提供商）。OPENQUERY 在定义为连接服务器的 OLE DB 数据源上运行通过查询。该连接服务器已由存储过程 sp_addlinkedserver 创建。

语法

```
OPENQUERY（linked_server,' query'）
```

变量

linked_server	linked_server 是连接的服务器名。
'query'	'query' 是在连接的服务器上执行的查询。

示例

下面是 OPENQUERY 函数的示例：

```
SELECT *
FROM OPENQUERY(Orcl_Linked_Srv1, ' SELECT Column1 FROM dbo.DBA_Table ' )
```

本例中，选择出 Oracle 服务器中的数据。

OPENROWSET

OPENROWSET 函数用于查询的 FROM 子句中，以替代表名，它也可以是 INSERT、UPDATE 或 DELETE 语句的目标表（如果能使用 OLE DB 提供商）。OPENROWSET 传送连接信息，以便访问 OLE DB 数据源上的远程数据。可以用它替代 OPENQUERY。也可以用它替代访问连接服务器中的表格，作为用 OLE DB 访问远程数据的 ad hoc 方式。

语法

```
OPENROWSET(' provider_name'
{
 ' datasource ' ;' user_id ' ;' password '
| ' provider_string '
},
```

```
{
 [catalog.] [schema.] object
| 'query'
})
```

变量

'provider_name'	provider 是注册表中 OLE DB 提供商的友好名称。
'datasource'	'datasource' 是 OLE DB 数据源的字母串。提供商用该串定位数据库或文件。
'user_id'	'user_id' 是发送给 OLE DB 提供商的用户名。
'password'	password 是为用户准备的，并传送给 OLE DB 提供商。
'provider_string'	provider_string 是专门用于提供商的，它在 DBPROP_INIT_PROVIDERSTRING 属性中给出，以初始化 OLE DB 提供商。
Catalog	catalog 是对象所在的目录或数据库名。
Schema	schema 是模式名或对象的拥有者。
Object	object 是对象名。
'query'	query 是提供商执行的通过查询。在提供商

没有通过表名显示列表数据，而只通过命令语言显示该数据时，使用通过查询是很有用的。

示例

下面是 OPENROWSET 函数的示例：

```
SELECT t1.*
FROM OPENROWSET('MSDASQL',
'DRIVER={SQL Server};SERVER=server1;UID=sa;PWD=server1pwd',
db1.dbo.MyTable) AS t1
ORDER BY t1.Column1, t1.Column2
GO
```

本例中，ODBC 和 SQL Server 驱动程序的 OLE DB 提供商发送了运行远程服务器的查询。

OR

OR 检查两个条件，如果其中一个条件为 TRUE，就返回 TRUE，先运算 AND，但括号的位置可控制运算的顺序。

语法

boolean_expression OR boolean_expression

变量 boolean_expression 是任一合法的 SQL 表达式，其值为 TRUE，FALSE 或 UNKNOWN。

ORDER BY

ORDER BY 确定 SELECT 语句中的排列顺序。

PARSENAME

PARSENAME 返回一个对象的对象名，拥有者的名称，数据库名和服务器名。但不通知用户该对象是否存在。

语法

PARSENAME（' object_name' ,object_piece}）

变量

' object_name'	object_name 是对象名，可以限定它。
object_piece	object_piece 是要返回的对象部分。该变量的合法值为：

1=对象名，2=拥有者的名称，3=数据库名，4=服务器名。

示例

下面是 PARSENAME 的函数示例：

```
SELECT PARSENAME('mydb..MyTable', 2) AS 'Owner Name'
```

本例中，解析出数据库 mydb 的表 MyTable 的拥有者名称，并返回 NULL。

PATINDEX

PATINDEX 函数返回表达式中遇到的第一个模式的起始位置，如果表达式中不存在模式，就返回 0。

语法

```
PATINDEX ('%pattern%',expression)
```

变量

pattern	**pattern 是可以包含通配符的字符串。**
Expression	**expression 是一个字符串，通常它是要搜索的列名。**

示例

下面是 PATINDEX 函数的示例：

```
SELECT PATINDEX('%MS%',Column1)
FROM MyTable
WHERE My_id=062554
```

PERMISSIONS

PERMISSIONS 函数返回的 32 位位图可显示当前对话中用户的语句、对象、列许可。存储过程 sp_helpprotect 也可以用来替代 PERMISSIONS 函数。

低 16 位是当前对话中用户的安全帐户许可。例如，返回值为 12（没有变量 objectid）表示当前用户拥有执行 CREATE PROCEDURE（小数位 4）和 CREATE VIEW（小数位 8）语句的许可。

高 16 位是当前对话中的用户可以授予其他用户的许可。高 16 位与低 16 位的内涵完全相同，但要乘以 65536。

返回的语句许可位图值是小数、位和许可；1=0x1=CREATE DATABASE;2=0x2=CREATE TABLE,4=0x4=CREATE PROCEDURE,8=0x8=CREATE VIEW,16=0x10=CREATE RULE,32=0x20=CREATE DEFAULT,64=0x40=BACKUP DATABASE,128=0x80=BACKUP LOG,256=0x100=BACKUP TABLE.

当提供了 objectid 时，返回的对象许可位图值是小数、位和许可；1=0x1=SELECT ALL，2=0x2=UPDATEALL，4=0x4=REFERENCES ALL,8=0x8=INSERT,16=0x10=DELETE,32=0x20=EXECUTE(存储程序),4096=0x1000=SELECT ANY,8192=0x2000=UPDATE

ANY,16384=0x4000=REFERENCES ANY.

当提供了 objectid 和列时，返回的列对象许可位图值是小数、位和许可；1=0x1=SELECT,2=0x2=UPDATE,4=0x4=REFERENCES.

如果不存在 objectid 和列，就返回 NULL。

语法

PERMISSIONS([objectid [, 'column']])

变量

objectid	为当前数据库中对象的 OBJECT_ID。如果没有提供 Objectid 语句，许可权返回给当前用户。
'column'	可选的要返回许可权的列名。

示例

下面是 PERMISSIONS 函数的示例：

```
IF PERMISSIONS()&1=1
CREATE TABLE DBA_Table (Column1 INT, Column2 VARCHAR)
ELSE
GOTO ErrorRtn
```

PI

PI 函数返回 pi 的值 3.14159265358979。
语法

```
PI ( )
```

示例
下面是 PI 函数的示例：

```
SELECT PI ( )
```

本例中，返回 pi 的值 3.14159265358979。

POWER

函数返回给定数值表达式的幂。
语法

```
POWER（numeric_expression，y）
```

变量

numeric_expression	numeric_expression 的数据类型可以是小数、浮点、整型、货币、数值、实型、短整型、短货币或微整型。
y	表达式的幂指数。

PRINT

PRINT 语句将用户定义的信息返回到客户机，该信息最多可以有 1024 个字符。

语法

PRINT{ ' any ASCII text ' |@local_variable | @@global_variable | string_expr}

变量

' any ASCII text '	any ASCII text 是一字符串。
@local_variable	@local_variable 是局部变量。
@@global_variable	@@global_variable 是返回字符串的全局变量。
string_expr	string_expr 是返回字符串的表达式。

QUOTENAME

QUOTENAME 函数返回 Unicode 字符串，此 Unicode 字符串将字符串变量转换成有效的引用标识符。

语法

```
QUOTENAME('character_string' [,'quote_character'])
```

变量

'character_string'	character_string 是一 Unicode 字符串。
'quote_character'	quote_character 是用作分隔符的字符串。该变量的可能值为单引号，左括号，右括号或双引号。

RADIANS

RADIANS 为一函数，在提供以度表示的数字表达式时，RADIANS 返回其弧度值。

语法

RADIANS(numeric_expression)

变量 numeric_expression 可以是小数、浮点数、整型数、货币、数字、实数、短整型数、短整型数，小货币或微短整型数等。

RAISERROR

RAISERROR 是一个函数，它返回用户定义的错误信息，并设置系统标志以指明出现的错误。此信息可能是来自 sysmessages 系统表的错误，也可能是带有用户定义的严重级别和状态的函数内部的错误。如果错误在运行期间出现，信息就发送到客户机上。

如果使用了 sysmessages 系统表，并使用变量 msg_str 的格式选项定义信息，那么变量就传送给 msg_id。系统存储过程 sp_addmessage 和 sp_dropmessage，用于建立或删除用户定义的错误信息。

@@ERROR 存储最近产生的错误号，缺省状态下设置为 0，严重级别为 1 到 10。

语法

```
RAISERROR ({    msg_id | msg_str   } {,    severity, state   }
 [,    argument
 [,...   n   ]]  )
 [WITH    option   ]
```

变量

msg_id	msg_id 是用户定义的保存在 sysmessages 表中的错误 信息。应使用户定义的错误信息的错误号大于 50,000。Ad hoc 信息引发错误 50,000。msg_id 的最大值为 2,147,483,647。
msg_str	msg_str 是最多可达 8000 个字符的 ad hoc 信息，标准信息 ID 为 14,000。字符%表示这是 ad hoc 信息，格式化为： %[[flag] [width] [precision] [{h | l}]]type flag=对不用单引号的信息，定义其间距和调整的代码。 '-'=域内的左边调整。 +=表示带有正或负号的值。 0=零填充字符数据类型。 #=非十进制数的词根前缀。 ' '=空格插入正值，使用+标志时可以忽略。 Width=整数和最小宽度，*表示由精度定义宽度。 Precission=字符的最大数或小数的最小位数，*表示由变量定义精度。 {h | 1}type=修改量(h 表示短整型数，或 1 表示长整型数)用数字类型定义数字变量的容量。

	类型有： Di=带符号的整数 :u=无符号的整数 :o=无符号的八进制数 :x 或 X=无符号的十六进制数 :P=指针
severity	severity 是此信息的用户定义的严重等级。 1-18:所有用户 19-25：系统管理员且必须用 WITHLOG 20-25：中断客户机连接的致命错误，且在 SQL Server 错误记录中和 NT 事件记录中注册。
state	state 是表示错误状态的从 1 到 127 的任意整数。
argument	argument 表示在 msg_str 中变量的替代参数，最大值为 20，参数可以局部变量或数据类型为整数 1，整数 2、整数 4、字符、可变字符、二进制或可变二进制的常数。
Option	option 表示错误的定制选项。 LOG：在服务器错误记录和事件记录中记录错误，只能由系统管理员来使用。 NOWAIT：信息立即送至客户机。 SETERROR：不管严重等级是多少，@@error 都是 ad hoc 信息的 msg_id 或 50000。

示例
下面是两个 RAISERROR 函数的示例：

RAISERROR (' Customerid 4 expects the district of 44. ' ,16,-1)

RAISERROR (' The district for customer_id:%d should be

between %d and %d. ' ,
16,-1,@ @ customer_id, @ @ min_dist,@@max_dist)

第一个例子中，建立了简单的信息。第二 RAISERROR 动态建立信息。

RAND

RAND 返回 0～1 之间的随机浮点值。
语法

RAND([seed])
变量 seed 决定起始值，它可以为整型数、短整型数、微短整型数据类型。

READTEXT

READTEXT 返回文本或图像列中的文本、ntext 或图像值，它从规定的偏置开始，返回规定字节数。可以用 TEXTPTR 函数获取表格的有效 text_ptr 值。参阅本章后面的“TEXTPTR”和“SETTEXTSIZE”。

语法

READTEXT{table.column text_ptr offset size} [HOLDLOCK]

变量

table.column	表格和列的名称。
text_ptr	text_ptr 是二进制(16)的有效文本指针。
Offset	offset 在放入文本或图像数据前要忽略的字符数（字节）。Ntext 数据类型必须以字符方式给出（每个数据两个字节），文本或图像数据类型必须以字节的方式给出。
Size	size 是要读取的数据的字节数，其中 0 表示数据为 4K 字节。
HOLDLOCK	事务处理结束前锁定文字值，这些文字值可以读取，但不能改变。

示例

下面是 READTEXT 函数的示例：

```
DECLARE @ varbinary(16)
SELECT @pntr = TEXTPTR(MyTable)
READTEXT MyTable.Column1 @pntr 2 55
```

本例读取出 MyTable 中 Column1 的第 3 到第 57 个字符。

实型

参阅本章前面的“ Data Type ”。

RECONFIGURE

RECONFIGURE 更新当前配置，对于某些配置选项，RECONFIGURE 更新由 sp_configure 修改的当前运行值。对于某些配置选项，Microsoft SQL Server 必须中止再启动，才能更新它们的当前运行值。

语法

```
RECONFIGURE [WITH OVERRIDE]
```

变量

RECONFIGURE	RECONFIGURE 更新当前配置，对于某些配置选项，RECONFIGURE 更新由 sp_configure 修改的当前运行值。它还检查无效值和坏值。
WITH OVERRIDE	WITH OVERRIDE 禁止对允许的更新，恢复间隔或时间段配置选项的检查，并用 sp_configure 提供的值重新配置。

示例

下面是 RECONFIGURE WITH OVERRIDE 语句的示例：

```
EXEC sp_configure 'allow updates',1
RECONFIGURE WITH OVERRIDE
GO
```

本例中，重新配置服务器，以允许更新系统表。即使该值为真，SQL Server 也不通知用户不推荐该值。

REPLACE

REPLACE 用串表达式 3 取代串表达式 1 中的串表达式 2。

语法

REPLACE (' string_expression 1 ' , ' string_expression 2 ' , ' string_expression 3 ')

变量

' string_expression 1 '	**' string_expression 1 ' 是在其中搜索 string_expression2 的串表达式。可以使用字符或二进制数据。**
' string_expression 2 '	**string_expression2 是在 string_expression1 中查找的串表达式，也是用 string_expression 3 取代的串表达式。**
' string_expression 3 '	**string_expression 3 是取代 string_expression 2 的新串表达式。**

示例

下面是 REPLACE 的示例：

SELECT REPLACE (' MyDog ' , ' Dog ' , ' Cat ')

本例中，用 Cat 取代了 MyDog 中的 Dog，结果为 MyCat。

REPLICATE

REPLICATE 将字符表达式复制用户定义的次数。

语法

REPLICATE(character_expression，integer_expression)

变量

character_expression	character_expression 是字符或二进制常数，变量或列。
integer_expression	integer_expression 是一个正整数，表示复制字符表达式的次数。

RESTORE

RESTORE 语句恢复数据库、记录或数据库文件，可以进行以下类型的恢复：备用服务器的恢复、整个数据库的恢复，差异数据库的恢复(差异备份是用 RESTORE DATABASE 语句恢复的)、事务处理记录的恢复、个别文件或文件组的恢复。文件或文件组可以从文件或文件组备份中恢复，或者从整个数据库备份中恢复。恢复文件或文件组时，必须使用事务处理记录。

只有 RESTORE DATABASE 影响返回数据库备份设置的自动关闭、自动压缩、自动剪切、只有 dbo 使用、只读、单用户、选择／成批复制或检测已拆分下的页面等设置。

语法

恢复整个数据库的语法：

```
RESTORE DATABASE { database_name | @database_name_var }
 [FROM <backup_device> [,... n ]]
 [WITH
 [DBO_ONLY]
 [[,] FILE = file_number ]
 [[,] MEDIANAME = { media_name | @media_name_variable }]
 [[,] MOVE 'logical_file_name' TO 'operating_system_file_name
']
 [,...p]
 [[,] {NORECOVERY | RECOVERY | STANDBY = undo_file_name
 }]
 [[,] {NOUNLOAD | UNLOAD}]
 [[,] REPLACE]
 [[,] RESTART]
 [[,] STATS [= percentage ]]
]
```

恢复特定文件或文件组的语法：

```
RESTORE DATABASE { database_name | @database_name_var }
```

```
<file_or_filegroup> [,... m ]
 [FROM <backup_device> [,... n ]]
 [WITH
 [DBO_ONLY]
 [[,] FILE = file_number ]
 [[,] MEDIANAME = { media_name | @media_name_variable }]
 [[,] (NORECOVERY |RECOVERY | STANDBY = undo_file_name }]
 [[,] {NOUNLOAD | UNLOAD}]
 [[,] REPLACE]
 [[,] RESTART]
 [[,] STATS [= percentage ]]
]
```

恢复事务处理记录的语法：

```
RESTORE LOG { database_name | @database_name_var }
 [FROM<backup_device> [,... n ]]
 [WITH
 [DBO_ONLY]
 [[,] FILE = file_number ]
 [[,] MEDIANAME = { media_name | @media_name_variable} ]
 [[,] MOVE 'logical_file_name' TO 'operating_system_file_name' ]
```

```
 [,...   p   ]
 [[,]  {NORECOVERY | RECOVERY | STANDBY =    undo_file_name}  ]
 [[,]  {NOUNLOAD | UNLOAD}]
 [[,]  RESTART]
 [[,]  STATS  [ =    percentage  ]]
 [[,]  STOPAT = {   data_time | @date_time_var   }]
]

<backup_device> :: =
{
{   ' backup_device_name '  | @backup_device_name_var}
| {DISK | TAPE | PIPE} =
   {  ' temp_backup_device '  | @ temp_backup_device_var}
}

<file_or_filegroup> :: =
{
FILE = {   logical_file_name | @ logical_file_name_var}
|
FILEGROUP = {   logical_filegroup_name | @ logical_filegroup_name_var    }
}
```

变量

DATABASE	如果使用 DATABASE，就通知 Microsoft SQL Server 恢复完整的数据库。也可以和一系列文件或文件组一起使用 DATABASE，限制将数据库恢复到指定的文件或文件组。现在，Microsoft SQL Server 恢复备份文件时，只备份足以创建带有完整事务处理的数据库的记录。
{database_name | @database_name_var}	database_name 是与恢复程序有关的数据库的名称。可以用变量或常数串的形式传输数据库名称。
FROM	FROM 限定从哪个备份设备处恢复。忽略 FROM 子句可用于恢复用 NORECOVERY 选项恢复的非可疑数据库，或者用于转换至备用服务器，如果省去 FROM 子句，必须给出 NORECOVERY，RECOVERY，或者 STANDBY。
<backup_device>	backup_device 是 sp-addumpdevice 建立的备份设备的逻辑名称。
{DISK | TAPE | PIPE}= ' temp_backup_device ' | @temp_backup_device_var	允许从指定的磁盘、磁带或管道设备中恢复备份。DISK=' c:＼mssql7＼backup＼myback.dat ' 或 TAPE=' ＼＼.＼TAPE0 '。管道类型的设备可用通过第三方厂商使用的客户应用程序来命名管道。
n	n 表示前面的项可以重复。
DBO_ONLY	DBO_ONLY 表示只有 Database Owner 可以访问恢复

	的数据库。
FILE=file_number	file_number 是要恢复为显示在介质上的备份集的数目。
MEDIANAME={media_name | @media_name_variable}	MEDIANAME 为整个备份集的介质名称。它必须与备份卷中的介质名称相符。在 BACKUP 和 RESTORE 操作中使用介质名称是一种可靠的检查。
MOVE 'logical_file_name' TO 'operating_system_file_name'	MOVE 决定将 logical_file_name 移至 operating_system_file_name 中，缺省值为将 logical_file_name 恢复至其开始位置。可以用此变量将数据库重新定位到另一个服务器上。
p	p 表示可以列出多个 MOVE 语句。
NORECOVERY	不重新运行未提交的事务处理。必须将 NORECOVERY 或 RECOVERY 指定为缺省值。恢复数据库和事务处理记录时，除最后一个 RESTORE 以外都使用 NORECOVERY。当所有的事务处理记录都没有应用时，NORECOVERY 使数据库不能使用。如果使用 STANDBY，在只读模式中能使用数据库。RECOVERY 为缺省值。
RECOVERY	重新运行未提交的事务处理，使数据库可用。当另一个事务处理记录应用于数据库时，不应指定 RECOVERY。

STANDBY=undo_file_name	这是存储数据以撤消恢复影响的文件名。与 NORECOVERY 不同，可以查看事务处理记录恢复之间的数据库。如果文件不在操作系统上，就建立文件。如果文件存在，就覆盖文件。
NOUNLOAD	NOUNLOAD 表示恢复以后，磁带不会自动从磁带驱动器上卸载。该设置一直保持此状态，直到恢复指令中发出 UNLOAD 改变该设置为止。
UNLOAD	此变量决定恢复完成以后，磁带将自动倒带并卸载。这是对磁带驱动器的缺省操作，设置一直保持此状态，直到恢复指令中发出 NOUNLOAD 改变该设置为止。
REPLACE	REPLACE 表示 SQL Server 将删除数据库，并根据备份集规范重新建立数据库。如果不使用 REPLACE 选项，下列情况下不恢复数据库： 数据库已经存在于服务器中——没有 REPLACE，不会进行恢复。 数据库名称与备份集中的数据库名称不相同——没有 REPLACE，不会进行恢复。 数据库中的文件集合与备份集中包含的数据库文件集不一致——没有 REPLACE，不会进行恢复。 恢复过程中 Microsoft SQL Server 不分辨文件容量的差别，进行恢复时，也不检查文件容量的差别。

RESTART	RESTART 是中断时允许重新开始恢复。要重新开始恢复，重复 RESTORE 指令并指定 RESTART 选项。但是要注意，该选项只能用于从磁带介质中恢复，以及通过多个介质卷写入的恢复。
STATS [=percentage]	如果指定 STATS，将显示完成的百分比统计数字。如果不提供 STATS，缺省值为 10%。
<file_or_filegroup>	当数据库的容量太大，从性能上讲不适于全集数据库备份时，可使用此变量。此变量可以是一个文件或多个文件，也可以是一个文件组或者多个文件组。
FILE={logical_file_name | @logical_file_name_var}	FILE 定义要包含在数据库恢复的文件。
FILEGROUP={logical_filegroup_name | @logical_filegroup_name_var}	FILEGROUP 定义要包括在数据库恢复中的文件组。最后一个文件或文件组的恢复操作将文件向前滚动到数据库的其余文件处时，必须立即将记录应用到数据库文件上。如果备份以后文件没有修改，不必使用记录，如果最后一次备份这些文件后，在文件组上建立了索引，就可能需要多个文件组。
m	m 表示一个 RESTORE 语句中可以使用多个文件和文件组。
LOG	只恢复事务处理记录的备份。要使用多个事务处理

	记录，除最后一个记录的恢复以外，都必须使用 NORECOVERY 变量。
STOPAT=date_time | @data_time_var	STOPAT 控制停止恢复的准确日期和时间，只有在 STOPAT 日期和时间以前提交的事务处理能恢复到数据库中。

示例

下面是用文件、文件组和事务处理记录恢复数据库的示例：

```
RESTORE DATABASE MyDB
FILE = 'MyDB_data_1',
FILE = 'MyDB_data_2',
FILEGROUP = 'My_filegroup'
FROM MyDB_1
WITH NORECOVERY
GO
RESTORE LOG MyDB
FROM MyDBLog1
GO
```

本例中，通过恢复 MyDB 数据库的文件、文件组和记录恢复了数据库。

示例

下面是恢复整个数据库和差异备份的示例：

```
RESTORE DATABASE MyDB
FROM MyDB_1
WITH NORECOVERY
GO
RESTORE DATABASE MyDB
FROM MyDB_1
WITH FILE = 2
GO
```

本例中，恢复了整个数据库和差异数据库。差异数据库作为文件 2 放在备份文件中。

示例

下面是恢复整个数据库和事务处理记录的示例：

```
RESTORE DATABASE MyNwind
FROM MyNwind_1
WITH NORECOVERY,
MOVE   'MyDB'   TO   'C:\mssql7\data\MyNewDB.mdf',
MOVE   'MyDBLog1'   TO   'C:\mssql7\data\MyNewDB.ldf'
RESTORE LOG MyDB
```

```
FROM MyDBLog1
WITH RECOVERY
```

本例中，恢复了数据库与事务处理记录。MOVE 语句将要恢复的数据和记录文件移动到指定的位置。

示例

下面是用 STOPAT 恢复到指定时刻的示例：

```
RESTORE DATABASE MyDB
FROM MyDB_1,MyDB_2
WITH NORECOVERY
GO
RESTORE LOG MyDB
FORM MyDBLog1
WITH NORECOVERY
GO
RESTORE LOG MyDB
FROM MyDBLog2
WITH RECOVERY,STOPAT = ' Sept 15,1998 12:00 AM'
GO
```

本例中，数据库恢复到 1998 年 9 月 15 日的上午 12：00。

示例

下面是恢复整个数据库备份的示例：

```
RESTORE DATABASE MyDB
FROM MyDB_1
```

本例中，恢复了整个数据库。

RESTORE FILELISTONLY

RESTORE FILELISTONLY返回备份集中一系列的数据库和记录文件。

语法

```
RESTORE FILELISTONLY
FROM<backup_device>
 [WITH
{FILE = file_number>
 [[,]  {NOUNLOAD|UNLOAD}]
]
<backup_device>::=
{
{  'backup_device_name' |  @backup_device_name_var  }
|{DISK|TAPE|PIPE}=
{  'temp_backup_device'  |@temp_backup_device_var  }
}
```

式中各个变量参阅 RESTORE 语句。

RESTORE HEADERONLY

RESTORE HEADERONLY 返回备份设备中所有备份集的备份标题信息。

语法

```
RESTORE HEADERONLY
FROM <backup_device>
 [WITH {NOUNLOAD | UNLOAD}]

<backup_device>::=
{
{    'backup_device_name' | @backup_device_name_var    }
| {DISK | TAPE | PIPE} =
{    'temp_backup_device'  | @temp_backup_device_var    }
}
```

式中各变量参阅 RESTORE 语句。

RESTORE LABELONLY

RESTORE LABELONLY 返回备份设备中备份介质的信息：

语法

```
RESTORE LABELONLY
FROM <backup_device>
<backup_device>::=
{
{    ' backup_device_name '  | @backup_device_name_var    }
| {DISK | TAPE | PIPE} =
{    ' temp_backup_device '   | @ temp_backup_device_var    }
}
```

式中各变量参阅 RESTORE 语句。

RESTORE VERIFYONLY

RESTORE VERIFYONLY 检验备份集是否完整，并检验是否所有的卷都能读取。它不恢复备份或检验数据。

语法

```
RESTORE VERIFYONLY
FROM <backup_device> [,... n ]
 [WITH
 [FILE = file_number ]
 [[,] {NOUNLOAD|UNLOAD}]
 [[,] LOADHISTORY]
]

<backup_device>::=
{
{ 'backup_device_name' |@backup_device_name_var }
|{DISK|TAPE|PIPE} =
{ 'temp_backup_device' |@temp_backup_device_var }
}
```

式中各变量参阅 RESTORE 语句。

RETURN

RETURN 语句从查询或过程中退出，而不执行 RETURN 语句后面的语句。用户定义的返回状态值不应与 SQL Server 保留的值（0～-99）有冲突。

如果没有提供返回值，就使用 SQL Server 值。SQL Server 现在使用值 0～

-14。这些值与其使用说明如下所示：

0=成功地执行完程序，-1=对象失踪(不见)，-2=数据类型有错误，-3=过程被选择未死锁牺牲品，-4=许可有错误，-5=语法有错误，-6=其他用户错误，-7=资源错误，-8=非致命内部错误，-9=达到系统极限，-10=致命内部矛盾，-11=致命内部矛盾，-12=表格或索引崩溃，-13=数据库崩溃，-14=硬件有错误

语法

```
RETURN [(integer_expression)]
```

变量 integer_expression 是返回至调用存储过程或者应用程序的整数值。

示例

下面是使用 RETURN 的示例：

```
CREATE PROC MyProc
AS
DECLARE @rtn int
SELECT *
FROM SYOBJECTS
If @@error = 0
BEGIN
SELECT @rtn = 0
END
```

```
ELSE
BEGIN
SELECT @rtn = 1
RETURN @rtn
@END
GO
```

本例中，通过执行存储过程 MyProc，并检查成功或失败的返回代码，就可以进一步操作。

```
DECLARE @rtn int
EXEC @rtn = MyProc
IF @rtn = 0
PRINT ' Successful execution of MyProc stored procedure '
ELSE
PRINT ' Failed execution of MyProc stored procedure '
GO
```

REVERSE

REVERSE 返回字符表达式的倒序。如果传输 ' pam ' ，将接收到 ' map ' 。

语法

REVERSE(character_expression)

变量 character_expression 是包括字母和数字的字符或二进制表达式、常数、变量或列。

REVOKE

参阅第 5 章。

RIGHT

RIGHT 返回字符串中从右数给定字符数的字符串。
语法

RIGHT(character_expression，integer_expression)

变量

character_expression　　character_expression 是字符或二进制包括

integer_expression	字母和数字的表达式、常数、变量或者列。integer_expression是正整数。

ROLLBACK TRANSACTION

ROLLBACK TRANSACTION 删除开始进行事务处理以来的更改的所有数据，调用它可将事务处理返回到保存点或事务处理的起始点。

语法

```
ROLLBACK [TRAN [SACTION] [transaction_name |
@tran_name_variable |
savepoint_name |
@savepoint_variable] ]
```

变量

transaction_name	transaction_name 是 BEGIN TRANSACTION 给事务处理起的名称。
@tran_name_variable	@tran_name_variable 是用户定义的变量名称，它保存数据类型为 CHAR，VARCHAR，NCHAR，或者 NVARCHAR 的事务处理名。
savepoint_name	savepoint_name 是 SAVE TRANSACTION 语句给

	出的 savepoint_name。
@savepoint_variable	@savepoint_variable 是用户定义的变量名称，它保存数据类型为 CHAR，VARCHAR，NCHAR，或者 NVARCHAR 的保存点的名称。

ROLLBACK WORK

除 ROLLBACK TRANSACTION 可以接收事务处理名称以外，ROLLBACK WORK 与 ROLLBACK TRANSACTION 相同。

语法

```
ROLLBACK [WORK]
```

ROUND

ROUND 返回给出的数学表达式，并四舍五入至指定的长度或精度。

语法

```
ROUND(numeric_expression, length [,function] )
```

变量

numeric_expression	numeric_expression 可以是小数表达式、浮点数表达式、整型数表达式、货币表达式、数字表达式、实数表达式、短整型数表达式、小货币表达式或微短整型数表达式。
length	length 是四舍五入的精度。它可以是微短整型数、短整型数或整型数。当长度为负时，数字表达式在小数点左边进行四舍五入。
function	function 是微短整型数、短整型数或整型数，是要执行的操作。如果其值为 0，将数字表达式四舍五入。如果其值不为 0，将数字表达式舍位。

RTRIM

RTRIM 删去尾部的空格。

语法

RTRIM(character_expression)

变量 character_expression 是字符或二进制包含字母和数字的表达式，常量、变量或列。

SAVE TRANSACTION

SAVE TRANSACTION 在事务处理中建立保存点。
语法

```
SAVE TRAN [SACTION] {savepoint_name | @savepoint_variable}
```

变量

savepoint_name	**存储点的名称。**
@savepoint_variable	**用户定义变量的名称，它存储数据类型为 CHAR, VARCHAR NCHAR 或 NVARCHAR 的存储点名。**

SELECT

SELECT 语句返回数据库表中的行。
语法

```
SELECT
 [ALL | DISTINCT]
 [
```

```
TOP    n    [PERCENT]    [WITH TIES]
]
{ <result_data>:: =
{ *
|  [   column_heading = ]  column_name
|    column_name    [[AS]     column_heading   ]
|    column_name    AS    expression
|    expression    [AS]     column_heading   ]
|GROUPING (   column_name   )
|IDENTITYCOL
|ROWGUIDCOL
|    local_or_globa1_variable
|    new_column_name    = IDENTITY(   data_type,seed,increment   )
|    fulltext_table.   RANK
}  [,...   n   ]
|<variable_assignment>:: = {   @local_variable = expression} [,...n   ]



}
 [INTO    new_table_name   ]
 [FROM
{
```

```
<table_or_view>
|(    select_statement    )  [AS]       alias
|<table_or_view>
 [  {CROSS|INNER}
|
{
FULL
|LEFT
|RIGHT
}  [OUTER]    [<join_hint>]  JOIN <table_or_view>
ON     search_conditions
]
|CONTAINSTABLE
(    table,{ column     |*},'<    contains_search_condition>'
)
|FREETEXTTABLE
(table,{colunn|*},'freetext_string'
)
} [,...n]
]
 [  WHERE
{   [<search_conditions>::=
```

```
 [NOT] <predicate> [ {AND|OR} [NOT] <predicate>]
] [,... n ]
 [ |CONTAINS
({ column |*},' <contains_search_condition> '
)
|FREETEXT
(
{ column |* }, ' freetext_string '
)
] [,... n ]
| fulltext_table.fulltext_key_column = alias. [KEY]
}
]
 [{
GROUP BY [ALL] { column_name } [,... n ]]
 [WITH {CUBE|ROLLUP}]
}]
 [HAVING search_conditions ]
 [ORDER BY
{
{
{ table.|view.}column_name
```

```
|   select_list_number
|    expression
}  [ASC | DESC]
} [,...   n   ]
]
 [COMPUTE    row_aggregate({expression | column_name}  [,...n]    )
 [BY  {    expression | column_name) [,...n]]
]
 [FOR BROWSE]
 [OPTION (<query_hint>  [,...   n   ] )]
<table_or_view>::=
{
   table     [[AS]    alias   ]   [WITH(<table_hint> [...   m   ] )]
|    view    [[AS]     alias   ]
|OPENROWSET(    ' provider_name ' ,
{
   ' datasource ' ;' user_id ' ;' password '
 |    ' provider_string '
}
 {
   [ catalog.] [schema.] object_name
|    ' query '
```

```
})
|OPENQUERY(    linked_server,' query '    )
}

<table_hint>::=
( [INDEX=
{
   index_name
|    index_id
}  [,...   n   ]
 [FASTFIRSTROW]
 [HOLDLOCK] [NOLOCK] [PAGLOCK] [READCOMMITTED]
 [READPAST] [READUNCOMMITTED] [REPEATABLEREAD]
 [ROWLOCK] [SERIALIZABLE] [TABLOCK] [TABLOCKX]
 [UPDLOCK] )
<join_hint>::=
{
HASH
| LOOP
|MERGE
}
```

```
<query_hint>::=
{ {HASH | ORDER} GROUP
|{MERGE | HASH | CONCAT}UNION
|FAST     n
| FORCE ORDER
| ROBUST PLAN
}
```

变量

SELECT ALL	ALL 允许在结果集中包含相同的行。
DISTINCT	DISTINCT 只检索结果集中的独有的行。
TOP n	TOP 表示只返回前 n 行。
RERCENT	PERCENT 返回前 n% 行。
WITH TIES	WITH TIES 只和 ORDER BY 子句一起使用，也可与 TOP 和 PERCENT 一起使用。如果 ORDER BY 列中具有相同的值的行彼此相联系，有时 WITH TIES 返回的行数多于 n(用户定义的值)。
<result_data>	result_data 是希望 SQL Server 返回的值。它可以是一列(或几列)、表达式、变量的赋值或者星号(*)，*表示表格中所有的列。
Column_heading	如果希望建立一个自己想要的列标题，可以用

	Column_heading 替代 SQL Server 提供的列标题。可以使用下述语法： column_name AS column_heading column_heading= column_name column_name column_heading
GROUPING	GROUPING 跟 GROUP BY 和 CUBE 或 ROLLUP 一起使用，适用于返回集中的列名，也就是 GROUP BY 列。GROUPING 输出一附加列，如果列值为 1，表示 CUBE 或 ROLLUP 添加累加行，如果列值为 NULL 或 0 值，表示返回集中的列值无效。
IDENTITYCOL	IDENTITYCOL 表示表格中标识列的名称。
ROWGUIDCOL	ROWGUIDCOL 表示唯一标识符列是一行全局唯一标识符列。
local_or_global_variable	local_or_global_variable 指定局部或全局变量的名称。
new_column_name	new_column_name 定义数据类型为 NOT NULL 微短整数、短整型数、整型数、小数或数字的新列。
IDENTITY	IDENTITY 与 SELECT INTO 一起使用，表示该列将使用 IDENTITY 属性。
data_type	data_type 指标识列的数据类型，它可以为整型数、短整型数、微短整型数、小数或数字。
seed	seed 和 IDENTITY 一起使用，是表格中第一行的值。

	缺省值为 1。
increment	increment 是将下一行添加到表格时根值的增量值。然后，它从添加的最后一行开始递增，一直到插入表格中的下一行。
n	n 表示前面的项可以重复。
fulltext_table	fulltext_table 是为全文查询标志的表格。
RANK	RANK 和全文查询一起使用，显示每行的序列值。
<variable_assignment>	variable_assignment 是局部变量的赋值。
@local_variable=expression	@local_variable=expression 是局部变量的赋值。如果可能，最好使用 SET 语句，因为，如果 SELECT 语句返回多个值，就将变量设置为所返回的最后一行。如果不返回行，变量将保持 SELECT 语句执行前的值。如果子查询程序进行赋值，又没有返回行，值就设置为 NULL。
INTO new_table_name	INTO new_table_name 用返回集中的列建立表格。
<table_or_view>	<table_or_view>是表格或视图的名称。
alias	alias 是列或视图的别名。派生表需要使用别名。它通常比实际名称要短，这样，在 SELECT 语句中使用的两个表格有相同的列名时，更容易区分含义不明的相同列名。
<table_hint>	table_hint 指示 SQL Server 使用优化方法。表

	格提示应放在圆括号内。虽然提供表格提示，但 SQL Server 通常能自己找出最佳优化方式。有效的表格提示包括： PAGLOCK，NOLOCK，ROWLOCK，TABLOCK，TAB -LOCKX，HOLDLOCK，NOLOCK，READCOMMITED ，REPEATABLEREAD，SERIALIZABLE，NOLOCK 和 READPAST。
INDEX={index_name | index_id}	INDEX=表格使用的索引名称。出现成簇索引表示 INDEX=0 使用成簇索引；INDEX=1 进行成簇索引扫描。没有成簇索引时使用 INDEX=0 将进行表格扫描，使用 INDEX=1 会出错。
n	n 表示前面的项可以重复。
FASTFIRSTROW	FASTFIRSTROW 与使用 FAST 1 相同，表示优化程序将尽快返回第一行，并继续生成查询程序中的其他行。
HOLDLOCK	HOLDLOCK 和 SERIALIZABLE 相同，和 FOR BROWSE 子句相互排斥。它包含事务处理期间的共享锁定，而不是释放完成页面的存取。
NOLOCK	NOLOCK 进行无效的读取，且不进行共享或单独的锁定。它可以读取未提交的事务处理，即 NOLOCK 可以读取不一致的数据，并可能引起错误 605，606，624，或 625；这表示应再试着执行一次语句。
PAGLOCK	PAGLOCK 发出共享页面的锁定，而不是共享表格的

	锁定。
READCOMMITTED	READCOMMITTED 通过保存共享锁定来读取数据，且读取数据时，不允许无效读取，而允许修改数据，此时 READCOMMITTED 提供较高的兼容性，这可导致非重复性读取。
READPAST	READPAST 用行级锁定跳行。
READUNCOMMITTED	READUNCOMMITTED 和 NOLOCK 相同。
REPEATABLEREAD	REPEATABLEREAD 用于在查询中所有的数据上设置锁定，但仍允许其他用户插入行，仍能生成幻象行，并允许缺省状态下较少的并发性。
ROWLOCK	ROWLOCK 产生共享行锁定，而不是共享页面或共享表格锁定。
SERIALIZABLE	SERIALIZABLE 与 HOLDLOCK 相同。
TABLOCK	TABLOCK 在表格上产生共享表格锁定，直到语句结束。
TABLOCKX	TABLOCKX 产生专用表格锁定，直到语句或事务处理结束。
UPDLOCK	UPDLOCK 产生更新锁定，直到事务处理或语句结束。
<join_hint>	join_hint 在联结的执行策略方面指导 SQL Server。
{LOOP | HASH | MERGE}	{LOOP | HASH | MERGE}允许通过查询优化程序控制所使用的联结策略，并根据指定的内容执行嵌套的

	循环连接、散列连接或合并连接。
OpenRowset	OpenRowset 定义从数据源中查询远程数据的联接信息。
OpenQuery	OpenQuery 用来执行通过查询。
select_statement	select_statement 是从数据库中返回行的查询程序。
CROSS JOIN	CROSS JOIN 是返回行的两个表格的交叉积，就象没有 WHERE 语句一样。
INNER	INNER 连接是缺省设置，它返回匹配的行。
LEFT [OUTER]	即使 RIGHT 表格中没有对应的行，LEFT [OUTER] 连接，返回左边表格中的所有行;但是 RIGHT 表格中的列为 NULL。
RIGHT [OUTER]	即使 LEFT 表格中没有对应的行，RIGHT [OUTER] 连接，返回右边表格中的所有行;但是 LEFT 表格中的列为 NULL。
FULL [OUTER]	FULL [OUTER] 联接用于返回左边或右边表格中不符合连接条件的行，另一表格中返回为 NULL 的行，以及符合联接标准的其它匹配行。
CONTAINSTABLE	CONSTAINSTABLE 返回精确匹配或不太精确匹配的字符列的结果表。
FREETEXTTABLE	FREETEXTTABLE 返回意义匹配而不是单词匹配的字符列结果表。

<search_conditions>	search_conditions 是查询中限制的条件。
CONTAINS	CONTAINS 搜索精确或者不太精确匹配的字符列。
FREETEXT	FREETEXT 为搜索意义匹配的字符列。
fulltext_table	fulltext_table 是为全文查询标志的表格名或别名。
fulltext_key_column	fulltext_key_column 是为全文查询标志的 fulltext_table 中的全文关键列。
[KEY]	[KEY] 表示选择满足 <contains_search_condition> 的行作为选定范围。
GROUP BY	GROUP BY 将行分成组，这些组在 GROUP BY 列中有相同的值，SELECT 语句还可以为每一组的列总汇包含集合函数。
ALL	ALL 表示即使不符合 WHERE 子句中的搜索条件，也返回所有的组和结果集。对于不符合搜索条件的组，在总汇列中返回 NULL。ALL 必须跟在 GROUP BY 子句后。
CUBE	CUBE 不仅返回 GROUP BY 集合行，而且返回总汇行。这些总汇行在 GROUP BY 子句中的列或表达式的子集上的组合来建立，称为超集合行。
ROLLUP	ROLLUP 表示超集合子集，它和 GROUP BY 子句中通常的集合行一起建立。

HAVING	HAVING 定义 GROUP BY 子句的条件，与 WHERE 子句定义 SELECT 语句中单独行的条件相同。
ORDER BY	ORDER BY 确定 SQL Server 返回的列的排列次序。
select_list_number	select_list_number 是<result_data>中表达式的位置，此数值可以代替 ORDER BY 子句中的表达式。
ASC	ASC(递增)表示 ORDER BY 子句返回的分类数据由小到大排列。
DESC	DESC(递减)表示 ORDER BY 子句返回的分类数据从最大开始排列。
COMPUTE	COMPUTE 在结果集中建立行集合函数的汇总行。COMPUTE 子句用于计算子组合中的总计值，或用于对同一组计算多个集合函数，或在集合函数 SUM，AVG，MIN，MAX 和 COUNT 的组中中断汇总变动。然后就可以查看细节和总计行。使用 COMPUTE BY 子句时必须使用 ORDER BY 子句;可是如果用 COMPUTE 而不用 BY 时，ORDER BY 是可选项。
row_aggregate	row_aggregate 函数有 AVG,COUNT,MAX,MIN 和 SUM。
BY	BY 表示希望行集合函数计算子组。当子组值改变时，将出现中断，集合函数创建为结果集中的新行。使用 BY 表示 ORDER BY 必须和 COMPUTE BY 一起使用。如果 BY 后面有多个表达式，会使集合函数在每一级别上进行汇总。

FOR BROWSE	如果用 DB-Library 在应用程序中查看数据，FOR BROWSE 允许更新。
<query_hint>	<query_hint>指导查询优化程序如何优化查询;然而，用户不指定查询提示，SQL Server 通常知道如何进行优化。
{HASH｜ORDER}GROUP	{HASH｜ORDER}GROUP 指导 SQLServer 去散列或排列 ORDER BY 或 COMPUTE 子句中的聚合。
{MERGE ｜ HASH ｜ CONCAT}UNION	{MERGE｜HASH｜CONCAT}UNION 指导 SQL Server 对 UNION 操作进行合并，散列和链接。
FAST n	FAST n 表示优化程序将尽快返回前 n 行，并继续生成查询中的其余行。
FORCE ORDER	FORCE ORDER 指导 SQL Server 在优化查询过程中保持查询语法中指定的连接顺序。
ROBUST PLAN	ROBUST PLAN 指导 SQL Server 查询优化程序器为可能最大的行容量建立查询计划。

示例

下面是 SELECT 语句的示例：

```
SELECT a.Column1,b.Column2
FROM MyTable a,
YourTable b
```

```
WHERE a.Column3 = b.Column3
ORDER BY a.Column1
```

本例中，执行 INNER JOIN，并对结果集排序。

SESSION_USER

SESSION_USER 是一个 niladic 函数，它提供执行当前对话的用户名称值。
语法

```
SESSION_USER
```

SET @local_variable

SET @local_variable 将局部变量设置为某值。
语法

```
SET
{
   {@local_variable = expression}
```

```
| { @cursor_variable =
{ @cursor_variable
| cursor_name
| { CURSOR
[FORWARD_ONLY | SCROLL]
[STATIC | KEYSET | DYNAMIC]
[READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
FOR select_statement
[FOR {READ ONLY | UPDATE [OF column_list ] } ] }}}}
```

变量

@local_variable	**@local_variable 为一变量，但不是游标变量。**
expression	**expression 是任意的 Microsoft SQL Server 表达式。**
cursor_variable	**cursor_variable 指游标变量的名称。**
cursor_name	**cursor_name 指游标的名称。**
CURSOR	**CURSOR 表示此 SET 语句声明了一个游标。**
SCROLL	**SCROLL 描述游标，表示游标可以进行 FIRST，LAST，NEXT，PRIOR，RELATIVE 和 ABSOLUTE 的滚动。**
FORWARD_ONLY	**FORWARD_ONLY 将游标滚动限制为 FETCH NEXT。**
STATIC	**STATIC 游标用 tempdb 中的临时表格保存游标数**

	据，更新数据库中的数据不为游标读取所注意。
KEYSET	KEYSET 表示打开游标时，游标中的行与顺序固定不变。唯一的键集插入到名为键集的 tempdb 表格中。游标上卷时，可以看到当前用户或其他用户对非键值的更改。但看不到并行用户进行的插入。要读取删去的行将返回@@FETCH_STATUS 的值-2。其他用户对关键列的更新可以是一个插入或删除，但看不到插入。要读取删去的行将返回@@FETCH_STATUS 的值-2。可是使用 WHERE CURRENT OF 语法就能看到新行。
DYNAMIC	对于一个游标，DYNAMIC 可使该游标滚动时能看到所有的数据改变，并且每一次读取都改变游标。即绝对读取和相对读取都不和 DYNAMIC 一起使用。
READ_ONLY	READ_ONLY 定义不能进行更新的只读游标。
SCROLL LOCKS	SCROLL LOCKS 表示读取时锁定游标行，以后进行修改就不会遇到阻碍。
OPTIMISTIC	OPTIMISTIC 表示如果读取行后对行进行了改动，就不能通过游标进行更新或删除。这通过比较时间戳值来完成，如果没有时间戳列，就通过比较检查和来完成。
FOR select_statement	FOR select_statement 是返回游标数据的查询程序。

READ ONLY　　READ ONLY 与 READ_ONLY 一样。
UPDATE [OF column_list]　　UPDATE [OF column_list] 列出可更新的游标列。

示例
下面为 SET 语句的示例：

```
DECLARE Cursor1 CURSOR GLOBAL FOR SELECT *FROM MyTable
DECLARE @Cursor1_variable CURSOR
SET @Cursor1_variable = Cursor1
DEALLOCATE Cursor1 ／* There is now only a LOCAL variable
reference (@my_variable) to the my_cursor
cursor. *／
```

本例中，局部变量的设置参照游标 Cursor1。

SET

SET 语句根据某些设置修改当前对话的行为。

SET ANSI_DEFAULTS

SET ANSI_DEFAULTS 打开或关闭确定 SQL-92 标准状态的 Microsoft SQL Server 设置。

语法

SET ANSI_DEFAULTS { ON | OFF}

SET ANSI_NULL_DFLT_OFF

当用 CREATE 或 ALTER 语句添加列时，SET ANSI_NULL_DFLT_OFF 置 ON 将改变在新列中允许为空的缺省值。如果该设置为 ON，则新列需要某个值，而不是 NULL。

语法

SET ANSI_NULL_DFLT_OFF {ON | OFF}

SET ANSI_NULL_DFLT_ON

当用 CREATE 或 ALTER 语句添加列时，SET ANSI_NULL_DFLT_ON 置 OFF 将改变在新列中允许为空的缺省值。如果此设置为 OFF，则新列需要一个值，而不是 NULL。

语法

```
SET ANSI_NULL_DFLT_ON {ON | OFF}
```

SET ANSI_NULLS

SET ANSI_NULLS 用等号和不等号来计算将 NULL 值与未知值相比较的值。如果某个与 NULL 比较的值为 NULL，将 SET ANSI_NULLS 设置为 OFF 将使该值等于真，否则为假。

语法

```
SET ANSI_NULLS {ON | OFF}
```

SET ANSI_PADDING

建议将 SET ANSI_PADDING 设置为 ON，它决定字符、二进制、可变字符，和可变二进制数据类型如何存储尾部的空格。若将新字符列的这个设置设定为 ON，插入尾部的空格会填满整个列;对于新二进制列，插入尾部的零会填满整个列。对于新可变字符和可变二进制列，不会截短与尾随值一起插入的值，尾随值不会添加至添满整个列。当该设置为 OFF 时，NOT NULL 字符和二进制列会和尾随值一起插入至添满整个列，但是 NULL 字符和二进制列与可变字符和可变二进制相同。在这些情况下，将截去尾部的值，尾部的值不会插入至添满整个新列。

语法

```
SET ANSI_PADDING {ON | OFF}
```

SET ANSI_WARNINGS

如果在集合函数中有 NULL，SET ANSI_WARNINGS 置 ON 就用信号报警。被 0 除和溢出错误会卷回语句并用信息报警。如果新值的长度比字符、Unicode 或二进制列的长度还长，将终止 INSERT 和 UPDATE 语句。

如果集合函数中有 NULL，SET ANSI_WARNINGS 置 OFF 不会用信号报警。被 0 除或溢出错误返回 NULL。如果新值的长度比字符、Unicode 或二进制列的长度

还长，INSERT 语句与 UPDATE 语句将返回 NULL。

语法

```
SET ANSI_WARNINGS {ON | OFF}
```

SET ARITHABORT

如果 SET ARITHABORT 置 ON，被 0 除或溢出错误将导致 Transact-SQL 终止和／或卷回事务处理。

如果 SET ARITHABORT 置 OFF，被 0 除或溢出错误就用信号报警，但是 Transact-SQL 继续执行。

语法

```
SET ARITHABORT {ON | OFF}
```

SET ARITHIGNORE

如果 SET ARITHIGNORE 和 SET ANSI_WARNINGS 都设置为 OFF，对被 0 除和算术溢出错误都不会发出警告。

语法

SET ARITHIGNORE {ON | OFF}

SET CONCAT_NULL_YIELDS_NULL

SET CONCAT_NULL_YIELDS_NULL 置 ON 时，将字符串与 NULL 值并置，会返回 NULL。

SET CONCAT_NULL_YIELDS_NULL 置 OFF 时，将字符串与 NULL 值并置，会返回字符串。

语法

SET CONCAT_NULL_YIELDS_NULL{ON | OFF}

SET CURSOR_CLOSE_ON_COMMIT

SET CURSOR-CLOSE-ON-COMMIT 置 ON 时，将关闭委托上打开的游标。SET CURSOR_CLOSE_ON_COMMIT 置 OFF 时，不关闭委托上打开的游标。

语法

SET CURSOR_CLOSE_ON_COMMIT {ON | OFF}

SET DATEFIRST

SET DATEFIRST 规定一星期中的第一天，其中 1 表示周一，2 表示周二，等等从 1 到 7 的数字。美国英语缺省为 7(周日)。

语法

SET DATEFIRST{ number | @number_var }

变量 number | @number_var 是一个整数，美国英语中缺省为周日。

SET DATEFORMAT

SET DATEFORMAT 决定日期数据类型中日期部分，月、日、年的顺序。

语法

SET DATEFORMAT{ format | @format_var }

变量 format | @format_var 的有效值为 mdy,dmy,ymd,ydm,myd 和 dym，美国英语中缺省为 mdy。

SET DEADLOCK_PRIORITY

SET DEADLOCK_PRIORITY 确定死锁时当前对话中发生的情况。

语法

```
SET DEADLOCK_PRIORITY{LOW | NORMAL | @    deadlock_var   }
```

变量

LOW	LOW 将当前对话选择为死锁牺牲品，并返回事务处理。
NORMAL	NORMAL 允许 SQL Server 根据缺省情形决定死锁牺牲品。
@deadlock_var	@deadlock_var 是一个字符串变量。

SET DISABLE_DEF_CNST_CHK

此设置只为了向后兼容，此版本中它不能置为 ON。

SET FIPS_FLAGGER

SET FIPS_FLAGGER 用基于 SQL-92 标准的 FIPS 127-2 标准控制警告信息的兼容阈级。

语法

```
SET FIPS_FLAGGER '    level    '
```

变量 level 必须是 ENTRY、FULL、INTERMEDIATE 或 OFF。

SET FMTONLY

SET FMTONLY 置 ON 不返回行，只返回元数据。

语法

```
SET FMTONLY{ON | OFF}
```

SET FORCEPLAN

SET FORCEPLAN 指示 SQL Server 按照连接表在 FROM 子句中的顺序进行连接。

语法

SET FORCEPLAN {ON | OFF}

SET IDENTITY_INSERT

SET IDENTITY_INSERT 置 ON 允许将用户定义的值插入到标识列中。

语法

SET IDENTITY_INSERT [database. [owner.]] {table }{ON | OFF}

变量

database	**database 指数据库名称。**
Owner	**owner 指作用表格的拥有者。**
Table	**table 为包含标识列的表格名称。**

SET IMPLICIT_TRANSACTIONS

SET IMPLICIT_TRANSACTIONS 置 On 可将连接设置为隐式事务处理模式，这表示如果连接不在事务处理中，FETCH、DROP、OPEN、DELETE、SELECT、REVOKE、CREATE、ALTER TABLE、TRUNCATE TABLE、GRANT、INSERT、UPDATE 启动事务处理。如果设置为 OFF，连接就处于自动务物处理模式中。

语法

```
SET IMPLICIT_TRANSACTIONS{ON | OFF}
```

SET LANGUAGE

SET LANGUAGE 可根据日期格式和系统信息来决定对话中的语言。

语法

```
SET LANGUAGE{language | @language_var}
```

变量 language | @language_var 是 syslanguage 系统表中的语言名称。此变量必须是 sysname 数据类型。

SET LOCK_TIMEOUT

SET LOCK_TIMEOUT 是过程需要锁定的资源时等待的毫秒数。
语法

```
SET LOCK_TIMEOUT [ timeout_period ]
```

变量 timeout_period 为释放锁定或返回错误前的毫秒数，缺省值-1 表示不等待。

SET NOCOUNT

SET NOCOUNT 置 ON 可使行计数的信息性消息不在结果集中显示。
语法

```
SET NOCOUNT{ON | OFF}
```

SET NOEXEC

SET NOEXEC 置 ON 将不执行后面 SET NOEXEC 置 OFF 前的 Transact-SQL 语句，但是这些语句仍会进行编译。

语法

```
SET NOEXEC {ON | OFF}
```

SET NUMERIC_ROUNDABORT

在表达式中，当圆整到列中的精度比结果的精度低，SET NUMERIC_ROUNDABORT 置 ON 就会出错。

语法

```
SET NUMERIC_ROUNDABORT{ON | OFF}
```

SET OFFSETS

SET OFFSETS 用于 DB-Library 应用程序中，返回与用户定义的语句开头有关的位置，此开头是用逗号分开的 Transact-SQL 关键词列表，这些关键词可以是:SELECT，FROM，ORDER，COMPUTE，TABLE，PROCEDURE，STATEMENT，PARAM 和 EXECUTE。

语法

```
SET OFFSETS     keyword_list
```

变量 keyword_list 是用逗号分开的 Transact-SQL 关键词列表。

SET PARSEONLY

SET PARSEONLY 检查语法但不编译或执行 Transact-SQL 语句。

语法

```
SET PARSEONLY {ON | OFF}
```

SET PROCID

如果使用 SET PROCID ON，在发送存储过程生成的结果集前，要将该存储过程的标识符号发送到 DB_Library 应用程序。

语法

```
SET PROCID {ON | OFF}
```

SET QUERY_GOVERNOR_COST_LIMIT

如果变量为 0，SET QUERY_GOVERNOR_COST_LIMIT 将允许执行所有的查询程序；如果查询值比变量提供的整数值大，就不执行查询。

语法

```
SET QUERY_GOVERNOR_COST_LIMIT    value
```

变量 value 是一整数。

SET QUOTED_IDENTIFIER

如果使用 SET QUOTED_IDENTIFIER ON，就允许标识符名称作为关键词，如果标识符名称放到双引号中，还可以包含专用字符。文字必须放在单引号中。文字中也可能包含单引号，此时，可用两个单引号代替单引号以防出现语法错误。

如果使用 SET QUOTED_IDENTIFIER OFF，标识符名称不能加双引号，而且必须按规则执行，而文字可以放在一对单引号或双引号中，如果在双引号中引用文字，文字中还可以包括单引号。如果在存储过程中使用 SET QUOTED_IDENTIFIER，可以忽略上述情况。用于存储过程的 SET QUOTED_IDENTIFIER 取决于编译时的设置。sp_dboption 存储过程使用引用的标识符变量也会影响该设置。

由于 SET QUOTED_IDENTIFIER 改变了双引号的功能，所以可能引起混乱。因此，最好一直使用单引号定界文字串，使用方括号([,])可获得和 Quoted Identifiers 相同的功能，且不需改变双引号的使用规则。这一点和其他系统存储过程参阅 Chapter 23。

语法

SET QUOTED_IDENTIFIER{ON | OFF}

SET REMOTE_PROC_TRANSACTIONS

如果使用 SET REMOTE_PROC_TRANSACTIONS ON，注意执行远程存储过程时，Microsoft Distributed Transaction Coordinator（MS DTC）会启动分布式事务处理。

如果用 SET REMOTE_PROC_TRANSACTIONS OFF，注意不会启动分布式事务处理。

语法

```
SET REMOTE_PROC_TRANSACTIONS{ON | OFF}
```

SET ROWCOUNT

如果使用 SET ROWCOUNT，并将它设置为特定的数，返回行后，查询将退出处理过程。

语法

```
SET ROWCOUNT {    number | @number_var    }
```

变量 number | @number_var 是一整数。

SET SHOWPLAN_ALL

如果使用 SET SHOWPLAN_ALL ON，将不执行 Transact-SQL 语句。返回的是信息，而不是查询优化程序计划如何执行 Transact-SQL 语句，也不是基于价值的优化程序的查询值。将 SET SHOWPLAN_ALL 写入可格式化输出，其方式与程序扫描和处理输出的方式一致。

语法

```
SET SHOWPLAN_ALL{ON | OFF}
```

SET SHOWPLAN_TEXT

如果使用 SET SHOWPLAN_TEXT ON，就不会执行 Transact-SQL 语句。返回的将是信息，而不是优化程序计划如何执行 Transact-SQL 语句。此版本的显示图样用于人们处理并检查输出。

语法

```
SET SHOWPLAN_TEXT {ON | OFF}
```

SET STATISTICS IO

如果使用 SET STATISTICS IO ON，将显示表格名称、扫描计数、逻辑读取(从数据高速缓存中读取的页数)，物理读取(从磁盘中读取的页数)，和超前读取(放在查询高速缓存中的页数)的信息。

语法

```
SET STATISTICS IO{ON | OFF}
```

SET STATISTICS PROFILE

SET STATISTICS PROFILE 显示 ad hoc 查询、视图、触发器和存储过程的简要信息。

语法

```
SET STATISTICS PROFILE {ON | OFF}
```

SET STATISTICS TIME

如果使用 SET STATISTICS TIME ON，将显示分析、编译和执行 Transact-SQL 语句所用的毫秒数。

语法

```
SET STATISTICS TIME{ON | OFF}
```

SET TEXTSIZE

将 SET TEXTSIZE 置为用户定义的数值，当执行 SELECT 语句时，会将要显示的和 ntext 数据类型的长度设置为该数值。

语法

```
SET TEXTSIZE{number | @number_var}
```

变量 number | @number_var 是最大为 2GB 的整数。

SET TRANSACTION ISOLATION LEVEL

如果使用 SET TRANSACTION ISOLATION LEVEL，并将其设置下面可用选项之一： READ COMMITTED，READ UNCOMMITTED，REPEATABLE READ 或 SERIALIZABLE 缺省的事务处理锁定将变化为当前对话中 SELECT 语句的选定项。

语法

```
SET TRANSACTION ISOLATION LEVEL
{READ COMMITTED | READ UNCOMMITTED | REPEATABLE READ |
SERIALIZABLE}
```

变量

READ COMMITTED	**READ COMMITTED 是存储 SELECT 语句的共享锁定的隔离级别，是 Microsoft SQL Server 的缺省值。此隔离级别预防无效读取，表示没有读取未提交的数据。该隔离级别可能造成非重复性读取的情形，这表示其他对话可以改变数据，如果重读数据，事务处理结束前数据可能已发生改变。此事务处理也可能产生幻象数据，这表示事务处理结束前可以添加新数据。**

READ UNCOMMITTED	READ UNCOMMITTED(也叫 0 隔离级别)允许无效读取。此隔离级别不进行锁定，并忽略其他用户所做的独占锁定，结果造成无效读取，非重复性读取和幻象数据。和使用 NOLOCK 一样，此隔离级别提供的并行性最多，一致性最少。
REPEATABLE READ	REPEATABLE READ 隔离级别通过在当前事务处理所使用的所有数据上设置锁定，来阻止其他用户更新数据，可是，仍可能出现幻象行。此选项可能对并行性有消极影响，使用时应注意，不必要时尽量不用。
SERIALIZABLE	SERIALIZABLE 隔离级别通过在数据上设置一定范围的锁定，来阻止其他用户更新事务处理所使用的数据，或防止插入新行。它提供的并行性最少，一致性最高，但是使用时应注意，因为它会影响其他用户，并对并行性有消极影响，这些方面和使用 HOLDLOCK 相同。

SET XACT_ABORT

如果使用 SET XACT_ABORT ON，当前事务处理将卷回执行期间的错误。

语法

SET XACT_ABORT{ON | OFF}

SETUSER

建议不在 Microsoft SQL Server 中使用 SETUSER，它只向后兼容。SETUSER 函数由系统管理员和数据库的拥有者使用，以便假定其他用户的许可。

语法

SETUSER ['username' [WITH NORESET]]

变量

'username'	**SA 或 DBO 假定许可的安全帐户名。如此变量没有提供为一个参数，假定为最初的等同值。**
WITH NORESET	**WITH NORESET 不需使用所提供的用户名，确定后来发布的 SETUSER 语句，不重置为 SA 或 DBO。**

SHUTDOWN

SHUTDOWN 命令停止 Microsoft SQL Server。

语法

```
SHUTDOWN  [WITH NOWAIT]
```

变量 WITH NOWAIT 在停止 SQL Server 前不完成检验点。正执行的所有事务处理都将卷回。

SIGN

如果表达式为正，SIGN 函数返回 1；如果表达式为负，SIGN 函数返回-1，如果表达式为 0，SIGN 函数返回 0。

语法

```
SIGN(numeric_expression)
```

变量 numeric_expression 可以是小数、浮点数、整型数、货币、数字、实数、短整型数、小货币，或微短整型数。

SIN

系统存储过程 SIN 显示角度的三角正弦值，角度是以弧度表示的浮点表达式。

语法

SIN(float_expression)

变量 float_expression 是浮点数据类型的表达式。

smalldatetime

参阅本章的“ Data Types ”。

smallint

参阅本章的“ Data Types ”。

smallmoney

参阅本章的“Data Types”。

SOME | ANY

SOME 也称作 ANY，如果初始表达式的比较和子查询程序中至少有一个值返回真，SOME 和 ANY 都为真，否则，就返回假。

语法

```
scalar_expression {= | <> | != | > | >= | !> | < | <= | !<}
{SOME | ANY}( subquery )
```

变量

scalar_expression	scalar_expression 是任意有效的 SQL 表达式。
{= | <> | != | > | >= | !> | < | <= | !<}	这些是可用于比较操作的 Microsoft SQL Server 运算符。
SOME | ANY	SOME 或 ANY 为关键词，表示要进行比较。
Subquery	用于此处的 SUBQUERY 只能返回和变量列表中 scalar_expression 的数据类型相同的一列。

SOUNDEX

SOUNDEX 用于确定两个字符串的发音是否相似，并返回 4 个字符的代码。代码的第一字节是一个字符，表示变量的第一个字母。其余的代码包括数字。元音忽略不计，除非元音位于字符串的开头。字母 y，双字母和字母 h 也不考虑。相关的函数有 DIFFERENCE 函数。参阅本章的“DIFFERENCE”。

语法

```
SOUNDEX(character_expression)
```

变量 character_expression 是一个字符串，它可以是变量、常数或表格中的列。

示例

下面是 SOUNDEX 函数的示例：

```
SELECT SOUNDEX('FOREX'), SOUNDEX('FORX')
```

本例中，'FOREX' 和 'FORX' 的返回值都为 F620，因为它们发音非常相似。

SPACE

SPACE返回一串重复的空格，然而，如果要在Unicode数据中包含空格，用REPLICATE函数代替SPACE。

语法

```
SPACE(integer_expression)
```

变量integer_expression为一整数。如果integer_expression为负数，此函数返回NULL。

SQUARE

SQUARE为一函数，它将浮点数据类型的值与其自身相乘。

语法

```
SQUARE(float_expression)
```

变量float_expression是一个浮点型数据。

SQRT

SQRT 是返回浮点型数据的平方根的函数。
语法

SQRT(float_expression)
变量 float_expression 是一个浮点型数据。

STATS_DATE

STATS_DATE 是返回最后一次更新索引的统计日期及时间的函数。
语法

STATS_DATE(table_id, index_id)

变量

table_id	**table_id 是表格的 OBJECT_ID。**
index_id	**index_id 是索引的 indid。**

STDEV

STDEV 是一个返回标准偏差的集合函数。它与其他集合函数(如 SUM 函数)大体相同，只适用于数值型数据类型。在所提供的列中，会排除输入数据集中的 NULL 值。

语法

```
STDEV(expression)
```

变量 expression 可以是一个常量、列名或函数，也可以包含运算符。

示例

下面是 STDEV 的示例：

```
SELECT STDEV(MyNumericColumn)
FROM MyTable
```

本例中，返回 MyTable 表的 MyNumericColumn 列中所有数值的标准偏差。

STDEVP

STDEVP 是返回人口标准偏差的集合函数。该函数与其他集合函数（如 SUM

函数）大体相同，只适用于数值型数据，并忽略 NULL 值。

语法

```
STDEVP(expression)
```

变量 expression 可以是一个常量、列名或函数，也可以包含运算符。

示例

下面是 STDEVP 函数的示例：

```
SELECT STDEVP(MyNumericColumn)
FROM MyTable
```

本例中，返回 MyTable 表的 MyTableColumn 列中所有数值的人口标准偏差。

STR

STR 可将数值型数据转换为字符型数据，并返回字符型数据。它提供的功能比 CAST 函数更广泛，因为它允许控制小数的格式。如果长度不够则返回星号。

语法

```
STR(float_expression [, length [, decimal]] )
```

变量

float_expression	float_expression 是浮点型表达式，可以包含小数点。
Length	length 是想要的字符串的总长度。如果没有提供这个可选变量部分，缺省值为 10。
Decimal	decimal 是小数点右边的位数。

STUFF

STUFF 函数可以将字符表达式的一部分用所提供的另一个字符串替代，替代部分用起始位置和长度来定义。

语法

STUFF(character_expression, start, length, character_expression)

变量

character_expression	character_expression 可以是字符或二进制型常量、变量或列。
Start	start 是定义要替换的字符表达式的起始位置的整数。如该变量为负值或大于字符表达式的总长，则返回 NULL。

Length	length 是决定要替换的字符数的整数。

示例

下面是 STUFF 函数的示例：

```
SELECT STUFF(' My dog Fido ', 8, 4, ' Sally ')
GO
```

本例中，从第 8 个字符起，删去 4 个字符“ Fido”，并在原位置处插入 5 个字符“ Sally”。这正是 STUFF 名称的由来。如果愿意，替换的字符串的字节数可以多于、等于或少于被替换的字符串的字节数。

SUBSTRING

SUBSTRING 函数返回字符型、二进制型、文本或图形表达式的一部分。

语法

```
SUBSTRING(expression, start, length)
```

变量

expression	expression 可以是字符串、二进制串、文本、图形或列名;但不是集合函数。

Start	start 是表明开始截取表达式的位置的整数。
Length	length 是决定表达式截取长度的整数。

示例

下面是 SUBSTRING 函数的示例：

```
SELECT SUBSTRING('Microsoft',6,4)
```

本例中，返回表达式 Microsoft 中的 4 个字符“soft”。

SUM

SUM 是一个集合函数，它返回表格的选定行中所有选定的数值型列中的值的总和，或只返回表格的选定行中所有选定的数值型列中的 DISTINCT 值。表格的选定行中一个选定的数值型列中有 Null 值，表示排除该行。

语法

```
SUM( [ALL | DISTINCT]      expression    )
```

变量

All	All 关键字 DISTINCT 关键字相区别，它是缺省值，表示对所有选定的值求和。

DISTINCT	DISTINCT 关键字用来确定只对独有值求和。
Expression	expression 是任意有效的、返回数值的 SQL Server 表达式，但不能是集合函数和子查询程序。

SUSER_ID

SUSER_ID 函数返回 SQL Server 用户的标识号。它只用于向后兼容。7.0 版本中，人物编码用 SUSER_SID 代替。

语法

```
SUSER_ID( [ 'login' ] )
```

变量 login 是可选项。它是登录 ID 的名称。如果没有提供该变量，则默认为当前用户。

SUSER_NAME

SUSER_NAME 函数返回 SQL Server 用户的登录 ID 名，它只用于向后兼容。7.0 版本中的人物编码用 SUSER_SNAME 代替。

语法

```
SUSER_NAME( [ server_user_id ] )
```

变量 server_user_id 是可选项，它是登录 ID 号。如果没有提供该变量，则默认为当前用户。

SUSER_SID

SUSER_SID 返回用户所提供的 Microsoft SQL Server 安全帐户名所对应的安全标识号（也叫 SID)。

语法

```
SUSER_SID( [ 'login' ] )
```

变量 login 是可选的，它是安全帐户名称。如果没有提供该变量，则默认为当前用户。

SUSER_SNAME

SUSER_SNAME 从用户的安全标识号（也称 SID）中返回 Microsoft SQL Server 安全帐户名。

语法

```
SUSER_SNAME( [ server_user_sid ] )
```

变量 server_user_id 是可选项，它是安全标识号。如果没有提供该变量，默认为当前用户。

SYSTEM_USER

如果 NT Authentication 用来验证用户，SYSTEM_USER 函数就提供 NT 帐户名的系统提供值，或者如果使用 SQL Server Authentication，SYSTEM_USER 函数就提供 SQL Server 已验证的安全帐户名的系统提供值。这个函数可以用作常规函数，如果不存在其他缺省值，它就是插入表格的缺省值;或用于 DEFAULT CONSTRAINT 中。

语法

```
SYSTEM_USER
```

TAN

TAN 函数返回浮点变量的正切值。

语法

```
TAN(float_expression)
```

变量 float_expression 是一个角度，且是浮点或实型数据。

text

参看本章 Data Type 部分。

TEXTPTR

TEXTPTR 函数用来指向第一个文本页，并以变长二进制型数据返回。
语法

TEXTPTR(column)

变量 column 是列的名称。

TEXTVALID

TEXTVALID 函数用于文本、ntext 或图像列,以检查文本指针的有效性。如果有效，则返回 1;无效则返回 0。UPDATETEXT、WRITETEXT 和 READTEXT 都依赖于这一函数的使用。
语法

TEXTVALID('table.column', text_ptr)

变量

table	**table 是表格名称。**
Column	**column 是列名称。**
text_ptr	**text_ptr 是有效化的文本指针。**

timestamp

timestamp 是列的数据类型，在每个表格中只有一个 timestamp 列，它包含二进制(8)型值 (如果允许 NULL 值存在，则可包含 varbinary(8)型值)，该值在数据库中是唯一的。每次插入或更新行时，都要更新该列。

tinyint

参看本书“ Data Type”部分。

TRIGGER_NESTLEVEL

TRIGGER_NESTLEVEL返回表格中对应UPDATE、INSERT或DELETE操作的所执行的触发器数。

语法

```
TRIGGER_NESTLEVEL( [    object_id    ] )
```

变量object-id是触发器的ID，如果未提供该变量，返回的整数表示UPDATE、INSERT或DELETE操作中所有触发器被激活的次数。

TRUNCATE TABLE

TRUNCATE TABLE语句删除表格中所有未注册的行，但不激活触发器。

语法

```
TRUNCATE TABLE    name
```

变量name是要截取的表名。

TYPEPROPERTY

TYPEPROPERTY 函数以字符型或二进制型数据的属性形式返回信息的类型。

语法

TYPEPROPERTY (type, property)

变量

type	**type 数据类型**
property	**property 可以是以下几种：** **precision(数字位数或字符数)** **Scale(小数位数)** **AllowsNull(0 = FALSE, 1=TRUE)** **UsesAnsiTrim(0 = FALSE, 1 = TRUE)**

UNICODE

UNICODE 函数返回变量第一个字节的 Unicode 标准整数值。

语法

UNICODE(' ncharacter_expression ')

变量 ncharacter-expression 是 nchar 或 nvarchar 型数据。

UNION

UNION 语句可以提取多个 SELECT 语句的结果，并合并这些结果。

语法

```
select_statement
UNION [ALL]
select_statement
 [UNION [ALL] select_statement] [,...n]
```

变量

select_statement	select_statement 是返回数据的查询语句。
UNION	UNION 合并多个查询的结果集。
ALL	ALL 返回包括复本在内的所有行。但如果未提供此关键字，则不返回复本。
N	表明前述项目是可重复的。

示例

下面是 UNION 语句的示例：

```
SELECT FirstName, LastName
```

```
FROM MyFirstTable
UNION
SELECT FirstName, LastName
FROM MySecondTable
```

本例中，合并了 MyFirstTable 的结果与 MySecondTable 的结果集，并返回一个数据集。

uniqueidentifier

参看本章“Data Type”部分。

UPDATE

UPDATE 语句通过更改表格中已有的行来修改表中的数据。
语法

```
UPDATE {<table_or_view>}
SET
{   column_name    = {   expression     | DEFAULT}
 |     @variable     =     expression    }  [,...   n   ]
```

[FROM
{
<table_or_view>
| (select_statement) [AS] table_alias [(column_alias [,...
m])]
| <table_or_view> CROSS JOIN <table_or_view>
| INNER [<join_hints>] JOIN
<table_or_view> ON <join_condition>
| <rowset_function>
} [,... n]
]
[WHERE
<search_conditions>
| CURRENT OF
{ { [GLOBAL] cursor_name } | cursor_variable_name } }
]
[OPTION (<query_hints>, [,... n])]
<table_or_view> :: =
{ table_name [[AS] table_alias] [<table_hints > [... m]]

| view_name [[AS] table_alias]

```
}
<table_hints> ::=
{ INDEX = {    index_name    |    index_id    } [,...    n    ]
  |  FASTFIRSTROW
  |  READPAST
  |  { HOLDLOCK | PAGLOCK | READCOMMITTED
  |  READUNCOMMITTED | REPEATABLEREAD
  |  ROWLOCK | SERIALIZABLE | TABLOCK | TABLOCKX
}
<table_hints> ::=
{ INDEX(    index_name    |    index_id    )
  |  FASTFIRSTROW
  |  HOLDLOCK
  |  PAGLOCK
  |  READCOMMITTED
  |  READPAST
  |  READUNCOMMITTED
  |  REPEATABLEREAD
  |  ROWLOCK
  |  SERIALIZABLE
  |  TABLOCK
  |  TABLOCKX}
```

```
<join_hints> ::=
{ HASH | LOOP | MERGE }

<query_hints> :: =
{ { HASH | ORDER } GROUP
 | { CONCAT | HASH | MERGE } UNION
 |  FAST     number_rows
 |  FORCE ORDER
 |  ROBUST PLAN}
<join_condition> :: =
{    table_name    |    table_alias    |    view_name    }.   column_name

<logical_operator>
{    table_name    |    table_alias    |    view_name    }.   column_name

<logical_operator>:: =
{=|> | <|>=|<=|<>| !=| !<|!>}
<rowset_function>::=
{OPENQUERY (    linked_server, ' query '    )
 |  OPENROWSET
(    ' provider_name ' ,
{   ' datasource ' ;' user_id ' ;' password '    |    ' provider_string '    },
```

```
{    [catalog.] [schema.] object_name    |    'query'    })}
<search_conditions> ::=
{  [NOT]  <predicate>  [  { AND | OR } [NOT]  <predicate>]
}  [, ...  n  ]
<predicate> ::=
{
   expression    { = | <> | != | > | >= | !> | < | <= | !< }    expression
 |    string_expression    [NOT] LIKE    string_expression
 [ESCAPE    'escape_character'   ]
 |    expression    [NOT] BETWEEN    expression    AND    expression

 |     expression    IS  [NOT]  NULL
 |    expression     [NOT]  IN (   subquery | expression     [,...   n   ] )
 |     expression    { = | <> | != | > | >= | !> | < | <= | !< }
{ALL | SOME | ANY} (   subquery   )
 |  EXISTS (   subquery   )}
```

变量

<table_or_view> **table_name | view_name**	**table_or_view 是表或视图名的标题。** **table_name | view_name 是数据被更新的表**

	或视图的名称。只能更新视图中的一个表。
table_alias	alias 是表或视图的简短名称。
column_alias	column_alias 是列标题的别名。
<table_hints>	table_hints 指示 SQL Server 查询优化器如何进行查询。参看本书第 6 章“新优化程序提示”。
INDEX(index_name | index_id)	如果给出了索引优化器提示，该索引用以处理查询。
m	m 表明列的别名可重复。
OPENQUERY	OPENQUERY 运行通过查询。参看本章“OPENQUERY”。
OPENROWSET	OPENROWSET 包含连接的定义以使用数据源中的远程数据。
SET	SET 关键字后面的列是要更新的列。
column_name	column_name 是格式为 form 的用逗号分隔开的列表,column_name=表达式。
Expression	是任何有效的 SQL 表达式。
DEFAULT	DEFAULT 将某列中的值设置成由 DEFAULT 约束为该列定义的缺省值。
@variable	已声明的局部变量，设置为表达式返回的值。
n	n 表示前述各项可重复。

FROM	FROM 确定用来为 UPDATE 提供值的表格。
CROSS JOIN	CROSS JOIN 是返回行的两个表格的交叉积。就好象没有 WHERE 子句一样。
INNER	INNER 连接是缺省值，返回匹配的行。
<join_hints>	<join_hints>指示 SQL Server 连接的运行策略。
ON<join_condition>	ON<join_condition>是连接条件。
WHERE<search_conditions>	WHERE<search_conditions>是查询中的限制条件。
CURRENT OF	CURRENT OF 是游标的当前位置，指向修改后的行。
GLOBAL	GLOBAL 表明游标是一个全局游标。
cursor_name	cursor_name 是决定可更新的数据集的游标名。
cursor_variable-name	cursor_variable-name 是游标变量的名称。
OPTION(<query_hints> , <,...n>)	OPTION(<query_hints>，<,...n>)指示查询优化器如何优化查询，然而，通常无须用户指定查询提示，SQL Server 也能自动进行优化。
{HASH | ORDER}GROUP	{HASH | ORDER}GROUP 指示 SQL Server 在 ORDER BY 或 COMPUTE 子句中对集合运算使

	用散列或排序。
{MERGE | HASH | CONCAT}UNION	{MERGE | HASH | CONCAT}UNION 指示 SQL Server 为 UNION 操作使用合并、散列或连接。
FAST n	FAST n(number_of_rows)表示在查询中，优化器将尽快返回前 n 行，之后继续返回余下的各行。
FORCE ORDER	FORCE ORDER 指示 SQL Server 在查询优化过程中保持查询语法中指定的连接顺序。
ROBUST PLAN	ROBUST PLAN 指示 SQL Server 查询优化器为可能的最多行数建立查询计划。

示例

下面是 UPDATE 语句的示例：

```
UPDATE MyTable
SET Column1 = b.Column1
FROM MyTable a,
MyTable2 b
WHERE a.MyId = b.MyId
```

在本例中，若 MyTable 和 MyTable 2 表的 MyId 列中各行具有相匹配的值，

则表 MyTable 中的 Column1 中的值更新为 MyTable2 的 Column1 中的值。

UPDATE STATISTICS

UPDATE STATISTICS 语句更新与表格索引相关的系统表中的信息。这些信息是 SQL Server 用以建立规划的列值的分布式数据。这些数据的大部分发生变更后，就应更新这些数据。另外，用户也可以利用自动统计功能完成这一步（通过 sp_autostats），而无需执行更新统计。

语法

```
UPDATE STATISTICS {table} [index | (index_or_column [, ...n] )]
 [WITH [[FULLSCAN] | SAMPLE number {PERCENT | ROWS}]]
 [[,] [ALL | COLUMNS | INDEX] [[,] NORECOMPUTE]]
```

变量

table	table 是表格名称。
Index	index 是索引名称。
index_or_column	index_or_column 是要更新统计的索引或列的名称。索引或列的名称必须循标识符的规则。只有设定了 INDEX 或 COLUMN 选项，才需要 index_or_column 变量。

N	表示前述各项可重复。
FULLSCAN	FULLSCAN 对表格的全部内容进行浏览。，而不是对表格取样。
SAMPLE number{PERCENT | ROWS}	SAMPLE number{PERCENT | ROWS}表示表格中行的百分比或取样的行数。如果给定百分比或数目太小，显示效果不佳，则 Microsoft SQL Server 会更改数字。
ALL | COLUMNS | INDEX	ALL | COLUMNS | INDEX 确定列统计式、数字、索引统计数字或所有的统计数字是否会受到影响，缺省值是只有索引会受到影响。
NORECOMPUTE	NORECOMPUTE 禁止 Microsoft SQL Server 自动重建统计，禁止自动重建统计。执行 sp_autostats 系统存储过程，或运行不带 NORECOMPUTE 的 UPDATE STATISTICS 可恢复统计的自动计算。

UPDATETEXT

UPDATETEXT 用来更新文本、ntext 或图像列的一部分。WRITETEXT 用来替代整个文本、ntext 或图像列。

语法

```
UPDATETEXT {table_name.dest_column_name dest_text_ptr}
{NULL | insert_offset} {NULL | delete_length}
 [WITH LOG]
 [inserted_data | [{ table_name.src_column_name src_text_ptr}]
```

变量

table_name.dest_column_name	**table_name.dest_column_name 是表和文本、ntext 或图像列的名称。**
dest_text_ptr	**dest_text_ptr 由 TETXPTR 函数返回，是要更新的数据的指针。**
insert_offset	**insert_offset 是新数据插入的起始点。如果第一个字符的偏置为 0，第二个字符的偏置就为 1，依次类推。如果列是文本或图像列，insert_offset 表示在查找插入数据的点之前，要从列首开始传送的字节数。如果该列是 ntext 型数据，insert_offset 表示字符数，而不是字节数，因为 ntext 型数据占两个字节。如果提供的是 0，数据就插入到开头，并将 NULL 追加到数据的末尾。**
delete_length	**delete_length 是要删除的字符数。如果**

	列是文本或图像列，delete_length 表示在查找要删除的数据的点之前，要从 insert_offset 的位置开始传送的字节数。如果该列是 ntext 型数据，delete_length 表示字符数，而不是字节数，因为 ntext 型数据占两个字节。如果提供的是 0，就删除从 insert_offset 到数据的末尾的所有数据。
WITH LOG	WITH LOG 指定记录 UPDATETEXT 操作，但快速增加记录文件的尺寸。如果没有提供 WITH LOG，就必须打开 select into／bulkcopy database 选项，该选项可以用系统存储过程 sp_dboption 打开。请查阅 23 章的系统存储过程。
inserted_data	inserted_data 表示要在 insert_offset 处将数据类型为文本、ntext 或图像的数据插入到列中。
table_name.src_column_name	数据类型为文本、ntext 或图像的限定表的列名，其中包含要插入的数据。
src_text_ptr	用 TEXTPTR 函数获取包含要插入的数据的文本、ntext 或图像列的指针。

示例

下面是 UPDATETEXT 语句的示例：

```
EXEC sp_dboption ' MyDB ', ' select into／bulkcopy ', ' true '
GO
DECLAER @pointervar binary(16)
SELECT@pointervar = TEXTPTR(MyTable)
FROM MyTable a, MyTable2 b
WHERE a.MyID= b.MyID
UPDATETEXT MyTable.MyColumn @pointervar 56 3 ' COM '
GO
EXEC sp_dboption ' MyDB ', ' select into／bulkcopy ', ' false '
GO
```

本例中，TEXTPTR 放在一个局部变量中，更新了 MyColumn。

UPPER

UPPER 函数用大写字符替代小写字符。

语法

UPPER（character_expression）

变量 character_expression 是字符或二进制常数，变量。

USE

USE 确定要使用的数据库。
语法

```
USE{database}
```

变量 database 是数据库名。

USER

该语句与 USER_NAME 相同。

USER_ID

USER_ID 是系统函数，它返回用户的数据库用户标识号。
语法

```
USER_ID ([   'user'   ])
```

变量 user 是用户名称。

USER_NAME

USER_NAME 函数在提供了用户的标识号时返回数据库的用户名。

语法

```
USER_NAME ([ id ])
```

变量 id 是用户的标识号。

VAR

VAR 是集合函数，它返回数值列或表达式中所有值的变体。如果列中的值为 NULL，该列就排除在外。

语法

```
VAR(expression)
```

变量 expression 是一个常数、列名或函数，它可以包含运算符，但不能包含集合函数和子查询程序。

varbinary

请查阅本章的“Data Type”。

varchar

请查阅本章的“Data Type”。

VARP

VARP 是一个集合函数，它返回数值列或表达式中所有值的个数变体。如果列中的值为 NULL，该列就排除在外。

语法

VARP(expression)

变量 expression 是一个常数、列名或函数，它可以包含运算符，但不能包含集合函数和子查询程序。

WAITFOR

WAITFOR 语句允许在继续进行处理前等待一段时间。

语法

```
WAITFOR {DELAY    'time'    |TIME    'time'    }
```

变量

DELAY	DELAY 是一个关键字，后跟要等待的时间，该时间的最大值为 24 小时。
'time'	time 的格式为 hh:mm:ss，可以使用局部变量。
TIME	TIME 是一个关键字，后跟要等到的时刻。

WHERE

WHERE 子句在用于 SELECT，UPDATE，INSERT 或 DELETE 语句时确定要检索的行。

语法

```
WHERE<search_conditions>
```

变量<search_conditions>定义语句所影响的行的限制条件。

WHILE

WHILE 为 sql_statement 或语句块的重复执行设置条件。只要特定的条件为真，这些语句就一直重复执行。WHILE 循环中语句的执行可由循环内的 BREAK 或 CONTINUE 关键字来控制。

语法

```
WHILE Boolean_expression
{sql_statement |statement_block}
[BREAK]
{sql_statement |statement_block}
[CONTINUE]
```

变量

Boolean_expression	Boolean_expression 返回 TRUE 或 FALSE。
{sql_statement|statement_block}	{sql_statement |statement_block} 是 Transact-SQL 语句，该语句带有使用

	BEGIN 和 END 语法的块。
BREAK	BREAK 从 WHILE 循环中退出。
CONTINUE	CONTINUE 跳过 CONTINUE 后、WHILE 循环末尾前的语句，从 WHILE 循环的开头执行。

WRITETEXT

WRITETEXT 允许已有的文本、ntext 或图像列的不记录的交互更新。该语句完全覆盖了它所操作的列中的任何已有数据。WRITETEXT 不能用于视图中的文本、ntext 或图像列。在缺省状态下，不记录 WRITETEXT 语句，因此，事务处理记录不会被由这些数据类型组成的大量数据所填满。

语法

```
WRITETEXT { table.column text_ptr }
[WITH LOG] { data }
```

变量

table.column	table.column 是文本、ntext 或图像列的限定表名。
text_ptr	用 TEXTPTR 函数获取包含要写入的数据的文本、

	ntext 或图像列的指针。
WITH LOG	WITH LOG 指定记录 WRITETEXT 操作，但记录文件的尺寸会迅速增大。如果没有提供 WITH LOG，就必须打开 select into／bulkcopy database 选项，该选项可以用系统存储程序 sp_dboption 打开。请查阅第 23 章的系统存储过程。
Data	data 是要写入数据库的文本、ntext 或图像型数据。这些数据最大为 120KB。

示例

下面是 WRITETEXT 语句的示例：

```
EXEC sp_dboption ' MyDB ', ' select into／bulkcopy ', ' true '
GO
DECLARE @pointervar binary(16)
SELECT@pointervar = TEXTPTR(MyTable)
FROM MyTable a, MyTable2 b
WHERE a.MyID= b.MyID
WRITETEXT MyTable.MyColumn @pointervar  ' These types of columns
are used to hold long text ntext or image data and can hold up to 120
kilobytes at a time.'
GO
EXEC sp_dboption ' MyDB ', ' select into／bulkcopy ', ' false '
```

GO

在本例中，TEXTPTR 放在一个局部变量中，并得到一个值。

YEAR

YEAR 函数返回表示所给定日期的年的整数。

语法

YEAR(data)

变量 data 是日期表达式。