网 络 技 术 精 通 系 列

精 通 Web 数 据 库

金磊　康晓宁　邓就庆　等　编著

人 民 邮 电 出 版 社

需要全本请访问 www.ertongbook.com

第 10 章　Transaction-SQL 与基本操作

10.1　数据类型

10.1.1　预定义数据类型

SQL Server 为使用者预定了很多数据类型，因此绝大多数的需求用这些系统已经预定好的类型就已经足够了。所有的预定义数据类型列于表 10.1 中。

表 10.1　　预定义数据类型

类型类别	类型名称	说　明
整数类型	int	从-2147483648 到 2147483647 的整数。Int 使用 32 个 bit
	smallint	从-32768 到 32767。Smallint 使用的是 16 个 bit
	tinyint	从 0 到 255。Tinyint 使用的是 8 个 bit
	bit	0 或者 1
固定精度数字	decimal	从-10^38 -1 到 10^38 -1 的固定精度的数字
	numeric	同 decimal
货币类型	money	从-922337203685477.5808 到 922337203685477.5807 的带 4 位小数精度的数字
	smallmoney	从-214748.3648 到 214748.3647 的带 4 位小数精度的数字
浮点类型	float	从-1.79E + 308 到 1.79E + 308 的浮点数字
	real	从-3.40E + 38 到 3.40E + 38 的浮点数字
时间类型	datetime	从 1753 年 1 月 1 日到 9999 年 12 月 31 日，可以表示的精度为千分之三秒
	smalldatetime	从 1900 年 1 月 1 日到 2079 年 6 月 6 日，可以表示的精度是一分钟
数字类型	cursor	游标指针
	timestamp	数据库系统内唯一的时间戳
	uniqueidentifier	唯一标志符号
字符串类型	char	定长字符串，只能使用非 Unicode 类型的数据，最大长度为 8000 个字符
	varchar	变长字符串，只能使用非 Unicode 类型的数据，最大长度为 8000 个字符
	text	变长字符串，只能使用非 Unicode 类型的数据，最大长度为 2147483647 个字符

续表

类型类别	类型名称	说　明
Unicode 字符串类型	nchar	定长字符串，使用 Unicode 类型的数据，最大长度为 4000 个字符
	nvarchar	变长字符串，使用 Unicode 类型的数据，最大长度为 4000 个字符
	ntext	变长字符串，只能使用非 Unicode 类型的数据，最大长度为 1073741823 个字符
二进制数据类型	binary	固定长度的二进制数据，最大长度为 8000 个字节
	varbinary	长度可变的二进制数据，最大长度为 8000 个字节
	image	长度可变的二进制数据，最大长度为 2147483647 个字节

1．整数类型

int、smallint、tinyint、bit 都属于整数类型，但是它们占用的字节大小不一样。int 占用 4 个字节，smallint 用两个字节，tinyint 用一个字节。bit 的存储方式和其他整数类型的方式不太一样，当有 1 至 8 个 bit 类型的数据时，它们共同占用一个字节，当有 9 到 16 个 bit 类型的数据时，它们占用 2 个字节，依此类推。正因为这种压缩的存储结构，所以 SQL Server 不能对这种 bit 类型的数据列做索引。另外，int 类型的标准和 SQL-92 中 integer 类型的标准是完全一样的。

2．固定精度数字类型

decimal 类型要指定数字的精度。指定精度的格式为：

decimal[(p[,s])]

p 指定在十进制数字中，最大能记录多少位。这个位的个数是小数点左边的整数部分的位数加上小数点右边的小数部分的位数之和。p 必须是个大于 0 的整数，而且在一般情况下，p 不应该超过 28，当 SQL Server 是带着参数 p 启动的时候，p 的范围可以扩大到 38。

s 指定在小数点右边最多能保留几位小数。s 也必须是大于等于 0 的整数，且它不能大于 p。

例如，如果要在数据库中保存 1234.1234 这个数字，那么为了完整地保留精度，至少要用 decimal(8,4)才行。

通过固定精度数字的结构可以看出，固定精度数字占用空间的大小决定于参数 p，如表 10.2 所示。

表 10.2　　固定精度数字类型占用的空间

P 的值	整个数字占用的字节数
1～9	5
10～19	9
20～28	13
29～38	17

在 SQL-92 标准中，decimal 被表示为 dec。

3. 货币类型

货币类型主要是为表示大数量的货币以及货币要求的精度而设置的，实际上，它与固定精度的数字类型没有本质的区别。money 占用 8 个字节的容量，而 smallmoney 占用 4 个字节的容量。

4. 浮点类型

使用 float 可以指定该数字的尾数，指定的格式为

float[(n)]

其中 n 就表示该数字的尾数可以占用多少个 bit。n 的范围必须是从 1 到 53。SQL Server 的这种 float[(n)]的格式可以适应 SQL-92 标准的浮点数的任何数据格式。例如 SQL-92 标准中的 double precision 类型就是 float(53)。

Real 类型也是一种浮点数，它可以表示从–3.40E + 38 到 3.40E + 38 的浮点数字，其实它就是 float(24)。

关于 float 的精度和占用的空间，可见表 10.3。

表 10.3 浮点数的精度及占用的空间

n	精度	占用的字节数
1～24	7	4
25～53	15	8

5. 时间类型

由于 datetime 的精度是千分之三秒，所以 datetime 的最小增量是 3 毫秒。也就是说 SQL Server 只能用 0、3、7 来表示时间的毫秒位。对于其他的毫秒位上的数字，例如 1、2、9 等，系统都会用一个四舍五入的办法来把它们近似到 0、3、7 这三个数字上，四舍五入的对应关系如表 10.4 所示。

表 10.4 毫秒的四舍五入

实际时间	四舍五入后的时间
1998-01-01 23:59:59.999	1998-01-02 00:00:00.000
1998-01-01 23:59:59.995	1998-01-01 23:59:59.997
1998-01-01 23:59:59.996	
1998-01-01 23:59:59.997	
1998-01-01 23:59:59.998	
1998-01-01 23:59:59.992	1998-01-01 23:59:59.993
1998-01-01 23:59:59.993	
1998-01-01 23:59:59.994	
1998-01-01 23:59:59.990	1998-01-01 23:59:59.990
1998-01-01 23:59:59.991	

Datatime 在数据库内部其实是用了两个 4 字节长的整数来表示的。第一个 4 字节长的数字表示当前日期与一个标准日期之间差的天数，第二个字节表示从当天午夜以后到现在经过了多少毫秒。所以 datatime 共占用 8 个字节。

Smalldatetime 在数据库内部用是两个 2 字节的整数来表示的。第一个 2 字节长的数字也表示当前日期是与标准日期所相差的天数，第二个字节表示从当天午夜以后到现在经过了多少秒。所以 smalldatetime 共占用 4 个字节。

6. 特殊数字类型

cursor 是指向一个游标的指针，它可以是一个用在存储过程或触发器中的变量，但不能是一张表或者视图中的数据类型。任何被声明为 cursor 类型的数据都可以为空。

timestamp 是个时间戳。一个表中只能有一列数据是 timestamp 类型的。当一个表中的某一条数据被插入或者修改的时候。SQL Server 会自动更新这个时间戳。这个时间戳的信息在数据库中是唯一的。它占用 8 个字节的空间。虽然时间戳是唯一的，但是这种类型的列却不适合做键，尤其是主键。因为数据行上的任何的修改都会导致时间戳的更新，从而导致键的位置的变化。最明显的就是聚簇索引，当主键的值发生改变的时候，不光是索引要重新调整，而且数据实际存储的位置都要变化。

Uniqueidentifier 是一个唯一标志。为了避免重复，不能由使用者随便指定 uniqueindentifier 的值。使用者要么用 NEWID 函数生成一个新的唯一标志，要么从一个十六进制转化过来。例如 6F9619FF-8B86-D011-B42D-00C04FC964FF 就是一个 uniqueidentifier 的值。使用唯一标志的时候，可以用逻辑比较符号，例如=、<>或者 IS NULL、NOT NULL，但是这种类型的数据不能进行排序比较，如>、<等。

7. 字符串类型

char 是定长度字符串类型，使用者可以指定该类型的最大长度。最大长度的范围是从 1 到 8000。由于 char（n）是定长的，所以就算向 char（n）中存储很少几个字符，SQL Server 也要使用 n 个字节的空间，n 个字节中空余的部分将用空格填满。

varchar 是变长度字符串类型。使用者也可以指定该类型的最大长度。最大长度的范围是从 1 到 8000。在 SQL Server 中，变长类型的实际存储格式是先在数据前面将实际数据的长度记录下来，然后才是实际存储的数据。每次访问数据的时候，先读出数据的实际长度，然后按照这个实际长度将后面的数据读出。这样的存储结构可以节省很多空间，让数据更紧凑，但也使数据行的长度各不相同而导致存入与读取数据操作的复杂度增加。

text 的存储结构与 varchar 不同，但它也是可变长的字符串类型。它的最大长度可以达到 2^30- 1 (1,073,741,823)个字符。

8. Unicode 字符串类型

nchar、nvarchar、ntext 分别对应普通字符串类型的 char、varchar、text。它们的区别主要是 nchar、nvarchar 和 ntext 可以支持 unicode。unicode 实际上是用两个字节记录一个字符以实现多语言的支持。所以 nchar、nvarchar 和 ntext 的最大长度都是普通对应字符串类型的一半，这里所说的长度指的是字符的个数，而不是占用的字节数。nchar 最多可以记录 4000 个字符，nvarchar 也是一样。由于 unicode 的一个字符占用两个字节，所以 nchar、nvarchar、ntext 的最大长度所占用的空间分别与 char、varchar、ntext 没什么区别。

需要全本请访问 www.ertongbook.com

9. 二进制数据类型

binary、varbinary 类型与 char、varchar 分别对应，唯一不同的一点是 binary、varbinary 存储的是二进制的数据，而 char、varchar 存储的是字符类型的数据。

Image 的存储结构与 varbinary 不同，但它也是可变长的二进制数据类型。它的最大长度可以达到 2^30- 1 (1,073,741,823)个字符。

10. 大容量数据类型

text、ntext、image 都是大容量的数据类型，它们的存储方式与普通的数据类型有很大的不同。简单地说，就是大容量数据类型都不和普通的数据放置在一起。关于大容量数据的存储方式，在上一章中的 SQL Server 7.0 体系结构一节中有详细的介绍。

10.1.2 自定义数据类型

SQL Server 7.0 允许使用者定义自己的数据类型。自定义数据类型只能由数据库本身提供的预定义数据类型组成，并指定该类型的数据是否可以为空。

使用自定义类型的最大好处是可以将数据的类型统一。例如有一个数据库中有很多表中都用到了身份证号码，于是我们就可以用 varchar(18)定义一个身份证号码的类型，以后在定义表的时候，只要用这个自定义身份证号码类型就可以了，而不必每次都要指定 varchar（18）。

1. 创建

创建新的自定义数据类型的方法有两种，一种是直接执行 Transaction-SQL，另一种是用 SQL Server7.0 提供的 Enterprise Manager 图形化管理工具。

使用 Transaction-SQL 定义数据类型的指令格式如下：

```
sp_addtype [@typename=] type,
        [@phystype=]  system_data_type
      [,[@nulltype=]  'null_type']
```

注意：Transaction-SQL 的指令的格式中，中括号内的部分都是可选参数。

各个参数的说明列于表 10.5。

表 10.5　　sp_addtype 参数说明

参数名称	说明
Type	指定自定义类型的类型名称，这个名称要在系统中唯一
System_data_type	系统预定义数据类型，指明自定义数据类型的实际类型。这个实际类型可以为除了 cursor 和 timestamp 之外的任何一种数据类型。当数据类型中需要指定长度的时候，数据类型外面要用单引号括起来。如'varchar(20) '
null_type	指定该种自定义数据类型是否可以为空。可以填入的值为'NOT NULL'、'NULL'、'NONULL'

例如我们要在数据库中增加两种数据类型。一种数据类型是表示电话的，该类型不允许空，另一种数据类型是表示传真的，该类型可以为空。那么 T-SQL 语言如下：

```
EXEC sp_addtype tel, 'varchar(20) ', 'NOT NULL'
EXEC sp_addtype fax, 'varchar(24) ', 'NULL'
```

于是在运行这两条语句的数据库中将会增加两种数据类型。在 Enterprise Manager 中也可以做同样的事情。操作步骤如下：

（1）启动 Enterprise Manager，登录到 SQL Server 服务器，选择数据库。

（2）打开该数据库的对象列表，选择 User Defined Data Types。如图 10.1 所示。

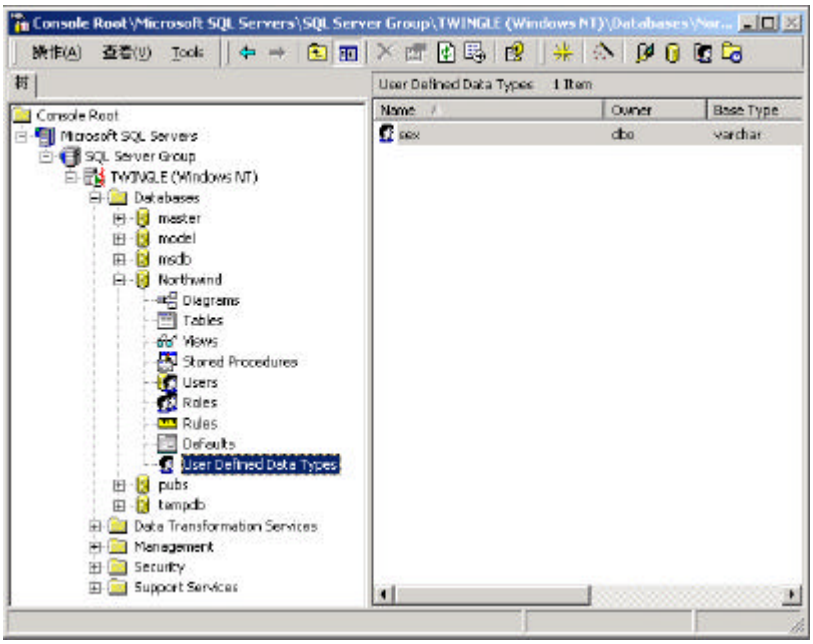

图 10.1　选择 User Defined Data Types

（3）在 User Defined Data Types 上点击鼠标右键，弹出菜单，选择 New User Defined Data Type，如图 10.2 所示。



图 10.2　User Defined Data Types 子菜单

（4）弹出对话框如图 10.3 所示。在 Name 输入框中输入要定义的类型的名称。在 Data type 输入框中选择预定义的类型。如果所选定的数据类型需要填写长度，则在 Length 中填入相应的数字。选中 Allow NULLs 则定义的类型允许出现 NULL 值。

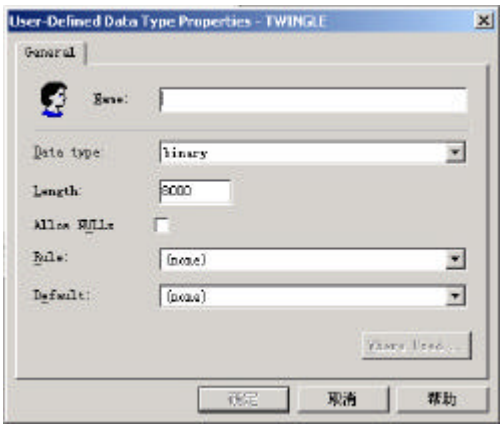

图 10.3 User Defined Data Types 对话框

2. 删除

删除一个自定义的类型也很简单。在 T-SQL 中，使用 sp_droptype 即可。但是如果这个类型正在被某些表使用的话，就无法删除这个自定义的数据类型。

使用 T-SQL 语言删除电话和传真自定义数据类型的例子如下：

```
EXEC sp_droptype tel
EXEC sp_droptype fax
```

在 Enterprise manager 中，首先选择要被删除的自定义数据类型，然后点击鼠标右键，弹出菜单，如图 10.4 所示，选择“删除”。

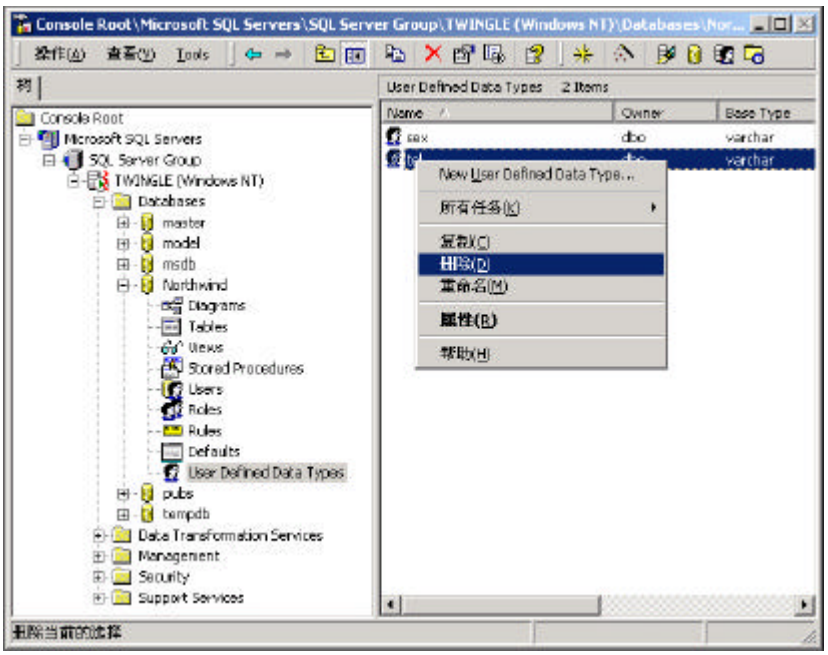

图 10.4 删除自定义数据类型

3. 重命名

在 Enterprise manager 中，重命名的步骤和删除差不多，在图 10.4 的菜单中选择“重命名”，然后直接修改自定义类型的名称即可。

使用 T-SQL 也可以修改自定义类型的名称。语法如下：

```
sp_rename [@objname =] 'object_name',
        [@newname =] 'new_name'
```

其中 object_name 是自定义数据类型原先的名称，而 new_name 是要被修改成的名称。例如要把名为 tel 的自定义数据类型改名为 telephone，则

```
EXEC sp_rename 'tel', 'telephone'
```

需要注意的是，当修改了自定义的数据类型以后，所有使用这个数据类型的表的定义也随之修改了。例如，我们原先有一个表中的 home_tel 列是 tel 类型的，那么当上面的语句执行后，home_tel 的数据类型就是“telephone”了。

sp_rename 不但可以更改自定义数据类型的名称，也可以修改所有数据库中对象的名称，关于 sp_rename 的其他用法，我们将在以后提到。

10.1.3　数据类型间的转换

SQL Server 7.0 提供两个函数来进行数据类型的转换。它们是 CAST 和 CONVERT。

1. CAST

CAST 的使用方法为：

```
CAST (expression as data_type)
```

expression 是一个合法的表达式，也就是要被转换的数据。data_type 是要被转换成的数据类型。下面举几个例子来说明 CAST 的具体用法和注意事项。

在 SQL Server 7.0 中有一个样本数据库 pub，其中表 titles 的部分情况如下：

title	price
The Busy Executive's Database Guide	19.9900
Cooking with Computers: Surreptitious Balance Sheets	11.9500
You Can Combat Computer Stress!	2.9900
Straight Talk About Computers	19.9900
Silicon Valley Gastronomic Treats	19.9900
The Gourmet Microwave	2.9900
The Psychology of Computer Cooking	NULL
But Is It User Friendly?	22.9500
Secrets of Silicon Valley	20.0000
Net Etiquette	NULL
Computer Phobic AND Non-Phobic Individuals: Behavior Variations	21.5900
Is Anger the Enemy?	10.9500
Life Without Fear	7.0000

Prolonged Data Deprivation: Four Case Studies	19.9900
Emotional Security: A New Algorithm	7.9900
Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean	20.9500
Fifty Years in Buckingham Palace Kitchens	11.9500
Sushi, Anyone?	14.9900

【例 1】把 money 类型转换成字符串类型

```
SELECT 'This price is ' + CAST(price AS varchar(10))    title
FROM titles where
WHERE price>10
```

结果为：

```
title
------------------------
This price is 19.99
This price is 11.95
This price is 19.99
This price is 19.99
This price is 22.95
This price is 20.00
This price is 21.59
This price is 10.95
This price is 19.99
This price is 20.95
This price is 11.95
This price is 14.99
```

首先，由于 price 中的数字都是 money 类型的，如果我们不使用 CAST(price as varchar(10)) 的话，那么 SQL Server 不能把两种类型的数据做连接运算，也就是不能把“This price is”与 price 连成一个字符串输出。其次，原始数据中的 price 都保留了四位小数，而使用 CAST 把 price 转化成字符串类型，就会自动保留两位小数。

另外，如果转换成的字符串类型长度无法完整地表达数字的话，SQL Server 系统会用星号来代替或者干脆根本不执行。

【例 2】money 类型转换成的字符串类型长度不够，系统报错

```
SELECT 'This price is ' + CAST(price as varchar(3)) as title
FROM titles where
WHERE price>10
```

结果为：

```
There is insufficient result space to convert a money value to varchar.
```

【例 3】整数转换成的字符串类型长度不够，转换成星号

```
SELECT 'This price is ' + CAST (CAST(price as int) AS char(1))   title
FROM titles
WHERE price>10
```

结果为：

```
title
---------------
This price is *
This price is *
This price is *
This price is *
This price is *
This price is *
This price is *
This price is *
This price is *
This price is *
This price is *
This price is *
```

【例 3】实际上做了两步转换，第一次是把 money 类型转换成整数类型，第二次是把整数类型转换成字符串类型。在第二次转换中，由于字符串的最大长度只有 1，所以 SQL Server 不得不把结果换成星号来输出。【例 4】具体说明了浮点数和货币型数字转换成整数类型的情况。

【例 4】货币型数字转化成整数。

```
SELECT CAST(price as int) price
FROM titles
WHERE price>10
```

结果为：

```
price
-----------
20
12
20
20
23
20
22
11
20
```


需要全本请访问 www.ertongbook.com

21

12

15

对比上原始数据可以看出，在进行数据类型转换的时候，SQL Server 先会对浮点数字或货币型数字进行四舍五入的操作，再取其整数部分做为结果。

2. CONVERT

CONVERT 的使用方法为：

CONVERT(data_type,expression[,style])

其中，参数 data_type、expression 与 CAST 函数中的完全相同。Style 是一个可选参数，它将规定输入和输出数据的样式。这个样式一般只在时间类型(datetime、smalldatetime)与字符串类型（nchar、nvarchar、char、varchar）相互转换的时候才被用到。

例如，我们这样可以获得当前的时间

```
SELECT getdate()
```

结果为

```
now
---------------------------
2000-09-14 21:31:54.547
```

但是，这个结果只是适用于部分国家的人，而其他国家的时间文字习惯可能并不是这个样子。例如英语国家的人习惯将月份放在最前面，日子放在中间，最后才是年份。另外，如果我们的用户只想知道今天的日期，而不需要知道几点几分几秒，那怎么办呢？这就需要指定 style。style 是一个数字。如

```
SELECT CONVERT(varchar(30),getdate(),1) now
```

结果为

```
now
------------------------------
09/14/00
```

在这个例子里面，分隔符变成了“ / ”，月份被排在了最前面，中间是日子，最后是年份，年份用了两位数而不是四位数。要想把年份显示成 4 个数字的，一般需要把 style 加上 100。如：

```
SELECT CONVERT(varchar(30),getdate(),101) now
```

结果为

```
now
------------------------------
09/14/2000
```

style 的具体含义如表 10.6 所列。

表 10.6　　style 数字在转换时间时的含义

Style（两位表示年）	Style（四位表示年）	输入输出样式
-	0 or 100	mon dd yyyy hh:miAM (或 PM)
1	101	mm/dd/yy
2	102	yy.mm.dd
3	103	dd/mm/yy
4	104	dd.mm.yy
5	105	dd-mm-yy
6	106	dd mon yy
7	107	mon dd, yy
8	108	hh:mm:ss
-	9 or 109	mon dd yyyy hh:mi:ss:mmmAM (或 PM)
10	110	mm-dd-yy
11	111	yy/mm/dd
12	112	yymmdd
-	13 or 113	dd mon yyyy hh:mm:ss:mmm(24 小时制)
14	114	hh:mi:ss:mmm(24 小时制)
-	20 or 120	yyyy-mm-dd hh:mi:ss(24 小时制)
-	21 or 121	yyyy-mm-dd hh:mi:ss.mmm(24 小时制)

Style 不但在时间类型与字符串类型之间转换时可以使用，也可以在浮点、货币类型与字符串类型之间转换时使用。

浮点与字符串类型之间的转换的例子如下：

```
SELECT CONVERT(char(50),CAST (1203342.12342225 AS float),2) floattochar
```

结果为

```
floattochar
----------------------------------------------------
1.203342123422250e+006
```

style 为 2 的含义表示保留 16 位有效数字，同时用科学计数法表示。关于 style 在浮点数与字符串转换时的作用，可以参见表 10.7。

表 10.7　　style 在浮点数与字符串转换时的作用

Style 的值	说明
0	不指定 style 时的缺省设置。系统最多用 6 位有效数字表示一个浮点数，若需要时，可能用科学计数法表示
1	有效数字为 8 位，并且用科学计数法表示
2	有效数字为 16 位，并且用科学计数法表示

货币类型与字符串类型之间的转换的例子如下：

```
SELECT CONVERT(varchar(50),CAST (1203342.12342225 AS money),1) moneytochar
moneytochar
----------------------------------------------------
1,203,342.12
```

style **为** 2 **的含义是在小数点后面保留** 2 **位小数，并且在整数部分每隔** 3 **位添加一个逗号。关于** style **在货币类型与字符串类型之间的转换时的作用，可以参见表** 10.8**。**

表 10.8　　style **在货币类型与字符串类型转换时的作用**

Style 的值	说　明
0	**不指定** style **时的缺省设置。不使用逗号分割整数部分，同时只保留** 2 **位小数。例如** 1234567.23
1	**使用逗号分割整数部分，同时保留** 2 **位小数。例如** 123,456.34
2	**不使用逗号分割整数部分，同时保留** 4 **位小数。例如** 123456.3456

10.2　流程控制

10.2.1　BEGIN…END **结构**

BEGIN…END **的语法结构为：**

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

其中 sql_statement|statement_block **指的是一条或多条** SQL **语句。**

例如：

```
IF (SELECT COUNT(*)
      FROM titles
WHERE price>10)>0
--如果选择出来的数据记录条数大于0
BEGIN
  PRINT 'There is a book'
  PRINT 'Not very cheap'
```

```
    END
    ELSE
        BEGIN
      PRINT 'There is a book'
      PRINT 'Not very cheap'
    END
```

BEGIN…END **的主要作用是将** SQL **语句分成块，以方便流程控制。另外，**BEGIN…END **结构可以进行嵌套。**

10.2.2　IF…ELSE…

IF…ELSE…**的语法结构为：**

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

Boolean_expression **为一个布尔表达式。当这个表达式的值为真的时候，系统执行** IF **后面的语句块，当表达式的值为假的时候，系统执行** ELSE **后面的语句块。当然，**ELSE **也可以不存在，那么当表达式为假的时候，就跳过** IF **后面的语句或语句快而继续执行下面的语句。**

sql_statement|statement_block **表示一条语句或者一个语句块。当语句块中超过一条语句的时候，需要用** BEGIN…END **套起来以组成一个语句块。**

IF…ELSE…**结构可以嵌套使用。例如：**

```
    IF (SELECT COUNT(*)
        FROM titles
    WHERE price>10)>0
    BEGIN
      IF (SELECT COUNT(*)
        FROM titles WHERE price>20)>0
          PRINT 'There is a very very expensive book'
       ELSE
        BEGIN
      PRINT 'There is a book'
          PRINT 'Not very cheap'
            END
    END
    ELSE
        BEGIN
      PRINT 'There is a book'
```

```
    PRINT 'Not very cheap'
END
```

10.2.3 CASE

CASE…WHEN…**也是一种分支结构，与** IF…ELSE…**的两分支结构不同的是，它是多分支结构。它的语法结构如下：**

```
CASE input_expression
    WHEN when_expression THEN result_expression
         [...n]
    [
         ELSE else_result_expression
    ]
END
```

或

```
CASE
    WHEN Boolean_expression THEN result_expression
         [...n]
    [
         ELSE else_result_expression
    ]
END
```

…n **指的是可以把** WHEN…THEN…**的结构重复多次。**

ELSE **是可选的参数，当使用这个参数的时候，如果所有的** WHEN **都没有匹配，则执行** ELSE **后面的** else_result_expression**。**

第一种结构只能进行等值比较，也就是 SQL Server **把** input_expression **和每一个** when_expression **进行比较，如果相等，整个** CASE **语句的值就是** result_expression.

第二种结构比较灵活，它判断每个 boolean_expression **的真假。这个** boolean_expression **可以是个等值表达式，也可以是比较大小的表达式。如果这个表达式是真，那么整个** CASE **语句的值就是** result_expression.

第一种结构的例子如下，这个例子把按书的类型进行统计，同时利用 CASE **语句把类型的简称转换成类型的全称。**

```
Psychology          Prolonged Data Deprivatio      19.9900
Psychology          Computer Phobic AND Non-P      21.5900
                                                   avg
                                                   ==========
                                                   13.5040

Traditional Cooking Fifty Years in Buckingham      11.9500
```

```
Traditional Cooking Sushi, Anyone?                    14.9900
Traditional Cooking Onions, Leeks, and Garlic         20.9500
                                                SELECT    Category =
        CASE type
            WHEN 'popular_comp' THEN 'Popular Computing'
            WHEN 'mod_cook' THEN 'Modern Cooking'
            WHEN 'business' THEN 'Business'
             WHEN 'psychology' THEN 'Psychology'
            WHEN 'trad_cook' THEN 'Traditional Cooking'
            ELSE 'Not yet categorized'
        END,
    CAST(title AS varchar(25)) AS 'Shortened Title',
     price AS Price
FROM titles
WHERE price IS NOT NULL
ORDER BY type, price
COMPUTE AVG(price) BY type
```

结果为：

```
Category             Shortened Title             Price
-------------------  -------------------------   ---------------------
Business             You Can Combat Computer     2.9900
Business             Cooking with Computers:     11.9500
Business             The Busy Executive's Data   19.9900
Business             Straight Talk About Compu   19.9900
                                                 avg
                                                 ====================
                                                 13.7300


Modern Cooking       The Gourmet Microwave       2.9900
Modern Cooking       Silicon Valley Gastronomi         19.9900
                                                 avg
                                                 ====================
                                                 11.4900

Popular Computing    Secrets of Silicon Valley   20.0000
Popular Computing    But Is It User Friendly?    22.9500
                                                 avg
                                                 ====================
```


需要全本请访问 www.ertongbook.com