第 9 章 Transact-SQL 语言

SQL Server 在支持标准 SQL 语言的同时，对其进行了扩充，引入了 T-SQL，即 Transact-SQL。通过 T-SQL，可以定义变量、使用流控制语句、自定义函数和存储过程，极大地扩展了了 SQL Server 的功能。

本章的学习要点：

* T-SQL 的编程基础

*流程控制语句

*系统自定义函数

9.1 Transact-SQL 语言

Transact-SQL（简称为T-SQL）语言是Microsoft公司在关系型数据库管理系统SQL Server中实现的一种计算机高级语言，是微软对 SQL 的扩展。T-SQL 语言具有 SQL 的主要特点，同时增加了变量、运算符、函数、流程控制和注释等语言元素，使得其功能更加强大。T-SQL 语言对 SQL Server 十分重要，SQL Server 中使用图形界面能够完成的所有功能，都可以利用 T-SQL 语言来实现。使用 T-SQL 语言操作时，与 SQL Server 通信的所有应用程序都通过向服务器发送 T-SQL 语句来进行，而与应用程序的界面无关。

T-SQL 语言既允许用户直接查询存储在数据库中的数据，也可以把语句嵌入到某种高级程序设计语言中。在关系型数据库管理系统中，T-SQL 可以实现数据的检索、操纵和添加等功能，它同其它程序设计语言一样，有自己的数据类型、表达式和流程控制语句等。

9.2 标识符、数据类型

9.2.1 语法约定

表 9- 1 列出了 T-SQL 的语法约定，并进行了说明。本书相关描述遵循这些约定。

表 9- 1　SQL 语法约定

约定	用于
大写	T-SQL 关键字。
斜体	用户提供的 T-SQL 语法的参数。
粗体	数据库名、表名、列名、索引名、存储过程、实用工具、数据类型名以及必须按所显示的原样键入的文本。
<u>下划线</u>	指示当语句中省略了包含带下划线的值的子句时应用的默认值。
|（竖线）	分隔括号或大括号中的语法项。只能使用其中一项。
[ ]（方括号）	可选语法项。不要键入方括号。
{ }（大括号）	必选语法项。不要键入大括号。

[,...*n*]	指示前面的项可以重复 *n* 次。各项之间以逗号分隔。

续

约定	用于
<label> ::=	语法块的名称。此约定用于对可在语句中的多个位置使用的过长语法段或语法单元进行分组和标记。可使用语法块的每个位置由括在尖括号内的标签指示：<标签>。

9.2.2 注释语句

注释语句不是可执行语句，不参与程序的编译、注释语句通常是用来说明代码的功能或者对代码的实现方式给出简单的解释和提示。

在 T-SQL 中可使用两类注释符：

1. “--”用于单行注释；
1. “/*　*/”用于多行注释。“/*”用于注释文字的开头，“*/”用于注释文字的结束。

9.2.3 标识符

数据库对象的名称即为其标识符。SQL Server 中的所有内容都可以有标识符，比如服务器、数据库和数据库对象（例如表、视图、列、索引、触发器、过程、约束及规则等）的名称都是标识符。大多数对象要求必须有标识符，但对有些对象（例如约束），标识符是可选的。对象标识符是在定义对象时创建的，随后标识符用于引用该对象。

1. 标识符的分类

SQL Server 有两类标识符：

1）常规标识符：符合标识符的格式规则。在 T-SQL 语句中使用常规标识符时不用将其分隔开。

常规标识符的定义规则如下：

1） 名称的长度可以从 1 到 128（对于本地临时表，标识符最多可以有 116 个字符）。

1） 名称的第一个字符必须是一个字母或者“_”、“@”和“#”中的任意字符。

1） 在中文版 SQL Server 中，可以直接使用中文名称。

1） 名称中不能有空格

1） 不允许使用 SQL Server 的保留字。

说明：	● 在 SQL Server 中，标识符中拉丁字母的大小写等效。 ● 在 SQL Server 中，某些位于标识符开头位置的符号具有特殊意义。 ✧ 以 @ 符号开头的常规标识符始终表示局部变量或参数，并且不能用作任何其他类型的对象的名称。 ✧ 以一个数字符号(#)开头的标识符表示临时表或过程。以两个数字符号 (##) 开头的标识符表示全局临时对象。 ✧ 某些 Transact-SQL 函数的名称以(@@) 开头。为了避免与这些函数混淆，不应使用以 @@ 开头的名称。

2）分隔标识符：在 T-SQL 中，不符合常规标识符定义规则的标识符必须用分隔符双引号 (") 或者方括号 ([ ])分隔，称为分隔标识符。例如：Select * from [my table]中，因为“my table”中间含有空格，不符合常规标识符的定义规则，因此必须用分隔符 ([ ])进行分隔。

2. 标识符的使用

用户在引用对象时，需要给出对象的名字。可用两种方式来引用对象：完全限定对象名和部分限定对象名。

完全限定对象名由 4 个标识符组成：服务器名称、数据库名称、所有者名称和对象名称，每部分用“.”隔开。完全限定对象名通常用于分布式查询或远程存储过程调用。

其语法格式为：

```
[[[server.][database].][owner_name].] object_name
```

服务器、数据库和所有者的名称称为对象名称限定符。例如，“jiaowu.student.dbo.studentinfo”用来标识数据库中的表对象“studentinfo”。其中，“jiaowu”、“student”和“dbo”分别表示服务器名称、数据库名称和所有者名称。

通常，当引用一个对象时，不需要指定服务器、数据库和所有者，而用句号标出它们的位置，这种为部分限定对象名。SQL Server 可根据系统当前的工作环境确定部分限定对象名中省略的部分，使用以下的默认值：

1）server：本地服务器；

2）database：当前数据库；

3）owner_name：在数据库中与当前连接会话登录标识相关联的数据库用户名或数据库所有者（dbo）。

例如，需要引用 customer 数据库中 employee 表的 telephone 列，可指定 customer..employee.telephone。

需要注意的是，在使用时，服务器名和数据库名常被省略，默认为本地服务器和当前数据库；如果当前用户就是拥有者，那么，owner_name 也可以被省略。

9.2.3 数据类型

在 SQL Server 中，每个列、局部变量、表达式和参数都具有一个相关的数据类型。 数据类型是一种属性，用于指定对象可保存的数据的类型：整数数据、字符数据、货币数据、日期和时间数据、二进制字符串等。

常用的数据类型介绍如下：

1. 整数数据类型

使用整数数据的精确数字数据类型。具体见表 9- 2。其中，int 数据类型是 SQL Server 中的主要整数数据类型。

表 9- 2 整数数据类型

数据类型	范围	存储
bigint	-2^{63} 到 $2^{63}-1$	8 字节
int	-2^{31} 到 $2^{31}-1$	4 字节
smallint	-2^{15} 到 $2^{15}-1$	2 字节
tinyint	0 到 255	1 字节

2. 带固定精度和小数位数的数值数据类型

主要有 decimal[(p[,s])]和 numeric[(p[,s])]，有效值从 $-10^{38}+1$ 到 $10^{38}-1$。numeric 在功能上等价于 decimal。

（1）p 代表精度，指最多可以存储的十进制数字的总位数，包括小数点左边和右边的位数。该精度必须是从 1 到最大精度 38 之间的值。默认精度为 18。

（2）s 代表小数位数，指小数点右边可以存储的十进制数字的最大位数。小数位数必须是从 0 到 p 之间的值。仅在指定精度后才可以指定小数位数。默认的小数位数为 0；因此，$0 \le s \le p$。

3. 日期时间数据类型

SQL Server 的常用日期时间类型有 datatime 和 smalldatetime。

在 SQL Server 2008 中，日期时间类型的最大转换就是在这两种数据类型的基础上又引入了 date、time、datetime2 和 datetimeoffset 类型。其中，可将 datetime2 视作现有 datetime 类型的扩展，其数据范围更大，默认的小数精度更高，并具有可选的用户定义的精度。datetimeoffset 数据类型具有时区偏移量，是指定某个 time 或 datetime 值相对于 UTC（世界标准时间）的时区偏移量。时区偏移量可以表示为 [+|-] hh:mm，具体介绍见表 9- 3。

表 9- 3　日期时间数据类型

数据类型	格式及范围	精度	存储长度	举例
date	YYYY-MM-DD，范围从 0001 年 1 月 1 号 到 9999 年 12 月 31 号	1 天	3 字节	2007-05-08
time	hh:mm:ss[.nnnnnnn]，其中，[.nnnnnnn] 是 0 到 7 位数字，范围为 0 到 9999999，它表示秒的小数部分。	默认的精度是 100ns。	5 字节	12:35:29.1234567
datetime	日期范围 1753 年 1 月 1 日 到 9999 年 12 月 31 号；时间范围 00:00:00.000 到 23:59:59.999	默认的小数部分为一个 0 到 3 位的数字，范围为 0 到 999。	8 字节	2007-05-08 12:35:29.123
smalldatetime	1900 年 1 月 1 日 到 2079 年 12 月 31 号	1 分钟	4 字节	2007-05-08 12:35:00
datetime2	YYYY-MM-DD hh:mm:ss[.nnnnnnn], 范围从 0001 年 1 月 1 号 到 9999 年 12 月 31 号,[.nnnnnnn] 代表 0 到 7 位数字，范围从 0 到 9999999，它表示秒小数部分。	准确度为 100ns。	最多 8 字节	2007-05-08 12:35:29.1234567
datetimeoffset	YYYY-MM-DD hh:mm:ss[.nnnnnnn] [{+|-}hh:mm]，与 datetime 范围相同	100 ns	默认值为 10 个字节的固定大小	2007-05-08 12:35:29.1234567 +12:15

说明：　因为 SQL Server 中有专门的处理日期时间类型的函数，所以建议凡是日期时间的数据最好用日期时间的数据类型。

4. 字符数据类型（非 Unicode 字符数据）

1） char[(n)]：长度为 n 个字节的固定长度字符数据。n 的取值范围为 1 至 8,000。

char 数据类型是一种长度固定的数据类型。

如果插入值的长度比 char 列的长度小，将在值的右边填补空格直到达到列的长度。例如，如果某列定义为 char(10)，而要存储的数据是“music”，则 SQL Server 将数据存储为“music_____”，这里“_”表示空格。

如果要存储的数据比允许的字符数多，则数据就会被截断。例如，如果某列被定义为 char(10)，现要将值“This is a really long character string”存储到该列，则 SQL Server 将把该字符串截断为“This is a”。

2） varchar[(n|max)]：可变长度的非 Unicode 字符数据。n 的取值范围为 1 至 8,000。max 指示最大存储大小是 2^{31}-1 个字节。

varchar 数据类型是一种长度可变的数据类型，存储长度为实际数值长度。

varchar 数据可以有两种形式：

（1） 指定最大字符长度。例如，varchar(6) 表示此数据类型最多存储六位字符；

（1） varchar(max)形式。此数据类型可存储的最大存储大小是 2^{31}-1 个字节。

说明：	使用 char 或 varchar 的建议： 如果列数据项的大小一致，则使用 char； 如果列数据项的大小差异相当大，则使用 varchar。

3）text：服务器代码页中长度可变的非 Unicode 数据，最大长度为 231-1 个字符。

5. Unicode 字符数据类型（双字节数据类型）

Unicode（统一码）是一种在计算机上使用的字符编码，它为每种语言中的每个字符设定了统一且唯一的二进制编码，以满足跨语言、跨平台进行文本转换、处理的要求。由于它的设计中涵盖世界上所有语言的全部字符，因此不需要不同的代码页来处理不同的字符集。支持国际化客户端的数据库应始终使用 Unicode 数据，SQL Server 支持 Unicode 标准 3.2 版。SQL Server 使用 UCS-2 编码方案存储 Unicode 数据。在这一机制下，所有 Unicode 字符均用 2 个字节存储。

字符数据类型有 nchar 长度固定，nvarchar 长度可变的 Unicode 字符数据类型。

1） nchar[(n)]：n 个字符的固定长度的 Unicode 字符数据。n 值必须在 1 到 4,000 之间（含）。

2）nvarchar[(n|max)] ：可变长度 Unicode 字符数据。n 值在 1 到 4,000 之间（含）。max 指示最大存储大小为 2^{31}-1 字节。

说明：	SQL Server 将所有文字系统目录数据存储在包含 Unicode 数据类型的列中。 数据库对象（如表、视图和存储过程）的名称存储在 Unicode 列中。

6. 二进制数据类型

二进制数据类型存储的是由 0、1 组成的文件。表 9- 4 中列出了 SQL Server 支持的二进制数据库类型。

表 9- 4 二进制数据类型

数据类型	说明
BINARY(n)	存储固定大小的二进制数据，n 从 1 到 8000
VARBINARY(n)	存储可变大小的二进制数据，n 从 1 到 8000
VARBINARY(max)	存储可变大小的二进制数据，最大为 2^{31}-1 字节
IMAGE	存储可变大小的二进制数据，最大为 2^{31}-1 字节

说明：	在 Microsoft SQL Server 的未来版本中将删除 **ntext**、**text** 和 **image** 数据类型，改用 nvarchar(max)、varchar(max) 和 varbinary(max)。

9.3 常量和变量

9.3.1 常量

常量，也称为文字值或标量值，是表示一个特定数据值的符号。常量的格式取决于它所

表示的值的数据类型。

1. 字符串常量

字符串常量括在单引号内。空字符串用中间没有任何字符的两个单引号表示，以下是字符串的示例：'Cincinnati'，'Process X is 50% complete.'。

说明：	● 如果将 QUOTED_IDENTIFIER 选项设置成 OFF，则字符串也可以使用双引号括起来，但 SQL Server 中 SET QUOTED_IDENTIFIER ON 为默认设置。当 SET QUOTED_IDENTIFIER 为 ON 时，标识符可以由双引号分隔，而文字必须由单引号分隔，不允许用双括号括住字符串常量。 ● 如果单引号中的字符串包含一个嵌入的引号，可以使用两个单引号表示嵌入的单引号，如 'I''m ZYT'。对于嵌入在双引号中的字符串则没有必要这样做。

2. Unicode 字符串

Unicode 字符串的格式与普通字符串相似，但它前面有一个 N 标识符（N 代表 SQL-92 标准中的区域语言）。N 前缀必须是大写字母。例如，'Michel' 是字符串常量而 N'Michel' 则是 Unicode 常量。对于字符数据，存储 Unicode 数据时每个字符使用 2 个字节，而不是每个字符 1 个字节。

3. 二进制常量

二进制常量具有前辍 0x 并且是十六进制数字字符串。这些常量不使用引号括起。

下面是二进制字符串的示例：0xAE。 空二进制常量：0x 。

4. datetime 常量

datetime 常量使用特定格式的字符日期值来表示，并被单引号括起来。

下面是 datetime 常量的示例：

'December 5, 1985'； '5 December, 1985'； '12/5/98'

'851205'（其中的 0 不能省略）

下面是时间常量的示例：'14:30:24'； '04:24 PM'

5. Integer（int） 常量

int 常量以没有用引号括起来并且不包含小数点的数字字符串来表示。integer 常量必须全部为数字且不能包含小数。integer 常量的示例：1894。

6. decimal 常量

decimal 常量由没有用引号括起来并且包含小数点的数字字符串来表示。decimal 常量的示例：1894.1204。

7. float 和 real 常量

float 和 real 常量使用科学记数法来表示。float 或 real 值的示例：101.5E5。

8. money 常量

money 常量以前缀为可选的小数点和可选的货币符号的数字字符串来表示。money 常量不使用引号括起。money 常量的示例：$542023.14，￥30。

9. 全局唯一标识符

全局唯一标识符（Globally Unique Identification Numbers, GUID）是 16 字节长的二进制数据类型，是 SQL Server 根据计算机网络适配器地址和主机时钟产生的唯一号码生成的全局唯一标识符。例如：6F9619FF-8B86-D011-B42D-00C04FC964FF 即为有效的 GUID 值。

世界上的任何两台计算机都不会生成重复的 GUID 值。GUID 主要用于在拥有多个节点、多台计算机的网络或系统中，分配必须具有唯一性的标识符。在 Windows 平台上，GUID 应用非常广泛：注册表、类及接口标识、数据库、甚至自动生成的机器名、目录名等。

9.3.2 变量

变量就是在程序执行过程中其值可以改变的量。Transact-SQL 语言允许使用两种变量：一种是用户自己定义的局部变量（Local Variable），另一种是系统提供的全局变量（Global Variable）。

1. 局部变量

局部变量是用户自己定义的变量。它的作用范围就在程序内部。通常只能在一个批处理或存储过程中使用，用来存储从表中查询到的数据；或当作程序执行过程中暂存变量使用。当这个批处理或存储过程结束后，这个局部变量的生命周期也就结束了。

局部变量使用 DECLARE 语句定义，并且指定变量的数据类型，然后可以使用 SET 或 SELECT 语句为变量初始化。局部变量必须以“@”开头，而且必须先声明后使用。

1）局部变量声明

```
DECLARE  @变量名 变量类型[,@变量名 变量类型…]
```

注意：

（1） 变量名必须以 @ 开头且符合有关标识符的规则。

（1） 变量先声明或定义，然后就可以在 SQL 命令中使用。

说明： 如果声明字符型的局部变量，一定要在变量类型中指明其最大长度，否则系统认为其长度为 1。

2）赋值

变量声明后，默认初值为 NULL。局部变量不能使用“变量=变量值”的格式进行初始化，必须使用 SELECT 或 SET 语句来设置其初始值。赋值方式为：

格式：

```
SET  @变量名=表达式
或：SELECT  @局部变量=变量值
```

SELECTt 可以同时给多个变量赋值，而 set 不能。

如 SELECT @x=1,@y=1 是允许的。

而用 SET 的话，需要写两个 SET 语句，

SET @x=1 和 SET @y=1。

另外，在 SELECT 查询语句时，变量也可以通过 SELECT 列表中当前所引用的列赋值。

格式为：

```
SELECT  @变量名=输出值  FROM 表 where …
```

SELECT　@变量名=输出值用于将单个值返回到变量中。

（1） 若 SELECT 语句返回多个值，则将返回的最后一个值赋给变量。

（1） 若 SELECT 语句没有返回值，变量保留当前值；

（1） 若表达式是不返回值的子查询，则变量为 NULL。

例 9- 1：声明变量并赋值。

```
declare @a int,@b int,@c int
set @a=1
set @b=2
set @c=@a+@b
print 'the value of c is'+convert(char(10),@c)
select @a,@b,@c
```

例 9- 2： select 赋值。

```
--声明变量
DECLARE @x int;
--将 sc 表中的最高成绩赋给变量
SELECT @x = MAX(grade) FROM sc;
```

例 9- 3：SELECT 命令赋值，多个返回值中取最后一个。

```
DECLARE @stu_name varchar(8)
--查询结果赋值，返回的是整个列的全部值，但最后一个赋给变量
SELECT @stu_name =Sname  FROM student
--显示局部变量的结果
SELECT @stu_name AS '学生姓名'
```

例 9- 4：SET 命令赋值。

```
DECLARE @no varchar(10)
--变量赋值
SET @no='Bj10001'
--显示指定学生学号、姓名
SELECT Sid,Sname FROM student WHERE SID=@no
```

2. 全局变量

全局变量是 SQL Server 系统内部使用的变量，其作用范围并不局限于某一程序，而是任何程序均可随时调用。

全局变量通常存储一些 SQL Server 的配置设置值和效能统计数据。用户可在程序中用全局变量来测试系统的设定值或者 Transact_SQL 命令执行后的状态值。

用户不能定义与全局变量同名的局部变量。从 SQL Server 7.0 开始，全局变量就以系统函数的形式使用。

格式：@@变量名。表 9- 5 列出了 SQL Server 常用的全局变量。

说明：	全局变量由系统定义和维护，用户只能显示和读取，不能建立全局变量，也不能用 SET 语句改变全局变量的值。

表 9- 5 SQL Server 常用的全局变量

名称	说明
@@ERROR	最后一个 T-SQL 错误的错误号
@@IDENTITY	最后一个插入的标识值
@@LANGUAGE	当前使用语言的名称
@@MAX_CONNECTIONS	可以创建的同时链接的最大数目
@@ROWCOUNT	受上一个 SQL 语言影响的行数
@@SERVERNAME	本地服务器的名称
@@SERVICENAME	该计算机上的 SQL 服务的名称
@@TIMETICKS	当前计算机上每刻度的微秒数
@@TRANSCOUNT	当前连接打开的事务数
@@VERSION	SQL Server 的版本信息

例 9- 5：显示本地服务器名。代码为：

```
select @@servername
```

9.3.3 运算符

运算符是一种符号，用来执行列间或变量间的数学运算和比较操作等。

1. 算术运算符

+（加）、-（减）、*（乘）、/（除）、%（求余）。

2. 位运算符

& （位与） 、|（位或） 、^（位异或）

3. 比较运算符

比较运算符测试两个表达式是否相同。除了 text、ntext 或 image 数据类型的表达式外，比较运算符可以用于所有的表达式。

比较运算符包括：=（等于）、> （大于） 、< （小于）、 >=（大于等于） 、<=（小于等于） 、<> （不等于） 、!=（不等于） 、!<（不小于）、!>（不大于）。

4. 逻辑运算符

逻辑运算符对某些条件进行测试，以获得其真实情况。逻辑运算符和比较运算符一样，返回带有 TRUE、FALSE 或 UNKNOWN 值的 Boolean 类型数据。在 T-SQL 中可以使用的逻辑运算符如表 9- 6 所示。

表 9- 6 逻辑运算符

逻辑运算符	含义
AND	如果两个布尔表达式都为 TRUE，那么就为 TRUE。
OR	如果两个布尔表达式中的一个为 TRUE，那么就为 TRUE。
NOT	对任何其他布尔运算符的值取反。
BETWEEN	如果操作数在某个范围之内，那么就为 TRUE
IN	如果操作数等于表达式列表中的一个，那么就为 TRUE。
LIKE	如果操作数与一种模式相匹配，那么就为 TRUE。
EXISTS	如果子查询包含一些行，那么就为 TRUE。
ALL	如果一组的比较都为 TRUE，那么就为 TRUE
ANY	如果一组的比较中任何一个为 TRUE，那么就为 TRUE
SOME	如果在一组比较中，有些为 TRUE，那么就为 TRUE

6. 字符串串联运算符

加号（+）是字符串串联运算符，可以用它将字符串串联起来。如'abc' + 'def' 的结果为 'abcdef'。其他所有字符串操作都使用字符串函数（如 SUBSTRING）进行处理。

7. 复合运算符

SQL Server 2008 新增复合运算符，可执行操作并将变量设置为结果。符合运算符如表 9- 7 所示。

表 9- 7 复合运算符

运算符	操作	运算符	操作
+=	将原始值加上一定的量	%=	将原始值除以一定的量
-=	将原始值减去一定的量	&=	对原始值执行位与运算

续

运算符	操作	运算符	操作
*=	将原始值乘上一定的量	^=	对原始值执行位异或运算
/=	将原始值除以一定的量	|=	对原始值执行位或运算

如：

```
DECLARE @x1 int = 27;
SET @x1 += 2 ;
SELECT @x1  -- 返回 29
```

9.3.4 表达式

表达式是标识符、值和运算符的组合。访问或更改数据时，可在多个不同的位置使用表达式。例如，可以将表达式用作要在查询中检索的数据的一部分，也可以用作查找满足一组条件的数据时的搜索条件。

在以下示例中的查询使用了多个表达式。例如，Name、ProductNumber、ListPrice 都是列名，1.5 是常量值。

```
SELECT Name,  SUBSTRING('This is a long string', 1, 5) AS SampleText,
ProductNumber, ListPrice * 1.5 AS NewPrice FROM Product;
```

9.4 系统内置函数

9.4.1 字符串函数

1. ASCII 函数

函数格式：

```
ASCII（character_expression）
```

功能：求 character_expression（char 或 varchar 类型）左端第一个字符的 ASCII 码。

返回值数据类型：int。例如：

```
Select ASCII（'abcd'）          --结果为字符 a 的 ASCII 码 97。
```

2. CHAR 函数

函数格式：

```
CHAR（integer_expression）
```

功能：求 ASCII 码 integer_expression 对应的字符，integer_expression 的有效范围为[0,255]，如果超出范围，则返回值 NULL。

返回值数据类型：CHAR。例如：

```
Select CHAR（97）          --结果为'a'。
```

CHAR 可用于将控制字符插入字符串中。表 9- 8 显示了一些常用的控制字符。

表 9- 8 控制字符及值

控制字符	值
制表符	CHAR(9)
换行符	CHAR(10)
回车	CHAR(13)

例9- 6: 使用回车符。

```
select '**'+char(13)+'***'
```

结果如图 9- 1 所示（注意，显示该结果时，在查询编辑器中，需在查询工具栏选择“以文本格式显示结果”）。

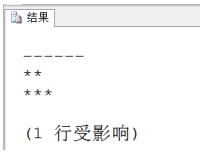


图 9- 1 char 函数示例

3. UNICODE 函数

语法:

```
UNICODE ( 'ncharacter_expression' )
```

功能：按照 Unicode 标准的定义，返回输入表达式的第一个字符的整数值。

返回类型：int。例如：

```
Select unicode(N'kerge')        --返回字符 k 的 unicode 值 107
```

4. NCHAR 函数

语法：NCHAR (integer_expression)

功能：根据 Unicode 标准所进行的定义，用给定整数代码返回 Unicode 字符。integer_expression 介于 0 与 65535 之间的所有正整数。如果指定了超出此范围的值，将返回 NULL。

返回类型：nchar(1)。例如：

```
select nchar(107)        --返回 unicode 字符 k
```

5. CHARINDEX 函数

函数格式：

```
CHARINDEX（expression1, expression2[,start]）
```

功能：在 expression2 中由 start 指定的位置开始查找 expression1 第一次出现的位置，如果没有找到，则返回 0。如果省略 start，或 start≤0，则从 expression2 的第一个字符开始。

返回类型：int。例如：

```
Select CHARINDEX('ab', '123abc123abc')            --结果为 4。
```

6. LEFT 函数

函数格式：

```
LEFT（expression1,n）
```

功能：返回字符串 expression1 从左边开始 n 个字符组成的字符串。如果 n=0，则返回一个空字符串。

返回类型：varchar。例如：

```
LEFT('abcde', 3)                    --结果为'abc'。
```

例 9- 7：找出 dbo.student 中名字以'刘'开头的学生信息。

```
select sno,sname from dbo.student where left(sname,1)='刘'
```

7. RIGHT 函数

函数格式：

```
RIGHT（expression1,n）
```

功能：返回字符串 expression1 从右边开始 n 个字符组成的字符串。如果 n=0，则返回一个空字符串。

返回类型：varchar。例如：

```
Select RIGHT('abcde', 3)              --结果为'cde'。
```

8. SUBSTRING 函数

函数格式：

```
SUBSTRING（expression1,start,length）
```

功能：返回 expression1（数据类型为字符串、binary、text 或 image）中从 start 开始长度为 length 个字符或字节的子串。

返回值：数据类型与 expression1 数据类型相同，但 text 类型返回值为 varchar，image 类型返回值为 varbinary，next 类型返回值为 nvarchar。例如：

```
Select SUBSTRING('abcde123',3,4)          --结果为'cde1'。
```

例 9- 7 也可以写成：

```
select sno,sname from dbo.student where substring(sname,1,1)='刘'
```

9. LEN 函数

函数格式：

```
LEN（expression1）
```

功能：返回字符串 expression1 中的字符个数，不包括字符串末尾的空格。

返回类型：int。例如：

```
Select LEN('abcde   ')          --结果为 5。
select len('刘')                --结果为 1。
```

10. DATALENGTH 函数

```
DATALENGTH ( expression )
```

功能：返回用于表示任何表达式的字节数。

返回类型：int。例如：

```
Select DATALENGTH('abcde   ')          --结果为 8（含 3 个空格）。
select DATALENGTH('刘')                --结果为 2。
```

11. LOWER 函数

函数格式：

```
LOWER（expression1）
```

功能：将字符串 expression1 中的大写字母替换为小写字母。

返回类型：varchar。例如：

```
Select LOWER('12ABC45*%^def')     --结果为'12abc45*%^def'。
```

12. UPPER 函数

函数格式：

```
UPPER（expression1）
```

功能：将字符串 expression1 中的小写字母替换为大写字母。

返回类型：varchar。例如：

```
Select UPPER('12ABC45*%^def')     --结果为'12ABC45 *%^DEF'。
```

13. LTRIM 函数

函数格式：

```
LTRIM（expression1）
```

功能：删除字符串 expression1 左端的空格。

返回类型：varchar。例如：

```
Select LTRIM('   12AB')          --结果为'12AB'。
```

14. RTRIM 函数

函数格式：

```
RTRIM（expression1）
```

功能：删除字符串 expression1 末尾的空格。

返回类型：varchar。例如：

```
Select RTRIM(LTRIM('   12AB   '))        --结果为'12AB'。
```

15. REPLACE 函数

函数格式：

```
REPLACE（expression1, expression2, expression3）
```

功能：将字符串 expression1 中所有的子字符串 expression2 替换为 expression3。

返回值数据类型：varchar。例如：

```
Select REPLACE('abcde','de','12')     --结果为'abc12'。
```

16. STUFF 函数

语法:

```
STUFF (character_expression,start,length,character_expression )
```

功能：删除指定长度的字符并在指定的起始点插入另一组字符。

返回类型:如果 *character_expression* 是一个支持的字符数据类型，则返回字符数据。

例如：

```
Select STUFF('abcde',4,2,'12')     --结果为'abc12'。
```

17. REVERSE 函数

函数格式：

```
REVERSE（expression1）
```

功能：按相反顺序返回字符串 expression1 中的字符。

返回值数据类型：varchar。例如：

```
Select REVERSE ('edcba')       --结果为 abcde。
```

18. SPACE 函数

函数格式：

```
SPACE（n）
```

功能：返回包含 n 个空格的字符串，如果 n 为负数，则返回一个空字符串。

返回值数据类型：char。

19. STR 函数

函数格式：

```
STR（expression1[,length[,decimal]]）
```

功能：将数字数据转换为字符数据。length 为转换得到的字符串总长度，包括符号、小数点、数字或空格。如果数字长度不够，则在左端加入空格补足长度，如果小数部分超过总长度，则进行四舍五入，length 的默认值为 10，decimal 为小数位位数。

返回值数据类型：char。例如：

```
select str(123,6)              --结果为'   123'
select str(123.456,5)          --结果为'  123'
select str(123.456,5,2)        --结果为'123.5'
select str(123.456,8,2)        --结果为'  123.46'
```

20. REPLICATE(character_expression,times)

语法：

```
REPLICATE(character_expression,times)
```

功能：返回多次复制后的字符表达式。times 参数的计算结果必须为整数。例如：

```
select replicate('*',3)          --返回‘***’
```

例 9- 8：以“*”方式输出菱形。

输出结果见 图 9- 2（在查询编辑器中以文本格式显示结果）。

```
declare @i int
set @i=1
while @i<=4
begin
 print space(4-@i)+
replicate('*',2*@i-1)
set @i=@i+1
end
set @i=1
while @i<=3
begin
 print space(@i)+replicate('*',7-2*@i)
 set @i=@i+1
end
```

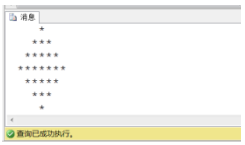


图 9- 2 输出菱形

9.4.2 日期函数

1. GETDATE 函数

函数格式：

```
GETDATE（）
```

功能：按 SQL Server 内部标准格式返回系统日期和时间。

返回值数据类型：datetime。例如：

```
Select getdate()        --结果为 2012-08-13 21:51:32.390
```

1. YEAR 函数

函数格式：

```
YEAR（date）
```

功能：返回指定日期 date 中年的整数。

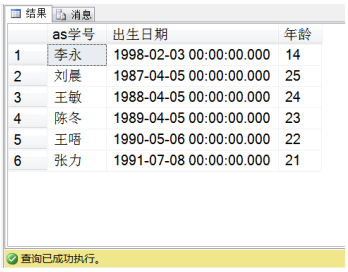
返回值数据类型：int。例如：

```
Select year('2004-3-5')        --结果为 2004
```

例 9- 9：返回学生表中学生的年龄。

```
select sname as 学号, BirthDate as 出生日期,
YEAR(getdate())-YEAR(birthdate) as 年龄
from dbo.student
```

结果见图 9- 3 所示。



结果 消息

	as学号	出生日期	年龄
1	李永	1998-02-03 00:00:00.000	14
2	刘晨	1987-04-05 00:00:00.000	25
3	王敏	1988-04-05 00:00:00.000	24
4	陈冬	1989-04-05 00:00:00.000	23
5	王晤	1990-05-06 00:00:00.000	22
6	张力	1991-07-08 00:00:00.000	21

查询已成功执行。

图 9- 3 YEAR 函数示例

1. MONTH 函数

函数格式：

```
MONTH（date）
```

功能：返回指定日期 date 中月份的整数。

返回值数据类型：int。例如：

```
Select month('2004-3-5')            --结果为 3
```

1. DAY 函数

函数格式：

```
DAY（date）
```

功能：返回指定日期 date 中天的整数。

返回值数据类型：int。例如：

```
Select day('2004-3-5')              --结果为 5
```

1. DATENAME 函数

函数格式：

```
DATENAME（datepart,date）
```

功能：返回日期 date 中由 datepart 指定的日期部分的字符串。

返回类型：nvarchar。

参数 datepart 可用的短语如表 9- 9 所示。

表 9- 9 datapart 参数说明

日期部分	缩写	日期部分	缩写
年份	yy、yyyy	工作日	dw
季度	qq、q	小时	hh
月份	mm、m	分钟	mi、n
每年的某一日	dy、y	秒	ss、s
日期	dd、d	毫秒	ms
星期	wk、ww	工作日	dw

例如：

```
select datename(yy, getdate())        --返回当前年份
select  datename (m, getdate())       --返回当前月份
select  datename (WEEKDAY, getdate()) --结果为当前日期是星期几
```

1. DATEPART 函数

函数格式：

```
DATEPART（dateprrt,date）
```

功能：与 DATENAME 类似，只是返回值为整数。

返回值数据类型：int。

1. DATEADD 函数

函数格式：

```
DATEADD（dateprrt,n,date）
```

功能：在 date 指定日期时间的 datepart 部分加上 n，得到一个新的日期时间值。

返回值数据类型：datetime，如果参数 date 为 smalldatetime，则返回值为 smalldatetime 类型。参数 datepart 可以使用如表 9- 9 所示中的短语或缩写。例如：

```
Select dateadd(yy,2,'2012-3-4') --结果为'2014-03-04 00:00:00.000'
Select dateadd(m,2,'2012-3-4')  --结果为'2012-05-04 00:00:00.000'
Select dateadd(d,2,'2012-3-4')  --结果为'2012-03-06 00:00:00.000'
```

1. DATEDIFF 函数

格式：

```
DATEDIFF ( datepart , startdate , enddate )
```

功能：返回指定的 startdate 和 enddate 之间所跨的指定 datepart 边界的计数（带符号的整数）。

例 9- 9：返回学生表中学生的年龄。

```
select sname as 学号, BirthDate as 出生日期,
DATEDIFF(yy, birthdate,getdate()) as 年龄 from dbo.student
```

9.4.3 数学函数

表 9- 10 数学函数及说明

函数	说明
ABS(numeric_expression)	返回数值表达式的绝对值。
CEILING (numeric_expression)	返回大于或等于指定数值表达式的最小整数。
FLOOR (numeric_expression)	返回小于或等于指定数值表达式的最大整数。
POWER(numeric_expression,power)	返回对数值表达式进行幂运算的结果。Power 参数的计算结果必须为整数。
PI ( )	返回 PI 的常量值。
SQRT (float_expression)	返回指定浮点值的平方根。
SQUARE (float_expression)	返回指定浮点值的平方。
RAND ([seed])	返回从 0 到 1 之间的随机 float 值。
ROUND (numeric_expression , length [,function])	返回一个数值，舍入到指定的长度或精度。ROUND(x,n[,f])函数按由 n 指定的精度和由 f 指定格式对 x 四舍五入，如果省略参数 f，其默认值为 0，则按由 n 指定的精度四舍五入，如果 f 为其他值，则执行截断。参数 n 如果为负数，并且 n 的绝对值大于 x 整数部分的数字个数，则结果为 0。

例 9- 10：ROUND 函数举例如下：

```
select ROUND(534.56, 1)         --结果为 534.60
select ROUND(534.56, 0)         --结果为 535.00
select ROUND(534.56, -1)        --结果为 530.00
select ROUND(534.56, -2)        --结果为 500.00
select ROUND(534.56, -3)       --结果为 1000.00
select ROUND(534.56, -4)       --结果为 0.00
```

9.4.4 其它常用函数

1. ISDATE 函数

语法：

```
ISDATE( expression )
```

功能：如果 expression 是 datetime 或 smalldatetime 数据类型的有效日期或时间值，则返回 1；否则，返回 0。例如：

```
select ISDATE('2009/2/29')    --返回 0
```

1. ISNULL (check_expression , replacement_value)

语法：

```
ISNULL ( check_expression , replacement_value )
```

功能：如果 check_expression 不为 NULL，则返回它的值；否则，在将 replacement_value 隐式转换为 check_expression 的类型（如果这两个类型不同）后，则返回前者。例如：

```
--如果成绩为 NULL，替换为 0
select grade as 成绩,ISNULL(grade,0) as ISNULL_结果 from sc
```

1. NULLIF 函数

```
NULLIF ( expression , expression )
```

功能：如果两个指定的表达式相等，则返回空值。

1. ISNUMERIC 函数

语法：

```
 ISNUMERIC ( expression )
```

功能：确定表达式是否为有效的数值类型。

9.4.5 转换函数

1. CAST 函数

功能：将一种数据类型的表达式转换为另一种数据类型的表达式。

语法：

```
CAST ( expression AS data_type [ (length ) ])
```

例：

```
--将日期时间类型转换为 char 类型
select cast('2010-3-2' as char(10))
```

1. CONVERT 函数

CONVERT 函数将一种数据类型的表达式转换为另一种数据类型的表达式。

语法：

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

有关参数的 style 值及说明的节选见表 9- 11。

表 9- 11 style 取值说明

不带世纪位 (yy)	带世纪数位 (yyyy)	标准	输入/输出
-	0 或 100 (*)	默认值	mon dd yyyy hh:miAM（或 PM）
1	101	美国	mm/dd/yyyy
2	102	ANSI	yy.mm.dd
3	103	英国/法国	dd/mm/yy
4	104	德国	dd.mm.yy
5	105	意大利	dd-mm-yy
6	106	-	dd mon yy
…	…	…	…

例 9- 11：查看不同 style 的转换结果。

```
select birthdate as 出生日期,
convert(char(20),birthdate,4)as style 为的转换后结果,
convert(char(20),birthdate,104)as style 为的转换后结果
 from dbo.student
```

结果见图 9- 4。

结果 消息

	出生日期	style为4的转换后结果	style为104的转换后结果
1	1998-02-03 00:00:00.000	03.02.98	03.02.1998
2	1987-04-05 00:00:00.000	05.04.87	05.04.1987
3	1988-04-05 00:00:00.000	05.04.88	05.04.1988
4	1989-04-05 00:00:00.000	05.04.89	05.04.1989
5	1990-05-06 00:00:00.000	06.05.90	06.05.1990
6	1991-07-08 00:00:00.000	08.07.91	08.07.1991

图 9- 4 style 取值示例

9.5 批处理和流程控制语句

9.5.1 批处理

1. 概念

批处理是指包含一条或多条 T-SQL 语句的语句组,它从应用程序一次性发送到 SQL Server 执行。SQL Server 将批处理编译成一个可执行单元，称为执行计划。批处理中如果某处发生编译错误，整个执行计划都无法执行。

书写批处理时,GO 语句作为批处理命令的结束标志，当编译器读取到 GO 语句时，会把 GO 语句前的所有语句当作一个批处理，并将这些语句打包发送给服务器。GO 语句本身不是 T -SQL 语句的的组成部分，只是一个表示批处理结束的前端指令。

说明：　局部（用户定义）变量的作用域限制在一个批处理中，不可在 GO 命令后引用。

2. 批处理有以下规则：

1） Create default , Create rule , Create trigger, Create procedure 和 Create view 等语句在同一个批处理中只能提交一个。

1） 不能在删除一个对象之后，在同一批处理中再次引用这个对象。

1） 不能把规则和默认值绑定到表字段或者自定义字段上之后，立即在同一批处理中使用它们。

1） 不能定义一个 check 约束之后，立即在同一个批处理中使用。

1） 不能修改表中一个字段名之后，立即在同一个批处理中引用这个新字段。

1） 若批处理中第一个语句是执行某个存储过程的 execute 语句，则 execute 关键字可以省略。若该语句不是第一个语句，则必须写上。

3. 几种指定批处理的方法

1） 应用程序作为一个执行单元发出的所有 SQL 语句构成一个批处理，并生成单个执行计划。

1） 存储过程或触发器内的所有语句构成一个批处理，每个存储过程或触发器都编译为一个执行计划。

1） 由 EXECUTE 语句执行的字符串是一个批处理，并编译为一个执行计划。

例如，以下的语句就是一个批处理。

```
DECLARE @Rate int
SELECT  @Rate = max(Rate)FROM EmployeePayHistory
PRINT @Rate
GO
```

9.5.2 流程控制语句

T-SQL 语言支持基本的流程控制逻辑，它允许按照给定的某种条件执行程序流和分支。

1. Print 语句

Print 语句用于向客户端返回用户定义消息。使用 PRINT 可以帮助排除 Transact-SQL 代码中的故障、检查数据值或生成报告。

格式：

```
PRINT msg_str | @local_variable | string_expr
```

其中，

1） msg_str ：字符串或 Unicode 字符串常量。

1） @ local_variable：任何有效的字符数据类型的变量。@local_variable 的数据类型必须为 **char** 或 **varchar**，或者必须能够隐式转换为这些数据类型。

1） string_expr ：返回字符串的表达式。

例 9- 12：返回当前日期的值。

代码如下：

```
DECLARE @PrintMessage varchar(50);
SET @PrintMessage = 'This message was printed on '
   + RTRIM(CAST(GETDATE() AS varchar(30)))+ '.';
PRINT @PrintMessage;
GO
```

2. IF…ELSE 语句

用于指定 T-SQL 语句的执行条件。如果满足条件，则在 IF 关键字及其条件之后执行 T-SQL 语句。可选的 ELSE 关键字引入另一个 T-SQL 语句，当不满足 IF 条件时就执行该语句。语法：

```
IF Boolean_expression                    /* 条件表达式 */
{ sql_statement | statement_block }     /* 条件表达式为 TRUE 时执行 */
[ ELSE
{ sql_statement | statement_block } ]    /* 条件表达式为 FALSE 时执行 */
```

例 9- 13：判断 1 号课程的平均成绩是否超过 60 分。

代码如下，执行结果如图 9-5 所示。

```
DECLARE  @avgscore decimal
select @avgscore=avg(grade)
from dbo.SC   where cno='1'
--判断成绩
IF @avgscore>=60
--输出结果
SELECT '课程平均成绩为'+CONVERT(varchar(10),@avgscore)+',超过 60 分'
ELSE
```

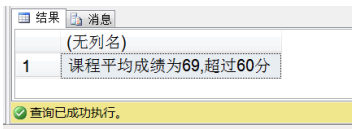


图 9-5 IF…ELSE 语句示例

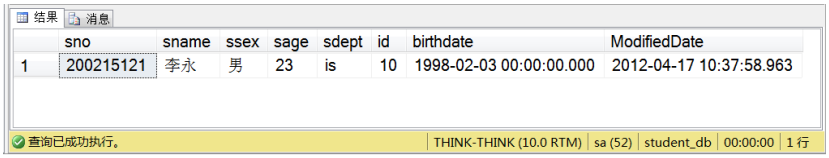
```
    --输出结果
     SELECT '课程平均成绩为'+CONVERT(varchar(10),@avgscore)
     +',不超过 60 分'
GO
```

经常利用 if 语句和 exists 或 not exists 关键字来判断 Select 查询结果是否有记录。

例 9- 14： 利用 if 语句和 exists 判断是否存在姓李的学生。

代码如下，结果见图 9- 6。

```
--exists 检查存在性
 if exists(select * from dbo.student  where sname like '李%')
-- 如果有，显示该同学的记录
select * from dbo.student where sname like '李%'
--没有显示'not available'
 else
    print 'not available'
Go
```



结果 | 消息

	sno	sname	ssex	sage	sdept	id	birthdate	ModifiedDate
1	200215121	李永	男	23	is	10	1998-02-03 00:00:00.000	2012-04-17 10:37:58.963

查询已成功执行。 THINK-THINK (10.0 RTM) | sa (52) | student_db | 00:00:00 | 1 行

图 9- 6 if exists 示例

在实际程序中，IF…ELSE 语句中不止包含一条语句，而是一组的 SQL 语句。为了可以一次执行一组 SQL 语句，这时就需要使用 BEGIN…END 语句将多条语句封闭起来。其语法格式为：

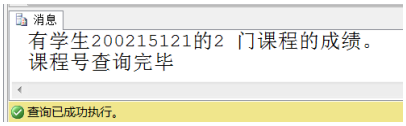
```
BEGIN
{sql_statement | statement_block }              /* 语句块 */
END
```

说明：　BEGIN...END 语句块允许嵌套。

例 9- 15: 查找学号为 200215121 的成绩。

代码如下，结果见图 9- 7。

```
DECLARE @message varchar(255),@grade_num int
--得到 200215121 号同学的选修课程的数目
SELECT @grade_num=COUNT(grade) FROM sc
WHERE  sno='200215121'
IF @grade_num=0
   BEGIN
     SET @message= '没有学生 200215121 的成绩'
     PRINT @message
   END
ELSE
   BEGIN
```



消息

有学生200215121的2 门课程的成绩。
课程号查询完毕

查询已成功执行。

图 9- 7 Begin…End 示例

```
        SET @message= '有学生 200215121 的'
        + convert(char(2),@grade_num)+ '门课程的成绩。'
        PRINT @message
      END
   SET @message='课程号查询完毕'
   PRINT @message
   GO
```

3. CASE 分支语句

CASE 关键字可根据表达式的真假来确定是否返回某个值，可在允许使用表达式的任意位置使用这一关键字。

CASE 可以进行多个分支的选择，具有两种格式：

1）简单 CASE 表达式

将某个表达式与一组简单表达式进行比较以确定结果。

2）CASE 搜索表达式

计算一组逻辑表达式以确定结果。

两种格式都支持可选的 ELSE 参数。

1）简单式

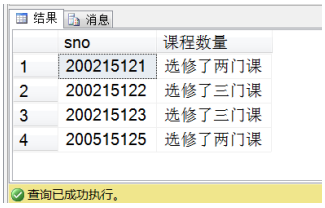
```
   CASE 表达式
      WHEN 表达式的值 1  THEN 返回表达式 1
      WHEN 表达式的值 2  THEN 返回表达式 2
      …
```

```
ELSE 返回表达式 n
END
```

例 9- 16： 显示学生选课的数量。

代码如下，结果如 图 9- 8 所示。

```
SELECT sno,'课程数量'=
CASE count(*)
 WHEN 1 THEN '选修了一门课'
 WHEN 2 THEN '选修了两门课'
 WHEN 3 THEN '选修了三门课'
END
FROM sc GROUP BY sno
```



结果 消息

	sno	课程数量
1	200215121	选修了两门课
2	200215122	选修了三门课
3	200215123	选修了三门课
4	200515125	选修了两门课

查询已成功执行。

图 9- 8 Case 语句示例 1

（2）搜索式。

```
CASE
    WHEN 逻辑表达式 1 THEN 返回表达式 1
    WHEN 逻辑表达式 2 THEN 返回表达式 2
    …
ELSE 返回表达式 n
END
```

例 9- 17： 将例 9- **16** 改写为搜索式写法。

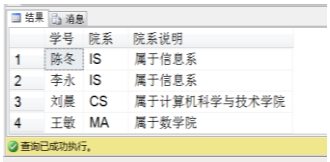
代码如下：

```
SELECT sno,count(*) AS 数量,课程数量=
CASE
 WHEN count(*)=1 THEN '选修了一门课'
 WHEN count(*)=2 THEN '选修了两门课'
 WHEN count(*)=3 THEN '选修了三门课'
END
FROM sc GROUP BY sno
```

例 9- 18： 为学生表的每个院系添加说明。

代码如下，结果如图 9- 9 所示。

```
SELECT sname AS 学号,sdept AS 院系, '院系说明'=
CASE sdept
--分别为各个院系添加说明
 WHEN 'IS' THEN '属于信息系'
 WHEN 'MA' THEN '属于数学院'
 WHEN 'CS' THEN '属于计算机科学与技术学院'
 ELSE '其他院系'
 END
FROM student ORDER BY sname    --按照姓名排序
```



结果 消息

	学号	院系	院系说明
1	陈冬	IS	属于信息系
2	李永	IS	属于信息系
3	刘晨	CS	属于计算机科学与技术学院
4	王敏	MA	属于数学院

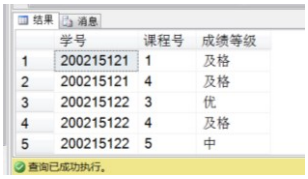
查询已成功执行。

图 9- 9 Case 语句示例 2

例 9- 19： 显示学生的成绩等级。

代码如下，结果见图 9- 10。

```
select Sno as 学号,Cno as 课程号,case
when grade>=90 then '优'
when grade>=80 then '良'
when grade>=70  then '中'
when grade>=60 then '及格'
else '不及格'
end
as '成绩等级'
 from sc
```



	学号	课程号	成绩等级
1	200215121	1	及格
2	200215121	4	及格
3	200215122	3	优
4	200215122	4	及格
5	200215122	5	中

图 9- 10 Case 语句示例 3

4. WHILE 语句

设置重复执行 SQL 语句或语句块的条件。只要指定的条件为真，就重复执行语句。可以使用 BREAK 和 CONTINUE 关键字在循环内部控制 WHILE 循环中语句的执行。

1） BREAK 语句：导致从最内层的 WHILE 循环中退出。将执行出现在 END 关键字（循环结束的标记）后面的任何语句。

1） CONTINUE：使 WHILE 循环重新开始执行，忽略 CONTINUE 关键字后面的任何语句。

WHILE 语句语法：

```
WHILE 逻辑表达式
Begin
   T-SQL 语句组
[break]          /*终止整个语句的执行*/
[continue]       /*结束一次循环体的执行*/
END
```

说明：	如果嵌套了两个或多个 WHILE 循环，则内层的 BREAK 将退出到下一个外层循环，将首先运行内层循环结束之后的所有语句，然后重新开始下一个外层循环。

例 9- 20：求 1 到 10 之间偶数的和。

代码如下，结果见图 9- 11。

```
DECLARE @i smallint,@sum smallint
SET @i=0
SET @sum=0
WHILE @i>=0
 BEGIN
   SET @i=@i+1
   IF @i>10
     BEGIN
SELECT '1 到之间偶数的和'=@sum
     BREAK
     END
   IF (@i%2)!=0
     CONTINUE
   ELSE
```

	1到10之间偶数的和
1	30

图 9- 11 while 语句

```
    SET @sum=@sum+@i
END
```

5. GOTO 语句

GOTO 语句将执行语句无条件跳转到标签处，并从标签位置继续处理。GOTO 语句和标签可在过程、批处理或语句块中的任何位置使用。其语法格式为：

```
GOTO label
```

6. WAITFOR 语句

WAITFOR 语句，称为延迟语句，就是暂停执行一个指定的时间间隔或者到一个指定的时间。它可以悬挂起批处理、存储过程或事务的执行，直到超过指定的时间间隔或到达指定的时间为止。其语法格式为：

```
WAITFOR
{ DELAY 'time_to_pass'       /* 设定等待时间 */
| TIME 'time_to_execute'     /* 设定等待到某一时刻 */
}
```

例 9- 21：延迟 30 秒执行查询。

```
WAITFOR DELAY '00:00:30'
SELECT  * FROM  student
```

例 9- 22：在时刻 21:20:00 执行查询。

```
WAITFOR TIME '21:20:00'
SELECT  * FROM  student
```

7. TRY-CATCH

TRY 块包含一组 T-SQL 语句。如果 TRY 块的语句中发生任何错误，控制将传递给 CATCH 块。CATCH 块包含另外一组语句，这些语句在错误发生时执行。如果 TRY 块中没有错误，控制将传递到关联的 END CATCH 语句后紧跟的语句。如果 END CATCH 语句是存储过程或触发器中的最后一条语句，控制将传递到调用该存储过程或触发器的语句。

```
BEGIN TRY
{ sql_statement | statement_block }
END TRY
BEGIN CATCH
{ sql_statement | statement_block }
END CATCH [ ; ]
```

TRY 块以 BEGIN TRY 语句开头，以 END TRY 语句结尾。在 BEGIN TRY 和 END TRY 语句之间可以指定一个或多个 Transact-SQL 语句。CATCH 块必须紧跟 TRY 块。CATCH 块以 BEGIN CATCH 语句开头，以 END CATCH 语句结尾。在 Transact-SQL 中，每个 TRY 块仅与一个 CATCH 块相关联。

在 CATCH 块中，可以使用以下的系统函数来确定关于错误的信息：

1） ERROR_NUMBER() 返回错误号。

1） ERROR_MESSAGE() 返回错误消息的完整文本。此文本包括为任何可替换参数（如长度、对象名称或时间）提供的值。

1） ERROR_SEVERITY() 返回错误严重性。

1） ERROR_STATE() 返回错误状态号。

1） ERROR_LINE() 返回导致错误的例程中的行号。

1） ERROR_PROCEDURE() 返回出现错误的存储过程或触发器的名称。

例 9- 23：插入重复值的错误信息。

代码如下：

```
BEGIN TRY
INSERT INTO dbo.student (sno,sname)  VALUES('200215121', '李永')
END TRY
BEGIN CATCH
   SELECT 'There was an error! '
    + ERROR_MESSAGE() AS   ErrorMessage,
      ERROR_LINE()        AS ErrorLine,
      ERROR_NUMBER()       AS ErrorNumber,
      ERROR_PROCEDURE()    AS ErrorProcedure,
      ERROR_SEVERITY()     AS ErrorSeverity,
      ERROR_STATE()       AS ErrorState
END CATCH
```

因为 student 表中已存在该生的信息，因此，系统会给出错误信息。结果见图 9- 12。

图 9- 12 try…catch 语句示例

8. Raiserror

有时会遇到 SQL SERVER 实际并不知道的一些错误，但我们希望能在客户端产生运行错误，而客户端使用的时候能够唤醒异常处理并进行相应的处理。要完成这一点，就需要在 T-SQL 中使用 RAISERROR 命令。RAISERROR 生成的错误与数据库引擎代码生成的错误的运行方式相同。

语法如下：

```
RAISERROR(<message ID | message_string>, <severity>,<state> [,<argument>
[,<…n>]]) [WITH option[,…n]]
```

参数说明见表 9- 12。

表 9- 12 Raiserror 参数及说明

参数	说明
msg_id	定制消息的错误代码。 RAISERROR 接受任何大于 13000 的数字，但是定制信息的 msg_id 要大于等于 50000。
msg_str	定制信息的文本。
severity	定制信息的级别。从 0 到 25，19-25 是重大错误代码。任何用户都可以指定 0 到 18 之间的严重级别。只有 sysadmin 固定服务器角色成员或具有 ALTER TRACE 权限的用户才能指定 19 到 25 之间的严重级别。若要使用 19 到 25 之间的严重级别，必须选择 WITH LOG 选项。20 或者更高的级别会自动终止用户的连接。
state	呈现导致错误的状态，状态值可以是 1-127 之间的任意值。。

argument	定义在错误信息中的可以替换的值。
WITH…	有三个选项: ·WITH LOG 纪录错误。只能用于级别高于 19 的错误。 ·WITH NOWAIT 将错误立刻发送到客户端 ·WITH SETERROR 将@@ERROR 设置@@ERROR 的值等于错误 ID。

说明:	错误由 master.dbo.sysmessages 表维护。每一个错误代码都有相应的级别和描述。

例 9- 24：在存储过程中使用 RAISERROR，当给定的两个时间差不等于 8 时，返回错误信息。

```
--创建存储过程 prc_error，判断给定的两个时间差是否等于 8
create procedure prc_error
@Start datetime,@End datetime
as
begin
Declare @Date_diff int
Select @Date_diff = DATEDIFF(hh, @Start, @End)
--两个时间差不等于 8，返回错误
If (@Date_diff != 8)
  RAISERROR('Error Raised', 16, 1)
End
--执行 prc_error
exec prc_error '2011-01-01 23:00:00.000','2011-01-01 10:00:00.000'
```

结果如图 9- 13 所示。

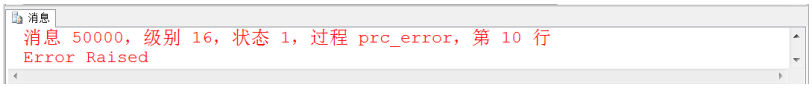


图 9- 13 RAISERROR 示例

9.6 游标

9.6.1 游标概述

由 SELECT 语句返回的行集包括满足该语句的 WHERE 子句中条件的所有行。这种由语句返回的完整行集称为结果集。应用程序，特别是交互式联机应用程序，并不总能将整个结果集作为一个单元来有效地处理。这些应用程序需要一种机制以便每次处理一行或一部分行。游标就是提供这种机制的对结果集的一种扩展。

游标（Cursor）是一种从包括多条数据记录的结果集中每次提取一条记录以便处理的机制，可以看做是查询结果的记录指针。

1. 游标的作用：

1） 允许定位在结果集的特定行。

1） 从结果集的当前位置检索一行或一部分行。

1） 支持对结果集中当前位置的行进行数据修改。

1） 为由其他用户对显示在结果集中的数据所做的更改提供不同级别的可见性支持。

1） 提供脚本、存储过程和触发器中用于访问结果集中的数据的 T-SQL 语句。

2. 使用游标的步骤：

1） 将游标与 T-SQL 语句的结果集相关联，并且定义该游标的特性，例如是否能够更新游标中的行。

1） 执行 T-SQL 语句以填充游标。

1） 从游标中检索想要查看的行。从游标中检索一行或一部分行的操作称为提取，执行一系列提取操作以便向前或向后检索行的操作称为滚动。

1） 根据需要，对游标中当前位置的行执行修改操作（更新或删除）。

1） 关闭游标。

9.6.2 使用游标

游标的语法有 SQL-92 中的语法和 T-SQL 扩展语法，以下我们主要介绍 SQL-92 中的语法。

1. 声明游标

语法如下：

```
DECLARE cursor_name [ INSENSITIVE ] [ SCROLL ] CURSOR
FOR select_statement
[ FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ] ] } ]
```

语法中参数说明如下：

1） cursor_name：指游标的名字。

1） INSENSITIVE：表示声明一个静态游标。

静态游标的含义：SQL SERVER 会将游标定义所选取出来的数据记录存放在一临时表内（建立在 tempdb 数据库下），对该游标的读取操作皆由临时表来应答。因此，对基本表的修改并不影响游标提取的数据，即游标不会随着基本表内容的改变而改变，同时也无法通过游标来更新基本表。如果不使用该保留字，那么对基本表的更新、删除都会反映到游标中。另外应该指出，当遇到以下情况发生时，游标将自动设定 INSENSITIVE 选项。

（1） 在 SELECT 语句中使用 DISTINCT、 GROUP BY、 HAVING UNION 语句；

（1） 使用 OUTER JOIN；

（1） 所选取的任意表没有索引；

（1） 将实数值当作选取的列。

3）SCROLL：表示声明一个动态游标。

动态游标的含义：滚动游标时，动态游标反映结果集中所做的所有更改，所有用户做的全部 UPDATE、INSERT 和 DELETE 均通过游标可见，因此消耗资源较多。动态游标时，所有的提取操作（如 FIRST、 LAST、 PRIOR、 NEXT、 RELATIVE、 ABSOLUTE）都可用。如果不使用该保留字，那么只能进行 NEXT 提取操作。由此可见，SCROLL 极大地增加了提取数据的灵活性，可以随意读取结果集中的任一行数据记录，而不必关闭再重开游标。

4）select_statement：定义结果集的 Select 语句。应该注意的是，在游标中不能使用 COMPUTE、COMPUTE BY、FOR BROWSE、INTO 语句。

5）READ ONLY 表明不允许游标内的数据被更新，而且在 Update 或 Delete 语句的 Where CURRENT OF 子句中，不允许对该游标进行引用。

6）Update [OF column_name[,…n]] 定义在游标中可被修改的列，如果不指出要更新的列，那么所有的列都将被更新。

当游标被成功创建后，游标名成为该游标的惟一标识，如果在以后的存储过程、触发器或 T_SQL 脚本中使用游标，必须指定该游标的名字。

例 9- 25：声明一个游标，用来指向学生表（Student）的学号（sno）和姓名（sname）数据。

如下的代码声明了一个名为 cur 的滚动式游标。

```
declare cur scroll cursor for select sno,sName from student
```

2. 打开游标

声明了游标以后，在做其他操作前，应打开游标。语法如下：

```
OPEN cursor_name
```

如果使用 INSENSITIVE 选项声明了游标，那么 OPEN 将创建一个临时表以保留结果集。如果结果集中任意行的大小超过 SQL Server 表的最大行大小，OPEN 将失败。

在游标被成功打开之后，@@CURSOR_ROWS 全局变量将用来记录游标内数据行数。如果所打开的游标在声明时带有 SCROLL 或 INSENSITIVE 保留字，那么@@CURSOR_ROWS 的值为正数且为该游标的所有数据行。如果未加上这两个保留字中的一个，则@@CURSOR_ROWS 的值为-1， 说明该游标内只有一条数据记录。

3. 提取数据

当游标被成功打开以后，就可以从游标中逐行地读取数据，以进行相关处理。从游标中读取数据主要使用 FETCH 命令。其语法规则为：

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

参数说明如下：

1） NEXT：返回结果集中当前行的下一行。如果 FETCH NEXT 是第一次读取游标中数据，则返回结果集中的是第一行而不是第二行。NEXT 为默认的游标提取选项。

1） PRIOR：返回结果集中当前行的前一行。如果 FETCH PRIOR 是第一次读取游标中数据，则无数据记录返回，并把游标位置设为第一行。

1） FIRST：返回游标中第一行。

1） LAST：返回游标中的最后一行。

1） ABSOLUTE {n | @nvar}：如果 n 或 @nvar 为正数，则返回从游标头开始向后的第 n 行，并将返回行变成新的当前行。如果 n 或 @nvar 为负，则返回从游标末尾开始向前的第 n 行，并将返回行变成新的当前行。如果 n 或 @nvar 为 0，则不返回行。n 必须是整数常量，并且 @nvar 的数据类型必须为 smallint、tinyint 或 int。

1） RELATIVE {n | @nvar}：如果 n 或@nvar 为正，则返回从当前行开始向后的第 n 行，并将返回行变成新的当前行。如果 n 或@nvar 为负，则返回从当前行开始向前的第 n 行，并将返回行变成新的当前行。如果 n 或 @nvar 为 0，则返回当前行。在对游标进行第一次提取时，如果在将 n 或 @nvar 设置为负数或 0 的情况下指定 FETCH RELATIVE，则不返回行。n 必须是整数常量，@nvar 的数据类型必须为 smallint、tinyint 或 int。

1） INTO @variable_name[,...n]：允许将使用 FETCH 命令读取的数据存放在多个变量中。在变量行中的每个变量必须与游标结果集中相应的列相对应，每一变量的数据类型也要与游标中数据列的数据类型相匹配。

@@FETCH_STATUS 全局变量返回上次执行 FETCH 命令的状态。在每次用 FETCH 从游标中读取数据时，都应检查该变量，以确定上次 FETCH 操作是否成功，来决定如何进行下一步处理。@@FETCH_STATUS 变量有三个不同的返回值:

1）返回值为 0，说明 FETCH 语句成功。

2）返回值为-1，说明 FETCH 语句失败或行不在结果集中。

3）返回值为-2，说明提取的行不存在。

在使用 FETCH 命令从游标中读取数据时，应该注意以下的情况：

当使用 SQL-92 语法来声明一个游标时，没有选择 SCROLL 选项时，只能使用 FETCH NEXT 命令来从游标中读取数据，即只能从结果集第一行按顺序地每次读取一行，由于不能使用 FIRST、 LAST、 PRIOR， 所以无法回滚读取以前的数据。

如果选择了 SCROLL 选项，则可能使用所有的 FETCH 操作。

例 9- 26：FETCH 示例：

```
declare @sID varchar(4),@sName nvarchar(4)
declare cur  scroll cursor for select sno,sName from student
open cur
fetch next from cur        --下一行
fetch absolute 3 from cur  --第三行
fetch relative -2 from cur --当前行的前二行
fetch prior from cur      --上一行
fetch last from cur       --最后一行
close cur                --关闭游标
deallocate cur           --释放游标
```

例 9- 27：通过游标读出学生表（Student）的每一个学生的学号、姓名。结果如图 9- 14 所示。

```
declare cur scroll cursor for select sno,sName from student
open cur
fetch next from cur
--@@FETCH_STATUS=0 表示上一个 fetch 语句取数据成功。
while @@FETCH_STATUS=0
fetch next from cur
close cur
deallocate cur
```

图 9- 14 提取数据示例

4. 关闭游标

1）使用 CLOSE 命令关闭游标

在处理完游标中数据之后必须关闭游标来释放数据结果集和定位于数据记录上的锁。CLOSE 语句关闭游标，但不释放游标占用的数据结构。如果准备在随后的使用中再次打开游标，则应使用 CLOSE 命令。

其关闭游标的语法规则为：

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

2）自动关闭游标

游标可以应用在存储过程、触发器和 T-SQL 脚本中。如果在声明游标与释放游标之间使用了事务，则在结束事务时游标会自动关闭。

假设游标的使用情况如图 9- 15 所示的情况：

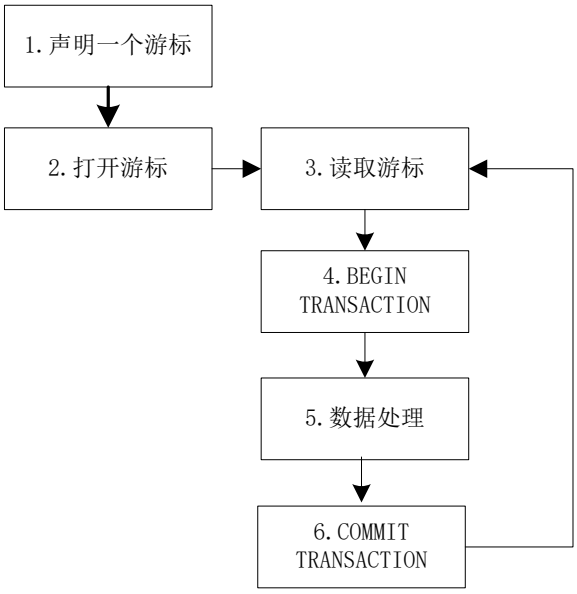


图 9- 15 游标和事务使用示意图

当从游标中读取一条数据记录进行以 BEGIN TRANSATION 为开头，COMMIT TRANSATION 或 ROLLBACK 为结束的事务处理时，在程序开始运行后，第一行数据能够被正确返回。

但是程序回到步骤 3， 读取游标的下一行，此时常会发现游标未打开的错误信息。其原因就在于当一个事务结束时，不管其是以 COMMIT TRANSATION 还是以 ROLLBACK TRANSATION 结束，SQL SERVER 都会自动关闭游标，所以当继续从游标中读取数据时就会造成错误。

解决这种错误的方法就是使用 SET 命令将 CURSOR_CLOSE_ON_COMMIT 这一参数设置为 OFF 状态。其目的就是让游标在事务结束时仍继续保持打开状态，而不会被关闭。使用 SET 命令的格式为： SET CURSOR_CLOSE_ON_COMMIT OFF。

5. 释放游标

在使用游标时，各种针对游标的操作或者引用游标名，或者引用指向游标的游标变量。当 CLOSE 命令关闭游标时，并没有释放游标占用的数据结构。因此常使用 DEALLOCATE 命令删除掉游标与游标名或游标变量之间的联系，并且释放游标占用的所有系统资源。

其语法规则为：

```
DEALLOCATE { { [GLOBAL] cursor_name } | @cursor_variable_name}
```

当使用 DEALLOCATE @cursor_variable_name 来删除游标时，游标变量并不会被释放，除非超过使用该游标的存储过程、触发器的范围(即游标的作用域)。

6. 利用游标修改、删除数据

可更新游标支持通过游标更新行的数据修改语句。

当定位在可更新游标中的某行上时，可以执行更新或删除操作，这些操作是针对用于在游标中生成当前行的基表行的，称之为“定位更新”。

定位更新在打开游标的同一个连接上执行。这就允许数据修改操作共享与游标相同的事务空间，并且使游标持有的锁不会阻止更新。

在游标中执行定位更新的方法可通过 UPDATE 或 DELETE 语句中的 T-SQL 中的 WHERE CURRENT OF 子句。WHERE CURRENT OF 子句通常在需要根据游标中的特定行进行修改时用于 Transact-SQL 存储过程、触发器以及脚本。

存储过程、触发器或脚本中使用可更新游标的步骤：

1）DECLARE 和 OPEN 游标。

2）使用 FETCH 语句定位到游标中的某一行。

3）用 WHERE CURRENT OF 子句执行 UPDATE 或 DELETE 语句。使用 DECLARE 语句中的 cursor_name 作为 WHERE CURRENT OF 子句中的 cursor_name。

例 9- 28：利用游标将成绩表（SC）中不及格的成绩改为 60 分。

代码如下，结果如图 9- 16 所示。

```
--声明变量
declare @v_sno varchar(10),
@v_cno varchar(10),@v_grade int
--声明游标
declare cur   scroll cursor
for select sno,cno,grade from sc
--打开游标
open cur
--取出第一行记录
fetch next from cur into @v_sno,@v_cno,@v_grade
--循环取值
while @@FETCH_STATUS=0
begin
--判断当前记录的成绩值
if @v_grade<60
   begin
--显示修改前的成绩
  select sno as 学号,cno as 课程号,grade as 更改前的成绩
  from sc  where sno=@v_sno and cno=@v_cno
--修改游标所在行的成绩
 update sc set grade=60  where current of  cur
--显示修改后的成绩
  select sno as 学号,cno as 课程号,grade as 更改后的成绩
  from sc  where sno=@v_sno and cno=@v_cno
  end
 --取下一行记录
 fetch next from cur into @v_sno,@v_cno,@v_grade
end
--关闭游标
close cur
--释放游标
deallocate cur
```

结果 消息

	学号	课程号	更改前的成绩
1	200215121	1	40

	学号	课程号	更改后的成绩
1	200215121	1	60

	学号	课程号	更改前的成绩
1	200215121	4	40

	学号	课程号	更改后的成绩
1	200215121	4	60

图 9- 16 利用游标修改数据

习题 9

1. 预测以下陈述的输出：Select Round(1234.567,1)

A. 1234.5　　B. 1234.6　　C. 1234　　D. 1234.56