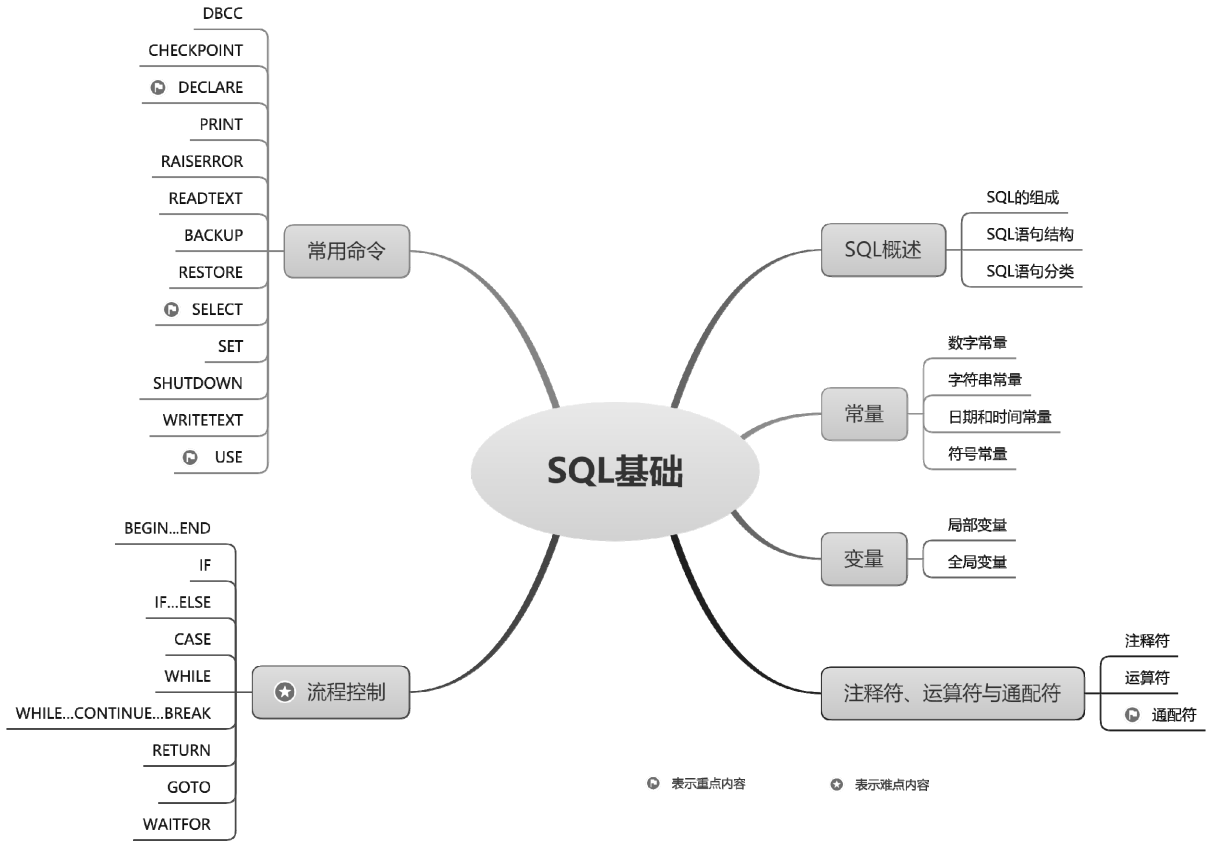
第 6 章

SQL 基础

本章将介绍 SQL 的基础知识，包括数据类型、常量、变量、运算符、流程控制语句以及一些常用的命令等，学习这些内容后读者可以掌握 SQL 语句的基本知识，为 SQL 语句编程打下良好的基础。

本章知识架构及重难点如下：



6.1 SQL 概述

SQL 是关系型数据库系统的标准语言，标准的 SQL 语句几乎可以在所有的关系型数据库上不加修改地使用。Access、Visual FoxPro、Oracle 这样的数据库同样支持标准的 SQL 语句。

6.1.1　SQL 的组成

SQL 主要由以下 3 个部分组成。

- ☑ 数据定义语言：用于在数据库系统中对数据库、表、视图、索引等数据库对象进行创建和管理。
- ☑ 数据控制语言（Data Control Language，DCL）：实现对数据库中数据的完整性、安全性等的控制。
- ☑ 数据操纵语言（Data Manipulation Language，DML）：用于插入、修改、删除和查询数据库中的数据。

6.1.2　SQL 语句结构

每条 SQL 语句均由一个谓词（Verb）开始，该谓词描述这条语句要产生的动作，如 SELECT 或 UPDATE 关键字。谓词后紧接着一个或多个子句（Clause），子句中给出被谓词作用的数据或提供谓词动作的详细信息。每一条子句都由一个关键字开始。下面介绍 SELECT 语句的主要结构。语法如下：

```
SELECT  子句
[INTO 子句]
FROM 子句
[WHERE 子句]
[GROUP  BY 子句]
[HAVING  子句]
[ORDER BY 子句]
```

【例 6.1】在 Student 数据表中查询女同学的信息。运行结果如图 6.1 所示。**（实例位置：资源包\TM\sl\6\1）**

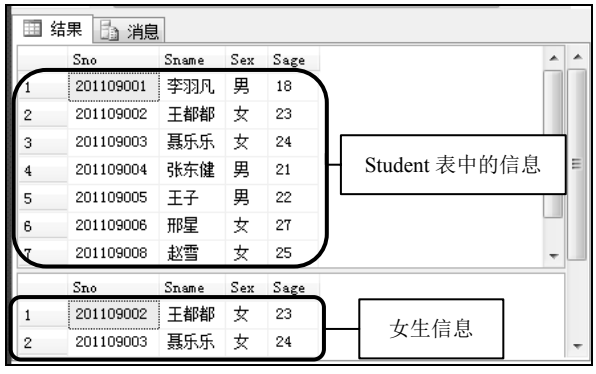


图 6.1　查询 Student 表中女生的信息

SQL 语句如下：

```
use db_Test
select * from Student
where Sex='女' order by Sage
```

误区警示

SQL 语句中的关键字不区分大小写，这点一定要注意。例如，例 6.1 代码中的 select，跟 SELECT、Select、sELECT 等表示的都是查询的意思，都可以正确执行。

6.1.3 SQL 语句分类

SQL 语句的分类如下。

（1）变量说明语句：说明变量的命令。

（2）数据定义语句：建立数据库、数据库对象和定义列，大部分是以 CREATE 开头的命令，如 CREATE TABLE、CREATE VIEW 和 DROP TABLE 等。

（3）数据操纵语句：操纵数据库中数据的命令，如 SELECT、INSERT、UPDATE、DELETE 和 CURSOR 等。

（4）数据控制语句：控制数据库组件的存取许可、存取权限等命令，如 GRANT、REVOKE 等。

（5）流程控制语句：设计应用程序流程的语句，如 IF WHILE 和 CASE 等。

（6）内嵌函数：说明变量的命令。

（7）其他命令：嵌于命令中使用的标准函数。

6.2 常　　量

数据在内存中存储始终不变化的量叫作常量。常量，也称文字值或标量值，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。

6.2.1 数字常量

数字常量包括整数常量、小数常量以及浮点常量。

整数常量和小数常量在 SQL 中被写成普通的小数数字，前面可加正负号。例如：

```
12, -37, 200.45
```

在数字常量的位之间不能加逗号。例如，123123 不能表示为 123,123。

浮点常量使用符号 e 指定，例如：

```
1.5e3, -3.14e1,2.5e-7
```

e 后面数字是几表示“乘 10 的几次幂”。

6.2.2 字符串常量

字符串常量括在单引号内，包含字母和数字字符（a～z、A～Z 和 0～9）以及特殊字符，如感叹号

（!）、at 符（@）和数字号（#）。

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。

以下是字符串的示例：

```
'MingRi'
'O'   'Brien'
'Process X is 50% complete.'
```

6.2.3　日期和时间常量

SQL 规定日期、时间和时间间隔的常量值被指定为日期和时间常量。例如：

```
'1984-03-10 ' ,'03/03/1976'
```

日期和时间根据国家不同，书写方式也不同。例如，美国表示为 mm/dd/yyyy，欧洲表示为 dd.mm.yyyy，日本表示为 yyyy-mm-dd 等。

6.2.4　符号常量

除了用户提供的常量外，SQL 包含几个特有的符号常量，这些常量代表不同的常用数据值。

例如，CURRENT_DATE 表示当前的日期，类似的如 CURRENT_TIME、CURRENT_TIMESTAMP 等。这些符号常量也可以通过 SQL Server 的内嵌函数访问。

6.3　变　　量

数据在内存中存储可以变化的量叫作变量。为了在内存中存储信息，用户必须指定存储信息的单元，并为该存储单元命名，以方便获取信息，这就是变量的功能。SQL 可以使用两种变量：一种是局部变量，另一种是全局变量。局部变量和全局变量的主要区别在于存储的数据作用范围不同。

6.3.1　局部变量

局部变量是用户可自定义的变量，它的作用范围仅在程序内部。局部变量的名称是用户自定义的，命名的局部变量名要符合 SQL Server 标识符命名规则，局部变量名必须以@开头。

1．声明局部变量

局部变量的声明需要使用 DECLARE 语句。语法如下：

```
DECLARE
{
@varaible_name datatype    [ ,… n  ]
}
```

参数说明如下。

- ☑ @varaible_name：局部变量的变量名，必须以@开头，另外变量名的形式必须符合 SQL Server 标识符的命名方式。
- ☑ datatype：局部变量使用的数据类型，可以是除 text、ntext 或者 image 类型外所有的系统数据类型和用户自定义数据类型。一般来说，如果没有特殊的用途，建议在应用时尽量使用系统提供的数据类型。这样做可以减少维护应用程序的工作量。

例如，声明局部变量@songname。SQL 语句如下：

```
declare   @songname    char(10)
```

2．为局部变量赋值

为变量赋值的方式一般有两种：一种是使用 SELECT 语句，一种是使用 SET 语句。使用 SELECT 语句为变量赋值的语法如下：

```
SELECT     @varible_name    =   expression
[FROM      table_name [ ,… n ]
WHERE     clause   ]
```

SELECT 语句的作用是给变量赋值，而不是从表中查询数据。而且在使用 SELECT 语句赋值的过程中，不一定必须使用 FROM 关键字和 WHERE 子句。

【例 6.2】在 tb_Student 数据表中，把“所学专业”是“会计学”的信息赋值局部变量@songname，并把它的值用 print 关键字显示。运行结果如图 6.2 所示。（**实例位置：资源包\TM\sl\6\2**）

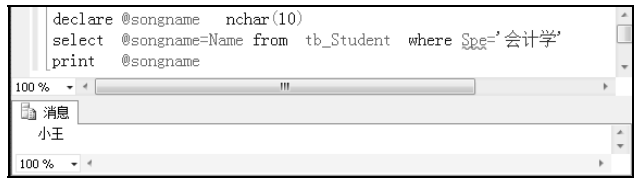


图 6.2　把查询内容赋值给局部变量

SQL 语句如下：

```
USE db_Test
declare @songname     nchar(10)
select   @songname=Name from    tb_Student    where Spe='会计学'
print     @songname
```

SELECT 语句赋值和查询不能混淆。例如，声明一个局部变量名是@b 并给它赋值的 SQL 语句如下：

```
declare   @b   int
select   @b=1
```

另一种为局部变量赋值的方式是使用 SET 语句，常用语法如下：

```
{ SET   @varible_name   =   expression } [ ,… n ]
```

下面是一个简单的赋值语句：

```
DECLARE   @song   char(20)
SET   @song   =   'I   love   flower'
```

还可以为多个变量一起赋值，相应的 SQL 语句如下：

```
declare   @b   int, @c   char(10),@a   int
select @b=1, @c='love',@a=2
```

注意

数据库语言和编程语言都有一些关键字。关键字是在某一环境下能够促使某一操作发生的字符组。为避免冲突和产生错误，在命名表、列、变量以及其他对象时应避免使用关键字。

6.3.2　全局变量

全局变量是 SQL Server 系统内部事先定义好的变量，不需用户参与定义，对用户而言，其作用范围并不局限于某一程序，任何程序均可随时调用。全局变量通常用于存储一些 SQL Server 的配置设定值和效能统计数据。

SQL Server 一共提供了 30 多个全局变量，本节只对一些常用的全局变量的功能和使用方法进行介绍。全局变量的名称都是以@@开头的。

- ☑ @@CONNECTIONS：记录自最后一次服务器启动以来，所有针对这台服务器进行的连接次数，包括没有连接成功的尝试。使用@@CONNECTIONS 可以让系统管理员很容易地得到今天所有试图连接本服务器的连接次数。
- ☑ @@CUP_BUSY：记录自上次启动的工作时间，无论连接成功还是失败，都是以 ms 为单位的 CPU 工作时间。
- ☑ @@CURSOR_ROWS：返回在本次服务器连接中，打开游标（游标是获取一组数据并能够一次与一个单独的数据进行交互的方法。详见第 13 章）取出数据行的行数。
- ☑ @@DBTS：返回当前数据库中 timestamp 数据类型的当前值。
- ☑ @@ERROR：返回执行上一条 SQL 语句所返回的错误代码。在 SQL Server 服务器执行完一条语句后，如果执行成功，则返回@@ERROR 的值为 0；如果发生错误，则返回错误信息，返回@@ERROR 的值为错误代码，该代码将一直保持下去，直到下一条语句执行为止。由于@@ERROR 在每一条语句执行后被清除并且重置，因此应在语句验证后立即检查，或将其保存到一个局部变量中以备事后查看。
- ☑ @@FETCH_STATUS：返回上一次使用游标 FETCH 操作所返回的状态值，且返回值为整型，其描述如表 6.1 所示。

表 6.1　@@FETCH_STATUS 返回值的描述

返　回　值	描　　述
0	FETCH 语句成功
−1	FETCH 语句失败或此行不在结果集中
−2	被提取的行不存在

例如，到了最后一行数据还要取下一行数据时，返回的值为−2，表示行不存在。

- ☑ @@IDENTITY：返回最近一次插入的 identity 列的数值，返回值是 numeric。
- ☑ @@IDLE：返回以 ms 为单位计算 SQL Server 服务器自最近一次启动以来处于停顿状态的时间。
- ☑ @@IO_BUSY：返回以 ms 为单位计算的 SQL Server 服务器自最近一次启动以来输入和输出使用的时间。

- ☑ @@LOCK_TIMEOUT：返回当前对数据锁定的超时设置。
- ☑ @@PACK_RECEIVED：返回 SQL Server 服务器自最近一次启动以来从网络接收数据分组的总数。
- ☑ @@PACK_SENT：返回 SQL Server 服务器自最近一次启动以来向网络发送数据分组的总数。
- ☑ @@PROCID：返回当前存储过程的 ID 标识。
- ☑ @@REMSERVER：返回在登录记录中记载远程的 SQL Server 服务器名。
- ☑ @@ROWCOUNT：返回上一条 SQL 语句影响数据行的行数。对所有不影响数据库数据的 SQL 语句，这个全局变量返回的结果是 0。在进行数据库编程时，经常要检测@@ROWCOUNT 的返回值，以便明确执行的操作是否达到了目标。
- ☑ @@SPID：返回当前服务器进程的 ID 标识。
- ☑ @@TOTAL_ERRORS：返回自 SQL Server 服务器启动以来，遇到读写错误的总数。
- ☑ @@TOTAL_READ：返回自 SQL Server 服务器启动以来，读磁盘的次数。
- ☑ @@TOTAL_WRITE：返回自 SQL Server 服务器启动以来，写磁盘的次数。
- ☑ @@TRANCOUNT：返回当前连接中，处于活动状态事务的总数。
- ☑ @@VERSION：返回当前 SQL Server 服务器安装日期、版本以及处理器的类型。

6.4 注释符、运算符与通配符

注释符对代码进行解释或说明。常见的运算符有算术运算符、赋值运算符、比较运算符和逻辑运算符等。常用的通配符有%、_（下画线）、[ ]、[^]。

6.4.1 注释符

注释语句不是可执行语句，不参与程序的编译，通常是一些说明性的文字，对代码的功能或者代码的实现方式给出简要的解释和提示。

在 SQL 中，可使用以下两类注释符。

- ☑ ANSI 标准的注释符（--），用于单行注释；如下面 SQL 语句所加的注释。

```
use  pubs     --打开数据表
```

- ☑ 与 C 语言相同的程序注释符，即“/*”和“*/”。“/*”用于注释文字的开头，“*/”用于注释文字的结尾，可在程序中标识多行文字为注释。

  例如，有多行注释的 SQL 语句如下：

```
USE  db_Test
declare @songname    char(10)
select  @songname=Stu_col  from  tb_Student  where Stu_spe='会计学'
print    @songname
/*打开 db_Test 数据库，定义一个变量
把查询到的结果赋值给所定义的变量*/
```

说明

把所选的行一次都注释的快捷键是 Shift+Ctrl+C，一次取消多行注释的快捷键是 Shift+Ctrl+R。

6.4.2　运算符

运算符是一种符号，用来进行常量、变量或者列之间的数学运算和比较操作，它是 SQL 很重要的部分。运算符的常见类型有算术运算符、赋值运算符、比较运算符、逻辑运算符、位运算符和连接运算符。

1．算术运算符

算术运算符在两个表达式上执行数学运算，这两个表达式可以是数字数据类型分类的任何数据类型。

算术运算符包括+（加）、-（减）、×（乘）、/（除）、%（取余）。

【例 6.3】求 2 对 5 取余。在查询分析器中运行的结果如图 6.3 所示。**（实例位置：资源包\TM\sl\6\3）**

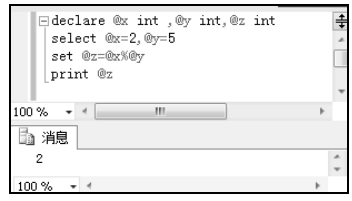


图 6.3　求 2 对 5 取余的结果

SQL 语句如下：

```
declare @x int ,@y int,@z int
select @x=2,@y=5
set @z=@x%@y
print @z
```

注意

取余运算两边的表达式必须是整型数据。

2．赋值运算符

SQL 有一个赋值运算符，即等号（=）。在下面的示例中，创建了@songname 变量，然后利用赋值运算符将@songname 设置成一个由表达式返回的值。代码如下：

```
DECLARE @songname    char(20)
SET @songname='loving'
```

还可以使用 SELECT 语句进行赋值，并输出该值。

```
DECLARE @songname    char(20)
SELECT @songname ='loving'
print @songname
```

3．比较运算符

比较运算符测试两个表达式是否相同。除 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>（大于）、<（小于）、=（等于）、>=（大于等于）、<=（小于等于）、<>（不等于）、!=（不等于）、!>（不大于）、!<（不小于），其中，!=、!<、!<不是 ANSI 标

准的运算符。

比较运算符的结果为布尔数据类型，它有 3 个值：TRUE、FALSE 及 UNKNOWN。那些返回布尔数据类型的表达式被称为布尔表达式。

和其他 SQL Server 数据类型不同，不能将布尔数据类型指定为表列或变量的数据类型，也不能在结果集中返回布尔数据类型。例如：

```
3>5=FALSE,6<>9=TRUE
```

4．逻辑运算符

逻辑运算符对某个条件进行测试，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 或 FALSE 的布尔数据类型。SQL 支持的逻辑运算符如表 6.2 所示。

表 6.2　SQL 支持的逻辑运算符

运　算　符	行　　为
ALL	如果一个比较集中全部都是 TRUE，则值为 TRUE
AND	如果两个布尔表达式均为 TRUE，则值为 TRUE
ANY	如果一个比较集中任何一个为 TRUE，则值为 TRUE
BETWEEN	如果操作数在某个范围内，则值为 TRUE
EXISTS	如果子查询包含任何行，则值为 TRUE
IN	如果操作数与一个表达式列表中的某个相等，则值为 TRUE
LIKE	如果操作数匹配某个模式，则值为 TRUE
NOT	对任何其他布尔运算符的值取反
OR	如果任何一个布尔表达式是 TRUE，则值为 TRUE
SOME	如果一个比较集中的某些为 TRUE，则值为 TRUE

【例 6.4】在 Student 数据表中，查询学生中年龄大于 24 岁的女生信息。运行结果如图 6.4 所示。**（实例位置：资源包\TM\sl\6\4）**

图 6.4　查询年龄大于 24 岁的女生信息

SQL 语句如下：

```
USE   db_Test
Select   *   from   Student
where Sex='女' and Sage>24
```

当 NOT、AND 和 OR 出现在同一表达式中时，优先级是：NOT、AND、OR。例如：

```
3>5 or 6>3 and not 6>4=FALSE
```

先计算 not 6>4=FALSE；然后再计算 6>3 AND FALSE =FALSE，最后计算 3>5 or FALSE= FALSE。

5．位运算符

位运算符的操作数是整数数据类型或二进制串数据类型（image 数据类型除外）范畴的。SQL 支持的位运算符如表 6.3 所示。

表 6.3　SQL 支持的位运算符

运　算　符	说　　明	运　算　符	说　　明
&	按位 AND	^	按位互斥 OR
|	按位 OR	～	按位 NOT

6．连接运算符

连接运算符“+”用于连接两个或两个以上的字符或二进制串、列名或者串和列的混合体，将一个串加入另一个串的末尾。语法如下：

```
<expression1>+<expression2>
```

【例 6.5】用“+”连接两个字符串。运行结果如图 6.5 所示。**（实例位置：资源包\TM\sl\6\5）**

图 6.5　用“+”连接两个字符串

SQL 语句如下：

```
declare @name char(20)
set @name=' 最爱'
print '我喜爱的电影是'+@name
```

7．运算符优先级

当一个复杂表达式中包含多个运算符时，运算符的优先级决定了表达式计算和比较操作的先后顺序。运算符的优先级由高到低的顺序如下。

（1）+（正）、−（负）、～（位反）

（2）*（乘）、/（除）、%（取余）

（3）+（加）、+（连接运算符）、−（减）

（4）=、>、<、>=、<=、<>、!=、!>、!<（比较运算符）

（5）^（按位异或）、&（按位与）、|（按位或）

（6）NOT

（7）AND

（8）ALL ANY BETWEEN IN LIKE OR SOME（逻辑运算符）

（9）=（赋值）

若表达式中含有相同优先级的运算符，则从左向右依次处理。还可以使用括号提高运算的优先级，在括号中的表达式优先级最高。如果表达式有嵌套的括号，那么首先对嵌套最内层的表达式求值。

例如：

```
DECLARE   @num   int
SET @num = 2 * (4 + (5 - 3) )
```

上面的代码中，先计算（5-3），然后再加 4，最后再和 2 相乘。

6.4.3 通配符

匹配指定范围内或者属于方括号指定的集合中的任意单个字符。可以在涉及模式匹配的字符串比较（如 LIKE 和 PATINDEX）中使用这些通配符。

在 SQL 中通常用 LIKE 关键字与通配符结合实现模糊查询。其中，SQL 支持的通配符的描述和示例如表 6.4 所示。

表 6.4 SQL 支持的通配符的描述和示例

通 配 符	描 述	示 例
%	包含零个或更多字符的任意字符	loving%可以表示：loving，loving you，loving?
_（下画线）	任何单个字符	loving_ 可以表示：lovingc，后面只能再接一个字符
[ ]	指定范围（[a～f]）或集合（[abcdef]）中的任何单个字符	[0～9]123 表示以 0～9 任意一个字符开头，以 123 结尾的字符
[^]	不属于指定范围（[a～f]）或集合（[abcdef]）的任何单个字符	[^0～5]123 表示不以 0～5 任意一个字符开头，却以 123 结尾的字符

6.5 流 程 控 制

流程控制语句是用来控制程序执行流程的语句。使用流程控制语句可以提高编程语言的处理能力。与程序设计语言（如 C 语言）一样，SQL 提供的流程控制语句如下：

```
BEGIN…END       IF          IF...ELSE
CASE            WHILE       WHILE...CONTINUE... BREAK
RETURN          GOTO        WAITFOR
```

6.5.1 BEGIN...END

BEGIN...END 语句用于将多个 SQL 语句组合为一个逻辑块。当流程控制语句必须执行一个包含两条或两条以上的 SQL 语句的语句块时，使用 BEGIN...END 语句。语法如下：

```
BEGIN
{sql_statement...}
END
```

其中，sql_statement 是指包含的 SQL 语句。

BEGIN 和 END 语句必须成对使用，任何一条语句均不能单独使用。BEGIN 语句后为 SQL 语句块，END 语句指示语句块结束。

【例 6.6】在 BEGIN…END 语句块中完成两个变量的值交换。运行结果如图 6.6 所示。**（实例位置：资源包\TM\sl\6\6）**

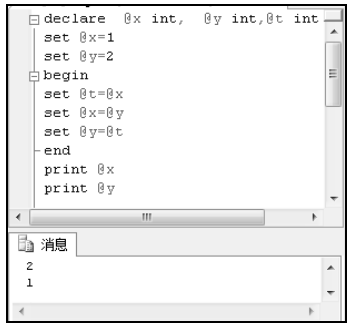


图 6.6　交换两个变量的值

SQL 语句如下：

```
declare   @x int,   @y int,@t int
set @x=1
set @y=2
begin
set @t=@x
set @x=@y
set @y=@t
end
print @x
print @y
```

此实例不用 BEGIN...END 语句结果也完全一样，但 BEGIN...END 和一些流程控制语句结合起来更有用。在 BEGIN...END 中可嵌套另外的 BEGIN...END 定义另一程序块。

6.5.2　IF

在 SQL Server 中为了控制程序的执行方向，也会像其他语言（如 C 语言）一样有顺序、选择和循环 3 种控制语句，其中，IF 就属于选择判断结构。IF 结构的语法如下：

```
IF<条件表达式>
    {命令行|程序块}
```

其中，<条件表达式>可以是各种表达式的组合，但表达式的值必须是逻辑值 TRUE 或 FALSE。其中命令行和程序块可以是合法的 SQL 任意语句，但含两条或两条以上语句的程序块必须加 BEGIN...END 子句。

执行顺序是：遇到选择结构 IF 子句，先判断 IF 子句后的条件表达式，如果条件表达式的逻辑值是 TRUE，就执行后面的命令行或程序块，然后再执行 IF 结构下一条语句；如果条件表达式的逻辑值

是 FALSE，就不执行后面的命令行或程序块，直接执行 IF 结构的下一条语句。

【例 6.7】判断数字“3”是否是正数。运行结果如图 6.7 所示。（**实例位置：资源包\TM\sl\6\7**）

SQL 语句如下：

```
declare   @x int
set @x=3
if @x>0
print '@x 是正数'
print'end'
```

【例 6.8】判断数字“8”的奇偶性。运行结果如图 6.8 所示。（**实例位置：资源包\TM\sl\6\8**）

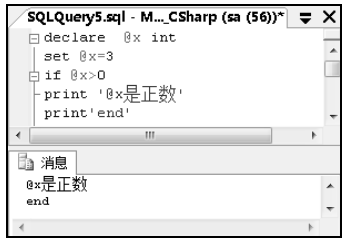


图 6.7　判断“3”的正负

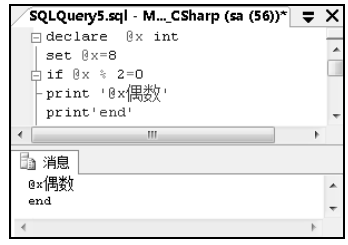


图 6.8　判断“8”的奇偶性

SQL 语句如下：

```
declare   @x int
set @x=8
if @x % 2=0
print '@x 偶数'
print'end'
```

6.5.3　IF…ELSE

IF 选择结构可以带 ELSE 子句。IF...ELSE 的语法如下：

```
IF<条件表达式>
    {命令行 1|程序块 1}
ELSE
    {命令行 2|程序块 2}
```

如果逻辑判断表达式返回的结果是 TRUE，那么程序接下来会执行命令行 1 或程序块 1；如果逻辑判断表达式返回的结果是 FALSE，那么程序接下来会执行命令行 2 或程序块 2。无论哪种情况，最后都要执行 IF...ELSE 语句的下一条语句。

【例 6.9】判断两个数“8”和“3”的大小。运行结果如图 6.9 所示。（**实例位置：资源包\TM\sl\6\9**）

SQL 语句如下：

```
declare   @x int,@y int
set @x=8
set @y=3
if @x>@y
print '@x 大于@y'
```

```
else
print'@x 小于等于@y'
```

IF...ELSE 结构还可以嵌套解决一些复杂的判断。

【例 6.10】输入一个坐标值（8,−3），然后判断它在哪一个象限。运行结果如图 6.10 所示。（**实例位置：资源包\TM\sl\6\10**）

图 6.9　判断两个数的大小

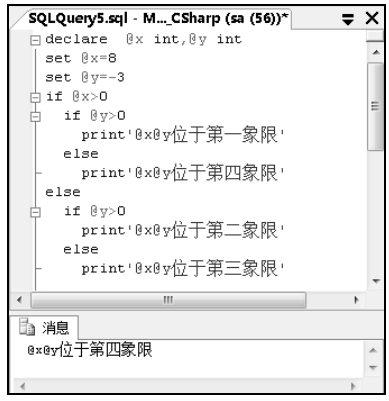


图 6.10　判断坐标位于的象限

SQL 语句如下：

```
declare   @x int,@y int
set @x=8
set @y=-3
if @x>0
  if @y>0
    print'@x@y 位于第一象限'
  else
    print'@x@y 位于第四象限'
else
  if @y>0
    print'@x@y 位于第二象限'
  else
    print'@x@y 位于第三象限'
```

6.5.4　CASE

使用 CASE 语句可以很方便地实现多重选择的情况，比 IF...THEN 结构有更多的选择和判断的机会，从而避免编写多重的 IF...THEN 嵌套循环。

SQL 支持 CASE 有以下两种语句格式。

简单 CASE 函数：

```
CASE input_expression
    WHEN when_expression THEN result_expression
          [ ...n ]
      [
```

```
        ELSE else_result_expression
    END
```

CASE 搜索函数：

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

CASE 函数的参数及其说明如表 6.5 所示。

表 6.5　CASE 函数的参数及其说明

参　数	描　述
input_expression	使用简单 CASE 格式时所计算的表达式。input_expression 是任何有效的 Microsoft® SQL Server™ 表达式
WHEN when_expression	使用简单 CASE 格式时 input_expression 所比较的简单表达式。when_expression 是任意有效的 SQL Server 表达式。input_expression 和每个 when_expression 的数据类型必须相同，或者是隐性转换
n	占位符，表明可以使用多个 WHEN when_expression THEN result_expression 子句或 WHEN Boolean_expression THEN result_expression 子句
THEN result_expression	当 input_expression = when_expression 取值为 TRUE，或者 Boolean_expression 取值为 TRUE 时返回的表达式。result_expression 是任意有效的 SQL Server 表达式
ELSE else_result_expression	当比较运算取值不为 TRUE 时返回的表达式。如果省略此参数并且比较运算取值不为 TRUE，CASE 将返回 NULL 值，else_result_expression 是任意有效的 SQL Server 表达式。else_result_expression 和所有 result_expression 的数据类型必须相同，或者必须是隐性转换
WHEN Boolean_expression	使用 CASE 搜索格式时所计算的布尔表达式。Boolean_expression 是任意有效的布尔表达式

下面介绍简单 CASE 函数和 CASE 搜索函数两种格式的执行顺序。

1．简单 CASE 函数

（1）计算 input_expression，然后按指定顺序对每个 WHEN 子句的 input_expression=when_expression 进行计算。

（2）如果 input_expression = when_expression 为 TRUE，则返回 result_expression。

（3）如果没有取值为 TRUE 的 input_expression = when_expression，则当指定 ELSE 子句时，SQL Server 将返回 else_result_expression；若没有指定 ELSE 子句，则返回 NULL 值。

2．CASE 搜索函数

（1）按指定顺序为每个 WHEN 子句的 Boolean_expression 求值。

（2）返回第一个取值为 TRUE 的 Boolean_expression 的 result_expression。

（3）如果没有取值为 TRUE 的 Boolean_expression，则当指定 ELSE 子句时，SQL Server 将返回 else_result_expression；若没有指定 ELSE 子句，则返回 NULL 值。

【例 6.11】在 tb_Grade 表（见图 6.11）中，查询每个同学的成绩。如果成绩大于等于 90，显示

成绩优秀；如果成绩小于 90 大于等于 80，显示成绩良好；如果成绩小于 80 大于等于 70，显示成绩及格。否则将显示不及格。运行结果如图 6.12 所示。**（实例位置：资源包\TM\sl\6\11）**

ID	Name	Subjet	Grade
1	婷子	语文	90
2	小明	语文	85
3	小心	语文	70
1	婷子	数学	85
2	小明	数学	95
3	小心	数学	65

图 6.11　tb_Grade 表

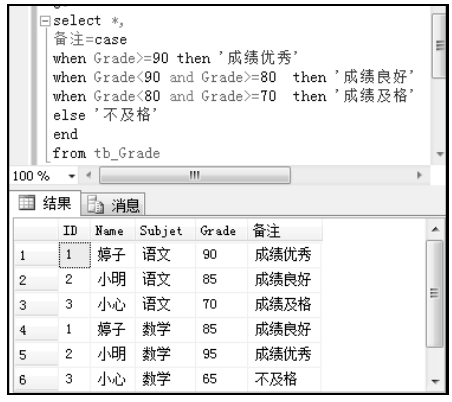


图 6.12　用 CASE 查询 tb_Grade 表

SQL 语句如下：

```
use db_Test
go
select *,
备注=case
when Grade>=90 then '成绩优秀'
when Grade<90 and Grade>=80    then '成绩良好'
when Grade<80 and Grade>=70    then '成绩及格'
else '不及格'
end
from tb_Grade
```

6.5.5　WHILE

WHILE 子句是 SQL 语句支持的循环结构。在条件为 TRUE 的情况下，WHILE 子句可以循环地执行其后的一条 SQL 命令。如果想循环执行一组命令，则需要配合 BEGIN…END 子句使用。WHILE 的语法如下：

```
WHILE<条件表达式>
BEGIN
    <命令行|程序块>
END
```

遇到 WHILE 子句，先判断条件表达式的值。当条件表达式的值为 TRUE 时，执行循环体中的命令行或程序块，遇到 END 子句自动地再次判断条件表达式的值的真假，决定是否执行循环体中的语句。只有当条件表达式的值为 FALSE 时，才结束执行循环体的语句。

【例 6.12】求 1～10 的整数的和。运行结果如图 6.13 所示。**（实例位置：资源包\TM\sl\6\12）**

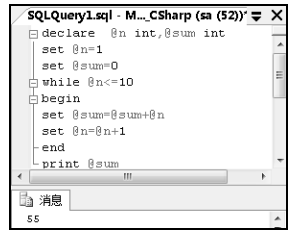


图 6.13　求 1～10 的整数的和

SQL 语句如下：

```
declare   @n int,@sum int
set @n=1
set @sum=0
while @n<=10
begin
set @sum=@sum+@n
set @n=@n+1
end
print @sum
```

6.5.6 WHILE…CONTINUE…BREAK

循环结构 WHILE 子句还可以用 CONTINUE 和 BREAK 命令控制 WHILE 循环中语句的执行。语法如下：

```
WHILE<条件表达式>
BEGIN
    <命令行|程序块>
    [BREAK]
    [CONTINUE]
    [命令行|程序块]
END
```

其中，CONTINUTE 命令可以让程序跳过 CONTINUE 命令之后的语句，回到 WHILE 循环的第一行命令。BREAK 命令则让程序完全跳出循环，结束 WHILE 命令的执行。

【例 6.13】求 1～10 偶数的和，并用 CONTINUE 控制语句输出。运行结果如图 6.14 所示。**（实例位置：资源包\TM\sl\6\13）**

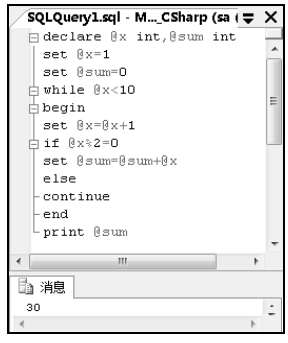


图 6.14　求 1～10 偶数的和

SQL 语句如下：

```
declare @x int,@sum int
set @x=1
set @sum=0
while @x<10
begin
```

```
set @x=@x+1
if @x%2=0
set @sum=@sum+@x
else
continue
end
print @sum
```

6.5.7　RETURN

RETURN 语句用于从查询或过程中无条件退出。RETURN 语句可在任何时候用于从过程、批处理或语句块中退出。位于 RETURN 之后的语句不会被执行。语法如下：

```
RETURN[整数值]
```

在括号内可指定一个返回值。如果没有指定返回值，SQL Server 根据程序执行的结果返回一个内定值。RETURN 命令返回的内定值及其含义如表 6.6 所示。

表 6.6　RETURN 命令返回的内定值及其含义

返　回　值	含　　义	返　回　值	含　　义
F	程序执行成功	−7	资源错误，如磁盘空间不足
−1	找不到对象	−8	非致命的内部错误
−2	数据类型错误	−9	已达到系统的极限
−3	死锁	−10 或−11	致命的内部不一致性错误
−4	违反权限原则	−12	表或指针破坏
−5	语法错误	−13	数据库破坏
−6	用户造成的一般错误	−14	硬件错误

【例 6.14】RETURN 语句实现退出功能。运行结果如图 6.15 所示。（**实例位置：资源包\TM\sl\6\14**）

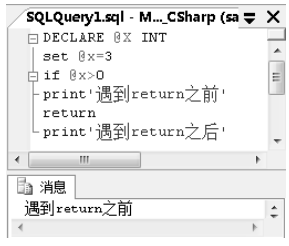


图 6.15　RETURN 实现退出功能

SQL 语句如下：

```
DECLARE @X INT
set @x=3
if @x>0
print'遇到 return 之前'
return
print'遇到 return 之后'
```

6.5.8 GOTO

GOTO 命令用来改变程序执行的流程，使程序跳到标识符指定的程序行再继续往下执行。语法如下：

```
GOTO  标识符
```

标识符需要在其名称后加上一个冒号“:”。例如：

```
"33: " "loving: "
```

【例 6.15】用 GOTO 语句实现跳转输出小于等于 3 的值。运行结果如图 6.16 所示。（**实例位置：资源包\TM\sl\6\15**）

SQL 语句如下：

```
DECLARE @X INT
SELECT @X=1
loving:
    PRINT @X
    SELECT @X=@X+1
WHILE  @X<=3   GOTO   loving
```

6.5.9 WAITFOR

WAITFOR 指定触发器、存储过程或事务执行的时间、时间间隔或事件；还可以用来暂时停止程序的执行，直到所设定的等待时间已过才继续往下执行。语法如下：

```
WAITFOR DELAY<'时间'>|TIME<'时间'>
```

其中，“时间”必须为 DATETIME 类型的数据，如“11:15:27”，但不能包括日期。各关键字含义如下。

☑ DELAY：用来设定等待的时间，最多可达 24 h。

☑ TIME：用来设定等待结束的时间点。

【例 6.16】等待 3 s 后显示“祝你节日快乐！”运行结果如图 6.17 所示。（**实例位置：资源包\TM\sl\6\16**）

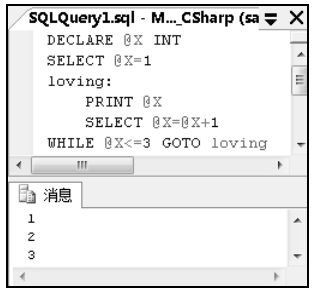


图 6.16　GOTO 实现跳转功能

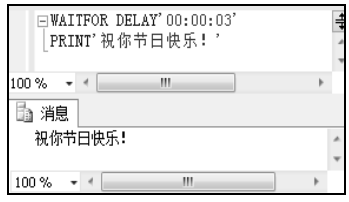


图 6.17　等待 3 s 后输出信息

SQL 语句如下：

```
WAITFOR DELAY'00:00:03'
PRINT'祝你节日快乐！'
```

【例 6.17】15:00 显示“《新三国演义》开始了”。（**实例位置：资源包\TM\sl\6\17**）

SQL 语句如下：

```
WAITFOR TIME'15:00:00'
PRINT'《新三国演义》开始了！"
```

6.6 常 用 命 令

本节介绍 SQL Server 中常用的命令，如常见的输出命令、数据备份命令、数据还原命令等，使用这些命令可以提高数据库的完整性和安全性。

6.6.1 DBCC

DBCC（DataBase Consistency Checker，数据库一致性检查）命令用于验证数据库完整性、查找错误和分析系统使用情况等。

DBCC 命令后必须加上子命令系统才知道要做什么。

1．DBCC CHECKALLOC

检查指定数据库的磁盘空间分配结构的一致性。

【例 6.18】执行 DBCC CHECKALLOC 命令检测 db_Test 数据库磁盘空间分配结构。运行结果如图 6.18 所示。（**实例位置：资源包\TM\sl\6\18**）

SQL 语句如下：

```
DBCC CHECKALLOC ('db_Test')
```

2．DBCC SHOWCONTIG

显示指定表的数据和索引的碎片信息。

【例 6.19】使用 OBJECT_ID 获得表 ID，使用 sys.indexes 获得索引 ID，使用 DBCC SHOWCONTIG 显示指定表的数据和索引的碎片信息。运行结果如图 6.19 所示。（**实例位置：资源包\TM\sl\6\19**）

图 6.18　检测 db_Test 数据库磁盘空间分配结构

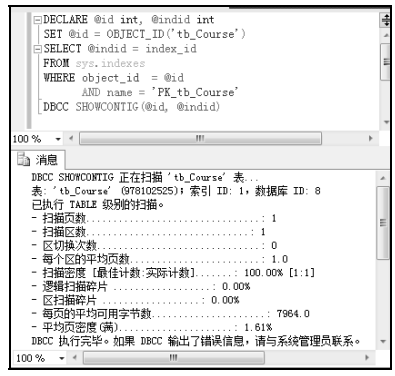


图 6.19　显示表数据和索引碎片信息

SQL 语句如下：

```
DECLARE @id int, @indid int
SET @id = OBJECT_ID('tb_Course')
SELECT @indid = index_id
FROM sys.indexes
WHERE object_id   = @id
        AND name = 'PK_tb_Course'
DBCC SHOWCONTIG(@id, @indid)
```

6.6.2 CHECKPOINT

CHECKPOINT 命令用于检查当前工作的数据库中被更改过的数据页或日志页，并将这些数据从数据缓冲器中强制写入硬盘。语法如下：

```
CHECKPOINT [ checkpoint_duration ]
```

参数 checkpoint_duration 表示以秒为单位指定检查点完成所需的时间。如果指定 checkpoint_duration，则 SQL Server 数据库引擎在请求的持续时间内尝试执行检查点。checkpoint_duration 必须是一个数据类型为 int 的表达式，并且必须大于零。如果省略该参数，SQL Server 数据库引擎将自动调整检查点持续时间，以便最大限度地降低对数据库应用程序性能的影响。

CHECKPOINT 权限默认授予 sysadmin 固定服务器角色、db_owner 和 db_backupoperator 固定数据库角色的成员且不可转让。

【例 6.20】使用 CHECKPOINT 命令检查 db_Test 数据库中被更改过的数据页或日志页，运行结果如图 6.20 所示。**（实例位置：资源包\TM\sl\6\20）**

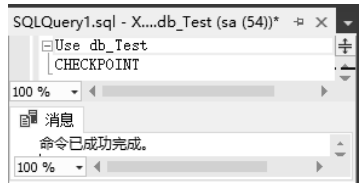


图 6.20　使用 CHECKPOINT 命令检查 db_Test 数据库

SQL 语句如下：

```
Use db_Test
CHECKPOINT
```

6.6.3 DECLARE

DECLARE 命令用于声明一个或多个局部变量、游标变量或表变量。语法如下：

```
DECLARE
      {
{{ @local_variable [AS] data_type } | [ = value ] }
     | { @cursor_variable_name CURSOR }
} [,...n]
     | { @table_variable_name [AS] <table_type_definition> | <user-defined table type> }
<table_type_definition> ::=
       TABLE ( { <column_definition> | <table_constraint> } [ ,... ]
        )
<column_definition> ::=
```

```
        column_name { scalar_data_type | AS computed_column_expression }
    [ COLLATE collation_name ]
    [ [ DEFAULT constant_expression ] | IDENTITY [ ( seed ,increment ) ] ]
    [ ROWGUIDCOL ]
    [ <column_constraint> ]
<column_constraint> ::=
    { [ NULL | NOT NULL ]
    | [ PRIMARY KEY | UNIQUE ]
    | CHECK ( logical_expression )
    }
```

DECLARE 命令的参数及其说明如表 6.7 所示。

表 6.7　DECLARE 命令的参数及其说明

参　数	描　述
@local_variable	变量的名称。变量名必须以@符开头。局部变量名称必须符合标识符规则
data_type	用户定义表类型或别名数据类型。变量的数据类型不能是 text、ntext 或 image
= value	以内联方式为变量赋值。值可以是常量或表达式，但它必须与变量声明类型匹配，或者可隐式转换为该类型
@cursor_variable_name	游标变量的名称
CURSOR	指定变量是局部游标变量
@table_variable_name	table 类型的变量的名称
<table_type_definition>	定义 table 数据类型。表声明包括列定义、名称、数据类型和约束
n	指示可以指定多个变量并对变量赋值的占位符
column_name	表中的列的名称
scalar_data_type	指定列是标量数据类型
computed_column_expression	定义计算列值的表达式
[COLLATE collation_name]	指定列的排序规则
DEFAULT	如果在插入过程中未显式提供值，则指定为列提供的值
constant_expression	用作列的默认值的常量、NULL 或系统函数
IDENTITY	指示新列是标识列
seed	装入表的第一行使用的值
increment	添加到以前装载的列标识值的增量值
ROWGUIDCOL	指示新列是行的全局唯一标识符列
NULL | NOT NULL	决定在列中是否允许 NULL 值的关键字
PRIMARY KEY	通过唯一索引对给定的一列或多列强制实现实体完整性的约束
UNIQUE	通过唯一索引为给定的一列或多列提供实体完整性的约束
CHECK	一个约束，该约束通过限制可输入一列或多列中的可能值强制实现域完整性
logical_expression	返回 TRUE 或 FALSE 的逻辑表达式

【例 6.21】定义一个变量，SQL 语句如下：

```
declare @x int
```

如果定义的变量是字符型，应该指定 data_type 表达式中其最大长度，否则系统认为其长度为 1。

【例 6.22】定义一个字符变量，SQL 语句如下：

```
declare @c char(8)
```

【例 6.23】使用 DECLARE 命令定义多个变量，中间用逗号相隔，SQL 语句如下：

```
declare @x int ,@y char(8),@z datetime
```

6.6.4 PRINT

PRINT 命令向客户端返回一个用户自定义的信息，即显示一个字符串（最长为 255 个字符）、局部变量或全局变量的内容。语法如下：

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

参数说明如下。

- ☑ msg_str：字符串或 Unicode 字符串常量。
- ☑ @local_variable：任何有效的字符数据类型变量。其数据类型必须为 char 或 varchar，或者必须能够隐式转换为这些数据类型。
- ☑ string_expr：返回字符串的表达式。可包括串联的文字值、函数或变量。

【例 6.24】定义一个变量，为其赋值。用 PRINT 命令显示变量，并生成字符串。运行结果如图 6.21 所示。（**实例位置：资源包\TM\sl\6\21**）

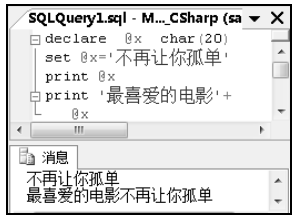


图 6.21 使用 print 命令输出信息

SQL 语句如下：

```
declare   @x   char(20)
set @x='不再让你孤单'
print @x
print '最喜爱的电影'+
    @x
```

6.6.5 RAISERROR

RAISERROR 命令用于在 SQL Server 系统中返回错误信息时同时返回用户指定的信息。语法如下：

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

RAISERROR 命令的参数及其说明如表 6.8 所示。

表 6.8　RAISERROR 命令的参数及其说明

参　数	描　述
msg_id	存储于 sysmessages 表中的用户定义的错误信息。用户定义错误信息的错误号应大于 50000。由特殊消息产生的错误是第 50000 号
msg_str	一条特殊消息，其格式与 C 语言中使用的 PRINTF 格式相似。此错误信息最多可包含 400 个字符。如果该信息包含的字符超过400个，则只能显示前397个并将添加一个省略号以表示该信息已被截断。所有特定消息的标准消息 ID 是 14000。msg_str 支持的格式有% [[flag] [width] [precision] [{h | l}]] type
@local_variable	一个可以为任何有效字符数据类型的变量，其中包含的字符串的格式化方式与 msg_str 相同。@local_variable 必须为 char 或 varchar，或者能够隐式转换为这些数据类型
severity	用户定义的与消息关联的严重级别。用户可以使用 0～18 的严重级别。19～25 的严重级别只能由 sysadmin 固定服务器角色成员使用。若要使用 19～25 的严重级别，必须将 WITH option 设置为 WITHLOG
state	1～127 的任意整数，表示有关错误调用状态的信息。state 的值默认为 1
argument	用于取代在 msg_str 中定义的变量或取代对应于 msg_id 的消息的参数。可以有 0 或更多的替代参数；替代参数的总数不能超过 20 个。每个替代参数可以是局部变量或这些任意数据类型，如 int1、int2、int4、char、varchar、binary 或 varbinary。不支持其他数据类型
WITH option	错误的自定义选项

6.6.6　READTEXT

READTEXT 命令用于读取 text、ntext 或 image 列中的值，从指定的位置开始读取指定的字符数。语法如下：

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

READTEXT 命令的参数及其说明如表 6.9 所示。

表 6.9　READTEXT 命令的参数及其说明

参　数	描　述
table.column	从中读取的表和列的名称。表名和列名必须符合标识符的规则。必须指定表名和列名，不过可以选择是否指定数据库名和所有者名
text_ptr	有效文本指针。text_ptr 必须是 binary(16)
offset	开始读取 text、image 或 ntext 数据之前跳过的字节数（使用 text 或 image 数据类型时）或字符数（使用 ntext 数据类型时）
size	要读取数据的字节数（使用 text 或 image 数据类型时）或字符数（使用 ntext 数据类型时）。如果 size 是 0，则表示读取了 4 KB 的数据
HOLDLOCK	使文本值一直锁定到事务结束。其他用户可以读取该值，但是不能对其进行修改

6.6.7　BACKUP

计算机在操作过程中难免出现意外，为了保证用户数据的安全性，防止数据库中的数据意外丢失，

应对数据库及时进行备份。BACKUP 命令用于将数据库内容或其事务处理日志备份到存储介质上（硬盘或磁带等）。BACKUP 命令的语法如下：

```
Backup Database { database_name | @database_name_var }
To < backup_device > [ ,...n ]
[ <MIRROR TO clause>][ next-mirror-to ]
[ WITH { DIFFERENTIAL | <general_WITH_options> [ ,...n ] }]
[ ; ]
```

BACKUP 命令的参数及其说明如表 6.10 所示。

表 6.10　BACKUP 命令的参数及其说明

参　数	描　述
Backup Database	关键字
{ database_name | @database_name_var }	备份事务日志、部分数据库或完整的数据库时所用的源数据库。如果作为变量（@database_name_var）提供，则可以将该名称指定为字符串常量（@database_name_var = database name）或指定为字符串数据类型（ntext 或 text 数据类型除外）的变量
To	关键字，用于指定备份设备
<backup_device>	一个备份设备，用于存储备份数据，其中 Disk 表示在磁盘上存储备份数据，Tape 表示在磁带上存储备份数据。physical_backup_device_name 表示磁盘或磁带上的物理路径，通常用于指定一个备份文件
n	一个占位符，表示可以在逗号分隔的列表中指定多个文件和文件组。数量没有限制
[next-mirror-to]	一个占位符，表示一个 BACKUP 语句除了包含一个 TO 子句外，最多还可包含 3 个 MIRROR TO 子句
DIFFERENTIAL	只能与 BACKUP DATABASE 一起使用，指定数据库备份或文件备份应该只包含上次完整备份后更改的数据库或文件部分。差异备份一般比完整备份占用更少的空间。对于上一次完整备份后执行的所有单个日志备份，使用该选项不必再进行备份

【例 6.25】把 db_Test 数据库备份到名称是 backup.bak 的备份文件中，运行结果如图 6.22 所示。**（实例位置：资源包\TM\sl\6\22）**

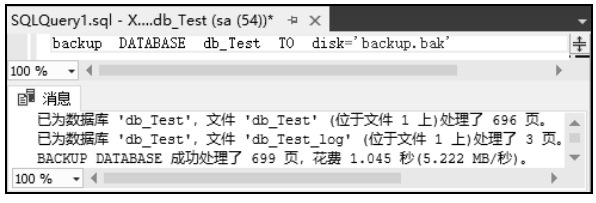


图 6.22　备份 db_Test 数据库

SQL 语句如下：

```
backup  DATABASE  db_Test  TO  disk='backup.bak'
```

6.6.8　RESTORE

如果数据库中的数据发生丢失或破坏，操作员应该及时还原数据库，尽可能地减少损失。RESTORE 命令用来将数据库或其事务处理日志备份文件由存储介质还原到 SQL Server 系统中。

通过 RESTORE 命令还原数据库，可以执行下列还原方案。

☑　基于完整数据库备份还原整个数据库（完整还原）。

☑　还原数据库的一部分（部分还原）。

☑　将特定文件或文件组还原到数据库（文件还原）。

☑　将特定页面还原到数据库（页面还原）。

☑　将事务日志还原到数据库（事务日志还原）。

☑　将数据库恢复到数据库快照捕获的时间点。

完整数据库还原的语法如下：

```
RESTORE DATABASE { database_name | @database_name_var }
 [ FROM <backup_device> [ ,...n ] ]
 [ WITH
    {
          [ RECOVERY | NORECOVERY | STANDBY =
          {standby_file_name | @standby_file_name_var }
         ]
     | ,   <general_WITH_options> [ ,...n ]
        | , <replication_WITH_option>
    | , <change_data_capture_WITH_option>
        | , <service_broker_WITH options>
        | , <point_in_time_WITH_options—RESTORE_DATABASE>
        } [ ,...n ]
 ]
[;]
```

RESTORE 命令的参数及其说明如表 6.11 所示。

表 6.11　RESTORE 命令的参数及其说明

参　数	描　述
RESTORE DATABASE	指定目标数据库
{ database_name | @database_name_var}	将日志或整个数据库备份还原到数据库
FROM <backup_device> [,...n]	通常指定要从哪些备份设备还原备份
RECOVERY	指示还原操作回滚任何未提交的事务。在恢复进程后即可随时使用数据库。如果既没有指定 NORECOVERY 和 RECOVERY，也没有指定 STANDBY，则默认为 RECOVERY
NORECOVER	指示还原操作不回滚任何未提交的事务。如果稍后必须应用另一个事务日志，则应指定 NORECOVERY 或 STANDBY 选项。如果既没有指定 NORECOVERY 和 RECOVERY，也没有指定 STANDBY，则默认为 RECOVERY
STANDBY = standby_file_name	指定一个允许撤销恢复效果的备用文件。STANDBY 选项可以用于脱机还原（包括部分还原），但不能用于联机还原。尝试为联机还原操作指定 STANDBY 选项将导致还原操作失败。如果必须升级数据库，也不允许使用 STANDBY 选项
<general_WITH_options> [,...n]	RESTORE DATABASE 和 RESTORE LOG 语句均支持常规 WITH 选项。一个或多个辅助语句也支持其中某些选项
replication_WITH_option>	此选项只适用于在创建备份时对数据库进行了复制的情况

【例 6.26】还原例 6.25 中备份了 db_Test 数据库的备份文件 backup.bak，运行结果如图 6.23 所示。**（实例位置：资源包\TM\sl\6\23）**

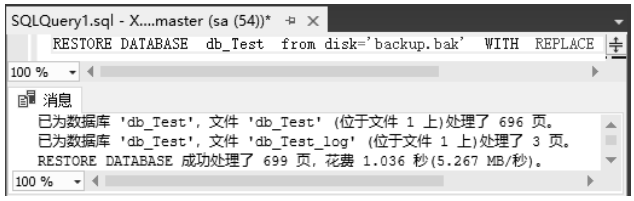


图 6.23　还原备份文件 backup.bak

SQL 语句如下：

```
RESTORE DATABASE    db_Test    from disk='backup.bak'    WITH    REPLACE
```

说明

执行还原操作时，需要将使用的数据库更改为 master 数据库。

6.6.9　SELECT

SELECT 语句除了有强大的查询功能外，还可用于给变量赋值。语法如下：

```
SELECT { @local_variable { = | += | -= | *= | /= | %= | &= | ^= | |= } expression } [ ,...n ] [ ; ]
```

参数说明如下。

☑ @local_variable：要为其赋值的声明变量。

☑ =：将右边的值赋给左边的变量。

☑ { = | += | -= | *= | /= | %= | &= | ^= | |= }：复合赋值运算符。

- ➢ +=：相加并赋值。
- ➢ −=：相减并赋值。
- ➢ *=：相乘并赋值。
- ➢ /=：相除并赋值。
- ➢ %=：取余并赋值。
- ➢ &=：“位与”并赋值。
- ➢ ^=：“位异或”并赋值。
- ➢ |=：“位或”并赋值。

☑ expression：任何有效的表达式。此参数包含一个标量子查询。

说明

SELECT @local_variable 通常用于将单个值返回到变量中。如果 expression 是列的名称，则可返回多个值。如果 SELECT 语句返回多个值，则将返回的最后一个值赋给变量。如果 SELECT 语句没有返回行，变量将保留当前值。如果 expression 是不返回值的标量子查询，则将变量设为 NULL。

【例 6.27】给一个变量赋值，SQL 语句如下：

```
declare   @x int
select @x=1
print @x
```

一个 SELECT 语句可以初始化多个局部变量。

【例 6.28】一次给多个变量赋值，SQL 语句如下：

```
declare   @x int,@y char(20),@z datetime
select @x=1,@y='LOVING',@z='2001/01/01'
print @x
print @y
print @z
```

【例 6.29】对 tb_Grade 表中的 Subjet 进行查询，并赋值给变量 courses。运行结果如图 6.24 所示。**（实例位置：资源包\TM\sl\6\24）**

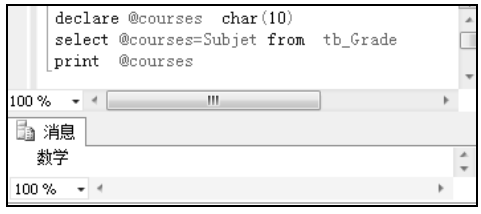


图 6.24　将查询的课程内容赋给变量

SQL 语句如下：

```
use   db_Test
declare @courses   char(10)
select @courses= Subjet
from   tb_Grade
print   @courses
```

6.6.10　SET

SET 命令有两种用法，具体说明如下。

1．用于给局部变量赋值

在用 DECLARE 命令声明后，所有的变量都被赋予初值 NULL。这时需要用 SET 命令给变量赋值。语法如下：

```
SET{{ @ local_variable=expression}
      |{ @ cursor_variable={@ cursor_variable|cursor_name
            |{CURSOR[FORWARD_ONLY|SCROLL]
                [STATIC|KEYSET|DYNAMIC|FAST_FORWARD]
                [READ_ONLY|SCROLL_LOCKS|OPTIMISTIC]
                [TYPE_VARNING]
            FOR select_statement
```

```
            [FOR {READ ONLY|UPDATE[OF column_name[,…n]]}
            ]
        }
    }}
}
```

【例 6.30】定义一个变量，并为该变量赋值，SQL 语句如下：

```
declare  @x  int
set  @x=1
print @x
```

SET 命令与 SELECT 命令赋值不同的是，SET 命令一次只能给一个变量赋值，但 SET 命令功能更强且更严密，因此，推荐使用 SET 命令给变量赋值。

2．用于执行 SQL 命令时 SQL Server 的处理选项设定

如果要对计算列或索引视图创建和操作索引，必须将 SET 选项 ARITHABORT、CONCAT_NULL_YIELDS_NULL、QUOTED_IDENTIFIER、ANSI_NULLS、ANSI_PADDING 和 ANSI_WARNINGS 设置为 ON，并将选项 NUMERIC_ROUNDABORT 设置为 OFF。

当从批处理或其他存储过程执行某个存储过程时使用的选项值，就是当前包含该存储过程的数据库中设置的选项值。

6.6.11 SHUTDOWN

SHUTDOWN 命令用于立即停止 SQL Server 的执行。语法如下：

```
SHUTDOWN[WITH NOWAIT]
```

参数说明如下。

WITH NOWAIT：当使用 NOWAIT 参数时，SHUTDOWN 命令立即终止所有的用户过程，并在对每一现行的事务发生一个回滚后退出 SQL Server。当没有用 NOWAIT 参数时，SHUTDOWN 命令将按以下步骤执行。

（1）终止任何用户登录 SQL Server。

（2）等待尚未完成的 SQL 命令或存储过程执行完毕。

（3）在每个数据库中执行 CHECKPOINT 命令。

（4）停止 SQL Server 的执行。

【例 6.31】使用 SHUTDOWN 命令停止 SQL Server 服务器，运行结果如图 6.25 所示。**（实例位置：资源包\TM\sl\6\25）**

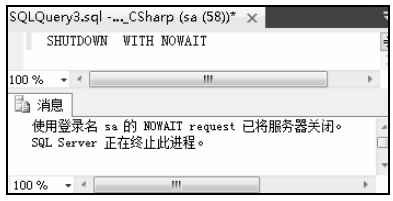


图 6.25　停止 SQL Server 服务器

SQL 语句如下：

```
SHUTDOWN　WITH NOWAIT
```

6.6.12　WRITETEXT

WRITETEXT 命令允许对数据类型为 text、ntext 或 image 的列进行交互式更新。WRITETEXT 语句不能用在视图中的 text、ntext 和 image 列上。语法如下：

```
WRITETEXT { table.column text_ptr }
[ WITH LOG ] { data }
```

参数说明如下。

- ☑ table.column：要更新的表和 text、ntext 或 image 列的名称。表名和列名必须符合标识符的规则。指定数据库名和所有者名是可选的。
- ☑ text_ptr：指向 text、ntext 或 image 数据的指针的值。text_ptr 的数据类型必须为 binary(16)。若要创建文本指针，可对 text、ntext 或 image 列用非 NULL 数据执行 INSERT 或 UPDATE 语句。
- ☑ WITH LOG：在 SQL Server 中忽略。日志记录由数据库的实际恢复模型决定。
- ☑ data：要存储的实际 text、ntext 或 image 数据。data 可以是字面值，也可以是变量。对于 text、ntext 和 image 数据，可以用 WRITETEXT 交互插入，文本的最大长度是 120 KB。

【例 6.32】将文本指针放到局部变量@val 中，使用 WRITETEXT 命令将新的文本字符串放到@val 所指向的行中。

SQL 语句如下：

```
USE pubs
EXEC sp_dboption 'pubs', 'select into/bulkcopy', 'true'
DECLARE @val binary(16)
SELECT @val = TEXTPTR(pr_info)
FROM pub_info p1, publishers p2
WHERE p2.pub_id = p1.pub_id
    AND p2.pub_name = 'New Moon Books'
WRITETEXT pub_info.pr_info @val 'New Moon Books (NMB) has just released another
 top ten publication. With the latest publication this makes NMB the hottest new
publisher of the year!'
EXEC sp_dboption 'pubs', 'select into/bulkcopy', 'false'
```

6.6.13　USE

USE 命令用于在前工作区打开或关闭数据库。语法如下：

```
USE {数据库}
```

数据库：用户上下文要切换到的数据库名。数据库名必须符合标识符的规则。

要使用 db_Test 数据库时，必须选中该数据库，一种方法是在工具栏上“数据库名称”列表框中选择数据库；另一种方法是使用 USE 命令打开。

【例 6.33】查询 Student 数据表中的全部信息。

SQL 语句如下：

```
USE db_Test
SELECT * FROM Student
```

如果没有在工具栏“数据库名称”列表中选择数据库，也没有使用 USE 命令打开数据库，就会提示如图 6.26 所示的错误。

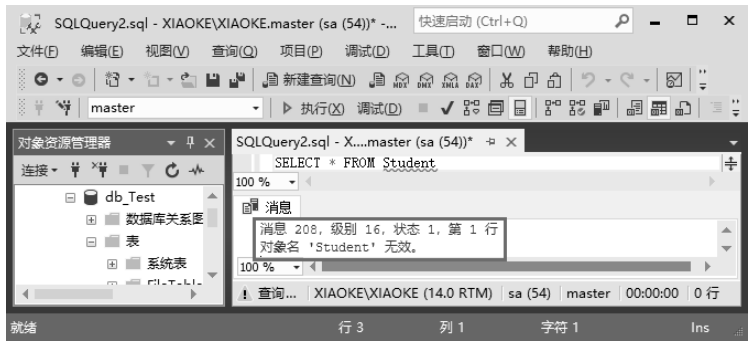


图 6.26　提示对象名无效

6.7　小　　结

本章介绍了 SQL 的基础知识，包括 SQL 的数据类型、常量、变量、流程控制语句和常用的命令等。学习本章时，读者需要了解与 SQL 相关的概念，区分常量和变量，熟悉常用命令，熟练地掌握流程控制语句的用法，因为流程控制语句可以提高 SQL 语句的处理能力。

6.8　实践与练习

1．通过使用分组查询中的 GROUPING SETS 运算符，在 books 表中，查询分别按照书名、出版社分组的销售价格。（**答案位置：资源包\TM\sl\6\26**）

2．使用 IF EXISTS 语句检测 Employee 表中性别为女、姓名为“赵小小”的人是否存在。（**答案位置：资源包\TM\sl\6\27**）