第5章

Transact-SQL程序设计进阶

Transact-SQL 包含许多 SQL 不具备的扩展编程功能，主要包括批处理、脚本、流程控制语句，以及存储过程和触发器等，本章主要介绍这些内容。

5.1 批处理和脚本

批处理是数据库系统中很重要的一个功能，能够对系统的性能进行有效优化。脚本将一组 Transact-SQL 语句以文本文件的形式存储，能够提高数据访问的效率，并进行相关的数据处理。事务是数据库系统上执行并发操作时的最小控制单元，能将逻辑单元的一组操作绑定在一起，便于使服务器保持数据的完整性。

5.1.1 批处理概述

1. 批处理的概念

批处理是指一次性地执行包含一条或多条 Transact-SQL 语句的语句组，它表示用户提交给数据库引擎的工作单元。批处理是作为一个单元进行分析和执行的，它要经历的处理阶段有分析（语法检查）、解析（检查引用的对象和列是否存在、是否具有访问权限）、优化（作为一个执行单元）。SQL Server 将批处理语句编译为单个可执行的单元，称为执行计划，执行计划中的语句每次执行一条，这种批处理方式有助于节省执行时间。

在书写批处理时，GO 语句作为批处理命令的结束标志，当编译器读取到 GO 语句时会将 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 Transact-SQL 语句的组成部分，只是一个表示批处理结束的前端指令。

2. 批处理的规则

下面的规则适用于批处理的使用：

(1) 不能被组合在同一个批处理中的语句包括创建默认值(CREATE DEFAULT)、创建函数(CREATE FUNCTION)、创建存储过程(CREATE PROCEDURE)、创建规则(CREATE RULE)、创建模式(CREATE SCHEMA)、创建触发器(CREATE TRIGGER)和创建视图(CREATE VIEW)。

(2) 批处理以 CREATE 语句开始，所有跟在该批处理后的其他语句将被解释为第一个 CREATE 语句定义的一部分。

(3) 不能在同一个批处理中更改表结构(例如修改字段名、新增字段、新增或更改约束等)，然后引用新列。因为 SQL Server 可能还不知道架构定义发生了变化，导致出现解析错误。

(4) 不能在同一个批处理中删除一个对象之后再次引用该对象。

(5) 不可在将规则和默认值绑定到表字段或自定义字段上之后立即在同一个批处理中使用。

(6) 使用 SET 语句设置的某些 SET 选项不能应用于同一个批处理中的查询。

(7) 如果批处理中的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。如果 EXECUTE 语句不是批处理中的第一条语句，则 EXECUTE 关键字必须保留。

3. 指定批处理的方法

指定批处理的方法有以下 4 种：

(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。

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

(3) 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。

(4) 由 SP_EXECUTESQL 存储过程执行的字符串是一个批处理，并编译为一个执行计划。

用户需要注意以下几点：

(1) 若应用程序发出的批处理过程中含有 EXECUTE 语句，则已执行字符串或存储过程的执行计划将和包含 EXECUTE 语句的执行计划分开执行。

(2) SP_EXECUTESQL 存储过程所执行的字符串生成的执行计划与 SP_EXECUTESQL 调用的批处理执行计划分开执行。

(3) 若批处理中的语句激活了触发器，则触发器的执行将和原始的批处理分开执行。

4. 批处理的结束与退出

1) 执行批处理语句

用 EXECUTE 语句执行标量值的用户自定义函数、系统过程、用户自定义存储过程

或扩展存储过程，同时支持 Transact-SQL 批处理内字符串的执行。

2）批处理结束语句

执行计划中的语句逐条执行，每次执行一条。所有的批处理语句都以 GO 命令作为结束的标志。当编译器读到 GO 时，它会把 GO 前面所有的语句作为一个批处理进行处理，并打包成一个数据包发送给服务器。

GO 命令和 Transact-SQL 语句不能在同一行，否则无法识别。但在 GO 命令行中可包含注释。用户必须遵照使用批处理的规则。

SQL Server 2005 以及更高的版本中对 GO 命令这个客户端工具进行了增强，让它可以支持一个正整数参数，表示 GO 之前的批处理将执行指定的次数。当需要重复执行批处理时，就可以使用这个增强后的命令。该命令使用的语法格式为：

```
GO [count]
```

其中，count 为一个正整数，用于指定 GO 之前的批处理将执行的次数。

【例 5-1】 查询 StudentMIS 数据库中学生的基本信息。

```
USE StudentMIS            -- 将当前使用的数据库切换到"StudentMIS"
GO
SELECT * FROM Student     -- 从 Student 表中查询学生信息
GO
```

3）批处理退出语句

批处理退出语句的基本语法格式为：

```
RETURN [整型表达式]
```

该语句可无条件终止查询、存储过程或批处理的执行。存储过程或批处理不执行 RETURN 之后的语句。当存储过程使用该语句时，RETURN 语句不能返回空值。用户可用该语句指定返回调用应用程序、批处理或存储过程的整数值。

5.1.2 脚本

脚本是存储在文件中的一系列 Transact-SQL 语句，是一系列顺序提交的批处理，脚本文件保存时的扩展名为.sql，该文件是一个纯文本文件。在脚本文件中可包含一个或多个批处理，GO 作为批处理结束语句，若脚本中无 GO 语句，则作为单个批处理。

使用脚本可以将创建和维护数据库时进行的操作保存在磁盘文件中，方便以后重复使用该段代码，还可以将此代码复制到其他计算机上执行。因此，对于经常操作的数据库，保存相应的脚本文件是一个良好的使用习惯。

5.2 流程控制语句

一般结构化程序设计语言的基本结构有顺序结构、条件分支结构和循环结构。Transact-SQL 语言也提供了类似的功能。Transact-SQL 提供了称为流程控制的特殊关

键字，用于控制 Transact-SQL 语句、语句块或存储过程的执行流程。

流程控制语句就是用来控制程序执行流程的语句，使用流程控制语句可以在程序中组织语句的执行流程，提高编程语言的处理能力。SQL Server 提供的流程控制语句如表 5-1 所示。

表 5-1 流程控制语句

流程控制语句	说 明
BEGIN…END	定义语句块
BREAK	跳出循环语句
CASE	分支语句
CONTINUE	重新开始循环语句
GOTO	无条件跳转语句
IF…ELSE	条件处理语句，如果条件成立，执行 IF 语句；否则执行 ELSE 语句
RETURN	无条件退出语句
WAITFOR	延迟语句
WHILE	循环语句

5.2.1 BEGIN…END 语句块

BEGIN…END 语句用于将多个 Transact-SQL 语句组合为一个逻辑块。在执行时，该逻辑块作为一个整体被执行。其语法格式如下：

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

其中，BEGIN 和 END 是控制语句的关键字，分别表示语句块的开始和结束；{sql_statement|statement_block}是任何有效的 Transact-SQL 语句或以语句块定义的语句分组。在任何时候，当流程控制语句必须执行一个包含两条或两条以上 Transact-SQL 语句的语句块时，都可以使用 BEGIN 和 END 语句。它们必须成对使用，任何一条语句均不能单独使用。BEGIN…END 语句块的功能类似于程序设计语言中的{…}。此外，BEGIN…END 语句块可以嵌套使用。

下面几种情况经常要用到 BEGIN…END 语句块：

(1) WHILE 循环需要包含语句块。

(2) CASE 函数的元素需要包含语句块。

(3) IF 或 ELSE 子句需要包含语句块。

在上述情况下，如果只有一条语句，则不需要使用 BEGIN…END 语句块。

5.2.2 IF…ELSE 语句

使用 IF…ELSE 语句，可以有条件地执行语句。其语法格式如下：

```
IF <逻辑表达式>
  <Transact-SQL 语句或用语句块定义的语句分组>
[ELSE
  <Transact-SQL 语句或用语句块定义的语句分组>]
```

其中,<逻辑表达式>可以返回 TRUE 或 FALSE。如果<逻辑表达式>中含有 SELECT 语句,必须用圆括号将 SELECT 语句括起来。

IF…ELSE 语句的执行方式是:如果逻辑表达式的值为 TRUE,则执行 IF 后面的语句块;否则执行 ELSE 后面的语句块。

在 IF…ELSE 语句中,IF 和 ELSE 后面的子句都允许嵌套,嵌套层数不受限制。但是,嵌套最好不要超过 3 层,否则会降低程序的可读性。

5.2.3 CASE 语句

使用 CASE 语句可以实现程序的多重分支选择。虽然使用 IF…ELSE 语句也能够实现多重分支结构,但是使用 CASE 语句的程序可读性更强。在 SQL Server 2016 中,CASE 语句有两种格式。

(1) 简单 CASE 语句:将某个表达式与一组简单表达式进行比较以确定结果。其语法格式为:

```
CASE <输入表达式>
  WHEN <when 表达式> THEN <结果表达式>
  WHEN <when 表达式> THEN <结果表达式>
  […n]
  [ELSE <else 结果表达式>]
END
```

简单 CASE 语句的执行过程为:将<输入表达式>与各个<when 表达式>进行比较,如果相等,则返回对应的<结果表达式>的值,然后跳出 CASE 语句,不再执行后面的语句;如果没有<输入表达式>等于<when 表达式>的值,则返回<else 结果表达式>的值。如果没有 ELSE 子句,则返回 NULL。需要注意的是,各条件分支返回的<结果表达式>的数据类型必须一致,最好明确地写上 ELSE 子句。

【例 5-2】 显示出版社数据库 pubs 中作者所在州的情况。

```
SELECT au_fname, au_lname,
  CASE state
    WHEN 'CA' THEN 'California'
    WHEN 'KS' THEN 'Kansas'
    WHEN 'TN' THEN 'Tennessee'
    WHEN 'MI' THEN 'Michigan'
    WHEN 'IN' THEN 'Indiana'
    WHEN 'MD' THEN 'Maryland'
  END AS StateName
FROM authors WHERE au_fname LIKE 'M%'
```

执行结果如下：

```
au_fname      au_lname      StateName
 ---------------------------------
Marjorie      Green         California
Michael       O'Leary       California
Meander       Smith         Kansas
Morningstar   Greene        Tennessee
Michel        DeFrance      Indiana
```

(2) 搜索 CASE 语句：计算一组布尔表达式以确定结果。其基本语法格式为：

```
CASE
  WHEN <逻辑表达式> THEN <结果表达式>
  WHEN <逻辑表达式> THEN <结果表达式>
  [ …n]
  [ ELSE < else 结果表达式>]
END
```

搜索 CASE 语句的执行过程为：如果<逻辑表达式>的值为 TRUE，则返回 THEN 后面的<结果表达式>，然后跳出 CASE 语句；否则继续测试下一个 WHEN 后面的<逻辑表达式>。如果所有的 WHEN 后面的<逻辑表达式>均为 FALSE，则返回 ELSE 后面的< else 结果表达式>。如果没有 ELSE 子句，则返回 NULL。WHEN 子句中的“<逻辑表达式>”就是类似“列＝值”这样，返回值为真值(TRUE、FALSE、UNKNOWN)的表达式。用户也可以将其看作使用＝、<>或者 LIKE、BETWEEN 等谓词编写出来的表达式。用户在编写 SQL 语句的时候需要注意，在发现为真的 WHEN 子句时，CASE 表达式的真假值判断就会中止，而剩余的 WHEN 子句会被忽略。为了避免引起不必要的混乱，在使用 WHEN 子句时要注意条件的排他性。

【例 5-3】 在 StudentMIS 数据库中，采用“优”“良”“中”“差”“不及格”五级打分制来显示 5 名学生成绩表的情况，结果如图 5-1 所示。

```
SELECT TOP 5 StuNo AS 学号, CNo AS 课程号,
  CASE
    WHEN Score >= 90 THEN '优'
    WHEN Score >= 80 THEN '良'
    WHEN Score >= 70 THEN '中'
    WHEN Score >= 60 THEN '差'
    ELSE '不及格'
  END
  AS '成绩等级'
FROM StudentMIS.dbo.SC
```

执行结果如图 5-1 所示。

【例 5-4】 在 StudentMIS 数据库中，统计每门课程选修的男生总数和女生总数，结果如图 5-2 所示。

```
SELECT CNo AS 课程号,
  SUM( CASE WHEN sex = '男' THEN 1 ELSE 0 END) AS 男生数,
  SUM (CASE WHEN sex = '女' THEN 1 ELSE 0 END) AS 女生数
FROM Student AS S INNER JOIN SC ON S.StuNo = SC.StuNo GROUP BY CNo
```

	学号	课程号	成绩等级
1	41756002	050115	良
2	41756002	050116	良
3	41756002	050118	优
4	41756002	050218	优
5	41756002	050264	良

图 5-1　例 5.3 的查询结果

	课程号	男生数	女生数
1	050115	3	1
2	050116	2	1
3	050118	3	1
4	050218	3	1
5	050264	3	1

图 5-2　例 5.4 的查询结果

上面这段代码所做的是分别统计选修每门课程的“男生”(即 sex＝'男')人数和“女生”(即 sex＝'女')人数。也就是说,这里是将“行结构”的数据转换成了“列结构”的数据。除了 SUM(),COUNT()、AVG()等聚合函数也都可以用于将行结构的数据转换成列结构的数据。

【例 5-5】 在 StudentMIS 数据库中,为了使学生的成绩更符合正太分布且方差尽可能小,将成绩高于 90 分的降低 5%,将成绩低于 80 分的提高 10%。

```
UPDATE SC SET
Score = CASE WHEN Score > 90 THEN Score * 0.95
             WHEN Score < 80 THEN Score * 1.1
             ELSE Score END
```

这样通过一条语句即可实现上述业务规则。尽管对例 5.5 分别执行下面两个 UPDATE 操作的结果与上面的执行效果是一致的,但这种逻辑是不正确的。问题在于,第一次的 UPDATE 操作执行后,学生的成绩发生了变化,如果继续拿它当作第二次 UPDATE 的判定条件,结果就会不准确。所以,例 5.5 只能使用 CASE 语句进行成绩的更新。

```
UPDATE SC SET Score = Score  *  0.95 WHERE Score > 90      -- 条件 1
UPDATE SC SET Score = Score  *  1.1 WHERE Score < 80       -- 条件 2
```

需要注意的是,SQL 语句最后一行的 ELSE Score 非常重要,必须写上。因为如果没有它,Score 介于 80 分和 90 分之间的学生成绩就会被更新成 NULL。

5.2.4　WHILE 语句

WHILE 语句可以设置重复执行 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。用户可以使用 BREAK 和 CONTINUE 关键字在循环内部控制 WHILE 循环中语句的执行。其语法格式如下:

```
WHILE <逻辑表达式>
BEGIN
  <Transact - SQL 语句或用语句块定义的语句分组>
  [ BREAK ]
  [ CONTINUE ]
  <Transact - SQL 语句或用语句块定义的语句分组>
END
```

其中,BREAK 语句无条件地退出 WHILE 循环,即从最内层的 WHILE 循环中退出,将执行出现在 END 关键字后面的语句,END 关键字为循环结束标记。CONTINUE 结束本次循环,进入下次循环,也就是使 WHILE 循环重新开始执行,忽略 CONTINUE 关键字后面的任何语句。

WHILE 语句的执行方式为:如果逻辑表达式的值为 TRUE,则反复执行 WHILE 语句后面的语句块;否则将跳过后面的语句块。

【例 5-6】 计算并输出 1+2+3+…+100 的和。

```
DECLARE @sum int, @i int
SET @i = 0
SET @sum = 0
WHILE @i <= 100
  BEGIN
  SET @sum = @sum + @i
  SET @i = @i + 1
END
PRINT '1～100 的和为:' + CAST(@sum AS char(25))
```

执行结果为:

```
1～100 的和为: 5050
```

5.2.5 GOTO 语句

GOTO 语句可以实现无条件跳转,让执行流程跳转到 SQL 代码中的指定标签处。其语法格式为:

```
GOTO 标签名
```

GOTO 语句的执行方式为:遇到 GOTO 语句后,直接跳转到“标签名”处继续执行,而 GOTO 后面的语句将不被执行。

5.2.6 RETURN 语句

使用 RETURN 语句,可以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,而不执行位于 RETURN 之后的语句。其语法格

式为：

```
RETURN [整数值]
```

5.2.7 WAITFOR 语句

使用 WAITFOR 语句，可以在指定的时间或者过了一定时间后执行语句块、存储过程或者事务。其语法格式为：

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

DELAY 指示 SQL Server 一直等到指定的时间过去，最长可达 24 小时。'时间'是要等待的时间。用户可以按 datetime 数据可接受的格式指定'时间'，也可以用局部变量指定此参数。注意不能指定日期，因此在 datetime 值中不允许有日期部分。TIME 指示 SQL Server 等待到指定时间。

【例 5-7】 等待 30 秒后对 SC 表执行 SELECT 语句。

```
WAITFOR DELAY '00:00:30'
SELECT * FROM StudentMIS.dbo.SC
```

【例 5-8】 指定在 15：30：00 时执行一个输出当前时间的语句。

```
WAITFOR TIME '15:30:00'
PRINT '现在是 15:30:00'
```

执行后，等计算机上的时间到了 15：30：00 时将出现结果“现在是 15：30：00”。

5.2.8 TRY…CATCH 语句

TRY…CATCH 语句实现类似于 Java 和 C++语言中的异常处理。Transact-SQL 语句或用语句块定义的语句分组可以包含在 TRY 中，如果 TRY 内部发生错误，则会将控制传递给 CATCH 中包含的另一个语句分组。其语法格式如下：

```
BEGIN TRY
  <Transact-SQL 语句或用语句块定义的语句分组>
END TRY
BEGIN CATCH
  <Transact-SQL 语句或用语句块定义的语句分组>
END CATCH
```

在 CATCH 模块中，可以使用下面的函数来实现错误处理。

(1) ERROR_NUMBER()：返回错误号。

(2) ERROR_MESSAGE()：返回错误消息的完整文本。

(3) ERROR_SEVERITY()：返回错误严重性。

(4) ERROR_STATE():返回错误状态号。

(5) ERROR_LINE():返回导致错误的例程中的行号。

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

5.2.9 PRINT 语句

PRINT 语句用于向客户端返回用户信息。PRINT 语句只允许显示常量、表达式或变量,不允许显示列名。它的语法格式如下:

```
PRINT 字符串|局部变量|字符串表达式
```

5.3 存储过程

5.3.1 存储过程概述

1. 存储过程的概念

存储过程(Stored Procedure)是一组为了完成特定功能、可以接收和返回用户参数的 Transact-SQL 语句的预编译集合,经过编译后存储在数据库中,以一个名称存储并作为一个单元处理。存储过程存储在数据库内,用户通过指定存储过程名及给出参数进行调用执行,而且允许用户声明变量、带参数执行以及拥有其他强大的编程功能。存储过程在第一次执行时进行语法检查和编译,执行后它的执行计划就驻留在高速缓存中,用于后续调用。存储过程可以接受和输出参数、返回执行存储过程的状态值,还可以嵌套调用。存储过程在数据库开发过程以及数据库维护和管理等任务中有非常重要的作用。

2. 存储过程的特点

存储过程可包含流程控制、逻辑以及对数据库的查询。它们可以接收参数、输出参数、返回单个或多个结果集以及返回值。与单纯的 Transact-SQL 语句相比,存储过程具有以下优点:

(1) 允许模块化的程序设计。存储过程被创建、存储在数据库中,可以在程序中被多次调用。用户可以在自己的存储过程内引用其他存储过程,这可以简化一系列复杂语句;而且存储程序可以独立于应用程序进行修改,极大地提高了程序的可移植性。

(2) 执行速度快,改善系统性能。存储过程在创建时即在服务器上进行编译和优化。程序调用一次后,它的执行计划就驻留在高速缓存中,下次调用时可以直接执行;而批处理的 Transact-SQL 语句在每次运行时都要进行编译和优化,因此速度相对较慢。

(3) 有效降低网络流量。用户可以在单个存储过程中执行一系列 Transact-SQL 语句。有了存储过程,在网络上只要一条语句就能执行一个存储过程,因此有效地减少了网络流量,提高了应用程序的执行效率。

(4) 保证数据库的安全性。通过隔离和加密的方法提高了数据库的安全性,通过授权可以让用户只能执行存储过程而不能直接访问数据库对象。

3. 存储过程的分类

在 SQL Server 2016 中存储过程分为 3 类，即系统存储过程、用户自定义存储过程以及扩展存储过程。

(1) 系统存储过程主要存放在 master 数据库中，并以“sp_”为前缀名。系统存储过程主要是从系统表中获取信息，从而为系统管理员管理 SQL Server 提供支持。SQL Server 2016 提供了一千多个系统存储过程，下面对常用的系统存储过程做一个简单的介绍。

- sp_tables：返回可在当前环境中查询的对象列表。这代表可在 FROM 子句中出现的任何对象。
- sp_stored_procedures：返回当前环境中的存储过程列表。
- sp_rename：在当前数据库中更改用户创建对象的名称。此对象可以是表、索引、列、别名等数据类型。
- sp_renamedb：更改数据库的名称。
- sp_help：报告有关数据库对象(sys. sysobjects 兼容视图中列出的所有对象)、用户定义数据类型或 SQL Server 2016 提供的数据类型的信息。
- sp_helptext：表示用户定义规则的定义、默认值、未加密的 Transact-SQL 存储过程、用户定义 Transact-SQL 函数、触发器、计算列、CHECK 约束、视图或系统对象(例如系统存储过程)。
- sp_who：提供有关 SQL Server Database Engine 实例中的当前用户和进程的信息。
- sp_password：为 SQL Server 登录名添加或更改密码。

(2) 用户自定义存储过程是由用户创建的、存放在用户创建的数据库中，并能完成某些特定功能的存储过程。本节主要介绍用户自定义存储过程。在 SQL Server 2016 中，用户自定义存储过程又分为 Transact-SQL 存储过程和 CLR 存储过程两种。Transact-SQL 存储过程是指保存 Transact-SQL 语句的集合，可以接收和返回用户提供的参数。CLR 存储过程是针对微软公司的. NET Framework 公共语言运行时(CLR)方法的引用，可以接收和返回用户提供的参数。

(3) 扩展存储过程是指使用其他编程语言(例如 C 语言)创建自己的外部存储过程，是 SQL Server 数据库的实例可以动态加载和运行的动态链接库(DLL)，扩展了 SQL Server 2016 的性能，常以“xp_”为前缀名，其内容并不存储在 SQL Server 2016 中，而是以 DLL 的形式单独存在。常用的扩展存储过程如下：

- xp_cmdshell：用来运行平常在命令提示符下执行的程序，例如 DIR(显示目录)和 MD(更改目录)命令等。
- xp_sscanf：将数据从字符串读入每个格式参数所指定的参数位置。
- xp_sprintf：设置一系列字符和值的格式并将其存储到字符串输出参数中。每个格式参数都用相应的参数替换。

存储过程与视图之间的关系如表 5-2 所示。

表 5-2 存储过程与视图之间的关系

对比项目	视 图	存储过程
语句	只能是 SELECT 语句	可以包含控制流程、逻辑以及 SELECT 语句
输入、返回结果	不能接受参数，只能返回结果集	可以有输入、输出参数，也可以有返回值
典型应用	多个表的连接查询	完成某个特定的较复杂的任务

5.3.2 创建存储过程

如果要使用存储过程，首先要创建一个存储过程，可以使用 Transact-SQL 语句的 CREATE PROCEDURE 语句，也可以使用 SQL Server Management Studio 来完成。使用 SQL Server Management Studio 创建容易理解，较为简单；使用 Transact-SQL 语句创建较为快捷。在创建存储过程时，需要确定存储过程的 3 个组成部分：

- 所有的输入参数及执行后传给调用者的输出参数。
- 被执行的针对数据库的操作语句，包括调用其他存储过程的语句。
- 返回给调用者的状态值，以指明执行是否成功。

1. 使用 SQL Server Management Studio 创建存储过程

使用 SQL Server Management Studio 创建存储过程的操作步骤如下：

(1) 打开 SQL Server Management Studio 并连接到目标服务器，在“对象资源管理器”窗口中找到“数据库”结点，打开要创建存储过程的数据库(例如 StudentMIS)。

(2) 展开“可编程性”结点，然后右击“存储过程”项，在打开的快捷菜单中选择“新建存储过程”命令。此时，右侧窗口中显示了 CREATE PROCEDURE 语句的框架，可以修改要创建的存储过程的名称，然后加入存储过程所包含的 SQL 语句，如图 5-3 所示。

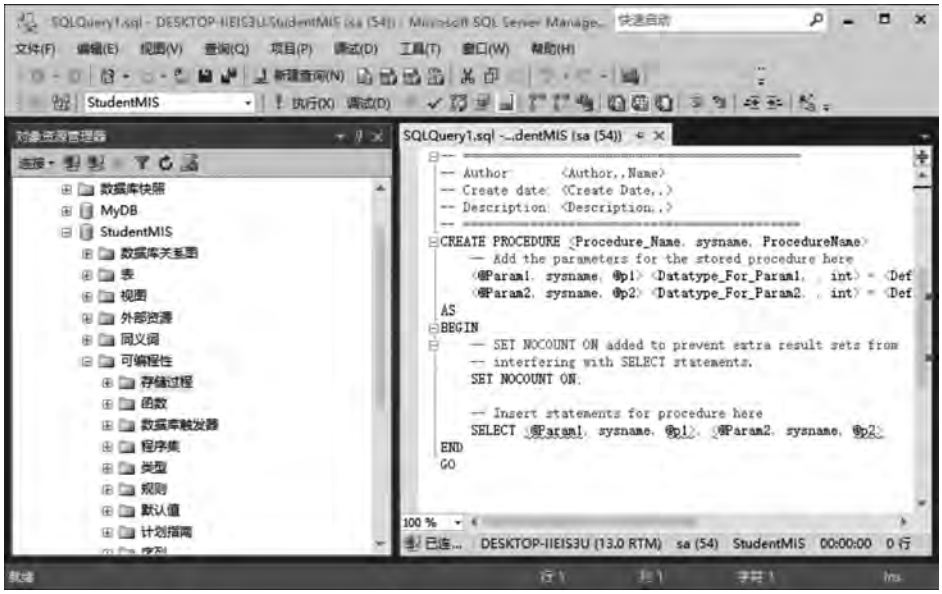

图 5-3 新建存储过程

(3) 在模板中输入完成后,单击工具栏上的“执行”按钮,可以立即执行 SQL 语句以创建存储过程。用户也可以单击“保存”按钮保存该存储过程的 SQL 语句。

2. 使用 Transact-SQL 语句创建存储过程

创建存储过程的 CREATE PROCEDURE 语句的语法为:

```
CREATE [ PROC | PROCEDURE] procedure_name [; number ]
[{ @parameter data_type }
[VARYING ] [ = default ] [ OUTPUT ] ] [ , …n ]
[WITH { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]
[FOR REPLICATION] AS sql_statement […n ]
```

其中,各参数的含义如下:

(1) procedure_name 是新建存储过程的名称,必须符合标识符命名规则。

(2) ;number 是可选的整数,用来对同名的存储过程分组,以便用一条 DROP PROCEDURE 语句即可将同组的存储过程一起删除。例如,名为 student 的应用程序使用的存储过程可以命名为“studentProc;1”“studentProc;2”等。DROP PROCEDURE studentProc 语句将删除整个组。

(3) @parameter 为存储过程的参数。在 CREATE PROCEDURE 语句中可以声明一个或多个参数。用户必须在执行过程时提供每个所声明参数的值(除非定义了该参数的默认值)。存储过程最多可以有 2100 个参数。

(4) data_type 为参数的数据类型。所有数据类型均可以用作存储过程的参数。不过,游标(CURSOR)数据类型只能用于 OUTPUT 参数。如果指定的数据类型为 CURSOR,必须同时指定 VARYING 和 OUTPUT 关键字。

(5) VARYING 指定作为输出参数支持的结果集(由存储过程动态构造,内容可以变化)。该参数仅适用于游标参数。

(6) default 为参数的默认值。如果定义了默认值,则不必指定该参数的值即可执行过程。默认值必须是常量或 NULL。

(7) OUTPUT 表明参数是返回参数。该选项的值可以返回给 EXEC[UTE]。使用 OUTPUT 参数可将信息返回给调用过程。

(8) 在{RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION}中,RECOMPILE 表明 SQL Server 不会缓存该过程的计划,该过程将在运行时重新编译;ENCRYPTION 表示 SQL Server 加密 syscomments 表中包含 CREATE PROCEDURE 语句文本的条目。

(9) FOR REPLICATION 指定不能在订阅服务器上执行为复制创建的存储过程。本选项不能和 WITH RECOMPILE 选项一起使用。

(10) sql_statement 为过程中要包含的任意数目和类型的 Transact-SQL 语句。

用户在创建存储过程时应该注意下面几点:

(1) 存储过程最大为 128MB。

(2) 用户自定义的存储过程只能在当前数据库中创建(临时存储过程除外,临时存储过程总是在 tempdb 中创建)。

(3) 在单个批处理中,CREATE PROCEDURE 语句不能与其他 Transact-SQL 语句组合使用。

(4) 存储过程可以嵌套使用,在一个存储过程中可以调用其他的存储过程。嵌套的最大深度不能超过 32 层。

(5) 存储过程如果创建了临时表,则该临时表只能用于该存储过程,而且当存储过程执行结束后,临时表自动被删除。

【例 5-9】 使用 CREATE PROCEDURE 语句创建一个存储过程 studentProc,查询某籍贯某学生的情况:

```
USE StudentMIS
GO
CREATE PROCEDURE studentProc
@name NVARCHAR(64), @address NVARCHAR(256)
AS SELECT * FROM Student WHERE StuName = @name AND Address = @address
GO
```

【例 5-10】 使用 CREATE PROCEDURE 语句创建计算两个整数和的存储过程 SumProc,并将结果返回给用户。

```
CREATE PROCEDURE SumProc
@i1 INT, @i2 INT, @result INT OUTPUT
AS
SET @result = @i1 + @i2
```

从存储过程中返回一个或多个值是通过在创建存储过程的语句中定义输出参数来实现的,通过关键字 OUTPUT 指明这是一个输出参数。值得注意的是,输出参数必须位于所有输入参数之后,如例 5.10 中输出参数@result 位于最后。

5.3.3 执行存储过程

存储过程与函数不同,存储过程不返回取代其名称的值,其参数不需要用括号括起,存储过程也不能直接在表达式中使用。执行存储过程使用 EXECUTE 语句,其完整语法格式如下:

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

其中,各参数的含义如下:

(1) @return_status 是一个可选的整型变量,用于保存存储过程的返回状态。这个

变量用在 EXECUTE 语句前，必须在批处理、存储过程或函数中声明过。

(2) procedure_name 为调用的存储过程名称。

(3) number 是可选的整数，用于将相同名称的过程进行组合，使得它们可以用 DROP PROCEDURE 语句删除。

(4) @procedure_name_var 是局部定义变量名，代表存储过程名称。

(5) @parameter 是存储过程参数，在 CREATE PROCEDURE 语句中定义，参数名称前必须加上符号@。

(6) value 是存储过程中参数的值。

(7) @variable 是用来保存参数或者返回参数的变量。

(8) OUTPUT 指定存储过程必须返回一个参数。该存储过程的匹配参数也必须由关键字 OUTPUT 创建。在使用游标变量做参数时使用该关键字。

(9) DEFAULT 为根据存储过程的定义提供参数的默认值。

(10) WITH RECOMPILE 为强制编译新的计划。

在调用存储过程时有两种传递参数的方法：第一种是在传递参数时，使传递的参数和定义时的参数顺序一致，对于使用默认值的参数可以用 DEFAULT 代替；第二种是采用参数名称引导的形式(例如@name='张颖')，此时各个参数的顺序可以任意排列。当输入参数较多时，建议使用参数名称引导的形式调用存储过程。

例如要执行例 5.9 的存储过程，使用如下语句：

```
EXECUTE studentProc@name = '张颖',@address = '北京'        --参数由参数名标识,顺序任意
或 EXECUTE studentProc '张颖', '北京'                    --参数以位置标识,必须按照定义参数的顺序
```

如果要执行例 5.10 的存储过程，使用如下语句：

```
DECLARE @answer INT
EXEC SumProc 10,20, @answer OUTPUT
SELECT @answer '两个整数相加的结果'
```

存储过程在执行后都会返回一个整型值。如果执行成功，返回 0；否则返回－1～－99 的数值。用户也可以使用 RETURN 语句来指定一个返回值。

5.3.4 查看存储过程

在存储过程被创建之后，存储过程的名称被存储在系统表 sysobjects 中，它的源代码被存储在系统表 syscomments 中。用户可以使用系统存储过程来查看用户自定义的存储过程。

(1) sp_help 用于显示存储过程的参数及其数据类型，其语法格式如下：

```
sp_help [[@objname = ]存储过程名]
```

（2）sp_helptext 可以查看未加密的存储过程的定义信息，其语法格式如下：

```
sp_helptext [[@objname = ]存储过程名]
```

【例 5-11】 查看 studentProc 存储过程的定义信息。

可以执行下面的 SQL 语句：

```
EXEC sp_helptext studentProc
```

执行的结果如图 5-4 所示。

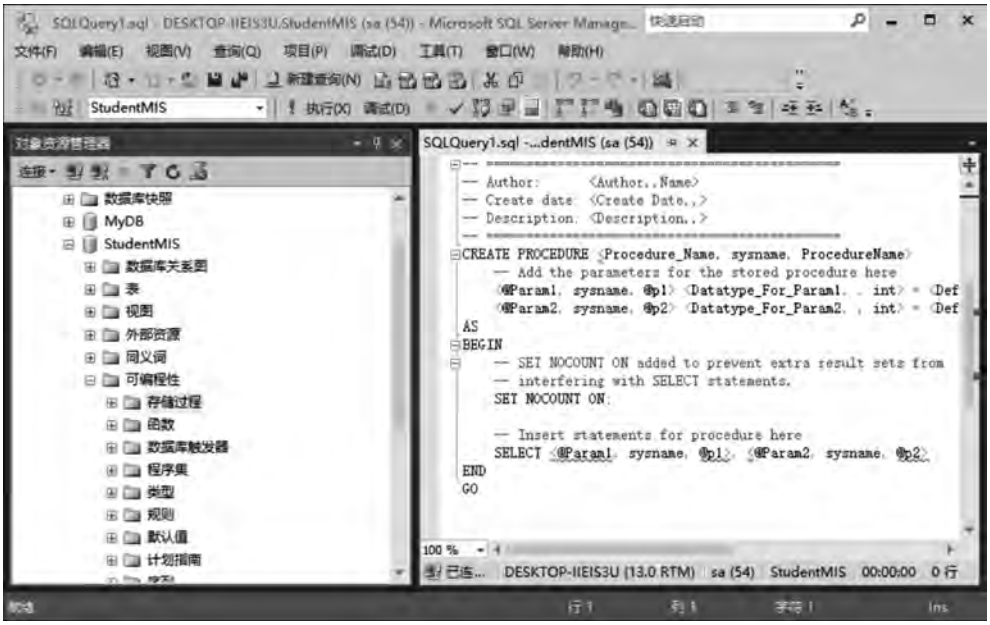

图 5-4 查看 studentProc 存储过程的定义信息

如果在创建存储过程中使用了 WITH ENCYPTION 选项，那么使用 sp_helptext 就无法看到存储过程的定义。

5.3.5 修改存储过程

1. 使用 SQL Server Management Studio 修改存储过程

使用 SQL Server Management Studio 修改存储过程较简单，步骤如下：

（1）打开 SQL Server Management Studio 并连接到目标服务器，在“对象资源管理器”窗口中找到“数据库”结点，然后选择存储过程所在的数据库（例如 StudentMIS）。

（2）依次展开“可编程性”结点和“存储过程”结点，然后右击要修改的存储过程名，例如 studentProc，在弹出的快捷菜单中选择“修改”命令，则会在右侧窗口中打开查询编辑器，显示该存储过程的源代码。

（3）修改代码后重新执行，保存即可。

2. 使用 Transact-SQL 语句修改存储过程

在 SQL Server 2016 中，可以使用 ALTER PROCEDURE 语句修改已经存在的存储过程，即直接将创建中的 CREATE 关键字替换为 ALTER。虽然删除并重新创建该存储过程也可以达到修改存储过程的目的，但是将丢失与该存储过程相关联的所有权限。修改存储过程的语法格式如下：

```
ALTER [ PROC | PROCEDURE] procedure_name [ ; number ]
[ { @parameter data_type }
[ VARYING ] [ = default ] [ OUTPUT ] ] [ , …n ]
[ WITH { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]
[ FOR REPLICATION ] AS sql_statement [ …n ]
```

通过对 ALTER PROCEDURE 语句语法的分析可以看出，其与 CREATE PROCEDURE 语句的语法构成完全一致，各参数的说明请参考 CREATE PROCEDURE 语句的语法说明。

【例 5-12】 出于对安全性的考虑，对例 5.9 中创建的存储过程进行加密处理。

```
USE StudentMIS
GO
ALTER PROCEDURE studentProc
@name NVARCHAR(64), @address NVARCHAR(256)
WITH ENCRYPTION      -- 增加了加密处理
AS SELECT * FROM Student WHERE StuName = @name AND Address = @address
GO
```

5.3.6 删除存储过程

1. 使用 SQL Server Management Studio 删除存储过程

使用 SQL Server Management Studio 删除存储过程的步骤如下：

(1) 打开 SQL Server Management Studio 并连接到目标服务器，在“对象资源管理器”窗口中找到“数据库”结点，然后选择存储过程所在的数据库(例如 StudentMIS)。

(2) 依次展开“可编程性”结点和“存储过程”结点，然后右击要删除的存储过程名，例如 studentProc，在弹出的快捷菜单中选择“删除”命令，则会弹出“删除对象”对话框，单击“确定”按钮，完成删除存储过程。

2. 使用 Transact-SQL 语句删除存储过程

使用 DROP PROCEDURE 语句可以在当前数据库中删除一个或多个存储过程，其语法格式如下：

```
DROP[ PROC | PROCEDURE] procedure_name [ , …n ]
```

【例 5-13】 删除例 5.9 创建的 studentProc 存储过程。

```
DROP PROC studentProc
```

5.3.7 重命名存储过程

使用系统存储过程 sp_rename 可以重命名存储过程。其语法格式如下：

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

其中，各参数的说明如下：

(1) [@objname=]'object_name'为存储过程的当前名称。

(2) [@newname=]'new_name'为要执行存储过程的新名称。

(3) [,[@objtype=]'object_type']为要重命名的对象的类型。当对象类型为存储过程或触发器时，其值为 OBJECT。

5.4 触发器

5.4.1 触发器概述

1. 触发器的概念

触发器(Trigger)是一种特殊类型的存储过程，是一个在修改指定表值的数据时执行的存储过程，它的执行不是由程序显式地调用或执行，也不是手工启动，而是通过事件触发被执行。例如，当对一个表进行操作(INSERT、UPDATE 或 DELETE 语句)时就会激活触发器的执行。但是触发器又与存储过程不同，存储过程可以由用户直接使用 EXEC 语句调用并执行，但是触发器不能被直接调用并执行，它只能自动执行。与存储过程相比，触发器通常可以完成一定的业务规则，用于 SQL Server 约束、默认值和规则的完整性检查，实施完整性和强制执行业务规则。

2. 触发器的特点

触发器作为一种非程序调用的存储过程，在应用过程中具有如下优点：

(1) 预编译、已优化、自动执行且效率高，避免了 SQL 语句在网络传输后再解释的低效率。

(2) 业务逻辑封装性好，数据库中很多问题都是可以在程序代码中去实现的，但是将其分离出来在数据库中处理，这样逻辑上更加清晰，对于后期维护和二次开发的作用比较明显。

(3) 触发器可通过数据库中的相关表实现级联更改。但是，通过级联引用完整性约束可以更有效地执行这些更改。

(4) 与 CHECK 约束定义相比，触发器可以强制定义更为复杂的约束。与 CHECK

约束不同，触发器可以引用其他表中的列。例如，触发器可以使用另一个表中的SELECT比较插入或修改的数据，以及执行其他操作，例如修改数据或显示用户定义错误信息。

(5) 触发器也可以评估数据更新前后的表状态，并根据其差异采取对策。

(6) 一个表中的多个同类触发器(INSERT、UPDATE或DELETE)允许采取多个不同的对策，以响应同一个更新语句。

(7) 确保数据规范化。使用触发器可以维护非正规化数据库环境中的记录级数据的完整性。

3. 触发器的分类

按照触发事件的不同，SQL Server 2016将系统提供的触发器分为3类，即DML触发器、DDL触发器和登录触发器。在SQL Server中，可以创建CLR(Common Language Runtime，公共语言运行库)触发器，它既可以是DML触发器，也可以是DDL触发器。CLR触发器将执行在托管代码(在.NET Framework中创建并在SQL Server中加载的程序集的成员)中编写的方法，而不需要执行Transact-SQL存储过程。

1) DML触发器

当数据库中发生数据操纵语言(DML)事件时将调用DML触发器。DML事件包括在指定表或视图中更新数据的INSERT语句、UPDATE语句或DELETE语句。按照DML触发器事件类型的不同，可以把SQL Server系统提供的DML触发器分成3种类型，即INSERT类型、UPDATE类型和DELETE类型。这也是DML触发器的基本类型。

DML触发器按照触发时机通常分为两类，即AFTER触发器和INSTEAD OF触发器。AFTER触发器在数据更新完成后被激活，执行顺序为：数据表约束检查→更新表中的数据→激活触发器。INSTEAD OF触发器会取代原来要进行的操作，在数据更改之前发生，数据如何更新完全取决于触发器的内容，执行顺序为：激活触发器→若触发器涉及数据更新，则检查表约束。

2) DDL触发器

在CREATE、ALTER、DROP和其他DDL语句上操作时发生的触发器称为DDL触发器。DDL触发器用于执行管理任务，并强制影响数据库的业务规则。它们通常在数据库或服务器中某一类型的所有命令执行时激活。

3) 登录触发器

登录触发器将为响应LOGON事件而激发触发器。与SQL Server实例建立用户会话时将触发该事件。登录触发器将在登录的身份验证阶段完成之后且用户会话实际建立之前触发。因此，来自触发器内部且通常将到达用户的所有消息传送到SQL Server错误日志。如果身份验证失败，将不触发登录触发器。

任何触发器都可以包含影响另外一个表的INSERT、UPDATE或DELETE语句。当允许触发器嵌套时，一个触发器可以修改触发第二个触发器的表，第二个触发器又可以触发第三个触发器。在默认情况下，系统允许触发器最多嵌套32层。

4. inserted 表和 deleted 表

在触发器执行的时候会产生两个临时表——inserted 表和 deleted 表。它们的结构和触发器所在的表的结构相同，SQL Server 2016 自动创建和管理这些表。用户可以使用这两个临时的驻留在内存中的表测试某些数据修改的效果及设置触发器操作的条件；然而，用户不能直接对这两个表中的数据进行修改，但可以读取。触发器执行完成后，与该触发器相关的这两个表也会被删除。

deleted 表用于存储 DELETE 和 UPDATE 语句所影响的行的副本。在执行 DELETE 或 UPDATE 语句时，行从触发器表中删除，并传输到 deleted 表中。deleted 表和触发触发器的表中不会有相同的行。

inserted 表用于存储 INSERT 和 UPDATE 语句所影响的行的副本。在一个插入或更新事务处理中，新建行被同时添加到 inserted 表和触发器表中。inserted 表中的行是触发器表中新行的副本。

在对具有触发器的表进行 INSERT、DELETE 和 UPDATE 操作时，其操作过程如下：

(1) 执行 INSERT 操作。插入触发器表中的新行被插入 inserted 表中。

(2) 执行 DELETE 操作。从触发器表中删除的行被插入 deleted 表中。

(3) 执行 UPDATE 操作。先从触发器表中删除旧行，然后再插入新行。其中被删除的旧行被插入 deleted 表中，插入的新行被插入 inserted 表中。

5.4.2 创建触发器

在 SQL Server 中，创建触发器可以使用 Transact-SQL 语句的 CREATE TRIGGER 语句，也可以使用 SQL Server Management Studio 来完成，本书只介绍使用 Transact-SQL 语句创建触发器的语法，使用 SQL Server Management Studio 创建触发器的方法请读者自学完成。

1. 创建 DML 触发器

在创建 DML 触发器前，用户应该考虑到下列问题：

(1) CREATE TRIGGER 语句必须是批处理中的第一个语句。将该批处理中随后的其他所有语句解释为 CREATE TRIGGER 语句定义的一部分。

(2) 创建触发器的权限默认分配给表的所有者，且不能将该权限转给其他用户。

(3) 触发器为数据库对象，其名称必须遵循标识符的命名规则。

(4) 虽然触发器可以引用当前数据库以外的对象，但只能在当前数据库中创建触发器。

(5) 虽然不能在临时表或系统表上创建触发器，但是触发器可以引用临时表。注意，不应引用系统表，而应使用信息架构视图。

(6) 在含有用 DELETE 或 UPDATE 操作定义的外键的表中，不能定义 INSTEAD OF 和 INSTEAD OF UPDATE 触发器。

(7) 虽然 TRUNCATE TABLE 语句(用于删除行)类似没有 WHERE 子句的 DELETE 语句,但它并不会激活 DELETE 触发器,因为 TRUNCATE TABLE 语句没有记录。

触发器可以由 CREATE TRIGGER 语句创建,其语法格式如下:

```
CREATE TRIGGER trigger_name ON { table | view }
[ WITH ENCRYPTION ]
{{ FOR | AFTER | INSTEAD OF } { [DELETE] [,] [INSERT] [,] [UPDATE] }
  [ WITH APPEND ] [ NOT FOR REPLICATION ] AS
  [ { IF UPDATE (column) [ { AND | OR } UPDATE (column) ] […n ]
  | IF ( COLUMNS_UPDATED () { bitwise_operator } updated_bitmask )
  { comparison_operator } column_bitmask […n ]
  } ]
  sql_statement […n ]
}
```

其中,各参数的含义如下:

(1) trigger_name 为触发器的名称。

(2) table | view 是在其上执行触发器的表或视图,有时称为触发器表或触发器视图。

(3) WITH ENCRYPTION 为加密 syscomments 表中包含 CREATE TRIGGER 语句的条目。

(4) AFTER 指定触发器只有在触发 SQL 语句中指定的所有操作都已成功执行后才触发。所有的引用级联操作和约束检查也必须成功完成后才能执行此触发器。如果仅指定 FOR 关键字,则 AFTER 是默认设置。注意,不能在视图上定义 AFTER 触发器。

(5) INSTEAD OF 指定执行触发器而不是执行触发 SQL 语句,从而替代触发语句的操作。INSTEAD OF 触发器不能在 WITH CHECK OPTION 的可更新视图上定义。

(6) {[DELETE] [,] [INSERT] [,] [UPDATE]}是指定在表或视图上执行哪些数据更新语句时将激活触发器的关键字,必须至少指定一个选项。当向表中插入或者修改记录时,INSERT 或者 UPDATE 触发器被执行。在一般情况下,这两种触发器常用来检查插入或者修改后的数据是否满足要求。DELETE 触发器通常用于两种情况:①防止那些确实要删除,但是可能会引起数据一致性问题的情况,一般是为那些用作其他表的外键记录。②级联删除操作。

(7) WITH APPEND 指定应该添加现有类型的其他触发器。注意,只有当兼容级别是 65 或更低时才需要使用该可选子句。

(8) NOT FOR REPLICATION 表示当复制进程更改触发器所涉及的表时不执行该触发器。

(9) AS 是触发器要执行的操作。

(10) sql_statement 是触发器的条件和操作。触发器条件指定其他准则,以确定 DELETE、INSERT 或 UPDATE 语句是否导致执行触发器操作。

IF 子句说明了触发器条件中的列值被修改时才触发触发器。判断列是否被修改有

以下两种办法。

(1) UPDATE (column): 参数为表或者视图中的列名称,说明这一列的数据是否被 INSERT 或者 UPDATE 操作更新过。如果更新过,返回 TRUE; 否则返回 FALSE。

(2) (COLUMNS_UPDATED(){bitwise_operator}updated_bitmask){comparison_operator}column_bitmask[…n]: COLUMNS_UPDATED()检测指定列是否被 INSERT 或者 UPDATE 操作更新过。它返回 varbinary 位模式,表示插入或修改了表中的哪些列。COLUMNS_UPDATED()函数以从左到右的顺序返回位,最左边的为最不重要的位。最左边的位表示表中的第一列; 向右的下一位表示第二列,以此类推。如果在表上创建的触发器包含 8 列以上,则 COLUMNS_UPDATED()返回多个字节,最左边的为最不重要的字节。在 INSERT 操作中 COLUMNS_UPDATED()将对所有列返回 TRUE 值,因为这些列插入了显式值或隐性(NULL)值。bitwise_operator 是用于比较运算的位运算符。updated_bitmask 是整型位掩码,表示实际修改或插入的列。例如,表 t1 包含列 C1、C2、C3、C4 和 C5。假定表 t1 上有 UPDATE 触发器,若要检查列 C2、C3 和 C4 是否都有更新,指定值 14(对应二进制数为 01110); 若要检查是否只有列 C2 有更新,指定值 2(对应二进制数为 00010)。comparison_operator 是比较运算符。使用等号(=)检查 updated_bitmask 中指定的所有列是否都实际进行了更新,使用大于号(>)检查 updated_bitmask 中指定的任一列或某些列是否已更新。column_bitmask 是要检查列的整型位掩码,用来检查是否已修改或插入了这些列。

在创建触发器时需要指定下面的选项: 触发器的名称,必须遵循标识符的命名规则; 在其上定义触发器的表; 触发器将何时激活; 激活触发器的数据修改语句,有效选项为 INSERT、UPDATE 或 DELETE; 多个数据修改语句可激活同一个触发器,例如触发器可由 INSERT 或 UPDATE 语句激活; 执行触发操作的编程语句。

1) INSERT 触发器

INSERT 触发器通常被用来验证被触发器监控的字段中的数据满足要求的标准,以确保数据完整性。

【例 5-14】 为 StudentMIS 数据库中的 Student 表创建一个名为 tr_student_ins 的 INSERT 触发器,在用户插入记录时,该触发器被触发,并自动显示表中的内容。

```
USE StudentMIS
GO
/* 如果触发器 tr_student_ins 存在,则删除 */
IF EXISTS (SELECT name FROM sysobjects WHERE name = 'tr_student_ins' AND type = 'TR')
DROP TRIGGER tr_student_ins
GO
/* 创建触发器 tr_student_ins */
CREATE TRIGGER tr_student_ins ON Student FOR INSERT AS
SELECT * FROM Student
GO
```

单击“执行”按钮,创建该触发器。后面例子中的触发器也需要输入代码后单击“执行”按钮创建,这里不再赘述。

说明：

(1) 当触发 INSERT 触发器时，新的数据记录就会被插入触发器所在的表和 inserted 表中。inserted 表包含了 INSERT 语句中已记录的插入动作。

(2) 触发器通过检查 inserted 表来确定是否执行触发器动作或如何执行它。

2) DELETE 触发器

当触发 DELETE 触发器后，SQL Server 2016 将被删除的记录转存到 deleted 表中，此时触发器所在的表中将不再存在该记录。也就是说，触发器所在的表和 deleted 表中不可能有相同的记录信息。临时表 deleted 存放在内存中，以提高系统性能。

【例 5-15】 为 StudentMIS 数据库中的 Student 表创建一个名为 tr_student_del 的 DELETE 触发器，因为 SC 表中包含学生的学号和成绩，如果还存在一个 Student 表，其中包含学生的学号和姓名，它们之间以学号相关联。如果要删除 Student 表中的一条记录，则与该记录的学号对应的学生成绩也应该删除。

```
USE StudentMIS
GO
IF EXISTS (SELECT name FROM sysobjects WHERE name = 'tr_student_del' AND type = 'TR')
DROP TRIGGER tr_student_del
GO
CREATE TRIGGER tr_student_del ON Student AFTER DELETE AS
DELETE FROM SC WHERE SC.StuNo = deleted.StuNo
GO
```

此时，要删除 Student 表中的记录，则 SC 表中对应的记录也被删除。如果使用 SELECT 语句来查询 SC 表，将看到对应的记录已经被删除，这样就保证了数据的参照完整性。

3) UPDATE 触发器

UPDATE 触发器的工作过程相当于删除一条旧的记录，插入一条新的记录。因此，可将 UPDATE 语句看成两步操作：

(1) 捕获更新前的记录的 DELETE 语句。

(2) 捕获更新后的记录的 INSERT 语句。

当在定义有触发器的表上执行 UPDATE 语句时，更新前的记录被存储到 deleted 表，更新后的记录被存储到 inserted 表。

【例 5-16】 为 StudentMIS 数据库中的 Student 表创建一个名为 tr_student_update 的 UPDATE 触发器，用户对 Student 表执行更新操作后触发，并返回更新的记录信息。

```
USE StudentMIS
GO
IF EXISTS (SELECT name FROM sysobjects WHERE name = 'tr_student_update' AND type = 'TR')
DROP TRIGGER tr_student_update
```

```
GO
CREATE TRIGGER tr_student_update ON Student AFTER UPDATE AS
BEGIN
    SELECT StuNo AS 更新前学号, StuName 更新前姓名 FROM deleted
    SELECT StuNo AS 更新后学号, StuName 更新后姓名 FROM inserted
END
GO
```

4) INSTEAD OF 触发器

与前面介绍的3种AFTER触发器不同,SQL Server服务器在执行触发AFTER触发器的SQL代码后,先建立临时的inserted和deleted表,然后执行SQL代码中对数据的操作,最后才激活触发器中的代码。而对于INSTEAD OF触发器,SQL Server服务器在执行触发INSTEAD OF触发器的代码时,先建立临时的inserted和deleted表,然后直接触发INSTEAD OF触发器,而拒绝执行用户输入的DML操纵语句。基于多个基本表的视图必须使用INSTEAD OF触发器来支持引用多个表中数据的插入、更新和删除操作。

【例5-17】 为StudentMIS数据库中的SC表创建一个名为tr_sc_insteadof_ins的INSERT触发器,当用户插入SC表中的成绩大于100分时,拒绝插入,并给出"插入成绩不能大于100分"的提示信息。

```
USE StudentMIS
GO
IF EXISTS (SELECT name FROM sysobjects WHERE name = 'tr_sc_insteadof_ins' AND type = 'TR')
DROP TRIGGER tr_sc_insteadof_ins
GO
CREATE TRIGGER tr_sc_insteadof_ins ON SC INSTEAD OF INSERT AS
BEGIN
    DECLARE @score decimal(18,2);
    SELECT @score = (SELECT score FROM inserted)
    IF @score > 100
        PRINT '插入成绩不能大于100分'
END
GO
```

2. 创建DDL触发器

在CREATE、ALTER、DROP和其他DDL语句上操作时发生的触发器称为DDL触发器。DDL触发器常用于以下情况:防止对数据库架构进行某些更改,以响应数据库架构中的更改;记录数据库架构中的更改或事件。

创建DDL触发器的语法格式如下:

```
CREATE TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[WITH ENCRYPTION ]
{ FOR | AFTER } { event_type} [ ,…n ]
AS
  sql_statement
```

其中,各参数的说明如下:

(1) ALL SERVER 表示将 DDL 触发器的作用域应用于当前服务器。如果指定了该参数,则当前服务器中的任何数据库都能触发该触发器。

(2) DATABASE 表示将 DDL 触发器的作用域应用于当前数据库。如果指定了该参数,则只有当前数据库能触发该触发器。

(3) AFTER 表示事后触发器,DDL 触发器没有 INSTEAD OF 触发器。

(4) event_type 指定触发 DDL 触发器的 Transact-SQL 语言事件的名称。每一个 DDL 事件都对应一个 Transact-SQL 语句,DDL 事件的名称是 DDL 语句的语法经过修改,在关键字之间包含了下画线(_)。例如删除表事件为 DROP_TABLE,修改表事件为 ALTER_TABLE,修改索引事件为 ALTER_INDEX 等。

【例 5-18】 创建一个触发器,用于防止用户删除和修改 StudentMIS 数据库中的任一数据表。

```
USE StudentMIS
GO
CREATE TRIGGER tr_deny_delete ON DATABASE          -- 指定作用域
FOR DROP_TABLE, ALTER_TABLE AS                     -- 指定触发事件
BEGIN
    PRINT '禁止删除或修改该数据表!'
    ROLLBACK TRANSACTION
END
GO
```

5.4.3 查看触发器

因为触发器是一种特殊的存储过程,所以可以使用查看存储过程的方法来查看触发器的内容。因此,用户可以使用系统存储过程 sp_help、sp_helptext 和 sp_depents 分别查看触发器的不同信息。

- sp_ help:显示触发器的所有者和创建时间。
- sp_ helptext:显示触发器的源代码。
- sp_depends:显示该触发器参考的对象清单。

【例 5-19】 查看 tr_student_ins 触发器的源代码。

```
USE StudentMIS
GO
sp_helptext tr_student_ins
```

5.4.4 修改触发器

当触发器不满足需求时,可以修改触发器的定义和属性。在 SQL Server 中可以通过两种方式进行修改:先删除原来的触发器,再重新创建与之同名的触发器;直接修改现有触发器的定义。修改触发器的定义可以使用 ALTER TRIGGER 语句。

1. 修改 DML 触发器

修改 DML 触发器可以使用 ALTER TRIGGER 语句,其语法格式如下:

```
ALTER TRIGGER trigger_name ON (table | view)
[ WITH ENCRYPTION ]
{ ( FOR | AFTER | INSTEAD OF ) { [ DELETE ] [ , ] [ INSERT ] [ , ] [ UPDATE ] }
 [ NOT FOR REPLICATION ] AS sql_statement […n ] } |
{ ( FOR | AFTER | INSTEAD OF ) { [ INSERT ] [ , ] [ UPDATE ] }
[ NOT FOR REPLICATION ]
AS { IF UPDATE (column) [ { AND | OR } UPDATE (column) ] […n ]
| IF ( COLUMNS_UPDATED () { bitwise_operator } updated_bitmask ) { comparison_operator }
 column_bitmask […n ] } sql_statement […n ] }
```

各参数的含义和 CREATE TRIGGER 语句相同,这里不再介绍。

2. 修改 DDL 触发器

修改 DDL 触发器的语法格式如下:

```
ALTER TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[WITH ENCRYPTION ]
{ FOR | AFTER } { event_type} [ ,…n ]
AS
 sql_statement
```

5.4.5 删除触发器

当触发器不再使用时,可以将其删除。删除触发器不会影响其操作的数据表,而当某个表被删除时,该表上的触发器也同时被删除。用户可以使用 DROP TRIGGER 语句来删除触发器,其语法格式如下:

```
DROP TRIGGER { trigger_name } [ ,…n ]                                    -- 删除 DML 触发器
DROP TRIGGER { trigger_name } [ ,…n ] ON { ALL SERVER | DATABASE }   -- 删除 DDL 触发器
```

其中,trigger_name 是要删除的触发器名称,而 n 是表示可以指定多个触发器的占位符。

【例 6-20】 删除 DML 触发器 tr_student_ins。

在查询编辑器中执行下面的 Transact-SQL 语句：

```
DROP TRIGGER tr_student_ins
```

5.4.6 重命名触发器

重命名触发器使用 sp_rename 命令，其语法格式为：

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

其参数说明和用例请参考重命名存储过程的说明，在此不再赘述。

5.4.7 启用和禁用触发器

在默认情况下，触发器创建之后便启用了，如果暂时不需要使用某个触发器，可以将其禁用。触发器被禁用后并没有删除，它仍然作为对象存储在当前数据库中。

1. 禁用触发器

当不再需要某个触发器时可将其禁用。禁用触发器可以使用 ALTER TABLE 语句或者 DISABLE TRIGGER 语句。使用 DISABLE TRIGGER 语句的语法格式如下：

```
DISABLE TRIGGER {ALL|trigger_name [,…n]}
ON {table | view | DATABASE | ALL SERVER}
```

2. 启用触发器

已禁用的触发器可以被重新启用。启用触发器可以使用 ALTER TABLE 语句或者 ENABLE TRIGGER 语句。使用 ENABLE TRIGGER 语句的语法格式如下：

```
ENABLE TRIGGER {ALL| trigger_name[,…n]}
ON {table | view | DATABASE | ALL SERVER}
```

5.5 本章知识点小结

批处理是一次性将多个 Transact-SQL 语句发送给服务器以完成执行的工作方式，这有助于节省语句的执行时间。脚本是指存储在文件中的一系列 SQL 语句，将常用的 Transact-SQL 语句保存为脚本文件，可方便以后重复使用或复制到其他计算机上执行。

Transact-SQL 中提供了一些常用的流程控制语句，通过这些语句使得 Transact-SQL 除了具备标准 SQL 的优点之外，还实现了顺序、选择、循环等程序结构的流程控制。

本章介绍了存储过程与触发器的概念、特点和作用，介绍了创建和管理存储过程与触

发器的方法与技巧。存储过程是 SQL 语句和可选控制流语句的预编译集合，以一个名称存储并作为一个单元处理。触发器则是一种特殊的存储过程，在对表执行 INSERT、DELETE 和 UPDATE 操作时自动执行。存储过程和触发器在数据库开发过程以及数据库维护和管理等任务中有非常重要的作用。使用存储过程和触发器可以有效地检查数据的有效性和数据的完整性、一致性。

5.6 习题

1. 什么是批处理？使用批处理有何限制？
2. 什么是存储过程？存储过程分为哪几类？使用存储过程有什么好处？
3. 什么是触发器？其主要功能是什么？
4. 触发器分为哪几种？
5. INSERT 触发器、UPDATE 触发器和 DELETE 触发器有什么不同？
6. AFTER 触发器和 INSTEAD OF 触发器有什么不同？
7. 创建一个存储过程，用于查询订单信息，包括订单日期、客户名称、定购的书籍名称、单价、数量和总价。
8. 创建一个触发器，用于在向 author 表或者修改插入数据时检查 telephone 字段的长度不大于 13 位(必须为区号＋电话号码的格式，例如 0471-11111111)。