实验 7 数据库编程初步

7.1 实验目的

1．掌握常用的流程控制语句；

2．掌握存储过程、触发器编写方法。

7.2 内容提要

7.2.1 流程控制语句

T-SQL 中用来编写流程控制模块的语句有：BEGIN...AND 语句、IF...ELSE 语句、CASE 语句、WHILE、RETURN 等语句。

1．BEGIN...AND 语句

该语句块是多条 Transact-SQL 语句组成的代码段，从而可以执行一组 Transact-SQL 语句。**BEGIN** 和 **END** 是控制流语言的关键字。

BEGIN...END 语句块通常包含在其他控制流程中，用来完成不同流程中有差异的代码功能。

示例 1：以下代码定义整形变量 count，循环输出 count 值到 count<9 为止。

```
DECLARE @count INT       --定义整形变量 count
SELECT @count = 0        --赋值，也可采用 SET @count = 0 语句
WHILE @count < 10        --while 语句，以下 BEGIN …END 语句块包含代码的为循环体
BEGIN
    PRINT 'count = ' + CONVERT(VARCHAR(10), @count)
    SELECT @count = @count + 1
END
PRINT 'loop finished, count = ' + CONVERT(VARCHAR(10), @count)
```

2．IF...ELSE 语句

IF...ELSE 语句用于在执行一组代码之前进行条件判断，根据判断的结果执行不同的代码。IF...ELSE 语句对布尔表达式进行判断，如果布尔表达式返

回为 TRUE，则执行 IF 关键字后面的语句块；如果布尔表达式返回 FALSE，则执行 ELSE 关键字后面的语句块。

语法规则如下：

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

示例 2：

```
DECLARE @score INT
SET @score = 100
IF @score >= 60
    PRINT '及格'
ELSE
    PRINT '不及格'
```

3．CASE 语句

CASE 语句是多条件分支语句，相比 IF...ELSE 语句，CASE 语句进行分支流程控制可以使代码更加清晰，易于理解。CASE 语句根据表达式逻辑值的真假来决定执行的代码流程。

语法规则如下 2 种形式：

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


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

示例 3：

```
DECLARE @score INT
SET @score = 100
SELECT
```

```
    CASE @score
        WHEN 100 THEN '满分'
        WHEN 60 THEN '及格'
      END
  AS '成绩'
```

示例 4：

```
DECLARE @score INT
SET @score = 100
SELECT
    CASE
        WHEN @score >= 90 THEN '优秀'
        WHEN @score >= 80 THEN '良好'
        WHEN @score >= 70 THEN '中等'
        WHEN @score >= 60 THEN '及格'
        ELSE '不及格'
     END
 AS '成绩'
```

4. WHILE 语句

WHILE 语句根据条件重复执行一条或多条 T-SQL 代码，只要条件表达式为真，就循环执行语句。

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。

语法：

```
WHILE Boolean_expression
        { sql_statement | statement_block | BREAK | CONTINUE }
```

参数：

Boolean_expression：返回 TRUE 或 FALSE 的表达式。 如果布尔表达式中含有 SELECT 语句，则必须用括号将 SELECT 语句括起来。

{sql_statement | statement_block}：Transact-SQL 语句或用语句块定义的语句分组。 若要定义语句块，请使用控制流关键字 BEGIN 和 END。

BREAK：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。

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

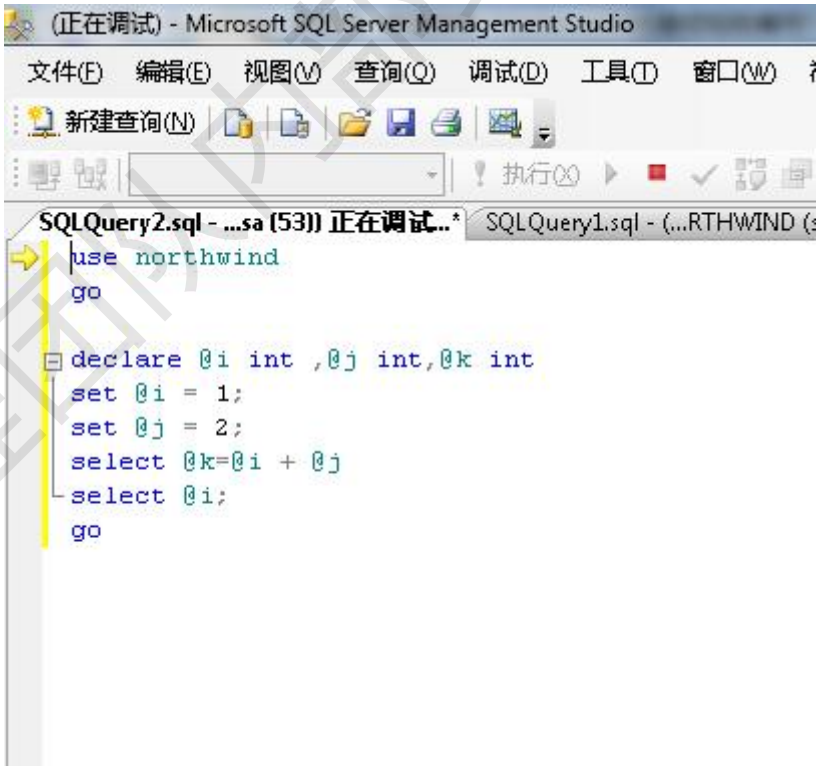
5.调试 T-SQL 语句

Debug T-SQL 语句，SQL 代码如下：

```
declare @i int, @j int, @k int
 set @i = 1;
 set @j = 2;
 set @k=@i + @j
 select @i;
```

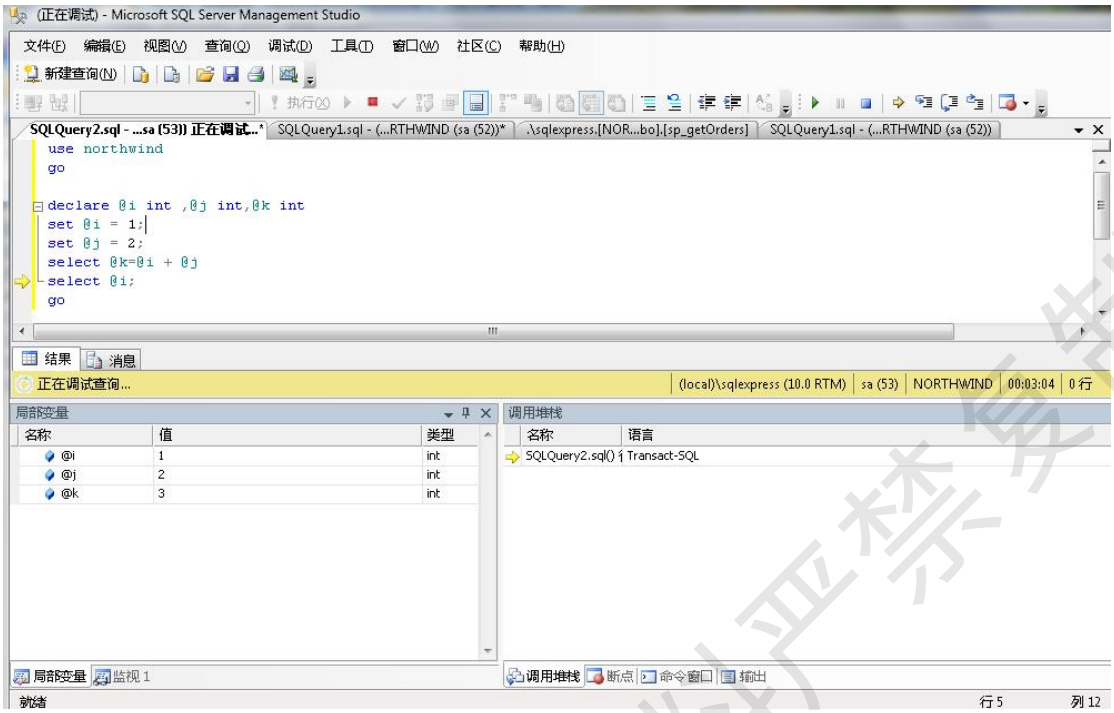
非常简单的定义了 三个 INT 型变量:i、j、k 并且对这些变量进行简单的逻辑运算，在 Management Studio 中只要轻松的按 **F11 键**，即可调试以上代码块，该功能需要防火墙打开 135 端口，或调试时关闭防火墙。

截图如下：



接着点击 F11 **逐语句调试** 或者 F10 **逐过程调试**代码。

截图如下：



这个 Debug 的场面您是否觉得已经和 VS 相差无几了呢？

7.2.2 存储过程

存储过程是一组为了完成特定功能的 SQL 语句集合，它经编译后存储在数据库中，用户通过指定的存储过程名称并给出相应的参数就可以对其进行执行。

SQLSERVER 2008 主要包括用户自定义存储过程，扩展存储过程和系统存储过程，本实验主要介绍用户自定义存储过程创建与执行方法。

1. 创建存储过程

使用create PROCEDURE或create proc创建存储过程，语法规则：

```
-- ============================================
-- Author:     <Author,,Name>
-- Create date: <Create Date,,>
-- Description:   <Description,,>
-- ============================================
CREATE PROCEDURE <Procedure_Name, sysname, ProcedureName>
    -- Add the parameters for the stored procedure here
    <@Param1, sysname, @p1> <Datatype_For_Param1, , int> =
<Default_Value_For_Param1, , 0>,
    <@Param2, sysname, @p2> <Datatype_For_Param2, , int> =
<Default_Value_For_Param2, , 0>
AS
```

```
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT <@Param1, sysname, @p1>, <@Param2, sysname, @p2>
END
```

示例 1：创建无参数存储过程，输出仓库表所有记录

```
-- ================================================
-- Author:     <Author,,Name>
-- Create date: <Create Date,,>
-- Description:   创建名为MyPRO的无参数存储过程
-- 输出仓库表所有记录
-- ================================================
create procedure MyPRO
as
begin
    select * from 仓库
end
```

执行过程：

```
exec mypro  --exec 表示执行存储过程
```

示例 2：创建无参数存储过程，输出仓库表所有记录

```
-- ================================================
-- Author:     <Author,,Name>
-- Create date: <Create Date,,>
-- Description:   创建有个参数，名为Pro_Order的存储过程，
-- 输出满足条件的订单信息
-- ================================================
CREATE PROCEDURE Pro_Order
    @EmployeeID VARCHAR(50)
    @OrderDate  DATETIME
AS
begin
    SELECT 职工号,订购单号,订购日期
    FROM  订购单
    WHERE 职工号=@EmployeeID AND 订购日期=@OrderDate
end
```

执行过程：

```
EXEC Pro_Order @EmployeeID = 'E3',@OrderDate = '2002-06-23'
```

示例 3：将示例 2 修改为默认参数过程，当不输入参数值时，返回全部记录

```
-- =============================================
-- Author:     <Author,,Name>
-- Create date: <Create Date,,>
-- Description:   创建有个参数，名为Pro_Order的存储过程,
-- 输出满足条件的订单信息
-- =============================================
    ALTER PROCEDURE [dbo].[Pro_Order]
        @EmployeeID VARCHAR(50)=null, --默认参数
        @OrderDate  DATETIME=null     --默认参数
    AS
    begin
        SELECT 职工号,订购单号,订购日期
        FROM  订购单
        WHERE
            (case when @EmployeeID is null or 职工号=@EmployeeID  then 1
            else 0 end = 1)
            AND
            (case when @OrderDate is null or 订购日期=@OrderDate  then 1
            else 0 end = 1)
    End
```

不输入参数，执行上述过程：

```
EXEC   Pro_Order
```

7.2.3 触发器

1.触发器简介

触发器是 SQL 提供的一种维护数据库完整性的工具。

一个触发器只适用于一个表，每个表最多能有 INSERT、UPDATE、DELETE 触发器。

每个触发器有两个特殊的表，即 inserted、deleted 表，这两个表由系统管理，存储在内存中，不允许用户直接对其编辑。

2.触发器工作原理

1)INSERT 触发器:先向 inserted 表中插入一行副本，并检查 inserted 表中新行的有效性，确认是否阻止该插入操作，如不干预，插入数据行道表中。

2)UPDATE 触发器：先将原数据行移动到 deleted 表中，然后将一个新行插入 inserted 表中，最后计算 deleted 表和 inserted 表中的值以确定是否要阻止。

3)DELETE 触发器：先将原数据行移动到 deleted 表中，计算 deleted 表中的值决定是否干预，如不干预，数据行删除。

3.创建触发器

使用 CREATE TRIGGER 创建触发器，语法如下：

```
CREATE TRIGGER <Schema_Name, sysname, Schema_Name>.<Trigger_Name, sysname, Trigger_Name>
   ON  <Schema_Name, sysname, Schema_Name>.<Table_Name, sysname, Table_Name>
   AFTER <Data_Modification_Statements, , INSERT,DELETE,UPDATE>
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
    -- Insert statements for trigger here

END
```

示例：为员工表创建触发器，当插入或更新数据时，保证员工工资值大于 0

```
-- ============================================
-- Author:     <Author,,Name>
-- Create date: <Create Date,,>
-- Description:   当插入或更新数据时，保证员工工资值大于
-- ============================================
CREATE TRIGGER TRI_Employee
   ON   职工
   AFTER  INSERT, UPDATE
AS
    IF(SELECT 工资 FROM inserted) <0
BEGIN
    -- Insert statements for trigger here
    PRINT ''
    ROLLBACK TRANSACTION
END
```

读者可通过 INSERT、UPDATE 语句验证其有效性。

7.3 实验任务及步骤

7.3.1 创建存储过程

1)创建存储过程，通过员工号查询员工的姓名，订单号，供应商号，默认职工号为 E1。

2)执行该存储过程，输入员工号 E3，显示该员工订单信息。

3)另外创建存储过程，输入员工号，订购日期，统计该员工订单数，如果某月订单大于 1 单，则该员工该月为“优秀”，如果没有订单，则显示“加油”。

7.3.2 创建触发器

1)为供应商表建立触发器，禁止删除目前有订购单的供应商。

7.3.3 实验总结

1.记录实验全过程，并写出实验报告。

2.详细记录实验过程中遇到的问题，以及问题的解决方法。