第10章　存 储 过 程

10.1　存储过程的定义

存储过程是存储在服务器上的预编译好的 SQL语句集。你可以将存储过程类比为 SQL Server提供的用户自定义函数，你可以在后台或前台调用它们。

实际上，存储过程是Transact-SQL对ANSI-92 SQL标准的扩充。它允许多个用户访问相同的代码。它提供了一种集中且一致的实现数据完整性逻辑的方法。存储过程用于实现频繁使用的查询、业务规则、被其他过程使用的公共例行程序。

存储过程分为三类：系统提供的存储过程、用户定义的存储过程和扩展存储过程。

- 系统提供的存储过程 —— 在安装SQL Server时，系统创建了很多系统存储过程。系统存储过程主要用于从系统表中获取信息，也为系统管理员和合适用户（即有权限用户）提供更新系统表的途径。他它们中的大部分可以在用户数据库中使用。系统存储过程的名字都以“ sp_ ”为前缀。关于系统存储过程，我们会在相关的每一章内进行讲解。
- 用户定义的存储过程 —— 用户定义的存储过程是由用户为完成某一特定功能而编写的存储过程。我们在本章详细介绍用户定义的存储过程。
- 扩展存储过程 —— 扩展存储过程是对动态链接库（ DLL）函数的调用。我们也在本章最后讲解。

10.2　存储过程的优点

存储过程具有如下的优点：

- 使用存储过程可以减少网络流量。这是因为存储过程存储在服务器上，并在服务器上运行。只有触发执行存储过程的命令和返回的结果才在网络上传输。所以，可以减少网络流量。客户端无需将数据库中的数据通过网络传输到本地进行计算，再将结果数据通过网络送到服务器。从而减少了网络流量。
- 增强代码的重用性和共享性。一个存储过程是为了完成某一个特定功能而编写的一个模块，该模块可以被很多用户重用，也可以被很多用户共享。所以，存储过程可以增强代码的重用性和共享性，加快应用的开发速度，提供开发的质量和效率。
- 使用存储过程可以加快系统运行速度。第一次执行后的存储过程会在缓冲区中创建查询树，使得第二次执行时不用进行预编译，从而加快速度。
- 使用存储过程保证安全性。因为可以不授予用户访问存储过程中涉及的表的权限，而只授予访问存储过程的权限。这样，既可以保证用户通过存储过程操纵数据库中的数据，又可以保证用户不能直接访问与存储过程相关的表，从而保证表中数据的安全性。

10.3　存储过程的创建

创建存储过程的语法为：

```
CREATE PROCEDURE [OWNER.]procedure_name[;number]
 [({[@]parameter data_type}[VARYING][=default][OUTPUT])]
 [,...n]
[WITH {RECOMPILE|ENCRYPTION|RECOMPILE,ENCRYPTION}]
AS
  sql_statements
```

上一个语法中包含以下符号：

- procedure_name：为新创建的存储过程指定的名字。
- parameter：在存储过程中包含的输入和输出参数。
- WITH RECOMPILE：重编译选项。具体含义见第4节。
- Default：为这个参数指定的缺省值。它必须是一个常量。如果该参数使用在 LIKE子句中，则该缺省值可以含有通配符。如果给出了缺省值，那么在执行该存储过程时，如果没有向具有缺省值的参数传递参数值时，则具有缺省值的参数就可使用它们的缺省值。
- WITH ENCRYPTION：对包含CREATE PROCEDURE文本的SYSCOMMENTS表中的项加密。
- number:用于将名字相同的过程编组。
- sql_statements：在存储过程中要执行的动作。

具体创建的例子，我们将在下节中仔细讲解。

10.3.1 创建步骤

以下是创建存储过程所推荐的4个步骤：

1) 写SQL语句

如：查看书的个数。 SELECT COUNT(*)FROM titles

2) 测试SQL语句

执行这些SQL语句。确认符合要求。

3) 若得到所需结果,则创建过程

如果发现符合要求，则按照存储过程的语法，定义该存储过程。

```
CREATE PROC count_title
AS SELECT COUNT(*)FROM titles
```

4) 执行过程

执行存储过程，验证正确性。

```
EXEC count_title
```

例 下面我们为library数据库创建一个存储过程 totalbooksloaned，显示借出书的详细信息。

```
USE library
GO

CREATE PROC dbo.TotalBooksLoaned
AS
SELECT l.member_no, m.lastname, m.firstname, m.middleinitial,
  COUNT(l.member_no) TotalBooksLoaned
FROM member m
JOIN loanhist l
```

```
 ON l.member_no = m.member_no
GROUP BY l.member_no, lastname, firstname, middleinitial
ORDER BY totalbooksloaned DESC
```

结果显示如下，表示创建成功。

```
The command(s) completed successfully.
```

下面，我们讲解如何执行存储过程。

10.3.2 执行存储过程

执行存储过程的完整语法如下：

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

我们会在每节详细讲解每个参数的使用。现在，你只需要知道： EXEC后面带上存储过程的名称就可以执行这个存储过程。

如：EXEC totalbooksloaned，它将显示借出书的详细信息。以下是部分执行结果。

member_no	lastname	firstname	middleinitial	TotalBooksLoaned
130	Durkin	William	A	104
344	Martin	Darlene	A	104
1206	Sherman	Jose	B	104
1263	Thomas	Mary Anne	B	104
1494	Erickson	Karl	B	104

10.3.3 创建带输入参数的存储过程

输入参数是指由调用程序向存储过程传递的参数。它们在创建存储过程语句中被定义，而在执行该存储过程中给出相应的变量值。

具体语法如下:

```
@parameter_name datatype[=default]
```

其中,

- @parameter_name是存储过程的输入参数名，必须以 @符号为前缀。当执行该存储过程时，应该向输入参数提供相应的值。
- Datatype是该参数的数据类型说明，它可以是系统提供的数据类型，也可以是用户定义的数据类型。
- DEFAULT 如果执行存储过程时未提供该参数值，则使用 DEFAULT值。

例 为library数据库创建一个存储过程，添加一个成人信息到 Member和Adult表。

```
USE library
GO
/* 如果有重名的存储过程，则删除之 */
IF OBJECT_ID('dbo.addadult') IS NOT NULL
DROP PROCEDURE dbo.addadult
```

```
GO

CREATE PROCEDURE dbo.addadult
   @lastname          shortstring          = NULL,
   @firstname         shortstring          = NULL,
   @middleinitial     letter      = NULL,
   @street    shortstring          = NULL,
   @city      shortstring          = NULL,
   @state     statecode            = NULL,
   @zip       zipcode   = NULL,
   @phone_no          phonenumber       = NULL
AS
IF @lastname              IS NULL OR
@firstname         IS NULL OR
@street             IS NULL OR
      @city        IS NULL OR
      @state       IS NULL OR
      @zip          IS NULL

  BEGIN
    PRINT 'Please reenter this Adult Member.'
    PRINT 'You must provide Lastname, Firstname, Middle Initial, '
    PRINT 'Street, City, State, Zip, and Phone number.'
    PRINT '(Middle initial and Phone number can be null.)'
    RETURN
  END

  BEGIN TRANSACTION
    INSERT member
       (lastname,  firstname,  middleinitial)
    VALUES
       (@lastname, @firstname, @middleinitial)
    IF @@error <> 0
     BEGIN
       ROLLBACK TRAN
       RETURN
     END

    DECLARE @insertmem_no member_no
    SELECT @insertmem_no=@@identity

     INSERT adult
       (member_no,  street,  city,  state,  zip,  phone_no, expr_date)
     VALUES
       (@insertmem_no, @street, @city, @state, @zip, @phone_no, DATEADD(year,1,GETDATE()) )
     IF @@error <> 0
      BEGIN
       ROLLBACK TRAN
       RETURN
      END
```

```
        PRINT '*** Adult Member added ***   '

    COMMIT TRANSACTION
GO
```

结果显示如下，表示创建成功。

```
The command(s) completed successfully.
```

下面，我们来执行这个存储过程。SQL Server提供了以下两种方法传递参数。

1. 按位置传送

这种方法是在执行存储过程语句中，直接给出参数的传递值。当有多个参数时，值的顺序与创建存储过程语句中定义参数的顺序相一致。也就是说，参数传递的顺序就是参数定义的顺序。其格式是：

```
[EXECUTE] proc_name [value...]
```

其中，PROC_NAME是存储过程名字，value是传递给该输入参数的值。如：EXEC addadult'YourLastName'，'YourFirstName'，NULL，'YourStreetAddress', 'YourCity','YourState', 'YourPostalCode',NULL

2. 通过参数名传送

这种方法是在执行存储过程中，指出创建该存储过程语句中的参数名字和传递给它的值。其格式如下：

```
 [EXECUTE]proc_name  [@parameter=value]
```

其中，proc_name是存储过程名字，parameter是输入参数的名字，value是传递给该输入参数的值。

如：执行addadult存储过程添加一个新的成人信息。

```
USE library
GO

EXEC addadult
    @lastname = 'YourLastName',
    @firstname = 'YourFirstName',
    @middleinitial = NULL,
    @street = 'YourStreetAddress',
    @city = 'YourCity',
    @state = 'YourState',--2 character abbreviation
    @zip ='YourPostalCode',
    @phone_no = NULL
```

结果显示如下，表示执行成功。

```
(1 row(s) affected)
(1 row(s) affected)
*** Adult Member added ***
```

使用按参数名传递参数的好处是，参数可以以任意顺序给出。

10.3.4 创建带输出参数的存储过程

我们可以从存储过程中返回一个或多个值。这是通过在创建存储过程的语句中定义输出参数来实现的。具体语法如下：

```
@parameter name datatype[=default]  OUTPUT
```

其中，

- @parameter_name是存储过程的输出参数名，必须以@符号为前缀。
- Datatype是该参数的数据类型说明，它可以是系统提供的数据类型，也可以是用户定义的数据类型。
- 保留字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说明之后。

例如：创建一个完成乘法的存储过程。

```
CREATE PROCEDURE mathtutor
@m1 smallint,
@m2 smallint,
@result smallint OUTPUT
AS
SELECT @result=@m1*@m2
GO
```

结果显示：The command(s) completed successfully.。

为了接收某一存储过程的返回值，在调用该存储过程的程序中，也必须声明作为输出的传递参数。这个输出传递参数声明为局部变量，用来存放返回参数的值。

具体语法如下：

```
[[EXECUTE]
 {
   [@return_status=]
    {procedure_name[;number]|@procedure_name_var}
 }
[[@parameter_name=]{value|@variable [OUTPUT]|[default]}][,...n]
[WITH RECOMPILE]
```

其中，

- proc_name是存储过程名字
- [@parameter_name=]{value|@variable}是输入参数传递值
- [@parameter_name=] @variable OUTPUT是传递给该输出参数的变量。其中， @variable是用来存放返回参数的值。 OUTPUT指明这是一个输出传递参数，与相应的存储过程中的输出参数相匹配。

下面我们来执行刚刚创建的存储过程：

```
DECLARE @guess smallint
SET @guess=50
EXECUTE mathtutor 5,6,@guess OUTPUT
SELECT 'the result IS:',@guess
GO
```

结果显示：

```
-------------- ------
the result IS: 30

(1 row(s) affected)
```

在上述例子中，若调用中省略 OUTPUT,调用仍执行,但@guess值不改变（50）。若在存储过程的定义中省略OUTPUT,调用时会出错。

实际上，每个存储过程的执行，都将自动返回一个返回状态（可以通过 @return_status获得，见上面语法），用于告诉调用程序“ 执行该存储过程的状况 ”。调用程序可根据返回状态作相应的处理。一般而言，系统使用 0表示该存储过程执行成功。其他是系统的一些保留值。用户也可以在存储过程中使用 RETURN来返回指定的值。

例如：下面我们为 library数据库创建一个名叫 loancount的存储过程，如果该成员还借着书就返回1，否则返回0。

```
USE library
GO

CREATE PROC dbo.loancount
   @member_no member_no,
   @loancount int OUTPUT
AS
IF EXISTS
 (SELECT * FROM loan WHERE member_no = @member_no)
 BEGIN
    SELECT @loancount=COUNT(*)
    FROM loan
    WHERE member_no = @member_no
    RETURN (1)
 END
ELSE RETURN (0)
GO
```

结果显示如下，表示创建成功。

```
The command(s) completed successfully.
```

下面我们来执行这个存储过程。

```
USE library
GO

DECLARE
 @member_no  int,
 @msg        varchar(80),
 @retcode    int,
 @numtitles  int

SET @member_no = 123   -- change this number as desired

EXEC @retcode = loancount @member_no, @numtitles OUTPUT

IF @retcode = 1
BEGIN
  SELECT @msg =
    'Member_no ' +
    RTRIM(CONVERT(char(8),@member_no)) +
    ' has ' +
    RTRIM(CONVERT(char(8),@numtitles)) +
    ' title(s) on loan.'
  RAISERROR (@msg, 10 ,1)
      END
```

```
ELSE
BEGIN
  SELECT @msg =
    'Member_no ' +
    RTRIM(convert(char(8),@member_no)) +
    ' has NO titles on loan.'
  RAISERROR (@msg, 10 ,1)
END
GO
```

结果显示如下：

```
Member_no 123 has NO titles on loan.
```

10.3.5 library数据库中的一些存储过程

下面是在library数据库中所需要的一些存储过程，每个存储过程的作用请见存储过程前面的注释。

```
/*以下脚本为library数据库创建所有的存储过程。还创建了 4个自定义错误信息*/
USE library
GO

/* 如果存在同名的存储过程，则删除之 */

IF OBJECT_ID('dbo.addjuvenile') IS NOT NULL
   DROP PROCEDURE dbo.addjuvenile
IF OBJECT_ID('dbo.removemember') IS NOT NULL
   DROP PROCEDURE dbo.removemember
IF OBJECT_ID('dbo.loanbook') IS NOT NULL
   DROP PROCEDURE dbo.loanbook
IF OBJECT_ID('dbo.getmember') IS NOT NULL
   DROP PROCEDURE dbo.getmember

IF OBJECT_ID('dbo.getloans') IS NOT NULL
   DROP PROCEDURE dbo.getloans

IF OBJECT_ID('dbo.gettitle') IS NOT NULL
   DROP PROCEDURE dbo.gettitle

IF OBJECT_ID('dbo.recordloan') IS NOT NULL
   DROP PROCEDURE dbo.recordloan

IF OBJECT_ID('dbo.returnbook') IS NOT NULL
   DROP PROCEDURE dbo.returnbook

IF OBJECT_ID('dbo.recordreturn') IS NOT NULL
   DROP PROCEDURE dbo.recordreturn

IF EXISTS ( SELECT error FROM master..sysmessages
         WHERE error = 50010 )
   EXEC sp_dropmessage 50010
```

```
IF EXISTS ( SELECT error FROM master..sysmessages
          WHERE error = 50011 )
   EXEC sp_dropmessage 50011

IF EXISTS ( SELECT error FROM master..sysmessages
          WHERE error = 50012 )
   EXEC sp_dropmessage 50012

IF EXISTS ( SELECT error FROM master..sysmessages
          WHERE error = 50013 )
   EXEC sp_dropmessage 50013
GO


/* ADDJUVENILE: 添加小孩信息到Member和Juvenile表*/
CREATE PROCEDURE dbo.addjuvenile
   @lastname          shortstring = NULL,
   @firstname         shortstring = NULL,
   @middleinitial     letter = NULL,
   @adult_member_no   member_no = NULL,
   @birth_date        datetime = NULL
AS

   IF   @lastname = NULL OR
        @firstname = NULL OR
        @adult_member_no =  NULL OR
        @birth_date      =  NULL

   BEGIN
      PRINT 'Please reenter this Juvenile Member.'
      PRINT 'You must provide Last name, First name, '
      PRINT 'Middle initial, Adult Member Number, Birth_date.'
        PRINT 'Middle initial can be null.'
      RETURN
   END

   BEGIN TRANSACTION

     IF NOT EXISTS
        (SELECT *
         FROM adult
         WHERE member_no = @adult_member_no)
      BEGIN
         PRINT 'ERROR:  Adult Member Number not found in Adult table.'
         PRINT 'Transaction terminated.'
         ROLLBACK TRAN
         RETURN
      END

      INSERT member
         (lastname,  firstname,  middleinitial, photograph)
```

```
        VALUES
           (@lastname, @firstname, @middleinitial, NULL )
        IF @@error <> 0
         BEGIN
          ROLLBACK TRAN
          RETURN
         END

        DECLARE @insertmem_no member_no
        SELECT @insertmem_no=@@identity

        INSERT juvenile
           (member_no,  adult_member_no,  birth_date)
        VALUES
           (@insertmem_no, @adult_member_no, @birth_date)
         IF @@error <> 0
           BEGIN
            ROLLBACK TRAN
             RETURN
           END

         PRINT '***  Juvenile Member added  *** '

   COMMIT TRANSACTION
GO


/*
REMOVEMEMBER：删除借阅者信息，并同时删除在 adult表或juvenile表的相关信息，也删除该读者的预约信息。如果该读者还借着书，则不能删除。如果该读者是在 juvenile表中某个小孩的父/母亲，则也不能删除。否则，违反参照完整性。
*/

/* 为removemember存储过程定义一些自定义信息*/
EXEC sp_addmessage 50010, 10, 'Member number not found.'
EXEC sp_addmessage 50011, 10, 'Member cannot be deleted.'
EXEC sp_addmessage 50012, 10, 'Member is responsible for Juvenile member.'
EXEC sp_addmessage 50013, 10, 'Member currently has books on loan.'
GO

CREATE PROCEDURE dbo.removemember
   @member_no member_no = NULL
AS
   IF @member_no is null
   BEGIN
   PRINT 'You must supply a member number'
   RETURN
   END

   IF NOT EXISTS
         (SELECT *
```

```
    FROM member
    WHERE member_no = @member_no)
BEGIN
    RAISERROR (50010, 10, 1) --Member not found.
    RETURN
END

IF EXISTS (SELECT member_no
                FROM juvenile
                WHERE @member_no = adult_member_no)
BEGIN
RAISERROR (50011, 10, 1) --Member cannot be deleted
RAISERROR (50012, 10, 1) --Member responsible for juveniles
SELECT juvenile.member_no,
              juvenile_name=RTRIM(lastname) + ' ', firstname, birth_date
FROM juvenile
JOIN member ON juvenile.member_no = member.member_no
WHERE @member_no = adult_member_no
PRINT 'Remove the juvenile member(s) first or specify a new responsible adult member'
         RETURN
END

IF EXISTS (SELECT member_no
                FROM loan
                WHERE member_no = @member_no)
BEGIN
   RAISERROR (50013, 10, 1) --Books on loan.
   RAISERROR (50011, 10, 1) --Member cannot be deleted.
   SELECT due_date, isbn, copy_no, title
   FROM OnloanView
   WHERE member_no = @member_no
    RETURN
END

IF EXISTS (SELECT member_no
               FROM loanhist
               WHERE member_no = @member_no)
BEGIN
SELECT 'Member owes $' +
               ISNULL(CAST(sum(ISNULL(fine_assessed,0.00) - ISNULL(fine_paid,0.00))
                 - sum(ISNULL(fine_waived,0.00)) AS char(6)),0.00)+' fines'
FROM loanhist
WHERE member_no = @member_no
END

BEGIN TRANSACTION

IF EXISTS (SELECT member_no
               FROM reservation
               WHERE member_no = @member_no)
BEGIN
```

```
    PRINT 'Deleting Loan Reservation information'
        DELETE reservation
      WHERE member_no = @member_no
    END

    IF EXISTS (SELECT member_no
                  FROM juvenile
                  WHERE member_no = @member_no)
    BEGIN
       DELETE juvenile
    WHERE member_no = @member_no
    END

    ELSE IF EXISTS (SELECT member_no
                        FROM adult
                        WHERE member_no = @member_no)
    BEGIN
       DELETE adult
       WHERE member_no = @member_no
    END

    DELETE member
    WHERE member_no = @member_no
    SELECT 'Member ' + CONVERT(char(6), @member_no)
     + 'has been removed from the library database.'

    COMMIT TRANSACTION
GO

/* GETMEMBER: Supports LOANBOOK stored procedure. */
CREATE PROCEDURE dbo.getmember
    @member_no member_no
AS

SELECT       Member = member_no,
      Name   = convert (char(20), lastname + ', ' + firstname),
      Street = street,
      City   = city,
      State  = state,
\     Zip    = zip,
      Phone  = phone_no,
      Expire = expr_date
FROM AdultwideView
WHERE member_no = @member_no
UNION
SELECT       Member = member_no,
      Name   = convert (char(20), lastname + ', ' + firstname),
      Street = street,
      City   = city,
      State  = state,
      Zip    = zip,
```

```
    Phone  = phone_no,
    Expire = expr_date
FROM ChildwideView
WHERE member_no = @member_no

RETURN (@@ROWCOUNT)
GO

/* GETLOANS: Supports LOANBOOK stored procedure. */
CREATE PROCEDURE dbo.getloans
  @member_no member_no
AS
SELECT      Isbn   = isbn,
    Copy   = copy_no,
    Title  = title,
    Out    = out_date,
    Due    = due_date
FROM OnloanView
WHERE member_no = @member_no

RETURN (@@ROWCOUNT)
GO

/* GETTITLE: Supports LOANBOOK stored procedure. */
CREATE PROCEDURE dbo.gettitle
  @isbn     isbn,
  @copy_no  smallint
AS
SELECT      Isbn   = isbn,
    Copy   = copy_no,
    Title  = title,
    Out    = getdate(),
    Due    = dateadd(day, 14, getdate())
FROM LoanableView
WHERE isbn = @isbn
AND copy_no = @copy_no

RETURN (@@ROWCOUNT)
GO

/* RECORDLOAN: Supports LOANBOOK stored procedure. */
CREATE PROCEDURE dbo.recordloan
  @member_no  member_no,
  @isbn       isbn,
  @copy_no    smallint
AS
  IF (SELECT COUNT(*)
    FROM loan
    WHERE isbn = @isbn
    AND copy_no = @copy_no) > 0
  BEGIN
```

```
    DELETE loan
    WHERE isbn = @isbn
    AND copy_no = @copy_no
  END

  INSERT INTO loan
      (isbn, copy_no, title_no, member_no, out_date, due_date)
  SELECT
      isbn,
      copy_no,
      title_no,
      @member_no,
      getdate(),
      dateadd(day, 14, getdate())
  FROM copy
  WHERE copy.isbn = @isbn
  AND copy.copy_no = @copy_no

  RETURN (@@ROWCOUNT)
GO

/*
LOANBOOK: 用于借书的存储过程。向Loan表中插入一行借书信息。并更新 Copy表的相关数据。本存储过程
调用4个存储过程： GETMEMBER、GETLOANS、GETTITLE和RECORDLOAN。这四个存储过程都会返回
行数。
*/
CREATE PROCEDURE dbo.loanbook
  @member_no member_no,
  @isbn      isbn,
  @copy_no   smallint
AS

DECLARE
  @rowcount int

BEGIN
  EXEC @rowcount = getmember @member_no
  IF @rowcount = 0
  BEGIN
    PRINT 'Error:  Member not found as Adult nor Juvenile.'
    PRINT 'Transaction terminated.'
    RETURN
  END

  EXEC @rowcount = getloans  @member_no

  EXEC @rowcount = gettitle @isbn, @copy_no
  IF @rowcount = 0
  BEGIN
    PRINT 'Error:  Isbn / copy_no not found as a loanable copy.'
    PRINT 'Transaction terminated.'
```

```
      RETURN
   END

   IF (SELECT count(*)
      FROM loan
      WHERE isbn = @isbn
      AND copy_no = @copy_no) > 0
   BEGIN
      PRINT 'WARNING:  Copy already on loan.'
      PRINT 'Transaction continuing.'
   END

   BEGIN TRANSACTION
   EXEC @rowcount = recordloan @member_no, @isbn, @copy_no
   IF @rowcount = 0
   BEGIN
      PRINT 'ERROR:  Unable to add record into loan table.'
      PRINT 'Transaction terminated.'
      ROLLBACK TRANSACTION
      RETURN
   END
   ELSE
   BEGIN
      COMMIT TRANSACTION
   END

   EXEC @rowcount = getloans  @member_no

   PRINT 'Transaction complete.'
END
GO

/*
RECORDRETURN: 记录借还书的历史信息。向Loanhist表中插入一行借还书信息。并删除Loan表中的借书信息和更新Copy表的相关书的状态信息。
*/
CREATE PROCEDURE dbo.recordreturn
  @isbn          isbn,
  @copy_no       smallint,
  @fine_assessed  money    = NULL,
  @fine_paid      money    = NULL,
  @remarks        remarks  = NULL
AS

BEGIN TRANSACTION

   INSERT loanhist
      (isbn, copy_no, out_date, title_no, member_no, due_date,
      in_date, fine_assessed, fine_paid, fine_waived, remarks)
     SELECT
               @isbn,
```

```
                @copy_no,
                loan.out_date,
                loan.title_no,
                loan.member_no,
                loan.due_date,
                GETDATE(),
                @fine_assessed,
                @fine_paid,
                NULL,
                @remarks
        FROM loan
        WHERE isbn = @isbn
        AND copy_no = @copy_no

    DELETE loan
      WHERE isbn = @isbn
      AND copy_no = @copy_no

    UPDATE copy
      SET on_loan = 'N'
      WHERE isbn    = @isbn
      AND copy_no = @copy_no

COMMIT TRANSACTION

RETURN (@@ROWCOUNT)
GO
/*
RETURNBOOK: Receive a loaned book back from the member.
Calls the RECORDRETURN stored procedure that deletes loan table row,
updates copy table row, and inserts a row into loanhist table.
*/
CREATE PROCEDURE dbo.returnbook
  @isbn          int,
  @copy_no       smallint,
  @fine_assessed  money    = NULL,
  @fine_paid      money    = NULL,
  @remarks        remarks  = NULL
AS

    IF (SELECT COUNT(*)
      FROM OnloanView
      WHERE isbn    = @isbn
        AND copy_no = @copy_no) = 0
      BEGIN
        PRINT 'Note:  Records show copy as not currently loaned.'
      END

    SELECT isbn,
              title,
```

```
                author,
            lastname,
             firstname
    FROM OnloanView
    WHERE isbn    = @isbn
      AND copy_no = @copy_no

    EXEC recordreturn @isbn, @copy_no, @fine_assessed, @fine_paid, @remarks

    PRINT '       *** RETURN COMPLETE ***'
GO

/* List all stored procedures in Library database to verify script created all. */

SELECT name
FROM sysobjects
WHERE type = 'P'
AND name NOT LIKE 'dt_%'
ORDER BY name
GO
```

结果显示如下：

```
(1 row(s) affected)

New message added.

(1 row(s) affected)

New message added.

(1 row(s) affected)

New message added.

(1 row(s) affected)

New message added.
name
----------------------------------------------------
addadult
addjuvenile
getloans
getmember
gettitle
loanbook
recordloan
recordreturn
removemember
returnbook
TotalBooksLoaned

(11 row(s) affected)
```

10.4 存储过程的处理

当SQL Server接收到创建一个存储过程的命令（CREATE PROC ...）时，由SQL Server的查询处理器对该存储过程中的 SQL语句进行语法分析，检查其是否合乎语法规范。并将该存储过程的源代码存放在当前数据库的系统表 syscomments中。也在sysobjects表中存放该存储过程的名字。

当存储过程第一次运行时， SQL Server首先对该存储过程进行预编译，即为该存储过程建立一棵查询树，这个过程被称为“ resolution ”—— 分解SQL语句中的对象，并为该存储过程建立一个规范化的查询树。然后， SQL Server为这个存储过程完成编译（compilation）。该步骤分成两步：查询优化（optimization）和在高速缓存（procedure cache）中建立查询计划。最后，系统就可以执行这个存储过程了。

这也是一般SQL语句处理的步骤。

10.5 重编译选项

在某些应用中，你可能改变了数据库的逻辑结构（如：为表新增列），或者为表新增了索引，这样的话，你可能要求 SQL Server在执行存储过程时对它重新编译，以便该存储过程能够重新优化并建立新的查询计划（如：选择新建的索引等）。以下是重编译选项的3种方法。

1. CREATE PROCEDURE中的RECOMPILE

在创建存储过程时带上重编译选项。具体语法如下：

```
CREATE PROCEDURE...[WITH RECOMPILE]
```

这样的话， SQL Server对这个存储过程不重用查询计划，在每次执行时都被重新编译和优化,并创建新的查询计划。

例 为library数据创建一个存储过程，用于查询某本书的借 /还历史信息。

```
CREATE PROC testproc
@title_no title_no
WITH RECOMPILE
AS
SELECT*FROM loanhist
WHERE title_no=@title_no
```

结果显示：

```
The command(s) completed successfully.
```

2. EXECUTE中的RECOMPILE

在EXECUTE语句中使用WITH RECOMPILE选项，让SQL Server在执行一个存储过程时，重新编译该存储过程。

语法：EXECUTE procedure_name[parameter][WITH RECOMPILE]

例：EXEC sp_help WITH RECOMPILE

作用：在执行存储的过程期间创建新的查询计划 ,新的执行计划存放在高速缓存中。

3. SP_RECOMPILE表名

使用sp_recompile系统存储过程。

语法：sp_recompile表名

例 EXEC sp_recompile title

作用：使指定表的存储过程和触发器在下一次运行时被重新编译。

10.6 自动执行存储过程

在SQL Server启动时，可以让SQL Server自动执行一个或多个存储过程。那么，这些存储过程就称为自动执行存储过程。以下是一些系统存储过程，可以帮助你创建和停止自动执行存储过程。下面也提供了查看自动执行存储过程的系统存储过程。

系统存储过程	描　述
sp_makestartup	使已有的存储过程成为启动存储过程
sp_unmarestartup	停止在启动时执行过程
sp_helpstartup	提供所有在启动时执行的存储过程的列表

值得注意的是，在执行自动执行存储过程时 ,每一个启动过程要消耗一个连接。你可以修改NT注册表中的跟踪标志，使得系统在启动时跳过自动执行存储过程。

10.7 查看、修改和删除存储过程

10.7.1 查看存储过程

存储过程被创建以后，它的名字存放在系统表 sysobjects中，它的创建源代码存放在syscomments系统表中。可以通过 SQL Server提供的系统存储过程来查看关于用户创建的存储过程信息。

- 查看创建存储过程的源代码：

sp_helptext 存储过程名

例如：查看totalbooksloaned存储过程的源代码，可执行 sp_helptext totalbooksloaned。结果显示如下：

```
Text
------------------------------------------------------------
CREATE PROC dbo.TotalBooksLoaned
AS
SELECT l.member_no, m.lastname, m.firstname, m.middleinitial,
  COUNT(l.member_no) TotalBooksLoaned
FROM member m
JOIN loanhist l
  ON l.member_no = m.member_no
GROUP BY l.member_no, lastname, firstname, middleinitial
ORDER BY totalbooksloaned DESC
```

- 查看存储过程的一般信息（如创建日期等）：

sp_help 存储过程名

- 查看系统表

如：查看totalbooksloaned 存储过程的对象ID号。

```
  SELECT OBJECT_ID('dbo.totalbooksloaned')
```

结果显示如下：

```
-----------
1669580986

(1 row(s) affected)
```

10.7.2 修改存储过程

为了满足用户的新需求，存储过程可能要求被修改。我们可以使用 ALTER PROCEDURE 语句来实现。SQL Server可以增、删一些选项（如：WITH ENCRYPTION等），或者在保留原来名称的同时修改存储过程的内容。具体语法如下：

```
ALTER PROCEDURE  [OWNER.]procedure_name[;number]
[({[@]parameter data_type}[varying][=default][output])]
[,...n]
[WITH{RECOMPILE|ENCRYPTION|RECOMPILE,ENCRYPTION}]
AS
 sql_statements
```

上述语法中包含以下符号：

- procedure_name：要修改的存储过程指定的名字。
- parameter：在存储过程中包含的输入和输出参数。
- WITH RECOMPILE：重编译选项。具体含义见第4节。
- default：为这个参数指定的缺省值。它必须是一个常量。如果该参数使用在LIKE子句中，则该缺省值可以含有通配符。如果给出了缺省值，那么在执行该存储过程时，如果没有向具有缺省值的参数传递参数值时，则具有缺省值的参数就可使用它们的缺省值。
- WITH ENCRYPTION：对包含CREATE PROCEDURE文本的syscomments表中的项加密。
- number:用于将名字相同的过程编组。
- aql_statements：在存储过程中要执行的动作。

例如：我们修改returnbook存储过程，加入了对用户输入参数的验证。当用户归还书本时，系统执行returnbook存储过程。returnbook存储过程的作用是，调用 recordreturn存储过程，删除Loan表中的借书信息，更新书库（ Copy表）中的这本书的状态，并在 Loanhist表中记录这次借书的历史信息。脚本如下：

```
USE library
GO

ALTER PROCEDURE dbo.returnbook
   @isbn          int          = NULL,-- Must specify NULL for student code to work
   @copy_no       smallint = NULL,-- Must specify NULL for student code to work
   @fine_assessed  money    = NULL,
   @fine_paid      money    = NULL,
   @remarks        remarks  = NULL
AS

/* 新添加的代码*/
  IF  @isbn          IS NULL OR
      @copy_no       IS NULL

    BEGIN
      PRINT 'You must enter the isbn and copy number.'
      PRINT 'If applicable, also enter fine assessed, fine paid, and any comments.'
      PRINT 'Example:  returnbook 125, 1, 2.50, 2.50, "Fine paid in full."'
      RETURN
```

```
    END
/*添加代码结束*/
    IF (SELECT COUNT(*)
        FROM OnloanView
        WHERE isbn    = @isbn
          AND copy_no = @copy_no) = 0
        BEGIN
          PRINT 'Note:  Records show copy as not currently loaned.'
        RETURN
        END

    SELECT isbn,title,author,lastname, firstname FROM OnloanView
    WHERE isbn    = @isbn AND copy_no = @copy_no

    EXEC recordreturn @isbn, @copy_no, @fine_assessed, @fine_paid, @remarks

    PRINT '      *** RETURN COMPLETE ***'
GO
```

结果显示如下，表示修改成功。

```
The command(s) completed successfully.
```

使用ALTER PROCEDURE命令修改存储过程，可以让你保留该存储过程的权限分配，避免重新分配权限。

10.7.3 删除存储过程

删除存储过程是指删除由用户创建的存储过程。可以使用DROP PROC命令删除存储过程。具体语法如下：

```
DROP  PROC  procedure_name
```

procedure_name 是要被删除的存储过程名字。

删除存储过程也可以在企业管理器中完成。

10.8 扩展存储过程

虽然 SQL Server的扩展存储过程听起来近似于存储过程，但是，实际上它们两者相差很远。存储过程是一系列预编译的 Transact-SQL语句，而扩展存储过程是对动态链接库（ DLL）函数的调用。扩展存储过程名以“ xp_ ”开始，后跟它的名字。

虽然SQL Server提供了一些扩展存储过程（SQL Server内置的扩展存储过程已经被安装到SQL Server中了），但是扩展存储过程也可以由开发人员来编写。自行编写的扩展存储过程在可以使用之前必须在 SQL Server中进行安装。

你可以编写自己的DLL，然后以扩展存储过程的模式将它安装在 SQL Server上。大多数DLL包括多个函数（任务），每个函数有各自唯一的名字，这意味着在将其作为扩展存储过程安装到SQL Server之前既需要知道DLL的名字，又需要知道所需函数的名字。你可以从这个DLL的开发人员那里得到这部分信息。只有系统管理员可以安装扩展存储过程。

1. 安装扩展存储过程

向SQL Server的master数据库中安装扩展存储过程是使用一个系统存储过程来实现的，语法如下：

```
sp_addextendedproc function_name，dll_name
```

在上面的语法中，“function_name”是指DLL中函数的名字。这个函数名将成为这个扩展存储过程的名字。另外，“dll_name”是DLL的名字。

从上面语法中看出，你可以用开发语言生成一个 DLL，然后使用上述的系统存储过程将它添加到SQL Server中。

2. 删除扩展存储过程

如果不再需要定制的扩展存储过程，你可以使用以下的语法将其从 SQL Server中删除：

```
sp_dropextendedproc function_name
```

在上面的语法中，“function_name”既是DLL中函数的名字又可以说是扩展存储过程的名字。

3. 使用安装过的扩展存储过程

在向SQL Server中添加了扩展存储过程之后，这个扩展存储过程就可以像任何内置的扩展存储过程一样被使用。你只需要像执行任何其它 Transact-SQL语句一样来执行扩展存储过程。你可以在Query Analyzer或者SQL查询工具中运行扩展存储过程。如果要被执行的扩展存储过程需要参数，那么你还需要输入相应的参数。

例如：xp_cmdshell存储过程的作用是，显示在指定目录下的文件。类似执行 DIR命令。

```
EXEC xp_cmdshell 'dir e:\mssql7\data'
```

这是我目前做实验的机器上显示的结果：

```
output
-------------------------------------------------------------------------
驱动器E中的卷没有卷标。
卷的序列号是 C043-CC25
NULL
 e:\mssql7\data 的目录
NULL
99-10-03  05:37p       <DIR>
99-10-03  05:37p       <DIR>
98-11-13  03:28a         1 048 576 distmdl.ldf
98-11-13  03:28a         3 145 728 distmdl.mdf
99-11-01  01:27p         1 048 576 distribution.LDF
99-11-01  01:27p         3 670 016 distribution.MDF
99-10-31  03:58p         2 097 152 fff_Data.NDF
99-11-01  01:27p        20 971 520 library.ldf
99-11-01  01:27p        52 428 800 library.mdf
99-11-01  01:27p         9 240 576 master.mdf
99-11-01  01:27p         1 310 720 mastlog.ldf
99-11-01  01:27p         1 048 576 model.mdf
99-11-01  01:27p           786 432 modellog.ldf
99-11-01  01:27p         8 388 608 msdbdata.mdf
99-11-01  01:27p         1 572 864 msdblog.ldf
99-11-01  01:27p         2 097 152 mydb_Data.MDF
99-11-01  01:27p         1 048 576 mydb_Log.LDF
99-11-01  01:27p         1 048 576 northwnd.ldf
99-11-01  01:27p         3 801 088 northwnd.mdf
99-10-31  11:34a         2 097 152 other_Data.NDF
99-11-01  01:27p         1 835 008 pubs.mdf
```

```
99-11-01  01:27p              786 432 pubs_log.ldf
99-12-04  10:08p            8 388 608 TEMPDB.MDF
99-12-04  10:20p              786 432 TEMPLOG.LDF
          24 个文件     128 647 168 字节
                  1 137 787 392 字节可用

(31 row(s) affected)
```

如果执行如下命令来查看xp_cmdshell的源代码，我们会发现它是DLL。

```
exec sp_helptext xp_cmdshell
```

结果显示如下：

```
Text
----------------------------------
xpsql70.dll
```