

数据库编程技术

本章主要介绍游标的概念以及游标的使用方法,运用触发器完成复杂的完整性约束和审计功能,通过存储过程完成复杂的业务处理和查询统计工作。

5.1 相关知识

本章要求学习使用多种工具,提高解决实际问题的能力。进一步理解并掌握游标和触发器、游标和存储过程的灵活运用。

5.1.1 游标

游标是一种允许用户访问单独的数据行的数据访问机制。游标主要用在存储过程、触发器和 T-SQL 脚本中。使用游标,可以对由 SELECT 语句返回的结果集记录进行逐行处理。使用游标必须经历五个步骤:

- (1) 定义游标: DECLARE。
- (2) 打开游标: OPEN。
- (3) 逐行提取游标集中的行: FETCH。
- (4) 关闭游标: CLOSE。
- (5) 释放游标: DEALLOCATE。

1. 定义游标

语法:

```
DECLARE cursor_name SCROLL CURSOR FOR sql_statements  
[FOR [READ ONLY|UPDATE {OF column_name_list [,...n]]]
```

其中:

- cursor_name: 用户定义的游标名。
- SCROLL: 指定所有的提取选项 (FIRST、LAST、PRIOR、NEXT、RELATIVE、ABSOLUTE) 均可用。如果在 DECLARE CURSOR 中未指定 SCROLL, 则 NEXT 是唯一支持的提取选项。

- sql_staments: 定义游标结果集的标准 SELECT 语句。
- FOR: 后面的短语定义游标属性只读或更新,缺省时为 UPDATE。
- UPDATE {OF column_name_list}: 定义游标内可更新的列。如果指定 OF column_name_list [, ...n] 参数,则只允许修改所列出的列。如果在 UPDATE 中未指定列的列表,则可以更新所有列。
- READ ONLY: 在 UPDATE 或 DELETE 语句的 WHERE CURRENT OF 子句中不能引用游标。该选项替代要更新的游标的默认功能。

注意:

- (1) 当游标移至尾部时,不可以再读取游标,必须关闭游标然后重新打开游标。
- (2) 可以通过检查全局变量 @@fetch_status 来判断是否已读完游标集中的所有行。

2. 打开游标

使用 OPEN 语句执行 SELECT 语句并生成游标。语法为:

```
OPEN curser_name
```

3. 提取游标

- (1) 逐行提取游标集中的行:

```
FETCH curser_name [INTO @variable_name [,...n]]
```

- (2)

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

其中,

- NEXT: 返回紧跟当前行之后的结果行,并且当前行递增为结果行。如果 FETCH NEXT 为对游标的第一次提取操作,则返回结果集中的第一行。NEXT 为默认的游标提取选项。
- PRIOR: 返回紧临当前行前面的结果行,并且当前行递减为结果行。如果 FETCH PRIOR 为对游标的第一次提取操作,则没有行返回并且游标置于一行之前。
- FIRST: 返回游标中的第一行并将其作为当前行。
- LAST: 返回游标中的最后一行并将其作为当前行。
- ABSOLUTE {n|@nvar}: 如果 n 或 @nvar 为正数,返回从游标头开始的第 n 行并将返回的行变成新的当前行。如果 n 或 @nvar 为负数,返回游标尾之前的第 n 行并将返回的行变成新的当前行。如果 n 或 @nvar 为 0,则没有行返回。n 必须为整型常量且 @nvar 必须为 smallint、tinyint 或 int。

- RELATIVE {n|@nvar}: 如果 n 或 @nvar 为正数,返回当前行之后的第 n 行并将返回的行变成新的当前行。如果 n 或 @nvar 为负数,返回当前行之前的第 n 行并将返回的行变成新的当前行。如果 n 或 @nvar 为 0,返回当前行。如果对游标的第一次提取操作时将 FETCH RELATIVE 的 n 或 @nvar 指定为负数或 0,则没有行返回。n 必须为整型常量且 @nvar 必须为 smallint、tinyint 或 int。
- INTO @variable_name [, ...n]: 把每列中的数据转移到指定的变量中。

4. 关闭游标

关闭游标可以释放某些资源,如游标结果集和对当前行的锁定。如果重新发出一个 OPEN 语句,则该游标结构仍可用于处理。语法为:

```
CLOSE curser_name
```

5. 释放游标

DEALLOCATE 语句则完全释放分配给游标的资源,包括游标名称。在游标被释放后,必须使用 DECLARE 语句重新生成游标。语法为:

```
DEALLOCATE curser_name
```

6. 删除游标集中当前行

语法:

```
DELETE FROM table_name WHERE CURRENT OF curser_name
```

注意:从游标中删除一行后,游标定位于被删除的游标之后的一行,必须再用 FETCH 得到该行。

7. 更新游标集中当前行

语法:

```
UPDATE table_name SET column_name=expression [,column_name=expression]  
WHERE CURRENT OF curser_name
```

5.1.2 存储过程

SQL Server 提供了一种方法,它可以将一些固定的操作集中起来由 SQL Server 数据库服务器来完成,以实现某个任务,这种方法就是存储过程。存储过程是经过编译和优化后存储在数据库服务器中 SQL 语句写的过程,使用时只要调用即可。存储过程的优点是:

- 提供了在服务器端快速执行 SQL 语句的有效途径。
- 降低了客户机和服务器之间的通信量。

- 方便实施企业规则。
- 业务封装后,对数据库系统提供了一定的安全保证。

(1) 使用 CREATE PROCEDURE 命令创建存储过程前,应考虑下列几个事项:

- ① 不能将 CREATE PROCEDURE 语句与其他 SQL 语句组合到单个批处理中。
- ② 创建存储过程的权限默认属于数据库所有者,该所有者可将此权限授予其他用户。
- ③ 存储过程是数据库对象,其名称必须遵守标识符规则。
- ④ 只能在当前数据库中创建存储过程。
- ⑤ 一个存储过程的最大规模为 128MB。

(2) 创建存储过程时,需要确定存储过程的 3 个组成部分:

- ① 所有的输入参数以及传给调用者的输出参数。
- ② 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。
- ③ 返回给调用者的状态值,以指明调用是成功还是失败。

1. 创建存储过程

语法:

```
CREATE PROCEDURE procedure_name [;number] [{@parameter datatype}
    [OUTPUT]][, ...n]
AS
    sql_statement [, ...n]
```

其中,

- procedure_name: 存储过程的名称。创建临时过程,在 procedure_name 前面加一个编号符,即 # procedure_name;创建全局临时过程,在 procedure_name 前面加两个编号符,即 ## procedure_name。完整的名称(包括 # 或 ##)不能超过 128 个字符。过程所有者的名称是可选的。
- number: 是可选的整数,用来对同名的过程分组,以使用一条 DROP PROCEDURE 语句即可将同组的过程一起删除。

例如,名为 orders 的应用程序使用的过程可以命名为 orderproc; 1、orderproc; 2 等。DROP PROCEDURE orderproc 语句将除去整个组。

- @parameter: 过程中的参数,最多可以有 2100 个参数。
- datatype: 参数的数据类型。所有数据类型(包括 text、ntext 和 image)均可以用作存储过程的参数。
- OUTPUT: 表明参数是输出参数,text、ntext 和 image 参数可用作 OUTPUT 参数。使用 OUTPUT 关键字的输出参数可以是游标占位符。
- n: 表示最多可以指定 2100 个参数的占位符。
- AS: 指定过程要执行的操作。
- sql_statement: 过程中的 Transact-SQL 语句。

2. 执行存储过程

语法:

```
EXECUTE {procedure_name [; number] | @Parameter_name} [OUTPUT] [, ...n]
```

其中:

- procedure_name: 拟调用的存储过程名。
- @procedure_name_var: 局部定义的变量名。
- @parameter_name: 过程参数,在 CREATE PROCEDURE 语句中定义。参数名称前必须加上符号@。在以 @parameter_name=value 格式使用时,参数名称和常量不一定按照 CREATE PROCEDURE 语句中定义的顺序出现。但是,如果有一个参数使用 @parameter_name=value 格式,则其他所有参数都必须使用这种格式。
- OUTPUT: 指定存储过程必须返回一个参数。使用 OUTPUT 参数,目的是在调用批处理或过程的其他语句中使用其返回值,参数值必须作为变量传递。在执行过程之前,必须声明变量的数据类型并赋值。返回参数可以是 text 或 image 数据类型以外的任意数据类型。

3. 重命名存储过程

语法:

```
ALTER PROCEDURE procedure_name1 RENAME TO procedure_name2
```

4. 删除存储过程

语法:

```
DROP PROCEDURE procedure_name
```

5.1.3 触发器

触发器是一种特殊的存储过程,当 INSERT、DELETE 或 UPDATE 语句修改指定表的一行或多行时,自动执行触发器。

(1) 在触发器的使用中,系统会自动产生两个临时表 Deleted 和 Inserted。用户不能直接修改这两个表的内容。

① Deleted 表: 存储 DELETE 和 UPDATE 语句执行时所影响的行的副本。在 DELETE 和 UPDATE 语句执行前被作用的行转移到 Deleted 表中。

② Inserted 表: 存储 INSERT 和 UPDATE 语句执行时所影响的行的副本。在 Insert 和 UPDATE 语句执行期间,新行被同时加到 Inserted 和触发器表中。实际上 UPDATE 命令是删除后紧跟着插入,旧行首先复制到 Deleted 表中,新行同时复制到 Inserted 和触发器表中。

(2) 触发器仅在当前 DB 中生成。触发器有 3 种类型,即插入、删除和更新。

① 插入(INSERT)类型的触发器:当对指定表 TableName 执行了插入操作时系统自动执行触发器代码。

② 更新(UPDATE)类型的触发器:当对指定表 TableName 执行了更新操作时系统自动执行触发器代码。

③ 删除(DELETE)类型的触发器:当对指定表 TableName 执行了删除操作时系统自动执行触发器代码。

(3) 在触发器内不能使用如下 SQL 命令:

① 所有数据库对象的生成命令,如 CREATE TABLE、CREATE INDEX 等。

② 所有数据库对象的结构修改命令,如 ALTER TABLE、ALTER DATABASE 等。

③ 创建临时保存表。

④ 所有 DROP 命令。

⑤ GRANT 和 REVOKE 命令。

⑥ TRUNCATE TABLE 命令。

⑦ LOAD DATABASE 和 LOAD TRANSACTION 命令。

⑧ RECONFIGURE 命令。

1. 创建触发器

语法:

```
CREATE TRIGGER trigger_name
ON table_name
FOR <INSERT|UPDATE|DELETE>
AS
    sql_statement
```

2. 删除触发器

语法:

```
DROP TRIGGER trigger_name
```

3. 修改触发器

语法:

```
ALTER TRIGGER trigger_name
ON table_name
FOR <INSERT|UPDATE|DELETE>
AS
    sql_statement
```

5.2 实验十二 游标与存储过程

5.2.1 实验目的与要求

- (1) 掌握游标的定义和使用方法。
- (2) 掌握存储过程的定义、执行和调用方法。
- (3) 掌握游标和存储过程的综合应用方法。

5.2.2 实验案例

下面以简单实例介绍游标的具体用法。

[例 5.1] 利用游标查询业务科员工的编号、姓名、性别、部门和薪水,并逐行显示游标中的信息。

```

DECLARE cur_emp SCROLL CURSOR FOR
SELECT employeeNo, employeeName, sex, department, salary
FROM employee
WHERE department='业务科'
ORDER BY employeeNo           /* 定义游标 */
OPEN cur_emp                 /* 打开游标 */
SELECT 'CURSOR 内数据条数'=@@cursor_rows /* 显示游标内记录的个数 */
FETCH NEXT FROM cur_emp      /* 逐行提取游标中的记录 */
WHILE (@@FETCH_status<>-1)  /* 判断 FETCH 语句是否执行成功 */
BEGIN
    SELECT 'cursor 读取状态'=@@FETCH_status /* 显示游标的读取状态 */
    FETCH NEXT FROM cur_emp /* 提取游标下一行信息 */
END
CLOSE cur_emp                /* 关闭游标 */
DEALLOCATE cur_emp           /* 释放游标 */

```

本例中,@@cursor_rows 是返回连接上最后打开的游标中当前存在的合格行的数量。具体参数信息如表 5-1 所示。

表 5-1 @@cursor_rows 参数返回值的含义

返回值	含 义
-m	游标被异步填充,返回值(-m)是键集中当前的行数
-1	游标为动态。因为动态游标可反映所有更改,所以符合游标的行数不断变化,因而永远不能确定地说所有符合条件的行均已检索到
0	没有被打开的游标,没有符合最后打开的游标的行,或最后打开的游标已被关闭或被释放
n	游标已完全填充,返回值(n)是在游标中的总行数

@@FETCH_status 是返回被 FETCH 语句执行的最后,而不是任何当前被连接打

开的游标的状态。具体参数如表 5-2 所示。

表 5-2 @@FETCH_status 参数返回值的含义

返回值	含 义
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 5.2] 利用游标查询业务科员工的编号、姓名、性别、部门和薪水,并以格式化的方式输出游标中的信息。

```

DECLARE @emp_no char(8), @emp_name char(10), @sex char(1), @dept char(4)
DECLARE @salary numeric(8,2), @text char(100) /* 用户自定义的几个变量 */
DECLARE emp_cur SCROLL CURSOR FOR
SELECT employeeNo, employeeName, sex, department, salary
FROM Employee
WHERE department='业务科'
ORDER BY employeeNo /* 定义游标 */
SELECT @text='=====业务科员工情况列表===== '
PRINT @text
SELECT @text=' 编号      姓名      性别  部门  薪水'
PRINT @text
SELECT @text='----- '
PRINT @text /* 按照用户要求格式化输出相关信息 */
OPEN emp_cur /* 打开游标 */
FETCH emp_cur INTO @emp_no, @emp_name, @sex, @dept, @salary
/* 提取游标中的信息传递并分别给内存变量 */
WHILE (@@FETCH_status=0) /* 判断是否提取成功 */
BEGIN
    SELECT @text=@emp_no+' '+@emp_name+' '+@sex+' '+
           @dept+' '+convert(char(10), @salary) /* 给@text 赋字符串值 */
    PRINT @text /* 打印字符串值 */
    /* 提取游标中的信息并分别传递给内存变量 */
    FETCH emp_cur into @emp_no, @emp_name, @sex, @dept, @salary
END
CLOSE emp_cur /* 关闭游标 */
DEALLOCATE emp_cur /* 释放游标 */

```

本例中,主要结合 SELECT 和 PRINT 命令将创建游标后逐行提取游标的信息以格式化的方式输出,这就提高了脚本的可读性。

[例 5.3] 不带参数的存储过程:利用存储过程计算出 E2015002 业务员的销售总金额。

① 创建存储过程。

```
CREATE PROCEDURE sales_tot1
```

```
AS
SELECT sum(orderSum)
FROM OrderMaster
WHERE salerNo='E2015002'
```

② 执行存储过程。

```
EXEC sales_tot1
```

上述操作能够统计业务员 E2015002 的销售业绩,但执行此存储过程不能统计任一业务员的销售业绩。

[例 5.4] 带输入参数的存储过程:统计某业务员的销售总金额。

① 创建存储过程。

```
CREATE PROCEDURE sales_tot2 @e_no char(8)
AS
SELECT sum(orderSum)
FROM OrderMaster
WHERE salerNo= @e_no
```

② 执行存储过程。

```
EXEC sales_tot2 'E2015003'
```

注意: 程序中使用@符号表示一个变量来指定参数名称,且每个过程的参数仅用于该过程本身。

上述操作只要在执行存储过程时添加输入参数(即被统计的业务员的编号)就能统计任一业务员的销售业绩。但是任一业务员的销售总金额如何被其他用户/程序方便地调用呢?

[例 5.5] 带输入/输出参数的存储过程:统计某业务员的销售总金额并返回其结果。

① 创建存储过程。

```
CREATE PROCEDURE sales_tot3 @e_no char(8), @p_tot int OUTPUT
AS
SELECT @p_tot=sum(orderSum)
FROM OrderMaster
WHERE salerNo=@e_no
```

② 执行存储过程。

```
DECLARE @tot_amt int
EXEC sales_tot3 'E2008003', @tot_amt OUTPUT
SELECT 销售总额=@tot_amt
```

上述操作可以统计任一业务员的销售业绩并能实现其结果的调用。

[例 5.6] 带通配符参数的存储过程(模糊查找):统计所有姓陈的员工的销售业绩并输出其姓名和所在部门。

① 创建存储过程。

```
CREATE PROCEDURE emp_name @e_name varchar(10)
AS
    SELECT a.EmployeeName, a.department, ssum
    FROM Employee a, ( SELECT SalerNo, ssum=sum(OrderSum)
                      FROM OrderMaster
                      GROUP BY SalerNo) b
WHERE a.EmployeeNo=b.SalerNo AND a.EmployeeName LIKE @e_name
```

② 执行存储过程。

```
EXEC emp_name @e_name='陈%'
```

[例 5.7] 重命名存储过程：将存储过程 sales_tot2 销售总额改名为 sale_tot。

```
Sp_rename 'sales_tot2', 'sale_tot'
```

[例 5.8] 删除存储过程：将存储过程 sale_tot 删除。

```
DROP PROCEDURE sale_tot
```

[例 5.9] 游标和存储过程的综合应用：请使用游标和循环语句编写一个存储过程 emp_tot。根据业务员姓名，查询该业务员在销售工作中的客户信息及每一位客户的销售记录，并输出该业务员的销售总金额。

① 创建存储过程。

```
CREATE PROCEDURE emp_tot @v_emp_name char(10)
AS
BEGIN
    DECLARE @sv_emp_name varchar(10), @v_custname varchar(10), @p_tot int
    DECLARE @sum int, @count int, @order_no varchar(10)
    SELECT @sum=0, @count=0
    DECLARE get_tot CURSOR FOR
        SELECT EmployeeName, CustomerNo, b.OrderNo, OrderSum
        FROM Employee a, OrderMaster b
        WHERE a.EmployeeName=@v_emp_name AND a.EmployeeNo=b.SalerNo
    OPEN get_tot
    FETCH get_tot INTO @sv_emp_name, @v_custname, @order_no, @p_tot
    WHILE (@@FETCH_status=0)
    BEGIN
        SELECT 业务员=@sv_emp_name, 客户=@v_custname,
              订单编号=@order_no, 订单金额=@p_tot
        SELECT @sum=@sum+@p_tot
        SELECT @count=@count+1
        FETCH get_tot INTO @sv_emp_name, @v_custname, @order_no, @p_tot
    END
    CLOSE get_tot
```