数据库系统原理与设计

（第3版）

树立健康的人生观、世界观

- 胸怀正面的世界观，维持自己的人格
- 不靠别人，要靠自己，为自己的未来念书
- 不搞自我中心，具有团队精神，待人和善有礼
- 要看到客观场景（Context）：
  - 要全面，不片面
  - 不光看当前，也看过去与未来

数据库系统原理与设计

（第 3 版）

第7章　SQL数据定义、更新及数据库编程

目　录

SQL数据定义语言

SQL数据更新语言

视　　图

T-SQL语言简介

游　　标

存储过程

触 发 器

7.4　T-SQL语言简介

■ 变量

- 局部变量：变量名前加1个@符号
- 全局变量：变量名前加2个@@符号。如：
  - @@ERROR：当事务成功时为0，否则为最近一次的错误号
  - @@ROWCOUNT：返回受上一语句影响的行数
  - @@FETCH_STATUS：返回最近的FETCH语句执行后的游标状态

■ 变量的声明与赋值

- 声明变量的语法：

  DECLARE <@variableName> <datatype> [, <@variableName> <datatype> …]

- 单个变量赋值的语法：SET <@variableName> = <expr>
- 变量列表赋值(或显示表达式的值)的语法：

  SELECT <@variableName> [= <expr | columnName>]
  [, <@variableName> [= <expr | columnName>] …]

7.4　T-SQL语言简介

■ 运算符

- ●算术运算符：+，-，*，/，%(取余)
- ●比较运算符：>，>=，<，<=，=，<>，!=
- ●逻辑运算符：AND，OR，NOT
- ●位运算符：&(按位与)，|(按位或)，~(按位非)，^(按位异或)
- ●字符串连接运算符：+
- ●赋值语句：
  - ➢ SELECT：一次可赋值多个变量，或显示多个表达式的值
  - ➢ SET：一次仅能给一个变量赋值

■ 函数：数学函数、字符串函数、日期和时间函数、聚合函数和系统函数等

- ● 数学函数：绝对值函数abs、随机数函数rand、四舍五入函数round、上取整函数ceiling、下取整函数floor、指数函数exp、平方根函数sqrt等

7.4　T-SQL语言简介

■ **函数：** 数学函数、字符串函数、日期和时间函数、聚合函数和系统函数等

● **字符串函数：**

表 7-1　部分字符串函数

函数名	函数功能
charindex(*expr*1, *expr*2 [, *start_location*])	返回字符串中指定表达式的起始位置
left(*character_expr*, *integer_expr*)	返回从字符串左边开始指定字符个数的字符串
len(*string_expr*)	返回给定字符串的长度(不包含尾随空格)
lower(*character_expr*)	将大写字符转换为小写字符后返回字符表达式
ltrim(*character_expr*)	删除起始空格后返回字符表达式
replicate(*character_expr*, *integer_expr*)	以指定的次数重复字符表达式。
right(*character_expr*, *integer_expr*)	返回字符串中右边的 *integer_expr* 个字符
rtrim(*character_expr*)	截断所有尾随空格后返回一个字符串
space(*integer_expr*)	返回由重复的空格组成的字符串
str(*float_expr* [, *length* [, *decimal*]])	由数字数据转换来的字符数据
substring(*expr*, *start*, *length*)	提取子串函数
upper(*character_expr*)	返回将小写字符数据转换为大写的字符表达式

7.4　T-SQL语言简介

■ **函数：** 数学函数、字符串函数、日期和时间函数、聚合函数和系统函数等

● **日期和时间函数：**

表 7-2　日期和时间函数

函数名	函数功能
dateadd(*datepart*, *number*, *date*)	在指定日期上加一段时间，返回新的 datetime 值
datediff(*datepart*, *startdate*, *enddate*)	返回两个指定日期的日期和时间边界数
datename(*datepart*, *date*)	返回指定日期的指定日期部分的字符串
datepart(*datepart*, *date*)	返回指定日期的指定日期部分的整数
day(*date*)	返回指定日期中日(day)的整数
getdate()	返回当前系统日期和时间
getutcdate()	返回世界时间坐标或格林尼治标准时间的 datetime 值
month(*date*)	返回指定日期中月(month)的整数
year(*date*)	返回指定日期中年(year)的整数

7.4　T-SQL语言简介

■ **函数：** 数学函数、字符串函数、日期和时间函数、聚合函数和系统函数等

● **系统函数：**

表 7-3　系统函数

函数名	函数功能
convert(*data_type* [(*length*)], *expr* [, *style*])	将某种数据类型的表达式转换为另一种数据类型
current_user()	返回当前的用户，等价于 user_name()
datalength(*expr*)	返回任何表达式所占用的字节数
@@ERROR	返回最后执行的 SQL 语句的错误代码
isnull(*check_expr*, *replacement_value*)	使用指定的替换值替换 NULL
@@ROWCOUNT	返回受上一语句影响的行数
session_user()	返回当前会话的用户名
user_name()	返回给定标识号的用户名
host_name()	返回工作站名称
user()	当前数据库用户名

7.4　T-SQL语言简介

■ **函数**：数学函数、字符串函数、日期和时间函数、聚合函数和系统函数等

● **系统函数**：　**convert**(*data_type* [(*length*)], *expr* [, *style*])

- *data_type*：系统所提供的数据类型
- *length*：字符数据类型的可选参数，用于控制字符串的长度
- *expr*：任何有效的SQL Serve表达式
- *style*：日期格式样式(详见表7-4)。

◆ 例如，经常将datetime数据或数值数据表达式*expr*转换为字符数据类型*data_type*，然后可用于字符串的连接输出

[例 7.40] 将当前系统的时间按 104 格式输出。

```
SELECT convert(char(20), getdate(), 104)
```

运行结果如图 7-3 所示。

	(无列名)
1	19.03.2017

图 7-3　例 7.40 的运行结果

[例 7.41] 将当前系统的时间按 120 格式输出。

```
SELECT convert(char(20), getdate(), 120)
```

运行结果如图 7-4 所示。

	(无列名)
1	2017-03-19 00:51:26

图 7-4　例 7.41 的运行结果

7.4　T-SQL语言简介

■ 流程控制语句：

表 7-5　流程控制语句

关键字	功能描述
BEGIN…END	定义语句块
BREAK	退出当前层的 WHILE 循环
CASE WHEN [ELSE] END	多分支语句
CONTINUE	重新开始当前层的 WHILE 循环
GOTO *label*	将程序流程转向到标号 *label* 处继续执行
IF [ELSE]	分支(选择)语句
RETURN	无条件退出
WAITFOR	为语句的执行设置延迟
WHILE	循环语句

7.4　T-SQL语言简介

■ 程序实例：

● [例7.38] 在ScoreDB数据库中，查询Score表中的最高成绩，如果最高成绩大于95分，则显示“very good!”。

```
USE ScoreDB
GO
DECLARE @score numeric
SELECT @score =
        ( SELECT max(score) FROM Score )
IF @score>95
    PRINT 'very good!'
```

```
use ScoreDB
GO
DECLARE @score numeric
SELECT @score = (select
MAX(score) from Score)
if @score > 80
PRINT 'good'
```

7.4　T-SQL语言简介

■ 程序实例：

- **[例7.38] 在ScoreDB数据库中，查询Score表中的最高成绩，如果最高成绩大于95分，则显示“very good!”。**
- **[例7.39] 声明两个局部变量@sno和@score，用于接受SELECT语句查询返回的结果(005号刘方晨)，并显示其结果。**

```
USE ScoreDB
GO
DECLARE @score numeric
SELECT @score =
        ( SELECT max(sc
IF @score>95
    PRINT 'very good!'
```

```
DECLARE @sno char(7), @score numeric
SELECT @sno=a.studentNo, @score=score
FROM Score a, Student b
WHERE a.studentNo=b.studentNo
    AND courseNo='005' AND studentName='刘方晨'
IF @@ROWCOUNT = 0
    PRINT 'Warning: No rows were selected'
ELSE
    SELECT @sno, @score
```

7.4　T-SQL语言简介

■ 程序实例：

● [例7.45] 在学生表Student中，如果有蒙古族学生，则显示：存在蒙古族的学生。

```
IF EXISTS (SELECT * FROM Student WHERE nation='蒙古族')
    PRINT '存在蒙古族的学生'
```

7.4　T-SQL语言简介

■ 程序实例：

● [例7.45] 在学生表Student中，如果有蒙古族学生，则显示：存在蒙古族的学生。

```
IF EXISTS (SELECT * FROM Student WHERE nation='蒙古族')
    PRINT '存在蒙古族的学生'
```

目　录

SQL数据定义语言

SQL数据更新语言

视　　图

T-SQL语言简介

游　　标

存储过程

触 发 器

7.5　游标

- 对SELECT语句的结果集进行**逐行处理**，需使用游标。
- 游标(cursor)是系统为用户开设的一个**数据缓冲区**，用于存放SQL语句的执行结果(元组集合)。每个游标都有一个名字，用户可以用SQL提供的语句从游标中逐一获取元组(记录)，并赋给主变量，交由主语言进一步处理。
- 可对游标的**当前位置**进行更新、查询和删除，使用游标需要经历5个步骤：
  - 定义游标：DECLARE
  - 打开游标：OPEN
  - 逐行提取游标集中的行：FETCH
  - 关闭游标：CLOSE
  - 释放游标：DEALLOCATE

7.5.1 游标的定义与使用

7.5.2 当前游标集的修改与删除

7.5.1 游标的定义与使用

■ 定义游标

- 语法为：

  DECLARE *<cursorName>* **CURSOR**
  FOR *<SQL-Statements>*
  [FOR { READ ONLY | UPDATE [OF *<columnName_list>*] }]

- 在使用游标之前，必须先定义游标。其中：
  - *<cursorName>*：所定义游标的名称；
  - *<SQL-Statements>*：游标要实现的功能程序，即SQL子查询；
  - *<columnName_list>*：属性列名列表；
  - **[FOR { READ ONLY | UPDATE [OF** *<columnName_list>*] }]：
    - ✓ **READ ONLY**表示当前游标集中的元组仅可查询，不能修改；
    - ✓ **UPDATE [OF** *<columnName_list>*]表示可对当前游标集中的元组进行更新操作。
    - ✓ 如果有**OF** *<columnName_list>*，表示仅可以对游标集中指定的属性列进行修改操作；缺省为**UPDATE**

7.5.1 游标的定义与使用

需要经历5个步骤：

- 定义游标：DECLARE
- 打开游标：OPEN
- 逐行提取游标集中的行：FETCH
- 关闭游标：CLOSE
- 释放游标：DEALLOCATE

■ 打开游标

- 语法为：

  OPEN <*curserName*>

- 游标定义后，如果要使用游标，必须要先打开游标。
  - 打开游标操作表示：
    - ✓ 系统按照游标的定义从数据库中将数据检索出来，放在内存的游标集中(如果内存不够，会放在临时数据库中)
    - ✓ 为游标集指定一个游标(相当于一个指针)，该游标指向游标集中的第1个元组

7.5.1 游标的定义与使用

- **获取当前游标值**：即获取当前游标所指向元组的值，语法是

  FETCH *<curserName>* **INTO** *<@variableName_list>*

  - 执行一次该SQL语句，系统将当前游标所指向的元组属性值放到变量中，然后游标自动下移一个元组。
  - 当前游标所指向元组的每个属性值必须分别用一个变量来接收，即变量个数、数据类型必须与定义游标中的SELECT子句所定义的属性(或表达式)个数、数据类型相一致。
  - 当游标移至尾部，不可再读取游标，必须关闭游标，然后重新打开游标。
  - 通过检查全局变量*@@FETCH_STATUS*来判断是否已读完游标集中所有行(元组)。*@@FETCH_STATUS*的值有：
    - ✓ 0：FETCH 语句成功，表示已经从游标集中获取了元组值
    - ✓ -1：FETCH 语句失败或此行不在结果集中
    - ✓ -2：被提取的行不存在

7.5.1 游标的定义与使用

- 关闭游标：游标不使用了，必须关闭，其语法为：

  CLOSE <*curserName*>

- 释放游标(集)所占用的空间：关闭游标并没有释放游标所占用的内存和外存空间，必须释放游标，其语法为：

  DEALLOCATE <*curserName*>

需要经历5个步骤：

- 定义游标：DECLARE
- 打开游标：OPEN
- 逐行提取游标集中的行：FETCH
- 关闭游标：CLOSE
- 释放游标：DEALLOCATE

7.5.1 游标的定义与使用

- [例7.48] 创建一个游标，逐行显示选修了《计算机原理》课程的学生姓名、相应成绩和该课程的平均分。
- 分析：

① 选修《计算机原理》课程的同学可能不止一个，需要使用游标查询选修该门课程的学生姓名和相应的选课成绩。

定义游标为：

```
DECLARE myCur CURSOR FOR
    SELECT studentName, score, termNo
    FROM Student a, Course b, Score c
    WHERE a.studentNo=c.studentNo AND b.courseNo=c.courseNo
        AND courseName='计算机原理'
    ORDER BY studentName
```

7.5.1 游标的定义与使用

② 要获得该课程的平均分，必须首先计算选课人数和总分

- 声明计数器和累加器变量@*countScore*、@*sumScore*，赋初值为0

```
DECLARE @countScore smallint, @sumScore int
SET @countScore=0
SET @sumScore=0
```

③ 声明3个变量@*sName*、@*score*和@*termNo*，用于接收游标集中当前游标中的学生姓名、选课成绩和选课学期

```
DECLARE @sName varchar(20), @score tinyint, @termNo char(3)
```

④ 由于FETCH命令每次仅从游标集中提取一条记录，必须通过一个循环来重复提取，直到游标集中的全部记录被提取

- 全局变量 @@*FETCH_STATUS*用于判断是否正确地从游标集中提取到了记录；
- @@*FETCH_STATUS*=0表示已经正确提取到了游标记录；
- 循环语句为：

```
WHILE ( @@FETCH_STATUS = 0 )
```

7.5.1 游标的定义与使用

⑤ 在循环体内：

- 首先，显示提取到的学生姓名、选课成绩和选课学期，使用语句：

```
PRINT  convert(char(10), @sName) + convert(char(10), @score) +
       convert(char(10), @termNo)
```

- 其次，计数器@countScore进行计数，并将提取到的成绩累加到变量@sumScore中。语句为：

```
SET  @sumScore = @sumScore + @score          -- 计算总分
SET  @countScore = @countScore + 1           -- 计算选课人数
```

- 提取当前游标所指向元组，并下移(即使其指向游标集中下一元组)：

```
FETCH  myCur  INTO  @sName, @score, @termNo
```

- 重复⑤，直到全部游标记录处理完毕，退出循环。

⑥ 处理完全部游标记录后：

- 关闭和释放游标
- 对计数器@countScore进行判断：如果为0，表示没有同学选修，其平均分为0；否则，平均分等于总分除以选课人数。

7.5.1 游标的定义与使用

⑦ 程序如下：

```
/* 声明变量及赋初值 */
DECLARE @sName varchar(20), @score tinyint, @termNo char(3)
DECLARE @countScore smallint, @sumScore int
SET @countScore=0
SET @sumScore=0
-- 定义游标
DECLARE myCur CURSOR FOR
    SELECT studentName, score, termNo
    FROM Student a, Course b, Score c
    WHERE a.studentNo=c.studentNo AND b.courseNo=c.courseNo
        AND courseName='计算机原理'
    ORDER BY studentName
OPEN myCur     -- 打开游标，游标指向游标集(查询结果集)的第1个元组
PRINT convert(char(10), '学生姓名')+convert(char(10), '课程成绩')+convert(char(10), '选课学期')
PRINT replicate('-', 30)    -- 输出表头信息
```

7.5.1 游标的定义与使用

```
--获取当前游标的值(即第1个元组值)放到变量@sName、@score和@termNo中
FETCH myCur INTO @sName, @score, @termNo   --获取第1个元组值, 游标下移
WHILE ( @@FETCH_STATUS = 0 ) -- 循环处理游标集中的每一个元组
BEGIN
    -- 显示变量@sName、@score和@termNo中的值
    PRINT convert(char(10), @sName) + convert(char(10), @score) + convert(char(10), @termNo)
    SET @sumScore = @sumScore + @score          -- 计算总分
    SET @countScore = @countScore + 1           -- 计算选课人数
    FETCH myCur INTO @sName, @score, @termNo   --获取当前游标所指向元组值, 游标下移
END
PRINT replicate('-', 30)    -- 输出表格底线
PRINT '课程平均分'          -- 输出选修《计算机原理》课程的所有学生的平均分
IF @countScore>0
    PRINT @sumScore/@countScore
ELSE            -- 选修人数为0，即没有学生选修《计算机原理》课程
    PRINT 0.00
CLOSE myCur                 -- 关闭游标
DEALLOCATE myCur            -- 释放游标
```

7.5.2 当前游标集的修改与删除

- 游标可以放在触发器和存储过程中使用
- 可以对游标集中的当前元组执行删除和修改操作
- 删除游标集中的当前元组(即游标所指向的元组)

  DELETE FROM *<tableName>*

  WHERE CURRENT OF *<curserName>*

  - 从游标集中删除当前元组后，游标定位于被删除元组的下一行，但还需要用FETCH语句提取该行的值。
- 修改游标集中的当前元组(即游标所指向的元组)

  UPDATE *<tableName>*

  SET *<columnName>* = *<expr>* [, *<columnName>* = *<expr>*...]

  WHERE CURRENT OF *<curserName>*

目　录

SQL数据定义语言

SQL数据更新语言

视　　图

T-SQL语言简介

游　　标

存储过程

触 发 器

7.6 存储过程

- **存储过程**是为了完成特定功能汇集而成的一组命名了的SQL语句集合
  - 该集合编译后存放在数据库中，可根据实际情况重新编译；
  - 存储过程可直接在服务器端运行，也可在客户端远程调用运行，远程调用时存储过程还是在服务器端运行。
- 使用存储过程具有如下优点：
  - **将业务操作封装**
    - 可为复杂的业务操作编写存储过程，放在数据库中；
    - 用户可调用存储过程执行，而业务操作对用户是不可见的；
    - 若存储过程仅修改了执行体，没有修改接口(即调用参数)，则用户程序不需要修改，达到业务封装的效果。
  - **便于事务管理**
    - 事务控制可以用在存储过程中；
    - 用户可依据业务的性质定义事务，并对事务进行相应级别的操作。

7.6 存储过程

- 实现一定程度的安全性保护
  - 存储过程存放在数据库中，且在服务器端运行；
  - 对于不允许用户直接操作的基本表或视图，可通过调用存储过程来间接地访问这些基本表或视图，达到一定程度的安全性；
  - 这种安全性缘于用户对存储过程只有执行权限，没有查看权限；
  - 拥有存储过程的执行权限，自动获取了存储过程中对相应基本表或视图的操作权限；
  - 这些操作权限仅能通过执行存储过程来实现，一旦脱离存储过程，也就失去了相应操作权限。
- **注意**：对存储过程只需授予执行权限，不需授予基本表或视图的操作权限。
- 特别适合统计和查询操作
  - 一般统计和查询，尤其是期末统计，往往涉及数据量大、表多，若在客户端实现，数据流量和网络通信量较大；
  - 很多情况下，管理信息系统的设计者，将复杂的查询和统计用存储过程来实现，免去客户端的大量编程。

7.6 存储过程

- 减少网络通信量
  - 存储过程仅在服务器端执行，客户端只接收结果；
  - 由于存储过程与数据一般在同一个服务器中，可减少大量的网络通信量。
- 使用存储过程前，首先要创建存储过程。可对存储过程进行修改和删除。
- 创建存储过程后，必须对存储过程授予执行EXECUTE的权限，否则该存储过程仅可以供创建者执行。

■ 7.6.1 创建存储过程

■ 7.6.2 执行存储过程

■ 7.6.3 修改和删除存储过程

7.6.1 创建存储过程

■ 语法：

```
CREATE PROCEDURE <procedureName>
    [(<@parameterName> <datatype> [= <defaultValue>] [OUTPUT]
    [, <@parameterName> <datatype> [= <defaultValue>] [OUTPUT] ] ) ]
AS
    <SQL-Statements>
```

● 其中：

- <procedureName>：存储过程的名称，必须符合标识符规则，且在同一个数据库中唯一；
- <@parameterName>：参数名，存储过程可不带参数，形式参数是变量，但实际参数可以是变量、常量和表达式；
- **OUTPUT**：说明该参数是输出参数，被调用者获取使用。缺省时表示是输入参数。

7.6.1 创建存储过程

- 如果存储过程的输出参数取集合值，则该输出参数不在存储过程的参数中定义，而是在存储过程中定义一个临时表来存储该集合值。
  - 临时表的表名前加一个#符号，如*#myTemp*
  - 在存储过程尾部，使用语句：

    SELECT * FROM *#myTemp*

    将结果集合返回给调用者。
  - 存储过程结束后，临时表自动被删除。
- **注意：**
  - 用户定义的存储过程只能在当前数据库中创建；
  - 一个存储过程最大不能超过128MB。若超过128MB，可将超出的部分编写为另一个存储过程，然后在存储过程中调用。

7.6.1 创建存储过程

■ [例7.50] 输入某个同学的学号，统计该同学的平均分。

```
CREATE PROCEDURE proStudentByNo1(@sNo char(7))
AS
    SELECT a.studentNo, studentName, avg(score)
    FROM Student a, Score b
    WHERE a.studentNo=b.studentNo
        AND a.studentNo=@sNo
    GROUP BY a.studentNo, studentName

EXECUTE proStudentByNo1 '1500003'
```

存储过程创建，openGauss下建立

- 通过窗体和命令行均可以建立。
- 其基本格式为：

操作列表>存储过程>CREATE PROCEDURE

CREATE PROCEDURE

功能描述

创建一个新的存储过程。

注意事项

- 如果创建存储过程时参数或返回值带有精度，不进行精度检测。
- 创建存储过程时，存储过程定义中对表对象的操作建议都显示指定模式，否则可能会导致存储过程执行异常。
- 在创建存储过程时，存储过程内部通过SET语句设置current_schema和search_path无效。执行完函数search_path和current_schema与执行函数前的search_path和current_schema保持一致。
- 如果存储过程参数中带有出参，SELECT调用存储过程必须缺省出参，CALL调用存储过程调用非重载函数时必须指定出参，对于重载的package函数，out参数可以缺省，具体信息参见CALL的示例。
- 存储过程指定package属性时支持重载。
- 在创建procedure时，不能在avg函数外面嵌套其他agg函数，或者其他系统函数。

语法格式

```
postgres=# CREATE [ OR REPLACE ] PROCEDURE procedure_name
    [ ( {[ argmode ] [ argname ] argtype [ { DEFAULT | := | = } expression ]}[,...]) ]
    [
       { IMMUTABLE | STABLE | VOLATILE }
       | { SHIPPABLE | NOT SHIPPABLE }
       | {PACKAGE}
       | [ NOT ] LEAKPROOF
       | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
       | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER | AUTHID DEFINER | AUTHID CURRENT_US
ER}
       | COST execution_cost
       | ROWS result_rows
       | SET configuration_parameter { [ TO | = ] value | FROM CURRENT }
    ][ ... ]
```

7.6.1 创建存储过程(openGauss)

- [例7…] 输入某个同学的学号，插入该学生信息

```
create or replace procedure insert_data(IN sno char(7))
is
begin
INSERT INTO student (studentno,studentname,sex,birthday,native,nation,classno)
 VALUES (sno,'李志强','男     ','1999-12-21 00:00:00','北京','汉族','CP1602');
end;
/
```

```
CALL insert_data ( '1500003')
```

7.6.1 创建存储过程(openGauss)

■ [例7.50] 输入某个同学的学号，统计该同学的平均分。

```
CREATE OR REPLACE PROCEDURE proStudentByNo1( IN sNo CHAR(7))
AS
DECLARE temp1 char(7)…
BEGIN
    SELECT a.studentNo, studentName, avg(score) into temp1,temp2,temp3
    FROM Student a, Score b
    WHERE a.studentNo=b.studentNo
        AND a.studentNo= sNo
    GROUP BY a.studentNo, studentName;
END;
```

```
CALL proStudentByNo1( '1500003')
```

7.6.1 创建存储过程

- SQL Server数据库还可以返回一个数据集合
  - 该数据集合在客户端的程序中可以被网格类的对象接收；
  - 可以对其进行逐行处理；
  - 游标中可以嵌套游标。

[例7.51] 输入某同学的学号，使用游标统计该同学的平均分，并返回平均分，同时逐行显示该同学的姓名、选课名称和选课成绩。

```
CREATE PROCEDURE proStudentByNo2(@sNo char(7),
                                 @avg numeric(6, 2) OUTPUT)
AS
BEGIN
    DECLARE @sName varchar(20), @cName varchar(20)
    DECLARE @score tinyint, @sum int, @count tinyint
    SELECT @sum = 0, @count = 0
```

7.6.1 创建存储过程

```
-- 定义、打开、获取游标
DECLARE curScore CURSOR FOR
    SELECT studentName, courseName, score
    FROM Score a, Student b, Course c
    WHERE b.studentNo=@sNo
        AND a.studentNo=b.studentNo AND a.courseNo=c.courseNo
OPEN curScore
FETCH curScore INTO @sName, @cName, @score
WHILE (@@FETCH_STATUS = 0)
BEGIN
    -- 业务处理
    SELECT @sName, @cName, @score   --逐行显示该同学的选课信息
    SET @sum=@sum+@score
    SET @count=@count+1
    FETCH curScore INTO @sName, @cName, @score
END
```

7.6.1 创建存储过程

```
    CLOSE curScore
    DEALLOCATE curScore
    IF @count = 0
       SELECT @avg = 0
    ELSE
       SELECT @avg=@sum/@count
END
```

- 本例使用了SELECT语句来显示变量的值，即

```
SELECT @sName, @cName, @score
```

- 由于存储过程仅在服务器端执行，其显示的内容只在服务器端出现，并不返回给客户端，这样的输出结果是**没有价值的**。
  - 显示内容在调试存储过程时有作用，一旦存储过程调试正确，使用存储过程的修改命令将存储过程中的显示命令删除。

7.6.2 执行存储过程

- 使用存储过程时，必须执行命令EXECUTE
- 语法：

```
EXECUTE  <procedurName>
[ [<@parameterName> =] <expr>,
[<@parameterName> =] <@variableName>  [OUTPUT]
[, [<@parameterName> =] <expr>,
[<@parameterName> =] <@variableName>  [OUTPUT] ] ]
```

  - 注意：EXECUTE的参数必须与对应的PROCEDURE的参数相匹配。

- [例7.53] 执行存储过程proStudentByNo1

```
EXECUTE  proStudentByNo1  '1600001'
--CALL proStudentByNo1 （’1600001’）；--openGauss下
```

7.6.3 修改和删除存储过程

- 修改存储过程
- 语法为：

```
ALTER PROCEDURE <procedureName>
        [ <@parameterName> <datatype> [= <defaultValue>] [OUTPUT]
        [, <@parameterName> <datatype> [= <defaultValue>] [OUTPUT] ] ]
AS
    <SQL-Statements>
```

- **注意**：由于存储过程是在服务器端执行，程序中不需要有输出命令SELECT，由SELECT引出的输出不会在客户端出现。

7.6.3 修改和删除存储过程

■ 删除存储过程

- 语法：

DROP PROCEDURE *<procedureName>*

[例7.57] 删除存储过程proStudentByNo1

```
DROP PROCEDURE proStudentByNo1
```

目　录

SQL数据定义语言

SQL数据更新语言

视　　图

T-SQL语言简介

游　　标

存储过程

触 发 器

7.7　触发器

- 触发器(trigger)是用户定义在关系表上的一类由事件驱动的存储过程，由服务器自动激活。
- 触发器是一种特殊的存储过程，不管什么原因造成的数据变化都能自动响应，对于每条SQL语句，触发器仅执行一次，事务可用于触发器中。
- 事务定义：

  BEGIN TRANSACTION [<*transactionName*>]
  COMMIT TRANSACTION [<*transactionName*>]
  ROLLBACK TRANSACTION [<*transactionName*>]

- 有两个特殊的表用在触发器语句中，不同的数据库管理系统其名称不一样：
  - 在SQL Server中使用deleted表和inserted表；
  - Oracle数据库使用old表和new表。
  - openGuass数据库使用OLD表和NEW表。

7.7 触发器

- 下面以SQL Server为例:注意:
  - deleted表、inserted表的结构与触发器作用的基本表结构完全一致;
  - 当针对触发器作用的基本表(简称作用表)的SQL语句开始执行时，自动产生deleted表、inserted表的结构与内容;
  - 当SQL语句执行完毕， deleted表、inserted表也随即被删除。
- deleted表
  - 存储当DELETE和UPDATE语句执行时所影响的行的拷贝，即在DELETE和UPDATE语句执行前，先将该语句所作用的行转移到deleted表中，即将被删除的元组或修改前的元组值存入deleted表中。
- inserted表
  - 存储当INSERT和UPDATE语句执行时所影响的行的拷贝，即在INSERT和UPDATE语句执行期间，新行被同时加到inserted表和触发器作用的表中。即将被插入的元组或修改后的元组值存入inserted表中，同时更新触发器作用的基本表。

Update是如何工作的？

7.7 触发器-openGauss数据库中

- 下面以openGauss数据库为例:注意:
  - OLD表、NEW表的结构与触发器作用的基本表结构完全一致;
  - 当针对触发器作用的基本表(简称作用表)的SQL语句开始执行时,自动产生OLD表、 NEW表的结构与内容;
  - 当SQL语句执行完毕, OLD表、 NEW表也随即被删除。
- OLD表
  - 存储当DELETE和UPDATE语句执行时所影响的行的拷贝,即在DELETE和UPDATE语句执行前,先将该语句所作用的行转移到OLD表中,即将被删除的元组或修改前的元组值存入OLD表中。
- NEW表
  - 存储当INSERT和UPDATE语句执行时所影响的行的拷贝,即在INSERT和UPDATE语句执行期间,新行被同时加到NEW表和触发器作用的表中。即将被插入的元组或修改后的元组值存入NEW表中,同时更新触发器作用的基本表。

Update是如何工作的?

7.7　触发器

- 实际上，UPDATE命令是删除后紧跟着插入，旧行首先拷贝到deleted表中，新行同时拷贝到inserted表和基本表中。
- 触发器仅在当前数据库中被创建
  - 触发器有3种类型，即插入、删除和修改；
  - 插入、删除或修改也可组合起来作为一种类型的触发器；
  - 查询操作不会产生触发动作，没有查询触发器类型。
- 7.7.1 创建触发器
- 7.7.2 修改和删除触发器

7.7.1 创建触发器

■ 创建触发器的语法：

```
CREATE  TRIGGER  <triggerName>
ON  <tableName>
FOR  { INSERT | UPDATE | DELETE }
AS  <SQL-Statement>   -- 触发动作的执行体，即触发器代码
```

● 其中：

- <*triggerName*>：触发器的名称，在1个数据库中必须唯一；
- <*tableName*>：触发器作用的基本表，该表也称为触发器的目标表；
- { INSERT | UPDATE | DELETE }：触发器事件，触发器的事件可以是插入INSERT、修改UPDATE或删除DELETE事件，也可以是这几个事件的组合。

7.7.1 创建触发器

- **INSERT 类型**的触发器是指：当对指定基本表<*tableName*>执行了插入操作时**系统自动执行**触发器**代码**。
- **UPDATE 类型**的触发器是指：当对指定基本表<*tableName*>执行了修改操作时**系统自动执行**触发器**代码**。
- **DELETE类型**的触发器是指：当对指定基本表<*tableName*>执行了删除操作时**系统自动执行**触发器**代码**。
- <*SQL-Statement*>：触发动作的执行体，即触发器**代码**
  - ✓如果该触发器**代码**执行失败，则激活触发器的事件就会终止，且触发器的目标表<*tableName*>及触发器可能影响的其它表不发生任何变化，即执行事务的**回滚操作**。

7.7.1 创建触发器

- [例7.58] 创建触发器，保证学生表中的性别仅能取男或女。
- 分析：
  - 本例需要使用插入和修改两个类型的触发器，因为可能破坏约束“性别仅能取男或女”的操作是插入和修改操作。
  - 违约条件是：
    - 如果在inserted表中存在有性别取值不为“男”或“女”的记录(由于inserted表保存了修改后的记录，只要对inserted表进行判断即可)，则取消本次操作——取消本次的所有操作。
  - 插入类型的触发器

```
CREATE TRIGGER sexIns    -- 创建插入类型的触发器
ON Student               -- 触发器作用的基本表
FOR INSERT               -- 触发器的类型，即触发该触发器被自动执行的事件
AS
  IF EXISTS (SELECT * FROM inserted WHERE sex NOT IN ('男', '女'))
     ROLLBACK   -- 事务的回滚操作，即终止触发该触发器的插入操作
```

7.7.1 创建触发器

- 修改类型的触发器

```
CREATE TRIGGER sexUpt     -- 创建修改类型的触发器
ON Student
FOR UPDATE
AS
   IF EXISTS (SELECT * FROM inserted WHERE sex NOT IN ('男', '女'))
      ROLLBACK   -- 事务的回滚操作，即终止触发该触发器的修改操作
```

- 该例也可以合并为一个触发器：

```
CREATE TRIGGER sexUptIns
ON Student
FOR INSERT, UPDATE
AS
   IF EXISTS (SELECT * FROM inserted WHERE sex NOT IN ('男', '女'))
      ROLLBACK
```

- 本例的inserted表结构与Student表结构相同。

```
CREATE TRIGGER sexUptIns
ON Student
FOR INSERT, UPDATE
AS
    IF EXISTS ( SELECT * FROM inserted WHERE sex NOT
IN ('男', '女') )
        ROLLBACK
    ELSE
    BEGIN
        insert into class values('1112','计科班','信院
','2019',NULL)
    END
```

openGauss数据库命令

功能描述：创建一个触发器。 触发器将与指定的表或视图关联，并在特定条件下执行指定的函数。

注意事项：当前仅支持在普通行存表上创建触发器，不支持在列存表、临时表、unlogged表等类型表上创建触发器。如果为同一事件定义了多个相同类型的触发器，则按触发器的名称字母顺序触发它们。触发器常用于多表间数据关联同步场景，对SQL执行性能影响较大，不建议在大数据量同步及对性能要求高的场景中使用。创建触发器，一般先创建触发函数，之后建立触发器。

```
CREATE  TABLE classbk (
classno character(6) PRIMARY KEY,
classname character varying(64) NOT NULL,
institute character varying(32) NOT NULL,
grade smallint DEFAULT 0::smallint NOT
NULL,
classnum smallint
);
```

```
CREATE OR REPLACE FUNCTION public.tri_insert_func_class()
 RETURNS trigger
 LANGUAGE plpgsql
 NOT FENCED NOT SHIPPABLE
AS $$ DECLARE BEGIN INSERT INTO
classbk(classno,classname,institute,grade,classnum)
VALUES(NEW.classno, NEW.classname,NEW.institute, NEW.grade,
NEW.classnum);
RETURN NEW;
END
$$
/
```

```
CREATE OR REPLACE FUNCTION
tri_update_func_class() RETURNS TRIGGER
AS
$$
DECLARE BEGIN UPDATE classbk SET classname =
NEW.classname,classnum = NEW.classnum
 WHERE classno=OLD.classno;
RETURN OLD;
END
$$ LANGUAGE PLPGSQL;
```

```
CREATE OR REPLACE FUNCTION
TRI_DELETE_FUNC_class() RETURNS TRIGGER
AS
$$
DECLARE BEGIN DELETE FROM classbk WHERE
classno=OLD.classno;
RETURN OLD;
END
$$ LANGUAGE PLPGSQL;
```

```
CREATE TRIGGER insert_trigger_class BEFORE INSERT ON class
FOR EACH ROW EXECUTE PROCEDURE tri_insert_func_class();

CREATE TRIGGER update_trigger_class AFTER UPDATE ON class
FOR EACH ROW EXECUTE PROCEDURE tri_update_func_class();

CREATE TRIGGER delete_trigger_class BEFORE DELETE ON class
FOR EACH ROW EXECUTE PROCEDURE tri_delete_func_class();
```

7.7.2 修改和删除触发器

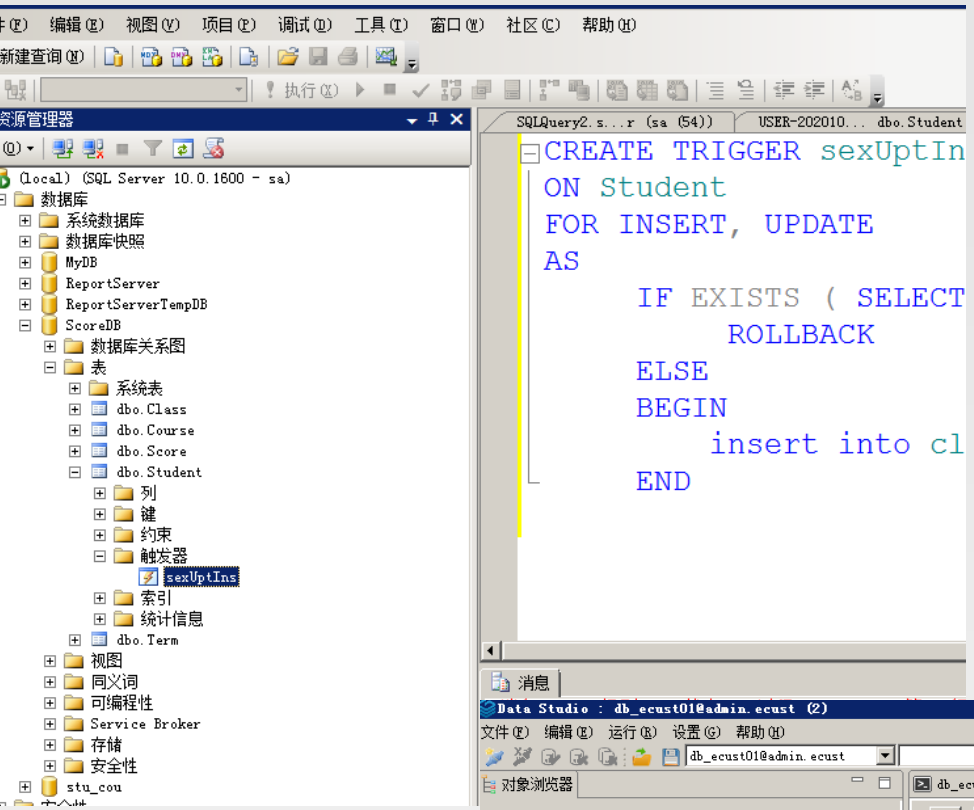
- 触发器不需要时可以删除，删除语法：

```
DROP  TRIGGER  <triggerName>
```

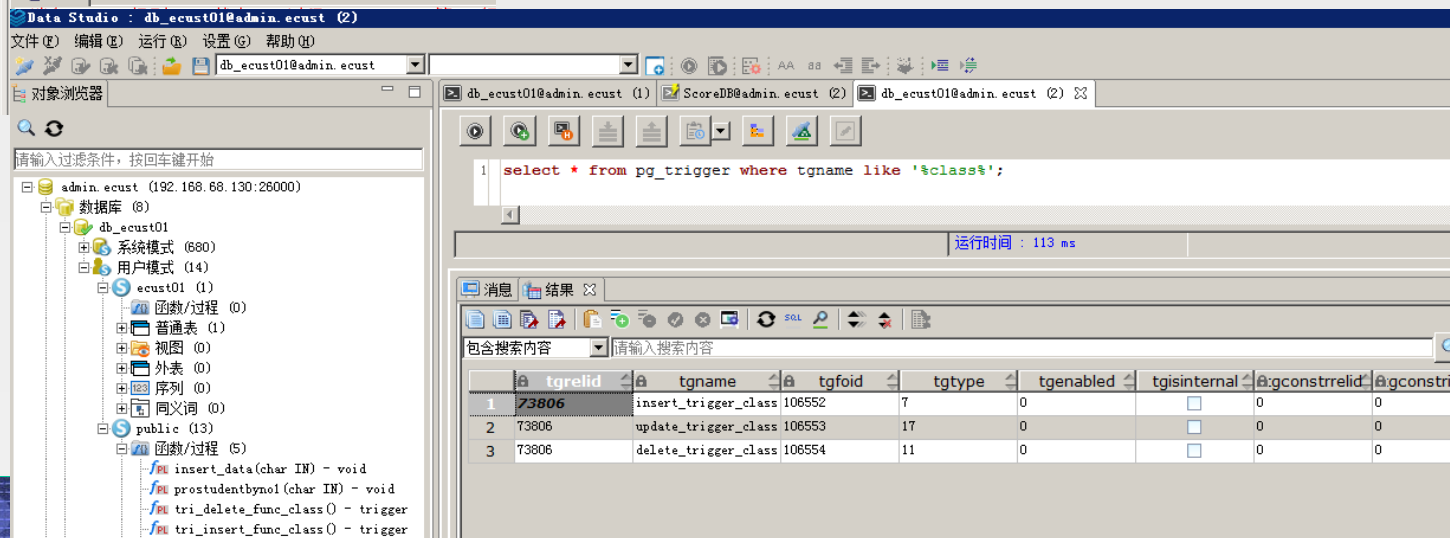
- [例7.62] 删除触发器ClassInsMany。

```
DROP  TRIGGER  ClassInsMany
```

触发器查看


编辑(E)
视图(V)
项目(P)
调试(D)
工具(T)
窗口(W)
社区(C)
帮助(H)
新建查询(N)
执行(X)
资源管理器
(local) (SQL Server 10.0.1600 - sa)
数据库
系统数据库
数据库快照
MyDB
ReportServer
ReportServerTempDB
ScoreDB
数据库关系图
表
系统表
dbo.Class
dbo.Course
dbo.Score
dbo.Student
列
键
约束
触发器
sexUptIns
索引
统计信息
dbo.Term
视图
同义词
可编程性
Service Broker
存储
安全性
stu_cou

SQLQuery2.s...r (sa (54))
USER-202010... dbo.Student
CREATE TRIGGER sexUptIn
ON Student
FOR INSERT, UPDATE
AS
IF EXISTS (SELECT
ROLLBACK
ELSE
BEGIN
insert into cl
END


消息
Data Studio : db_ecust01@admin.ecust (2)
文件(F)
编辑(E)
运行(R)
设置(G)
帮助(H)
对象浏览器
admin.ecust (192.168.68.130:26000)
数据库 (8)
db_ecust01
系统模式 (880)
用户模式 (14)
ecust01 (1)
函数/过程 (0)
普通表 (1)
视图 (0)
外表 (0)
序列 (0)
同义词 (0)
public (13)
函数/过程 (5)
insert_data(char IN) - void
prostudentbyno1(char IN) - void
tri_delete_func_class() - trigger
tri_insert_func_class() - trigger
db_ecust01@admin.ecust (1)
ScoreDB@admin.ecust (2)
db_ecust01@admin.ecust (2)
select * from pg_trigger where tgname like '%class%';
运行时间 : 113 ms
消息
结果
包含搜索内容
tgrelid
tgname
tgfoid
tgtype
tgenabled
tgisinternal
1 73806 insert_trigger_class 106552 7 0 0 0
2 73806 update_trigger_class 106553 17 0 0 0
3 73806 delete_trigger_class 106554 11 0 0 0