南京大学 NANJING UNIVERSITY

奋进行动 团结奋斗 争先进位 坐言起行 应势而动

数 据 管 理 基 础

第8章 数据库编程

（过程化SQL & 存储过程与函数）

智能软件与工程学院

第8章 数据库编程

8.1 嵌入式SQL

8.2 过程化SQL

8.3 存储过程和函数

8.4 ODBC编程

8.5* OLE DB

8.6* JDBC编程

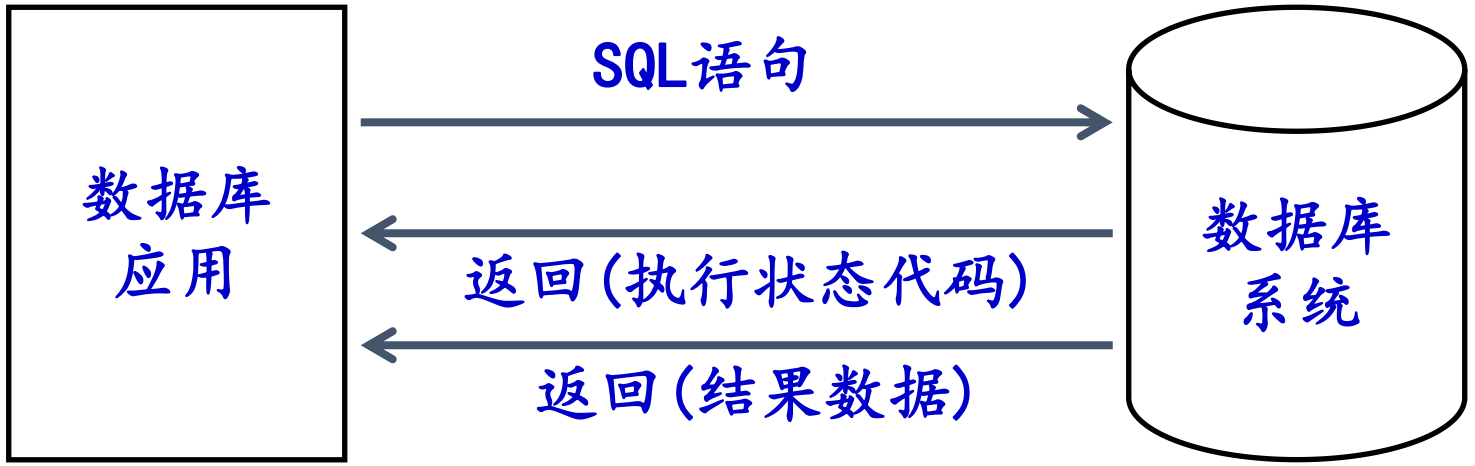
8.7 小结

review：学习导引

❑ SQL语言的三种使用方式

	使用方式	应用场景
交互式SQL	命令行 / 批处理	可独立运行，一般供临时用户操作访问数据库用(即席查询, ad-hoc query)
嵌入式SQL	主语言 + ESQL	数据库应用开发
过程化SQL	SQL编程	• 兼有SQL数据访问和高级程序设计语言的流程控制、简单数值处理功能； • 可在数据库服务器中独立运行； • 常用于编写存储过程、存储函数、触发器。

❑ 数据交换模型



8.2 过程化SQL

- 为了能够在数据库中实现部分应用程序逻辑，需要在交互式SQL语言的基础上扩充部分过程式程序设计语言的成份，构成一个可编程的SQL语言，称为过程化SQL，又被称为过程式SQL、可编程SQL。
- 过程化SQL的作用：
  - 可用于定义触发器、存储过程、存储函数对应的SQL程序块
- 过程化SQL相对于嵌入式SQL的优势
  - 可独立编程，不再需要区分 主变量 与 SQL变量
  - 不需要经历从预编译到编译的处理
  - 在数据库服务器内部实现数据交换与处理
- 最常用的两种过程化SQL： PL/SQL and T-SQL

8.2 过程化SQL - 学习导引

- 过程化SQL的内容
  - 交互式SQL语句
  - SQL中的数据交换，包括游标、诊断及动态SQL
  - 事务控制
  - 传统程序设计语言的主要成份，包括：
    - 变量、常量、参数的定义
    - 流程控制及循环语句：
      - begin.....end块，if...then...else结构，while/for循环结构，case...switch结构，异常处理块
      - return语句
    - 函数/过程调用语句
    - 服务性的函数库、类库等

8.2.1 过程化SQL的块结构

8.2.2 变量和常量的定义

8.2.3 流程控制

8.2.1 过程化SQL的块结构

- 过程化SQL
  - 是对SQL的扩展，增加了过程化语句功能
  - 过程化SQL的基本结构是‘块’（图8.2）
    - 块之间可以互相嵌套
    - 每个块完成一个逻辑操作

过程化SQL块

定义部分

```
DECLARE        /* 定义的变量、常量等只能在该基本块中使用 */
变量、常量、游标、异常等 /* 当基本块执行结束时，定义就不再存在 */
```

执行部分

```
BEGIN
    SQL语句、过程化SQL的流程控制语句
EXCEPTION
    异常处理部分
END;
```

/* 遇到不能继续执行的情况称为‘异常’*/

/* 在出现异常时，采取措施来纠正错误或报错 */

8.2.1 过程化SQL的块结构

8.2.2 变量和常量的定义

8.2.3 流程控制

8.2.2 变量和常量的定义

1. 变量定义

变量名 数据类型 [[NOT NULL] := 初值表达式]

变量名 数据类型 [[NOT NULL] 初值表达式]

2. 常量定义

常量名 数据类型 CONSTANT := 常量表达式

- 常量必须要给一个值，并且该值在存在期间或常量的作用域内不能改变。如果试图修改它，过程化SQL将返回一个异常。

3. 赋值语句

- 变量名称 := 表达式

PL/SQL中的变量定义

Oracle 中常用的基本数据类型

数据类型	描述
Number	数字型，范围10^{-130}~10^{125}
Char	字符型，最大2000个字符
Date	日期型，包括日期、小时、分、秒
Long	大文本
Boolean	逻辑型，取值true,false,null
Varchar2	变长字符型，最大为4000个字符

PL/SQL中的变量定义（续）

- 简单变量定义的一般格式：

<*variablename*> [CONSTANT]<*datatype*> [[NOT NULL] {DEFAULT|:=} <*expression*>];

- 说明
  - 在**PL/SQL**中使用的变量、常量、游标和异常处理的名字都必须先定义后使用。
  - 定义部分是包括在关键字**DECLARE**和**BEGIN**之间的部分，每条语句后用‘;’结束。
  - 每行只能定义一个标识符。
  - 如果加上关键字**CONSTANT**，则表示所定义的标识符为一个常量，必须为它赋初值。
  - 标识符的命名规则与**SQL**的规则基本相同，即每个标识符必须以字母开头，而且不分大小写。
  - 为标识符赋值时，使用赋值符号‘**:=**’，默认值为空。

PL/SQL中的变量定义（续）

- 变量定义的几种方式

1. 基本数据类型
2. %TYPE 类型
3. %RECORD 类型
   ① 自定义记录类型
   ② 基于数据库表结构的记录类型

PL/SQL的变量定义示例（1）

1、使用基本数据类型定义变量

```
sex  BOOLEAN :=TRUE;

today  DATE not null:=sysdate;

age  number(3) not null:=25;
```

PL/SQL的变量定义示例（2）

2、%TYPE类型

- 声明一个变量，使它的类型与某个变量或数据库表中某个列的数据类型一致
- 当我们不知道该变量或列的数据类型时，可以使用 **%TYPE**

```
DECLARE
teacher_name char（5）;
student_name teacher_name%TYPE;
BEGIN
……
END
```

```
DECLARE
no EMP.EMPNO%TYPE;
BEGIN
……
END
```

PL/SQL的变量定义示例（3）

3、记录类型定义的一般格式（自定义记录类型）

```
TYPE <recordtypename> IS RECORD (
    <field> <datatype> [[NOT NULL]{DEFAULT|:=} <expression>]
    [,<field> …]
);
```

- 说明：
  - 标识符 <recordtypename>是定义的记录类型名；
  - 要定义记录型变量，定义方法与前面标量型变量定义一样。
  - 记录类型变量的属性引用方法是‘.’引用。

PL/SQL的变量定义示例（4）

- 声明记录类型和记录类型变量

```
DECLARE
    TYPE student IS RECORD （
        id NUMBER(4) NOT NULL default 0,
                        /* 非空时必须加上缺省值 */
        name CHAR(10) ,
        sex BOOLEAN NOT NULL default true,
        birthdate DATE,
        physics NUMBER(3),
        chemistry NUMBER(3)
    );
    /*下面定义一个student类型的变量 */
    student1  student;
```

PL/SQL的变量定义示例（5）

4、声明一个记录型变量，使它的类型与某个基本表的数据结构一致，可以使用%ROWTYPE的形式定义。

```
DECLARE
    emp_val  emp%ROWTYPE;
/* 定义一个与关系表emp的表结构相同的记录型变量emp_val */
```

- 变量引用方式
  - emp_val.empno　　/* empno是表emp中的一个属性 */
  - emp_val.ename　　/* ename也是表emp中的一个属性 */

PL/SQL中的全局变量与局部变量

- 局部变量
  - 在过程块中定义的变量
  - 只能在块内引用
- 全局变量
  - 在过程块之外定义的变量
  - 使用时需要加':'前缀

```
variable fac number;  /*全局变量*/
DECLARE
    NUM NUMBER(3):=5;  /*局部变量*/
BEGIN
    :fac := 1;
    if NUM>0 then
        for  i  in  1..NUM  loop
            :fac := :fac * i;
        end loop;
    end if;
END;
```

PL/SQL中的游标属性

❑ PL/SQL中的游标属性

- ➢ 游标名%ISOPEN
  - 布尔型。如果游标已经打开，返回TRUE，否则返回FALSE。
- ➢ 游标名%FOUND
  - 布尔型。如果最近一次使用FETCH语句，有返回结果则为TRUE，否则为FALSE;
- ➢ 游标名%NOTFOUND
  - 布尔型。如果最近一次使用FETCH语句，没有返回结果则为TRUE，否则为FALSE;
- ➢ 游标名%ROWCOUNT
  - 数值型。描述的是到目前为止实际从游标工作区取的记录数。

8.2.1 过程化SQL的块结构

8.2.2 变量和常量的定义

8.2.3 流程控制

8.2.3 流程控制

- 过程化SQL功能
  1. 条件控制语句
  2. 循环控制语句
  3. 错误处理

流程控制（续）- 条件控制语句

1. 条件控制语句：IF-THEN，IF-THEN-ELSE 和 嵌套的IF语句

（1）

```
IF condition THEN
    Sequence_of_statements;
END IF;
```

（2）

```
IF condition THEN
    Sequence_of_statements1;
ELSE
    Sequence_of_statements2;
END IF;
```

（3）在THEN和ELSE子句中还可以再包含IF语句，即IF语句可以嵌套

流程控制（续）- 循环控制语句

2. 循环控制语句：LOOP，WHILE-LOOP和FOR-LOOP

(1) 简单的循环语句LOOP

```
LOOP
        Sequence_of_statements;
END LOOP;
```

- 多数数据库服务器的过程化SQL都提供EXIT、BREAK 或 LEAVE等循环结束语句，保证LOOP语句块能够结束循环。

(2) WHILE-LOOP 循环

```
WHILE  condition  LOOP
        Sequence_of_statements;
END LOOP;
```

- 每次执行循环体语句之前，首先对条件进行求值。如果条件为真，则执行循环体内的语句序列；如果条件为假，则跳过循环并把控制传递给下一条语句。

(3) FOR-LOOP 循环

```
FOR  count  IN [REVERSE]  bound1 … bound2  LOOP
        Sequence_of_statements;
END LOOP;
```

3. 错误处理

- 如果过程化SQL在执行时出现异常，则应该让程序在产生异常的语句处停下来，根据异常的类型去执行异常处理语句。
- SQL标准对数据库服务器提供什么样的异常处理做出了建议，要求过程化SQL管理器提供完善的异常处理机制。

❑ 在PL/SQL中，异常处理的一般格式

```
EXCEPTION
    WHEN 异常情况1 [OR 异常情况2…] THEN
        …… ;
    WHEN异常情况3 [OR 异常情况4…] THEN
        …… ;
    WHEN OTHERS THEN
        …… ;
```

数据管理基础

8.3 存储过程和函数

智能软件与工程学院

8.3 存储过程和函数

❑ 存储过程

- 用过程化SQL编写的过程，经编译和优化后存储在数据库服务器中，因而被称为‘存储过程’
- 可以被应用程序或其他存储过程调用

❑ 函数

- 函数的定义与存储过程类似，也是用过程化SQL编写，经编译和优化后存储在数据库服务器中，因而也被称为‘存储函数’或‘自定义函数’。
- 与存储过程不同之处是函数必须定义返回值的类型。

存储过程与函数的区别

- 使用场景不同
  - 存储过程通常用于复杂业务逻辑的实现和完成数据库运维管理任务
  - 函数一般用于计算和数据查询任务
- 结果返回方式不同
  - 存储过程本身没有返回值，只能通过访问存储过程的输出参数或输入/输出参数来获取执行结果；
  - 函数有返回类型定义和返回值，它把执行结果返回给调用者；
  - 因此，存储过程中可以没有返回语句**RETURN**，或者使用不带参数的返回语句；函数必须通过带参数的返回语句结束。
- 执行方式不同
  - 存储过程只能被直接调用执行
  - 函数可以出现在**SQL**语句和表达式中

使用存储过程/函数 的优点

- 存储过程/函数 的优点
  - 运行效率高
    - 已经经过编译和优化，在调用时不需要再进行语法分析和优化工作，运行效率高；
  - 降低客户机和数据库服务器之间的通讯开销
    - 只需要提交过程名和参数值，并返回最终的处理结果
  - 方便用户业务逻辑的实现和管理
    - 由数据库管理系统统一集中管理，既有利于集中管理，又便于维护

❑过程化SQL块类型

❑ 命名块

- 编译后保存在数据库中，可以被反复调用，运行速度较快。
- 过程 和 函数 是命名块

❑ 匿名块

- 每次执行时都要进行编译，它不能被存储到数据库中，也不能在其他过程化SQL块中被调用。

```
DECLARE
    ..... /* 定义声明部分 */
BEGIN
    .....
    EXCEPTION
    ..... /* 异常处理部分部分 */
END
```

/* 执行部分 */

8.3.1 存储过程

8.3.2 函数

8.3.3 过程化SQL中的游标

8.3.1 存储过程

- 存储过程
  - 由过程化SQL语句书写的过程，经编译和优化后存储在数据库服务器中，使用时只要调用即可。

- 存储过程的优点
  - 运行效率高
    - 在创建时进行语法分析和优化工作，提高了运行效率
  - 降低了客户机和服务器之间的通信量
  - 方便实施企业规则
    - 由数据库管理系统统一集中管理，既有利于集中管理，又便于维护

8.3.1 存储过程

- 存储过程的用户接口
  - （1）创建存储过程
  - （2）执行存储过程
  - （3）修改存储过程
  - （4）删除存储过程

存储过程的用户接口-创建存储过程 1

❑ 创建存储过程

CREATE [OR REPLACE] PROCEDURE 过程名([参数1,参数2,...]) /* 存储过程首部 */

AS <过程化SQL块>; /*存储过程体，描述该存储过程的操作 */

- 过程名：存储过程的过程名
- 参数列表：
  - 用参数名来标识调用时给出的参数值，必须指定参数值的数据类型。
  - 参数也可以被定义为 输入参数、输出参数 或 输入/输出参数
    - 参数名 **IN ...** /* 输入参数，也是默认的参数类型 */
    - 参数名 **OUT ...** /* 输出参数，用于存放存储过程的执行结果 */
    - 参数名 **IN OUT ...** /* 输入/输出参数 */
- 过程体：是一个<过程化SQL块>，包括声明部分和可执行部分（包括可执行的过程化SQL语句，以及可能的异常处理部分）

存储过程的用户接口-创建存储过程 2

- [例8.8] 利用存储过程来实现下面的应用：从账户1转指定数额的款项到账户2中。

```
CREATE OR REPLACE PROCEDURE TRANSFER
    ( inAccount  INT,  outAccount  INT,  amount  FLOAT )
    /* 定义存储过程TRANSFER，如果该过程已经存在则覆盖其原有定义 */
    /* 三个输入参数：转入账户、转出账户、转账额度 */
AS
DECLARE              /* 定义变量 */
    totalDepositOut  Float;
    totalDepositIn  Float;
    inAccountnum  INT;
```

存储过程的用户接口-创建存储过程 3

```
BEGIN    /* 过程体执行部分的开始 */
    /* 检查转出账户及其余额 */
    SELECT Total INTO totalDepositOut FROM Accout WHERE accountnum=outAccount;

    IF  totalDepositOut  IS NULL  THEN   /* 转出账户不存在或账户中没有存款 */
        ROLLBACK;              /* 回滚事务 */
        RETURN;
    END IF;

    IF  totalDepositOut < amount  THEN        /*如果账户存款不足*/
        ROLLBACK;         /* 回滚事务 */
        RETURN;
    END IF;
```

存储过程的用户接口-创建存储过程 4

```
    /* 检查转入账户 */
    SELECT Accountnum INTO inAccountnum FROM Account WHERE accountnum=inAccount;

    IF  inAccount  IS NULL  THEN        /* 如果转入账户不存在 */
        ROLLBACK;     /* 回滚事务 */
        RETURN;
    ENDIF;

    /* 修改转出账户余额，减去转出额 */
    UPDATE  Account  SET total = total - amount  WHERE  accountnum = outAccount;
    /* 修改转入账户余额，增加转入额 */
    UPDATE  Account  SET  total = total + amount  WHERE  accountnum = inAccount;

    COMMIT;     /* 提交转账事务 */
END;    /* 过程体执行部分的结束 */
```

存储过程的用户接口-执行存储过程

- 执行存储过程

  CALL/PERFORM PROCEDURE 过程名([参数1,参数2,...]);

  - 使用CALL或者PERFORM等方式激活存储过程的执行
  - 在过程化SQL中，数据库服务器支持在过程体中调用其他存储过程

- [例8.9] 从账户01003815868转10000元到01003813828账户中。

```
CALL PROCEDURE TRANSFER(01003813828,01003815868,10000);
```

存储过程的用户接口-其他

- 修改存储过程：对存储过程重命名或重编译
  - 重命名

    ALTER PROCEDURE 过程名1 RENAME TO 过程名2;
  - 重新编译

    ALTER PROCEDURE 过程名 COMPILE;

- 删除存储过程

  DROP PROCEDURE 过程名;

8.3 存储过程和函数

8.3.1 存储过程

8.3.2 函数

8.3.3 过程化SQL中的游标

8.3.2 函数

- 使用过程化SQL编写的‘函数’，也被称为‘存储函数’或‘自定义函数’

- 函数和存储过程的异同
  - 同：都是持久性存储模块
  - 异：函数必须指定返回的类型

❑ PL/SQL中的函数定义语句格式

```
CREATE [OR REPLACE] FUNCTION  function_name
    [(parameter1 [{IN|OUT|IN OUT}] datatype [{:=|DEFAULT} expression]
    [, parameter2 [{IN|OUT|IN OUT}] datatype [{:=|DEFAULT} expression]
    …])]
RETURN  returntype    /* 定义函数的返回值类型 */
{IS|AS}
    [declarations]
BEGIN
        code
    [EXCEPTION
        exception_handlers]
END
```

8.3.2 函数

- 函数的执行语句格式

  CALL/SELECT 函数名 ([参数1,参数2,...]);

- 修改函数
  - 重命名
    - ALTER FUNCTION 过程名1 RENAME TO 过程名2;
  - 重新编译
    - ALTER FUNCTION 过程名 COMPILE;

存储过程示例

```
/* 定义存储过程：连续输出某个字符 */
CREATE OR REPLACE PROCEDURE printLine
    (width IN INTEGER, chr IN CHAR DEFAULT '-')  IS
BEGIN
    FOR i IN 1..width LOOP
        dbms_output.put(chr);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('');
END printLine;

/* 调用存储过程 */
begin
    printline(40, '*');
end;
```

函数示例

```
/* 定义函数：返回指定职工的总收入 */
CREATE OR REPLACE FUNCTION totalsal(v_empno in emp.empno%type)
     RETURN number   /* 定义函数返回值的数据类型 */
IS
    totalsal1 number;
    begin
        select  sal + comm  into  totalsal1
        from  emp
        where  empno = v_empno;
        return totalsal1;
    end;

/* 在SQL语句中调用函数 */
select empno, totalsal(empno) from emp where totalsal(empno)>300;
```

8.3.1 存储过程

8.3.2 函数

8.3.3 过程化SQL中的游标

8.3.3 过程化SQL中的游标

- 过程化SQL中的游标，与嵌入式SQL中的游标及其使用方式一样，只是在语句格式上稍有差别。

- [例8.11] 带参数的游标

```
CREATE OR REPLACE PROCEDURE proc_cursor() AS
DECLARE
    cno  CHAR(3);
    cname  CHAR(8);

    CURSOR  mycursor(leaderno  CHAR(3))  FOR  /* 带参数游标mycursor */
        SELECT lno, lname FROM leader WHERE lno=leaderno;
        /* 检索leader表中属性lno的值 等于 游标参数leaderno值的记录 */
```

[例8.11] 带参数的游标（续）

```
BEGIN
    OPEN  mycursor( 'L01' );   /* 使用参数 L01 打开游标 */
    FETCH  mycursor  INTO  cno, cname;   /* 获取lno='L01'的游标元组 */
    INSERT INTO temp(lno,lname) VALUES(cno,cname);
          /* 将游标元组插入临时表temp中 */
    CLOSE  mycursor;  /* 关闭游标 */

    OPEN  mycursor( 'L02' );   /* 使用新的参数 L02 重新打开游标 */
    FETCH  mycursor  INTO  cno, cname;   /* 获取lno='L02'的新的游标元组 */
    INSERT INTO temp(lno,lname) VALUES(cno,cname);
          /* 将新的游标元组插入临时表temp中 */
    CLOSE  mycursor;   /* 关闭游标 */
END;
```

用过程化SQL创建触发器（1）

- [例5.21]当对表SC的Grade列进行修改时，若分数增加了10%则将此次操作记录到表SC_U(Sno,Cno,Oldgrade,Newgrade)中去。其中：Oldgrade是修改前的分数，Newgrade是修改后的分数。（P170页）

```
CREATE TRIGGER  SC_T
AFTER UPDATE OF Grade ON SC
REFERENCING
                    OLD  row   AS  OldTuple,
                    NEW  row  AS  NewTuple
FOR EACH ROW
WHEN (NewTuple.Grade >= 1.1*OldTuple.Grade)
INSERT INTO SC_U(Sno,Cno,OldGrade,NewGrade)
VALUES(OldTuple.Sno, OldTuple.Cno, OldTuple.Grade, NewTuple.Grade);
```

用过程化SQL创建触发器（2）

- [例5.23] 定义一个BEFORE行级触发器，为教师表Teacher定义完整性规则“教授的工资不得低于4000元，如果低于4000元，自动改为4000元”。（P171页）

```
CREATE TRIGGER Insert_Or_Update_Sal
BEFORE INSERT OR UPDATE ON Teacher
    /*触发事件是插入或更新操作*/
REFERENCING NEW ROW AS newTuple
FOR EACH ROW       /*行级触发器*/
BEGIN    /*定义触发动作体，是一个PL/SQL过程块*/
    IF (newTuple.Job = '教授') AND (newTuple.Sal < 4000)
    THEN  newTuple.Sal := 4000;
    END IF;
END;
```

用过程化SQL创建触发器（3）

```
CREATE OR REPLACE TRIGGER logemp
BEFORE  INSERT OR UPDATE OR DELETE ON EMP
FOR  EACH  ROW
DECLARE
    statementtype CHAR(20);
BEGIN
    IF INSERTING THEN
        statementtype:='INSERT TRIGGER!';
    ELSIF UPDATING THEN
        statementtype:='UPDATE TRIGGER!';
    ELSE
        statementtype:='DELETE TRIGGER!';
    END IF;
    DBMS_OUTPUT.PUT_LINE(statementtype);
END;
```

小结

- 过程化SQL
- 存储过程 & 函数
- 如何用过程化SQL编写存储过程和函数

复习思考题

设有一个如下表所示的学生选课数据库，其中：带下划线属性是码；同一门课同一个学生只能有一条选课记录；成绩采用百分制。

关系	属性集	关系模式
学生	学号, 学生姓名, 就读院系, 年级	Student (sno, sname, dept, grade)
课程	课程号, 课程名, 开课院系, 学分	Course (cno, cname, dept, credit)
教师	教师编号, 教师姓名, 工作院系	Teacher (tno, tname, dept)
选课	学号, 课程号, 授课教师编号, 成绩, 授课年份	Scorelist (sno, cno, tno, score, year)

6. 请创建一个存储过程：根据输入的课程号，将该门课程的所有选课成绩从百分制转换成等级制，并插入到临时表temp(sno, cno, scgrade)中去。要求如下：
   ① 有一个输入参数‘课程号’和 两个输出参数‘课程选修人数’、成绩等级为A的学生人数；
   ② 等级转换规则：>=90为A，80~89为B，70~79为C，60~69为D，低于60为E
   ③ 如果成绩为空值，则等级也是空值

7. 请创建一个函数：根据输入的学号，统计该学生的平均成绩（不包括成绩为空值的选课记录）

习 言 道 | 我 们 都 是 奋 斗 者