

第5章 PL/SQL



课程知识结构

Chp.1 数据库系统概述

Chp.2 数据库系统体系结构

Chp.3 关系数据模型

Chp.9 完整性

Chp.4 SQL

Chp.6 关系数据库模式设计

Chp.10 安全性

Chp.5 PL/SQL

Chp.7 数据库设计

Chp.11 事务与恢复

Chp.8 数据库应用系统设计

Chp.12 并发控制

Chp.13 高级主题

本章主要内容

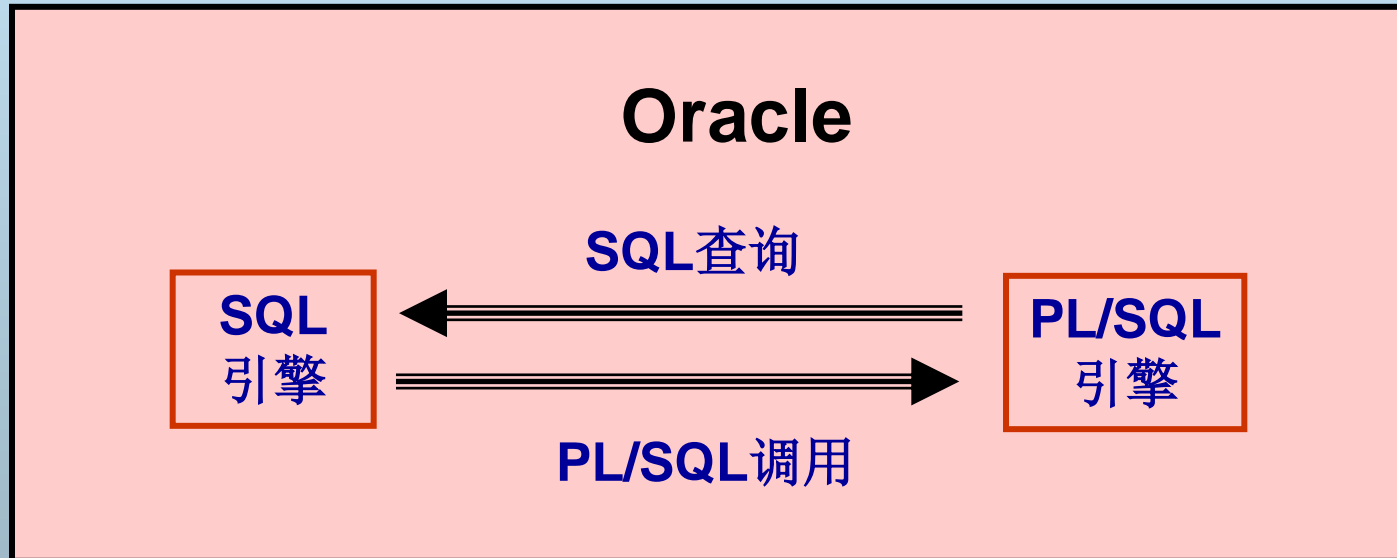
- **PL/SQL vs. SQL**
- **PL/SQL编程**
- **存储过程 (Stored Procedure)**
- **触发器 (Trigger)**

一、PL/SQL与SQL

- **SQL是描述性语言，PL/SQL是过程性语言**
- **PL/SQL是Oracle对SQL的一个扩展，是一种过程化的程序设计语言**
 - **SQL本身并不能建立数据库应用程序**
 - **PL/SQL是包含SQL的一种过程性语言，它不仅支持SQL，还支持一些过程性语言特性**
- **其它商用DBMS一般也都提供类似的扩展**
 - **Microsoft T-SQL**
 - **Informix E-SQL**

一、PL/SQL与SQL

- 二者均可以在Oracle数据库中运行，可以相互调用



一、PL/SQL与SQL

从A账号转帐
100到B账号

Total 5

客户端多次计算
多次网络传输

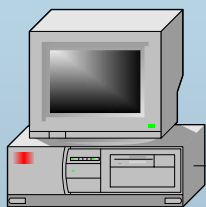
SQL

客户机

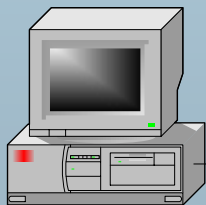
- 1、通过SQL查询账号A是否存在
- 2、通过SQL查询账号B是否存在
- 3、查询账号A上的余额
- 4、修改A上的余额
- 5、修改B上的余额

SQL

Result



客户端



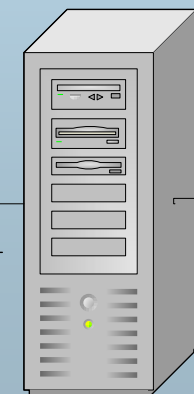
客户端

SQL

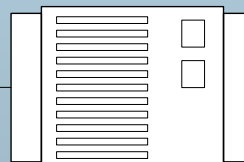
网络

SQL

结果



Oracle DBMS



Oracle数据库文件

一、PL/SQL与SQL

从A账号转帐
100到B账号

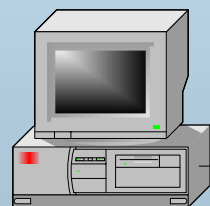
Total 1

客户端更少计算
更少网络传输

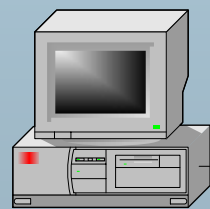
客户机

执行一个预先编写好并存储
在服务器上的**PL/SQL**程序
完成转帐

调用**PL/SQL**程序



客户端



客户端

SQL

Result

SQL

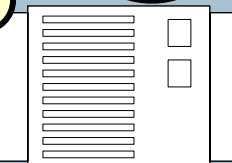
结果

网络

SQL

Oracle DBMS

任务都加在
这儿了



Oracle数据库文件

一、PL/SQL与SQL

■ 在程序中使用SQL

- 客户机计算任务多
- 网络传输重
- 服务器计算任务少

} 胖客户机、瘦服务器

■ 在程序中使用PL/SQL

- 可以完成一些SQL不能完成的复杂计算
- 客户机计算任务少
- 服务器计算任务加重
- 网络传输少

} 瘦客户机、胖服务器

二、PL/SQL程序的结构

■ **DECLARE** —— 变量声明，必须放在首部

.....

BEGIN

.....

EXCEPTION

.....

END

} 程序体

1、一个例子：返回001学生的姓名

```
DECLARE
```

```
    name varchar2(20);
```

```
BEGIN
```

```
    Select sname Into name From Student Where s#='001';
```

```
    DBMS_OUTPUT.PUT_LINE('学号001的学生姓名是: ' || sname)
```

```
EXCEPTION
```

```
    When NO_DATA_FOUND Then
```

```
        DBMS_OUTPUT.PUT_LINE('学号为001的学生不存在' );
```

```
    When others Then
```

```
        DBMS_OUTPUT.PUT_LINE('发生了其它错误' );
```

```
END;
```

2、PL/SQL的编程

■ 赋值

- :=

- **Select Into <变量> From.....**

■ 注释

- --

■ 运行

- /

三、变量声明

- 必须放在**DECLARE**段

- **<变量名> <类型>**

- 如果编写过程或函数，则过程首部将取代**Declare**段

- **Create Procedure**

-**

- IS**

- 变量声明**

- BEGIN**

1、变量声明例子

■ 例1：一般性声明

- **Declare**

- sno Number(3);**

- name Varchar2(10);**

■ 例2：声明为正在使用的表中的某个列类型

- **Declare**

- sno student.s#%TYPE;**

- name student.sname%TYPE;**

- 可以保证代码与数据库结构之间的独立性

1、变量声明例子

■ 例3：声明一个记录类型

- 记录类型是由多个相关变量构成的一个单元

◆ **TYPE <记录名> IS RECORD (**
 变量1 类型1,
 变量2 类型2,

)

- 通过定义记录类型，可以保存表中的记录

1、变量声明例子

■ 例3：声明记录类型

- 通过 “记录类型变量.成员名” 访问内部成员

```
DECLARE
```

```
    TYPE stu IS RECORD(  
        s# varchar2(10),  
        name varchar2(10),  
        age number  
    );
```

val stu; —— 声明了一个记录类型的变量 **val**

```
BEGIN
```

```
    Select * Into val From Student Where s#='001';
```

```
    DBMS_OUTPUT.PUT_LINE('学号001的学生姓名是: ' || val.name)
```

```
END;
```

1、变量声明例子

■ 例4：声明为一个表的行类型

- **Declare**
stu Student%ROWTYPE;
- **Stu**被声明为与**student**表相匹配的记录类型，**student**表的列自动称为**stu**的成员
 - ◆ **Stu.s#**
 - ◆ **Stu.sname**
 - ◆ **Stu.age**

四、分支控制语句

■ **IF <表达式> THEN**

<语句>

ELSEIF <表达式> THEN

<语句>

.....

ELSE

<语句>

END IF;

```
IF x=5 THEN
```

```
    DBMS_OUTPUT.PUT_LINE('x=5');
```

```
END IF;
```

五、循环语句

- **WHILE**循环
- **FOR**循环
- **LOOP**循环

1、WHILE循环

- **While** <表达式>
Loop
 <语句>
End Loop;
- 当表达式为真时执行语句

Declare

x Number;
total Number;

Begin

x:=1;

total:=0;

While x<=100 Loop

total:=total+x;

x:=x+1;

End Loop

END ;

2、FOR循环

- **For** <计数变量> **In** [**Reverse**] <开始值>..**<结束值>**
Loop
 <语句>
End Loop;
- 循环体每执行一次
 计数变量自动加**1**
- 若有**Reverse**，则每
 次循环计数变量自动
 减**1**

Declare

x Number;
total Number;

Begin

total:=0;

For x In 1..100 Loop

total:=total+x;

End Loop

END ;

3、LOOP循环

- **Loop**
 <语句>
End Loop
- 无内部控制结构的循环结构，循环执行其中的<语句>
- 必须在循环体中显式地结束循环
- **Exit**和**Exit When**两种方式退出循环

```
Declare
    x Number:=1;
    total Number:=0;
Begin
    Loop
        If x<=100 Then
            total:=total+x;
            x:=x+1;
        Else
            Exit;
        End If
    End Loop
END ;
```

```
Declare
    x Number:=1;
    total Number:=0;
Begin
    Loop
        total:=total+x;
        x:=x+1;
        Exit When x>100
    End Loop
END ;
```

六、处理异常

■ Exception

- 可以捕捉程序运行中出现的错误或意外情况，并加以处理

■ Exception

When <错误名1> **OR** <错误名2>..... **Then**
 <错误处理语句>

When <错误名1> **OR** <错误名2>..... **Then**
 <错误处理语句>

When Others Then
 <错误处理语句>

1、例子：返回001学生的姓名

DECLARE

name varchar2(20);

BEGIN

Select sname Into name From Student Where s#='001';

DBMS_OUTPUT.PUT_LINE('学号001的学生姓名是: ' || sname)

EXCEPTION

When NO_DATA_FOUND Then

DBMS_OUTPUT.PUT_LINE('学号为001的学生不存在');

When others Then

DBMS_OUTPUT.PUT_LINE('发生了其它错误');

END;



系统定义的标准异常

2、系统定义的标准异常（共20种）

■ NO_DATA_FOUND

- 执行Select...Into语句却没有找到匹配记录

■ TOO_MANY_ROWS

- 执行Select...Into语句却返回了多行记录

■ VALUE_ERROR

- 变量赋值错误，可能是类型不匹配，或者是值太大或太长

■ ZERO_DIVIDE

- 除零错误

■ TIMEOUT_ON_RESOURCE

- 资源等待超时

3、人工生成异常

■ 直接生成异常

- **Raise_application_error**

■ 声明并触发一个自定义异常

- **Raise**
- **Raise_application_error**

(1) 直接生成异常

Raise_application_error (自定义错误号-20000~-20999, 错误信息)

--插入一个新学生 '001'

DECLARE

sno varchar2(20);

BEGIN

Select s# Into sno From Student Where s#='001';

If SQL%FOUND Then

raise_application_error(-20001, '学生已存在');

Else

Insert Into student(s#) values('001');

End If

END;

(2) 声明并触发一个自定义异常

--插入一个新学生 '001'

DECLARE

sno varchar2(20);

exp Exception; --声明一个Exception变量

BEGIN

Select s# Into sno From Student Where s#='001';

If SQL%FOUND Then

raise exp; --生成一个异常

Else

Insert Into student(s#) values('001');

End If

EXCEPTION

When exp Then

raise_application_error(-20001, '学生已存在');

When others Then

DBMS_OUTPUT.PUT_LINE('发生了其它错误 ');

END;

Raise_application_error返回一个异常
，而**PUT_LINE**直接输出异常信息

4、两个系统参数

DECLARE

.....

BEGIN

.....

EXCEPTION

When NO_DATA_FOUND Then

DBMS_OUTPUT.PUT_LINE('数据不存在 ');

When others Then

DBMS_OUTPUT.PUT_LINE

(' 错误号: ' || SQLCODE || '错误描述: ' || SQLERRM);

END;

七、游标

- 游标概念
- 游标操作
- 游标属性
- 使用游标**FOR**循环
- 操纵游标的当前行

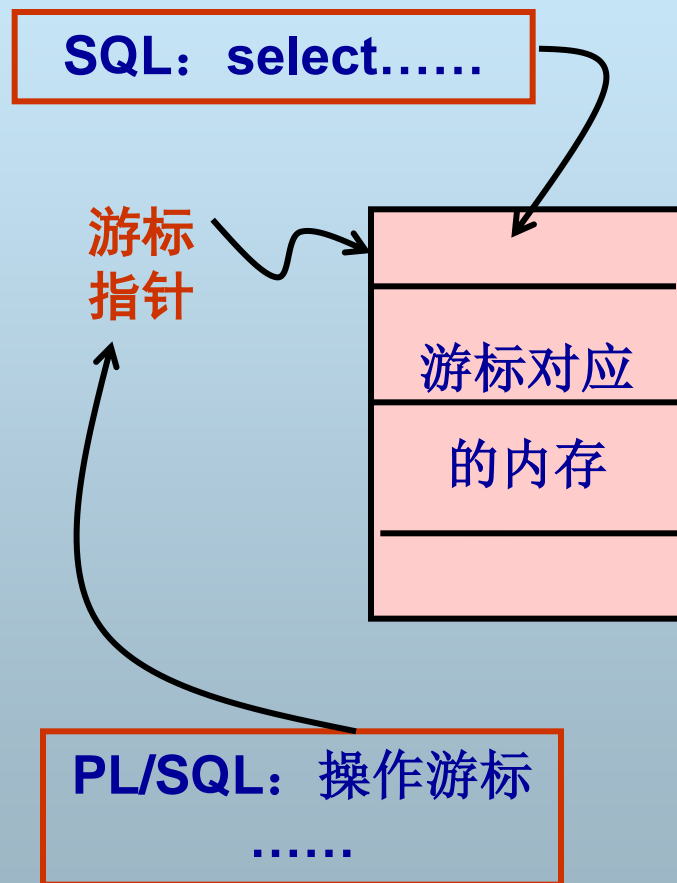
1、游标概念

■ 动机

- **PL/SQL**是过程性语言，每次只能处理单个记录；而**SQL**是描述性语言，每次可以处理多行记录。**PL/SQL**支持**SQL**，问题：
- **PL/SQL**如何支持多行记录的操作？

■ 解决方法：游标


- 游标是客户机或数据库服务器上开辟的一块内存，用于存放**SQL**返回的结果
- 游标可以协调**PL/SQL**与**SQL**之间的数据处理矛盾
- **PL/SQL**中可以通过游标来过程化存取**SQL**返回的结果



2、游标操作

- 声明一个游标
- 打开游标
- 读取游标中的记录
- 关闭游标

一般的操作顺序



(1) 声明游标

■ Declare

Cursor <名称> IS <Select语句>

- 声明中的SQL语句在声明时并不执行，只是给出了游标对应的数据定义

```
--声明一个游标，用于存放所有学生记录
```

```
DECLARE
```

```
Cursor cs_stu IS select * from student;
```


(2) 打开游标

■ Open <游标名>

- 打开游标时，**SELECT**语句被执行，其结果放入了游标中

```
--声明一个游标，用于存放所有学生记录
```

```
DECLARE
```

```
        Cursor cs_stu IS select * from student;
```

```
BEGIN
```

```
        Open cs_stu;
```

(3) 读取游标中的记录

■ Fetch <游标名> Into <变量表或记录 变量>

- 打开游标后，游标指向了第一条记录
- **Fetch**后指向下一条记录
- 若要读取游标中的数据，一般需使用一个循环

--返回所有学生记录

DECLARE

Cursor cs_stu IS select * from student;

stu student%ROWTYPE;

BEGIN

Open cs_stu;

Fetch cs_stu Into stu;

While cs_stu%FOUND Loop

DBMS_OUTPUT.PUT_LINE(...);

Fetch cs_stu Into stu;

End Loop;

.....

(4) 关闭游标

■ Close <游标名>

--返回所有学生记录的学号和姓名

DECLARE

Cursor cs_stu IS select s#,sname from student;

1

sno student.s#%TYPE;

name student.sname%TYPE;

BEGIN

Open cs_stu;

2

Loop

Fetch cs_stu Into sno,name;

Exit When cs_stu%NOTFOUND;

DBMS_OUTPUT.PUT_LINE(...);

End Loop;

Close cs_stu;

4

END;

3

3、游标属性

■ PL/SQL使用游标属性判断游标的当前状态

● **Cursor%FOUND**

◆ 布尔型，当前FETCH返回一行时为真

● **Cursor%NOTFOUND**

◆ 布尔型，当前FETCH没有返回一行时为真

● **Cursor%ISOPEN**

◆ 布尔型，若游标已经打开则为真

● **Cursor%ROWCOUNT**

◆ 数值型，显示目前为止已从游标中取出的记录数

4、使用游标FOR循环

- 由于游标总是使用循环处理，因此可以简化这种处理过程
- 游标**FOR**循环：简化了游标操作
 - 自动声明一个与游标中的数据记录类型一致的变量，并自动打开游标，读取游标，并在读完后自动关闭游标

```
--返回所有学生记录  
DECLARE  
  
  Cursor cs_s IS select * from student;  
  
BEGIN  
  
  For s IN cs_s Loop  
  
    DBMS_OUTPUT.PUT_LINE(...);  
  
  End Loop;  
  
END;
```

自动声明s、打开游标、
读取数据并关闭游标

5、带参游标

■ Declare

Cursor <名称> (参数表) IS <Select语句>

--返回给定年龄的学生记录

DECLARE

Cursor cs_s(val Number(3)) IS select * from student where age=val;

BEGIN

For s IN cs_s(20) Loop

DBMS_OUTPUT.PUT_LINE(...);

End Loop;

END;

八、PL/SQL的输入输出

■ 输出：使用DBMS_OUTPUT包

- ◆ **PUT_LINE (.....)** : 输出并换行
- ◆ **PUT (.....)** : 输出但不换行
- ◆ **NEWLINE**: 生成一个新行

■ 输入

--返回给定年龄的学生记录

DECLARE

Cursor cs_s IS select * from student where age=&val;

BEGIN

For s IN cs_s Loop

DBMS_OUTPUT.PUT_LINE(...);

End Loop;

END;

运行时将提示输入
val的值

九、存储过程和函数

■ 匿名PL/SQL块

- 以**DECLARE/BEGIN**开始，每次运行都要编译，程序在数据库中不保存

■ 命名PL/SQL块

- 可以存储在数据库中，可以随时运行，也可以被**SQL**或其它程序调用
- 存储过程、函数、触发器、包

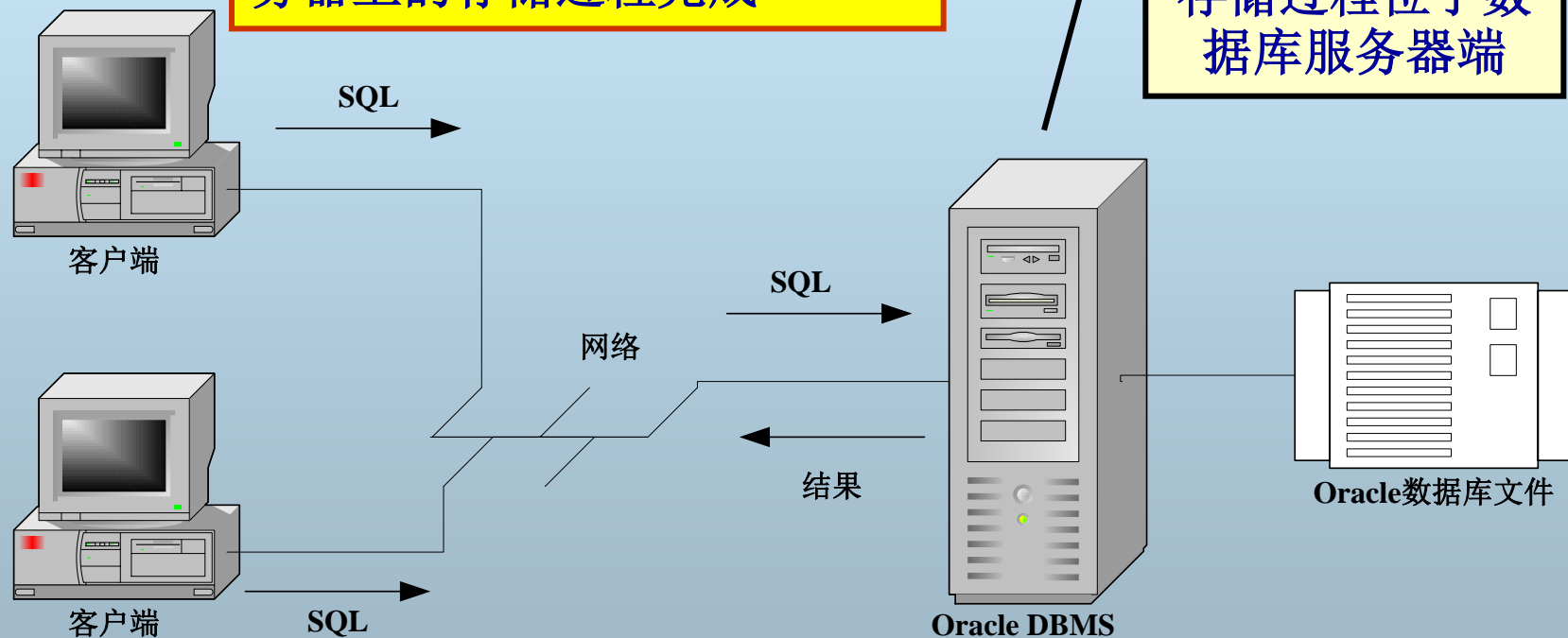
1、存储过程概念

- 一类存储在数据库中的PL/SQL程序，可以通过过程名调用

1、存储过程概念

Client/Server计算模式：

客户端将业务处理任务交给服务器上的存储过程完成



```
Create or Replace Procedure AddStudent(  
    v_s# IN varchar2, v_sname IN varchar2, v_age IN number)  
AS  
BEGIN  
    Insert Into student(s#,sname,age) Values(v_s#,v_sname,v_age);  
END;
```

可在SQL PLUS中直接调用运行

```
SQL>Execute AddStudent('001','John',21);
```

可在其它PL/SQL程序中使用

```
BEGIN  
.....  
AddStudent(s,n,a); --s, n, a是变量  
.....  
END;
```

2、存储过程定义

■ **Create or Replace Procedure** <名称>(
 参数表
)
 AS | IS
 变量定义
 BEGIN
 PL/SQL代码
 EXCEPTION
 错误处理
 END;

} 与匿名PL/SQL程序
 格式相同

3、参数定义

- 参数名 **IN** | **OUT** | **IN OUT** 数据类型 [:=默认值]
 - 例 name **IN** varchar2, result **OUT** number
- **IN**参数
 - 输入参数，在程序中不能修改
- **OUT**参数
 - 输出参数，在程序中只能对其赋值
- **IN OUT**
 - 既可作为**IN**参数使用，也可作为**OUT**参数使用

4、删除存储过程

■ **Drop Procedure** <存储过程名>

5、函数

- 具有返回值的存储过程
- **Create or Replace Function** <名称>(
 参数表
)
 RETURN <类型>
 AS | **IS**
 变量定义
 BEGIN
 PL/SQL代码
 EXCEPTION
 错误处理
 END;

例：创建返回一个系的学生总人数的函数

```
Create or Replace Function StudentCount(DeptNo IN varchar2)
Return Number
AS
    v_count Number:=0;
BEGIN
    select count(s#) Into v_count From Student where dept=deptno;
    return v_count;
END;
```

使用函数

```
Declare
    a number:=0;
BEGIN
    a:=StudentCount('cs');
    DBMS_OUTPUT.PUT_LINE('cs系的学生人数为' || a);
END;
```


6、使用过程和函数的注意点

■ 参数类型

- 不能指定长度（与变量定义不同）
 - ◆ 变量：name varchar2(20) 必须指定长度
 - ◆ 参数：name IN varchar2 不能指定长度
- 可使用%TYPE

■ 参数传递

- 按位置：sam(1,2,3,4)
- 命名传递：sam(b=>2,a=>1,d=>4,c=>3)
 - ◆ 与位置无关

■ IN,OUT,IN OUT参数的使用

十、触发器 (Trigger)

- 触发器的概念
- 触发器的种类
- 触发器的创建
- 触发器的触发顺序
- :old和:new系统变量

1、触发器的概念

- 与特定表关联的存储过程。当在该表上执行DML操作时，可以自动触发该存储过程执行相应的操作
 - 触发操作：Update、Insert、Delete
 - 通过触发器可以定制数据库对应用程序文件的反应
 - 一个触发器只能属于一个表，一个表可有多多个触发器

2、触发器概念示例

- Student (s#, sname, age, status)
- Sc(s#, c#, score)
- 规定当学生有3门课不及格时，将该学生的status标记为‘不合格’
- 通过SC上的触发器实现：当在SC中插入或更新记录时，自动检查是否有学生满足不合格条件

S#	Sname	age	status
01	aaa	22	合格
02	bbb	21	合格

S#	C #	Score
01	c1	55
01	c2	50
02	c1	80
01	c3	55

插入该记录后01学生的
status自动改为‘不合格’

3、Oracle触发器的种类

按执行先后

- **先触发器 (Before Trigger)** : 在DML语句执行之前触发
- **后触发器 (After Trigger)** : 在DML语句执行之后触发

按执行方式

- **行级触发器**: 对由触发的DML语句所导致变更的每一行触发一次 (一个DML语句可能触发多次)
- **语句级触发器**: 一个DML语句触发一次 (只触发一次)

4、触发器的创建

■ Create Or Replace Trigger <名称>

[Before | After | Delete | Insert | Update [Of <列名表>] [OR

定义触发动作

Before | After | Delete | Insert | Update [Of <列名表>] ...]

先触发器还是后触发器

ON <表名>

[For Each Row]
<PL/SQL块>

是否定义为行级触发器

END;

- 注意：没有参数。因为触发器是自动执行的，不能向它传参数

5、系统变量:old和:new

- 对于行级触发器，系统变量:old和:new存储每一行的更新前值 (:old) 和更新后值 (:new)
- 可以在触发器程序中需要时访问它们

操作 变量	Insert	Update	Delete
:old的值	空	原记录	删除的记录
:new的值	新记录	新记录	空

6、触发器例子：自动更新学生状态

Create or Replace Trigger **SetStatus**

After Insert Or Update Of score on SC

For Each Row

Declare

a Number:=0;

行级触发器

Begin

Why?

Select count(*) into a From SC where s#:= :new. s# and score<60;

If a>=3 Then

Update student Set status='不合格' Where s#= :new. s#;

Else

Update student Set status='合格' Where s#= :new. s#;

End If

End;

7、触发器例子：自动统计学生人数

- 学校表：University(U#,uname, s_count)
- 学生表：Student(s#,sname,age)

Create or Replace Trigger **TotalStudent**

After Insert Or Delete On Student

Declare

a Number:=0;

Begin

Select count(*) into a From Student ;

Update University Set s_count=a;

End;

语句级触发器

8、触发器的触发顺序

1. 语句级先触发器
2. 对于受语句影响的每一行
 - ① 行级先触发器
 - ② 执行语句
 - ③ 行级后触发器
3. 语句级后触发器

本章小结

- **PL/SQL与SQL**
- **PL/SQL程序要素**
- **游标**
- **存储过程**
- **触发器**