十五、PLSQL编程

# PLSQL的介绍

## PLSQL的概念及优缺点

PLSQL(procedural language)是Oracle在标准语言上的扩展，pl/sql不仅允许嵌入sql语言，还可以定义变量和常量，允许使用条件语句和循环语句，允许使用异常处理各种错误，这使得Oracle的功能变得更加强大，这也是Oracle的优势所在；

* 过程，函数，触发器是用pl/sql编写的；
* 过程，函数，触发器存储在Oracle中；
* pl/sql是非常强大的数据库过程语言；
* 过程、函数可以在java程序中调用；

使用pl/sql的优点：

* 提高应用程序的运行性能；例如：函数、过程在调用之前已经被编译成直接执行的代码，只需要传入参数即可执行数据操作；而传统的SQL语句需要先编译再执行，效率较低；
* 模块化设计思想；例如：可以将一些特定功能的数据操作封装成一个存储过程，比如分页过程，订单过程，转账过程；降低了耦合，修改数据库操作，不会影响上层调用；
* 降低网络传输流量；例如：传统数据库直接传输SQL语句，而存储过程传输参数即可；
* 提高安全性；例如：函数、过程将数据的具体操作封装到内部，调用函数、过程的程序并不知道其内部实现；

使用pl/sql的缺点：

* 移植性不好：函数、存储过程针对于特定的数据库，更换数据库就需要重新写；

**写程序不只在于运行，更在于优化与效率；**

## PLSQL开发工具

* sqlpuls

1. 编写一个存储过程，该过程可以向一个表中添加数据；

1)登录scott用户，创建一个表mytest；

|  |
| --- |
| CREATE TABLE mytest(  name VARCHAR2(30),  PASSWORD VARCHAR2(30)  ); |

2)创建存储过程；

|  |
| --- |
| CREATE OR REPLACE PROCEDURE my\_first\_pro IS  BEGIN  --执行部分  INSERT INTO mytest('FUZHENGIXN','FUZHENGIXN');  END; |
|  |

3)数据库报出以上存储过程带有错误，查看错误可以用命令：**SHOW ERROR;**

|  |
| --- |
| SHOW ERROR; |
|  |

4)修改insert语句：

|  |
| --- |
| CREATE OR REPLACE PROCEDURE my\_first\_pro IS  BEGIN  --执行部分  INSERT INTO mytest VALUES('FUZHENGIXN','FUZHENGIXN');  END; |
|  |

5)调用存储过程

方法1：**EXEC 过程名(参数1，参数2，…);**

**或EXECUTE过程名(参数1，参数2，…);**

方法2：**CALL 过程名(参数1，参数2，…);**

|  |
| --- |
| EXEC my\_first\_pro; |
|  |

* pl/sql developer

1. 编写一个存储过程，该过程可以删除一个表中的一条记录；

|  |
| --- |
| CREATE OR REPLACE PROCEDURE my\_second\_pro IS  BEGIN  *--执行部分*  DELETE FROM mytest WHERE name='FUZHENGXIN';  END;  / |
|  |

# PLSQL的基础

## 分类

使用pl/sql主要可以编写以下几种代码：

|  |
| --- |
|  |

## 编写规范

注释：

* 单行注释：--
* 多行注释：/\*\*/

标志符命名规范：

* 定义变量，以v\_开头；
* 定义常量，以c\_开头；
* 定义游标，以\_cursor作尾缀；
* 定义异常，以e\_开头；

## 块

块（block）是pl\sql的基本程序单元，编写pl\sql实际是编写pl\sql块。要实现简单的功能，可以编写一个块，若要实现复杂功能，则可能需要多个块嵌套；

块主要有三部分组成：定义部分，执行部分，异常处理部分；结构如下：

|  |
| --- |
| DECLARE  */\*定义部分：非必须，用来定义变量、常量、游标、异常、复杂数据类型\*/*  BEGIN  */\*执行部分：必须，执行pl/sql语句和sql语句 \*/*  EXCEPTION  */\*异常处理部分：非必须，处理运行时出现的各种异常\*/*  END; |

1. 一个简单的块，输出一行语句；

|  |
| --- |
| SET SERVEROUTPUT ON *--打开输出选项，从而显示输出信息*  BEGIN  dbms\_output.put\_line('Hello World!');  END;  /  *--注：dbm\_output是Oracle提供的一个包，而put\_line即为其中的一个过程；* |
|  |

1. 从控制台接受变量作为emp表的empno，根据empno查询员工姓名和薪水并打印；

|  |
| --- |
| DECLARE  v\_name VARCHAR2(5); *--定义一个长度为5的字符串变量*  v\_sal NUMBER(5);  BEGIN  SELECT ename,sal INTO v\_name,v\_sal  FROM emp  WHERE empno=&no;  dbms\_output.put\_line('雇员姓名：' || v\_name);  dbms\_output.put\_line('雇员薪水：' || v\_sal);  END;  /  *--注：&表示要从控制台接收参数，no表示变量的名称但不能直接打印* |
|  |

1. 添加异常处理，对上例中的员工号进行异常处理，若输入不存在的编号，则处理异常；

|  |
| --- |
|  |

添加异常处理：

|  |
| --- |
| DECLARE  v\_name VARCHAR2(5); *--定义一个长度为5的字符串变量*  v\_sal NUMBER(5);  BEGIN  SELECT ename,sal INTO v\_name,v\_sal  FROM emp  WHERE empno=&no;  dbms\_output.put\_line('雇员姓名：' || v\_name);  dbms\_output.put\_line('雇员薪水：' || v\_sal);  EXCEPTION*--异常处理：当数据为空时，Oracle会定义一个no\_data\_found的异常*  WHEN no\_data\_found THEN  dbms\_output.put\_line('员工号输入有误，请重新输入！');  END;  /  *--注：&表示要从控制台接收参数，no表示变量的名称* |
|  |

**注：**

* 在声明部分，不能对变量进行赋值；如需要重新赋值，则需要在执行部分进行；
* 在pl/sql中，不能对输入参数进行重新赋值；

## 控制结构

### 条件分支语句

|  |
| --- |
| **IF THEN END IF;** |
| **IF THEN ELSE END IF;** |
| **IF THEN ELSIF THEN [ELSE] END FF;** |

1. 编写一个过程，可输入一个雇员名，如果该雇员工资低于2000，就给该雇员增加10%的工资；

|  |
| --- |
|  |

1. IF THEN ELSE END;

|  |
| --- |
|  |
|  |

1. 多重条件分支

|  |
| --- |
|  |
|  |

### 循环语句

|  |  |
| --- | --- |
| LOOP WHEN END LOOP | 至少执行一次 |
| WHILE LOOP END LOOP |  |
| FOR IN REVERSE LOOP END LOOP |  |

1. 简单循环

|  |
| --- |
|  |

1. WHILE循环

|  |
| --- |
|  |
|  |

1. FOR循环

### GOTO 语句

|  |
| --- |
|  |
|  |

### NULL语句

|  |
| --- |
|  |
|  |

# 存储过程

过程用于执行特定的操作，当建立过程时，既可以指定输入参数（in），也可以指定输出参数（out）。通过在过程中使用输入参数，可以将数据传递到执行部分；通过使用输出参数，可以将执行部分的数据传递到应用环境。在 sql plus中可以使用CREATE PROCEDURE语句来建立过程。

**注：**在过程中，传的参数只需要定义类型即可，不需要指定类型的大小；

1. 编写一个过程，可以输入雇员名，新工资 可修改雇员的工资；

|  |
| --- |
| CREATE PROCEDURE CHANGE\_SAL(emp\_name VARCHAR2 , new\_sal Number) IS  BEGIN  UPDATE emp SET sal=new\_sal  WHERE ename=emp\_name;  END CHANGE\_SAL;  / |
| EXEC CHANGE\_SAL('SCOTT',4500); |
|  |

## 调用存储过程的方法：

|  |
| --- |
| **EXEC 过程名** |
| **EXECUTE 过程名** |
| **CALL 过程名** |

## 在java中调用存储过程

存储过程的主要目的是为了给程序调用；

1. Java中调用存储过程修改emp中员工工资；

|  |
| --- |
| **try** {  //加载驱动类  Class.*forName*("oracle.jdbc.driver.OracleDriver");    //获取Connection对象  String url="jdbc:oracle:thin:@FS\_INDUSTRY-PC:1521:ORCL";  java.sql.Connection conn=java.sql.DriverManager.*getConnection*(url,"scott","tiger");    //获取statement  java.sql.CallableStatement cs=conn.prepareCall("{CALL CHANGE\_SAL(?,?)}");    //设置参数  cs.setString(1, "SMITH");  cs.setInt(2, 4000);    //执行过程  cs.execute();    //关闭连接  cs.close();  conn.close();  } **catch** (SQLException e) {    e.printStackTrace();  } **catch** (ClassNotFoundException e) {    e.printStackTrace();  } |
|  |

注：方法二

|  |
| --- |
|  |

1. 无返回值的存储过程

|  |
| --- |
|  |
|  |
|  |
|  |

1. 有返回值的存储过程

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

1. 返回结果集的存储过程

|  |
| --- |
|  |
|  |
|  |

# 函数

函数用于返回特定的数据，当建立函数时，在函数的头部必须包含RETURN子句，而在函数体内必须包含RETRUN子句所返回的数据，用CREATE FUNCTION语句来建立函数。

语法：

|  |
| --- |
|  |

1. 建立函数，输入雇员姓名，返回该雇员的年薪；

|  |
| --- |
|  |
|  |

**注**：上面操作显示编译出错，执行**SHOW ERROR命令**,显示错误，原来是多写了一个NUMBER;

|  |
| --- |
| CREATE OR REPLACE FUNCTION CAL\_ANNUAL\_SAL(empname VARCHAR2)  *--声明返回值类型*  RETURN NUMBER IS  *--定义变量存储年薪值*  annual\_sal NUMBER(7,2);  BEGIN *--执行部分*  SELECT (sal+NVL(comm,0))\*12 INTO annual\_sal  FROM emp  WHERE ename=empname;  RETURN annual\_sal;  END CAL\_ANNUAL\_SAL;  / |
|  |

函数的调用方法：

|  |
| --- |
| *--创建变量*  VAR 变量名 变量类型;  CALL 函数名(参数) INTO:变量名;*--此处冒号与变量名之间不能有空格*  PRINT 变量名; |
| 若在JAVA中则可以使用以下语句：  SELECT 函数名(参数) FROM dual;  然后用ResultSet的getInt()等方法获取返回值； |

总结函数与过程的区别：

* 函数必须有返回值，过程可以没有；
* 函数和过程在JAVA中调用的方式不同；函数用“select”语句调用，过程用CallableStatement对象调用；

# 包

包用于在逻辑上组成过程和函数，它由包规范和包体两部分组成。包规范只包含了过程和函数的声明，但没有过程和函数的实现代码，包体用于实现包规范中的过程和函数。可以使用CREATE PACKAGE来创建包规范，使用CREATE PACKAGE BODY来创建包体。

1. 创建包，封装上例中的过程和函数；

|  |
| --- |
| *--包规范*  CREATE PACKAGE EMP\_TOOL\_PACKAGE IS  *--函数和过程的声明*  PROCEDURE CHANGE\_SAL(emp\_name VARCHAR2 , new\_sal Number);  FUNCTION CAL\_ANNUAL\_SAL(empname VARCHAR2) RETURN NUMBER;  END EMP\_TOOL\_PACKAGE;  / |
| *--包体*  CREATE PACKAGE BODY EMP\_TOOL\_PACKAGE IS  *--函数和过程的实现*  PROCEDURE CHANGE\_SAL(emp\_name VARCHAR2 , new\_sal Number) IS  BEGIN  UPDATE emp SET sal=new\_sal  WHERE ename=emp\_name;  END CHANGE\_SAL;    FUNCTION CAL\_ANNUAL\_SAL(empname VARCHAR2)  *--声明返回值类型*  RETURN NUMBER IS  *--定义变量存储年薪值*  annual\_sal NUMBER(7,2);  BEGIN *--执行部分*  SELECT (sal+NVL(comm,0))\*12 INTO annual\_sal  FROM emp  WHERE ename=empname;  RETURN annual\_sal;  END CAL\_ANNUAL\_SAL;  END EMP\_TOOL\_PACKAGE;  / |

**注：**包规范和包体要分别进行编译，不能放在一起执行编译；

当调用包中的过程或函数时，需要在过程或函数名前跟上包名；

例如：

|  |
| --- |
|  |

# 触发器

触发器是指隐含执行的存储过程。当定义触发器时，必须要指定触发的事件和触发的操作，常用的触发事件包括INSERT,UPDATE,DELETE语句，而触发操作实际就是一个PL/SQL块。可以使用create trigger来创建触发器。触发器非常有用，可以用来维护数据库的安全性和一致性。

考虑问题：

|  |
| --- |
| 当用户登录到Oracle时，用一张表记录登录到Oracle的用户名和时间等信息； |
| 禁止用户在星期天对某一张表进行删除操作； |
| 在用户删除一张表的时候，自动把删除的记录备份到另外一张表中； |

以上问题可以使用触发器来解决；

触发器可以看做是一个通过一定条件自动执行的存储过程；触发器仍然是一种数据对象；

## 触发器的分类

* dml触发器：insert,update,delete;
* ddl触发器：create,drop;
* 系统触发器：与系统相关的触发器；如用户登录，退出，启动数据库，关闭数据库；

## 创建触发器

基本语法：

|  |
| --- |
| create [or replace] trigger trigger\_name  {before | after}*--表示在事件之前还是之后触发*  {insert | delete | update [of column1 [,column2...]]}  on [shcema.]table\_name *--指定要被监控的数据表*  [for each row]*--指定为行级触发器，若未指定，则为表级触发器；*  [when condition]  begin  trigger\_body;  end; |

1. 在emp中添加一条记录时，提示“添加了一条记录”；

|  |
| --- |
| create or replace trigger emp\_insert\_trigger  after insert on scott.emp  begin  dbms\_output.put\_line('添加了一条记录！');  end; |
|  |
|  |

1. 在emp表中执行多行语句的update操作；提示“修改了一条数据”；

若按照之前的写法，则修改了15条记录后，仍然只显示一条提示；这是因为，在默认情况下，触发器为表级触发器，即对整个表的事件进行响应；若需要对每行数据都有一个响应，则需要通过 for each row 来指定；

|  |
| --- |
|  |
| create or replace trigger emp\_update\_trigger  after update on scott.emp  for each row  begin  dbms\_output.put\_line('修改了一条记录！');  end; |
|  |

1. 禁止工作人员在休息日删除emp表中的信息；提示：通过before来控制；

|  |
| --- |
| create or replace trigger emp\_delete\_trigger  before delete on scott.emp  for each row  begin  if to\_char(sysdate,'day') in('星期六','星期日','星期一')  then  *--通过抛出自定义错误来阻止删除操作；*  *--注：自定义错误码范围是-20000到-20999*  raise\_application\_error(-20001,'对不起，休息日期间不能删除emp表中记录！');  end if;  end; |
|  |

**注**：raise\_application\_error是一个过程，由oracle提供，包含两个参数，第一个参数为错误编码，范围在（-20999，-20000）；第二个参数是错误提示信息；

谓词：

若在一个触发器中包含多个触发事件（例如同时包含insert update delete）时，需要通过谓词来区分；

|  |  |
| --- | --- |
| inserting | 插入操作 |
| updating | 更新操作 |
| deleting | 删除操作 |

1. 禁止工作人员在休息日添加，修改，删除数据，并且分别对每个操作输出提示；

|  |
| --- |
| create or replace trigger emp\_change\_forbiden  before  insert or update or delete  on scott.emp  for each row  begin  if to\_char(sysdate,'day') in('星期六','星期日','星期一')  then  case  when inserting then  dbms\_output.put\_line('请不要插入数据');  raise\_application\_error(-20001,'请不要插入数据');  when updating then  dbms\_output.put\_line('请不要修改数据');  raise\_application\_error(-20002,'请不要修改数据');  when deleting then  dbms\_output.put\_line('请不要删除数据');  raise\_application\_error(-20003,'请不要删除数据');  end case;  end if;  end; |

## 使用:old和:new

若在触发器中，需要使用操作前后数据的值，则需要用到old和new

|  |  |
| --- | --- |
| **:old** | 访问操作完成之后记录的值； |
| **:new** | 访问操作完成之前记录的值； |

**注：**:old和:new必须在行级触发器中使用；

1. 在修改emp表雇员薪水时，显示雇员工资修改前和修改后的值；且确保修改后的工资不低于原来的工资；

|  |
| --- |
| create or replace trigger emp\_output\_info  before update  on scott.emp  for each row  begin  *--判断修改后薪水是否大于修改前薪水*  if :new.sal<:old.sal  then  dbms\_output.put\_line('修改后的工资不能低于原来的工资！');  raise\_application\_error(-20001,'修改后的工资不能低于原来的工资！');  else  dbms\_output.put\_line('工号：'||:new.empno||' | 原来的工资是：'||:old.sal||' | 现在的工资是：'||:new.sal);  end if;  end; |
|  |
|  |

1. 编写一个触发器，保证当用户在删除一张表（emp）记录的时候，自动把删除的记录备份到另外一张表（emp\_bak）中；

|  |
| --- |
| *--编写一个触发器，保证当用户在删除一张表（emp）记录的时候，自动把删除的记录备份到另外一张表（emp\_bak）中；*  create or replace trigger backup\_on\_delete  before delete  on scott.emp  for each row  begin  insert into scott.emp\_bak(empno,ename,job,mgr,hiredate,sal,comm,deptno)  values(:old.empno,:old.ename,:old.job,:old.mgr,:old.hiredate,:old.sal,:old.comm,:old.deptno);  end;  / |

1. 编写一个触发器，控制员工的工资不低于原来的工资，也不能高于原来工资的20%；

|  |
| --- |
| *--编写一个触发器，控制员工的工资不低于原来的工资，也不能高于原来工资的20%；*  create or replace trigger sal\_control  before update  on scott.emp  for each row  begin  if (:new.sal<:old.sal) or (:new.sal>:old.sal\*1.2)  then  dbms\_output.put\_line(:new.empno||'更新员工工资超出范围！');  raise\_application\_error(-20001,'更新员工工资超出范围！');  end if;  end; |

1. 编写一个触发器，阻止把小于18岁的用户增加到数据库emp表中；

|  |
| --- |
| *--编写一个触发器，阻止把小于18岁的用户增加到数据库emp表中；*  create or replace trigger age\_limition  before insert  on scott.emp\_bak  for each row  begin  if months\_between(sysdate,:new.birthday)<18\*12  then  raise\_application\_error(-20001,'不可以雇佣小于18岁的员工！');  end if;  end; |

## 系统触发器

上面所述的触发器都是dml触发器；都是根据数据的更新操作而触发的事件来执行相应操作，而系统触发器是指根据一些Oracle系统本身的一些事件而响应的触发器；例如用户的登录，数据库的打开与关闭等。需要系统管理员权限。

|  |
| --- |
|  |

* 建立登录和退出触发器

基本语法：

|  |
| --- |
| create or replace trigger 触发器名  after[before] logon[logoff]  on database  begin  *--执行语句。。。*  end; |

1. 创建一张表，记录用户登录和退出数据库的信息，包括用户名、登录时间、退出时间、ip地址；（注：需要用到管理员权限）

|  |
| --- |
| *--创建存放登录信息的数据表*  create table login\_info(  username varchar2(100),  logon\_time timestamp,  logoff\_time timestamp,  ip varchar2(50)  );  *--创建登录触发器*  create or replace trigger logon\_info  after logon  on database  begin  insert into login\_info(username,logon\_time,ip)  values(ora\_login\_user,systimestamp,ora\_client\_ip\_address);  end;  *--创建退出登录触发器*  create or replace trigger logoff\_info  before logoff  on database  begin  insert into login\_info(username,logoff\_time,ip)  values(ora\_login\_user,systimestamp,ora\_client\_ip\_address);  end; |
|  |

**注：**systimestamp使用的是UTC时间；

## ddl触发器

create,alter,drop等ddl语句触发的事件；响应这些事件的触发器称为ddl触发器；这种触发器也需要管理员权限；

基本语法：

|  |
| --- |
| create or replace tirgger 触发器名  after dll *–注：ddl没有before关键字；*  on 方案名.schema  begin  *--执行语句。。。*  end; |

1. 创建一张表，记录用户的ddl操作信息，包括操作类型，用户名，操作时间；

|  |
| --- |
| *--创建ddl操作信息统计触发器*  create or replace trigger ddl\_info  after ddl  on scott.schema  begin  insert into ddl\_operation\_info(operation\_type,user\_name,operation\_time)  values(ora\_sysevent,ora\_login\_user,systimestamp);  end; |
|  |

## 管理触发器

|  |
| --- |
|  |

# 定义和使用变量

编写PL/SQL时，可以定义变量和常量；PL/SQL中包含的变量和常量有：

* 标量类型(scalar)
* 复合类型(composite)
* 参照类型(reference)
* LOB(large object)

## 标量

语法：

|  |
| --- |
| 变量名称 [CONSTANT] 数据类型 [NOT NULL] [:= | DEFAULT] [变量|表达式|常量|函数] |

1. 定义一个变长字符串

|  |
| --- |
| v\_ename VARCHAR2(20); |

1. 定义一个小数，范围-9999.99 ~ 9999.99；

|  |
| --- |
| v\_sal NUMBER(6,2); |

1. 定义一个小数并赋初值5.4；

|  |
| --- |
| v\_sal NUMBER(6,2):=5.4; |

1. 定义一个日期数据类型

|  |
| --- |
| v\_hiredate DATE; |

1. 定义一个布尔变量，不能为空，初始值为FALSE；

|  |
| --- |
| v\_valid BOOLEAN NOT NULL DEFAULT FALSE; |

使用标量

1. 输入员工号，显示员工姓名，工资，个人所得税（假设税率为0.03）；

|  |
| --- |
| DECLARE  *--税率*  c\_tax\_rate CONSTANT NUMBER(3,2):=0.03;  *--员工姓名*  v\_ename VARCHAR2(10);  *--工资*  v\_sal NUMBER(7,2);  *--税*  v\_sal\_tax NUMBER(7,2);    BEGIN  *--执行*  SELECT ename,sal INTO v\_ename,v\_sal  FROM emp  WHERE empno=&no;    *--计算所得税*  v\_sal\_tax:=v\_sal\*c\_tax\_rate;  *--输出*  DBMS\_OUTPUT.PUT\_LINE('员工姓名：'||v\_ename||' | 员工工资：'||v\_sal||' | 个税：'||v\_sal\_tax);    END;  / |
|  |

为了方便操作，可以使用“表名.字段名%TYPE”来标识数据类型，在定义变量时，不必显示指定数据类型，只要告诉编译器，该变量的类型和哪个字段的类型和大小一致就行；故上例可以修改如下：

1. %TYPE的使用；

|  |
| --- |
| DECLARE  *--税率*  c\_tax\_rate CONSTANT NUMBER(3,2):=0.03;  *--员工姓名*  v\_ename emp.ename%TYPE;  *--工资*  v\_sal emp.sal%TYPE;  *--税*  v\_sal\_tax NUMBER(7,2);    BEGIN  *--执行*  SELECT ename,sal INTO v\_ename,v\_sal  FROM emp  WHERE empno=&no;    *--计算所得税*  v\_sal\_tax:=v\_sal\*c\_tax\_rate;  *--输出*  DBMS\_OUTPUT.PUT\_LINE('员工姓名：'||v\_ename||' | 员工工资：'||v\_sal||' | 个税：'||v\_sal\_tax);    END;  / |

## 复合变量(composite)

复合变量是用于存放多个值的变量；包括：

* PL/SQL记录
* PL/SQL 表
* 嵌套表
* Varray

1. PL/SQL 记录

类似于高级语言中的结构体，当引用记录成员时，必须要加记录变量作为前缀（记录变量.记录成员）

1. 定义一个记录，存放emp表中的ename,sal,job字段；

|  |
| --- |
| *--定义记录*  DECLARE    *--定义一个记录类型，存放emp表的ename,sal,job字段*  TYPE emp\_record\_type IS RECORD(empname emp.ename%TYPE,sal emp.sal%TYPE,job emp.job%TYPE);  *--定义一个记录类型的变量*  emp\_record emp\_record\_type;    BEGIN  *--查询emp表中信息，放入记录变量中*  SELECT ename,sal,job INTO emp\_record  FROM emp  WHERE empno=7788;    *--打印记录*  DBMS\_OUTPUT.PUT\_LINE('员工姓名：'||emp\_record.empname || '工资：' || emp\_record.sal || '工作:' || emp\_record.job);  END;  / |
|  |

1. PL/SQL 表

相当于高级语言中的数组，但是高级语言中数组的下标不能为负数，而PL/SQL中是可以为负数的，且表的元素下标没有限制。

1. 定义一个PL/SQL表，存放emp的ename字段；

|  |
| --- |
| *--定义PL/SQL表*  DECLARE    *--定义一个PL/SQL表类型，存放emp表的ename字段*  TYPE emp\_table\_type IS TABLE OF emp.ename%TYPE INDEX BY BINARY\_INTEGER;*--此处表示下标按整数排序,可以为负数*  *--定义一个PL/SQL表变量*  emp\_table emp\_table\_type;  BEGIN    SELECT ename INTO emp\_table(0)  FROM emp  WHERE empno=7788;  DBMS\_OUTPUT.PUT\_LINE('员工姓名：' || emp\_table(0));  END;  / |
|  |

**注：上例中的PL/SQL表中一个索引元素只能存放一条记录，如果有多条，则会报错；**

|  |
| --- |
| *--定义PL/SQL表*  DECLARE    *--定义一个PL/SQL表类型，存放emp表的ename字段*  TYPE emp\_table\_type IS TABLE OF emp.ename%TYPE INDEX BY BINARY\_INTEGER;*--此处表示下标按整数排序,可以为负数*  *--定义一个PL/SQL表变量*  emp\_table emp\_table\_type;  BEGIN    SELECT ename INTO emp\_table(0)  FROM emp;  DBMS\_OUTPUT.PUT\_LINE('员工姓名：' || emp\_table(0));  END;  / |
|  |

## 参照变量（重要）

参照变量是指用来存放数值指针的变量，通过使用参照变量，可以使得应用程序共享相同对象，从而降低占用空间。在编写PL/SQL程序时，可以使用游标变量(REF CURSOR)和对象类型变量(REF OBJ\_TYPE)两种参照变量类型。

1. 游标变量

使用游标时，当定义游标时不需要指定相应的SELECT语句，但当使用时（即OPEN时）需要指定SELECT语句，这样游标就与SELECT语句结合了；

1. 编写一个块，可以输入部门号，并显示该部门所有员工姓名和工资；

|  |
| --- |
| *--定义游标变量，取出查询结果；*  DECLARE  *--定义游标类型*  TYPE emp\_ref\_cr\_type IS REF CURSOR;  *--定义游标变量*  emp\_ref\_cr emp\_ref\_cr\_type;    *--ename*  v\_ename emp.ename%TYPE;  *--sal*  v\_sal emp.sal%TYPE;    BEGIN    *--使用游标*  OPEN emp\_ref\_cr FOR  SELECT ename,sal  FROM emp  WHERE deptno=&no;    *--循环取出数据*  LOOP    FETCH emp\_ref\_cr INTO v\_ename,v\_sal;  *--判断游标是否为空*  EXIT WHEN emp\_ref\_cr%NOTFOUND;    *--输出*  DBMS\_OUTPUT.PUT\_LINE('员工姓名：' || v\_ename || '薪水' || v\_sal );  END LOOP;    *--关闭游标*  CLOSE emp\_ref\_cr;    END;  / |
|  |

1. 编写一个块，可以输入部门号，并显示该部门所有员工姓名和工资；若某个雇员工资低于200元，就加薪100元；

|  |
| --- |
| *--编写一个块，可以输入部门号，并显示该部门所有员工姓名和工资；*  *--若某个雇员工资低于200元，就加薪100元；*  declare  type rc\_empinfo is ref cursor;*--定义游标，存放员工信息；*  v\_empinfo\_rc rc\_empinfo;*--定义游标变量；*  v\_deptno emp.empno%type;*--定义变量，存放部门编号；*  v\_ename emp.ename%type;*--存放员工姓名；*  v\_sal emp.sal%type;*--存放员工薪水；*  v\_empno emp.empno%type;*--存放员工编号；*  begin    *--接收输入的部门编号；*  v\_deptno := &in\_deptno;    *--根据部门编号查询员工信息；*  open v\_empinfo\_rc for  select emp.empno,emp.ename,emp.sal  from emp  where emp.deptno=v\_deptno;    *--遍历游标，打印员工信息*  loop  fetch v\_empinfo\_rc into v\_empno,v\_ename,v\_sal;  exit when v\_empinfo\_rc%notfound;    *--判断员工薪水是否大于200*  if v\_sal<200  then  update emp set emp.sal=emp.sal+100  where emp.empno=v\_empno;  v\_sal:=v\_sal+100;  end if;    *--打印员工信息*  dbms\_output.put\_line('员工姓名：'||v\_ename||' ||薪水：'||v\_sal);    end loop;    *--关闭游标*  close v\_empinfo\_rc;  end;  / |

# 案例

## 分页存储过程

分页查询的SQL语句

|  |
| --- |
|  |
| *--创建一个包*  CREATE OR REPLACE PACKAGE SEPARATE\_PAGE\_PKG IS    *--定义游标*  TYPE rc\_data IS REF CURSOR;  *--定义存储过程*  PROCEDURE QUERY\_BY\_PAGE(table\_name IN VARCHAR2,*--表名*  page\_size IN NUMBER,*--每页记录数*  page\_now IN NUMBER,*--当前页数*  total\_size OUT NUMBER,*--总记录数*  total\_page OUT NUMBER,*--总页数*  o\_rc\_data OUT SEPARATE\_PAGE\_PKG.rc\_data  );  END SEPARATE\_PAGE\_PKG; |
| *--创建包体*  CREATE OR REPLACE PACKAGE BODY SEPARATE\_PAGE\_PKG IS  *--定义存储过程*  PROCEDURE QUERY\_BY\_PAGE(table\_name IN VARCHAR2,*--表名*  page\_size IN NUMBER,*--每页记录数*  page\_now IN NUMBER,*--当前页数*  total\_size OUT NUMBER,*--总记录数*  total\_page OUT NUMBER,*--总页数*  o\_rc\_data OUT SEPARATE\_PAGE\_PKG.rc\_data)  IS    *--定义字符型变量，存放SQL字符串*  v\_sql VARCHAR2(1000);  *--定义记录查询的开始值和结束值*  v\_start\_index NUMBER:=(page\_now-1)\*page\_size +1;  v\_end\_index NUMBER:=page\_now\*page\_size;    BEGIN  *--创建动态SQL语句*  v\_sql:='SELECT \* FROM ( SELECT temp.\*, ROWNUM rn FROM ( SELECT \* FROM '|| table\_name  ||') temp WHERE rn<'|| v\_end\_index||') WHERE ROWNUM>='||v\_start\_index;    *--把游标和SQL关联*  OPEN o\_rc\_data FOR v\_sql;    *--计算总记录数和总页数*  v\_sql:='SELECT COUNT(\*) FROM '||table\_name;    *--执行SQL语句,获取总记录数*  EXECUTE IMMEDIATE v\_sql INTO total\_size;    IF MOD(total\_size,page\_size)=0  THEN  total\_page:=total\_size/page\_size;  ELSE  total\_page:=total\_size/page\_size+1;  END IF;    END QUERY\_BY\_PAGE;  END SEPARATE\_PAGE\_PKG; |

1. 分页例题

|  |
| --- |
|  |

# 异常处理

## 预定义异常

常用预定义异常:

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

## 自定义异常

|  |
| --- |
|  |