Oracle 数据库存储过程技术文档

BY:Stone

本文中各例均使用 Oracle 数据库 demo 用户.

用户名:scott 用户口令:tiger

数据结构建立:

/*使用 system 用户及口令登录 oracle 数据库*/

$SQLPLUS system/passwd

/*建立 scott 用户口令为 tiger*/

$SQL>create user scott identified by tiger;

/*给 scott 用户授权*/

$SQL>grant create session to scott;

$SQL>exit;

$SQLPLUS scott/tiger

$SQL>start $ORACLE_HOME/sqlplus/demo/demobld.sql

主要数据结构:

```
CREATE TABLE EMP
(EMPNO NUMBER(4) NOT NULL,
      ENAME VARCHAR2(10),
      JOB VARCHAR2(9),
      MGR NUMBER(4),
      HIREDATE DATE,
      SAL NUMBER(7,2),
      COMM NUMBER(7,2),
      DEPTNO NUMBER(2));
```

```
CREATE TABLE BONUS
 (ENAME VARCHAR2(10),
      JOB VARCHAR2(9),
       SAL NUMBER,
       COMM NUMBER);
```

```
CREATE TABLE DEPT
     (DEPTNO NUMBER(2),
      DNAME VARCHAR2(14),
      LOC VARCHAR2(13) );
```

```
CREATE TABLE SALGRADE
     ( GRADE NUMBER,
       LOSAL NUMBER,
       HISAL NUMBER );
```

```
CREATE TABLE DUMMY
     ( DUMMY NUMBER );
```

第一章 **oracle** 存储过程概述

Oracle 存储过程(store procedure)作为 PL/SQL 语言的子程序,使用 PL/SQL 语言对数据处理逻辑,数据存储,数据操纵进行描述和封装,通过 oracle 其他工具(Pro*c&sqlplus 等)对存储过程调用,实现相应功能.

Oracle 存储过程在创建时经过数据库编译,作为数据库对象存储在数据库中,使用存储过程名称和输入输出参数实现存储过程描述的功能.

存储过程是由流控制和 SQL 语句书写的过程，这个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。在 ORACLE 中，若干个有联系的过程可以组合在一起构成程序包。

使用存储过程有以下的优点：

1. 存储过程的能力大大增强了 SQL 语言的功能和灵活性。存储过程可以用流控制语句编写，有很强的灵活性，可以完成复杂的判断和较复杂的运算.

2. 可保证数据的安全性和完整性。

3. 通过存储过程可以使没有权限的用户在控制之下间接地存取数据库，从而保证数据的安全。

4. 通过存储过程可以使相关的动作在一起发生，从而可以维护数据库的完整性。

5. 再运行存储过程前，数据库已对其进行了语法和句法分析，并给出了优化执行方案。这种已经编译好的过程可极大地改善 SQL 语句的性能。由于执行 SQL 语句的大部分工作已经完成，所以存储过程能以极快的速度执行。

6. 可以降低网络的通信量。

7. 使体现企业规则的运算程序放入数据库服务器中，以便集中控制。

当企业规则发生变化时在服务器中改变存储过程即可，无须修改任何应

用程序。企业规则的特点是要经常变化，如果把体现企业规则的运算程序放入应用程序中，则当企业规则发生变化时，就需要修改应用程序工作量非常之大（修改、发行和安装应用程序）。如果把体现企业规则的运算放入存储过程中，则当企业规则发生变化时，只要修改存储过程就可以了，应用程序无须任何变化.

Oracle 存储函数(FUNCTION)作为特殊的存储过程,与 C/C++语言函数相似,具备函数名,输入输出参数以及返回值.

存储过程和存储函数都是相对独立的实体.

Oracle 包(Package)为了管理上的方便,把一些相关的程序结构如存储过程,存储函数,变量,游标等组织在一起,构成一个包.Oracle 包具有面向对象程序设计语言的特点,是对 PL/SQL 程序设计元素的封装.包类似于 C++和 JAVA 语言中的类,其中变量相当于类中的成员变量,存储过程和存储函数相当于类方法.包中的元素分为共有元素和私有元素,两种元素允许访问的程序范围不同.

存储过程基本结构**(PROCEDURE)**

1.1.1 创建存储过程

```
CREATE  [OR REPLACE] PROCEDURE  存储过程名
  （参数定义标）
IS/AS
变量定义
BEGIN
 PL/SQL 语句块
EXCEPTION
```

例外处理

END 存储过程名

定义说明:

1. 参数定义表:

存储过程可以有三类参数

IN 数据从调用环境传入存储过程

OUT 数据从存储过程传入调用环境

INOUT 数据可以传入或传出存储过程

参数使用原则:

参数类型可以为 ORACLE 允许的任意类型,也可为%TYPE(与其他某一变量类型一致)或%ROWTYPE(与数据库中某一对象表,游标等数据类型一致) 类型

指定参数时,不能指定长度

所有输出参数(OUT)只能出现在 SELECT INTO 语句或赋值语句中.

尽量减少 IN 参数个数.

2. 变量定义

变量类型可以为 ORACLE 允许的任意类型,也可为%TYPE(与其他某一变量类型一致)或%ROWTYPE(与数据库中某一对象表,游标等数据类型一致) 类型.

3. 例外处理

存储过程例外处理与 PL/SQL 错误处理一致,可按条件执行相应的操作

```
/*CREATE 创建存储过程*/
/*REPLACE 替换存储过程*/
/*CREATE OR  REPLACE 如存储过程不存在则创建,否则替换*/
CREATE OR REPLACE PROCEDURE
/*存储过程名为 raise_sal*/
/*参数为输入 NUMBER 型 emp_id 和输入 NUMBER 型 add_sal*/
     raise_sal(emp_id IN NUMBER,add_sal IN NUMBER)
AS
/*无局部变量声明*/
BEGIN
     /*PLSQL 语句块*/
    UPDATE emp
    SET sal = sal+ add_sal
     WHERE empno = emp_id;
EXCEPTION
   /*例外处理 NO_DATA_FOUND 数据未找到时执行*/
     WHEN NO_DATA_FOUND THEN
     /*raise_application_error(错误代码,'错误信息') 向调用环境返回错误信息*/
                        raise_application_error(-20011,'InvalidEmployee'||
TO_CHAR(emp_id));
END raise_sal;
```

存储过程简例

1.1.2 存储过程删除

$SQLPLUS> DROP PROCEDURE 过程名

1.1.3 调用存储过程

1.SQLPLUS 环境

语法 $SQLPLUS> EXECUTE 存储过程名

参数 SQLPLUS 中的变量或常量

例: $SQL>EXECUTE raise_sal(10,1000);

2.SQLDBA 环境

语法 $SQLPLUS> EXECUTE 存储过程名

参数 SQLPLUS 中的变量或常量

3.SQLFORMS

语法:过程名

参数:SQLFORMS 中的域或全局变量

4.PLSQL 或其他存储过程

语法:过程名　参数:PLSQL 局部变量

5.

```
#include <stdio.h>
EXEC SQL INCLUDE SQLCA;
main(){
/*声明宿主变量*/
 EXEC SQL BEGIN DECLARE SECTION;
                char    *oid="scott/tiger";
                int     tt;
EXEC SQL END DECLARE SECTION;
/*连接数据库*/
        EXEC SQL CONNECT :oid;
        if(sqlca.sqlcode != 0) {printf("connect database  error \n");exit(0);}
/*调用存储过程 raise_sal*/
        EXEC SQL EXECUTE
         BEGIN
            Raise_sal(100,1000);
          END;
        END-EXEC;
/*调用完毕*/
        if(sqlca.sqlcode != 0) {printf("Execute  error \n");exit(0);}
        /*断开数据库连接*/
EXEC SQL COMMIT WORK RELEASE;
}
```

1.2 存储函数(FUNCTIONE)

存储函数是一类特殊的存储过程,与一般存储过程不同的是存储函数必须返回一个值.

1.2.1 创建存储函数

CREATE [OR REPLACE] FUNCTION 存储函数名

RETUNR 返回值类型

IS/AS

变量声明

BEGIN

PLSQL 语句块

EXCEPTION

例外处理

END 存储函数名

备注: 返回值类型不带长度

1.2.2 删除存储函数

$SQL>drop function 存储函数名

例 1.3

从员工信息表(emp)中选择部门代号为 v_empno 员工的工资

```
CREATE OR REPLACE FUNCTION get_sal(v_empno IN emp.empno%TYPE)
RETURN NUMBER
IS
        v_emp_sal emp.sal%TYPE:=0;
BEGIN
        SELECT sal INTO v_emp_sal
        FROM emp
        WHERE empno=v_empno;
        RETURN(v_emp_sal);
END get_sal;
```

创建存储函数

1.3 包(package)

1.3.1 包的基本结构

包中可以包含过程(procedure),函数(function),变量(variable) 游标(cursor),常量(constant),例外处理(exception).

包由两部分组成:包定义和包体

包定义:对包的公共元素如过程,函数,变量,常量,游标,例外情况等进行说明,可在包外独立使用这些公共元素.

包体部分包括包中使用的私有元素和包的公共元素的定义.

1.3.2 包的创建

1.创建包定义

CREATE〔OR REPLACE〕PACKAGE package_name

IS/AS

公共元素声明

END package_name;

2.创建包体

CREATE〔OR REPLACE〕 PACKAGE BODY package_name

IS/AS

私用元素定义

公共元素定义

BEGIN

PLSQL 语句

END package_name;

例 1.4

```
CREATE OR REPLACE PACKAGE emp_package
AS
/*声明函数*/
FUNCTION
 hire_emp(name VARCHAR2,job VARCHAR2,mgr NUMBER,hiredate DATE,sal NUMBER
,comm NUMBER,deptno NUMBER)
RETURN NUMBER;
/*声明过程*/
PROCEDURE fire_emp(emp_id NUMBER);
PROCEDURE sal_raise(emp_id NUMBER,sal_id NUMBER);
END emp_package;
```

创建包定义

```
CREATE OR REPLACE PACKAGE BODY emp_package  AS
  /*定义函数*/
  FUNCTION
    hire_emp(name VARCHAR2,job VARCHAR2,mgr NUMBER,hiredate
                DATE,sal NUMBER,comm NUMBER,deptno NUMBER)
    RETURN NUMBER  IS
      new_empno NUMBER(10);
    BEGIN
      SELECT emp_sequence.NEXTVAL INTO new_empno FROM emp;
      INSERT INTO emp
      VALUES(new_empno,name,job,mgr,hiredate,sal,comm,deptno);
    RETURN (new_empno);
    END hire_emp;
   /*定义过程*/
   PROCEDURE fire_emp(emp_id NUMBER) IS
     BEGIN
        DELETE FROM emp WHERE empno=emp_id;
        IF SQL%NOTFOUND THEN
  raise_Application_error(-20011,'Ivalid EmployeeNUMBER'||TO_CHAR(emp_id));
        END IF;
    END fire_emp;
   PROCEDURE sal_raise(emp_id NUMBER,sal_id NUMBER) AS
     BEGIN
       UPDATE emp  SET sal=sal+sal_id  WHERE empno=emp_id;
       IF SQL%NOTFOUND THEN
  raise_Application_error(-21011,'Ivalid EmployeeNUMBER'||TO_CHAR(emp_id));
       END IF;
     END sal_raise;
/*结束包定义*/
 END emp_package;
```

创建包体

1.3.3 调用包中元素

$SQL>EXECUTE 包名.元素名(参数列表);

例: $SQL>EXECUT emp_package.raise_sal(7654,100);

1.3.4 包的修改和删除

删除包:$SQL>drop PACKAGE 包名

$SQL>drop PACKAGE BODY 包名

备注:包定义和包体应该同时修改,并保持一致.

第二章 oracle 存储过程基础——PL/SQL

Oracle 存储过程以 PL/SQL 作为其流程控制语言,可以理解 Oracle 存储过程为具有名称和输入输出参数的 PL/SQL 语句块,因此,本章介绍 PL/SQL 的语法和使用.

pl/sql 基础

2.1.1 PL/SQL 简介

一 **PL/SQL** 优点

1.过程化能力

PL/SQL 称为 SQL 过程语言,他将高级程序设计语言中所具备的过程能力与非过程化的 SQL 语言有机的结合在一起,形成了一个集成式的 Oracle 数据库事务处理应用开发工具,为应用开发者提供了增强生产力的机制.

PL/SQL 以块(blocks)为单位,较大的块中可以镶嵌子块,可以将复杂的问题分解成一组易于控制的,很好定义的逻辑模块.

在 PL/SQL 块中可以进行变量定义,例外处理,然后在 SQL 语句中调用.PL/SQL 块中可以使用过程化语言控制结构进行程序设计,包括条件转移,循环控制,游标.

2.改进处理性能

使用 PL/SQL,Oracle 数据库将 PL/SQL 语句块作为一组,一次提交给 Oracle 服务进程,减少 Oracle 客户服务进程间的交互.

3.良好的应用移植性

由于 PL/SQL 是模块化结构,在进行应用移植时可以将模块内部的复杂处理忽略,二只考虑模块间的数据交换.

4.与关系数据库管理系统(RDBMS)集成

使用 PL/SQL,可以将许多用户都可能用到的处理编程封装过程或包,与关系数据库管理系统有效集成.这样,用户可使用 Oracle 工具直接调用,提高了开发效率,减少了再编译时间,提高系统性能.

二 **PL/SQL** 应用环境

SQL*PLUS;SQL*FORMS;Oracle CDE 工具;Pro*C

三 **PL/SQL** 块的基本结构

基本的 PL/SQL 块由定义部分,执行部分,例外处理部分组成

```
DECLARE
  定义部分
BEGIN
  执行部分
EXCEPTION
  例外处理部分
END
```

PL/SQL 块基本结构

1.定义部分:

定义在程序执行部分使用的常量,变量,游标和例外处理名称

2.可执行部分

包括数据库操作语句和 PL/SQL 块控制语句

3.例外处理部分

对执行部分的所有 PL/SQL 语句的执行进行监控,如执行发生例外,则程序跳到该部分执行

2.1.2 一个简单的 PL/SQL 块

```
/*一个简单 PLSQL 块开始*/
/*定义变量*/
DECLARE
 /*ROWTYPE 类型 定义变量 myrecord 为一结构,与表 emp 各字段数据类型一致*/
 myrecord emp%ROWTYPE;
 /*定义变量 myempno 类型为 number(4) 变量非空,初值为 8000*/
 myempno  number(4) NOT NULL:=8000;
 /*TYPE 类型  定义变量 myname 类型与表 emp 中 ename 字段一致*/
 myname   emp.ENAME%TYPE;
 /*CONSTANT 关键字 定义常量 addsal 值为 500*/
 addsal   CONSTANT number(4):=500;
BEGIN
 SELECT * INTO myrecord
 FROM EMP
 WHERE ENAME='SMITH';
 myname:='WUCHEN';
 INSERT INTO EMP(EMPNO,ENAME,SAL,COMM,JOB,HIREDATE,DEPTNO)
 VALUES(myempno,myname,myrecord.sal,myrecord.comm,myrecord.job,myrecord.h
iredate,myrecord.deptno);
 UPDATE EMP SET sal = sal + addsal;
END;
```

一个简单的 PL/SQL 块程序

执行步骤:

$SQLPLUS scott/tiger

$SQL>start ./PLSQL 块名称

$SQL>. (输入符号点)

$SQL>r (字母 r 或符号/执行程序)

一 定义变量

在 PLSQL 中所使用的变量必须在变量定义部分明确定义.变量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.

定义格式:

变量标示符 [CONSTANT] 数据类型 [NOT NULL]

[:=缺省值或 PLSQL 表达式];

变量标示符命名规则应遵循 SQL 实体命名规则

定义常量时必须加关键字 CONSTANT 必须为其赋值

如该变量不允许为空值,必须加参数 NOT NULL

变量赋值时,可使用:=或使用关键字 DEFAULT.

每行只能定义一个变量.

数据类型

简单数据类型(标量数据类型):

NUMBER(m,n)	数字类型 m 为总长度,n 为小数长度
CHAR(m)	字符型 m 为变量长度
VARCHAR2(m)	可变长字符型 m 为最大长度
DATE	日期型
LONG	长型
BOONEAN	布尔型 值为 TRUE FALSE NULL

已定义变量%TYPE 定义成与已定义变量一致类型

复合数据类型

变量标示符 对象标示符%ROWTYPE;

对象标示符可为表,游标等,变量被定义成与数据库对象一致的类型结构,当数据库结构改变时,不必改变改变量的定义.

使用%ROWTYPE 分为两种不同情况:

1. 作为查询结果存放空间时:

select 字段列表 INTO %ROWTYPE 型变量

2. 作为单个成员使用:

%ROWTYPE 变量名.字段名

二 变量赋值

变量赋值时需使用 PLSQL 变量赋值操作符(:=)

1.常量赋值: 变量名 := 常量

2.变量赋值: 变量名 := 同类型变量

3.为%ROWTYPE 型变量赋值

a . select 列表 into %ROWTYPE 型变量 from 表

b. 为%ROWTYPE 变量每个成员单独赋值

如 %ROWTYPE 变量.变量成员 = 值

4.表达式赋值: 变量名:=表达式或函数

三 **PLSQL** 中使用的 **SQL** 语句

在 PL/SQL 块中,所有对数据库的访问和操作还是要经由 SQL 语言进行,在 PL/SQL 块中可以使用数据查询语言,数据操纵语言和数据控制语言,但不能使用数据定义语言具体地说可以使用 select,insert,update,delete,commit,rollback, 但不能使用 create,alter,drop,grant,revoke.

1.PL/SQL 块中使用查询语句

在 PL/SQL 中使用 select 时必须加 INTO 语句.

INTO 子句后的变量个数和位置必须与 SELECT 后的字段列表相同

SELECT 语句中的 WHERE 条件可以包含 PL/SQL 块中定义的变量及表达式.

SELECT 语句必须保证有且仅有一条记录返回,否则出错:

TOO_MANY_ROWS -1422 记录多于一条

NO_DATA_FOUND -1403 没找到记录

在 SQL 语句中使用的变量名应与数据库字段名区分开.

```
DECLARE
 /* 变量 emp_rec 为 ROWTYPE 型结构变量,结构类型与表 EMP 结构类型一致*/
       emp_rec EMP%ROWTYPE;
 /*变量 v_ename 与表 emp 中 ename 字段类型一致*/
 v_ename EMP.ename%TYPE:='SMITH';
BEGIN
 SELECT * INTO emp_rec FROM emp
 /*ENAME 字段 = 变量 v_ename 值*/
 WHERE ENAME=v_ename;
END;
```

PL/SQL 使用查询语句

```
DECLARE
 v_empno emp.empno%TYPE NOT NULL:=8000;
 v_ename emp.ename%TYPE:='Bill';
 v_job emp.job%TYPE:='MANAGER';
 v_sal emp.sal%TYPE:=2000;
 v_comm emp.comm%TYPE:=1000;
  v_hiredate emp.hiredate%TYPE:=SYSDATE;
 v_deptno emp.deptno%TYPE:=10;
 v_addsal emp.sal%TYPE;
BEGIN
 INSERT INTO emp(empno,ename,job,sal,comm,hiredate,deptno)
 VALUES (v_empno,v_ename,v_job,v_sal,v_comm,v_hiredate,v_deptno);
 v_addsal:=1000;
 UPDATE emp SET sal=sal+v_addsal WHERE empno=v_empno;
 DELETE FROM emp WHERE empno > 8000;
 COMMIT WORK;
END;
```

PL/SQL 使用数据操纵语句

3.PL/SQL 块中使用事物控制语句

提交命令(COMMIT):结束当前事物,对数据库作永久性改变

语法 COMMIT [WORK]

回退命令(ROLLBACK): 结束当前事物,并放弃对数据库所作修改

语法 ROLLBACK [WORK]

保存点(SAVEPOINT):为了避免一处失败导致全部事物回滚,可以使用 SAVEPOINT 和 ROLLBACKTO 语句

语法 SAVEPOINT 标记

ROLLBACK TO 标记

2.1.3 PL/SQL 流程控制

PL/SQL 具有与高级语言类似的流程控制语句.PL/SQL 主要控制语句有:

条件控制语句

循环控制语句

跳转控制语句

1.条件控制语句:

IF_THEN_ELSE 语句

语法: IF 条件 THEN

语句;

ELSE

语句;

END IF;

条件可为 IS NULL 或 NOT IS NULL 以及 AND, OR, NOT,逻辑运算符

例:将 emp 表中的雇员名为 SMITH 雇员的工资修改,如果工资大于$2000,则加$500,否则加$1000.

```
DECLARE
 v_ename emp.ename%TYPE:='SMITH';
 v_addsal emp.sal%TYPE;
 v_sal emp.sal%TYPE;
BEGIN
  SELECT sal INTO v_sal
 FROM emp WHERE ename=v_ename;
 IF v_sal > 2000 THEN
 v_addsal:=500;
 ELSE
 v_addsal:=1000;
 END IF;
 UPDATE emp SET sal=sal+v_addsal
 WHERE ename=v_ename;
END;
```

PL/SQL 控制语句 IF_THEN_ELSE 例程

IF_THAN_ELSIF 语句:

语法:IF 条件 THEN

语句;

ELSIF 条件 THEN

语句;

[ELSIF 条件 THEN 语句;]

[ELSE 语句;]

END IF

根据 emp 表中的工种为 SMITH 修改工资,若工种为 MANAGER,工资加$1000,工种为 SALESMAN,工资加$500,工种为 ANALYST,工资加$200,否则加$100.

```
DECLARE
 v_job emp.job%TYPE;
 v_addsal emp.sal%TYPE;
BEGIN
 SELECT job INTO v_job FROM emp
 WHERE ename='SMITH';

 IF v_job='MANAGER' THEN
 v_addsal :=1000;
 ELSIF v_job='SAIESMAN' THEN
 v_addsal :=500;
 ELSIF v_job='ANALYST' THEN
 v_addsal :=200;
 ELSE
 v_addsal :=100;
 END IF;

 UPDATE emp SET sal=sal+v_addsal
 WHERE ename='SMITH';
 COMMIT WORK;
END;
```

条件控制语句例程

2.循环控制语句:

LOOP 循环:

语法: LOOP

语句;

[EXIT [WHEN 条件]];

END LOOP

例 给 10 号部门增加新雇员,只确定雇员代号,其他信息忽略.

```
DECLARE
 v_empno emp.empno%TYPE:=8000;
BEGIN
 LOOP
 INSERT INTO emp(deptno,empno)
 VALUES(10,v_empno);
 v_empno:=v_empno+100;
 /*如果雇员代号>=9000 则退出循环*/
 EXIT WHEN v_empno >= 9000;
 END LOOP;
END;
```

FOR 循环:

语法: FOR 计数器 IN [REVERSE] 下界…上界 LOOP

语句;

END LOOP;

计数器用于控制循环次数的变量,无需在定义部分做出定义,系统隐含定义为整数,REVERSE 表示计数器从上界到下界递减计数,下界定义初值,上界定义终值,下界应小于上界.对计数器不可作赋值操作.

例 同上例

```
DECLARE
 v_deptno emp.deptno%TYPE:=10;
BEGIN
 FOR i IN 1...10 LOOP
 INSERT INTO emp(deptno,empno)
 VALUES(v_deptno,8000+i*100);
 END LOOP;
 COMMIT WORK;
END;
```

循环控制例程

WHILE 循环

语法: WHILE 条件 LOOP

语句;

END LOOP;

例 同上例

```
DECLARE
 i number(2):=1;
BEGIN
 WHILE i<=10 LOOP
 INSERT INTO emp(deptno,empno)
 VALUES(10,8000+i*100);

 i:=i+1;
 END LOOP;
 COMMIT WORK;
END;
```

循环控制例程

3.跳转控制语句:

语法: 《标号》

其他语句;

GOTO 标号;

跳转语句可在统一块语句间跳转

跳转语句可从子块跳转倒父块中，但不能从父块跳转到子块中

跳转语句不能在 IF 语句体外跳到 IF 体内

跳转语句不能从循环体外跳到循环体内

例 同上例

```
DECLARE
  v_empno emp.empno%TYPE := 8000;
BEGIN
 <<repeat>>
 INSERT INTO emp(deptno,empno)
 VALUES(10,v_empno);
 v_empno:=v_empno+100;
 IF v_empno <= 9000 THEN
 GOTO repeat;
 END IF;
END;
```

跳转控制例程

游标**(CURSOR)**

在 PL/SQL 查询语句中,有时会返回多条记录,这时如使用 SQL 语句则会出错.因此,在查询语句返回多条记录或不知返回结果数目时,必须使用游标.

游标的概念

PL/SQL 中游标的使用与 Pro*C 中游标使用相似,包括定义,打开,提取数据,关闭四个步骤.一般游标在定义，打开后使用循环语句逐条处理提取的数据。

一 定义游标

语法: CURSOR 游标名称 IS

SELECT 语句;

定义游标应写在 PL/SQL 语句的 DECLARE 变量定义部分

定义游标时 SELECT 语句中不可有 INTO 子语句

在 SELECT 语句中使用的变量必须在定义游标前定义

二 打开游标

语法:OPEN 游标名;

在 BEGIN 语句之后,可以打开游标,在打开游标之前,必须对游标所涉及到的变量赋值.

三 利用游标提取数据

语法: FETCH 游标名 INTO 变量 1,变量 2,…..

游标每次只能取到一条数据,同时游标指针下移,等待取下一条数据.该条语句变量列表应与定义游标时的参数列表一致

四 关闭游标

语法: CLOSE 游标名

关闭游标,释放资源,游标关闭后不能再提取数据.

游标的属性

游标的属性标示游标的运行状况.

%ISOPEN	布尔型	表示游标是否打开
%NOTFOUND	布尔型	描述最后一次 FETCH 的结果
%FOUND	布尔型	描述最后一次 FETCH 的结果,与 NOTFOUND 相反
%ROWCOUNT	数字型	描述当前取值的条数

例 查询 10 号部门所有雇员的姓名工资，并插入到临时表 tmp 中

tmp 表结构为 t1 char(20),t2 number(10)

以下使用三种不同的循环方法实现该例：

例 查询 10 号部门所有雇员的姓名工资，并插入到临时表 tmp 中

```
DECLARE
        v_deptno emp.deptno%TYPE:=10;
            /*定义游标*/
        CURSOR C1 IS
                SELECT ename,sal FROM EMP
                WHERE DEPTNO=v_deptno;
            /*定义 ROWTYPE 型变量 emp_rec 变量为与游标 C1 结构一致的结构型，即
ename,sal 结构*/
        emp_rec C1%ROWTYPE;
BEGIN
            /*打开游标*/
        OPEN C1;
            /*循环开始*/
        LOOP
                        /*提取数据到变量 emp_rec 结构中*/
                FETCH C1 INTO emp_rec;
                        /*如果数据提取完毕，退出*/
                EXIT WHEN C1%NOTFOUND;
                INSERT INTO tmp
                 VALUES(emp_rec.ename,emp_rec.sal);
            /*结束循环*/
        END LOOP;
            /*关闭游标*/
        CLOSE C1;
        COMMIT WORK;
END;
```

游标使用 NOTFOUND 属性

```
/*如果游标已经打开*/
IF C1%ISOPEN  THEN
      FETCH  C1  INTO  emp_rec;
/*如果游标未打开，先打开游标再提取数据*/
ELSE
      OPEN  C1;
      FETCH  C1 INTO emp_rec;
END IF;
```

游标使用 ISOPEN 属性

```
DECLARE
       v_deptno emp.deptno%TYPE:=10;
       CURSOR C1 IS
              SELECT ename,sal FROM EMP
              WHERE DEPTNO=v_deptno;
       emp_rec C1%ROWTYPE;
BEGIN
       OPEN C1;
 FETCH C1 INTO emp_rec;
 WHILE C1%FOUND LOOP
              INSERT INTO tmp
               VALUES(emp_rec.ename,emp_rec.sal);
              FETCH C1 INTO emp_rec;
       END LOOP;
       CLOSE C1;
       COMMIT WORK;
END;
```

游标使用 FOUND 属性

游标中 **FOR** 循环的使用

游标使用 FOR 循环可以简化游标的操作，使用 FOR 循环时，系统隐含定义了一个数据类型为%ROWTYPE 的变量为循环计数器，此时在 PL/SQL 块中不用显式的打开，关闭游标。

语法： FOR 组合变量名 IN 游标名 LOOP

语句;

END LOOP;

```
DECLARE
        v_deptno emp.deptno%TYPE:=10;
        CURSOR C1 IS
                SELECT ename,sal FROM EMP
                WHERE DEPTNO=v_deptno;
        emp_rec C1%ROWTYPE;
BEGIN
 FOR emp_rec IN C1 LOOP
                INSERT INTO tmp
 VALUES(emp_rec.ename,emp_rec.sal);
                FETCH C1 INTO emp_rec;
        END LOOP;
        COMMIT WORK;
END;
```

游标使用 FOR 循环

带参数游标的使用方法

在定义游标时，可以加入参数，参数再游标中使用。

语法： DECLARE

CURSOR 游标名（参数列表） IS

SELECT 语句;

```
DECLARE
 /*游标 C1 带参数 v_deptno*/
        CURSOR C1(v_deptno emp.deptno%TYPE) IS
                SELECT ename,sal FROM EMP
                WHERE DEPTNO=v_deptno;
        emp_rec C1%ROWTYPE;
BEGIN
 /*参数值为 10*/
        FOR emp_rec IN C1(10) LOOP
                INSERT INTO tmp
                 VALUES(emp_rec.ename,emp_rec.sal);
                FETCH C1 INTO emp_rec;
        END LOOP;
 /*参数值为 20*/
        FOR emp_rec IN C1(20) LOOP
                INSERT INTO tmp
                 VALUES(emp_rec.ename,emp_rec.sal);
                FETCH C1 INTO emp_rec;
        END LOOP;
        COMMIT WORK;
END;
```

使用带参数游标

动态 **SQL** 语句

动态 SQL 语句是指语句文本在应用程序运行时才被建立的 SQL 语句或 PL/SQL 块，动态 SQL 语句文本中可以包含结合参数占为符。在使用占位符时，必须在其前面加冒号（：）前缀。使用动态 SQL 语句还能执行在 PL/SQL 块中不能静态执行的 SQL 语句，如 DDL 语句。

语法： EXECUTE IMMDIATE 动态语句串

[INTO {variable[,variable]….|record}]

[USING [IN|OUT|IN OUT] bind_argument

[,[IN|OUT|IN OUT] bind_argument]…];

其中动态语句串表示 SQL 语句或 PL/SQL 块文本，对于 SQL 语句不能使用语句结束符（；），对于 PL/SQL 块文本，必须加结束符。INTO 子句只能用于单行查询语句，将查询结果存储到指定变量 variable 或记录 record 变量中。USING 子句使用 bind_argument 值替换动态语句串中的占位符。USING 语句不能传递布尔型变量（TRUE FALSE NULL）。由于 Oracle 自动将所有未赋值的变量设置为 NULL,因此在需要传递 NULL 变量时，使用未赋值变量即可。

例：
```
DECLARE
    no_initialization NUMBER;
BEGIN
    EXECUTE IMMEDIATE 'DELETE FROM emp WHERE comm=:x'
        USING no_initialization;
END;
```

使用未赋值变量传递 NULL

USING 子句中，不能使用结合参数传递对象名称。

例：

```
DECLARE
    Tab_name VARCHAR2(30);
BEGIN
    Tab_name :='scott.mytab'; ----对象名称
/**错误执行*/
    EXECUTE IMMDIATE 'DROP TABLE :tab' USING tab_name;
/*正确执行*/
    EXECUTE IMMDIATE 'DROP TABLE '||tab_name;
END;
```

USING 语句中不能传递对象

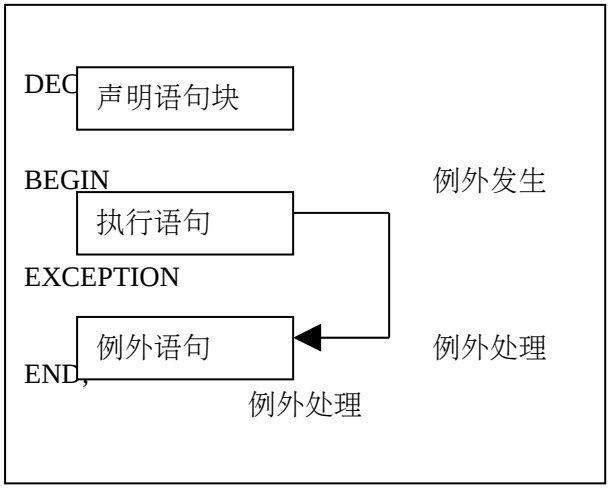
USING 语句中结合变量默认参数模式为输入参数，当结合变量为输出参数或输入输出参数时应加 OUT 或 IN OUT 选项说明。

```
DECALRE
    Dyna_sql VARCHAR2(128);
    Old_loc  VARCHAR2(15);
BEGIN
                                    /*para 为占位符*/
    Dyna_sql:='update dept set loc=''BEIJING''  where deptno=90 returning loc into :para';
    EXECUTE IMMDIATE dyna_sql USING OUT old_loc;
END;
                                    Old_loc 为输出变量
```

例外处理

例外处理是指程序在执行过程中的警告或错误的处理。

语法:




```
EXCEPTION
    WHEN 例外情况 1[OR 例外情况 2…] THEN
        语句;
    WHEN 例外情况 3[OR 例外情况 4…] THEN
        语句;
    [WHEN OTHERS THEN  语句;  ]
```

常见系统预定义例外情况

NO_DATA_FOUND	ORA_01403	执行 SELECT 时未找到数据
TOO_MANY_ROWS	ORA_01427	未使用游标的 SELECT 语句返回了多行数据
INVALID_CURSOR	ORA_1001	非法的游标操作
VALUES_ERROR	ORA_06502	出现数字，数据转换，截字符串错误
INVALID_NUMBER	ORA_01722	字符串向数字转换失败
ZERO_DIVIDE	ORA_01476	分母未零
DUP_VAL_ON_INDEX	ORA_0001	向具有唯一索引表中插入重复键值
TIMEOUT_ON_RESOURCE	ORA_00051	等待资源超时
INVALID_CURSOR	ORA_1001	试图关闭一个未打开的游标
NOT_LOGGED_ON	ORA_1012	数据库未联接
LOGIN_DENIED	ORA_1017	登录数据库失败
SYS_INVALID_ROWID	ORA_1410	无效字符串
STORAGE_ERROR	ORA_30625	PL/SQL 用尽内存或内存被破坏
ROWTYPE_MISMATCH	ORA_6504	赋值时，类型不匹配
CURSOR_ALREADY_OPEN	ORA_6511	试图打开一个已达开的游标
PROGRAM_ERROR	ORA_6501	PL/SQL 块内部错误

例 删除 EMP 表中的 SMITH 信息。

```
DECLARE
 v_ename emp.ename%TYPE:='SMITH';
BEGIN
 DELETE FROM emp WHERE ename=v_ename;
 COMMIT WORK;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 INSERT INTO tmp(t1) VALUES('SMITH is not found');
 WHEN TOO_MANY_ROWS THEN
 ROLLBACK WORK;
            INSERT INTO tmp(t1) VALUES('SMITH is too many rows');
       WHEN OTHERS THEN
 ROLLBACK WORK;
            INSERT INTO tmp(t1) VALUES('other error  occurred');
END;
```

系统预定义例外处理

用户可以自定义例外处理，例外处理的名称需在定义部分声明，在引发例外处理时，需使用 RAISE 子句：

语法： RAISE 例外处理名称

```
DECLARE
 v_sal_err emp.sal%TYPE:=5000;
 CURSOR C1 IS SELECT sal FROM EMP;
 v_sal emp.sal%TYPE;
sal_error EXCEPTION;  --定义例外名称
BEGIN
      OPEN C1;
            FETCH C1 INTO v_sal;
            LOOP
                  EXIT WHEN C1%NOTFOUND;
                        IF v_sal > 5000 THEN
                              RAISE sal_error; ---调用例外处理
                        ELSE
                              FETCH C1 INTO v_sal;
                        END IF;
            END LOOP;
      CLOSE C1;
EXCEPTION
       WHEN sal_error THEN   ---例外处理
              INSERT INTO tmp(t1) values ('found error sal>5000');
              CLOSE C1;
END;
```

2.5 一个完整的 PL/SQL 实例

```
DECLARE
        CURSOR dept_cursor IS SELECT distinct deptno FROM DEPT;
        deptrec dept_cursor%ROWTYPE;
        CURSOR emp_cursor(v_deptno emp.deptno%TYPE) IS
           SELECT ename,sal FROM EMP WHERE deptno=v_deptno order by sal;
        emprec emp_cursor%ROWTYPE;
        allsal  NUMBER(5):=0;
        allemp  NUMBER(3):=0;
        ERR     EXCEPTION;
BEGIN
        FOR deptrec IN dept_cursor LOOP
                FOR emprec IN emp_cursor(deptrec.deptno) LOOP
                        IF emprec.sal > 5000 THEN
                                RAISE ERR;
                        ELSIF emprec.sal > 2000 THEN
                 INSERT INTO highsal(ename,sal)VALUES(emprec.ename,emprec.sal);
                        ELSE
                 INSERT INTO lowsal(ename,sal) VALUES(emprec.ename,emprec.sal);
                        END IF;
                        allsal:=allsal+emprec.sal;
                END LOOP;
                INSERT INTO summary(deptno,sum_emp,sum_sal)
                VALUES (deptrec.deptno,allemp,allsal);
                allsal:=0;
        END LOOP;
        COMMIT WORK;
EXCEPTION
        WHEN ERR THEN
                ROLLBACK WORK;
                INSERT INTO tmp(t1) VALUES('found sal > 5000');
                COMMIT WORK;
        WHEN OTHERS THEN
                ROLLBACK WORK;
                INSERT INTO tmp(t1) VALUES('found other error');
                COMMIT WORK;
END;
```

第三章 **oracle** 存储过程讨论

Oracle 存储过程是一组有名字并且可以传递参数的 PL/SQL 语句集合，Oracle 函数是具有返回值的存储过程，包是一组具有类似属性的存储过程的集合。

函数（**FUNCTION**）

PL/SQL 用户定义函数是一组 PL/SQL 语句的集合，它具有两个特点：

1． 每个函数具有固定的名称。

2． 函数向其调用者返回数据。

3.1.1 用户函数创建，编译，删除

用户函数建立：

语法

CREATE [OR　REPLACE] FUNCTIONE [schema.]function_name

（[argument [IN|OUT[NOCOPY] | IN OUT [NOCOPY]] datatype [,….]]）

RETURN datatype

[invoker_rights_clause] [DETERMINISTIC] [PARALLEL_ENABLE]

{IS|AS}

PL/SQL block;

从功能上分可将该语句分为两部分：{IS|AS}前为函数定义部分，说明函数的参数数量，类型；第二部分为{IS|AS}以后部分，这是函数体，实现函数功能

OR REPLACE 语句说明当创建用户函数时，该函数已经存在，则用当前函数定义替换原函数定义。

Schema 说明所创建的函数的模式名称，默认为当前用户。

Function_name 用户函数的名称。

Argument 说明参数名称，参数类型，参数为输入，输出，输入输出。

Datatype　说明参数类型，它可以是 PL/SQL 支持的所有数据类型。在制定数据类型时，不能指定长度，精度和小数位数以及 NOT NULL 等约束。

NOCOPY 说明参数数据传递方法。地址传递或值传递。当作值传递时，大量数据需在内存中拷贝，影响效率，因次考虑使用 NOCOPY 参数，进行地址传递。

RETURN 说明函数返回值的数据类型。

Invoker_rights_clause 说明应用程序在调用函数时所使用的权限模式:

AUTHID DEFINER　定义者权限

AUTHID CURRENT_USER 当前用户权限

DETERMINISTIC Oracle 优化提示选项，他提示系统可以使用函数返回值的存储备份

PARALLEL_ENABLE 说明函数能够被并行查询操作所执行。

在 Oracle 服务器执行 CREATE FUNCTION 语句时，用户应具备一定的权限，用户在其自己的模式下创建函数时，应拥有 CREATE PROCEDURE 权限，用户在其他用户模式下创建函数时，需要拥有 CREATE ANY PROCEDURE 权限。

例：创建函数，查询指定部门的工资总和。

```
CREATE OR REPLACE FUNCTION get_salary(
        dept_no NUMBER, --部门编号
        emp_count OUT INTEGER) --输出参数，部门人数
RETURN NUMBER IS
        v_sum NUMBER(10,2);--返回指定部门的工资总和
BEGIN
        SELECT sum(sal),count(*) INTO v_sum,emp_count
                FROM EMP
                WHERE deptno=dept_no;
        RETURN v_sum;
END get_salary;
```

创建函数

修改函数：

语法： ALTER FUNCTION [schema.]function_name COMPILE [DEBUG]

DEBUG 选项指示 PL/SQL 编译起在编译时生成 PL/SQL 调试程序所使用的符号代码。

用户执行 ALTER FUNCTION 时必须拥有 ALTER ANY PROCEDURE 权限

删除函数：

语法：DROP FUNCTION [schema.]function_name;

用户可以删除自己用户下的所用函数，删除其他用户的函数时必须具有 DROP

ANY PROCEDURE 系统权限。

3.1.2 参数传递

应用程序在调用函数时，可使用以下三种方法向函数传递参数：

1． 位置表示法

语法：Argument_value1[,argument_value2….]

所传递的参数数量，数据类型和参数模式必须与定义时一致。

```
例：统计 30 号部门的工资和人数
DECLARE
      V_num INTEGER;
      V_sum NUMBER(8,2);
BEGIN  --用 v_sum 接收函数返回值
      V_sum:=get_salary(30,v_num);--用 v_num 变量做函数输出参数
END;
```

2． 名称表示法：

语法：argument=>parameter [,….]

argument 为形式参数名称，必须与函数定义时的形式参数名称相同， parameter 为实际参数。在这种格式下，形式参数与实际参数一一对应，因此参数的顺序可以任意排列。

```
例：
      DECLARE
            V_num INTEGER;
            V_sum NUMBER（8，2）；
      BEGIN
            /*改变参数传入的顺序*/
            V_sum:=get_salary(emp_count=>v_num,dept_nu=>30);
      END;
```

3. 混合表示法:

同时使用位置表示法和名称表示法，位置表示法参数必须放在名称表示法参数的前面，即使用名称表示法后不能再使用位置表示法。

例:

```
CREATE OR REPLACE FUNCTION demo_fun
    (name VARCHAR2,age INTEGER,sex VARCHAR2)
RETURN VARCHAR2 AS
    V_var VARCHAR2(32);
BEGIN
    V_var:=name||':'||TO_CHAR(age)||'岁'||sex;
    RETURN v_var;
END;
```

创建函数 demo_fun

例：函数调用

```
DECLARE
    Var VARCHAR(32);
BEGIN
    Var:=demo_fun('user1',30,sex=>'男');
    Var:=demo_fun('user2',age=>40,sex=>'男');
    Var:=demo_fun('user3',sex=>'女',age=>20);
    /*以上均正确*/
    var:=demo_fun('user1',name=>30,'男');
    /*以上调用错误*/
END;
```

存储过程

过程与函数统称为 PL/SQL 子程序，他们是命名的 PL/SQL 块，存储在数据库中，并通过输入，输出参数或输入输出参数与其调用者交换信息。函数与过程唯一区别是函数总是返回参数，而过程不返回参数。

语法:

创建过程:

```
CREATE [OR REPLACE] PROCEDURE [schema.] proc_name
    ([argument [IN|OUT [NOCOPY]|INOUT[NOCOPY]] datatype[,…]])
        [invoker_rights_clause]
    {IS|AS}
        PL/SQL_block;
```

```
DECLARE
    V_num INTEGER;
    V_sum  NUMBER(8,2);
        --建立本地过程;
    PROCEDURE proc_demo1（
        Dept_no NUMBER DEFAULT 10 , --输入参数，部门编号
        Sal_sum OUT NUMBER,--输出参数，说明部门工资总和
        Emp_count OUT INTEGER –输出函数，说明部门人数
    ）
    IS
    BEGIN
        SELECT sum(sal),count(*) INTO sal_sum,emp_count
            FROM emp WHERE deptno = dept_no;
    END proc_demo1;
BEGIN
    Proc_demo1(30,v_sum,v_num);
    Proc_demo1（sal_sum=>v_sum,emp_count=>v_num）;
        /*部门代号使用默认值 10*/
END;
```

本地过程及其调用

包

包是一组相关的过程，函数，变量，常量和游标等 PL/SQL 程序设计元素的组合，它具有面向对象程序设计语言的特点，是对这些 PL/SQL 程序设计元素的封装。包类似于 C++中的类，其变量相当于类中的成员变量，过程和函数相当于方法。

与类相同，包中的程序元素也分为共有元素和私有元素两种，这两种元素区别时允许的访问程序范围不同，即作用域不同。共有元素可以被包中的函数，过程访问，也可被包外的 PL/SQL 程序访问。私有元素只能被包内的函数合过程访问。

在 PL/SQL 程序设计中，使用包不仅可以使程序设计模块化，对外隐藏包内所使用的信息，而且可以提高程序的执行效率。以为当程序首次调用包内函数或过程时，Oracle 把整个包调进内存，当再次访问包内元素时，Oracle 直接从内存中读取。

创建包

创建包分为创建包定义和创建包体两个过程,包定义部分声明包内数据类型,变量,常量,游标,子程序,异常处理等元素,这些元素为公有元素,而包体则是包定义部分的实现,他定义了包定义部分所声明的游标和子程序,包体内还可以声明私有元素.

包定义:

语法:CREATE [OR REPLACE] PACKAGE package_name

[AUTHID {CURRENT_USER|DEFINER}]

{IS|AS}

[公有数据类型定义]

[公有游标定义]

[公有变量，常量声明]

[公有子程序声明]

END [package_name];

说明：AUTHID 参数说明应用程序在调用函数时说使用的权限模式

CURRENT_USER：当前使用者权限

DEFINER: 定义者权限

包体定义:

语法:

CREATE [OR REPLACE] PACKAGE BODY package_name{IS|AS}

[私有数据类型定义]

[私有变量，常量声明]

[公有游标定义]

[公有子程序定义]

[BEGIN 语句]

END [package_name];

删除包

删除包:$SQL>drop PACKAGE 包名

$SQL>drop PACKAGE BODY 包名

备注:包定义和包体应该同时修改,并保持一致.

应用举例

例：定义包定义部分

```
CREATE OR REPLACE PACKAGE demo_pack
        AUTHID CURRENT_USER
    IS
    Deptrec  dept%ROWTYPE;
    V_sqlcode NUMBER;
    V_sqlerr  VARCHAR2(2048);
    FUNCTION add_dept(dept_no NUMBER,dept_name VARCHAR2,location VARCHAR2)
        RETURN NUMBER;
    FUNCTION remove_dept(dept_no NUMBER)
        RETURN NUMBER;
    PROCEDURE query_dept(dept_no IN NUMBER);
END demo_pack;
```

例： 定义包体

```
CREATE OR REPLACE PACKAGE BODY  demo_pack
IS
    --声明包私有变量
    flag INTEGER;
    --声明包私有函数
    FUNCTION check_dept(dept_no NUMBER)
        RETURN INTEGER;
    --以下为公有函数
    FUNCTION add_dept(dept_no NUMBER,dept_name VARCHAR2,location
VARCHAR2)
        RETURN NUMBER
    IS
    BEGIN
        IF check_dept(dept_no) = 0 THEN
            INSERT INTO dept
                VALUES(dept_no,dept_name,location);
            RETUEN 1;
        ELSE
            RETURN;
        END IF;
```

```
--接上页
    EXCEPTION
        WHEN OTHERS THEN
            V_sqlcode:=SQLCODE;    --捕获异常信息
            V_sqlerr:=SQLERRM;
            RETURN –1;
    END add_dept;

FUNCTION remove_dept(dept_no NUMBER)
        RETURN NUMBER
    IS
    BEGIN
        V_sqlcode := 0;
        V_sqlerr :=NULL;
        IF check_dept(dept_no) = 1 THEN
            DELETE FROM dept WHERE deptno = dept_no;
            RETURN 1;
        ELSE
            RETURN 0;
        END IF;
    EXCEPTION
        WHEN OTHERS THEN
            V_sqlcode:=SQLCODE;
            V_sqlerr:=SQLERRM;
            RETURN –1;
    END remove_dept;
    PROCEDURE query_dept(dept_no IN NUMBER)
    Is
    BEGIN
        IF check_Dept(dept_no) = 1 THEN
            SELECT * INTO DeptRec FROM dept
                WHERE deptno = dept_no;
        END IF;
    END query_dept;
    FUNCTION check_dept(dept_no NUMBER)
        RETURN INTEGER
    IS
    BEGIN
        SELECT count(*) INTO flag  FROM dept
            WHERE deptno = dept_no;
        IF flag > 0 THEN flag:=1;
        END IF;
        RETURN flag;
```

例：接上页

```
    END check_dept;
BEGIN  --包体初始化，对公有变量初始化
    V_sqlcode : =NULL;
    V_sqlerr : =NULL;
END demo_pack;
```

例：调用 demo_pack 包内函数对 dept 表进行插入，查询和修改操作，通过 demo_pack 包中的记录变量 deptrec 显示查询到的信息。

```
DECLARE
    Var NUMBER;
BEGIN
    Var : = demo_pack.add_dept(90,'abc','def');
        --调用包 demo_pack 中 add_dept 函数向数据库表 dept 插入数据
    IF var = -1 THEN
        Dbms_output.put_line(demo_pack.v_sqlerr);
            --使用函数 dbms_output.put_line 输出错误信息
    ELSE
        Demo_pack.query_dept(90);
            --查询部门号为 90 信息
        Dbms_output.put_line(demo_pack.deptrec.deptno||demo_pack.deptrec.dname||
demo_pack.deptrec.loc);
        Var:=demo_pack.remove_dept(90);
            --删除部门号为 90 的信息
        IF var  = -1 THEN
            Dbms_output.put_line(demo_pack.v_sqlerr);
        END If;
    END IF;
END;
```

UTL_FILE 包的使用

Oracle 通过包 UTL_FILE 操作文件，常用过程为

文件控制：FOPEN FCLOSE IS_OPEN FCLOSE_ALL

文件输出：PUT PUT_LINE NEW_LINE PUTF FFLUSH

文件输入：GET_LINE

相关参数： init.ora 文件

UTL_FILE_DIR=目录

3.4.1 文件控制：

1． FUNCTION FOPEN(location IN VARCHAR2,filename IN VARCHAR2,open_mode IN VARCHAR2)

RETURN FILE_TYPE;

FOPEN 函数提供文件打开功能。

Location 为文件存放目录，制定的目录必须存在，FOPEN 不会自己创建不存在的目录。如果模式是'w'模式，将覆盖已存在的文件。

Filename 为文件名称

Open_mode 文件打开模式‘w’（写模式）,’r’（读模式）,’a’（附加模式）；

返回值为打开文件的句柄。

FOPEN 引发的异常现象：

UTL_FILE.INVALID_PATH 目录不存在或不可访问

UTL_FILE.INVALID_MODE

UTL_FILE.INVALID_OPERATION

UTL_FILE.INVALID_ERROR

2．FCLOSE 文件操作完毕后，使用 FCLOSE 关闭文件。

PROCEDURE FCLOSE(file_handle IN OUT FILE_TYPE);

参数说明：file_handle 关闭文件的句柄

FCLOSE 引发的异常现象：

UTL_FILE.WRITE_ERROR 写操作时发生错误

UTL_FILE.INVALID_FILEHANDLE 句柄非法

3．IS_OPEN 判断文件是否已经打开

FUNCTION IS_OPEN(file_handle IN FILE_TYPE)

RETURN BOOLEAN

参数说明：输入句柄。

返回值说明：文件打开时返回 TRUE ,否则返回 FALSE。

4．FCLOSE_ALL 关闭所有打开的文件

PROCEDURE FCLOSE_ALL

3.4.2 文件输出：

1． PUT 将指定的字符串输入到文件中，输入后不换行。

PROCEDURE PUT(file_handle IN FILE_TYPE,buffer IN VARCHAR2);

2． NEW_LINE 向文件中写入行终结束符。相当于换行。

PROCEDURE NEW_LINE(file_handle IN FILE_TYPE,lines IN NATURAL:=1);

参数说明：file_handle 文件句柄

lines 输入换行个数，默认为 1。

3． PUT_LINE 向文件写入一行。

PROCEDURE PUT_LINE(file_handle IN FILE_TYPE,buffer IN VARCHAR2);

4． PUTF 格式输出字符串，与 C 语言 printf()相似。

PROCEDURE PUTF(file_handle IN FILE_TYPE,

Format IN VARCHAR2,

Arg1 IN VARCHAR2 DEFAULT NULL,

……);

3 参数说明 format 与 C 语言 printf 一致的输出格式。比如'%s'等

5．FFLUSH :使用 PUT PUT_LINE PUTF NEW_LINE 输出的数据通常被写在缓冲区中，缓冲区满了才向文件些数据，FFLUSH 强令缓冲区中的数据立即写入文件中。

PROCEDURE FFLUSH(file_handle IN FILE_TYPE);

3.4.3 文件输入：

GET_LINE 从文件中读入一行数据。不包括新行字符

PROCEDURE GET_LINE (file_handle IN FILE_TYPE, buffer OUT VARCHAR2)

参数说明：file_handle 文件句柄

buffer 缓冲区

3.4.4 应用举例

```
DECLARE
 v_OutputFile UTL_FILE.FILE_TYPE;
 v_Name VARCHAR2(10):='scott';
BEGIN
 v_OutputFile:=UTL_FILE.FOPEN('/tmp','aaaa','w');
 UTL_FILE.PUTF(v_OutputFile,'hi there !\n my name is %s,and i am a %s maj
or.\n',v_Name,'Computer Sclence');
  UTL_FILE.FCLOSE(v_OutputFile);
END;
```

文件操作

Wrapper 应用

对于 Oracle 存储过程，包，函数或其他 PL/SQL 块，是以明码的形式存储在数据库中，视图名为 user_source,对于一些加密算法，业务逻辑等不希望对客户公开的源代码，我们可以使用 wrapper 工具，将明码转换为 16 进制码，然后再使用数据库进行编译。

wrapper 是一个操作系统可执行文件。在大多数系统中名字为 wrap.

语法:

wrap iname=input_file [oname=output_file]

iname 是包含一条 CREATE OR REPLACE 语句的文件名字，该文件可为任意扩展名，缺省为.sql。如果指定了 output_file,则 output_file 为输出文件名称。否则输入文件与输出文件名称一致，后缀为.plb.

.plb 文件与.SQL 文件一样均可以在 SQL*PLUS 等 Oracle 工具中使用。

第四章 存储过程运行环境

存储过程以及 PL/SQL 执行环境

SQL*PLUS 环境

1.在 PL*SQL 中操纵块

在 SQL*PLUS 中执行一条 SQL 语句时，应该使用分号结束该语句。分号不是该语句的一部分，他是语句终结符。当 SQL*PLUS 读到这个分号时，他便知道该语句已经结束，并把语句发送到数据库。另一方面，在 PL/SQL 块中，该分号是 PL/SQL 块的一个语法成分，而不再是结束符。当输入 DECLARE 或 BEGIN 关键字时，SQL*PLUS 会检测正在运行的是 PL/SQL 块而不是 SQL 语句，这时需要使用斜线（/）来指明 PL/SQL 块结束。

2． 替换变量

替换变量为 SQL 语句和 PL/SQL 语句块提供了与 SQL*PLUS 交互的媒介，通

过替换变量可以将参数值传入到 PL/SQL 块中。替换变量可以在 SQL 语句或 PL/SQL 语句块内的任何地方使用，包括替换数据库对象。

```
SQL> select ename,sal
  2  from &table_name     --替换变量  替换表名
  3  where deptno=&dept_no  --替换变量  替换部门代号
  4  ;
Enter value for table_name: emp   --数据库提示 输入表名 emp
old   2: from &table_name      --原语句
new   2: from emp             --新语句
Enter value for dept_no: 20       --提示输入部门代号  20
old   3: where deptno=&dept_no   --原语句
new   3: where deptno=20        --新语句

ENAME           SAL
---------- ----------
JONES          3475
SCOTT          3500
ADAMS          1600
FORD           3500
WUCHEN          3300
```

使用替换变量

3．使用联编变量：

SQL*PLUS 以可以分配内存单元，分配的内存块可以在 PL/SQL 块或 SQL 语句内部使用。该内存单元是在块的外面分配的，因此可以连续被多个块使用，并且一个块

执行完毕后可以打印该内存单元的内容，该内存单元被称为联编变量。

```
SQL> VARIABLE v_count NUMBER   --定义联编变量
SQL> BEGIN
  2    SELECT count(*)
  3          INTO :v_count   --使用联编变量时变量前应加冒号（：）
  4    FROM emp;
  5  END;
  6  /
PL/SQL procedure successfully completed.
SQL> print v_count    --打印联编变量
   V_COUNT
----------
        48
SQL>
```

使用联编变量

4 使用 EXECUTE 调用内置存储过程

语法：$>SQLPLUS scott/tiger;

$SQL>EXECUTE 存储过程名称（参数列表）；

5。使用文件

PL/SQL 块或存储过程编写完成后，通常是存放在文本文件中，我们在 SQL*PLUS 中使用以下语法在 SQL*PLUS 中执行文件。

语法一： $SQL>@文件名

语法二： $SQL>start 文件名

$SQL>/

Pro*c 预编译环境

1.在 Pro*C 程序中静态调用存储过程

语法:

EXEC SQL EXECUTE

BEGIN

存储过程名称（参数列表）/PLSQL 块

END;

END-EXEC;

参数列表中各参数应是在 Pro*C 程序中预先声明的宿主变量。

2.关于指示器变量的使用

在 C 语言中没有 NULL 的概念，在需要使用 NULL 时使用空字符串模拟字符串的 NULL ,但无法模拟 NULL 整数。因此在预编译器中提供指示变量弥补该缺陷。

指示器编量为一二位短整型变量。

语法:

EXEC SQL BEGIN DECLARE SECTION;

Short 指示器变量名称;

声明其他宿主变量;

EXEC SQL END DECLARE SECTION;

SQL 语句 ： 宿主变量[INDICATOR]：指示器变量

指示器变量紧随宿主变量之后，INDICATOR 参数指明其后为指示器变量，该参数可以省略，指示器变量前应加冒号（：）前缀。

指示器变量为 0： 宿主变量检索成功

指示器变量为-1： 返回值为 NULL

指示器变量为-2： 宿主变量过短，无法完全存放返回值。

3.编译器选项

为了编译嵌套 PL/SQL 语句块的程序，应设置一些必须预编译参数。

SQLCHECK = SEMNTICS

USERID = 用户名/口令

存储过程调试方法

4.2.1 SQL*PLUS 环境中显示错误

$SQL>show errors

show errors 显示存储过程编译时产生的错误

$SQL>select * from user_errors;

user_errors 视图存储编译存储过程是的出错信息

$SQL>select * from user_source;

user_source 视图存储存储过程的源代码。

4.2.2 插入测试表调试存储过程

方法：编写调试包，将存储过程执行时的局部变量取值插入到临时表中。通过判断局部变量的取值查找错误

简单的 debug 包：

```
CREATE OR REPLACE PACKAGE Debug AS
    /*调试过程，向临时表中插入值*/
    PROCEDURE Debug(p_description IN VARCHAR2,p_value IN VARCHAR2);
    /*过程初始化*/
    PROCEDURE Reset;
END Debug
```

创建包体:

```
CREATE OR REPLACE PACKAGE BODY Debug AS
    V_linecount NUMBER;
PEOCEDURE Debug (p_Description IN VARCHAR2,p_Value IN VARCHAR2) IS
BEGIN
    INSERT INTO debug_table(linecount,debug_str)
        VALUES(v_linecount,p_Descrition||':'||p_Value);
    COMMIT WORK;
    V_linecount：=v_lincount +1;
END Debug;

PROCEDURE Reset IS
BEGIN
    V_linecount:=1;
    DELETE FROM debug_table;
    COMMIT WORK;
END Reset;
BEGIN  /*包初始化*/
    Reset;
END Debug;
```

使用 Debug 包时，需先调用过程 Debug.Reset 对临时表初始化，然后调用过程 Debug.Debug 相临时表记录局部变量值。

4.2.3 DBMS_OUTPUT 系统内置包

PL/SQL 通过内置包 DBMS_OUTPUT 加入了输出功能。

DBMS_OUTPUT 内置包中主要包括以下几类过程：

1. PUT 类过程：输出变量值，不换行。

PROCEDURE PUT（a VARCHAR2）

PROCEDURE PUT (a NUMBER)

PROCEDURE PUT (a DATE)

2. PUT_LINE 类过程：输出变量值，输出后换行。

PROCEDURE PUT_LINE(a VARCHAR2)

PROCEDURE PUT_LINE(a NUMBER)

PROCEDURE PUT_LINE(a DATE)

PUT_LINE 相当于先调用 PUT 然后调用 NEW_LINE 过程

3. NEW_LINE 过程

PROCEDURE NEW_LINE

在缓冲区中放入新行字符，表示一行的结束。

4. GET_LINE:

PROCEDURE GET_LINE(line OUT VARCHAR2,status OUT INTEGER);

Line 参数为包含缓冲区一行的字符串，status 指明是否成功检索，一行最大长度为 255 字节。如检索成功 status 值为 0，否则为 1。

DBMS_OUTPUT 包的使用：

在 SQLPLUS 中应设置 serveroutput 参数

语法：$SQL>SET SERVEROUTPUT ON SIZE buffer_size

例： 输出测试

```
SQL> SET SERVEROUTPUT ON SIZE 2000
SQL> BEGIN
  2    DBMS_OUTPUT.PUT_LINE('打印测试');
  3  END;
  4  /
打印测试     ---输出打印结果

PL/SQL procedure successfully completed.
SQL>
```

附录一 **sql*plus** 工具

sql*plus 是 oracle 数据库管理员和普通用户最常用的实用程序,他提供一个交互式 SQL 语句,PL/SQL 语句块和 sql*plus 命令的编辑,编译和执行环境.sql*plus 作为交互式管理,操作工具,使用命令行方式实现数据的操作,数据库的管理等工作.

附录 **1.1 sql*plus** 启动和关闭

语法一: sql*plus 启动语法格式:

$> sqlplus [<option>] [logon] [@<file>[.ext] [<param1><param2>..]]

<option> 选项 {- | -? | -s[ilent]}

$>sqlplus - 显示 sqlplus 使用帮助

$>sqlplus -? 显示 sqlplus 版本号和运行时间

$>sqlplus –s[ilent] 启动过程中不显示任何信息,启动后不显示提示信息,经常在其

他程序中调用 sqlplus 时使用.

<logon>

选项 {<username>[/<passwd>][@<connect_string>]|/|/nolog

$sqlplus username/passwd

连接数据库用户 username 使用 passwd,如果不输入 passwd,sqlplus 会使用交互方式提示输入密码.

$sqlplus username/passwd@connect_string

连接远程数据库用户,connect_string 为数据库网络服务名称.

$sqlplus /

使用系统认证方式自动连接数据库,采用自动登陆方式.数据库用户名称应与系统用户名称对应.sco unix 系统 oracle7.3.2 外部用户为 ops$user_name.

$sqlplus /nolog

sqlplus 登录时不与用户建立连接,使用用户时

$sql>connect user_name 建立连接,

使用$sql > disconnect 断开连接

[@<file>[.ext][<param1><param2>…]]选项

@<file>[.ext]为 sqlplus 登录成功后立即执行该文件,文件扩展名为.ext,默认扩展名为.sql.

<param1><param2>为文件中使用的参数.

例 1

查询文件 selectdept.sql 内容为

SELECT * FROM dept WHERE deptno=&1;

执行$>sqlplus –s scott/tiger @selectdept 30

其中$>为系统提示符 –s 为 sqlplus 不显示提示信息

scott/tiger 为用户名和密码

@为文件执行命令

selectdept 为文件名称

30 为&1 使用的参数值.

语法二:关闭 sqlplus

{exit|ouit}

[success|failure|warning|n|variabe|:bindvariable]

[commit|rollback]

exit 命令或 quit 命令正常退出 sql*plus

$sql>exit 或$sql>quit

$sql>exit commit 提交 sqlplus 所作操作

$sql>exit rollback 回退 sqlplus 所作操作

附录 **1.2 sql*plus** 环境设置

语法一 修改环境变量

def[ine] [variable]|[variable=text]

$sql>def 显示已定义的环境变量

$sql>def _editor = vi 设置编辑器为 vi

$sql>def new_name = old_name

定义新环境变量,在 sql 语句中使用

例:$sql>def col(新变量名) = job(数据库表列名)

$sql>select &col from emp 与

$sql>select job from emp 等价

语法二 释放环境变量

undef[ine] 变量名

例:$sql>undef col

附录 **1.3** 设置环境参数

基本环境参数存储在数据库站点概要文件中,oracle8.1.5 站点概要文件为 $oracle_home/sqlplus/admin/glogin.sql

用户设置参数存储在用户概要文件中,默认文件名为 login.sql

语法一: set 系统变量 变量值

$sql>set feed[back] n/off/on

执行查询语句时是否显示记录统计信息 (如行数),n 为查询记录数大于 n 行时统计信息,off 不显示统计信息,on 显示统计信息.

$sql>set heading on/off

查询时是(on)否(off)打印列名

$sql>set lin[esize] n

每行显示的最多字符数,默认为 80

$sql>set newp[age] n

设置每页首行前的空行数

$sql>set null <name>

查询时返回的空值用 name 替换

$sql>set numf[ormat] format

设置数字显示格式:

$99999 数字前加$符号,长度与 9 的个数相同

999999 数字前导 0 换成空格

000000 数字前导零保留

XXXX 十六进制,字母大写 xxxx 十六进制,字母小写

99G999D99 G 位显示逗号分隔符,D 位显示小数点

L999D99 L 显示本地货币符号

$sql>set num[width]

设置数字显示宽度

$sql>set pages[ize]

设置每页显示行数

$sql>set pau[se]

设置换页时是(on)否(off)暂停

附录 **1.4 sqlplus** 命令的执行

$sql>SQL 语句 ;

分号执行命令(;) sqlplus 执行分号前的 SQL 语句

$sql>SQL 语句

$sql> /

/执行命令

$sql>SQL 语句

$sql> r(un)

run 执行命令

$sql>PL/SQL 块

$sql> .

.执行命令,PL/SQL 块遇到.命令开始执行

$sql>start file_name

使用 start 命令执行外部 SQL 语句文件

$sql>@ file_name

使用@命令执行外部 SQL 语句文件

附录 **1.5 sql*plus** 编辑命令

显示命令

$sql>L[ist]/L[ist] * 显示当前行内容

$sql>L n 显示第 n 行内容,并将第 n 行设为当前行

$sql>L n * 显示第 n 行到当前行

$sql>L m n 显示第 m 行到第 n 行

$sql>L last 显示最后一行

添加命令

$sql>A(ppend) text 在当前行尾添加 text

修改命令

$sql>C(hange) /old/new

将当前行中 old 替换为 new

$sql>C(hange) /old/

删除当前行中的 old

$sql>CL(ear) buff

清除缓冲区

$sql>del m(*) n(*)

删除行

$sql>I(nput)

在当前行后添加新行

$sql>I(nput) text

在当前行后添加文本 text