1、最简单的创建存储过程

CREATE [OR REPLACE] PROCEDURE 存储过程名[(参数[IN|OUT|IN OUT] 数据类型...)]

{AS|IS}

[说明部分]

BEGIN

可执行部分

[EXCEPTION

错误处理部分]

END [过程名];

其中：

可选关键字OR REPLACE 表示如果存储过程已经存在，则用新的存储过程覆盖，通常用于存储过程的重建。

参数部分用于定义多个参数(如果没有参数，就可以省略)。参数有三种形式：IN、OUT和IN OUT。如果没有指明参数的形式，则默认为IN。 关键字AS也可以写成IS，后跟过程的说明部分，可以在此定义过程的局部变量。(注意：大小写不敏感)。

简单的存储过程示例：

create or replace procedure hello\_world as

say\_hi varchar2(20);

begin

say\_hi := 'Hello World!';

dbms\_output.put\_line(say\_hi);

end;

2 删除存储过程

语法：

DROP PROCEDURE 存储过程名;

如：drop procedure hello\_world;

3.存储过程的查看

可以通过对数据字典的访问来查询存储过程或函数的有关信息，如果要查询当前用户的存储过程或函数的源代码，可以通过对USER\_SOURCE数据字典视图的查询得到。

3.1查看存储过程的脚本

如查询存储过程hello\_world的脚本

Select text from user\_source where name = ‘HELLO\_WORLD’;

(这里的过程名必须大写)

在PL/SQL Developer 中快速查看存储过程代码可以通过Ctrl+鼠标左键完成（与在eclipse中查看方法体代码类似）

3.2 查看存储过程的状态

如：select status from user\_objects where object\_name = ‘HELLO\_WORLD’;

说明：VALID表示该存储过程有效(即通过编译)，INVALID表示存储过程无效或需要重新编译。当Oracle调用一个无效的存储过程或函数时，首先试图对其进行编译，如果编译成功则将状态置成VALID并执行，否则给出错误信息。

当一个存储过程编译成功，状态变为VALID，会不会在某些情况下变成INVALID。结论是完全可能的。比如一个存储过程中包含对表的查询，如果表被修改或删除，存储过程就会变成无效INVALID。所以要注意存储过程和函数对其他对象的依赖关系。

4.存储过程参数说明

三种形式的参数

1. IN 定义一个输入参数变量，用于传递参数给存储过程

2. OUT 定义一个输出参数变量，用于从存储过程获取数据

3. IN OUT 定义一个输入、输出参数变量，兼有以上两者的功能

参数的定义形式和作用如下：

4.1 IN参数

语法：参数名 IN 数据类型 DEFAULT 值;

定义一个输入参数变量，用于传递参数给存储过程。在调用存储过程时，主程序的实际参数可以是常量、有值变量或表达式等。DEFAULT 关键字为可选项，用来设定参数的默认值。如果在调用存储过程时不指明参数，则参数变量取默认值。在存储过程中，输入变量接收主程序传递的值，但不能对其进行赋值。

4.2 OUT参数

语法：参数名 OUT 数据类型;

定义一个输出参数变量，用于从存储过程获取数据，即变量从存储过程中返回值给主程序。

在调用存储过程时，主程序的实际参数只能是一个变量，而不能是常量或表达式。在存储过程中，参数变量只能被赋值而不能将其用于赋值，在存储过程中必须给输出变量至少赋值一次。

4.3 IN OUT参数

语法：参数名 IN OUT 数据类型 DEFAULT 值;

定义一个输入、输出参数变量，兼有以上两者的功能。在调用存储过程时，主程序的实际参数只能是一个变量，而不能是常量或表达式。DEFAULT 关键字为可选项，用来设定参数的默认值。在存储过程中，变量接收主程序传递的值，同时可以参加赋值运算，也可以对其进行赋值。在存储过程中必须给变量至少赋值一次。

示例：

create or replace procedure say\_hi(to\_whom in varchar2 default '张三', who out varchar2)

as

who\_name varchar(20); --定义常量部分

begin

who\_name := '李四';

who := who\_name; --为输出变量赋值

dbms\_output.put\_line('Say Hi to '||to\_whom);

end;

create or replace procedure invoke\_say\_hi

as

who varchar2(20);

whom varchar2(20);

begin

whom := '小明';

say\_hi(whom, who);

say\_hi(who => who);

dbms\_output.put\_line(who||'say hi to '||whom);

end;  
运行结果：

Say Hi to 小明

Say Hi to 张三

李四Say hi to小明

参数的值由调用者传递，传递的参数的个数、类型和顺序应该和定义的一致。如果顺序不一致，可以采用以下调用方法:

过程名(参数名 => 参数的值，参数名 => 参数的值， ~~~~~);

如上面的示例： say\_hi(who => who); =>运算符左侧是参数名，右侧是参数表达式.

存储过程参数宽度：

无法在存储过程的定义中指定存储参数的宽度，也就导致了我们无法在存储过程中控制传入变量的宽度。这个宽度是完全由外部传入时决定的。

参数或变量的定义时的类型也可以是这样：

[变量名] in/out 表名.字段名%type --表示变量的类型与此表的此字段同类型

name in t\_user.name%type

5.游标

5.1 普通游标

游标定义： cursor [游标名] is [sql语句]

普通游标把整个查询已经写死,调用时不可以作任何改变

注意：这里的is不能用as代替

示例：

create or replace procedure cursor\_sample

as

user\_name varchar2(20);

user\_id varchar2(20);

cursor select\_user

is select t.user\_name, t.user\_id from t\_user t; --在定义变量时定义游标

begin

open select\_user;

loop

fetch select\_user into user\_name, user\_id;

exit when select\_user%notfound;

dbms\_output.put\_line('UserId：'||user\_id||' UserName: '||user\_name);

end loop;

close select\_user;

end;

5.2动态查询游标

动态查询游标，查询条件的参数由变量决定。

例：

create or replace procedure cursor\_sample2

as

vuser\_id varchar2(20);

vuser\_name varchar2(20);

cursor get\_user\_by\_id

is select t.user\_id, t.user\_name from t\_user t where t.user\_id = vuser\_id;

begin

vuser\_id := 'root2';

open get\_user\_by\_id;

loop

fetch get\_user\_by\_id into vuser\_id, vuser\_name;

exit when get\_user\_by\_id%notfound;

dbms\_output.put\_line('UserName:'||vuser\_name||' UserId:'||vuser\_id);

end loop;

close get\_user\_by\_id;

end;

5.3游标变量

先定义了一个引用游标类型，然后再声明了一个游标变量. 然后再用open for 来打开一个查询。需要注意的是它可以多次使用,用来打开不同的查询。

示例：

create or replace procedure cursor\_sample3

as

vuser\_name varchar2(20);

vuser\_id varchar2(20);

type cursor\_type is ref cursor;

select\_user cursor\_type;

begin

open select\_user for select t.user\_name, t.user\_id from t\_user t;

loop

fetch select\_user into vuser\_name, vuser\_id;

exit when select\_user%notfound;

dbms\_output.put\_line('UserName:'||vuser\_name||' ||UserId:'||vuser\_id);

end loop;

close select\_user;

dbms\_output.put\_line('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

dbms\_output.put\_line('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*第二次使用游标\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

open select\_user for select t.user\_name, t.user\_id from t\_user t where t.user\_id = 'root2';

loop

fetch select\_user into vuser\_name, vuser\_id;

exit when select\_user%notfound;

dbms\_output.put\_line('UserName:'||vuser\_name||' ||UserId:'||vuser\_id);

end loop;

close select\_user;

end;

5.4游标循环的方法

(1)游标的%found和%notfound属性。

能从游标中取出记录，得到的结果为%found，取不到记录为%notfound。在打开一个游标之后，马上检查它的%found或%notfound属性，它得到的结果即不是true也不是false.而是null.必须执行一条fetch语句后，这些属性才有值。

三种循环方法

5.4.1【Loop循环】

loop

fetch select\_user into vuser\_name, vuser\_id;

exit when select\_user%notfound;

【Do something】;

end loop;

close select\_user;

5.4.2【while循环】(感觉比较麻烦，不爱用这种方式)

fetch select\_user into vuser\_name, vuser\_id;

while select\_user%found loop

dbms\_output.put\_line('UserName:'||vuser\_name||' ||UserId:'||vuser\_id);

fetch select\_user into vuser\_name, vuser\_id;

end loop;

close select\_user;

说明：我们知道了一个游标打开后，必须执行一次fetch语句，游标的属性才会起作用。所以使用while 循环时，就需要在循环之前进行一次fetch动作。

而且数据处理动作必须放在循环体内的fetch方法之前。循环体内的fetch方法要放在最后。否则就会多处理一次。这一点也要非常的小心。

使用while来循环处理游标是最复杂的方法。

5.4.3【for循环】

create or replace procedure cursor\_sample4

as

vuser\_name varchar2(20);

vuser\_id varchar2(20);

cursor select\_user is select t.user\_name, t.user\_id from t\_user t;

begin

for v\_pos in select\_user loop

vuser\_name := v\_pos.user\_name;

vuser\_id := v\_pos.user\_id;

dbms\_output.put\_line('UserName:'||vuser\_name||'UserId'||vuser\_id);

end loop;

end;

说明：for循环是比较简单实用的方法。

首先，它会自动open和close游标。解决了你忘记打开或关闭游标的烦恼。

其次,自动定义了一个记录类型及声明该类型的变量，并自动fetch数据到这个变量中。 我们需要注意v\_pos 这个变量无需要在循环外进行声明，无需要为其指定数据类型。 它应该是一个记录类型，具体的结构是由游标决定的。这个变量的作用域仅仅是在循环体内。 把v\_pos看作一个记录变量就可以了，如果要获得某一个值就像调用记录一样就可以了。 如v\_pos.user\_name由此可见，for循环是用来循环游标的最好方法。高效，简洁，安全。

6.异常处理

语法：exception when [异常名] then [dosomething] …

When others then [dosomething];

示例：

create or replace procedure exception\_sample

as

vuser\_name varchar2(20);

vuser\_id varchar2(20);

vsqlcode varchar2(10);

vsqlerrm varchar2(1000);

begin

select t.user\_name, t.user\_id into vuser\_name, vuser\_id from t\_user t where 1 = 0;

exception

when others

then

vsqlcode := sqlcode;

vsqlerrm := sqlerrm;

dbms\_output.put\_line('Exception, code:'||vsqlcode||' Error message:'||sqlerrm);

end;

另外：若在代码中需要抛出异常时，用raise+异常名

如以下例子：

create or replace procedure exception\_sample2

as

vuser\_name varchar2(20);

vuser\_id varchar2(20);

yourexception exception; --自定义异常

cursor select\_user is select t.user\_name, t.user\_id from t\_user t;

begin

open select\_user;

loop

fetch select\_user into vuser\_name, vuser\_id;

exit when select\_user%notfound;

if vuser\_id = 'root2'

then raise yourexception; --抛出异常

end if;

end loop;

exception

when yourexception then

dbms\_output.put\_line('人品异常');

when others then

dbms\_output.put\_line(sqlcode||sqlerrm);

end;

已经命名的异常：

命名的系统异常 产生原因

ACCESS\_INTO\_NULL 未定义对象

CASE\_NOT\_FOUND CASE 中若未包含相应的 WHEN ，并且没有设置

ELSE 时

COLLECTION\_IS\_NULL 集合元素未初始化

CURSER\_ALREADY\_OPEN 游标已经打开

DUP\_VAL\_ON\_INDEX 唯一索引对应的列上有重复的值

INVALID\_CURSOR 在不合法的游标上进行操作

INVALID\_NUMBER 内嵌的 SQL 语句不能将字符转换为数字

NO\_DATA\_FOUND 使用 select into 未返回行，或应用索引表未初始化的

TOO\_MANY\_ROWS 执行 select into 时，结果集超过一行

ZERO\_DIVIDE 除数为 0

SUBSCRIPT\_BEYOND\_COUNT 元素下标超过嵌套表或 VARRAY 的最大值

SUBSCRIPT\_OUTSIDE\_LIMIT 使用嵌套表或 VARRAY 时，将下标指定为负数

VALUE\_ERROR 赋值时，变量长度不足以容纳实际数据

LOGIN\_DENIED PL/SQL 应用程序连接到 oracle 数据库时，提供了不

正确的用户名或密码

NOT\_LOGGED\_ON PL/SQL 应用程序在没有连接 oralce 数据库的情况下

访问数据

PROGRAM\_ERROR PL/SQL 内部问题，可能需要重装数据字典& pl./SQL

系统包

ROWTYPE\_MISMATCH 宿主游标变量与 PL/SQL 游标变量的返回类型不兼容

SELF\_IS\_NULL 使用对象类型时，在 null 对象上调用对象方法

STORAGE\_ERROR 运行 PL/SQL 时，超出内存空间

SYS\_INVALID\_ID 无效的 ROWID 字符串

TIMEOUT\_ON\_RESOURCE Oracle 在等待资源时超时

7.过程内部块

我们知道了存储过程的结构，语句块由begin开始，以end结束。这些块是可以嵌套。在语句块中可以嵌套任何以下的块：

Declare … begin … exception … end;

例：

create or replace procedure innerblock\_sample

as

vuser\_name varchar2(20);

vuser\_id varchar2(20);

cursor select\_user is select t.user\_name, t.user\_id from t\_user t;

begin

open select\_user;

loop

fetch select\_user into vuser\_name, vuser\_id;

Exit when select\_user%notfound;

dbms\_output.put\_line('UserName:'||vuser\_name||'UserId'||vuser\_id);

end loop;

close select\_user;

declare

vuser\_level varchar2(20);

cursor get\_level is select t.user\_level from t\_user t;

begin

open get\_level;

loop

fetch get\_level into vuser\_level;

exit when get\_level%notfound;

dbms\_output.put\_line('UserLevel:'||vuser\_level);

end loop;

close get\_level;

exception when others then

dbms\_output.put\_line(sqlcode||sqlerrm);

end;

exception when others then

dbms\_output.put\_line(sqlcode||sqlerrm);

end;