1. **游标**
2. 隐式游标

隐式游标无需定义（DECLARE）游标。直接在BEGIN END；块中添加程序段。系统会为其中的语句自动创建设置游标。隐式游标的名字为SQL ，这是Oracle定义好了的。可以通过SQL%\_\_\_\_\_来获取程序块中语句的执行结果的状态。

注:SQL%\_\_\_\_获取的是最新一条一句执行完的状态。

SQL%ROWCOUNT 整型 代表DML语句成功执行的数据行数

SQL%FOUND 布尔型 值为TRUE代表插入、删除、更新或单行查询操作成功

SQL%NOTFOUND 布尔型 与SQL%FOUND属性返回值相反

SQL%ISOPEN 布尔型 DML执行过程中为真，结束后为假

DML操作和单行SELECT语句会使用隐式游标，它们是：

\* 插入操作：INSERT。

\* 更新操作：UPDATE。

\* 删除操作：DELETE。

\* 单行查询操作：SELECT ... INTO ...。eg:

Declare

Sname varchar(20);

Begin

Select name into sname from student; --注意into语句在select和from中间

……….

END

1. 显式游标

显示游标使用之前需要定义游标。格式如:

declare

CURSOR mycur IS -----定义变量

SELECT \* from student;

Myrecord mycur%ROWTYPE -------- 基于游标定义记录变量

Begin -------开始

Open mycur； --------打开游标 注意尾部的；

…………..对游标的操作

Close mycur; --------关闭游标，注意结尾的;

END; --------结束 ，注意结尾的;

游标中的变量

游标中使用变量需要提前在DECLARE语句中声明，例如:

DECLARE

 Studentno number(3); -------只定义，没有赋值 Studentno number(3):=100; ----定义并初始化赋值

注意 :=为赋值， = 为比较是否相等

游标中的分支

If 表达式 then ---游标与存储过程中的if后面的表达式不带()

elsif -----注意是 elsif 不是 elseif what the f\*\*\*

End if; ------注意 尾部的；

游标中的循环

1.for 循环

declare

i number(3);

begin

i:=1;

for i in 1..101 loop ----- 非常简洁的便利的方式

DBMS\_OUTPUT.PUT\_LINE(i); -----输出1到101 注 不用自己i:=i+1

end loop; -----这种for循环是程序自己实现i++的

end;

declare

cursor mycur is (select \* from student);

myrec mycur%rowtype;

begin

for myrec in mycur loop ----遍历游标，不用自己打开游标，也不用关闭游标，直接使用， 贼几把方便

DBMS\_OUTPUT.PUT\_LINE(myrec.no||'---'||myrec.name);

end loop;

end;

2,直接loop循环

declare

i number(3);

begin

i:=1;

loop -----直接loop 开启循环

exit when i=100; ------在loop中添加判断退出的语句

DBMS\_OUTPUT.PUT\_LINE(i);

i:=i+1;

end loop;

end;

几条重要的语句:

DBMS\_OUTPUT.PUT\_LINE( ); ---该语句有点类似Java中的System.out.println();

里面输出的字符串拼接的时候 使用 || 类似于java中的+

fetch mycur into myrecord; ---移动游标读取下一条，游标打开时候是处于第0行的

exit when mycur%notfound; ----退出循环的方法

1. **存储过程**

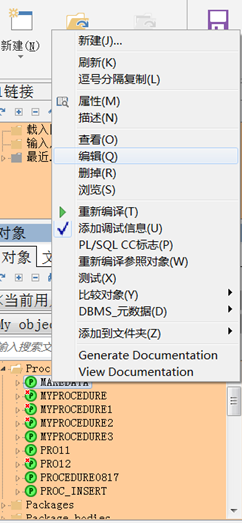
存储过程的优点:

1.执行效率很高，因为存储过程是预编译的，即创建时编译，而SQL语句是执行一次，编译一次。调用存储过程可以大大减少同数据库的交互次数。

2.降低网络通信量，因为存储过程执行的时候，只需要call存储过程名，不需要传递大量的SQL语句。

3.有利于复用

注:自己在pl/sql中写存储过程的时候很容易出错，调用时候报错也不详细，很难找到bug。此时应该打开调试工具。如下:

 然后就可以看到具体的报错信息了。方便修改。

1.基本结构

CREATE OR REPLACE PROCEDURE 存储过程名字 ( -----定义需要传入的参数

参数1 IN NUMBER, ----是number 不是number(n) IN表示输入 还有OUT类

参数2 IN varchar2 ----同上，千万不要定义大小，不然错都不知道错哪里

)

IS

变量1 INTEGER :=0; -----声明一些变量

变量2 DATE;

CURSOR

BEGIN

Do something;

END 存储过程名字;

存储过程若不带输入输出参数 则定义的时候 省去空括号()。 调用的时候直接名字调用，不要加后面的括号.

---------------------------------------------------简单举例--------------------------------------------------------------

create or replace procedure MYPROCEDURE0817 (

input in number, -----参数中的数据类型不带大小

output out varchar2

)

IS

str varchar(30); -----但是定义自定义变量时候需定义大小

begin

select a.name into output from student a where a.no=input;

DBMS\_OUTPUT.PUT\_LINE(output);

end MYPROCEDURE0817; ------以上为创建存储过程

declare ----写个游标调用存储过程

output1 varchar2(30);

begin

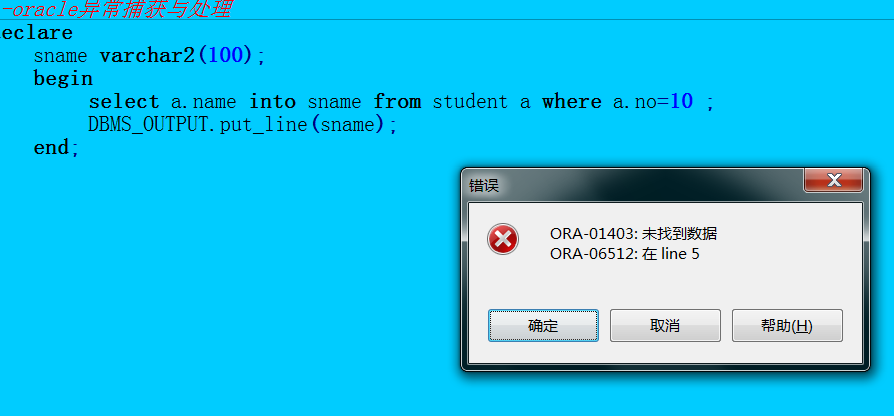
MYPROCEDURE0817(4,output1);

DBMS\_OUTPUT.PUT\_LINE(output1||' from cursor');

end;

1. **oracle错误捕获机制**

如果在程序的子块中发生了错误，但子块没有错误处理部分，则错误会传递到主程序中。例如没有查找到数据的时候，就会这样：



所以为了不影响程序的正常运行，需要错误捕处理机制。

1、Oralce中预定义的一些错误:

错 误 名 称 错误代码 错 误 含 义

CURSOR\_ALREADY\_OPEN ORA\_06511 试图打开已经打开的游标

INVALID\_CURSOR ORA\_01001 试图使用没有打开的游标

DUP\_VAL\_ON\_INDEX ORA\_00001 保存重复值到惟一索引约束的列中

ZERO\_DIVIDE ORA\_01476 发生除数为零的除法错误

INVALID\_NUMBER ORA\_01722 试图对无效字符进行数值转换

ROWTYPE\_MISMATCH ORA\_06504 主变量和游标的类型不兼容

VALUE\_ERROR ORA\_06502 转换、截断或算术运算发生错误

TOO\_MANY\_ROWS ORA\_01422 SELECT…INTO…语句返回多于一行的数据

NO\_DATA\_FOUND ORA\_01403 SELECT…INTO…语句没有数据返回

TIMEOUT\_ON\_RESOURCE ORA\_00051 等待资源时发生超时错误

TRANSACTION\_BACKED\_OUT ORA\_00060 由于死锁，提交失败

STORAGE\_ERROR ORA\_06500 发生内存错误

PROGRAM\_ERROR ORA\_06501 发生PL/SQL内部错误

NOT\_LOGGED\_ON ORA\_01012 试图操作未连接的数据库

LOGIN\_DENIED ORA\_01017 在连接时提供了无效用户名或口令

**注:**

凡是出现在WHEN后面的错误都是可以捕捉到的错误，其他未被捕捉到的错误，将在WHEN OTHERS部分进行统一处理，OTHENS必须是EXCEPTION部分的最后一个错误处理分支。

一个简单的异常捕获例子:

--oracle异常捕获与处理

declare

sname varchar2(100);

begin

select a.name into sname from student a where a.no=10 ; ---只存在1-5的数据

DBMS\_OUTPUT.put\_line(sname);

exception

when NO\_DATA\_FOUND then

DBMS\_OUTPUT.PUT\_LINE('不存在这条数据');

when OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('I dont know');

end;

2、自定义异常

（1）如果一个系统错误没有在标准包中定义，则需要在说明部分定义，语法如下：

Declare

MyException EXCEPTION ------ 定义异常

PRAGMA EXCEPTION\_INIT(MyException,ORA-01400); ------ 绑定异常

BEGIN

………

EXCEPTION

WHEN MyException then

.....

WHEN OTHERS then

….

END

还有一种方式: RAISE\_APPLICATION\_ERROR(-1400, '失败');

不需要定义与声明，达到上面代码一样的目的。

注：可以通过SQLCODE()和SQLERRM()方法在DBMS\_OUTPUT输出的时候输出来查看具体信息。

(2)纯自定义异常

用户定义的错误不能由系统来触发，必须由程序显式地触发 eg:

declare

myexcp1 exception;

begin

insert into student values(11,'11');

if SQL%found then

commit;

raise myexcp1; ----------显示触发自定义异常

end if;

insert into student values(12,'12'); ----------触发异常之后会转入Exception程序段，剩下的程序段就不执行了，12是插入不到数据当中的。

if SQL%found then

commit;

raise myexcp1;

end if;

exception

when myexcp1 then

DBMS\_OUTPUT.PUT\_LINE('触发错误');

end;

一个触发异常 并转移错误信息到自己创建的异常表的eg:

declare

myexcp1 exception;

err\_code varchar2(20);

err\_des varchar2(100);

begin

insert into student values(13,null);

exception

when others then

err\_code:=SQLCODE();

err\_des:=SQLERRM();

insert into ERRORTEST values(ERRORTEST$SEQ.nextval,err\_code,err\_des);

DBMS\_OUTPUT.PUT\_LINE('触发错误,信息已转入errortest表中');

end;