**pl/sql的介绍(未学习)**

为什么需要pl/sql编程？

因为使用纯的sql语句来操作数据库，有先天性的技术缺陷：

1、不能模块编程；

2、执行速度慢；

3、安全性有问题；

4、浪费带宽。

pl/sql是什么？

**pl/sql(procedural language/sql)是oracle在标准的sql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，允许使用例外处理各种错误，这样使得它的功能变得更加强大。**

学习必要性

1、提高应用程序的运行性能；

2、模块化的设计思想[分页的过程，订单的过程，转账的过程...]；

3、减少网络传输量；

4、提高安全性。

缺点:移植性不好。

sqlplus开发工具

sqlplus是oracle公司提供的一个工具。

**举一个简单案例：**

**编写一个存储过程，该过程可以向某表中添加记录。**

创建add\_emp存储过程：

create procedure add\_emp is

begin

insert into emp (empno,ename) values(4444,'4444');

end;

/

执行：

exec add\_emp;

pl/sql developer开发工具

pl/sql developer是用于开发pl/sql块的集成开发环境(IDE)，他是一个独立的产品，而不是oracle的一个附带品。

**举一个简单案例：**

**编写一个存储过程，该过程可以删除某表记录。**

创建删除del\_emp过程

create procedure del\_emp(in\_empno number) is

begin

delete from emp where empno=in\_empno;

end;

/

执行过程：

exec del\_emp(4444);

**创建过程基本语法：**

create procedure 过程名(参数1,...)

is

begin

执行语句;

end;

/

**执行过程语法:**

exec 过程名(传入参数,...)

**pl/sql基础知识--介绍**

开发人员使用pl/sql编写应用模块时，不仅需要掌握sql语句的编写方法，还要掌握pl/sql语句及语法规则。pl/sql编程可以使用变量和逻辑控制语句，从而可以编写非常有用的功能模块。

比如：分页存储过程模块、订单管理存储过程模块、转账存储过程模块。而且如果使用pl/sql编程，我们可以轻松的完成非常复杂的查询要求。

**pl/sql简单分类**

**块(编程)包含：**

**过程(存储过程)、函数、触发器、包**

**pl/sql基础知识--编写规范**

**1、注释**

单行注释：--

多行注释：/\*...\*/来划分

**2、标识符号的命名规范**

1)当定义变量时，建议用v\_作为前缀；如：v\_sal

2)当定义常量时，建议用c\_作为前缀；如：c\_rate

3)当定义游标时，建议用\_cursor作为后缀；如：emp\_cursor

4)当定义例外时，建议用e\_作为前缀；如：e\_error

**pl/sql块介绍**

块(block)是pl/sql的基本程序单元，编写pl/sql程序实际上就是编写pl/sql块。要完成相对简单的应用功能，可以只需要编写一个pl/sql块；但是如果要想实现复杂的功能，可能需要在一个pl/sql块中嵌套其它的pl/sql块。

块结构示意图：

**pl/sql块由三个部分构成：定义部分、执行部分、例外处理部分。**

如下所示：

**declare**

/\*定义部分----定义常量、变量、游标、例外、复杂数据类型\*/

**begin**

/\*执行部分----要执行的pl/sql语句和sql语句\*/

**exception**

/\*例外处理部分----处理运行的各种错误\*/

**end;**

**重要说明：**

1、定义部分是从declare开始的，该部分是可选的；

**2、执行部分是从begin开始的，该部分是必需的；**

3、例外处理部分是从exception开始的，该部分是可选的。

**pl/sql实例：只包括执行部分的pl/sql块**

**案例：输出hello,world**

相关说明：**dbms\_output**是oracle所提供的包(类似java的开发包)，该包包含一些过程，put\_line就是dbms\_output的一个过程。

--开发一个只包括执行部分的block块

set serveroutput on;

begin

dbms\_output.put\_line('hello,world');

end;

/

**特别说明：在默认情况下，dbms\_output.put\_line是不输出内容的，需要set serveroutput on;才可输出。**

**实例2：包含定义部分和执行部分的pl/sql块**

**案例：根据用户输入的雇员编号，显示该雇员的名字**

相关说明：

&表示要接收从控制台输入的变量

||表示把两个串拼接起来

declare

--定义变量的格式：变量名称 变量类型

v\_ename varchar2(64);

begin

select ename into v\_ename from emp where empno=&empno;--把查询到的ename放到v\_ename变量中

--输出v\_ename

dbms\_output.put\_line('雇员名字：'||v\_ename);

end;

/

将上面的块改为过程

create procedure pro3(in\_empno number) is

v\_ename varchar2(64);

begin

select ename into v\_ename from emp where empno=in\_empno;

dbms\_output.put\_line('雇员名字：'||v\_ename);

end;

/

**实例3--包含定义部分、执行部分和例外处理部分**

**为了避免pl/sql程序的运行错误，提高pl/sql的健壮性，应该对可能的错误进行处理，这个很有必要：**

1、比如在实例2中，如果输入了不存在的雇员号，应当做例外处理；

2、有时出现异常，希望用另外的逻辑处理。比如，如果不存在就加入编号为1，名字为“马大哈”这么一个人。

我们看看如何完成1的要求：

相关说明：oracle事先预定义了一些例外，no\_data\_found就是找不到数据的例外。

可以查看pl/sql官方文档看看oracle提供了哪些例外。

declare

v\_ename varchar2(64);

begin

select ename into v\_ename from emp where empno=&empno;

dbms\_output.put\_line('雇员名字：'||v\_ename);

exception

when no\_data\_found then

dbms\_output.put\_line('你查询的雇员信息不存在!');

end;

/

对该案例的细节说明：

这里我们涉及到异常处理，

**异常的基本语法：**

**exception**

**when 异常的名称1 then**

**//对异常处理的代码**

**when 异常的名称2 then**

**//对异常处理的代码**

**end;**

**异常处理的作用：**

1、可以捕获异常，可以给出明确提示；

2、有时可以利用异常来进行业务处理。

declare

v\_ename varchar2(64);

begin

select ename into v\_ename from emp where empno=&empno;

dbms\_output.put\_line('雇员名字：'||v\_ename);

exception

when no\_data\_found then

dbms\_output.put\_line('你查询的雇员信息不存在!加入一条信息');

isnert into emp (empno,ename) values(1,'马大哈');

end;

/

java捕获异常

try{

//如果用户输入字串

int abc=Integer.parse(str);

}catch(Exception e){

//对不起你输入的字串不是一个数

}



**pl/sql基础知识--过程快速入门**

**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sqlplus中可以使用create procedure命令来建立过程。

实例如下：

1、请考虑编写一个过程，可以输入雇员名，新工资可修改雇员的工资

2、如何**调用过程有两种方法：**

**exec 过程名(参数值,..)**

**call 过程名(参数值,..)**

**创建存储过程基本语法：**

**create or replace procedure 过程名(变量 in 变量类型,..,变量 out 变量类型) is**

**//定义变量**

**begin**

**//执行语句;**

**end;**

**/**

**特别说明：**or replace在创建存储过程中可带也可不带。带or replace是指在存储过程名字相同时将其覆盖。不带则无法覆盖。在使用or replace时要小心，建议不使用or replace对原存储过程进行覆盖。

举例：请考虑编写一个过程，可以输入雇员名，新工资可修改雇员的工资

create procedure update\_sal(in\_name in varchar2,in\_new\_sal in number) is

begin

update emp set sal=in\_new\_sal where ename=in\_name;

dbms\_output.put\_line('更新成功!');

end;

/

**特别说明：**当编写过程出现错误时，查看具体错误信息。输入**show error;**

**java中调用过程**

3、如何在java程序中调用一个存储过程？

动手体验：我们写一个java程序来调用前面的存储过程。

课堂小练习：编写一个过程，可以接受id和薪水，更新薪水，如果id不存在，需要在exception中捕获，并给出提示!

如何使用过程返回值？

特别说明：对于过程我们会在以后给大家详细具体的介绍，现在请大家先有一个概念。

Java代码:

**package** com.test;

**import** java.sql.CallableStatement;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**public** **class** TestProcedure {

//调用oracle中update\_sal存储过程

**public** **static** **void** main(String[] args) {

Connection ct=**null**;

CallableStatement cs=**null**;

**try** {

//加载驱动

Class.*forName*("oracle.jdbc.driver.OracleDriver");

//得到连接

ct=DriverManager.*getConnection*("jdbc:oracle:thin:@127.0.0.1:1521:orcl","scott", "tiger");

//创建CallableStatement接口引用对象

cs=ct.prepareCall("{call update\_sal(?,?)}");

//给？赋值

cs.setString(1, "BOSS");

cs.setFloat(2, 8888f);

//执行我们的语句

cs.execute();

//提交

ct.commit();

} **catch** (Exception e) {

e.printStackTrace();

}**finally**{

**try** {

**if**(cs!=**null**){

cs.close();

}

**if**(ct!=**null**){

ct.close();

}

} **catch** (Exception e2) {

e2.printStackTrace();

}

cs=**null**;

ct=**null**;

}

}

}

**对前面的java程序的SQLHelper类进行升级，添加一个可以调用存储过程的方法**

代码如下：

**private** **static** CallableStatement *cs*=**null**;

//调用存储过程的方法

**public** **static** **void** executeProcedure(String sql,String [] parameters){

**try** {

*ct*=DriverManager.*getConnection*(*url*,*username*,*password*);

*cs*=*ct*.prepareCall(sql);

**if**(parameters!=**null**){

**for**(**int** i=0;i<parameters.length;i++){

*cs*.setString(i+1, parameters[i]);

System.*out*.println(parameters[i]);

}

}

//执行

*cs*.execute();

} **catch** (Exception e) {

e.printStackTrace();

**throw** **new** RuntimeException(e.getMessage());

}**finally**{

*close*(*rs*, *cs*, *ct*);

}

}

调用方法：

//当我们需要去调用过程的时候传SQL语句

String sql="{call update\_sal(?,?)}";

String paras[]={"BOSS","1520"};

SQLHelper.*executeProcedure*(sql, paras);

课堂小练习：编写一个过程，可以接受id和薪水，更新薪水，如果id不存在，需要在exception中捕获，并给出提示!需要在控制台和java程序中都调用。

oracle控制台

create procedure update\_sal2(in\_empno in number,in\_new\_sal in number) is

v\_ename varchar2(32);

begin

select ename into v\_ename from emp where empno=in\_empno;

update emp set sal=in\_new\_sal where empno=in\_empno;

dbms\_output.put\_line('更新成功!');

exception

when no\_data\_found then

dbms\_output.put\_line('输入的ID不存在!');

end;

/

**pl/sql基本知识--函数快速入门**

oracle函数

**函数用于返回特定的数据，当建立函数时，在函数头部必须包含return子句，而在函数体内必须包含return语句返回的数据。**我们可以使用create function来建立函数，实际案例：

**建立函数的基本语法：**

**create function 函数名(参数1,...)**

**return 数据类型 is**

**定义变量;**

**begin**

**执行语句;**

**end;**

**/**

**函数调用的基本语法：**

**var 变量名 变量类型**

**call 函数名(参数值,...) into :变量名;**

**print 变量名**

**select 函数名(参数,...) from dual;**

案例：请编写一个函数，可以接收用户名并返回该用户的年薪。

create function inName\_outSal(v\_in\_name varchar2)

return number is

v\_annual\_sal number;

begin

select (sal+nvl(comm,0))\*13 into v\_annual\_sal from emp where ename=v\_in\_name;

return v\_annual\_sal;

end;

/

**函数和过程的区别：**

**1、函数必须有返回值，而过程可以没有；**

**2、函数和过程在java中调用的方式不一样；**

**java中调用oracle函数可以在select语句中直接调用，如：select 自定义的函数名(参数) from 表;**

**过程则是使用CallableStatement完成调用。**

Java调用函数方式

**package** com.test;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**public** **class** TestFunction {

//如何在java中调用自己编写的函数

**public** **static** **void** main(String[] args) {

String sql="select inName\_outSal('KING') annual from dual";

ResultSet rs=SQLHelper.*executeQuery*(sql, **null**);

**try** {

**if**(rs.next()){

System.*out*.println(rs.getDouble("annual"));//此处可以用数字或别名接收返回值

}

} **catch** (SQLException e) {

e.printStackTrace();

}**finally**{

**if**(rs!=**null**){

**try** {

rs.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

rs=**null**;

}

}

}

**pl/sql基本知识--包**

包用于在逻辑上组合过程和函数，它由包规范和包体两部分组成。

1、我们可以**使用create package命令来创建包**：

**建包基本语法：**

**create [or replace] package 包名 is**

**procedure 过程名(变量名 变量类型,...);**

**function 函数名(变量名 变量类型,...) return 数据类型;**

**end;**

**/**

**包的规范**只包含了过程和函数的说明，但是没有过程和函数的实现代码。**包体**用于实现包规范的过程和函数。

请编写一个包，该包有一个过程，该过程可以接收用户名和新的薪水。(将来用于通过用户去更新薪水)还有一个函数，该函数可以接收一个用户名(将来要实现得到该用户的年薪是多少)

create package emp\_package is

procedure update\_sal(v\_in\_ename varchar2,v\_in\_newsal number);

function inName\_outSal(v\_in\_name varchar2) return number;

end;

2、**建立包体可以使用create package body 命令**

**建立包体基本语法：**

**create or replace package body 包名 is**

**procedure 过程名(变量名 变量类型,...) is**

**--声明变量;**

**begin**

**--执行语句;**

**exception**

**when 异常名 then**

**--异常处理;**

**end;**

**function 函数名(变量名 变量类型,...)**

**return 数据类型 is**

**--声明变量;**

**begin**

**--执行语句;**

**end;**

**end;**

**/**

案例：请实现前面定义的包中的过程和函数。

create or replace package body emp\_package is

procedure update\_sal(v\_in\_ename varchar2,v\_in\_newsal number) is

v\_empno number;

begin

select empno into v\_empno from emp where ename=v\_in\_ename;

update emp set sal=v\_in\_newsal where ename=v\_in\_ename;

dbms\_output.put\_line('员工号为：'||v\_empno||'的薪水更新成功');

exception

when no\_data\_found then

dbms\_output.put\_line('您输入的人员信息不存在!');

end;

function inName\_outSal(v\_in\_name varchar2)

return number is

v\_annual\_sal number;

begin

select (sal+nvl(comm,0))\*13 into v\_annual\_sal from emp where

ename=v\_in\_name;

return v\_annual\_sal;

end;

end;

/

**细节说明：**

1、包体中要现实的函数或过程，应当在包规范中声明；

2、在调用包中的某个函数或过程的时候，需要使用对应的方法才可以调用。

3、如何调用包的过程或函数

当调用包的过程或是函数时，在过程和函数前需要带有包名，如果要访问其它方案的包，还需要在包名前加方案名。

**调用基本方法：**

**exec 方案名.包名.过程名(参数,...);**

**call 方案名.包名.函数名(参数,...);**

**也可以直接用select 方案名.包名.函数名(参数,...) from dual;**

**在java中调用oracle包下的过程和函数与之前java调用的过程和函数是一致的，只是尽可能的将方案名.包名加上。**

**String sql="{call scott.emp\_package.update\_sal(?,?)}";//过程**

**String sql="select scott.emp\_package.inName\_outSal('KING') annual from dual";//函数**

**特别说明：包是pl/sql中非常重要的部分，在使用过程分页时，将会再次体验它的威力。**

**pl/sql基础知识--触发器**

触发器简单介绍

**触发器是指隐含的执行的存储过程。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 来建立触发器。**

特别说明：

我们会在后面详细为大家介绍触发器的使用，因为触发器是非常有用的，可维护数据库的案例和一致性。

**pl/sql基础知识--定义并使用变量**

介绍

在编写pl/sql程序时，可以定义变量和常量；在pl/sql程序中包括有：

1、标量类型(scalar)

2、复合类型(composite)

3、参照类型(reference)

4、lob(large object)

**标量(scalar)--常用类型**

在编写pl/sql块时，如果要使用变量，需在定义部分定义变量。

**pl/sql中定义变量和常量的语法如下：**

identifier [constant] datatype [not null] [:=|default expr]

名称 [指定常量] 数据类型 [不为null] [:=(赋初值) | default(默认值) expr(指定初始值)]

**说明：**

identifier:名称

constant:指定常量。需要指定它的初始值，且其值是不能改变的。

datatype:数据类型

not null:指定变量值不能为null

:=给变量或是常量指定初始值

default:用于指定初始值

expr:指定初始值的pl/sql表达式，可是文本值、其它变量、函数等。

标量定义的案例：

1、定义一个变长字符串

v\_ename varchar2(10);

2、定义一个小数范围-9999.99～9999.99

v\_sal number(6,2);

3、定义一个小数并给一个初始值为5.4 :=是pl/sql的赋值号

v\_sal2 number(6,2):=5.4

4、定义一个日期类型的数据

v\_hiredate date;

5、定义一个布尔变量，不能为空，初始值为false

v\_valid boolean not null default false;

**特别说明：**pl/sql在定义一个变量的时候，如果要赋初值，则需要使用:=，如果只是=则是用于判断两个值是否相等。

标量(scalar)使用标量

在定义好变量后，就可以使用这些变量。这里需要说明的是pl/sql块为变量赋值不同于其它的编程语言，需要在等号前加冒号(:=)

案例：以输入员工号，显示雇员姓名、工资、个人所得税(税率为0.03)为例。说明变量的使用，看看如何编写？

create or replace procedure emp\_info(in\_empno number) is

v\_ename varchar2(32);

v\_sal number;

v\_tax number;

c\_tax\_rate number(3,2):=0.03;

begin

select ename,sal,sal\*c\_tax\_rate into v\_ename,v\_sal,v\_tax from emp where empno=in\_empno;

dbms\_output.put\_line('姓名：'||v\_ename||' 工资：'||v\_sal||' 个人所得税：'||v\_tax);

exception

when no\_data\_found then

dbms\_output.put\_line('你的输入有误!');

end;

/

**标量(scalar)--使用%type类型**

对于上面的pl/sql块有一个问题：

就是如果员工的姓名超过了5字符的话，就会有错误，为了降低pl/sql程序的维护工作量，可以使用%type属性定义变量，这样它会按照数据库列来确定你定义的变量的类型和长度。

我们看看这个怎么使用：

**%type类型使用的基本语法：**

**标识符名 表名.列名%type;**

create or replace procedure emp\_info(in\_empno number) is

v\_ename emp.ename%type;--为了让v\_ename的类型更加灵活，我们使用%type，这样就会自适应

v\_sal emp.sal%type;

v\_tax number;

c\_tax\_rate number(3,2):=0.03;

begin

select ename,sal,sal\*c\_tax\_rate into v\_ename,v\_sal,v\_tax from emp where empno=in\_empno;

dbms\_output.put\_line('姓名：'||v\_ename||' 工资：'||v\_sal||' 个人所得税：'||v\_tax);

exception

when no\_data\_found then

dbms\_output.put\_line('你的输入有误!');

end;

/

**复合变量(composite)--介绍**

**用于存放多个值的变量。常用的包括：1、pl/sql记录；2、pl/sql表**

**复合类型--pl/sql记录**

**类似与高级语言中的结构体，需要注意的是，当引用pl/sql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)**如下：

**复合变量定义基本语法：**

**type 自定义的pl/sql记录名 is record(**

**变量名 变量类型,**

**变量名 变量类型**

**);**

//使用自定义的pl/sql记录

**复合变量基本使用语法：**

**变量名 自定义的pl/sql记录名;**

请编写一个过程，该过程可以接收一个用户编号，并显示该用户的名字，薪水，工作岗位(注意：要求用pl/sql记录实现)

create or replace procedure inEmpno(in\_empno number) is

--定义一个记录数据类型

type my\_emp\_record is record(

v\_ename emp.ename%type,

v\_sal emp.sal%type,

v\_job emp.job%type

);

--定义一个变量，该变量的类型是my\_emp\_record

v\_emp\_record my\_emp\_record;

begin

select ename,sal,job into v\_emp\_record from emp where empno=in\_empno;

dbms\_output.put\_line('名字：'||v\_emp\_record.v\_ename||' 工资：'||v\_emp\_record.v\_sal||' 职位：'||v\_emp\_record.v\_job);

exception

when no\_data\_found then

dbms\_output.put\_line('你的输入有误!');

end;

/

**复合类型--pl/sql表(了解即可)**

**相当于高级语言中的数组，但是需要注意的是在高级语言中数组的下标不能为负数，而pl/sql是可以为负数的，并且表元素的下标没有限制。(可以理解为是oracle下的数组)**实例如下:

复合类型pl/sql表的基本语法：

type 自定义的pl/sql表名 is table of 对应表.列名%type

index by binary\_integer;

//使用自定义的pl/sql表

变量名 自定义的pl/sql表名;

declare

type sp\_table\_type is table of emp.ename%type

index by binary\_integer;

sp\_table sp\_table\_type;--定义一个变量：sp\_table类型

begin

select ename into sp\_table(-1) from emp where empno=7788;

dbms\_output.put\_line('员工名：'||sp\_table(-1));

end;

说明：

sp\_table\_type 是pl/sql表类型

emp.ename%type 指定了表的元素的类型和长度

sp\_table 为pl/sql表变量

sp\_table(0) 则表示下标为0的元素

**参照变量--介绍(重点，必须掌握)**

参照变量是指用于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，从而降低占用的空间。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。

**游标变量**

**通过游标可以取得返回结果集(这个结果集，往往是select语句的结果)的任何一行数据，从而提供共享的效率。**

参照变量--游标(ref cursor)使用

**定义游标基本语法：**

**type 自定义游标名 is ref cursor;**

**变量名 自定义游标名;**

**打开游标基本语法：**

**open 游标变量 for select 语句;**

**取出当前游标指向的行基本语法：**

**fetch 游标变量 into 其它变量;**

**判断游标是否指向记录最后基本语法：**

**游标变量%notfound**

**参照变量--游标(ref cursor)变量**

**使用游标时，当定义游标时不需要指定相应的select语句，但是当使用游标时，(open时)需要指定select语句，这样一个游标就写一个select语句结合了。**

**实例如下：**

1、请使用pl/sql编写一个过程，可以输入部门号，并显示该部门所有员工姓名和他的工资。

create or replace procedure test\_cursor(in\_deptno in number) is

--定义一个记录数据类型

type my\_emp\_record is record(v\_ename emp.ename%type,v\_sal emp.sal%type);

--定义一个变量，该变量的类型是my\_emp\_record

v\_emp\_record my\_emp\_record;

--先定义一个游标变量类型

type my\_emp\_cursor is ref cursor;

--定义一个游标变量

v\_emp\_cursor my\_emp\_cursor;

begin

--打开游标，执行语句

open v\_emp\_cursor for select ename,sal from emp where deptno=in\_deptno;

--取出游标指向的每行数据，使用循环语句取出

loop

fetch v\_emp\_cursor into v\_emp\_record;--会引起游标向下走

--判断当前游标是否到达最后

exit when v\_emp\_cursor%notfound;--判断游标是否为空，为空时退出循环

dbms\_output.put\_line('姓名：'||v\_emp\_record.v\_ename||' 工资：'||v\_emp\_record.v\_sal);

end loop;

--关闭游标[游标使用完后，一定要关闭，避免资源浪费]

close v\_emp\_cursor;

end;

/

2、在1基础上，如果某个员工的工资低于200元，就增加100元。

create or replace procedure test\_cursor(in\_deptno in number) is

--创建一个记录数据类型

type my\_emp\_record is record(v\_ename emp.ename%type,

v\_sal emp.sal%type,

v\_empno emp.empno%type

);

v\_emp\_record my\_emp\_record;

--创建游标

type my\_emp\_cursor is ref cursor;

v\_emp\_cursor my\_emp\_cursor;

begin

open v\_emp\_cursor for select ename,sal,empno from emp where deptno=in\_deptno;

loop

fetch v\_emp\_cursor into v\_emp\_record;

exit when v\_emp\_cursor%notfound;

--判断工资低于200的人，加100块

if v\_emp\_record.v\_sal<200 then

v\_emp\_record.v\_sal:=v\_emp\_record.v\_sal+100;

update emp set sal=v\_emp\_record.v\_sal where empno=v\_emp\_record.v\_empno;

end if;

dbms\_output.put\_line('姓名：'||v\_emp\_record.v\_ename||' 工资：'||v\_emp\_record.v\_sal);

end loop;

close v\_emp\_cursor;

end;

/

**pl/sql练习题：**

使用pl/sql块编程实现，注意必需的异常处理。

1、输入一个员工号，输出该员工的姓名、薪金和大概的服务年限(按年月日显示)

declare

v\_ename emp.ename%type;

v\_sal emp.sal%type;

v\_year varchar2(20);

begin

select ename,sal,to\_char(to\_date('00010101','yyyymmdd')+(sysdate-hiredate)-366-31,'yy"年"mm"个月"dd"天"') into v\_ename,v\_sal,v\_year from emp where empno=&empno;

dbms\_output.put\_line('姓名：'||v\_ename||' 薪金：'||v\_sal||' 服务年限：'||v\_year);

exception

when no\_data\_found then

dbms\_output.put\_line('您输入的员工编号不存在!');

end;

/

2、接收一个员工号，输出该员工所在部门的名称

declare

v\_ename emp.ename%type;

v\_dname dept.dname%type;

begin

select e.ename,d.dname into v\_ename,v\_dname from emp e,dept d where e.deptno=d.deptno and empno=&empno;

dbms\_output.put\_line('姓名：'||v\_ename||' 所在部门：'||v\_dname);

exception

when no\_data\_found then

dbms\_output.put\_line('您输入的员工编号不存在!');

end;

/

3、接收一个部门号，如果该员工职位是MANAGER,并且在DALLAS工作那么就给他薪金加15%；如果该员工职位是CLERK,并且在NEW YORK工作，那么就给他薪金扣除5%,其他情况不作处理。

declare

v\_deptno number;

v\_deptno1 dept.deptno%type;

v\_deptno2 dept.deptno%type;

begin

v\_deptno:=&deptno;

select deptno into v\_deptno1 from dept where loc='DALLAS';

select deptno into v\_deptno2 from dept where loc='NEW YORK';

if v\_deptno=v\_deptno1 then

update emp set sal=sal\*1.15 where job='MANAGER' and deptno=v\_deptno1;

dbms\_output.put\_line('已将job为MANAGER且在DALLAS工作的员工薪资提高了15%');

elsif v\_deptno=v\_deptno2 then

update emp set sal=sal\*0.95 where job='CLERK' and deptno=v\_deptno2;

dbms\_output.put\_line('已将job为CLERK且在NEW YORK工作的员工薪资降低了5%');

else

dbms\_output.put\_line('数据未做修改或输入的部门号不存在!');

end if;

end;

/

4、接收一个员工号，输出这个员工所在部门的平均工资

declare

v\_deptno emp.deptno%type;

v\_avgsal number(7,2);

begin

select deptno into v\_deptno from emp where empno=&empno;

select avg(sal+nvl(comm,0)) into v\_avgsal from emp where deptno=v\_deptno;

dbms\_output.put\_line(v\_deptno||'部门的平均工资为：'||v\_avgsal);

exception

when no\_data\_found then

dbms\_output.put\_line('您输入的员工编号不存在!');

end;

/

5、以交互的方式给部门插入一条记录，如果出现主键冲突的异常，请显示“部门号已被占用”的字样。

declare

v\_deptno dept.deptno%type;

v\_dname dept.dname%type;

v\_loc dept.loc%type;

begin

v\_deptno:=&deptno;

v\_dname:=&dname;

v\_loc:=&loc;

insert into dept (deptno,dname,loc) values(v\_deptno,v\_dname,v\_loc);

dbms\_output.put\_line('部门编号：'||v\_deptno||'部门名称：'||v\_dname||'所在地：'||v\_loc||'已成功添加!');

exception

when dup\_val\_on\_index then

dbms\_output.put\_line('部门号已被占用!');

end;

/

**过程函数练习题：**

1、建立一个存储过程用来接收一个员工号，返回他的工资和他所在的部门的平均工资并作为传出参数传出。

create or replace procedure inempno\_outsaldname(in\_empno in number,out\_sal out number,out\_avgsal out number) is

v\_deptno emp.deptno%type;

begin

select sal,deptno into out\_sal,v\_deptno from emp where empno=in\_empno;

dbms\_output.put\_line('部门号：'||v\_deptno);

select avg(sal) into out\_avgsal from emp where deptno=v\_deptno;

dbms\_output.put\_line('平均工资：'||out\_avgsal);

end;

/

create or replace procedure in\_out\_emp(in\_empno number) is

v\_sal number;

v\_avgsal number(7,2);

begin

inempno\_outsaldname(in\_empno,v\_sal,v\_avgsal);

dbms\_output.put\_line('员工号：'||in\_empno||'工资为：'||v\_sal||'平均工资为：'||v\_avgsal);

exception

when no\_data\_found then

dbms\_output.put\_line('您输入的员工编号不存在!');

end;

/

2、建立一个存储过程用来接收一个部门号，找出其中的两位最老的员工的员工号，并打印。

create or replace procedure indeptno\_outhiredate(in\_deptno number) is

type my\_emp\_cursor is ref cursor;

v\_emp\_cursor my\_emp\_cursor;

v\_i number;

type my\_emp\_record is record(v\_empno emp.empno%type,v\_ename emp.ename%type,v\_hiredate emp.hiredate%type);

v\_emp\_record my\_emp\_record;

begin

v\_i:=0;

open v\_emp\_cursor for select empno,ename,hiredate from emp where deptno=in\_deptno order by hiredate;

loop

fetch v\_emp\_cursor into v\_emp\_record;

v\_i:=v\_i+1;

if(v\_i=3)then

exit;

end if;

dbms\_output.put\_line('员工编号：'||v\_emp\_record.v\_empno||'姓名：'||v\_emp\_record.v\_ename||'入职日期：'||to\_char(v\_emp\_record.v\_hiredate,'yyyy-mm-dd'));

end loop;

close v\_emp\_cursor;

exception

when no\_data\_found then

dbms\_output.put\_line('您输入的部门编号不存在!');

end;

/

3、编写一个过程用来传入一个员工号，在emp表中删除一个员工，当该员工是该部门的最后一个员工时就在dept表中删除该员工所在的部门。

create or replace procedure inempno\_deldept(in\_empno number) is

v\_count number;

v\_empno emp.empno%type;

v\_deptno emp.deptno%type;

begin

v\_empno:=in\_empno;

select deptno into v\_deptno from emp where empno=v\_empno;

delete from emp where empno=v\_empno;

dbms\_output.put\_line('工号：'||v\_empno||'成功删除!');

select count(\*) into v\_count from emp where deptno=v\_deptno;

dbms\_output.put\_line(v\_deptno||'部门还有'||v\_count||'雇员!');

if v\_count=0 then

delete from dept where deptno=v\_deptno;

dbms\_output.put\_line(v\_deptno||'删除部门成功!');

end if;

exception

when no\_data\_found then

dbms\_output.put\_line('您输入的员工编号不存在!');

end;

/

**pl/sql的进阶**

Procedural Language/SQL叫做过程化SQL编程语言，是oracle对SQL语句的扩展。在普通SQL语句的使用上增加了编程语言的特点，所以pl/sql就是把数据操作和查询语言组织在pl/sql代码的过程性单元中，通过逻辑判断，循环等操作实现复杂的功能或者计算的程序语言。

pl/sql进阶--控制结构

在任何计算机语言(c,java,c#,c++)都有各种控制语句(条件语句，循环语句，顺序控制结构..)在pl/sql中也存在这样的控制结构。

条件分支语句

pl/sql中提供了三种条件分支语句

if--then,if--then--else,if--then--elsif--elsif—else

这里我们可以和java语句进行一个比较。

简单的条件判断if--then

**基本语法：**

if 条件表达式 then

执行语句...;

end if;

编写一个过程，可以输入一个雇员名，如果该雇员的工资低于2000，就给该雇员工资增加10%。

create or replace procedure inempno\_upsal(in\_ename varchar2) is

v\_sal emp.sal%type;

begin

select sal into v\_sal from emp where ename=in\_ename;

if v\_sal<2000 then

update emp set sal=sal\*1.1 where ename=in\_ename;

end if;

exception

when no\_data\_found then

dbms\_output.put\_line('您输入的姓名不存在!');

end;

/

二重条件分支if--then--else

**基本语法：**

if 条件表达式 then

执行语句;

else

执行语句;

end if;

编写一个过程，可以输入一个雇员名，如果该雇员的补助不是0就在原来的基础上增加100；如果补助为0就把补助设为200；

create or replace procedure inname\_upcomm(in\_ename varchar2) is

v\_comm emp.comm%type;

begin

select nvl(comm,0) into v\_comm from emp where ename=in\_ename;

if v\_comm<>0 then

update emp set comm=comm+100 where ename=in\_ename;

else

update emp set comm=comm+200 where ename=in\_ename;

end if;

exception

when no\_data\_found then

dbms\_output.put\_line('您输入的姓名不存在!');

end;

/

多重条件分支if--then--elsif--else

**基本语法：**

if 条件表达式 then

执行语句;

elsif 条件表达式 then

执行语句;

else

执行语句;

end if;

编写一个过程，可以输入一个雇员编号，如果该雇员的职位是PRESIDENT就给他的工资增加1000，如果该雇员职位是MANAGER就给他的工资增加500，其它职位的雇员工资增加200。

create or replace procedure inempno\_upsal(in\_empno number) is

v\_job emp.job%type;

begin

select job into v\_job from emp where empno=in\_empno;

if v\_job='PRESIDENT' then

update emp set sal=sal+1000 where empno=in\_empno;

elsif v\_job='MANAGER' then

update emp set sal=sal+500 where empno=in\_empno;

else

update emp set sal=sal+200 where empno=in\_empno;

end if;

exception

when no\_data\_found then

dbms\_output.put\_line('您输入的编号有误!');

end;

/

**循环结构--loop**

是pl/sql中最简单的循环语句，这种循环语句以loop开头，以end loop结尾，这种循环至少会被执行一次。

**基本语法：**

loop

执行语句;

exit when 条件表达式;

end loop;

案例：现在有一张表users，表结构如下：

|  |  |
| --- | --- |
| 用户ID | 用户名 |
|  |  |

请编写一个过程，可以输入用户名和添加用户的个数n;循环添加n个用户到users表中，用户编号从1开始增加，直到n

create table users(userId number primary key,userName varchar2(32));

create or replace procedure inname\_adduser(in\_username varchar2,in\_n number) is

v\_i number:=0;

begin

loop

exit when in\_n<=0;

v\_i:=v\_i+1;

insert into users values(v\_i,in\_username);

exit when v\_i=in\_n;

end loop;

end;

/

**循环语句--while循环**

基本循环至少要执行循环体一次，而对于while循环来说，只有条件为true时，才会执行循环体语句，while循环以while..loop开始，以end loop结束。

**基本语法：**

while 条件表达式 loop

执行语句;

end loop;

案例：现在有一张表users，表结构如下：

|  |  |
| --- | --- |
| 用户ID | 用户名 |
|  |  |

请编写一个过程，可输入用户名，并循环添加10个用户到users表中，用户编号从11开始增加。

create table users(userId number primary key,userName varchar2(32));

create or replace procedure inname\_addusers(in\_name varchar2) is

v\_i number:=0;

v\_j number:=10;

begin

while v\_i<10 loop

v\_j=v\_j+1;

insert into users values(v\_j,in\_name);

end loop;

end;

/

看下面题判断是否正确

下面的过程是否正确，如果不正确，应该怎么改？

create or replace procedure sp\_pro6(spName varchar2) is

v\_test varchar2(40);

v\_test:='aaa';--赋初值需在定义时就直接赋值，或在定义之后在begin中进行赋值。

begin

dbms\_output.put\_line(v\_test);

end;

create or replace procedure sp\_pro6(spName varchar2) is

v\_test varchar2(40):='aaa';

begin

spName:='你好';--spName为传入变量，不能重复赋值。

dbms\_output.put\_line(v\_test||spName);

end;

**说明：**

1、在is--begin之间只能定义变量类型同时初始化赋值，或定义变量类型后在begin内进行赋值，不能在定义变量类型之后再对变量赋值。

2、传入的参数变量不能在存储过程中再次赋值。

**循环语句--for循环**

基本for循环的基本结构如下：

begin

for i in reverse 1..10 loop

insert into users values(i,'顺平');

end loop;

end;

**基本语法：**

for 变量 in reverse 开始值..结束值 loop

执行语句;

end loop;

我们可以看到控制变量i,在隐含中就在不停的增加

**注意：推荐使用loop循环结构，不推荐使用for循环。**

**顺序控制语句--goto,null**

**1、goto语句**

goto语句用于跳转到特定标号去执行语句，注意由于使用goto语句会增加程序的复杂性，并使得应用程序可读性变差，所以在做一般应用开发时，**建议大家不要使用goto语句。**基本语法如下：goto lable，其中lable是已定义好的标号名。

基本语法：goto 标号;

标号定义：<<标号>>

例：

declare

i number:=1;

begin

<<start\_loop>>

loop

dbms\_output.put\_line('输出i='||i);

if i=12 then

goto end\_loop;

end if;

i:=i+1;

if i=10 then

goto start\_loop;

end if;

end loop;

<<end\_loop>>

dbms\_output.put\_line('循环结束');

end;

--输出1至12 循环结束。

**2、null**

null语句不会执行任何操作，并且会直接将控制传递到下一条语句。使用null语句的主要好处是可以提高pl/sql的可读性。

例：

declare

v\_sal emp.sal%type;

v\_ename emp.ename%type;

begin

select ename,sal into v\_ename,v\_sal from emp where empno=&no;

if v\_sal<3000 then

update emp set comm=sal\*0.1 where ename=v\_ename;

else

null;

end if;

end;

**pl/sql进阶--编写分页过程**

介绍

分页是任何一个网站(bbs、网上商城、blog)都会使用到的技术，因此学习pl/sql编程开发一定要掌握该技术。

**无返回值的存储过程**

古人云：欲速则不达，为了让大家比较容易接受分页过程编写，还是从简单到复杂，循序渐进的给大家讲解。首先是掌握最简单的存储过程，无返回值的存储过程：

案例：现在有一张表book,表结构如下：

字段名 字段类型

id number(5)

name varchar2(100)

pubHouse varchar2(100)

请编写一个过程，可以向book表添加书，要求通过java程序调用该过程。

提示查看jdk,看看CallableStatement是怎么调存储过程的!

建book表

create table book(id number(5) primary key,name varchar2(100) not null,pubHouse varchar2(100));

create or replace procedure inBook(in\_id number,in\_name varchar2,in\_pubHouse varchar2)

begin

insert into book values(in\_id,in\_name,in\_pubHouse);

exception

when dup\_val\_on\_index then

dbms\_output.put\_line('错误：序号不能为空或不能重复!');

end;

**java调用无返回值的存储过程代码:**

//当我们需要去调用过程的时候传SQL语句

String sql="{call 存储过程名称(?,?)}";//调用格式{call 用户区.包名.过程名(参数?)}

String paras[]={"过程中的参数1","过程中的参数2"};

SQLHelper.*executeProcedure*(sql, paras);

**public** **class** SQLHelper {

//定义三个变量

**private** **static** Connection *ct*=**null**;

**private** **static** PreparedStatement *ps*=**null**;

**private** **static** ResultSet *rs*=**null**;

**private** **static** CallableStatement *cs*=**null**;

//连接数据库的用户名，密码，url，驱动

//说明：在实际开发中，我们往往把这些变量写到一个外部文件中

//当程序启动时，我们读入这些配置信息。java.util.Properites

**private** **static** String *username*;

**private** **static** String *password*;

**private** **static** String *driver*;

**private** **static** String *url*;

//使用静态块加载驱动(驱动只需要加载一次)

**static**{

//使用Properties类，来读取配置文件

Properties pp=**new** Properties();

FileInputStream fis=**null**;

**try** {

fis=**new** FileInputStream("dbinfo.properties");

//让pp与dbinfo.properties文件关联起来

pp.load(fis);

//获取dbinfo.properties文件内信息

*username*=pp.getProperty("username");

*password*=pp.getProperty("password");

*driver*=pp.getProperty("driver");

*url*=pp.getProperty("url");

//获得驱动

Class.*forName*(*driver*);

} **catch** (Exception e) {

e.printStackTrace();

}**finally**{

**try** {

**if**(fis!=**null**){

fis.close();

}

} **catch** (Exception e) {

e.printStackTrace();

}

fis=**null**;

}

}

//调用存储过程的方法

**public** **static** **void** executeProcedure(String sql,String [] parameters){

**try** {

*ct*=DriverManager.*getConnection*(*url*,*username*,*password*);

*cs*=*ct*.prepareCall(sql);

**if**(parameters!=**null**){

**for**(**int** i=0;i<parameters.length;i++){

*cs*.setString(i+1, parameters[i]);

System.*out*.println(parameters[i]);

}

}

//执行

*cs*.execute();

*ct*.commit();

} **catch** (Exception e) {

e.printStackTrace();

**throw** **new** RuntimeException(e.getMessage());

}**finally**{

*close*(*rs*, *cs*, *ct*);

}

}

//把关闭资源写成函数

**public** **static** **void** close(ResultSet rs,Statement ps,Connection ct){

//关闭资源

**if**(rs!=**null**){

**try** {

rs.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

rs=**null**;

}

**if**(ps!=**null**){

**try** {

ps.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

ps=**null**;

}

**if**(ct!=**null**){

**try** {

ct.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

ct=**null**;

}

}

}

**dbinfo.properties 连接Oracle数据库配置文件**

username=scott

password=tiger

driver=oracle.jdbc.driver.OracleDriver

url=jdbc\:oracle\:thin\:@127.0.0.1\:1521\:orcl

**有返回值的存储过程(非列表，只有一个返回值)**

再看如何处理有返回值的存储过程：

**建立有返回值的存储过程基本语法：**

create or replace procedure 过程名(参数名 in 类型,..,参数名 out 类型,..) is

定义变量..;

begin

执行语句..;

exception

when 错误提示 then

处理或提示语句;

end;

案例：编写一个过程，可以输入雇员的编号，返回该雇员的姓名。

create or replace procedure inEmpno\_outEname(in\_v\_empno in number,out\_v\_ename out varchar2) is

begin

select ename into out\_v\_ename from emp where empno=in\_v\_empno;

exception

when no\_data\_found then

dbms\_output.put\_line('您输入的雇员编号不存在!');

end;

在java中去调用该过程，并接受返回的用户名。

[TestProcedureOutValues.java]源码示例：

**package** com.test;

**import** java.sql.CallableStatement;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**public** **class** TestProcedureOutValues {

//调用oracle存储过程并获得存储过程的返回值。

**public** **static** **void** main(String[] args) {

//定义需要的变量

Connection ct=**null**;

CallableStatement cs=**null**;

ResultSet rs=**null**;

**try** {

//加载驱动

Class.*forName*("oracle.jdbc.driver.OracleDriver");

//得到连接

ct=DriverManager.*getConnection*("jdbc:oracle:thin:@127.0.0.1:1521:orcl","scott","tiger");

//创建CallableStatement接口

cs=ct.prepareCall("{call inEmpno\_outEname(?,?)}");//第一个?输入值，第二个?是获得值。

//给第一个?赋值

cs.setString(1, "7839");

//给第二个?注册(因为它是输出值)

cs.registerOutParameter(2, oracle.jdbc.OracleTypes.*VARCHAR*);

//执行

cs.execute();

//取出输出的值

String ename=cs.getString(2);

System.*out*.println("用户的名字是："+ename);

} **catch** (Exception e) {

e.printStackTrace();

}**finally**{

//关闭资源

**try** {

**if**(cs!=**null**){

cs.close();

}

**if**(rs!=**null**){

rs.close();

}

**if**(ct!=**null**){

ct.close();

}

} **catch** (Exception e) {

e.printStackTrace();

}

cs=**null**;

rs=**null**;

ct=**null**;

}

}

}

**说明：**

1、对于过程的输入值，使用set方法，对于输出值使用registerOutParameter来注册接收返回值。问号的顺序要对应，同时考虑类型。

2、取出过程返回值的方法是CallableStatement提供的get方法(输出参数的位置)；同时要考虑输出的参数类型。

java调用关键代码：

CallableStatement cs=null;

cs=ct.prepareCall("{call 过程名(?,?)}");

cs.registerOutParameter(输出参数的在第几个问号,oracle.jdbc.OracleTypes.类型);//Types.类型是输出参数的类型。

cs.get类型(输出参数在问号的位置);//不同的类型要用不同的get方法接收。

案例扩展：编写一个过程，可以输入雇员的编号，返回该雇员的姓名、工资和岗位。

create or replace procedure inEmpno\_outAllinfo(in\_v\_empno in number,out\_v\_ename out varchar2,out\_v\_sal out number,out\_v\_job out varchar2) is

begin

select ename,sal,job into out\_v\_ename,out\_v\_sal,out\_v\_job from emp where empno=in\_v\_empno;

exception

when no\_data\_found then

dbms\_output.put\_line('您输入的雇员编号不存在!');

end;

在java中去调用该过程，并接受返回的用户名、工资、职位。

[TestProcedureOutValues.java]返回多个值的调用存储过程源代码。

**package** com.test;

**import** java.sql.CallableStatement;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**public** **class** TestProcedureOutValues {

//调用oracle存储过程并获得存储过程的返回值。

**public** **static** **void** main(String[] args) {

//定义需要的变量

Connection ct=**null**;

CallableStatement cs=**null**;

ResultSet rs=**null**;

**try** {

//加载驱动

Class.*forName*("oracle.jdbc.driver.OracleDriver");

//得到连接

ct=DriverManager.*getConnection*("jdbc:oracle:thin:@127.0.0.1:1521:orcl","scott","tiger");

//创建CallableStatement接口

cs=ct.prepareCall("{call inEmpno\_outAllinfo(?,?,?,?)}");//第一个?输入值，第二个?是获得值。

//给第一个?赋值

cs.setString(1, "7839");

//给第二-四个?注册(因为它是输出值)

cs.registerOutParameter(2, oracle.jdbc.OracleTypes.*VARCHAR*);

cs.registerOutParameter(3, oracle.jdbc.OracleTypes.*NUMBER*);

cs.registerOutParameter(4, oracle.jdbc.OracleTypes.*VARCHAR*);

//执行

cs.execute();

//取出输出的值

String ename=cs.getString(2);

Float sal=cs.getFloat(3);

String job=cs.getString(4);

System.*out*.println("姓名："+ename+" 工资："+sal+" 职位："+job);

} **catch** (Exception e) {

e.printStackTrace();

}**finally**{

//关闭资源

**try** {

**if**(cs!=**null**){

cs.close();

}

**if**(rs!=**null**){

rs.close();

}

**if**(ct!=**null**){

ct.close();

}

} **catch** (Exception e) {

e.printStackTrace();

}

cs=**null**;

rs=**null**;

ct=**null**;

}

}

}

**有返回值的存储过程(列表[结果集])**

案例：编写一个过程，输入部门号，返回该部门所有雇员信息。

对该题分析如下：

由于oracle存储过程没有返回值，它的所有返回值都是通过out参数来替代的，列表同样也不例外，但由于是集合，所以不能用一般的参数，必须要用package了，步骤如下：

1、建一个包。

2、建立存储过程。

3、下面如何在java程序中调用。

**创建包同时定义一个游标类型**

create or replace package empPackage is

--定义一个游标数据类型

type my\_emp\_cursor is ref cursor;

end;

**创建存储过程**

create or replace procedure indeptno\_outAllInfo(v\_in\_deptno in number,v\_out\_result out empPackage.my\_emp\_cursor) is

begin

open v\_out\_result for select \* from emp where deptno=v\_in\_deptno;

--close v\_out\_result;--此处不能关闭游标，需要在程序中关闭游标。

end;

**编写java程序获得存储过程返回的结果集。**

[TestProcedureOutAllValues.java]源代码

**package** com.test;

**import** java.sql.CallableStatement;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**public** **class** TestProcedureOutAllValues {

//调用oracle存储过程并获得存储过程的返回结果集。

**public** **static** **void** main(String[] args) {

//定义需要的变量

Connection ct=**null**;

CallableStatement cs=**null**;

ResultSet rs=**null**;

**try** {

//加载驱动

Class.*forName*("oracle.jdbc.driver.OracleDriver");

//得到连接

ct=DriverManager.*getConnection*("jdbc:oracle:thin:@127.0.0.1:1521:orcl","scott","tiger");

//创建CallableStatement接口

cs=ct.prepareCall("{call indeptno\_outAllInfo(?,?)}");

//给?赋值

cs.setInt(1, 20);

//给第二个?注册

cs.registerOutParameter(2, oracle.jdbc.OracleTypes.*CURSOR*);

//执行

cs.execute();

/\*这里是关键所在，java没有接收结果集的get方法，所以只能用getObject来接收结果集，接收到后需要使用ResultSet强转才可以。\*/

rs=(ResultSet)cs.getObject(2);

//循环取出

**while**(rs.next()){

System.*out*.println(rs.getString("ename")+" "+rs.getString("sal"));

}

} **catch** (Exception e) {

e.printStackTrace();

}**finally**{

//关闭资源

**try** {

**if**(cs!=**null**){

cs.close();

}

**if**(rs!=**null**){

rs.close();

}

**if**(ct!=**null**){

ct.close();

}

} **catch** (Exception e) {

e.printStackTrace();

}

cs=**null**;

rs=**null**;

ct=**null**;

}

}

}

**编写分页过程**

**有了上面的基础，相信大家可以完成分页存储过程了。**

**要求：请大家编写一个存储过程，要求可以输入表名、每页显示记录数、当前页，排序字段(deptno降序)。返回总记录数，总页数和返回的结果集。**

**把一个字符串，当做sql语句执行，并把查询得到的结果赋给某个变量，语法如下：**

**execute immediate v\_sql into myrows;**

**基本语法：**

**execute immediate 变量(sql拼接语句) into 输出变量名;**

温馨提示：

如果大家忘了oracle中如何分页，请参考第三天的内容

提示：为了讲的清楚明白，这里使用循序渐进的方法，逐步增加传入的参数来讲解。

**先简化再复杂**

通过输入表名、每页显示记录数、当前页，返回结果集。

**1、创建包同时创建游标**

create or replace package pagingPackage is

type paging\_cursor is ref cursor;

end;

**2、创建分页存储过程**

create or replace procedure paging\_cursor(v\_in\_table in varchar2,v\_in\_pagesize in number,v\_in\_pagenow in number,v\_out\_result out pagingPackage.paging\_cursor) is

--定义需要的变量

v\_sql varchar2(4000);

v\_start number;

v\_end number;

begin

--执行代码

--计算v\_start和v\_end是多少

v\_start:=v\_in\_pagesize\*(v\_in\_pagenow-1)+1;

v\_end:=v\_in\_pagesize\*v\_in\_pagenow;

v\_sql:='select t2.\* from (select t1.\*,rownum rn from (select \* from '||v\_in\_table||') t1 where rownum<='||v\_end||') t2 where rn>='||v\_start;

--打开游标，让游标指向结果集

open v\_out\_result for v\_sql;

end;

**java调用分页存储过程**

[TestProcedurePaging.java]源代码

**package** com.test;

**import** java.sql.CallableStatement;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**public** **class** TestProcedurePaging {

//调用oracle分页存储过程并获得存储过程的返回结果集。

**public** **static** **void** main(String[] args) {

//定义需要的变量

Connection ct=**null**;

CallableStatement cs=**null**;

ResultSet rs=**null**;

**try** {

//加载驱动

Class.*forName*("oracle.jdbc.driver.OracleDriver");

//得到连接

ct=DriverManager.*getConnection*("jdbc:oracle:thin:@127.0.0.1:1521:orcl","scott","tiger");

//创建CallableStatement接口

cs=ct.prepareCall("{call paging\_cursor(?,?,?,?)}");

//给in?赋值

cs.setString(1,"emp");//传表名

cs.setInt(2, 6);//传入pagesize，每页显示多少条记录

cs.setInt(3, 1);//传入pagenow，显示第几页。

//给out?注册

cs.registerOutParameter(4, oracle.jdbc.OracleTypes.*CURSOR*);

//执行

cs.execute();

/\*这里是关键所在，java没有接收结果集的get方法，所以只能用getObject来接收结果集，接收到后需要使用ResultSet强转才可以。\*/

rs=(ResultSet)cs.getObject(4);

//循环取出

**while**(rs.next()){

System.*out*.println(rs.getString("ename")+" "+rs.getString("sal"));

}

} **catch** (Exception e) {

e.printStackTrace();

}**finally**{

//关闭资源

**try** {

**if**(cs!=**null**){

cs.close();

}

**if**(rs!=**null**){

rs.close();

}

**if**(ct!=**null**){

ct.close();

}

} **catch** (Exception e) {

e.printStackTrace();

}

cs=**null**;

rs=**null**;

ct=**null**;

}

}

}

**要求：请大家编写一个存储过程，要求可以输入表名、每页显示记录数、当前页，排序字段(deptno降序)。返回总记录数，总页数和返回的结果集。**

**1、创建包同时创建游标**

create or replace package pagingPackage is

type paging\_cursor is ref cursor;

end;

**2、创建分页存储过程**

create or replace procedure paging\_cursor(v\_in\_table in varchar2,v\_in\_pagesize in number,v\_in\_pagenow in number,v\_out\_result out pagingPackage.paging\_cursor,v\_out\_rows out number,v\_out\_pagecount out number) is

--定义需要的变量

v\_sql varchar2(4000);

v\_sql\_select varchar2(4000);

v\_start number;

v\_end number;

begin

--执行代码

--计算v\_start和v\_end是多少

v\_start:=v\_in\_pagesize\*(v\_in\_pagenow-1)+1;

v\_end:=v\_in\_pagesize\*v\_in\_pagenow;

v\_sql:='select t2.\* from (select t1.\*,rownum rn from (select \* from '||v\_in\_table||') t1 where rownum<='||v\_end||') t2 where rn>='||v\_start;

--打开游标，让游标指向结果集

open v\_out\_result for v\_sql;

--查询共有多少条记录

v\_sql\_select:='select count(\*) from '||v\_in\_table;

execute immediate v\_sql\_select into v\_out\_rows;

--统计多少页记录

if mod(v\_out\_rows,v\_in\_pagesize)=0 then

v\_out\_pagecount:=v\_out\_rows/v\_in\_pagesize;

else

v\_out\_pagecount:=v\_out\_rows/v\_in\_pagesize+1;

end if;

end;

**java调用分页存储过程(完整)源代码**

[TestProcedurePaging.java]源代码

**package** com.test;

**import** java.sql.CallableStatement;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**public** **class** TestProcedurePaging {

//调用oracle分页存储过程并获得存储过程的返回结果集。

**public** **static** **void** main(String[] args) {

//定义需要的变量

Connection ct=**null**;

CallableStatement cs=**null**;

ResultSet rs=**null**;

**try** {

//加载驱动

Class.*forName*("oracle.jdbc.driver.OracleDriver");

//得到连接

ct=DriverManager.*getConnection*("jdbc:oracle:thin:@127.0.0.1:1521:orcl","scott","tiger");

//创建CallableStatement接口

cs=ct.prepareCall("{call paging\_cursor(?,?,?,?,?,?)}");

//给in?赋值

cs.setString(1,"emp");//传表名

cs.setInt(2, 6);//传入pagesize，每页显示多少条记录

cs.setInt(3, 1);//传入pagenow，显示第几页。

//给out?注册

cs.registerOutParameter(4, oracle.jdbc.OracleTypes.*CURSOR*);

cs.registerOutParameter(5, oracle.jdbc.OracleTypes.*INTEGER*);

cs.registerOutParameter(6, oracle.jdbc.OracleTypes.*INTEGER*);

//执行

cs.execute();

/\*这里是关键所在，java没有接收结果集的get方法，所以只能用getObject来接收结果集，接收到后需要使用ResultSet强转才可以。\*/

rs=(ResultSet)cs.getObject(4);

//循环取出

**while**(rs.next()){

System.*out*.println(rs.getString("ename")+" "+rs.getString("sal"));

}

//取出总记录数

**int** rowCount=cs.getInt(5);

//取出总页数

**int** pageCount=cs.getInt(6);

System.*out*.println("共有记录："+rowCount+"条! "+"共有记录："+pageCount+"页!");

} **catch** (Exception e) {

e.printStackTrace();

}**finally**{

//关闭资源

**try** {

**if**(cs!=**null**){

cs.close();

}

**if**(rs!=**null**){

rs.close();

}

**if**(ct!=**null**){

ct.close();

}

} **catch** (Exception e) {

e.printStackTrace();

}

cs=**null**;

rs=**null**;

ct=**null**;

}

}

}

**oracle视图**

介绍

视图是一个**虚拟表**，其内容由查询定义。同真实的表一样，视图包含一系列带有名称的列和行数据。但是，视图并不在数据库中以存储的数据值形式存在。行和列数据来自由定义视图的查询所引用的表，并且在引用视图时动态生成。

视图是oracle又一种数据对象，视图的主要的用处是简化操作，提高案例，满足不同用户的查询需求，视图不是一个真正存在的物理表，它是根据别的表动态生成的。

**创建视图基本语法：**

create view 视图名 as select语句 [with read only]

特别说明：with read only如果带上的话，只能查询，不能改写。

**创建或修改视图基本语法：**

create op replace view 视图名 as select语句 [with read only]

特别说明：with read only如果带上的话，只能查询，不能改写。

**删除视图基本语法：**

drop view 视图名;

案例：创建和emp表(empno,ename,job)完全一致的视图，看看带with read only和不带的区别。

注意：当表结构过于复杂时，请使用视图。

create or replace view empview as select empno,ename,job from emp;

视图可以简化操作(可以将多表中的字段合并在一张视图中)

例：我们希望查询雇员的名字和部门编号和部门名称

create or replace view empdeptview as select e.ename,e.deptno,d.dname from emp e,dept d where e.deptno=d.deptno with read only;

**视图与表的区别**

1、表需要占用磁盘空间，而视图不需要；

2、视图不能添加索引；

3、使用视图可以简化复杂查询；

4、使用视图利于提高安全性。

**oracle触发器**

几个问题：

请大家考虑一个需求：当一个用户登录到oracle时，在一张表中记录登录到oracle的用户和登录时间等信息，怎么办？再一个需求：禁止用户在星期天对某一张表进行删除操作，怎么办？再一个需求：当用户在删除一张表的时候，自动把删除的记录备份到另外一张表中...

**解决之道**

**很多关系数据库中都提供一种技术，可以在用户进行某种操作的时候，自动的进行另外一个操作，我们把这种技术称为触发器技术。**

触发器是指存放在数据库中，被隐含执行的存储过程，可以支持dml触发器，还支持基于系统事件(启动数据库，关闭数据库，登陆)和ddl操作建立触发器。

当发生特定事件时(如修改表、建立对象、登陆到数据库)，oracle就会自动的去执行相应的代码。

**触发器分类：**

**1、dml触发器；**

**2、系统事件触发器；**

**3、ddl触发器；**

触发器由触发事件，触发条件，触发操作三个部分构成。

**触发器--创建基本语法：**

create [or replace] trigger trigger\_name

{before|after}

{insert|delete|update[of column [,column...]]}

or {insert|delete|update[of column [,column...]]}

on [schema.] table\_name|[schema.]view\_name

[for each row]

[when condition]

begin

trigger\_body;

end;

create [or replace] trigger 触发器名称

{befor|after}

{insert|delete|update[of column [,column...]]}

or {insert|delete|update[of column [,column...]]}

on [schema.] 表名|[schema.]视图

[for each row]

[when condition]

begin

执行语句;

end;**详细说明：**

or replace带上则为覆盖

befor在触发事件之前执行

after在触发事件之后执行

insert/delete/update在插入、删除、更新操作时触发

or可以多个操作同时定义触发器

on对哪一个表或视图进行监控

for each row带上是对每一条数据都记录

when condition 条件表达式。

**快速入门**

在某张表(my\_emp)添加一条数据的时候，提示“添加了一条数据”

1、建表

create table my\_emp(id number,name varchar2(32));

2、创建一个触发器

create or replace trigger trigger1

after insert on scott.my\_emp

begin

dbms\_output.put\_line('添加了一条数据');

end;

在某张表(my\_emp)修改多条数据的时候，提示多次“修改了数据”

行级触发器和语句级触发器的区别

在创建触发器的时候，带不带for each row

create or replace trigger trigger2

after update on scott.my\_emp

for each row--**这是一个行级触发器**

begin

dbms\_output.put\_line('修改了数据');

end;

**快速之门2**

为了禁止工作人员在休息日改变员工信息，开发人员可以建立before语句触发器，从而实现数据的安全。

create or replace trigger trigger3

before insert or update or delete on scott.my\_emp

for each row

begin

if to\_char(sysdate,'day') in ('星期六','星期日') then

raise\_application\_error(-20001,'对不起，休息日不能对数据进行改动。');

end if;

end;

**特别注意：**

procedure raise\_application\_error(error\_number\_in in number,error\_msg\_in in varchar2);

error\_number\_in[自定义]从-20000至-20999之间，这样就不会与oracle的任何错误代码发生冲突。error\_msg\_in[自定义]的长度不能超过2k,否则截取2k。

**触发器--dml触发器**

使用条件谓词

当触发器中同时包含多个触发事件(insert,update,delete)时，为了在触发器代码中区分具体的触发事件，可以使用三个条件：

inserting

updating

deleting

为了禁止工作人员在休息日改变员工信息，开发人员可以建立before语句触发器，从而实现数据的安全，在给出提示时，明确提示用户是进行的insert、update还是delete操作。

create or replace trigger trigger4

before insert or update or delete on scott.my\_emp

for each row

begin

case

when inserting then

raise\_application\_error(-20001,'对不起，不能对insert数据。');

when updating then

raise\_application\_error(-20002,'对不起，不能对update数据。');

when deleting then

raise\_application\_error(-20003,'对不起，不能对delete数据。');

end case;

end;

**触发器--dml触发器**

使用:old和:new

问题：当触发器被触发时，要使用被插入、更新或删除的记录中的列值，有时要使用**操作前、后列的值**。

:new 修饰符访问操作完成后的列的值

:old 修饰符访问操作完成前的列的值

|  |  |  |  |
| --- | --- | --- | --- |
| **特性** | **insert** | **update** | **delete** |
| **old** | null | 有效 | 有效 |
| **new** | 有效 | 有效 | null |

案例：1、在修改emp表雇员的薪水时，显示雇员工资修改前和修改后的值。2、如何确保在修改员工工资不能低于原有工资。

create or replace trigger trigger5

before update on scott.emp

for each row

begin

if :new.sal<:old.sal then

dbms\_output.put\_line('工资不能低于原工资');

raise\_application\_error(-20004,'工资不能低于原工资');

else

dbms\_output.put\_line('原工资为：'||:old.sal||'现工资为：'||:new.sal);

end if;

end;

**触发器--dml触发器课堂练习**

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**到另处一张表(emp\_bak)中

create table emp\_bak as select \* from emp;

delete from emp\_bak;

create or replace trigger trigger6

before delete on scott.emp

for each row

begin

insert into emp\_bak values

(:old.empno,:old.ename,:old.job,:old.mgr,:old.hiredate,:old.sal,:old.comm,:old.deptno);

dbms\_output.put\_line('删除的'||:old.empno||'数据已备份到emp\_bak表中');

end;

**实现精细化控制**

编写一个新的触发器，如何控制员工的新工资不能低于原来的工资，同时也不能高出原来工资的20%，使用约束显示无法实现该规则。

create or replace trigger trigger7

before update on scott.emp

for each row

begin

if(:new.sal<:old.sal or :new.sal>:old.sal\*1.2) then

raise\_application\_error(-20005,'新工资不能低于原工资或不能高出原工资20%!');

end if;

end;

阻止把小于18岁的用户增加到数据库表中，请编写一个触发器完成上述任务。

create table users(id number primary key,name varchar2(32) not null,birthday date not null);

create or replace trigger trigger8

before insert on scott.users

for each row

begin

if add\_months(:new.birthday,18\*12)>sysdate then

raise\_application\_error(-20006,'年龄未满18岁，不能使用童工!');

end if;

end;

**特别说明：在存储过程中可以使用oracle提供的系统函数。**

**触发器--系统触发器**

系统事件是指基于oracle事件(例如logon/logout和startup/shutdown)所建立的触发器。通过使用系统事件触发器，提供了跟踪系统或是数据库变化的机制。

下面介绍一些常用的系统事件属性函数，和建立各种事件触发器的方法在建立系统事件触发器时，我们需要使用事件属性函数，常用的事件属性函数如下：

ora\_client\_ip\_address //返回客户端的ip

ora\_database\_name //返回数据库名称

ora\_login\_user //返回登陆用户名

ora\_sysevent //返回触发器的系统事件名

ora\_des\_encrypted\_password //返回用户des(md5)加密后的密码

**事件属性函数表**

|  |  |
| --- | --- |
| Ora\_client\_ip\_address | 返回客户端的ip地址 |
| Ora\_database\_name | 返回当前数据库名 |
| Ora\_des\_encrypted\_password | 返回des加密后的用户口令 |
| Ora\_dict\_obj\_name | 返回ddl操作所对应的数据库对象名 |
| Ora\_dict\_obj\_name\_list(name\_list out ora\_name\_list\_t) | 返回在事件中被修改的对象名列表 |
| Ora\_dict\_obj\_owner | 返回ddl操作所对应的对象的所有者名 |
| Ora\_dict\_obj\_owner\_list(owner\_list out ora\_name\_list\_t) | 返回在事件中被修改的对象的所有者列表 |
| Ora\_dict\_obj\_type | 返回ddl操作所对应的数据库对象的类型 |
| Ora\_grantee(user\_list out ora\_name\_list\_t) | 返回授权事件的授权者 |
| Ora\_instance\_num | 返回例程号 |
| Ora\_is\_alter\_column(column\_name in varchar2) | 检测特定列是否被修改 |
| Ora\_is\_creating\_nested\_table | 检测是否正在建立嵌套表 |
| Ora\_is\_drop\_column(column\_name in varchar2) | 检测特定列是否被删除 |
| Ora\_is\_servererror(error\_number) | 检测是否返回了特定oracle错误 |
| Ora\_login\_user | 返回登录用户名 |
| Ora\_sysevent | 返回触发器的系统事件名 |

**系统触发器创建基本语法：**

create or replace trigger 系统触发器名称

after[before] logon[logoff] on datebase

begin

执行语句;

end;

**详细说明：**

after 事件之后触发

before 事件之前触发

logon 登陆触发

logoff 登出触发

startup 开启系统触发

shutdown关闭系统触发

**下面给出系统触发器的种类和事件出现的时机（前或后）：**

|  |  |  |
| --- | --- | --- |
| **事件** | **允许的时机** | **说明** |
| STARTUP | AFTER | 启动数据库实例之后触发 |
| SHUTDOWN | BEFORE | 关闭数据库实例之前触发（非正常关闭不触发） |
| SERVERERROR | AFTER | 数据库服务器发生错误之后触发 |
| LOGON | AFTER | 成功登录连接到数据库后触发 |
| LOGOFF | BEFORE | 开始断开数据库连接之前触发 |
| CREATE | BEFORE，AFTER | 在执行CREATE语句创建数据库对象之前、之后触发 |
| DROP | BEFORE，AFTER | 在执行DROP语句删除数据库对象之前、之后触发 |
| ALTER | BEFORE，AFTER | 在执行ALTER语句更新数据库对象之前、之后触发 |
| DDL | BEFORE，AFTER | 在执行大多数DDL语句之前、之后触发 |
| GRANT | BEFORE，AFTER | 执行GRANT语句授予权限之前、之后触发 |
| REVOKE | BEFORE，AFTER | 执行REVOKE语句收权限之前、之后触犯发 |
| RENAME | BEFORE，AFTER | 执行RENAME语句更改数据库对象名称之前、之后触犯发 |
| AUDIT/NOAUDIT | BEFORE，AFTER | 执行AUDIT或NOAUDIT进行审计或停止审计之前、之后触发 |

**系统触发器事件属性**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **事件属性\事件** | **Startup/Shutdown** | **Servererror** | **Logon/Logoff** | **DDL** | **DML** |
| **事件名称** | □＊ | □＊ | □＊ | □＊ | ＊ |
| **数据库名称** | □＊ |  |  |  |  |
| **数据库实例号** | □＊ |  |  |  |  |
| **错误号** |  | □＊ |  |  |  |
| **用户名** |  |  | □＊ | ＊ |  |
| **模式对象类型** |  |  |  | □＊ | ＊ |
| **模式对象名称** |  |  |  | □＊ | ＊ |
| **列** |  |  |  |  | □＊ |

**除DML语句的列属性外，其余事件属性值可通过调用ORACLE定义的事件属性函数来读取。**

|  |  |  |
| --- | --- | --- |
| **函数名称** | **数据类型** | **说明** |
| Ora\_sysevent | VARCHAR2(20) | 激活触发器的事件名称 |
| Instance\_num | NUMBER | 数据库实例名 |
| Ora\_database\_name | VARCHAR2(50) | 数据库名称 |
| Server\_error(posi) | NUMBER | 错误信息栈中posi指定位置中的错误号 |
| Is\_servererror(err\_number) | BOOLEAN | 检查err\_number指定的错误号是否在错误信息栈中，如果在则返回TRUE，否则返回FALSE。在触发器内调用此函数可以判断是否发生指定的错误。 |
| Login\_user | VARCHAR2(30) | 登陆或注销的用户名称 |
| Dictionary\_obj\_type | VARCHAR2(20) | DDL语句所操作的数据库对象类型 |
| Dictionary\_obj\_name | VARCHAR2(30) | DDL语句所操作的数据库对象名称 |
| Dictionary\_obj\_owner | VARCHAR2(30) | DDL语句所操作的数据库对象所有者名称 |
| Des\_encrypted\_password | VARCHAR2(2) | 正在创建或修改的经过DES算法加密的用户口令 |

建立登陆和退出触发器

为了记录用户的登陆和退出事件，我们可以建立登陆和退出触发器为了记录用户名称、时间、ip地址。我们首先建立一张信息表。

**特别说明：系统触发器的级别较高，由系统管理员来创建。**

SQL>conn system/orcl as sysdba;

create table log\_table(username varchar2(32),logon\_time date,logoff\_time date,ip\_address varchar2(20));

--创建登陆触发器

create or replace trigger log\_tri

after logon on database

begin

insert into log\_table (username,logon\_time,ip\_address) values

(ora\_login\_user,sysdate,ora\_client\_ip\_address);

end;

--创建退出触发器

create or replace trigger exit\_tri

before logoff on database

begin

insert into log\_table (username,logoff\_time,ip\_address) values

(ora\_login\_user,sysdate,ora\_client\_ip\_address);

end;

**触发器--ddl触发器**

什么是ddl(data definition language)，说白了就是我们常用的create、alter和drop这些数据定义语句。

**创建ddl触发器基本语法：**

create or replace trigger ddl触发器名称

after ddl on 方案名.schema

begin

执行语句;

end;

请编写一个触发器，可以记录某个用户进行的ddl操作：

为了记录系统所发生的ddl事件，应该建立一张(my\_ddl\_event)用来存相关信息。

**特别注意：ddl触发器需要system用户数据库管理员权限才可以建立。**

SQL>conn system/orcl as sysdba;

create table my\_ddl\_event(event varchar2(20),username varchar2(20),time date);

create or replace trigger ddltri

after ddl on scott.schema

begin

insert into my\_ddl\_event values(ora\_sysevent,ora\_login\_user,sysdate);

end;

**特别说明：在oracle中dml语句需要手动commit(提交)，如果没有手动提交，在正常退出客户端时，oracle会自动提交；ddl语句是自动commit(提交)。**

**触发器--管理触发器**

**特别注意：管理触发器使用system登陆。**

**禁止触发器**

是指让触发器临时失效。

alter trigger 触发器名称 disable;

**激活触发器**

alter trigger 触发器名称 enable;

**禁止或是激活表的所有触发器**

alter table 表名 disable all triggers;

alter table 表名 enable all triggers;

**删除触发器**

drop trigger 触发器名称;

**特别注意：触发器是针对所有客户端的操作，只要是对输出了触发器的表进行操作，在满足触发条件，均会触发相应的触发器。**

**pl/sql进阶--例外处理**

在pl/sql的执行过程中发生异常时系统所作的处理称为一个例外情况(exception)。通常例外情况的种类有三种：

1、预定义的oracle例外情况oracle预定义的例外情况大约有24个，对于这种例外情况无需在程序中定义，由oracle自动地触发(重点)

2、非预定义的oracle例外情况由使用者增加定义例外情况，然后oracle自动将其触发执行。

3、自定义例外，这个用的较少。

**自定义例外的基本语法：**

exception

when <异常情况名> then

<异常处理代码>

when <异常情况名> then

<异常处理代码>

...

when others then

<异常处理代码>

**捕获异常的两个目的：**

1、给用户提示更加明确，方便对过程优化；

2、可能需要对异常进行业务处理。

例外传递

如果不处理例外我们看看会出现什么情况

案例：编写一个过程，可接收雇员的编号，并显示该雇员的姓名。

问题是，如果输入的雇员编号不存在，怎样去处理呢？

create or replace procedure inempno\_listename(v\_in\_empno number) is

v\_ename emp.ename%type;

begin

select ename into v\_ename from emp where empno=v\_in\_empno;

exception

when no\_data\_found then

dbms\_output.put\_line('你输入的雇员编号不存在!');

when others then

dbms\_output.put\_line('错误不明!');

end;

常用的预定义的例外

oracle预定之例外情况的处理，下列出常见的几个：

|  |  |
| --- | --- |
| **例外情况** | **错误代码** |
| NO\_DATA\_FOUND | ORA-01403 |
| TOO\_MANY\_ROWS | ORA-01427 |
| INVALID\_CURSOR | ORA-01001 |
| VALUE\_ERROR | ORA-06502 |
| INVALID\_NUMBER | ORA-01722 |
| ZERO\_DIVIDE | ORA-01476 |
| DUP\_VAL\_ON\_INDEX ORA-00001 | 试图向具有唯一键值的索引中插入一个重复键值。 |
| CASE\_NOT\_FOUND | ORA-06592 |
| CURSOR\_NOT\_OPEN ORA-06511 | 游标没有打开 |

**预定义说明的部分ORACLE异常错误**

|  |  |  |
| --- | --- | --- |
| **错误号** | **异常错误信息名称** | **说明** |
| ORA-0001 | Dup\_val\_on\_index | 违反了唯一性限制 |
| ORA-0051 | Timeout-on-resource | 在等待资源时发生超时 |
| ORA-0061 | Transaction-backed-out | 由于发生死锁事务被撤消 |
| ORA-1001 | Invalid-CURSOR | 试图使用一个无效的游标 |
| ORA-1012 | Not-logged-on | 没有连接到ORACLE |
| ORA-1017 | Login-denied | 无效的用户名/口令 |
| ORA-1403 | No\_data\_found | SELECT INTO没有找到数据 |
| ORA-1422 | Too\_many\_rows | SELECT INTO 返回多行 |
| ORA-1476 | Zero-divide | 试图被零除 |
| ORA-1722 | Invalid-NUMBER | 转换一个数字失败 |
| ORA-6500 | Storage-error | 内存不够引发的内部错误 |
| ORA-6501 | Program-error | 内部错误 |
| ORA-6502 | Value-error | 转换或截断错误 |
| ORA-6504 | Rowtype-mismatch | 宿主游标变量与 PL/SQL变量有不兼容行类型 |
| ORA-6511 | CURSOR-already-OPEN | 试图打开一个已处于打开状态的游标 |
| ORA-6530 | Access-INTO-null | 试图为null 对象的属性赋值 |
| ORA-6531 | Collection-is-null | 试图将Exists 以外的集合( collection)方法应用于一个null pl/sql 表上或varray上 |
| ORA-6532 | Subscript-outside-limit | 对嵌套或varray索引得引用超出声明范围以外 |
| ORA-6533 | Subscript-beyond-count | 对嵌套或varray 索引得引用大于集合中元素的个数. |

**pl/sql进阶--例外处理**

预定义例外case\_not\_found

在开发pl/sql块中编写case语句时，如果在when子句中没有包含必须的条件分支，就会触发case\_not\_found的例外。

预定义例外zero\_divide

当执行2/0语句时，则会触发该例外。

预定义例外no\_data\_found

下面是一个pl/sql块，当执行select into 没有返回行，就会被触发该例外。

预定义例外too\_many\_rows

当执行select into语句时，如果返回超过了一行，则会触发该例外。

**案例：**

--cast\_not\_found案例

create or replace procedure sp\_pro6(spno number) is

v\_sal emp.sal%type;

begin

select sal into v\_sal from emp where empno=spno;

case

when v\_sal<1000 then

update emp set sal=sal+100 where empno=spno;

when v\_sal<2000 then

update emp set sal=sal+200 where empno=spno;

end case;

exception

when case\_not\_found then

dbms\_output.put\_line('case语句没有与'||v\_sal||'相匹配的条件');

end;

--no\_data\_found案例

declare

v\_sal emp.sal%type;

begin

select sal into v\_sal from emp

where ename='&name';

exception

when no\_data\_found then

dbms\_output.put\_line('不存在该员工');

end;

--too\_many\_rows错误案例

declare

v\_ename emp.ename%type;

begin

select ename into v\_ename from emp;

exception

when too\_many\_rows then

dbms\_output.put\_line('返回了多行');

end;

**如何处理多个例外：**

set serveroutput on;

declare

var\_name varchar(60);

begin

select ename into var\_name from emp

where deptno=&deptno;

exception

when no\_data\_found then

dbms\_output.put\_line('没有匹配数据!');

when too\_many\_rows then

dbms\_output.put\_line('返回多行数据!');

when others then

dbms\_output.put\_line('提示错误不明!');

end;

**oracle--dba和表的备份与恢复**

数据库管理员

每个oracle数据库应该至少有一名数据库管理员(dba)，对于一个小的数据库，一个dba就够了，但是对于一个大的数据库可能需要多个dba分别担负不同的管理职责，那么一个数据库管理员的主要工作是什么呢？

**dba的职责：**

1、安装和升级oracle数据库

2、建库、表空间、表、视图、索引...

3、制定并实施备份与恢复计划

4、数据库权限管理，调优，故障排除

5、对于高级dba，要求能参与项目开发，会编写sql语句、存储过程、触发器、规则、约束、包

**管理数据库的用户主要是sys和system**

在前面我们已提到这两个用户，**区别主要是：**

**1、最重要的区别，存储的数据重要性不同。**

sys：所有oracle的数据字典的基表和视图都存放在sys用户中，这些基表和视图对于oracle的运行是至关重要的，由数据库自己维护，任何用户都不能手动更改。sys用户拥有dba(角色)，sysdba(系统权限)，sysoper(系统权限)角色或权限，是oracle权限最高的用户。

system：用于存放次一级的内部数据，如oracle的一些特性或工具的管理信息。system用户拥有dba，sysdba系统权限。

**2、其次的区别，权限的不同。**

sys用户必需以as sysdba或as sysoper形式登录，不能以normal方式登录数据库。

system如果正常登录，它其实就是一个普通的dba用户，但是如果以as sysdba登录，其结果实际上它是作为sys用户登录的，这一点类似linux里面的su的感觉，从登录信息里面我们可以看出来。

sysdba和sysoper权限区别

sysdba和sysoper具体的权限可以看下表：

|  |  |  |
| --- | --- | --- |
| **系统权限** | **sysdba** | **sysoper** |
| **区别** | startup(启动数据库) | startup |
| shutdown(关闭数据库) | shutdown |
| alter database open/mount/backup | alter database open/mount/backup |
| 改变字符集 | none |
| create database(创建数据库) | none不能创建数据库 |
| drop database(删除数据库) | none |
| create spfile | create spfile |
| alter database archivelog(归档日志) | alter database archivelog |
| alter database recover(恢复数据库) | 只能完全恢复，不能执行不完全恢复 |
| 拥有restricted session(会话限制) | 拥有restricted session权限 |
| 可以让用户作为sys用户连接 | 可以进行一些基本的操作，但不能查看用户数据 |
| 登录之后用户是sys | 登录之后用户是public |

**dba权限的用户**

dba用户是指具有dba角色的数据库用户。特权用户可以执行启动实例，关闭实例等特殊操作，而dba用户只有启动数据库后才能执行各种管理工作。

**数据库管理员--管理初始化参数**

管理初始化参数

初始化参数用于设置实例或是数据库的特征，oracle10g提供了200多个初始化参数，并且每个初始化参数都有默认值。

**显示初始化参数命令**

show parameter

如何修改参数

需要说明的如果你希望**修改这些初始化的参数，可以到文件：**

盘符:\oracle目录\admin\数据库实例\pfile\init.ora文件中去修改。

**数据库(表)的逻辑备份与恢复--介绍**

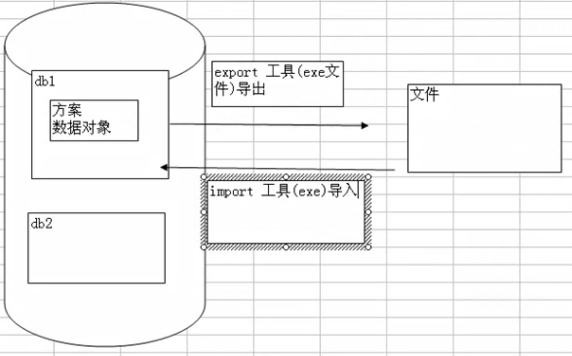
逻辑备份是指使用工具export将数据对象的结构和数据导出到文件的过程；

逻辑恢复是指当数据库对象被误操作而损坏后使用工具import利用备份的文件把数据对象导入到数据库的过程。

物理备份即可在数据库open的状态下进行也可在关闭数据库后进行；

但是逻辑备份和恢复只能在open的状态下进行。

原理图(所有数据库一样，只是使用的工具不一样而以。)



**数据库(表)的逻辑备份与恢复--导出**

**导出具体的分为：导出表、导出方案、导出数据库三种方式。**

**导出使用exp命令来完成的**，该命令常用的选项有：

userid：用于指定执行导出操作的用户名、口令、连接字符串；

tables：用于指定执行导出操作的表；

owner：用于指定执行导出操作的方案；

full=y：用于指定执行导出操作的数据库；

inctype：用于指定执行导出操作的增量类型；

rows：用于指定执行导出操作是否要导出表中的数据；

file：用于指定导出文件名。

导出表

**1、导出自己的表**

exp userid=用户名/密码@数据库实例名 tables=(表名1,表名2,..) file=备份路径

例：exp userid=scott/tiger@orcl tables=(emp,dept) file=d:/scott\_emp\_dept.dmp

**2、导出其它方案的表**

**如果用户要导出其它方案的表，则需要DBA的权限或是exp\_full\_database的权限**，比如system就可以导出scott的表

exp useris=用户名/密码@数据库实例名 tables=(方案名.表名1,方案名.表名2,..) file=备份路径

例：exp userid=system/orcl@orcl tables=(scott.emp,scott.dept) file=d:/scott\_emp\_

dept2.dmp

**特别说明：备份文件的后缀名一般是.dmp，但是这不是必须的。**

**3、导出表的结构**

exp userid=用户名/密码@数据库实例名 tables(表名1,表名2,..) file=备份路径 rows=n

例：exp userid=system/orcl@orcl tables(scott.emp,scott.dept) file=d:/scott\_em

p\_dept.demp rows=n

**4、使用直接导出方式**

exp userid=用户名/密码@数据库实例名 tables(表名1,表名2,..) file=备份路径 direct=y

例：exp userid=system/orcl@orcl tables(scott.emp,scott.dept) file=d:/scott\_em

p\_dept.demp direct=y

这种方式比默认的常规方式速度要快，当数据量大时，可以考虑使用这样的方法。

这时需要**数据库的字符集要与客户端字符集完全一致，否则会报错。**

**导出方案**

导出方案是指使用export工具导出一个方案或是多个方案中的所有对象(表、索引、约束..)和数据，并存放到文件中。

**1、导出自己的方案**

exp 用户名/密码@数据库实例名 owner=所有者 file=备份路径

例：exp scott/tiger@orcl owner=scott file=d:/scott\_owner.dmp

**2、导出其它方案**

如果用户要导出其它方案，则需要dba的权限或者exp\_full\_database的权限。例如system用户就可以导出任何方案

exp 用户名/密码@数据库实例名 owner=(所有者1,所有者2,..) file=备份路径

例：exp system/orcl@orcl owner=scott,system file=d:/system\_scott\_owner.dmp

**特别注意：如果用户的权限够大，则可一次性导出多外用户的方案。比如system。**

**导出完整数据库**

需要dba的权限或者exp\_full\_database的权限。

exp 用户名/密码@数据库实例名 full=y file=备份路径

例：exp system/orcl@orcl full=y file=d:/orcl\_full.dmp

**导入表**

**导入自身表**

imp 用户名/密码@数据库实例名 file=备份路径 tables=(表名1,表名2,..)

例：imp scott/tiger@orcl file=d:/emp\_dept.dmp

**特别注意：如果目的地有相同的表，则导入不成功。**

**导入其它方案的表**

需要有dba权限或者imp\_full\_database的权限。

imp 用户名/密码@数据库实例名 file=备份路径 tables=(表名1,表名2,..) fromuser=方案名 touser=system

例：imp system/orcl@orcl file=d:/scott\_emp\_dept.demp tables=emp,dept fromuser=scott touser=system

注意：提示成功，但警告，表示此对象有scott导出，而不是当前用户。

**导入方案**

**导入自身方案**

imp 用户名/密码@数据库实例名 file=备份路径

例：imp scott/tiger@orcl file=scott\_owner.dmp

**导入其它方案**

需要有dba权限或者imp\_full\_database权限

imp 用户名/密码@数据库实例名 file=备份路径 fromuser=方案名 touser=system

例：imp system/orcl@orcl file=scott.dmp fromuser=scott touser=system

**导入完整数据库**

需要有dba权限或者imp\_full\_database权限

imp 用户名/密码@数据库实例名 file=备份路径 full=y

例：imp system/orcl@orcl file=orcl\_full.dmp full=y

**oracle的卸载**

1.停止所有与ORACLE相关的服务。

2. 使用OUI（Oracle Universal Installer）卸载Oracle软件。

　“开始”->“程序”->“Oracle-OraDb110g\_home1|Oracle installation product|Universal

installer.

3.删除注册表内容。

运行regedit命令，删除下面内容：HKEY\_LOCAL\_MACHINE|SOFTWARE|ORACLE注册表键，删除此键。

　　HKEY\_LOCAL\_MACHINE|SYSTEM|CurrentControlSet|Services，删除Services键下所有以oracle为首的键。

HKEY\_LOCAL\_MACHINE|SYSTEM|CurrentControlSet|Services|Eventlog|Application， 删除此键下所有以oracle为首的键。

　　HKEY\_CLASSES\_ROOT，删除此键下所有以Ora，Oracle，Orcl，EnumOra 为前缀的键。

HKEY\_CURRENT\_USER|Software|Microsoft|Windows|CurrentVersion|Explorer|MenuOrder|Start Menu|Programs， 删除此键下所有以oracle为首的键。

　　HKEY\_LOCAL\_MACHINE|SOFTWARE|ODBC|ODBCINST.INI注册表键，删除了Microsoft ODBC FOR ORACLE注册表键以外的所有有Oracle字样的键值。

　　HKEY\_LOCAL\_MACHINE|SYSTEM|CurrentControlSet|Services，删除以Oracle或OraWeb为前缀的键。

4. 删除环境变量。

　 删除环境变量CLASSPATH，PATH中含有Oracle字样的值。

5.最后在文件系统内删除ORACLE相关的文件及目录：删除系统盘符：\Progrm Files\Oracle目录；删除ORACLE\_BASE目录。

　无法删除\oracle目录，重新启动机器之后才删除。