Oracle 第四天

一、PL/SQL 编程语言

- **游标（光标 Cursor）**

为什么要使用游标？

示例： 按员工的工种长工资,总裁1000元，经理长800元其，他人员长400元。

```
1  set serveroutput on;
2
3  declare
4    ptilte varchar(20); --记录员工的工种
5  begin
6    select job into ptilte from emp; --得到员工的工种
7    --判断员工的工种
8    if ptitle = 'PRESIDENT' then update emp set sal +  1000;
9    elsif ptitle = 'MANAGER' then update emp set sal + 800;
10   else  update emp set sal = sal + 400;
11   end if;
12 end;
13 /
```

- 问题1：返回多行
- 问题2：没有指定where条件

在写 java 程序中有集合的概念，那么在 pl/sql 中也会用到多条记录，这时候我们就要用到游标，游标可以存储查询返回的多条数据。

语法：

CURSOR 游标名 [(参数名 数据类型,参数名 数据类型,...)] IS SELECT 语句;

例如：`cursor c1 is select ename from emp;`

游标的使用步骤：

- 打开游标： open c1; (打开游标执行查询)
- 取一行游标的值：fetch c1 into pjob; (取一行到变量中)
- 关闭游标： close c1;(关闭游标释放资源)
- 游标的结束方式 exit when c1%notfound
- 注意： 上面的 pjob 必须与 emp 表中的 job 列类型一致：
  定义：pjob emp.empjob%type;

范例 1：使用游标方式输出 emp 表中的员工编号和姓名

```
declare
  cursor pc is
    select * from emp;
  pemp emp%rowtype;
begin
  open pc;
  loop
    fetch pc
      into pemp;
    exit when pc%notfound;
    dbms_output.put_line(pemp.empno || ' ' || pemp.ename);
  end loop;
  close pc;
end;
```

范例 2：写一段 PL/SQL 程序，为部门号为 10 的员工涨工资。

```
declare
  cursor pc(dno myemp.deptno%type) is
    select empno from myemp where deptno = dno;
  pno myemp.empno%type;
begin
  open pc(20);
  loop
    fetch pc
      into pno;
    exit when pc%notfound;
    update myemp t set t.sal = t.sal + 1000 where t.empno =
pno;
  end loop;
  close pc;
end;
```

● 例外

异常是程序设计语言提供的一种功能，用来增强程序的健壮性和容错性。

✓ 系统定义异常

no_data_found (没有找到数据)
too_many_rows (select ...into 语句匹配多个行)
zero_divide (被零除)
value_error (算术或转换错误)
timeout_on_resource (在等待资源时发生超时)

范例 1：写出被 0 除的异常的 plsql 程序

```
declare
  pnum number;
```

```
begin
  pnum := 1 / 0;
exception
  when zero_divide then
    dbms_output.put_line('被0除');
  when value_error then
    dbms_output.put_line('数值转换错误');
  when others then
    dbms_output.put_line('其他错误');
end;
```

✓ 用户也可以自定义异常，在声明中来定义异常

```
DECLARE
    My_job     char(10);
    v_sal      emp.sal%type;
    No_data    exception;
    cursor c1 is select distinct job from emp    order by job;
    如果遇到异常我们要抛出 raise no_data;
```

范例 2：查询部门编号是 50 的员工

```
declare
  no_emp_found exception;
  cursor pemp is
    select t.ename from emp t where t.deptno = 50;
  pename emp.ename%type;
begin
  open pemp;
  fetch pemp
    into pename;
  if pemp%notfound then
    raise no_emp_found;
  end if;
  close pemp;
exception
  when no_emp_found then
    dbms_output.put_line('没有找到员工');
  when others then
    dbms_output.put_line('其他错误');
end;
```

二、存储过程

存储过程（Stored Procedure）是在大型数据库系统中，一组为了完成特定功能的 SQL 语句集，经编译后存储在数据库中，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过程是数据库中的一个重要对象，任何一个设计良好的数据库应用程序都应该用到存储过程。

创建存储过程语法：

```
create [or replace] PROCEDURE 过程名[(参数名 in/out 数据类型)]
AS
begin
        PLSQL 子程序体；
End;
```

或者

```
create [or replace] PROCEDURE 过程名[(参数名 in/out 数据类型)]
is
begin
        PLSQL 子程序体；
End 过程名;
```

范例 1：给指定的员工涨 100 工资，并打印出涨前和涨后的工资

分析：我们需要使用带有参数的存储过程

```
create or replace procedure addSal1(eno in number) is
  pemp myemp%rowtype;
begin
  select * into pemp from myemp where empno = eno;
  update myemp set sal = sal + 100 where empno = eno;
  dbms_output.put_line('涨工资前' || pemp.sal || '涨工资后' ||
(pemp.sal + 100));
end addSal1;
```

调用

```
begin
  -- Call the procedure
  addsal1(eno => 7902);
  commit;
end;
```

三、存储函数

```
create or replace function 函数名(Name in type, Name out type, ...) return 数据类型 is
  结果变量 数据类型;
begin

  return(结果变量);
end[函数名];
```

存储过程和存储函数的区别

一般来讲，过程和函数的区别在于函数可以有一个返回值；而过程没有返回值。

但过程和函数都可以通过 **out** 指定一个或多个输出参数。我们可以利用 **out** 参数，在过程和函数中实现返回多个值。

范例：使用存储函数来查询指定员工的年薪

```
create or replace function empincome(eno in emp.empno%type)
return number is
  psal  emp.sal%type;
  pcomm emp.comm%type;
begin
  select t.sal into psal from emp t where t.empno = eno;
  return psal * 12 + nvl(pcomm, 0);
     end;
```

使用存储过程来替换上面的例子

```
create or replace procedure empincomep(eno in emp.empno%type,
income out number) is
  psal emp.sal%type;
  pcomm emp.comm%type;
begin
  select t.sal, t.comm into psal, pcomm from emp t where
t.empno = eno;
  income := psal*12+nvl(pcomm,0);
end empincomep;
```

调用：

```
declare
  income number;
begin
  empincomep(7369, income);
  dbms_output.put_line(income);
end;
```

四、Java 程序调用存储过程

1.java 连接 oracle 的 jar 包

可以在虚拟机中 xp 的 oracle 安装目录下找到 jar 包 :ojdbc14.jar

2.数据库连接字符串

```
String driver="oracle.jdbc.OracleDriver";
String url="jdbc:oracle:thin:@192.168.56.10:1521:orcl";
String username="scott";
String password="tiger";
```

测试代码：

```
@Test
public void testJdbc(){
    String driver="oracle.jdbc.OracleDriver";
    String url="jdbc:oracle:thin:@192.168.56.10:1521:orcl";
    String username="scott";
    String password="tiger";

    try {
        Class.forName(driver);
        Connection con  = DriverManager.getConnection(url, username, password);

        Statement st = con.createStatement();

        ResultSet rs = st.executeQuery("select * from emp");
        while(rs.next()){
            System.out.println(rs.getObject(1)+","+rs.getObject(2));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

3.实现过程的调用

1.调用过程

1.过程定义

```
--统计年薪的过程
create or replace procedure proc_countyearsal(eno in number,esal
out number)
as
begin
   select sal*12+nvl(comm,0) into esal from emp where empno=eno;
end;


--调用
declare
   esal number;
begin
   proc_countyearsal(7839,esal);
   dbms_output.put_line(esal);
end;
```

2.过程调用

```
@Test
    public void testProcedure01(){
        String driver="oracle.jdbc.OracleDriver";
        String url="jdbc:oracle:thin:@192.168.56.10:1521:orcl";
        String username="scott";
        String password="tiger";

        try {
            Class.forName(driver);
            Connection  con    =  DriverManager.getConnection(url,
username, password);

            CallableStatement  callSt  =  con.prepareCall("{call
proc_countyearsal(?,?)}");
```

```
            callSt.setInt(1, 7839);
            callSt.registerOutParameter(2, OracleTypes.NUMBER);

            callSt.execute();

            System.out.println(callSt.getObject(2));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

4.游标引用的 java 测试

1.定义过程，并返回引用型游标

```
--定义过程
create or replace procedure proc_cursor_ref(dno in number,empList out
sys_refcursor)
as
begin
  open empList for select * from emp where deptno = dno;
end;

--pl/sql 中调用
declare
  mycursor_c sys_refcursor;
  myempc emp%rowtype;
begin
  proc_cursor_ref(20,mycursor_c);

  loop
    fetch mycursor_c into myempc;
    exit when mycursor_c%notfound;
    dbms_output.put_line(myempc.empno||','||myempc.ename);
  end loop;
  close mycursor_c;
end;
```

2.java 代码调用游标类型的 **out** 参数

```
@Test
    public void testFunction(){
        String driver="oracle.jdbc.OracleDriver";
        String url="jdbc:oracle:thin:@192.168.56.10:1521:orcl";
        String username="scott";
        String password="tiger";

        try {
            Class.forName(driver);
            Connection con = DriverManager.getConnection(url,
username, password);

            CallableStatement callSt = con.prepareCall("{call
proc_cursor_ref (?,?)}");

            callSt.setInt(1, 20);
            callSt.registerOutParameter(2, OracleTypes.CURSOR);

            callSt.execute();

            ResultSet rs =
((OracleCallableStatement)callSt).getCursor(2);
            while(rs.next()){

    System.out.println(rs.getObject(1)+","+rs.getObject(2));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

思考：**hibernate** 怎么调用存储过程.

五、触发器

数据库触发器是一个与表相关联的、存储的 **PL/SQL** 程序。每当一个特定的数据操作语句(**Insert,update,delete**)在指定的表上发出时，**Oracle** 自动地执行触发器中定义的语句序列。

1.触发器作用

- 数据确认
  - 示例：员工涨后的工资不能少于涨前的工资
- 实施复杂的安全性检查
  - 示例：禁止在非工作时间插入新员工
- 做审计，跟踪表上所做的数据操作等
- 数据的备份和同步

2.触发器的类型

✓ 语句级触发器 ：

在指定的操作语句操作之前或之后执行一次，不管这条语句影响了多少行 。

✓ 行级触发器（FOR EACH ROW） ：

触发语句作用的每一条记录都被触发。在行级触发器中使用 **old** 和 **new** 伪记录变量，识别值的状态。

语法：

```
CREATE  [or REPLACE] TRIGGER  触发器名
    {BEFORE | AFTER}
    {DELETE | INSERT | UPDATE [OF 列名]}
    ON  表名
    [FOR EACH ROW [WHEN(条件) ] ]
declare
     ……
begin
    PLSQL 块
End 触发器名
```

范例：插入员工后打印一句话“一个新员工插入成功”

```
create or replace trigger testTrigger
  after insert on person
declare
  -- local variables here
begin
  dbms_output.put_line('一个员工被插入');
  end testTrigger;
```

范例：不能在休息时间插入员工

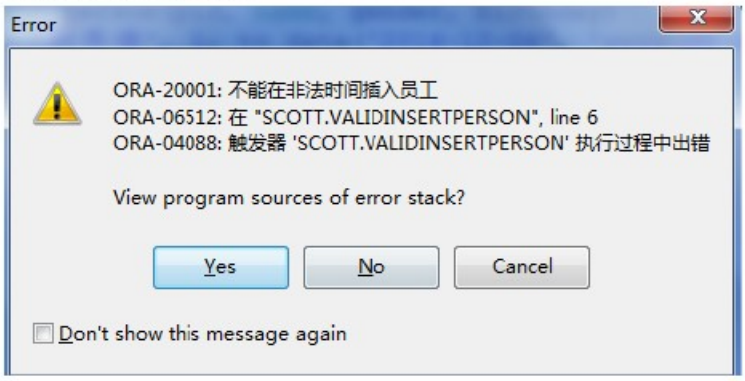
```
create or replace trigger validInsertPerson
```

```
  before insert on person

declare
  weekend varchar2(10);
begin
  select to_char(sysdate, 'day') into weekend from dual;
  if weekend in ('星期一') then

    raise_application_error(-20001, '不能在非法时间插入员工');
  end if;
  end validInsertPerson;
```

当执行插入时会报错



在触发器中触发语句与伪记录变量的值

触发语句	:old	:new
Insert	所有字段都是空(null)	将要插入的数据
Update	更新以前该行的值	更新后的值
delete	删除以前该行的值	所有字段都是空(null)

范例：判断员工涨工资之后的工资的值一定要大于涨工资之前的工资

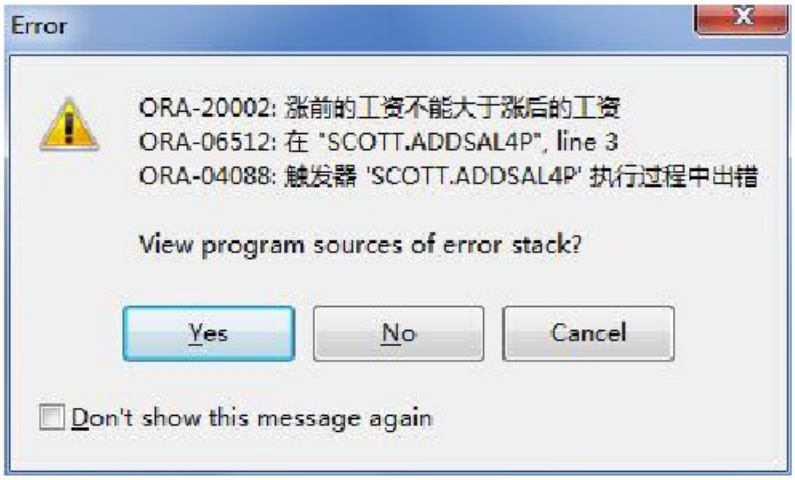
```
create or replace trigger addsal4p
  before update of sal on myemp
  for each row
begin
  if :old.sal >= :new.sal then

    raise_application_error(-20002, '涨前的工资不能大于涨后的工

资');
  end if;
  end;
```

调用

```
update myemp t set t.sal = t.sal - 1;
```



3.触发器实际应用

需求：使用序列，触发器来模拟 **mysql** 中自增效果

1. 创建序列

1、建立表
代码如下:

```
create table user
(
    id   number(6) not null,
    name   varchar2(30)   not null primary key
)
```

2、建立序列 **SEQUENCE**
代码如下:

```
create sequence user_seq;
```

2.创建自增的触发器

分析：创建一个基于该表的 **before insert** 触发器，在触发器中使用刚创建的 **SEQUENCE**。
代码如下:

```
create or replace trigger user_trigger
before insert on user
for each row
```

```
begin
      select   user_seq.nextval  into:new.id from sys.dual ;
end;
```

3.测试效果

```
insert into itcastuser(name) values('aa');
commit;
insert into itcastuser(name) values('bb');
commit;
```