

Oracle 第三天

一、PL/SQL 编程语言

● 什么是 PL/SQL?

PL/SQL (Procedure Language/SQL)

PLSQL 是 Oracle 对 sql 语言的过程化扩展，指在 SQL 命令语言中增加了过程处理语句（如分支、循环等），使 SQL 语言具有过程处理能力。把 SQL 语言的数据操纵能力与过程语言的数据处理能力结合起来，使得 PLSQL 面向过程但比过程语言简单、高效、灵活和实用。

范例 1：为职工涨工资，每人涨 10% 的工资。

```
update emp set sal=sal*1.1
```

范例 2：例按职工的职称长工资，总裁涨 1000 元，经理涨 800 元，其他人员涨 400 元。

这样的需求我们就无法使用一条 SQL 来实现，需要借助其他程序来帮助完成，
也可以使用 pl/sql。

● PL/SQL 的语法

```
declare
    说明部分 (变量说明, 光标申明, 例外说明)
begin
    语句序列 (DML语句) ...
exception
    例外处理语句
End;
/
```

● 常量和变量的定义

❖ 说明变量 (`char, varchar2, date, number, boolean, long`)

```
var1      char(15) ; 说明变量名、数据类型和长度后用分号结束说明语句。
married   boolean :=true ;
psal      number(7,2);
my_name   emp.ename%type; 引用型变量，即my_name的类型与
                           emp表中ename列的类型一样
emp_rec   emp%rowtype; 记录型变量
```

代表一行

补充：--也可以定义常量

➤ 引用变量

```
Myname emp.ename%type;
```

引用型变量，即 my_name 的类型与 emp 表中 ename 列的类型一样

在 sql 中使用 **into** 来赋值

```
declare
    emprec emp.ename%type;
begin
    select t.ename into emprec from emp t where t.empno = 7369;
    dbms_output.put_line(emprec);
end;
```

➤ 记录型变量

```
Emprec emp%rowtype
```

记录变量分量的引用

```
emp_rec.ename:='ADAMS';
declare
    p emp%rowtype;
begin
    select * into p from emp t where t.empno = 7369;
    dbms_output.put_line(p.ename || ' ' || p.sal);
end;
```

● If 语句

语法：

1. IF 条件 THEN 语句1;
语句2;
END IF;

2. IF 条件 THEN 语句序列1;
ELSE 语句序列2;
END IF;

3. IF 条件 THEN 语句;
ELSIF 语句 THEN 语句;
ELSE 语句;
END IF;

范例 1：如果从控制台输入 1 则输出我是 1

```
declare
    pnum number := &num;
begin
    if pnum = 1 then
        dbms_output.put_line('我是1');
    end if;
end;
```

补充：(接收键盘输入)
accept num prompt '请输入一个数字';
(num是地址值，需要地址符&num获取num上的值)

范例 2：如果从控制台输入 1 则输出我是 1 否则输出我不是 1



```
declare
    mynum number := &num;
begin
    if mynum = 1 then
        dbms_output.put_line('我是1');
    else
        dbms_output.put_line('我不是1');
    end if;
end;
```

范例 3:判断人的不同年龄段 18 岁以下是未成年人，18 岁以上 40 以下是成年人，40 以上是老年人

```
declare
    mynum number := &num;
begin
    if mynum < 18 then
        dbms_output.put_line('未成年人');
    elsif mynum >= 18 and mynum < 40 then
        dbms_output.put_line('中年人');
    elsif mynum >= 40 then
        dbms_output.put_line('老年人');
    end if;
end;
```

● 循环

语法:

```
WHILE total <= 25000
LOOP
...
total := total + salary;
END LOOP;
```

```
Loop
EXIT [when 条件];
.....
```

```
End loop
```

```
FOR I IN 1..3
LOOP
语句序列 ;
END LOOP ;
```

比较好用

范例:使用语法 1 输出 1 到 10 的数字

```
declare
    step number := 1;
begin
    while step <= 10 loop
        dbms_output.put_line(step);
        step := step + 1;
    end loop;
end;
```

范例:使用语法 2 输出 1 到 10 的数字

```
declare
    step number := 1;
begin
    loop
        exit when step > 10;
        dbms_output.put_line(step);
        step := step + 1;
    end loop;
end;
```

范例:使用语法 3 输出 1 到 10 的数字

```
declare
    step number := 1;
begin
    for step in 1 .. 10 loop
        dbms_output.put_line(step);
    end loop;
end;
```

表示1到10

● 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**800**元其, 他人员**400**元。

```

1 set serveroutput on;
2
3 declare
4     ptitle varchar(20); --记录员工的工种
5 begin
6     select job into ptitle 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;

Cursor是静态游
标, 适用于静态sql
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;
    type emprowtype;
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(10);
    loop
        fetch pc
        into pno;
        exit when pc%notfound;
        update myemp t set t.sal =
    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)

```

-- 给员工涨工资，总裁1000 经理800 其他400
declare
    -- 定义光标
    cursor cemp is select empno,job from emp;
    pempno emp.empno%type;
    pjob   emp.job%type;
begin
    -- 打开光标
    open cemp;
    loop
        -- 取一个员工
        fetch cemp into pempno,pjob;
        exit when cemp%notfound;

        -- 判断职位
        if pjob = 'PRESIDENT' then update emp set sal=sal+1000 where empno=pempno;
        elsif pjob = 'MANAGER' then update emp set sal=sal+800 where empno=pempno;
        else update emp set sal=sal+400 where empno=pempno;
        end if;

    end loop;
    -- 关闭光标
    close cemp;
end;

```

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;
        pname emp.ename%type;
begin
    open pemp;
    fetch pemp
        into pname;
    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

把sql的逻辑写在plsql中，可以提高执行效率。但plsql是不能直接被java调用，但java可以直接调用‘存储过程’和‘函数’，而‘存储过程’和‘函数’也是用plsql写的。

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;
```

无需指定位数，只需写数据类型！
比如：直接写varchar，不能写varchar(20)

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

完全可以由存储过程代替

```
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();
    }
}
```

补充：程序包

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;
```

系统定义的动态游标，主要用在过程中返回结果集。

这里也可以使用程序包的形式，
在包声明中来定义自定义的游标和存储过程，
在包体中实现这个存储过程。



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();
    }
}
```

五、触发器

触发器也是存储过程

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：不论影响多少行都只执行一次

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。

影响多少行就执行多少次。

另一区别：行级触发器可以通过: old 和: new 来获取更新前后的值。

语法：而语句级不能，这也是因为语句级触发器影响多行只执行一次，无法确定是哪一个行。

```
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

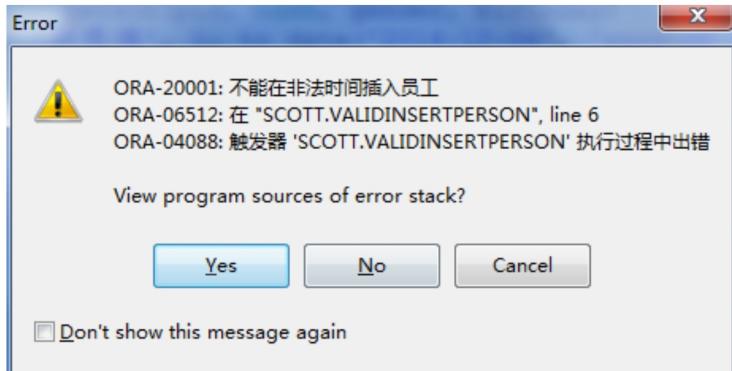
```
create or replace trigger validInsertPerson
```



```
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

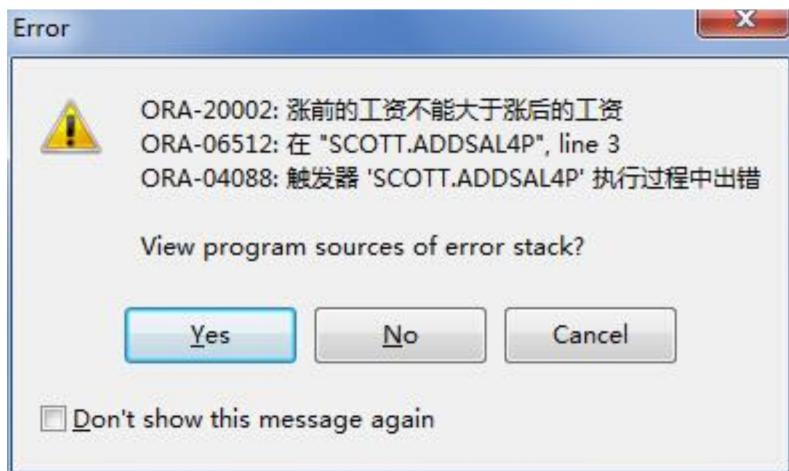
触发语句	:old	:new
Insert	所有字段都是空(null)	将要插入的数据
Update	更新以前该行的值	更新后的值
delete	删除以前该行的值	所有字段都是空(null)

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

复制代码 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 increment by 1 start with 1 minvalue 1 maxvalue 999999999999
nocache order;
```

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;
```