**达内**

**郑州嵩山路中心**

**Java方向面试题知识点**

**编辑：何新峰\魏凯**

**如您身边有想要学习或了解技术的请联系**

**项目经理: 魏老师 电话:18739933113**

**PL/SQL**

PL/SQL也是一种程序语言，叫做过程化SQL语言（Procedural Language/SQL）,是Oracle对标准数据库语言SQL的过程化扩充，它将数据库技术和过程化程序设计语言联系起来，使SQL成为一种高级程序设计语言，支持高级语言的块操作，条件判断，循环语句，嵌套，异常处理,过程和函数等，与数据库核心的数据类型集成，使SQL 的程序设计效率更高.

**--PL/SQL结构**

1 declare

2 声明部分

3 begin

4 执行部分

5 exception

6 异常处理部分

7 end

**说明:**

行1: declare是声明关键字,表示要在行2部分进行声明一些变量等.  
行2: 该部分可以声明变量,游标等.   
行3: BEGIN关键词表明PL/SQL体的开始。   
行4: 该部分是PL/SQL要执行的具体逻辑体

行5: exception表示捕获异常

行6: 该部分是捕获到的异常信息进行处理  
行7: END关键词表明PL/SQL体的结束

注意: declare和exception可选部分

**--第一个PL/SQL语句块**

declare

uname varchar2(10) default '张三';

age number default 20;

height int :=170;

sal number(7,2):=5000.55;

begin

dbms\_output.put\_line(uname ||'的年龄是'||age ||'岁,身高'||height||'cm,工资是'||sal);

end;

执行前需要先开打控制台输出,执行命令: set serveroutput on;

注:以上PL/SQL语句块执行完毕会输出: 张三的年龄是20岁,身高170cm,工资是5000.55 dbms\_output.put\_line():相当于java中的输出语句,将在控制台输出结果

dbms\_ out\_put:数据库管理系统中的输出类库

put\_line();类库中的一个内置输出函数,用于向控制台输出结果

**分解说明:**

declare

uname varchar2(10) default '张三';

age number default 20;

height int :=170;

sal number(7,2):=5000.55;

-- declare关键字表示声明方框选中的变量

--uname,age,height,sal表示是变量名

--varchar2,number,int是变量的类型,其中varchar2 是字符串类型,number是数值类型的

父类型,可以包含整数和小数,int是number的扩展类型表示整数.

--default:表示uname变量的默认值是张三,:=表示声明后赋值,两种用法一样,等号前边有:

是为了区分where 语句后的=

特别提示:声明变量时不能声明关键字或者已存在的内置函数名

begin

dbms\_output.put\_line(uname ||'的年龄是'||age ||'岁,身高'||height||'cm,工资是'||sal);

end;

方框选中的是逻辑体,可以写分支,循环,SQL语句等逻辑结构,这里是有的是输出语句.

||是字符串拼接符,相当于java中的+

**--PL/SQL流程控制**

**--if分支:语法如下**

if 条件 then

. . .

end if;

**--if else:语法如下**

if 条件 then

. . .

else

. . .

end if;

**--if else if:语句如下**

if 条件 then

. . .

elsif 条件 then

. . .

end if;

案例:

1.判断张三和李四的年龄

declare

zhangsan number(3):=30;

lisi number(3):=25;

begin

if zhangsan >lisi then

dbms\_output.put\_line('张三比李四年龄大');

end if;

end;

2.判断两个值的大小

declare

a number(3):=10;

b number(3):=50;

begin

if a>b then

dbms\_output.put\_line('a大于b');

else

dbms\_output.put\_line('a小于b');

end if;

end;

3.判断每个月有多少天

declare

month number(2):=2;

begin

if month=1 or month=3 or month=5 or month=7 or month=8 or month=10 or

month=12 then

dbms\_output.put\_line(month || '月份有31天');

elsif month=2 then

dbms\_output.put\_line(month || '月份有28天');

else

dbms\_output.put\_line(month || '月份有30天');

end if;

end;

**--CASE WHEN:**相当于java中的switch ... case分支

语法:

case 匹配变量

when 匹配值 then 执行语句

when 匹配值 then 执行语句

else 默认执行语句

end case;

或者

case

when 条件 then 执行语句

when 条件 then 执行语句

else 默认执行语句

end case;

案例:查询成绩等级所对应的分值范围

declare

grade char(1):='A'

begin

case grade

when 'A' then

dbms\_output.put\_line('90~100分为A等级');

when 'B' then

dbms\_output.put\_line('80~89分为B等级');

when 'C' then

dbms\_output.put\_line('70~79分为C等级');

else

dbms\_output.put\_line('70分以下为D等级');

end case;

end;

或

declare

grade char(1):='A'

begin

case

when grade='A' then

dbms\_output.put\_line('90~100分为A等级');

when grade= 'B' then

dbms\_output.put\_line('80~89分为B等级');

when grade ='C' then

dbms\_output.put\_line('70~79分为C等级');

else

dbms\_output.put\_line('70分以下为D等级');

end case;

end;

第二种方式相对灵活,when 后面可以跟任意条件语句,只要为true既能执行then后面的执行语句.

**--PL/SQL循环控制**

--基本循环:loop

loop

循环体

end loop;

注意:该循环如果不加条件则是死循环,所以一般搭配IF语句使用

案例:循环输出1-10的值

declare

x number:=1;

begin

loop

x:=x+1;

if x >=10 then

exit;

end if;

dbms\_output.put\_line(x);

end loop;

end;

注意:exit是结束循环,相当于java中的break;

--while循环

while 条件 loop

循环体

end loop;

案例:使用while循环输出求1-10的和

declare

x number:=1;

resu number:=0;

begin

while x>10 loop

resu:=resu+x;

x:=x+1;

end loop;

dbms\_output.put\_line('1-10的和为:' || result);

end;

--loop exit when:该循环,类似于java中的do while,但又有区别,when 循环条件为退

出的条件,而不是再次循环的条件,当条件为true则退出循环

loop

循环体

exit when 循环条件;

end loop;

案例:声明i和total变量均为0,total等于自身加上i的值,当total 的值>1000的时候,i的值是几

declare

i number:=0;

total number:=0;

begin

loop

i:=i+1;

total:=total+i;

exit when total>1000;

end loop;

dbms\_output.put\_line('当total的值是'||total||'时,i的值是'||i);

end;

--for 循环

for 循环变量 in [reverse]起始值..终止值

loop

循环体

end loop;

注:reverse反转的意思,可以让循环变量的值反转执行,比如1-10,变为10-1递减.

案例:输出循环变量

begin

for i in 1 .. 10 loop

dbms\_output.put\_line(i);

end loop;

end;

案例:倒序输出循环变量

begin

for i in revsere 1 .. 10 loop

dbms\_output.put\_line(i);

end loop;

end;

案例:循环输出变量,跳过3和5

begin

for i in 1 ..10 loop

if i=3 or i=5 then

continue;

end if;

dbms\_output.put\_line(i);

end loop;

end;

**--声明常量:使用constant关键字声明,声明过的常量值不能改变,声明的同时赋值**

案例:

declare

pi constant number:=3.1415926;

begin

dbms\_output.put\_line(pi);

pi:=pi+1;

end;

注意:以上案例会报错:报错的原因就是声明的pi是常量,但是在语句块内尝试改变pi的值,所以报错.

**--PL/SQL中的数组/集合/枚举类型**

**--PL/SQL中提供了叫VARRAY的数据结构,可以有序的存储相同类型的一组变量的集合.**

语法:

create or replace type 数组类型名 is varray(3) of 数组元素类型;

案例:

create or replace type names is varray(3) of varchar2(10);

以上案例的意思是创建了一个叫names的数组,长度为3,里边的每一个元素都是varchar2类型,初始值为null;

--给数组赋值

declare

name1 names:=names('tom','andy','jack');

begin

dbms\_output.put\_line('name1的长度为:'||name1.count||',第一个元素是:'||name1(1));

end;

解释:以上案例中的name1.count用来获取数组长度类型,name1(1)是获取第一个元素的值,数据库中VARRAY从1开始计数,若想修改声明过的VARRAY的长度可以用ALTER TYPE进行修改.

**--另外一种创建方式,在declare中声明,比较常用**

declare

type nums is varray(5) of integer;

num1 nums:=nums(10,20,50,8,9);

begin

dbms\_output.put\_line('num1的长度是:'||num1.count);

for i in 1..num1.count loop

dbms\_output\_put\_line(num1(i));

end loop;

end;

区别:声明在declare中的VARRAY类型只对当前PL/SQL语句块有效,使用create type创建的对所有PL/SQL语句块都有效.

**--利用表或列快速定义数据类型**

--具体方法:

--变量名 表名%rowtype:可以使用该表中一条记录为一个类型,包含其各个字段的类型

--变量名 表名.列名%type:使用该表中的某个字段的类型,使用的时候直接是变量名

案例:查询出emp表中指定编号的员工姓名以及指定编号的员工所有信息赋值给eno和names和emps,并输出.

declare

eno emp.empno%type:=7369;

names emp.ename%type;

emps emp%rowtype;

begin

select ename into names from emp where empno= eno;

dbms\_output.put\_line(names);

select \* into emps from emp where empno=7369;

dbms\_output.put\_line(emps.empno||','||emps.ename||','||emps.job ||','||emps.sal);

end;

解释:

declare

eno emp.empno%type:=1001;

names emp.ename%type;

emps emp%rowtype;

声明eno变量类型引用emp表中empno字段的类型,为number类型.

声明names变量,类型引用emp表中ename字段的类型,为varchar2类型

声明emps变量,类型引用emp表中一条记录的所有字段的类型,存储的也就是该条记录的所有字段信息.

begin

select ename into names from emp where empno= eno;

dbms\_output.put\_line(names);

select \* into emps from emp where empno=7369;

dbms\_output.put\_line(emps.empno||','||emps.ename||','||emps.job ||','||emps.sal);

end;

两个框里选中的部分:

ename into names:将查询到的ename通过into赋值给names变量

\* into emps:将查询出来的该条记录的所有字段通过into保存到emps变量中,在使用的时候通过该变量名点字段名即可.

**--异常操作**

PL/SQL支持程序员在程序中使用异常块捕获程序运行中出现的错误等异常信息,分为:

* 系统定义的异常
* 用户定义的异常

**--系统级异常**

--语法如下:

begin

exception

when 异常名then

语句...

when others then

语句...

end;

以下是oracle数据库中系统定义好的异常的名称和产生的原因:

|  |  |
| --- | --- |
| 异常名 | 原因 |
| ACCESS\_INTO\_NULL | 当一个空对象会自动分配一个值引发此异常 |
| CASE\_NOT\_FOUND | 当没有任何选择，在当选择一个CASE语句的子句且没有ELSE子句时被引发 |
| COLLECTION\_IS\_NULL | 当程序试图申请其他收集方法不是存在未初始化的嵌套表或VARRAY，或程序试图值分配给未初始化的嵌套表或变长数组的元素时被引发 |
| DUP\_VAL\_ON\_INDEX | 当重复值试图被存储在具有唯一索引的列时被引发 |
| INVALID\_CURSOR | 当尝试以使这是不允许的，如关闭一个未打开的游标的游标操作时被引发 |
| INVALID\_NUMBER | 当一个字符串转换成一个数失败，因为字符串不表示一个有效的数据被引发 |
| LOGIN\_DENIED | 当程序试图登录到数据库使用无效的用户名或密码被引发 |
| NO\_DATA\_FOUND | 它被引发当一个SELECT INTO语句无任何行返回 |
| NOT\_LOGGED\_ON | 在不连接到数据库发出数据库调用它被引发 |
| PROGRAM\_ERROR | 它被引发当PL/SQL有一个内部问题 |
| ROWTYPE\_MISMATCH | 当游标取值有不兼容的数据类型的变量被引发 |
| SELF\_IS\_NULL | 它被引发，当对象的成员方法被调用，但对象类型的实例没有被初始化。 |
| STORAGE\_ERROR | 它被引发当PL/SQL内存不足或内存已损坏 |
| TOO\_MANY\_ROWS | 它被引发当SELECT INTO语句返回多行 |
| VALUE\_ERROR | 当算术，转换，截短，或大小约束错误时引发 |
| ZERO\_DIVIDE | 它被引发当一个数试图除以零。 |

案例:

查询部门表中有某个编号的部门,给其一个不存在的编号来演示NO\_DATA\_FOUND异常:该异常在SELECT INTO语句查询不到结果时触发.

另外给其一个能够返回多个行记录的编码来演示TOO\_MANY\_ROWS异常:该异常在SELECT INTO语句查询后返回多个行记录时触发.

declare

dnum number:=60;

depts dept%rowtype;

dna dept.dname%type;

begin

select \* into depts.deptno,dna,depts.loc from dept where deptno>dnum;

dbms\_output.put\_line('部门编号:'||depts.deptno||',部门名称:'||dna||',城

市:'||depts.loc);

exception

when NO\_DATA\_FOUND then

dbms\_output.put\_line('没有查找到数据');

when TOO\_MANY\_ROWS then

dbms\_output.put\_line('返回了多个行记录');

when others then

dbms\_output.put\_line('其他异常');

end;

**--自定义异常**

自定义异常需要现在declare中声明,类型为exception类型,然后需要在begin中给其设置触发条件,然后就可以在exception后的when中使用.

自定义异常三步骤:

1.declare中定义异常

2.begin中设置触发条件,由raise关键字触发.注: raise中文发音:瑞爱Z

3.exception中使用

案例:查询指定id的员工名字来演示自定义异常

declare

cid number:=-1;

cname emp.ename%type;

ex\_invalid\_id exception; ---1.定义异常

begin

if cid <0 then ---2.设定触发条件

raise ex\_invalid\_id;

end if;

select ename into cname from emp where empno=cid;

dbms\_output.put\_line('员工名字为:'||cname);

exception

when ex\_invalid\_id then ---3.使用异常

dbms\_output.put\_line('id不可用,不能为负数');

when no\_data\_found then

dbms\_output.put\_line('未查到任何数据');

when others then

dbms\_output.put\_line('其他异常,请联系管理员');

end;

**游标 Cursor**

**Cursor 中文发音:可搜儿**

游标是系统为用户在内存中开设的一个数据缓冲区,也成为上下文区域，用于处理SQL语句的相关信息比如行数处理等.

游标分为:

隐式游标: 在DML语句执行时,当没有显式游标的语句时隐式游标自动创建,程序员无法控

制隐式游标在内存中的信息,只能获取一些简单的影响行等信息。

显式游标: 由程序员定义游标来获得更多对游标的控制信息,显式游标应在PL/SQL块的声

明部分中定义,一般在SELECT语句中使用.l

**--隐式游标:隐式游标的名字为系统默认的为SQL**

--常用属性

SQL%isopen 判断游标是否打开

SQL%rowcount 对于增删改,是受影响行数,对于select无意义,只有返回1行时才能读到该

值,值为1

SQL%found 布尔值,是否查到有数据的行

SQL%notfound 与found相反

案例:对20部门的员工涨薪100,然后查看该DML语句游标影响的行数

begin

update emp set sal=sal+100 where deptno=20;

if SQL%found then

dbms\_output.put\_line('修改了'||SQL%rowcount||'行');

end if;

end;

**--显示游标:cursor,需要自己声明,一般是用在select语句**

游标名%isopen 判断游标是否打开

游标名%rowcount 当前已fetch到的行

游标名%found 上次fetch是否得到数据

游标名%notfound 与found相反

--显示游标使用四步骤

1.声明游标:用于初始化游标

2.打开游标:分配内存

3.获取数据:从游标中获取信息

4.关闭游标:释放内存

语法:

declare

cursor 游标名 is select 语句;

begin

open 游标名;

fetch 游标名 into 变量名;

close 游标名;

end;

案例:创建一个显示游标存储查询部门信息结果集的相关信息,然后遍历游标,取出每个部门的编号及名称

declare

cursor depts is select \* from dept; --声明游标

e dept%rowtype;

begin

open depts; --打开游标

loop

fetch depts into e; --获取游标中信息

exit when depts%notfound;

dbms\_output.put\_line('部门编号:'||e.deptno||',部门名称:'||e.dname);

end loop;

dbms\_output.put\_line('一共查询到了'||depts%rowcount||'行');

close depts; --关闭游标

end;

**存储函数 FUNCTION**

**FUNCTION中文发音:范克绅**

**--存储函数有返回值类型和返回值,存储过程没有返回值**

案例1:比较两个数,并返回最大值

CREATE OR REPLACE FUNCTION maxnum(a int,b int) return int

AS

max1 number:=0;

BEGIN

if a>b then

max1:=a;

else

max1:=b;

end if;

RETURN max1;

END;

**调用:**

BEGIN

dbms\_output.put\_line(maxnum (5,10));

END;

**存储函数一般在sql语句中使用:**

select maxnum (5,10) from dual;

案例2:查询某个员工的年薪(包含月薪和奖金)

CREATE OR REPLACE FUNCTION queryyearsal(eno in number)

RETURN number

AS

--定义两个变量保存员工的薪水和奖金

psal emp.sal%type;

pcomm emp.comm%type;

BEGIN

--得到该员工的月薪和奖金

SELECT sal,comm INTO psal,pcomm FROM emp WHERE empno=eno;

--直接返回年薪

RETURN psal\*12+NVL(pcomm,0);

END;

**存储函数一般在sql语句中使用:**

select queryyearsal(7369) from dual;

**存储过程 Stored Procedure**

**Procedure中文发音:扑若谁酒**

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，存储在数据库中，经过第一次编译后再次调用不需要再次编译，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。

**Oracle存储过程基本语法:**

　　1 CREATE OR REPLACE PROCEDURE 存储过程名   
　　2 AS    
　　3 BEGIN   
　　4 NULL;   
　　5 END;

**说明:**

行1: CREATE REPLACE PROCEDURE 是一个SQL语句通知Oracle数据库去创建一个叫

做某某某名字的存储过程, 如果存在就覆盖它;   
行2: AS关键词表明后面将跟随一个PL/SQL体。   
行3: BEGIN关键词表明PL/SQL体的开始。   
行4: NULL PL/SQL语句表明什么事都不做，这句不能删去，因为PL/SQL体中至少需要有

一句,该处的NULL可以替换为要具体执行的SQL语句  
行5: END关键词表明PL/SQL体的结束

**--第一个存储过程,打印helloworld**

CREATE OR REPLACE PROCEDURE sayhelloworld

AS

BEGIN

dbms\_output.put\_line('Hello World');

END;

以上存储过程创建完毕,在调用后会在控制台上打印出Hello World

注: dbms\_output.put\_line('Hello World');

dbms\_ out\_put:数据库管理系统中的输出类库

put\_line();类库中的一个内置输出函数,用于向控制台输出结果

**在执行前需要使用如下命令打开控制台输出:**

set serveroutput on;

**打开输出后可以执行调用存储过程,有两种方式:**

1)使用exec 关键字跟上要执行的存储过程名

exec sayhelloworld();

2)使用begin end

begin

sayhelloworld();

end;

3)使用call关键字跟上要执行的存储过程名

call sayhelloworld();

**--第二个存储过程(带参)**

**--为指的员工涨薪100块钱,并且打印涨前和涨后的薪水?**

CREATE OR REPLACE PROCEDURE raisesalary (eno in number)

AS

--定义一个变量来存储涨前的薪水

psal emp.sal%type;

BEGIN

SELECT sal into psal from emp where empno=eno; --得到员工涨前的薪水

update emp set sal =sal+100 where empno=eno; --给员工涨薪100

dbms\_output.put\_line('涨前:'||psal|| '涨后:'||(psal+100)); --打印

END;

--调用有参存储过程

BEGIN

raisesalary(7369);

raisesalary(7521);

commit;

END;

**如上存储过程是一个带参数的存储过程:**

CREATE OR REPLACE PROCEDURE raisesalary (eno in number)

**跟在存储过程名后的括号为传入的参数:**

eno 是参数名, in 表示为入参,number为入参的类型

**定义变量:**

psal emp.sal%type;

psal:定义的变量名

emp.sal%type:前面定义的变量与emp表中sal字段的类型自动匹配(%type匹配类型)

**into关键字:**

SELECT sal into psal from emp where empno=eno;

into前面跟表中查询后要显示的字段,后面跟要赋值的变量名

上述sql语句的意思就是:

将员工表中符合where过滤条件的员工的工资查询出来赋值给psal变量.

--注意:一般不在存储过程和函数中使用commit和rollback,因为过程和函数是被调用的

--如果在过程中和函数中提交了事务,则无法保证过程和被调用时在一个事务中

--应该有谁调用谁提交

**--关于in,out,in out三种不同参数类型的解释**

--in:调用时传入的参数,但在存储过程内部不能再次更改传入参数的值

--out:调用时无法接收传入的参数,作为传出的参数给调用前声明的变量

--in out:既能作为入参也能作为传出参数

案例:--in

create or replace procedure p\_age(age in number)

as

begin

dbms\_output.put\_line('内:'||age);

age:=50; --不能再次更改,会报错,去掉后编译通过

end;

调用:

declare

age1 number:=30;

begin

p\_age(age1);

dbms\_output.put\_line('外:'||age1);

end;

此时输出的结果为:内:30,外:30

由于p\_age中声明的age是接收传入的参数,age接收到了传入的age1的值,第一次输30,内部不能修改age的值也不能回传,所以两次输出都是调用时声明的age1的值.

案例:--out

create or replace procedure p\_age1(age out number)

as

begin

dbms\_output.put\_line('内:'||page);

age:=50;

end;

调用:

declare

age1 number:=30;

begin

p\_age1(age1);

dbms\_output.put\_line('外:'||age1);

end;

此时输出的结果为:内: ,外:50

由于p\_age1中声明的age是传出的参数,内部不能接收调用传入的age1的值,所以输出空,在过程内部改变age的值,类型为传出,则将修改后age的值传出赋给age1,所以输出50;

案例:--in out

create or replace procedure p\_age2(age in out number)

as

begin

dbms\_output.put\_line('内:'||age);

age:=50;

end;

调用:

declare

age1 number:=30;

begin

p\_age2(age1);

dbms\_output.put\_line('外:'||age1);

end;

此时输出的结果为:内:30,外:50

由于p\_age2中声明的age是传入,传出的参数,过程内部既能接收到外部传入过来的age1的值,第一次输出30,然后重新给age进行赋值,然后将值传出给age1,第二次输出则是修改后的值.

**--第三个存储过程(无参:即在过程名后不加括号以及传入参数即可)**

**--查询10号部门的平均工资?**

CREATE OR REPLACE PROCEDURE deptavgsal

AS

dno emp.deptno%type;

avgsal emp.sal%type;

BEGIN

SELECT deptno,AVG(sal) INTO dno,avgsal FROM emp GROUP BY deptno

HAVING deptno=10;

dbms\_output.put\_line(dno||':'||avgsal);

END;

**调用:**

BEGIN

deptavgsal();

END;

**触发器 TRIGGER**

**TRIGGER中文发音:吹哥儿**

触发器是一个特殊的存储过程,但并不是人为主动调用的,而是通过相关的事件触发的该过程,每当一个特定的(DML)数据库操作语句(insert,update,delete),在指定的表上发出时,oracle自动执行触发器中定义的语句序列.

触发事件：引起触发器被触发的事件。

例如：DML语句(INSERT, UPDATE, DELETE语句对表或视图执行数据处理操作)、DDL语句（如CREATE、ALTER、DROP语句在数据库中创建、修改、删除模式对象）、数据库系统事件（如系统启动或退出、异常错误）、用户事件（如登录或退出数据库）。

注:selete语句是没有触发器的

--第一个触发器

--每当成功插入新员工后自动打印"成功插入新员工"

CREATE TRIGGER saynewemp

AFTER INSERT --INSERT针对插入操作 after在插入执行之后调用PL/SQL

ON emp --on指明那一张表

DECLARE --PL/SQL程序以declare开始,该declare可省略

BEGIN

dbms\_output.put\_line('成功插入新员工');

END;

通过如下insert事件触发:

INSERT INTO emp(empno,ename,sal,deptno) VALUES(1001,'Tom',3000,10);

**触发器的语法:**

CREATE OR REPLACE TRIGGER 触发器名

{BEFORE|AFTER} --在之前还是之后执行

{DELETE|INSERT|UPDATE[OF 列名]}--更新操作可执行列名,比如列名是薪水,只有更新薪

--水的时候才会被触发

ON 表名

[FOR EACH ROW [WHEN(条件)]]--指明触发器的类型:有是行级,没有则是语句级,WHEN

--条件,当WHEN old.deptno=10时只有部分是10号时

--才会触发PL/SQL块

**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,语

句触发器针对的是表.

**行级触发器**:触发语句执行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.行级触发器针对的是行

比如:insert into emp10 select\*from emp where deptno=10;

假设得到3条记录

语句级触发器: 如上语句,如果在emp10表上定义了语句级触发器,向emp10表中插入查询出来的3条记录只会触发一次触发器

行级触发器:如果在emp10上针对插入操作定义了行级触发器,会触发调用3次,行级的对象时每一行,插入三行会触发3次触发器.

案例:

准备工作,复制表不带数据(大家各自建一张自己的表,因为一张表只能12个触发器)

create table emp10 as select \* from emp where 1=2;

查询一下是否复制成功:

select \* from emp10;

--语句级触发器:不加for each row

create or replace trigger insertdatatrig1

after insert

on emp10

begin

dbms\_output.put\_line('成功插入一条数据');

end;

测试:

insert into emp10 select \* from emp where deptno=10;

结果:以上插入语句查3条数据,仅提示了一次,说明语句级触发器针对的是语句和表本身.

--语句级触发器:加for each row

create or replace trigger insertdatatrig2

after insert

on emp10

for each row

begin

dbms\_output.put\_line('成功插入一条数据');

end;

测试:

insert into emp10 select \* from emp where deptno=20;

结果:以上插入语句查5条数据,提示了5次,说明行级触发器针对的是每一行的数据.

**--触发器触发次序**

1. 执行 BEFORE语句级触发器;

2. 对与受语句影响的每一行：

执行 BEFORE行级触发器

执行 DML语句

执行 AFTER行级触发器

3. 执行 AFTER语句级触发器

**--行级触发器的伪记录变量:old,:new**

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、 后相关字段的值,可以交给:old和:new来记录

:old:用来记录操作前的字段值

:new:用来记录操作后的字段的值

|  |  |  |  |
| --- | --- | --- | --- |
| 伪记录变量 | | | |
| 特性 | INSERT | UPDATE | DELETE |
| :old | null | 实际值 | 实际值 |
| :new | 实际值 | 实际值 | null |
|  |  |  |  |

案例:

计算涨薪前和涨薪后的薪水差是多少?

create or replace trigger subtract

after update of sal

on emp10

for each row

declare

resu number;

begin

resu:=:new.sal-:old.sal;

dbms\_output.put\_line('涨薪前:'||:old.sal||', 涨薪后:'||:new.sal||',本次涨薪:'||resu);

end;

测试结果:

update emp10 set sal=sal+1000 where empno=7369;

控制台输出:

涨薪前:1500, 涨薪后:2500,本次涨薪:1000

**--RAISE\_APPLICATION\_ERROR**:

异常信息处理函数:是将异常信息返回给调用的客户端

之前我们一直使用的dbms\_output.put\_line()输出语句函数,该语句是在控制台上打印提示信息,并不能把信息返回给应用程序运行过程中的调用者.若想把相关信息返回给调用者并且中断执行操作,那么可以使用RAISE\_APPLICATION\_ERROR.

语法:

raise\_application\_error( error\_numbe in number, error\_msg in varchar2);

其中:

error\_numbe: 用于定义不正确号，该不正确号必须在-20000到-20999之间的负整数；

error\_msg: 用于指定不正确消息，并且该消息的长度无法超过2048字节；

案例: 不允许对ename是KING的员工进行任何操作

create or replace trigger executedisabled

before update or delete or insert

on emp10

for each row

when (old.ename='KING')

begin

raise\_application\_error(-20001,'禁止操作该员工');

end;

测试:

update emp10 set sal=sal+100 where ename='KING';

案例:禁止向数据库中插入年龄小于18岁的员工

--emp表中有hiredate字段,DATE类型

--ADD\_MONTHS(时间参数,增加多少个月):在时间参数基础上增加多少个月

create or replace trigger illegalage

before insert

on emp10

for each row

begin

if ADD\_MONTHS(:new.hiredate,18\*12)>sysdate then

raise\_application\_error(-20002,'年龄不满18岁不能录用');

end if;

end;

测试:

insert into emp10(empno,ename,hiredate)

values(1001,'tom',to\_date('1999-10-10','yyyy-mm-dd'));

**--条件谓词**

条件谓词：当在触发器中包含多个触发事件（INSERT、UPDATE、DELETE）的组合时，为了分别针对不同的事件进行不同的处理，需要使用ORACLE提供的如下条件谓词。

|  |  |
| --- | --- |
| 触发事件的条件谓语 | |
| inserting | 触发事件是INSERT时，取值为true，否则为false |
| updating('字段') | 触发事件是UPDATE且修改的是指定字段时,取值为true,否则为false,指定字段可选 |
| deleting | 触发事件是DELETE时，取值为true，否则为false |

案例:多重触发事件禁止10号部门员工的新增,删除和工资降薪和降奖金.

create or replace trigger test\_iud\_trig

before insert or update of sal,comm or delete

on emp10

for each row

begin

case

when inserting then

if :new.deptno=10 then

raise\_application\_error(-20003,'10号部门不需要增加新员工');

end if;

when updating('sal') then

if :old.sal>:new.sal and :old.deptno=10 then

raise\_application\_error(-20004,'10号部门的员工工资不能降');

end if;

when updating('comm') then

if nvl(:old.comm,0)>:new.comm and :old.deptno=10 then

raise\_application\_error(-20005,'10号部门的员工奖金不能降');

end if;

when deleting then

if :old.deptno=10 then

raise\_application\_error(-20006,'10号部门的员工不能删除');

end if;

end case;

end;

**--触发器的四个应用场景:**

**1.复杂的安全性检查:**比如:禁止在非工作时间插入新员工

**2.数据库的确认:**比如:涨工资,工资应该越长越多的,如果越长越少就不叫涨工资了

**3.数据库审计:**比如:跟踪表上操作的记录,比如什么时间什么人操作了数据库,操作了表上的 记录是什么等

**4.数据库的备份和同步:**比如有两个数据库一个在北京一个在上海,在北京的数据库是主数据库,在上海的数据库是备用数据库,在主数据库中的数据被修改了以后可以通过触发器监听,如果被修改会将修改的数据传递给备份数据库,当主数据崩溃以后不影响数据的使用

**触发器使用场景一:**

复杂的安全性检查

禁止在非工作时间插入新员工

--周末:select to\_char(sysdate,'day') from dual in ('星期六','星期日');

--上班前,下班后:select to\_number(to\_char(sysdate,'hh24')) not between 9 and 18;

--不管插入10个员工还是1个员工,插入操作都只是对表操作一次,所以使用语句级触发器

CREATE OR REPLACE trigger securityemp

before insert

on emp

begin

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

to\_number(to\_char(sysdate,'hh24'))not between 9 and 18 then

--禁止insert新员工

raise\_application\_error(-20007,'禁止在非工作时间插入新员工');

--自定义的错误代码比如在-20000-20009区间

end if;

end;

执行如下插入语句时当不满足9:00-18:00之间或者在非周末时间会触发以上触发器

insert into emp(empno,ename,sal,deptno) values(1002,'Tom',3000,10);

**触发器使用场景二:数据的确认**

涨工资不能越涨越少

涨工资的时候可能给1个员工涨工资,也可能给很多个员工涨工资,对每一条记录都要做检查

所以要使用行级触发器

CREATE OR REPLACE trigger checksalary

before update --涨工资之前需要检查一下涨后的工资是否少于涨钱的工资

on emp

for each row

begin

--if 张后的薪水<涨钱的薪水 then

if :new.sal<:old.sal then

raise\_application\_error(-20008,'涨后的薪水不能少于涨前的薪水,涨之后的薪水:'||:new.sal||' 涨之前的薪水:'||:old.sal);

end if;

end;

--:ord和:new 他们代表的是同一条记录,

--:ord是表示操作该行之前,这一行的值

--:new是表示操作该行之后,这一行的值

执行下面更新语句不会触发触发器:

update emp set sal=sal+1 where empno=7839;

执行下面更新语句会触发触发器,因为修改后的工资比修改前要低

update emp set sal=sal-1 where empno=7839;

**触发器使用场景三:数据库的审计--->基于值的审计功能**

给员工涨工资,当涨后的薪水超过6000块钱的时候,我们审计该员工的信息

准备工作,创建一张表用于保存审计信息

create table audit\_info\_emp(

information varchar2(200));

给涨工资的每个员工都需要触发该触发器,所以应使用行级触发器

CREATE OR REPLACE TRIGGER do\_audit\_emp\_salary

after update

on emp

for each row

begin

--当涨后的薪水大于6000,插入审计信息

if:new.sal>6000 then

insert into audit\_info\_epm values(:new.empno||' '||:new.ename||' '||:new.sal);

end if;

end;

使用如下更新语句给所以员工涨2000工资,涨薪后超过6000的员工会被写入审计表

update emp set sal=sal+2000;

commit;

**触发器使用场景四:数据库的备用和同步**

利用触发器实现数据的同步部分

准备工作:emp表:源数据表 emp\_back表:模拟emp表的备份表

--不加where条件或者加了where条件为true为复制表结构及数据

CREATE table emp\_back as select \* FROM emp;

当给员工涨完工资后,自动备份新的工资到备份表中

由于每一位涨工资的员工都需要触发该触发器,所以也是使用行级触发器

CREATE OR REPLACE TRIGGER sync\_salary

after update

on emp

for each row

begin

-- 当主表更新后,自动更新备份表

update emp\_back set sal=:new.sal where empno=:new.empno;

end;

使用如下更新语句会触发该触发器同步数据到备份表

update emp set sal=sal+10 where empno=7839;

总结:存储过程,存储函数和触发器的区别:

相同点: 存储过程和函数目的是为了 可重复地 执行操作数据库的sql语句的集合。

区别:

存储过程和存储函数:区别是写法和调用上。

写法上：

存储过程的参数列表可以有输入参数、输出参数、可输入输出的参数；

函数的参数列表只有输入参数，并且有return <返回值类型，无长度说明>。

调用方式上：

存储过程的调用方式有：

1)、exec <过程名>;

2)、execute <过程名>;

3)、在PL/SQL语句块中直接调用,也就是begin end;

4). call <过程名>;

函数的调用方式有：

在PL/SQL语句块中直接调用.也就是begin end;

BEGIN

dbms\_output.put\_line(add\_three\_numbers(2,4,5));

END;

sql语句中调用:

SELECT add\_three\_numbers(3, b=>4,c=>2) FROM DUAL;

存储过程和触发器的其他区别:

触发器与存储过程的主要区别在于调用和执行的方式:

存储过程通过用户调用实现.

触发器通过事件触发,比如当UPDATE发生时使用一个更新触发器，当INSERT发生时使用一个插入触发器，当DELETE发生时使用一个删除触发器。

使用触发器的注意事项:

1. 触发器不接受参数，一个表上最多可有12个触发器，但同一时间、同一事件、同一类型的触发器只能有一个。并各触发器之间不能有矛盾。

2.在一个表上的触发器越多，对在该表上的DML操作的性能影响就越大。

触发器最大为32KB。若确实需要，可以先建立过程，然后在触发器中用CALL语句进行调用。

3.在触发器的执行部分只能用DML语句（SELECT、INSERT、UPDATE、DELETE），不能使用DDL语句（CREATE、ALTER、DROP）。

4.触发器中不能包含事务控制语句(COMMIT，ROLLBACK，SAVEPOINT)。因为触发器是触发语句的一部分，触发语句被提交、回退时，触发器也被提交、回退了。

5.在触发器主体中调用的任何过程、函数，都不能使用事务控制语句。

6.触发器命名时可以和表,视图,存储函数,存储过程同名,因为他们都是独立的数据库对象,给其分配的都是自己的内存区域,但不能在不同的表上创建同名的触发器.会抛出已经在另一表上存在, 无法替换异常.

**包Package**

包(PACKAGE)的全称叫程序包，是由一组相关的函数，过程，变量，游标等PL/SQL程序设计元素的组合。

包类似于java中的类,把公有的变量,常量,函数,过程等都封装到该包中.

说其像java中的类,其实更像java中的接口,因为包是包含两部分:

--包接口规范:相当于java中定义了一个接口,封装公有的属性,抽象方法.

--包体:相当于实现类,将抽象方法的方法体进行实体化.

包接口规范语法:

CREATE [OR REPLACE] PACKAGE package\_name

IS|AS

[公有数据类型定义]

[公有游标声明]

[公有变量、常量声明]

[公有函数声明]

[公有过程声明]

END [package\_name];

包体语法:

CREATE [OR REPLACE] PACKAGE BODY package\_name

IS|AS

[私有数据类型定义]

[私有变量、常量声明]

[私有异常错误声明]

[私有函数声明和定义]

[私有函过程声明和定义]

[公有游标定义]

[公有函数定义]

[公有过程定义]

BEGIN

--执行部分

END package\_name;

执行调用包:

BEGIN

package\_name.函数名/过程名(有参传参);

END;

案例:

准备数据:create table emp12 as select \* from emp where 1=1;

--声明一个包接口,定义关于emp12包的返回新插入员工的编号的函数,查询指定员工信息的存储过程,返回某部门全年薪资支出存储过程

第一步:先写包规范:

create or replace package emp12\_msg

as

deptid emp12.deptno%type:=30;

function quarynewempid(eno emp12.empno%type, ena emp12.ename%type,

ejob emp12.job%type, emgr emp12.mgr%type,

ehir emp12.hiredate%type,esal emp12.sal%type,

ecomm emp12.comm%type,edno emp12.deptno%type)

return number;

procedure empmsg(eno emp12.empno%type);

procedure deptyearsal(dno emp12.deptno%type);

end;

第二步:写包体

create or replace package body emp12\_msg

as

--返回新插入员工的编号的函数

function quarynewempid(eno emp12.empno%type,ena emp12.ename%type,

ejob emp12.job%type, emgr emp12.mgr%type,

ehir emp12.hiredate%type,esal emp12.sal%type,

ecomm emp12.comm%type,edno emp12.deptno%type)

return number

as

newempno emp12.empno%type;

begin

insert into emp12 values(eno,ena,ejob,emgr,ehir,esal,ecomm,edno);

select empno into newempno from emp12 where ename=ena;

return newempno;

end quarynewempid;

--查询指定员工信息的存储过程

procedure empmsg(eno emp12.empno%type)

as

xemp emp12%rowtype;

begin

select empno,ename,sal into xemp.empno,xemp.ename,xemp.sal from emp12

where empno=eno;

dbms\_output.put\_line('编号:'||xemp.empno||',名字:'||xemp.ename||',工资:'||xemp.sal);

end empmsg;

--返回某部门全年薪资支出存储过程

procedure deptyearsal(dno emp12.deptno%type)

as

yearcomm emp12.comm%type;

yearsal emp12.sal%type;

sumsalcomm number(10);

begin

select sum(sal),sum(NVL(comm,0)) into yearsal,yearcomm from emp12

where deptno=dno;

sumsalcomm:=yearsal\*12+yearcomm\*12;

dbms\_output.put\_line(dno||'号部门,全年薪资支出为:'||sumsalcomm||'元');

end deptyearsal;

end;

第三步:调用

declare

resu number;

begin

--调用全局变量

dbms\_output.put\_line(emp12\_msg.deptid);

--返回新插入员工的编号的函数

resu:=emp12\_msg.quarynewempid(1001,'jack','临时工',7369,

to\_date('2018-4-19','yyyy-mm-dd'),5000,3000,70);

dbms\_output.put\_line(resu);

--查询指定员工信息的存储过程

emp12\_msg.empmsg(7369);

--返回某部门全年薪资支出存储过程

emp12\_msg.deptyearsal(emp12\_msg.deptid);

end;

**存储过程练习题:**

1.创建存储过程查询某个部门的平均工资和这个部门的最高工资?

2.创建存储过程查询某个部门的人数?

3.创建存储过程查询某个指定编号的员工,并输出其名字,员工编号,工资,部门编号,部门名称?

4.创建存储过程查询工资在某个范围(比如1500~3000)的员工名字和薪资? (提示:由于返回多条数据,需要用到游标)

5.创建存储过程修改某个员工的工资:(提示:需要用到分支和异常)

5.1)先判断该员工是否存在?

存在再判断该员工是否是MANAGER职位,如果是则加薪1000,不是该职位加薪500.

并打印涨薪前的工资和涨新后的工资

5.2)若不存在,则抛出异常给出提示"没有该编号员工"

6.创建存储过程查看员工表部门人数大于3人的部门的平均工资?(提示:由于返回多条数据,需要用到游标)

7.创建存储过程分页查询工资降序排名在指定范围的员工名字和工资?(提示:oracle分页用rownum,返回多条数据,需要用到游标)

8.创建存储过程添加员工:(提示:需要分支和异常,员工入职日期需要使用to\_data()转换下)

8.1)先判断要添加员工是否存在,若存在则不添加,抛出异常给用户提示"该用户已存在".

8.2)不存在,则执行添加操作,添加完毕后给用户输出提示"添加成功"

9.创建存储过程查询某个员工的领导的编号及名字?

10.创建存储过程查询全年工资支出总和?(提示:该存储过程不需要参数)