PL/SQL

I. 变量

一个变量是程序中可以操纵的存储区域的名称

PL/SQL中的每个变量都有一个指定的数据类型，决定变量内存的大小和布局

1. 变量声明

在声明部分声明`PL/SQL`变量当作全局变量。当声明一个变量时，`PL/SQL`为变量的值分配内存，并且存储位置由变量名称标识。

```
variable_name [constant] datatype [NOT NULL] [:= | DEFAULT
initial_value];
```

`datatype`必须是有效的PL/SQL数据类型或者任何用户定义的数据类型

```
x_salary employee.salary %type;
pi constant double precision := 3.1415;
name varchar2(25);
```

如果使用`not null`必须为变量显式分配初始值

2. 变量作用域

PL/SQL允许块的嵌套，每个程序块可以包含另一个内部块。

如果在一个内部块中声明了一个变量，外部块不可以访问内部变量。

- 局部变量，在内部块中声明的变量，外部块不可访问；
- 全局变量，在最外部块中生命的变量

3. 将SQL查询结果分配给PL/SQL变量

`SELECT INTO`语句：对于`SELECT`列表中的每个项目，`INTO`列表中必须有一个对应的类型兼容变量。

```
DECLARE
    E_SSN EMPLOYEE.SSN %TYPE := 888665555;
    E_FNAME EMPLOYEE.FNAME %TYPE;
    E_ADDRESS EMPLOYEE.ADDRESS %TYPE;
    E_SAL EMPLOYEE.SALARY %TYPE;
BEGIN
    SELECT fname,address,salary
    into e_fname,e_address,e_sal
    from employee
    where ssn=e_ssn;
    dbms_output.put_line('employee' || e_fname || 'from' ||
e_address ||'earns' || e_sal);
END;
```

II. 存储过程

1.创建存储过程

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN|OUT|IN OUT] TYPE [,...] )]
{IS | AS}
BEIGN
    ...
END procedure_name;
```

- procedure_name: 要创建的存储过程的名字
- [OR REPLACE]： 选项允许修改现有的进程
- 可选参数列表包含参数的名称，模式和类型。 IN表示将从外部传递的值，OUT表示将用于返回过程外的值的参数

```
PROCEDURE GREETINGS
AS
begin
    dbms_output.put_line('hello world');
end;
```

2. 执行独立程序

- 使用`EXECUTE`关键字
- 在`PL/SQL`块中调用过程的名称

```
EXECUTE greetings;
```

```
BEGIN
    greetings;
END;
/
```

3.删除独立存储过程

```
DROP PROCEDURE procedure_name;
```

4. 子程序中的参数模式

1. IN
2. OUT
3. IN OUT

`IN`	`IN` 参数允许将值传递给子程序。它是一个只读参数。在子程序中， `IN` 参数的作用如常数，它不能被赋值。可以将常量，文字，初始化的变量或表达式作为 `IN` 参数传递。也可以将其初始化为默认值; 然而，在这种情况下，从子程序调用中省略它。 它是参数传递的默认模式。参数通过引用传递。
`OUT`	`OUT` 参数返回一个值给调用程序。在子程序中， `OUT` 参数像变量一样。 可以更改其值并在分配该值后引用该值。实际参数必须是可变的，并且通过值传递。
`IN OUT`	`IN OUT` 参数将初始值传递给子程序，并将更新的值返回给调用者。 它可以分配一个值，该值可以被读取。对应于 `IN OUT` 形式参数的实际参数必须是变量，而不是常量或表达式。正式参数必须分配一个值。实际参数(实参)通过值传递。

```
create or replace procedure greeting is
h number;
g char(20);

begin
    select extract(hour from current_timestamp)
    into h from dual;

    if h>=20 or h<=5 then
    g:='goodnight';
    elsif h>5 and h<=12 then
    g:='goodmorning';
    elsif h>12 and h<=17 then
    g:='goodafternoon';
    else
    g:='good evening';
    end if;

    dbms_output.put_line(g);
end;
```

III. 游标

Oracle创建一个称为上下文区域的内存区域，用于处理SQL语句，包含处理该语句所需的所有信息；例如，处理的行数等。

游标是指向此上下文区域的指针，`PLSQL`通过游标控制上下文区域，游标保存sql语句返回的行（一个或多个），游标所在的行集称为活动集

常用属性

`%FOUND`	如果 `INSERT` ， `UPDATE` 或 `DELETE` 语句影响一行或多行，或者兄 `SELECT INTO` 语句返回一行或多行，则返回 `TRUE` ，否则返回 `FALSE` 。
`%NOTFOUND`	与 `%FOUND` 的逻辑相反。 如果INSERT，UPDATE或DELETE语句没有影响任何行，或SELECT INTO语句未返回任何行，则返回TRUE。 否则返回FALSE。
`%ISOPEN`	由于Oracle在执行关联的SQL语句后会自动关闭SQL游标，因此总是为隐式游标返回 `FALSE` 。
`%ROWCOUNT`	返回受 `INSERT` ， `UPDATE` 或 `DELETE` 语句，或者受 `SELECT INTO` 语句影响的行数。

任何SQL游标属性将被访问为`sql%attribute_name`

隐式游标

```
SET SERVEROUTPUT ON SIZE 99999;
DECLARE
   total_rows number(2);
BEGIN
   UPDATE customers
   SET salary = salary + 500;
   IF sql%notfound THEN
      dbms_output.put_line('没有找到客户信息~');
   ELSIF sql%found THEN
      total_rows := sql%rowcount;
      dbms_output.put_line('一共有: ' || total_rows || ' 个客户的工资被更新! ');
   END IF;
END;
/
```

III. Record

记录可以容纳不同种类的数据项的数据结构。记录由不同的字段组成，类似于数据库的一行。

- 基于表的记录
- 基于游标的记录
- 用户定义的记录

1. 基于表的记录

`rowtype`属性能够创建基于表和基于游标的记录

```
SET SERVEROUTPUT ON SIZE 99999;
DECLARE
   customer_rec customers%rowtype;
BEGIN
   SELECT * into customer_rec
   FROM customers
   WHERE id = 5;
   dbms_output.put_line('客户ID: ' || customer_rec.id);
   dbms_output.put_line('客户姓名: ' || customer_rec.name);
   dbms_output.put_line('客户地址: ' || customer_rec.address);
   dbms_output.put_line('客户薪资: ' || customer_rec.salary);
END;
/
```

2.用户定义的记录

```
TYPE
type_name IS RECORD
( field_name1  datatype1  [NOT NULL]  [:= DEFAULT EXPRESSION],
   field_name2   datatype2   [NOT NULL]  [:= DEFAULT EXPRESSION],
   ...
   field_nameN  datatypeN  [NOT NULL]  [:= DEFAULT EXPRESSION);
record-name  type_name;
```

```
```

```
declare
    TYPE EmpRecord
    IS RECORD(SSN employee.ssn%type,
              lname employee.lname%type,
              dname department.dname%type,
              bounspayment number(6));

    InactiveEmp EmpRecord;

begin
    select essn,lname,dname,0
    into InactiveEmp
    from employee e
    inner join department d on e.dno=d.dnumber
    inner join works_on w on e.ssn=w.essn
    where hours = (select MIN(hours) from works_on)
    and rownum<=1;

    update employee
    set superssn=null
    where superssn=inactiveemp.ssn;

    update department
    set mgrssn=null
    where mgrssn=inactiveemp.ssn;

    delete from dependent
    where essn=inactiveemp.ssn;

    delete from works_on
    where essn=inactiveemp.ssn;

    delete from employee
    where ssn=inactiveemp.ssn;

    dbms_output.put_line('Least active employee has been
transferred: ' ||
                         InactiveEmp.LName);
end;
```