

PL/SQL存储过程基础知识

▼ 一、PL/SQL简介

- (一) ORACLE PL/SQL语言：结合了结构化查询与ORACLE自身过程控制为一体的强大语言（SQL查询语言+过程控制语言结合）

▼ (二)PL/SQL语言体系结构

- 1、执行逻辑：是一种块结构的语言，将一组语言放在一个块中，一次性发送给服务器，PL/SQL引擎分析收到PL/SQL语句中的内容，把其中的过程控制语句在PL/SQL引擎自身的执行器去执行，把PL/SQL块中的SQL语句交给Oracle服务器的SQL语句执行器执行
- 2、PL/SQL的优点：PL/SQL块一次性向数据库发送多条SQL，减少网络传输交互的次数，减少网络开销，减小对服务器性能的影响，提升应用程序性能

▼ (三) PL/SQL的优点

- 1、支持SQL：PL/SQL支持所有的SQL数据操作命令、游标控制命令、事务控制命令、SQL函数、运算符和伪列，支持所有SQL数据类型和NULL值；
- 2、支持面向对象编程：在PL/SQL中可以创建类型，可以对类型进行继承，可以在子程序中重载方法等；
- 3、更好性能：SQL是非过程语言，只能一条一条执行，而PL/SQL把一个PL/SQL块统一进行编译后执行，同时还可以把编译好的PL/SQL块储存起来，以备重用，减少了应用程序和服务器之间的通信时间，PL/SQL快速而高效；
- 4、可移植性：使用PL/SQL编写的应用程序，可以移植到任何操作系统平台上的ORACLE服务器，同时还可以编写可移植程序库，在不同环境中重用；
- 5、安全性：可以通过存储过程对客户机和服务器之间的应用程序逻辑进行分隔，这样可以限制对Oracle数据库的访问，数据库还可以授权和撤销其他用户访问的能力；

- (四) PL/SQL的目标：实现SQL的自动化

▼ 二、PL/SQL存储过程 语法基础（语法组成和结构、循环判断、条件控制等）

- (一) PL/SQL是一种编程语言，与JAVA和C#一样，除了有自身独有的数据类型、变量声明和赋值以及流程控制语句外，还有自身的语言特性
- (二) 语言特性：1、对大小写不敏感；2、每一条语句都必须以分号结束

▪ (三) 特殊符号说明

PL/SQL中的特殊符号说明：

类型	符号	说明
赋值运算符	<code>:=</code>	Java和C#中都是等号，PL/SQL的赋值是： <code>=</code>
特殊字符	<code> </code>	字符串连接操作符。
	<code>--</code>	PL/SQL中的单行注释。
	<code>/*,*/</code>	PL/SQL中的多行注释，多行注释不能嵌套。
	<code>..</code>	范围操作符，比如： <code>1..5</code> 标识从 1 到 5
算术运算符	<code>+, -, *, /</code>	基本算术运算符。
	<code>**</code>	求幂操作，比如： <code>3**2=9</code>
关系运算符	<code>>, <, >=, <=, =</code>	基本关系运算符， <code>=</code> 表示相等关系，不是赋值。
	<code><>, !=</code>	不等关系。
逻辑运算符	<code>AND, OR, NOT</code>	逻辑运算符。

▼ (四) PL/SQL块语法

- 1、PL/SQL 是一种块结构语言，一个PL/SQL程序包含了一个或多个逻辑块，逻辑块中可以声明变量，变量在使用之前必须声明，除了正常执行程序外，PL/SQL还提供了专门的异常处理部分，每个逻辑块分成3个部分

▼ 2、PL/SQL块的语法

- 1) 包含3个部分：声明、执行和异常处理

- 2) 语法结构：PL/SQL块的语法

```
[DECLARE  
    --declaration statements] ①  
BEGIN  
    --executable statements ②  
[EXCEPTION  
    --exception statements] ③  
END;
```

▼ 3) 语法解析

- ①声明部分：声明部分包含了变量和常量的定义。这个部分由关键字DECLARE开始，如果不声明变量或者常量，可以省略这部分。
- ②执行部分：执行部分是 PL/SQL块的指令部分，由关键字BEGIN开始，关键字END结尾。所有的可执行PL/SQL语句都放在这一部分，该部分执行命令并操作变量。其他的PL/SQL块可以作为子块嵌套在该部分。PL/SQL块的执行部分是可选的。注意END关键字后面用分号结尾。
- ③异常处理部分：该部分是可选的，该部分用EXCEPTION关键字把可执行部分分成两个小部分，之前的程序是正常运行的程序，一旦出现异常就跳转到异常部分执行。

▼ 4) 各个部分语法结构

▼ (1)声明部分

- A. 语法结构：DECLARE 变量名 数据类型[:=初始值]

- ▼ B. 注：

- 声明变量必须指明变量的数据类型，也可以声明变量时对变量初始化，变量声明必须在声明部分
- 字符型变量一定要定义长度，数字型变量可以不用定义长度

▼ C. 声明属性数据类型

- %ROWTYPE：引用数据库中的一行（所有字段）作为数据类型

代码演示：

--【例】找出员工编号为7934的员工名称和工资

```
DECLARE
  V_EMP EMP%ROWTYPE; ---定义变量为EMP表的所有字段的数据类型
BEGIN
  SELECT *      ---将EMP表的数据插入到V_EMP表中
  INTO V_EMP
  FROM EMP
  WHERE EMPNO = 7934;
  DBMS_OUTPUT.PUT_LINE(V_EMP.EMPNO ||
  CHR(13)||V_EMP.ENAME); ---CHR(13)为换行符号
END;
```

- %TYPE：引用数据库中的某列的数据类型或某个变量的数据类型

代码演示：

```
DECLARE
  V_EMPNO EMP.EMPNO%TYPE; ---定义变量
  V_ENAME EMP.ENAME%TYPE;
  V_HIREDATE EMP.HIREDATE%TYPE;
BEGIN
  SELECT EMPNO,
         ENAME,      ---查询数据库并为变量赋值
         HIREDATE
  INTO V_EMPNO,V_ENAME,V_HIREDATE
  FROM EMP
  WHERE EMPNO = &EMPNO; ----输入参数来空值运算逻辑
  DBMS_OUTPUT.PUT_LINE(V_EMPNO||CHR(13)||V_ENAME||CHR(13)||v_
  HIREDATE);
END;
```

▼ D. 常量

- 常量在声明时赋予初值，并且在运行时不允许重新赋值。
- 声明常量时使用关键字CONSTANT，常量初值可以使用赋值运算符(:=)赋值，也可以使用DEFAULT关键字赋值

- E. &赋值符号：可以输入内容控制运行逻辑

▪ 代码演示

```
DECLARE
  SNAME VARCHAR2(20) :='JERRY'; ①
BEGIN
  SNAME:=SNAME||'AND TOM'; ②      --直接赋值
  DBMS_OUTPUT.PUT_LINE(SNAME); ③
END;
```

▼ 其他语法注意项

- dbms_output.put_line是输出语句
- 对变量赋值还可以使用SELECT...INTO 语句从数据库中查询数据对变量进行赋值。但是查询的结果只能是一行记录，不能是零行或者多行记录

▪ (2) 执行部分

▼ (3) 异常处理

- 异常处理的定义：程序在运行时出现的错误，称异常，异常发生后，语句将停止执行，PL/SQL引擎立即将掌控权转到PL/SQL块的异常处理部分，异常处理机制简化了代码中的错误检测。PL/SQL中任何异常出现时，每一个异常都对应一个异常码和异常信息。
- 异常处理逻辑：当语句执行部分出现异常----PL/SQL引擎跳到异常处理部分----异常处理部分处理完异常后退出

▼ 3 预定义异常

- 为了Oracle开发和维护的方便，在Oracle异常中，为常见的异常码定义了对应的异常名称，称为预定义异常

▼ 常见的预定义异常有

异常名称	异常码	描述
DUP_VAL_ON_INDEX	ORA-00001	试图向唯一索引列插入重复值
INVALID_CURSOR	ORA-01001	试图进行非法游标操作。
INVALID_NUMBER	ORA-01722	试图将字符串转换为数字
NO_DATA_FOUND	ORA-01403	SELECT INTO语句中没有返回任何记录。
TOO_MANY_ROWS	ORA-01422	SELECT INTO语句中返回多于1条记录。
ZERO_DIVIDE	ORA-01476	试图用0作为除数。
CURSOR_ALREADY_OPEN	ORA-06511	试图打开一个已经打开的游标

▼ 4 异常处理部分语法结构

- 语法格式：异常处理

```
BEGIN
    --可执行部分
    EXCEPTION -- 异常处理开始
        WHEN 异常名1 THEN
            --对应异常处理
        WHEN 异常名2 THEN
            --对应异常处理
        .....
        WHEN OTHERS THEN
            --其他异常处理
END;
```

▼ 语法解析

- 常发生时，进入异常处理部分，具体的异常与若干个WHEN子句中指定的异常名匹配，匹配成功就进入对应的异常处理部分，如果对应不成功，则进入OTHERS进行处理。

▼ 代码演示

- 预定义异常处理

代码演示：

----EXCEPTION编译时的错误处理

DECLARE

X NUMBER :=&X; ---使用参数化的数值，可以输入任意值

Y NUMBER :=&Y;

Z NUMBER :=&Z;

BEGIN

Z := X + Y ; --- 两个数相加

DBMS_OUTPUT.PUT_LINE('X + Y = '||Z);

Z := X/Y ;

DBMS_OUTPUT.PUT_LINE('X/Y = '||Z);

EXCEPTION ---异常处理部分

WHEN ZERO_DIVIDE THEN ---预定义异常，处理被0除的异常

DBMS_OUTPUT.PUT_LINE('除数不能为0!!! ');

END;

- 自定义异常：使用Raise作为关键字，触发异常

代码演示：

案例：向表中插入新员工，新员工的姓名不能重复

--RAISE 触发异常写法

DECLARE

E_DUPLICATE_NAME EXCEPTION; ---定义异常的名称

V_ENAME EMP.ENAME%TYPE; ---保存姓名的变量

V_NEWNAME EMP.ENAME%TYPE := 'SMITH'; ---新插入的员工姓名

BEGIN

SELECT ENAME INTO V_ENAME FROM EMP WHERE EMPNO = 7369; ---查询员工编号为7369的姓名

IF V_ENAME = V_NEWNAME

THEN

RAISE E_DUPLICATE_NAME; ---如果名字相同，运行会出现异常，就会触发自定义的异常E_DUPLICATE_NAME

END IF;

INSERT INTO EMP ---如果名字不相同，就会执行插入语句

VALUES(7881,V_NEWNAME,'CLERK',NULL,TRUNC(SYSDATE),2000,200,20);

EXCEPTION

WHEN E_DUPLICATE_NAME THEN

DBMS_OUTPUT.PUT_LINE('不能插入重复的员工姓名!!!');

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE('异常编码: '||SQLCODE||' 异常信息: '||SQLERRM);

END;

▼ (五) 存储过程中的条件判断，PL/SQL关于条件控制的关键字主要种类

▼ 1、IF - THEN

▼ 语法结构：

- IF 条件 THEN --条件成立结构体
ELSE --条件不成立结构体
END IF;

▼ 语法解析

- 该结构先判断一个条件是否为TRUE，条件成立则执行对应的语句块。
- ①用IF关键字开始，END IF关键字结束，注意END IF后面有一个分号。
- ②条件部分可以不使用括号，但是必须以关键字THEN来标识条件结束，如果条件成立，则执行THEN后到对应END IF之间的语句块内容。如果条件不成立，则不执行条件语句块的内容。
- ③条件可以使用关系运算符符合逻辑运算符。
- ④在PL/SQL块中可以使用事务控制语句，该COMMIT同时也能把PL/SQL块外没有提交的数据一并提交，使用时需要注意。

▼ 2、IF - THEN -ELSE :

▼ 语法结构：

- IF 条件 THEN --条件成立结构体 ELSE --条件不成立结构体 END IF;
- 把ELSE与IF-THEN连在一起使用，如果IF条件不成立则执行就会执行ELSE部分的语句。

▼ 3、IF - THEN - ELSIF :

▼ 语法结构：

- IF --条件1 THEN ---条件1成立结构体 ELSIF 条件2 THEN--条件2成立结构体
ELSE --以上条件都不成立结构体 END IF;
- PL/SQL中的再次条件判断中使用关键字ELSIF。

▼ 4、CASE :

▼ 语法结构：

- CASE [selector] WHEN 表达式1 THEN 语句序列 1 ; WHEN 表达式 2 THEN 语句序列 2; WHEN 表达式 3 THEN 语句序列 3;ELSE 语句序列 N; END CASE;

代码演示:

```
DECLARE
  V_SAL NUMBER;
BEGIN
  SELECT SAL INTO V_SAL FROM EMP WHERE ENAME = 'JAMES';
  CASE
    WHEN V_SAL > 1500 THEN
      UPDATE EMP SET COMM = 100 WHERE ENAME = 'JAMES';
    WHEN V_SAL >= 900 AND V_SAL <= 1500 THEN
      UPDATE EMP SET COMM = 800 WHERE ENAME = 'JAMES';
    ELSE
      UPDATE EMP SET COMM = 400 WHERE ENAME = 'JAMES';
  END CASE;
END;
```

▼ 语法分析

- CASE是一种选择结构的控制语句，可以根据条件从多个执行分支中选择相应的执行动作。
- 如果存在选择器selector，选择器selector与WHEN后面的表达式匹配，匹配成功就执行THEN后面的语句。如果所有表达式都与selector不匹配，则执行ELSE后面的语句。
- 如果不使用CASE中的选择器，直接在WHEN后面判断条件，第一个条件为真时，执行对应THEN后面的语句序列。

▼ 六、循环控制

- (一) PL/SQL 提供了丰富的循环结构重复执行一些语句，循环中EXIT用来强制结束循环。

▼ (二) 循环控制的类型

▼ 1、LOOP无限 循环

- (1) LOOP循环是最简单的循环，也称为无限循环，LOOP和END LOOP是关键字。循环退出机制在循环体内

▼ (2) 语法结构:

- LOOP
 - 循环体
 END LOOP;

▼ (3) 语法解析

- 1) 循环体在LOOP和END LOOP之间，在每个LOOP循环体中，首先执行循环体中的语句序列，执行完后再重新开始执行。
- 2) 在LOOP循环中可以使用EXIT或者[EXIT WHEN 条件]的形式终止循环。否则该循环就是死循环。

▼ (4) 使用EXIT 退出循环

- LOOP循环中可以使用IF条件判断结构嵌套EXIT关键字退出循环
- 使用EXIT WHEN 来替换上述IF 循环体，满足条件时退出循环
- (5) 代码演示

代码演示：LOOP循环

```

【例】执行1+2+3+...+100的值
DECLARE
A NUMBER := 0;
B NUMBER := 0;
BEGIN
LOOP
A := A + 1; --1 2 3 4 .....99 100
B := B + A; --0+1 1+2 1+2+3 1+2+3+4 .....1+2+...+98+99 1+...+99+100
IF A >= 100 THEN
EXIT; ----loop 循环中可以使用IF结构嵌套EXIT关键字退出循环
END IF;
---- EXIT WHEN A >= 100; ---可以替换IF条件判断。当满足条件时退出循环
END LOOP;
DBMS_OUTPUT.PUT_LINE(B);
END;

```

- (6) 退出机制：退出机制在循环体内

▼ 2、WHILE循环

- (1)WHILE循环，先判断条件，条件成立再执行循环体，循环退出机制在循环体外

▼ (2) 语法结构：

- WHILE 条件 LOOP --循环体 END LOOP，在WHILE后 LOOP前设置结束条件

▪ (3) 代码演示

代码演示：WHILE循环

```

【例】执行1+2+3+...+100的值
DECLARE
A NUMBER := 0;
B NUMBER := 0;
BEGIN
WHILE A < 100 LOOP
A := A + 1;
B := B + A;
END LOOP;
DBMS_OUTPUT.PUT_LINE(B);
END;

```

▼ 3、FOR 循环

- (1)FOR循环需要预先确定的循环次数，可通过给循环变量指定下限和上限来确定循环运行的次数，然后循环变量在每次循环中递增（或者递减）。

- (2)FOR X IN ...LOOP -END LOOP，变量要遍历所有遍历，每遍历一次，执行一次循环体，FOR循环不用声明变量

▼ (3)语法结构：

- FOR 循环变量 IN [REVERSE] 循环下限..循环上限 LOOP
--循环体
END LOOP;

▼ (4)语法解析

- 循环变量：该变量的值每次循环根据上下限的REVERSE关键字进行加1或者减1。
- REVERSE：指明循环从上限向下限依次循环。
- FOR循环不用声明变量

▪ (5) 代码演示

代码演示：FOR循环

```
【例】执行1+2+3+...+100的值
DECLARE
  B NUMBER := 0;
BEGIN
  FOR A IN 0..100 LOOP
    B := B + A ;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE(B);
END;
```

▼ 三、PL/SQL存储过程基础（游标、静态SQL）

▼ （一）游标

- Oracle在执行SQL语句时，总是需要创建一块内存区域，这块内存区域称为上下文区域。在上下文区域中包含了处理语句的信息，这些信息包含当前语句已经处理了多少行、指向被分析语句的指针和查询语句返回的数据行集。

游标是一个指向上下文区域的指针，游标的作用就是用于提取临时存储在内存的数据块，该数据块是从数据库中提取的。因为在某些情况下，需要把数据从存放在磁盘的表中调到计算机内存中进行处理，最后将处理结果显示出来或最终写回数据库。这样数据处理的速度才会提高，否则频繁的磁盘数据交换会降低效率。

当使用SELECT语句查询返回多行的数据时，可以通过游标来指向结果集中的每一行，使用循环语句依次对每一行进行处理。

- 游标在PL/SQL块中的运行逻辑：PL/SQL块将多条SQL发送给PL/SQL执行器，其中的SQL语句发送给SQL执行器，处理和读写数据后传到硬盘上，硬盘上的数据再加载到内存中，当使用硬盘上的ROWID直接返回数据给PL/SQL执行器时，会读写很慢。但在内存里读写就很快，所以加载到内存后，使用游标在内存的查询结果集中将结果指向PL/SQL执行器

- 1、定义：游标是指向查询结果集的一个指针，通过游标可以将查询结果集中的记录逐一取出，并在PL/SQL程序块中进行处理
- ▼ 2、游标的种类
 - 根据对应查询结果集的行数分类，查询结果集为多行的为显性游标，查询结果集为单行的为隐性游标
- ▼ (1) 显示游标
 - 1) 定义：：在PL/SQL块的声明区域中显式定义的，用来处理返回多行记录查询的游标。
- ▼ 2) 特点：
 - 需要声明、打开、抓取、关闭
 - 指向多行查询结果集
- ▼ 3) 显性游标的逻辑：采用OPEN、FETCH和CLOSE语句来控制游标
 - OPEN用于打开游标并使游标指向结果集的第一行
 - FETCH会检索当前行的信息并把游标指向下一行
 - 当最后一行被处理完后，CLOSE就会关闭游标。
- ▼ 4) 语法结构
 - ▼ 声明游标
 - CURSOR 游标名[(参数1 数据类型[, 参数2 数据类型...])]
IS SELECT语句; --游标的声明
 - 游标命名规范：C_游标名
 - ▼ 执行游标
 - OPEN 游标名[(实际参数1[, 实际参数2...]); --打开游标
FETCH 游标名 INTO 变量名1[, 变量名2...];
或
FETCH 游标名 INTO 记录变量; --提取数据
CLOSE 游标名; --关闭游标（千万别忘了！）
 - 最后一定要CLOSE 关闭游标

- ● FETCH一次只能抓取一行数据，所以一般结合循环使用

---FETCH只能一次抓取一行

--代码演示:

```
DECLARE
V_DNAME VARCHAR2(20);
V_DEPTNO NUMBER;
V_LOC VARCHAR2(20);
CURSOR C_DEPT
IS
SELECT DNAME,DEPTNO,LOC FROM DEPT;
BEGIN
OPEN C_DEPT;
FETCH C_DEPT INTO V_DNAME,V_DEPTNO,V_LOC;
DBMS_OUTPUT.PUT_LINE(V_DNAME || ' '||V_DEPTNO|| ' '||V_LOC);
FETCH C_DEPT INTO V_DNAME,V_DEPTNO,V_LOC;
DBMS_OUTPUT.PUT_LINE(V_DNAME || ' '||V_DEPTNO|| ' '||V_LOC);
FETCH C_DEPT INTO V_DNAME,V_DEPTNO,V_LOC;
DBMS_OUTPUT.PUT_LINE(V_DNAME || ' '||V_DEPTNO|| ' '||V_LOC);
FETCH C_DEPT INTO V_DNAME,V_DEPTNO,V_LOC;
DBMS_OUTPUT.PUT_LINE(V_DNAME || ' '||V_DEPTNO|| ' '||V_LOC);
CLOSE C_DEPT;
END;
```

- 代码演示 (结合loop循环)

---使用游标属性和Loop 循环抓取数据 --显性游标

```
DECLARE
V_DNAME VARCHAR2(10);
V_DEPTNO NUMBER;
V_LOC VARCHAR2(10);
CURSOR C_DEPT1
IS
SELECT DNAME, DEPTNO,LOC FROM DEPT;
BEGIN
OPEN C_DEPT1;
LOOP
FETCH C_DEPT1 INTO V_DNAME,V_DEPTNO,V_LOC;
EXIT WHEN C_DEPT1%NOTFOUND; ---%ROWCOUNT = 3; 当抓取到第三行
的时候退出
DBMS_OUTPUT.PUT_LINE(V_DNAME||' '||V_DEPTNO|| ' '||V_LOC);
END LOOP;
CLOSE C_DEPT1;
END;
```

▼ 5) 游标属性

- %FOUND:用于判断游标是否从结果集中提取数据。如果提取到数据，则返回值为TRUE，否则返回值为FALSE。
- %NOTFOUND: 该属性与%FOUND相反，如果提取到数据则返回值为FALSE；如果没有，则返回值为TRUE。
- %ROWCOUNT: 判断提取到第几条数据

▼ (2) 隐性游标

- 1) 定义：由PL/SQL自动为DML语句或SELECT-INTO语句分配的游标，包括只返回一条记录的查询操作
- 当在PL/SQL中执行SELECT和DML语句时，如果只查询单行数据，比如使用SELECT INTO语句或执行DML语句时，Oracle会为它们分配隐含的游标。

事实上，执行每一个DML操纵语句时，Oracle都会在PGA中的一个上下文区域中具有一个隐式的游标。举个例子，当对emp表中的提成栏进行更新的时候，在执行UPDATE语句时会自动具有一个隐式游标，可以通过隐含的游标变量来访问隐式游标的状态。

▼ 2) 特点：

- 系统自动声明，不需要声明，不需要打开和关闭
- 指向单行查询结果集

▼ 3) DML操作和单行SELECT语句会使用隐式游标

- 插入操作：INSERT
- 更新操作：UPDATE
- 删除操作：DELETE
- 单行查询操作：SELECT ... INTO ...

▪ 代码演示

--查询JAMES的工资，如果大于1500元，则发放奖金100元，如果工作大于900元小于等于1500，则发奖金800元，否则发奖金400元。

代码演示

```
DECLARE
NEWSAL EMP.SAL%TYPE;
BEGIN
SELECT SAL INTO NEWSAL FROM EMP WHERE ENAME = 'JAMES';
IF NEWSAL > 1500 THEN
UPDATE EMP SET COMM = 100 WHERE ENAME = 'JAMES';
ELSIF NEWSAL BETWEEN 900 AND 1500 THEN
UPDATE EMP SET COMM = 800 WHERE ENAME = 'JAMES';
ELSE
UPDATE EMP SET COMM = 400 WHERE ENAME = 'JAMES';
END IF;
COMMIT;
DBMS_OUTPUT.PUT_LINE(NEWSAL);
END;
```

---隐性游标，单行查询结果集，DDL语句一般是隐性游标

```
DECLARE
V_NUM NUMBER;
BEGIN
----不需要声明游标
SELECT COUNT(1) INTO V_NUM FROM EMP;
DBMS_OUTPUT.PUT_LINE(V_NUM);
END;
```

▼ 3、游标变量

- 游标FOR循环不需要事先定义，它会隐式声明一个代表当前的循环索引变量。系统自动打开游标，当所有行都被处理后，就会自动关闭游标，不需要人为操作。

- 代码演示

```
----方法二，for循环，不需要声明游标变量，不需要open fetch 和close
DECLARE
CURSOR C_DEPT2
IS
SELECT DNAME,DEPTNO,LOC FROM DEPT;
BEGIN
FOR X IN C_DEPT2 LOOP
DBMS_OUTPUT.PUT_LINE(X.DNAME||' '||X.DEPTNO||' '||X.LOC);
END LOOP;
END;
```

▼ 4、用游标批量提取数据

- 如果操作涉及大量的数据，则可以通过把大量的数据进行一次性处理来提升性能，比如可以将数据放到索引表、嵌套表和变长数组中，通过FORALL或BULK COLLECT INTO等批处理语句，一次性处理大的数据量，提升性能。

- 代码演示：

▼ (二) 静、动态SQL

▼ 1、静态SQL

- (1)PL/SQL块执行的任何SQL语句，其查询的对象必须存在。比如要查询员工表中的员工名称，必须存在员工表、必须知道员工 名的字段名称。这种 SQL称为静态SQL
- (2)注意：静态SQL仅能为DML语句，所以当需要在PL/SQL块执行DDL操作时，就需要动态SQL

▼ 2、动态SQL


- (1) 定义：动态SQL是指在运行时才编译执行的SQL语句,且它在PL/SQL块中以字符串的形式存在。比如在PL/SQL块中不能执行DDL 语句和DCL语句，那么可以使用EXECUTE IMMEDIATE来执行动态拼合而成的SQL语 句。
- (2) 注意：更常见的情形是在执行时才知道要查询哪个表中的数据，或者执行时才知道要查询哪 些列，则可以使用动态SQL语句。
- (3) 动态SQL执行逻辑：PL/SQL块是先编译然后再执行的，而动态SQL语句在编译时不能确定，在编译阶段它以字符串形式存在，程序不会对字符串中的内容进行编译。只有在运行阶段时，才会对字符串中的SQL语句进行编译和执行

▼ (4) 动态SQL语法

- 语法格式：
EXECUTE IMMEDIATE 动态语句字符串
[INTO 变量列表]
[USING 参数列表]
- 语法解析：
如果动态语句是SELECT语句，可以把查询的结果保存到INTO后面的变量中。如果动态语句中存在参数，USING为语句中的参数传值。
动态SQL中的参数格式是：[:参数名]，参数在运行时需要使用USING传值。
- 可以随意拼接

- (5) 通过表名、列名控制SQL执行逻辑就只能通过动态SQL，执行DDL语句

▼ 四、PL/SQL存储过程（匿名块和命名块）

-  PL/SQL分类匿名块和命名块

▼ 1、匿名块

- 1、以BEGIN 或 DECLARE开始，在每次执行时必须重新编译，不能存在数据库中，不能被其他语句块调用
- 2、语法结构： DECLARE 声明 BEGIN 执行逻辑 EXCEPTION 异常处理块 END;

语法结构：PL/SQL块的语法
[DECLARE
--declaration statements] ①
BEGIN
--executable statements ②
[EXCEPTION
--exception statements] ③
END;

语法解析：

- ①声明部分：声明部分包含了变量和常量的定义。这个部分由关键字DECLARE开始，如果不声明变量或者常量，可以省略这部分。
- ②执行部分：执行部分是 PL/SQL块的指令部分，由关键字BEGIN开始，关键字END结尾。所有的可执行PL/SQL语句都放在这一部分，该部分执行命令并操作变量。其他的PL/SQL块可以作为子块嵌套在该部分。PL/SQL块的执行部分是必选的。注意END关键字后面用分号结尾。
- ③异常处理部分：该部分是可选的，该部分用EXCEPTION关键字把可执行部分分成两个小部分，之前的程序是正常运行的程序，一旦出现异常就跳转到异常部分执行。

▼ 2、命名块

- (1) 定义：包含了PL/SQL子程序（过程和函数）、包及触发器。可以存储在数据字典中，可以被其他块调用。不需要在每次执行时重新编译
- ▼ (2) 命名块的种类（子程序（包含过程和函数）、包、触发器）
 - ▼ 1) 子程序（过程和函数）

- ① 子程序定义：包含了过程和函数两大类，与匿名块一样，子程序在组成和结构上有声明部分、执行部分及可选的异常处理部分，但子程序能够接收参数，并被其他的程序调用
- ▼ ②子程序的种类
 - ▼ a. 存储过程(procedure)
 - 定义：过程是一个命名的程序块，包括过程的名称、过程使用的参数以及过程执行的操作。如果在应用程序中经常需要执行某些特定的操作，1 那么就可以基于这些操作创建一个特定的过程，过程经编译后存储在数据库中
 - ▼ 2 语法格式（创建、调用、删除）
 - ▼ ● 创建存储过程
 - CREATE [OR REPLACE] PROCEDURE 过程名(参数1 [IN|OUT|INOUT] 数据类型, 参数2 [IN|OUT|INOUT] 数据类型.....) IS | AS
PL/SQL过程体;
 - ▼ 创建过程的语法解析
 - CREATE OR REPLACE：表示创建过程，如果过程已存在则是替换已有的过程
 - IN 表示传入参数，不可以赋值，OUT表示传出参数，可以被赋值，且不能对过程的参数的类型添加长度约束
 - IS/AS：在IS / AS后声明变量不需要添加DECLARE语句
 - ▼ 命名的规范
 - 存储过程命名规范：SP_目标表名
 - 存储过程传入参数命名规范：P_参数名（P_START_DATE）
 - 存储过程变量命名规范：V_变量名（V_END_DATE）

▪ 代码演示

---创建存储过程

--- 1、不带参数的过程

代码演示

```
CREATE OR REPLACE PROCEDURE SP_WINDS
IS--AS
BEGIN
  DBMS_OUTPUT.PUT_LINE('W_INDS');
END;
```

---调用该过程

```
BEGIN
  SP_WINDS;
END;
```

---2、带输入参数的过程 ----给过程里的变量赋值来控制过程

CREATE OR REPLACE PROCEDURE SP_NAME(P_NAME IN VARCHAR2) ---相当于声明变量来接收参数

```
IS
BEGIN
  DBMS_OUTPUT.PUT_LINE(P_NAME);
END;
```

----调用该过程

```
BEGIN
  SP_NAME('WINDS');
END;
```

---3、带输入和输出参数的过程，输出的参数要作为另一个过程的输入参数

```
CREATE OR REPLACE PROCEDURE SP_INOT_SAL(P_EMPNO IN
NUMBER,P_SAL OUT NUMBER)
AS
BEGIN
  SELECT SAL INTO P_SAL FROM EMP WHERE EMPNO =
P_EMPNO;
END;
```

---调用该过程，有输出参数时，需要定义一个变量接收输出参数，

```
DECLARE
V_SAL NUMBER;
BEGIN
  SP_INOT_SAL(7369,V_SAL);
  DBMS_OUTPUT.PUT_LINE(V_SAL);
END;
```

- 存储过程的语句：一般可以先创建匿名块，再根据匿名块进行修改换成命名块，即更换参数

▼ ● 调用存储过程

- BEGIN
过程名称[(参数)];
END;

- 代码演示

```
--调用该过程  
BEGIN  
    SP_NAME;  
END;
```

- ▼ ● 删除存储过程

- DROP PROCEDURE 过程名;

- ● 怎么查看过程是否正确，主要用来调试过程：鼠标放在过程名，右键--view-运行查看是否正确

- ● 直接使用命名块过程建表，会出现权限不足的情况，这时候在存储过程名称后面添加Authid Current_Suer

- ▼ b. 函数(function)

- ① 函数：与过程非常相似，都是命名块的程序块，语句结构也非常相似

- ▼ ② 函数与过程的区别

- 函数会具有一个返回值，而过程只是为了执行一系列的行为
 - 在调用时，函数可以作为表达式的一部分被调用，而过程只能作为一个PL/SQL语句进行调用

- ▼ ③ 语法格式

- ▼ ● 创建函数

- 语法格式：创建函数

```
CREATE [OR REPLACE] FUNCTION 函数名(参数1 数据类型, 参数  
2, [IN] 数据类型.....)  
RETURN 返回的数据类型  
IS|AS  
PL/SQL函数体; --里面必须要有一个RETURN子句
```

- 代码演示

--输入N，就返回N的阶乘

```
CREATE OR REPLACE FUNCTION F_N1(P_N IN NUMBER)
```

```
RETURN NUMBER ---必须有一个返回值
```

```
IS
```

```
A NUMBER;
```

```
B NUMBER := 1 ;
```

```
BEGIN
```

```
FOR A IN 1..P_N LOOP
```

```
  B := B * A;
```

```
END LOOP;
```

```
RETURN B;
```

```
END;
```

---调用函数

```
BEGIN
```

```
  DBMS_OUTPUT.PUT_LINE(F_N1(4));
```

```
END;
```

---可以作为表达式被调用

```
SELECT F_N1(10) FROM DUAL;
```

- ▼ ● 删除函数

- 语法格式：删除函数

```
DROP FUNCTION 函数名;
```

- ● 注意：与过程中的参数类似，不能对函数的参数或返回值的类型添加长度约束。

- 代码演示

代码演示：

--输入表名，就返回一个表的数据量

---运行时才知道要查询那个表，所以使用动态SQL

```
CREATE OR REPLACE FUNCTION F_ROWNUMS(P_TABLE
```

```
VARCHAR2)
```

```
RETURN NUMBER
```

```
AS
```

```
V_ROWNUMS NUMBER;
```

```
SQL_QUERY VARCHAR2(100) := 'SELECT COUNT(*) FROM '||
```

```
P_TABLE; ---定义动态SQL语句
```

```
BEGIN
```

```
  EXECUTE IMMEDIATE SQL_QUERY INTO V_ROWNUMS; ---动态执行SQL并返回结果值
```

```
  RETURN V_ROWNUMS; ---返回函数结果
```

```
END;
```

- ▼ c. 存储过程 (procedure) 和函数 (function) 的区别

- 1.返回值的区别,函数有1个返回值,而存储过程是通过参数返回的,可以有多个或者没有
 - 2.调用的区别,函数可以在查询语句中直接调用,而存储过程必须单独调用.

- 函数一般情况下是用来计算并返回一个计算结果而存储过程一般是用来完成特定的数据操作（比如修改、插入数据库表或执行某些DDL语句等等）

▼ 2、触发器（TRIGGER）

- (1)定义：本身就是一个命名的语句块，定义的方式与PL/SQL语句块的定义没有太多区别。不同之处在于调用方式，触发器总是隐式地被调用，不能接收任何参数

▼ （2）触发器的使用

- 完成表的变更校验：当表的数据发生INSERT、UPDATE或DELETE操作
①时，提供验证逻辑，比如验证更改的数据的正确性、检查完整性约束、记录事件日志等操作。
- ②自动数据库维护：通过使用系统级的触发器，可以在数据库系统启动或退出时，通过触发器完成系统的初始化和清除操作。
- 控制数据库管理活动：可以使用触发器来精细地控制数据库管理活动，
③比如删除或修改表等操作，通过将逻辑放到这种触发器中，使得DDL操作的检查有了可保证性。

▼ （3）使用触发器来同步更新

▼ 扩展：merge into

- 在进行SQL语句编写时，我们经常会遇到大量的同时进行Insert/Update的语句，
也就是说当存在记录时，就更新(Update)，不存在数据时，就插入(Insert)。

▪ 代码演示

---触发器作业2：同步更新

--同步跟新

CREATE OR REPLACE TRIGGER t_emp2

AFTER INSERT OR DELETE OR UPDATE ON emp3 --触发器作用的表对象以及触发的条件和触发的动作

FOR EACH ROW --行级别的触发器

BEGIN

IF INSERTING OR UPDATING THEN

MERGE INTO EMP2 A

USING DUAL

ON (A.EMPNO = :NEW.EMPNO)

WHEN MATCHED THEN

UPDATE SET A.ENAME = :NEW.ENAME,

A.JOB = :NEW.JOB,

A.MGR = :NEW.MGR,

A.HIREDATE = :NEW.HIREDATE,

A.SAL = :NEW.SAL,

A.COMM = :NEW.COMM,

A.DEPTNO = :NEW.DEPTNO

WHEN NOT MATCHED THEN

INSERT (A.EMPNO,

A.ENAME,

A.JOB,

A.MGR,

A.HIREDATE,

A.SAL,

A.COMM,

A.DEPTNO)

VALUES (:NEW.EMPNO,

:NEW.ENAME,

:NEW.JOB,

:NEW.MGR,

:NEW.HIREDATE,

:NEW.SAL,

:NEW.COMM,

:NEW.DEPTNO);

ELSIF DELETING THEN

DELETE FROM EMP2 WHERE EMPNO = :OLD.EMPNO;

END IF;

END;

SELECT * FROM EMP2;

INSERT INTO EMP2 VALUES(1111,'WINDS','SALESMAN',111,TO_DATE('2020-01-06','YYYY-MM-DD'),2000,300,50);

SELECT * FROM EMP3;

▼ 3、程序包 (package)

▼ (1)定义

- 包就是把相关的存储过程、函数、变量、常量和游标等PL/SQL程序组合在一起，并赋予一定的管理功能的程序块

▼ (2) 程序包的组成：包定义（即包规范）和包体

▼ ❶ 包定义

- 包定义主要是包的一些定义信息，不包含具体的代码实现部分，也可以说包规范是PL/SQL程序和其他应用程序的接口部分，包含类型、记录、变量、常量、异常定义、游标和子程序的声明

▼ ● 创建包规范

- 包规范创建语法：CREATE [OR REPLACE] PACKAGE package_name
{IS |AS}
type_definition | procedure—specification | function_specification |
variable—specification | exception—declaration | cursor—
declaration | pragma_declaration
END [package_name]

语法格式：创建包头

CREATE [OR REPLACE] PACKAGE 包名

IS|AS

变量、常量及数据类型定义；

游标定义头部；

函数、过程的定义和参数列表以及返回类型；

END [包名];

▼ 语法解析

- CREATE [OR REPLACE] PACKAGE 表示将创建一个包规范，OR REPLACE 是可 选的关键字，如果不使用该关键字，在创建包规范时检测到同名的包时会报错。使用了 OR REPLACE之后，如果有同名的包规范，则先删除现有的包，然后创建 一个新的包。
- package_name是包名称，遵循PL/SQL的标识符命名规范。
- 从type_definition到cursor_declaration是在包规范中可以定义的各种类型，比如集合、记录、变量、常量、异常、游标、过程和函数等
- pragma_declaration用来指定包规范中的编译提示。
- ▼ 除了过程和函数之外，包规范中的元素与匿名块中声明部分一样，声明段的语法规则如下：
 - 1.包规范中声明的元素可以以任何顺序出现，但是引用的每个对象都必须先进行声明。
 - 2. 任何过程和函数的声明都必须是预先声明的，而不包含代码。实现包的过程和函数的代码在包体中。

▪ 代码演示：

```
CREATE OR REPLACE PACKAGE MYPACKAGE
AS
A NUMBER;
PROCEDURE MY_SP(P_A IN NUMBER);
FUNCTION MY_FUN(F_A NUMBER) RETURN NUMBER;
END;
```

- 代码演示

```
--定义包规范
CREATE OR REPLACE PACKAGE EMP_ACTION_PKG
AS
    V_DEPTNO NUMBER(3) := 20; ---包公开的变量
    PROCEDURE SP_NEWDEPT( ----定义一个新增加的员工过程
        P_DEPTNO DEPT.DEPTNO%TYPE, ----部门编号
        P_DNAME DEPT.DNAME%TYPE, ---部门名称
        P_LOC DEPT.LOC%TYPE ---位置
    );
    FUNCTION getraisesalary (P_EMPNO EMP.EMPNO%TYPE) ---定义一个
    获取员工加薪数量的函数
    RETURN NUMBER;
END EMP_ACTION_PKG; ----停止运行
```

- ▼ 2 包体

- 包体是对包规范中声明的子程序的实现部分，包体的内容对于外部应用程序来说是不可见的，包体就像一个黑匣子一样，是对包规范的实现
- 一个包可以没有包体部分，而且可以调试、改进和替换包体而无须改变包的规范部分

- ▼ 语法格式：创建包体

- CREATE [OR REPLACE] PACKAGE BODY package_name {IS | AS} _
type_definition | procedure—specification | function—specification |
variable—specification | exception_declaration | cursor—declaration
| pragma—declaration | cursor_body BEGIN —
sequence_of—statements END [package_name];
- 包体的声明和包规范的声明一样，都可以有集合、记录、子程序等的声明，只不过包规范中的声明是全局的，在包的任何部分都是可见的，而包体部分的声明只是对于包体是可见的，只是包的私有部分，外部的程序是看不到的
- 代码演示

- ▼ 3 包的调用

- BEGIN
包名.变量名|常量名
包名.游标名[(参数)]
包名.函数名[(参数)]|过程名[(参数)]
END;
- 代码演示
---调用包
BEGIN
EMP_ACTION_PKG.NEWDEPT(70,NULL,NULL);
DBMS_OUTPUT.PUT_LINE(EMP_ACTION_PKG.getraisesalary(7369));
END;

- ▼ 4 创建完整包的代码演示

▪ 代码演示

--1、定义包规范

```
CREATE OR REPLACE PACKAGE EMP_ACTION_PKG
AS
    V_DEPTNO NUMBER(3) := 20; ---包公开的变量
    PROCEDURE NEWDEPT( ---定义一个新增加的员工过程
        P_DEPTNO DEPT.DEPTNO%TYPE, ---部门编号
        P_DNAME DEPT.DNAME%TYPE, ---部门名称
        P_LOC DEPT.LOC%TYPE ---位置
    );
    FUNCTION getraisesalary (P_EMPNO EMP.EMPNO%TYPE) ---定义一个
    获取员工加薪数量的函数
        RETURN NUMBER;
END EMP_ACTION_PKG; ----停止运行
```

---2、定义包体

```
CREATE OR REPLACE PACKAGE BODY EMP_ACTION_PKG
AS
```

---2.1、公开，实现包规范中定义的NEWDEPT过程

```
PROCEDURE NEWDEPT(
    P_DEPTNO DEPT.DEPTNO%TYPE, ---部门编号
    P_DNAME DEPT.DNAME%TYPE, ---部门名称
    P_LOC DEPT.LOC%TYPE ---位置
)
AS
    V_DEPTCOUNT NUMBER; --声明一个变量，保存是否存在部门编号
    BEGIN
        SELECT COUNT(*) INTO V_DEPTCOUNT FROM DEPT WHERE
        DEPTNO = P_DEPTNO; ---查询在dept表中是否存在部门编号 P_DEPTNO
        IF V_DEPTCOUNT > 0 ---如果存在相同的部门记录
            THEN ---抛出异常
                RAISE_APPLICATION_ERROR (-20002,'出现相同的部门记录');----触发异
                常，该命名块运行结束
            END IF;
        INSERT INTO DEPT(DEPTNO,DNAME,LOC)
            VALUES(P_DEPTNO,P_DNAME,P_LOC);----如果不存在相同的部门记
            录，则插入新的记录
        COMMIT;
    END NEWDEPT; ---结束包体中的过程块
```

---2.2、公开，实现包规范中定义的 getraisesalary 函数

```
FUNCTION getraisesalary (P_EMPNO EMP.EMPNO%TYPE)
    RETURN NUMBER
AS
    V_JOB EMP.JOB%TYPE; ---职位变化
    V_SAL EMP.SAL%TYPE; ---薪资变化
    V_SALARYRATIO NUMBER(10,2); ---调信比率
    BEGIN
        SELECT JOB ,SAL INTO V_JOB,V_SAL FROM EMP WHERE EMPNO =
        P_EMPNO; ---获取员工表的薪资信息
        CASE V_JOB --根据不同的职位获取调薪比率
            WHEN 'CLERK' THEN
```

```

V_SALARYRATIO := 1.09;
WHEN 'SALESMAN' THEN
V_SALARYRATIO := 1.11;
WHEN 'MANAGER' THEN
V_SALARYRATIO := 1.18;
ELSE
V_SALARYRATIO := 1;
END CASE;
IF V_SALARYRATIO != 1
THEN
RETURN ROUND(V_SAL * V_SALARYRATIO,2);
ELSE
RETURN V_SAL;
END IF;
EXCEPTION
WHEN NO_DATA_FOUND THEN
RETURN 0;
END getraisesalary; ---结束函数块

```

```

--3、私有，该函数在包规范中并不存在，只能在包体内被引用
FUNCTION CHECKDEPTNO(P_DEPTNO DEPT.DEPTNO%TYPE)
RETURN NUMBER
AS
V_COUNTER NUMBER(2);
BEGIN
SELECT COUNT(*) INTO V_COUNTER FROM DEPT WHERE DEPTNO =
P_DEPTNO;
RETURN V_COUNTER;
END CHECKDEPTNO; ---结束私有包体
END EMP_ACTION_PKG; ---结束包

```

- 代码的实现过程如以下步骤

- (1)在包规范emp_action_pkg中，定义了一个公共的包变量v_depto和两个子程序，这3个元素将被公开，可以被任何其他的包或语句块调用。
- (2)在包体的实现中，除了对包规范中的两个元素进行实现之外，还定义了一个私有的函数checkdeptno,该函数仅可以被包体内的其他子程序调用，而不能由外部的包或语句块调用。

▼ 5 包的删除

- 语法格式：删除包

DROP PACKAGE 包名