PL/SQL 基础

单世民

大连理工大学

PL/SQL 简介

- PL/SQL（Procedural Language/SQL）
  PL/SQL是基于Ada编程语言的结构化编程语言，是由Oracle公司从版本6开始提供的专用于Oracle产品的数据库编程语言。用户可以使用PL/SQL语言编写过程、函数、程序包、触发器等PL/SQL代码，并且把这些代码存储起来，以便由具有适当权限的数据库用户重新使用。
  PL/SQL对大小写不敏感，因此用户应该选择符合自己的编码标准来描述性地规范自己的PL/SQL代码形式。
  PL/SQL代码使用了程序块（代码块），利用模块化方式进行构建。

PL/SQL 代码块

- PL/SQL语言基本的程序单元就是PL/SQL代码块，简称块。块是指令的集合，这些指令包括：
  - Oracle要执行的指令
  - 向屏幕显示信息的指令
  - 将数据写入文件的指令
  - 调用其它程序的指令
  - 操作数据的指令
  - 。。。
- PL/SQL程序至少会包含一个代码块

PL/SQL 代码块

- PL/SQL代码块的实现形式：
  - 只执行一次且永不存储的PL/SQL程序
  - 存储在数据库中以备后用的块

- PL/SQL代码块支持所有的DML语句，并可以通过使用本地动态SQL（Native Dynamic SQL，NDS）或内置的DBMS_SQL包运行DDL语句。

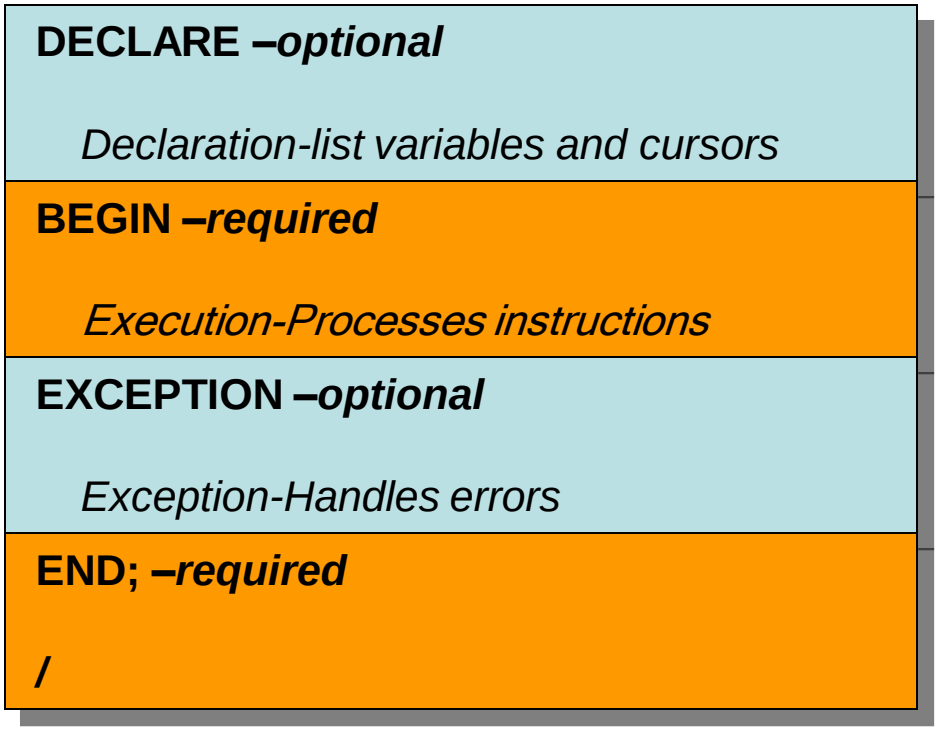
PL/SQL 代码块结构

- 最小的一个PL/SQL代码块结构

```
BEGIN
   NULL;
END;
/
```

PL/SQL 代码块结构

- 代码块的基本结构



PL/SQL 代码块组成部分

- 声明部分（DECLARE）
  声明部分是一个可选代码段，它将要在块内使用的变量和这些变量支持的数据类型捆绑在一起。声明部分是定义和归档在程序中用到的所有局部变量的地方。

```
DECLARE
    v_date DATE;
……
```

PL/SQL 代码块

- 执行部分（BEGIN）
  执行部分是代码块中唯一的一个不可缺少的代码段。执行部分的代码必须是完整的，才能让代码块进行编译。所谓完整，指的是在BEGIN和END关键字之间的必须是要送给PL/SQL引擎的一个完整的指令集。

- 执行部分支持DML的所有命令和SQL*Plus的一些内置功能，它还支持使用本地动态SQL和（或）DBMS_SQL内置包的DDL命令。

PL/SQL 代码块组成部分

- 异常处理部分（EXCEPTION）
  异常处理部分也是一个可选代码段，它捕获在程序执行过程中产生的错误。
- 如果删除异常处理部分，不会对代码的执行产生任何不良影响，不过这样会使用户无法获知在代码块执行过程中产生的任何错误

```
……
EXCEPTION
     WHEN OTHERS
     THEN
          DBMS_OUTPUT.PUT_LINE(sqlerrm);
END;
/
```

PL/SQL 代码块种类

- 匿名块

  匿名块没有名称，也不在数据库中存储，它们能够调用其它程序，但是自己不能被其他程序调用。

- 如何运行匿名块
  - 可以从某个文件上运行这个匿名块
  - 直接在SQL>提示符下键入匿名块的全部代码以后立即运行

PL/SQL 代码块种类

- 命名块
  命名块与匿名块的主要区别是命名块都有名称。命名块在匿名块的基本块结构上增加了一个头（HEADER）部分。头部分告诉Oracle该块的名称，以及该块是一个过程还是一个函数。

- 运行的时候，命名块并不会立即执行，而是先通过编译，然后在数据库中存储，最后才会执行。

PL/SQL 代码块种类

- 命名块-编译错误
  命名块的一个最大优势是：命名块中语法、依赖关系和权限等错误都是在过程或函数的编译过程中被捕获，而不是在运行时被捕获。
  创建过程或函数的时候，Oracle会首先编译代码，并检查其依赖性和语法是否正确。如果存在问题，Oracle就会返回一个错误信息，并将该命名块标识为无效

PL/SQL 代码块种类

- 嵌套块

  块中还可以包含其他子块，即嵌套，在代码块的异常处理部分和执行部分都允许存在嵌套块，但是在声明部分不允许存在嵌套块。

PL/SQL 代码块种类

- 触发器
  触发器提供了PL/SQL的一种特殊实现。它们存储在数据库中，但又不是存储过程或函数。触发器由事件驱动，并且与执行在数据库中的某种操作关联在一起。

PL/SQL的语言规则与约定

- 标识符
  标识符用于为PL/SQL对象和数据库对象提供命名标识。它让我们可以不通过Oracle的一些内部引用，只使用名称就可以引用所需对象。
- 标识符的命名约束包括：
  - 标识符名称的长度必须在30个字符以内（包括30）
  - 标识符的名称必须以字母开头
  - 标识符名称中可以包含$、#、_以及全部数字字符，但是不能将它们作为名称的第一个字符
  - 名称中不能包含标点符号、空格和连字符号（“-”）
  - 不可以使用Oracle关键字

PL/SQL的语言规则与约定

- 字面值

  字面值指的是没有使用标识符进行描述的字符、字符串、数字、布尔和日期/时间等类型的值

  - 字符　　‘C’ ‘c’ ‘~’
  - 字符串　　‘数据库’
  - 数字　　-6 +3.1 -2.32E+5
  - 布尔值　　TRUE,FALSE,NULL
  - 日期/时间　　‘2004-06-05 22:14:01’

PL/SQL的语言规则与约定

- 注释

  - 单行注释
    使用双连字符号（--）打头，紧接连字符的这一行文本就是注释文本

  - 多行注释
    多行注释以一个注释分隔符“/*”打头，以另一个注释分隔符“*/”终止。注释的范围可以跨过多条代码行，也可以跨越多个代码段

PL/SQL的数据类型

- PL/SQL的数据类型可以分为：
  - ¤ 标量类型
    - 字符/字符串类型
    - 数字类型
    - 布尔类型
    - 日期/时间类型
  - ¤ 引用类型
    - REF CURSOR
    - REF
  - ¤ 复合类型
    - 记录
    - 嵌套表
    - Index-by表
    - Varrays
    - 对象类型
  - ¤ LOB（大对象）类型

PL/SQL中的变量（与常量）

- 用户使用的所有变量和常量都必须在程序块中的声明部分定义。对于每一个变量，用户都必须规定名称和数据类型，以便在可执行部分为其赋值。
- 在何处为变量赋值是可以选择。既可以选择在可执行部分中为变量赋值，也可以选择在声明变量时同时为其赋值。

PL/SQL中的变量

- 定义变量的语法

```
variable_name [CONSTANT] type [NOT NULL] [:=value];
```

- 示例

```
DECLARE
     example_number_variable number:= 60;
BEGIN
     null;
END;
/
```

PL/SQL中的变量

- 用户可以使用很多方法为变量(在程序块的声明部分和可执行部分)和常量(在声明部分)赋值。最常用的赋值方法是使用PL/SQL的赋值运算符(:=)。赋值运算符的语法格式如下：

```
variable := expression;
```

- 在程序块的声明部分初始化变量时，可以使用default关键字。使用default关键字为变量赋值，表示在可执行部分既可以直接引用该变量的值，也可以重新为该变量赋值。
- 在声明变量时，可以为变量指定not null属性。not null属性表示该变量不允许空，必须为其赋予明确的值。

PL/SQL中的变量

```
DECLARE
     example_variable_1 varchar2(100);
     example_variable_2 varchar2(100) := '';
     example_variable_3 varchar2(100) := null;
     example_variable_4 varchar2(100) default null;
BEGIN
     null;
END;
/
```

- 在PL/SQL程序块中，如果引用某个已经声明的变量，且没有为该变量赋值，那么该变量的值就是null。也就是说，null通常会赋予在声明时没有赋值的变量。
- null关键字表示缺少、不可知或不适用等含义。从本质上来讲，null关键字表示没有内容。

PL/SQL中的变量

- %TYPE和%ROWTYPE
  变量的声明可以直接映射到数据库的某一列上或整个表上，这样可以采用%TYPE或%ROWTYPE将这个变量锚定到数据库相应的列或表：

  - 当用户声明单独的变量而不是记录时——%type。
  - 当用户声明表示表、视图或游标的完整行的记录变量时——%rowtype。

PL/SQL中的变量

- 变量的生存范围（可视性和作用域）
  变量的生存范围指的是代码在代码块中的可访问性和可用性。只有在它的生存范围内，变量才是有效的。

```
DECLARE
    v_video_name VARCHAR(100);

BEGIN
    DECLARE
        v_director VARCHAR(20);
    BEGIN
    ...
    END;
EXCEPTION
 ...
END;
/
```

PL/SQL的程序流控制

- 条件判断
  IF-THEN；IF-THEN-ELSE；
  IF-THEN-ELSIF；CASE；
- 循环执行
  简单LOOP；WHILE LOOP；FOR 数字 LOOP
- 跳转
  GOTO label_name;
  配合使用标签：<<label_name>>

- 示例演示

PL/SQL程序示例

- 输出方法

```
DECLARE
    x varchar2(10);
BEGIN
    x:='this is ...';
    dbms_output.put_line('x的值为:'||x);
END;
/
```

注意SERVEROUTPUT参数的设置

PL/SQL程序示例

- IF

```
DECLARE
    a number;
    b varchar2(10);
BEGIN
    a:=2;
    IF a=1 THEN
       b:='A';
    ELSIF a=2 THEN
       b:='B';
    ELSE
       b:='C';
    END IF;
    dbms_output.put_line('b的值是'||b);
END;
/
```

PL/SQL程序示例

- CASE

```
DECLARE
    a number;
    b varchar2(10);
BEGIN
    a:=2;
    CASE
      WHEN a=1 THEN b:='A';
      WHEN a=2 THEN b:='B';
      WHEN a=3 THEN b:='C';
    ELSE
      b:='OTHERS';
    END CASE;
    dbms_output.put_line('b的值是'||b);
END;
/
```

PL/SQL程序示例

- CASE

```
DECLARE
    a number;
    b varchar2(10);
BEGIN
    a:=2;
    CASE a
       WHEN 1 THEN b:='A';
       WHEN 2 THEN b:='B';
       WHEN 3 THEN b:='C';
    ELSE
      b:='OTHERS';
    END CASE;
    dbms_output.put_line('b的值是'||b);
END;
/
```

PL/SQL程序示例

- LOOP

```
DECLARE
    x number;
BEGIN
    x:=0;
    LOOP
       x:=x+1;
       IF x>=3 THEN
           EXIT;
       END IF;
       dbms_output.put_line( '内:x='||x );
    END LOOP;
    dbms_output.put_line('外:x='||x);
END;
/
```

PL/SQL程序示例

- LOOP

```
DECLARE
    x number;
BEGIN
    x:=0;
    LOOP
       x:=x+1;
       EXIT WHEN x>=3;
       dbms_output.put_line( '内:x='||x );
    END LOOP;
    dbms_output.put_line('外:x='||x);
END;
/
```

PL/SQL程序示例

- WHILE

```
DECLARE
    x number;
BEGIN
    x:=0;
    WHILE x<=3 LOOP
       x:=x+1;
       dbms_output.put_line( '内:x='||x );
    END LOOP;
    dbms_output.put_line('外:x='||x);
END;
/
```

PL/SQL程序示例

- FOR

```
BEGIN
    FOR i IN [REVERSE] 1..5 LOOP
       dbms_output.put_line( 'i='||i );
    END LOOP;
    dbms_output.put_line('end of loop');
END;
/
```

PL/SQL程序示例

- GOTO-标签

```
DECLARE
    x number;
BEGIN
    x:=0;
    <<repeat_loop>>
    x:=x+1;
    dbms_output.put_line('内:x='||x);
    IF x<3 THEN
       GOTO repeat_loop;
    END IF;
    dbms_output.put_line('外:x='||x);
END;
/
```

错误处理

PL/SQL中错误的类型

错误类型	报告者	处理方法
编译时错误	PL/SQL编译器	交互式地处理；编译器报告错误，必须更正这些错误
运行时错误	PL/SQL运行时引擎	程序化地处理；异常由异常处理子程序引发并进行捕获

异常

- 在PL/SQL中的一个警告或错误均被称为异常

  PL/SQL的异常和Java的异常非常相似（异常的引发和捕获方式）。但是，与Java异常不同，PL/SQL的异常不是对象，也没有定义在PL/SQL异常上的方法。

异常的分类

- 发生运行时错误时，就会引发异常
- 异常分为两类：
  - 预定义异常（系统异常）
  - 用户自定义异常

预定义异常

- 预定义异常对应最常见的Oracle错误，这些异常的标识符都是STANDARD标准包中定义的。
- 常见的预定义异常

Oracle错误编号	等价异常的名称	说明
ORA-0001	DUP_VAL_ON_INDEX	违反唯一性约束
ORA-1403	NO_DATA_FOUND	没有找到数据
ORA-1422	TOO_MANY_ROWS	SELECT INTO语句查询到的记录超过了一行
ORA-1476	ZERO_DIVIDE	被0整除
ORA-6502	VALUE_ERROR	切割运算、算术运算或转换运算中出现的错误

用户自定义异常

- 用户自定义异常就是有程序员自己定义的一个错误。
  用户自定义异常是在PL/SQL块的声明部分声明的。与变量相似，异常也有一个类型和有效范围
- 例：

```
DELCARE
    e_DuplicateActors EXCEPTION;
```

异常的引发

- 自定义异常：使用RAISE语句**显式**地引发
- 预定义异常：在与异常相关联的Oracle错误发生时**隐式**引发

- 如果发生一个错误，而此错误没有和异常相关联，也会引发一个异常，此异常可用OTHERS子程序进行捕获

预定义异常也可以使用raise语句显式地进行引发

EXCEPTION_INIT编译器指令

- 思考：有没有必要让一个自定义指令和某个特定的Oracle错误相关联？

- 有必要，并且可以通过编译器指令EXCEPTION_INIT实现（编译指令也叫pragma）。

```
PRAGMA EXCEPTION_INIT(exception_name,oracle_error_number);
```

- 此编译指令只能出现在代码的声明部分。
- 事实上，预定义异常就是在SYS.STANDARD包中，使用EXCEPTION_INIT编译指令与相应的oracle错误关联在一起的。

RAISE_APPLICATION_ERROR

- 可以使用RAISE_APPLICATION_ERROR创建自己的错误消息，这种方式比命名异常更具有描述性。

```
RAISE_APPLICATION_ERROR(error_number,error_mesg,[keep_errors]);
```

- error_number:-20000到-20999
- error_mesg:长度必须小于512个字符

RAISE_APPLICATION_ERROR

- 可以使用RAISE_APPLICATION_ERROR的一个好处在于：可以在创建错误消息文本的时候，将程序当前的某些变量值包含在错误消息文本中。

- 可以使用RAISE_APPLICATION_ERROR，以与Oracle相同的处理方式，向用户返回产生错误的原因。

异常的处理

- 语法结构

```
EXCEPTION
    WHEN exception_name THEN
            sequence_of_statements1;
    WHEN exception_name THEN
            sequence_of_statements2;
    [WHEN OTHERS THEN
            sequence_of_statements3;]
```

可以为多个异常设计一个共同的异常处理子程序。在WHEN子句中，用关键字OR分隔各个异常的名称即可
但是，某个给定的异常最多只能由一个异常处理子程序处理。

OTHERS异常处理子程序

- 当前异常处理部分定义的所有WHEN语句都没有处理的任意一个已引发的异常，都会导致执行这个OTHERS异常处理子程序。该异常处理子程序应该总是作为代码块的最后一个异常处理子程序。

- 思考：OTHERS异常处理子程序与Java异常处理中的那种情况相类似？Exception类还是Finally？

异常的传播

- 没有处理的异常将沿检测异常调用程序传播到外面，当异常被处理并解决或到达程序最外层传播停止。

在声明部分抛出的异常将控制转到上一层的异常部分。

异常的范围

- 异常像变量一样，有一定的范围。如果用户自定义异常传播到它的范围之外，就不能再通过名称应用它。
  这个时候怎么办？

- 定义一个公有包
- 使用RAISE_APPLICATION_ERROR

OTHERS异常处理子程序

- 较好的应用方式是将OTHERS处理程序放置在程序的最顶层，及最外层的代码块中，以保证不漏掉任何错误。

- 否则，被遗漏的错误就会传播到外部的调用环境中。此时，事务会被服务器自动回滚。

- 思考：如下处理是否合理？

```
WHEN OTHERS THEN NULL;
```

查看错误堆栈

- SQLCODE（过程化函数）
  - 返回当前的错误代号
  - 对于用户定义异常，返回1
- SQLERRM（过程化函数，最大512个字符）
  - 返回当前的错误信息文本
  - 对于用户定义异常，返回“User-defined Exception”
  - SQLERRM可以使用一个数字来调用，参数应该是负数。0-成功结束；+N-non-ORACLE Exception；+100-no data found
  - 思考：使用不带参数的SQLERRM好还是使用带参数的SQLERRM好？

查看错误堆栈

- DBMS_UTILITY.FORMAT_ERROR_STACK
  - 包函数，可以直接在SQL语句中使用
  - 返回值与SQLERRM相似，并且限制文本的最大长度为2000个字节
- DBMS_UTILITY.FORMAT_ERROR_BACKTRACE
  - 包函数，可以直接在SQL语句中使用
  - 不受2000字节长度的限制

示例

```
EXCEPTION
    WHEN e_exception2 THEN
          insert into…;
    WHEN e_exception1 THEN
          insert into…;
    WHEN e_exception1 OR e_exception2 THEN
          insert into…;
    WHEN e_exception1 AND e_exception2 THEN
          insert into…;
```

引发编译错误：某个给定的异常最多只能由一个异常处理子程序程序进行处理

引发编译错误：不能为为多个同时发生的异常定义同一个异常处理子程序。事实上，根本就不存在多个异常同时发生的情况。

PL/SQL 代码块

- 编译时警告
  通过编译的命名块，在一定程度上说明代码没有出现语法错误，所依赖的对象也都存在并且可用。但是，它并不能保证代码一定能保证代码能够顺利执行，也不能保证执行效率。

- 编译匿名块的时候，编译时警告会提供一些附加的反馈信息——编译匿名块的时候，不会有警告信息。

警告信息

- 可以为某一PL/SQL程序（如某一个命名块、当前会话或整个数据库实例中要编译的所有命名块），启用不同的警告设置。

- 查看警告设置的方法

```
SHOW PARAMETER PLSQL_WARNINGS
```

```
SELECT DBMS_WARNING.GET_WARNING_SETTING_STRING() WARNING_LEVEL
FROM  dual
```

警告信息

- 警告信息
  - ALL 包括所有可用的警告条件和警告信息
  - PERFORMANCE 仅返回与执行性能相关的警告
  - INFORMATIONAL 将那些对程序可能没有任何作用的代码标记为可以移除的或可以更正的。
  - SEVERE 被标记为严重的问题，表示代码可能会存在逻辑问题
  - Specific Error 对某一个错误信息而言，该警告可能是具体的。

警告信息

- 更改警告设置的方法

系统级

```
ALTER SYSTEM
SET PLSQL_WARNINGS='ENABLE:PERFORMANCE','ENABLE:SEVERE';
```

会话级

```
CALL DBMS_WARNING.
     SET_WARNING_SETTING_STRING('ENABLE:ALL','SESSION');
```

小结

- PL/SQL简介
- PL/SQL代码块
  - 简介
  - 结构
  - 组成部分（声明，执行，异常处理）
  - 类型（匿名块，命名块，嵌套块，触发器）
  - 命名规则与约定
  - 数据类型与变量
  - 程序流控制
  - 异常（异常的处理，错误分类，异常分类，异常的引发与传播）
  - 警告信息

The End