O'REILLY®

异步图书
www.epubit.com.cn

第6版
下册

Oracle PL/SQL 程序设计

Oracle PL/SQL Progr

[美] Steven Feuerstein 著
Bill Pribyl
方鑫 译

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

O'REILLY®

Oracle PL/SQL 程序设计（第6版）

（下册）

[美] Steven Feuerstein　Bill Pribyl　著

方　鑫　译

人民邮电出版社

北京

图书在版编目（CIP）数据

Oracle PL/SQL程序设计 : 第6版 : 全2册 / (美)史蒂芬·弗伊尔斯坦 (Steven Feuerstein) , (美) 比尔·普里比尔 (Bill Pribyl) 著 ; 方鑫译. -- 2版. -- 北京 : 人民邮电出版社, 2017.7
ISBN 978-7-115-44875-0

Ⅰ. ①O… Ⅱ. ①史… ②比… ③方… Ⅲ. ①关系数据库系统－程序设计 Ⅳ. ①TP311.138

中国版本图书馆CIP数据核字(2017)第100091号

版权声明

◆ 著　[美] Steven Feuerstein　Bill Pribyl
译　方　鑫
责任编辑　傅道坤
责任印制　焦志炜

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路 11 号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：800×1000　1/16
印张：70
字数：1461 千字　2017 年 7 月第 2 版
印数：8 001 – 10 500 册　2017 年 7 月河北第 1 次印刷

著作权合同登记号　图字：01-2013-8979 号

定价：188.00 元（上、下册）

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

目录（下）

第 5 部分　构造 PL/SQL 应用程序

第 6 部分 高级 PL/SQL 主题

第 5 部分

构造 PL/SQL 应用程序

本书的这一部分对前面的内容进行汇总。从本书开篇到现在，我们已经掌握了基础知识：我们了解了变量的声明和工作特点，也知道了如何进行异常处理和循环构建。现在，我们可以开始构造应用程序了——构建程序块，将过程、函数、程序包及触发器组合到一起——第 17 章到第 20 章介绍这些内容。在第 20 章，我们讨论如何管理这些 PL/SQL 代码，包括测试和调试程序、管理对象间的依赖关系；第 20 章还会对 Oracle 数据库 11g 第 2 版引入的基于版本的重定义功能进行简单介绍。第 21 章介绍如何通过各种工具，使我们的 PL/SQL 程序获得最佳性能。第 22 章介绍 PL/SQL 的 I/O 技术，包括 DBMS_OUTPUT（将结果输出到屏幕）、UTL_FILE（输入输出到文件）、UTL_MAIL（输出到邮件）及 UTL_HTTP（从 Web 页面获取数据）等。

第 17 章

过程、函数和参数

本章的前面部分、详细探究了 PL/SQL 语言的各种组件：游标、异常、循环、变量等。在使用 PL/SQL 编写应用时，当然需要了解这些组件，但把这些组件组合在一起，编写出结构优良、易读易懂、便于维护的程序，才是更重要的。

通常来说，我们接到的编程任务很少简单明了的。只有在极少数情况，才会出现解决方案一目了然，立刻就可以开始敲击键盘的情况。通常我们构建的系统，都是复杂而庞大的，有相当多互相影响甚至是冲突的组件。更进一步说，用户总是“贪得无厌”地要求程序功能再强大一些、使用再简单一些，这就使得应用程序内部的实现更加复杂化了。

当今我们专业面临的最大挑战是如何降低环境的复杂性。当面临一个复杂的难题时，我们会情不自禁地退缩，从何处着手？如何从众多需求和特性纠缠的丛林中杀出一条血路？

人类大脑的并行能力不如大型并行计算机，哪怕是最强大脑，也无法同时处理 7 件（上下浮动 2）以上的任务。所以我们必须把庞大的、令人生畏的项目分解成小的、更容易管理的组件，然后把这些组件再细分成易于理解的各个小程序。这样，我们就知道该如何构建和测试这些程序了。最后，我们再把这些组件组合在一起，构建成完整的应用程序。

无论我们使用的是“自顶至下”的设计方法（也称为逐步分解法，在 17.5 节我们将进行详述）还是其他设计方法，毋庸置疑的是，通过把程序代码模块化成过程、函数和对象类型，我们将会得到高质量和低管理成本的应用程序。

17.1 代码模块化

“模块化”是这样一个过程：把大的代码块拆分成若干个较小的片段（模块），这些小的模块又可以被其他模块调用。代码的模块化非常类似于数据的规范化，除了类似的好处之外，还有一些其他益处。利用模块化，我们的代码可以：

重用性更好

当我们把一大段代码（或整个应用程序）分解成单个的组件，并通过“即插即用”的思路把它们组合在一起时，我们会发现，在当前应用中，许多模块都可以被其他模块调用。如果设计得足够合理，一些工具程序甚至可以用于其他应用程序！

管理性更好

一个 1000 行的程序和 5 个各 200 行的互相调用的程序相比，哪个更容易调试？在处理小任务时，我们的注意力更容易集中，也能做得更好。在代码模块化的环境下，我们可以对每个程序单元进行测试和调试（我们称之为单元测试），然后再将各单元组合起来进行复杂的集成测试。

可读性更佳

模块可以被命名，通过名字来描述其行为。通过程序接口隐藏起来的代码越多，程序的行为就越容易被阅读和理解。模块化能帮助我们聚焦程序全局而不是一个个可执行语句。最终我们甚至可以研发出最高端的软件：完全自我说明的程序。

可靠性更高

以模块化思路开发的代码的错误会更少，而在一个模块中找到并排除错误也相对容易。而且，由于代码量减少、可读性提高，我们的代码维护起来也更容易。

一旦我们能够熟练地使用 PL/SQL 语言中的各种循环、条件、游标结构（IF 语句，循环等），我们就可以开始准备编程了。不过在此之前，我们还应该知道如何创建和组合 PL/SQL 模块。

PL/SQL 提供下述这些模块结构，我们可以用不同的方法实现代码的模块化：

过程

过程是一个可以像执行 PL/SQL 语句一样调用的程序，一个过程可以执行一个或多个动作。通过过程的参数列表，我们可以与过程交互信息。

函数

函数是一个通过 RETURN 子句返回数据的程序，使用起来就像一个 PL/SQL 表达式。我们也可以通过它的参数列表向其内部传递信息。函数也可以通过参数列表向外传递数据，但通常不建议这么做。

数据库触发器

当数据库中发生了某一事件时，一些列命令会被触发执行（如登录，修改了表中的一行记录，执行了一个 DDL 语句）等。

程序包

一个由若干过程、函数、类型和变量组成的被命名的集合叫作程序包。程序包并不是一个模块（它更像是一个元模块），但它很像一个模块，所以我们在这里一并列出。

对象类型或对象类型的实例

对象类型是面向对象类的 Oracle 版本（或仿真版本）。对象类型封装了状态和行为，把数据（如一个关系表）和规则（操作这些数据的过程和函数）整合在一起。

我们将在第 18 章讨论程序包，在第 19 章将会探讨数据库触发器，在第 26 章则有关于对象类型的详解。本章专注于讨论如何构建过程和函数，以及如何设计作为良好模块一部分的参数列表。

我们使用的数据“模块”泛指函数和过程。和许多其他编程语言一样，一个模块可以调用另一个命名模块，通过参数与模块传递信息。最后，PL/SQL 模块结构也紧密集成了异常处理机制，从而提供了一个全面的错误检查手段（详见第 6 章）。

本章将探讨如何定义过程和函数，然后深入设置参数列表的细节，对于在程序构建过程中可能会遇到的一些“怪异”的内容，包括局部模块、超载、前向引用、确定性函数和表函数等，也会有所涉及。

17.2 过程

一个过程，就是执行一个或多个动作的模块。由于在 PL/SQL 中，对过程的调用就是一个单独的可执行语句，因此，一个 PL/SQL 块中可以没有其他内容和只有一个过程调用语句。过程是模块化代码的关键组成部分，通过过程，我们可以把程序逻辑进行强化并在将来进行重用。

一个 PL/SQL 过程的通用形式如下：

```
PROCEDURE [schema.]name[( parameter[, parameter...] ) ]
   [AUTHID DEFINER | CURRENT_USER ]
   [ACCESSIBLE BY (program_unit_list)]
IS
   [declarations]

BEGIN
   executable statements

[ EXCEPTION
     exception handlers]

  END [name];
```

其中每个元素的作用如下。

schema

可选元素，拥有这个过程的 schema 的名字。默认是当前用户。如果拥有过程的 schema 不是当前用户，则当前用户必须具有 CREATE ANY PROCEDURE 权限，方能在其他 schema 下创建过程。

name

过程名。

(parameters[, parameter...])

参数列表，可选元素。我们通过这个列表，将信息在调用程序与过程间传入与传出。

***AUTHID* 子句**

这个子句用来定义该过程是用定义者（所有者）的权限还是当前用户的权限来运行。前一种（默认）模式称之为“定义者权限模型”，后一种模式称之为“调用者权限模型”。在第 24

章有对这些模型的详述。

declarations

过程内的局部标识符的声明。如果没有任何声明，则 IS 和 BEGIN 语句紧靠在一起。

***ACCESSIBLE BY* 子句（仅存在于 *Oracle Database 12c*）**

仅仅括号内的程序单元才有权限访问这个过程。第 24 章会对这一特性进行详细介绍。

executable statements

过程被调用时要执行的语句。在关键字 BEGIN 之后，关键字 END 或 EXCEPTION 之前，必须至少存在一个可执行语句。

exception handlers

过程中的异常处理机制。如果我们没有显式的异常处理规则，可以只用一个 EXCEPTION 关键字，然后用 END 关键字直接终止过程的执行。

图 17-1 演示了 apply_discount 过程，这个过程包含了一个有名字的 PL/SQL 代码块的全部 4 个部分及 1 个参数列表。

```
PROCEDURE apply_discount                                          ← 头部
 (company_id_in IN company.company_id%TYPE, discount_in IN NUMBER)
IS
 min_discount CONSTANT NUMBER:=.05;
 max_discount CONSTANT NUMBER:=.25;                               ← 声明部分
 invalid_discount EXCEPTION;
BEGIN
 IF discount_in BETWEEN min_discount AND max_discount
  THEN
   UPDATE item
    SET item_amount:=item_amount*(1-discout_in);
   WHERE EXISTS (SELECT 'x' FROM order
     WHERE order.order_id=item.order_id                           ← 执行部分
      AND order.company_id=company_id_in);
  IF SQL%ROWCOUNT = 0 THEN RAISE NO_DATA_FOUND; END IF;
 ELSE
  RAISE invalid_discount;
 END IF;
EXCEPTION
 WHEN invalid_discount
 THEN
  DBMS_OUTPUT.PUT_LINE('The specified discount is invalid.');
                                                                  ← 异常处理部分
 WHEN NO_DATA_FOUND
 THEN
  DBMS_OUTPUT.PUT_LINE('No orders in the system for company:'||
     TO_CHAR(company_id_in));
END apply_discount;
```

图 17-1　apply_discount 过程

17.2.1 调用一个过程

一个过程是作为一个可执行的 PL/SQL 语句被调用的。换而言之，过程的调用必须以分号（;）结尾，且在 PL/SQL 块中的可执行部分调用，之前或之后可以有其他 SQL 语句或 PL/SQL 语句（如果有的话）。

下面的可执行语句调用了 apply_discount 过程：

```
BEGIN
   apply_discount( new_company_id, 0.15 ); -- 申请了 15%的折扣。
END;
```

如果过程中没有任何参数，那么在调用过程时，可以带括号，也可以不带。

```
display_store_summary;
display_store_summary();
```

17.2.2 过程头部

在过程定义中，位于关键字 IS 之前的部分叫作过程的头部或过程签名。头部为程序员提供了调用该过程所需的全部信息，即：

- 过程名；
- AUTHID 子句（如果有的话）；
- 参数列表（如果有的话）；
- Accessible-by 列表（如果有的话），其为 Oracle 数据库 12c 的新特性。

理想情况下，一个程序员只需看到过程的头部就应该知道该过程可以完成哪些任务，以及如何调用该过程。

前一节提到的 apply_discount 过程的头部如下：

```
PROCEDURE apply_discount (
    company_id_in IN company.company_id%TYPE
  , discount_in IN NUMBER
)
```

它包含了模块类型、模块名称和含有两个参数的参数列表。

17.2.3 过程体

过程体就是实现这个过程的代码。它包含了过程的声明、执行和异常单元。关键字 IS 之后的所有代码组成了过程体。声明和异常单元是可选的。如果没有异常处理单元，就把 EXCEPTION 关键字留下来，用 END 关键字直接结束过程就可以了。如果没有声明单元，BEGIN 语句后面就会紧接着 IS 关键字。

一个过程至少要有一个可执行语句，通常来说，这不成问题。相反，要小心执行单元部分变得过

长而难于管理。我们应该努力保持执行单元的精炼和可读性。本章后面的部分有关于可执行单元管理的专门介绍，尤其是 17.5.1 节中关于提高可读性的部分。

17.2.4 END 标签

结束过程定义时，我们可以在关键字 END 后面直接带上过程的名字，像这样：

```
PROCEDURE display_stores (region_in IN VARCHAR2) IS
BEGIN
   ...
END display_stores;
```

过程名在这里作为一个标签，与过程起始的过程名相呼应，明确地标示出过程的结束。我们应该养成一个使用 END 标签的良好习惯，尤其是当一个过程的代码长达几页，或者是一个程序包体内的一系列过程与函数中的一个时，使用 END 标签会大大提高程序的可读性。

17.2.5 RETURN 语句

RETURN 语句通常与函数一起使用，因为函数需要通过 RETURN 语句来返回值(或是一个异常)。有趣的是，PL/SQL 也支持在过程中使用 RETURN 语句。过程中的 RETURN 不能带表达式，因此不能给调用程序单元传回值，它只是终止过程的执行，将控制权返回给调用程序单元。

通常而言，我们不会有太多的理由和机会在过程中使用 RETURN 语句，因为这会导致的代码的非结构化，使得过程至少会有两个出口，程序流变得难以理解和维护。在程序单元中，我们应该尽量避免使用 RETURN 和 GOTO 语句来绕开正常的控制结构和处理流程。

17.3 函数

所谓函数，就是通过 RETURN 语句而不是 IN 和 OUT 变量，进行数据返回的模块。和过程调用不同的是，过程可以作为一个可执行语句单独存在，而对函数的调用只是一个可执行语句的一部分，比如一个表达式中的一个元素，或变量声明时所赋予的默认值。

因为函数返回的是一个值，所以它有数据类型的属性。函数可以用来替换 PL/SQL 语句中具有相同数据类型的表达式。

对于构建模块化代码来说，函数是其中至关重要的结构。例如，应用程序中的每个业务规则或者公式都应该放到一个函数里，以便不同的程序重复调用，而不是在不同的程序反复编写相同的查询（“通过员工 ID 查询姓名”、“查询某个公司 ID 的最后订单记录”，等等）。这样做的结果是使得代码更易于调试、优化和维护。

提示

有一些程序员更喜欢用过程的参数列表来返回状态信息，而不是函数。如果我们也是其中之一，就要确保我们的业务规则、公式和单行查询都很好地隐藏在过程里！

如果一个应用程序缺少对函数的定义和使用，就很有可能随着时间的推移变得难以维护。