O'REILLY®

异步图书
www.epubit.com.cn

第6版
上册

Oracle PL/SQL 程序设计

Oracle PL/SQL Programm

[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）

（上册）

[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（上）

第 1 部分　用 PL/SQL 编程

第 2 部分 PL/SQL 程序结构

第 3 部分 PL/SQL 程序数据

第 4 部分 PL/SQL 中的 SQL

第 1 部分

用 PL/SQL 编程

本书的第 1 部分对 PL/SQL 进行说明，介绍如何创建和运行 PL/SQL 代码，以及 PL/SQL 语言的基础知识。在第 1 章中，我们提出这样的基本问题：PL/SQL 从何而来？它能干什么？PL/SQL 的主要特性是什么？在第 2 章中，我们让读者能够尽快上手使用 PL/SQL 语言，包括清晰、简单的 PL/SQL 代码执行指导，以及常见的环境说明。第 3 章中，我们回答了关于语言结构和关键字的常见问题：如何创建一个 PL/SQL 声明？PL/SQL 块结构是什么？如何在 PL/SQL 代码中添加注释？

第 1 章

PL/SQL 介绍

PL/SQL 是“结构化查询语言的过程化语言扩展”（Procedural Language extensions to the Structured Query Language）的英文缩写。SQL 是无处不在的关系型数据库查询和更新使用的语言。Oracle 公司引入 PL/SQL 来克服 SQL 中的一些短板，以给那些意欲在 Oracle 数据库上运行关键应用的企业提供一个更完整的编程解决方案。本章介绍 PL/SQL 的起源、它的不同版本，以及 PL/SQL 在最新的 Oracle 版本（Oracle 12c）中新特性的简单汇总，并为 PL/SQL 开发者提供了一些信息资源和建议。

1.1 什么是 PL/SQL

Oracle 的 PL/SQL 语言有一些决定性的特征，如下所示。

它是高度结构化、可读和易懂的语言

如果我们刚开始学习编程，PL/SQL 可以作为一个绝好的开端。我们会发现，PL/SQL 语言易于学习，它有丰富的关键字和结构来清晰地表达代码的内容。而如果我们已有其他编程语言的使用经验，则会很容易熟悉 PL/SQL 的语法。

它是标准的和可移植的 *Oracle* 开发语言

如果我们编写了一些 PL/SQL 过程或函数，并运行在自己的笔记本电脑上的 Oracle 数据库中，那么我们就可以不做修改（当然，数据库版本要兼容）地把这些过程或函数移植到我们所在公司的数据库中并运行它们。“一次编程，各地运行”是在 Java 出现之前长期流传的 PL/SQL 的口头禅。对于 PL/SQL 来说，“各地”意味着“有 Oracle 数据库的各个地方”。

它是内嵌式语言

PL/SQL 设计的初衷不是作为一种独立的语言，而是在宿主环境中被调用使用。因此，我们可以在数据库中运行 PL/SQL 程序（当然，通过 SQL*Plus 接口），或者，我们可以在 Oracle Developer form 或 report（这种方式称为客户端 PL/SQL）中定义和执行 PL/SQL 程序。但我们不能创建一个自己直接运行的 PL/SQL 可执行程序。

它是高性能、高度集成的数据库语言

近年来，当我们想针对 Oracle 数据库编写软件时，我们有多种选择：我们可以使用 Java 和 JDBC；我们可以用 Visual Basic 和 ODBC；我们还可以用 Delphi、C++等。然而，我们会发现在处理 Oracle 数据库时，PL/SQL 比其他语言更易写出高效率的代码。尤其是，Oracle 提供了针对 PL/SQL 的特定的增强，如 FORALL 声明，可以把数据库性能提高一个数量级甚至更多。

1.2 PL/SQL 的起源

Oracle 一度引领着声明式、非过程性数据库和应用设计的发展，Oracle 数据库服务器技术是世界上最先进、最强大、最稳定的关系数据库。它的应用开发工具，如 Oracle Forms，通过“在屏幕上作画”的方式，以及它提供的大量默认功能，使得开发者避免了繁重的定制开发工作量，由此极大地提高了开发的效率。

1.2.1 早期的 PL/SQL

在 Oracle 早期，SQL 的声明方式结合其突破性的关系型技术，足以满足开发者的需要。但随着行业的成熟和人们期望值的提高，人们对 SQL 有了更高的要求。开发者需要深入到产品的内层，在它们的表单和数据库脚本中构建复杂的公式和异常处理以及规则。

在 1988 年，Oracle 公司发布了 Oracle 版本 6，在关系型数据库技术中迈进了一大步。此版本中有一个叫作“过程性选项”或“PL/SQL”的关键组件，几乎就在同时，Oracle 发布了令人期待已久的升级版本 SQL*Forms 2.3（Oracle Forms 或 Forms Developer 最初的名字）。在工具方面，SQL*Forms v3 第一次纳入了 PL/SQL 引擎，使得开发者能以自然、简单的方式开发自己的过程性逻辑代码。

PL/SQL 最早版本的功能非常有限：在服务器端，我们只能使用 PL/SQL 来构建“批处理”脚本以及 SQL 语句，而不能构建模块化的应用或存储业务规则；在客户端，SQL*Formsv 3.0 倒是允许我们创建过程和函数，但其对函数的支持并没有对应文档，因此在之后的几年间都没有大量的开发人员在使用它。而且，此版本的 PL/SQL 不支持数组，不能与操作系统进行交互（输入和输出），它还是一个远远不够成熟的编程语言。

尽管存在局限性，但 PL/SQL 在开发人员社区受到了友好的甚至是热烈的欢迎，他们对在 SQL*Forms 里实现简单 IF 语句编码的渴望是强烈的。批处理执行多条 SQL 语句的需求压倒一切。

当时，少数开发者意识到，在 PL/SQL 背后的原始动机和主要期望，已经不再止于在像 SQL*Forms 这样的产品里，能够对控制进行编程的水平。在 Oracle 数据库和工具的生命周期早期，Oracle 公司就意识到，在它们的架构中，存在两个关键弱点：缺乏可移植性和执行权限管理。

1.2.2 提高应用的可移植性

对作者这些了解 Oracle 公司市场及技术战略的人来说，对应用可移植性存在顾虑显得有点奇怪，从 20 世纪 70 年代初开始，可移植性就是 Oracle 解决方案的标志之一。随着 PL/SQL 的出现，建筑在 C 语言之上的数据库就运行在许多不同的操作系统和硬件平台上。SQL*Plus 和 SQL*Forms 可以轻松适应不同配置的终端。但尽管有着这些适应性，仍然有一些应用程序需要 Oracle 提供

更复杂的和更细粒度的控制，像它们的宿主语言如 COBOL、C 和 FORTRAN 所做的那样。一旦开发者不采用端口中立的 Oracle 工具，就会导致应用程序不再可移植。

PL/SQL 语言曾经（现在也是）的意图是，在完全独立于操作系统的编程工具里，扩大对应用需求的处理能力。现在，Java 和其他编程语言也提供类似的可移植性。当然，PL/SQL 是这个领域的先锋，它继续支持着开发者们的高可移植性应用代码。

1.2.3 提高执行权限控制和交易完整性

比可移植性更重大的缺陷是执行权限控制。数据库和 SQL 语言使得我们对任意特定表中数据的访问和变更有着完全的控制，例如，用 GRANT 命令，我们可以限定只有特定的角色和用户可以对一个给定表进行 UPDATE 操作。而另一方面，这个 GRANT 命令，却没法保证让用户按照正确的顺序对一个或多个表进行变更，而通常大多数商业规则都对交易顺序有所要求。

PL/SQL 语言在逻辑交易处理上，能够提供严格控制和管理。PL/SQL 实现其控制和管理的方法之一，就是实施了执行权限控制。我们不再对一个角色或用户授权以允许其对表进行更改，取而代之的，是只授权其执行一个过程，由这个过程控制并提供对其内部隐含的数据结构的访问。过程的所有者是数据库中的另一个 schema（程序的“定义者”），这个 schema 被真正赋予了过程执行时所涉及的表的修改权限。由此，过程成为了交易的“守门员”。一个程序要执行某项交易，唯一的方法是调用这个过程，通过这个方法，整个应用交易的完整性得到了保证。

从 Oracle 数据库 8i 开始，Oracle 通过 AUTHID 子句，给 PL/SQL 执行权限控制模块增加了相当的灵活性。有了 AUTHID，我们不但可以在前面描述的模式下继续运行我们的程序，我们也可以选择使用 AUTHID CURRENT_USER（调用者的权限），在这种情况下，程序是运行在调用者（当前）schema 的权限之下的。这里我们只是将调用者权限作为一个示例，来展示 PL/SQL 随着时间的推移，如何变得成熟以及更灵活。

1.2.4 低调开始，持续改进

SQL 虽然强大，但仍然无法提供开发一个完整应用程序所需的灵活性和强大功能。而 Oracle 的 PL/SQL 语言，能够确保我们在独立于操作系统的 Oracle 环境中，开发出满足用户需求的高效率应用程序。

PL/SQL 是低调开始的，在 PL/SQL 1.0 时，这样的情形屡见不鲜：开发者告诉他们的经理“PL/SQL 做不了这个”。现在，这句话已经从事实变成了托词。如果我们曾经接到某个需求并发现“PL/SQL 做不了”，请不要立刻去向老板汇报，而是先对 PL/SQL 进行深度挖掘，或搜寻 Oracle 提供的 PL/SQL 程序包。时至今日，PL/SQL 已经基本能够实现任何我们想要的功能。

多年来，Oracle 公司展示了它对 PL/SQL，其编程语言旗舰的承诺。随着每个新数据库版本的发布，Oracle 也对 PL/SQL 语言本身进行了稳步的、重大的改进。它自带（内置）了大量程序包，用不同的方法和从各个角度，对 PL/SQL 进行了功能扩充。它引入了面向对象的能力，实施了一系列类似数组的数据结构，增强了编辑器，可以对代码进行优化和对代码质量、性能进行告警，在总体上对语言的深度与广度做了提升。

本章的下一节将展示一些 PL/SQL 的示例，使我们了解并熟悉基本的 PL/SQL 程序设计。

1.3 这就是 PL/SQL

如果你是个编程新手，或完全未接触过 PL/SQL（或 SQL），那么学习 PL/SQL 可能看起来令人生畏。如果是这样，请不要担心，我相信你会发现它比想象的要简单。作者的乐观依据包括以下两点。

- 总的来说，计算机语言并不难学，至少与人类的第二、第三外语学习相比并不难学。个中原因仅仅是因为计算机还不够“聪明”（它们的“思考”——操作的执行——非常迅速，但完全没有创造性），我们必须依赖一种非常严格的语法来告诉计算机我们需要它干什么。这导致所使用的语言也是刚性的（没有例外!），因此，我们容易学习上手。
- 与其他编程语言相比，PL/SQL 本身就比较简单。它在程序的不同部分，使用高度结构化的“程序块”，而且程序块都由显式、自说明的关键字进行定义。

下面，让我们看一些示例，从结构和功能两个方面展示 PL/SQL 的关键要素。

1.3.1 与 SQL 的集成

PL/SQL 的一个最重要的表现就是其与 SQL 的紧密集成。我们不必依赖其他中间软件作为“胶水”去运行 PL/SQL 中的 SQL 语句，如 ODBC（开放数据库连接）或 JDBC（Java 数据库连接）。相反，我们只需在代码中插入 UPDATE 或 SELECT 语句，如下所示：

```
1  DECLARE
2     l_book_count INTEGER;
3
4  BEGIN
5     SELECT COUNT(*)
6       INTO l_book_count
7       FROM books
8      WHERE author LIKE '%FEUERSTEIN, STEVEN%';
9
10    DBMS_OUTPUT.PUT_LINE (
11       'Steven has written (or co-written) ' ||
12        l_book_count ||
13        ' books.');
14
15    -- Oh, and I changed my name, so...
16    UPDATE books
17       SET author = REPLACE (author, 'STEVEN', 'STEPHEN')
18     WHERE author LIKE '%FEUERSTEIN, STEVEN%';
19 END;
```

让我们仔细看看下面表中对代码的分析。

行　号	说　明
1～3	这是所谓“匿名”PL/SQL 块的声明部分，作者（即“我”）在其中声明了一个整型变量，以存储我编写或合编的书的数量（第 3 章将更详细地讲解 PL/SQL 块结构）
4	BEGIN 关键字，表示执行部分的开始——当我把“块”传递给 SQL*Plus 时

续表

行　号	说　明
5～8	我运行一个查询，得到我编写或合编的书的数量。第 6 行最有意思：INTO 子句严格地说并不是 SQL 语句的一部分，而是作为数据库到本地 PL/SQL 变量的桥梁
10～13	我利用 DBMS_OUTPUT.PUT_LINE 内置过程（Oracle 提供的 DBMS_OUTPUT 软件包中的一个过程）来显示书的数量
15	一个单行注释行，解释 UPDATE 的目的
16～18	我决定把我名字的拼写改成“Stephen”，所以我对书本表进行了一个修改。我利用内置的 REPLACE 函数，把所有“STEVEN”替换成了“STEPHEN”

1.3.2 控制和条件逻辑

PL/SQL 提供了全方位的语句，使得我们可以严格控制程序中哪些行会被执行。这些语句如下所示。

***IF* 和 *CASE* 语句**

它们进行逻辑条件判断，例如“如果一本书的页数多于 1000，则……”

一个完整的循环或迭代控制

它们包括 FOR 循环、WHILE 循环，以及简单循环。

***GOTO* 语句**

是的，PL/SQL 甚至还提供了一个 GOTO 语句，使得我们可以无条件地将执行跳转到程序的另一个部分。然而，这并不意味着，我们当真应该去“使用”这个功能。

下面的存储过程（一个可以根据名字被调用，可重复使用的代码块），演示了前面提到的一些特性：

```
 1  PROCEDURE pay_out_balance (
 2     account_id_in IN accounts.id%TYPE)
 3  IS
 4     l_balance_remaining NUMBER;
 5  BEGIN
 6     LOOP
 7        l_balance_remaining := account_balance (account_id_in);
 8
 9        IF l_balance_remaining < 1000
10        THEN
11           EXIT;
12        ELSE
13           apply_balance (account_id_in, l_balance_remaining);
14        END IF;
15     END LOOP;
16  END pay_out_balance;
```

在下面的表格中，我们对上述代码做了详细分析。

行　号	说　明
1～2	为过程的头部，指明过程的名称（其意义是用账户余额来支付账单）。第 2 行是过程的参数列表，在本例中，它是一个输入值，表示账户的识别号
3～4	过程的声明段。请注意，我们没有像之前的例子那样使用 DECLARE 这个关键字，而是使用了 IS（或 AS）作为关键字来分开头部和声明
6～15	一个简单循环的示例，这个循环依赖 EXIT 语句（第 11 行）来终止循环；而 FOR 循环和 WHILE 循环则要分别指明终止条件
7	这里调用了一个叫作 account_balance 的函数，来获取对应账号的余额。这本身又是一个示例，演示如何在一个可复用的程序块里调用另一个可复用程序块；第 13 行演示了如何在本过程里调用另一个过程
9～14	这是一个 IF 语句，其含义是：如果账户余额少于 1000 美元，则不再用此账户来支付账单；否则，下次账单支付时，还用此账户

1.3.3 出错处理

无论是发现错误还是处理错误，PL/SQL 语言都有一个强大的机制。在下面的过程中，我们根据 ID 获得一个账户的名称和余额，随后判断余额是否过少，如果过少，就要显式地引发异常，终止过程的继续执行：

```
 1  PROCEDURE check_account (
 2     account_id_in IN accounts.id%TYPE)
 3  IS
 4     l_balance_remaining        NUMBER;
 5     l_balance_below_minimum    EXCEPTION;
 6     l_account_name             accounts.name%TYPE;
 7  BEGIN
 8     SELECT name
 9       INTO l_account_name
10       FROM accounts
11      WHERE id = account_id_in;
12
13     l_balance_remaining := account_balance (account_id_in);
14
15     DBMS_OUTPUT.PUT_LINE (
16        'Balance for ' || l_account_name ||
17         ' = ' || l_balance_remaining);
18
19     IF l_balance_remaining < 1000
20     THEN
21        RAISE l_balance_below_minimum;
22     END IF;
23
24  EXCEPTION
25     WHEN NO_DATA_FOUND
26     THEN
27        -- No account found for this ID
28        log_error (...);
29        RAISE;
30     WHEN l_balance_below_minimum
31     THEN
```