

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 Programming

[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

Copyright © 2014 by O'Reilly Media, Inc.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and Posts & Telecom Press, 2016.
Authorized translation of the English edition, 2014 O'Reilly Media, Inc., the owner of all rights to publish
and sell the same.

All rights reserved, including the rights of reproduction in whole or in part in any form.

本书中文简体字版由 **O'Reilly Media, Inc.** 授权人民邮电出版社出版。未经出版者书面许可, 对本书
的任何部分不得以任何方式复制或抄袭。

版权所有, 侵权必究。

-
- ◆ 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: 8001—10500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1章 PL/SQL 介绍.....3	2.3.1 启动 SQL*Plus 21
1.1 什么是 PL/SQL3	2.3.2 运行 SQL 语句 22
1.2 PL/SQL 的起源4	2.3.3 运行 PL/SQL 程序 22
1.2.1 早期的 PL/SQL4	2.3.4 运行一个脚本 24
1.2.2 提高应用的可移植性4	2.3.5 什么是“当前目录” 24
1.2.3 提高执行权限控制和交易完整性5	2.3.6 其他 SQL*Plus 任务 25
1.2.4 低调开始, 持续改进5	2.3.7 SQL*Plus 中的异常处理 29
1.3 这就是 PL/SQL6	2.3.8 为什么 SQL*Plus 让我们又爱又恨 30
1.3.1 与 SQL 的集成6	2.4 执行基本的 PL/SQL 任务 30
1.3.2 控制和条件逻辑7	2.4.1 创建存储程序 31
1.3.3 出错处理8	2.4.2 执行存储的程序 33
1.4 关于 PL/SQL 版本9	2.4.3 显示存储程序 34
1.4.1 Oracle 数据库 12c 中 PL/SQL 的新特性 10	2.4.4 存储程序的授权和别名 35
1.5 可供 PL/SQL 开发者使用的资源 12	2.4.5 删除一个存储程序 36
1.5.1 O'Reilly 的 PL/SQL 系列图书 13	2.4.6 隐藏存储程序的源代码 36
1.5.2 网络上的 PL/SQL 资源 14	2.5 编辑 PL/SQL 的环境 37
1.6 一些建议 15	2.6 从其他语言中调用 PL/SQL 37
1.6.1 别急, 慢慢来 15	2.6.1 C 语言, 使用 Oracle 预编辑器 (Pro*C) 38
1.6.2 不要畏惧寻求帮助 16	2.6.2 Java: 使用 JDBC 39
1.6.3 采用有创造性的甚至激进的方法 17	2.6.3 Perl: 使用 Perl DBI 和 DBD::Oracle 40
第2章 创建和运行 PL/SQL 代码 18	2.6.4 PHP: 使用 Oracle 扩展 41
2.1 在数据库中导航 18	2.6.5 PL/SQL Server Pages 42
2.2 创建和编辑源代码 19	2.6.6 其他 43
2.3 SQL*Plus 19	第3章 语言基础 44
	3.1 PL/SQL 块结构 44
	3.1.1 匿名块 46
	3.1.2 命名块 47

3.1.3 嵌套块.....	47	3.4.2 在一个直接量字符串中嵌入 单引号.....	61
3.1.4 作用范围.....	48	3.4.3 数字直接量.....	61
3.1.5 规范 SQL 语句中对变量和 列的引用.....	49	3.4.4 布尔直接量.....	62
3.1.6 可见性.....	51	3.5 分号分隔符.....	62
3.2 PL/SQL 字符集.....	54	3.6 注释.....	63
3.3 标识符.....	56	3.6.1 单行注释语法.....	63
3.3.1 保留字.....	57	3.6.2 多行注释语法.....	63
3.3.2 空白和关键字.....	58	3.7 PRAGMA 关键字.....	64
3.4 直接量.....	59	3.8 标签.....	65
3.4.1 NULL.....	60		

第 2 部分 PL/SQL 程序结构

第 4 章 条件与顺序控制.....69

4.1 IF 语句.....	69
4.1.1 IF-THEN 组合.....	69
4.1.2 IF-THEN-ELSE 的组合.....	71
4.1.3 IF-THEN-ELSIF 组合.....	73
4.1.4 避免 IF 语法陷阱.....	73
4.1.5 嵌套的 IF 语句.....	75
4.1.6 短路估算.....	75
4.2 CASE 语句和表达式.....	77
4.2.1 简单的 CASE 语句.....	77
4.2.2 搜索 CASE 语句.....	79
4.2.3 嵌套 CASE 语句.....	81
4.2.4 CASE 表达式.....	81
4.3 GOTO 语句.....	83
4.4 NULL 语句.....	84
4.4.1 提高程序的可读性.....	84
4.4.2 在标签后使用 NULL.....	84

第 5 章 循环迭代处理.....86

5.1 循环的基础知识.....	86
5.1.1 不同循环的示例.....	86
5.1.2 PL/SQL 循环的结构.....	88
5.2 简单循环.....	89
5.2.1 终止简单循环: EXIT	

和 EXIT WHEN.....	89
5.2.2 模仿 REPEAT UNTIL 循环.....	90
5.2.3 故意的无限循环.....	91
5.3 WHILE 循环.....	92
5.4 数值型 FOR 循环.....	93
5.4.1 数值型 FOR 循环的规则.....	94
5.4.2 数值型 FOR 循环的示例.....	94
5.4.3 处理特殊增量.....	95
5.5 游标 FOR 循环.....	96
5.5.1 游标 FOR 循环的示例.....	97
5.6 循环标签.....	98
5.7 CONTINUE 语句.....	99
5.8 迭代处理技巧.....	102
5.8.1 为循环索引使用可理解的 名称.....	102
5.8.2 以正确的方式说再见.....	102
5.8.3 获取 FOR 循环执行的信息.....	103
5.8.4 循环 SQL 语句.....	104

第 6 章 异常处理.....106

6.1 异常处理概念和术语.....	106
6.2 定义异常.....	108
6.2.1 声明命名异常.....	108
6.2.2 关联异常名称与错误代码.....	109
6.2.3 命名的系统异常.....	112

6.2.4 异常作用范围	114	代码	127
6.3 引发异常	115	6.5 构建有效的错误管理架构	129
6.3.1 RAISE 语句	115	6.5.1 确定我们的异常管理策略	129
6.3.2 使用 RAISE_APPLICATION_ ERROR	116	6.5.2 对不同类型异常进行标准化 处理	130
6.4 处理异常	117	6.5.3 程序特定错误代码的组织 使用	133
6.4.1 内置错误函数	118	6.5.4 使用标准的错误管理程序	133
6.4.2 单一处理句柄中结合多个 异常	122	6.5.5 使用自己的异常“对象”	135
6.4.3 未处理异常	123	6.5.6 创建常见错误处理的标准 模板	137
6.4.4 未处理异常的传播	123	6.6 充分利用 PL/SQL 错误 管理	138
6.4.5 继续过去的异常	125		
6.4.6 编写 WHEN OTHERS 处理			

第3部分 PL/SQL 程序数据

第7章 使用程序数据

7.1 程序数据的命名	141
7.2 PL/SQL 数据类型概述	143
7.2.1 字符数据	143
7.2.2 数字	144
7.2.3 日期、时间戳和时间间隔	145
7.2.4 布尔类型	145
7.2.5 二进制数据类型	146
7.2.6 ROWID	146
7.2.7 REF CURSOR	146
7.2.8 Internet 数据类型	147
7.2.9 “Any”数据类型	147
7.2.10 用户自定义数据类型	147
7.3 程序数据的声明	147
7.3.1 声明一个变量	148
7.3.2 声明常量	148
7.3.3 NOT NULL 语句	149
7.3.4 锚定声明	149
7.3.5 游标和表的锚	151
7.3.6 使用锚定声明的益处	152
7.3.7 NOT NULL 数据类型的锚	153
7.4 程序员定义的子类型	153

7.5 数据类型转换	154
7.5.1 隐式类型转换	155
7.5.2 显式类型转换	156

第8章 字符串

8.1 字符串类型	162
8.1.1 VARCHAR2 数据类型	163
8.1.2 CHAR 数据类型	164
8.1.3 String 子类型	164
8.2 使用字符串	165
8.2.1 指定字符串常量	165
8.2.2 不可打印字符	167
8.2.3 拼接字符串	168
8.2.4 处理大小写	169
8.2.5 传统的检索、提取和替换	172
8.2.6 填充	174
8.2.7 剪裁	176
8.2.8 正则表达式的检索、提取和 替换	177
8.2.9 使用空字符串	187
8.2.10 混用 CHAR 和 VARCHAR2	188
8.3 字符串函数快速参考	190

第 9 章 数字	199
9.1 数值型数字类型	199
9.1.1 NUMBER 类型	200
9.1.2 PLS_INTEGER 类型	204
9.1.3 BINARY_INTEGER 类型	205
9.1.4 SIMPLE_INTEGER 类型	205
9.1.5 BINARY_FLOAT 和 BINARY_DOUBLE 类型	207
9.1.6 SIMPLE_FLOAT 和 SIMPLE_DOUBLE 类型	212
9.1.7 数字子类型	212
9.2 数字转换	213
9.2.1 TO_NUMBER 函数	213
9.2.2 TO_CHAR 函数	216
9.2.3 CAST 函数	221
9.2.4 隐式转换	222
9.3 数字运算符	224
9.4 数字函数	224
9.4.1 四舍五入和截断函数	224
9.4.2 三角函数	225
9.4.3 数字函数的快速参考	225
第 10 章 日期和时间戳	230
10.1 Datetime 数据类型	230
10.1.1 声明日期时间变量	233
10.1.2 选择日期时间数据类型	233
10.2 获取当前日期和时间	234
10.3 INTERVAL 数据类型	236
10.3.1 声明 INTERVAL 变量	237
10.3.2 什么时候使用 INTERVAL	238
10.4 日期时间转换	240
10.4.1 从字符串到日期时间	240
10.4.2 从日期时间到字符串	242
10.4.3 使用时区	245
10.4.4 精确匹配需要格式掩码	247
10.4.5 让精确匹配更轻松	248
10.4.6 解释滑动窗口中两位数字的 年份	248

10.4.7 把时区转换成字符串	249
10.4.8 用填充模式把输出补齐	250
10.5 日期和时间戳直接量	251
10.6 时间间隔的转换	252
10.6.1 从数字到时间间隔的转换	252
10.6.2 把字符串转换成间隔	253
10.6.3 时间间隔的格式化显示	254
10.7 时间间隔直接量	254
10.8 CAST 和 EXTRACT	256
10.8.1 CAST 函数	256
10.8.2 EXTRACT 函数	258
10.9 日期时间的算法	258
10.9.1 时间间隔和日期时间的 算法	259
10.9.2 DATE 数据类型的日期 算法	260
10.9.3 计算两个日期时间之间的 时间间隔	260
10.9.4 DATE 和 TIMESTAMP 混合 计算	262
10.9.5 时间间隔的加减运算	263
10.9.6 时间间隔的乘除运算	264
10.9.7 使用不受限制的时间间隔 类型	264
10.10 日期/时间函数的快速 参考	266

第 11 章 记录类型	269
11.1 PL/SQL 中的记录	269
11.1.1 使用记录的好处	270
11.1.2 声明记录	271
11.1.3 程序员自定义的记录类型	273
11.1.4 使用记录类型	275
11.1.5 记录的比较	281
11.1.6 触发器伪记录	282

第 12 章 集合	284
12.1 集合概述	285
12.1.1 集合概念和术语	285

12.1.2	集合类型	287	12.4.4	嵌套表中的去重	346
12.1.3	集合示例	288	12.5	schema 级别集合的维护	347
12.1.4	使用集合的场合	291	12.5.1	必需的权限	347
12.1.5	选择一个集合类型	296	12.5.2	集合和数据字典	348
12.2	集合方法 (内置)	297	第 13 章	其他数据类型	349
12.2.1	COUNT 方法	298	13.1	BOOLEAN 类型	349
12.2.2	DELETE 方法	299	13.2	RAW 数据类型	350
12.2.3	EXISTS 方法	300	13.3	UROWID 和 ROWID 数据类型	351
12.2.4	EXTEND 方法	300	13.3.1	获取 ROWID	352
12.2.5	FIRST 和 LAST 方法	301	13.3.2	使用 ROWID	352
12.2.6	LIMIT 方法	302	13.4	LOB 数据类型	353
12.2.7	PRIOR 和 NEXT 方法	303	13.5	使用 LOB	354
12.2.8	TRIM 方法	304	13.5.1	理解 LOB 定位符	356
12.3	使用集合	305	13.5.2	LOB 的空和 NULL	357
12.3.1	声明集合类型	306	13.5.3	向 LOB 中写入数据	359
12.3.2	集合变量的声明和初始化	310	13.5.4	读取 LOB 数据	361
12.3.3	用数据填充集合	313	13.5.5	BFILE 的不同之处	363
12.3.4	访问集合内的数据	318	13.5.6	SecureFiles 和 BasicFiles	367
12.3.5	使用字符串索引的集合	319	13.5.7	临时 LOB	369
12.3.6	复杂数据类型的集合	324	13.5.8	原生的 LOB 操作	372
12.3.7	多级集合	327	13.5.9	LOB 转换函数	376
12.3.8	在 SQL 中使用集合	335	13.6	预定义的对象类型	376
12.4	嵌套表的多重集合操作	342	13.6.1	XMLType 类型	376
12.4.1	测试嵌套表是否相等及成员 归属	343	13.6.2	URI 类型	379
12.4.2	检查元素是否是嵌套表的 成员	344	13.6.3	Any 类型	381
12.4.3	执行高级别集合操作	345			

第 4 部分 PL/SQL 中的 SQL

第 14 章	DML 和事务管理	387	14.2	事务管理	397
14.1	PL/SQL 中的 DML	388	14.2.1	COMMIT 语句	397
14.1.1	DML 简介	388	14.2.2	ROLLBACK 语句	398
14.1.2	DML 操作符的游标属性	391	14.2.3	SAVEPOINT 语句	399
14.1.3	从 DML 语句返回信息	392	14.2.4	SET TRANSACTION 语句	399
14.1.4	DML 和异常处理	393	14.2.5	LOCK TABLE 语句	400
14.1.5	DML 和记录	394	14.3	自治事务	400
			14.3.1	定义自治事务	401

14.3.2	自治事务的规则和限制	402	15.5.6	从游标变量中提取数据	440
14.3.3	事务的可见性	403	15.5.7	游标变量的使用规则	442
14.3.4	何时使用自治事务	403	15.5.8	将游标变量作为参数传递	445
14.3.5	创建自治日志记录机制	404	15.5.9	游标变量的约束限制	447
第 15 章	数据提取	407	15.6	游标表达式	447
15.1	游标基础	408	15.6.1	使用游标表达式	448
15.1.1	一些数据提取术语	408	15.6.2	游标表达式的约束限制	450
15.1.2	典型的查询操作	410	第 16 章	动态 SQL 和动态 PL/SQL	451
15.1.3	游标属性介绍	411	16.1	NDS 语句	452
15.1.4	在游标中引用 PL/SQL 变量	413	16.1.1	EXECUTE IMMEDIATE 语句	452
15.1.5	显式与隐式游标之间的选择	414	16.1.2	OPEN FOR 语句	455
15.2	使用隐式游标	414	16.1.3	4 种动态 SQL 方法	460
15.2.1	隐式游标示例	415	16.2	绑定变量	462
15.2.2	隐式游标的异常处理	416	16.2.1	参数模式	463
15.2.3	隐式 SQL 游标的属性	418	16.2.2	重复的占位符	465
15.3	使用显式游标	419	16.2.3	传递 NULL 值	465
15.3.1	声明显式游标	420	16.3	使用对象和集合	466
15.3.2	打开显式游标	423	16.4	动态 PL/SQL	468
15.3.3	从显式游标获取	424	16.4.1	建立动态 PL/SQL 块	469
15.3.4	显式游标中的列别名	425	16.4.2	用动态块替换重复代码	470
15.3.5	关闭显式游标	426	16.5	NDS 建议	471
15.3.6	显式游标属性	427	16.5.1	对共享程序使用调用者权限	471
15.3.7	游标参数	429	16.5.2	预测并处理动态错误	472
15.4	SELECT...FOR UPDATE	432	16.5.3	使用绑定而非拼接	474
15.4.1	COMMIT 释放锁定	433	16.5.4	减少代码注入的危险	475
15.4.2	WHERE CURRENT OF 子句	434	16.6	何时使用 DBMS_SQL	478
15.5	游标变量和 REF CURSOR	435	16.6.1	获得查询列信息	478
15.5.1	为什么使用游标变量	436	16.6.2	实现第四种方法的动态 SQL 需求	479
15.5.2	与静态游标的相似之处	437	16.6.3	最小化动态游标解析	485
15.5.3	声明 REF CURSOR 类型	437	16.6.4	Oracle 数据库 11g 新动态 SQL 特性	486
15.5.4	声明游标变量	438	16.6.5	DBMS_SQL 增强安全	490
15.5.5	打开游标变量	439			

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？

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 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 的紧密切集成。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

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，来获取对应账户的余额。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

```