

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 Program

[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/SOL 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 (C I P) 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: 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 17 章 过程、函数和参数497

17.1 代码模块化497

17.2 过程499

17.2.1 调用一个过程501

17.2.2 过程头部501

17.2.3 过程体501

17.2.4 END 标签502

17.2.5 RETURN 语句502

17.3 函数502

17.3.1 函数的结构503

17.3.2 返回的数据类型504

17.3.3 END 标签506

17.3.4 调用函数506

17.3.5 不带参数的函数507

17.3.6 函数头508

17.3.7 函数体508

17.3.8 RETURN 语句509

17.4 参数510

17.4.1 定义参数511

17.4.2 实参和形参511

17.4.3 参数模式512

17.4.4 在 PL/SQL 中显式地关联 实参和形参515

17.4.5 NOCOPY 参数模式限定符519

17.4.6 缺省值519

17.5 局部或者嵌套模块520

17.5.1 使用局部模块的益处521

17.5.2 局部模块的作用范围523

17.5.3 用局部模块使得代码更 整洁524

17.6 模块的重载524

17.6.1 重载的益处525

17.6.2 重载的限制528

17.6.3 数字类型的重载528

17.7 前置声明529

17.8 高级主题530

17.8.1 在 SQL 内部调用我们的 函数530

17.8.2 表函数536

17.8.3 确定性函数545

17.8.4 隐式游标结果 (Oracle 数据库 12c)546

17.9 将模块化进行到底547

第 18 章 程序包548

18.1 为什么使用程序包548

18.1.1 演示程序包的能力549

18.1.2 与程序包相关的一些概念552

18.1.3 图示私有性553

18.2 构建程序包的规则554

18.2.1 程序包说明554

18.2.2 包体555

18.2.3 包的初始化557

18.3 包元素的调用规则561

18.4 使用包数据562

18.4.1 在一个 Oracle 会话内全局 可见562

18.4.2 全局公有数据563

18.4.3 包游标563

18.4.4 包的串行化568

18.5 何时使用包570

18.5.1 封装对数据的访问570

18.5.2 避免直接量的硬编码573

18.5.3 提高内置特性的可用性575

18.5.4 把逻辑上相关的功能组织 在一起576

18.5.5 缓存静态的会话数据576

18.6 包和对象类型.....	577	19.5.3 ORA_SPACE_ERROR_INFO 函数.....	624
第 19 章 触发器	578	19.5.4 DBMS_RESUMABLE 包.....	625
19.1 DML 触发器.....	579	19.5.5 捕获多个时间.....	626
19.1.1 DML 触发器的概念.....	580	19.5.6 是否该处理?.....	627
19.1.2 创建 DML 触发器.....	581	19.6 维护触发器	628
19.1.3 DML 触发器的例子: 严禁 作弊!.....	586	19.6.1 禁用、启用以及删除 触发器.....	628
19.1.4 同一类型的多个触发器.....	591	19.6.2 创建一个禁用的触发器.....	628
19.1.5 如何对触发顺序排序.....	592	19.6.3 查看触发器.....	629
19.1.6 突变表的错误.....	594	19.6.4 检查触发器的有效性.....	630
19.1.7 复合触发器: 聚在一处.....	595	第 20 章 管理 PL/SQL 代码	631
19.2 DDL 触发器.....	598	20.1 管理数据库内的代码.....	632
19.2.1 创建 DDL 触发器.....	598	20.1.1 数据字典视图概述.....	632
19.2.2 可用事件.....	600	20.1.2 显示存储对象的信息.....	634
19.2.3 可用属性.....	601	20.1.3 源代码的显示和搜索.....	635
19.2.4 使用事件和属性.....	602	20.1.4 根据程序的大小确定 Pinning 需求.....	637
19.2.5 删除不可删除的.....	606	20.1.5 获得存储代码的属性.....	637
19.2.6 INSTEAD OF CREATE 触发器.....	606	20.1.6 通过视图分析和更改触发器 状态.....	638
19.3 数据库事件触发器.....	607	20.1.7 分析参数信息.....	639
19.3.1 创建数据库事件触发器.....	608	20.1.8 分析标识符的使用 (Oracle 数据库 11g 的 PL/Scope).....	640
19.3.2 STARTUP 触发器.....	609	20.2 管理依赖关系及重编译 代码	643
19.3.3 SHUTDOWN 触发器.....	610	20.2.1 通过数据字典视图分析依赖 关系.....	643
19.3.4 LOGON 触发器.....	610	20.2.2 细粒度依赖 (Oracle 数据库 11g).....	647
19.3.5 LOGOFF 触发器.....	610	20.2.3 远程依赖.....	648
19.3.6 SERVERERROR 触发器.....	611	20.2.4 Oracle 的远程调用模式的 限制.....	650
19.4 INSTEAD OF 触发器.....	615	20.2.5 重编译无效的程序单元.....	651
19.4.1 创建 INSTEAD OF 触发器.....	615	20.3 编译时刻警告	655
19.4.2 INSTEAD OF INSERT 触发器.....	616	20.3.1 一个快速示例.....	655
19.4.3 INSTEAD OF UPDATE 触发器.....	618	20.3.2 开启编译时刻告警.....	656
19.4.4 INSTEAD OF DELETE 触发器.....	619	20.3.3 一些有用的警告.....	657
19.4.5 填充表.....	619	20.4 测试 PL/SQL 程序	664
19.4.6 嵌套表的 INSTEAD OF 触发器.....	620	20.4.1 典型的、华而不实的测试 技术.....	665
19.5 AFTER SUSPEND 触发器.....	621		
19.5.1 建立 AFTER SUSPEND 触发器.....	622		
19.5.2 看看真实的触发器.....	623		

20.4.2	PL/SQL 代码测试的一般建议	668	数据库 11g)	718
20.4.3	PL/SQL 的自动测试选项	669	21.3.4	缓存总结
20.5	跟踪 PL/SQL 的执行	670	21.4	重复的 SQL 的语句批处理
20.5.1	DBMS_UTILITY.FORMAT_CALL_STACK	671	21.4.1	通过 BULK COLLECT 加速查询
20.5.2	UTL_CALL_STACK (Oracle 数据库 12c)	673	21.4.2	使用 FORALL 加速 DML
20.5.3	DBMS_APPLICATION_INFO	676	21.5	利用管道化的表函数提升性能
20.5.4	使用 opp_trace 进行跟踪	677	21.5.1	用基于管道化函数的加载方式替换基于行的插入
20.5.5	DBMS_TRACE 工具包	678	21.5.2	用管道函数调优 Merge 操作
20.6	PL/SQL 程序的调试	681	21.5.3	用并行管道函数进行异步数据导出
20.6.1	错误的调试方法	682	21.5.4	并行管道函数中的分区和流子句对性能的影响
20.6.2	调试技巧和策略	683	21.5.5	管道函数和基于成本的优化器
20.7	使用白名单来控制对程序单元的访问	687	21.5.6	用管道函数优化负载的数据加载
20.8	存储代码的保护	689	21.5.7	管道函数结束语
20.8.1	封装的约束和局限	690	21.6	专用的优化技术
20.8.2	使用封装程序	690	21.6.1	使用 NOCOPY 参数模式提示符
20.8.3	使用 DBMS_DDL 进行动态封装	690	21.6.2	使用正确的数据类型
20.8.4	封装代码的使用指导	691	21.6.3	SQL (12.1 及更高版本) 的函数性能优化
20.9	基于版本的重定义 (Oracle 数据库 11g R2 版本)	692	21.7	性能回顾
第 21 章 PL/SQL 的性能优化		695	第 22 章 I/O 操作和 PL/SQL	
21.1	辅助优化的工具	696	22.1	显示信息
21.1.1	内存使用分析	696	22.1.1	启用 DBMS_OUTPUT
21.1.2	发现 PL/SQL 代码中的瓶颈	697	22.1.2	向缓存中写入行
21.1.3	计算花费时间	701	22.1.3	从缓存中读取内容
21.1.4	选择最快的程序	703	22.2	文件的读写
21.1.5	避免无限循环	704	22.2.1	UTL_FILE_DIR 参数
21.1.6	性能相关的警告	706	22.2.2	使用 Oracle 目录
21.2	优化编译器	706	22.2.3	打开文件
21.2.1	优化器工作原理	707	22.2.4	文件已经打开了吗?
21.2.2	循环 Fetch 操作的运行时优化	710	22.2.5	关闭文件
21.3	数据缓存技术	710	22.2.6	读取文件
21.3.1	基于包的缓存	711		
21.3.2	确定性函数的缓存	716		
21.3.3	函数结果缓存 (Oracle			

22.2.7	向文件中写	792	22.4	使用基于 Web 的数据 (HTTP)	808
22.2.8	复制文件	795	22.4.1	“分片”获得一个 Web 页面	808
22.2.9	删除文件	795	22.4.2	把页面提取到一个 LOB 中	809
22.2.10	改名和移动文件	796	22.4.3	使用 HTTP 的用户名/密码验证	810
22.2.11	提取文件属性	797	22.4.4	获取一个 SSL 加密的 Web 页面 (使用 HTTPS)	811
22.3	发送邮件	798	22.4.5	通过 GET 或者 POST 向 Web 页面提交数据	812
22.3.1	Oracle 的前提条件	798	22.4.6	禁用 cookie 或者使 cookie 持久化	816
22.3.2	设置网络安全	799	22.4.7	从 FTP 服务器获取数据	816
22.3.3	发送一个短的 (小于 32767 字节) 的纯文本消息	799	22.4.8	使用代理服务器	817
22.3.4	在邮件地址中加上“界面友好的”的名字	801	22.5	PL/SQL 中可用的其他 I/O 类型	817
22.3.5	发送任意长度的纯文本消息	802	22.5.1	数据库管道、队列、告警	817
22.3.6	发送带有小附件 (小于 32767 字节) 的消息	803	22.5.2	TCP Socket	818
22.3.7	以附件形式发送一个小文件 (小于 32767 字节)	805	22.5.3	Oracle 的内置 Web 服务器	818
22.3.8	任意大小的附件	805			

第 6 部分 高级 PL/SQL 主题

第 23 章 应用系统安全与

PL/SQL

23.1	安全概述	821	23.3.2	一个简单的 RLS 示例	847
23.2	加密	822	23.3.3	静态与动态策略	850
23.2.1	密钥长度	823	23.3.4	使用列敏感的 RLS	854
23.2.2	算法	824	23.3.5	RLS 调试	857
23.2.3	填补和连接	825	23.4	应用程序上下文	861
23.2.4	DBMS_CRYPTO 包	825	23.4.1	使用应用程序上下文	862
23.2.5	数据加密	827	23.4.2	上下文的安全	863
23.2.6	LOB 的加密	830	23.4.3	把上下文用作 RLS 的谓词条件	863
23.2.7	安全文件	830	23.4.4	识别出非数据库的用户	867
23.2.8	数据解密	831	23.5	细粒度审计	868
23.2.9	生成密钥	832	23.5.1	为什么要学习 FGA	869
23.2.10	密钥的管理	833	23.5.2	一个简单的 FGA 示例	870
23.2.11	加密哈希	838	23.5.3	访问多少列	872
23.2.12	使用消息验证码	839	23.5.4	查看审计跟踪信息	873
23.2.13	使用透明数据加密 (TDE)	841	23.5.5	使用绑定变量	874
23.2.14	透明的表空间加密	843	23.5.6	使用句柄模块	875
23.3	行级安全	844	第 24 章	PL/SQL 架构	877
23.3.1	为什么要学习 RLS	846	24.1	DIANA	877
			24.2	Oracle 如何执行 PL/SQL	

代码	878	25.2.1 国家字符集的数据类型	929
24.2.1 一个示例	879	25.2.2 字符编码	929
24.2.2 编译器的限制	881	25.2.3 和全球化支持相关的参数	930
24.3 PL/SQL 的缺省包	882	25.2.4 Unicode 函数	931
24.4 执行权限模型	884	25.3 字符语义	938
24.4.1 定义者权限模型	885	25.4 字符串排序顺序	941
24.4.2 调用者权限模型	889	25.4.1 二进制排序	942
24.4.3 组合权限模型	891	25.4.2 单语言排序	943
24.4.4 给 PL/SQL 程序单元授予角色 (Oracle 数据库 12c)	892	25.4.3 多语言排序	945
24.4.5 “谁调用了我?” 函数 (Oracle 数据库 12c)	895	25.5 多语言信息检索	946
24.4.6 视图的 BEQUEATH CURRENT_ USER 子句 (Oracle 数据库 12c)	895	25.5.1 信息检索和 PL/SQL	948
24.4.7 调用者权限优点的限制 (Oracle 数据库 12c)	897	25.6 日期/时间	950
24.5 条件编译	898	25.6.1 时间戳数据类型	951
24.5.1 条件编译的示例	899	25.6.2 日期/时间格式	952
24.5.2 查询指令	900	25.7 货币转换	955
24.5.3 \$IF 指令	903	25.8 PL/SQL 的全球化开发 工具箱	957
24.5.4 \$ERROR 指令	904	25.8.1 UTL_I18N 工具包	957
24.5.5 将代码与包常量同步	905	25.8.2 UTL_LMS 异常处理包	960
24.5.6 用查询指令实现程序专有 设置	906	25.8.3 GDK 实现选项	961
24.5.7 使用预处理后的代码	907	第 26 章 PL/SQL 的面向对象 特性	963
24.6 PL/SQL 和数据库实例 内存	908	26.1 Oracle 对象特性的介绍	963
24.6.1 SGA、PGA 和 UGA	908	26.2 对象类型示例	965
24.6.2 游标、内存及其他	909	26.2.1 创建一个基类	966
24.6.3 减少内存使用的技巧	910	26.2.2 创建子类型	967
24.6.4 内存用光了怎么办	920	26.2.3 方法	968
24.7 原生式编译	922	26.2.4 在 Oracle 数据库 11g 及以后 版本中调用父类的方法	972
24.7.1 什么时候使用解释模式	922	26.2.5 保存、提取、使用持久化 对象	974
24.7.2 什么时候使用原生模式	922	26.2.6 演变和创建	981
24.7.3 原生编译和数据库版本	923	26.2.7 回到指针吗?	983
24.8 一些须知	923	26.2.8 泛化数据: ANY 类型	989
第 25 章 PL/SQL 的全球化和本地化	925	26.2.9 我们自己做	993
25.1 概述和术语	926	26.2.10 对象的比较	996
25.2 Unicode 入门	928	26.3 对象视图	1001
		26.3.1 一个关系型系统的示例	1002
		26.3.2 带有集合属性的对象视图	1003
		26.3.3 对象子视图	1006
		26.3.4 带有反关系的对象视图	1008

26.3.5	INSTEAD OF 触发器	1008
26.3.6	对象视图和对象表的区别	1010
26.4	维护对象类型和对象视图	1012
26.4.1	数据字典	1012
26.4.2	权限	1013
26.5	来自一个关系开发者的总结 思考 (C551, E1200)	1015
第 27 章	从 PL/SQL 中调用 Java	1017
27.1	Oracle 和 Java	1017
27.2	准备好在 Oracle 中使用 Java	1018
27.2.1	安装 Java	1019
27.2.2	创建和编译我们的 Java 代码	1019
27.2.3	设置 Java 开发和执行的 权限	1020
27.3	一个简单的演示	1022
27.3.1	查找 Java 功能	1023
27.3.2	创建一个自定义 Java 类	1023
27.3.3	编译和加载到 Oracle	1025
27.3.4	创建一个 PL/SQL 的 包装器	1026
27.3.5	从 PL/SQL 删除文件	1027
27.4	使用 loadjava	1028
27.5	使用 dropjava	1030
27.6	管理数据库中的 Java	1030
27.6.1	Oracle 中的 Java 命名空间	1030
27.6.2	检查加载的 Java 元素	1031
27.7	使用 DBMS_JAVA	1032
27.7.1	LONGNAME: 转换 Java 长名字	1032
27.7.2	GET_、SET_ 和 RESET_ COMPILER_OPTION: 得到和设置 (一些) 编译器选项	1033
27.7.3	SET_OUTPUT: 允许从 Java 中输出	1034
27.7.4	EXPORT_SOURCE、EXPORT_ RESOURCE 和 EXPORT_	

	CLASS: 导出模式对象	1034
27.8	在 PL/SQL 中发布与 使用 Java	1036
27.8.1	调用规范	1036
27.8.2	一些调用规范的规则	1037
27.8.3	映射数据类型	1038
27.8.4	在 SQL 中调用 Java 方法	1039
27.8.5	Java 的异常处理	1040
27.8.6	扩展文件 I/O 功能	1042
27.8.7	其他示例	1046
第 28 章	外部过程	1049
28.1	外部过程介绍	1050
28.1.1	示例: 调用一个系统命令	1050
28.1.2	外部过程的架构	1052
28.2	Oracle 网络配置	1053
28.2.1	定义监听配置	1053
28.2.2	配置的安全特性	1055
28.3	设置多线程模式	1056
28.4	创建一个 Oracle 库	1058
28.5	编写调用规范	1059
28.5.1	调用规范: 整体语法	1060
28.5.2	参数映射: 示例重温	1061
28.5.3	参数映射: 完整的内容	1063
28.5.4	更多的语法: 参数子句	1064
28.5.5	参数属性	1065
28.6	从调用的 C 程序中引发 一个异常	1068
28.7	非默认的代理	1071
28.8	维护外部过程	1073
28.8.1	删除库	1073
28.8.2	数据字典	1074
28.8.3	规则和警示	1074

附录 A	正则表达式元字符和函数 参数	1075
-------------	---------------------------	-------------

附录 B	数字格式模型	1080
-------------	---------------	-------------

附录 C	日期格式模型	1083
-------------	---------------	-------------

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。

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”是这样一個过程：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
(company_id_in IN company.company_id%TYPE, discount_in IN NUMBER)
IS
min_discount CONSTANT NUMBER:=.05;
max_discount CONSTANT NUMBER:=.25;
invalid_discount EXCEPTION;
BEGIN
IF discount_in BETWEEN min_discount AND max_discount
THEN
UPDATE item
SET item_amount:=item_amount*(1-discount_in);
WHERE EXISTS (SELECT 'x' FROM order
WHERE order.order_id=item.order_id
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.');
```

图 17-1 展示了 apply_discount 过程的 PL/SQL 代码块，代码块被分为四个部分，分别由箭头指向：

- 头部：PROCEDURE apply_discount (company_id_in IN company.company_id%TYPE, discount_in IN NUMBER)
- 声明部分：IS min_discount CONSTANT NUMBER:=.05; max_discount CONSTANT NUMBER:=.25; invalid_discount EXCEPTION;
- 执行部分：BEGIN IF discount_in BETWEEN min_discount AND max_discount THEN UPDATE item SET item_amount:=item_amount*(1-discount_in); WHERE EXISTS (SELECT 'x' FROM order WHERE order.order_id=item.order_id AND order.company_id=company_id_in); IF SQL%ROWCOUNT = 0 THEN RAISE NO_DATA_FOUND; END IF; ELSE RAISE invalid_discount; END IF;
- 异常处理部分：EXCEPTION WHEN invalid_discount THEN DBMS_OUTPUT.PUT_LINE('The specified discount is invalid.');

```
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。