



程序设计 (第5版)

O'REILLY®

[美] Steven Feuerstein & Bill Pribyl 著

张晓明 译

 人民邮电出版社
POSTS & TELECOM PRESS



程序设计与实现

O'REILLY

11g 数据库编程 PL/SQL 入门与进阶
第 2 版

人民邮电出版社
POST & TELECOM PRESS

Oracle PL/SQL 程序设计 (第 5 版)

(下册)

[美] Steven Feuerstein Bill Pribyl 著

张晓明 译

人 民 邮 电 出 版 社

北 京

目录

(下册)

第5部分 构造 PL/SQL 应用程序

第 17 章 过程、函数与参数	543
17.1 代码模块化	543
17.2 过程	545
17.2.1 调用一个过程	547
17.2.2 过程的头部	548
17.2.3 过程体	548
17.2.4 END 标签	548
17.2.5 RETURN 语句	549
17.3 函数	549
17.3.1 函数的结构	549
17.3.2 返回的数据类型	552
17.3.3 END 标签	552
17.3.4 调用函数	553
17.3.5 不带参数的函数	554
17.3.6 函数的头部	554
17.3.7 函数体	555
17.3.8 RETURN 语句	555
17.4 参数	557
17.4.1 定义参数	558
17.4.2 形参和实参	558
17.4.3 参数模式	559
17.4.4 在 PL/SQL 中明确地把形参和实参关联在一起	562
17.4.5 NOCOPY 参数模式限定符	566
17.4.6 缺省值	566
17.5 局部或者嵌套模块	567
17.5.1 使用局部模块的好处	568
17.5.2 局部模块的作用范围	571
17.5.3 用局部模块让我们的代码更整洁	571

17.6	模块重载	572
17.6.1	使用重载的好处	573
17.6.2	重载的限制	576
17.6.3	关于数字类型的重载	576
17.7	前置声明	577
17.8	高级主题	579
17.8.1	在 SQL 中调用我们的函数	579
17.8.2	表函数	581
17.8.3	确定性函数	591
17.9	把模块化进行到底	592
第 18 章	包	593
18.1	为什么是包?	593
18.1.1	演示包的能力	594
18.1.2	有关包的一些概念	597
18.1.3	图示私有性	599
18.2	构建包的规则	599
18.2.1	包规范	600
18.2.2	包体	601
18.2.3	包的初始化	603
18.3	包成员的调用规则	607
18.4	使用包数据	608
18.4.1	在一个 Oracle 会话内全局可见	609
18.4.2	全局公有数据	609
18.4.3	包游标	610
18.4.4	包的串行化	614
18.5	何时使用包	617
18.5.1	封装对数据的访问	617
18.5.2	避免直接量的硬编码	620
18.5.3	改善内置特性的可用性	622
18.5.4	把逻辑上相关的功能组织在一起	623
18.5.5	缓存静态的会话数据	624
18.6	包和对象类型	624
第 19 章	触发器	626
19.1	DML 触发器	627
19.1.1	DML 触发器的概念	628

19.1.2	创建 DML 触发器	630
19.1.3	DML 触发器的例子: 不许作弊!	635
19.1.4	同一类型的多个触发器	641
19.1.5	何去何从	642
19.1.6	突变表的错误	644
19.1.7	复合触发器: 聚集一堂	645
19.2	DDL 触发器	648
19.2.1	创建 DDL 触发器	649
19.2.2	可用事件	651
19.2.3	可用属性	652
19.2.4	使用事件和属性	653
19.2.5	删除不可删除的	657
19.2.6	INSTEAD OF CREATE 触发器	657
19.3	数据库事件触发器	659
19.3.1	创建数据库事件触发器	659
19.3.2	STARTUP 触发器	660
19.3.3	SHUTDOWN 触发器	661
19.3.4	LOGON 触发器	661
19.3.5	LOGOFF 触发器	661
19.3.6	SERVERERROR 触发器	662
19.4	INSTEAD OF 触发器	666
19.4.1	创建 INSTEAD OF 触发器	666
19.4.2	INSTEAD OF INSERT 触发器	668
19.4.3	INSTEAD OF UPDATE 触发器	670
19.4.4	INSTEAD OF DELETE 触发器	671
19.4.5	填充表	671
19.4.6	嵌套表的 INSTEAD OF 触发器	672
19.5	AFTER SUSPEND 触发器	674
19.5.1	构建 AFTER SUSPEND 触发器	674
19.5.2	看看真实的触发器	676
19.5.3	ORA_SPACE_ERROR_INFO 函数	677
19.5.4	DBMS_RESUMABLE 包	678
19.5.5	捕获多个时间	679
19.5.6	该不该处理?	680
19.6	管理触发器	680
19.6.1	禁用、启用以及删除触发器	680

19.6.2	创建一个禁用的触发器	681
19.6.3	查看触发器	682
19.6.4	检查触发器的有效性	684
第 20 章 管理 PL/SQL 代码		685
20.1	管理数据库内的代码	686
20.1.1	数据字典视图概述	687
20.1.2	显示存储对象的信息	688
20.1.3	源代码的显示和搜索	689
20.1.4	根据程序的大小确定 Pinning 需求	691
20.1.5	获得存储代码的属性	692
20.1.6	通过视图分析和更改触发器状态	693
20.1.7	分析参数信息	693
20.1.8	分析标识符的使用 (Oracle 数据库 11g 的 PL/Scope)	695
20.2	依赖关系的管理以及代码的重编译	697
20.2.1	通过数据字典视图分析依赖关系	698
20.2.2	细粒度依赖 (Oracle 数据库 11g)	702
20.2.3	远程依赖	703
20.2.4	Oracle 的远程调用模式的限制	706
20.2.5	失效的程序单元的重编译	707
20.3	编译时刻警告	711
20.3.1	一个入门例子	711
20.3.2	启用编译时刻警告	712
20.3.3	一些有用的警告	714
20.4	PL/SQL 程序的测试	722
20.4.1	典型的、华而不实的测试技术	723
20.4.2	PL/SQL 代码测试的一般建议	726
20.4.3	PL/SQL 的自动测试选项	727
20.4.4	用 utPLSQL 测试	728
20.4.5	用 Quest Code Tester for Oracle 测试	730
20.5	跟踪 PL/SQL 的执行	732
20.5.1	DBMS_APPLICATION_INFO	734
20.5.2	Quest Error Manager 跟踪	736
20.5.3	DBMS_TRACE 功能	738
20.6	PL/SQL 程序的调试	741
20.6.1	错误的调试方法	741
20.6.2	调试技巧和策略	743

20.7	保护存储过程代码	747
20.7.1	包装的约束和限制	747
20.7.2	使用包装功能	748
20.7.3	通过 DBMS_DDL 动态包装	748
20.7.4	包装过的代码的使用指南	749
20.8	基于版本的重定义 (Oracle 数据库 11g R2 版本)	750
第 21 章	PL/SQL 的性能优化	753
21.1	辅助优化的工具	754
21.1.1	内存使用分析	755
21.1.2	发现 PL/SQL 代码中的瓶颈所在	755
21.1.3	计算消耗时间	760
21.1.4	选择最快的程序	762
21.1.5	避免无限循环	763
21.1.6	性能相关警告	764
21.2	优化过的编译器	765
21.2.1	优化器的工作原理	766
21.2.2	循环 Fetch 操作的运行时优化	769
21.3	数据缓存技术	770
21.3.1	基于包的缓存	771
21.3.2	确定性函数的缓存	776
21.3.3	函数结果缓存 (Oracle 数据库 11g)	778
21.3.4	缓存总结	790
21.4	多行 SQL 的批处理	790
21.4.1	通过 BULK COLLECT 加速查询	792
21.4.2	通过 FORALL 加速 DML	798
21.5	利用管道化的表函数提升性能	808
21.5.1	用基于管道化函数的加载方式替换基于行的插入	809
21.5.2	用管道函数调优 Merge 操作	816
21.5.3	用并行管道函数进行异步的数据卸载	818
21.5.4	并行管道函数中的分区子句和流子句对性能的影响	822
21.5.5	管道函数和基于成本的优化器	823
21.5.6	用管道函数优化复杂的数据加载	829
21.5.7	管道函数结束语	836
21.6	专用的优化技术	837
21.6.1	使用 NOCOPY 参数模式提示符	837
21.6.2	使用正确的数据类型	840

21.7 回顾性能	841
第 22 章 I/O 操作和 PL/SQL	843
22.1 显示信息	843
22.1.1 启用 DBMS_OUTPUT	844
22.1.2 向缓存中写入行	844
22.1.3 从缓存中读取内容	845
22.2 文件的读写	846
22.2.1 UTL_FILE_DIR 参数	847
22.2.2 使用 Oracle 目录	848
22.2.3 打开文件	850
22.2.4 文件已经打开了吗?	852
22.2.5 关闭文件	852
22.2.6 读取文件	853
22.2.7 向文件中写	855
22.2.8 拷贝文件	858
22.2.9 删除文件	859
22.2.10 文件改名和文件移动	859
22.2.11 提取文件属性	860
22.3 发送邮件	861
22.3.1 Oracle 的前提条件	862
22.3.2 配置网络安全	863
22.3.3 发送一个短的 (32 767 或者更少) 的纯文本消息	863
22.3.4 在邮件地址中加上“友好”的名字	865
22.3.5 发送任意长度的纯文本消息	866
22.3.6 发送带有小附件 (<32 767) 的消息	867
22.3.7 以附件形式发送一个小文件 (<32 767)	869
22.3.8 任意大小的附件	870
22.4 使用 Web 数据 (HTTP)	872
22.4.1 “分片”获得一个 Web 页面	873
22.4.2 把页面提取到一个 LOB 中	874
22.4.3 使用 HTTP 的用户名/密码验证	875
22.4.4 获取一个 SSL 加密的 Web 页面 (通过 HTTPS)	876
22.4.5 通过 GET 或者 POST 给一个 Web 页面提交数据	877
22.4.6 禁用 Cookies 或者 Cookies 持久化	881
22.4.7 从 FTP 服务器获取数据	881
22.4.8 使用代理服务器	882

22.5	PL/SQL 中可用的其他 I/O 类型	882
22.5.1	数据库管道、队列、告警	883
22.5.2	TCP Socket	883
22.5.3	Oracle 的内置 Web 服务器	883

第 6 部分 高级 PL/SQL 主题

第 23 章	应用安全与 PL/SQL	887
23.1	安全概述	887
23.2	加密	888
23.2.1	密钥长度	890
23.2.2	算法	890
23.2.3	填补和连接	892
23.2.4	DBMS_CRYPTO 包	892
23.2.5	数据加密	894
23.2.6	LOB 的加密	897
23.2.7	安全文件	897
23.2.8	数据解密	898
23.2.9	生成密钥	899
23.2.10	密钥管理	900
23.2.11	加密散列	905
23.2.12	使用消息验证码	907
23.2.13	使用透明数据加密 (TDE)	908
23.2.14	透明的表空间加密	910
23.3	行级安全	912
23.3.1	为什么要学习 RLS	914
23.3.2	一个简单的 RLS 示例	915
23.3.3	使用动态策略	919
23.3.4	使用列敏感的 RLS	923
23.3.5	RLS 调试	927
23.4	应用程序上下文	930
23.4.1	使用应用程序上下文	931
23.4.2	上下文的安全	932
23.4.3	把上下文用作 RLS 的谓词条件	933
23.4.4	识别出非数据库的用户	936
23.5	细粒度审计	938
23.5.1	为什么要学习 FGA	939

23.5.2	一个简单的 FGA 示例	940
23.5.3	访问多少列	942
23.5.4	查看审计跟踪信息	943
23.5.5	使用绑定变量	943
23.5.6	使用句柄模块	945
第 24 章	PL/SQL 架构	947
24.1	DIANA 是谁 (或者是什么)	947
24.2	Oracle 是如何执行 PL/SQL 代码的	948
24.2.1	一个例子	949
24.2.2	编译器的限制	952
24.3	PL/SQL 的缺省包	952
24.4	执行权限模型	955
24.4.1	定义者权限模型	956
24.4.2	调用者权限模式	960
24.4.3	组合权限模型	962
24.5	条件编译	963
24.5.1	条件编译的例子	964
24.5.2	查询指令	965
24.5.3	\$IF 指令	968
24.5.4	\$ERROR 指令	970
24.5.5	把代码和包常量同步	970
24.5.6	用查询指令实现程序专有设置	971
24.5.7	使用预处理后的代码	972
24.6	PL/SQL 和数据库实例内存	974
24.6.1	PGA、UGA 和 CGA	974
24.6.2	游标、内存及其他	975
24.6.3	减少内存用的技巧	977
24.6.4	如果内存用光了该怎么办	987
24.7	原生式编译	990
24.7.1	什么时候使用解释模式	990
24.7.2	什么时候使用原生模式	991
24.7.3	原生编译和数据库版本	991
24.8	我们需要知道的	991
第 25 章	PL/SQL 的全球化和本地化	993
25.1	概述和术语	995
25.2	Unicode 入门	996

25.2.1	国家字符集的数据类型	998
25.2.2	字符编码	998
25.2.3	和全球化支持相关的参数	999
25.2.4	Unicode 函数	1000
25.3	字符语义	1007
25.4	字符串排序顺序	1011
25.4.1	二进制排序	1012
25.4.2	单语言排序	1013
25.4.3	多语言排序	1015
25.5	多语言信息检索	1016
25.5.1	信息检索和 PL/SQL	1018
25.6	日期/时间	1021
25.6.1	时间戳数据类型	1021
25.6.2	日期/时间格式	1022
25.7	货币转换	1026
25.8	PL/SQL 的全球化开发工具箱	1028
25.8.1	UTL_I18N 工具包	1028
25.8.2	UTL_LMS 异常处理包	1031
25.8.3	GDK 实现选项	1032
第 26 章	PL/SQL 的面向对象特性	1034
26.1	Oracle 对象特性的介绍	1034
26.2	对象类型示例	1036
26.2.1	创建一个基类	1037
26.2.2	创建子类型	1039
26.2.3	方法	1040
26.2.4	在 Oracle 数据库 11g 中调用父类的方法	1045
26.2.5	保存、提取、使用持久化对象	1046
26.2.6	演变和创建	1054
26.2.7	回到指针吗?	1056
26.2.8	泛化数据: ANY 类型	1063
26.2.9	我也可以自己做	1067
26.2.10	对象的比较	1071
26.3	对象视图	1075
26.3.1	一个示例的关系系统	1077
26.3.2	带有集合属性的对象视图	1078
26.3.3	对象子视图	1081

26.3.4 带有反关系的对象视图	1083
26.3.5 INSTEAD OF 触发器	1084
26.3.6 对象视图和对象表的区别	1086
26.4 维护对象类型和对象视图	1087
26.4.1 数据字典	1087
26.4.2 权限	1089
26.5 来自一个关系开发者的总结思考	1091
附录 A 正则表达式的元字符和函数参数	1093
A.1 元字符	1093
A.2 函数和参数	1096
A.2.1 正则表达式函数	1096
A.2.2 正则表达式参数	1097
附录 B 数字格式模型	1099
附录 C 日期格式模型	1102

构造 PL/SQL 应用程序

这一部分对前面的内容进行了综合。现在，你已经掌握了基础知识。你知道如何声明变量和使用变量。你俨然已经是一个错误处理和循环结构方面的专家了。是时候构建应用程序了——你可以构建块、创建过程、函数、包、触发器，这些在第 17 章到第 19 章中介绍。第 20 章讨论了对 PL/SQL 代码的管理，包括对程序的测试和调试以及管理依赖关系；这一章还对 Oracle 数据库 11g R2 版本引入的基于版本的重定义功能进行了简要的介绍，第 21 章，这是第 5 版新加的一章，主要关注的是如何可以通过若干工具和技术帮助你的 PL/SQL 程序达到最好的性能。第 22 章介绍了 PL/SQL 的 I/O 技术，从 DBMS_OUTPUT（输出内容到屏幕）和 UTL_FILE（文件的读写）到 UTL_MAIL（发送邮件）和 UTL_HTTP（从 Web 页面提取数据）。

第 17 章 过程、函数与参数

第 18 章 包

第 19 章 触发器

第 20 章 管理 PL/SQL 代码

第 21 章 PL/SQL 的性能优化

第 22 章 I/O 操作和 PL/SQL

过程、函数与参数

本书前面的内容中已经详细的介绍了 PL/SQL 语言的各种组件：包括游标、异常、循环、变量等。要想用 PL/SQL 编写应用程序，确实也需要知道所有这些组件，不过比这些内容更重要的是该如何把这些片段组装在一起，进而创建一个结构良好的、容易理解的、便于维护的程序。

交给我们的任务很少有简单直接的。而需要的解决办法也很少能不假思索就能写到纸上或者输入到键盘的。我们构建的系统通常是庞大并且复杂的，会有许多相互影响甚至是相互冲突的组件组成。而且，从最终用户的角度来说，他们想要得到东西是比之前更容易使用的、功能更强大的应用程序，这些应用程序的内部实现也相应的变得更加复杂了。

今天我们工作所面临的最大挑战来自于该如何简化我们的环境的复杂性。当面临一个复杂的问题时，感觉上是很恐怖的。我们该从那里入手呢？我们该如何快刀斩乱麻，从一堆需求和特性中找到出路呢？

我们人类毕竟赶不上大型的并行计算机。就算是最聪明的人也不可能同时处理 7 个任务（加减二）。我们需要把一个庞大的、吓人的项目分解成小的、更容易管理的组件，然后再进一步把这些组件分解成一个个好理解的单独程序。这样我们就知道该如何构建和测试这些程序了，最后再用这些组件构成一个完整的应用程序。

无论我们用的是“从顶到下的设计”（也叫做逐步分解，在“局部或者嵌模块”一节会有详细的介绍）还是某种其他的方法学，毋庸置疑的是，通过把程序模块化成一个个过程、函数或者对象类型，我们会得到一个高质量的容易管理的应用程序。

17.1 代码模块化

模块化是把一个大的代码块拆分成若干个小片段（模块）的过程，然后就可以在其他

模块中调用这些模块了。代码的模块化非常类似于数据的规范化，得到的好处是相同的，而且还有一些额外的优点。利用模块化，我们的代码可以：

重用性更好

通过把一大段代码或者整个程序分解成独立的“即插即用”的组件，我们经常发现，许多模块其实可以被当前应用中的其他程序使用。只要有良好的设计，这些工具程序甚至可以用于其他应用程序！

管理性更好

一个 1 000 行的程序或者 5 个单独的每个只有 200 行的并且互相调用的程序，你更愿意调试哪一个？在处理小任务时，我们的注意力会更加集中，也能处理的更好。我们可以在每个程序范围进行测试和调试（也叫做单元测试），然后再把这些独立的模块合并起来，进行一个更复杂的集成测试。

可读性更好

模块可以被命名，可以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，程序行为就越容易阅读和理解。通过模块化，我们可以更专注于程序全局而不是一个个单独的可执行语句。我们甚至可以实现最隐蔽的软件：完全自我说明的代码。

更可靠的代码

按照模块化思路生成的代码的错误会更少。而且就算是有错误，也会因为它们已经被隔离在模块范围中而更容易被修正。而且，由于代码量更少并且可读性更好，我们维护起来也更容易。

只要你能熟练地使用 PL/SQL 语言中的各种不同的循环、条件、游标结构（IF 语句、循环等），你其实就已经为构建应用系统做好准备了。不过，你还必须知道如何创建、整合 PL/SQL 的模块，这才真正可以构建应用程序。

PL/SQL 语言中提供了下面这些模块结构，你可以用不同的方法实现代码的模块化：

过程

过程是一个可以像可执行 PL/SQL 语句一样调用的程序，一个过程可以执行一个或者多个动作。我们可以通过参数列表向过程传递或者从过程传出信息。

函数

函数是一个通过 RETURN 语句返回数据的程序，使用起来就像是一个 PL/SQL 表达式。我们可以通过参数列表给函数传入参数。也可以通过参数列表传出参数，不过通常情况下这么做并不好。