开发人员专业技术丛书

HZ BOOKS

Oracle PL/SQL by Example

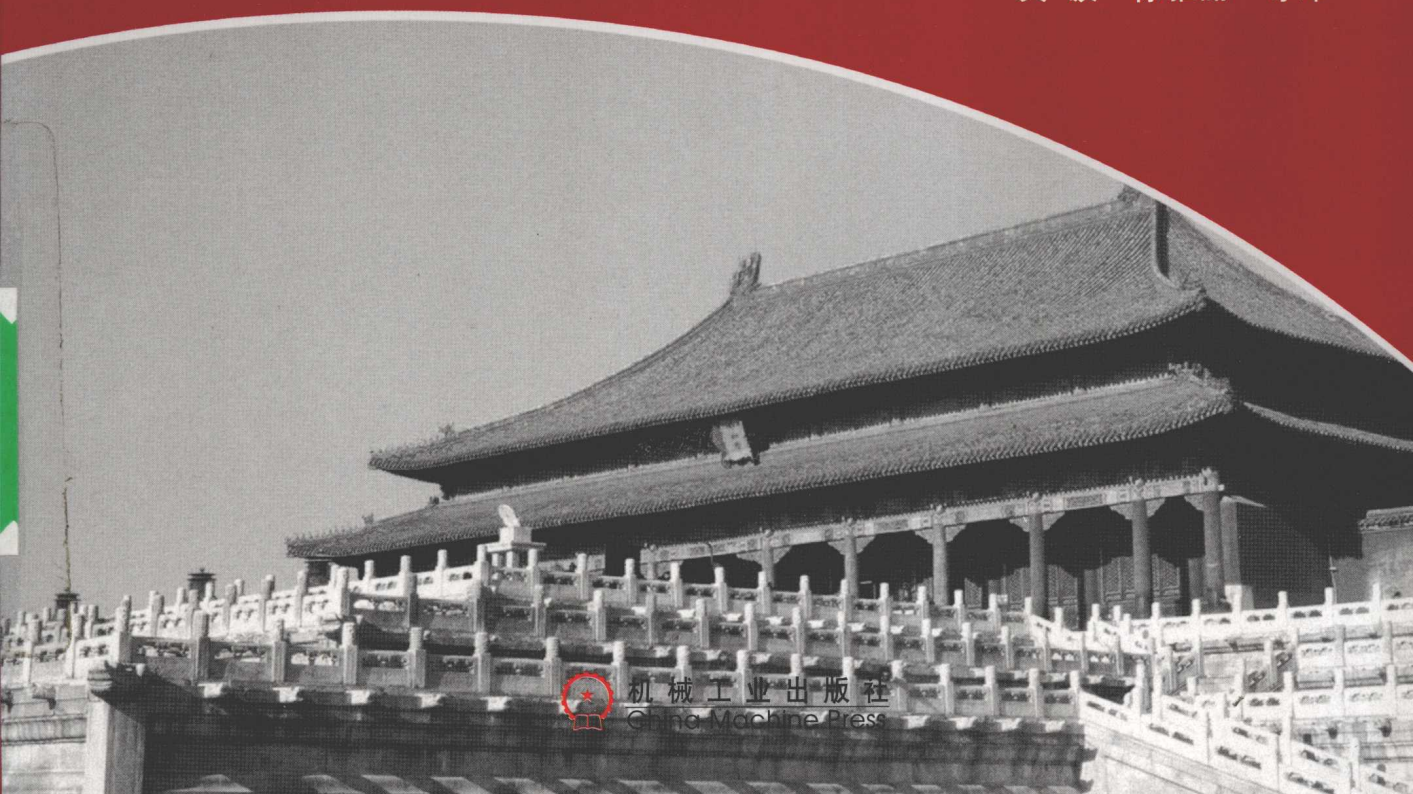
Fourth Edition

Oracle PL/SQL实例精解

（原书第4版）

（美） Benjamin Rosenzweig
Elena Silvestrova Rakhimov 著

龚 波 徐雅丽 等译



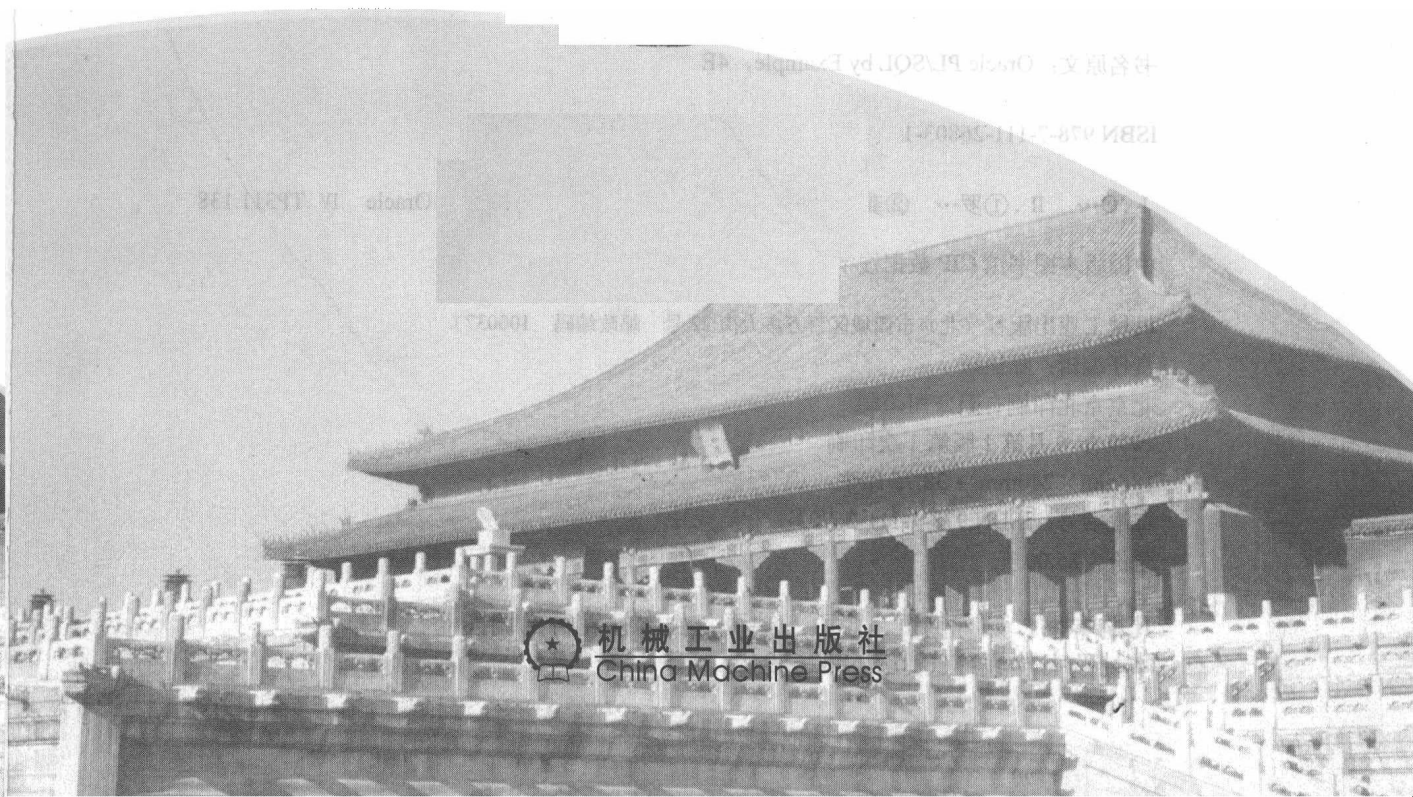
机械工业出版社
China Machine Press

开发人员专业技术丛书

Oracle PL/SQL by Example
Fourth Edition

Oracle PL/SQL实例精解

（原书第4版）



机械工业出版社
China Machine Press

本书是一本逐步分解的，详尽的 PL/SQL 编程教程，使用真实场景的试验、范例和练习来介绍读者所需的 PL/SQL 编程技能，涵盖 Oracle 11g 的最新特性。作者的写作手法源自于在哥伦比亚大学教授 PL/SQL 编程技术的经验，深度探索 PL/SQL 编程技术，融合自己的最佳实践。使用本书，读者可以快速掌握 PL/SQL 编程基本知识，并建立工程化的概念，是市面上难得的 PL/SQL 教程。

本书版权登记号：图字：01-2009-1366

图书在版编目（CIP）数据

Oracle PL/SQL 实例精解 /（美）罗森维格（Rosenzweig, B.）等著；龚波等译 . —北京：机械工业出版社，2009.6

（开发人员专业技术丛书）

书名原文：Oracle PL/SQL by Example, 4E

ISBN 978-7-111-26803-1

Ⅰ.O… Ⅱ.①罗… ②龚… Ⅲ.关系数据库－数据库管理系统，Oracle Ⅳ.TP311.138

中国版本图书馆 CIP 数据核字（2009）第 055727 号

机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码 100037）

责任编辑：陈佳媛

北京京北印刷有限公司印刷

2009 年 6 月第 1 版第 1 次印刷

186mm×240mm • 38.75 印张

标准书号：ISBN 978-7-111-26803-1

定价：85.00 元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

译 者 序

Oracle 11g 是一个功能极其强大，并且非常灵活的关系数据库系统。为了基于 Oracle 设计有用的应用程序，则有必要理解 Oracle 是如何操作存储在本系统中数据的。PL/SQL 就是这样一种实现数据操作的重要工具，你不仅可以在 Oracle 内部使用它，而且可以在自己的应用程序中应用它。在多种环境下 PL/SQL 都是非常有用的，并且在不同环境下，它具有不同的优势。

本书是一本面向任务的 Oracle 教材，适合于课堂教学或者自学，特别适合于需要明确实例和练习的读者。本书随着 Oracle 版本发布而不断推陈出新，目前已经是第 4 版，主要面向于 Oracle 11g 版本的开发和使用。本版与前几版相比，其特点包括：内容更新到 Oracle 11g，提供成百上千的范例、问题和答案，展示真实的实验场景等。

本书作者是 Oracle 系列图书的畅销作者，编写了多本有关 PL/SQL 的图书。本书不仅适用于 PL/SQL 的初学者，而且适合于 Oracle 开发人员参考。从根本上来说，本书是非常实用的，而且是非常宝贵的参考资源！

我们很荣幸能够有机会承担本书的翻译工作。在翻译过程中，经常为一句话、一个术语进行反复的讨论，到处查找资料，力图使本书的翻译能正确、贴切地反映原文的意思，同时注意使句子、段落符合中国人的语言习惯。我们真挚地希望你能从本书中获益，这是作者的初衷，也是我们的愿望！

本书由龚波、徐雅丽等组织翻译，很多人为本书的翻译和出版付出努力工作。参加本书翻译的其他人员包括：张平、李平芳、冯军、龚志翔、李志、姜南、任志宏、王强、刘刚等。

由于时间仓促，且译者经验和水平有限，译文中难免有不妥之处，恳请读者批评指正！

译 者

2009 年 1 月

致　谢

Benjamin Rosenzweig：衷心感谢本书的合著者——Elena Silvestrova Rakhimov，非常荣幸能够与这位优秀且知识渊博的伙伴合作。感谢 Douglas Scherer 给予我编写本书的机会，以及在编写过程中所提供的一贯支持和帮助。非常感谢 Prentice Hall 的工作团队，其成员有 Trina MacDonald、Songlin Qiu、Todd Taber、Shahdad Moradi 和 Oleg Voskoboynikov。他们的真知灼见、建议和编辑工作有效地改进了原稿，使本书得以出版发行。最后，感谢我的家人和很多好朋友，尤其是 Edward Clarin 和 Edward Knopping，感谢他们帮助我度过漫长的编写过程，容忍我在很多晚上和周末时间无法与他们呆在一起。

Elena Silvestrova Rakhimov：本书的完成得到很多人给予的帮助和建议。这里尤其感谢我的合作伙伴 Benjamin Rosenzweig，是他把本项目变成一个有价值和有意义的工作体验。特别感谢 Prentice Hall 的 Trina MacDonald、Songlin Qiu、Todd Taber 等人，本书是在他们的努力下才得以推向市场。感谢 Shahdad Moradi 和 Oleg Voskoboynikov 提供的有价值评论和建议。最重要的是，要感谢我的家人，他们的欣喜、热情、激情和支持激励着我更加努力工作，让我始终沐浴在爱中。

作者简介

Benjamin Rosenzweig 是 Misys Treasury & Capital Markets 公司的软件开发经理，自从 2002 年就在那里工作。在此之前的 3 年多时间内，他担任 Oracle 公司的 Custom Development Department 的主要咨询师。此外，他曾在尼泊尔的加德满都市开发藏文 - 英文的电子字典，来支持位于 Goldman Sachs 的演示中心，以及管理 TIAA-CREE 的交易系统。从 1998 年，在纽约，Rosenzweig 始终是哥伦比亚大学计算机技术及应用（CTA）的讲师。在 2002 年，他被 CTA 委员会的主席授予“杰出技术大奖”。他拥有 Reed College 的学士学位，以及哥伦比亚大学所颁发的数据库开发和设计方面的证书。他曾在 Prentice Hall 出版了《Oracle Forms Developer：The Complete Video Course》和《Oracle Web Application Programming for PL/SQL Developers》。

Elena Silvestrova Rakhimov 具有 15 年以上企业和业务环境的数据库开发经验，所涉及领域包括非营利组织和 Wall Street（华尔街）。现在，她就职于 Alea Software，是高级开发人员（Senior Developer）和团队领导（Team Lead）。她倡导实践，Rakhimov 在专业学术领域很突出，始终在哥伦比亚大学卓越的计算机技术及应用专业课程教授关系数据库编程。在哥伦比亚大学，她接受过数据库分析和设计方面的专业教育，并且在阿塞拜疆的巴库州立大学学习过应用数学。目前，她居住在加拿大的温哥华。

Oracle 11g 中 PL/SQL 新特性简介

Oracle 11g 引入 PL/SQL 很多新特性和增强之处。这里会概述本书没有覆盖的特性；对于本书所涉及的特性，也会明确指出讨论细节的章节。下面的特性列表也可以从《PL/SQL Language Reference》手册的“What's New in PL/SQL”部分获取；这个参考手册是 Oracle 在线帮助的一部分。

新的 PL/SQL 特性和增强之处包括：

- 增强正则表达式内置 SQL 函数。
- SIMPLE_INTEGER、SIMPLE_FLOAT 和 SIMPLE_DOUBLE 数据类型。
- CONTINUE 语句。
- PL/SQL 表达式中的序列。
- 动态 SQL 增强。
- PL/SQL 子程序调用中的命名和混合表示法。
- 跨会话的 PL/SQL 函数结果缓存。
- 对触发器的更多控制。
- 复合触发器。
- 数据库驻留连接池。
- 自动子程序内联。
- PL/Scope。
- PL/SQL 分层配置器。
- PL/SQL 本地编译器直接生成本地代码。

增强正则表达式内置 SQL 函数

在这个 Oracle 版本中，引入一个新的正则表达式内置函数：REGEXP_COUNT。这个函数返回源字符串中出现指定搜索模式的次数。例如：

示例

```
SELECT
   REGEXP_COUNT ('Oracle PL/SQL By Example Updated for Oracle 11g',
                 'ora', 1, 'i')
  FROM dual;

REGEXP_COUNT('ORACLEPL/SQLBYEXAMPLEUPDATEDFORORACLE11G','ORA',1,'I')
--------------------------------------------------------------------
                                                                   2
```

REGEXP_COUNT 函数会返回在源字符串 'Oracle PL/SQL... ' 中搜索模式 'ora' 出现的次数；1 表示从源字符串的第几个字符开始搜索，'i' 说明模式匹配时不区分大小写。

已有的正则表达式内置函数 REGEXP_INSTR 和 REGEXP_SUBSTR 都有一个名为 SUBEXPR 的新参数。这个参数是搜索模式的子表达式。实际上是圆括号中搜索模式的一部分，用于限制模式匹配，如下例所示：

示例

```
SELECT
   REGEXP_INSTR ('Oracle PL/SQL By Example Updated for Oracle 11g',
                 '((ora)(cle))', 1, 2, 0, 'i')
  FROM dual;

REGEXP_INSTR('ORACLEPL/SQLBYEXAMPLEUPDATEDFORORACLE11G',...)
------------------------------------------------------------
                                                          38
```

REGEXP_INSTR 函数会返回在 'Oracle PL/SQL...' 字符串中搜索模式（ora）（cle）中第一个子表达式 'ora' 的第二次出现的位置。1 表示从源字符串的第几个字符开始搜索；2 表示在源字符串中子表达式的出现次数；0 表示满足匹配条件的第一个字符的位置；'i' 表示匹配时不区分大小写。

SIMPLE_INTEGER、SIMPLE_FLOAT 和 SIMPLE_DOUBLE 数据类型

这些数据类型分别是 PLS_INTEGER、BINARY_FLOAT 和 BINARY_DOUBLE 的预定义子类型。因此，它们与其基类型拥有相同的值范围。除此之外，这些子类型没有 NOT NULL 约束。

当 PLSQL_CODE_TYPE 参数被设置为 NATIVE 时，相对于各自的基类型，这些子类型具有明显的性能优势。其中原因是，这些子类型的算术运算是在硬件层直接实现的。请注意，当 PLSQL_CODE_TYPE 被设置为 INTEPRETED（默认值）时，性能优势要小得多。下面的范例可以说明这一点。

示例

```
SET SERVEROUTPUT ON
DECLARE
    v_pls_value1     PLS_INTEGER := 0;
    v_pls_value2     PLS_INTEGER := 1;

    v_simple_value1 SIMPLE_INTEGER := 0;
    v_simple_value2 SIMPLE_INTEGER := 1;

    -- Following are used for elapsed time calculation
    -- The time is calculated in 100th of a second
    v_start_time     NUMBER;
    v_end_time       NUMBER;

BEGIN
    -- Perform calculations with PLS_INTEGER
    v_start_time := DBMS_UTILITY.GET_TIME;

    FOR i in 1..50000000 LOOP
```

```
      v_pls_value1 := v_pls_value1 + v_pls_value2;
   END LOOP;

   v_end_time := DBMS_UTILITY.GET_TIME;
   DBMS_OUTPUT.PUT_LINE ('Elapsed time for PLS_INTEGER: '||
      (v_end_time - v_start_time));

   -- Perform the same calculations with SIMPLE_INTEGER .
   v_start_time := DBMS_UTILITY.GET_TIME;

   FOR i in 1..50000000 LOOP
      v_simple_value1 := v_simple_value1 + v_simple_value2;
   END LOOP;

   v_end_time := DBMS_UTILITY.GET_TIME;
   DBMS_OUTPUT.PUT_LINE ('Elapsed time for SIMPLE_INTEGER: '||
      (v_end_time - v_start_time));
END;
```

上述脚本借助于数值型 FOR 循环，比较 PLS_INTEGER 数据类型与子类型 SIMPLE_INTEGER 之间的性能差异。请注意，PLSQL_CODE_TYPE 参数被设置为默认值 INTERPRETED。

```
Elapsed time for PLS_INTEGER: 147
Elapsed time for SIMPLE_INTEGER: 115

PL/SQL procedure successfully completed.
```

CONTINUE 语句

类似于 EXIT 语句，CONTINUE 语句能够控制循环迭代。然而，EXIT 语句都会终止当前循环，把执行权传递到循环之外；CONTINUE 语句也会终止循环的当前迭代，并把执行权传递到本循环的下一次迭代中。本书第 7 章会详细讨论 CONTINUE 语句。

PL/SQL 表达式中的序列

在 Oracle 11g 之前，在 PL/SQL 中，只能通过查询才可以访问序列伪列 CURRVAL 和 NEXTVAL。从 Oracle 11g 开始，就可以使用表达式访问这些伪列了。这种改变不仅可以改进 PL/SQL 源代码，也可以改进运行时性能和可扩展性。例如：

示例

```
CREATE SEQUENCE test_seq START WITH 1 INCREMENT BY 1;

Sequence created.

SET SERVEROUTPUT ON
DECLARE
   v_seq_value NUMBER;
BEGIN
   v_seq_value := test_seq.NEXTVAL;
   DBMS_OUTPUT.PUT_LINE ('v_seq_value: '||v_seq_value);
END;
```

当在 Oracle 10g 中执行上述脚本时，会出现如下错误：

```
   v_seq_value := test_seq.NEXTVAL;
                          *
ERROR at line 4:
ORA-06550: line 4, column 28:
PLS-00357: Table,View Or Sequence reference 'TEST_SEQ.NEXTVAL' not
allowed in this context
ORA-06550: line 4, column 4:
PL/SQL: Statement ignored
```

但是，当在 Orcale 11g 中执行时，可以正确完成任务：

```
v_seq_value: 1

PL/SQL procedure successfully completed.
```

下面的范例演示：当 PL/SQL 表达式用于处理序列时，能够带来性能改进。

示例

```
SET SERVEROUTPUT ON
DECLARE
   v_seq_value  NUMBER;

   -- Following are used for elapsed time calculation
   v_start_time NUMBER;
   v_end_time   NUMBER;

BEGIN
   -- Retrieve sequence via SELECT INTO statement
   v_start_time := DBMS_UTILITY.GET_TIME;

   FOR i in 1..10000 LOOP
      SELECT test_seq.NEXTVAL
        INTO v_seq_value
        FROM dual;
   END LOOP;

   v_end_time := DBMS_UTILITY.GET_TIME;
   DBMS_OUTPUT.PUT_LINE
      ('Elapsed time to retrieve sequence via SELECT INTO: '||
       (v_end_time-v_start_time));

   -- Retrieve sequence via PL/SQL expression
   v_start_time := DBMS_UTILITY.GET_TIME;

   FOR i in 1..10000 LOOP
      v_seq_value := test_seq.NEXTVAL;
   END LOOP;

   v_end_time := DBMS_UTILITY.GET_TIME;
```

```
   DBMS_OUTPUT.PUT_LINE
      ('Elapsed time to retrieve sequence via PL/SQL expression: '||
       (v_end_time-v_start_time));
END;

Elapsed time to retrieve sequence via SELECT INTO: 52
Elapsed time to retrieve sequence via PL/SQL expression: 43

PL/SQL procedure successfully completed.
```

动态 SQL 增强

在这个版本中，Oracle 已经大大增强了本地动态 SQL（native dynamic SQL）和 DBMS_SQL 包。

借助于本地动态 SQL，可以生成大于 32K 的动态 SQL 语句。换句话说，它支持 CLOB 数据类型。本书第 17 章会详细讨论本地动态 SQL。

现在，DBMS_SQL 包支持本地动态 SQL 支持的所有数据类型，其中包括 CLOB 数据类型。除此之外，使用函数 DBMS_SQL.TO_REFCURSOR 和 DBMS_SQL.TO_CURSOR_NUMBER，可以在本地动态 SQL 和 DBMS_SQL 包之间切换。

PL/SQL 子程序调用中命名和混合表示法

在 Oracle 11g 之前，调用函数的 SQL 语句必须按照位置表示法来设定参数。在这个版本中，同时支持混合和命名表示法。本书第 21 章和第 23 章提供有关位置、命名和混合表示法的详细说明。

考虑下面的范例：

示例

```
CREATE OR REPLACE FUNCTION test_function
   (in_val1 IN NUMBER, in_val2 IN VARCHAR2)
RETURN VARCHAR2
IS
BEGIN
   RETURN (in_val1||' - '||in_val2);
END;

Function created.

SELECT
   test_function(1, 'Positional Notation') col1,
   test_function(in_val1 => 2, in_val2 => 'Named Notation') col2,
   test_function(3, in_val2 => 'Mixed Notation') col3
  FROM dual;

COL1                    COL2                COL3
----------------------  ------------------  ------------------
1 - Positional Notation 2 - Named Notation  3 - Mixed Notation
```

请注意，混合表示法有个限制：位置表示法也许不会遵循命名表示法。下面的 SELECT 语句演示了这种情况：

```
SELECT
   test_function(1, 'Positional Notation') col1,
   test_function(in_val1 => 2, in_val2 => 'Named Notation') col2,
   test_function(in_val1 => 3, 'Mixed Notation') col3
  FROM dual;

   test_function(in_val1 => 3, 'Mixed Notation') col3
   *
ERROR at line 4:
ORA-06553: PLS-312: a positional parameter association may not follow
a named association
```

跨会话的 PL/SQL 函数结果缓存

结果缓存函数是在缓存中存储参数值和结果的函数。这意味着，当使用相同的参数值调用函数时，会从缓存中检索函数结果，而不会重新计算。这种缓存机制称为单会话缓存，因为每个会话都需要存储函数参数及其结果的缓存的副本。

从 Oracle 11g 开始，结果缓存函数的缓存机制已经扩展为跨会话缓存。换句话说，现在，结果缓存函数的参数值和结果存储在共享的全局区域（shared global area, SGA）中，适用于任何会话。请注意，当应用程序从单会话缓存转换为跨会话缓存时，它需要更多的 SGA，而不会减少系统内存。

考虑下面的范例，它会演示如何创建结果缓存的函数。

示例

```
-- Package specification
CREATE OR REPLACE PACKAGE test_pkg AS

   -- User-defined record type
   TYPE zip_record IS RECORD
      (zip   VARCHAR2(5),
       city  VARCHAR2(25),
       state VARCHAR2(2));

   -- Result-cached function
   FUNCTION get_zip_info (in_zip NUMBER) RETURN zip_record
   RESULT_CACHE;
END test_pkg;
/

-- Package body
CREATE OR REPLACE PACKAGE BODY test_pkg AS

-- Result-cached function
FUNCTION get_zip_info (in_zip NUMBER) RETURN zip_record
RESULT_CACHE
RELIES_ON (ZIPCODE)
IS
   rec zip_record;
BEGIN
```

```
    SELECT zip, city, state
      INTO rec
      FROM zipcode
     WHERE zip = in_zip;
    RETURN rec;

EXCEPTION
   WHEN NO_DATA_FOUND THEN
      RETURN null;
END get_zip_info;

END test_pkg;
/
```

请注意 RESULT_CACHE 和 RELIES_ON 子句的用法。RESULT_CACHE 指定函数是结果缓存函数，RELIES_ON 指定函数结果所依赖的表或者视图。

对触发器的更多控制

从 Oracle 11g 开始，CREATE OR REPLACE TRIGGER 子句也许会包含 ENABLE、DISABLE 和 FOLLOWS 选项。借助于 ENABLE 和 DISABLE 选项，能够分别在启用或者禁用状态创建触发器。使用 FOLLOWS 选项可以指定触发器被触发的顺序。请注意，FOLLOWS 选项适用于在相同表所定义的触发器，并在相同的定时点（timing point）进行触发。本书第 13 章会详细讨论触发器。

复合触发器

复合触发器是一种新型的触发器，能够把不同类型的触发器合并为一个触发器。尤其需要注意的是，可以合并如下触发器：

- 在触发语句之前触发的语句触发器。
- 在触发语句所影响每行记录之前触发的行触发器。
- 在触发语句所影响每行记录之后触发的行触发器。
- 在触发语句之后触发的语句触发器。

这意味着，当某事务发生时，一个触发器也许在不同时间触发。本书第 14 章会详细讨论复合触发器。

数据库驻留连接池

数据库驻留连接池（Database Resident Connection Pool，DRCP）提供被不同多层进程共享的连接池。使用新的包 DBMS_CONNECTION_POOL，数据库管理员可以启动和停止 DRCP，并配置其参数。

自动子程序内联

PL/SQL 编译器会把 PL/SQL 代码转换为机器代码。从 Oracle 11g 开始，当编译 PL/SQL 代码时，PL/SQL 编译器可以使用性能优化器。使用性能优化器，PL/SQL 编译器可以重新组织 PL/SQL 代码，以提升处理性能。PL/SQL 编译器所使用的优化等级是由参数 PLSQL_OPTIMIZE_LEVEL 控制的。这个参数的值在 0 和 2 之间，默认值是 2。这意味着，在默认情况下，PL/SQL 编译器会执行优化操作。

从 Oracle 11g 开始，PL/SQL 编译器可以执行子程序内联。子程序内联会使用被调用子程序的真实副本来替换子程序调用语句。通过指定 PRAGMA INLINE 参数或者把 PLSQL_OPTIMIZE_LEVEL 参数设置为 3，就可以实现这个目标。当 PLSQL_OPTIMIZE_LEVEL

参数被设置为 3 时，PL/SQL 编译器就会在合适时执行自动的子程序内联。但是，在有些情况下，PL/SQL 编译器也许会选择不执行子程序内联，原因是它认为这是不可取的。

下面的范例演示如何使用 PRAGMA INLINE 参数。请注意，在这个例子中，PLSQL_OPTIMIZE_LEVEL 参数已经被设置为默认值 2。

示例

```
SET SERVEROUTPUT ON
DECLARE
   v_num    PLS_INTEGER := 1;
   v_result PLS_INTEGER;

   -- Following are used for elapsed time calculation
   v_start_time NUMBER;
   v_end_time   NUMBER;

   -- Define function to test PRAGMA INLINE
   FUNCTION test_inline_pragma
      (in_num1 IN PLS_INTEGER, in_num2 IN PLS_INTEGER)
   RETURN PLS_INTEGER
   IS
   BEGIN
      RETURN (in_num1 + in_num2);
   END test_inline_pragma;

BEGIN
   -- Test function with INLINE PRAGMA enabled
   v_start_time := DBMS_UTILITY.GET_TIME;

   FOR i in 1..10000000 LOOP
      PRAGMA INLINE (test_inline_pragma, 'YES');
      v_result := test_inline_pragma (1, i);
   END LOOP;

   v_end_time := DBMS_UTILITY.GET_TIME;
   DBMS_OUTPUT.PUT_LINE
      ('Elapsed time when PRAGMA INLINE enabled: '||
       (v_end_time-v_start_time));

   -- Test function with PRAGMA INLINE disabled
   v_start_time := DBMS_UTILITY.GET_TIME;

   FOR i in 1..10000000 LOOP
      PRAGMA INLINE (test_inline_pragma, 'NO');
      v_result := test_inline_pragma (1, i);
   END LOOP;

   v_end_time := DBMS_UTILITY.GET_TIME;
   DBMS_OUTPUT.PUT_LINE
      ('Elapsed time when INLINE PRAGMA disabled: '||
```

```
    (v_end_time-v_start_time));
END;

Elapsed time when PRAGMA INLINE enabled: 59
Elapsed time when PRAGMA INLINE disabled: 220

PL/SQL procedure successfully completed.
```

请注意，当 PRAGMA INLINE 参数置于如下语句之前时，PRAGMA INLINE 参数影响对指定子程序的每个调用：

- 赋值语句。
- 调用语句。
- 条件语句。
- CASE 语句。
- CONTINUE-WHEN 语句。
- EXECUTE IMMEDIATE 语句。
- EXIT-WHEN 语句。
- LOOP 语句。
- RETURN 语句。

PL/Scope

PL/Scope 会收集和组织 PL/SQL 代码中有关用户定义标识符的数据。这个工具主要用于交互式开发环境，比如 SQL Developer 或者 Jdeveloper，而不会直接在 PL/SQL 中使用。

PL/SQL 分层配置器

使用 PL/SQL 分层配置器（hierarchical profiler），可以监控 PL/SQL 应用程序。换句话说，它会收集有关应用程序执行的统计信息，比如 SQL 和 PL/SQL 的执行次数、应用程序调用特定子程序的次数，以及子程序自身执行所花费的时间。

分层配置器是借助于 Oracle 所提供的包 DBMS_HPROF 实现的。本书第 24 章会详细讨论 DBMS_HPROF。

直接生成本地代码（native code）的 PL/SQL 本地编译器（native compiler）

在这个 Oracle 版本中，PL/SQL 本地编译器会直接生成本地代码。从前，PL/SQL 代码会被翻译为 C 代码，然后 C 编译器把 C 代码翻译为本地代码。在有些情况下，这样做会显著提升性能。PL/SQL 编译器类型是由 PLSQL_CODE_TYPE 参数决定的，后者可以被设置为 INTERPRETED（默认值）或者 NATIVE。

目　录