

THE EXPERT'S VOICE® IN ORACLE

Expert PL/SQL Practices

for Oracle Developers and DBAs

*DEEP INSIGHTS INTO DATABASE
PROGRAMMING WITH PL/SQL*

John Beresniewicz, Adrian Billington, Martin Büchi, Melanie Caffrey, Ron Crisco, Lewis Cunningham, Dominic Delmolino, Sue Harper, Torben Holm, Connor McDonald, Arup Nanda, Stephan Petit, Michael Rosenblum, Robyn Sands, and Riyaj Shamsudeen

Apress®

Expert PL/SQL Practices

for Oracle Developers and DBAs



**John Beresniewicz, Adrian Billington, Martin Büchi,
Melanie Caffrey, Ron Crisco, Lewis Cunningham,
Dominic Delmolino, Sue Harper, Torben Holm,
Connor McDonald, Arup Nanda, Stephan Petit,
Michael Rosenblum, Robyn Sands, Riyaj Shamsudeen**

Apress®

Expert PL/SQL Practices: for Oracle Developers and DBAs

Copyright © 2011 by John Beresiewicz, Adrian Billington, Martin Büchi, Melanie Caffrey, Ron Crisco, Lewis Cunningham, Dominic Delmolino, Sue Harper, Torben Holm, Connor McDonald, Arup Nanda, Stephan Petit, Michael Rosenblum, Robyn Sands, Riyaj Shamsudeen

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN 978-1-4302-3485-2

ISBN 978-1-4302-3486-9 (eBook)

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

President and Publisher: Paul Manning

Lead Editor: Jonathan Gennick

Technical Reviewers: Chris Beck, Mike Gangler, Toon Koppelaars

Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Frank Pohlmann, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh

Coordinating Editor: Corbin Collins

Copy Editor: Mary Behr

Production Support: Patrick Cunningham

Indexer: SPI Global

Artist: April Milne

Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media, LLC., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales—eBook Licensing web page at www.apress.com/bulk-sales.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

Contents at a Glance

About the Authors.....	xvii
About the Technical Reviewers	xx
Introduction	xxi
■ Chapter 1: Do Not Use.....	1
■ Chapter 2: Dynamic SQL: Handling the Unknown	19
■ Chapter 3: PL/SQL and Parallel Processing.....	45
■ Chapter 4: Warnings and Conditional Compilation	71
■ Chapter 5: PL/SQL Unit Testing.....	97
■ Chapter 6: Bulk SQL Operations.....	121
■ Chapter 7: Know Your Code.....	171
■ Chapter 8: Contract-Oriented Programming.....	213
■ Chapter 9: PL/SQL from SQL	235
■ Chapter 10: Choosing the Right Cursor	291
■ Chapter 11: PL/SQL Programming in the Large.....	313
■ Chapter 12: Evolutionary Data Modeling	367
■ Chapter 13: Profiling for Performance	395
■ Chapter 14: Coding Conventions and Error Handling	425
■ Chapter 15: Dependencies and Invalidations	445
Index	475

Contents

About the Authors	xvii
About the Technical Reviewers	xx
Introduction	xxi
Chapter 1: Do Not Use	1
Row-by-Row Processing	1
Nested Row-by-Row Processing	4
Lookup Queries	5
Excessive Access to DUAL	8
Arithmetics with Date	8
Access to Sequences.....	9
Populating Master-Detail Rows	10
Excessive Function Calls	10
Unnecessary Function Execution.....	11
Costly Function Calls	13
Database Link Calls	16
Excessive Use of Triggers.....	17
Excessive Commits.....	17
Excessive Parsing.....	17
Summary	18

■ Chapter 2: Dynamic SQL: Handling the Unknown	19
The Hero	20
Native Dynamic SQL	21
Dynamic Cursors	24
DBMS_SQL	27
Sample of Dynamic Thinking	28
Security Issues	33
Performance and Resource Utilization	37
Anti-Patterns	38
Comparing Implementations of Dynamic SQL	39
Object Dependencies	41
Negative Effects	41
Positive Effects	42
Summary	43
■ Chapter 3: PL/SQL and Parallel Processing	45
Why Parallel Processing?	45
Laws Affecting Parallel Processing	46
The Rise of Big Data	47
Parallel vs. Distributed Processing	47
Parallel Hardware Architectures	47
Identify Your Goals	48
Speedup	49
Scaling Up	49
Degree of Parallelism	49
Candidate Workloads for Parallel Processing	50
Parallelism and OLTP	50
Parallelism and Non-OLTP Workloads	50

The MapReduce Programming Model	51
Before Looking to PL/SQL	51
Processes Available for Parallel Activities	52
Using Parallel Execution Servers for MapReduce	52
Pipelined Table Functions	52
Guidance	69
Parallel Pipelined Table Functions Summary	69
Summary	69
Chapter 4: Warnings and Conditional Compilation	71
PL/SQL Warnings	71
Basics	71
Using Warnings	73
Promoting Warnings to Errors	76
Ignoring Warnings	78
Compilation and Warnings	80
Final Words on Warnings	82
Conditional Compilation	83
Basics	83
What Part of the Code is Running?	86
Benefits of Preprocessing Code	87
Invalidations	89
Controlling Compilation	91
Inquiry Variables	93
Final Words on Conditional Compilation	94
Summary	96

Chapter 5: PL/SQL Unit Testing.....	97
Why Test Your Code?	97
What Is Unit Testing?	98
Debugging or Testing?	98
When Should You Build Tests?	98
Tools for Building Unit Tests	99
utPLSQL: Working with Command Line Code	99
Quest Code Tester for Oracle.....	99
Oracle SQL Developer	100
Preparing and Maintaining the Unit Test Environment	100
Creating the Unit Test Repository	101
Maintaining the Unit Test Repository.....	102
Importing Tests.....	103
Building Unit Tests	103
Using the Unit Test Wizard	104
Creating the First Implementation	104
Adding Startup and Teardown Processes.....	105
Collecting Code Coverage Statistics	106
Specifying Parameters	106
Adding Process Validation	107
Saving Tests	108
Debugging and Running Tests.....	108
Broadening the Scope of the Tests.....	109
Creating Lookup Values	109
Seeding Test Implementations	111
Creating a Dynamic Query	112

Supporting Unit Test Features	113
Running Reports	113
Creating a Library of Components	114
Exporting, Importing, and Synchronizing Tests	117
Building Suites.....	119
Running Tests from the Command Line	119
Summary	120
■ Chapter 6: Bulk SQL Operations.....	121
The Hardware Store.....	121
Setting for the Examples in this Chapter	122
Bulk Operations in PL/SQL.....	123
Getting Started with BULK Fetch	124
Three Collection-Style Datatypes	127
Why should I bother?.....	128
Monitoring Bulk Collect Overheads	131
Refactoring Code to Use Bulk Collect.....	135
Bulk Binding	143
Getting Started with Bulk Bind	144
Measuring Bulk Binding Performance.....	145
Monitoring Memory Usage	148
Improvements in 11g.....	150
Error Handling with Bulk Bind	151
SAVE EXCEPTIONS with Batches	155
LOG ERRORS Clause	156
Robust Bulk Bind	157
A Justification for Massive Collections.....	162
The Real Benefit: Client Bulk Processing.....	164
Summary	168

Chapter 7: Know Your Code	171
What This Chapter Will (and Will Not) Cover	171
Automated Code Analysis	173
Static Analysis	173
Dynamic Analysis	174
When to Analyze?	174
Performing Static Analysis	175
The Data Dictionary	176
PL/SCOPE	183
Performing Dynamic Analysis.....	197
DBMS_PROFILER and DBMS_TRACE.....	197
DBMS_HPROF	206
Summary	212
Chapter 8: Contract-Oriented Programming	213
Design by Contract	213
Software Contracts.....	213
Basic Contract Elements	214
Assertions	215
References	215
Implementing PL/SQL Contracts.....	216
Basic ASSERT Procedure.....	216
Standard Package-Local ASSERT.....	218
Enforcing Contracts Using ASSERT	220
An Additional Improvement	222
Contract-Oriented Function Prototype.....	223
Example: Testing Odd and Even Integers	224

Useful Contract Patterns	226
Not-NULL IN / Not-NULL OUT.....	227
FUNCTION RETURN Not-NULL.....	227
FUNCTION RETURN BOOLEAN Not-NULL.....	228
Check Functions: RETURN TRUE OR ASSERTFAIL.....	228
Principles For Bug-Free Code	229
Assert Preconditions Rigorously.....	229
Modularize Ruthlessly.....	230
Adopt Function-Based Interfaces.....	231
Crash on ASSERTFAIL.....	231
Regression Test Your Postconditions.....	231
Avoid Correctness-Performance Tradeoffs.....	232
Oracle 11g Optimized Compilation.....	233
Summary	233
■ Chapter 9: PL/SQL from SQL	235
The Cost of Using PL/SQL Functions in SQL	235
Context-Switching.....	236
Executions.....	241
Suboptimal Data Access.....	245
Optimizer Difficulties.....	249
The Read-Consistency Trap.....	253
Other Issues.....	255
Reducing the Cost of PL/SQL Functions	256
A Sense of Perspective.....	256
Using SQL Alternatives.....	257
Reducing Executions.....	264

Assisting the CBO	273
Tuning PL/SQL	285
Summary	289
Chapter 10: Choosing the Right Cursor	291
Explicit Cursors	292
The Anatomy of an Explicit Cursor.....	293
Explicit Cursors and Bulk Processing.....	294
REF Cursors in Brief.....	295
Implicit Cursors.....	296
The Anatomy of an Implicit Cursor	297
The Implicit Cursor and the Extra Fetch Theory	299
Static REF Cursors	301
Cursor Variable Restrictions Laundry List.....	303
Your Client and REF Cursors.....	304
A Few Words about Parsing	305
Dynamic REF Cursors	307
Example and Best Use.....	308
The Threat of SQL Injection	309
Describing REF Cursor Columns.....	311
Summary	312
Chapter 11: PL/SQL Programming in the Large.....	313
The Database as PL/SQL-Based Application Server	313
Case Study: The Avaloq Banking System	313
Strengths of Business Logic in the Database with PL/SQL.....	315
Limits of the Database as PL/SQL-Based Application Server.....	318
Soft Factors	318
Requirements of Programming in the Large.....	319

Uniformity through Conventions	319
Abbreviations	320
Pre- and Suffixes for PL/SQL Identifiers.....	323
Modularization of Code and Data.....	326
Packages and Associated Tables as Modules.....	327
Modules Containing Multiple Packages or Submodules.....	332
Schemas as Modules.....	336
Modularization within Schemas	339
Modularization with Schemas vs. within Schemas.....	342
Object-Oriented Programming with PL/SQL	343
Object-Oriented Programming with User-Defined Types	344
Object-Oriented Programming with PL/SQL Records	348
Assessment	356
Memory Management.....	356
Measuring Memory Usage.....	356
Collections	362
Summary	365
■ Chapter 12: Evolutionary Data Modeling	367
Lessons from Two Decades of System Development.....	368
The Database and Agile Development.....	369
Evolutionary Data Modeling	370
Refactoring the Database	372
Creating an Access Layer via PL/SQL.....	376
The Agile Manifesto	390
Using PL/SQL with Evolutionary Data Modeling.....	392
Define the Interface.....	392
Think Extensible	392

Test Driven Development	393
Use Schemas and Users Wisely	394
Summary	394
■ Chapter 13: Profiling for Performance	395
What Is Performance?	396
Functional Requirements.....	396
Response Time	397
Throughput	397
Resource Utilization.....	397
Performance Is a Feature	398
What Is Profiling?.....	399
Sequence Diagrams	400
The Magic of Profiles.....	400
Benefits of Profiling.....	401
Instrumentation	402
Why is This Program Slow?.....	402
Measurement Intrusion	404
Identification	404
Conditional Compilation.....	409
Built-in Profilers.....	409
Extended SQL Trace Data (Event 10046)	410
Instrumentation Library for Oracle (ILO)	410
Problem Diagnosis.....	414
Method R	414
ILO Example.....	416
Profiling Example.....	419
Summary	422

Chapter 14: Coding Conventions and Error Handling	425
Why coding conventions?	425
Formatting	427
Case	427
Comments	427
Indentation	428
Dynamic Code	431
Packages	432
Stored Procedures	433
Naming	433
Parameters	434
Calls	435
Local Variables	435
Constants	435
Types	436
Global Variables	436
Local Procedures and Functions	436
Procedure Metadata	436
Functions	437
Error Handling	437
Error Trapping	438
Error Reporting	438
Error Recovery	440
Test First. Display Second.	441
Summary	441

Chapter 15: Dependencies and Invalidations	445
Dependency Chains	445
Shortening Dependency Chains.....	452
Datatype Reference	457
View for Table Alterations.....	458
Adding Components into Packages	462
Synonyms in Dependency Chains.....	466
Resource Locking	466
Forcing Dependency in Triggers.....	467
Creating Triggers Disabled Initially.....	471
Summary	473
 Index	 475

About the Authors

■ **John Beresniewicz** is a consulting member of the technical staff at Oracle headquarters in Redwood Shores, CA. He joined Oracle in 2002 to work on Enterprise Manager in the database performance area and has played significant role in the design of Diagnostic and Tuning packs, Real Application Testing, Support Workbench, and Exadata. He has been a frequent speaker on database performance and PL/SQL programming over many years at Oracle Openworld and other conferences. He is co-author of *Oracle Built-in Packages* (O'Reilly & Associates, 1998) with Steven Feuerstein and is a founding member of the OakTable network.

■ **Adrian Billington** is a consultant in application design, development, and performance tuning who has been working with Oracle databases since 1999. He is the man behind www.oracle-developer.net, a web site full of SQL and PL/SQL features, utilities, and techniques for Oracle developers. Adrian is also an Oracle ACE and a member of the OakTable Network. He lives in the UK with his wife Anji and three children Georgia, Oliver, and Isabella.

■ **Martin Büchi** has worked since 2004 as Lead Software Architect for Avaloq, a provider of a standardized banking software built on the Oracle RDBMS with 11 million lines of PL/SQL code. Together with two colleagues he defines the system architecture and reviews the designs and code of 170 full-time PL/SQL developers, looking for simplicity, efficiency, and robustness. Martin regularly speaks at Oracle conferences. In 2009 he was named PL/SQL Developer of the Year by *Oracle Magazine*. Before getting into the Oracle database, Martin worked in object-oriented systems, formal methods, and approximate record matching. He holds an MSc from the Swiss Federal Institute of Technology and a PhD from the Turku Center for Computer Science in Finland. In his spare time, Martin enjoys various outdoor sports with his family.

■ **Melanie Caffrey** is a Senior Development Manager for Oracle Corporation, providing front-end and back-end Oracle solutions for the business needs of various clients. She is co-author of several technical publications including *Oracle Web Application Programming for PL/SQL Developers*, *Oracle DBA Interactive Workbook*, and *Oracle Database Administration: The Complete Video Course*, all published by Prentice Hall. She has instructed students in Columbia University's Computer Technology and Applications program in New York City, teaching advanced Oracle database administration and PL/SQL development. She is a frequent Oracle conference speaker.

■ **Ron Crisco** has been a software designer, developer, and project leader for 28 years and has worked with Oracle databases for 21 years. He works at Method R Corporation, designing and developing software, managing software products (like Method R Profiler, MR Tools, and MR Trace), consulting, and teaching courses. His specialty is simplifying complex work, which is especially valuable in helping the people around him accomplish extraordinary things.

■ **Lewis Cunningham** has been working in IT for over 20 years and has worked with Oracle databases since 1993. His specialties are application design, database design, and coding of high volume, VLDB databases. He is currently a Senior Database Architect at a financial services company in St Petersburg, FL working on very large, high transaction rate analytical databases and applications. He spends an inordinate amount of time keeping up with current technology and trends and speaking at user groups and doing webinars. Lewis is also an Oracle ACE Director and Oracle Certified Professional. He has written several articles for the Oracle Technology Network and maintains an Oracle technology blog at <http://it.toolbox.com/blogs/oracle-guide>. Lewis has written two books: *EnterpriseDB: The Definitive Reference* (Rampant Techpress, 2007) and *SQL DML: The SQL Starter Series* (CreateSpace, 2008). He lives in Florida with his wife and two sons. You can contact him at lewisc@databasewisdom.com.

■ **Dominic Delmolino** is the Lead Oracle and Database Technologist for Agilex Technologies, a consulting firm specializing in assisting government and private enterprises to realize the value of their information. Dominic has over 24 years of database experience, including more than 20 years as an Oracle Database Engineering and Development professional. He is a member of the Oak Table Network and regularly presents at conferences, seminars, and user group meetings in Europe and the US. He also maintains www.oraclemusings.com, a site focused on database coding and design practices related to database application development. Dominic holds a Bachelor of Science degree in computer science from Cornell University, Ithaca, NY.

■ **Sue Harper** is a Product Manager for Oracle SQL Developer and SQL Developer Data Modeler in the Database Development Tools group. She has been at Oracle since 1992 and is currently based in London. Sue is a regular contributor to magazines, maintains a technical blog, and speaks at many conferences around the world. She has authored the technical book *Oracle SQL Developer 2.1* (Packt Publishing, 2009) When not at work, Sue is a keen walker and photographer. Sue takes time out to work with a charity in the slums of New Delhi, where she works with the women and children.

■ **Torben Holm** has been in the computer business as a developer since 1987. He has been working with Oracle since 1992; his first four years as system analyst and application developer (Oracle 7 and Forms 4.0/Reports 2.0 and DBA), then two years as developer (Oracle6/7, Forms 3.0 and RPT, and DBA). He spent several years working for Oracle Denmark in the Premium Services group as a Senior Principal Consultant performing application development and DBA tasks. He also worked as an instructor in PL/SQL, SQL, and DBA courses. Torben now works for Miracle A/S (www.miracleas.dk) as a consultant with a focus in application development (PLSQL, mod_plsql, Forms, ADF) and database administration. He has been at Miracle A/S 10 years. He is an Oracle Certified Developer and a member of OakTable.net.

■ **Connor McDonald** has worked with Oracle since the early 1990s, cutting his teeth on Oracle versions 6.0.36 and 7.0.12. Over the past 11 years, Connor has worked with systems in Australia, the U.K., Southeast Asia, Western Europe, and the United States. He has come to realize that although the systems and methodologies around the world are very diverse, there tend to be two common themes in the development of systems running on Oracle: either to steer away from the Oracle-specific functions or to use them in a haphazard or less-than-optimal fashion. It was this observation that led to the creation of a personal hints and tips web site (www.oracledba.co.uk) and more presenting on the Oracle speaker circuit in an endeavor to improve the perception and usage of PL/SQL in the industry.

■ **Arup Nanda** has been an Oracle DBA since 1993, which has exposed him to all facets of database administration, from modeling to disaster recovery. He currently leads the global DBA team at Starwood Hotels, the parent of chains such as Sheraton and Westin, in White Plains, NY. He serves as a contributing editor of *SELECT Journal*, the publication of Independent Oracle Users Group (IOUG); speaks at many Oracle Technology events such as Oracle World and local user groups such as New York Oracle User Group; and has written many articles for both print publications such as *Oracle Magazine* and online publications such as *Oracle Technology Network*. Arup has coauthored two books: *Oracle Privacy Security Auditing* (Rampant, 2003) and *Oracle PL/SQL for DBAs* (O'Reilly, 2005). Recognizing his professional accomplishments and contributions to user community, Oracle chose him as the DBA of the Year in 2003. Arup lives in Danbury, Connecticut, with his wife Anindita and son Anish. He can be reached at arup@proligence.com.

■ **Stephan Petit** began his career in 1995 at CERN, the European Laboratory for Particle Physics, located in Geneva, Switzerland. He is now in charge of a team of software engineers and students delivering applications and tools to the laboratory and beyond. One of these tools is the Engineering and Equipment Data Management System, also known as the CERN EDMS. Projects like CERN's Large Hadron Collider (LHC) have a lifetime of 40 years or more. The EDMS is the digital engineering memory of the laboratory. More than a million documents relating to more than a million pieces of equipment are stored in the EDMS, which is also used as CERN's Product Lifecycle Management (PLM) and Asset Tracking system. EDMS is based almost entirely on PL/SQL and is intended to have a lifetime at least as long as the LHC.

Stephan and his team have been polishing coding conventions and best practices in PL/SQL in order to meet their very interesting mix of challenges: maintainability over decades, reliability, efficient error handling, scalability, and reusability of the modules. These challenges are compounded by the frequent rotation of team members, most of whom are students only temporarily at CERN. The oldest piece of code was written in 1995 and is still in use — with success! Apart from polishing PL/SQL, Stephan also enjoys being on stage from time to time as rock band singer at the CERN's rock & roll summer festival and as actor in various plays.

■ **Michael Rosenblum** is a Software Architect/Development DBA at Dulcian, Inc. where he is responsible for system tuning and application architecture. Michael supports Dulcian developers by writing complex PL/SQL routines and researching new features. He is the co-author of *PL/SQL for Dummies* (Wiley Press, 2006) and author of a number of database-related articles (IOUG Select Journal, ODTUG Tech Journal). Michael is an Oracle ACE, a frequent presenter at various regional and national Oracle user group conferences (Oracle OpenWorld, ODTUG, IOUG Collaborate, RMOUG, NYOUG, etc), and winner of the ODTUG Kaleidoscope 2009 Best Speaker Award. In his native Ukraine, he received the scholarship of the President of Ukraine, a Master of Science degree in Information Systems, and a diploma with honors from the Kiev National University of Economics.

■ **Robyn Sands** is a Software Engineer for Cisco Systems, where she designs and develops embedded Oracle database products for Cisco customers. She has been working with Oracle since 1996 and has extensive experience in application development, large system implementations, and performance measurement. Robyn began her work career in industrial and quality engineering, and she has combined her prior education and experience with her love of data by searching for new ways to build database systems with consistent performance and minimal maintenance requirements. She is a member of the OakTable Network and a coauthor of two books on Oracle: *Expert Oracle Practices* and *Pro Oracle SQL* (both Apress, 2010). Robyn occasionally posts random blog entries at <http://adhdoddbba.blogspot.com>.

■ **Riyaj Shamsudeen** is the Principal Database Administrator and President of OraInternals, a performance/recovery/EBS11i consulting company. He specializes in real application clusters, performance tuning, and database internals. He also frequently blogs about these technology areas in his blog <http://orainternals.wordpress.com>. He is also a regular presenter in many international conferences such as HOTSOS, COLLABORATE, RMOUG, SIOUG, UKOUG, etc. He is a proud member of OakTable Network. He has over 16 years of experience using Oracle technology products and over 15 years as an Oracle/Oracle applications database administrator.

About the Technical Reviewers

■ **Chris Beck** has a degree in computer science from Rutgers University and has been working with multiple DBMS's for more than 20 years. He has spent the last 16 years as an Oracle employee where he is currently a Master Principal Technologist focusing on core database technologies. He is a co-inventor of two US Patents on software methodologies that were the basis for what is now known as Oracle Application Express. Chris has reviewed other Oracle books including *Expert One-On-One* (Peer Information Inc., 2001) and *Expert Oracle Database Architecture* (Apress, 2005), both by Tom Kyte and is himself the co-author of two books, *Beginning Oracle Programming* (Wrox Press, 2005) and *Mastering Oracle PL/SQL* (Apress, 2005). He resides in Northern Virginia with his wife Marta and four children; when not spending time with them, he can usually be found wasting time playing video games or watching Series A football.

■ **Mike Gangler** is a Database Specialist and Infrastructure Architect with over 30 years of data processing industry experience, primarily as a Database Technical Lead, Systems Analyst, and DBA on large corporate Information Systems projects. As a past President of the Detroit Oracle Users Group and Ann Arbor Oracle User groups, and charter member of the Board of Directors of the International Oracle Users Group, Mike has attained worldwide recognition as an accomplished and Certified Oracle DBA and relational database expert.

■ **Toon Koppelaars** is a long-time Oracle technology user, having used the Oracle database and tools software since 1987 (Oracle version 4). During this time, he has been involved in application development and database administration. He is an ACE Director (database development) and frequent speaker at Oracle-related events. He's also the declarer of "The Helsinki Declaration (IT version)" which describes a database-centric approach to modern application development. Together with Lex de Haan, Toon has co-authored *Applied Mathematics for Database Professionals* (Apress, 2007).

Introduction

Rarely do I take the opportunity to introduce a book that I've helped create. Normally I am content with my place in the background where book editors rightfully belong. I make an exception this time because the content in this book brings back so many memories from own experiences as a developer in days gone by.

Expert PL/SQL Practices is about wielding PL/SQL effectively. It's not a book about syntax. It's a book about how to apply syntax and features along with good development practices to create applications that are reliable and scalable—and maintainable over the long term.

With any tool, one of the first things to know is when to wield it. Riyaj Shamsudeen deftly tackles the question of when to use PL/SQL in his opening chapter *Do Not Use!* I put that chapter first in the book because of personal experience: My best-ever performance optimization success came in the late 1990s when I replaced a stack of procedural code on a client PC with a single SQL statement, taking a job from over 36 hours to just a couple of minutes. PL/SQL was not the culprit, but the lesson I learned then is that a set-based approach—when one is possible—is often preferable to writing procedural code.

Michael Rosenblum follows with an excellent chapter on dynamic SQL, showing how to write code when you don't know the SQL statements until runtime. He reminded me of a time at Dow Chemical in the early 1990s when I wrote a data-loading application for a medical record system using Rdb's Extended Dynamic Cursor feature set. I still remember that as one of the most fun applications that I ever developed.

Dominic Delmolino tackles parallel processing with PL/SQL. He covers the benefits that you can achieve as well as the candidate workloads. Just be careful, okay? One of my biggest-ever blunders as a DBA was when I once unthinkingly set a degree of parallelism on a key application table in order to make a single report run faster. It was as if the Enter key was connected to my telephone, because my phone rang within about a minute of my change. The manager on the other end of the line was most displeased. Needless to say, I decided then that implementing parallelism deserved just a tad bit more thought than I had been giving it. Dominic's chapter will help you avoid such embarrassment.

Several chapters in the book cover code hygiene and good programming practices. Stephan Petit presents a set of useful naming and coding conventions. Torben Holm covers PL/SQL Warnings and conditional compilation. Lewis Cunningham presents a thought-provoking chapter on code analysis and the importance of truly understanding the code that you write and how it gets used. Robyn Sands helps you think about flexibility and good design in her chapter on evolutionary data modeling. Melanie Caffrey tours the various cursor types available, helping you to make the right choice of cursor for any given situation.

Other chapters relate to debugging and troubleshooting. Sue Harper covers PL/SQL unit testing, especially the supporting feature set that is now built into SQL Developer. (I remember writing unit test scripts on paper back in the day). Save yourself the embarrassment of regression bugs. Automated unit tests make it easy and convenient to verify that you've not broken two new things while fixing one.

John Beresniewicz follows with a chapter on contract-oriented programming. A key part of John's approach is the use of asserts to validate conditions that should be true at various points within your code. I first learned of the assert technique while doing PowerBuilder programming back in the Stone Age. I've always been happy to see John promote the technique in relation to PL/SQL.

Arup Nanda helps you get control over dependencies and invalidations. Dependency issues can be a source of seemingly random, difficult-to-repeat application errors. Arup shows how to get control over what must inevitably happen, so that you aren't caught out by unexpected errors.

We could hardly leave performance and scalability out of the picture. Ron Crisco talks about profiling your code to find the greatest opportunities for optimization. Adrian Billington talks about the performance aspects of invoking PL/SQL from within SQL statements. Connor McDonald covers the tremendous performance advantages available from bulk SQL operations.

An unusual aspect of scalability not often thought about is that of application size and the number of developers. Is PL/SQL suited for large-scale development involving dozens, perhaps hundreds of programmers? Martin Büchi shows that PL/SQL is very much up to the task in his chapter on PL/SQL programming in the large by recounting his success with an 11-million line application maintained by over 170 developers.

You can probably tell that I'm excited about this book. The authors are top notch. Each has written on an aspect of PL/SQL that they are passionate and especially knowledgeable about. If you're past the point of learning syntax, then sit down, read this book, and step up your game in delivering applications using the full power of PL/SQL and Oracle Database.

Jonathan Gennick
Assistant Editorial Director, Apress

CHAPTER 1

Do Not Use

By Riyaj Shamsudeen

Congratulations on buying this book. PL/SQL is a great tool to have in your toolbox; however, you should understand that use of PL/SQL is not suitable for all scenarios. This chapter will teach you when to code your application in PL/SQL, how to write scalable code, and, more importantly, when not to code programs in PL/SQL. Abuse of some PL/SQL constructs leads to unscalable code. In this chapter, I will review various cases in which the misuse of PL/SQL was unsuitable and lead to an unscalable application.

PL/SQL AND SQL

SQL is a set processing language and SQL statements scale better if the statements are written with set level thinking in mind. PL/SQL is a procedural language and SQL statements can be embedded in PL/SQL code.

SQL statements are executed in the SQL executor (more commonly known as the SQL engine). PL/SQL code is executed by the PL/SQL engine. The power of PL/SQL emanates from the ability to combine the procedural abilities of PL/SQL with the set processing abilities of SQL.

Row-by-Row Processing

In a typical row-by-row processing program, code opens a cursor, loops through the rows retrieved from the cursor, and processes those rows. This type of loop-based processing construct is highly discouraged as it leads to unscalable code. Listing 1-1 shows an example of a program using the construct.

Listing 1-1. Row-by-Row Processing

```
DECLARE
  CURSOR c1 IS
    SELECT prod_id, cust_id, time_id, amount_sold
    FROM sales
```



```

    WHERE amount_sold > 100;
    c1_rec c1%rowtype;
    l_cust_first_name customers.cust_first_name%TYPE;
    l_cust_last_name customers.cust_last_name%TYPE;
BEGIN
    FOR c1_rec IN c1
    LOOP
        -- Query customer details
        SELECT cust_first_name, cust_last_name
        INTO l_cust_first_name, l_cust_last_name
        FROM customers
        WHERE cust_id=c1_rec.cust_id;
        --
        -- Insert in to target table
        --
        INSERT INTO top_sales_customers (
            prod_id, cust_id, time_id, cust_first_name, cust_last_name, amount_sold
        )
        VALUES
        (
            c1_rec.prod_id,
            c1_rec.cust_id,
            c1_rec.time_id,
            l_cust_first_name,
            l_cust_last_name,
            c1_rec.amount_sold
        );
    END LOOP;
    COMMIT;
END;
/

```

PL/SQL procedure successfully completed.

Elapsed: 00:00:10.93

In Listing 1-1, the program declares a cursor `c1`, and opens the cursor implicitly using `cursor-for-loop` syntax. For each row retrieved from the cursor `c1`, the program queries the `customers` table to populate `first_name` and `last_name` to variables. A row is subsequently inserted in to the `top_sales_customers` table.

There is a problem with the coding practice exemplified in Listing 1-1. Even if the SQL statements called in the loop are highly optimized, program execution can consume a huge amount of time. Imagine that the SQL statement querying the `customers` table consumes an elapsed time of 0.1 seconds, and that the `INSERT` statement consumes an elapsed time of 0.1 seconds, giving a total elapsed time of 0.2 seconds per loop execution. If cursor `c1` retrieves 100,000 rows, then the total elapsed time for this program will be 100,000 multiplied by 0.2 seconds: 20,000 seconds or 5.5 hours approximately. Optimizing this program construct is not easy. Tom Kyte termed this type of processing as *slow-by-slow processing* for obvious reasons.

■ **Note** Examples in this chapter use SH schema, one of the example schemas supplied by Oracle Corporation. To install the example schemas, Oracle-provided software can be used. You can download it from http://download.oracle.com/otn/solaris/oracle11g/R2/solaris.sparc64_11gR2_examples.zip for 11gR2 Solaris platform. Refer to the Readme document in the unzipped software directories for installation instructions. Zip files for other platforms and versions are also available from Oracle's web site.

There is another inherent issue with the code in Listing 1-1. SQL statements are called from PL/SQL in a loop, so the execution will switch back and forth between the PL/SQL engine and the SQL engine. This switch between two environments is known as a *context switch*. Context switches increase elapsed time of your programs and introduce unnecessary CPU overhead. You should reduce the number of context switches by eliminating or reducing the switching between these two environments.

You should generally avoid row-by-row processing. Better coding practice would be to convert the program from Listing 1-1 into a SQL statement. Listing 1-2 rewrites the code, avoiding PL/SQL entirely.

Listing 1-2. *Row-by-Row Processing Rewritten*

```
--
-- Insert in to target table
--
INSERT
INTO top_sales_customers
(
  prod_id,
  cust_id,
  time_id,
  cust_first_name,
  cust_last_name,
  amount_sold
)
SELECT s.prod_id,
       s.cust_id,
       s.time_id,
       c.cust_first_name,
       c.cust_last_name,
       s.amount_sold
FROM sales s,
     customers c
WHERE s.cust_id = c.cust_id and
s.amount_sold > 100;

135669 rows created.

Elapsed: 00:00:00.26
```

The code in Listing 1-2, in addition to resolving the shortcomings of the row-by-row processing, has a few more advantages. Parallel execution can be used to tune the rewritten SQL statement. With the use of multiple parallel execution processes, you can decrease the elapsed time of execution sharply. Furthermore, code becomes concise and readable.

■ **Note** If you rewrite the PL/SQL loop code to a join, you need to consider duplicates. If there are duplicates in the customers table for the same cust_id columns, then the rewritten SQL statement will retrieve more rows than intended. However, in this specific example, there is a primary key on cust_id column in the customers table, so there is no danger of duplicates with an equality predicate on cust_id column.

Nested Row-by-Row Processing

You can nest cursors in PL/SQL language. It is a common coding practice to retrieve values from one cursor, feed those values to another cursor, feed the values from second level cursor to third level cursor, and so on. But the performance issues with a loop-based code increase if the cursors are deeply nested. The number of SQL executions increases sharply due to nesting of cursors, leading to a longer program runtime.

In Listing 1-3, cursors c1, c2, and c3 are nested. Cursor c1 is the top level cursor and retrieves rows from the table t1; cursor c2 is opened, passing the values from cursor c1; cursor c3 is opened, passing the values from cursor c2. An UPDATE statement is executed for every row retrieved from cursor c3. Even if the UPDATE statement is optimized to execute in 0.01 seconds, performance of the program suffers due to the deeply nested cursor. Say that cursors c1, c2, and c3 retrieve 20, 50, and 100 rows, respectively. The code then loops through 100,000 rows, and the total elapsed time of the program exceeds 1,000 seconds. Tuning this type of program usually leads to a complete rewrite.

Listing 1-3. Row-by-Row Processing with Nested Cursors

```

DECLARE
  CURSOR c1 AS
    SELECT n1 FROM t1;
  CURSOR c2 (p_n1) AS
    SELECT n1, n2 FROM t2 WHERE n1=p_n1;
  CURSOR c3 (p_n1, p_n2) AS
    SELECT text FROM t3 WHERE n1=p_n1 AND n2=p_n2;
BEGIN
  FOR c1_rec IN c1
  LOOP
    FOR c2_rec IN c2 (c1_rec.n1)
    LOOP
      FOR c3_rec IN c3(c2_rec.n1, c2_rec.n2)
      LOOP
        -- execute some sql here;
        UPDATE ... SET ..where n1=c3_rec.n1 AND n2=c3_rec.n2;
      EXCEPTION
        WHEN no_data_found THEN

```

```

        INSERT into.. END;
    END LOOP;
END LOOP;
END LOOP;
COMMIT;
END;
/

```

Another problem with the code in the Listing 1-3 is that an `UPDATE` statement is executed. If the `UPDATE` statement results in a `no_data_found` exception, then an `INSERT` statement is executed. It is possible to offload this type of processing from PL/SQL to the SQL engine using a `MERGE` statement.

Conceptually, the three loops in Listing 1-3 represent an equi-join between the tables `t1`, `t2`, and `t3`. In Listing 1-4, the logic is rewritten as a SQL statement with an alias of `t`. The combination of `UPDATE` and `INSERT` logic is replaced by a `MERGE` statement. `MERGE` syntax provides the ability to update a row if it exists and insert a row if it does not exist.

***Listing 1-4.** Row-by-Row Processing Rewritten Using MERGE Statement*

```

MERGE INTO fact1 USING
(SELECT DISTINCT c3.n1,c3.n2
 FROM t1, t2, t3
 WHERE t1.n1      = t2.n1
 AND t2.n1        = t3.n1
 AND t2.n2        = t3.n2
 ) t
ON (fact1.n1=t.n1 AND fact1.n2=t.n2)
WHEN matched THEN
    UPDATE SET .. WHEN NOT matched THEN
    INSERT .. ;
COMMIT;

```

Do not write code with deeply nested cursors in PL/SQL language. Review it to see if you can write such code in SQL instead.

Lookup Queries

Lookup queries are generally used to populate some variable or to perform data validation. Executing lookup queries in a loop causes performance issues.

In the Listing 1-5, the highlighted query retrieves the `country_name` using a lookup query. For every row from the cursor `c1`, a query to fetch the `country_name` is executed. As the number of rows retrieved from the cursor `c1` increases, executions of the lookup query also increases, leading to a poorly performing code.

***Listing 1-5.** Lookup Queries, a Modified Copy of Listing 1-1*

```

DECLARE
CURSOR c1 IS
    SELECT prod_id, cust_id, time_id, amount_sold
    FROM sales
    WHERE amount_sold > 100;

```

```

l_cust_first_name customers.cust_first_name%TYPE;
l_cust_last_name customers.cust_last_name%TYPE;
l_Country_id countries.country_id%TYPE;
l_country_name countries.country_name%TYPE;
BEGIN
FOR c1_rec IN c1
LOOP
-- Query customer details
SELECT cust_first_name, cust_last_name, country_id
INTO l_cust_first_name, l_cust_last_name, l_country_id
FROM customers
WHERE cust_id=c1_rec.cust_id;

-- Query to get country_name
SELECT country_name
INTO l_country_name
FROM countries WHERE country_id=l_country_id;
--
-- Insert in to target table
--
INSERT
INTO top_sales_customers
(
    prod_id, cust_id, time_id, cust_first_name,
    cust_last_name, amount_sold, country_name
)
VALUES
(
    c1_rec.prod_id, c1_rec.cust_id, c1_rec.time_id, l_cust_first_name,
    l_cust_last_name, c1_rec.amount_sold, l_country_name
);
END LOOP;
COMMIT;
END;
/
PL/SQL procedure successfully completed.
Elapsed: 00:00:16.18

```

The example in Listing 1-5 is simplistic. The lookup query for the `country_name` can be rewritten as a join in the main cursor `c1` itself. As a first step, you should modify the lookup query into a join. In a real world application, this type of rewrite is not always possible, though.

If you can't rewrite the code to reduce the executions of a lookup query, then you have another option. You can define an associative array to cache the results of the lookup query and reuse the array in later executions, thus effectively reducing the executions of the lookup query.

Listing 1-6 illustrates the array-caching technique. Instead of executing the query to retrieve the `country_name` for every row from the cursor `c1`, a key-value pair, (`country_id`, `country_name`) in this example) is stored in an associative array named `l_country_names`. An associative array is similar to an index in that any given value can be accessed using a key value.

Before executing the lookup query, an existence test is performed for an element matching the `country_id` key value using an `EXISTS` operator. If an element exists in the array, then the `country_name` is retrieved from that array without executing the lookup query. If not, then the lookup query is executed and a new element added to the array.

You should also understand that this technique is suitable for statements with few distinct values for the key. In this example, the number of executions of the lookup query will be *probably* much lower as the number of unique values of `country_id` column is lower. Using the example schema, the maximum number of executions for the lookup query will be 23 as there are only 23 distinct values for the `country_id` column.

Listing 1-6. *Lookup Queries with Associative Arrays*

```

DECLARE
  CURSOR c1
  IS
    SELECT prod_id, cust_id, time_id, amount_sold
    FROM sales WHERE amount_sold > 100;
    l_country_names country_names_type;
    l_country_id countries.country_id%TYPE;
    l_country_name countries.country_name%TYPE;
    l_cust_first_name customers.cust_first_name%TYPE;
    l_cust_last_name customers.cust_last_name%TYPE;
TYPE country_names_type IS
TABLE OF VARCHAR2(40) INDEX BY pls_integer;
l_country_names country_names_type;
BEGIN
  FOR c1_rec IN c1 LOOP
    -- Query customer details
    SELECT cust_first_name, cust_last_name, country_id
    INTO l_cust_first_name, l_cust_last_name, l_country_id
    FROM customers
    WHERE cust_id=c1_rec.cust_id;
    -- Check array first before executing a SQL statement

IF ( l_country_names.EXISTS(l_country_id)) THEN
  l_country_name := l_country_names(l_country_id);
ELSE
  SELECT country_name INTO l_country_name
  FROM countries
  WHERE country_id = l_country_id;
  -- Store in the array for further reuse
  l_country_names(l_country_id) := l_country_name;
END IF;
  --
  -- Insert in to target table
  --
  INSERT
  INTO top_sales_customers
  (
    prod_id, cust_id, time_id, cust_first_name,
    cust_last_name, amount_sold, country_name
  )

```