TechJournal

New York Oracle Users Group

First Quarter 2009

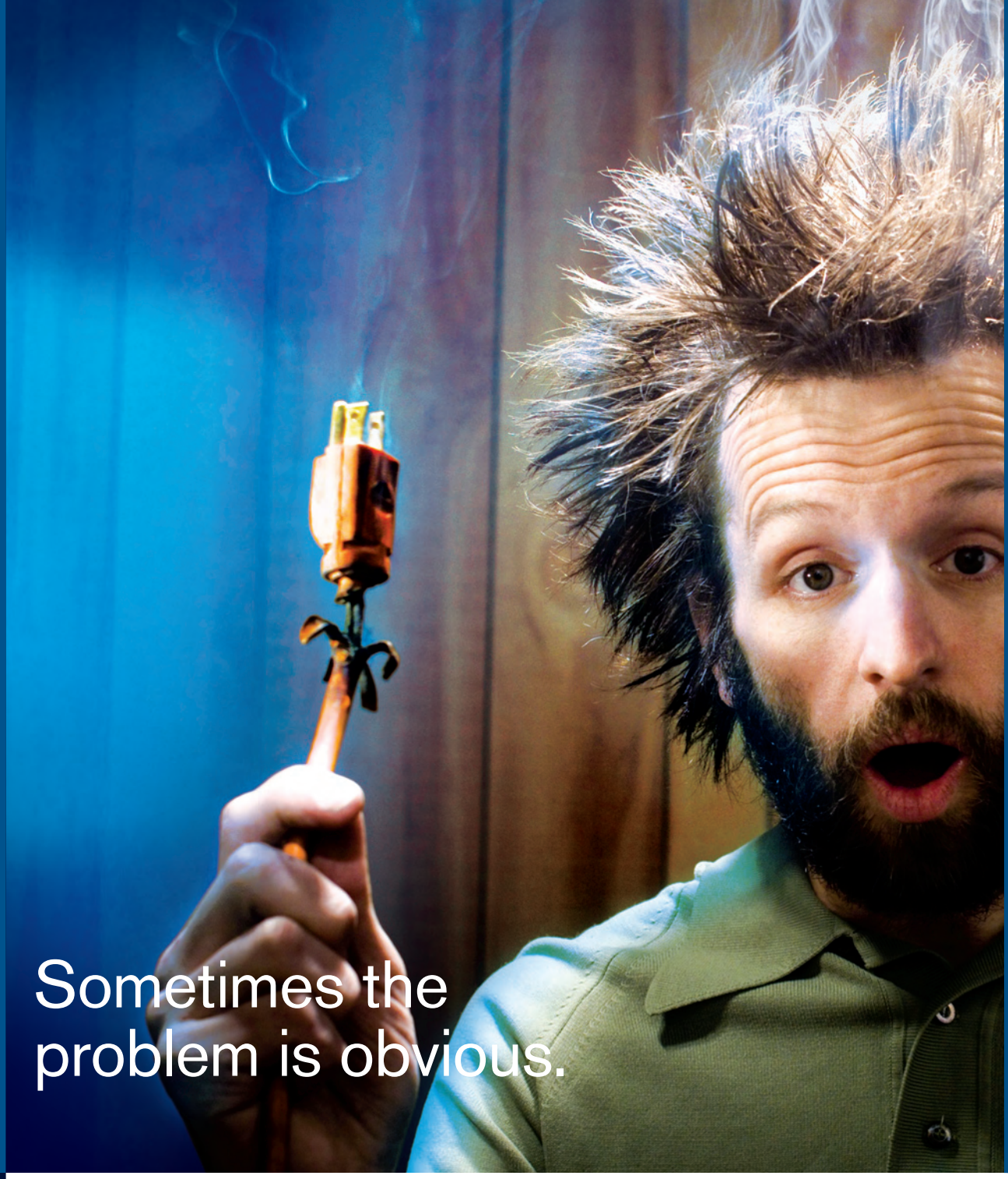
First Quarter General Meeting

Thursday, March 19, 2008
St. John's University – Manhattan Campus
101 Murray Street

Sponsored by:
Confio Software and TUSC

Free for Paid 2009 Members
Don't Miss It!

In This Issue –



Usually, it's harder to pinpoint.

Amazing what you can accomplish once you have the information you need.

When the source of a database-driven application slowdown isn't immediately obvious, try a tool that can get you up to speed. One that pinpoints database bottlenecks and calculates application wait time ***at each step***. Confio lets you unravel slowdowns at the database level with no installed agents. And solving problems where they exist costs a ***tenth*** of working around it by adding new server CPU's. Now that's a vision that can take you places.

A smarter solution makes everyone look brilliant.

CONFIO SOFTWARE

Download your FREE trial of Confio Ignite™ at www.confio.com/obvious

Download our FREE whitepaper by visiting www.oraclewhitepapers.com/listc/confio

NYOUG Officers / Chairpersons

ELECTED OFFICERS - 2009

President
Michael Olin
president@nyoug.org

Vice President
Mike La Magna
vicepresident@nyoug.org

Executive Director
Caryl Lee Fisher
execdir@nyoug.org

Treasurer
Robert Edwards
treasurer@nyoug.org

Secretary
Thomas Petite
secretary@nyoug.org

CHAIRPERSONS

Chairperson / WebMaster
Thomas Petite
info@nyoug.org

Chairperson / Technical Journal Editor
Melanie Caffrey
editor@nyoug.org

Chairperson / Member Services
Robert Edwards
membership@nyoug.org

Chairperson / Speaker Coordinator
Caryl Lee Fisher
speakers@nyoug.org

Co-Chairpersons / Vendor Relations
Sean Hull
Irina Cotler
vendorcoordinator@nyoug.org

Chairperson / DBA SIG
Simay Alpoge
dbasig@nyoug.org

Chairperson / Data Warehousing SIG
Vikas Sawhney
dwsig@nyoug.org

Chairperson / Web SIG
Coleman Leviter
websig@nyoug.org

Chairperson / Long Island SIG
Simay Alpoge
lisig@nyoug.org

Director / Strategic Planning
Carl Esposito
planning@nyoug.org

CHAIRPERSON / VENUE COORDINATOR

Michael Medved
venuecoordinator@nyoug.org

EDITORS – TECH JOURNAL

Associate Editor
Jonathan F. Miller
jonathanfmiller@earthlink.net

Contributing Editor
Arup Nanda - DBA Corner

Contributing Editor
Jeff Bernknopf - Developers Corner

ORACLE LIAISON

Kim Marie Mancusi
Kim.Marie.Mancusi@oracle.com

PRESIDENTS EMERITUS OF NYOUG

Founder / President Emeritus
Moshe Tamir

President Emeritus
Tony Ziemba

Chairman / President Emeritus
Carl Esposito
cesposi@bers.nyc.gov

President Emeritus
Dr. Paul Dorsey

Table of Contents

New York Oracle Users Group, Inc.
#0208
110 Wall Street, 11th floor
New York, NY 10005-3817
(212) 978-8890

General Meeting – March 19, 2009 Agenda

sponsored by Confio Software & TUSC

AGENDA

Time	Activity	Track/Room	Presenter
8:30-9:00	**REGISTRATION AND BREAKFAST**		
9:00-9:30	**Opening Remarks** **General Information**	(single session) Auditorium	Michael Olin NYOUG President
SESSION 1 9:30-10:30	**KEYNOTE:** The Best Oracle Database 11g New Features	(single session) Auditorium	Rich Niemiec TUSC
10:30-10:45	**BREAK**		
SESSION 2 10:45 -11:45	Tuning the Oracle Grid	DBA Auditorium	Rich Niemiec TUSC
	APEX & 11i – Custom Reporting via Blackberry	Developer Room 118	Brian Caunitz illy caffe
SESSION 3 11:45 -12:30	**Ask the Experts Panel**	(single session) Auditorium	Michael Olin Moderator
12:30 -1:30	**LUNCH - ROOM 123**		
SESSION 4 1:30-2:30	Honey I Shrunk the Data Warehouse!	DBA Auditorium	Shyam Varan Nath Deloitte Consulting
	Oracle Fusion Middleware - Tales from the Trenches	Developer Room 118	Dr. Paul Dorsey Dulcian, Inc.
2:30-2:45	**BREAK**		
SESSION 5 2:45-3:45	Tuna Helper – Proven Process for Tuning SQL	DBA Auditorium	Dean Richards Confio Software
	How to Secure Your APEX Applications	Developer Room 118	Josh Millinger Niantic Systems, LLC
3:45-4:00	**BREAK**		
SESSION 6 4:00-5:00	Advanced Performance Diagnostics: What the GUI Dosen't Show You	DBA Auditorium	Nicholas J Donatone Oracle Corporation
	Integrating Oracle 10g XML: A Case Study Part II	Developer Room 118	Coleman Leviter Arrow Electronics

ABSTRACTS

9:30-10:30 AM KEYNOTE: The Best Oracle Database 11*g* New Features

This presentation will look at which 11g new features should be investigated for use. There will be simple examples to show the basic functionality of the following new features:

- Memory Target
- Partition Advisor
- Security Enhancements
- DDL Lock Timeout
- The Invisible Index
- Automatic Diagnostics Repository
- SQL Plan Management
- Real Application Testing (Workload Capture and Replay)
- SQL Repair Advisor
- ADDM Enhancements
- Interval Partitioning
- Optimizer Enhancements

Richard J. Niemiec is President of TUSC. He is respected around the world as a master in database administration. In 2001, Rich was named by Oracle Corp. as an Oracle Certified Master , one of the first six so recognized around the world. In 2007, Rich authored *Oracle10g Performance Tuning Tips & Techniques*, a follow-up to *Oracle9i Performance Tuning Tips & Techniques* that he wrote four years earlier. Rich is former president of the International Oracle Users Group and currently serves as vice president for its Real Application Clusters special interest group. In addition, Rich is the current president of the Midwest Oracle Users Group. He previously was the executive editor of *Exploring Oracle DBMS* magazine and editor of the MOUG Newsletter. Rich has delivered hundreds of lectures about Oracle technology to users around the world. Past accomplishments include top presenter honors at international Oracle users conferences. He has also conducted numerous expert class presentations at Oracle OpenWorld and other major shows.

10:45-11:45 AM DBA TRACK: Tuning the Oracle Grid

This presentation will discuss the top ten tips when tuning your architecture for grid computing. Oracle has come a long way from when clustering was introduced. The tools continue to improve and the installation continues to get better and easier. This presentation will show many Grid Control screen shots so attendees can see the depth of the product. Topics to be covered are:

1. Know the basics; Introduction to RAC
2. Oracle Direction
3. Market direction - Consolidation
4. Grid Basics - Start with RAC
5. Grid Basics - Scaling it
6. Interconnect and block coordination
7. Tuning quick tips using Statspack & AWR Report
8. Use Grid Control to monitor
9. Grid Control for Multi-Node Systems
10. Use Grid Control to tune systems

See bio for Rich Niemiec above.

10:45-11:45 AM DEVELOPER TRACK: APEX & 11i - Custom Reporting via BlackBerry

Oracle's free developer tools, SQL Developer and Application Express (APEX) can provide Oracle Applications developers with quick and easy reports designed for multiple platforms and multiple audiences. This presentation will show how to successfully build a free APEX-based, fast, and effective reporting environment and how to make reports BlackBerry accessible. Attendees will learn how develop sophisticated multi-platform (Web and BlackBerry) reports using Application Express and SQL Developer against an Oracle Apps 11.5.10 environment.

Brian Caunitz has been working with Oracle Applications for the last ten years as a DBA/Developer at Oracle Consulting, UPS and most recently, illy caffè North America. He has a B.S. in Computer Science from Siena College**.**

1:30-2:30 PM DBA TRACK: Honey, I Shrunk the Data Warehouse!

This presentation will examine use of the Oracle Database compression features to help shrink the size of data warehouses. Shyam will discuss SecureFiles used to reduce the storage needs for unstructured data. To complete the full solution, it is necessary to compress the metadata, dump files during the export using the Data Pump, and backup files. The last piece of the puzzle is the compression of data in motion over the network such as the redo logs. Some live examples and a case study of quantitative benefits on the compressed data alongside the uncompressed data will be shown. Examples of table/partitions, materialized view, and index compression will also be discussed along with the impact of compression on regulatory compliance and how this can help to achieve the industry-specific goals of increased retention of data in a tamper-proof environment.

Shyam Varan Nath is an Oracle BIDW professional with over 18 years of industry experience. Shyam is a part of Deloitte's BI practice and formerly worked for Oracle Corporation's BI consulting practice. Shyam is the founder of the BIWA SIG and is a regular speaker at NYOUG, Oracle OpenWorld, IOUG Collaborate, BIWA, ODTUG, and other regional user groups. He is a Certified DBA with hands-on experience in Data Warehouse architecture, OBIEE implementations, database tuining and optimization. He has worked on large data warehouse implementation projects. Shyam was awarded the Oracle IOUG Contribution Award in 2007.

1:30-2:30 PM DEVELOPER TRACK: Oracle Fusion Middleware - Tales from the Trenches

The Oracle Fusion Technology stack is large and complex, encompassing many components. Some parts are free or nearly so such as JDeveloper, whereas other portions carry large licensing fees (WebCenter). What portions of the Fusion Middleware stack are organizations really using? How have their projects fared so far? This presentation will discuss the results of a number of projects from organizations that are using one or more parts of the Fusion Middleware technology stack. This presentation describes the results of a semi-formal survey where I interviewed some of the leading Fusion Middleware developers and ask them about their projects. Since the results are anonymous, the appraisals of the technology are candid.

Dr. Paul Dorsey is president of Dulcian, Inc. an Oracle consulting firm specializing in business rules and web-based application development and chief architect of Dulcian's BRIM® tool. Paul has co-authored 7 Oracle Press books on JDeveloper, UML Modeling, and Oracle database tools as well as *PL/SQL For Dummies*. He is an Oracle ACE Director, SELECT Associate Editor, ODTUG Symposium chairperson, past IOUG and ODTUG volunteer of the year and an Oracle 9*i* Certified Master. Dr. Dorsey's submission of a Survey Generator built to collect data for The Preeclampsia Foundation was the winner of the 2007 Oracle Fusion Middleware Developer Challenge and Oracle selected him as the 2007 PL/SQL Developer of the Year. Paul can be contacted at paul_dorsey@dulcian.com.

2:45-3:45 PM DBA TRACK: Tuna Helper - Proven Process for Tuning SQL

Many DBAs and developers are faced with tuning poorly performing SQL statements. However, many tuning projects fail because the process being used is inefficient. This presentation will walk through a process Confio Software uses with

great success and will include topics such as: indexing strategies, use of histograms, SQL wait event data, column selectivity and several more that will help you succeed on future tuning projects.

Dean Richards has over 20 years of Oracle project development, implementation and strategic database architecting experience. Before coming to Confio, Dean held engineering positions at McDonnell Douglas and Daugherty Systems - an Oracle solution provider. Dean was also a technical director for Oracle Corporation managing all aspects of key telecommunications accounts including short and long-term technical planning and strategic alliances. As a highly successful liaison between management and technical staff, Dean has proven to be an effective collaborator implementing cutting-edge Oracle solutions.

2:45-3:45 PM DEVELOPER TRACK: How to Secure Your APEX Applications

As important as it is, security is almost always added to an application as a last step, if at all. Designing your APEX application with security in mind is critical and should not be overlooked. This session will discuss how to build secure APEX applications from the ground up so that a hacker cannot exploit them. Josh will also discuss how to segment functionality inside of an APEX application so that only authorized users can perform specific tasks. The presentation includes information about what can be done to secure the underlying data of an APEX application and outline a list of things that need to be secured in an installation of APEX, as well as the associated infrastructure components.

Josh Millinger is the President of Niantic Systems, LLC which he co-founded with Raj Mattamal in 2007. Josh focuses on delivering complex APEX applications to customers across verticals in the Higher Education, Healthcare and Commercial arenas He has Computer Science degrees from UW-Madison and Johns Hopkins University.

4:00-5:00 PM DBA TRACK: Advanced Performance Diagnostics: What the GUI Doesn't Show You

The presentation will review "Performance Methodology", "Review - AWR versus ASH", a few interesting reports, mining your data and a Case Study. Using Enterprise Manager it will be shown how to tune your system for a given workload, identify operations consuming most DB Time, identify resource/capacity related bottlenecks and how to reduce "DB Time" consumed for the workload.

Nicholas J. Donatone is a Grid Sales Consultant for Oracle Corporation. Nick has been working with Oracle software since version 4 (1986). Nick is one of the founders of the NJOUG and was president/co president of the NJOUG for more than 18 years. He is Vice President of the Philadelphia Area Oracle User Group (PHLOUG). Nick has presented at NYOUG, NJOUG, IOUG, ODTUG, SEOUC and Oracle OpenWorld.

4:00-5:00 PM DEVELOPER TRACK: Integrating Oracle 10g XML: A Case Study Part II

Many times during a project life cycle, new technology is introduced that presents first time challenges. This presentation includes information about a project using Oracle 10g XML DB. The presentation will discuss why XML DB was chosen, how XML DB was used and technical issues encountered. Coleman will provide several examples using XMLTYPE, CLOBs, XML DB methods, XMLAGG, XMLELEMENT and XMLFOREST. Additionally, namespace examples and an introduction to XML Schema Definition (XSD) will be presented.

Coleman Leviter is an IT Software Systems Engineer at Arrow Electronics. He has presented at IOUG's Collaborate 07 and 08. He is the NYOUG Web SIG chair and sits on the Steering Committee. He has worked in the financial services and aerospace industries where he developed Navigation, Flight Control and Reconnaissance software for the F-14D Tomcat at Grumman Aerospace. Coleman has a BSEE from Rochester Institute of Technology, an MBA from C.W. Post and an MSCS from New York Institute of Technology. He has recently completed OCP (Oracle Certified Professional) training. He can be contacted at cleviter@ieee.org.

www.pillardata.com

Pillar Axiom.
Application-Aware Storage for Oracle Applications.

Pillar Axiom® can more than double your efficiency in running Oracle applications, because it sees storage from an applications point of view.

Pillar's integrated support for Oracle Enterprise Manager provides an ideal platform for monitoring your application SLAs. Since Pillar is Application-Aware, business policies are easily enforced and automated for Oracle E-Business Suite, PeopleSoft, Siebel, JD Edwards, and Retek.

- Oracle E-Business Suite
- PeopleSoft
- Siebel
- JD Edwards
- Retek

Pillar's support for Oracle Validated Configurations assures a faster, easier, lower-cost platform for all Oracle 11g Database and BI/DW customers, too. And Pillar and Oracle provide joint accelerator solutions for customers looking to expedite database and applications upgrades.

Find out why Oracle chose Pillar to eliminate cost and drive the highest level of efficiencies. And how we can do the same for you, with Application-Aware storage for Oracle applications. They're made for each other, so you can get the best out of both of them.

Call 1.877.252.3706 or visit www.pillardata.com

The First and Only True Application-Aware Storage™

pillar® DATA SYSTEMS

Message from the President's Desk

Michael Olin

Spring, 2009

The New York Oracle Users Group (NYOUG) Celebrates 25 Years of Serving the Greater NYC Area Oracle User Community

2009 marks the 25th anniversary of the founding of the New York Oracle Users Group. It was formed in 1984 with the goal of exchanging ideas, providing assistance and support to users of Oracle software products. The organization consists of volunteer users, consultants, and vendors of Oracle-related products and services.

NYOUG is one of the oldest and largest Oracle user groups in North America. It all started simply enough. In 1983, Moshe Tamir read about this new "Oracle database product", a commercial implementation of E.F. Codd's relational database using the SQL language described in IBM's research papers. Mr. Tamir, realizing that he had just glimpsed the future of databases, quit his job writing microcode for telephone switches and searched for a job where he could work with this amazing new software product. He landed a position as both DBA and developer at Hebrew National Kosher Foods, one of the first Oracle customers in the New York City area. At that time, Hebrew National was running Oracle Version 3 on Data General computers. Looking to share his experiences with Oracle and to learn from others who were using the software, Tamir asked Ken Jacobs (almost two decades before he earned the moniker "Dr. DBA"), who was the Oracle employee responsible for sales on the entire East coast, if he could provide contact information for others who were using Oracle on Data General hardware. That contact information became the mailing list for the first Oracle Users' Newsletter, and the stage was set for the founding of NYOUG.

In 1984, Tamir began working at the New York Blood Center. Along with IT manager Ken Brown, they organized the first official meeting of NYOUG, which was held in a conference room at the United Nations and attracted 10 attendees. Early meetings featured many speakers from "Belmont," the location of Oracle's headquarters at the time. These speakers included Oracle's top technical staff among them, the first manager of Oracle's centralized Support Center, Mary Winslow, who told the group about the new 800 number they could use to reach her staff of 30 support technicians.

NYOUG has been in the forefront of the Oracle user community ever since. Some interesting highlights over the years included:

- The keynote speaker at an NYOUG meeting at the Fashion Institute of Technology was Oracle's president, Ray Lane. Mr. Lane and his walkie-talkie carrying entourage arrived several hours late and tried to cut his appearance short so that he could get to the meetings he had scheduled for later in the day. The membership of NYOUG was anything but cooperative, asking question after question of Oracle's second in command until all hope of making those afternoon meetings was lost.
- NYOUG also met at the midtown offices of Morgan Stanley, hosted by one of the top research analysts following the software industry, Charles E. Phillips, Jr. Long before he became one of Oracle's presidents, Mr. Phillips (or, as his toll free number and website referred to him - "Mr. Chuck") would regularly host the NYOUG. He routinely polled the membership after returning from meeting with Oracle's executives on the West coast. "This is what they told me in California," he would say, "now what is the truth?"

NYOUG's members and officers have authored numerous books on Oracle related topics, presented at Oracle conferences throughout the world, and even taught database systems and Oracle to university students. Among the membership are Oracle ACEs, members of the Oak Table Network and even an Oracle "DBA of the Year" (Arup Nanda, 2003) and "PL/SQL Developer of the Year" (Dr. Paul Dorsey, 2007).

NYOUG's General and SIG meetings (which now include separate DBA, Web, Data Warehousing and Long Island meetings) attract some of the top speakers and leaders in the industry. Past meeting keynote speakers have included Oracle's Ray Lane, Charles Philips, Ken Jacobs, Tom Kyte, Thomas Kurian, Dai Clegg, Roel Stalman, Vijay Tella, and Wim Coekaerts, as well as other leading Oracle gurus like Steven Feuerstein, Rich Niemiec, Michael Abbey, and Arup Nanda. While the regular general meetings routinely attract over 100 attendees, NYOUG's "Metro Area" events draw over 500. The current NYOUG Technical Journal quarterly issues consist of over 50 pages full of useful white papers, tips

and techniques from the meeting speakers and NYOUG members. The entire archive of presentations from NYOUG meetings is available on the web (http://www.nyoug.org) and the group is exploring the possibility of podcasting meeting presentations.

Over the past 25 years, NYOUG has seen its paid membership grow to over 700 and its weekly email blasts reach over 4,000 IT professionals. Although part of the IOUG Regional User Group structure, NYOUG is a completely independent and self-funded group, supported by revenue from its membership dues and meeting vendor sponsors. NYOUG has been a resource for the Oracle user community for almost as long as there have been Oracle users. The group has accomplished quite a bit in that time and is well positioned for the next 25 years.

Editor's Corner

Melanie Caffrey

And the Award Goes To …

The Editor's Choice Award for 2008 is awarded to **Peter Koletzke**, author of the paper, **Introduction to Java – PL/SQL Developers Take Heart!,** published in the current issue of the NYOUG Tech Journal. Peter presented this topic at the December 2008 general meeting. He is a technical director and principal instructor for the Enterprise e-Commerce Solutions practice at Quovera, in Mountain View, California, and has over 24 years of industry experience. Peter has presented at various Oracle users group conferences more than 230 times and has won numerous user group awards. He is an Oracle Certified Master, Oracle Fusion ACE Director, and coauthor with Dr. Paul Dorsey, Avrom Roy-Faderman, and Duncan Mills of seven Oracle Press Books about Oracle development tools. It is a pleasure to have Peter Koletzke associated with the NYOUG.

Candidates for the 2009 award appear beginning with this issue. Many of the papers included in the Tech Journal are based on presentations given at NYOUG general meetings, but presentation is not a requirement to have a paper or article published in the newsletter. Please send your paper to: editor@nyoug.org, by April 1, to appear in the June issue. Due dates for later issues are: July 1 (September) and October 1 (December).

Bringing Back the DBA and Developer's Corners

The NYOUG Tech Journal is happy to be hosting the DBA and Developer Corners, a forum whereby presenters/authors are welcome to submit articles specific to DBA and Developer topics. Additionally, if you have a topic that you feel is too broad to fit into one journal issue, per se, but may instead be spread out across several issues in a series, the submission of such pieces is always welcome.

Newsletter Close and Publication Dates – 2009

Meeting Date	Location	Newsletter Publication Date	Article Submission Deadline	Ad Closing Date
June 2009	St. John's University	June 1	April 1	May 1
September 2009	TBD	September 1	July 1	August 1
December 2009	St. John's University	December 1	October 1	November 1

Conference Schedule – 2009

Be sure to participate in the following Oracle events and other events of interest in 2009:

OAUG Connection Point – Dubai, UAE	April 1-2
InSync '09 – Sydney, Australia	April 20-21
UKOUG Ireland '09 – Dublin, Ireland	April 22
Oracle Open World Japan – Tokyo, Japan	April 22-24
COLLABORATE '09 – Orlando, FL	May 3-7
NoCOUG Spring Conference – Pleasanton, CA	May 21
ODTUG Kaleidoscope '09 – Monterey, CA	June 21-25
SAOUG '09 – Durban, South Africa	June 22-24
Virginia Oracle Users Group '09 – Richmond, VA	October 7-8
Oracle Open World '09 – San Francisco, CA	October 11-15
UKOUG '09 – Birmingham, England	November 30–December 2

Value of TimesTen

Shig Hiura, Oracle Corporation

INTRODUCTION

The Oracle TimesTen In-Memory Database is a memory-optimized relational database that delivers very low response time and very high throughput for performance-critical systems. It is targeted to run in the application tier, close to applications, and optionally in process with applications. It may be used as the database of record, or as a cache to the Oracle Database.

THE GROWTH OF REAL-TIME APPLICATIONS

Real-Time Industries

For many companies, real-time applications aren't elective. They are a necessity to be in business. Network equipment manufacturers, telecom operators, securities exchanges and brokerages, airlines, shipping and logistics companies and defense and intelligence agencies are prime examples.

Real-Time Enterprises

With the growing velocity of messages moving through business networks, the use of real-time processing to capture, analyze and respond intelligently to key events is increasingly becoming the benchmark for corporate excellence. This isn't just important for the execution and management of critical business processes. Customers expect highly-tailored interactions and the utmost responsiveness from any company with whom they do significant business.

BEYOND THE TRADITIONAL DATABASE

When people think of a relational database, they think of a traditional client-server architecture where the client connects to a server over a network and issues a request. That request is processed by the server, and the results are returned back to the client, usually over the same network. Even with the web and grid computing, this model has remained pretty much the same, and has served well for most enterprise applications.

However, for some applications the performance requirements exceed the capabilities of this architecture. These applications lie at the edge of the performance envelope, and some push beyond the limits of what a client-server database architecture offers. Performance is typically measured in terms of latency (responsiveness) and concurrency (throughput), and factors such as network connections, inter-process communication, and database process architecture define their limits. To overcome these limits, developers "bring the data" closer to the application by storing them in memory structures local to the application, thereby making the data quickly accessible. This addresses the short term needs of performance. However, over time the shortcomings of this "home-grown cache" approach become evident:

- Can the data be persisted in case of failure?
- Can more than one process access the cache?
- Can updates be made to the cached data?
- Can data be made consistent across caches (on different servers)?
- Can cache data be queried in many different ways?
- Can cache data be managed easily?

To address these concerns, developers must create their own persistence, concurrency, transaction, and replication mechanisms, going down a path of proprietary code that is specific to the application and expensive to maintain. It is not sufficient to merely collect and cache data next to applications, nor is it practical to co-locate the corporate database on the same platform as one of the applications.

What if we can have the benefits of the home-grown in-memory cache, with the benefits of a relational database? TimesTen provides:

- A fully relational RDBMS supporting standard interfaces such as ODBC, JDBC, and SQL
- A robust database, providing the Atomicity, Concurrency, Integrity, and Durability (ACID properties) required by today's applications, as well as advanced features such as replication
- A proven database with a 10+ year history of successful deployments
- A lightweight, embeddable engine that can be managed by the application

PRODUCTS

The Oracle TimesTen product line consists of a base product, the Oracle TimesTen In-Memory Database, and two optional products:

- Replication: TimesTen to TimesTen
- Cache Connect to Oracle

TimesTen In-Memory Database

The In-Memory Database is a self-contained read/write relational database fully loaded into memory, and accessible via standard SQL, JDBC and ODBC. TimesTen supports Oracle data types and character sets, providing portability of data between Oracle and the In-Memory Database.

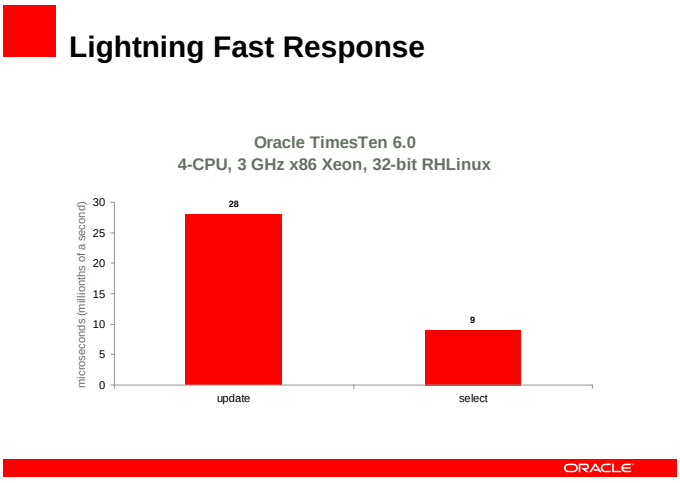
To ensure that data and transactions are not lost, local disks are used for logging changes and for recovery after a system failure. TimesTen provides the robust capabilities of a disk-based database with support for transactions with ACID properties, with the performance of an in-memory database.

The In-Memory Database's administration can be performed via command line utilities or SQL. This makes TimesTen readily applicable to embedded applications, where lights-out self-management is required.

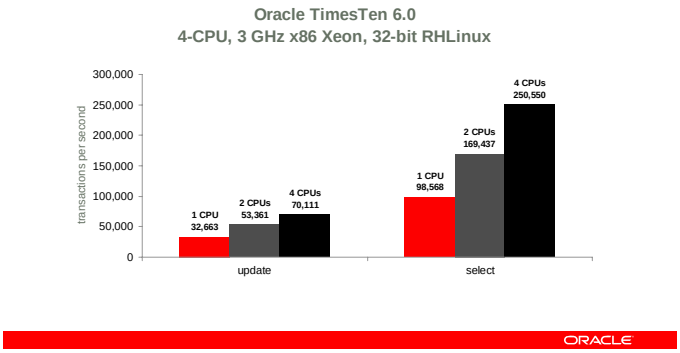
TimesTen is usually deployed on the application tier, close to the business logic. This proximity along with the streamlined in-memory architecture results in high throughput and low latency.

Performance Illustrated

Latency is one measure of database performance, and it supports an application's responsiveness to requests. The following chart depicts response times for simple updates and query benchmarks, using a low-cost commodity platform. At first look the results seem typical for any database. The distinction here is that while disk-based relational databases deliver response times in units of *milliseconds*, TimesTen delivers in units of *microseconds*.

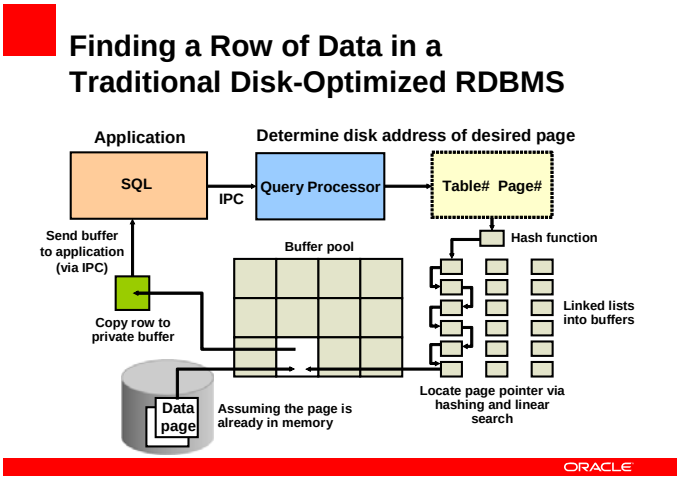


Another measure of performance is throughput and scalability. The following chart illustrates the near-linear growth in throughput as the number of processors increases. However, the more noteworthy point of this chart is the outstanding transaction throughput that is possible using low-cost commodity servers with TimesTen.



Source of TimesTen performance

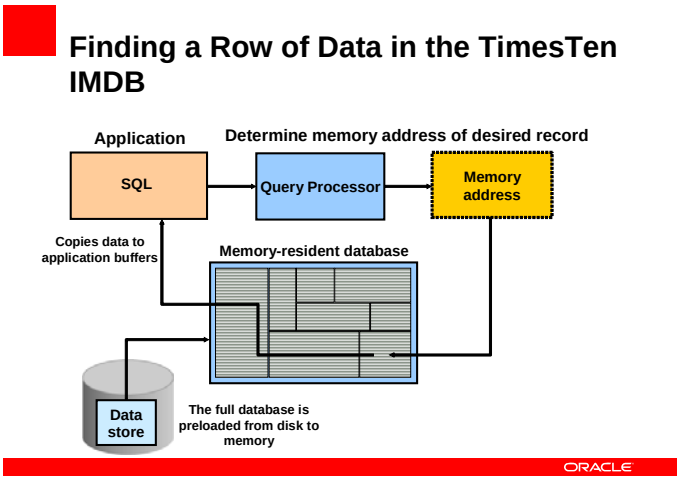
All traditional disk-optimized relational database systems incur an overhead in accessing data. The exact implementation varies from system to system, but every disk-optimized RDBMS must retrieve its data either from memory buffers or from disk into memory buffers before it can turn that data over to the application that requested it. A general architectural depiction of a disk-based RDBMS is provided below.



Regardless of implementation specifics, a record identifier of some sort must be translated into a memory address within the buffer pool of the RDBMS. This is assuming that the data has already been brought into memory by a previous request for that data. If the data has not been placed in the memory-resident buffer pool, then it must be read from disk, which is many times slower than memory access. The translation of the record identifier into a memory address within the buffer pool is just one element of the CPU-intensive overhead of a disk-optimized RDBMS.

When the SQL query processor requires a page of data, it first searches the buffer pool for that data in memory. Even if that data is in the buffer pool, it must be copied out of the pool for subsequent processing. This buffer pool maintenance and management, coupled with additional data copies, adds significant overhead of making the data available to the application.

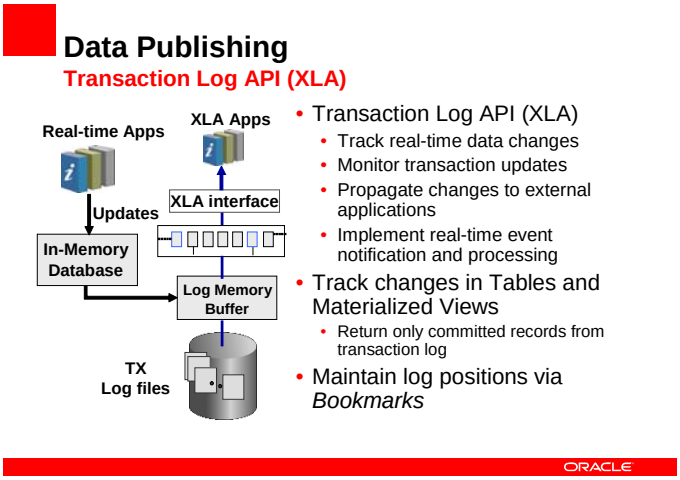
Although necessary in disk-optimized data management solutions, buffer pools become unnecessary in memory-based data management. The TimesTen IMDB has no buffer pool because the entire data store resides in main memory.



When the TimesTen IMDB accesses a row, performance is optimized by keeping CPU consumption to a minimum. Therefore, instead of CPU-intensive calculations of row identifiers and the overhead of buffer managers and buffering mechanisms, memory addresses are used wherever possible for direct memory addressing of data. There are far fewer steps needed to retrieve the data; about one-tenth the amount of CPU instructions compared to a traditional disk-optimized RDBMS. This results in large performance gains even over a traditional disk-optimized RDBMS database that is fully cached in memory.
Therefore, an in-memory database is fast not merely because it avoids disk I/O but because it reduces CPU consumption. After disk I/O is removed from query processing, the only bottleneck in retrieving the data is CPU instructions.

Data Publishing with TimesTen

Many customers need to be able to connect their real-time applications with other applications. One of the key features offered by TimesTen is the ability to track real-time data changes for notification. An open Transaction Log API (XLA) with a standard JMS interface is provided for reading the transaction log. This is useful for creating applications that react to database updates. In this regard, XLA is a lightweight "trigger". XLA enables the application to subscribe to data changes in order to propagate the changes to an external application or to use the real-time data tracking to implement event notification and event processing. It's also a way to build custom data replication from Oracle TimesTen to other database systems.



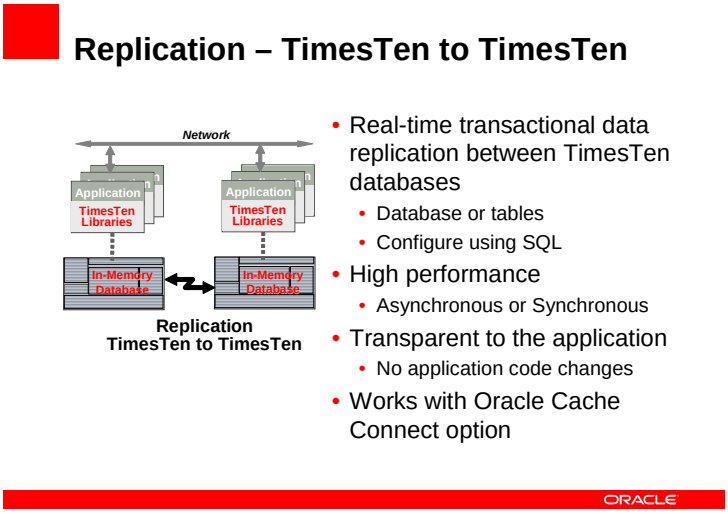
Replication – TimesTen to TimesTen

TimesTen Replication facilitates real-time copying of data between databases. The fundamental motivation behind Oracle TimesTen replication is to make data continuously available to mission critical applications with minimal performance impact. In addition to its role in failure recovery, replication is also useful for distributing user loads across multiple databases for maximum performance and for facilitating online upgrades and maintenance.

Oracle TimesTen follows a “master-subscriber” replication model, whereby committed changes are copied from their source to one or more subscriber databases. Replication is configured through SQL statements and can apply to designated tables or an entire database. To enable high efficiency and low overhead, a transaction-log based replication scheme is used.

Replication at each master and subscriber database is controlled by replication agents that communicate through TCP/IP stream sockets. The replication agent on the master database reads the records from its transaction log and forwards any detected changes to replicated elements to the replication agent on the subscriber database. The replication agent on the subscriber then applies the updates to its local database. If the subscriber agent is not running when the updates are forwarded by the master, the master retains the updates in its transaction log until they can be applied at the subscriber.

Data replication configurations include active-standby, active-active and n-way, using asynchronous or synchronous transmission, with conflict detection and resolution. Data replication is fully compatible with the Cache Connect to Oracle option.

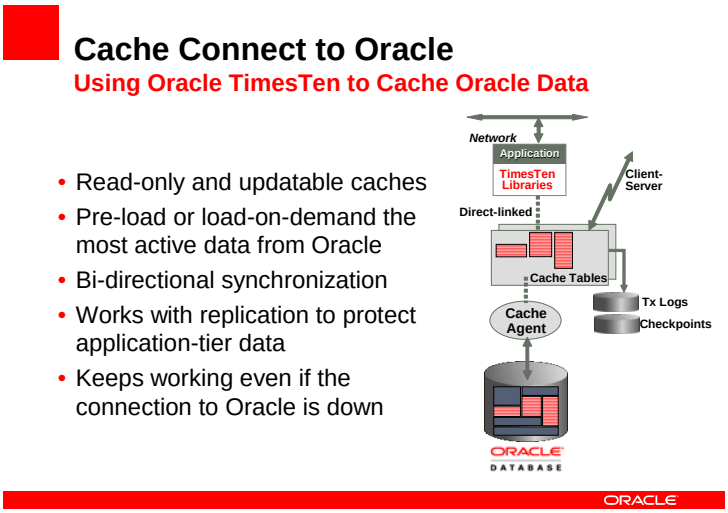


Cache Connect to Oracle

It’s common for Oracle TimesTen to be deployed in cooperation with a disk-based RDBMS, whereby TimesTen is used when data needs to be captured or processed in real-time. As the data transitions to a non real-time state (e.g. when a stock trade is complete or a call detail record has been rated) the information is transferred from TimesTen to the backend RDBMS. There are multiple ways in which this integration can be achieved.

Applications can connect to both Oracle TimesTen and the backend database and do the data movement through regular API requests. This is the most flexible approach, but is the least transparent to the applications, and could require complex programming.

A derivative of the above approach is to connect Oracle TimesTen and the backend RDBMS through a publish and subscribe message bus, and to write new modules separate from existing applications that “listen” for changes to publish to the bus and duplicate changes picked up from the bus. An application can use Oracle TimesTen’s transaction log API (XLA) to register for and receive notice of updates made to the database. Most RDBMSs offer a trigger feature that can be used to signal changes, or provide an API into the transaction log, similar to XLA. An easier and more transparent alternative for integrating Oracle TimesTen databases and Oracle Databases is through the addition of Cache Connect to Oracle, with its built-in connections.



Cache Connect to Oracle is an option to the Oracle TimesTen In-Memory Database that creates a real-time, updatable cache for Oracle data, residing in the application tier. It offloads computing cycles from backend systems and enables remarkably responsive and scalable real-time applications. Cache Connect to Oracle loads a subset of Oracle data into TimesTen, propagates updates in both directions, automates pass through of SQL requests for non-cached data and automatically resynchronizes data after failures. Cache Connect to Oracle is fully compatible with the Replication – TimesTen to TimesTen option.

Unlike most caches, which are read-only, Cache Connect to Oracle supports read/write caching of Oracle data and bi-directional propagation of updates between TimesTen and Oracle Databases. Another distinction is the concept of a "cache group", which describes a group of Oracle TimesTen tables that map to all or a subset of the tables in an Oracle Database. A cache group can consist of all or a subset of the rows and columns in these Oracle Database tables. Cache Connect to Oracle provides controls to specify how long the data from the Oracle Database should remain in the cache. In addition to the ability to set up automatic functions, a cache group can be loaded, refreshed, flushed, and unloaded on demand through SQL statements.

Cache Connect to Oracle interacts with the Oracle database to perform all of the synchronous cache group operations, such as create a cache group, load the cache group from Oracle, and propagate updates between the cache group and Oracle. In addition, there is a background process, called the Oracle Agent that performs all of the asynchronous cache operations, such as automatically propagating updates from Oracle Database to a cache group in Oracle TimesTen.

Cache Connect to Oracle applications can send SQL statements to either a cache group or Oracle Database through a single connection. This single-connection capability is enabled by a pass-through feature that checks if the SQL statement can be handled locally by the cache tables or if it must be redirected to Oracle Database. Cached data can be updated in either the Oracle TimesTen cache group or Oracle Database. Cache Connect to Oracle provides the ability to automatically propagate updates from the cache group to Oracle Database, as well from Oracle Database to the cache group in TimesTen.

CONCLUSION

With the growing velocity of messages moving through business networks, the use of real-time processing to capture, analyze and respond intelligently to key events is increasingly becoming the benchmark for corporate excellence. This isn't just important for the execution and management of critical business processes. Customers expect highly-tailored interactions and the utmost responsiveness from any company with whom they do significant business.

What's needed is a generation of lightweight infrastructure software that presents familiar, powerful interfaces and query languages that are widely in use – that can easily interface with existing back-office databases, messaging systems and application servers – and exploits the full performance potential of today's networked, memory-rich computing platforms. This is the generation of infrastructure software for real-time data management provided by Oracle TimesTen.

Oracle TimesTen products provide application-tier data management for performance-critical systems, optimized for blazing-fast response and real-time caching of Oracle data. Hundreds of companies worldwide use Oracle TimesTen in production applications, including Amdocs, Aspect, Avaya, Bombay Stock Exchange, Cisco, Ericsson, JP Morgan, Lucent, NEC, Nokia, Salesforce.com and Sprint.

JVM Cutover for Oracle Forms

Gilbert Standen, Sr. Consultant, TUSC, standeng@tusc.com

Managing Partner, Stillman Real Consulting LLC, mail@consultingcommandos.com
Oracle RDBMS Consulting for 12 years to private industry and government

INTRODUCTION

This paper will describe how to reconfigure your Oracle Forms to use the generic Sun JVM instead of Oracle Jinitiator ("Jinit") and will offer compelling reasons why this cutover should be explored and scheduled for implementation in development and then prod environments.

CUTOVER RATIONALE

The main reason to implement a cutover to Sun JVM from your current Oracle Jinit is to facilitate upgrading your Oracle Forms to a more fully SOA-BPEL-enabled version of Forms which can register an interest in a BPEL event.
It appears this shall be Forms 11 and consequently you would then deploy J2EE version 6 to your client machines (Sun JRE 1.6.x_x) at that point, and at which time Jinitiator must be decommissioned and can (optionally) be deinstalled from clients. Forms cannot yet (as of version 10.1.2) easily register an interest in a BPEL event and automatically be notified if input is needed from Forms. In version 11 of Forms Oracle intends per their SOD to have functionality in place that will make this much easier (watch for updates on this from Oracle Corp). Forms 11 also includes the Java Importer tool which is PL/SQL wrappers for Java which is also a key element of integrating Forms into the SOA-BPEL landscape. Therefore it is suggested to set your sights on upgrading your Oracle Forms to Forms ver. 11. This version of Forms is currently slated to be the more fully "SOA-enabled" version of Forms.
The widely-discussed incompatibility of Jinitiator with Windows Vista can be considered a peripheral "smoke" issue and is not a primary driver, unless your organization is firmly committed to a Windows Vista enterprise upgrade in which case you must make the cutover from Jinit to Sun JRE. It is worth noting that acceptance and deployment of Windows Vista has been deferred, or cancelled (or even undone) by significant numbers of large enterprise business clients. For example, "The agency that governs educational technology in the United Kingdom has advised schools in the country to keep Microsoft Windows Vista operating system and its Office 2007 software out of the classroom and administrative offices. A British educational report suggests the upgrade would increase costs and create software compatibility problems while providing little benefit."[1] This view is typical of many enterprises, whether giants or leprechauns. Many other examples can be found on the web. In fact, Microsoft is "quietly allowing PC makers to offer a downgrade option to buyers that get machines with the new operating system but want to switch to Windows XP," but the program only applies to Vista Business and Ultimate editions. The likes of Fujitsu, HP, Lenovo and Dell all have done this."[2] This prompted many businesses to delay upgrading to Vista and even caused some people who had upgraded to Vista to replace their installations with Windows XP or other operating systems. These results, publicized by online reviews and articles, further led to low adoption levels of Windows Vista and largely negative public review, as reflected by its title from PC World as the biggest tech disappointment of 2007[3]. The market share for Windows Vista, taking the median from various sources, was 9.03% as of February 2008.

---

[1] http://www.informationweek.com/news/management/showArticle.jhtml?articleID=205602879

[2] http://www.engadget.com/2007/09/21/microsoft-giving-vista-business-ultimate-users-downgrade-to/

[3] PC World: The 15 Biggest Tech Disappointments of 2007 "#1. No Wow No How: Windows Vista".

Other reasons include[4][5][6]:

1. Oracle has no plans to certify Jinit with Oracle E-business version 12;
2. Eliminate JVM vs. Jinit client workstation conflicts;
3. Deploy a single JVM to client workstations.

Key Forms Implementation Goals:

- Forms which has Java Importer capability (PL/SQL wrappers for Java).
- Forms which can register and get notifications from BPEL-compliant services
- Forms 3-tier architecture deployment. Need to be on Forms 11 at a minimum
- Timely application/awareness of SOA-BPEL-related Forms patches/upgrades

All of the above are encompassed in Forms 11 and so this is a logical point at which to make the cutover to Sun JRE and decommission Jinitiator.

SUGGESTED UPGRADE STRATEGY

This suggests that a possible strategy would be to start planning ahead now to do one sweeping, major upgrade event:

Proposed "Leveraged Strategy" Upgrade Plan Steps

1. Upgrade Forms to version 11
2. Upgrade Database(s) to 11g
3. Recompile all Forms applications (RSF-Pl/Sql)
4. Reconfigure Forms to push Sun JRE 6 to clients
5. Decommission and Uninstall Jinitiator from clients
6. Optionally can now upgrade clients to Win Vista.

Rationale and Justification for Upgrade Strategy

- Positions existing Forms as SOA-BPEL compatible, enabled and "compliant".
- Positions existing Forms to be using J2EE version 6 of the client JRE (JRE 1.6.x_x) which is the generic Sun JRE version certified for Forms 11.
- Implements the REQUIRED Jinitiator retirement.
- Requires your organization to only deal ONCE with the re-compiling of all Forms applications due to the Database RSF – PL/SQL compatibility requirement across major database releases (e.g. 10g -11g)[7]
- Leverages new Oracle 11g RDBMS features.
- Positions you to be able to go to Windows Vista on clients (optional).

---

[4] Oracle Forms 10*g* Release 2 (10.1.2.x) Statement of Directions (May 22, 2008)
http://www.oracle.com/technology/products/forms/htdocs/10gR2/clientsod_forms10gR2.html

[5] Oracle Forms in the SOA World by Robin Zimmermann
http://www.oracle.com/technology/oramag/oracle/05-mar/o25forms.html

[6] Oracle Forms & a Service Oriented Architecture (SOA): A Whitepaper from Oracle Inc. June 2007
http://www.oracle.com/technology/products/forms/pdf/10gR2/forms-soa-wp.pdf

[7] Smoother version-to-version upgrade commitment from Oracle should make this easier."Oracle will allow compatibility changes between Database Required Support Files (RSF's) and PL/SQL versions to occur ONLY across major releases of the database"

CUTOVER PROCEDURE

Oracle forms is configured to use Sun JVM in the formsweb.cfg file. We give the example of version 1.5.0_12 of Sun JRE which is the version that would be used for Forms 10. However, as mentioned above, Forms 11 will use 1.6.x_x so make the changes as necessary in the procedures that follow if doing edits of formsweb.cfg for Forms 11. The formsweb.cfg file can be edited manually or it can be edited with Enterprise Manager Application Server Control Console (this is the "lightweight" OEM that comes bundled with the 10gAS). Figure 14 below shows the GUI you would you use in the OEM Server Control Console. If you opt to edit formsweb.cfg manually, be sure to make a backup of the file first. The formsweb.cfg file is located at $ORACLE_HOME/forms/server/formsweb.cfg.

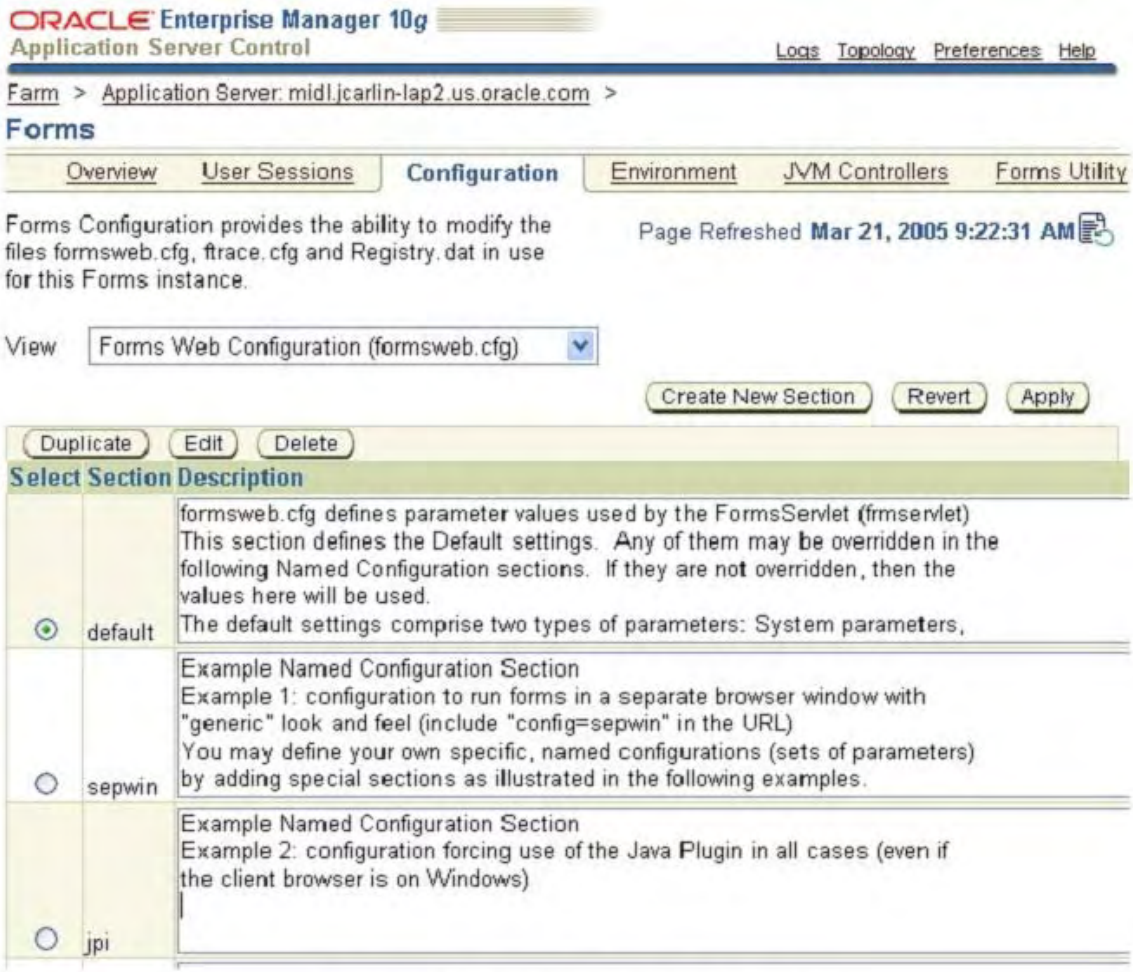


Figure 14: EM Configurations page

The parameters which must be changed in formsweb.cfg are as follows. Note that these are changed in the top beginning section of formsweb.cfg which is not a "named" section.

1. baseHTMLie change to basejpi.htm
2. baseHTMLjpi change to basejpi.htm
3. baseHTMLjinitiator change to basejpi.htm
4. baseHTML change to basejpi.htm
5. IE irrelevant (can be set to "native" or "Jinitiator")

The setting of IE=native or IE=JInitiator is no longer relevant. Basically, what this does is determine if the setting for baseHTMLjinitiator (IE=JInitiator) or baseHTMLie (IE=native) should be used. Since we changed both base-parameters to point to the same file, this is no longer relevant.
The basejpi.htm uses several variables from the formsweb.cfg in the EMBED and OBJECT tags. These are: jpi_classid, jpi_codebase, jpi_mimetype and jpi_download. These must also be reset to new values. These are four settings in your formsweb.cfg file that are relevant to the Sun JPI configuration and they are enumerated below for easy reference as follows:

1. jpi_classid – The ClassID of the Sun JVM to use (Internet Explorer specific);
2. jpi_codebase – Download location of the CAB file for the Sun JVM (IE specific);
3. jpi_mimetype – Used primarily for Netscape/Firefox but also passed on to IE;
4. jpi_download_page – Download location of JRE installer for Netscape/Firefox.

Jpi_classid

JPI_CLASSID should be set to clsid:CAFEEFAC-<major version>-<minor version>-<patch version>-ABCDEFFEDCBA for static versioning. For example, the classID for Sun JPI v1.5.0_12 would be clsid:CAFEEFAC-0015-0000-0012-ABCDEFFEDCBA.
Note that dynamic versioning can be used, but is not recommended due to unpredictability of Oracle Forms performance on a non-standardized JVM deployment. The setting for JPI_CLASSID for dynamic versioning, however, for reference is to set it to clsid:8AD9C840-044E-11D1-B3E9-00805F499D93 for dynamic versioning.

Jpi_codebase

With static versioning JPI_CODEBASE should be a URL to the .CAB file to download this exact version. A list of possible URLs is available at Sun. For dynamic versioning the URL consists of two parts, separated by a #-sign. The last part specifies the minimum required Java version. An example is Version=1,5,0,12 for Java v1.5.0_12. The first part of the URL should point to the CAB file to download when the required (or a higher) version is not installed on the workstation. You could point to the CAB file of specific version (e.g. http://java.sun.com/update/1.5.0/jinstall-1_5_0_12-windows-i586.cab if you wanted the Sun Java v.1.5.0_12).
You can also just use the version number of a family to automatically download the latest version of this family (e.g. http://java.sun.com/update/1.5.0/jinstall-1_5-windows-i586.cab will download the latest version in the 1.5 family). However, this is strongly discouraged for use with Oracle Forms due to the need to validate and standardize your application on a specific version of known validated results delivery for users.
Also, for applications run over HTTPS, external URLs in the HTML must be changed to also use HTTPS, otherwise Internet Explorer raises a warning about combining secure and non-secure items on the same page. Since Sun's download sites are also available through HTTPS this can also be used.

Jpi_mimetype

Use a format like "application/x-java-applet;<version_type>=<implementation_version>". For static versioning, <version_type> should be set to "jpi-version". You can just require a family or a very specific version/patch. Use "version=1.5.0_12" to require at least v1.5.0_12. This makes the entire mime-type "application/x-java-applet;version=1.5.0_12". However, this may not seem to work with Firefox, so you may want to try using the more generic "application/x-java-applet;version=1.5.0" with firefox.

Jpi_download_page

JPI_DOWNLOAD_PAGE is the URL where Netscape/Firefox users can download the JRE. Note that for many Oracle shops, which use Internet Explorer this setting is not relevant, but it is best to set it. Sun's examples show this as http://java.sun.com/j2se/1.5.0/download.html. Again, you can change this to use https to prevent warnings in your browser, making it https://java.sun.com/j2se/1.5.0/download.html

NAMED CONFIGURATION SECTIONS OF FORMSWEB.CFG FILE

Remember that if you have named sections in your formsweb.cfg file which also require JVM you will need to make changes there as well. For example, if you are using Oracle Webutil, you would need to make these changes as well to ensure use of Sun JVM. The section to be changed will be found under [webutil]

1. baseHTMLJinitiator – Should point at the WebUtil JPI, (and not JInitiator) template file, for example, baseHTMLJinitiator=webutiljpi.htm.
2. baseHTMLjpi - Should point to the baseHTML file for the Java Plug-in, for example baseHTMLjpi=webutiljpi. baseHTMLjpi is used when JInitiator cannot be used, such as with non-Windows platforms.
3. baseHTML - Should point to the baseHTML for WebUtil, e.g. baseHTML=webutilbase.htm. baseHTML is for running WebUtil using Internet Explorer's native JVM. However, this can also be set to use webutiljpi.htm.

Notice how these named sections have their own jpi htm file that must be referenced.

EXAMPLE SNIPS FROM FORMSWEB.CFG FILE

The complete default formsweb.cfg file is not shown here because a listing of the entire file would take up too much publication space. However, the relevant sections which must be changed are shown.

```
# formsweb.cfg defines parameter values used by the FormsServlet (frmservlet)
# This section defines the Default settings.  Any of them may be overridden in the
# following Named Configuration sections.  If they are not overridden, then the
# values here will be used.
…
[default]
# System parameter: default base HTML file
baseHTMLie=basejpi.htm
baseHTML=base.htm                          # (Change this to basejpi.htm)
# System parameter: base HTML file for use with JInitiator client
baseHTMLjinitiator=basejini.htm     # (Change this to basejpi.htm)
# System parameter: base HTML file for use with Sun's Java Plug-In
baseHTMLjpi=basejpi.htm
# System parameter: delimiter for parameters in the base HTML files
HTMLdelimiter=%
...
# Page displayed to Netscape users to allow them to download Oracle JInitiator.
# Oracle JInitiator is used with Windows clients.
# If you create your own page, you should set this parameter to point to it.
##########################################################################
# These are the old jinit parameters.  They are moot because we are now using “jpi”
There is no need to comment these out.  They only become active again if the baseHTML
parameters are changed back.
jinit_download_page=/forms/jinitiator/us/jinit_download.htm
# Parameter related to the version of JInitiator
jinit_classid=clsid:CAFECAFE-0013-0001-0022-ABCDEFABCDEF
# Parameter related to the version of JInitiator
jinit_exename=jinit.exe#Version=1,3,1,22
# Parameter related to the version of JInitiator
jinit_mimetype=application/x-jinit-applet;version=1.3.1.22
##########################################################################
# Page displayed to users to allow them to download Sun's Java Plugin.
# Sun's Java Plugin is typically used for non-Windows clients.
# (NOTE: you should check this page and possibly change the settings)
jpi_download_page=http://java.sun.com/products/archive/j2se/1.4.2_06/index.html
# Parameter related to the version of the Java Plugin
jpi_classid=clsid:CAFEEFAC-0015-0000-0012-ABCDEFFEDCBA
# Parameter related to the version of the Java Plugin.  Downloads an exact version.
jpi_codebase=http://java.sun.com/products/plugin/autodl/jinstall-1_5_0-windows-
```

```
i586.cab#Version=1,5,0,12
# Parameter related to the version of the Java Plugin
jpi_mimetype=application/x-java-applet;jpi-version=1.5.0_12
# EM config parameter
…
[webutil]  # (Note here is a named section which requires changes as well)
WebUtilArchive=frmwebutil.jar,jacob.jar
WebUtilLogging=off
WebUtilLoggingDetail=normal
WebUtilErrorMode=Alert
WebUtilDispatchMonitorInterval=5
WebUtilTrustInternal=true
WebUtilMaxTransferSize=16384
baseHTMLjinitiator=webutiljini.htm   # (Change to webutiljpi.htm)
baseHTMLjpi=webutiljpi.htm
archive_jini=frmall_jinit.jar
archive=frmall.jar
lookAndFeel=oracle
```

JAR FILE CONSIDERATIONS

"Oracle provides two Jar files (f90all.jar and f90all_jinit.jar). f90all.jar is a standard Jar file, and f90all_jinit.jar is a Jar file with extra compression that can only be used with Oracle JInitiator."
You may wish to experiment when switching to Sun JVM with using f90all.jar vs. f90all_jinit.jar. You may see performance improvement.
Reference: http://oraclesvca2.oracle.com/docs/cd/B10464_01/web.904/b10470/appa.htmOracle Application Server Forms Services Deployment Guide 10g (9.0.4) Part Number B10470-01.

XML CONSIDERATIONS

Advanced users might want to edit the web.xml file to Use a Forms Servlet config file other than the std. one in <ORACLE_HOME>/forms90/server/formsweb.cfg). To do this, uncomment and change the f90servlet "configFileName" servlet parameter. Reasons for this may include:

- Run Oracle Forms using static HTML (instead of Forms Servlet).
- When Oracle Forms applications are run using a method other than the Forms Servlet (for example, static HTML pages, or JSPs), parameter settings in the formsweb.cfg file are NOT used.
- You may need to define servlet parameters for the Listener Servlet, such as workingDirectory and envFile (specifying the current working directory for the Forms runtime processes, and the file containing environment settings to be used).

The location of the xml file is:
`j2ee/OC4J_BI_FORMS/applications/forms90app/forms90web/WEB-INF`

(underneath <ORACLE_HOME>)

FRM-92160 ERROR ON CLIENT MACHINE

When changing over to JVM from Jinit for the first time, clients may get the FRM-92160 Error. Try the following steps to fix this error: On client:

1. Exit Browser
2. Start => Control Panel
3. Open Jinit (or JVM) console
4. Click "Cache" Tab
5. Click button to "Clear JAR Cache"

Restart browser and login to Forms app again.

This will often clear the FRM-92160 error on client machines.

REFERENCES

Oratransplant, "***Settings for dynamic versioning with Sun JPI and Oracle Forms***", at weblink http://www.oratransplant.nl/2005/05/24/settings-for-dynamic-versioning-with-sun-jpi-and-oracle-forms/

Oratransplant, "***How to configure Forms to use Sun JPI",*** at http://www.oratransplant.nl/2005/06/16/how-to-configure-forms-to-use-sun-jpi/

Oracle® Application Server Forms Services Deployment Guide, 10g Release 2 (10.1.2), Oracle Part No. B14032-03, Sections 4.3.1 and 4.3.1.1 at http://download-uk.oracle.com/docs/cd/B14099_19/web.1012/b14032/configure003.htm#CACFIBHE

Oracle Forms: Configuring Webutil, section "***The Formsweb.cfg File***" at http://www.oracle.com/webapps/online-help/forms/10g/state/content/navId.3/navSetId._/vtAnchor.CBDGDEBB/vtTopicFile.web_util%7Cconfig%7Ehtm/

Oracle® Application Server Forms Services Deployment Guide, 10g Release 2 (10.1.2), Oracle Part No. B14032-03, Appendix C.1 Example Default formsweb.cfg file.

For Mission Critical Systems, Do You Have Both Eyes Open?

For your Oracle systems, see how GoldenGate gives you one solution for both high availability and real-time data integration.

- **Oracle 8i or 9i to 10g/11g migrations** – No Downtime
- **Active-Active** – Highest Availability
- **Direct feeds for Data Warehousing** – Enable Operational BI
- **Off-load Data to a Real-Time Reporting Database** – Better Performance
- **Heterogeneous Data Replication** – Extremely Flexible

With GoldenGate, you solve many business needs with one technology.

- Global Headquarters: +1 415 777 0200
- sales@goldengate.com
- **www.goldengate.com**

GOLDENGATE®

Real-Time Access to Real-Time Information

The Right Way to Monitor an Oracle Database

Michael S. Abbey
Ntirety
michael.abbey@ntirety.com

The number of ways to monitor a database is as plentiful as databases there are out there. The approach I use to "baby-sit" an Oracle database has proven to be fundamentally different than many other practitioners. The mantra I use is simple:

If you cannot fix it immediately, do not page … simply email!!!

It is the responsibility of administrators to inform the keepers of their databases about events and conditions that may be service affecting; to alert and attend to all errors encountered, in my opinion, is unnecessary and serves no real purpose. This paper and the presentation at the September 10 meeting go hand in hand. We discuss the following in this paper:

- alert log monitoring
- rationale for paging oncall personnel
- how often to run monitoring scripts

and look into other areas of interest in the PowerPoint presentation. Let's get started by looking at the source of most errors detected while an Oracle database runs … the instance alert log.

All of what I discuss in this paper and the accompanying presentation is outside the OEM/Database Control umbrella. My decades plus experience with these GUI-based solutions is that they provide a way too much information and are not easily configurable to be trained to deliver the alerts that really mean something to the smooth operation of the Oracle database.

Alert Log Monitoring

My nickname for this text file is Oracle's *over-alert log*. Whilst most of the information deposited in the file is of interest to technical personnel, painstakingly alerting on every *ORA-* line in this file is overkill. The following logic outlines my approach to the alert log:

```
a) if ORA-00600 then
b)    if accompanied by data corruption message(s) then
c)       page oncall personnel
d)    else
e)       email appropriate personnel
f)    end if
g) elsif ORA-07445 then
h)    email appropriate personnel
i) elsif error contained in paging text file then
j)    page oncall personnel
k) elsif error in email text file then
l)    email appropriate personnel
m) else
n)    ignore error
o) end if
```

Now for the inevitable discussion points on the above logic:

	Dialogue
a)	Often, Oracle deposits error message numbers into the alert log using *non-standard* formats. The "normal" mask for messages is ORA-*nnnnn*, with numbers less than 10000 zero-padded to the left to make them 5 digits. Thus, discovering ORA-01149 and ORA-1149, and identifying them as the same error is prudent.

In most cases, the arguments accompanying the ORA-00600 message gives adequate information to guide the DBA on the attention they require. MetaLink provides a robust lookup utility for these messages, located at this location.[8] Many DBAs end up devoting a great deal of time investigating 600's and at the end of the journey, still struggle with the disposition of errors and importance of many arguments.

b) The evidence of data corruption does not always jump out at you. It is often buried in the few lines of text following the error message. On many occasions, the applications will return information to the user session that may be more informative and timely than the alert log occurrence. Almost always, data corruption errors are accompanied by ORA-01578 and information that immediately points you at the offending location in the database:

ORA-01578: Oracle data block corrupted (file # 59, block # 763411)

c) Going from Oracle V6 to Oracle7 (circa 1993), we noticed the plethora of ORA-00600 errors drop dramatically. Upon further investigation, we encountered a new suite of arguments accompanying this pesky ORA-07445. Suffice to say, similar to its ORA-00600 counterpart, close to all the occurrences of the 7445 are not service-affecting and should be treated following the *Forrest Gump* theory.[9]

e) When deciding to even pursue this email route, weigh the likelihood that the recipient(s) will even see and read the material. When management personnel receive 3,219 emails on the average day, how likely is it that the ORA-00600 or 07445 notification will serve any purpose?

i) This *paging text file* is created in house, and contains ORA- message numbers that are of interest to an alert log monitoring routine. Keep in mind that most, if not all, messages are prefixed by the text ORA- though you may also encounter prefixes *TNS-* and *RMAN-*.

k) Same note as point i).

To Alert or Not to Alert: That is the Question

Regardless of whether you are the recipient of alerts at 3:15AM or not, the following discussion is pertinent. If you are "unlucky" enough to be that person, extraneous alerts in the wee hours of the morning contribute to employee burnout and a premature MFDJ disorder (Must Find a Day Job). My mantra for alerting is a follows:

If something needs immediate attention and, if not dealt with, the smooth running of the database is in jeopardy … ALERT, otherwise INFORM.

In other words, if one can answer *yes* to ANY of the following questions, then an issue should be brought to the immediate attention of the oncall administrator:

1. Is the database down—this often ends up being a false alarm; better safe than sorry.
2. Is the monitorer incapable of connecting to the database to carry out its work—this could be indicative of a down listener or some other sort of network glitch that is keeping connection requests from being brokered successfully.
3. Did a looming/actual space deficiency get detected that could/did cause transaction failure—the best-case scenario is for the monitorer to catch space related issues before application interaction. Would you not rather head off these

[8] For those reading hard copy, enter document ID 153788.1 into an Advanced Search at MetaLink to access this utility.

[9] Ask me in NYC if interested, and not familiar with this nomenclature.

issues pro-actively rather than your perennially cranky CFO?[10]

4. Has the database failed to communicate with your administration systems for more than the length of time negotiated with the application owners—whether it's a formal service level agreement or a loose understanding between you and your users, regular communication success is required to facilitate event communication.[11]
5. Has an event been detected that, if not attended to, could lead to a service outage—this is indeed the catch-all question whose answer should be negotiated between DBA and application owner personnel.
6. Is the issue something that can be fixed immediately—if it can't be fixed at 3:15AM, notification can wait until morning.
7. Do the recipients of the alert have the "authority" to implement a fix and inform interested parties of what they did—if a DBA needs to "get permission" to do something and a person can be contacted for the go-ahead, by all means page 24x7.

How Often Should Monitoring Scripts Run

There are a few factors that influence this choice of frequency:

- What have you promised—within reason, you must commit to a number of checks per hour upon which you can deliver. Looking ahead into an ever-increasing number of databases being monitored, it may be realistic to start out doing 10 cycles of checking every hour; that may work when you are taking care of 18 database. How about (best-case scenario) two or three years down the road when you are looking after 612!
- The DBA-to-database ratio of your team—if you divide the number of databases by the number of DBAs today and arrive at something like 25, you then need some metrics on who is occupying the most of your personnel's time. The more often monitoring scripts run, the higher the likelihood of more than one emergency surfacing at the same time. Smart techies understand that the best way to triage database issues is *slowly*, *carefully*, and *methodically* according to skills you have gleaned from previous interventions. Try applying those three adverbs to this situation:
  - 07:19PM—4 production databases at TGR page that they are down
  - 07:21PM—connectivity issues reported at JAM's two quality assurance databases
  - 07:24PM—74 of the 168 databases that you monitor fail to communicate with your internal system for the 3rd time in the past 30 minutes
- Those three adverbs, under the stress of receiving 80 pages in a five minute period??? I don't think so.
- Delays that enter into alert processing—there are a few items that influence how rapidly you can respond to alerts:
  a. Network lag in the transport mechanism between the server being monitored and your internal systems.
  b. Whether the monitoring scripts are trained to *not run a second copy* of themselves if one is already running. Suppose someone insists you run eight checks on their databases every five minutes. If those checks normally take longer than five minutes, successive runs could collide with one another.[12]
  c. How many alerts are being ingested by your internal server and how robust it is in processing many hundreds of items that may be arriving every few minutes.

From my past experiences, a doable and deliverable frequency is either three or four times per hour. Also, if a server supports more than one database, ensure the monitoring runs are nicely staggered as shown next (the cells underneath each database name show the minutes past the hour each check should run):

	prd	***tst***	***qau***	***nau***
alert log	0,15,30,45	4,19,34,49	8,23,38,53	12,27,42,57
next extent	10,40	15,45	25,55	5,35
max extents	9	24	39	54
listener	2	17	32	47

[10] Not to be construed as suggesting your CFO is one of these ☺ …

[11] Even more important is ensuring that if the monitorer has not contacted your systems for *x* minutes, an alert is delivered notifying this missing communication.

[12] How could a check take more than five minutes to run … try looking for objects that may be unable to extend themselves in an Oracle Applications environment with thousands and thousands of objects.

invalid objects	3	18	33	48

Jobs are set up in UNIX from the crontab and Windows using the Task Scheduler. Setting these up to run every 10 minutes for example on Windows is an interesting feat; when defining the task, do the following:

1. click Advanced in the Schedule tab
2. click in the Repeat Task checkbox
3. fill in the interval beside Every (e.g., **10** minutes)
4. for Duration, enter **24 hours**

There are a few other factors that influence how often you should run your monitoring scripts:

- Does the client find the time to get back to you—often issues that we are expected to monitor command little or no attention from the user community. Continual attempts to contact the individuals identified in a database escalation matrix go nowhere. In situations where we are not given carte blanche "fix these problems and let us know via email", efforts are derailed by inability to communicate. You have to find some way to adjust your alert notifications so as not to waste your time (and rob sleep from your employees).
- The speed with which the client gets back to you when contacted—as individuals are identified in the escalation matrix, how long (if at all) does it take people to get back to you. At the start of a contract, it can be a give-and-take process agreeing with the client about what needs to be monitored in the first place. Critical situations have been identified and contact information for personnel recorded. The auspices under which you will be contacting them before implementing solutions has been decided.

Picture the following all too frequent occurrence:

- it's 3:30AM, an alert has just arrived
- its content falls into the contact-the-client-before-fixing bucket
- contact is initiated and you are sent to voice mail
- you update the ticket accordingly
- you go to the second contact individual
- at 3:45AM, the same alert arrives
- …
- …

By 4:30AM, five alerts have arrived for the same issue, and still nobody to speak with.[13] Suppose a new database is coming online, the key client personnel having been identified. Regardless of what you normally do with alerting, the clients insist they want you to alert on every ORA-00060[14]. You negotiate, but the people will not budge … after all they are the client. ORA-00060 alerts arrive at 3:05AM, 1:20AM, 5:19AM, and 6:12AM over the first three days of monitoring. Each alert is followed by a call to the main client contact's cell. The first call yields a "go-ahead" to do whatever it needs to fix the issue, as long as the smooth operation of the production database is not compromised. The second leads to "permission" from a very groggy contact person, difficult to hear anything on that person's cell due to what seems to be a screaming infant in the background. The third call goes un-answered, and the fourth goes nowhere as the reception is so poor no information is exchanged. On the first working day after the 4th occurrence, you have email and voice mail from the client suggesting the paging and contact rules of engagement need to be re-visited.
The ORA-00060 I used deliberately here as it falls into a very grey area; 99 times out of 100 (if not more), the fix for this error will not be delivered until days or weeks after the alert arrives:

```
ORA-00060: deadlock detected while waiting for resource
Cause: Transactions deadlocked one another while waiting for resources.
```

[13] Naturally, a mature system has a hook to suspend monitoring for entities over a user-specified time period once the initial alert has arrived.

[14] Deadlock detected

```
Action: Look at the trace file to see the transactions and resources involved. Retry if
necessary.
```

As well, when one consults the trace file the error suggests you might want to look at, therein is contained a disclaimer from the vendor's pseudo legal counsel claiming this is not a database error but rather a programming problem.

Wrapup

My suggestions and approaches to database monitoring may be familiar to some readers. Others may suggest that systematic filtering of errors is reckless and prone to missing events that interfere with normal day-to-day operations of the Oracle database. On the contrary, experience has shown that attention to real or perceived service affecting problems is the only approach that is both supportable and reasonable. Exceptional situations that mean "the world" do users of database A may mean nothing to applications hitting database B. This drives my approach to monitoring the database after over 20 years living and breathing the Oracle software. Bottom-line … implement a monitoring strategy that serves the needs of the database users and adopt a strategy that sets reasonable and obtainable monitoring goals.

Michael Abbey has been a regular presenter at user group and vendor-sponsored events for north of 18 years. His areas of expertise centre on installation, upgrades, backup/recovery, and data organization. He is an avid volunteer in the Oracle user group programs, and keynotes regularly at a handful of regional events. He has experienced success with the Oracle Press offering "The Beginner's Guide" series with Ian Abramson and Mike Corey.

Introduction to Java — PL/SQL Developers Take Heart!

Peter Koletzke, Quovera

Out beyond ideas of wrongdoing and rightdoing there is a field.
I will meet you there.
When the soul lies down in that grass
the world is too full to talk about.
Ideas, language, even the phrase "each other"
doesn't make any sense.

—Jelaluddin Rumi (1207–1273), *A Great Wagon*,
(translated by Coleman Barks)

If you are a currently a PL/SQL developer or live or work close to such a person, you may have become aware of Oracle's current focus on Java. You know that Oracle has stated repeatedly that PL/SQL is here for the long term, but you also watch as Oracle implements more and more Java features in the database, application server, and development tools. All of this may have led you to the conclusion that it is now time to learn Java.

As a PL/SQL enthusiast, your first view of Java may be a bit discouraging because its object-oriented core makes it look very different. Also, you may be trying to sort from all the marketing hype basic concepts about Java's strengths and weaknesses and where it fits in the industry.

Demystifying the unknown usually helps. Therefore, an objective of this paper is to explain the basic concepts of and terms used in Java. When learning a new language, it is usually only necessary to learn the following:

- **The syntax for program control** such as iteration and conditional statements
- **How programs code is organized**, compiled, distributed, and run
- **The built in datatypes** offered by the language and how to declare and use variables

When you are learning Java, it is still necessary to understand these things. However, if you have not come in contact with object orientation (OO) concepts while developing code, when learning Java you also need to learn how OO works because Java is an OO language. This paper starts with a discussion of why you would use Java and how object orientation works. Often language explanations are a bit dry, so this paper then takes a slightly non-traditional route to explaining Java language elements. Assuming that you know a language such as PL/SQL as well as programming logic and can recognize parallels and syntax variations in another language, this paper shows an annotated example of the code and documents each line. This section acts as a guided tour of most key Java elements. The paper then discusses other elements necessary to understand Java.

Note: This white paper was adapted from the material in the *Oracle JDeveloper 10g Handbook*; McGraw-Hill/Osborne, 2004; by Dr. Avrom Roy-Faderman, Peter Koletzke, and Dr. Paul Dorsey.

Note

Naturally, it is not possible to explain an entire language in a short white paper such as this. To complete your understanding of Java, you will need further study, through either self-directed or formal training. A good resource for more information is the no-cost Java Tutorial available at Sun Microsystem's website, java.sun.com.

Why Java?

Java is a relatively new, object-oriented language (officially launched in 1995) that provides many ways to deploy the code. Object orientation offers benefits in analysis and design because business concepts are more easily matched with objects than with standard relational structures. These concepts map easily to programming elements in an object-oriented language such as Java.

If you are in the process of evaluating the Java language for use in a production environment, you need to consider both its benefits and drawbacks as well as what you will need to make the transition.

Benefits

The IT industry is proceeding at a breakneck speed into Java technologies (primarily Java 2 Platform, Enterprise Edition) because of the perceived benefits. It is useful to examine some of the main strengths that Java offers.

Flexibility

Java is implemented as a rich set of core libraries that you can easily extend because the language is object oriented. Distributing these extensions is a normal and supported part of working with Java.

Java supports light-client applications (through technologies such as JavaServer Pages), which only require a browser on the client side. Running the client in a browser virtually eliminates runtime installation and maintenance concerns, which were a stumbling point with client/server application environments such as Forms (before it could be web deployed). Java also supports deployment as a standalone application with a Java Virtual Machine (JVM) runtime on the client. It solves the problem of supporting different screen resolutions with layout managers that are part of the core libraries.

The Java language is a core component of standards such as Java Platform, Enterprise Edition (Java EE, formerly known as "J2EE"). It is used as the language in which basic libraries, such as those from which JavaServer Pages (JSP) code is built, are written. In addition, you can embed Java code snippets inside JSP files to perform actions specific to the web page.

A popular Java EE design pattern, Model-View-Controller (MVC) defines layers of application code that can be swapped out when the needs of the enterprise change. With MVC, the view (presentation) layer can be implemented as a Java application running on the client with Swing components or as HTML elements presented in a browser. Although these views are different, they can share the same model (data definition and access) and controller (behavior and operation) layers. This kind of flexibility is a benefit.

Current strategies for deployment of Java code emphasize multi-tier architectures that provide one or more application servers in addition to client and database server tiers. Although this feature is not unique to Java environments, it is one of the main design features of current Java web architectures. The application server approach offers flexibility and better scalability as the enterprise grows. For example, to add support for more clients, it is only necessary to add application servers and software that distribute the load among servers. The client and database tiers are unaffected by this scaling. A multi-tier approach also offers a central location to support business logic that is common to many applications.

Note

Another characteristic that makes Java attractive is its relative ease of use. For example, the Java runtime automatically handles memory management and garbage collection. Also, Java supports multiple threads so that you can write a program in Java that runs in multiple simultaneous threads of execution.

Wide Support from Vendors

A compelling reason to use Java is that it is supported by many vendors. Instead of one main vendor, as with other technologies (for example, Microsoft .NET Framework), hundreds of companies produce and support Java products. Oracle is one of these companies. Oracle has a large stake in the Java world and continues to offer its customers guidance

and robust product features for Java application development and deployment. Due to this wide support from vendors, the choice of Java as the language is not strongly tied to a single vendor who may not be viable or strong in the future.

Wide Support from Users

Another source of wide support is the user community. Java has a well-established user base that is not necessarily tied to a particular company. The Java community is reminiscent of the early days of Unix, when users made their work available to other users on a not-for-profit basis. The concept of *open source* (www.opensource.org) includes free access to the source code, no-cost licenses, and the ability for others to extend the product. For example, the Linux operating system started and continues to be enhanced through open-source channels.

Although the Java language is not an open-source venture, there are many Java products, such as the Apache web server, that are open-source products. Sample Java code is readily available from many sources on the Internet. In addition, many freeware (with no-cost licenses) or shareware (try before you buy) class libraries are available to Java developers.

Platform Independence

Java source code and runtime library and application files are not specific to a particular operating system. Therefore, you can create and compile Java class (runtime) files in a Windows environment and deploy the same files in a Unix environment without any changes. This aspect of Java, sometimes referred to as *portability*, is important to enterprises that find themselves outgrowing a particular operating environment but that need to support previously created systems in a new environment.

Drawbacks

Many of Java's drawbacks are derived from the same features as its benefits and result from the newness of the language.

Rapidly Changing Environment

The Java environment is less mature than traditional environments that access a relational database. This immaturity has two main effects: frequent updates that add significant new features, and shifts in technologies that occur more rapidly than in traditional environments. For example, when Java was first released, the main deployment environment was within a Java Virtual Machine (JVM) running on the client machine. As that environment matured, there were features added and features *deprecated* (supported, but specially marked as being removed or replaced in future releases).

In addition to updates in the language, additional technologies were added to the mix. Associated specifications such as Java Database Connectivity (JDBC), portlets and portals, and wireless Java guided how Java was used. Different environments were also developed. For example, in addition to the environment of Java running on the client, there are now many variations on web-deploying a system developed in Java. In fact, the Java web-deployment landscape is so complex that as part of the Java EE specifications, Sun Microsystems has created *blueprints* (called BluePrints*),* which are descriptions of proven techniques and best practices for deploying applications. Java EE also includes descriptions of proven, lower-level coding techniques called *design patterns* that are used as additional guidelines for development.

Multi-Vendor Support

Although multi-vendor support was listed as one of Java's strengths, it can also be thought of as a drawback. You may need to merge Java technologies from different vendors, and each vendor is responsible only for their part. Oracle offers a complete solution for development (JDeveloper) and deployment (Oracle Application Server), but you may find yourself in a multi-vendor situation if Oracle products were not selected or were extended with components from other vendors. In addition, Oracle is not responsible for the base Java language. In that respect, a Java environment will always be multi-vendor because Sun Microsystems, which is responsible for the language, does not offer a complete solution including a database.

Significant Language Skills Required

Java developers need to think in an object-oriented way as well as to understand all aspects of the language and how the code pieces tie together in a production application. Java coding is largely a 3GL effort at this point. The IDEs assist by generating starting code and providing frameworks (such as Oracle's Application Development Framework), but

developers also need to have solid programming skills to effectively create production-level Java programs. Java is a popular language now with colleges and universities, so many new graduates are well trained in Java.
In addition, for web-deployed Java, developers need to have skills in other languages and technologies such as HTML, XML, JavaScript, and JSP tags. These skills are easily obtained, but are essentially prerequisites to effective work in the Java web environment.
If developers are to be completely effective, they must also have solid knowledge of database languages. Tools such as JDeveloper's Application Development Framework Business Components (ADF BC) and other technologies (such as Oracle Application Server TopLink) hide the SQL statements from the developers. However, developers must be aware of how the SQL statements are produced by the tools so that the statements can be as efficient as possible.
In addition, although database-stored code (packages, procedures, functions, and triggers) can be coded in Java, developers may still need to interface with existing code built in PL/SQL and, therefore, will need to understand some PL/SQL.

Transitioning to Java

Working in a Java environment is very different from working with traditional database development environments such as Oracle Forms Developer or Visual Basic (VB). If you are not already using Java but the benefits of Java have convinced you of the need to make the transition, you will need to plan that transition carefully.
It will take time to learn the nuances of a Java environment. If you are committed to creating an organization-wide Java environment, building a traditional client/server application using a local Java client (Java running on the desktop) may still make sense if your application is used in a small group or departmental situation. Coding and deploying the application locally on the desktop in this way postpones the complexities of working with web deployments and can give you a feel for how Java works.
If your development team has skills in other languages, Java will require retraining and ramp-up time. Building client/server applications directly in Java can be a good first step. This method leverages the improved flexibility of Java and its ability to build sophisticated applications. It also makes the transition of your business to the Web easier because Java is a primary language of the Web. The smaller the application, the easier it will be to concentrate on the language and not the application. A prototype or internal administrative application that will not see extensive use might be a good candidate for this first effort.
Another variation on this transition advice is to develop a small web application in Java. This can be the next step after building a client/server application, or it can be the first step. Web applications add the complexity of application and web servers, and this will give you a taste of this extra layer of software.

Making the Leap

The transition to Java may not need to be (and probably should not be) a big bang where you move all new development to Java and start converting existing applications to Java. Although you want to minimize the number of tools and environments that you support, it is likely that you will have to support existing applications in the environments in which they were written. With all current development tools trying to improve their web-enabled capabilities, it becomes increasingly difficult to make a compelling argument for abandoning these technologies. For example, following a long evolution, Oracle Forms Developer running over the Web is now a stable and viable environment. Especially since you can extend Forms with Java plugins, there may be no reason to convert legacy Forms applications to Java.
Therefore, the best approach to transitioning into the Java environment is to leave core application development in whatever legacy environment you are comfortable with and to build a few systems of limited scope in Java using JDeveloper. Once you have some experience in building and deploying applications, you can make an informed decision about whether your organization is ready to make the transition to an entirely Java-based environment. There may still be good reasons to stay with a legacy environment for core applications and only to use a Java environment for e-commerce and other web-based applications.
As you become more comfortable with working in Java and have more Java projects under way, you can think about migrating current applications. However, some applications may never need to make the transition.

Note

As with many shops that support legacy COBOL-based programs, it is likely that you will have to support your current development environment for large enterprise-wide applications for some time.

Object-Orientation Concepts

If you have experience in the C++ language, you will notice keyword and syntax similarities between Java and C++. However, there are enough differences between Java and C++ that it is worth reviewing the basics of the language even if you understand the concepts behind C++. Java used C++ concepts as a springboard but was designed as an object-oriented language in its first incarnation (unlike C++). Understanding Java requires a comfort level with the concepts of object orientation (OO). If you have any experience with another object-oriented language, such as Smalltalk or C#, you may have already grasped the OO concepts you need to work with Java

The fundamental building block of an object-oriented language like Java is a structure called a *class*. The class acts as a pattern or blueprint from which *objects* are built. In object-speak, "an object is an instance of a class." An object is built from a class and has an identity (or name). This means that, primarily, the class is actually not used as a programmatic element except to create other elements that you will manipulate (assign values to and query values from). For example, an object called "someBox" can be instantiated from a class called "Box." In this example, the someBox object is created from the pattern defined by the Box class.

The concept of a class and its object loosely parallels the concept of datatyping (and record variable definition) in programming languages such as PL/SQL. In PL/SQL, a datatype is a pattern upon which variables are built. If the datatype concept were expressed in object-oriented terms, a variable would be the object that instantiates the datatype (acting as a class). In fact, as you will see later in this paper, Java objects are thought of as being typed from classes. In the Box example, you can say that "someBox is of type Box." In common Java parlance, you might also refer to "someBox" as "the Box" or "the Box object."

The parallel between a PL/SQL variable and a Java object is loose because, in addition to creating and initializing the Java object (as you do a variable in PL/SQL), you need code to "create" the Java object using code (you normally use the new operator to accomplish this creation). The creation operation runs constructor code to assign it some default data as well as to run code specific to the class. This concept will be discussed more a bit later in this paper.

A class contains both data (values in variables, also called *attributes* or *fields*) and behavior or application code logic (in methods). This makes it different from anything in the world of relational databases. The closest concept to the class data and behavior characteristics is a relational table with a dedicated package of procedures that are used for SQL operations such as INSERT, UPDATE, DELETE, and SELECT. In this case, the combination of relational table and procedural package contains data (in the table) and behavior (procedures and functions in the package assigned to the table). Another example in the Oracle relational database paradigm is a database view with a procedural package run by INSTEAD OF triggers. The database view (the data) combines with the package and INSTEAD OF trigger (the behavior) to form a rough equivalent to an object-oriented class.

The difference between this example from the relational/procedural world and the object-oriented paradigm is that there is only a conceptual link or loose coupling between the table and the code package. The table and package exist as separate objects and can be used separately. (Although you could link the table and PL/SQL package using table triggers, this mechanism is not required by or native behavior of the language.) In object orientation, the class is inherently both data and behavior; the link is tight and perfectly integrated. The class is used as a pattern to create objects that contain data and pointers to the code in the class. Figure 1 depicts the coupling difference between data and application code for the relational/procedural and object-oriented paradigms.

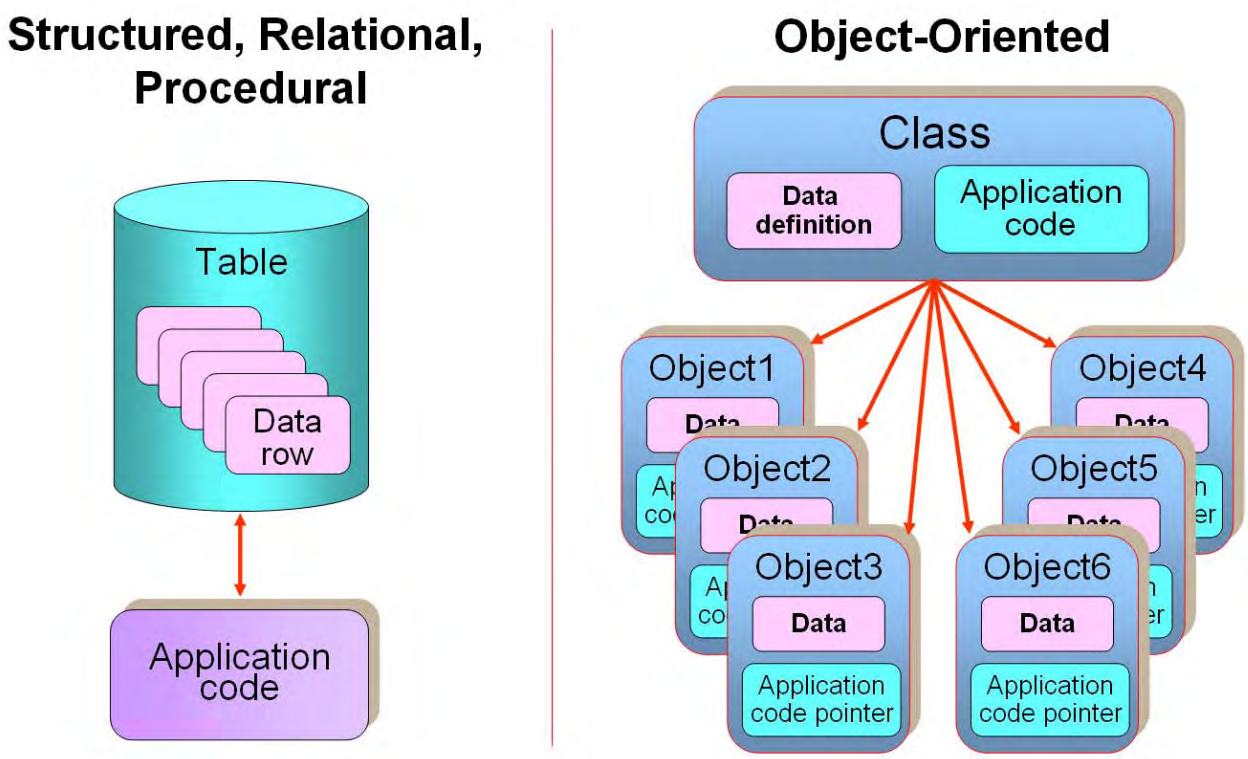


Figure 1: Relational/procedural and object-oriented paradigms

Handling and Storing Data

The ways in which data is handled in an object-oriented language such as Java and in a relational database system are also fundamentally different. Data is not inherently *persistent* (permanently stored) in Java. Therefore, data is available only for the time in which the Java program is running. There are ways to store data in between program sessions; the method included as part of the base language is *object serialization*. Object serialization includes the ability to write object values to, and read object values from, a persistent stream (such as a file).

Object serialization is the built-in Java way to handle persistence. However, many programmers of Java and other languages have become accustomed to using a fully featured relational database management system (RDBMS) to handle data persistence. An RDBMS provides solid facilities for fast and safe storage, retrieval, backup, and recovery of mission-critical data. However, the RDBMS, by definition, is built around the concept of storing data in relational tables. This concept does not correspond to the way in which the Java language handles data in objects. Figure 2 shows a conceptual mapping that you can make between relational and object-oriented data storage.

This diagram shows how a row in a table roughly corresponds to the data in an object. You can describe a table in object-oriented terms as a collection of records representing instances of related data that are defined by the structure of the table. The problem is the difference in the way that data appears in the two paradigms. A table contains rows that are accessible using a relational database language such as SQL, which addresses requests for sets of data to the table. Objects contain data and you address requests for that data directly to the object. You cannot use SQL to access data in an OO environment because the source structure of the data is different; data is distributed across many objects. The standard solution to this mapping problem is to create an array object (often called a *row set*) of values that represents multiple rows in a database table. The row set is a single object with methods for retrieving individual rows and column values.

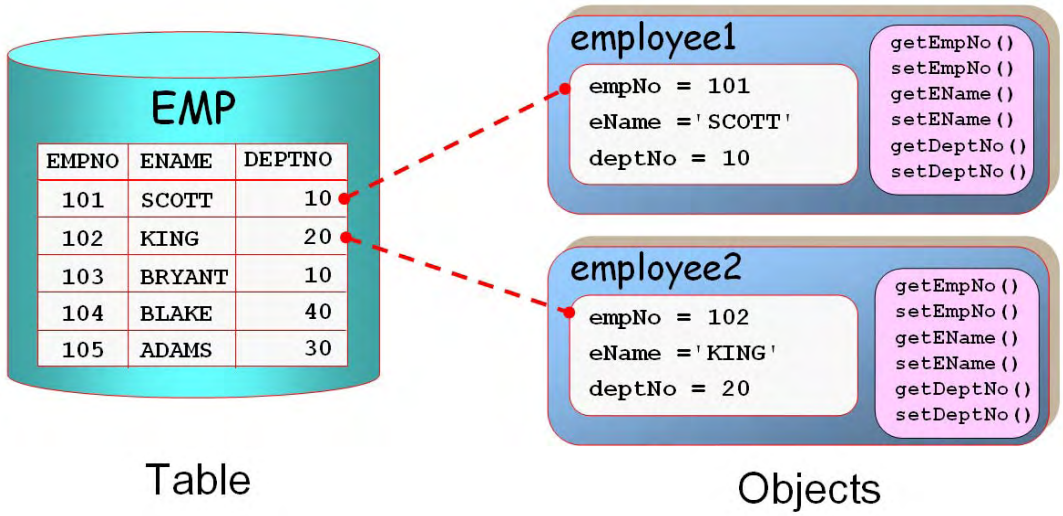


Figure 2: Mapping between relational and object data storage

Therefore, there are basic differences between the relational and the object paradigms in the areas of persistence, data structures and access code, and conceptual foundations. Using Java code to access a relational database is a common requirement, and there are many solutions to making an effective map between relational tables and objects. For example, architectures such as JDeveloper's ADF Business Components or Oracle Application Server TopLink hide the complexity of the relational-object mapping and provide programmers with object-oriented, Java-friendly structures that can be easily incorporated into application programs. Other strategies, such as JDBC, also ease the burden of accessing a relational database from an object-oriented language.

Inheritance

One of the key characteristics of object orientation is the ability of a class to automatically take on the attributes and methods of another class. This is the concept of *inheritance*, where one class is a parent of another. The parent, also called the *superclass* (base class or generalization), contains elements (data and behavior) that are available in the *subclass* (child class or specialization). The lines of inheritance can be deep, with one class acting as the grandparent or great-grandparent of another. To base a class on a parent class, you *extend* the parent class. The child class can then supplement, modify, or disable the attributes and behavior of the parent class. This kind of inheritance is shown in an annotated code example later in this paper.

Other Primary Object-Oriented Concepts

Two other major concepts are inherent to object orientation—*polymorphism* and *encapsulation*. Although they are key to object orientation and it is necessary to understand them when you are developing production systems using Java, you do not need to understand them fully to begin working with Java. The following provides a brief definition of these concepts:

- **Polymorphism** The ability of a class to modify or override inherited attributes or behavior. This is a key feature of the Java programming language allowing the developer to create template classes as well as extensions to these classes, which may (but are not required to) inherit attributes and behavior from the generalization (master) class.
- **Encapsulation** Only the important characteristics of an object are revealed; the internals are hidden. Encapsulation is accomplished in Java by means of access modifiers (explained later in this paper).

Java Language Overview

High-level, theoretical discussions of object orientation often glaze the eyes of the audience. The theory makes more sense when it is demonstrated using some code examples. The following section shows a code sample and explains its contents to demonstrate object orientation and Java language concepts. Even if you do not have extensive Java experience but have been exposed to other programming languages, you will be able to identify some of the language elements as well as the

structure of a typical source code file. The example also demonstrates, in the context of the Java language, some of the key object-oriented concepts just explained.

Annotated Java Code Example

All Java code is contained in class files. A class file is made up of a number of standard elements. The following is a representative class file that contains standard elements. The line numbers in the code listing serve as reference points and would not appear in the actual code file.

```
01: package shapes;
02: import java.lang.*;
03:
04: /*
05:    This class defines a shape with three dimensions
06: */
07: public class Box extends Shape {
08:   int height;   // override the Shape height
09:   int depth;    // unique to Box
10:
11:   public Box() {
12:     height = 4;
13:     width = 3;   // width is inherited from Shape
14:     depth = 2;
15:   }
16:
17:  public int getDepth() {
18:    return depth;
19:  }
20:
21:  public void setDepth(int newDepth) {
22:    depth = newDepth;
23:  }
24:
25:   // super.getWidth is the same as getWidth() here
26:   public int getVolume() {
27:     return height * super.getWidth() * getDepth();
28:   }
29: }
```

Note

All code in the Java language is case-sensitive. Therefore, a class called “BOX” is different from a class called “Box.” By convention, class names are mixed case, with each word in the name initial-capped. For example, a class that defines salary history would be called “SalaryHistory.”

Package Declaration

Line 01 identifies the location of this file. *Packages* are collections of class files in subdirectories in a file system. A package normally represents a directory in the file system. Packages of files can be archived into a .zip or .jar (*Java Archive* or *JAR*) file, and the Java runtime can search in this archive (sometimes called a *library*) for a specific class file. If a package is archived, the archive instead of the file system then contains the directory structure.

The *CLASSPATH* operating system environment variable contains a list of these archive files separated by colons (for Unix) or semicolons (for Windows), for example:

```
C:\JDev10g\jdk\jre\lib\rt.jar;C:\JDev10g\jdk\jre\lib\jce.jar;C:\JDev10g\
jdk\jre\lib\charsets.jar;C:\JDev10g\jdk\jre\classes;C:\JDev10g\jdev\lib\
```

```
jdev-rt.jar
```

(The path is a single variable value entered on one line.).
Whether a particular file is inside a file system directory or a directory inside an archive file is irrelevant, but the CLASSPATH must include the archive file name if the file required is in an archived directory. The CLASSPATH must include the name of the file system directory if the file required is in a file system directory.
Line 01 ends with a semicolon, as do all Java statements.

> **Note**
>
> You can view the contents of JAR files by using any file decompression utility.

Import

Line 02 defines an *import* (called an "include" in other languages such as C++), specifying the external or library classes required for the code to compile and execute. Imports are identified by class name as well as by the package in which they reside. Many import statements may appear if more than one package structure or class is required. The import statement can reference a single class file or an entire package as in line 02. The directory is listed using fully qualified dot syntax (package.subpackage.subpackage.*), with "*" indicating that all classes in that package will be available. Java libraries are often grouped by function into the same directory (package) so that associated functions can be called more easily. This example is provided for discussion purposes. The java.lang classes are automatically available without an import statement.

Comment

Lines 04–06 show a multi-line *comment* (using the beginning and ending symbols "/*" and "*/"). Line 08 shows a single-line comment (using the beginning symbol "//") at the end of the code line. As shown in line 25, this style of comment can be used on a line by itself. Java also offers special multi-line comments that start with "/**" and end with "*/" and are used to generate a type of documentation called "Javadoc." The javadoc program extracts these special comments and presents them in a formatted HTML file.

Class Declaration

Line 07 is the class declaration. It includes the keyword `public`, indicating that this class is available to all other classes. The keyword `public` is called an *access modifier* (or *access specifier*). Other choices for access modifiers are `private` (where access to the class member is limited to other members of the same class) and `protected` (where you cannot access the class from outside the package unless the calling class is a subclass). If you do not use a modifier keyword, this indicates the *default* modifier, where members are available only to code in the same package.

Code Block

The end of line 07 contains an opening curly bracket "{" indicating the start of a *block of code*. Between this and the matched closing curly bracket "}" are code elements and other blocks of code. The code blocks do not need to contain any code (unlike blocks in PL/SQL). Blocks of Java code may be nested. Blocks define the scope of variables and code structures such as `if` and `while`, as discussed later.

Using Brackets for Code Blocks

It is good coding practice to always use curly brackets to contain code in code structures (such as `if` and `for`). Technically, you do not need curly brackets if there is only one statement to execute as in the following example:

```
if (width == 10)
  depth = 20;
```

However, it is easy to make the mistake of adding a line of code under an `if` statement that has no curly brackets and assume that it will execute conditionally. Since only the first line of code under the `if` statement is part of the conditional logic, the next statement would always be executed if no curly brackets contain the statements, for example:

```
if (width == 10)
   depth = 20;
   height = 30; // this will always execute
```

Always using curly brackets, even for single-line blocks, will prevent this type of error as in the following example:

```
if (width == 10)
{
   depth = 20;
}
```

> **Note**
>
> No recognized standard exists for whether to place the starting bracket for a block at the end of the line above or on a new line. This paper shows examples of both techniques but for consistency and readability, you will want to make a particular starting bracket style a standard for your application code.

Subclassing (extends)

This code defines a class called `Box` that is built (subclassed) from a class called `Shape`. The `extends` keyword declares that `Shape` is the parent of `Box` and defines the inheritance for `Box`. The `Shape` class might be defined as follows:

```
package shapes;

public class Shape  {
  int height;
  int width;

  public Shape() {
    height = 1;
    width = 1;
  }

  public int getHeight() {
    return height;
  }

  public void setHeight(int newHeight) {
    height = newHeight;
  }
  // getWidth() and setWidth() methods go here
}
```

> **Note**
>
> A Java source code file can contain only one public class. The file name is the name of the public class in that file (with a .java extension).

Variable Declaration

Lines 08–09 (of the `Box` class) declare two variables (as `int` types). These constitute the data (attribute or field) for the class that was mentioned in the discussion of object-oriented concepts. Since the variables of the parent class are available

in the child class, the `Shape` variables `height` and `width` are available to the `Box` class. In addition, the `Box` class declares its own `height` variable. This variable is available to objects created from the `Box` class. The parent's `height` variable is also available using the symbol `super.height` (the `height` variable of the `Shape` superclass). This code also declares a variable that is not in the parent: `depth`. Variable names are formatted with initial caps for all words except the first.

Note

The modifier "super" can be also be used in the child class to access methods from the parent class. For example, the `Shape` class contains a `setHeight()` method, which is available to the `Box` class (subclass) as `super.setHeight()`. This modifier can also be used for constants and variables.

Technically, *variables* in Java are declared using primitive datatypes such as `int`, `float`, or `boolean`. Other data resides inside objects that are instantiations of classes such as `String` or `StringBuffer`. Datatypes are discussed in more detail later in this paper.

Code Unit—Method and Constructor

Lines 11–28 define methods and constructors, which are the main containers for functional code in the class.

Method

The standard unit of code in Java is called a *method.* The first line of the method is called the *method signature* or *signature* because it contains the unique characteristics of this code unit such as the arguments, access modifier, and return type.

You can use the same name for more than one method in the class if the methods with that name all have different argument lists (for example, `showWidth(int width)` and `showWidth(String widthUnit)`). Methods with the same name but different arguments are referred to as *overloaded methods*.

Methods implement the object-oriented concept of behavior in a class. Java does not distinguish between code units such as functions that return a value and procedures that do not return a value. It has only one unit of code—the method. Methods must declare a return type. They can return a primitive or class—as with functions in other languages—or they can return *void* (nothing)—as with procedures in other languages. Methods can only return one thing, but that thing could be an array or an object, which could be made up of many values.

Method names are, by convention, mixed case, with each word except for the first one initial-capped.

Constructor

Lines 11–15 define a unit of code called "Box," which has the same name as the class. This unit is called a *constructor* and is not a method. Constructors have a signature similar to methods, but do not return anything (not even `void`). If a return type is declared, the signature identifies a method not a constructor. Constructors are used to create an object that is based on the class (using the keyword `new`); in this case, the constructor just sets values for the variables in the object that is created from the class. The constructor code is contained within a block of code delimited by curly brackets. If you do not define a constructor, you can still create an object from the class because there is an implicit constructor that is available to calling programs.

Constructors must have the same name as the class. Since class names usually begin with an initial-capped word, the constructor also begins with an initial-capped word.

Accessor (Getter and Setter)

Lines 17–28 define getters and setters (also called *accessors* or *accessor methods*). A *getter* is a method that is usually named with a "get" prefix and variable name suffix. It returns the value of the variable for which it is named. Although this is a standard and expected method that you would write for each property or variable that you want to expose, it is not required for variables that you want to hide. Getters may include security features to restrict access to the values and may not be as simple as this example, which just returns the value of a single variable. In this example, the `getDepth()`

method returns the value of the variable, and the `getVolume()` method calculates the volume based on the three dimensions of the box. As with the variable prefix "super." before, this method references the superclass methods getWidth() and getDepth().

> **Note**
>
> For readability, getter methods for boolean variables are usually prefixed with "is" instead of "get." For example, the boolean variable `hasWheels` would be accessed using a getter `isHasWheels()`.

A *setter* is a method that is named with a "set" prefix and is normally used to change the value of the property for which it is named. As with the getter, it is normal and expected that you would write a setter for variables you want to expose to the caller; you would omit the setter if you did not want the variable changed. Setters can include validation logic (for example, not allowing the height to be set to a negative number), but they usually also assign a new value to the variable as in this example.

The object-oriented concept of encapsulation is implemented here by accessor methods that read and write to private variables in the class. Other code outside the class cannot see or manipulate the private variables in this class directly, but must go through the getters and setters to retrieve and change data values, respectively. The accessor methods can have specific logic that protects the variable values, for example, a setter called `setHeight()` could enforce the rule that the value for height may not be less than 0. If the variable were not private, the caller would be responsible for knowing and complying with the rule.

In this example, the `Box` class contains no setters and getters for the `height` and `width` variables. These would be defined in the parent class, `Shape`, and are available to `Box`.

Line 29 is the closing bracket for the class definition.

> **Note**
>
> Java is a case-sensitive language. If you are transitioning from a non-case-sensitive language, it is useful to keep reminding yourself of this fact.

Annotated Use of the Box Example Class

Code that uses a class such as `Box` demonstrates some other principles of the Java language. Consider the following example usage:

```
01: package shapes;
02:
03: public class TestBox  {
04:
05:   public static void main(String[] args) {
06:     // this creates an object called someBox
07:     Box someBox = new Box();
08:     someBox.setDepth(3);
09:     // getHeight() and setHeight() are from Shape
10:     // height shows the height variable from Box
11:     System.out.println (
12:       "The height of Shape is " + someBox.getHeight() +
13:       " and of someBox is " + someBox.height);
14:
15:     // getDepth and getVolume are from Box
16:     System.out.println (
17:       "The depth of someBox is " + someBox.getDepth() +
18:       " and the volume of someBox is " + someBox.getVolume());
```

```
19:   }
20: }
```

The output for the main method will be the following:

```
The height of Shape is 1 and of someBox is 4
The depth of someBox is 3 and the volume of someBox is 36
```

Lines 01–03 state the package and declare the class `TestBox`.

main() Method

Lines 05–19 define a method called `main()`. This is a specially named method that executes automatically when the JVM runs the class from the command line (for example: `java.exe client.TestBox`). The `main()` method can contain any accessible code; in this case, it shows messages in the Java console (that displays in the JDeveloper Log window).

The keyword `static` indicates that `main()` is a *class method* that can be run without declaring an instance of the class. Normally, you need to create an object and then call the method by prefixing it with the object name. With static methods, you can still run the method (in this example, `main()`) without having to create an object from the class.

The `main()` method signature also includes an argument within the parentheses that follow the method name. This argument is typed as a String and is named "args." The common parlance for expressing object type uses the datatype name, for example, "args is a String" or "args is [an instance] of the String type (or class)." (The words in square brackets are optional.)

The square brackets [ ] after args indicate that it is an array. *Arrays* are collections of similar objects. The String array args is used to pass any command-line arguments available when the class is executed. You can also use the expression "`String args[]`" to represent an array of strings.

Object Creation

Line 07 creates an object called someBox based on the `Box` class. The object is made from the `Box` class and, therefore, has the same variables (such as `someBox.width`) and methods (such as `someBox.getVolume()`) as the class. Line 07 accomplishes two tasks: it declares an object of type `Box` and creates the object by calling the constructor `Box()`. This line could also be expanded into the following two lines to separate the tasks:

```
Box someBox;
someBox = new Box();
```

Assign Values

Line 08 sets the value of the depth variable using the setDepth() method. This overwrites the value set in the constructor.

Console Output

Lines 11–13 output a message (using the `System.out.println()` method) that will be displayed in the Java console window. If you run a Java program from the command line, the Java console window will be the command-line window. If you run a Java program in JDeveloper, the message will appear in the Log window. You can concatenate literal strings (in quotes) and variables with the "+" operator regardless of type.

Variable and Accessor Usage

Line 12 references `getHeight()`, which is a method from the `Shape` class—the parent of `Box`, the class from which someBox is built. Since this method displays the value of the `height` variable in `Shape`, the value will be 1 (the default for that class). This demonstrates that you can call a method of a parent class from an object created from the subclass.

Line 13 references the `height` variable of `someBox` (which was built from the `Box` class). In this case, the `height` variable will be displayed as the default from the `Box` class ("4"). This output could be a bit confusing, and this system would probably not be used outside of demonstration purposes because the height of the parent class is set differently from the height of the subclass.

Lines 16–18 display the results of `someBox` method calls. Both `getDepth()` and `getVolume()` are declared in the `Box` class and will be output as 3 and 36, respectively.

Lines 19 and 20 close the method and class.

> **Note**
>
> When naming Java elements, you may use a combination of uppercase and lowercase letters, numbers, the underscore, and the dollar sign. However, you may not begin names with a number. There is no limit to the number of characters that you can use in a name.

Other Java Language Concepts

There are some other Java language concepts that were not demonstrated in the examples but that are useful to review.

The Code Development and Deployment Process

The typical Java development process, if you are not using an IDE such as JDeveloper, follows:

1. Write a source code file with a text editor, and name it using the name of the class that the file represents and a .java extension, for example, TestBox.java.
2. Compile the source code using the javac.exe executable (included in the Java SDK). If there are no syntax errors, the compiler creates a file with the same name and a class extension, for example, TestBox.class. This binary compiled file (called *bytecode*) is interpreted by the Java runtime engine when the program is executed. Java is compiled in this way, but it is considered an interpreted language. Java programs require a runtime interpreter—the JVM, a component of but often used synonymously with *Java runtime environment* or *JRE*.
3. Test the class file using the java.exe executable, the JVM (also included in the Java SDK). If the Java code is a Java application, the command line is as simple as the following example:
   ```
   java client.TestBox
   ```
4. Repeat steps 1–3 until the program performs as required.
5. Package the program file with the library files that it uses (libraries or classes declared in the import statements at the beginning of the program), and install the package on a client machine that has a Java runtime environment installed (containing the Java runtime engine—java.exe—and the base Java libraries such as java.lang.*).

Although an IDE such as JDeveloper automates many of these steps, the tasks are the same. Also, different types of Java programs have different requirements for the compile and runtime steps, but the concepts are the same. For example, working with JSP files requires the development of a .jsp file that is translated into a .java file and is compiled automatically into a .class file by a special Java runtime engine.

Naming Conventions

Since Java is a case-sensitive language, its keywords must always be entered in the case they are designed. All keywords for the Java syntax are lower case. User-defined symbols such as class, member, and variable names can be used in any case, just so their usage is consistent within the code. For example, if you define a method "getWidth()," it must be used with the same case designation in subsequent code.

Even though Java does not enforce a particular case for user-defined symbols, there are generally recognized that Java programmers used. These standards can be summarized as follows:

- **Package and library** (archive file) names are all lower case.
- **Class names** are mixed case, with each word in the name initial capped. For example, a class that defines salary history would be called "SalaryHistory."
- **Method**, **variable (object), and exception names** are mixed case in the same way but the first character is lower case, for example, "usefulBox."
- **Constant (final variable) names** are all uppercase with underscores between words.

Generally names do not start with an underscore "_" because that syntax is often used for internal purposes.

Control Statements

The idea of control statements is familiar to anyone who has written program code. Learning the Java control structures is usually just a matter of learning a different syntax (unless the other language is C++, which provided some of the syntax used in Java). This section reviews only the basic structures, since most structures are similar to those in other programming languages. You can refer to a standard Java language text to understand the variations and usage requirements for these control statements.

Sequence

One of the main concepts of control statements is sequence, and Java code is executed in the order in which it appears in the file. The method that is executed first varies with the style of Java program; for example, a Java application executes the `main()` method first; a Java applet executes the `init()` method first; JavaServer Pages applications execute a `service()` method first. The commands within these methods are executed in the order in which they appear in the code file. As in other languages, calls to other methods execute the method and return to the line of code after the method call. The keyword `return` jumps out of the current method and returns control to the statement in the calling unit after that method was called (or to the command line if the command line was the caller).

Conditional Branching

Java uses the statements `if-else` and `switch` to branch the code based upon a condition as follows:

```
class ShowQuarter {
  public static void main (String args[]) {
    int taxMonth = 10;
    String taxQuarter;

    if (taxMonth == 1 || taxMonth == 2 || taxMonth == 3) {
        taxQuarter = "1st Quarter";
    } else if (taxMonth == 4 || taxMonth == 5 || taxMonth == 6) {
        taxQuarter = "2nd Quarter";
    // more conditions would appear here
    } else {
        taxQuarter = "Not Valid";
    }
    System.out.println("Your current Tax Quarter is: " + taxQuarter );
  }
}
```

This is a branching statement that uses multiple `if` statements. The "||" symbol is a logical OR operator ("&&" is a logical AND). Logical conditions are enclosed in parentheses. The "= =" symbol is the equality comparison operator. Each condition is followed by a single statement or block of code. As mentioned before, it is a good idea to always define a block of code enclosed in curly brackets under the `if` statement.

The `switch` statement is an alternative to multiple `if` statements that test the same value. The following example could be used instead of the `if-then` example:

```
class ShowQuarter2 {
  public static void main (String args[]) {
    int taxMonth = 10;
    String taxQuarter;
    // The break statement jumps out of the conditional testing
    switch (taxMonth) {
      case 1:    case 2:    case 3:
        taxQuarter = "1st Quarter";
        break;
      case 4:    case 5:    case 6:
        taxQuarter = "2nd Quarter";
        break;
```

```
      // more conditions would appear here
      default:
        taxQuarter = "Not Valid";
    }    // end of the switch
    System.out.println("Your current Tax Quarter is: " + taxQuarter);
  }     // end of the main() method
}       // end of the class
```

Iteration or Looping

There are three loop statements: `for`, `while`, and `do-while`. The `for` loop controls loop iteration by incrementing and testing the value of a variable as shown in the following example:

```
class TestLoops
{
  public static void main (String args[]) {
    for (int i = 1; i <= 10; i++) {
       System.out.println("Loop 1 count is " + i);
    }
  }
}
```

The `while` and `do-while` loops test a condition at the start or end of the loop, respectively. Refer to a Java language reference, such as the Java Tutorial at java.sun.com, for more examples of loop structures.

Exception Handling

Exceptions can occur in the Java runtime environment when undefined conditions are encountered. To catch exceptions, you enclose the code in a block defined by the keywords `try`, `catch`, and (optionally) `finally` as in the following example:

```
public class TestException {
  public static void main(String[] args) {
    int numerator = 5, denominator = 0;
    int ratio;
    try {
      ratio = numerator / denominator ;
      System.out.println("The ratio is " + ratio);
    }
    catch (Exception e)  {
      // This shows an error message on the console
      e.printStackTrace();
    }
    finally {
      System.out.println("The end.");
    }
  }
}
```

If a `finally` block appears, it will be executed regardless of whether an exception is thrown. You may also raise an exception by using the keyword `throw` anywhere in the code.

Note

The preceding example shows how you can declare more than one variable (numerator and denominator in this example) of the same type on the same line.

Variable Scope

Variables can be declared and objects can be created anywhere in a class and are available within the block in which they are declared. For example, the following code shows a variable, `currentSalary`, that is available throughout the `main()` method. Another variable, `currentCommission`, is available only within the `if` block in which it is declared. The last print statement will cause a compilation error because the variable is out of scope for that statement.

```
class TestScope {
  public static void main (String[] args) {
    int currentSalary = 0;
    if (currentSalary < 0) {
      int currentCommission = 10;
      System.out.println("No salary but the commission is " + currentCommission);
    }
    else {
      System.out.println("Salary but no commission.");
    }
    // This will cause a compilation error.
    System.out.println(currentCommission);
  }
}
```

Note

Although Java does not use the concept of a variable declaration section (such as the DECLARE section of PL/SQL), Java variables have the scope of their enclosing curly brackets. It is good programming practice to put all variable declarations at the beginning of their scope. For example, if you are declaring method-wide variables, place their declarations at the beginning of the method. If you are declaring variables with a class scope, place their declarations under the class declaration statement. Positioning the variable declarations in this way makes the code easier to read.

In addition to the scope within a block, variable scope is affected by where and how the variable is declared in the class file. The following example demonstrates these usages:

```
class ShowSalary {
  static int previousSalary = 0;
  int commission = 10;

  public static void main (String[] args) {
    int currentSalary = 100;
    if (currentSalary == 0) {
      System.out.println("There is only a commission.");
    }
    else {
      System.out.println("Current salary is " + currentSalary);
    }
    System.out.println("{Previous salary is " + previousSalary);

    // The following would cause a compile error.
    // System.out.println(commission);
  }
}
```

This example demonstrates three usages for variables—instance variables (`commission`), class variables (`previousSalary`), and local variables (`currentSalary`). Both instance variables and class variables are

categorized as *member variables* because they are members of a class (not within a method or constructor). Member variables are available to any method within the class. Methods are also considered members of a class because the class is the container for the method.

Instance Variables

These variables are created outside of any method. In this example, the variable `commission` is an instance variable. It does not use the keyword `static` in the declaration and is not available to class methods (that are declared with the keyword `static`). Therefore, the variable `commission` is not available to the `main()` method in this example. Instance variables are available to objects created from the class. For example, you could create an object (instance) from this sample class using "`ShowSalary calcSalary = new ShowSalary();`", and the variable `calcSalary.commission` would be available. Each object receives its own copy of the class variable. Therefore, if you instantiate objects `salary1` and `salary2` from the `ShowSalary` class, `salary1.commission` and `salary2.commission` could contain different values.

Class Variables

As with instance variables, class variables, such as `previousSalary` in this example, are declared outside of any method. The difference with class variables is that their declaration includes the `static` keyword and they are available to class methods that are also declared with the keyword `static`. The variable can be used without creating an instance of the class using the syntax "Classname.VariableName" (for example, ShowSalary.previousSalary). There is only one copy of the class variable regardless of the number of objects that have been created from the class. Therefore, if you create `salary1` and `salary2` from the `ShowSalary` class, the same variable `previousSalary` will be available from both objects (as `salary1.previousSalary` and `salary2.previousSalary`). If `salary2` changes the value of this variable, `salary1` will see that new value because there is only one variable. The following is an example that demonstrates this principle:

```
class TestShowSalary
{
  public static void main(String[] args)
  {
    ShowSalary salary1 = new ShowSalary();
    ShowSalary salary2 = new ShowSalary();
    //
    System.out.println("From salary1: " + salary1.previousSalary);
    salary2.previousSalary = 300;
    System.out.println("After salary2 changed it: " + salary1.previousSalary);
  }
}
```

The output from this program follows:

```
From salary1: 0
After salary2 changed it: 300
```

Local Variables

This variable usage is declared inside a method. In the sample `ShowSalary` class, `currentSalary` is a local variable because it is declared inside a method (`main()`). The variable is available only within the scope of that method.

Constants and "final"

A variable can be marked as `final`, which means that its value cannot change. Since you cannot change the value, you must assign a value when you declare the variable. This is similar to the idea of a constant in other languages. The following is an example of a final "variable." Final variable names use all uppercase characters by convention.

```
final int FEET_IN_MILE = 5280;
```

You can also mark methods as `final`, which means that you cannot override the method in a subclass. Thus, if class A has a `final` method b( ), and if class C extends A, then class C cannot override the inherited method b( ) in class C. For example:

```
final int getCommission() {
```

```
}
```
Classes may be marked with `final` to indicate that they cannot be subclassed. That is, no class may extend that class. For example:
```
class final CalcSalary {
}
```

> **Caution**
>
> The keyword `final` stops inheritance (subclassing) of final classes, and the overriding of final methods, but does not stop the overriding of a final variable (constant).

Primitive Datatypes

Variable types fall into two categories: primitive and reference. *Primitive datatypes* can hold only a single value and cannot be passed by reference or pointers. Primitives are not based on classes and therefore have no methods. The primitive datatypes include `boolean` (for true and false values), several number types differentiated by the magnitude and precision of data they can represent (`byte`, `short`, `int`, `long`, `float`, `double`), and `char`.
A `char` is a single-byte number between 0 and 65,536 that is used to represent a single character in the Unicode international character set. A `char` datatype can be assigned in a number of ways as follows:

```
// decimal equivalent of the letter 'a'
char charDecimal = 97;
// using an actual character inside single quotes
char charChar = 'a';
// octal equivalent of the letter 'a'
char charOctal = '\141';
// Hex value for the letter 'a'
char charHex = 0x0061;
// Unicode (hex) value for the letter 'a'
char charUnicode = '\u0061';
```

> **Note**
>
> Assigning values to a byte requires surrounding the value in single quotes(‘). This is the only time that single quotes are used in Java. Double quotes (") are used to define a character string in code.

Reference Datatypes

Reference datatypes represent a memory location for a value or set of values. Since Java does not support pointers or memory addresses, you use the variable name to represent the reference. You can type an object using these reference datatypes, and the object instantiated in this way will have available to it the members in the class or referenced element (methods and variables). Reference datatypes may be arrays, interfaces, or classes.

Arrays

Collections are programmatic groups of objects or primitives. There are various types of collections available in Java, such as arrays, sets, dynamic arrays, linked lists, trees, hash tables, and key-value pairs (maps). Java provides a type of collection appropriately called Collection. This section discusses arrays. You will find information about the other categories of collections in Java language references.
Arrays in Java are collections of objects or primitives of similar type and may have one or more dimensions. Arrays are the only type of collection that can store primitive types. Elements within an array are accessed by indexes, which start at zero ([0]). To create an array, you declare it, allocate memory (size), and initialize the elements. These operations can be performed in two basic steps as shown here:

```
String animals[];
```

```
animals = new String[10];
```
The first line of code creates the array variable by adding a pair of square brackets to the variable name. The second line sets the size of the array (in this case, 10), which allocates memory, creates the object (animals), and initializes the elements. Arrays must be declared with a fixed number of members. This code could be condensed into the following line:
```
String animals[] = new String[10];
```
The next step is to store values in the array. In this example, the index numbers run from 0 to 9, and you store a value using that number as follows:
```
animals[3] = "Cat";
```
In Java, you can create arrays of arrays, more commonly known as *multi-dimensional arrays*. Since each array can be independently created, you can even create irregular combinations where array sizes vary within a given dimension. The more complex the array, the harder it is to keep track of, so moderation is advised. The following is a shorthand method for creating and assigning a two-dimensional array that stores pet owner names and the pet types:
```
class PetNames {
  public static main (String args[]) {
    String petFriends[ ][ ] = {
      {"George", "Snake", "Alligator" },
      {"Denise", "Butterfly"},
      {"Christine", "Tiger"},
      {"Robert", "Parrot", "Dove", "Dog", "Cat"}
    };
  }
}
```

Interfaces

An interface is somewhat like a PL/SQL package specification because it lists method signatures and constants without any method code body. Classes that *implement* (or inherit) from the interface must include all methods in the interface. Interfaces are useful for providing a common type for a number of classes. For example, if you have a method that needs to return a type that will be manipulated by three different classes (that execute slightly differently), you can use an interface as the return type. Each of the three classes would implement the interface and, therefore, the classes could be used in the same way by the method.

You can base a class on one or more interfaces, and this also provides a form of multi-parent inheritance. For example, if you had interfaces called SalaryHistory and CommissionHistory, you could define a class as follows:
```
public class HistoryAmounts extends CalcSalary implements SalaryHistory,
  CommissionHistory {
}
```
The `HistoryAmounts` class is a subclass of the `CalcSalary` class and will implement (provide method code declared in) the SalaryHistory and CommissionHistory interfaces. If you did not want to provide the code for the methods, you could declare `HistoryAmounts` as abstract (for example, `abstract class HistoryAmounts`). An *abstract class* cannot be instantiated but can be subclassed.

Classes

You can use any class to "type" an object (with the exception of abstract classes and classes with private constructors). The object becomes an instantiation of the class and has available to it the methods and member variables defined by the class. Therefore, classes can be used to create objects with the data and behavior characteristics defined in the class.

The Java language includes *wrapper* classes, such as `Boolean`, `Byte`, `Character`, `Double`, `Float`, `Integer`, `Long`, and `Number`, that implement the corresponding primitive datatypes and are commonly used as types for variables. These classes include methods that act upon the objects, such as a method that converts a `Long` to an `int`. For example, using a Long object called longVar, the `int` value is `longVar.intValue()`. The next section describes wrapper classes further.

Wrapper Classes

A *wrapper class* is a Java class that is created to control access to another Java class. A wrapper class can simplify the wrapped class' API, provide additional validation to the wrapped class, or change the access level of the wrapped class' methods. A wrapper class contains the wrapped class as an object member. For example, consider the following class:

```
package sample;
public class WrappedClass {
  public WrappedClass() {
  }
  int sumNumbers(int number1, int number2, boolean reallySum) {
    int result = 0;
    if (reallySum){
      result = number1 + number2;
    }
    return result;
  }
}
```

Note that the sumNumbers() method is not public so many classes will not be able to access it. In addition, sumNumbers() accepts three arguments. This class could be wrapped by the following wrapper class:

```
package sample;
public class WrapperClass {
  private final WrappedClass contents;
  public WrapperClass(WrappedClass newContents) {
    contents = newContents;
  }
  public int sumNumbers(int number1, int number2) {
    return contents.sumNumbers(number1, number2, true);
  }
}
```

Individual instances of WrappedClass are passed to the constructor and are stored in the `contents` field. The class exposes the API of WrappedClass, changing the number of arguments to `sumNumbers` and exposing it publicly. WrapperClass is said to wrap WrappedClass. In addition, instances of WrapperClass can be said to wrap the instances of WrappedClass that they contain.

Character String Classes

Two commonly used classes are `String` and `StringBuffer`.

String Class

A String object can be declared and assigned a set of characters as follows:

```
String stringVar = "This is a Java test string";
```

Objects built from `String` can take advantage of the methods in the `String` class. The methods provide functions to create strings from literals, chars, char arrays, and other string reference objects. The following Java Strings store the value “Java” by assigning a value to one String object and concatenating that object to another string using the `concat()` method that is part of the `String` class.

```
String startingLetters = "Ja";
String newString = startingLetters.concat("va");
```

Tip

To view the Javadoc for a basic Java class such as `String`, type “String” into the Code Editor (or find the class name “String” in the

file), place the cursor in the word, and select Quick Javadoc from the right-click menu. A window will pop up and display the documentation heading for that class. If you need to look at the entire Javadoc topic for the class including methods and other details, select Go to Javadoc. A window will appear with the full Javadoc topic containing methods and constants available to objects built from the class.

You can compare, concatenate, change the case of, find the length of, extract characters from, search, and modify strings. Strings in Java are considered *immutable*, that is, they cannot be changed. Whenever you alter a string through a string operation, the result is a new String object that contains the modifications. The old String object is no longer accessible because the object name points to the new object just created. You can take advantage of the overloading of the concatenation operator "+" to assign string values from number literals as in the following example:

```
// This assigns "The age is 235" to age.
String age = ("The age is " + 2 + 35);
// This assigns "The age is 37" to age.
String age = "The age is " + (2 + 35);
```

Note

In Java, the method `substring(int startIndex, int endIndex)` returns a portion of a string from the startIndex to the (endIndex –1). As with arrays, the index numbers start with zero. The following example will assign "This is a Java" to the newString variable:

```
String baseString = "This is a Java string";
String newString = baseString.substring(0, 15);
```

StringBuffer Class

`StringBuffer` is a sister class to `String` and represents character sequences that are *mutable*, that is, they can change size and/or be modified. What this means to the developer is that methods such as `append()` and `insert()` are available to modify a `StringBuffer` variable without creating a new object. Thus, the `StringBuffer` class is best if the character sequences being stored may need to be changed. The `String` class is good if the character sequence will not need to be changed.

The following shows an example usage of the `append()` method available to `StringBuffer`:

```
class StringAppend {
  public static void main (String args[]) {
    StringBuffer stringBuff = new StringBuffer("A string");
    stringBuff = stringBuff.append(" is added");
    System.out.println(stringBuff.toString());
  }
}
```

You could also append to a String variable using the String concat() method but, due to the immutable nature of String objects, that method would create a new String variable with the same name as the old variable. There is overhead and a bit of memory required by additional objects, so `StringBuffer` is better for concatenation.

Datatype Matching

Java is a semi-strongly typed language—every variable has a type, and every type is strictly defined. Type matching is strictly enforced in cases such as the following:

- The arguments passed to a method must match the argument types in the method's signature.
- Both sides of an assignment expression must contain the same datatype.
- Both sides of a Boolean comparison, such as an equality condition, must use matching datatypes.

There are few automatic conversions of one variable type to another. In practice, Java is not as restrictive as you might think, since most built-in methods are heavily overloaded (defined for different types of arguments). For example, you can combine strings, numbers, and dates using a concatenation operator (+) without formal variable type conversion, because the concatenation operator (which is, technically speaking, a base-language method) is overloaded.
In addition to overloading, an exact match is not always required, as shown in the following example:

```
public class TestCast {
  public static void main (String args[]) {
    byte smallNumber = 10;
    int largeNumber;
    largeNumber = smallNumber * 5;
    System.out.println("largeNumber is " + largeNumber);
    // smallNumber = largeNumber;
    smallNumber = (byte) largeNumber;
    System.out.println("smallNumber is " + smallNumber);
  }
}
```

The assignment starting with `largeNumber` assigns the `byte` variable `smallNumber` (times five) to the `int` variable `largeNumber`. In this case, there is a datatype mismatch (`byte` times `int`), but the code will compile without a problem because you are storing a smaller type (`byte`) in a larger type (`int`).
Rounding errors can occur from misuse of datatypes. The following shows an example of one of these errors:

```
int numA = 2;
int numB = 3;
System.out.println(numB/numA);
```

Although the division of these two variables results in "1.5," the print statement shows "1" because the output of the operator is the same type as the variables: `int`.

Casting Variables

In the preceding example class (`TestCast`), the statement that is commented out will generate a compilation error because it tries to store a larger-capacity datatype (`int`) in a smaller-capacity datatype (`byte`), even though the actual value of 50 is within the range of the `byte` datatype.
You can *cast* (explicitly convert) one type to another by preceding the variable name with the datatype in parentheses. The statement after the commented lines in this example corrects the typing error by casting `largeNumber` as a `byte` so that it can be stored in the `smallNumber` `byte` variable. The disadvantage of casting is that the compiler will not catch any type mismatch as it will for explicit, non-cast types. Another disadvantage with older JDKs is performance—the cast takes time; the more recent JDKs minimize or eliminate this overhead.

Casting Objects

You can also cast objects to classes and interfaces so that you can take advantage of the methods defined for the classes and interfaces. Casting allows you to match objects of different, but related, types. For example, the `Integer` class is a subclass of the `Number` class. The following code creates an object called `numWidth` as a Number cast from an Integer object. The cast is required because the `Number` class is abstract and you cannot instantiate it. The code then creates an object called `width` and assigns it the value of `numWidth`. Since `numWidth` is a Number object, which is less restrictive (or wider), this code needs to cast it to `Integer` match the new object.

```
Number numWidth = (Number) new Integer(10);
Integer width = (Integer) numWidth;
```

If the example were reversed so that the Integer was created first and the Number second, casting would not be required. Consider the following example:

```
Integer width2 = new Integer(10);
Number numWidth2 = width2;
```

Explicit casting of the Integer (`width2`) into the Number (`numWidth2`) is not required because Number is less restrictive (or wider). Casting to interfaces works in the same way.

Casting Literals

Floating-point literals (such as the value 34.5) default to the `double` datatype. If you want to assign a datatype of `float` to the literal, you must add an "F" suffix (for example, 34.5F). Alternatively, you may cast the literal using an expression such as `(float) 34.5`. Some examples for assigning datatypes to literals follow. ("L" is used for a `long` datatype, and "F" is used for a `float` datatype. It does not matter whether the suffix letters are upper- or lowercase.)

```
long population = 1234567890123456789L;
int age = 38;
float price = 460.95F;
float price = (float) 460.95;
double area, length = 3.15, width = 4.2;
area = length * width;
```

Non-floating literals (such as 38 in the example) will be assigned an `int` datatype. This can make an expression such as the following fail at compile time:

```
smallNumber = 5 + smallNumber;
```

The right side of the expression (`5 + smallNumber`) is assigned an `int` type because "5" is an `int` and `smallNumber` is implicitly cast up to match it. The right side does not match the left side because `smallNumber` is a `byte`. Explicit casting will solve the problem if you apply the cast to the entire side of the expression as follows:

```
smallNumber = (byte) (5 + smallNumber);
```

Conclusion

To combat confusion and fear about Java, PL/SQL enthusiasts only need a bit of knowledge about object orientation and basic Java concepts and elements. This paper has provided an introduction to those topics. Further study and some real-world experience will make you as expert in Java as you are in SQL and PL/SQL.

About the Author

Peter Koletzke is a technical director and principal instructor for the Enterprise e-Commerce Solutions practice at Quovera, in Mountain View, California, and has 25 years of industry experience. Peter has presented at various Oracle users group conferences more than 240 times and has won awards such as Pinnacle Publishing's Technical Achievement, Oracle Development Tools Users Group (ODTUG) Editor's Choice, ECO/SEOUC Oracle Designer Award, ODTUG Volunteer of the Year, and NYOUG Editor's Choice. He is an Oracle Certified Master, Oracle ACE Director, and coauthor of the Oracle Press Books: *Oracle JDeveloper 10*g *for Forms & PL/SQL Developers* (with Duncan Mills); *Oracle JDeveloper 10*g *Handbook* (from which the material in this white paper is taken) and *Oracle9*i *JDeveloper Handbook* (with Dr. Paul Dorsey and Avrom Roy-Faderman); *Oracle JDeveloper 3 Handbook, Oracle Developer Advanced Forms and Reports*, *Oracle Designer Handbook, 2nd Edition*, and *Oracle Designer/2000 Handbook* (all with Dr. Paul Dorsey).

Faster Cross-Platform Database Migration with RMAN

John Larkin – JP Morgan Chase

Introduction

You can always move a database from one OS to another with DATAPUMP or EXPORT and IMPORT, but these methods are time consuming. You can reduce the time needed to actually convert the datafiles by up to 88% or more when you use RMAN to parallelize the datafile conversions. There are some challenges with some versions of ORACLE but there are ways around these issues. And, based on Oracle documentation you might even get away with converting only a fraction of all the datafiles in the database. Our initial tests showed a serial runtime of 150 min and a parallelized runtime of 18 min. To test with a worst case scenario, both the input and output files were on the same mount point.
The genesis of this paper is based on a year-long technology refresh project to migrate and upgrade 88 Oracle 10g databases from 6 standalone Solaris / AIX servers to a Veritas-clustered environment with 10 Solaris servers (1 Production and 1 QA 4-node cluster and 1 Development 2-node cluster) and shared SAN storage. In this paper we will present our findings on RMAN behavior and performance as it is manifested when using the "CONVERT DATABASE" command with Source or Target conversion. The main goal during any given database migration was to balance the downtime required against the assurance of a successful migration. We hope to illustrate how our iterative approach to major production changes allowed us to modify and improve our plan as we moved through the project.

The PROJECT

The project involved migrating 88 databases on disparate operating systems (approximately a 50/50 split) on Solaris 8 and AIX 5.3 to a common Solaris 10 environment. The databases were mostly running Oracle Enterprise 10.2.0.2 with a few at 10.1.0.4 and one at 9.2.0.6. Nine of these databases included a manual dataguard solution (no broker) for disaster recovery.
The planning for this project included mapping existing databases to the new cluster by attempting to balance the relative load of each database with the desire to keep related databases on the same node within a cluster. Additionally we mapped database names to virtual IP addresses and registered them in our DNS server so that each database TNS entry referenced a virtual IP address that could be moved to any server in the cluster seamlessly. The majority of the storage resources were shared across all nodes within a cluster using the Veritas cluster filesystem. A small amount of storage was reserved for a node-specific filesystem used to house the OEM software and its logs. A second reason for this segregated storage was as a safety net in case we found an issue with the shared database software installation – mainly in the area of OS upgrades and RDBMS relinking requirements.
The ideas presented here are the result of many months of work by all of the DBA's in our group each working on sections of the project to contribute to the successful conclusion of this mass migration and upgrade.

The PLAN for Database Migrations

The actual migration phase of the project required 2 different approaches depending on the methods available for migration from each OS. We ended up with 2 main migration alternatives from each OS with some minor, but not necessarily inconsequential variations on those main choices.

Solaris to Solaris Migrations

To begin we'll look at the simpler options which as you might guess, are related to a "Same OS conversion", moving from one server to another with the same OS. As we mentioned earlier EXPORT / IMPORT is always an option, with a good record of favorable outcomes, but the drawback is that it takes quite a while on an intermediate-sized server to import an entire database. This is especially true if you were to have 2 or 3 database migrations occurring at the same time. Since we have many databases that are inter-related, staggering the migration of related databases would really just be another form of extended downtime since database links would be inoperable during two outage windows instead of just one. Additionally, since the target servers are by definition a shared resource, competition for common resources helped us rank IMPORT as a last resort. This left us with 2 other alternatives that held more promise for more efficient migrations. One was a simple FTP of the database files from one server to the other after a clean shutdown. The simplicity of this option made it very appealing, however the time needed to FTP files (or even use NFS mounts) to the target server at the time of migration made this a sub-optimal choice. This method would work fine for small non-Production databases, but as the size of the databases grew and with it the impact of the downtime, we would want other options. This method was mainly used for development databases that were less than 100 GB.

Our second alternative was to build a physical standby database on the target server (sometimes a secondary standby since many production databases already had a standby database for normal disaster recovery purposes) for the primary database. This would allow us to pre-stage the datafiles on the target server in a recoverable state eliminating many hours of downtime during a migration. As long as the archived logs were being shipped and applied to the standby database on the target server, the datafile migration could be completed with little more than activating the standby database. A twist on the standby migration was an attempt to perform a switchover the standby database, placing the original database in a standby role so that should we need to fall back to the original server the database would be current. It was a great idea but an issue with the control files required us to re-create them so that they would appear as primary controlfiles instead of standby controlfiles. This happened at the end of the Solaris-Solaris migrations, so there were no recurring problems related to this variation.

AIX to Solaris Migrations

The more complex migrations tended to be those that involved "Different OS conversion", moving from an AIX server to the target Solaris server. Again EXPORT / IMPORT is always an option, especially since it provided us with the ability to handle the cross-platform migration issues. The main drawback continued to be the additional time required for migration and our largest databases were on the AIX platform. We also ruled out the FTP of an idle database to the target server – this was not an option for us since datafile conversion would be needed. The same issue applied to using a physical standby database.

We were running out of options for speedy migrations when Devesh came up with the idea to use RMAN's Cross-Platform Transportable Database (TDB) feature. He found that it would do the datafile conversions needed and it had an option to parallelize the conversion. This was just what we were looking for - ease of conversion and speed. This option was not quite as simple as the documentation might lead you to believe, but with persistence (and Metalink) we were able to succeed in greatly reducing the datafile conversion times.

RMAN Conversion Basics

Up until about a year ago Recovery Manager (RMAN) was not one of my favorite Oracle utilities. It seemed to be cumbersome and finicky but I have since learned to appreciate the power of the tool. We'll review the basic features that would be needed to perform the cross-platform database migration.

As a backup and recovery tool RMAN needs to keep track of the information needed to backup and recover a database. This information is always stored in the target database's control files (RMAN repository) and if you also have a Recovery Catalog, this basic information as well as additional data can be stored there. For the purposes of this paper we only need access to the RMAN repository. The target database is the one requiring a backup, recovery or conversion process to be performed on it. RMAN also permits you to store basic configuration parameters so that you can retain settings persistently.

The basic startup / connection for RMAN that we need looks like : **`rman target /`**

This connects you the db identified by the value in your ORACLE_SID environment variable. You could optionally specify the **catalog=** parameter if you needed to use the recovery catalog, but we do not need that for our purposes.
To utilize parallelism we will need to modify one of the persistent settings with the configure command :

```
CONFIGURE DEVICE TYPE DISK PARALLELISM 6 BACKUP TYPE TO BACKUPSET;
```

This will allow RMAN to start 6 processes, each one will be able to process a separate datafile simultaneously. This can also be accomplished with the **PARALLELISM** parameter of the **CONVERT DATABASE** command

There are 2 options to the **CONVERT DATABASE** command which allow you to specify whether the actual datafile conversion is to occur on the source or target platform. Source platform conversion is the default and has no parameters. The target platform conversion is indicated by adding the **ON TARGET PLATFORM** clause to the convert command. The commands used for conversion follow:

SOURCE CONVERSION

```
run
{
CONVERT DATABASE
new database      'oradb1p'
TRANSPORT script   'oradb1p_transportscript.sql'
to platform   'Solaris[tm] OE (64-bit)'
db_file_name_convert    '/app/oraf'  ,     '/dbmig/oradb1p_migr' ;
}
```

This command will create on the SOURCE server :

1. a TRANSPORT script
2. a set of CONVERTED data files.
3. The following messages when the RMAN command completes.

Run SQL script oradb1P_transportscript.sql on the target platform to create database.
Edit init.ora file $ORACLE_HOME/dbs/init_00jq5163_1_0.ora. This PFILE will be used to create the database on the target platform.
To recompile all PL/SQL modules, run utlirp.sql and utlrp.sql on the target platform.
To change the internal database identifier, use DBNEWID Utility.

TARGET CONVERSION

```
run
{
CONVERT DATABASE
new database                  'oradb1p'
➔ON TARGET PLATFORM [1]
➔CONVERT   script     'oradb1p_convertscript.sql' [2]
TRANSPORT script  'oradb1p_transportscript.sql'
to platform  'Solaris[tm] OE (64-bit)'
db_file_name_convert
'/app/oraf/u000', '/app/oraf/u031/tempxfer/u000',
'/app/oraf', '/app/orax';
}
```

-- Notes

1. The order in db_file_name_convert is important – the most generic mapping must be last to work effectively.
2. Without the db_file_name_convert parameter mapping you get $ORACLE_HOME/dbs/omf file names.
3. There must be some difference between the FROM and TO mappings or you will get OMF names.

This command will create on the SOURCE server :
1. a TRANSPORT script
2. a CONVERT script
   ** NO converted datafiles will be created.
3. The following messages when the RMAN command completes.

Run SQL script oradb1p_transportscript.sql on the target platform to create database.
Edit init.ora file /app/oraprod/Oracle10gEE-10.2.0.2.0-64b/dbs/init_00jvr8a7_1_0.ora.
This PFILE will be used to create the database on the target platform
→Run RMAN script oradb1p_convertscript.sql on target platfm to convert datafiles [3]
To recompile all PL/SQL modules, run utlirp.sql and utlrp.sql on the target platform
To change the internal database identifier, use DBNEWID Utility.

[1.] specifies that a TARGET conversion is to be performed.
[2.] the RMAN script that you will run on the target server to complete the conversion.
[3.] the extra step that is needed when you perform a TARGET conversion.

Initial Setup

We needed to create a backup controlfile from the source database to be used for the RMAN conversion. We also required a single temporary filesystem to store the converted files before FTP'ing them to the destination server. Then we created a directory for each database being converted and below them were directories that represented the various mount points in order to maintain the mount point groupings from the source databases. This was not required but it did make it easier to group the files when they needed to be moved to the target server.

The Cross-Platform Migration Process

The source conversion method was used with several of our initial development database conversions. Most steps are the same for any of the RMAN migration methods that you may choose. The basic steps for conversion follow:

1. Get a location for the READ ONLY data files.
2. Build directories for the files (either 1 big dir or 1 for 1 dir names).
3. Set the AUDIT trail parameter to none
4. Disable Standby database log shipping
   a. log_archive_dest_state_2 defer
5. Get file sizes by mountpoint

```
col bytes    format 999,999,999,999
col maxbytes format 999,999,999,999

select substr(file_name, 1, 18) FILE_NAME,
count(*) Files,
sum(maxbytes) maxbytes
from dba_data_files
group by substr(file_name, 1, 18)

FILE_NAME          FILES        MAXBYTES
------------------------------------------------------------
/app/oraf/u006         8     12,280,922,112
```

6. Restart the database in READONLY mode
7. Do the prerequisite checks to verify the db is transportable.

a. What platforms are valid for TDB migration ?
select platform_name from v$db_transportable_platform;
PLATFORM_NAME

Solaris[tm] OE (32-bit)
Solaris[tm] OE (64-bit)
AIX-Based Systems (64-bit)

b. Do any files need conversion (to cheat later if we must)
select distinct(file_name)
from dba_data_files a,
dba_rollback_segs b
where a.tablespace_name = b.tablespace_name;

c. Can the database be transported to the target OS?
set serveroutput on
declare
retcode boolean;
begin
retcode := dbms_tdb.check_db('Solaris[tm] OE (64-bit) ',dbms_tdb.skip_none);
--(want TRUE)
IF retcode
THEN
-- seem to get TRUE even if CAPS is not matched
dbms_output.put_line('rc= TRUE');
ELSE
dbms_output.put_line('rc= FALSE');
end IF;
end;

OUTPUT :
"The following directories exist in the database:
SYS.APPL_FILE_IN, SYS.DATA_PUMP_DIR"

8. Identify external tables and bfiles – these are not migrated automatically.
(The Oracle-supplied code tdb_* can be found in Metalink).
a. identify external table files
select directory_path||'/'||location External_file_path
from dba_directories a, dba_external_locations b
where a.directory_name=b.directory_name;

9. Identify BFILE files that will need to be transferred to the target :
@tdb_get_bfile_dirs.sql

OUTPUT:
"The following directories contain external files for BFILE columns
Copy the files within these directories to the same path on the target system
There are 0 directories, 0 total BFILEs"

-- if necessary, list bfiles.
@tdb_get_bfiles.sql

10. Run either the NON-Parallelized or Parallelizable CONVERT DATABASE command on the SOURCE server.
11. Move to the TARGET server
12. Get the datafiles to the TARGET server.
13. Modify init.ora as needed. (copy your original and apply changes from the generated one.)
14. Create a temporary control file / database with the same name as the original. (with just the dictionary and undo files). This is used to run the RMAN convert script.
15. Run the RMAN convert script.
16. Run the SQL transport script generated from the CONVERT DATABASE command to create the real database.
17. Run utlirp.sql and utlrp.sql to recompile all PL/SQL modules.
18. Run the NEWID utility if you need to change the internal database id.
19. Your migrated database is ready to use.

The Issues with Parallelism

This process seems to be relatively straight forward, but what would a migration be without a little hitch. During our testing we encountered problems when we actually tried to get the parallelism feature to work.
There were various documented bugs that we encountered throughout the process:
Bug 5016125 - RMAN convert with parallelism > 1 fails (RMAN-3009 / ORA-19927) – This was a SOURCE conversion bug. The workaround was to not use the parallelism option or upgrade to 10.2.0.4 or 11g.
Note:417455.1 – Datafiles are not converted in parallel for transportable database during a source system conversion.
When performing a source system conversion, Bug 5016125 prevents the use of multiple channels during the datafiles conversion portion of CONVERT DATABASE. Workaround: Use a single channel OR Apply fix for BUG:5016125 (i.e. 10.2.0.4) OR Upgrade to 11g.
These issues ruled out the SOURCE system conversion. One of our main goals was to maintain the ability to fall back to the databases previous state as quickly as possible. Since our plan called for us to leave the original database intact on the source server so that backing out to the original database required nothing more than restarting the old database, we did not want to do an upgrade on the source system and jeopardize that contingency.
We moved on to testing the target system conversion which lead us to Bug 5641099. When performing a target system conversion, this bug causes the convert script to be created in a way that prevents the use of RMAN PARALLELISM during datafiles conversion. For RMAN PARALLELISM to work, multiple datafiles must be specified under a single CONVERT DATAFILE statement. This was a relatively simple fix – a few shell commands would ensure repeatability and automate most of the process of correcting the error and a few changes with vi would complete the changes needed in the CONVERT script. There were also some modifications required to the TRANSPORT script. The CONVERT DATABASE generated a transport script that used OMF type names for LOGFILE and TEMPFILE names similar to the following - '…/dbs/arch_D-ORADB1P_id-3931780729_S-460049_T-1_A-555085049_00jvr8a7'. Change these names to match your original names if desired.
The TRANSPORT script will have entries similar to the following: (this example is not a complete listing, only the entries requiring discussion are listed here)

```
-- The following commands will create a new control file and use it to open the database.
-- Data used by Recovery Manager will be lost.
-- The contents of online logs will be lost and all backups will
-- be invalidated. Use this only if online logs are damaged.
--
STARTUP NOMOUNT
CREATE CONTROLFILE REUSE SET DATABASE "ORADB1P" RESETLOGS  ARCHIVELOG
LOGFILE
  GROUP 1 (
    '/app/oraf/u003/redo_logs/ORADB1P/ORADB1P.g1.m1.rdo')
DATAFILE   '/app/oraf/u033/datafiles/ORADB1P/ORADB1P.system.01.dbf'
```

```
CHARACTER SET US7ASCII ;

-- Database can now be opened zeroing the online logs.
ALTER DATABASE OPEN RESETLOGS;

-- Add TEMPFILES
ALTER TABLESPACE ORACLE_TMP_SRT ADD TEMPFILE ‘xx/temp1.dbf’

prompt * Your database has been created successfully!

SHUTDOWN IMMEDIATE
STARTUP UPGRADE PFILE='../dbs/init_00jvr8a7_1_0.ora'
@@ ?/rdbms/admin/utlirp.sql
SHUTDOWN IMMEDIATE
STARTUP PFILE='../dbs/init_00jvr8a7_1_0.ora'
-- The following step will recompile all PL/SQL modules.
@@ ?/rdbms/admin/utlrp.sql
```

The CONVERT script requires the following modifications:
(Use FORMAT or CONVERT depending on whether or not you need to use an intermediate file name)
cat oradb1p_convertscript.sql | awk '{ if($1 == "FORMAT") print }' > oradb1p_convertscript_t2

vim oradb1p_convertscript_t2

:%s/CONVERT DATAFILE/,/
:%s/;//

add : “ RUN { CONVERT DATAFILE “ at the top of the script.

add :
FROM PLATFORM 'AIX-Based Systems (64-bit)'
db_file_name_convert '/app/oraf/u031/tempxfer/u000','/app/oraf/u000',
..
'/app/oraf/u032/tempxfer/u033','/app/oraf/u033',
'/app/orax', '/app/oraf';
} ;
at the bottom of the script
so we end up with a convert script (oradb1p_convertscript_t2) that looks like:

```
RUN {
CONVERT DATAFILE
 '/app/oraf/u032/tempxfer/u022/datafiles/ORADB1P/ORADB1P.appl1_opn_tbl.01.dbf'
  , '/app/oraf/u031/tempxfer/u006/datafiles/ORADB1P/ORADB1P.oracle_trn_undo.01.dbf'
  , '/app/oraf/u032/tempxfer/u033/datafiles/ORADB1P/ORADB1P.system.01.dbf'
  , '/app/oraf/u031/tempxfer/u006/datafiles/ORADB1P/ORADB1P.sysaux.01.dbf'
FROM PLATFORM 'AIX-Based Systems (64-bit)'
db_file_name_convert
'/app/oraf/u031/tempxfer/u006','/app/oraf/u006',
'/app/oraf/u032/tempxfer/u022','/app/oraf/u022',
'/app/oraf/u032/tempxfer/u033','/app/oraf/u033',
'/oracle/orafx',    '/oracle/orafs';
    }
```

These modifications will allow you to convert all datafiles 88% faster , using 6 degrees of parallelism on a Target system conversion with Oracle 10.2.0.3. This is a significant reduction in time, especially for database conversions that might take 24 hours or more. In this case a 24 hour conversion could be cut down to less than 12 hours overall with this method. (Remember the 88% reduction in runtime was for the datafile conversion piece – you would still need significant time to move the datafiles from the source to the target server.). Had we been able to use the Source system conversion method a greater reduction in overall runtime should have been achieved since we would have been able to take the source database files directly as input to the CONVERT DATAFILE statement and route the output to mount points that were NFS mounts of the destination server filesystem.
A method to further reduce the conversion time is described in Metalink Note 732053.1 – Avoid Datafile Conversion during Transportable Database. This note states that only datafiles that contain UNDO data require conversion. This includes any files that make up the SYSTEM and UNDO tablespaces and any other tablespaces that contain ROLLBACK segments. These tablespaces can be identified using this query : select distinct(file_name) from dba_data_files a, dba_rollback_segs b where a.tablespace_name=b.tablespace_name;

The Results

A quick review of the output from the RMAN CONVERT DATABASE and CONVERT DATAFILE statements can help us confirm the runtime improvement and provide some additional insight into the conversion process from RMAN's perspective.

```
Recovery Manager: Release 10.2.0.2.0 - Production on Sun Nov 16 10:21:53 2008
connected to target database: ORADB1P (DBID=9931799729)
RMAN>    run
2>      {
3>      CONVERT DATABASE NEW DATABASE 'oradb1p'
4>      ON TARGET PLATFORM
..
46>     }

Starting convert at 16-NOV-08
using target database control file instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=549 devtype=DISK

Directory SYS.APPL1_FILE_IN found in the database
Directory SYS.APPL1_FILE_OUT found in the database
Directory SYS.DATA_PUMP_DIR found in the database

User SYS with SYSDBA and SYSOPER privilege found in password file
channel ORA_DISK_1: starting to check datafiles
input datafile fno=00078 name=ORADB1P.appl1_opn_tbl.01.dbf
channel ORA_DISK_1: datafile checking complete, elapsed time: 00:00:00
channel ORA_DISK_1: starting to check datafiles
input datafile fno=00079 name=ORADB1P.appl1_opn_tbl.02.dbf
channel ORA_DISK_1: datafile checking complete, elapsed time: 00:00:00
..
channel ORA_DISK_1: starting to check datafiles
input datafile fno=00106 name=ORADB1P.appl2m_opn_tbl.37.dbf
channel ORA_DISK_1: datafile checking complete, elapsed time: 00:00:00
..
Finished backup at 16-NOV-08
```

Here we can see that on the source system where we executed the CONVERT DATABASE statement that only 1 channel is allocated. This also lists any directories defined in the database and the users in the password file. The interesting part

follows where we can see that all of the datafiles are checked (not converted) in less than one second each. This is reasonable since we had indicated to RMAN that we would be doing a TARGET system conversion.
We then FTP'ed the datafiles in parallel (4 separate FTP sessions) to the destination server we achieved a throughput of more that 2 GB per minute for each FTP session.

```
150 Opening data connection for ORADB1P.oracle_usr_tix.01.dbf (2097160192 bytes).
226 Transfer complete.
local: ORADB1P.oracle_usr_tix.01.dbf remote: ORADB1P.oracle_usr_tix.01.dbf
2097160192 bytes received in 31 seconds (65119.27 Kbytes/s)
150 Opening data connection for ORADB1P.appl2m_opn_idx.02.dbf (2097160192 bytes).
226 Transfer complete.
local: ORADB1P.appl2m_opn_idx.02.dbf remote: ORADB1P.appl2m_opn_idx.02.dbf
2097160192 bytes received in 40 seconds (51241.76 Kbytes/s)
```

The partial log from the CONVERT DATAFILE statement on the Target server follows :

```
ora1d3@destsvr::dba-work/uid01a/work/upgr_10203.528=>rman target / nocatalog
Recovery Manager: Release 10.2.0.2.0 - Production on Sun Nov 16 15:36:51 2008
connected to target database: ORADB1P (DBID=3939909829)
using target database control file instead of recovery catalog

RMAN> @oradb1p_convertscript_t2.sql

RMAN> RUN {
2> CONVERT DATAFILE
71>     '/app/oraf/u032/tempxfer/u033/datafiles/ORADB1P/ORADB1P.system.01.dbf'
72>   , '/app/oraf/u031/tempxfer/u006/datafiles/ORADB1P/ORADB1P.sysaux.01.dbf'
83>   , '/app/oraf/u031/tempxfer/u006/datafiles/ORADB1P/ORADB1P.oracle_adm_tix.01.dbf'
109> FROM PLATFORM 'AIX-Based Systems (64-bit)'
110> db_file_name_convert
117> '/app/oraf/u031/tempxfer/u006','/app/oraf/u006',
133> '/app/oraf/u032/tempxfer/u022','/app/oraf/u022',
144> '/app/oraf/u032/tempxfer/u033','/app/oraf/u033',
145> '/app/orax',   '/app/oraf';
146>    }
Starting backup at 16-NOV-08
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=547 devtype=DISK
allocated channel: ORA_DISK_2
channel ORA_DISK_2: sid=513 devtype=DISK
allocated channel: ORA_DISK_3
channel ORA_DISK_3: sid=541 devtype=DISK
allocated channel: ORA_DISK_4
channel ORA_DISK_4: sid=536 devtype=DISK
allocated channel: ORA_DISK_5
channel ORA_DISK_5: sid=535 devtype=DISK
allocated channel: ORA_DISK_6
channel ORA_DISK_6: sid=557 devtype=DISK
channel ORA_DISK_1: starting datafile conversion
channel ORA_DISK_2: starting datafile conversion
..
channel ORA_DISK_6: starting datafile conversion
input filename=ORADB1P.appl1_opn_tbl.01.dbf
input filename=ORADB1P.appl1_opn_tbl.02.dbf
input filename=ORADB1P.appl1_opn_tbl.03.dbf
input filename=ORADB1P.appl1_opn_tbl.04.dbf
input filename=ORADB1P.appl1_opn_tbl.05.dbf
input
```

```
filename=/app/oraf/u031/tempxfer/u006/datafiles/ORADB1P/ORADB1P.oracle_usr_tix.01.dbf
converted datafile=/app/oraf/u006/datafiles/ORADB1P/ORADB1P.oracle_usr_tix.01.dbf
channel ORA_DISK_6: datafile conversion complete, elapsed time: 00:00:45
channel ORA_DISK_6: starting datafile conversion
input filename=ORADB1P.appl2m_opn_tbl.06.dbf
converted datafile=ORADB1P.appl1_opn_tbl.02.dbf
channel ORA_DISK_2: datafile conversion complete, elapsed time: 00:00:49
channel ORA_DISK_2: starting datafile conversion
input filename=ORADB1P.appl2m_opn_tbl.07.dbf
converted datafile=ORADB1P.appl1_opn_tbl.04.dbf
channel ORA_DISK_4: datafile conversion complete, elapsed time: 00:00:51
channel ORA_DISK_4: starting datafile conversion
converted datafile=ORADB1P.appl1_opn_tbl.01.dbf
channel ORA_DISK_1: datafile conversion complete, elapsed time: 00:00:52
channel ORA_DISK_1: starting datafile conversion
converted datafile=ORADB1P.appl1_opn_tbl.03.dbf
channel ORA_DISK_3: datafile conversion complete, elapsed time: 00:00:52
channel ORA_DISK_3: starting datafile conversion
input filename=ORADB1P.appl2m_opn_tbl.08.dbf
input filename=ORADB1P.appl2m_opn_tbl.09.dbf
input filename=ORADB1P.appl2m_opn_tbl.10.dbf
converted datafile=ORADB1P.appl1_opn_tbl.05.dbf
channel ORA_DISK_5: datafile conversion complete, elapsed time: 00:00:53
Finished backup at 16-NOV-08

RMAN>
RMAN> **end-of-file**
```

From this excerpt of the log file we can see that our average throughput is better than 2 GB per minute per channel. We can also see that the value displayed in the “input filename” message is the first parameter we passed to the db_file_name_convert parameter and that the value displayed in the “converted datafile” message is the second parameter. We can also see that the various channels are each processing files in parallel.

Sample Error Messages

Missing a destination directory on a mount point.- The message is fairly explicit and it didn’t take long to find and fix the error:

```
input filename=ORADB1IP.appl2m_opn_tbl.15.dbf
RMAN-00569: ==== ERROR MESSAGE STACK FOLLOWS ==
RMAN-03009: failure of backup command on ORA_DISK_1 channel at 10/15/2008 15:21:15
ORA-19504: failed to create file "ORADB1I.appl2m_opn_tbl.15.dbf"
ORA-27040: file create error, unable to create file
SVR4 Error: 2: No such file or directory
```

```
Incomplete cleanup – The message above required a rerun of the CONVERT DATAFILE
statement. We tested to see how RMAN would react to existing destination files. The error
message below gave us the answer.
```

```
RMAN-00569: ==== ERROR MESSAGE STACK FOLLOWS ==
RMAN-03009: failure of backup command on ORA_DISK_1 channel at 10/15/2008 15:32:30
ORA-19504: failed to create file
"/app/oraf/u022/datafiles/ORADB1I/ORADB1I.appl1_opn_tbl.01.dbf"
ORA-27038: created file already exists
```

Conclusion

Even before the current economic challenges that we face there has been a drive to reduce operational costs by consolidating servers. One of the offshoots of this drive is that moving a database from one platform to another has become a more common task required of DBA's. While the move that we faced was not the worst case (ours did not involve a change in endianness) it gave us a chance to learn more about the inner workings of RMAN and datafile structure. The ability of parallelized processing to improve RMAN throughput to the extent that it did was a pleasant surprise and has helped to reinforce the desire to push beyond what I thought was reasonable. Last September Tom Kyte spoke at the NYOUG Oracle Day on the topic of "How do we know what we know" and how the old "truths" that we hung on to from Oracle 7 or 8i are not necessarily so in the Grid world. This project has helped to remind me that sometimes it's best to forget what I think I know and re-test what reality is in the 10g / 11g world.

Acknowledgements

I would like to extend special thanks to the entire Marketing DBA group (Devesh, Ravi, Dave, Suresh, Linda, Pradeep, Hanmanth, Sadiq) who have provided invaluable input into the design and testing of the various methods considered during the course of this project and recognition of the Marketing System Administrators for their willingness to humor us and our requests for system modifications to help us improve the migration process.

References

Oracle - Backup and Recovery Advanced User's Guide
Chapter 15 – RMAN Cross-Platform Transportable Databases and Tablespaces
Metalink – Bug reports, Technical Notes, How-To articles

APEX Debug Options

Karen Cannell, Integra Technology Consulting

The APEX Debug Conundrum

co·nun·drum [kə núndrəm] (*plural* co·nun·drums)
noun Definition: 1. **something confusing:** something that is puzzling or confusing

Debugging Oracle Application Express (APEX) can be a conundrum. It is a PL/SQL application, but there is no traditional way to "launch" it, and no traditional way to step through the code. It is an HTML page, but "View Source" does not give any information on what code ran, when and with what values. So how does a developer know what APEX is doing when? How does one determine if APEX is executing a code module, or why it is not? This paper aims to ease the conundrum by presenting a suite of APEX-supplied and external debug options readily available to APEX developers. APEX is a blend of technologies – PL/SQL and SQL that generates HTML pages using CSS, templates and JavaScript and perhaps web services. Tracking and resolving problems in this multi-"layer" environment necessitates a blend of tactics to examine what is happening in each components of the application.
The debug options described here include APEX-supplied utilities such as Debug mode, Session and Event information, remote PL/SQL debugging using Oracle SQL Developer, and the use of web development tools such as Web Developer and Firebug. The intent is to provide enough information to get a developer started in using these options. The reader is encouraged to consult the references included at the conclusion of this paper for more information on each topic presented. Application of a mixture of these debug tactics will help get one through most APEX debug challenges.

Change of Thought

Troubleshooting in APEX requires a different line of thinking, and an expanded set of tools than many of us – traditional PL/SQL or Oracle Forms developers - are used to. The shift from a traditional PL/SQL or Oracle Forms environment to APEX is a change from an enclosed application to the web. Web applications entail coding on multiple layers and therefore one must think along those same layers when troubleshooting. Gone are the days of step through the code in a debugger, or adding text message output to show our progress. Developers now need to examine the HTML, verify the style sheets, check for any JavaScript, review process and web service behavior, and, if warranted, step through code in a debugger.

The change of thought is mainly in the initial triage, when determining which component of a web application is the likely source of an issue. Depending on the application, a problem could be in generated or hand-coded HTML, a style sheet, JavaScript, in the SQL of a report, validation, computation, or condition, in a PL/SQL process or in a web service.

Knowing, or guessing, which component is causing the problem determines which debug tactic to start with. Making an educated guess is partly experience, and partly knowing how APEX works. APEX developers must have an understanding of the overall APEX web page generation process in addition to understanding of the business application logic. Several of the APEX utilities described below are designed to report on the flow of events that occur in constructing an APEX page. While not "debug tools" in the traditional sense, these activity and event reports are essential in targeting APEX issues because they show the steps APEX executes in generating a page. The next sections review debug options specific to Oracle APEX, remote PL/SQL debugging and the of use of generic web development tools for APEX troubleshooting.

Figure 1 – APEX Developer Toolbar

APEX –Supplied Debug Options

APEX has several built-in tools to assist a developer in troubleshooting application behavior without leaving the APEX environment. The APEX Developer Toolbar, the set of links at the bottom of every un-deployed APEX application page,

includes the Session, Debug and Activity options. These allow the developer to view session state, debug information, and application activity. Every developer should become thoroughly familiar with these options. Additional utilities for Application Reports, object dependencies and trace enable-ing further assist a developer in finding and resolving interface, logic and performance errors.

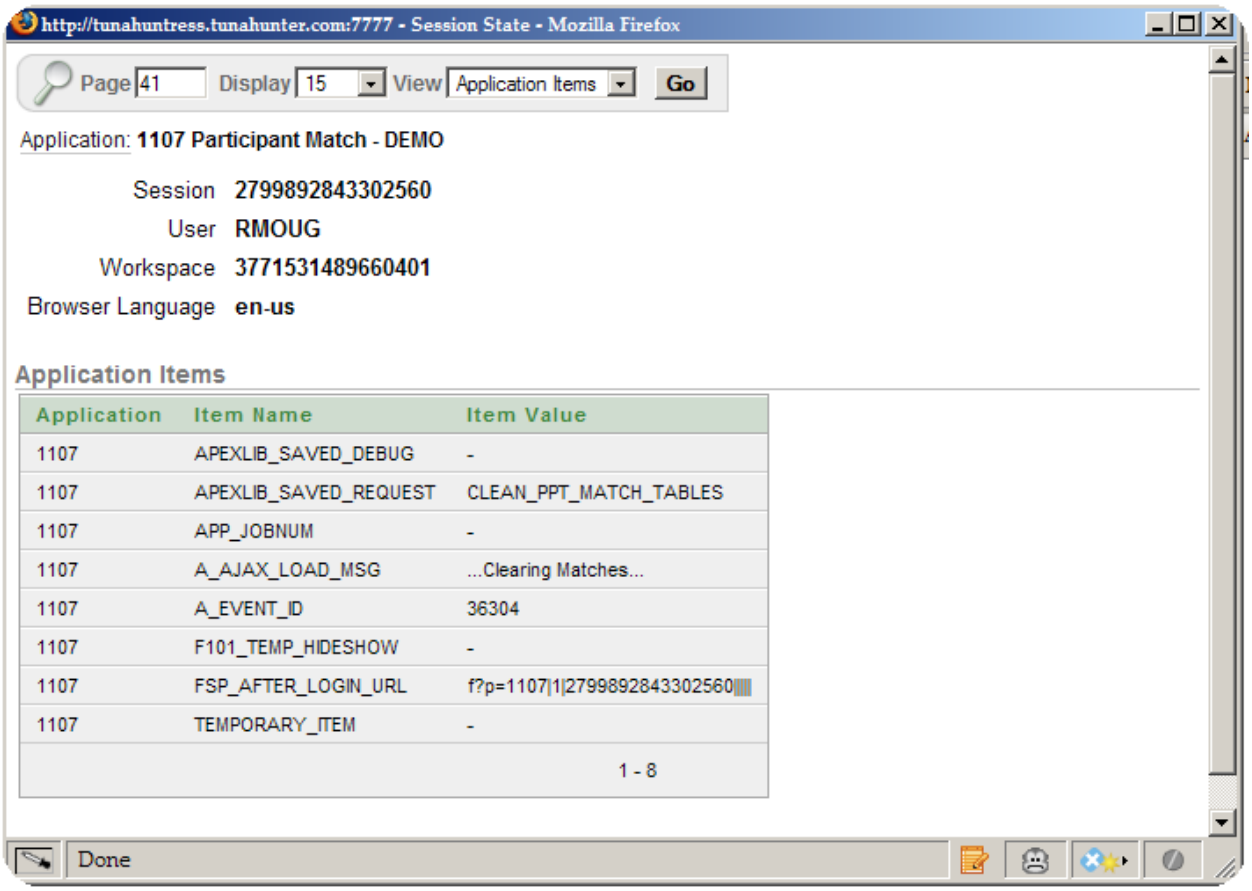


Figure 2 – APEX Session – Application Items

View Session State

The **Session** link in the Developer toolbar allows one to view session state values. The **Session** link opens a popup window, which displays session state by Application, Page Items, Application Items, and Session State. This interface displays the current values of the page and application items for the application. This is where to look to see if application or page items have been set correctly. Figure 2 illustrates the Session interface for Application Items.

The Page Items view of the Session interface lists the application, page, item name, display option, item value and status. The Page Item status is I, U or R for Inserted, Updated or Reset. The status settings help identify recently set or updated items.

ORA-01722: invalid number

Error in show_hide_collection_output(): ORA-01722: invalid number

Return to application.

Figure 3 – Application with Error

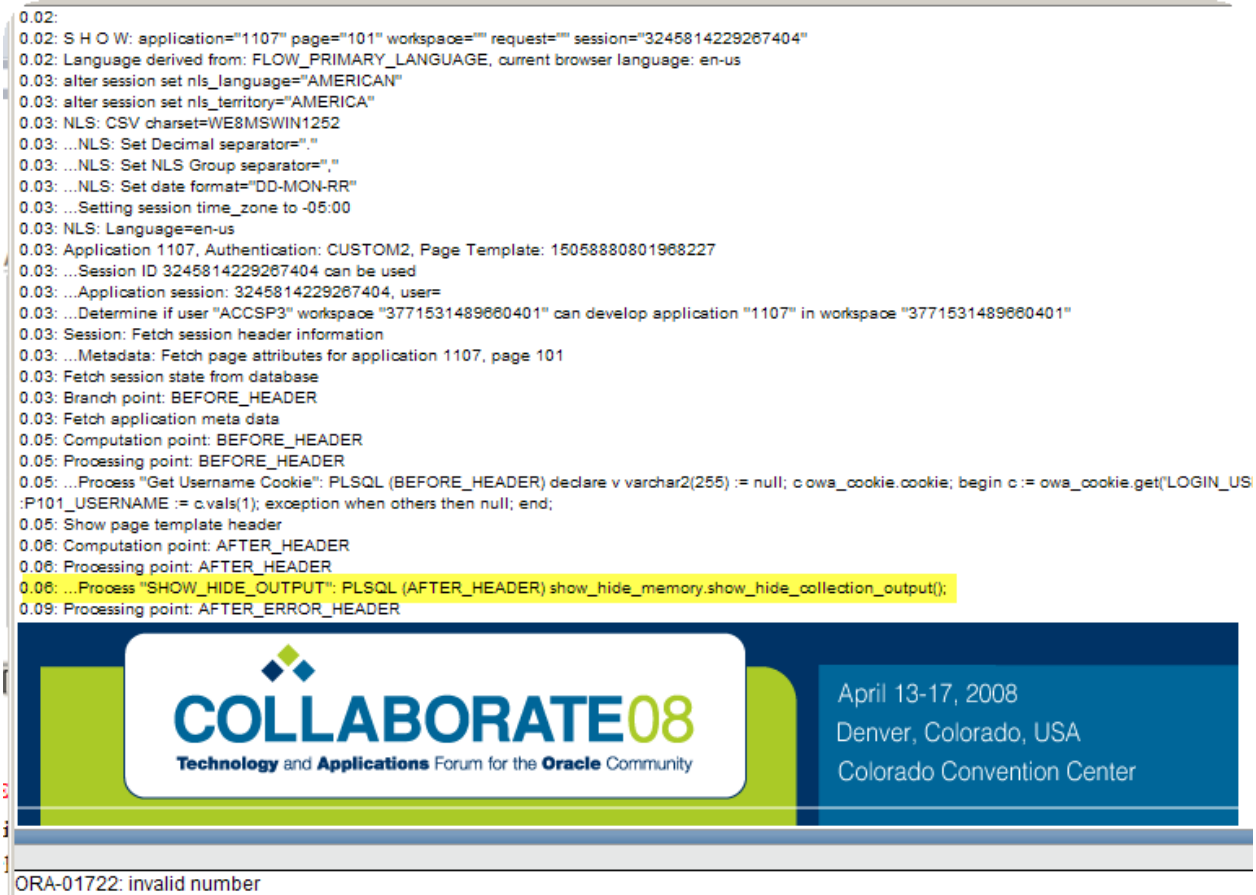


Figure 4 – Application with Error, Debug Mode Output. Error line highlighted

Debug Mode

The Debug link in the APEX Developer's Toolbar turns on the APEX Debug Mode. Debug Mode is perhaps the single most useful tool in diagnosing APEX application errors. In Debug Mode, APEX lists debug information for major steps in the page generation process on the application page, mixed in with displayed HTML elements. The list of debug messages is helpful in determining which computations, validations and processes executed, and in what order, with what values. Reviewing this list and comparing the listed actions with the intended actions, helps one discover where events are or are not firing, with correct or incorrect values. The usefulness of Debug Mode is best illustrated with a few examples. Figures 3 shows an application with a problem. Figure 4 displays the same application in Debug Mode. In Figure 4, the highlighted line indicates that the error message was thrown from the PL/SQL process SHOW_HIDE_OUTPUT, which calls show_hide_collection.show_hide_collection_output (). Now one knows where to look for the error.

```
0.01: Branch point: BEFORE_COMPUTATION
0.01: Computation point: AFTER_SUBMIT
0.01: Tabs: Perform Branching for Tab Requests
0.01: Branch point: BEFORE_VALIDATION
0.01: Perform validations:
0.01: Branch point: BEFORE_PROCESSING
0.01: Processing point: AFTER_SUBMIT
0.03: ...Do not run process "EXECUTE_PM", process point=AFTER_SUBMIT, condition type=ALWAYS, when button pressed=EXECUTE_PM
0.03: ...Do not run process "Create_MV_PINFO_ORIG_PPT", process point=AFTER_SUBMIT, condition type=NEVER, when button pressed=
0.03: ...Do not run process "BATCH_PRELOAD_REORG_TABLES", process point=AFTER_SUBMIT, condition type=, when button pressed=I
0.03: ...Do not run process "BATCH_PRELOAD_NEW_TABLES", process point=AFTER_SUBMIT, condition type=, when button pressed=PRI
0.03: ...Process "CLEAN_PPT_MATCH_TABLES": PLSQL (AFTER_SUBMIT) DECLARE v_status PLS_INTEGER := 0; BEGIN REORG.CLEA
<> 0 THEN RAISE_APPLICATION_ERROR( -20001,'Error Clearing Match Tables:'||SQLERRM); END IF; END;
0.32: ...Do not run process "CLEAR_REORG_TABLES", process point=AFTER_SUBMIT, condition type=, when button pressed=CLEAR_REC
0.32: ...Do not run process "BATCH_PROCESS_CONFIRMED", process point=AFTER_SUBMIT, condition type=, when button pressed=BATC
0.32: ...Do not run process "BATCH_PROCESS_TO_NEW_STRUCTURES", process point=AFTER_SUBMIT, condition type=, when button p
0.32: ...Do not run process "FLUSH_UNMATCHED_PPT", process point=AFTER_SUBMIT, condition type=, when button pressed=FLUSH_UN
0.32: ...Do not run process "FLUSH_REMAINING_PPT", process point=AFTER_SUBMIT, condition type=, when button pressed=FLUSH_UNI
0.32: ...Process "SET_AJAX_LOAD_MSG": ON_DEMAND (AFTER_SUBMIT) 9310448235151815
0.32: ...Session State: Saved Item "A_AJAX_LOAD_MSG" New Value="...Clearing Matches..."
0.32: Branch point: AFTER_PROCESSING
0.32: ...Evaluating Branch: AFTER_PROCESSING type: "REDIRECT_URL" button: 12360766432457077 branch: (Unconditional)
0.32: ...Evaluating Branch: AFTER_PROCESSING type: "REDIRECT_URL" button: 12361558400457080 branch: (Unconditional)
0.32: ...Evaluating Branch: AFTER_PROCESSING type: "REDIRECT_URL" button: 12360965129457079 branch: (Unconditional)
0.32: ...Evaluating Branch: AFTER_PROCESSING type: "REDIRECT_URL" button: 12361140368457079 branch: (Unconditional)
0.32: ...Evaluating Branch: AFTER_PROCESSING type: "REDIRECT_URL" button: 12361368434457080 branch: (Unconditional)
0.32: ...Evaluating Branch: AFTER_PROCESSING type: "REDIRECT_URL" button: 12361756104457080 branch: (Unconditional)
0.32: ...Evaluating Branch: AFTER_PROCESSING type: "REDIRECT_URL" button: 12069255230421234 branch: (Unconditional)
0.32: ...Evaluating Branch: AFTER_PROCESSING type: "REDIRECT_URL" button: 12362753122457081 branch: (Unconditional)
0.32: ...Evaluating Branch: AFTER_PROCESSING type: "REDIRECT_URL" button: 12373336686577997 branch: (Unconditional)
0.32: ...Evaluating Branch: AFTER_PROCESSING type: "REDIRECT_URL" button: 12376447130685154 branch: (Unconditional)
0.32: ...Evaluating Branch: AFTER_PROCESSING type: "REDIRECT_URL" button: 12426858599524165 branch: (Unconditional)
0.32: ...Evaluating Branch: AFTER_PROCESSING type: "REDIRECT_URL" button: (No Button Pressed) branch: (Unconditional)
0.34: ...Unconditional branch taken
0.00:
0.00: S H O W: application="1107" page="41" workspace="" request="" session="2799892843302560"
0.00: Language derived from: FLOW_PRIMARY_LANGUAGE, current browser language: en-us
0.00: alter session set nls_language="AMERICAN"
0.00: alter session set nls_territory="AMERICA"
```

Figure 5 – Debug Mode Debug Messages

Figure 5 shows an excerpt from a more complex set of debug messages on a page that contains numerous conditional processes. This trail of debug messages tells the developer that one process fired (CLEAN_PPT_MATCH_TABLES), and then another process (SET_AJAX_LOAD_MSG) fired. Other processes were recognized but not processed due to conditions on the processes. In this case, the second process was resetting an application item (A_AJAX_LOAD_MSG.) when it should not have. Correcting the condition on the second process resolved the problem of the wrong message being displayed.

In both examples, the debug messages helped identify where within APEX to address and solve the problem. Figures 3 and 4 illustrate how in Debug Mode APEX clearly identifies each processing point, computation point, branch point and validation point, and whether each conditional processes is executed or not. Each region and item is listed as encountered. The time element to the left of each line is a running total of CPU time. Figures 3 and 4 are abbreviated for space considerations, however, the debug messages continue interspersed with the displayed page elements through to the After Footer and End Show messages at the bottom of the page.
APEX debug messages tell which processes where firing, in what order, and with what values. From this information, one can discern whether conditions are correct and whether processes are firing in the correct order. Debug Mode is useful not only for diagnosing specific problems, but also for learning more about how APEX operates. The debug messages are a listing of each step in the APEX page generation process, from the request through all after footer actions.
Debug Mode can also be entered by setting the DEBUG component of the p parameter in the APEX URL to YES. The YES must be in uppercase to be recognized by APEX. To exit Debug Mode, click on the **No Debug** link in the Developer Toolbar, or set the APEX URL DEBUG flag to NO, again in uppercase.

Wwv_flow.debug

All of the debug messages displayed in figures 3 and 4 are APEX-generated debug messages. A developer can augment the APEX messages with additional, custom debug messages by adding code to output debug messages, much like adding debug output using DBMS_OUTPUT.PUT_LINE, by inserting calls to the APEX utility **wwv_flow.debug**.
Wwv_flow.debug takes a single string argument. The argument can be any text the developer wants to see in the debug message flow. Since the debug message flow is HTML, one can embed font and style information if desired.
Wwv_flow.debug calls can be inserted in APEX processes, computations, validations or other code constructs, as in the code in Listing 1.

```
wwv_flow.debug ( ‘In PROCESS_CONFIRMED process using value: ’||:P1_ITEM );
```

or in a stored procedure:

```
PROCEDURE show_hide_collection_output AS
  BEGIN
   -- debug message
    wwv_flow.debug( '<font color="red"><br> g_flow_id = '||
                    :APEX_APPLICATION.g_flow_id||
                    '<br> g_glow_step_id = '||
                    :APEX_APPLICATION.g_flow_step_id||
                    '</font>' );
    htp.prn('<script type="text/javascript">' || CHR(10));
    htp.prn('<!--' || CHR(10));
    htp.prn('window.onload=function(){' || CHR(10));
    FOR c1 IN
    …
```

Listing 1 – wwv_flow.debug Call

Activity

The Developer Toolbar **Activity** link displays an Activity interface, through which the user can view a series of application and session activity reports. The main categories of activity reports are Page Views, Caching, Developer Activity and Sessions, as in Figure 6. Most of these are informational and are of most value in learning how APEX operates and how developers and users are working with an application. The Developer Activity reports are perhaps more useful than others in diagnosing problems, as they form a trail map of the development process.

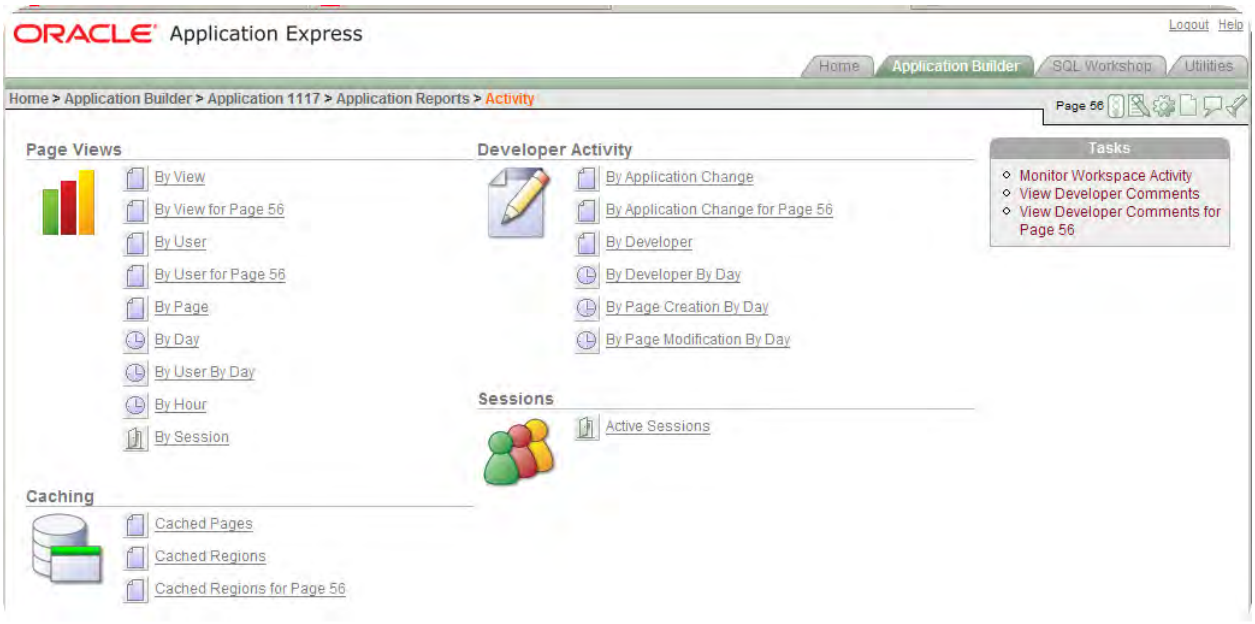
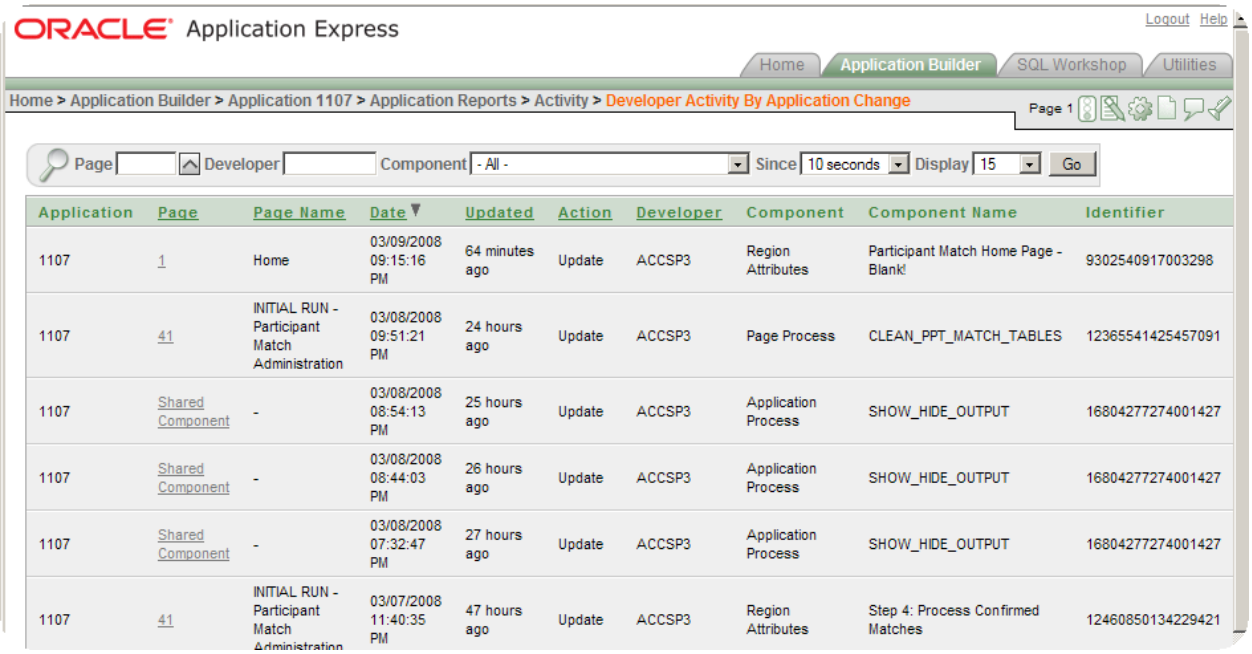


Figure 6 – APEX Activity Reports

For example, the **Developer Activity by Application Change** report, illustrated in Figure 7, is helpful in the debug process because it shows all changes made by a developer by application, page, date, developer, component and identifier. In brief, it shows who did what to what when. In cases where something used to work and now does not, this report is useful in tracing which components were touched, which is often a clue as to which components to inspect to resolve a problem.



Application	Page	Page Name	Date	Updated	Action	Developer	Component	Component Name	Identifier
1107	1	Home	03/09/2008 09:15:16 PM	64 minutes ago	Update	ACCSP3	Region Attributes	Participant Match Home Page - Blank!	9302540917003298
1107	41	INITIAL RUN - Participant Match Administration	03/08/2008 09:51:21 PM	24 hours ago	Update	ACCSP3	Page Process	CLEAN_PPT_MATCH_TABLES	12365541425457091
1107	Shared Component	-	03/08/2008 08:54:13 PM	25 hours ago	Update	ACCSP3	Application Process	SHOW_HIDE_OUTPUT	16804277274001427
1107	Shared Component	-	03/08/2008 08:44:03 PM	26 hours ago	Update	ACCSP3	Application Process	SHOW_HIDE_OUTPUT	16804277274001427
1107	Shared Component	-	03/08/2008 07:32:47 PM	27 hours ago	Update	ACCSP3	Application Process	SHOW_HIDE_OUTPUT	16804277274001427
1107	41	INITIAL RUN - Participant Match Administration	03/07/2008 11:40:35 PM	47 hours ago	Update	ACCSP3	Region Attributes	Step 4: Process Confirmed Matches	12460850134229421

Figure 7 – Developer Activity by Application Change Activity Report

View Select List on Page Definition

The View select list in the Page definition interface contains a series of useful options. Developers are most used to the default **Definition** option, which is the familiar page builder interface. The Events, Objects, History, Groups and Referenced selections contain useful information, but developers often overlook them as they are outside of the normal page builder flow. The **Events** and **References** options are most useful for learning and troubleshooting page construction. Figure 8 shows the full Page View select list.

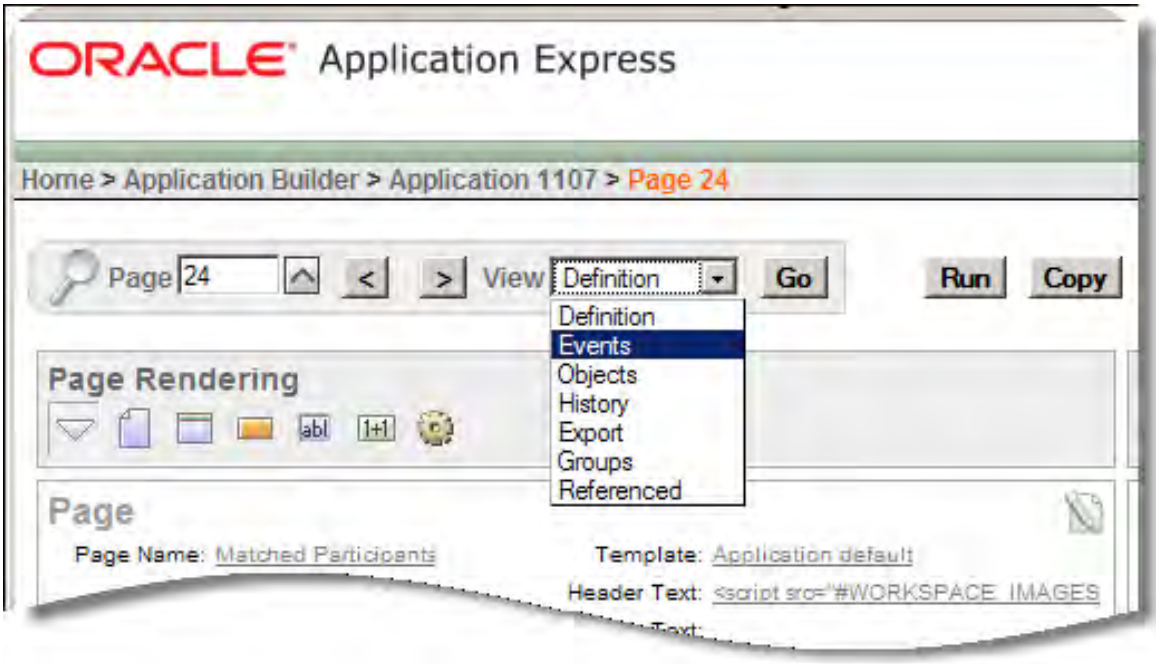


Figure 8 – APEX Page View Select List Options

Events

The Page View **Events** interface shows details of all defined page controls and processes for the current page. Events are displayed in chronological order of how APEX renders the page, executes logic and runs processes. Figure 9 shows a typical **Events** hierarchy. The default is to show all used controls and processes. The **Show All** radio option displays all possible controls and processes The **Events** interface is useful to understanding the order in which events are executed, which helps in understanding how APEX renders a page and execute the logic embedded in it. For example, the **Events** interface may show events occurring in a different order than expected, in which case a developer can take steps to correct the problem.

Referenced

The **Referenced** option of the Page View Select List displays page components and shared components referenced by the current page. These may include breadcrumbs, Page 0 items, lists, branches, tabs and parent tabs. This view is also useful in understanding the composition of the current application page. Understanding page composition helps when deciphering debug messages, watching for correct event processing.

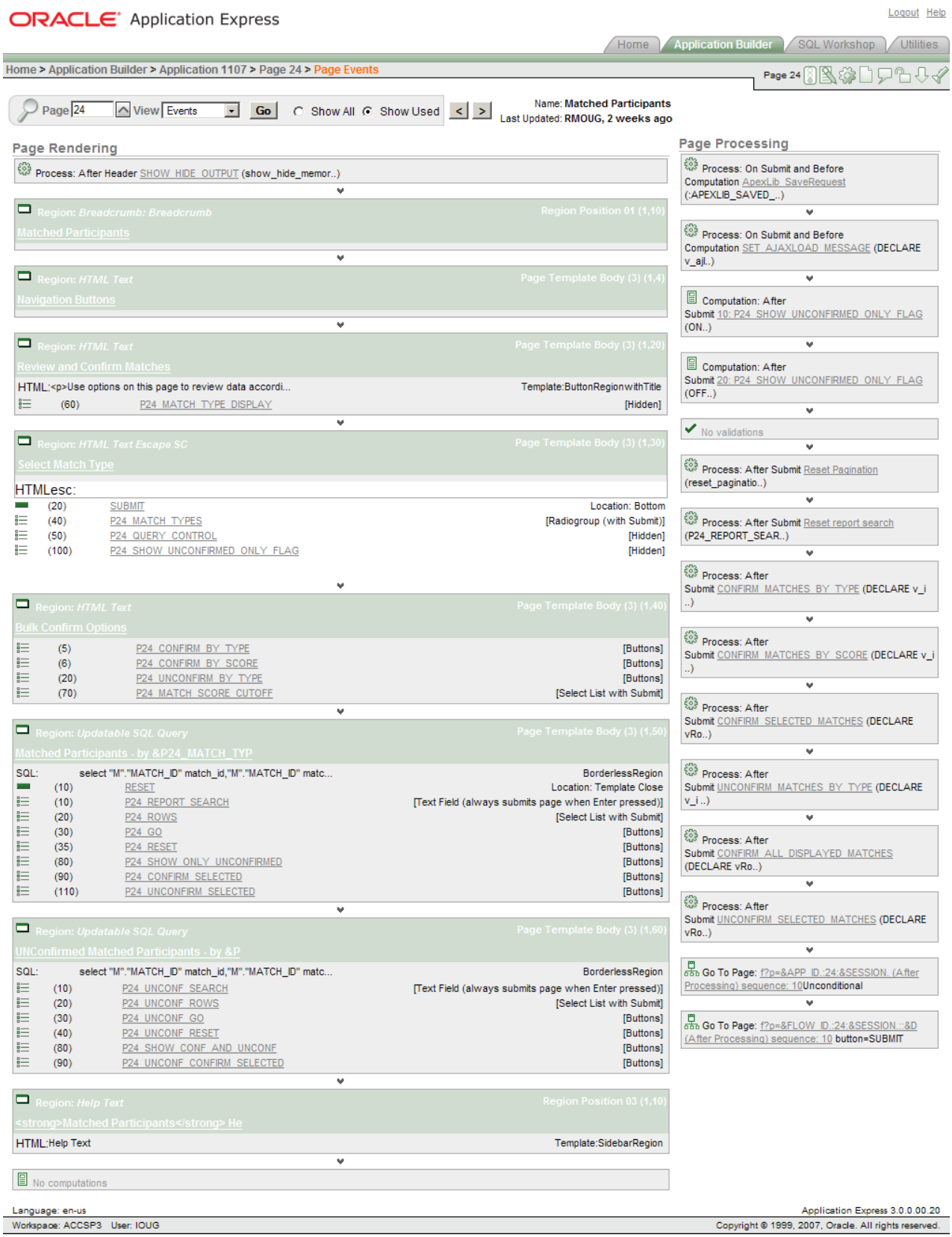


Figure 9 – Page View Events Report

Application Reports

The Application Reports are a collection of reports about the contents and activity of the current APEX application. Activity reports are one category of Application Reports. The Application Reports main menu can be reached through the Tasks menu in the sidebar region of the APEX builder application interface, as illustrated in Figure 10. These reports are not directly useful in debugging in the traditional sense of troubleshooting lines of code, but they do provide details about the content of the application and activity so far. The major report categories are Shared Components, Page Components, Activity and Cross Application.

The Application Reports → Shared Components → Database Object Dependencies report displays information on valid and invalid database objects, including PL/SQL processes. This report is described in detail in the Validation of PL/SQL Processes section below.

The reader is encouraged to check out the contents of the Application Reports to gain better understanding of APEX page generation.

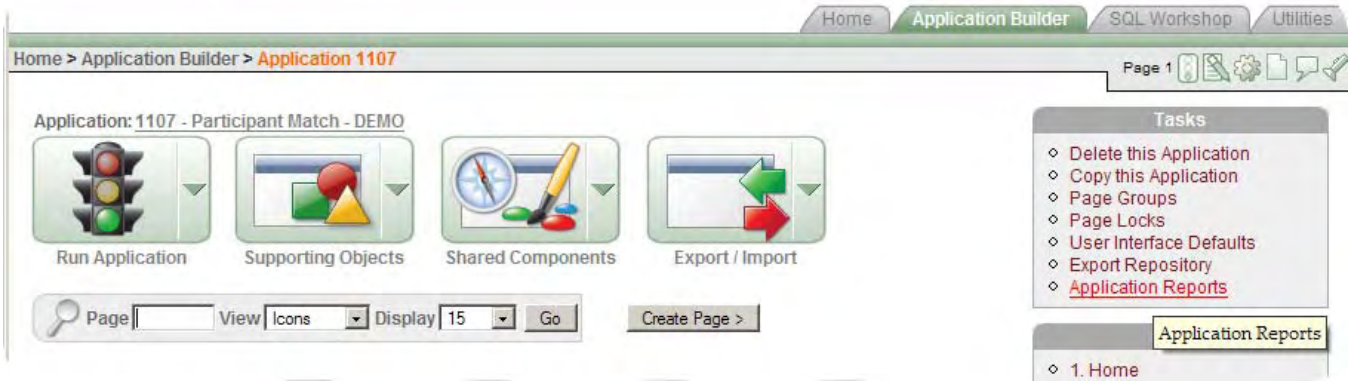


Figure 10 – Application Reports Access

DBMS_APPLICATION_INFO

APEX makes calls to DBMS_APPLICATION_INFO for each database session and each page rendered to set Module, Client Info and Client Identifier information. Module is set to APEX:APPLICATION <application number> PAGE <page number>, Identifier is set to the user and instance, and Client Info is set to the APEX logged in user. The code APEX uses to set application information is displayed in Listing 2.

```
BEGIN
  DBMS_APPLICATION_INFO.SET_CLIENT_INFO(g_user);
  DBMS_SESSION.SET_IDENTIFIER(SUBSTR( g_user,1,
                        (64 - LENGTH( g_instance)-1))||
                        ':'||TO_CHAR( g_instance));
  DBMS_APPLICATION_INFO.SET_MODULE('APEX:APPLICATION '||
                        TO_CHAR( g_flow_id),'PAGE '||
                        TO_CHAR( g_flow_step_id));
 EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
```

Listing 2 – APEX DBMS_APPLICATION_INFO Call

This information forms a log of which APEX applications and pages are executing, and what queries the pages are issuing. It can be viewed through queries to VSQL, VSQLAREA and for database versions 10.2.0.2 and higher through V$SESSION. Listings 3 and 4 show some sample queries. The APEX Utilities → Database Monitor → Top Sql reports also make use of this information, as shown in Figure 11.

```
SELECT sql_text,
       action,
       module,
       program_id,
       program_line#
   FROM  v$sql
WHERE module  LIKE '%APEX%'
```

Listing 3 – Sample V$SQL Query of APEX Application Info

```
SELECT  plsql_entry_object_id,
     plsql_entry_subprogram_id,
     plsql_object_id,
     plsql_subprogram_id,
     module,
     action
FROM   v$session
WHERE module = 'APEX:APPLICATION 1103'
```

Listing 4 – Sample V$SESSION Query of APEX Application Information – 10.2.0.2 and higher

The output of these queries shows which APEX application number and page are executing for which client name (APEX authenticated user name) and session.

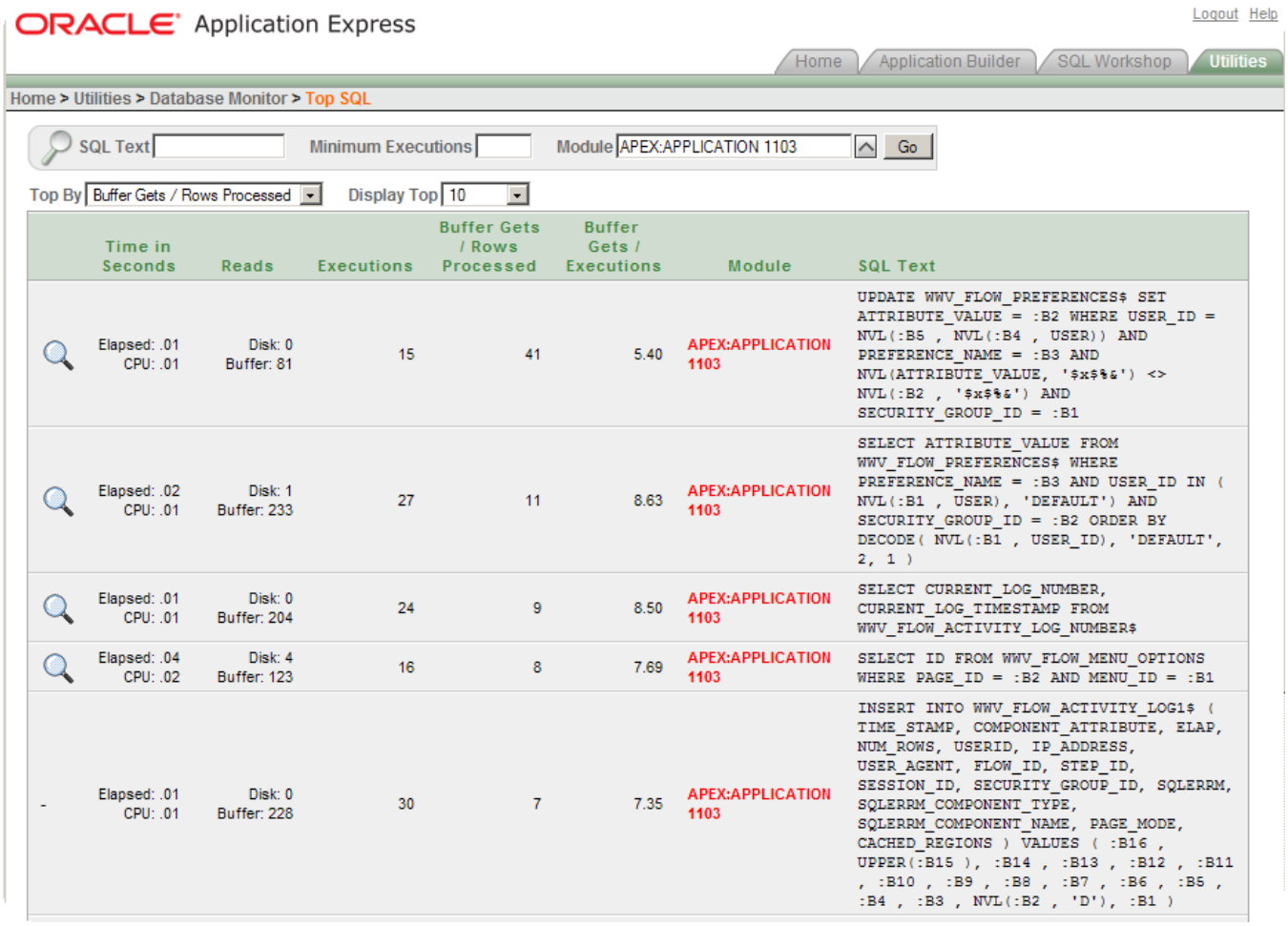


Figure 11 – APEX Utilities → Database Monitor → Top SQL Report

Developers can of course add DBMS_APPLICATION_INFO calls in stored procedures and APEX processes to further instrument an APEX application with information that will allow us to tracking execution or progress of various application processes.

?P_TRACE URL Argument

For cases when more detailed query performance and timing information is required to diagnose an APEX performance problem, one can initiate a trace file. Adding ?p_trace=YES to the APEX URL generates SQL Trace information for the page show/accept request cycle for the current session. The trace file is generated to the directory designated in the database **user_dump_dest** parameter.
An APEX URL with trace enabled looks like:
http://server.mycompany.com:7777/pls/apex/f?p=1107:1:8672053472734971:::::**?p_trace=YES**

The actual trace command that APEX issues is:

```
ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT FOREVER, LEVEL 12'
```

For alternate trace levels, one should instrument the alternate trace setting with an On Load – Before Header process to turn the alternate trace setting on:

```
EXECUTE IMMEDIATE 'ALTER SESSION SET EVENT ''<alternate trace setting>> CONTEXT FOREVER,
LEVEL 12''';
```

followed by an On Load – After Footer process to set the alternate trace setting off:

```
EXECUTE IMMEDIATE 'ALTER SESSION SET EVENTS ''< same trace settings>> CONTEXT OFF''';
```

Know your application.

This sounds obvious, but knowing what an application is supposed to do, how, and in what order is essential for clean development and testing. That a module runs with no errors is not enough, it must produce the desired result.

Modularize

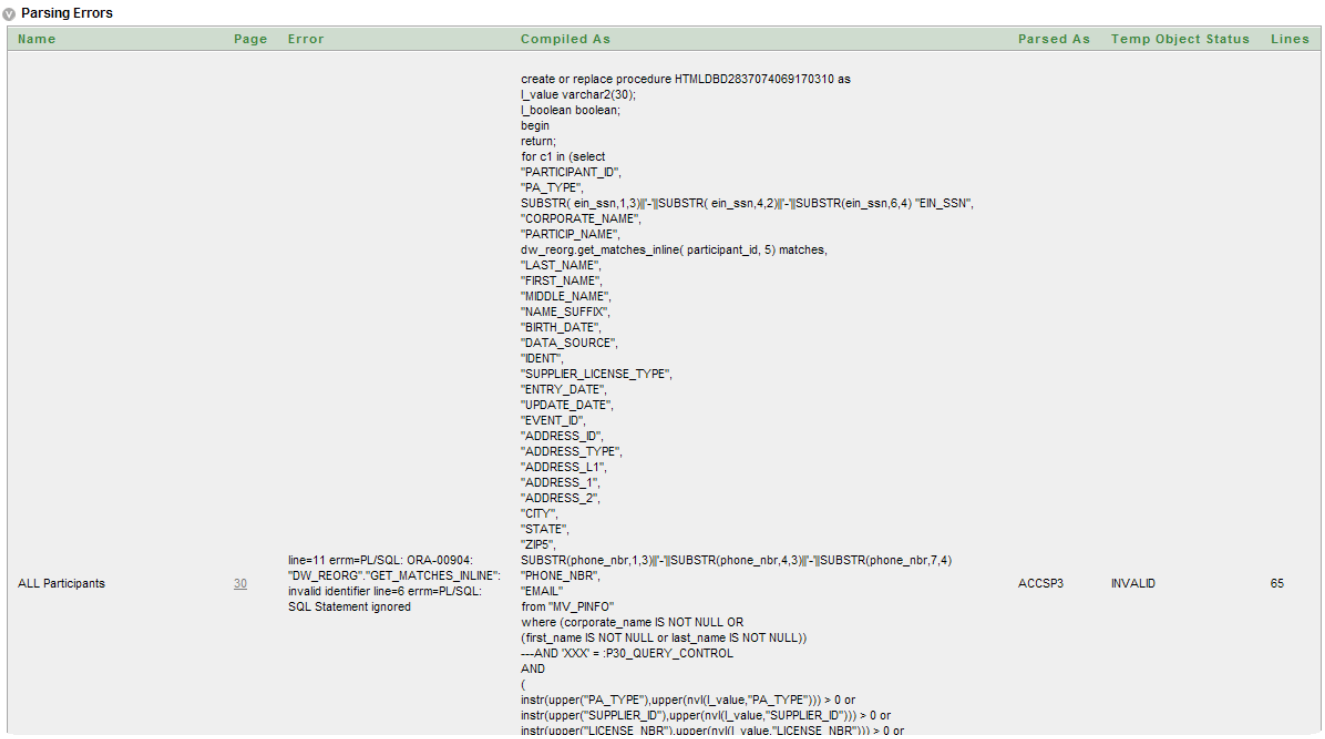
Separate business logic from presentation logic. Place business logic in code modules or web services that are distinctly separate from the presentation aspects of one's application. In APEX, the simplest way to do this is to place business logic in PL/SQL packages or web services and to call those modules in APEX page or application processes. This coding best practice isolates business logic for reuse, keeps it separate from the presentation interfaces and enables easier debug and maintenance of the modules.

Build and Tune Outside First

The first rule in building and optimizing SQL and PL/SQL within APEX is to ensure that the code runs outside of APEX. Build, test and tune queries and code modules at a SQL prompt, or in a PL/SQL IDE such as SQL Developer or TOAD. Perform all preliminary tuning and debugging of PL/SQL modules outside of APEX, in an environment where one can leverage the building and tuning features of the IDE.

Bind

Within APEX, use bind variables in queries, conditions and other application logic to reference session state values. As in PL/SQL, use of bind variables usually the more efficient way to reference values. Again, this is a matter of optimizing APEX performance through general PL/SQL best practices.



Parsing Errors

Name	Page	Error	Compiled As	Parsed As	Temp Object Status	Lines

ALL Participants	30	line=11 errm=PL/SQL: ORA-00904: "DW_REORG"."GET_MATCHES_INLINE": invalid identifier line=6 errm=PL/SQL: SQL Statement ignored	create or replace procedure HTMLDBD2837074069170310 as
			l_value varchar2(30);
			l_boolean boolean;
			begin
			return;
			for c1 in (select
			"PARTICIPANT_ID",
			"PA_TYPE",
			SUBSTR(ein_ssn,1,3)||'-'||SUBSTR(ein_ssn,4,2)||'-'||SUBSTR(ein_ssn,6,4) "EIN_SSN",
			"CORPORATE_NAME",
			"PARTICIP_NAME",
			dw_reorg.get_matches_inline(participant_id, 5) matches,
			"LAST_NAME",
			"FIRST_NAME",
			"MIDDLE_NAME",
			"NAME_SUFFIX",
			"BIRTH_DATE",
			"DATA_SOURCE",
			"IDENT",
			"SUPPLIER_LICENSE_TYPE",
			"ENTRY_DATE",
			"UPDATE_DATE",
			"EVENT_ID",
			"ADDRESS_ID",
			"ADDRESS_TYPE",
			"ADDRESS_L1",
			"ADDRESS_1",
			"ADDRESS_2",
			"CITY",
			"STATE",
			"ZIP5",
			SUBSTR(phone_nbr,1,3)||'-'||SUBSTR(phone_nbr,4,3)||'-'||SUBSTR(phone_nbr,7,4)
			"PHONE_NBR",
			"EMAIL"
			from "MV_PINFO"
			where (corporate_name IS NOT NULL OR
			(first_name IS NOT NULL or last_name IS NOT NULL))
			---AND 'XXX' = :P30_QUERY_CONTROL
			AND
			(
			instr(upper("PA_TYPE"),upper(nvl(l_value,"PA_TYPE"))) > 0 or
			instr(upper("SUPPLIER_ID"),upper(nvl(l_value,"SUPPLIER_ID"))) > 0 or
			instr(upper("LICENSE_NBR"),upper(nvl(l_value,"LICENSE_NBR"))) > 0 or
				ACCSP3	INVALID	65

Figure 12 – Database Object Dependencies → Compute Dependencies, Parsing Errors

Validate PL/SQL Processes – Compute Dependencies

APEX includes a utility to validate all PL/SQL processes in an application, in a somewhat round-about way. The Application Reports → Shared Components →Database Object Dependencies report menu contains a **Compute Dependencies** button. Compute Dependencies check for any PL/SQL errors in user-defined PL/SQL. Compute Dependencies essentially compiles each PL/SQL module checking for object dependencies. Generation of this report will take a few minutes. In the resulting report, the **Parsing Errors** in the Hide/Show region at the bottom of the report are cases where the PL/SQL module will not compile. Figure 12 shows an excerpt from the Parsing Errors section of a dependencies report. The Compute Dependencies option is a one-stop way to verify that all PL/SQL modules in an application are valid.

Remote Debugging

Sometimes one really needs to see what is happening when a PL/SQL module is called from within APEX. Finally, one gets to step through some code! One can debug the PL/SQL in an APEX session using an IDE that supports remote debugging. This paper describes remote debugging using SQL Developer, though TOAD or JDeveloper should work similarly.

Note that remote debugging only works when the debug tool has a clearly defined and accessible stored procedure – the code module must exist in the database. This is another reason to modularize code.

SQL Developer Remote Debug

Perhaps the most important feature SQL Developer offers the APEX developer is the ability to debug PL/SQL processes through the remote debug feature. The SQL Developer debug listener waits for a connection from the APEX session. Once the two are connected, a developer can step through PL/SQL code in the SQL Developer interface.

To initiate a remote debug session:

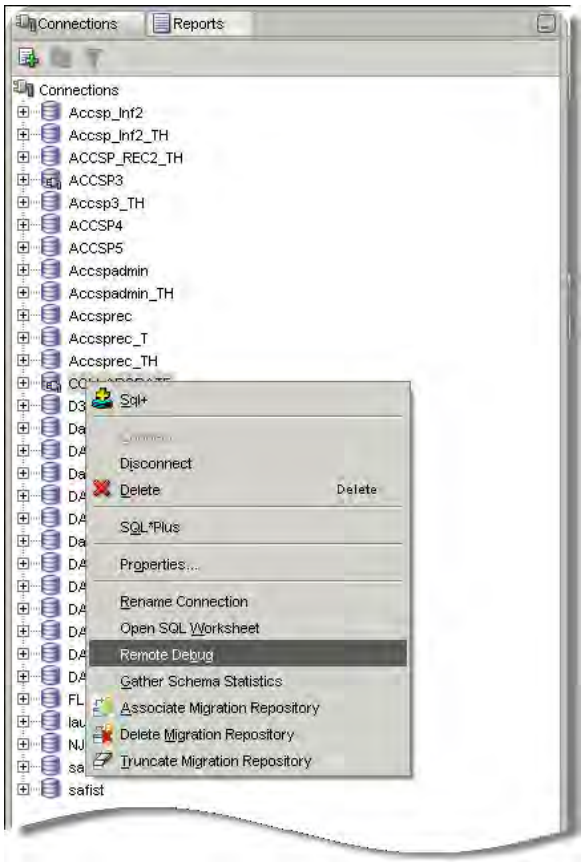


Figure 13 – SQL Developer Remote Debug Option

1. Create a SQL Developer database connection using File → New Connection and fill in the appropriate information.
2. Ensure the that user of the database connection has the necessary debug privileges:
```
DEBUG CONNECT SESSION
DEBUG ANY PROCEDURE
```
3. Connect to the database by opening the database connection.
4. Start a remote debug "listening" session from SQL Developer by **right clicking** on the Database Connection in the Connections navigator and selecting the Remote Debug option, as in Figure 13.

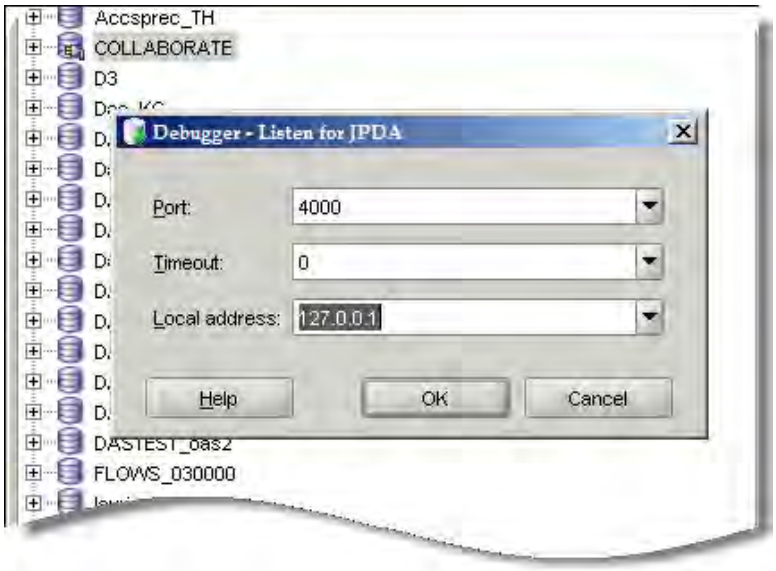


Figure 14 – Remote Debug Listen Port and IP Address

Enter the listening port number and the IP address of the machine with the database, as in Figure 14. The range of ports can be set in a SQL Developer Preferences (Tools→ Preferences from the SQL Developer menu).

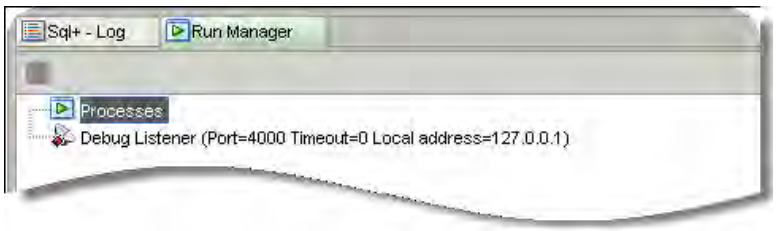


Figure 15 – SQL Developer Debug Listener Started

When the Debug Listener is started, the Run Manager will display the listener process, as in Figure 15.

5. Invoke the debugger from the APEX application by this call:

```
DBMS_DEBUG_JDWP.CONNECT_TCP( '127.0.0.1', 4000);
```

This can be accomplished in a variety of ways. For example, use a button that calls a page or application process(an easily removable feature that can deleted or set to Never display when debugging is done), or embed the call to the debugger at the start of the process to be debugged.

6. In SQL Developer, set breakpoints in the code mode to be debugged.

7. Run the application to execute the code to be examined. Control will automatically switch to SQL Developer when the application reaches the code with the breakpoint.
8. Debug the code through the SQL Developer interface.

Remote debugging is not an every-day tactic, but does help for cases where it is not obvious what values a code module is picking up from APEX, and there is no visible way to track code progress. Remember to end the Debug Listener process when through debugging.

SQL Developer - APEX Reports and User-Defined Reports

SQL Developer has built-in reports for viewing APEX applications, schemas, pages and workspaces. These high-level reports are not particularly useful for debug activities. However, they are examples of queries against the FLOWS schema. Developers can build their own queries against the FLOWS repository and save them as User Defined Reports.

Web Development Tools

The APEX-supplied and PL/SQL-focused debug options are mainly focused on the database-centered page generation processes. To address appearance and page behavior aspects of an APEX application, one must turn to a web development tool. The key feature of web development tools is the ability to view and edit web interface components such as the HTML source code, the CSS style sheet and JavaScript.

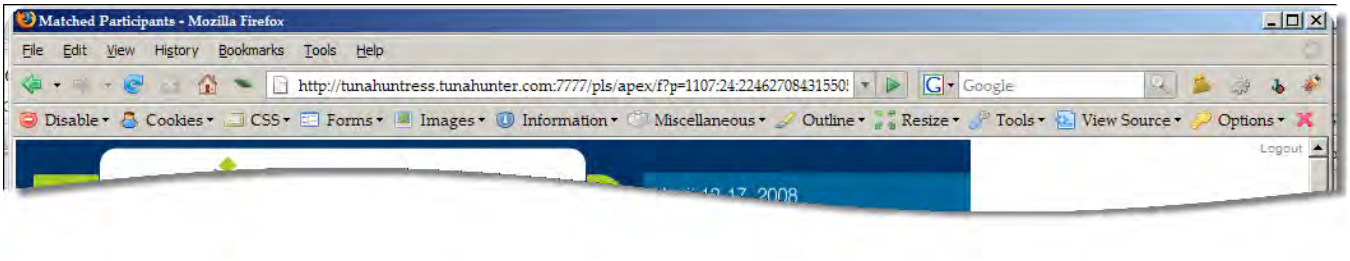


Figure 16 – Web Developer Tool Bar

When all of one's APEX development is within the APEX wizards and builder interface, a developer may never need the features of a web development tool. As soon as one ventures into customizations using the APEX_ITEM and collections API's, custom user interface templates, or specialized JavaScript, a web development tool is essential.

The details of web development tools are beyond the scope of this paper, however, APEX developers performing customizations should eventually become familiar with one or more of these tools. Web development tools range for expensive interfaces designed for professional use to shareware and freeware for general public web development use. Many are particularly suited to one form of development or another. Two tools that are particularly popular for and suited to APEX development are the Firefox add-ins Web Developer and Firebug. (This is sure hint that this author develops primarily in Firefox!). These are mentioned for their breadth of features, ease of use and cost (free). The following sections give an overview of each tool and a few examples of their use. One can of course use any web development tool that is compatible with the development browser.

Web Developer

Web Developer is a Firefox browser add-in that contains a wide variety of web developer utilities. Installation is a simple as clicking the Install Now button on the download page from a Firefox browser window. Web Developer will automatically install in the Firefox browser. When Firefox restarts, the Web Developer toolbar will appear, as in Figure 16. All Web Developer options are also available from the Firefox browser Tools → Web Developer menu.

The Web Developer main menu categories are Cookies, CSS, Forms, Images, Information, Miscellaneous, Outline, Resize, Tools view Source and Options. The category headings do not do the contents justice – there are a lot of useful tools here for figuring out exactly what's what with the HTML, CSS and JavaScript on your APEX page. APEX developers who add AJAX features will be most interested in the ability to examine element details and JavaScript. Developers who build custom page templates and themes will appreciate the Form, Information and Outline menus, and the myriad of options for displaying HTML element information.

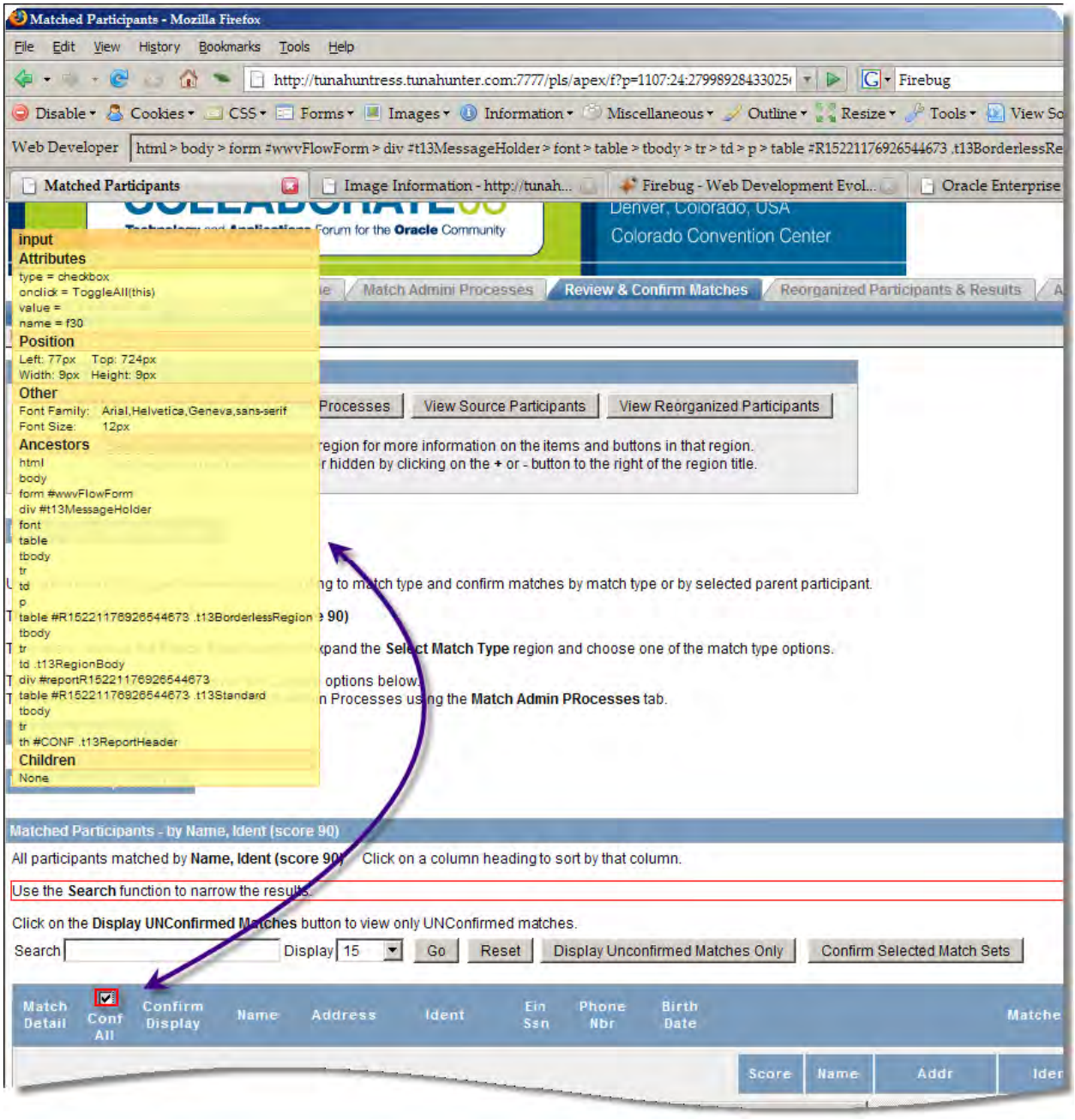


Figure 17 – Web Developer Display Id & Class Details Option

To illustrate Web Developer in action, consider the case where a developer has added a checkbox to a report, a Select All checkbox in the header, and written a PL/SQL process that reads the values of the checkbox and updates records in the database accordingly. Using the Web Developer Information→ Display Id & Class Details option helps the developer verify the Id of the checkbox element, so he or she can be sure to reference the correct element in the PL/SQL. The Information → Display Element Information displays details of each element on an HTML page, as one clicks on the element. Figure 17 shows the Display Element Information option for the highlighted checkbox.

The CSS→ View CSS and Edit CSS options are most helpful when editing page, region and item templates, or creating a new theme. View CSS displays the current class settings. Edit CSS allows one to change those settings and immediately see the results. One can then save or discard the edited CSS. This feature is handy when editing templates to match a corporate user interface, or any case where a developer needs to make element style modifications.

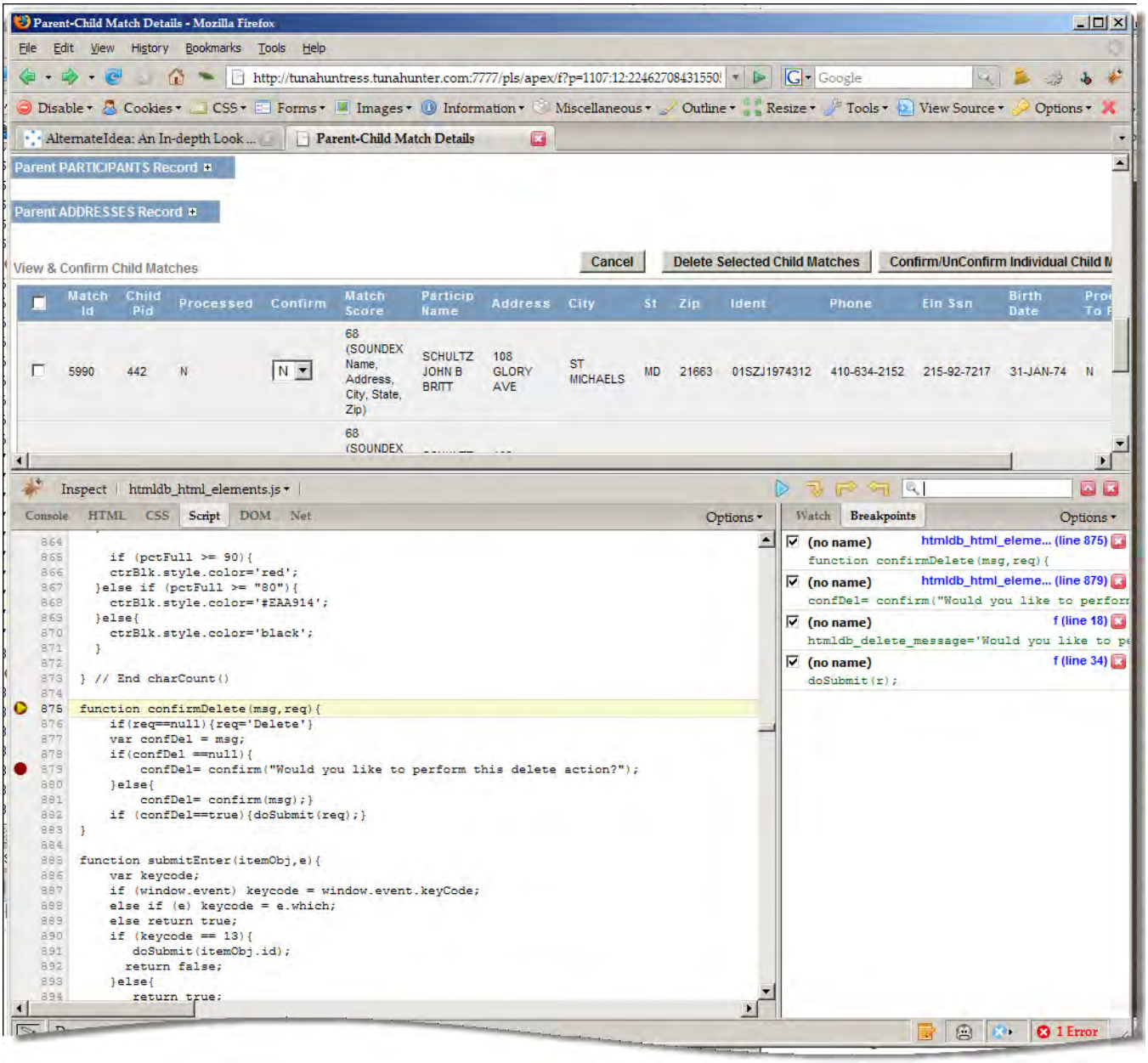


Figure 18 – Firebug JavaScript Debug and Breakpoint Interface

Firebug

Firebug offers many of the same features as Web Developer, in a slightly different format and package. Firebug is also a Firefox add-in, and is installed by clicking on the Install Now button on the Firebug home page. Firebug can be started as a pane in the same window or in a separate window. The main Firebug interface headings are Console, HTML, CSS, Script (for JavaScript) and Net.

The essential features to view and edit HTML, CSS, and JavaScript are all there. The Firebug JavaScript Console is very helpful for informational messages and for testing custom JavaScript. The HTML tab displays all HTML for the page in hierarchical format, expandable by main HTML tag. The CSS tab displays all CSS for the page. The Script tab displays all JavaScript and the JavaScript debug interface. The DOM tab displays all DOM element descriptions. The Net tab displays network header/response information and timing. Firebug offers the developer a lot to work with!
The JavaScript debugger is a great feature, though sometimes finicky to get started in this author's experience. When it is cooperating, use of the Firebug JavaScript debugger is a simple as opening Firebug, selecting the Script tab, selecting the JavaScript file to access, setting a break and refreshing the page. Figure 18 shows the Firebug JavaScript debugger in action.
A full explanation of the capabilities of Web Developer or Firebug is beyond the scope of this paper. Developers who venture beyond the APEX wizards into adding their own templates, themes, JavaScript or customized processes using the APEX API's should check out the features of Web Developer, Firebug, or some other web development tools that offers similar features.

Code Instrumentation

A paper on debug options would not be complete without a comment on code instrumentation. Developers often overlook this useful tool for tracking what is going on in an application during development and in production. Code instrumentation is the process of a developer purposely inserts informational messages, hidden or displayed (but usually hidden), to indicate what the application is doing when. Usually instrumentation is placed in an application to monitor progress, to monitor executions, to log executions, or for debugging. Instrumentation may be as simple as a DBMS_OUTPUT.PUT_LINE message (or its HTML equivalent htp.p), or a more complex system of inserts into journal tables to track execution details.
Code instrumentation is a topic in itself. It should be planned carefully and designed to serve its purpose without interrupting the normal production flow of an application. Developers have several ways to instrument APEX code, depending on the type of process, the type of code, and what needs tracking.
If adding debug messages, consider the IF V ('DEBUG') construct to toggle instrumentation on or off with the APEX Debug Mode. Use wwv_flow.debug('Your Message Here') to add custom debug messages. For saving information that is more extensive or a record-by-record log of execution progress, insert records into a separate "journaling" table, a table created specifically to hold the informational messages. Use calls to DBMS_APPLICATION_INFO in cases when one needs to track application execution in production environments. Keep messaging to a minimum and as unobtrusive as possible if instrumentation will be left in production code.

Test and Debug in Different Browsers

As with all web development, it is essential to test in all browsers. Do not assume that since APEX is an Oracle product that all APEX-produced applications will be clean in all browsers. Browsers update frequently, and user settings for cookies and JavaScript in particular vary greatly. Test at a minimum in Internet Explorer 6 and 7, Firefox and a MAC-orient browser such as Safari and Opera.

Debug Options – Triage

With so many APEX debug options, where does one start? In general, triage for an APEX problem is that same for any other coding problem. Logically figure out roughly where the problem is and then use the appropriate tool narrow in and correct it.
The first step is to Think. What is happening, what could cause that result. This most often narrows the issue down to SQL, PL/SQL, the APEX flow of events, HTML, CSS or JavaScript. Use APEX Debug Mode to determine what and where the problem is in the APEX flow. Once the issue is generally diagnosed, use the appropriate debug option to address and resolve the problem.
Incorrect report results or data results indicate SQL or PL/SQL or APEX process errors. For SQL and PL/SQL issues ensure that the code executes properly outside of APEX by using an IDE or SQLPlus. Ensure the correct values are in use through the Session interface. If necessary start a remote debug session to examine exactly what occurs as the code

executes. Performance issues that cannot be resolved by SQL and PL/SQL tuning outside of APEX may require a trace file to properly diagnose and address. Use of the #TIMING# indicator may narrow a performance issue to a certain region, so one can focus on the SQL in that region. Events or processes not firing indicate problems in the sequence of flow events, possibly caused by the conditions and session state values that control the conditions Use APEX Debug Mode, the Session interface and the APEX Activity Reports and utilities to further track these issues. Turning processes or page elements on and off by changing the element conditions also helps track and solve event flow problems. Incorrect or misaligned elements on a page indicate HTML errors or places where additional HTML tags or styles may be needed to produce the desired result. Use web development tools to view and edit the HTML, JavaScript and styles sheets to address interface issues.

Many issues will involve a combination of tactics. A methodical approach in resolving one "layer" of issues at a time is recommended. For example, correct all SQL and PL/SQL issues first, then ensure all events are firing in order and under the correct conditions, and then address interface issues. Above all knowing which debug tool and technique to employ comes from knowing one's application, knowing how APEX operates, and experience.

Debug Options – Less Confused?

To summarize, APEX is a blend of technologies – PL/SQL, SQL, HTML, CSS, JavaScript and more – and debugging APEX necessitates a blend of debug tactics. Fortunately, there is a variety of readily available options for debugging APEX applications. APEX supplies a series of utilities for viewing application, page and event information, ranging from the Application Reports series to the Page View Events list to the Database Objects Dependencies report that helps one locate database objects with errors. APEX Debug Mode displays informational messages of each step in the page generation process. The p_trace APEX URL argument enables creation of a trace file, which can be reviewed for SQL execution, timing and tuning information. One can debug PL/SQL modules using the SQL Developer Remote Debug capability. HTML, CSS and JavaScript issues can be view and resolved using web development tools such as Web Developer and Firebug. They key is to know how APEX generates pages, to know the tenets of one's application, and to employ a variety of tactics to meet the challenge at hand.

Thus, the APEX debug conundrum, "How do I debug APEX?" is solved. The remaining puzzle is "What is wrong with my code?" which is exactly where one should be at the start of a debug session.

Resources

The APEX OTN Forum - http://forums.oracle.com/forums/forum.jspa?forumID=137 . The APEX user community is active and growing. It is your best source for APEX how-to's and answers for users of all abilities. Know it, Bookmark it. Use it. Contribute to it.

SQL Developer - Home Page on OTN http://www.oracle.com/technology/products/database/sql_developer/index.html

Web Developer - http://chrispederick.com/work/web-developer/ Web Developer is a free Firefox (Flock, SeaMonkey) add-in that adds a menu and a toolbar that contains a wide variety of web developer tools.

Firebug - http://www.getfirebug.com – Firebug is a free Firefox extension web development tool for edit and monitoring of CSS, HTML and JavaScript.

The APEX Studio - http://htmldb.oracle.com/pls/otn/f?p=18326:1:3795991895340045::NO::: There are a growing number of APEX applications out here that are free to download and use as is or modify for your own purpose. This is the Oracle-hosted APEX hosting site. It includes applications, utilities, themes, tips and tricks. Watch for early adopter versions of APEX.

The APEX OTN Technology Center –
http://www.oracle.com/technology/products/database/application_express/index.html The Oracle Technology Network Application Express home page.

APEX Blogs – There are numerous APEX blogs with quality examples and advice. See the APEX Blog Aggregator for a start, http://www.apexblogs.info.

Tuning the Large Pool for RMAN

Anthony D. Noriega, MBA, MSCE, BSSE, OCP

Abstract

The purpose of this study is to summarize a set of benchmark results that affect the appropriate sizing of the Large Pool, in order to tune RMAN tasks, in particular, backup speed and backup size, by customizing an RMAN backup and recovery (BR), business continuity (BC) and planned disaster recovery (DR) strategies, for both Oracle-based and third-party vendor software or hardware solutions. The discussion introduces a comparison between Oracle recommended sizing for the Large Pool and the specific effect of sizing the Large Pool with different values, and the outcome affecting the size of backup pieces, overall backup size, and overall backup and restore turnaround time. Several case studies results are introduced while a set of practical conclusions on this experiment are presented in relation to size and time, as the Large Pool size is either decreased or increased. These results, being practical in nature, permit the optimization of disk and tape storage resource, and specific window of time.

1. Introduction: Background and Experimentation

There are many possible scenarios where options to size, control, and optimize performance. In this experiment the most common options are used. In a future experiment RMAN options such as compression and backup encryption will be carefully considered.

Among the key objectives to find the best approach to tune the Large-Pool tuning for RMAN, it is possible to highlight the following, namely:

- Provide a concise approach to tune the Large Pool for RMAN.
- Analyze and compare Oracle recommendations and custom settings.
- Present a robust approach to optimize RMAN time and storage space.
- Derive a series of technical arguments and useful rules to satisfy a BR/BC/DR optimization methodology.

While the database and backup sizes more significantly validate the following experimentation results, generically, it is for the large and very large databases (VLDB) where the most important contributions are possible in both terms of time and storage space. In the case of VLDB, we could contemplate some thresholds in operating system specific environments, which could translate into hours of economy when a database size threshold of several terabytes is in place, based on the expectation of this experimentation of historic backup duration. The results are also useful when there are constraints affecting the storage space or the duration of the RMAN operation, such as a reduced tape or disk space or a narrow window of time to perform the task. The experimentation does not contemplate compression usage, but sizing the large pool will have a proportional approach and congruent results.

Furthermore, unlike the Shared Pool, the Large Pool lacks a Least Recently Used (LRU) approach to segment management, and there is more interest in observing results rather than attempting to use existing literature or derive new algorithmic approach, for which RMAN metrics and statistics become of greater observational and correlation validity.

2. Large Pool Functionality and Oracle Architecture

The Large Pool obeys to a few functional tasks, namely:

- Providing the session memory for the shared server and the Oracle XA interface (used where transactions interact with more than one database)

- Supplying the memory for I/O server processes
- Supporting the Oracle Database backup and restore operations
- Meeting the space and functional requirements when used by backup processes for disk I/O buffers.

Besides, the Large Pool is a memory structure part of the System Global Area (SGA).

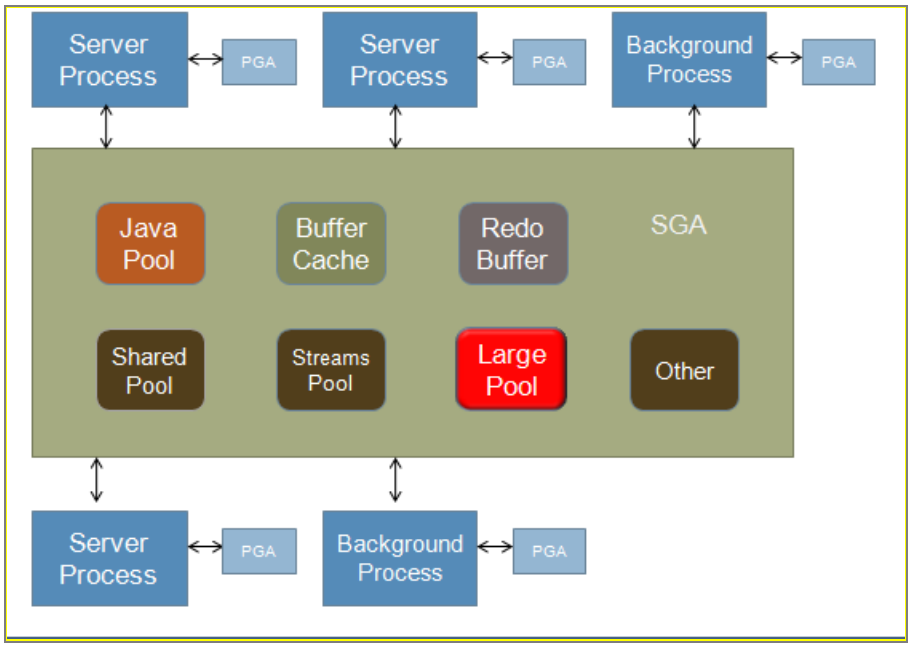


Exhibit 1. Oracle Instance Architecture: Memory Structures and Background Processes

3. Experimentation Environments

Although there is historic experimentation on Oracle systems using various flavors of UNIX –such as AiX, AT&T SVR4, HP-UX, and Solaris–, the outcome of this experiment is based on observations in the following environment:

- Linux 4-node RACs, 9i and 10g
- SAN architecture using Symantec (Veritas) Netbackup
- MPP (Linux Intel 64-bit 8/16 dual-core processor RH Linux servers).
- Oracle RAC and Data Guard in place (MAA).
- Incremental backup level 0 mostly.
- Block Change Tracking (Oracle10g/11g) was not available in most instances.

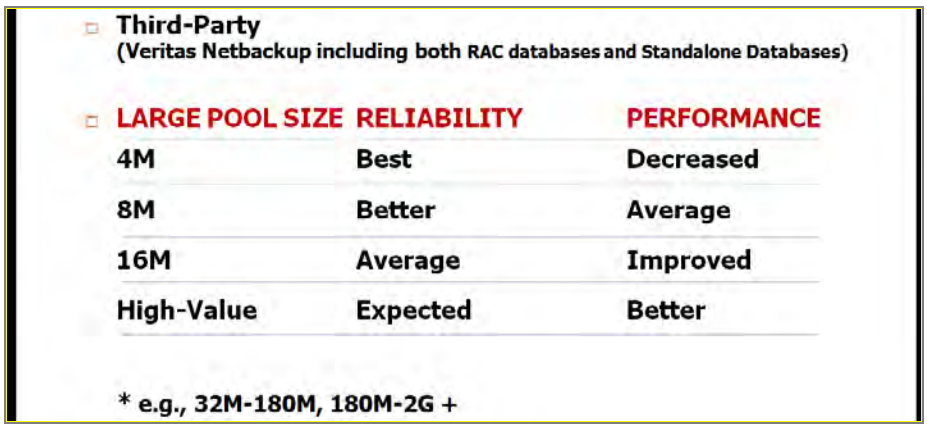
4. Oracle Recommendations to Size the Large Pool

Although tuning and sizing the Large Pool for RMAN is a custom task, Oracle recommends 16M for Orace10g, 8M for Oracle9i, and 16M-32M for Oracle11g, which is both Operating System and database architecture dependent. These are also usually the default values used by the Database Configuration Assistant (DBCA) for each one of these Oracle versions. Therefore, for an Oracle10g database, a DBA would normally see the following entry in the init.ora parameter:

```
LARGE_POOL_SIZE=16M
```

5. Custom Settings to Size the Large Pool

Although it is less likely to see a Large Pool Size setting of 180M or larger, there are scenarios where RMAN tasks may suggest the usage of large segments for the purpose to attain maximum performance and throughput, i.e., streaming of physical data onto the backup device, whether it is a tape (SBT) or a disk. The next two exhibits clearly convey the reliability and performance interdependence with respect to the Large Pool size.

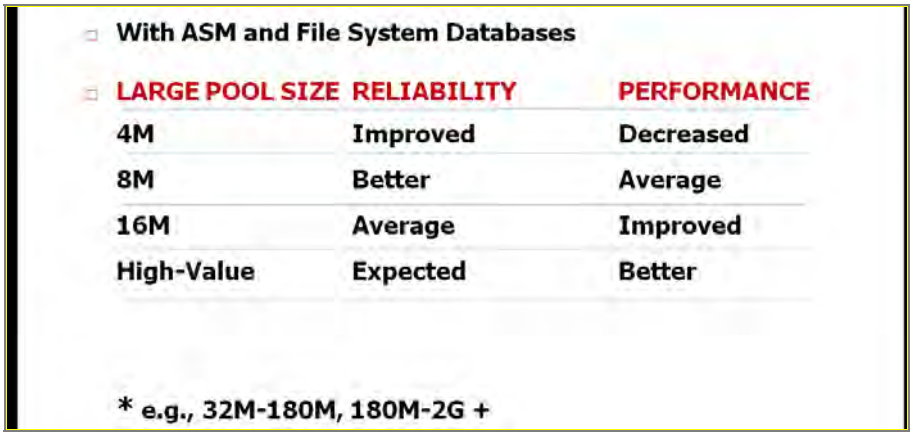


Third-Party
(Veritas Netbackup including both RAC databases and Standalone Databases)

LARGE POOL SIZE	RELIABILITY	PERFORMANCE
4M	Best	Decreased
8M	Better	Average
16M	Average	Improved
High-Value	Expected	Better

* e.g., 32M-180M, 180M-2G +

Exhibit 2. Custom Settings and Observed Results (with third-party vendor product)



With ASM and File System Databases

LARGE POOL SIZE	RELIABILITY	PERFORMANCE
4M	Improved	Decreased
8M	Better	Average
16M	Average	Improved
High-Value	Expected	Better

* e.g., 32M-180M, 180M-2G +

Exhibit 3. Custom Settings and Observed Results (ASM and conventional databases)

The main justification to properly size the Large Pool is that the usage of larger segments improves the performance of RMAN tasks. In fact the value of the LARGE_POOL_SIZE parameter typically represents the smallest segment used, which matches requirements that the Shared Pool will not normally guarantee to meet easily. The next exhibit clearly suggests this perception.

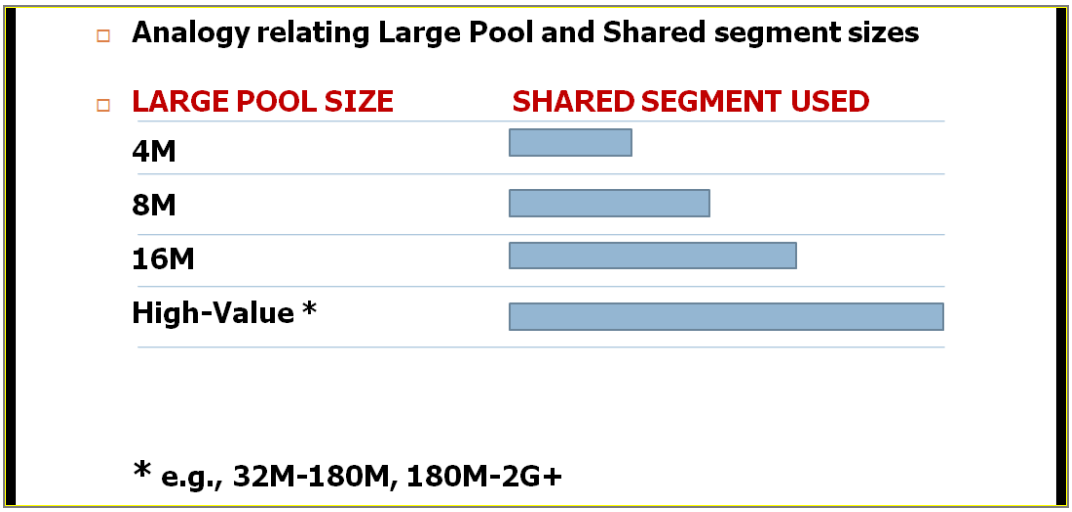


Exhibit 4. Analogy of the Large Pool Size compared to the Memory Segment Usage

6. Large Pool Size and Backup Size and Duration

A careful observation of the backup size and duration in relation to the Large Pool size is illustrated in the next exhibit.

LARGE POOL SIZE	BACKUP PIECE SIZE	ESTIMATED BACKUP SIZE $\sum_{i=1}^{n} s_i$	DURATION
2^jM	$S_i=S_{i-1}+j\Delta s$	$\sum_{i=1}^{n} s_i = n(S_0+j\Delta s)$	$t_0 = t_0 - \Delta t$
....	....	....	....
32M	$S_3=S_2+3\Delta s$	$\sum_{i=1}^{n} s_i = n(S_0+3\Delta s)$	$t_1 = t_0 - \Delta t$
16M	$S_2=S_1+2\Delta s$	$\sum_{i=1}^{n} s_i = n(S_0+2\Delta s)$	t_0
8M	$S_1=S_0+\Delta s$	$\sum_{i=1}^{n} s_i = n(S_0+\Delta s)$	$t_{-1} = t_0 + \Delta t$
4M	S_0	$\sum_{i=1}^{n} s_i = nS_0$	$t_{-2} = t_0 + 2\Delta t$

Exhibit 5. Large Pool Size and Backup Size and Duration Comparison Table

This exhibit clearly suggests that the backup size increases as the Large Pool Size increases while the RMAN task (e.g., backup) duration actually decreases, and vice versa; i.e., as the Large Pool size is decreased, as described, the RMAN task (e.g., backup) duration increases and the overall backup set size decreases. The small differential could run from a fraction of 1% to high single digit to low double variations, depending on the database size, the computing resources, and other network data streaming settings. Consequently, the following exhibits clearly correlate the Large Pool size and the corresponding RMAN task reliability and performance.

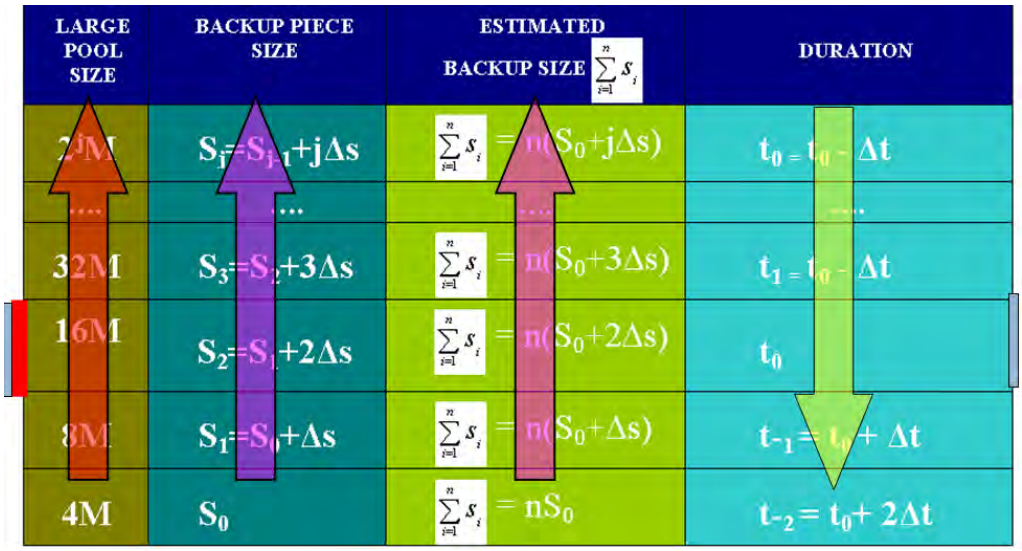


Exhibit 6. As the Large Pool Size Increases, Better Performance

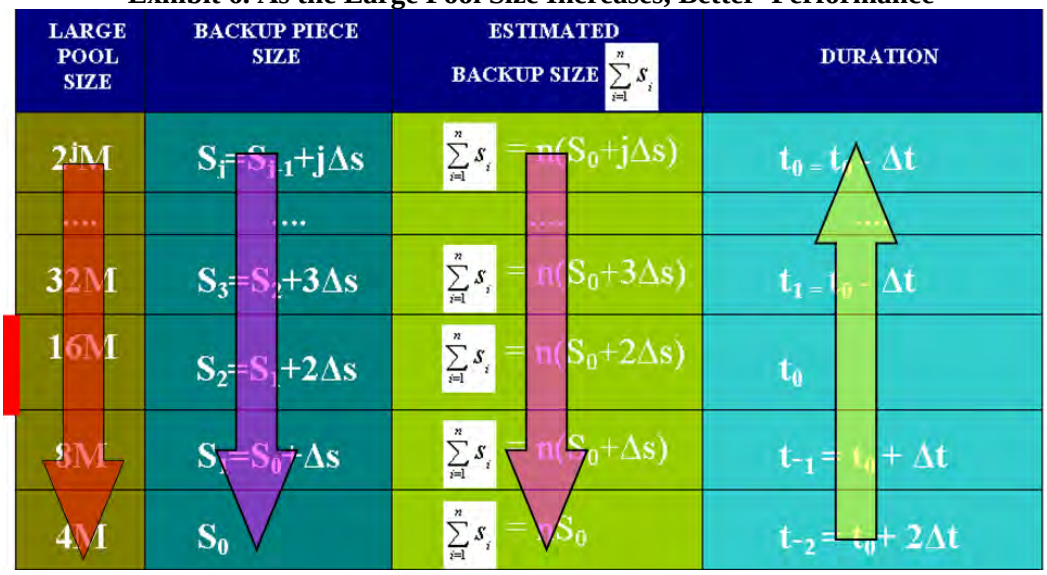


Exhibit 7. As the Large Pool Size Decreases, Improved Reliability (Mostly Third-party Products)

The results presented on the previous tables use absolute notation, in an effort to suggest the appropriate conclusions, rather than imputing results particularly associated with an average database size. Instead, these concluding observations are valid for all database sizes, the results tend to be more useful as the database size grows, or as the number of business constraints becomes significant enough to customize a backup strategy, to attain optimal reliability and performance throughput. The results taken were consistently reviewed and systematically verified for accuracy, within reasonable boundaries of the Large Pool sizing.

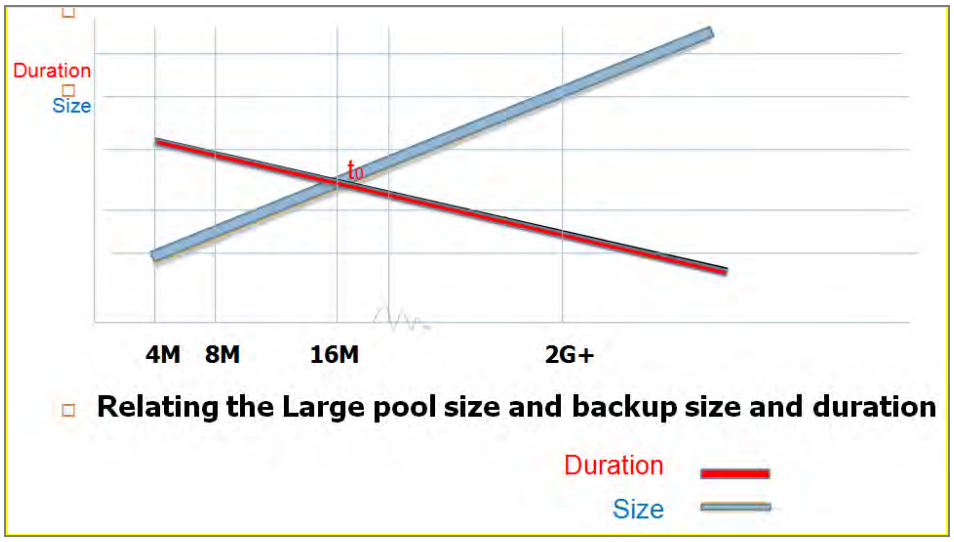


Exhibit 8. Large Pool Size and RMAN Backup Size and Duration

7. RMAN Reliability

In general, RMAN tasks are more reliable when using Oracle-recommended settings; and in most cases, with smaller settings when using third-party backup utilities. Similarly, RMAN tasks tend to increase performance when a Large Pool is set to a higher value. In general, the value of the LARGE_POOL_SIZE parameter can range from 300K to 2G or more, and it depends mostly on the operating system platform and computing resources.

8. RMAN Compatibility Table and Other Issues

Although issues such as the usage of compression or encryption could vary the absolute result of this experiment, it is predictable that the results are proportionally comparable based on previous case studies. Similarly, RMAN tasks are constrained by a compatibility table as exhibited next:

Target/Auxiliary Database	RMAN client	Recovery Catalog Database	Recovery Catalog Schema
8.0.6	8.0.6	>=8.1.7	>=8.0.6
8.1.7	8.0.6.1	>=8.1.7	>=8.1.7
8.1.7	8.1.7	>=8.1.7	>=RMAN client
8.1.7.4	8.1.7.4	>=8.1.7	8.1.7.4
8.1.7.4	8.1.7.4	>=8.1.7	>= 9.0.1.4
9.0.1	9.0.1	>=8.1.7	>= RMAN client
9.2.0	>=9.0.1.3 and <= target database executable	>=8.1.7	>= RMAN client
10.1.0	>=9.0.1.3 and <= target database executable	>=9.0.1	>= RMAN client
10.2.0	>=9.0.1.3 and <= target database executable	>=9.0.1	>= RMAN client
11.1.0	>=9.0.1.3 and <= target database executable	>=9.0.1	>= RMAN client

Exhibit 9. RMAN Compatibility Table

9. Final Analysis

The purpose of this analysis is to derive a set of rules of thumb that allows either validation of Oracle recommended settings or the establishment of custom settings for the database used. The fact that there is no LRU-approach to the Large Pool usage, DBAs need to convey a proactive monitoring to tuning the Large Pool based on backup size and task duration. Normally, appropriate sizing could derive performance optimization of about 10% in average (in terms of throughput and overall turnaround duration), or increased reliability with other vendor utilities, i.e., fewer vendor warning or error messages and consequently fewer backup tasks with abnormal terminations (ABENDs), either related to the SBT hardware or utility technology. Overall, this also translates conceptually into easier control of RMAN operations. Oracle Enterprise Management Grid Control is the recommended tool, and an alert-driven approach is extremely useful for monitoring purposes.

10. Oracle by-Version

The following tables illustrate the by-version Oracle relevancy and most important considerations when attempting to optimize the sizing of the Large Pool for reliability and performance purposes:

VERSION	LARGE POOL	REMARKS
Oracle9i	YES	Dynamic Parameter, BR disk buffers, Parallel Execution Message Buffer, No LRU, Optional. Defaults to 0. ksfqxcre internal shared memory allocation error Oracle recommended setting: 8M
		300 KB to at least 2 GB (actual maximum is operating system-specific)

Exhibit 10. Oracle9i Large Pool Highlights

VERSION	REMARKS
Oracle10g	Dynamic Parameter, BR disk buffers (Oracle allocates buffers that are a few hundred kilobytes) Parallel query, Parallel Execution Message Buffer No LRU and no aging out of pool Optional.
	Specified LARGE_POOL_SIZE value a minimum value for the memory pool (allocation heap).
	300 KB to at least 2 GB (actual maximum is operating system-specific)

Exhibit 11. Oracle10g Large Pool Highlights

VERSION	REMARKS
Oracle11g	Dynamic Parameter, BR disk buffers (Oracle allocates buffers that are a few hundred kilobytes) Parallel query, Parallel Execution Message Buffer No LRU and no aging out of pool Optional. Defaults to 0. Minimum Value 300K. 30M recommended for about 100 sessions (shared server, about 200-300k/session). For effective sort usage of LP, set sort_area_size and sort_area_retained_sime to same value.
	Practical in Automatic Memory Management.

Exhibit 12. Oracle11g Large Pool Highlights

11. Concluding Remarks

Upon completing this research experiment, the author has arrived at the following key factual correlations, namely:

- The Large Pool Size Setting could affect BR/DR/BC Processes, involving both reliability and performance.
- Large Pool Size is a factor when correlating database size/backup size and duration.
- The large pool size is directly proportional to the backup size and inversely proportional to the backup duration.

NYOUG 2009 Sponsors

The New York Oracle Users Group wishes to thank the following companies for their generous support.

Confio Software (www.confio.com)
Oracle (www.oracle.com)
Quest Software (www.quest.com)
TUSC (www.tusc.com)

Contact Sean Hull and Irina Cotler for vendor information, sponsorship, and benefits

TUSC acquires WhittmanHart Consulting
– expands into Enterprise Performance Management

ORACLE | Hyperion

With 20 years of experience delivering analytic applications that extend visibility into the enterprise and beyond, TUSC's new Enterprise Performance Management Group ensures that you articulate and seamlessly implement an end-to-end EPM strategy for your business. Leveraging a combination of Oracle EPM solutions as well as proprietary financial templates, driver-based planning models, Industry specific scorecards/dashboards, and reference architectures, we help you drive value from a financial, customer and shareholder value perspective.

The Oracle Experts

TUSC

A ROLTA COMPANY

TUSC's other Core Service Areas

Oracle's E-Business Suite
- Implementations
- Upgrades
- Remote Support

BI / Data Warehousing
- Strategy
- Implementation

Custom Development
- Fusion Middleware
- Web Services & SOA
- Application Migrations

Product Licensing
- Oracle
- Instant SOA

Managed Services
- Remote Support
  - DBA and Database
  - Oracle E-business

Database Services
- Health Checks
- Strategic Planning
- Infrastructure Services
- Full DBA Support
- Remote Support

Training
- Classroom
  - (offsite & onsite)
- Mentoring

ORACLE CERTIFIED ADVANTAGE PARTNER

GSA Contract Holder
Contract GS-35F-0149U