TechJournal

New York Oracle Users Group

Fourth Quarter 2013

Fourth Quarter General Meeting

Tuesday, December 17, 2013
St. John's University – Manhattan Campus
101 Murray Street

Sponsored by:
Dell Software, datAvail, and Axxana

Free for Paid 2013 Members
Don't Miss It!

In This Issue –

Nothing hunts down Oracle performance issues like Confio Ignite™.

Over 50% of DBAs who try Ignite resolve a performance problem on the first day.

Start your **free trial at Confio.com/p-hog**

CONFIO SOFTWARE

NYOUG Officers / Chairpersons

ELECTED OFFICERS - 2012

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
Cathy Wang-Wender
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

Chairperson / Vendor Relations
Caryl Lee Fisher
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 Emeritus

Kim Marie Mancusi

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
67 Wall Street, 22nd floor
New York, NY 10005-3198
(212) 978-8890

Winter General Meeting – December 17, 2013
Sponsored by Dell Software, datAvail, and Axxana

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: Oracle Database 12c: Engineered for Clouds and Big Data**	(single session) Auditorium	Willie Hardie Oracle Corp.
10:30-10:45	**BREAK**		
SESSION 2 10:45 -11:45	**New Database Replication and Data Integration with Hadoop and BI**	DBA Auditorium	Jeff Surretsky Dell Software
	Top 10 Oracle SQL Developer Tips and Tricks	Developer Room 118	Marc Sewtz Oracle Corp.
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	**An Introduction to New Features in Oracle 12c**	DBA Auditorium	Earl Shaffer Planet Payment
	Hitchhiker's Guide to Xquery with Oracle Database 11g & SQL Developer	Developer Room 118	Coleman Leviter IOUG
2:30-2:45	**BREAK**		
SESSION 5 2:45-3:45	**All the Leaves Aren't Brown: Many Ways to Profile your Application Code**	DBA Auditorium	Chuck Ezell datAvail
	APEX Unplugged	Developer Room 118	Dan McGhan Enkitec
SESSION 6 4:00-5:00	**Making Round the Clock Database Support Work without Losing Sleep – PANEL** Panel Members: Mike Vergato - Arrow Electronics Chuck Ezell - datAvail Chad Cleveland - datAvail Krishna Rote - datAvail	(single session) Auditorium	Keenan Phelan datAvail Moderator

ABSTRACTS

9:30-10:30 AM KEYNOTE: Oracle Database 12c: Engineered for Clouds and Big Data

Oracle Database 12c introduced a new multi-tenant architecture and a range of features that can help customers simplify database consolidation in the cloud, deliver database as a service, integrate big data into enterprise data warehouses, and derive greater efficiencies and cost savings with engineered systems. This roadmap session will discuss the latest developments from Oracle Database Server Technologies, see Oracle Database 12c in action, and get a preview of what's on the database horizon.

Willie Hardie is responsible for Oracle Database product management, focusing on growing Oracle's business in the global database, data warehousing, and embedded database markets. His areas of expertise includes Oracle Database and Oracle Exadata as well as Oracle Real Application Clusters, Oracle In-Memory Database Cache, Oracle Advanced Compression, and other key Oracle Database options. Mr. Hardie has been in IT for more than 25 years and has specialized in relational database technologies for more than 20 years. Originally from Edinburgh in Scotland, he has worked with Oracle Database since Release 5.

10:45-11:45 AM DBA TRACK: New Database Replication and Data Integration with Hadoop and BI

This presentation discusses a solution that ensures business continuity while meeting your database operational goals. It provides a real-time copy of production data without impacting your OLTP system's performance and availability. Attendees will learn how to:
Enable near real-time log-based replication of Oracle tables to Hadoop HDFS, HBase, and Hive environments
Achieve database operational goals without affecting system performance and business continuity
Improve decision making with fast, simplified, up-to-date data integration
This session provides insight on how you can improve the process, with continuous replication of changes from an Oracle database to a Hadoop cluster, maintaining real-time or near real-time images of the source tables for all your data analytics needs. In this insight-packed session, you'll discover how to simplify your work with impact-free Oracle database to Hadoop replication and near real-time data integration.

Jeffrey Surretsky is a Database Systems Consultant for Dell Software (formerly Quest Software) for over 13 years where he has focused on Database Performance, Optimization, Monitoring and Replication. Previous to that, he was a Database Administrator after having held other IT roles. Jeffrey graduated from Rutgers with a Bachelor's Degree in Computer and Science and a Master's Degree in Management of Information Systems from Fairleigh Dickinson University.

10:45-11:45 AM DEVELOPER Track: Top 10 Oracle SQL Developer Tips & Tricks

More than two and a half million people use Oracle SQL Developer, but how many of them are really getting the most out of the tool? Oracle SQL Developer's primary purpose is to save developers and DBAs time and energy without getting in their way. In this interactive session, see the most popular features and productivity tips that the SQL Developer Product Management has accumulated. All attendees are guaranteed to pick up two or three new techniques that will improve their Oracle Database experience.

Marc Sewtz is a Senior Software Development Manager for Oracle Application Express (APEX) in the Database Tools Group. Marc has 18 years of industry experience, including roles in Consulting, Sales, and Product Development. Marc joined Oracle in 1998, works in New York City and manages a global team of APEX developers and product managers. Marc and his team are responsible for product features such as Mobile Development, User Interface Design, Oracle Forms conversion, Reporting, Tabular Forms, PDF printing and the integration with BI

Publisher. Marc has a Master's degree in Computer Science from the University of Applied Sciences in Wedel, Germany**.**

1:30-2:30 PM DBA TRACK: An Introduction to New Features in Oracle 12c

As a former Oracle employee, Earl was able to learn about Oracle DB 12 starting in 2012. He attended a formal week-long class, as well as over 10 seminars, webinars, and briefings. In that time, Earl learned about the new features, improved features, and the ground-breaking new multi-tenant architecture and how that will change Oracle DBA work going forward. This presentation will cover 12c new features in the areas of DBA work, RMAN, Data Guard, RAC, ASM and more

Earl Shaffer is a 20+ year veteran of Oracle technology. He started with Oracle v5.1 in 1988. He has worked with, and upgraded to versions 6.0, 7, 8, 8i, 9i, 10g, 11g, and now 12c.Earl has presented papers at local, regional, national, and international conferences. He has also been in the executive team of regional and local Oracle User Groups.

1:30-2:30 PM DEVELOPER TRACK: Hitchhiker's Guide to Xquery with Oracle Database 11g and SQL Developer

XQuery has been around for a few short years. In that time, it has grown to become the standard for querying and shredding XML documents. Embedded in its syntax comes strange notation that usually presents an obstacle to first-time users of XQuery. Additionally, many of its features that are important to use may be overlooked. This session presents the fundamentals of XQuery using basic examples of XMLQUERY and XMLTABLE. Demonstrations using Oracle 11g and Oracle SQL Developer will be used to reinforce the session examples.

Coleman Leviter, OCP is employed as an IT Software Systems Engineer at Arrow Electronics. He has presented at IOUG's Collaborate and at Oracle Open World. He is the NYOUG Web SIG chairperson on the NYOUG Steering Committee. His articles have been published in Select Journal, IOUG Tips and Best Practices, and the OTDUG Journal. He has worked in the financial services industry and the aerospace industry where he developed Navigation, Flight Control and Reconnaissance software for the F-14D Tomcat. Additionally, Coleman is on the Board of Directors at the ioug.org. He may be contacted at cleviter@ieee.org

2:45-3:45 PM DBA TRACK: All the Leaves Aren't Brown – Many Ways to Profile Your Application Code

Finding improvements in your Client Side JavaScript, Java or .Net code is often easier said than done. Managing application performance problems due to memory leaks, poor coding habits, superfluous library usage, detecting deadlocks, measuring memory usage, and garbage collection frequency are activities often left to the DBA and System Administrators. Do we concern ourselves with the stability of the JVM on a user's computer or just the application server? Does it really make a difference what web browser the user chose to use that day? How do we know how the difference between a locked thread and a locked database object even though they seem to look the same from the users perspective? Attendees should expect to leave this presentation with an overview and introduction to several different methods of profiling application code. Chuck will cover free utilities and off-the- shelf products that are easily implemented as well as demonstrate the pros and cons of each. The session will end with a discussion and Q/A on best approaches that work for different tools in different environments.

With over 18 years of experience, **Chuck Ezell** supports some of the world's largest retailers with database performance tuning for their internal applications. Chuck helps Datavail locate and eliminate bottlenecks in database administration, while uncovering opportunities to improve performance. He excels at optimizing and customizing EBS systems. Chuck works with Java, SQL, .NET, and other languages on systems by Oracle, HP, IBM, among others. He uses AppDynamics, Splunk and Visual VM, along with other tools, for database tuning operations.

2:45-3:45PM DEVELOPER TRACK: APEX Unplugged

In Oracle Application Express 4.2, new features give developers the ability to easily create mobile Web applications that look like native mobile applications. But with new capabilities come new challenges. One such challenge with which developers must contend is that mobile devices often lose their Internet connections where signals are weak. Is there anything developers can do to prevent the loss of Internet from rendering mobile application built with Oracle Application Express useless? The short answer is yes thanks to several new technologies that have emerged to tackle this common problem. This session will discuss these exciting new technologies and how they can be utilized in Oracle Application Express applications.

Dan McGhan is Senior Technical Consultant at Enkitec who recently moved to NYC. He is an Oracle Application Express Certified Expert, an Oracle PL/SQL Developer Certified Associate, as well as an Oracle ACE. In addition to his "day job", he is one of the top contributors to the APEX forum, maintains his own Oracle and APEX blog at danielmcghan.us, and is a regular presenter at various events and user group meetings including Oracle OpenWorld and ODTUG's Kscope & APEXposed.

4:00-5:00PM Database Discussion Panel: Making Round-the-Clock Database Support Work Without Losing Sleep - PANEL DISCUSSION

This session brings together three Oracle Database Administrators from a Fortune 500 company who team together to manage the databases 24x7 for a data-intensive company. The team will discuss the best practices, communications and processes they follow to have a truly seamless, “follow the sun” support of major databases, enhancing the uptime, availability and performance of the databases. They will share examples of how they are able to more effectively and efficiently support their company’s databases, using both planned and ad-hoc communications, knowledge sharing, escalation and education processes, and reporting tools.

Keenan Phelan is Executive Vice President of Global Services at datAvail, an IT leader in database administration as a managed service. Prior to joining datAvail, Keenan led the Operations and Technical Presales team at CIBER-ITO and has held a number of Vice President and Director-level technical, sales and operations roles with IT consulting firms with a focus on the banking and financial sector. Keenan is an accomplished leader, has authored dozens of technical and organizational analysis documents and routinely presents at major conferences.

PANEL MEMBERS:
Mike Vergato - Director Oracle Architecture, Oracle Integration Oracle Database Administration, Arrow
Chuck Ezell, Oracle DBA and Applications Tuner, Datavail
Chad Cleveland, Oracle Database Administrator, Datavail
Krishna Rote, Senior DBA Tier 3 – Datavail (by video in India)

Message from the President's Desk

Michael Olin

Winter, 2013

You Have Been Nominated

Once again, in late November, I started received emails from both the Independent Oracle User Group (IOUG) and Oracle inviting me to a very special meeting. The first message, from Stacey Freeh, the IOUG's Membership and Volunteer Engagement Coordinator, informed me that I would shortly be hearing from Oracle. The message began:

> Dear IOUG Leader,
>
> You have been recognized for outstanding leadership within your user group and IOUG has nominated you to attend the 2014 Oracle User Group Leaders' Summit. The International Oracle User Group Community (IOUC) is a community of leaders representing Oracle user groups worldwide. The leaders of independent communities focused on Oracle products and technology meet in-person at IOUC every year to share ideas about fostering community growth, establish best practices and to learn more about current and upcoming Oracle products.

About a week and a half later, the "official" invitation arrived from May Lou Dopart, a Senior Director in Oracle's Global Customer Programs office. Ms. Dopart is the Oracle executive who serves as liaison between Oracle and the worldwide user group community. Ms. Dopart's invitation explained my "nomination" as follows:

> Michael,
>
> I would like to invite you to attend the 2014 International Oracle User Group Community (IOUC) Summit, scheduled to take place Tuesday, January 21, 2014 through Thursday, January 23, 2014 at Oracle Headquarters in Redwood Shores, CA. You have been nominated to attend the Summit because of your role with your Oracle user group.

Both messages included information about the agenda of the meeting and this very important logistical detail:

> Please be aware that all travel, accommodations and additional expenses are the responsibility of the attendee.

It seems that my leadership role in the Oracle user community provided me with the opportunity to travel to California, on my own time and at my own expense, to meet with other user group leaders who are also paying out of their own pockets, to do the following:

> …share ideas about fostering community growth, establish best practices and to learn more about current and upcoming Oracle products

From my perspective, the only thing missing from this "nomination" is the opportunity to purchase a bound copy of a book listing all of the nominated luminaries from the user group community. I am sure that most of those invited would be happy to spend a few hundred dollars more to get their copy of "Who's Who in the Oracle User Community."

I did not reply to either message, which resulted in reminder messages from both the IOUG and Oracle, arriving back-to-back in mid-December. I am also sure that a "last chance" message will arrive just after New Year's Day. If I were to reply, I suppose it would look something like this:

An Open Letter to the Organizers of the Oracle User Group Leaders' Summit

I am writing to thank you for your invitation to participate in the 2014 Oracle User Group Leaders' Summit. As President of NYOUG, one of the largest and most active Regional User Groups (RUG) in the Oracle User Community, I suppose that I should really give your invitation some consideration. Nevertheless, as I have since the inception of this event, I must once again decline. Although I do appreciate the fact that you have made changes to the format this year, reducing the length of the event to "two days to decrease leader time out of office," I cannot help feeling that this event is more about getting Oracle's message across rather than helping the user groups. Of course, having never attended, I could be completely wrong about the conference's focus. However, I hope you can see how agenda items like this may give me that impression:

> Oracle Campaigns
> User group leaders will learn how to leverage Oracle messages and understand the priority campaigns that will be delivered in Q3-4 FY14. This will help the groups synch with Oracle on content delivery!

I wonder if the people who plan this "Summit" have any idea what attendance at such a meeting looks like to a true volunteer leader in the user community. I've been involved with NYOUG since its founding in the mid 1980's. That is long before there was an IOUG (whether the "I" meant "International," as it did originally, or "Independent," the fig leaf that is currently employed), before Oracle had an executive responsible for managing interaction with user groups, and well before Oracle took over half of San Francisco so that some 60,000 people could attend Oracle OpenWorld. In all of those years, I have presented papers at several "International Oracle User Week" conferences, presented and run full-day sessions at 10 "East Coast Oracle (ECO)" conferences, attended and/or run well over 100 NYOUG General Meetings, planned NYOUG training days, and spent countless hours on conference calls and email related to managing a local users group. I have paid my own way for all of this, paid for my own IOUG membership and, while I don't have to use my vacation days to cover the days that I am out of the office volunteering as a user group leader, the reason is because as an independent consultant, I don't have any "vacation days" to use. If I am at a user group event, there is no client to bill for the time and I don't get paid. The truth is that I am hardly alone in these circumstances. I know dozens of user group leaders throughout the US who have been doing the same thing that I have, many of them for just as many years.

It's not that I don't think meeting with other user group leaders is worth my time. I speak with leaders of other RUGs frequently and I think that there is quite a bit they could learn from what we do at NYOUG and just as much that we could learn from them. I'd probably be willing to give up a day or two of billing to meet with them and even sit through a couple of hours of Oracle product managers giving us their talking points for the coming year. However, asking me to foot the bill for my own travel, meals and accommodations at the (luxurious, I'm sure) Embassy Suites Hotel in Burlingame is - well frankly - insulting. I know how much it costs to run a two-day conference in New York City, and I'm sure that things are just as expensive in the Bay Area. I'm willing to bet that picking up all of the expenses for the user group leaders you invite to this event would amount to little more than a rounding error compared with what Oracle spends on Open World. Perhaps having one less band at the "Appreciation Event" would cover the costs. Larry Ellison could probably personally fund the User Group Leaders Summit for years with a contribution of perhaps 10% of the next quarterly dividend payment on his Oracle holdings (after taxes). The user group community has always been supportive of Oracle and dedicated to helping Oracle users in our local areas be successful in their use of the company's products. If you truly believe that Oracle benefits from having a robust, independent, user group community, and you think that this Summit will help keep that community strong, it's time to stop asking those volunteers who give so much of themselves to their local communities to pick up the tab.

Michael Olin
President, NYOUG

ADF On-Ramp: What You Need to Know To Use ADF

Peter Koletzke, Quovera

> *Si nous ne trouvons pas des choses agréables,*
> *nous trouverons du moins des choses nouvelles.*
>
> (If we do not find anything pleasant,
> at least we shall find something new.)
>
> —Voltaire (1694-1778), *Candide* (Ch. xvii)

Developing a Java-oriented web application these days is an experience that many Oracle technologists find to be new but not necessarily very pleasant. Architecting such an application requires selecting a set of technologies from a dauntingly-large and ever-growing list. Up to now, the responsibility for combining these technologies and ensuring that they communicate and work together in an orderly way has been left up to each organization. The path of selecting and working with different Java-oriented frameworks can inevitably lead to wrong turns, especially for those who are new to the Java world. Depending upon when those wrong turns occur, the effect on the project can range from mild to devastating and will likely require rewriting some or most of the application.
Fortunately, Oracle has now provided guidance in the form of the set of technologies they have selected to build Fusion Applications—the new application products (parallel to E-Business Suite). Oracle has chosen open-standards technologies in the Java realm so parts of the application can be easily extended with little reliance on a specific vendor's product line, hardware set, or operating system. (Many other reasons for selecting open standard technologies—such as customer preferences—exist, but are a larger discussion that is not critical to the focus of this white paper.)

Fusion Technology Stack

Fusion developers within Oracle have been creating Fusion Applications using Application Development Framework (ADF) in JDeveloper with the following core technologies:

- ADF Business Components (ADF BC)
- ADF Faces Rich Client (ADF Faces RC)
- ADF Bindings and ADF Data Controls
- ADF Controller

In addition to those core technologies, Oracle uses high-level technologies or strategies such as the following to coordinate Fusion Applications' components and to fulfill additional architectural requirements:

- Service Oriented Architecture (SOA) with Business Process Execution Language (BPEL)
- Enterprise Service Bus (ESB)
- Oracle Business Rules
- Oracle WebCenter

Since Oracle is using Application Development Framework (ADF)—a facility in JDeveloper for working with code in a common way—to create Fusion Applications, you can use the term "ADF Fusion Technology Stack" to refer to all technologies in the core and high-level lists. Packaged application software is a large part of Oracle's business, and Oracle has a very compelling business reason to ensure that the technologies used in Fusion Applications will integrate properly and work successfully. Therefore, you can be relatively assured that you, too, can be successful in creating applications with the same technologies.

Retooling for Fusion Technology Work

Determining the list of technologies that an application will use is not enough. Planning for any application development effort must also include tasks and strategies for bringing current development staff up to speed on the techniques required for the new environment. If you determine that your current development staff cannot reach acceptable skill levels in the available time, you may need to employ additional resources. You will need to understand what tools, development techniques, and languages a developer needs to learn (for current staff) or to know (for additional resources) to be productive in the ADF Fusion Technology environment.

The main objective of this white paper is to explain just that—what developers need to know to be productive writing applications using the ADF Fusion Technology Stack. If you think of Oracle internal developers as drivers already speeding along on the Fusion Development Highway, this white paper is the on-ramp for others who are not yet on that road but who need to be there. To extend that analogy a bit, while the exit for Oracle developers is "Oracle Fusion Applications Production" and yours will be different, all have the same vehicle—ADF in JDeveloper—and type of fuel—the Fusion Technology Stack.

This white paper starts by explaining some preliminary concepts; then it explains and shows the kinds of code and techniques needed for productive work in ADF with the core technologies in the Fusion Technology Stack. The goal is to explain the main development techniques for only the core technology set. The high-level technologies are more strategic systems that an enterprise architect will select for a particular application. While heads-down developers may need to know about techniques for the high-level technologies, that type of work will vary depending upon architectural decisions and on the enterprise's environment. In fact, the core technology stack may suffice for some applications so no high-level technologies would be needed at all. The white paper closes by discussing the languages developers use for this type of work.

> **Note**: Although this white paper does not specifically address techniques required to extend Fusion Applications, if you currently develop or maintain custom extensions to Oracle E-Business Suite (Oracle Applications) or think you will find yourself doing so in the future, you will be using the same techniques and technologies discussed in this white paper for that work in Fusion Applications.

What is Fusion?

The word "Fusion" is used these days to refer to almost everything from cars to food to razors to drinks. Therefore, the first concept to understand is what Oracle means when they use that word. The word "Fusion" is used in various ways within the Oracle product line, but it generally refers to a strategic reorganization of Oracle products. Oracle invented Oracle Fusion after acquiring various companies who offered their own application products. Oracle's objective with Fusion is to merge the best of all those products into a single (fused) applications suite. This effort will take many years, but Oracle has started this work and we expect to see the premier version of these application products in the near future. You can summarize Oracle Fusion with the uses in the following terms:

- **Oracle Fusion Applications**, mentioned before, is the next version of Oracle E-Business Suite.
- **Oracle Fusion Middleware** is the toolset that Oracle is using to develop and deploy Fusion Applications. This toolset consists of virtually all Oracle development and runtime products (except for the Oracle database and Oracle packaged applications) such as JDeveloper and Oracle WebLogic Server (but not legacy tools such as Oracle Forms and Reports).
- **Oracle Fusion Architecture** outlines the way various technologies are used to build the applications. This Fusion usage is not as frequently used or seen as Oracle Fusion Middleware and Oracle Fusion Applications.

JDeveloper and ADF

JDeveloper 11*g* is the Fusion Middleware development tool. It is the common tool used for developing all types of code, regardless of the technology. Moreover, as mentioned, JDeveloper is the container for ADF. Therefore, Oracle is very focused on enabling JDeveloper 11*g* to support all requirements of the new Fusion Applications.

Although this white paper focusses on JDeveloper as the main tool to use for ADF, Oracle has published *ADF Essentials*, a package of no-license-fee ADF technologies that you can also plug into Eclipse through the Oracle Enterprise Pack for Eclipse. Instead of WebLogic Server, the public domain server Glassfish is used for deployment and runtime of ADF Essentials. More information is available on the Oracle ADF web pages with a good starting point being the FAQs: www.oracle.com/technetwork/developer-tools/adf/overview/adfessentialsfaq-1837249.pdf.

What is ADF?

To answer that question, you need to know that the word "framework" in the Java world refers to an application development technology. A framework is like an Application Programming Interface (API) or a code library in other disciplines: all offer generically built code that you can use in your application. The code that implements the framework supplies an entire service that you can access using a certain development method and calling interface. Although APIs and code libraries may have these characteristics, frameworks are built around the idea of a service. For example, instead of building from scratch some key facility such as a connection layer to the database, you use an existing framework such as ADF BC to supply that service.

One reason to use a framework is to tap into a standard way of supplying the functionality of the service to your application. You do not need to invent a service that you need for a piece of your application. Another related reason is that you do not need to redevelop code that many applications share. When using a framework, you leverage solid and (hopefully) well-debugged code in all your applications. In addition, the most popular frameworks offer solid support at least from the user community, if not from a vendor. The sidebar "Working with Java Frameworks" describes how you use frameworks in your application code.

Working with Java Frameworks

Framework code in the Java world usually consists of prebuilt Java classes. Those classes offer complete functionality for a service (like database access). They are set up to read configuration or application-specific definitions coded inside an Extensible Markup Language (XML) file. Therefore, the primary code you are responsible for when using a framework is XML-based. A good framework offers enough flexibility to handle most applications with this type of work. Moreover, developers using frameworks are most effective when they understand what the framework can accomplish so they can design their application code to fully leverage the framework.

Occasionally (and if you are using frameworks properly, it should only be occasionally), a developer will need to replace or add to a part of the service that cannot fulfill an application's requirement. In this case, the developer subclasses one or more framework classes and adds some code to customize the framework's behavior. This type of work requires intermediate-level knowledge of Java as well as a deep knowledge of the framework. Therefore, it is a technique to be used sparingly.

So ADF is…

Application Development Framework (ADF) is an architectural strategy within JDeveloper that allows you to build applications using common declarative and visual methods. For example, you can build database access code into your application using Enterprise JavaBeans (EJBs), ADF Business Components (ADF BC), or web services (among others). The code details and libraries that support these frameworks are different, but the actions you use in JDeveloper to create user interfaces based on these frameworks are the same. ADF, therefore, is really a meta-framework that integrates and offers common development methods to many other frameworks.

ADF Architecture

The ADF architecture model, depicted in Figure 1, divides the frameworks it supports into various code layers that loosely follow the Java EE design pattern Model-View-Controller (MVC). *MVC* defines three main layers of application code: *Model*—to manage the data portion of the application, *View*—to handle drawing the user interface screen, and *Controller*—to process user interface events (such as button clicks) and to control page flow (how one page is called from another page).

The ADF architecture layers follow the definition of MVC for the most part, but ADF adds another layer, *ADF Business Services*, a spin off from the Model layer. ADF Business Services provides code for accessing data sources such as a database. Business services are responsible for *persistence*—the physical storage of data for future retrieval—and *object-relational (OR) mapping*—translating storage units such as rows and columns in relational database tables to object-oriented structures such as arrays of objects with property values. ADF Business Components is a core Fusion technology in this layer.

The *ADF View* layer corresponds directly to the MVC View layer. It includes technologies that you use to draw the user interface. In the case of *web client code*—application code that is run in a Java runtime on an application server rather than locally on the desktop (as is *application client code*)—ADF View supports JavaServer Faces (JSF) and ADF Faces RC, core Fusion technologies.

The *ADF Controller* layer, which defines separate frameworks only for web client code, supports popular JSF and Struts controller frameworks. In addition, it adds an ADF-specific framework—ADF Controller ("ADF Task Flow Controller")—that allows you to create and control parts of a page. ADF Controller is a core Fusion technology in this layer.

The *ADF Model* layer corresponds to part of the MVC Model layer but specifically represents the connection mechanism from the Business Services layer to the View layer (through the Controller layer). The ADF Model layer is composed of the following two aspects:

- **ADF Bindings** This framework (really just an aspect of ADF Model) provides a standard way to access data values in the ADF Business Services layer from an ADF View user interface component such as a pulldown item. For example, if you defined a business service item to query the DEPARTMENTS table, you could add an expression to the *Value* attribute of a text input item referring to the DEPARTMENT_ID column of the query. When the screen is drawn, the data would automatically flow from the ADF Business Services object to the text item in the View layer by using ADF Bindings.
- **ADF Data Controls** This aspect of ADF Model supplies a list of prebound components based on the data model (data sources) defined in the ADF Business Services layer. For example, in JDeveloper, you could drag and drop a node from the Data Controls panel that represents the DEPARTMENTS query onto a JSF page. The IDE will determine the type of business service (in this case a *collection*—multiple rows and multiple columns) and will present a selection menu of different styles of display components (for example, forms, tables, trees, or navigation buttons). Selecting one of those options causes JDeveloper to lay out the appropriate display on the screen and bind the items on the screen to the business service.

Using both of those aspects, you do not need to write code to present data (for query and also for insert, update, and delete operations) in the user interface. Although no Java EE standard exists yet for bindings and data controls, Oracle and other parties were working on a *Java Specification Request* (JSR, the process by which a new feature or revision is made to the Java platform) to include this mechanism in the Java standards. (This JSR was removed in May 2011 but you can review its history by searching at jcp.org for JSR-227.)

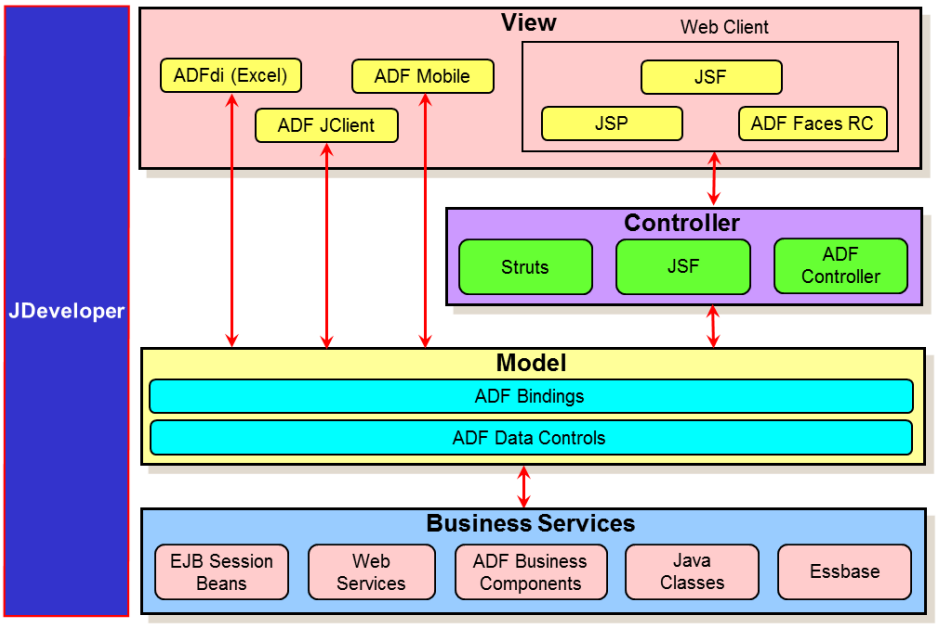


Figure 1. ADF Architecture Model

You will also notice in Figure 1 that JDeveloper sits to the side of the ADF framework layers because it is the tool you use to manipulate all ADF technologies.

> *Dans ce meilleur des mondes possibles ...*
> *tout est au mieux.*
> (In this best of all possible worlds ...
> everything is for the best.)
> —Voltaire (1694-1778), *Candide* (Ch. i)

Core ADF Fusion Technologies

The easiest way to describe the core ADF Fusion technologies is in the context of a working application. Although the ADF frameworks have many advanced features, the purpose of this white paper (to understand what you need to know) will be best served by looking at a simple application (shown in Figure 2) that provides the following basic data handling functions:

1. Querying the DEPARTMENTS table in read-only mode when the page opens.
2. Querying EMPLOYEES table records that are related to the displayed DEPARTMENTS record.
3. Navigating between DEPARTMENTS table records using **First**, **Previous**, **Next**, and **Last** buttons.
4. Editing the displayed DEPARTMENTS table using a separate page accessed with the **Edit Department** button.
5. Creating a DEPARTMENTS record using the edit page in Create mode accessed with the **New Department** button.

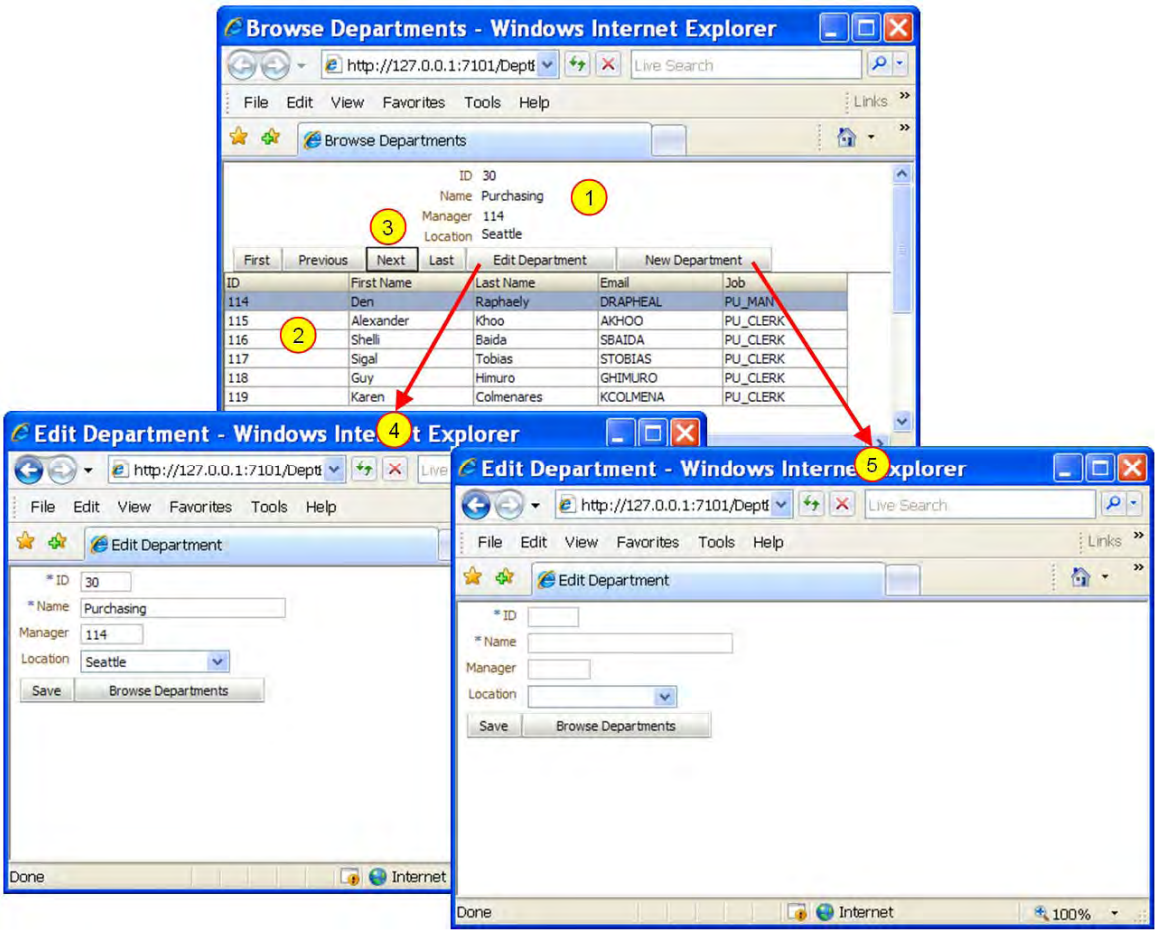


Figure 2. Sample Application Containing Basic Data Handling Functions

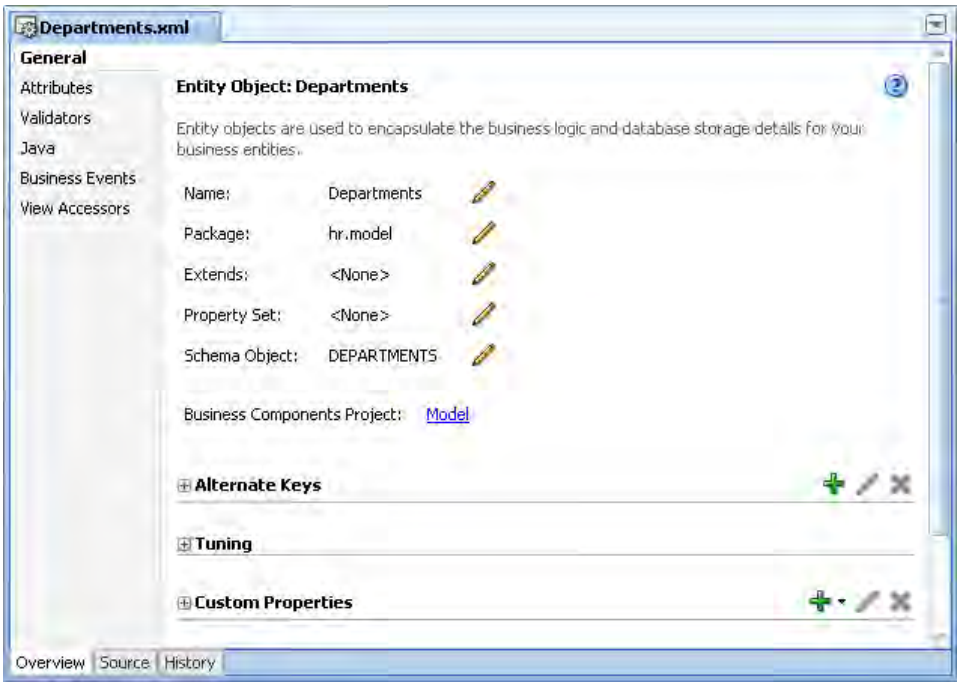
This application uses basic examples of these core Fusion technologies:

- **ADF Business Components** for ADF Business Services layer functions that access the database.
- **ADF Faces Rich Client** for ADF View layer functions that render the user interface in the web browser
- **ADF Bindings and ADF Data Controls** for ADF Model layer functions that connect database data to components on the web page
- **ADF Controller** for Controller layer functions that manage page flow and handle user event interactions

Let's see where those technologies are used in this sample application.

ADF Business Components

This application queries and updates data in an Oracle database. ADF Business Components (ADF BC) is the framework from the ADF Business Services layer used to perform the database-specific operations. For example, a representation of the DEPARTMENTS table is defined in an ADF BC *entity object*. You work with the entity object code in a declarative way. When you create an entity object, you follow a set of wizard pages. To change the entity object you would interact with a property editor such as the following for the Departments entity object:



Entity objects contain *attributes* that represent columns in the database table or view. The Attributes tab in the Entity Object Editor just shown allows you to modify the details about a specific entity attribute. Figure 3 shows an example of that screen.

Each attribute defines a Java field (for example, DepartmentId with a Java type of Number and a SQL type of NUMBER(4,0)) that ADF BC will use to prepare INSERT, UPDATE, and DELETE statements based on instructions issued through the user interface. These SQL statements are then passed to the database through Java Database Connectivity (JDBC) communication paths. All of the code that handles the JDBC calls as well as the code to create the SQL statements are provided by ADF BC. All you need do is declare at which table and columns the ADF BC framework should target.

The entity object wizard pages and property editor screens create XML code that is read by the framework files. The following code listing is a snippet from Departments.xml, the entity object definition file for the DEPARTMENTS table:

```
<Entity
  xmlns="http://xmlns.oracle.com/bc4j"
  Name="Departments"
  Version="11.1.1.53.41"
  DBObjectType="table"
  DBObjectName="DEPARTMENTS"
  AliasName="Departments"
  BindingStyle="OracleName"
  UseGlueCode="false">
  <DesignTime>
    <Attr Name="_codeGenFlag2" Value="Access"/>
    <AttrArray Name="_publishEvents"/>
  </DesignTime>
  <Attribute
    Name="DepartmentId"
    IsNotNull="true"
    Precision="4"
    Scale="0"
    ColumnName="DEPARTMENT_ID"
```

```
    SQLType="NUMERIC"
    Type="oracle.jbo.domain.Number"
    ColumnType="NUMBER"
    TableName="DEPARTMENTS"
    PrimaryKey="true">
    <DesignTime>
      <Attr Name="_DisplaySize" Value="22"/>
    </DesignTime>
    <Properties>
      <SchemaBasedProperties>
        <LABEL
          ResId="hr.model.Departments.DepartmentId_LABEL"/>
      </SchemaBasedProperties>
    </Properties>
  </Attribute>
```

This snippet shows how the entity object is declared and associated with the DEPARTMENTS table; it also sets up the DepartmentId attribute based on the DEPARTMENT_ID column. Similar definitions appear for other attributes in the entity object. When you change the entity object properties, the XML code is modified appropriately. Therefore, you do not need to modify (or even look at) entity object XML code.

Note: This declarative style of programming is found throughout work in JDeveloper and is a core strength of ADF.

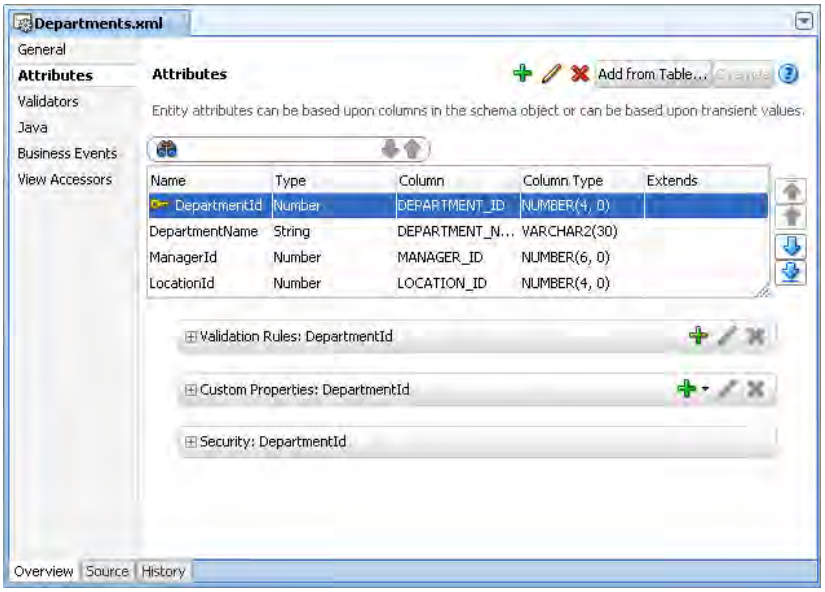


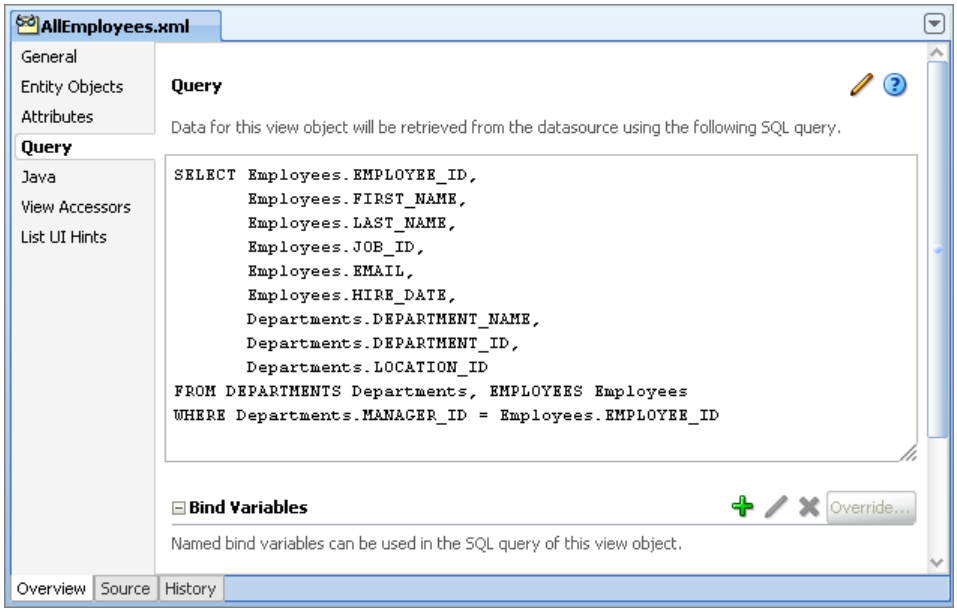
Figure 3. Attributes Page of the Entity Object Editor

Just as entity objects supply INSERT, UPDATE, and DELETE operations, *view objects* represent SELECT statements. View objects can be based on one or more entity objects, which then supply details about the table and columns, or on SELECT statements. You create and edit view objects in the same declarative way as entity objects. An XML code snippet for a view object follows.

```
<ViewObject
  xmlns="http://xmlns.oracle.com/bc4j"
  Name="AllEmployees"
```

```
  Version="11.1.1.53.41"
  SelectList="Employees.EMPLOYEE_ID,
       Employees.FIRST_NAME,
       Employees.LAST_NAME,
       Employees.JOB_ID,
       Employees.EMAIL,
       Employees.HIRE_DATE,
       Departments.DEPARTMENT_NAME,
       Departments.DEPARTMENT_ID,
       Departments.LOCATION_ID"
  FromList="DEPARTMENTS Departments, EMPLOYEES Employees"
  Where="Departments.MANAGER_ID = Employees.EMPLOYEE_ID"
  BindingStyle="OracleName"
  CustomQuery="false"
  PageIterMode="Full"
  UseGlueCode="false">
  ...
   <ViewAttribute
    Name="EmployeeId"
    IsUpdateable="false"
    IsNotNull="true"
    PrecisionRule="true"
    EntityAttrName="EmployeeId"
    EntityUsage="Employees"
    AliasName="EMPLOYEE_ID"/>
```

This view object is based on two entity objects, Employees and Departments; in the view object's XML you will find clauses used to construct a SELECT statement from those two tables. You can also read this query more directly in the view object editor as shown here:



The Bind Variables section of the editor just shown allows you to create variables that you work into the query so you can filter rows by values supplied by the application or by the user.
You can also create *view links* that represent foreign key constraints, master-detail relationships, or other logical attribute pairs that relate one view object to another. In the sample application, a view link is defined between the

DepartmentsView and EmployeesView view objects so when a department record is displayed, the employees for that department will be displayed. ADF BC automatically handles the master-detail synchronization if you define a view link.

ADF Controller

The JavaServer Faces standard of the Java Enterprise Edition platform specifications defines Controller functionality, which manages page flow (which page is loaded) as well handling user events (for example, by passing data from the Model layer to the View layer). ADF supplements the standard JSF Controller with the ADF Controller framework (also called "ADF Task Flow Controller"), which adds the ability to handle *page fragments* (parts of pages).
This ability has the following advantages over the standard JSF Controller:

- Page fragment processing can be faster (because fewer components are rerendered)
- Fragments can be reused more easily than full pages
- Additional functions or logic can be added into the flow between pages
- Flows between pages can be reused in different parts of the application.

The sample application does not specifically demonstrate page fragments; instead, as a simpler example, it shows a more standard set of two full pages: browse and edit. Navigating from one to the other is handled by the Controller as is the activity triggered by button clicks—for example, the **Next** and **Previous** buttons. Defining page flow is easiest using the diagrammer shown in Figure 4. You first create an ADF Controller file, and then drop View (page) and Control Flow Case (flow) components onto it. You then name all objects so you can refer to them in code later on. As with ADF BC, when you interact with the diagram editor, JDeveloper creates XML code such as the following:

```
<task-flow-definition id="dept-flow">
  <default-activity>deptBrowse</default-activity>
  <view id="deptBrowse">
    <page>/deptBrowse.jspx</page>
  </view>
  <view id="deptEdit">
    <page>/deptEdit.jspx</page>
  </view>
  <control-flow-rule>
    <from-activity-id>deptBrowse</from-activity-id>
    <control-flow-case>
      <from-outcome>toEdit</from-outcome>
      <to-activity-id>deptEdit</to-activity-id>
    </control-flow-case>
  </control-flow-rule>
  <control-flow-rule>
    <from-activity-id>deptEdit</from-activity-id>
    <control-flow-case>
      <from-outcome>toBrowse</from-outcome>
      <to-activity-id>deptBrowse</to-activity-id>
    </control-flow-case>
  </control-flow-rule>
</task-flow-definition>
```

After you set up a JSF page file, you can drop components such as buttons into the page. The button component's *Action* property can refer directly to the name of the control flow case. For example, the sample application's **Edit Department** button is defined in the JSF page using the following code:

```
<af:commandButton text="Edit Department" id="cb2" action="toEdit"/>
```

When the user clicks this button, the Controller finds the definition of the toEdit action in the task flow file. This code (listed earlier) declares that the flow toEdit defined in the from-outcome tag will load the deptEdit activity (in this case, a JSF page). The **Browse Departments** button on the edit page reverses this navigation using the toBrowse flow.

> **Note**: With ADF Controller, as well as with ADF BC, you can always write Java code to supplement or replace functionality. However, the more functionality you can define declaratively, the more you will be using the power of these frameworks.

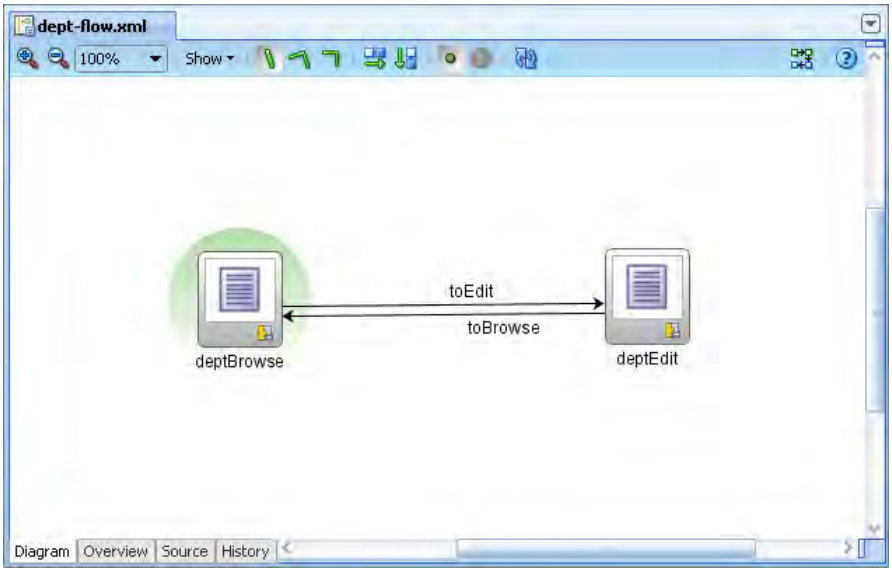


Figure 4. Task Flow Diagram

ADF Faces Rich Client

The ADF View layer constructs the user interface. In the case of a web application, the user interface is rendered in a Hypertext Markup Language (HTML) browser. Native HTML items such as text input items, buttons, selection lists, and radio buttons are limited in functionality. JSF defines higher-level items (called "components") that add functionality to HTML. ADF Faces Rich Client (available in JDeveloper 11*g* and abbreviated hereafter as "ADF Faces") is a set of JSF components with "rich" functionality. For example, ADF Faces offers a component called **`af:table`** (ADF Faces components are prefixed with "af" denoting the tag library in which they are found) that represents an HTML table in a web browser. Combining **`af:table`** with one or more **`af:column`** components allows you to define an entire HTML table without writing HTML. Here is a snippet of code for the Employees read-only table in the sample application:

```
<af:table value="#{bindings.EmployeesView3.collectionModel}" var="row"
          rows="#{bindings.EmployeesView3.rangeSize}"
          emptyText="#{bindings.EmployeesView3.viewable ? 'No data to display.' : 'Access
Denied.'}"
          fetchSize="#{bindings.EmployeesView3.rangeSize}"
          rowBandingInterval="0"
          selectedRowKeys="#{bindings.EmployeesView3.collectionModel.selectedRow}"
          selectionListener="#{bindings.EmployeesView3.collectionModel.makeCurrent}"
          rowSelection="single" id="t1" inlineStyle="width:100.0%;">
  <af:column sortProperty="EmployeeId" sortable="true"
             headerText="#{bindings.EmployeesView3.hints.EmployeeId.label}"
             id="c2">
    <af:outputText value="#{row.EmployeeId}" id="ot6">
```

```
  </af:column>
  <af:column sortProperty="FirstName" sortable="true"
             headerText="#{bindings.EmployeesView3.hints.FirstName.label}"
             id="c5">
    <af:outputText value="#{row.FirstName}" id="ot7"/>
  </af:column>
...
</af:table>
```

Notice that, like the Business Services and Controller layer code, ADF Faces is also XML code consisting of elements ("components" in ADF Faces) and attributes ("properties" in ADF Faces). The power of ADF Faces is in the flexibility of the component properties. In this sample code listing, the *value* property of **`af:table`** connects the table component to a data source (EmployeesView3 in this case—an instance of the EmployeesView view object) and assigns a variable name (called "row") to each record in the result set of that view object. Nested within the **`af:table`** component are two **`af:column`** components—representing the EmployeeId and FirstName attributes. Within each column component is an **`af:outputText`** (read-only text) component whose *value* property identifies the table data element within a single record (using the variable "row") that will be displayed in the HTML table cell. The **`af:table`** component is responsible for iterating rows appropriately for the data set.

> **Note**: As discussed more in the next section of this white paper, the "bindings" reference in the af:table component's value property points to the page binding, which connects the ADF BC objects to the components on the page.

Although the **`af:table`** code in the preceding snippet is functional code (many properties are defaulted and properties with default values are not represented in code) many more properties are available. Figure 5 shows JDeveloper's *Property Inspector* (the default property editor for most XML files) displaying the complete set of properties for **`af:table`**. (This display spreads across three columns although JDeveloper shows all properties in a single column.) You can zoom in for a closer look at individual property names, but the main point is that this component offers a lot of options for modifying its behavior or appearance. Some properties are data-oriented as just explained but some supply user-friendly features such as the following:

- **rowSelection** Setting this property to "single" will allow the user to select a row at runtime (by clicking it). The selected row can then be processed in a way you define (for example, to display a popup showing more detail). You can also define the ability to select multiple rows.
- **rowBandingInterval** Setting this property to "1" will shade every other row in the table to make rows visually easier to follow across a wide display
- **filterVisible** If you set this property to "true," the table component will display input fields above each column heading. The user can type a value into one or more of these fields and the displayed rows will be filtered by the entered values.

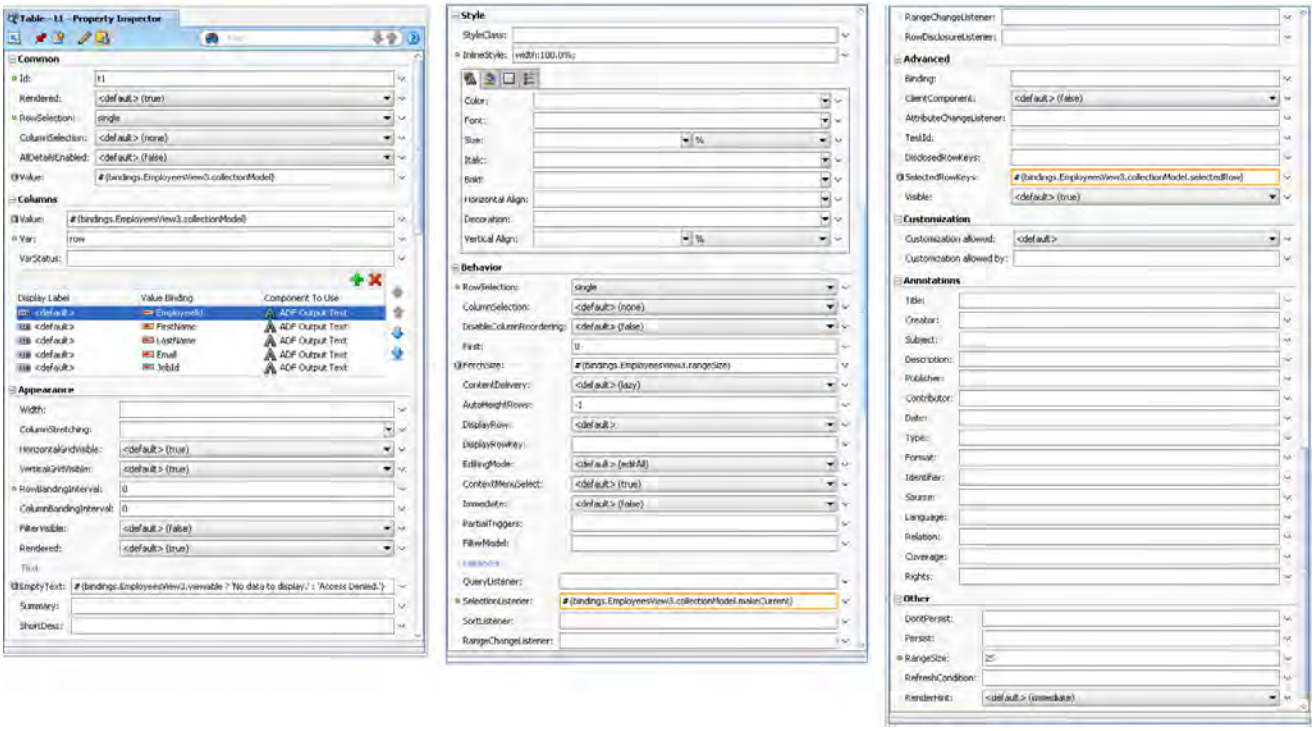


Figure 5. Property Inspector View of the af:table Properties

Declarative AJAX

The recent movement to make web applications more interactive has led to acceptance and wide use of *Asynchronous JavaScript and XML (AJAX)*. AJAX (sometimes spelled as "Ajax") consists of a number of technologies that have existed for some time (such as JavaScript and XML); it allows you to write code that refreshes only part of the page instead of the entire page. This enhances the user experience because the user does not need to wait for the entire page to redraw after clicking a button or link, changing a data value, or interacting with the page in some other way.

ADF Faces components are written with embedded AJAX features. For example, in the preceding code listing, the *sortable* property of the **`af:column`** components are set to "true." This sets up functionality that if the user clicks a column heading the rows displayed will be sorted based on the values in that column. Clicking the same column heading again reverses the sort. As the table is redrawn to display the rows in a different order, the rest of the page stays in place. That is, only the table contents are redrawn. This partial page drawing uses AJAX technology.

AJAX is built into the ADF Faces components so you do not need to write any JavaScript or XML code to cause the partial page redraw. However, with a small handful of properties, you can write your own partial page events, again without writing AJAX code. For example, by declaring property values for Price, Quantity, and Line Total fields, you can cause a refresh of the Line Total field when the user changes either Price or Quantity fields. The rest of the page would remain static. Only the value in the Line Total field would change when Prince or Quantity changes.

> **Note**: AJAX within ADF Faces is more properly called "Partial Page Rendering" (PPR), which specifically refers to the capability to define AJAX functionality by just declaring property values.

Visual Editor

In addition to the Property Inspector and source code view of the ADF Faces components in JSF file, you can view the components in a visual editor that emulates the component runtime. Figure 6 shows the Departments browse page as it appears in the visual editor.

This tool supports drag-and-drop actions for repositioning components. Changes you make in the visual editor are reflected in the source code just as changes you make in Task Flow Diagram are reflected in the controller source code. As an ADF application developer, you create code in any way that is most efficient and intuitive. For example, it is probably easier to reposition buttons by dragging and dropping them in the visual editor rather than reordering lines of code in the source code editor.

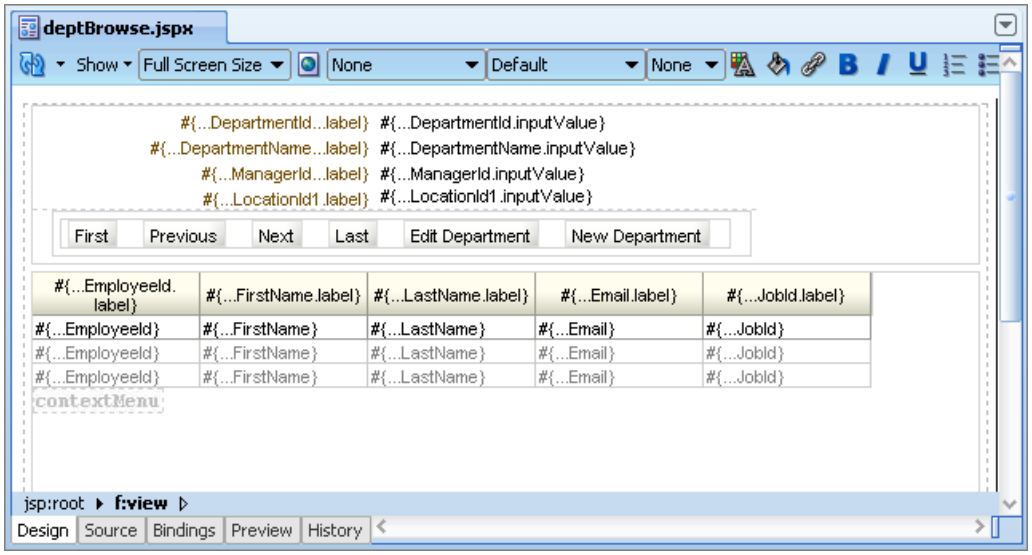
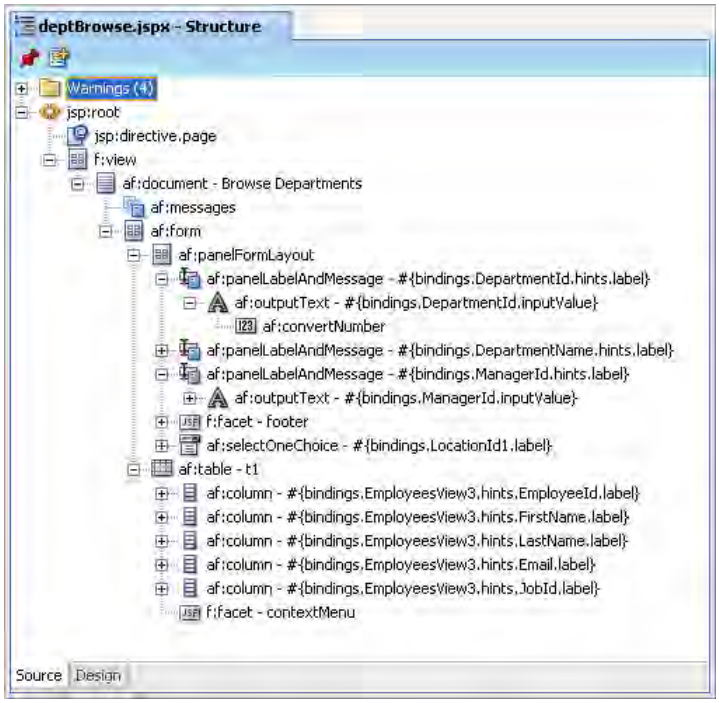


Figure 6. JDeveloper Visual Editor Display of the Departments Browse Page

In addition to the visual editor, you can interact with ADF Faces source code (as well as most other types of code) using the Structure window, shown here:

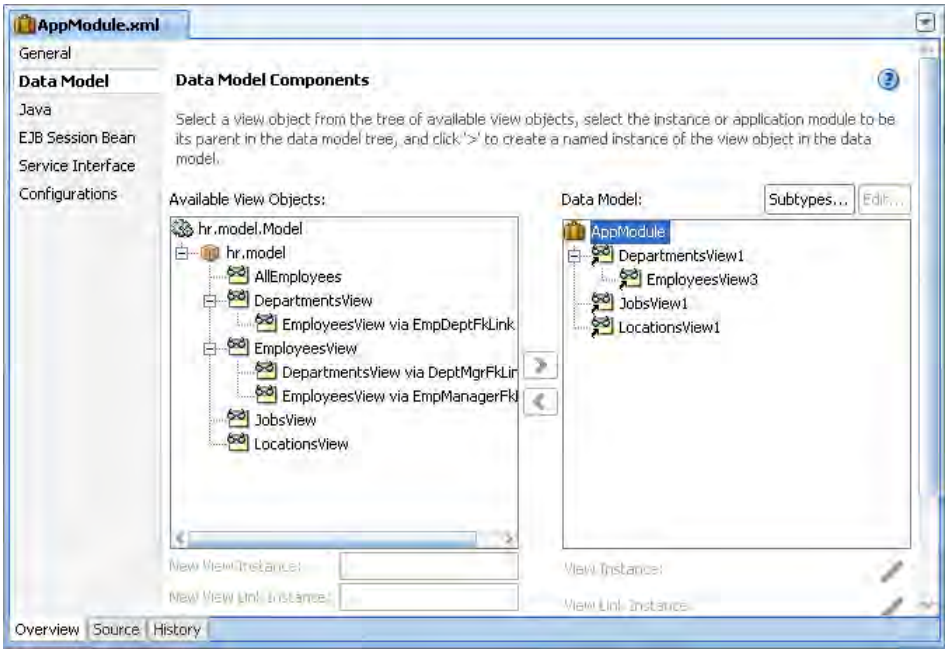


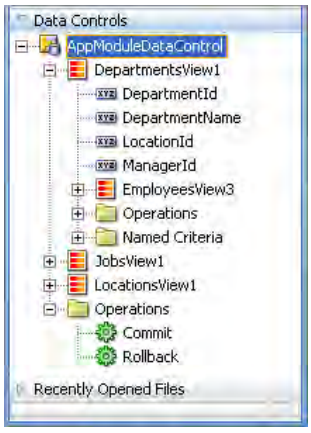
This window displays the hierarchy of ADF Faces and JSF component tags and allows repositioning them using drag-and-drop operations. In addition, you can select, delete, and copy nodes in this window to change the source code. The right-click menu on any node allows you to add components above, below, or inside that component. Errors and warnings are summarized at the top of this view and double clicking an error will open the source code editor to the problem line of code.
Although the sample application displays relatively standard interface components, ADF Faces offers nearly 150 components that you can use to create virtually any user interface you can envision. In addition to simple user input items—for example, text items and pulldowns—ADF Faces also supplies more complex input items such as a date input item with calendar popup, a shuttle control that serves as a multiple selection list, and a full-featured calendar widget. It also provides layout components that allow you to manage the relative positioning of components. In addition, a separate set of ADF Faces components called *Data Visualization Tools* (DVT) provides highly-interactive, Web 2.0, Flash-aware components such as graph, chart, gauge, hierarchy viewer, Gantt chart, map, and pivot table.

> *Le superflu, chose très nécessaire.*
> (The superfluous, a very necessary thing.)
> —Voltaire *(1694-1778), Le Mondian*

ADF Bindings and ADF Data Controls

The Model layer in ADF is composed of two aspects—ADF Data Controls and ADF Bindings. These frameworks link the database components written in ADF BC to user interface components (through the management of pages in the Controller layer). Wiring user interface components to database objects is relatively easy with these two technologies.
The story of how these ADF Model layer technologies work starts back in the ADF Business Services layer. An ADF BC component, the *application module*, manages database transactions (COMMIT and ROLLBACK) and defines the *data model*, a list of view objects and view links that the application uses. The data model is depicted within the Application Module Editor as a hierarchy as shown here:



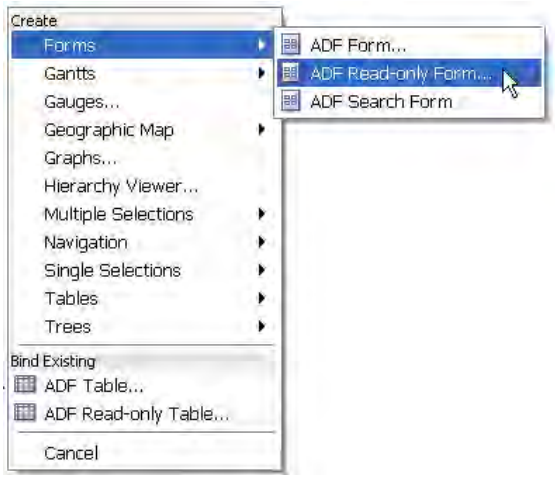


The *Data Model* area in this example defines view objects for DepartmentsView with a detail of EmployeesView (the suffix numbers indicate distinct usages of the view objects in the data model) linked through a view link. A master-level

instance of JobsView and LocationsView (used to supply unfiltered data for pulldowns or LOVs) is also part of this data model. This data model is defined completely within the ADF Business Components application module in the Business Services layer.

Returning to the Model layer, whenever you create a JSF page or page fragment, the Data Controls panel in the JDeveloper navigator will display the ADF BC application module's data model nodes as shown on the right. Additional nodes appear under each view object for attributes (for example, DepartmentId under DepartmentsView1), Operations (that provide actions you can take on the data collection, such as navigating the current record to the Next, Previous, First, or Last record in the set), and Named Criteria (which define which fields will be available for queries using search forms).

An almost magical thing occurs when you drag one of these nodes onto a JSF page or page fragment. For example, to build the sample application, the DepartmentsView1 node was dragged from the Data Controls panel and dropped onto the JSF page. The ADF Data Controls framework determines that the node is a collection-level (table-level) item and displays a menu of applicable components or component combinations as shown on the left with the **Forms** menu expanded.



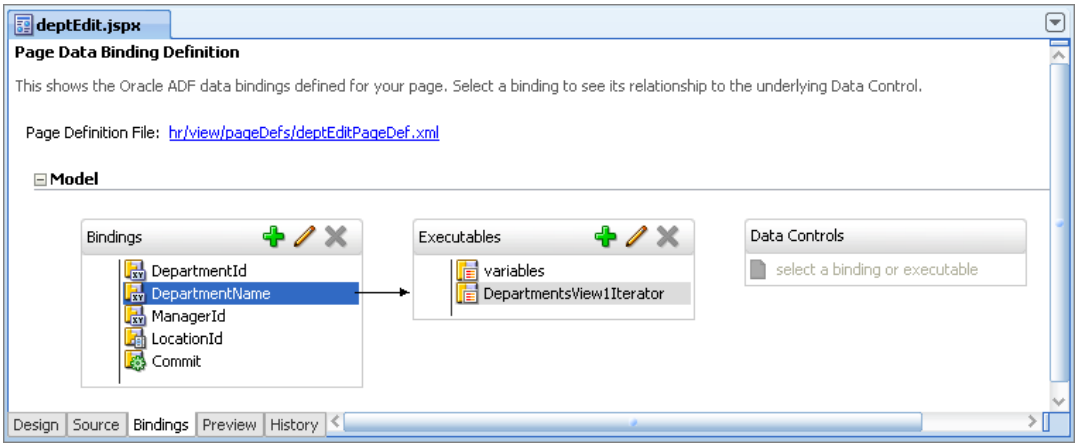
In the sample application, selecting **ADF Read-only Form** caused JDeveloper to create a display containing labeled fields with navigation buttons at the top of the Departments browse page. This drag-and-drop-and-select action interacts with the ADF Data Controls list. If an individual attribute node (such as DepartmentId) is dragged instead, a list of data controls appropriate to a single data value (for example, input text items, output items, and pulldowns) will display instead.

In addition to drawing user interface components on the screen, the drag-and-drop operation also creates bindings for those components. *Bindings* are code or definitions that declare which data from a business service will be connected to a user interface control or structure. Bindings appear in the ADF Faces' property values. The following example is an ADF Faces input text component from the Edit Department page:

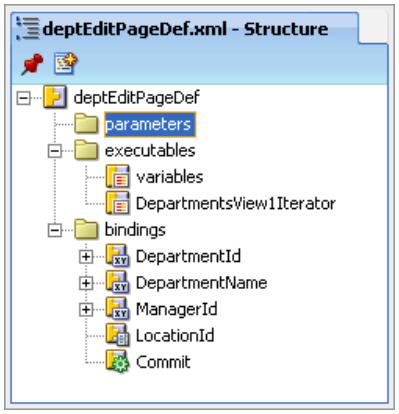
```
<af:inputText value="#{bindings.DepartmentId.inputValue}"
              label="#{bindings.DepartmentId.hints.label}"
              required="#{bindings.DepartmentId.hints.mandatory}"
              columns="#{bindings.DepartmentId.hints.displayWidth}"
              maximumLength="#{bindings.DepartmentId.hints.precision}"
              shortDesc="#{bindings.DepartmentId.hints.tooltip}"
              id="it1">
</af:inputText>
```

All of this code was created by the Data Controls panel drag-and-drop operation. This is one of the main advantages of the Data Controls panel: it builds all the property values for you and automatically binds the components to data. The property values defined using the "#{ }" delimiters are Expression Language expressions. *Expression Language* (EL) is a high-level, non-procedural language specified in the JavaServer Pages standards. It is used within JSF pages to refer to potentially dynamic sources of data that will supply property values at runtime.

In this case, all EL expressions begin with "bindings," which is the context for the values. This context refers to a PageDef (Page Definition bindings) file that JDeveloper creates for each JSF page. You can view the bindings in this file using the Bindings viewer for the page as shown here:



If you need to look at or manipulate the bindings code, you click the link next to the Page Definition File label to open the PageDef file—the container for the bindings definitions. The Structure window view of this page is shown on the right.



You will see an *executables* section for the queries (iterators) that occur when the page opens. You will also see a *bindings* section for the objects that refer to view object attributes. By now, you will not be surprised that JDeveloper creates XML code to define bindings; you will rarely need to touch this code. Here is a code snippet from the deptEditPageDef.xml file:

```
<bindings>
  <attributeValues IterBinding="DepartmentsView1Iterator"
      id="DepartmentId">
    <AttrNames>
      <Item Value="DepartmentId"/>
    </AttrNames>
  </attributeValues>
```

This file is processed by the ADF Bindings framework code and links the attribute, DepartmentId, to the iterator, DepartmentsView1Iterator. That iterator is defined for the DepartmentsView1 view object instance in the data model, and therefore represents a query of data. The EL bindings expressions in the ADF Faces component code point to this communication path and therefore to data. The EL expressions also further drill into a specific property of the ADF BC view attribute; for example, the label property of the example component is defined as "#{bindings.DepartmentId.hints.label},” which refers to the label property of the view attribute (in the hints property category). If no label property is defined, the default label is the attribute name.

Which Languages Are Important?

Now that you have sampled some ADF techniques for working with each of the core technologies, you know that JDeveloper creates a lot of application code automatically when you interact with its visual and declarative tools. However, you may still be wondering about which languages you will use when you need to supplement this code. First, remember that ADF was created as a visual and declarative environment to interact with many frameworks. Therefore, a key skill is knowing how to squeeze the most functionality out of the technologies by just defining property values and laying out components visually. The less code you need to write, the less code you need to debug.

With the goal of “declarative if at all possible” in mind, you can be quite productive without writing much code. However, you will come to a point where writing code is necessary and you will be using a combination of languages. The following list summarizes the main languages you need to know and how you will use them:

- **XML** As you have seen, work with frameworks makes heavy use of XML code. However, you work with most XML code in JDeveloper using declarative and visual tools. You will rarely need to type XML elements and attributes in these files, but the level of skill you will need at that time is very basic. You mainly need to know three

things about XML: elements need ending tags; elements have attributes that refine the element's use; and elements can be nested within elements to create an element hierarchy.

- **Java** You will write snippets of Java inside ADF BC classes and View layer code to perform customized tasks that the frameworks cannot provide. You can be quite productive in the ADF Fusion Technology Stack with a novice level knowledge of Java if you have someone on your team who understands Java at an expert level. This person can step in to assist if you run into a requirement that cannot be handled with a basic knowledge of Java.
- **HTML** For best use of JSF and ADF Faces concepts, you will avoid writing HTML code. Instead, you use high-level components that generate HTML for you.
- **Cascading Style Sheets (CSS)** ADF Faces components use CSS styles defined in a *skin*, a set of style selectors that provide a common look-and-feel to all your pages. You will use CSS to define the skin at the start of the first ADF application project, but will not need it much after that because you will apply the same skin to all applications in your organization.
- **JavaScript** ADF Faces components use JavaScript internally to provide user-friendly features such as refreshing part of a page when scrolling to the next set of records. You will usually not need to write JavaScript or AJAX code because the components provide many of the features you would normally need other languages to supply and allow you to declare AJAX functionality using only property values.
- **Expression Language** EL is used to supply dynamic values to JSF components' properties. The main learning curve for EL is in knowing how to start to build the correct expression. Fortunately, JDeveloper can assist. In the pulldown for most properties is an item for "Expression Builder." This selection displays a navigator that helps you create properly formatted EL expressions. It is a good learning tool as well as a way to enter proper EL.
- **Groovy** ADF Business Components allow you to write validation and message code using this language. As with EL, Groovy is used at a very basic level and understanding a few fundamentals as explained in the JDeveloper online help system will suffice.

Additional Resources

The intention of this white paper is to get you started thinking about ADF, Fusion, and techniques you will be using in JDeveloper to build web applications. The main source of all things ADF is the JDeveloper home page on Oracle Technology Network (www.oracle.com/technology/products/jdev/). Follow the links on that page to access tutorials and articles about specific techniques. In addition, the "Learn More" tab on that page currently displays a "Learn More About" link to information about ADF. The Technical Resources section contains links for Developer Guides; the "Fusion Developer's Guide" is a good starting point for a wealth of technical detail about ADF. This Oracle website guide is linked to within the JDeveloper help system as well.

Speaking of the help system (technically called the "Help Center"), the Help menu contains link to Tutorials (v.11.1.2) or Cue Cards (11.1.1), which step you through creating a specific type of code and are especially helpful when learning ADF. Another extremely useful Oracle resource can help in learning about ADF Faces RC components: the ADF Faces Rich Client Components Hosted Demo (available at jdevadf.oracle.com/). This demo shows all ADF Faces components and allows you to change properties to see how they work.

> *Le secret d'ennuyer est celui de tout dire.*
>
> (The secret of being a bore is to tell everything.)
>
> —Voltaire (1694-1778), *Sept Discours en Vers sur l'Homme*

Conclusion

Admittedly, this is a lot of information but, hopefully, you now have a better idea about Oracle Fusion and ADF as well as about the basics about each of the core technologies in the ADF Fusion Technology Stack: ADF BC, ADF Controller, ADF Faces RC, and ADF Model: Bindings and Data Controls. This white paper has shown the type of code you will be creating and the style of development work you will be performing to create that code in each of these technologies. This

overview information should help in your understanding of what you need to know to be productive with ADF and Fusion technologies and to start up the on-ramp to the Fusion Development Highway.
May that road rise up to meet you!

> *Il faut cultiver notre jardin.*
> (Let us cultivate our garden.)
> —Voltaire (1694-1778), *Candide* (Ch. xx)

About the Author

Peter Koletzke is a technical director and principal instructor for Quovera, in Mountain View, California, and has 29 years of industry experience, 24 of which is in the Oracle arena. Peter has presented at various Oracle users group conferences more than 310 times and has won awards such as Pinnacle Publishing's Technical Achievement, Oracle Development Tools Users Group (ODTUG) Editor's Choice (three times, one of which is for this white paper), ODTUG Best Speaker, ODTUG Volunteer of the Year, NYOUG Editor's Choice (three times), and ECO/SEOUC Oracle Designer Award. He is an Oracle Certified Master, Oracle ACE Director, and coauthor (variously with Dr. Paul Dorsey, Avrom Roy-Faderman, and Duncan Mills) of eight Oracle Press development tools books including *Oracle JDeveloper 11g Handbook*.

Upcoming Meeting Dates

SAVE THE DATE: NYOUG Spring 2014 General Meeting
DATE: Wednesday March 12, 2014
LOCATION: St. John's University – 101 Murray St. New York, NY 10007

RMOUG Training Days 2014
DATES: February 5 - 7
LOCATION: Denver, Colorado

HOTSOS Symposium 2014
DATES: March 2 - 6
LOCATION: Dallas, Texas

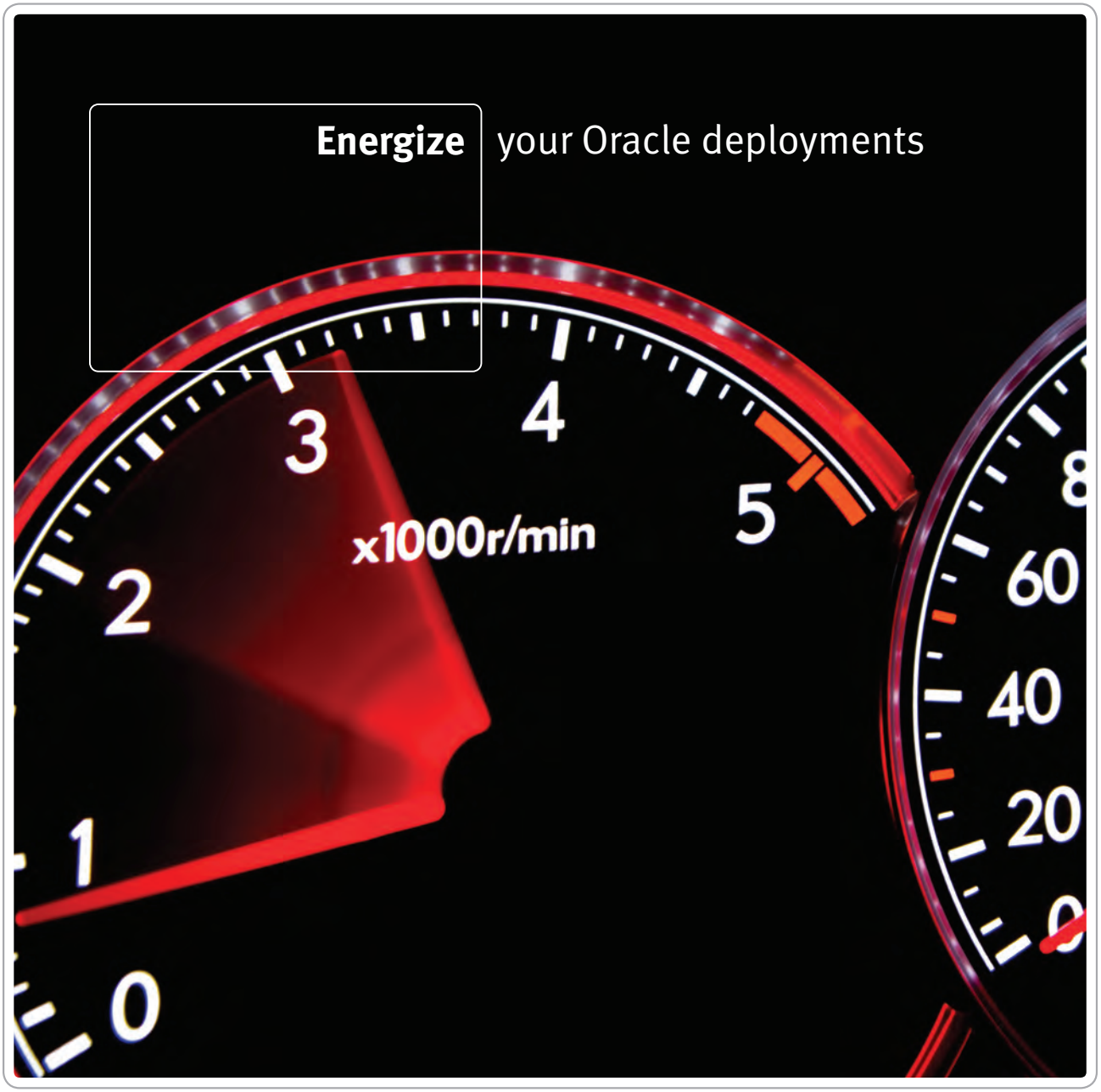
IOUG Collaborate 2014
DATES: April 7 - 11
LOCATION: Las Vegas, Nevada

OUG Scotland 2014
DATES: June 11 - 12
LOCATION: Linthithgow, Scotland

ODTUG Kaleidoscope 2014
DATES: June 22 - 26
LOCATION: Seattle, Washington

Oracle Open World 2014
DATES: September 27 – October 2
LOCATION: San Francisco, California

EMC²
where information lives®



Reduce risk and accelerate your application deployments by drawing on the power of our Oracle® expertise.

With EMC® Proven® Solutions, your information infrastructure accelerates towards greater productivity. Learn more at www.EMC.com/oraclesolutions.

EMC² PROVEN SOLUTIONS

Managing Statistics of Volatile Tables in Oracle

Iordan K. Iotzov, News America Marketing (NewsCorp)

Introduction

Adequate, up-to-date table and index statistics are of utmost importance for achieving optimal database performance. Unlike profiles, hints, and outlines, which can only help tune a narrow set of queries, database statistics assist the optimizer, a very sophisticated program, to deliver excellent results for every query every time.
Oracle's default statistics gathering process collects statistics every night and over the weekend. The process works well for tables that follow a traditional growth pattern or have low volatility; however, it is inadequate for tables which experience fluctuations in size or data distribution on a frequent basis. The challenge is even greater when we are not allowed to change the application design and queries, an increasingly common situation thanks to the growing use of off-the-shelf solutions.
After a definition of volume and distribution volatility, methods and consequences of reducing it are explored. Tradeoffs between statistics management simplicity and resource utilization are discussed.
Following a review of the benefits and the pitfalls of using dynamic sampling and locking statistics to manage the statistics of volatile tables, a robust algorithm that delivers both plan stability and system flexibility is proposed. The algorithm works by allowing the statistics to change only under specific circumstances.
Since handing volatile table statistics often involves directly invoking DBMS_STATS procedures, the missed opportunities due to the fact that Oracle's DBMS_STATS package issues an implicit commit are discussed. Transaction consistency and easiness to recover after a failure, need for functional testing, and inability to gather statistics in triggers are covered. JUST_STATS, a novel custom PL/SQL package for collecting table and index statistics, is proposed and explained. The JUST_STATS package is functionally equivalent to a subset of DBMS_STATS package, except that it does not issue a commit. Examples illustrating the use of JUST_STATS in batch processing and off-the-shelf applications, including statistics gathering in table triggers, are shown.

Definition of Volume and Distribution Volatility

Volatility, a term frequently used in finance, is a measure of dispersion. While the exact definition(s) of volatility is beyond the scope of this paper, it is typically measured by using the standard deviation or variance. Standard deviation (σ) is a basic statistical measure that can be estimated with:

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

where $\{x_1, x_2, \ldots, x_N\}$ are the observed values and $\bar{x}$ is the mean of those observations.
For the purposes of this paper, an Oracle table is volume volatile when the standard deviation of the number of its records is significant compared to its average number of records. This broad definition covers the variety of different scenarios we are going to review. Fortunately, volatility is a rather simple and intuitive concept. Figure 1 shows a side-by-side comparison of the number of records of a volatile table and a regular (non-volatile) one.

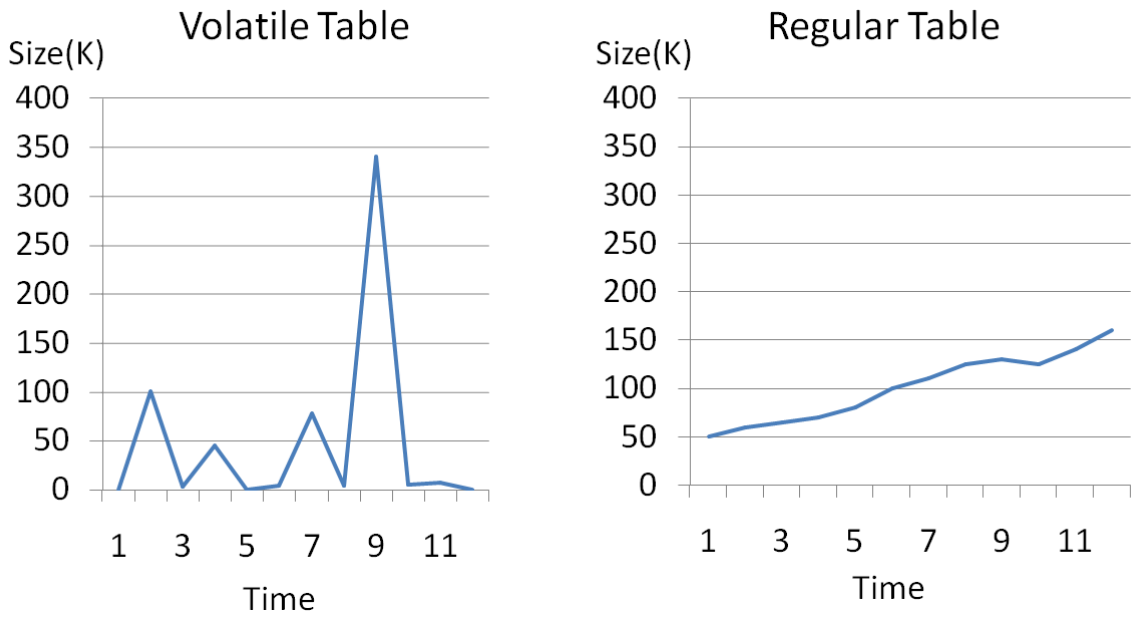


Figure 1. Volume Volatility

While the number of records is a very important characteristic of an Oracle table, there are also other attributes that could influence the Oracle's Cost Based Optimizer (CBO). Figure 2 shows the distribution of the records of a table column at two points in time. The data is sourced from the following query:

```
select      col1, count(*)
from        tab
group by    col1
```

At Time 1, AZRF and GDVR values have the most records; while at Time 2, BBTS and LTTP are the most popular. Even though the number of records in the table has not changed a lot, the distribution of records has. The change in column distribution can have profound effect on execution plans generated by the CBO, particularly if there is a histogram on that column.

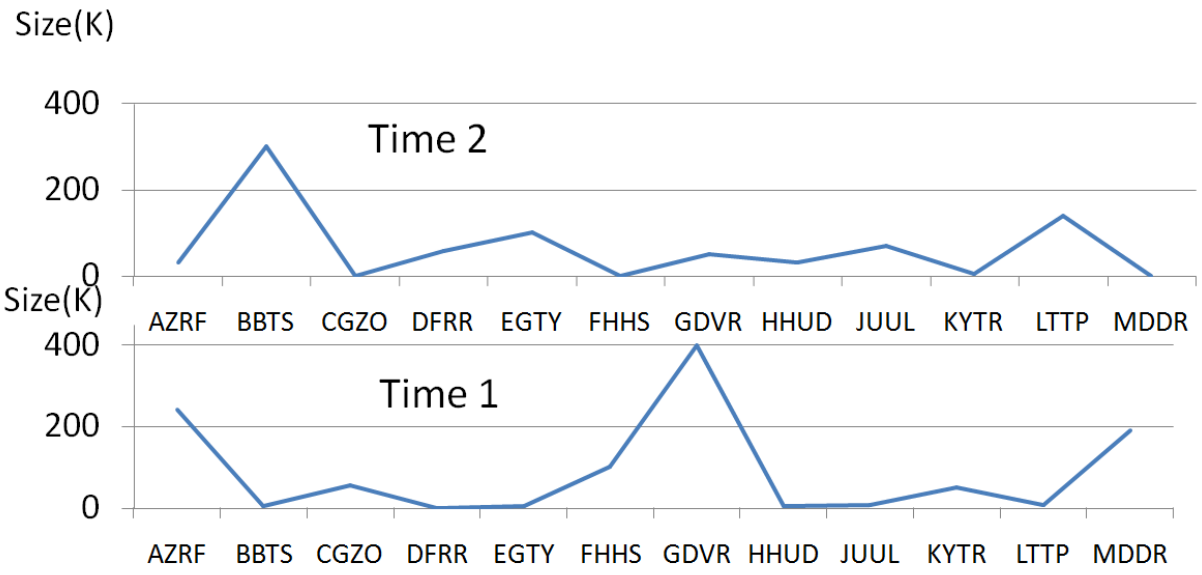


Figure 2. Distribution Volatility

Reducing Volatility

Oracle CBO's algorithm for generating SQL execution plans uses variety of information, yet nothing is more important than the statistics of the underlying application tables and indexes. Those statistics are stored in the data dictionary (DD). In most cases, CBO would not read any data from application tables or indexes while generating execution plan, but would rely entirely on the statistics of that data in the DD. The assumption that the statistics in the DD correctly represent the data they are describing is fundamental in performance optimization. For instance, if CBO creates an execution plan under the assumption (based on stats stored in DD) that a table has five records, but the table turns out to have a million records, then the generated execution plan would likely be subpar. Likewise, if an execution plan is for a million record table (as per stats in DD), but the table at time of execution has only a dozen records, then there is probably a better execution plan.

Table volatility could present an enormous challenge in any Oracle database because it increases the chance of discrepancy between the statistics stored in the DD and the actual data. Even though Oracle provides mechanisms for dealing with volatile tables and this paper introduces new techniques in that area, management of statistics of volatile tables continues to be a daunting task.

Proactive Volatility Reduction

Not creating volatile tables in the first place is by far the best approach one can take. People involved with database design should be aware of the pitfalls of using volatile tables and consider alternative options wherever possible. Frequently, volatile tables are not at the core of an application, but are rather used as an intermediary/staging storage. Since entities that store intermediate data are most frequently mapped to volatile tables, database designers and developers should consider consolidating processes and SQL statements, eliminating the need for those entities.

Reactive Volatility Reduction

Sometimes, for various reasons, we cannot just eliminate all volatile tables from our databases. Frequently, however, we have the ability to change the physical design of volatile tables in a way that would make them less volatile. The two-phase removal method not only reduces volatility, but can also speed up some delete operations. The idea behind this method is to use a binary delete flag to mark a record logically deleted instead of physically deleting the record from the table. To limit the resource footprint and prevent unlimited data growth, we must have a regular purge process to physically delete all records marked as deleted.

To convert a volatile table `tab` using two-phase removal, you need to create a table `tab_internal` that has the same columns as `tab`, plus a new `deleted` column. The `deleted` column would be the flag indicating whether the record was logically deleted. The content of the `tab` has to be inserted into `tab_internal` with appropriate `deleted` flag. Then the `tab` table has to be dropped and a new `tab` view has to be created. The new `tab` new would be functionally equivalent to the old `tab` table.

```
table tab
( col1 NUMBER,
…
col10 VARCHAR2)
```

=>

```
table tab_internal
( col1 NUMBER,
…
col10 VARCHAR2,
deleted VARCHAR2(1)
constraint del check
(deleted in ('Y','N')))
```

+

```
view tab as
select col1, col2,
… col10
from tab_internal
where deleted =
'N'
```

The effect of two-phase removal on table volatility can be seen in Figure 3. Two-phased removal leaves significantly larger footprint, but it has quite low volatility after the initial build-up. The original table by comparison was volatile, but consumed little disk space.

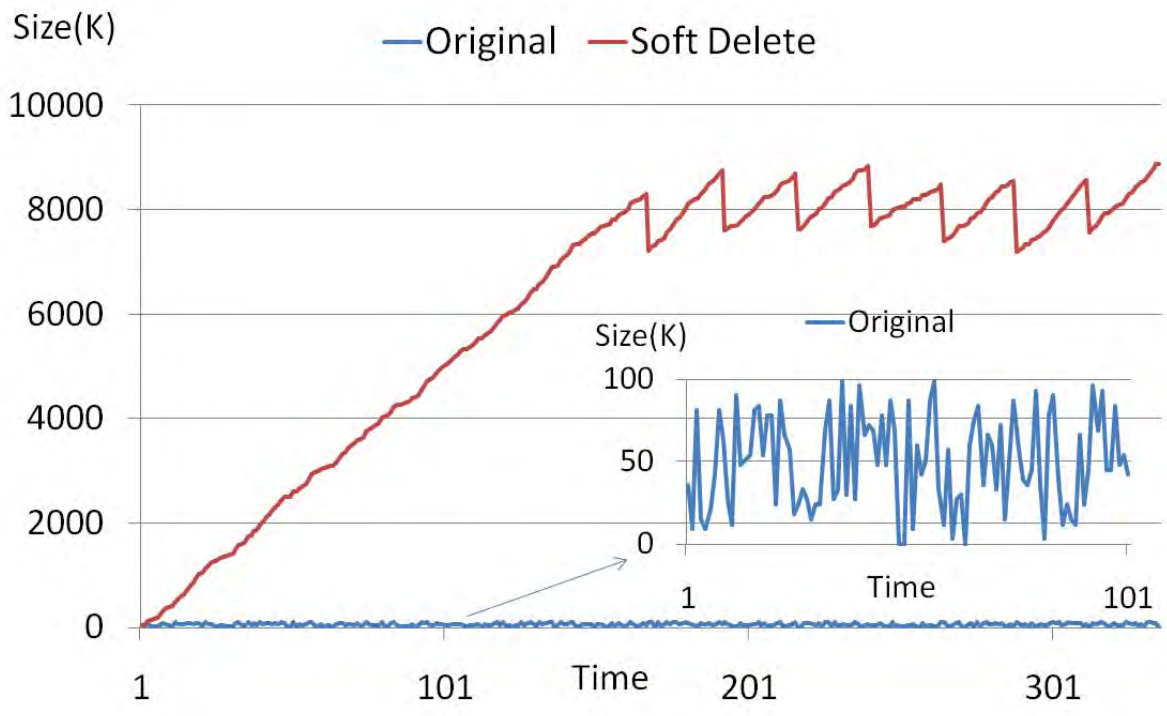


Figure 3. Comparison of Two-phase Removal vs Original

The pros and cons of this approach can be summarized in Table 1.

Pros	Cons
• No need to change select statements • Stable execution plans	• Larger footprint • Limited options for CBO(no FTS) • Does not help with distribution volatility • Column statistics represent average

Table 1. Pros and Cons of Two-Phase Removal

The two-phase removal approach requires little or no change of application SQL. SELECT statements, often the most frequently utilized SQL statements, are never affected. The substitute view (`tab`) restricts the output to records that are not deleted, making the view's content equivalent to the content the original table (`tab`) would have had. The reduced table volatility promotes stable execution plans, which in turn result in predictable SQL execution times. Those plans are often not the best, but produce reasonable and consistent execution times.

The two-phase removal approach has its drawbacks. The disk space required to accommodate the technique could be significant. While we could reduce the retention time of logically deleted records and thus decrease disk space footprint, doing so would increase table volatility, defeating the whole purpose of the project. Since underlying tables (`tab_internal`) are much larger now, as large as hundreds of times the size of the original table (`tab`), we effectively limit the CBO from utilizing full table scans (FTS) to get to the data. Also, the two-phase removal approach does not help with distribution volatility. Column statistics, including histograms, are created on all data, deleted and active (non-deleted), so there is no way to know the distribution of the active data alone. Since table and column statistics are averaged over the retention period, they would not properly account for unusually large active data set, since that large active set would only be a relatively small part of the overall data set (deleted plus active).

There are two implementations of two-phase removal. The first one requires changing all insert and delete statements, but is prone to few performance challenges, while the second one would not require any change in any SQL, but may introduce performance issues.

The first implementation requires application DMLs to be modified to refer to `tab_internal` directly. All insert SQLs should be modified according to this rule:

```
insert into tab
(col1,..col10)
values
(col1,..col10)
```

=>

```
insert into tab_internal
(col1,..col10,deleted)
values
col1,..col10,'N')
```

All delete SQLs should be transformed following this rule:

```
delete tab
where  col1= ..
```

=>

```
update into tab_internal
set deleted = 'Y'
where deleted = 'N'
and col1= ..
```

While the first implementation requires application code change, it allows the application to issue highly efficient bulk DML operations against volatile tables (Table 2).

The second implementation requires no application SQL changes whatsoever. The two phase removal is implemented with row level "instead of" triggers on the newly introduced `tab` view.

```
create or replace trigger v_t_tr instead
of insert on tab
 begin
   insert into tab_internal (col1,..col10,deleted)
   values (col1,..col10, 'N'  );
 end;
```

```
create or replace trigger v_t_del
 instead of delete on tab referencing new
as new old as old
   begin
     update tab_internal      set deleted = 'Y'
    where col1  = :old.col1
    and col2 = …
end;
```

An upside of this approach is that no application code change is needed. Please note that this does not imply that the application is still supported by its vendor – only the vendor can make that determination. Row-by-row internal processing imposed by the row-level triggers could have significant performance implications (Table 2).

	Pros	Cons
Implementation one (Keeping bulk DML operations solution)	• Ability to archive high performance by utilizing bulk operations	• Have to change the code
Implementation two (Trigger-based solution)	• No need to change the application code	• Some DML performance limited by row-by-row processing

Table 2. Pros and Cons of Two-Phase Removal Implementations

Dealing with Volatility

When reducing table volatility is not an option, we have to develop ways to deal with the performance challenges brought by volatile tables. There are two major database design choices for building systems that work well with volatile tables. The goal of the first approach, *robust execution plans*, is for the CBO to generate execution plans that perform reasonably well regardless of the exact number of records in a volatile table. It emphasizes stability and predictability at the expense of optimality. The goal of the other approach, *follow the change*, is to keep the CPO statistics in DD in sync with the content of the table, promoting optimal execution plans at all times.

Robust Execution Plans

Designing systems that can perform without failure under wide range of conditions has been a goal for scientists and engineers for many years. The difficulty of properly accounting for and effectively minimizing those variations is the driving force behind those efforts. The premise of robust design is to build systems that are insensitive to variations, also

called noise (Dehnad, 1989). Genichi Taguchi, a pioneer of robust design, introduced signal to noise ratio as a measure of system robustness.

$$SN = 10 \log_{10} \frac{E^2 Y}{Var\, Y},$$

where $E^2 Y$ is the mean, and $E^2 Y$ is the variance of Y.

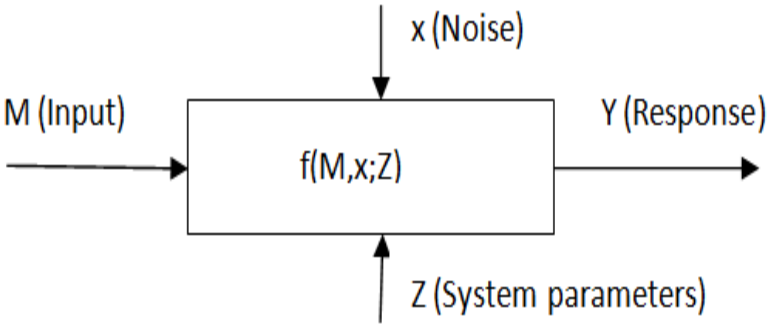


Figure 4. Robust Design Schema

Optimizing robust systems presents a major paradigm shift. A robust system could deliver slightly sub-optimal performance for any specific setting (M,x_1), as the optimal system parameters (Z) for that setting can be sub-par if the system experiences noise (M,x_2). A good robust system would deliver consistently good results regardless of noise.
In Oracle CBO's context, M (Input) would be the data dictionary (table, index and system) statistics, x (noise) would represent discrepancies between the statistics stored in DD and the actual statistics, Z(System parameters) would be the parameters we can use to influence CBO's behavior, such as system (init.ora) parameter, hints, custom DD manipulations, and Y (Response) would be the execution time of the plan generated by the CBO.
An execution plan specifies the join order of the tables, the join method and how the predicates/filters would be applied. Each of those components is very important.
The results of an empirical study about the two major join methods used in Oracle can be seen in Figure 5.The graph on left side shows the size of a volatile table (in K bytes). The graph on the right represents that execution time of a two table join between the volatile table (left side) and a table with static content. We can see that nested loops (NL) is the optimal type of join when the size of the volatile table is less than approximately 18K, while hash join (HJ) is the best for bigger sizes of the volatile table. As expected, the execution time fluctuates as the size of the volatile table fluctuates. The variation of the execution time, however, is quite different for the two join types. The execution time for hash join is associated with smaller variation adjusted to mean, which corresponds to higher signal-to-noise ratio and is therefore more robust than nested loops.

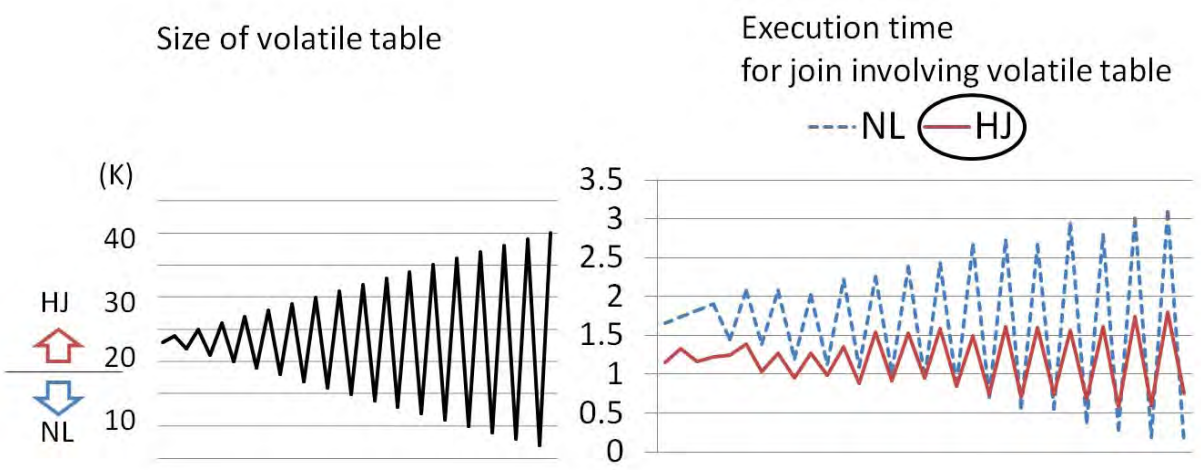


Figure 5. Robustness Analysis of Nested Loops (NL) and Hash Joint (HJ)

This finding does not imply that an optimal robust plan should contain only hash joins. If we know that the records in the volatile table cannot be more than a certain value, 18K in this example, then nested loops would always outperform hash join and would be the optimal join type.

Oracle 12c introduced Adaptive Execution Plans, a feature that allows the DB to decide the join type during runtime based on the actual number of records. The decision about the join type is done only at the first execution and it is reused by the consequent executions (Osborne, 2013). This feature can successfully mitigate stats inaccuracies, including those caused by volatile tables, in some cases.

Despite Oracle 12c improvements, such as Adaptive Execution Plans, table/index statistics continue to be a very important factor for getting a good execution plan. Stale statistics can cause suboptimal join order (Figure 6), which typically manifest itself with a huge intermediate data set. No Oracle feature, as far as I am aware, can compensate for that.

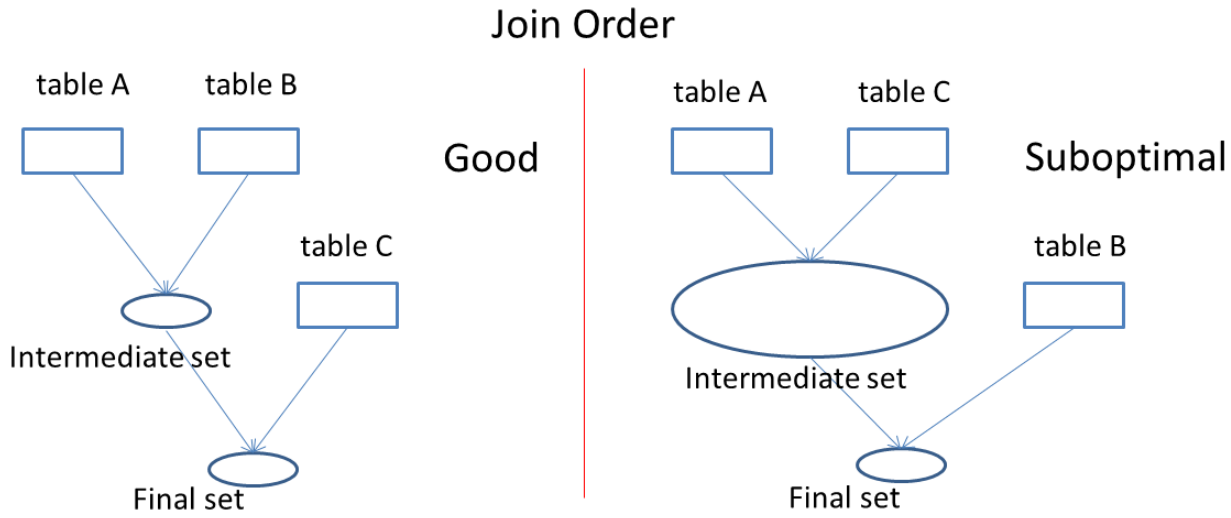


Figure 6. Join Order of a Query

Locking table and index statistics when the table has reached its maximum size is one of the recommended techniques for dealing with volatile tables (MOS,n.d. a). This advice is consistent with principles of robustness discussed earlier. Locking statistics at the high range of possible values could promote hash joins, the robust choice, instead of nested loops. The resulting query would be optimal when the volatile table is close to its maximum size, but it would have suboptimal, yet reasonable, performance when the volatile table has fewer records. The other alternative, locking the statistics to a value in the lower to mid range, could have disastrous performance effects. When a "nested loops" plan generated for a table with dozens of records is used when the table has thousands of records, it would naturally result in sub-optimal execution time. The problem is that unlike the previous scenario, the execution time could be tens to hundreds of times worse than the optimal execution time.

Simply locking the statistics of a volatile table is usually not a long term solution though. Data changes over time, so the content of most tables, including the volatile ones, also changes. Locking the statistics when the table has reached its maximum size is imperative, yet almost impossible proposition if the volatile table is to grow over time. Even if the number of records does not change, the content of the records almost certainly will. Column statistics, such as minimum and maximum values, are instrumental in the way CBO computes set cardinality (Lewis, 2006). Without histograms, CBO approximates the number of records for a value based on the density column statistic. As shown in Figure 7, for values outside this min/max range, the CBO gradually reduces the estimate, while for values well outside the min/max range, the CBO assumes that only one record would be found.

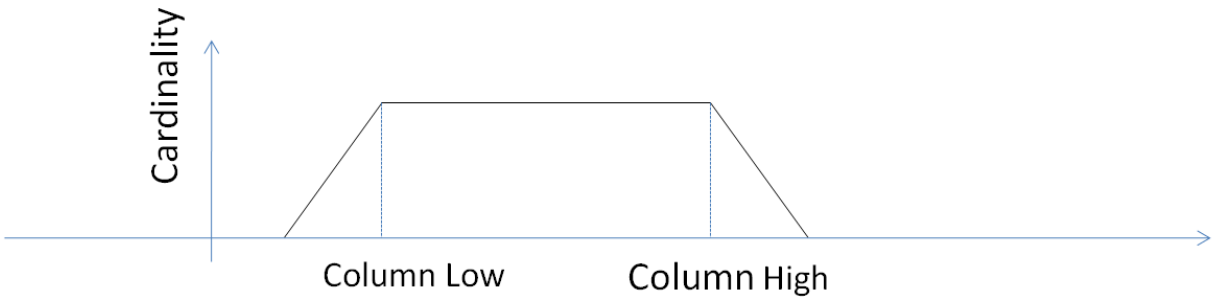


Figure 7. Cardinality Adjustments

The right portion of Figure 8 illustrates how the minimum and maximum values of a column in a volatile table change over time. The left portion shows the cardinality adjustments, and is based on the column statistics at starting time. Over time, the max value of the column grows, but its corresponding statistics stay the same. As a result, the cardinality estimate for the max value goes down and down, until it reaches one. The incorrect lower cardinality estimate negatively affects execution plans.

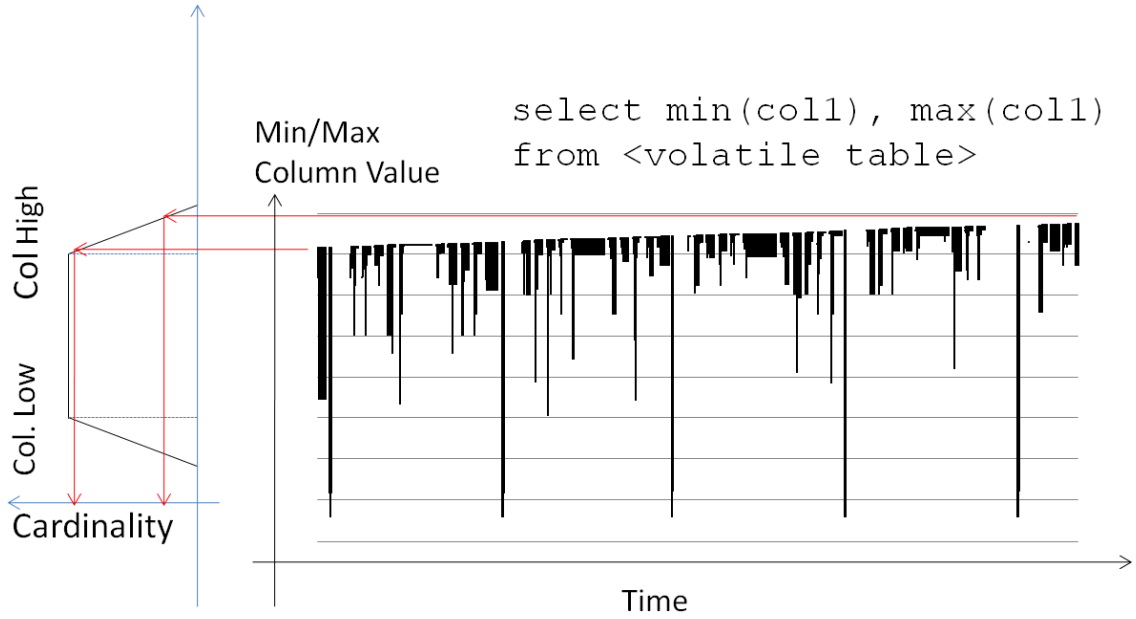


Figure 8. Min/Max Ranges of a Volatile Table

To overcome those limitations, I propose a new adaptive stats locking algorithm. It is flexible, yet delivers stable execution plans. The new approach is illustrated in Figure 9. First, we count the number of records in the volatile table. That number is compared to the previous number of records recorded in the DD, multiplied by a threshold value. If the new table count passes the test, we proceed with stats collection; otherwise we do not gather stats.

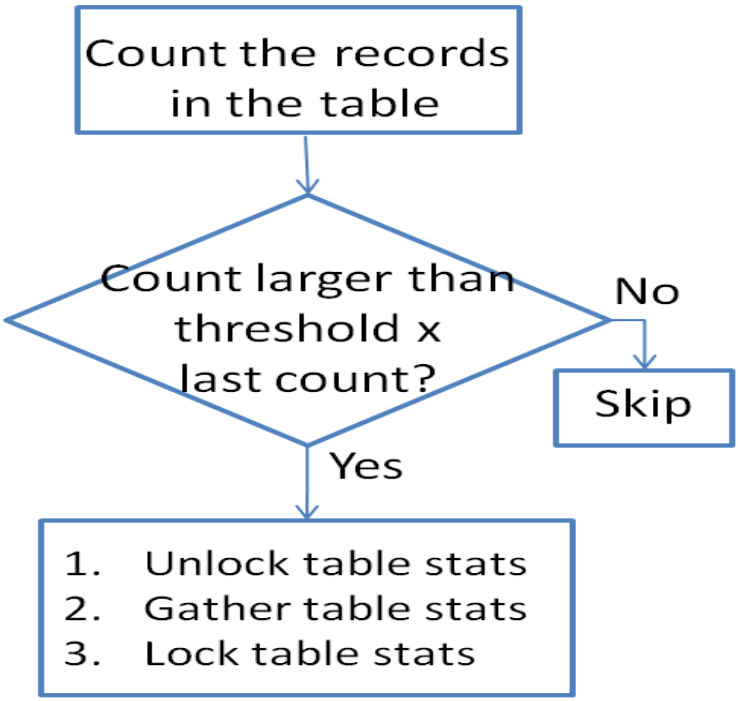


Figure 9. Adaptive Stats Locking

#
By comparing the new count with the count already stored in the DD, the algorithm does not allow the number of records statistic in the DD to drop sharply. This ensures that the statistics in DD are locked to values that represent maximum table size. The new algorithm is more flexible to data changes, because unlike the regular statistics locking, it does gather statistics on some occasions. The balance between stability and flexibility can be controlled with the threshold parameter. A good rule is to have the threshold parameter decline over time and get reset when a stats gathering occurs. While the exact parameters of the threshold function could differ from system to system, the structure shown in Figure 10 is usually adequate. The sample function there is defined as

```
threshold = 5%,if stats were gathers within a week
1/(5*(abs.days_since_last_stats_gather)-3), otherwise
#
```

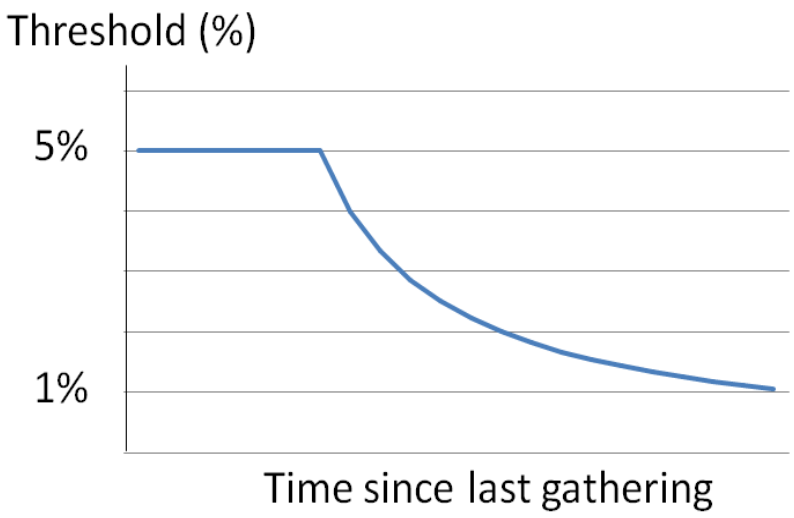


Figure 10. Adaptive Stats Locking Threshold Function

Figure 11 illustrates how the new algorithm works in practice. Even though the table size fluctuates a lot, going to zero in several occasions, the stats for that table in the DD are relatively stable. The number of records in the table according to the DD stats varies between around 300K and 1200K. Column statistics in the DD are updated every time new stats are gathered, a rather frequent occurrence in this case.

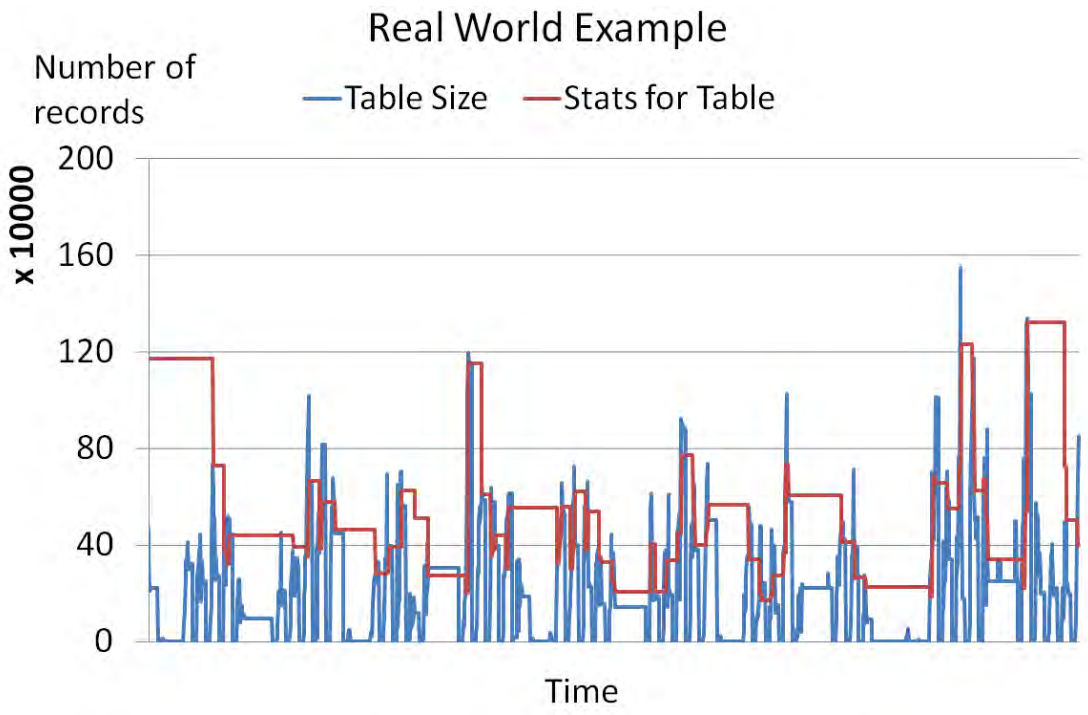


Figure 11. Examples of Adaptive Stats Locking

The fact that volatile tables can change their content very quickly could present some unique implementation challenges. The algorithm assumes that the volatile table does not change from the time we do the initial count until the time we gather the stats. That is not always true, so to prevent incorrect results, we have to explicitly verify that assumption every time we run the procedure.

The implementation for Oracle 10g is shown in Table 3. The first step is to back up the current statistics, in case we have to revert to them. After that, table statistics are gathered. The next step is to verify that the newly gathered stats are what we expected. An easy way to check that is to compare the count of records taken at the beginning of the procedure with the record count stored in the DD. If a significant discrepancy is found, the statistics backed up in step one are restored.

Step	SQL
Backup existing statistics	truncate table prev_stats ; execute DBMS_STATS.EXPORT_TABLE_STATS (<DB_USER>,<TAB>, stattab => 'prev_stats');
Gather statistics	exec dbms_stats.gather_table_stats(<DB_USER>,<TAB>)
Verify that the gathered stats are what was expected?	select num_rows from dba_tables where owner = <DB_USER> and table_name = <TAB>
If not – restore statistics from backup	exec DBMS_STATS.IMPORT_TABLE_STATS (<DB_USER>,<TAB>, stattab => 'prev_stats');

Table 3. Adaptive Stats Locking Implementation Oracle 10g

The implementation for Oracle 11g is a bit simpler – Table 4. First, we specify that the table stats for the particular table would go into a pending area. Then we gather stats. Next, we compare the newly gathered statistics, still in the pending area with the in values in the DD which represent the previous state. Finally, if the new stats are OK, we proceed with publishing them to the DD.

Step	SQL
Keep new stats in pending state	exec dbms_stats.set_table_prefs((<DB_USER>,<TAB>,'PUBLISH', 'false');
Gather statistics	exec dbms_stats.gather_table_stats(<DB_USER>,<TAB>)
Verify that the gathered stats are what was expected?	select num_rows from dba_tab_pending_stats where owner = <DB_USER> and table_name = <TAB>
If yes – publish the statistics	exec dbms_stats.publish_pending_stats(<DB_USER>,<TAB>);

Table 4. Adaptive Stats Locking Implementation Oracle 11g

In general, custom management of table/index statistics is rife with challenges. Non-overlapping column ranges is a problem that could show up when the stats for different tables are locked at different times. Figure 12 illustrates that situation. According to the DD, the maximum value for TRANS_ID in table A is lower than the minimum value for TRANS_ID in table B. Because of the non-overlapping ranges for TRANS_ID, the Oracle CBO would assign cardinality of 1 (Lewis, 2006) for the query below, regardless of the number of records and column selectivity.

```
select      *
from        a , b
where       a.trans_id = b.trans_id
```

The problem can be mitigated by artificially expanding the column ranges, significantly increasing the chances of overlap, which in turn would result in more reasonable projected cardinality numbers. The approach is shown in Figure 13.

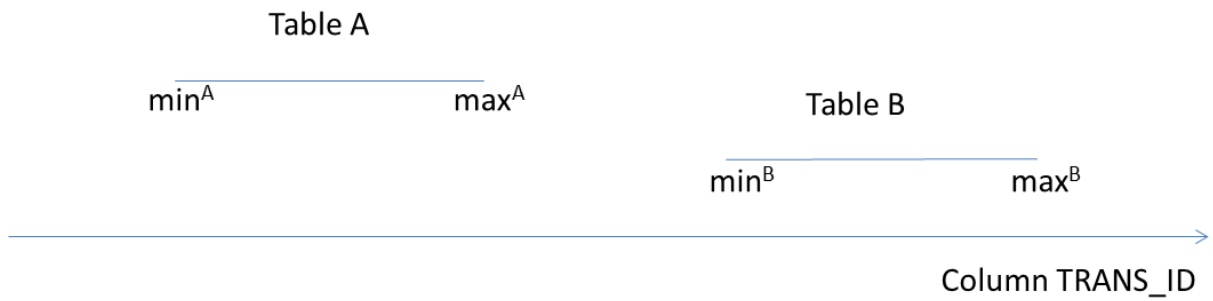


Figure 12. Non-Overlapping Column Ranges

One way for setting the expanded min/max values is to divide the min by 2 and double the max for columns with numerical data. Date/timestamp columns can be expanded by subtracting 365 days from the min value and adding 365 days to the max value. The constants used could be adjusted according to the specific business purposes. The stats manipulations above would reduce a little bit the quality of the single table cardinality/selectivity estimates, but they would significantly improve the quality of the join cardinality/selectivity estimates.

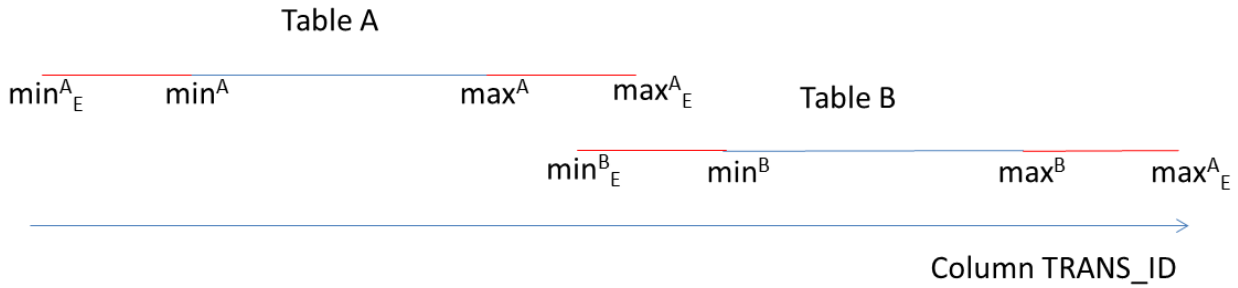


Figure 13. Overlapping Column Ranges - Manually Expanded Min/Max Values

Follow the Change

Keeping table statistics in sync with the content of the respective table at all times is a great way to achieve optimal performance. Oracle provides two major techniques to accomplish that – dynamic sampling and explicit statistics gathering after a significant data change using DBMS_STATS. While powerful, each of those two methods has limitations. To overcome them, I introduce JUST_STATS, a novel PL/SQL custom package that could enable us to solve even the most difficult performance issues related to volatile tables.

Dynamic sampling is on-the-fly statistics gathering that is triggered by hard parsing. The gathered statistics are used for generating the execution plan of the query that triggered the dynamic sampling, but are not persisted in the DD. Dynamic sampling with default level is quite easy to set up. All that is needed is for the table stats of the volatile table to deleted and locked. Since dynamic sampling affects performance only, its implementation does not require functional testing. Some off-the-shelf application may have challenges implementing dynamic sampling because it requires each SQL to be hard parsed. A great solution to that problem is utilizing VPD (virtual private databases) to force hard parsing without changing any application code (Geist, 2009) Dynamic sampling scans the table and gathers on-the-fly statistics every time a SQL statement is fired against a table set for dynamic sampling. That is a reasonable approach when there are only a couple select statements after a data modification. If a volatile table set for dynamic sampling is loaded once and selected many times before the next load or modification then the resources consumed by the dynamic sampling for each select statement are wasted. It makes no sense to constantly gather statistics through dynamic sampling while the table's content does not change.

Explicitly gathering statistics after a significant data change in a volatile table is another way we can keep the statistics in the DD and the content of the volatile table in sync. Oracle's DBMS_STATS package provides rich functionality and plenty of options to gather statistics efficiently. A drawback of this method is that it requires application code changes. All code fragments where a volatile table is modified have to be reviewed, and if appropriate, a statistics gathering statement added. Another significant challenge is that most procedures in DBMS_STATS package issue an implicit COMMIT. Until Oracle 12c, gathering stats after a data change was not scalable across multiple sessions, because the different sessions would overwrite each other's statistics. Session-private statistics for global temporary tables (GTT), an Oracle 12c feature, enable us to gather session specific statistics for GTTs without affecting the other sessions that use the same table. This great new feature greatly improves our ability to handle volatile tables in multi-user environments.

So, how big a deal is an "additional" COMMIT? The major purpose of a transaction in Oracle is to ensure data integrity. The commit points drive what data can be seen by other sessions and how easy it would be to recover in case of failure. A transaction should be committed when it must and never before (Kyte, 2010). Commit, implicit or explicit, is an important SQL statement that should not be used lightly. Issuing COMMIT for purely non-functional purpose, such as gathering statistics, is therefore not desired.

Granted, there are reasons for that behavior. DBMS_STATS package is treated as a DDL, and as such it requires a COMMIT to minimize the length of DDL locks (Kyte, 2010). Some in Oracle Support believe that the COMMIT issued by DBMS_STATS is not a problem because statistics should be gathered only on changes that have been committed. System testing, an essential part of almost any database project, could consume substantial time and resources. The purpose of non-functional testing is to verify that a non-functional change, such as changed init.ora parameter or a new index, produces the expected results. Since non-functional changes do not affect the application logic in any way, and are often easily reversible, they are less risky for the enterprise. There are number of great products, such as Oracle Real Application Testing and HP LoadRunner, that can automate non-functional testing. Functional testing, on the other hand, is usually human resource intensive. Only qualified professionals can accurately estimate the functional effect of a code or data change, making functional testing a slow and expensive process.

Even though statistics gathering within application code is supposed to affect execution plans only, a typical non-functional change, it still requires functional testing because of the implicit COMMIT it issues. To overcome those problems, I propose JUST_STATS PL/SQL package – a partial functional equivalent to DBMS_STATS package, but without the implicit COMMIT. The new package allows us to gather statistics without the fear of changing the application functionality. It opens the possibility to gather statistics in certain types of database triggers, an enormously powerful feature when dealing with volatile tables in a system that does not allow change of application code.

JUST_STATS is a limited version of DBMS_STATS. At this time, it has only the two most widely used procedures – GATHER_TABLE_STATS and GATHER_INDEX_STATS. Frequency histograms are implemented according to MOS (n.d. b) and height balanced histograms are implemented according to J. Lewis (2010). At this time, the package works with most widely used data types. Since JUST_STATS manipulates the DD using an autonomous transaction, it does not roll back the changes to the DD when the main transaction is rolled back. This weakness could be corrected with proper exception handling.

Figure 14 illustrates the architecture of JUST_STATS package. The procedures in the package issue select statements to obtain all relevant statistics, including number of table records, number of distinct column values, etc. The raw statistics data is saved into package variables, typically table of records. After all statistics are computed, an autonomous transaction reads the statistics from the package variables and writes them into DD.

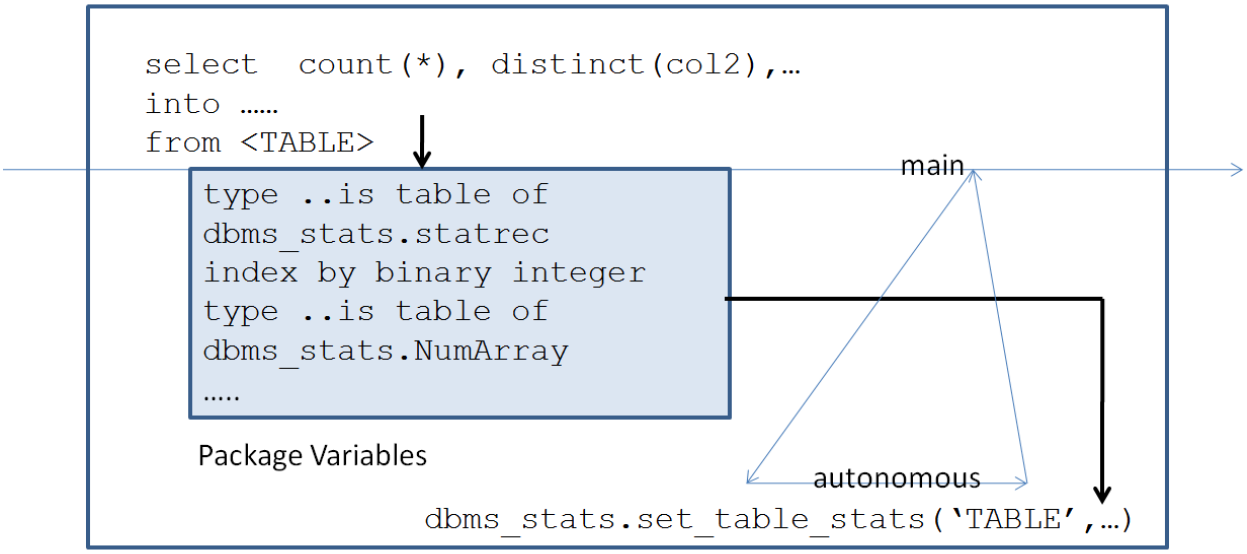


Figure 14. Inside JUST_STATS Package

JUST_STATS allows us to radically broaden our options when dealing with volatile tables. Since triggers are automatically invoked after any data change, they are a great place to gather statistics for volatile tables when application code change is not an option. Gathering statistics after any DML change can be easily accomplished with the following code:

```
create or replace trigger cust_stats
after insert or delete or update on <TAB>
begin
   just_stats.gather_table_stats('<USER>','<TAB>');
end;
```

While simple, the above trigger implementation for gathering statistics is not suitable for all cases. Sometimes, a table gets modified by statements that change few records as well as statements that change many. Since JUST_STATS consumes resources when gathering stats, automatically invoking it after every DML can be wasteful. Figure 15 illustrates how we can use before statement, after row and after statement table triggers to implement sophisticated and efficient statistics gathering solutions.

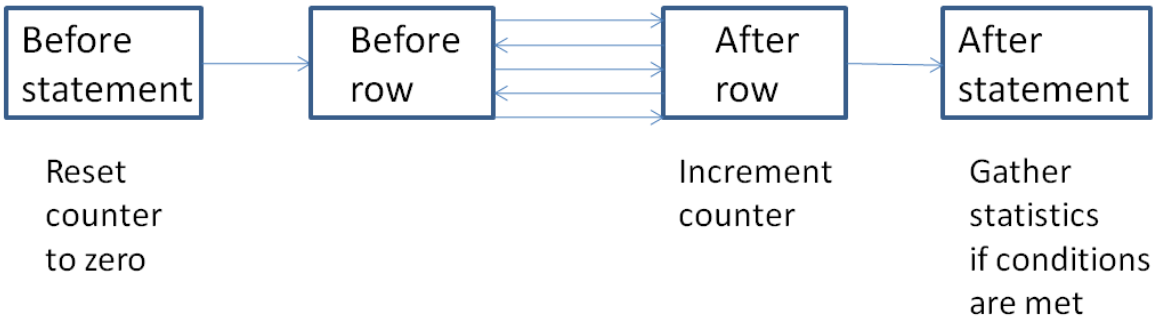


Figure 15. Customizations for Stats-gathering in Triggers

The number of records affected by a change is one important metric that we are going to track in this example. The number is going to be stored in a package variable.

```
create package stats_aux as
   cnt number;
end stats_aux;
```

The before statement trigger is fired once per statement, so it can be used to reset that counter.

```
create or replace
trigger stats_cnt_reset
before insert or delete
or update on <TABLE>
begin
   stats_aux.cnt:=0;
end;
```

The after row trigger is fired for every changed record, so it can be used to count the number of changed records. Introducing a row level trigger, even as simple as the one below, would inevitably affect DML performance. In most cases, this is a small price to pay for the ability to efficiently maintain accurate statistics of a volatile table.

```
create or replace
trigger stats_cnt_increment
before insert or delete
or update on <TABLE>
for each row
begin
  stats_aux.cnt:=stats_aux.cnt+1;
end;
```

Finally, the decision whether or not to gather statistics can be done in the post statement trigger, since it is fired only once per statement, after all changes are done. The trigger below compares the number of records in the table according to the DD with the number of records changed by the statement. It forces statistics gathering only if the statement changed more than 10% of the records. It is important to note that since the decision to gather statistics is made in PL/SQL code in a table trigger, there is enormous flexibility about the conditions that would trigger statistics gathering. We are able to base this algorithm on the current time or on the time since the last statistics gathering. We can achieve unparalleled customization by using session attributes and creating auxiliary statistics tables.

```
create or replace trigger cond_stats_gather
after insert or delete or update on <TAB>
declare
dd_cnt number;
begin
  select num_rows
  into dd_cnt
  from user_tables
  where table_name = '<TAB>';
  if stats_aux.cnt*10 > dd_cnt then
       just_stats.gather_table_stats('<USR>','<TAB>');
 end if;
end;
```

Gathering table and index statistics in triggers has challenges on its own. Since statistics are gathering on not-committed data, other sessions would have access to the newly gathered statistics before they are able to see the data those statistics were based upon. A great way to accommodate that discrepancy is to use the robust execution techniques outlined earlier in the paper. A simple implementation of such techniques would be gathering statistics only after inserts and updates, but not after deletes.

The need for such custom solution is somehow reduced in Oracle 12c due to the newly introduced "Online Statistics Gathering for Bulk Load" feature. Oracle 12c would automatically gather stats as part of CTAS and INSERT AS SELECT statements that use direct load. No index or histograms stats would be gathers though.

JUST_STATS, a PL/SQL package I designed, implemented and tested, is available for free. Even though I have successfully used it on hundreds of tables, I accept no liability or responsibility for JUST_STATS package, nor would I be able to support it. Use it at your own risk!

If you believe that there should be an option or parameter in Oracle's DBMS_STATS package that would allow gathering statistics without issuing a COMMIT, please contact Oracle support and request that bug/enhancement# 12897196 is promptly resolved. Let's bring these exiting new techniques to the mainstream!

Conclusion

Volatile tables present unique performance challenges. There are two major ways to address this problem – by reducing or eliminating the volatility, or by managing it. This paper shows how we can reduce table volatility by database refactoring, and what the consequences would be. The two main methods of managing volatile tables – *robust execution plans* and *follow the change* are reviewed in detail. Adaptive statistics locking, an innovative way to address some of the shortcomings of locking stats, is presented. The paper also introduces JUST_STATS package, a functional subset of DBMS_STATS that does not issues COMMIT, and gives guidance of its use in managing statistics of volatile tables.

References

Dehnad, K. (1989). *Quality Control, Robust design and the Taguchi Method* (pp. 241,270). Pacific Grove, CA :Wadsworth & Brooks/Cole

Geist, R. (2009). How to force a hard parse. Retrieved from http://oracle-randolf.blogspot.com/2009/02/how-to-force-hard-parse.html

Kyte, T. (2010). *Expert Oracle Database Architecture* (pp. 226,285). New York, NY:APRESS.

Lewis, J. (2010). Fake Histograms. Retrieved from http://jonathanlewis.wordpress.com/2010/03/23/fake-histograms/

Lewis, J. (2006). *Cost-Based Oracle Fundamentals*. New York, NY:APRESS

My Oracle Support. (n.d. a). **Best Practices for Automatic Statistics Collection [ID 377152.1].** Retrieved from https://support.oracle.com/

My Oracle Support. (n.d. b). **Setting histogram statistics on user-defined columns (based on type) [ID 228632.1].** Retrieved from https://support.oracle.com/

Osborne, K. (2013). **Adaptive Optimization, *Hotsos Symposium, Dallas, TX*** Retrieved from http://kerryosborne.oracle-guy.com/papers/12c_Adaptive_Optimization.pdf

ORACLE ACCELERATION

YEAH, IT'S KIND OF LIKE THAT FAST!

- Accelerate Oracle databases up to 10x
- Deploy with minimal disruption to operations
- Extend the life of your IT infrastructure

Put Your Big Data on the Fast Track.

The GridIron Systems TurboCharger™ data acceleration appliance seamlessly integrates into your existing IT environment without changes to applications, databases, servers, storage or operational processes.

Learn more at **www.gridironsystems.com/oracle**.

GRIDIRON
Big Data. Fast

NYOUG 2013 Sponsors

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

Axxana (www.axxana.com)
datAvail (www.datavail.com)
Dell Software (www.dell.com)
Oracle (www.oracle.com)

Contact Caryl Lee Fisher (execdir@nyoug.org) for vendor information, sponsorship, and benefits