Mike Dean  
All About Oracle Auditing

Mark W. Farnham  
Systematically Avoiding Deadlocks

John Jay King  
Planned Outages? No Thank You!

Peter Koletzke  
ADF: On-Ramp

Board Focus - John Peterson  
Comments From A Training Days Presenter
To explore how we can help your organization, contact Rolta AdvizeX at 800-220-8454 or visit Rolta AdvizeX online at www.advizex.com

Rolta AdvizeX brings you a complete solution for all your application, data, and infrastructure needs. Our customers use technology to drive innovation and achieve their most important business goals because we first help them focus on a strategy that leads to success, then bring the data, applications and infrastructure expertise needed to create and sustain that success.

#CloneAttack

What’s in it for you?

Going to RMOUG Training Days 2015? Try Delphix #CloneAttack free for 90 days. Install your own ready-to-use virtual machines and get hands-on experience with this important new capability.

Delphix experts will be ready to assist, demonstrate, or answer your questions in the lab at the rear of the exhibitor hall. Find out more about #CloneAttack on Twitter or contact us at cloneattack@delphix.com

#CloneAttack, Wed-Thu, February 18-19, 2015 at RMOUG Training Days 2015 in Denver, CO.
NEWSLETTER DIRECTOR
Vincent Giasolli
NewsletterDir@rmoug.org

Graphics & Printing by
PML Consulting
pat@patvanbuskirk.com

SUBSCRIPTIONS
Heidi Kuhn
admin@rmoug.org

CONTRIBUTING WRITERS
Mike Dean
Mark W. Farnham
Iggy Fernandez
John Jay King
Peter Koletzke
John Peterson

SQL>Update is published quarterly
by
Rocky Mountain Oracle Users Group
4869 S Coors Ct
Morrison, CO 80465
(303) 929-6266
Email: admin@rmoug.org
www.rmoug.org

Please submit advertising and editorial mate-
rial to the NewsletterDir@rmoug.org. RMOUG
reserves the right to edit, revise or reject all mate-
rial submitted for publication. Liability for errors
in ads will not be greater than the cost of the
advertising space. ©2014 Rocky Mountain Oracle
Users Group

monthly features
4 Letter From Our President
5 Stan Yellott Scholarship Fund
5 Scholarship Recipient - Alex Barton
28 Board Focus - John Peterson
30 RMOUG Calendar of Events
30 Commends From a Training Days
   Presenter - Iggy Fernandez
31 May 2015 QEW

On the Cover:
Cover photo by Vincent Giasolli. In addition to
being the Vice President and Newsletter Director for
RMOUG, one of Vincent’s favorite hobbies is photog-
raphy. This photo was taken along I-70 between
the Continental Divide and Idaho Springs using a
Nikon Coolpix AW120, his favorite travel camera.

features

6 ADF: On-Ramp
   by Peter Koletzke
   What You Need To Know To Use ADF

16 Planned Outages? No Thank You!
   by John Jay King
   Edition Based Redefinition in 12c

20 All About Oracle Auditing
   by Mike Dean
   How To Use The Audit Trail

24 Systemically Avoiding Deadlocks
   by Mark W. Farnham
   Achieve Superior Performance
From The President

John Peterson

“I’m convinced that about half of what separates successful entrepreneurs from the non-successful ones is pure perseverance.”

Steve Jobs

“Make a Difference” Rich Niemiec

What is important now?

Why Rocky Mountain Oracle Users Group?

As I write this, 2014 is rapidly moving to a the grand final events, the holidays, a time to remember an interesting year and a time to think about the future.

Oracle is important now, the company the products, the people, the careers of those who use Oracle Technology. This is a great place to be involved.

Focus on the future... what is going to happen in 2015 and beyond? Will it be some great idea that has turned into a technology or product we can actually use and help our lives and businesses?

There is an interesting list of possibilities, for your consideration,

Just what is the Internet of Things all about? How about Big Data? ...or NoSQL ...or Spark. Is there going to be really useful technology that will allow us to move into a production environment quickly? Or is that technology already here today?

Great new hardware and components are heading our way FAST.

What if performance tuning is as simple as plugging in a new appliance that makes those annoying old “patched to the nines” software package jump for joy.? Does this stuff work as advertised?

Back to Why RMOUG? Technology user groups are great places to get past some of the hype and into the reality. When a great user group is formed and is supported year after year, you have a place to find out just how real the new technology is, who is using it, how they are using it, where, when, why and what the new good stuff is really good for.

Coming to the Rocky Mountain Oracle Users Group in 2015 is our great international Oracle Users Group event called Training Days. This is a kickoff conference for the world of Oracle and the users of all Oracle Software and Hardware, This is where experts from around the world come once a year and share what they know and why Oracle stuff is so awesome!

RMOUG Training Days is just around the corner and this conference is filled with local as well as worldwide Oracle experts, providing over 100 world class presentations over a three day period in February. It is a good place to catch up with friends and make new ones – people involved with all things Oracle. Come see what the buzz is this year. You will meet new experts, see new products, get your hands on the stuff you have been reading about. We are all busy manufacturing software, project documents, manning the battle stations of oracle support or creating and designing new software to solve complicated problems every day. We need a day or two to pause, oil the machines, sharpen the saws, regenerate the excitement and find out something new. These are only a small sample of Why RMOUG.

We are looking forward to see you at Training Days and our Quarterly Education Workshops in the very bright future of the world of Oracle.

We are the people who use these amazing technologies and product.

Be well and Imagine Joy,

John Peterson
President Rocky Mountain Oracle Users Group
RMOUG Scholarship Mission

To provide educational opportunities to members of the organization about the information technology industry in general, and in particular the technology of Oracle Corporation to include databases, storage, networking and application development, specifically the products and services of the Oracle Corporation.

To collect, post and distribute information about Oracle technologies and other related technologies to members.

To provide members with the ability to help their peers maximize their knowledge and skills working with products in information technology and in particular Oracle products.

To provide a consolidated channel of communication, conveying needs, concerns, and suggestions, for members of the organization, to Oracle Corporation and other vendor corporations involved with Oracle related technology.

To encourage members to present their information technology experiences using Oracle and other products and services.

To provide a consolidated channel of communication between members of the RMOUG and other communities in related information technology industries.

To promote educational opportunities for students of information technology through directed funding and services for educational purposes.

RMOUG is committed to supporting others in the pursuit of technical knowledge.

The Scholarship Fund started in 2001 to encourage future IT professional in their efforts to broaden their knowledge. In 2007, RMOUG voted to rename the scholarship fund to honor the memory of Stan Yellott. Stan was a long time member of RMOUG where he supported the user community by serving on the RMOUG board. Stan focused on expanding Oracle educational opportunities. Stan’s vision was to include high school and college students as the next generation of IT professionals.

For Details, Visit the RMOUG Website

www.rmoug.org

Stan Yellott Scholarship Fund

Congratulations to Alex Barton for being chosen to receive a scholarship in the amount of $750 from the Stan Yellott Scholarship Fund. As a scholarship recipient, Alex also received a complimentary guest registration to our annual conference, Training Days 2015, to be held February 17-19, 2015 at the Colorado Convention Center. We plan to recognize Alex for his hard work and dedication during the conference.

Alex was granted the scholarship because of his passion for technology which was discovered through classes covering Oracle, Java, and C++ programming at Pine Creek High School. Alex wrote in his application,

“In high school, I grew a love for programming and technology, a drive to achieve, and most importantly I found the career field I want when I graduate from college. Becoming a DBA is my goal career after high school. My mom is a database administrator and I really enjoy the type of work she does. This scholarship will help me stay focused on my studies during the school year. I am a driven individual because I was able to balance my academics with sports, music, and programming throughout high school. An inspiring event was when I went to an RMOUG Training Days conference in 2013. I sat by Tim Gorman at lunch and talked to him a little about RMOUG. During the conference I learned that I want to be part of the RMOUG community and pursue my passion with programming. I would greatly appreciate this scholarship to help me work towards achieving my goals of getting my computer science degree and having a career working in technology.”

We are pleased to be able to help young students pursue a career in technology. The Stan Yellott scholarship provides a small reward for sharing your passion with our group and your enthusiasm encourages each of us to continue to mentor future technologists. Congratulations Alex, and thank you for sharing the RMOUG spirit with us! The next deadline for scholarship applications is March 15, 2015. More information on our webpage.
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 article.)

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 article 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 article 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 article 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

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)
What is ADF?

JDeveloper and ADF

JDeveloper 11g 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 11g to support all requirements of the new Fusion Applications.

Although this article focuses 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/afdevessentialsfaq-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.)

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 article (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.

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:

```xml
<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>
    </Attribute>
    <Properties>
        <SchemaBasedProperties>
            <LABEL
                ResId="hr.model.Departments.DepartmentId_LABEL"/>
        </SchemaBasedProperties>
    </Properties>
</Entity>
```
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.**

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.

```xml
<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"
    PagerIterMode="Full"
    UseGlueCode="false">
    ...
    <ViewAttribute
        Name="EmployeeId"
        IsUpdateable="false"
        IsNotNull="true"
        PrecisionRule="true"
        EntityAttrName="EmployeeId"
        EntityUsage="Employees"
        AliasName="EMPLOYEE_ID"/>
</ViewObject>
```

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 (next column).

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:

```xml
<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>
```
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:

```xml
<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.

![Figure 4. Task Flow Diagram](image)

**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.

### 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. JSP defines higher-level items (called “components”) that add functionality to HTML. ADF Faces Rich Client (available in JDeveloper 11g and abbreviated hereafter as “ADF Faces”) is a set of JSP 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:

```xml
<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 article, 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.

**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.

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 in Figure 6:

**Figure 6. JDeveloper visual editor display of the Departments browse page**

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.

> 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:

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.

The Data Model area in this example defines view objects for DepartmentsView with a detail of EmployeeView (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:

```xml
<af:inputText
    value="#{bindings.DepartmentId.inputValue}"
```
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>
</bindings>
```

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 article 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 online 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.

### 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 article 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!

---

**A Note From**

**Kellyn Pot’Vin-Gorman**

**Your Training Days Director**

*I was first introduced to Training Days by the recommendation of a Senior DBA I worked with back in 2004. I was amazed at the education opportunity and quality of the sessions and have continued recommending the conference to others since then. There is nothing more valuable than word of mouth and we are going to reward that at Training Days 2015. If you recommend someone new to Training Days and they list your name on their registration form as the one that recommended them, we’ll reward you with a $25 Amazon gift card after the conference!*

**Fine print:** Limit 10 gift cards per member and registration must be for attendee who is new or hasn’t attended in at least five years.

**Thank you!**

Kellyn
Planned Outages? No Thank You!
Edition Based Redefinition in 12c

by John Jay King, King Training Resources

Introducing Edition Based Redefinition
Oracle 11gR2’s Edition Based Redefinition (EBR) was trumpeted as the “killer feature” by many. Oracle improved EBR with Oracle 12c making it nearly irresistible. This article introduces Edition Based Redefinition (EBR), database objects that support EBR, and how EBR might be useful.

What is the problem EBR is meant to solve?
Planned outages are the problem. Many Oracle database applications have PL/SQL Packages, Procedures, and Functions. When PL/SQL is updated and re-installed in the production system an outage is usually needed.

What if you could update PL/SQL, and maybe your tables without scheduling an outage? EBR allows “Online Application Upgrade” so applications need not be taken down to apply upgrades. Online Application Upgrade means existing systems and users continue uninterrupted as upgraded code is introduced while selected users may use new code immediately (probably developers and testers). Both sets of code are in place in production; but a given session sees only code included in the session’s context. When all is ready, new code is enabled for everyone with no scheduled downtime. Oracle E-Business Suite 12.2 uses EBR to drastically reduce planned outages.

EBR uses database objects added or modified with Oracle 11gR2 and refined in Oracle 12c. EBR main object is a non-schema “edition” you might think of as a “release” or “version” of your applications. Editions include all “editionable” PL/SQL, views, and synonyms in the database for all schemas having “editioned” objects; each schema might have multiple versions of code belonging to different editions.

What is Editionable?
EBR manages editions (versions/releases) of “editionable objects” including:
- PL/SQL objects of all kinds
- Synonyms
- Views
- SQL translation profile (new in Oracle 12c)

Oracle 11gR2 non-editioned objects may NOT depend upon editioned objects. Oracle 12c Materialized Views and Virtual Columns may be dependent upon editionable objects making EBR easier for many shops to implement. Oracle 12c “editionable” objects may be marked “NONEDITIONABLE” to allow inclusion in non-editioned objects.

EBR is “Free”
EBR is safe, secure, and part of Oracle 11gR2 (EE and SE) and 12c; built in to the database, and without additional licensing cost. However, EBR might require considerable design investment, particularly if Editioning Views and Cross-Edition Triggers are involved.

EBR Database Objects
Oracle 11gR2 database objects are referenced internally using Edition Name, Schema Name, and Object Name. Edition name is never referenced in program SQL; edition is supplied by the current session context.

<table>
<thead>
<tr>
<th>Edition</th>
<th>Version representing editioned objects at a point in time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editioning View</td>
<td>Optional object type representing editioned view of table data</td>
</tr>
<tr>
<td>Cross-Edition Triggers</td>
<td>Optional PL/SQL synchronizing parent and child edition changes</td>
</tr>
</tbody>
</table>

Oracle database catalog view support editions:
- v$session (session_edition_id)
- dba_editions
- dbaEdition_comments
- dbaEditioning_views
- dbaEditioning_views_ae (all editions)
- dbaEditioning_view_cols
- dbaEditioning_view_cols_ae
- dba_triggers
- dba_objects (edition_name)
- more

Editions
Edition is a database object.

Each database has a default edition; the original default edition is “ORA$BASE” for all databases. New editions are created “as child of” an existing edition inheriting all objects from the parent edition:
create edition RELEASE2 as child of ORA$BASE;
create edition RELEASE3 as child of RELEASE2;
Each edition has exactly one parent (except ORA$BASE) and may have only one child. Oracle doesn’t duplicate code from prior editions; new editions “inherit” existing code. Existing editions (“pre-upgrade”) include editioned objects before changes happen. Changes in the new edition’s editioned objects FOR ALL SCHEMAS are stored as part of the child edition (“post-upgrade”). EBR Editions are non-schema objects with no owner (docs say “not owned”, ALL_OBJECTS says “SYS”).

**Edition Management**

Editions impact an entire database; so, only selected users should manage editions (probably a subset of the DBA staff) since editions cover ALL SCHEMAS.

To create a new edition requires “CREATE ANY EDITION” privilege.

create edition yyy as child of xxx;

Editions may be dropped (DROP EDITION…) but dropping might cause problems since editions do not copy definitions but inherit them. After upgrades have been applied and tested in a new edition, if the edition has errors it might be useful to DROP the latest edition. Administrators retire editions by revoking “use” privileges. Once an edition is no longer used: cross-edition triggers may be removed (probably should be) and superfluous columns may be dropped. Perform object “cleanup” as described on OTN’s EBR page (see “Self-contained Edition-based Redefinition Exercise”). Drop editions only as documented in the Advanced Application Developer’s Guide.

**EBR Access**

Edition access is user based; developers and users must be granted use.

grant use on edition yyy to someuser;

PL/SQL, Synonyms, and Views are editionable, a schema (user) must be “editions enabled” to have objects editioned:

```
alter user someschema enable editions;
```

Editions-enabling is IRREVERSIBLE. All editionable objects belonging to the user, both existing and yet to be created are editioned. Oracle 12c editions-enabled users may mark editionable objects “NONEDITIONABLE” to exclude them from editioning. Oracle 12c allows edition enabling a user for specific types.

```
alter user someschema enable editions for sometypel,sometype2;
```

Oracle flags users as editions-enabled in the DBA_USERS view EDITIONS_ENABLED column (“Y” or “N”). A schema object of a noneditionable type is always non-editioned, even if its owner is editions-enabled. In Oracle 11gR2 it is often useful to create a non-edition-enabled schema owning editionable objects that should not be editioned. Oracle E-Business Suite 12.2 uses two schemas: APPS and APPS_NE (Not Editioned).

**Edition Use is Session-based**

Sessions access one edition at a time via: database default edition, host-system environment variable of session, setting edition in TNS service definition, specification of edition at user login, or ALTER SESSION.

To set database default edition (grants PUBLIC access to edition implicitly):

```
alter database default edition = my_edition_name;
```

A session’s edition may be set using an environment variable.

```
ORA_EDITION=my_edition_name
```

A service definition may override the database default edition. The DBMS_SERVICE package’s CREATE_SERVICE and MODIFY_SERVICE procedures have EDITION and parameter settings.

```
BEGIN
  DBMS_SERVICE.modify_service(
    service_name => 'myservice',
    edition => 'my_edition_name',
    modify_edition => TRUE);
END;
```

Users may name a session when connecting.

```
connect ebr_user/ebr_user edition=my_edition_name;
```

Users with ‘ALTER SESSION’ privilege may switch editions:

```
alter session set edition=my_edition_name;
```

To check session edition:

```
select sys_context('Userenv','Current_Edition_name')
CurrentEdition
from dual;
```

**Three Ways to Use EBR**

Edition-Based Redefinition (EBR) use has three-levels of complexity:

1. EBR used to install new versions of PL/SQL, Views, or Synonyms
2. EBR used as above, plus, Editioning Views changing user’s view when column definitions must be added, modified, or removed,
3. EBR used as above (1&2) when multiple synchronized editions of tables need to be “live” simultaneously; adds Cross-Edition Triggers.

**PL/SQL, Views, Synonyms Only**

If only PL/SQL, Views, and Synonyms are changing little or no preparation is required:

- create new edition,
- make/test changes to desired PL/SQL, Views, and Synonyms,
- post tested changes to production system,
- test new edition by allowing use to selected users,
- make new edition available to all users.

**Reducing Downtime when Tables Change with EBR**

If tables might have column definitions added, removed, or altered between editions and EBR is desired, Editioning Views are needed. Editioning Views ensure that each edition’s users see only data relevant to the current edition. If it is necessary to synchronize data between editions, Cross-Edition Triggers are required.

**Editioning Views**

Editioning View preparation (or “readying”) will probably
require at least one outage. First, the base table must be renamed. Next, an Editioning View is created using the original base table name and listing columns in the underlying table. Once the Editioning View exists using the original table name, all existing code will work without change. Editioning Views provide potentially different projections of base table data for each edition. Each edition sees only its own columns.

Editioning Views represent desired base table columns unaltered and unfiltered; queries may not use: WHERE, joins, functions, expressions, operators, Group By, Order By, Distinct, or anything causing view data to misrepresent the underlying table data. Editioning Views act like tables and may have triggers and other table-like features. ALL application code references Editioning Views rather than the base tables.

alter table myemp rename to myemp_table;
create or replace editioning view myemp as
    select empno,ename,job,mgr,hiredate,sal,comm,deptno
    from myemp_table;

Sometimes renaming base table columns is useful. Suppose the original edition’s FIRST_NAME column is VARCHAR2(10) and you upgrade it to VARCHAR2(50); you might add a base-table column FIRST_NAME_50 and rename the original FIRST_NAME_10; then use FIRST_NAME in both editioning views. Existing code (parent edition) will use the shorter column and new code (child edition) will use the longer column. If both editions might be in use at the same time, UPDATES and INSERTS might require some synchronization via Cross-Edition Triggers.

Editioning Views merely PROJECT column data so the optimizer converts all activity to use the underlying table. SQL referencing Editioning Views gets EXACTLY the same execution plan as SQL using the base table. There is no additional performance cost (other than statement parsing) involved with Editioning Views.

Forward and Reverse Cross-Edition Triggers

Cross-Edition Triggers are needed when Editioned Views representing changes to underlying table definitions raise potential synchronization problems. Unlike Editioning Views, Cross-Edition Triggers will probably impact performance and may be logically complex. The idea is to allow safe writing to new and/or modified columns in editions by keeping changes synchronized:
- Changes in Parent propagate to Child (Forward),
- Changes in Child propagate to Parent (Reverse).

```
alter table myemp_table
add (avgjobpay number(9,2));
create or replace editioning view myemp
    select empno,ename,job,mgr,hiredate,sal,comm,deptno
    from myemp_table;
create or replace function getavgjobpay
    (injob in myemp.job%type) return number
    /* other code */
--
create or replace trigger myemp_table_avgpay
    before insert or update of sal on myemp_table
    for each row
    forward crossedition
disable
```

begin
    :new.avgjobpay := getavgjobpay(:new.job);
end;
alter trigger myemp_table_avgpay enable;

What About New Data?

If a base table has new and/or altered columns, data must be moved into them. In the past mass changes might have involved locking problems. Oracle improved ALTER TABLE to make most column additions non-blocking and dependency tracking was improved to allow “fine-grain dependency.” Oracle added the DBMS_PARALLEL_EXECUTE package to make changes in smaller chunks also limiting locking issues.

```
declare
    cptr number;
    retv number;
begin
    cptr := dbms_sql.open_cursor();
    dbms_sql.parse(c=>cptr,language_flag=>dbms_sql.
    native,
        statement=>'update myemp_table set empno =
    empno',
    apply_crossedition_trigger=>'MYEMP_TABLE_ AVGPAY');
    retv := dbms_sql.execute(cptr);
    dbms_sql.close_cursor(cptr);
    commit;
end;
```

Post-upgrade edition processing must not interfere with pre-upgrade processing (don’t break existing code!).

Oracle 12c EBR Enhancements

Oracle 12c fixes some 11gR2 limitations, notably: Materialized Views may use editioned resources, Virtual Columns may use editioned functions, Public synonyms may be editioned or noneditionable, and User objects may be noneditionable.

Materialized Views are not-editionable; but, Oracle 12c allows them to depend upon editioned objects using new clauses: EVALUATE USING and ENABLE QUERY REWRITE UNUSABLE BEGINNING WITH or UNUSABLE BEFORE CURRENT EDITION, EDITION XXX, or NULL EDITION (NULL eliminates editing).

```
create materialized view myemp_summary
    /* other code */
evaluate using current edition
enable query rewrite
    unusable before edition myedition3
    /* other code */
```

Oracle 12c non-editioned Virtual Columns may depend upon editioned objects by adding EVALUATION EDITION, UNUSABLE BEGINNING WITH, and UNUSABLE BEFORE clauses using CURRENT EDITION or EDITION XXX or NULL EDITION to limit Virtual Columns “visibility.”

```
create table testit
    (/* more columns */
    totpay number(8,2) as (get_totpay(sal,comm))
    evaluate using current edition
    unusable before edition myedition4,
    /* more columns */
    );
```
EBR Adoption

If EBR is being used for PL/SQL, Synonyms, and Views only it’s pretty simple, define your standards and just do it.

If EBR is more complex and involves Editioning Views and maybe Cross-Edition Triggers EBR can be adopted all at once or it may be phased-in. Editioning View “readying” work will require an outage. Oracle suggests tables be replaced by Editioning Views in one step to reduce future outages and limit the work required to begin using EBR. Or, you may “ready” tables as needed probably requiring requires future outages as needs change. Who will be EBR Administrator(s)? How will you control/stage editions? PL/SQL only or Editioning Views and Cross-Edition triggers?

EBR support page on OTN:

Detailed documentation and on Oracle OTN’s web site:


Wrapping It Up

EBR reduces (nearly eliminates) planned outages due to Oracle application upgrades.

Congratulations! You may be approaching your LAST planned Oracle outage!

---

John Jay King is a partner in King Training Resources tailoring and conducting training since 1988 (www.kingtraining.com). John customizes courses in topics including Oracle, MAF, and ADF. John presents frequently at conferences including RMOUG, ODTUG, IOUG, UKOUG, and others. John is an Oracle Ace Director, member of the Oak Table Network, and serves on the AZORA board.
Many organizations keep their most sensitive and valuable information in an Oracle database. Properly implemented auditing, as part of a defense-in-depth approach, will help keep it secure. In this white paper, I will discuss the most important aspects of auditing; from basic configuration to advanced techniques using Fine-Grained Auditing (FGA). I will also discuss what to look for in the Audit Trail as well as changes in 11g that, if not addressed, can cause serious issues. And I will even tell you how the Audit Trail can be used to make life easier for the DBA.

What is Oracle Auditing and why do I need it?

Auditing is the monitoring and recording, either in the database or in OS files, of selected database actions, from both database users and non-database users. In certain industries, you may be required to implement auditing for regulatory compliance. For example, healthcare companies are regulated by HIPAA and publicly traded companies are subject to Sarbanes-Oxley, both of which have extensive auditing requirements. Auditing can also help to ensure accountability by tracking who does what and when in your database. Auditing can deter inappropriate behavior in the first place as people are less likely to break the rules if they know they are being watched. If you do have a security incident of some sort, audit data can be very valuable for investigation purposes. It also allows you to monitor database activities, for example logon activity, table usage, database changes etc. And finally, it can make life easier for the DBA. This may seem counter-intuitive but I will give some examples later on to illustrate this point.

Mandatory Auditing

Mandatory auditing happens automatically in every database and cannot be turned off. It will record when the database is stopped or started as well as record when a user logs on with either SYSDBA or SYSOPER privileges. This data is written to files in a location determined by the AUDIT_FILE_DEST parameter on Unix or to the Event Viewer on Windows. If you have this parameter set to an invalid directory or run out of space in the directory, you won’t be able to start the database or connect as SYSDBA.

```
SQL> alter system set audit_file_dest='/bogus'
scope=spfile;
SQL> shutdown immediate
SQL> startup
ORA-09925: Unable to create audit trail file
Linux-x86_64 Error: 2: No such file or directory
Additional information: 9925
```

Depending on how often users connect as SYSDBA and/or SYSOPER, mandatory auditing can generate a lot of files so keep an eye on the free space for this location. You should also schedule periodic reviews and purging of these files.

Auditing Actions by SYS

By default, the SYS user is exempt from all auditing policies, and this can leave a huge hole in your audit trail. You can change this behavior by setting the parameter AUDIT_SYS_OPERATIONS=TRUE. When that is set to true, Oracle will record ALL statements issued by SYS to files in a location determined by the AUDIT_FILE_DEST parameter on Unix or to the Event Viewer on Windows. Or if AUDIT_SYSLOG_LEVEL is set, then it will write to the Unix syslog. Obviously, if you login as SYS a lot then this can generate a lot of data. In my opinion, you should set this parameter to TRUE so you can monitor SYS activities.

AUDIT_SYSLOG_LEVEL

This parameter allows you to integrate the database audit trail with the Unix Syslog and even send it to a remote server using remote syslogd functionality. This can be used in highly secure environments to remove the audit data from its source system. It can also be useful to consolidate audit data from multiple databases into a single file. In order to configure this, you specify a facility clause and priority clause in the AUDIT_SYSLOG_LEVEL parameter and then create a corresponding entry in /etc/syslog.conf specifying a filename.

For example:

```
AUDIT_SYSLOG_LEVEL = ‘LOCAL1.WARNING’;
```

```
/etc/syslog.conf:
LOCAL1.WARNING /var/log/dbaudit.log
```

Standard Auditing

Standard auditing is enabled by setting the AUDIT_TRAIL parameter and is configured with the AUDIT/NOAUDIT commands. It can write its records to either the SYS.AUD$ table or to OS files depending on the value of AUDIT_TRAIL. The following are valid values for this parameter:

- NONE = disables standard auditing
- OS = writes audit records to an OS file
- DB = writes audit records to the SYS.AUD$ table in the database
- DB_EXTENDED = writes audit records to the SYS.AUD$ and includes the complete SQL statement that was executed along with any bind values
- XML = writes audit records to an OS file in XML format
- XML_EXTENDED = writes audit records to an OS file
in XML format plus records the SQL statement and bind values to SYS.AUD$.

The database will need to be restarted for it to take effect. With Standard Auditing, you can track Statements, Privileges and Objects

Privilege auditing:

```sql
audit select any table;
```

Statement auditing:

```sql
audit select table;
```

Object auditing:

```sql
audit select on SCOTT.SALARY;
```

You can specify if you want to track only failed actions or only successful actions or both. You can specify either “by session” or “by access” which determines the granularity at which the audit record is written. You can also track activities by certain users. The details of your auditing config can be found in dba_stmt_audit_opts, dba_priv_audit_opts and dba_obj_audit_opts. You turn off the auditing with the NOAUDIT command.

```sql
audit select table by session | by access;
daudit select table whenever not successful
```

Oracle recommends that you audit BY ACCESS and not BY SESSION in your AUDIT statements. The audit records generated through the BY ACCESS audit option have more information, such as execution status (return code), date and time of execution, the privileges used, the objects accessed, the SQL text itself and its bind values. In addition, the BY ACCESS audit option captures the SCN for each execution and this can help flashback queries.

You can audit the use of any system privilege. Privilege auditing does not occur if the action is already permitted by the existing owner and object privileges. Privilege auditing is triggered only if the privileges are insufficient, that is, only if what makes the action possible is a system privilege. For example, suppose that user SCOTT has been granted the SELECT ANY TABLE privilege and the SELECT ANY TABLE is being audited. If SCOTT selects his own table (for example, SCOTT.EMP), then the SELECT ANY TABLE privilege is not used. Because he performed the SELECT statement within his own schema, no audit record is generated. On the other hand, if SCOTT selects from another schema (for example, the HR.EMPLOYEES table), then an audit record is generated. Because SCOTT selected a table outside his own schema, he needed to use the SELECT ANY TABLE privilege. If Scott has been given direct grants on the other schema's table (as he should be), then he would not need SELECT ANY TABLE and would not be audited.

### Fine Grained Auditing

Fine-Grained Auditing (FGA) is an Enterprise Edition only feature and enables you to create policies that define specific conditions that must take place for the audit to occur, allowing you to monitor data access based on content. It provides granular auditing of queries, and INSERT, UPDATE, and DELETE operations. For example, a CFO must track access to financial records to guard against employee snooping, with enough detail to determine what data was accessed. It is not enough to know that SELECT privilege was used by a specific user on a particular table. Fine-grained auditing provides this deeper functionality.

In general, fine-grained audit policies are based on simple, user-defined SQL predicates on table objects as conditions for selective auditing. During fetching, whenever policy conditions are met for a row, the query is audited.

For example, you can use fine-grained auditing to audit the following types of actions:

- Accessing a table outside of normal working hours
- Logging in from a particular IP address
- Selecting or updating a particular table column

Fine-grained auditing records are stored in the `SYS.FGA_LOG$` table. To find the records have been generated for the audit policies that are in effect, you can query the `DBA_FGA_AUDIT_TRAIL` view. The `DBA_COMMON_AUDIT_TRAIL` view combines both standard and fine-grained audit log records. The `DBA_AUDIT_TRAIL` view contains standard Auditing records.

To create a fine-grained audit policy, use the `DBMS_FGA_ADD_POLICY` procedure. This procedure creates an audit policy using the supplied predicate as the audit condition. Oracle Database executes the policy predicate with the privileges of the user who created the policy. The maximum number of fine-grained policies on any table or view object is 256. Oracle Database stores the policy in the data dictionary table, but you can create the policy on any table or view that is not in the SYS schema. After you create the fine-grained audit policy, it does not reside in any specific schema, although the definition for the policy is stored in the `SYS.FGA$` data dictionary table. You cannot modify a fine-grained audit policy after you have created it. If you need to modify the policy, drop it and then recreate it.

The following example will create an FGA policy to record all updates to the SALARY column unless it is done by the owner of the table, MIKE. I am using the `SYS_CONTEXT` function to find information about the user and using that to determine if auditing should occur.

```sql
begin
  DBMS_FGA.ADD_POLICY(
    object_schema => 'MIKE',
    object_name => 'EMPLOYEE',
    policy_name => 'salary_change',
    audit_column => 'SALARY',
    audit_condition => 'SYS_CONTEXT(’USERENV’,’SESSION_USER’) <> ’MIKE’,
    enable => TRUE,
    statement_types => 'UPDATE',
    audit_trail => DBMS_FGA.DB + DBMS_FGA.EXTENDED);
end;
```

One advantage of FGA over Standard Auditing is that you don’t need to set the `AUDIT_TRAIL` parameter for it to work. This means that you can enable auditing without restarting the database.

### What should you be auditing?

In general, you should audit changes to the database (alter database, alter system), DDL, system/object privileges, logon/logoff
and unsuccessful operations. See the script at the end of this paper for my recommended audit settings.

More specifically, you should audit data that is sensitive and/or important to your organization. (salaries, classified data, financial info, etc). This requires you to understand your data and is where Fine Grained Auditing may be most appropriate.

What to Look for in the Audit Trail?

Unless you are looking for something specific, you basically want to keep an eye out for anomalies in the Audit Trail. Of course, what is an anomaly in one database may not be in another. Here are some things you may want to look for:

- DDL not during a scheduled build
- Activity outside of normal working hours
- Failed attempts to access data or exceed privileges
  - ORA-00942: “table or view does not exist”
  - ORA-02004: “security violation”
  - ORA-01031: “insufficient privileges”
- Excessive failed login attempts
- Login attempts by non-existent users
- Alter system/Alter database commands
- Unauthorized privilege/object grants
- Unsuccessful operations – (where aud$.returncode != 0)

11g new Auditing features

Oracle 11g introduced a host of changes to the Auditing functionality. One change that could catch you off guard is Oracle 11 now implements auditing by default by setting the AUDIT_TRAIL=DB whenever the Database Configuration Assistant (DBCA) is used to create a database.

The script that gets run by DBCA is $ORACLE_HOME/rdbms/admin/secconf.sql. To turn off these audit settings, run the undoaud.sql file in that same directory.

- A summary of the changes in Oracle 11:
- Beware of default auditing! (11.1)
- Changes to AUDIT BY SESSION (11.2.0.1) - can dramatically increase the size of the audit trail if you had been auditing by session.
- Separate records for LOGON, LOGOFF now. In Oracle 10, the LOGON record would be updated when the user logged off. Now there are two records.
- DB_EXTENDED is now DB,Extended (11.2.0.1)
- BY ACCESS now the default (11.2.0.2)
- Audit Trail Cleanup process has been improved with DBMS_AUDIT_MGMT (11.2.0.1)
- Caution when upgrading to 11G with auditing turned on. Refer to Note 1329590.1

Oracle Database 11g audits the following SQL shortcuts by default:

<table>
<thead>
<tr>
<th>ROLE</th>
<th>SYSTEM AUDIT</th>
<th>PUBLIC SYNONYM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABASE LINK</td>
<td>PROFILE</td>
<td>SYSTEM GRANT</td>
</tr>
</tbody>
</table>

In addition to Oracle 11 Default, I recommend the following AUDIT settings. These will capture most DDL and grants as well as failed SQL.

- audit not exists;
- audit select any table;
- audit select any dictionary;
- audit SELECT TABLE whenever not successful;
- audit INSERT TABLE whenever not successful;
- audit UPDATE TABLE whenever not successful;
- audit DELETE TABLE whenever not successful;
- audit table; (shortcut for create, drop, truncate)
- audit alter table;
- audit procedure;
- audit trigger;
- audit view;
- audit index;
- audit grant procedure;
- audit grant table;

Examples of Audit Trail Use for the DBA

There many times over the years when I have found audit data to be useful to me as a DBA. Here are some examples:

- Performance issues caused by excessive logon activity. Being able to quantify which users are connecting how often and for how long can be valuable when faced with performance issues caused by excessive logon activity. This is often the result of a misconfigured connection pool and can be easily fixed once identified.
- “Table Or View Does Not Exist” or “Insufficient Privileges” – which table? When an application cannot access a table because it doesn’t exist or insufficient privileges, it will usually just return the Oracle error to the screen. You can easily identify the table(s) involved and the reason for the error by looking in the Audit Trail.
- Who dropped/altered this table/index/procedure? Even in tightly controlled environments, occasionally you have a change occur that no one will own up to.
- Unhandled Oracle errors. When an application doesn’t gracefully handle Oracle errors, they will show up in the Audit Trail.
- Inappropriate system privileges. Someone was granted DBA in Production without proper approval – who is responsible?
Managing the Audit Trail

The first step to managing the audit trail is to move it out of SYSTEM into its own tablespace. In previous versions, this would be done manually:

```
create table audx tablespace "AUDIT"
    storage (initial 50k next 50k pctincrease 0)
    as select * from aud$ where 1 = 2
/
rename AUD$ to AUD$$
/
rename audx to aud$
/
create index i_aud2
    on aud${sessionid, ses$tid}
    tablespace "AUDIT" storage(initial 50k next 50k pctincrease 0)
/
```

Be careful when doing any type of maintenance on the AUD$ or RGBA_LOG$ tables that might cause them to get locked or otherwise unavailable if you are currently auditing. I have seen this happen on a database that is auditing logon/logoff activity and it quickly causes a bottleneck as no one will be able to login. You should turn off auditing with the NOAUDIT command before moving the table to another tablespace.

As of 11g, there is the DBMS_AUDIT_MGMT package that can be used to manage the audit data. Here is a 4-step process than can be used to manage your audit data.

**Step 1: Move audit tables to dedicated tablespaces**

```sql
DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_LOCATION(
    AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_DB_STD,
    AUDIT_TRAIL_LOCATION_VALUE => 'AUD_DATA');
```

**Step 2: Initialize the Cleanup process and set the Cleanup interval**

```sql
DBMS_AUDIT_MGMT.INIT_CLEANUP(
    AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_DB_STD,
    DEFAULT_CLEANUP_INTERVAL => 12 );
```

**Step 3: After reviewing/archiving the audit data, set the archive timestamp**

```sql
DBMS_AUDIT_MGMT.SET_LAST_ARCHIVE_TIMESTAMP(
    AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_DB_STD,
    LAST_ARCHIVE_TIME => sysdate-90);
```

By setting the LAST_ARCHIVE_TIME to sysdate-90, you tell the clean-up job to delete everything older than 90 days.

**Step 4: Create the Purge job**

```sql
DBMS_AUDIT_MGMT.CREATE_PURGE_JOB (AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_DB_STD,
    AUDIT_TRAIL_PURGE_INTERVAL => 12,
    AUDIT_TRAIL_PURGE_NAME => 'Audit Purge',
    USE_LAST_ARCH_TIMESTAMP => TRUE );
```

What is the performance impact of Auditing?

In my experience, Auditing does not impose a significant burden of overhead and its benefits far outweigh the costs. Oracle has published a really good white paper discussing this subject called Oracle Database Auditing: Performance Guidelines and includes the following charts. It makes sense that the most resource intensive type of auditing is DB, Extended as it records the entire SQL statement and bind values. Keep in mind that these tests were generating 200-250 audit records per second which is likely a lot more than most systems would actually generate.

**Overhead of Standard Auditing during TPC-C OLTP benchmark**

<table>
<thead>
<tr>
<th>Audit Trail Setting</th>
<th>Additional Throughput Time</th>
<th>Additional CPU Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>1.39%</td>
<td>1.75%</td>
</tr>
<tr>
<td>XML</td>
<td>1.70%</td>
<td>3.51%</td>
</tr>
<tr>
<td>XML, Extended</td>
<td>3.70%</td>
<td>5.26%</td>
</tr>
<tr>
<td>DB</td>
<td>4.57%</td>
<td>8.77%</td>
</tr>
<tr>
<td>DB, Extended</td>
<td>14.09%</td>
<td>15.79%</td>
</tr>
</tbody>
</table>

**Overhead of Fine Grained Auditing during TPC-C OLTP benchmark**

<table>
<thead>
<tr>
<th>Audit Trail Setting</th>
<th>Additional Throughput Time</th>
<th>Additional CPU Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>3.66%</td>
<td>4.35%</td>
</tr>
<tr>
<td>XML, Extended</td>
<td>4.52%</td>
<td>9.09%</td>
</tr>
<tr>
<td>DB</td>
<td>6.60%</td>
<td>11.11%</td>
</tr>
<tr>
<td>DB, Extended</td>
<td>9.01%</td>
<td>20%</td>
</tr>
</tbody>
</table>

Source: Oracle Database Auditing: Performance Guidelines

Conclusion

Oracle provides a wealth of functionality that can be used to audit various aspects of your database. When properly implemented and reviewed on a regular basis, auditing is an important and useful tool for securing your database and maintaining compliance with your organizational security requirements. It can also be useful to the DBA for troubleshooting purposes.

Appendix

Use these commands to implement my recommended set of AUDIT settings:

- `AUDIT ALTER ANY PROCEDURE;`
- `AUDIT CREATE ANY LIBRARY;`
- `AUDIT DROP ANY TABLE;`
- `AUDIT ALTER ANY TABLE;`
- `AUDIT CREATE ANY PROCEDURE;`
- `AUDIT DROP PROFILE;`
- `AUDIT ALTER DATABASE;`
- `AUDIT CREATE ANY TABLE;`
- `AUDIT DROP USER;`

...continued on Page 31
Systematically Avoiding Deadlocks to Achieve Superior Performance

by Mark W. Farnham, Rightsizing, Inc.

Introduction

Oracle does a wonderful job of detecting and managing deadlocks generated by applications. But you still have to resubmit any changes rolled back in untangling the deadlock. Better performance can be achieved by avoiding deadlocks in the first place. Establishing an enterprise wide order of tables and row order within tables helps minimize deadlocks. This relatively simple concept applied with discipline across all applications in an enterprise limits the possibility for trivial deadlocks and sets the stage for avoiding more complex deadlocks. Remote submission of complete logical units of work and ensured multistage completion of transactions with business rules requiring an order of update different from the table and row order are additional techniques to improve concurrency, reduce locking delays, and further reduce transaction work that must be resubmitted after a deadlock is resolved. Employing these methods will reduce the possibility of application deadlocks to a few scenarios involving things like partially overlapping sets of rows or deferred constraints in support of a questionable schema design, while failing to employ a design method to avoid deadlocks can make the generation of deadlocks common and chronic. Because concurrent transaction processing does not routinely appear in unit testing and thus deadlocks do not routinely occur in testing, this principle is important to avoid surprises in production.

Beyond avoiding merely avoiding deadlocks, adding the discipline of describing an order of update across the enterprise will often have the side benefit of revealing duplications and previously unrecognized relationships amongst the applications and systems across an enterprise. Once put in place, this corporate data dictionary framework used to keep track of the order of update can be used to improve agility in deploying new systems. It can also be invaluable to analysis and business intelligence efforts.

When various rows and columns of tables are updated by multiple sessions a situation can arise such that one session holds exclusive access to something that another session is waiting for and is waiting for access to something another session holds. The simplest case requires only two sessions accessing two resources in the opposite order. Real world examples may involve several or many sessions that can be described by a deadlock graph. This article, however, is not about sorting out run time deadlocks actually experienced, but rather, it is about avoiding deadlocks in the first place.

In short, learn how to:

- Improve relations between application developers and dba staff
- Understand why avoiding deadlocks performs in a superior fashion to letting Oracle unwind them

Technical Discussions and Examples

Person first

11:50:27 SQL> @q_who_where

FNAME          NAME
-------------    -------------
Gary            Hotsos
Cary            Hotsos

11:50:37 SQL> @q_person

ID  FNAME       ADDRESS_ID
----------      -----------
1   Cary        1
2   Gary        1

11:51:05 SQL> @q_company

ID  NAME
---  ----
1    Hotsos
2    Hotsos

11:51:09 SQL> update person set address_id = 2 where id = 2;
1 row updated.

11:53:26 SQL> update company set name = 'MethodR' where id = 1;
1 row updated.

Company First

11:50:13 SQL> @q_who_where

FNAME          NAME
-------------    -------------
Gary            Hotsos
Cary            Hotsos

11:50:49 SQL> @q_company

ID  NAME
---  ----
1    Hotsos
2    Hotsos
So you can see that both editors see a problem when they query “who is where?” On the left hand side session the person is thinking “Person First”, dumps the data for person and company, and then begins to fix the data first by updating person. Then on the right hand side session the person is thinking “Company First”, and updates the company first.

Now the right hand session repairs person (the same person the left hand session has already updated, but not committed. And finally (at least for this example) the left hand must update the company name. And there, we have an example of the classic deadlock, which Oracle detects and handles. Notice, though, that Oracle does not do this instantly. Avoiding avoidable deadlocks therefore saves time and reduces the frequency of having to handle rolling back and resubmitting the original logical unit of work (LUW).

Of course if you only have two tables, it is truly trivial. But the real world is much more complex. And in a world of increasing consolidation, direct updates involving tables that were formerly not even accessible from a single application all of a sudden appear. A global corporate schema dictionary is therefore useful. And giving each object an update ordering number and normal order of access (for updates) can help you systematically minimize the possibility for deadlocks.

My friend, Graham Wood, has suggested that if you have no order of update defined across your corporation and applications, at the very least you can institute a simple alphabetical rule. Indeed this is much better than nothing at all, and it is difficult to come up with a simpler rule.

There are advantages, however, to having a more sophisticated rule if you have complex multipart LUWs where the results of one part of the LUW may indicate you should abandon the update (for example) prior to beginning an expensive piece of the transaction.

Note that you cannot make things completely bulletproof in some cases. Consider partially overlapping sets of rows being updated in two different sessions. Even if you are able to control the order the rows are updated, if the younger transaction begins farther down the order, it may be holding a lock on a row the elder transaction needs.
create or replace
package body update_person_company_pkg is
  -- procedure do_actual_update(p_company_id number,
p_person_id number, p_new_address_id number, p_new_co_name varchar2) is
  --   v_local_date date;
  -- begin
  --   select sysdate into v_local_date from dual;
  -- dbms_output.put_line('entering do_actual_update');
  -- dbms_output.put_line('just before update = ' ||
  -- to_char(v_local_date,'YYYY-MM-DD HH24:MI:SS'));
  --   update person set address_id = p_new_address_id
  -- where id = p_person_id;
  --   update company set name = p_new_co_name where id
  -- = p_company_id;
  --   commit;
  --   select sysdate into v_local_date from dual;
  -- dbms_output.put_line('just after update = ' ||
  -- to_char(v_local_date,'YYYY-MM-DD HH24:MI:SS'));
  -- dbms_output.put_line('leaving do_actual_update');
  -- end do_actual_update;

  procedure update_person_company is
    -- lv_co_id          number;
  end update_person_company_pkg;
  /
WAIT #5: name='SQL*Net message from client' ela=17556990 driver id=111838976 #bytes=1 p3=0 obj=-1
tim=63925666798
CLOSE #4: c=0,e=11,dep=0,type=1,tim=63925689198
CLOSE #5: c=0,e=29,dep=0,type=1,tim=63925689267

PARSING IN CURSOR #6 len=61 dep=0 uid=91
tim=63925689460 hv=3919849996
ad='7f8b45d880' sqld='c47k6gznua8ahc'
BEGIN update_person_company_pkg.update_person_company;
END;
END OF STMT
PARSE #6: c=0,e=137,p=0,cr=0,cu=0,ms=0,r=0,dep=0,og=1,plh=0,tim=63925689459

PARSING IN CURSOR #2 len=24 dep=1 uid=91
tim=63925689775 hv=12468195
ad='7f9c95f03008' sqld='c749bc43qqfz3'
SELECT SYSDATE FROM DUAL
END OF STMT
PARSE #2: c=0,e=30,p=0,cr=0,cu=0,ms=0,r=0,dep=1,og=1,plh=1388734953,tim=63925689774
EXEC #2: c=0,e=25,p=0,cr=0,cu=0,ms=0,r=0,dep=1,og=1,plh=1388734953,tim=63925689775
FETCH #2: c=0,e=26,p=0,cr=0,cu=0,ms=0,r=0,dep=1,og=1,plh=1388734953,tim=63925689838
STAT #2: id=1 cnt=1 pid=0 pos=1 obj=0 op='FAST DUAL (cr=0 pr=0 pw=0 time=0 us cost=2 size=0 card=1)'
CLOSE #2: c=0,e=1,dep=1,type=3,tim=63925690011

PARSING IN CURSOR #4 len=50 dep=1 uid=91
tim=63925690390 hv=1531184253
ad='7fbf45b770' sqld='1w4kx0ddn813x'
UPDATE PERSON SET ADDRESS_ID = :B2 WHERE ID = :B1
END OF STMT
PARSE #4: c=0,e=256,p=0,cr=0,cu=0,ms=1,r=0,dep=1,og=1,plh=0,tim=63925690389
EXEC #4: c=0,e=891,p=0,cr=7,cm=3,ms=1,r=1,dep=1,og=1,plh=2144681027,tim=63925691367
STAT #4: id=1 cnt=0 pid=0 pos=1 obj=0 op='UPDATE PERSON (cr=7 pr=0 pw=0 time=0 us)'!
STAT #4: id=2 cnt=1 pid=0 pos=1 obj=115436 op='TABLE ACCESS FULL PERSON (cr=7 pr=0 pw=0 time=0 us cost=3 size=6 card=1)'
CLOSE #4: c=0,e=2,dep=1,type=3,tim=63925691498

PARSING IN CURSOR #5 len=45 dep=1 uid=91
tim=63925691569 hv=4193271048
ad='7f9c98b7ce8' sqld='05g8rs7wzog88'
UPDATE COMPANY SET ADDRESS_ID = :B2 WHERE ID = :B1
END OF STMT
PARSE #5: c=0,e=35,p=0,cr=0,cu=0,ms=0,r=0,dep=1,og=1,plh=1979926739,tim=63925691569
EXEC #5: c=0,e=72,p=0,cr=7,cm=1,ms=0,r=1,dep=1,og=1,plh=1979926739,tim=63925691719
STAT #5: id=1 cnt=0 pid=0 pos=1 obj=0 op='UPDATE COMPANY (cr=7 pr=0 pw=0 time=0 us)'
STAT #5: id=2 cnt=1 pid=1 pos=1 obj=115437 op='TABLE ACCESS FULL COMPANY (cr=7 pr=0 pw=0 time=0 us cost=3 size=10 card=1)'
CLOSE #5: c=0,e=1,dep=1,type=3,tim=63925691806

PARSING IN CURSOR #2 len=6 dep=1 uid=91
tim=63925691837 hv=255718823 ad='0'
sqld='8ggw94h?mvdx7'

COMMIT
END OF STMT

Conclusion

The article is NOT about reading deadlock graphs. It is about AVOIDING deadlocks in the first place. Oracle handles deadlocks. If it did not, you would probably still be waiting for a lot of them. Oracle may not handle deadlocks the way you would like. Oracle detects deadlocks, determines the minimum offending lock held, does the minimum it can to release that lock (such as rolling back an update holding the last lock which is a component of a deadlock, but backtracking no further in that transaction’s uncommitted change stream), and it informs the affected session it has been affected. (Oracle does NOT decide how to pick up the pieces of the interrupted transaction stream.)

Network delays are huge piece of avoiding deadlocks. Plenty of deadlock prone situations exist in the wild in transactions that fly by each other without actually deadlocking because the potential deadlock is avoided by execution speed. As if the first typist in my trivial example finished all steps before the other started, but the transactions order of operations is just waiting for a time sliced collision. Often these situations are NOT exercised in application testing and await a peak load which slows things down just enough, or a situation where a high user count causes transactions to fire against each other that did not fire against each other in testing.

There are transactions that even if written in accordance with the ordering advice in this article cannot be avoided due to overlapping sets of row updates. Writing a transaction in a style that minimizes the real elapsed time from the first lock tossed in a transaction to the commit minimizes the chances of collision. It is not always the cheapest way to do the transaction in database time or even in total network time. But avoiding network waits within a transaction IS exactly on point in avoiding deadlocks.

References

For more information concerning deadlocks, I suggest starting with the oaktable.net article “Understanding and Interpreting Deadlocks or What to do When You Encounter ORA-00060" by Mark J. Bobak at http://www.oaktable.net/contribute/understanding-and-interpreting-deadlocks

The author learned about minimizing deadlocks by ordering the allocation of critical resources in the Kiewit Computation Center SysProg course taught by Philip DL Koch, Brig Elliott, and Carl Pedersen in about 1979. He first used the concept in an Oracle RDBMS context at DTSS on the SoftStore project in conjunction with Chief Project Architect Charlie Muntz. The concept is much older and was certainly well-known during the building of DTSS beginning in 1963 under the direction of Kemeny and Kurtz.

I graduated College in three years (woo hoo!). On to the next phase of life. After a couple of years digging trenches and doing Electrical Construction, I landed my first job that required a Bachelor's degree. It was with the government, insurance stuff. I started auditing the books of companies to see if they were submitting their insurance information correctly. (oh boy, somebody had to do it). One day that agency announced a contest to internal employees only, to compete to join the computer programming department. I thought

it sounded interesting and might be a good career move. The buzz around the office was "yes, you should go for it!", so I did and I won the competition and was offered a job as a Computer Programmer A, writing COBOL programs, (WOW!)

Then another, not so pleasant, "wow" happened right then, a pain in my side, and it hurt a lot. A few days later I was in surgery on my kidneys. Medicine has advanced a lot in the previous few years and I was in the hospital for 10 days. I spent the next 6 weeks recovering at home, finishing up some audits and starting to read and learn how to be a COBOL programmer. By reading the COBOL book by Shelly and Cashman, I learned to program on 80 column cards, which were just then starting to be replaced by something called a terminal, with electronic letters and numbers that would slowly appear on the screen in white or perhaps green lights. (WOW)

Things started to speed up and the long, strange, fun journey began. I was a real IT professional; or perhaps I was an ADP professional; or a Computer services professional; they had challenges with the title back then. I spent two years writing COBOL, moved quickly into a Systems Analyst position and had three programmers reporting to me that I had to keep busy.

Fast forward again - I landed at another small liberal arts college and learned a new language and database technology called BASIC and a database called POISE. I spent the next decade in small college Computer Service departments writing code in VAX Basic. I also did a little Business and Computer Science teaching and picked up another couple of languages, like DCL, C, Pascal and continued programming in COBOL, mostly related to Accounting and Finance applications.

Then the IT auditors arrived using the slogan, “We are here to help you.” Their helpful suggestion was that all those perfectly fine Vax Basic Accounting programs I was using and maintaining were no longer appropriate for the new, modern standards. I would need to find something with greater features, perhaps in a relational database, using 3rd normal form. This would make it completely impossible for folks in the office to get any data out of because the model is so complex.

We spent years listening to Requests For Proposals (RFP) along with all the other small colleges in order to replace our old systems with something new; something that would make the auditors happy. We spent countless hours listening to demos and presentations on this new paradigm shift and the robust technology.

One of the packages was running on a database called, wait for it....
Oracle. Hmmm, heard some good things about this relational database, we should no doubt get some of that, don’t you think? Well actually it was an extremely hard sell. The Oracle based product had to be sold to several other institutions before the end user committee reluctantly agreed with the IT geeks this Oracle stuff was the way to go.

We had reviewed and analyzed Oracle 5, yet by the time we purchased the Oracle systems, Oracle was running version 6 on VAX equipment. My real education on Oracle begins with training where I got to learn stuff like SQL, PL/SQL, SQL*Loader, RPT, RPF, SQL*Plus, Pro-C, RPT2c, Pro-Cobol, (really hard to get away from Cobol), SQL Report Writer, Oracle Forms, and all kinds of tuning stuff (Much better than BASIC don’t you think?)

I also had to learn about tablespaces filling up and Database Administration tasks and how to put all this great new stuff to use. For the next 8 years I wrote programs in SQL*Plus, PL/SQL, Pro-C, Pro-Cobol, Oracle forms and reports - all on Oracle 6. Then someone mentioned Oracle 7 was about to arrive with something called referential integrity- that is going to hurt a little and maybe leave a mark. (Smile)

The headhunters started to call. “Say, John, do you know anything about Oracle?”

Well as a matter of fact I did and I was told anyone with 5 years of experience doing something is considered an expert. So now I am an Oracle Expert. (woo hoo!) Later to be called a Subject Matter Expert or a SME. (heavy sigh)

It was also about that time I started to hear about this group of people that got together and had meetings on the topic of Oracle. I had been a member of the local Apple users group called Frosty Mug. (I am uncertain of the exact reference, yet, it gets cold in the Colorado Rockies in the winter, some might say Frosty. Or perhaps they were referring to a favorite beverage in the high country, (smile)). I had also been to DECUS (DEC equipment) meeting.

It was there the trip took another turn. I attended a session on SQL presented by a long haired happy hippie type guy named Stan Yellot. Rocky Mountain Oracle Users Group was populated with guys like Stan.

The Oracle life starts to take off. I started to gain experience and was hired away to a steel mill, implementing Oracle Financials 10.6 SC, and 10.7 NCA running on DEC Unix. Then, Boom!, the steel mill went on strike. I didn’t want to cross the picket lines so after a few weeks of searching on the thing called the internet for a new job, Oracle Corporation called and offered me a job as a Senior Consultant. (Woo Hoo!)

Suddenly I am working for Ray Lane and Larry Ellison with a group called Oracle Consulting out of Herndon VA. That was interesting! I was working remotely and would talk to guys in California, Phoenix and Washington DC. I remember this new idea called Webview, (it might be called APEX these days) for which I had to get permissions and systems administration privileges from a couple of guys in these offices. These guys were Mike Hichwa and Tom Kyte. Yes, I knew them then!

Following a contract at Santa Barbara CC, I would up “on the bench” at Oracle. An Oracle opening at the University of Colorado for an Oracle DBA was available so I went for it. We implemented a competitor to Oracle’s products called Peoplesoft on Sun Solaris and Sun hardware. I learned more tuning and created a lot more Oracle Databases for HR and Financials software, still in use today.

By the way, one of the guys I worked for was a huge fan of this group called The Grateful Dead. He used to share concert CD’s of Dead music. Hmmm sharing music with your buddies, interesting idea, I wonder if anyone can figure out how to make money doing that? (smile)

The headhunters starting calling again. The journey moves on to some great places, working for and with some really great people. An engineering firm (where I happen to be working currently as I write this), a dream job at an Ivy league school in New England, a major technology manufacturing company, another world class engineering college, a transportation company, an Aerospace company giant, an ultimate consulting company, back to the University, a financials services firm, a federal contract with a huge defense contractor. All of them using the Oracle Database and related software. It has been a great career.

I would recommend working with Oracle in some way to everyone, however, take lots of vacations! (Smile) My journey continues.

Be well my friends and Imagine Joy along the way, every once in awhile, when you get the chance, taste the dessert.

Keep smiling.

John
RMOUG Events Calendar

1/14/15 (TBD) RMOUG Board of Directors Meeting
1/15/15 RMOUG SQL>Update - Call For Articles and Cover Photo
2/15/15 RMOUG SQL>Update - Deadline For Articles and Cover Photo
2/17-2/19/2015 RMOUG Training Days, Denver (TD2015)
2/18/15 RMOUG Board of Directors Meeting
3/15/15 RMOUG SQL>Update - Publication Date
3/19/15 (TBD) RMOUG Board of Directors Meeting

Please note dates are subject to change. For the most current events calendar visit our website at www.rmoug.org.

SPONSOR A QUARTERLY EDUCATION WORKSHOP AND RECEIVE A HALF PRICE AD IN SQL>UPDATE

Join Us May 15, 2015 at the Oracle Broomfield Campus!

More Details www.rmoug.org

Contact Dan Hotka - Programs Director - ProgramsDir@rmoug.org

Comments About RMOUG Training Days From a Presenter

RMOUG is my favorite Oracle conference because it is so good to speakers. RMOUG makes it a point to offer a ride from and to the airport to all out-of-town speakers. Yes, I can afford a taxi but it feels very good to be welcomed by a smiling face.

It's the little things that matter.

In contrast, one major conference wanted me to pay a fine of $200 for not booking a room at one of the official conference hotels. They waived the fine after I explained that I was only flying in for the day but I don't bother going to that conference any more.

Also, RMOUG has sit-down lunches, not box-lunches. At one major conference I attended this year, the sandwiches and apples in the box-lunches were stale from having sat in a refrigerator for too long.

I might add that one's presentation actually has a shot at being accepted at RMOUG unlike other conferences which tend to favor established speakers. RMOUG stands alone in using a very democratic and fair evaluation process for speaker selection.

Little things like that!

Iggy Fernandez, editor of the NoCOUG Journal, the quarterly publication of the Northern California Oracle Users Group
The Spring QEW will be at the Oracle office in Broomfield on Friday May 15. RMOUG is planning a “Tools” day. Dell will have a TOAD Tips and Techniques presentation, Embarcadero will present a RapidSQL/DB Artisan/Software Management review, and there will be a Hyperion speaker. Dan Hotka, the RMOUG Programs Director, is also working on an OBIEE Tips and Techniques speaker as well as a SQL Developer Tips and Techniques speaker. We will have 6 topics covered with additional information available from the supporting vendors. Door prizes are a possibility as well. This is the morning event...come for the continental breakfast...stay for the nice lunch! Attend and learn some new areas of expertise with these various tools.

If you or your organization are interested in partnering with RMOUG to host an upcoming meeting, or to submit an abstract for presentation, please contact

Dan Hotka, Programs Director at ProgramsDir@rmoug.org

Watch RMOUG’s Web Page for Other QEW Training Topics www.rmoug.org
Highlights
- Two full days of training with a bonus half-day of two-hour deep-dive and hands-on sessions
- More than 100 technical sessions with new tracks
- ACE lunches, WIT, and career panel
- Full exhibition
- And more!

New This Year
- Project O.W.L. (Oracle Without Limits) brings you RAC Attack, CloneAttack, RepAttack, a “Stump the Chump” Table of Experts, and Oracle Hardware DEMOgrounds.
- Wednesday SIG meetup lunches; grab a box lunch and attend a SIG meeting

Keynote
Maria Colgan, Oracle Corporation  Oracle Database In-Memory Option – The Next Big Thing
For more than 30 years, Oracle Corporation has extended the database with industry-leading innovations, never breaking its pledge to bring your existing applications and DBAs along for the ride. That trend continues with the introduction of Oracle Database In-Memory. This session explains in detail what motivated Oracle to develop this new technology and provides a clear understanding of how this unique “dual-format” architecture works. It will also illustrate how the new in-memory functionality interacts with key database performance features such as RAC, partitioning, parallel execution, and much more.