INSIDE OCOUG

A MESSAGE FROM OUR PRESIDENT

FALL 2007 MESSAGE

Our 1st Annual OCOUG/LAOUG Joint Conference was a huge success. We had over 400 people in attendance and surprised all our friends at Oracle. We'd like to thank our vendor sponsors and all those that made it a great event. This special edition of our newsletter is being put together for the Oracle Technology Day in Costa Mesa on November 1, 2007. After the announcement of 11g, we're sure you have a lot of questions as to how everything will fit and what the product paths are. Our Long Beach conference went a long way to answering a lot of questions. With that in mind we gladly accepted Oracle's invitation to come to the Tech Day event and promote it to our users.

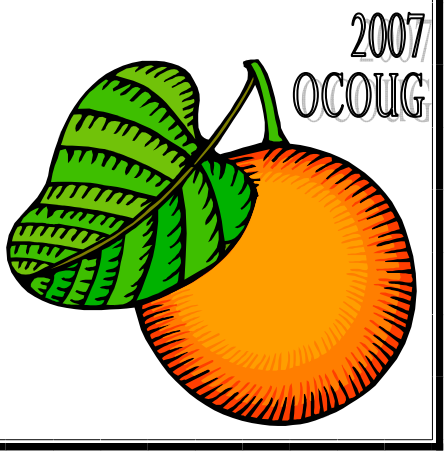
Please take this opportunity at the Tech Day to ask lots of questions and get started in your future migrations to 11g. There's no doubt that all the right people who have the knowledge will be right there at the Tech Day Conference. Many thanks go out to all our sponsors who are represented throughout the pages of this special edition of our newsletter. Special thanks to Oracle Corp. for inviting us to have a presence in Costa Mesa. We greatly appreciate all the support and feedback we got on Long Beach which will help us in the planning for next year's event. Please stop by our table to register for some great raffle prizes! Our goal is to bring you as much timely and useful information as possible and to optimize your time. We hope you enjoy the Tech Day. Thank you for attending and have a great time in Costa Mesa!

Marc Schissler

President OCOUG 2007

marc.schissler@ocoug.org

INSIDE THIS ISSUE:



The Orange County users Group invites you to check out our website at www.ocoug.org, serving Oracle professionals in the Orange County, California area. OCOUG provides a forum for the exchange of technical information related to Oracle products and services.

Officer	Name	Contact Information
President	Marc Schissler	marc.schissler@ocoug.org
Newsletter Director	Mercedes McShane	mercedes.mcshane@ocoug.org
Meeting Director	Max Lockie	mlockie@ocoug.org

OCOUG remaining meetings for 2007:

November 1, 2007

Oracle Tech Day

Westin South Coast Plaza Hotel, Costa Mesa

686 Anton Blvd.

Costa Mesa, CA 92626

(Bristol and Anton)

Register at ocoug.org

Visit the OCOUG Table and enter to win Books and valuable raffle prizes!

Agenda:

8:00 am	Registration and Breakfast
9:00 am	Welcome and Introductions
9:15 am	Executive Keynote
10:15 am	Break
10:30 am	Breakout Session #1
11:30 am	Lunch
12:30 pm	Breakout Session #2
1:30 pm	Breakout Session #3
2:30 pm	Wrap-Up and Raffle Drawing

Sizing up Performance

by Mike Hichwa

Tips and techniques for optimal Oracle Application Express performance.

As Oracle Application Express becomes more popular, many users are asking for guidance on sizing and performance tuning. In this column, I'll show you a quick and convenient way to estimate performance and sizing. I'll also demonstrate how to identify and deal with performance issues. Here are some of the more common questions this column will answer:

- How much hardware, especially how many CPUs, will I need to handle a given workload?
- How many users will my application support?
- How do I locate my performance bottlenecks?

Let's start with some background.

Understanding Oracle Application Express Application Performance

The key to optimal performance for most Oracle Application Express applications is to keep the average page-view times relatively short. Scaling is linear: For example, an application with average page-view times of 10 milliseconds (ms) will be able to handle about 10 times as many concurrent users as an application with average page-view times of 100 ms.

You can use the performance statistics collected by Oracle Application Express to easily approximate how well an Oracle Application Express application will scale. The performance statistics are available on the Monitor Activity page. Assuming that the application is well tuned, with efficient SQL and PL/SQL, the single most important sizing factor is the CPU.

For example, suppose you are developing an Oracle Application Express application that is required to support 1,000 page views per minute. On a dual-CPU system, the application would need to achieve 500 page views per minute, per CPU, or 8.33 page views per CPU per second. Meeting this requirement dictates that the average page-view time in the application must not exceed 120 ms.

The relationship between the available CPUs and the required page views per minute results in the average response time per page, which can be expressed in the following equation:

$$\frac{(N*60)}{P}=A$$

where N is the number of CPUs, P is the number of page views per minute, and A is the average response time per page.

Using this simple equation, you can approximate the required average page-view time to support a targeted number of page views per minute. By altering the number of CPUs or page views per minute, you can establish clear performance targets for your application.

An application's average page-view time can also help you predict the impact a change in the size of your user community will have as that community grows. To determine how many users the application can support, start by first determining the number of page requests during a given time period.

For example, if an average session comprising 50 page views takes 10 minutes, then the application supports 5 page views per minute for a typical session. If you are sizing the application to allow for 1,000 page views per minute, the application will support 200 concurrent users per session.

Extrapolating this data to approximate the daily user community, let's further assume that all users are in the same time zone and that, on average, each user performs two sessions per eight-hour day, resulting in 100 page views (2 * 50 page views per session). Because you are sizing the application for 1,000 page views per minute, multiply the number of minutes in eight hours (480) by the page views per minute (1,000) to determine that the application can support 480,000 page views per eight hours, or 4,800 users.

As a rule of thumb, you should size for your busiest hour, because an application may serve only 100 page views per minute in the course of a day but may serve 1,000 page views per minute during the peak hour.

Another rule of thumb is that you should ensure an average page-view time of 300 milliseconds or less. For high-volume applications, with many hundreds or thousands of concurrent users, the goal should be an average page-view time of 150 milliseconds or less.

Finding and Eliminating the Bottlenecks

The overhead associated with the generation of a page view in Oracle Application Express is fairly static. Any other processing of developer-created SQL and PL/SQL in an Oracle Application Express application is in addition to this static overhead.

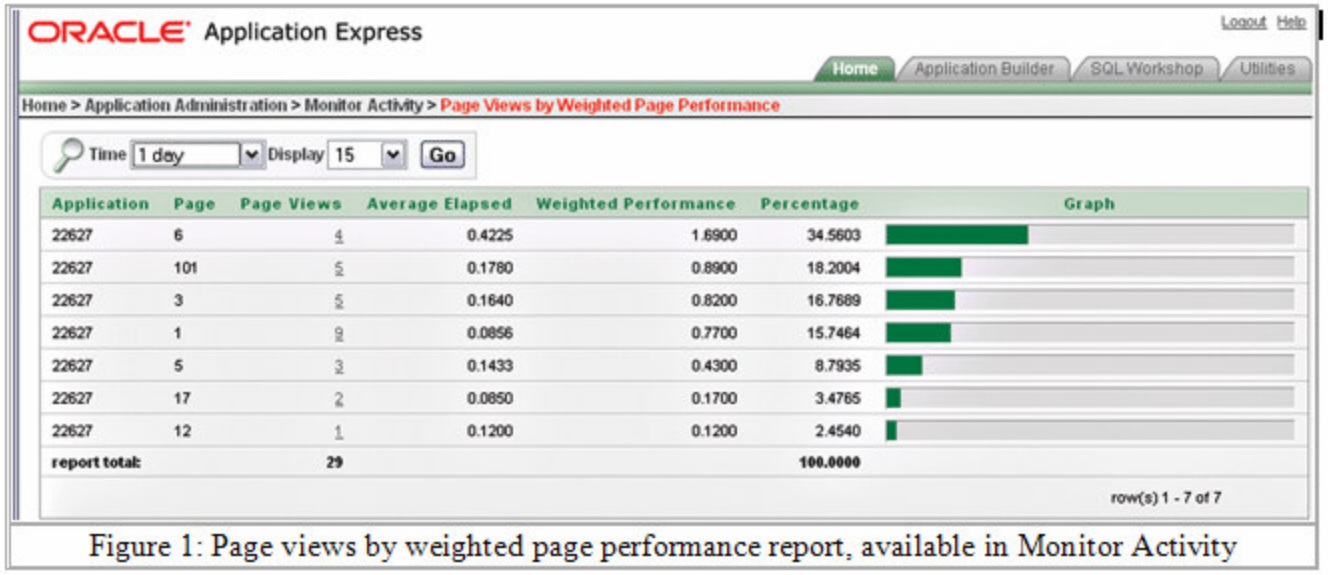


Figure 1: Page views by weighted page performance report, available in Monitor Activity

Then, using the Monitor Activity feature of Oracle Application Express, you can readily identify the poorest-performing pages in the application (see Figure 1).

Once you have identified the poorest-performing pages, you can examine them in Debug mode, by running the application and clicking the Debug link in the Developer Toolbar. Debug mode renders the page concurrently with timing information—you'll see time codes that correspond to specific Oracle Application Express actions as well as additional details about item names, computation, and processing points.

By identifying steep jumps in the elapsed time in the debug output, you will find the poorest-performing page elements. For example, here is a snippet from a page run in debug mode:

```
.
.
.
0.05: Region: Projects
0.06: Show report
0.06: Determine column headings
0.06: Activate sort
0.07: Parse query as: MIKE1
0.07: Binding: ":P24_SEARCH"="P24_SEARCH" value=""
0.07: Print column headings
0.07: Rows loop: 15 row(s)
0.18: Region: Icon View
.
.
.
```

Note the jump from 0.07 seconds elapsed to 0.18 seconds elapsed. This jump indicates that the query took 0.11 seconds to execute, which, in this example, is a large percentage of the total page time and thus an excellent target for performance tuning.

Tuning SQL

The most common area of tuning focus is SQL. The Oracle Application Express engine parses, binds, executes the statement, and fetches the results of the SQL for a reporting region. Use bind variables whenever possible to avoid unnecessary parsing and promote reuse of shared SQL by Oracle Database.

Additionally, ensure that an optimal query plan is being used for the query. Use Explain Plan from the SQL Commands menu to easily generate plans for a particular query.

For a thorough examination of every element of SQL and PL/SQL on a particular page, you can enable SQL tracing for the entire page view. SQL tracing generates a trace file on the server that you can analyze with the Oracle utility TKPROF. (See "Debugging an Application" in the Oracle Application Express User's Guide for more information about using TKPROF in Oracle Application Express.)

Tuning Page Elements

Page processes, computations, authorizations, validations, and conditions are other common tuning opportunities in Oracle Application Express applications. The performance of these elements on the page should be readily apparent when you run your application in Debug mode. Here are some guidelines for these page elements:

Set page processes to run on a per-page (rather than per-session) basis. If you are employing application-level processes, you can set the process to run once per session or per page view. Keep in mind that using the per-session option for an application process can affect all the other page views in the application. A poor-performing application-level process can affect every page view in an application.

Optimize page zero components. Page zero components are rendered on every page, so take special care to optimize all page zero logic. Consolidate numerous PL/SQL blocks into packages. If you are writing large PL/SQL blocks in a page, consolidate them into PL/SQL packages, which are then invoked from your application.

Use declarative conditions. Declarative conditions are faster than dynamic SQL and PL/SQL conditions; for example, using an "Item=Value" condition is faster than using the PL/SQL expression ":ITEM=value."

Use "Rows X to Y" pagination for reports that return numerous rows. "Rows X to Y of Z" takes longer to compute than the simple "Rows X to Y" pagination scheme. With "Rows X to Y of Z," if your report returns 900 rows, Oracle Application Express will need to fetch all rows to obtain the total row count; with "Rows X to Y," on the other hand, the reporting engine needs to fetch only Y + 1 rows.

Conclusion

Using the sizing guidelines presented in this column will help you estimate the performance and scalability of Oracle Application Express-based applications. If you're not happy with an application's performance, use these pointers to locate the slowest pages and then identify the poorly performing components within those pages. With this information, you can ensure that your Oracle Application Express applications meet performance expectations.

Mike Hichwa (michael.hichwa@oracle.com) is vice president of software development at Oracle and manages Oracle Application Express, Oracle SQL Developer, and other database development tools.

Standards-based Fusion

by Marta Bright

Oracle Fusion standards make applications hot-pluggable.

It wasn't so long ago that the introduction of hot-pluggable hardware made it possible to add and remove devices from a running desktop computer—adjusting capabilities on the fly. Soon hot-pluggable server and network hardware brought power and flexibility to the enterprise, and database clusters such as Oracle Real Application Clusters brought hot-pluggable database power to the data center.

Bringing the same level of interoperability to enterprise applications, however, requires a comprehensive architecture and adherence to multiple industry application standards. Hot-pluggable applications require a hot-pluggable architecture such as Oracle Fusion Architecture.

While there are dozens of standards behind Oracle Fusion Architecture, this article will focus on a handful, discussing five closely linked standards and standards families that include J2EE, Web services, JavaServer Faces (JSF), Business Process Execution Language (BPEL), and portal.

"The architecture we've built for Oracle Fusion is designed to be a comprehensive end-to-end architecture, all built on standard technologies and all with well-defined interfaces," explains Ted Farrell, chief architect and vice president, Oracle Tools and Middleware. "It's a J2EE platform for building the core functionality that an enterprise might want, and it includes JSF. Web services support enables interoperability between applications running on different platforms," he says. "What we have on the back end is the BPEL engine, which wires together business processes. Then we have the outward-facing elements, such as Oracle Portal, which actually expose all of the functionality to the end user."

Serving Flexibility

An application built on J2EE offers flexibility plus well-defined standards and interfaces, such as Enterprise JavaBeans (EJB) and Java Database Connectivity (JDBC) that exist throughout its architecture. "One of the key aspects of Oracle Fusion Architecture is that from a variety of levels you can actually get at information and integrate it everywhere, including from your business processes, your business rules, and your user interface [UI]," Farrell says. "J2EE makes the concept of hot-pluggable easy because of these standard interfaces."

With Oracle's J2EE offering, you can not only choose an application from a single vendor and have an end-to-end stack that's robust and performs but also plug in pieces of existing technology, mixing and matching it as necessary to fit your overall solution—ultimately lowering your overall cost of ownership. "Because you have interfaces at the various levels of Oracle Fusion Architecture, you can plug in different application servers, different message buses, different transaction monitors, or even different security providers throughout your architecture and create the environment that's right for you," Farrell adds.

Serving the Web

Farrell describes Web services as key to interoperability in Oracle Fusion Architecture, noting, "Anything a developer can use to build with, from a standard Java class to EJB to legacy information or PL/SQL running in a database, can be wrapped as Web services. Before Web services came along, people were stuck relying upon the owner of the data they wanted to provide it in a format they could consume via an adapter. With Web services, you can build a J2EE application where you can be talking to SAP or Fortran systems. Whatever it might be, through Web services you can get to the information you need."

Serving the User

Within Oracle Fusion Architecture, JSF—the new J2EE standard for building interfaces to Web applications—enables users to separate UI components from the way they're rendered. "JavaServer Faces is a key element in Oracle Fusion Architecture," Farrell points out. "We've been backing it for about three years, and I think it will be a big benefit to our customers."

With JavaServer Faces and Oracle ADF [Application Developer Framework] Faces, we can build Java UI components that are backed by a variety of render kits for HTML, Ajax, mobile, and Telnet," Farrell explains. "For Web browser targets, we provide HTML and Ajax render kits that will render out the components to a browser. Likewise for mobile device targets, we provide a HTML and Ajax render kits that will render out the components to a browser. Likewise for mobile device targets, we provide a mobile render kit that takes into account the limitations and characteristics of a small mobile device and then renders pages out to it accordingly."

Render kits greatly ease the burden placed on developers. For instance, if developers wanted to add mobile or Telnet capabilities

to an application, traditionally they would have to download a completely new development environment and then program for those technologies. With JSF and Oracle ADF Faces, developers have a single programming interface, regardless of the target device. "A developer simply uses the same standard interfaces, and the render kits take care of all of the complicated technologies required to actually render them out," says Farrell.

Modular Building with BPEL

BPEL is an XML-based language that's used for task sharing across a distributed or grid computing environment. One of the most important things that service-oriented architecture (SOA) standards such as BPEL offer is the ability to take existing systems and partition them into building blocks.

Edwin Khodabakchian, software development vice president, Oracle Server Technologies, puts the BPEL piece of Oracle Fusion Architecture in context, explaining, "If you're a telecom company and you have a system for payment, a system for registering your customers, and a system for provisioning your network, you can take each of those functionalities and expose them using XML, Web services, and Web Services Definition Language [WSDL], as modular building blocks," he says. "Once you have those services in place, you can use standards such as BPEL to easily and rapidly assemble them into end-to-end processes such as a subscription to DSL or cable TV."

Khodabakchian asserts that Oracle Fusion Architecture is not a "big bang" approach wherein Oracle is asking that customers start everything from scratch and rebuild. "Oracle Fusion Architecture employs a lot of technology that is available on the market today," he says. "It's about integrating what you already have. When it comes to service-oriented architecture, within Oracle Fusion Architecture there exists the ability to build modular business processes and change them very easily. I think this is one of the things our customers are excited about."

Facing Portal

Portlets are the basic building block in a portal environment. Portlets encapsulate functionality into a Web-based component that can be dropped into a Web page when the application is initially designed or when end users interact with the application at runtime. Regardless of when or how the portlets are consumed, they provide a standard pluggable model to allow users to build pages with the information they need to do their jobs. "From an Oracle Fusion Architecture perspective, we're looking to standards to provide Oracle's portal implementation as a consumer of standards-based portlets, which will provide for what we're calling the 'face of Fusion,'" says Todd Vender, director, Oracle Portal Development. "We envision the integration of all the various UI artifacts that the different Oracle Fusion Architecture applications will bring. Our goal of having business intelligence dashboards and line-of-business dashboards will all be enabled by this standards-based approach for Oracle Portal."

The portal-related standards include WSRP 1.0, the standard for Web Services for Remote Portlets, and JSR 168, a Java interface for portlet development. The JSR 168 standard describes a Java interface for developing local portlets within a J2EE application, while the WSRP 1.0 standard allows for the ability to access these portlets remotely. Various components such as remote and local portlets can integrate easily when built on the same standards. Moreover, these standards allow for customizable end-user dashboards. Vender continues, "In relation to Oracle Fusion Architecture and portal standards, 'hot-pluggable' means that we can consume other standards-based portlets—built by anyone—and have them integrate with our own Oracle Portal environment and our end-user dashboard environment. These components can also be consumed by other standards-based environments in the same way."

Oracle Portal provides an easy and fast solution to customers integrating their existing applications and tools in Oracle Fusion Architecture. "Because the face of Fusion is a standards-based portal, it doesn't have to be the case that all content that you expose through the portal is built using Oracle technology or provided by Oracle," says Vender. "Oracle Portal allows for the ability to integrate external applications in addition to everything Oracle brings to the table with Fusion."

People Know Standards

Farrell concludes, "We see this standards-based approach as so critical to Oracle Fusion Architecture that we're not only adopting it and consuming it in the architecture itself, but in many of these areas, organizations within Oracle are actively involved in shaping and taking these standards to the next level. Through Oracle Fusion Architecture, we're able to offer ease of use in an entirely standard stack. Another big advantage of Oracle Fusion Architecture being standards-based is that it is these same standards that the future developers of the world are learning in school right now."

Marta Bright (marta.bright@oracle.com) is a senior editor and writer specializing in business and technology issues.

Modeling Tables and Components

by Steve Muench

Maintain your data-centric business services with visual diagrams.

In previous columns, I've discussed how you can use Oracle JDeveloper to create data-bound Web pages for an application's view layer by dragging and dropping components in a visual editor. In a future column, I'll investigate similar capabilities that Oracle JDeveloper provides for building Swing applications. In this column, I focus on a different aspect of Oracle JDeveloper, examining two visual editors that let you work on "back-end" artifacts such as database tables and business components.

To follow along, make sure you're using the latest maintenance release of Oracle JDeveloper (10.1.3.1), which you can download (for free) from oracle.com/technology/jdev. As I have done in past columns, I'll be using the familiar EMP and DEPT tables in the SCOTT schema in Oracle Database to work through my examples.

Visualizing Your Database Schema

If you've ever worked with relational databases, chances are that you've used a database schema diagram to give you a bird's-eye view of the structure and relationships of your application's underlying tables. Oracle JDeveloper provides a database diagram capability that lets you design your database schema and synchronize your schema changes with the underlying database. To see how it works, let's create a simple diagram that displays the EMP and DEPT tables.

Using Oracle JDeveloper, start by creating a database connection that maps to the SCOTT schema; name it `scott`. Then create a new application by using the Web Application [JSF, ADF BC] template, specifying oramag as the application package prefix. Oracle JDeveloper will automatically create Model and ViewController projects for your new application. Next, create a third project, by selecting File -> New from the main menu to bring up the New Gallery wizard. Select the Project category, select Empty Project, and enter Database for the project name in the Create Project dialog box. Next, go to the Application Navigator and double-click the Database project to edit its project properties. Go to the Technology Scope page, shuttle Database to the Selected Technologies list, and click OK. This last step tells Oracle JDeveloper that I will be using this project for database modeling. It simplifies my database design work, by having Oracle JDeveloper filter out choices in various dialog boxes that aren't relevant to the task at hand.

The next step is to create a database diagram. Right-click the Database project in the Application Navigator, and select New. In the New Gallery dialog box, select the Diagrams category and Database Diagram. Name the diagram `Database Design`. Click OK, and a new diagram appears in the editor. Next, in the Connection Navigator, expand the Database folder, the scott connection, and the corresponding SCOTT schema to show the folders containing the SCOTT schema's objects. Expand the Tables folder, and hold down the Ctrl key while selecting both the DEPT and EMP tables. Drag the selected tables onto the diagram, and release the mouse button. In the Create from Database Object dialog box, click OK to create offline database objects. Your resulting diagram should look similar to Figure 1, showing two tables with a foreign key constraint between them.

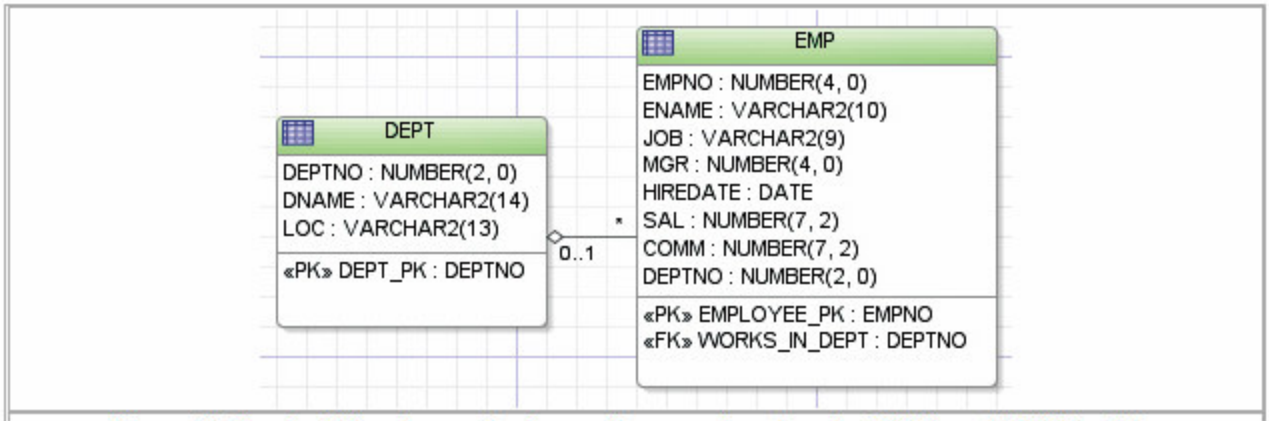


Figure 1: Oracle JDeveloper database diagram showing the EMP and DEPT tables

Next, select the DEPT table shape in the diagram. If you don't already see the Property Inspector and the Component Palette in your Oracle JDeveloper workspace, use the corresponding options on the View menu to display them. The Property Inspector enables you to configure any of the visual aspects of your selected table, and the Component Palette enables you to create new schema objects by dropping them onto the diagram. The Component Palette supports a variety of schema objects, including tables, views, sequences, and synonyms. Furthermore, you can perform many common tasks by making changes directly to the diagram. For example, to add a column to a table, simply click inside the dotted rectangle within the table shape, enter the new column name information (such as `MANAGER_ID:NUMBER(5,2)`), and press Enter. You can also rename a column or change its datatype by clicking the column entry and editing it in place. In addition, you can define one-to-many relationships between two tables by simply clicking the Foreign Key icon in the Component Palette, clicking the source table, and then clicking the target table. And if you double-click a schema object or relationship line in the diagram, the appropriate visual editor appears, allowing you to view and modify the object's full definition.

The tables and other schema objects are known as "offline" objects, because the changes you make in the diagram do not immediately cause updates to the database. Instead, you save your changes to disk, and when you are ready to make database changes, you generate a DDL script that creates or alters your database to match your database design. To generate a DDL script, right-click the database diagram and select Generate -> Data Definition Language for Diagram from the context menu.

Creating Your Business Components

You can use Oracle JDeveloper business components diagrams to visually document and maintain the Oracle Application Development Framework (ADF) objects that implement your business services. To demonstrate, let's create a simple diagram to work with entity objects, view objects, and an application module based on the underlying DEPT and EMP database tables. In the Application Navigator, right-click your Model project and select New to open the New Gallery dialog box. Select the Diagrams category and the Business Components Diagram icon. Name the diagram `Business Service`, provide `oramag.model` as the package name, and click OK. Next, go to the Connection Navigator, select the DEPT and EMP tables (as you did in the last section), and drag them onto the diagram. In the Create from Database Object dialog box, select Business Component Entity Objects, and click OK.

As a result, Oracle JDeveloper creates new Dept and Emp entity objects and adds them to the diagram. It also creates an association between the objects based on the foreign key constraint specified in the table definition. Note that you can now modify these components in much the same way you did in the previous section. You can use the Property Inspector to change a component's visual aspects, the Component Palette to create new components, and onscreen editing functions to perform common inline tasks such as adding or modifying attributes. Furthermore, you can double-click any component to bring up the respective component editor for in-depth editing.

Now drag a view object component from the Component Palette, and drop it onto the diagram. Give it the name `EmployeeList`, by typing that name directly into the new view object shape. Next, select the data the View Object query will reference. To do so, hold down the Ctrl key while selecting the Empno, Ename, and Sal attributes from the Emp entity on the diagram. Then drag the selected attributes to the new view object and release the mouse button. As a result, the Emp entity now becomes the primary entity for the new view object and the selected attributes are added to its attribute list.

You can also add related join information to the view object. For example, to add the department name of each employee, just select the Dname attribute from the Dept entity and drop it onto the view object. Note that Oracle JDeveloper automatically adds the primary key attribute (in this case, Dept.Deptno) from a secondary entity and shows a message alert indicating that it has done so.

Next, drop an Application Module component from the Component Palette onto the diagram. Give it the name `EmpModule`, by typing that name directly into the new application module shape. Add an instance of the EmployeeList view object to the EmpModule's data model, by dragging the view object and dropping it onto the application module. By default, Oracle JDeveloper gives the view instance the name EmployeeList1, but you can change it to something more meaningful (such as `AllEmployees`), by clicking the view instance shape and typing a new name. Finally, you can test the application module directly from the diagram by selecting it and selecting Test from the context menu.

Your diagram will automatically update to reflect changes you make to any component, regardless of how you make them. For example, I have used familiar business components wizards in past articles to create view objects and view links. In this example, you could create a Departments view object based on the Dept entity and a view link between Departments and EmployeeList. Then you could use the Application Module editor to add master/detail linked view instances named Departments and EmployeeList to its data model. After you do so, the diagram will update to automatically show the changes to the data model.

If you drop the new Departments view object and view link from the Application Navigator onto the diagram, fiddle a bit with the colors and layout of the shapes, and set a few options as described in the next section, you'll end up with a diagram similar to the one shown in Figure 2.

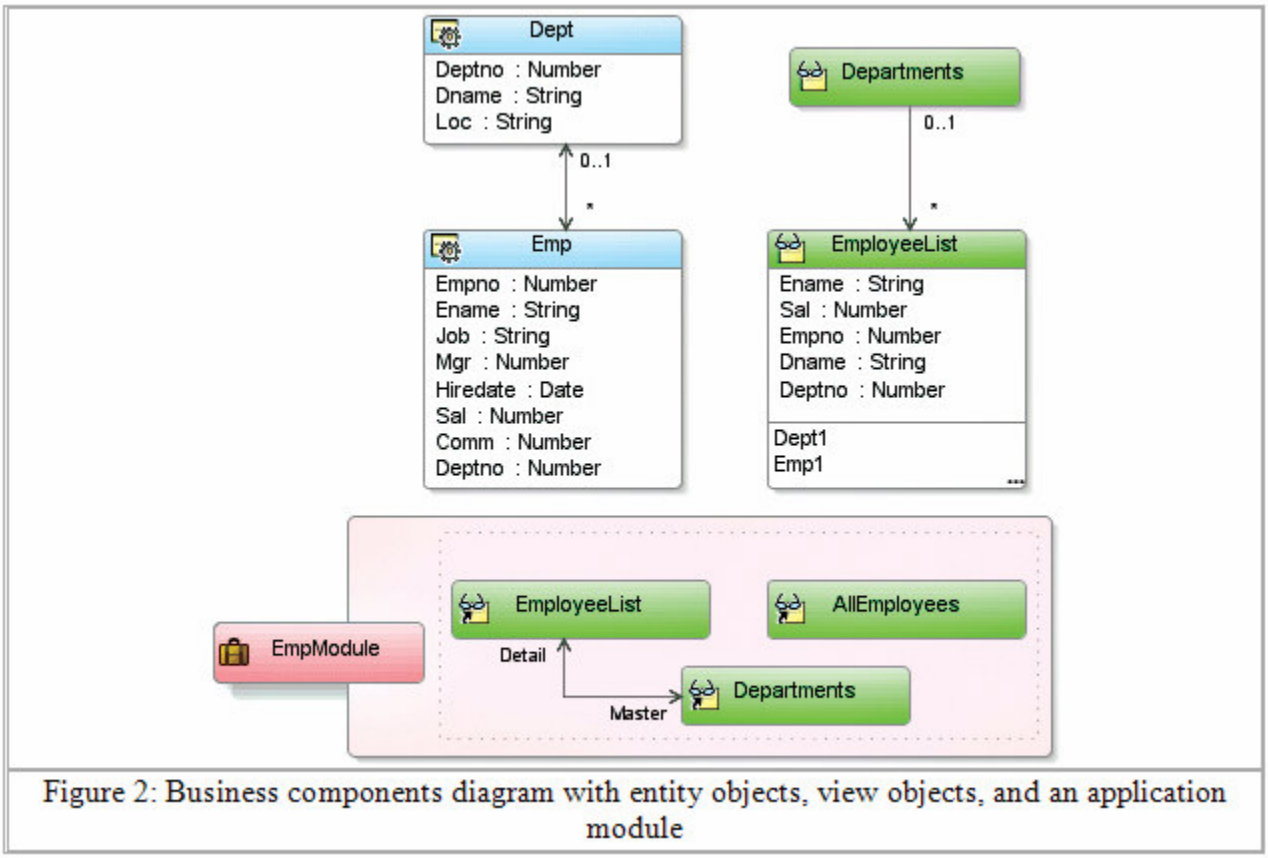


Figure 2: Business components diagram with entity objects, view objects, and an application module

Perfecting, Printing, and Publishing the Diagram

Oracle JDeveloper provides a lot of display options to help you get your diagrams to look exactly the way you want. By right-clicking the diagram and selecting Visual Properties from the context menu, you can control the grid spacing, page breaks, and snap-to-grid behavior of your diagram. Furthermore, by using the Property Inspector, you can control the exact display definition of each shape in your diagram. In this example, you can turn off the display of package names (as shown in Figure 2), by simply right-clicking each shape and unchecking the Show Package check box in the Visual Properties dialog box. Furthermore, you can specify the display mode, by selecting the View As menu from the shape's context menu. Every shape can be shown in a compact or standard display mode, and container shapes can also be shown in an expanded mode. For example, in Figure 2, the Departments view object is shown in compact mode whereas the EmployeeList view object is shown in standard mode.

You can also multiple-select items, and you can use the Property Inspector with multiple selections to set properties on multiple items in a single pass. One handy option is to use the Select All This Type option from a shape's context menu. This option selects all of the shapes on the diagram of the same kind as the one initially selected. You can use this option to easily color-code different kinds of components with just a few clicks.

When you're happy with your diagram, you can print it (by using File -> Print). If you need to include the diagram as a graphic in documentation or an e-mail, you can export the diagram to an external file. To do this, select Model -> Publish Diagram from the Oracle JDeveloper main menu. In the Publish Diagram dialog box, select one of the four supported export formats—PNG, JPEG, SVG, or SVGZ (compressed)—from the File Type list.

This whirlwind tour is meant to help you boost your own visual development skills up a notch. Hopefully, you'll use these and other diagramming tools provided in the Oracle JDeveloper environment on your next project.

Steve Muench is a consulting product manager for Oracle JDeveloper. In his more than 16 years at Oracle, he has developed and supported Oracle tools and XML technologies and continues to evangelize them. Muench coauthored Oracle ADF Developer's Guide for Forms/4GL Developers (Oracle, 2006), and he shares tips and tricks on OTN and in his Dive into BC4J and ADF We-blog.

Enhanced Calculation and Validation

by Steve Muench

Do even more in Oracle Application Development Framework 11g—without code.

In this column, I take another look at the upcoming Oracle JDeveloper and Oracle Application Development Framework (Oracle ADF) 11*g* release and experiment firsthand with several examples designed to improve developer productivity. This column shows how easy it now is to create calculated attributes, validate foreign key values, constrain mutually dependent attribute values, and define more-complex validation rules without writing any Java code.

To follow along with this article, make sure you're using the Oracle JDeveloper 11.1.1.0 (or later) Technology Preview release, which is available as a free download on Oracle Technology Network at otn.oracle.com/products/jdev/11. Furthermore, download the starter workspace at otn.oracle.com/oramag/oracle/07-nov/o67frame.zip.

After extracting the contents of the o67frame.zip file, start by opening the FrameworksNovDec2007.jws workspace in Oracle JDeveloper. Note that the starter workspace defines a familiar set of Emp and Dept entity objects, an EmpView view object, and an HRModule application module. Next, configure the scott connection in the starter workspace to point to the database you'll be using. To do this, expand the Application Resources zone of the Application Navigator, the Connection folder, and the Database node to reveal the scott connection. Right-click the scott connection, and select Properties… to view the database connection settings. Review these settings, and change them, if necessary, to point to the SCOTT database user you'll be working with. Click the Test Connection button to ensure that you can successfully connect to the SCOTT user, and then click OK.

Note that if you are using the free Oracle Database Express Edition, you may need to create a new SCOTT user account with CONNECT and RESOURCE privileges and run the CreateDeptEmpTables.sql script provided in the starter workspace to create the DEPT and EMP tables.

Simplifying Calculated Attributes

Groovy is a standards-based dynamic language for the Java platform, defined as Java Specification Request 241 (JSR 241). It provides a simpler syntax than Java for many common programming tasks, it interoperates seamlessly with any Java class, and it can be both compiled and interpreted on the fly. Oracle ADF 11*g* provides extensive support for the Groovy language, and the first example in this column demonstrates how to use Groovy expressions to define calculated attributes.

In the starter workspace referenced above, the Emp entity object already has a defined transient attribute called TotalComp. Let's update this attribute definition to be the sum of the employee's salary and that person's commission (as defined by the Sal and Comm attributes, respectively). In the formula, let's accommodate for the fact that Comm and Sal might be null.

In Oracle JDeveloper, double-click the Emp entity in the Application Navigator to open the Entity Object Editor. Click Attributes to go to the Attributes page, and double-click the row in the table containing the TotalComp attribute. In the Attribute Editor dialog box, ensure that the Expression radio button in the Value Type radio group is selected. In the Value field, enter the following formula:

```
(Sal!=null?Sal:0)+(Comm!=null?Comm:0)
```

This formula uses a ternary operator that tests a Boolean condition (`Sal!=null`) to return the value of Sal if it is not null and zero otherwise. It then performs a similar calculation on the value of Comm, and returns the sum of both calculations.

To complete the attribute assignment, go to the Dependencies page of the Attribute Editor, select the Sal attribute in the Available list, and click the Add (right-arrow) button to shuttle it to the Selected list. Perform the same steps to also add the Comm attribute to the Selected list. Finally, click OK to save the changes. To test your changes, right-click the HRModule application module in the Application Navigator and select Run from the menu that appears. When the Business Components Browser - Connect dialog box appears, click Connect. Double-click the Employees view object instance, and change the commission and/or salary values in any row to observe that the total compensation is always kept up to date.

Validating Foreign Keys

The last Frameworks column (September/October 2007), looked at how to define declarative lists of values (LOVs) for a view object attribute to assist users choosing existing foreign key lookup values. Keep in mind that, although these LOVs are handy for end users, they are not a substitute for proper validation of the foreign key at the entity object level. For example, some UI components, such as text fields with a pop-up LOV, allow the user to type a foreign key value directly. Furthermore, in a service-oriented architecture, the foreign keys in entity objects can be set programmatically by another application by use of a Web service interface. Fortunately, the new Key Exists validation rule makes it easy to validate foreign key attributes, making quick work of what is normally a programming chore.

The next example adds a Key Exists validation rule to the Emp entity in the Model project. On the General page of the Entity Object Editor, click the Add Validation Rule button, a green plus sign to the right of the Validation Rules heading on the page (on a smaller monitor, you may need to scroll down to see this section). In the Add Validation Rule dialog box, select Key Exists Validator from the Rule Types list. While on the Rule Definition tab, select WorksInDeptAssoc from the Association Name list. This selection denotes a one-to-many association between the Dept and Emp entity objects that represents the foreign key relationship to be validated. Next, on the Failure Handling tab, enter the error message `Department does not exist` in the Message Text box. Finally, click OK to define the new validation rule.

Run the HRModule again, and change the value of the department ID of an existing employee to any two-digit invalid number (such as 99). When you commit or navigate to a different row, an exception with your custom error message will be raised.

Constraining Dependent Values

Another common kind of validation involves comparing two attributes in the same row. The next example enforces a rule that says that an employee's commission must be less than that person's salary. This rule will be applied only when both the commission and salary are non-null, and it will be re-evaluated when either the commission or the salary value changes. The enhanced Compare validation rule in Oracle JDeveloper 11*g* makes this check easy to implement.

On the General page of the Entity Object Editor for Emp, go to the Validation Rules section and click the Add Validation Rule button again. In the Add Validation Rule dialog box, select Compare Validator from the Rule Types list. On the Rule Definition tab, select Comm from the Attribute list and LessThan from the Operator list. From the Compare With list, select Entity Attribute, and select the Sal attribute in the Select Entity Attribute box below. These steps set up the basic comparison in the validation rule. Next, on the Validation Execution tab, enter the formula `Sal!=null && Comm!=null` in the Conditional Execution Expression field. This field causes the rule to be applied only when the specified condition is true. Note that the expression is case-sensitive, so be sure to type Comm and not comm. Next, go to the Triggering Attributes section, select Sal from the Available Attributes list, and click the Add (right-arrow) button to shuttle Sal into the <,/span>Selected Attribute list, along with Comm. At runtime, when the value of any attribute in this list changes, the rule will be re-evaluated.

Finally, go to the Failure Handling tab and enter the following validation error message in the Message Text box:

```
The {attr1} of {val} must be less than
the {attr2}.
```

Be sure to include the three message expressions in curly braces. In the Error Message Expressions table below, click the row for the attr1 token and double-click the Expression cell in that row. Enter the expression `source.hints.Comm.label` to reference the value of the user-friendly display label for the Comm attribute of the source entity object being validated. Similarly, enter the expression `source.hints.Sal.label` for the attr2 message token, and enter the expression `Comm` for the val token to reference the value of the Comm attribute. These message expressions, like those in the calculated attribute example above, use Groovy syntax. Although these are very simple expressions, it's important to understand that developers can leverage the full power of Groovy when necessary. To finish, click OK to define the new rule.

Run the HRModule again to test the rule. Try to enter a value of `13000` for the commission of an existing employee. When you commit, the parameterized error message "The Commission of 13,000 must be less than the Salary" should appear. If you change the value of the salary of an employee with no commission, you can verify that no exception is raised, because the Comm value is null. Finally, you can verify that the triggering attributes work properly, by changing the salary of an employee to a value lower than that person's existing commission.

Writing Groovy Validation Rules

In my final example, I create a more complex validation rule, using the Groovy language. This rule specifies that if an employee belongs to a department whose name ends with the letter s, the salary must be a multiple of 5. This rule requires conditional logic and also accesses an attribute of the associated Dept entity object.

On the General page of the Entity Object Editor for Emp, go to the Validation Rules section and click the Add Validation Rule button again. In the Add Validation Rule dialog box, click the Rule Types list, scroll to the bottom, and select Expression Validator. On the Rule Definition tab, enter the following expression:

```
if (Dept?.Dname?.toUpperCase()?.endsWith("S")
&& Sal % 5 != 0) {
  return false;
}
return true;
```

The Boolean expression in the if statement references the Dept property of the current Emp object being validated to access the associated Dept entity object (if one exists). Then it references the Dname attribute of that Dept object, converts it to uppercase to perform a case-insensitive comparison, and uses the String class endsWith() method to test whether the Dname value ends with s. Note that instead of writing `Dept.Dname.toUpperCase().endsWith()`, I've replaced the normal dot operator with Groovy's "?." safe-navigation operator. This operator works like the dot operator to allow navigation from an object to properties on that object. However, if the left-side value is null, the Groovy operator doesn't throw a NullPointerException but instead evaluates to null. Conveniently, when null is encountered in Groovy as a Boolean value, it evaluates to false, so using the ?. operator can make lots of expressions more compact. After this initial check, the expression above uses the integer modulo operator (%) to test whether the salary is a multiple of 5. If the department name ends with s and the salary is not a multiple of 5, then it returns false, which causes the rule to fail. Otherwise, it returns true.

Next, go to the Failure Handling tab. Note that because Groovy validation rules can conditionally raise one of several exceptions, you can add multiple error messages in this tab. The message you select will be used when the validation rule returns false. To add a message, click the Add Message button (denoted by a green plus sign). When the Select Text Resource dialog box appears, click New.... In the Create Text Resource dialog box, enter the value `MultipleOfFiveSalForDeptsEndingInSMessage` in the Key field, and enter the following error message in the Value field:

```
If department name ends in S the salary must be multiple of five.
```

Then click Save and Select, and finally click OK to define the new validation rule.

Run the HRModule again, and try to enter a salary of 2001 for an employee in department 30 (SALES) or 40 (OPERATIONS). Then commit the change. Because these department names end with an s and the employee's salary is not a multiple of 5, the application will raise a validation error.

Hopefully, these simple examples will help you understand some of the new declarative development features that will be available in the next major release of Oracle JDeveloper and Oracle ADF. For more information on this new release, see the resources at otn.oracle.com/products/jdev/11. If you'd like to learn more about the Groovy language, visit http://groovy.codehaus.org.

Steve Muench has worked for Oracle since 1990. He is a consulting product manager for Oracle JDeveloper and an Oracle ACE who has developed and supported Oracle tools and XML technologies and continues to evangelize them. Muench coauthored Oracle ADF Developer's Guide for Forms/4GL Developers (Oracle, 2006) and wrote Building Oracle XML Applications (O'Reilly Media, 2000). His tips and tricks appear on OTN and in his Dive into ADF blog.

More Partitioning Choices

by Arup Nanda

Learn when and how to use new partitioning schemes in Oracle Database 11g.

The September/October 2006 issue of Oracle Magazine included an article I wrote on the various types of database partitioning and how to choose a partitioning strategy to meet your specific requirements. In Oracle Database 11*g*, the partitioning schemes have been greatly expanded to offer more functionality, including the ability to define new composite partitioning, choose a partition interval, specify a foreign key to inherit the partitioning key of its parent table, and partition on virtual columns.

Referential Partitioning

Consider a hypothetical company, Acme Hotels, for which you are building a hotel reservation system. One core table, named RES, stores the reservation information. Here are the columns of the RES table:

```
RES_ID      NUMBER
RES_DATE    DATE
HOTEL_ID    NUMBER(3)
GUEST_ID    NUMBER
```

The res_id, res_date, hotel_id, and guest_id columns refer to a unique ID number for the reservation, the date for which the reservation was made, the unique ID of the hotel for which the reservation was made, and the unique identifier of the guest who made the reservation, respectively. Because most users query on the res_date column and it is also used to identify records for partitioning, you decide to range-partition the table on that column with a partition per quarter, as shown in Listing 1.

There can be many transactions for a specific reservation, and each record is uniquely identified by a trans_id. Because TRANS is a child table of RES, there is a foreign key on the TRANS.res_id column, pointing to the RES table. Because the TRANS table has the same archival requirements as the RES table, you want to partition it in exactly the same way—range-partition on res_date, with one partition per quarter.

Code Listing 1: Creation of RES table

```
create table res (
   res_id      number primary key not null,
   res_date    date,
   hotel_id    number(3),
   guest_id    number
)
partition by range (res_date) (
   partition p1 values less than (to_date('01/01/2007','mm/dd/yyyy')),
   partition p2 values less than (to_date('04/01/2007','mm/dd/yyyy')),
   partition p3 values less than (to_date('07/01/2007','mm/dd/yyyy')),
   partition p4 values less than (to_date('10/01/2007','mm/dd/yyyy')),
   partition pm values less than (maxvalue)
);
```

Next, you want to create a table to hold the transactions resulting from the reservations. The table, called TRANS, looks like this:

```
TRANS_ID      NUMBER
RES_ID        NUMBER
TRANS_DATE    DATE
AMT           NUMBER
```

But there is a problem: the TRANS table does not have a res_date column, so how can you partition on a column that does not exist?

Enter Oracle Database 11*g*.

Oracle Database 11*g* provides a very useful new feature: referential partitioning. So instead of adding a res_date column to the TRANS table and specifying the partitioning clause, as you did for RES in Listing 1, you can specify a simple PARTITION BY REFERENCE clause in Oracle Database 11*g*, as shown in Listing 2. You must pass the foreign key constraint name as an argument that tells how the references are established. For instance, in this case, you are creating referential partitions on TRANS, referencing the foreign key as FK_TRANS_01, which points to the parent table RES. The TRANS table will inherit the partitioning strategy of the RES table, even though the partitioning column is not present in TRANS. Referential partitioning essentially instructs Oracle Database to equipartition the child table (TRANS, in this example) in exactly the same way as the parent table (RES).

Code Listing 2: Creating TRANS table, using reference partitioning

```
create table trans (
  trans_id     number not null,
  res_id        number not null,
  trans_date  date not null,
  amt           number,
  constraint fk_trans_01
    foreign key (res_id)
    references res
)
partition by reference
   (fk_trans_01);
```

You can see how the referential partitioning has been set up, by querying the USER_PART_TABLES dictionary view, as shown in Listing 3. The partitioning_type column shows the type of partitioning scheme; in Listing 3, the type is REFERENCE, and the ref_ptn_constraint_name column shows the foreign key constraint name, FK_TRANS_01, when the partition type is REFERENCE.

Code Listing 3: Checking for foreign key in reference partitioning

```
SQL> select table_name, partitioning_type, ref_ptn_constraint_name
  2    from user_part_tables
  3    where table_name in ('RES', 'TRANS');

 TABLE_NAME          PARTITIONING_TYP      REF_PTN_CONSTRAINT_NAME
------------         ------------------    -----------------------
 RES                 RANGE
TRANS                REFERENCE                    FK_TRANS_01
```

To check the boundaries of the partitions, query the USER_TAB_PARTITIONS dictionary view, as shown in Listing 4. Note that the boundaries of the partitions of the TRANS child table, shown in the HIGH_VALUE column, are all null. This indicates that the boundaries are the same as those of the partitions of the RES parent table and are not independently defined.

Code Listing 4: Checking partitions of tables

```
SQL> select table_name, partition_name, high_value
  2    from user_tab_partitions
  3    where table_name in ('RES', 'TRANS');

TABLE_NAME   PARTITION_NAME       HIGH_VALUE
----------   --------------       ----------------------------------------
RES          P1                   TO_DATE(' 2007-01-01 00:00:00', 'SYYYY-M
                                  M-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
RES          P2                   TO_DATE(' 2007-04-01 00:00:00', 'SYYYY-M
                                  M-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
RES          P3                   TO_DATE(' 2007-07-01 00:00:00', 'SYYYY-M
                                  M-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
RES          P4                   TO_DATE(' 2007-10-01 00:00:00', 'SYYYY-M
                                  M-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
RES          PM                         MAXVALUE
TRANS        P1
TRANS        P2
TRANS        P3
TRANS        P4
TRANS        PM
```

With referential partitioning, you can partition any child table in the same way as its parent to improve performance and meet archival requirements, even though the partition key column is not present in any of the child tables.

Suppose you purge the RES table by dropping partitions. All the child tables are consequently purged as well, automatically. When you add a partition to the parent, a corresponding partition is added to the child table, automatically. Similarly, when you split the PM partition of the RES table to carve out a new partition, the PM partition of the TRANS table is also split into two partitions, at exactly the same point. Here is how you can split the PM partition of the RES table at the value 401:

```
alter table res
split partition pm
at (401)
into (partition p4, partition pm);
```

Now if you query the partitions of the TRANS table, the child, you see

```
select partition_name
from user_tab_partitions
where table_name = 'TRANS';

PARTITION_NAME
---------------
P1
P2
P3
P4
SYS_P45
```

Note the new partition, SYS_P45, which was created when the PM partition was split. (The SYS_P45 name comes from a sequence number prefixed by SYS_P.) You may want to change its name to be consistent with the naming of the PM partition in RES. To rename it, use the following SQL:

```
alter table trans rename partition
SYS_P45 to PM;
```

Note that if the child table had any locally partitioned indexes, their corresponding partitions would have been split as well. When you drop or split partitions on the parent, the operations are performed on the corresponding partitions on the child. If the parent has multiple referentially partitioned child tables, the operations are performed automatically on all of them. Referential partitioning enables you to define an appropriate partitioning strategy for the parent table while leaving the details out for the child tables. This not only simplifies the administration of partitioned objects significantly but also allows equipartitioning even when it is not desirable to include the partition key columns in the child table—or even possible to include them, such as with prepackaged applications where adding an extra column in a table is not allowed.

Interval Partitioning

What happens when an incoming record in an INSERT statement has a value in the partitioning key column, for which no partitions have been defined? The INSERT will fail. However, if you have defined a default partition by, for example, using the MAXVALUE clause in a range-partition scheme (as shown in Listing 1) or the DEFAULT partition for a list-partitioned table, the INSERT will not fail but the new record will go into the default partition, which defeats the purpose of partitioning. So you have to ensure that you have a partition available for all possible records coming into the table; you have to carefully identify all possible values and then create partitions for all of them before the actual data arrives. For instance, suppose that the RES table is partitioned on the res_id column with partitions defined on value ranges 1-100, 101-200, and 201-300, so that you have defined partitions up to res_id = 300. You can monitor the maximum res_id column value, and just before it approaches 300, you can create a new p4 partition for values 301-400.

This task may be easier said than done. If you forget to add the partition, the new record with res_id = 301 will either fail to insert or go into the default partition, if you have defined one. Wouldn't it be great if the partitions were somehow created automatically?

Oracle Database 11*g* can do exactly that: create partitions automatically as needed without your intervention. To accomplish this, implement an interval partitioning method, as follows:

```
create table res (
   res_id      number not null,
   res_date    date,
   hotel_id    number(3),
   guest_id    number
)
partition by range (res_id)
interval (100) store in (users)
(
   partition p1 values less than (101)
);
```

This script creates a partition named p1 for records in which the value of the res_id column is 1-100. When records with a res_id value of less than 101 are inserted, they go into the p1 partition, but when a new record shows up with a res_id value equal to or greater than 101, Oracle Database 11*g* creates a new partition for it with a system-generated name. For instance, suppose you insert a record that has a res_id value of 901, as follows:

```
insert into res values (901,sysdate,1,1);
```

Now check the partitions defined on the table by executing the query shown in Listing 5. Note how the partition SYS_P82 was created automatically to hold the new 901 value, which was not within the boundary value of the P1 partition. At this time, if you insert another record with a res_id value of 301, the partition SYS_P82 is split in two, as shown in the output at the bottom half of Listing 5.

Code Listing 5: Checking partitions generated in interval partitioning

```
SQL> select partition_position, partition_name, high_value
  2    from user_tab_partitions
  3    where table_name = 'RES'
  4    order by 1;

PARTITION_POSITION    PARTITION_NAME     HIGH_VALUE
--------------------  --------------     ---------------
1                     P1                  101
2                     SYS_P82            1001
```

The result of the same query after inserting res_id = 301:

```
PARTITION_POSITION    PARTITION_NAME     HIGH_VALUE
--------------------  --------------     ---------------
1                     P1                  101
2                     SYS_P83             401
3                     SYS_P82            1001
```

You may want to address partitions by name, such as when you are truncating a specific partition. Given that the partition names are generated at runtime and you don't know them in advance, how can you address the specific partition of the table? For instance, suppose you want to truncate the partition that contains the value 901 as the res_id but you don't know the name of the partition. One way to find the partition name is to query the USER_TAB_PARTITIONS data dictionary view, as shown in Listings 4 and 5, but an easier way is to use the expanded partition access syntax in Oracle Database 11*g*. You can truncate that partition by issuing the following SQL:

```
alter table res truncate partition
for (901);
```

You can use the FOR (value) syntax in any direct partition access SQL statement for any kind of partitioned table, not only for interval-partitioned tables.

In interval partitioning, the first partition you specify in the table creation script is created in the default tablespace of the user but the subsequent partitions are created in the default tablespace of the database, not that of the user. Even if you change the attributes of the table to change the default tablespace, the new partitions still go into the default tablespace of the database. To force them to go to different tablespaces, you have to specify an additional clause at table creation time.

To specify the RESDATA1 and RESDATA2 tablespaces as the locations for all new partitions, include the STORE IN clause after the INTERVAL clause, as follows:

```
interval (100) store in (resdata1,resdata2)
```

Now the new partitions will be spread over these two tablespaces in a round-robin manner.

You can also use a time stamp as the partitioning interval. This comes in handy for creating a table containing date ranges and a partition for each month of records. To accomplish that, write the INTERVAL clause as follows:

```
interval (numtoyminterval(1, 'MONTH'))
```

Expanded Composite Partitioning

In my earlier partitioning article, I showed another important feature: composite partitioning. A composite partition is a partition further broken up into subpartitions. Up through Oracle Database 10*g* Release 2, you could divide only range partitions into hash or list subpartitions. Although this was adequate for most partitioning, some situations could really benefit from a range subpartition. For instance, consider the example of the hotel reservations table (RES) you saw earlier. Suppose the hotel ID shows the type of hotel: values 1-100 are for 5-star hotels, values 101-200 indicate 4-star hotels, and so on. Because a comparative revenue analysis is usually done within the same star rating, users tend to select the data for a specific range of hotel IDs, such as 1-100 only.

Therefore, it makes a lot of sense to range-partition the RES table on the hotel_id column. However, users also tend to select the most recent data and you may want to store the older data on cheaper disks to save storage costs. So a perfectly valid competing argument may be to range-partition the table on the res_date column. Both are attractive alternatives—which one should you choose?

Code Listing 6: Range-range composite subpartitioning

```
create table res (
   res_id     number not null,
   res_date   date,
   hotel_id   number(3),
   guest_id   number
)
partition by range (res_date)
interval (numtoyminterval(1,'MONTH')) store in (example)
subpartition by range (hotel_id)
subpartition template
(
  subpartition s1 values less than (101),
  subpartition s2 values less than (201),
  subpartition s3 values less than (301),
  subpartition s4 values less than (401),
  subpartition sm values less than (maxvalue)
)
(
   partition p1 values less than (to_date('01-FEB-2007','DD-MON-YYYY'))
);
```

Why not both? In Oracle Database 11*g*, you can create—in addition to already available range-hash and range-list composite partitioning—the following: range-range, list-range, list-hash, and list-list composite partitioning. In this example, you can create range partitions on res_date and then create a range subpartition on the hotel_id column, as shown in Listing 6. Note that I have deliberately chosen an interval scheme for the partitioning. I could have chosen explicit partition names and high values as well, but I wanted to show how you can use interval partitioning to ease maintenance. I could instead have chosen to reverse the order of partitioning—I could have partitioned on hotel_id and then subpartitioned on res_date. To find the subpartitions created, you can select from the USER_TAB_SUBPARTITIONS data dictionary view, as follows:

```
SQL> select partition_name,
  2    subpartition_name
  3    from user_tab_subpartitions
  4    where table_name = 'RES';

PARTITION_NAME     SUBPARTITION_NAME
------------------ ------------------
P1                 P1_SM
P1                 P1_S4
P1                 P1_S3
P1                 P1_S2
P1                 P1_S1
SYS_P106           SYS_SUBP105
SYS_P106           SYS_SUBP104
SYS_P106           SYS_SUBP103
SYS_P106           SYS_SUBP102
SYS_P106           SYS_SUBP101
```

Partitioning on Virtual Columns

Oracle Database 11*g* also offers a new feature called a virtual column. A virtual column is not actually stored in the table, but it is computed every time it is accessed at runtime and presented to the user. You can partition data on this virtual column as well. Take the example of the RES table. As explained earlier, the hotel_id column signifies the star rating of the hotel: 1-100 means 5-star, 101-200 indicates 4-star, and so on. Suppose you want to make the start-rating value of the hotel_id column value a part of the RES table.

The simplest way to do that is to create a column—STAR_RATING—and list-partition the table on that column. But how do you populate the column? One option is to rewrite the application to populate it, but this is not a very good option. Another option is to use a trigger to automatically update the column. In Oracle Database 11*g*, however, instead of populating the column with real values, you can just use a virtual column. Listing 7 shows the table creation script.

Code Listing 7: Virtual column partitioning

```
create table res (
   res_id            number not null,
   res_date          date,
   hotel_id          number(3),
   guest_id          number,
   star_rating       number(1)
      generated always as (
         substr(hotel_id,1,1)
      ) virtual
)
partition by list (star_rating)
(
   partition star5 values (5),
   partition star4 values (4),
   partition star3 values (3),
   partition others values (default)
)
```

So instead of making users figure out how to decipher the star rating, you have created a column that conveys the meaning in an unambiguous manner and have created a meaningful partitioning scheme based on this column. And you've done all this without writing a single line of code in a trigger to populate the column.

Conclusion

Partitioning is a powerful feature in modern database design that enables easier database administration with no change in logical design. In most cases, it helps accomplish the seemingly contradictory objectives of fine-grained storage, backup, archival, and retrieval—all with no application changes. In Oracle Database 11*g*, the partitioning option is further enhanced with the introduction of referential and interval partitioning, extended composite partitioning, and partitioning on virtual columns to offer a compelling reason to include partitioning in your physical database design.

Arup Nanda (arup@proligence.com) has been an Oracle DBA for more than 12 years, handling all aspects of database administration, from performance tuning to security and disaster recovery. He was Oracle Magazine's DBA of the Year in 2003.

MEMBERSHIP INFORMATION

For those unfamiliar with the OCOUG, we are an independent organization catering to ORACLE professionals (Business Managers, IT Managers, DBAs and Developers) that live and/or work in the Orange County area. Services we provide include:

- Host quarterly meetings where members gather to hear/share the latest information on Oracle products and services. There are usually 4 presentations at our meetings that focus on topics of interest to the member community. These meetings provide an excellent opportunity for professional networking (attendance has been averaging between 75 and 100) and continuing education.
- Publish a quarterly newsletter that includes articles from national/regional user groups, sponsor organizations, OCOUG members, or from Oracle itself. The newsletter provides timely access to industry news and white papers from experts on a range of Oracle topics.
- A web site (**www.ocoug.org**) with the latest updates and links to Oracle information.

The OCOUG has established the following fee structure to allow local professionals and businesses to participate in the User Group experience.

	INDIVIDUAL MEETING ($20/meeting)	INDIVIDUAL MEMBER ($50/year)	CORPORATE MEMBER ($125/year)
Quarterly Newsletter	No	1 Copy	3 Copies
Quarterly Meeting	1 Admission	1 Admission	3 Admissions
Web Site Access	Yes	Yes	Yes

"MEMBERSHIP DEFINITELY HAS IT'S PRIVILEGES"

If you are interested in becoming a member - or renewing your membership - for 2005, please fill out the membership form that was mailed to you and return it along with your membership fee. There is a membership form in the back of the newsletter on page 31. You can also sign up at our second meeting of the year – June 1, at the Hilton Irvine across from the Santa Ana/John Wayne Airport.

Thank you for supporting the OCOUG. Our goal is to provide a forum for self-improvement, career networking, and continuing education. Your participation goes a long way towards achieving those goals.

For our Corporate members, you may be interested in a sponsorship. Please contact me for all the exciting details and see how it can help your business!

Marc Schissler

Communications Chairman – OCOUG

marc.schissler@ocoug.org

(714) 609 - 4550

MEMBERSHIP FORM FOR OCOUG

Please submit your membership form to Marc Schissler at the meeting or send it to:

Orange County Oracle Users Group
C/O Ontime Consulting Services, Inc.
PO Box 580
Yorba Linda CA 92885-580.

Once the form is submitted you will receive an invoice in the mail for membership fees. For questions on membership see the exciting details on page 30 of this newsletter.

Please join Today!

Membership Order Form

First Name_____

Middle Name_____

Last Name_____

Title_____

Company_____

E-mail_____

Address1_____

Address2_____

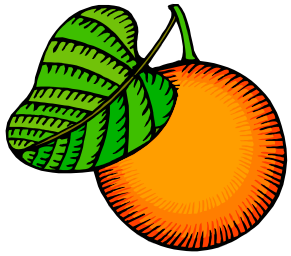
City_____

State_____

Zip Code_____

Phone Number_____

Special Instructions:



OCOUG Announces

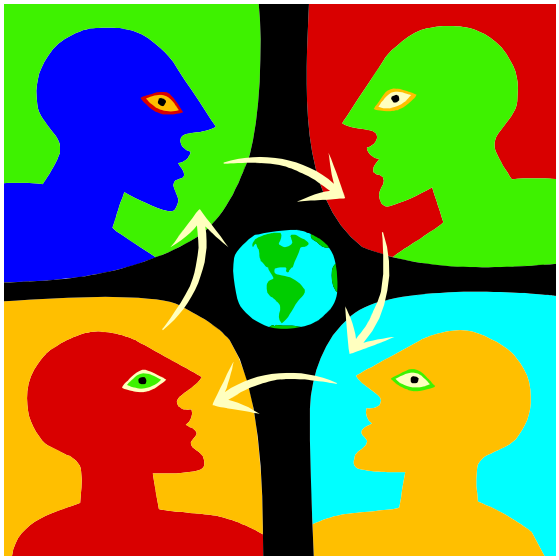
November 1, 2007

Oracle Tech Day

Westin South Coast Plaza Hotel, Costa Mesa

(686 Anton Blvd., Costa Mesa CA 92626)

Register at ocoug.org



From:

Orange County Oracle Users Group

c/o Ontime Consulting Services, Inc.

PO Box 580

Yorba Linda, CA 92885-580

STAMP

TO: