

The Eternal Donut of the Soul

5. Mod PL/SQL – Web Based Applications

podcast@xor.com.au

Introduction

The aim of this paper is to debate some of the common issues DBAs and developers come across when using Mod PL/SQL and building web applications.

1. Don't make me think.

This is the mantra espoused by web page developers. It's a good mantra. It makes sense. It works. It works well. HTML pages are developed simply and in an obvious fashion. A good example is the google home page. You go to it, it's simple and you don't have to think to use the page. Even my mother has no trouble understanding what to do on it.

So what's the issue? Not everyone who uses the web is a first time novice. Some are advanced users and well skilled in traversing internet sites. If you are building a web based applications you need to work out your user base and adjust your interface accordingly.

a. Novice users. These are your customers. They are ad-hoc users, maybe accessing the site only once or twice. For e-commerce sites and public service sites, they are the main focus. The web based application is primarily for them. For these users, the mantra "don't make me think" is to be driven home. The pages built must be simple, not requiring instructions and be thoroughly tested, and tested, and tested again by testers to determine the web pages can be used without thinking. It means simple pages, easy to read, easy to use and well thought out. This is incredibly hard in practice to do.

b. Intermediate users. There are users who access the site on a regular basis. They might want to use advanced functions, they will likely not want to see fancy graphics, and they typically want to do things quickly. More advanced functions can be built on the pages for these users with some simple help. These users will need to think a little bit to use the web page, but not too much. They have a basic understanding of what they want to do.

c. Advanced users. These users are expecting to think. They don't want simple pages, they want to achieve something. Maybe they want to do an advanced and very complex search and control how it runs. They are expecting additional functionality and they are happy to think to a point to do what they want. For example, these users might be researchers. They don't want delays in a web page appearing just so they can see some glitzy graphic which they have already seen many times before. They don't want to be held up by having to wait for flash to download. They don't want to see adverts or references to other pages which are not going to be useful to them. They will only want color on the pages if that color makes it easier to find information.

So which users do you build your web site for? With Mod PL/SQL and dynamic web page development, you build the pages to work with all three at the same time. Get away from static pages, semi-dynamic web page development and being constrained using template driven html code. This is the one big beef I have with .asp, php and similar coding environments. They don't encourage developers to build sites this way.(Note: I am willing to concede that using templates for novice users might make it easier and quicker to rapidly develop pages and ensure a page is developed where the user doesn't have to think to use it).

Also, don't expect the programmers to build perfect web pages that are user friendly. Be prepared to quickly adapt and change the interface based on serious feedback from testing. And build the smarts in so that the pages can adapt to the skill level of the user.

And finally, web sites are like fashion. What looks good and trendy today looks like a wide rimmed, plaid tie tomorrow and embarrassing to see. So don't get caught up in arguments on what makes a site pretty. Build your web sites to be adaptable and be able to be changed quickly with minimal fuss as the fashion changes. Keep the display separate from the content, but keep them close together to ensure performance.

2. Which one to use: Apache or Embedded Gateway?

The Oracle companion CD has Apache already to run on it with Mod PL/SQL. Oracle also has the embedded PL/SQL gateway, allowing you to deliver web pages through the XMLDB interface in the listener. Which one to use? Decisions, decisions.

What are the advantages of Apache? You can integrate a large number of additional capabilities like PHP, URL redirection etc. You can define virtual directories. Apache scales well, auto balances and can reside on a separate server to the database. There are number of books in the marketplace giving comprehensive instructions on configuring and using Apache.

Where does that leave the embedded gateway? It's lightweight and can be configured very quickly from within the database. Configuring it doesn't require a restart.

Mod PL/SQL applications work identically between Apache and the Embedded Gateway. From loading images to delivering multimedia, they are both identical in behavior. To separate them, it's best to look at their faults.

Apache faults:

Apache needs to be restarted every time a change is made. This is a big annoyance as any change made requires a complete restart, and on Windows Apache can sometimes get stuck requiring a machine reboot to resolve it. You need to use a Perl script to encrypt the password in the DAD (here is a recommendation to Oracle: allow the encryption to be done from within the database using a PL/SQL call). There are a number of configuration files that have to be edited to correctly configure it. There are so many options in Apache it can be overwhelming for the novice user trying to understand and use it. Though Apache is very safe from a security point of view, there could in theory be hacks in it that can be exploited. As Oracle is only using Apache 1.3 there is a lag time to have any issues fixed.

Embedded Gateway faults:

The embedded gateway can only be used to deliver Mod PL/SQL pages. There is no concept of a virtual address (you need to build your own program to support it). It's hard and requires some programming to setup static home pages. The listener is the webserver. If the listener fails, you lose access. The listener has to reside on the same node as the database (which might be a strength, but you don't have the option of moving it). But seriously, how often does the listener really fail?

If you are building an application that allows internet users to access the database then Apache and the Gateway both have strengths and weaknesses. For the embedded gateway, you need to configure the listener and open Port 80 on the server (default HTTP – you can configure another port). It's quite easy to redirect requests to the listener via an off the shelf router/firewall. If you can change the port number even simpler. You can have the listener, listening on any port number with external requests coming in being redirected to it.

If your database is internal, it means you don't need a DMZ. Most sites need a DMZ just to get comfortable with allowing external access to their site. Once they get on top of security, and determine they can't be hacked into, then having a DMZ becomes expensive for managing the site. This is especially seen when it comes to replicating data back and forth from an internal site to an external site. If you allow direct access to the database, management and control is simplified. You can then build firewalls within the Oracle database to add additional security.

Apache can be put in the DMZ and the database can also reside in the DMZ or the database can reside internally. If you want to enforce the rule that only valid HTTP packets can be passed to the database, then an inspection routine will be needed inside Apache.

3. To use cookies or not to use cookies.

I really don't like cookies, but at times they can be useful, so I tolerate their use. The big problem with them is when you use them for session authentication. Cookies make it hard to be connected multiple times to the application using different sessions. It can be done, but it is very hard. Setting cookies can be painful especially if you use frames. For example, try in the one call, accept a username and password, validating the account then saving the cookie by putting it into a frame. Then in the same call, opening up other frames, which need the cookie to have been set to validate the user session. The cookie hasn't been written yet, and yet it needs to have been set for the other frame pages to validate the session. Classic chicken and egg. It discourages the use of frames, but as any good DBA knows, frames are great for ensuring an application scales as they reduce the amount of repeated calls being made.

Then we have security. Permanent cookies are seen by a number of sites as being Trojans, spyware or just dangerous. They can be a security issue on public kiosks. Transient cookies are safer, but they are difficult to monitor and debug. In IE try viewing the value of a cookie or even see an audit trail how the value has changed. Firefox has some nice tools for viewing them, but it's still difficult. So if you rely on them for your application, you need to put in place rules to ensure that the browser and site accepts them.

The big advantage of cookies is that they allow a user to navigate away from your site and back again without having to re-authenticate themselves.

So when it comes to developing a web based application what is the best method to use? Time and time again, experience has shown that you can use two methods equally well. The first is on-login to create a random number session based key, and then using Oracle's encryption technique, you encrypt it, and pass that from screen to screen within the application (assume key and mapping is stored inside Oracle). Being a random number that is encrypted, it becomes near impossible for someone to guess the key and connect as you. By passing it from screen to screen the coding becomes slightly more complex, but not overly so. It does mean you can be logged in multiple times as different session in the one application.

If you then require it, you can store one of the encrypted session keys in a cookie. And then when the application is accessed, if no key is given to it, it uses the one in the cookie. It extracts it and then passes it to the application for use.

Should the IP address be attached to the session key for a stronger level of security? In this case, on login the IP address is noted against the encrypted session key. Subsequent validation also checks the IP address to make sure someone has not guessed the encrypted key or intercepted it. For internal networks where you know the IP address will not change for the session, then you can get away with this. For any internet applications then the answer is no, as some ISP's can rotate the IP address of the session between http calls. If there is concern about encrypted keys being intercepted over an internet connection then use SSL or attach a short timeout period to the session, so that if there is no activity against the key the session is disconnected (also have to be built in the application inside oracle).

4. Use the htp.p procedure only or diligently use all the htp procedures

Should one program using a procedure call like `htp.bold('My text')` or do I do it manually using htp.p package like `htp.p('My Text');` ?

My view is that a healthy use of both is ideal. I tend to use htp.p for Javascript and frame setups. It makes it easier to see what is happening and ensure the code has no errors in it.

One useful feature is conformance and ensuring syntactically correct HTML. This highlights a strength and weakness of the mod PL/SQL packages. When you use the procedure `htp.bold`, you don't need to worry about the markup used. You are assured that the syntax is correct. Whereas if you don't use it, then it becomes possible for programmers to write a statement like : `htp.p('My Text');` which is supported by most browsers but is not XHTML correct. They could make a simple typo and type in `htp.p('<bMy Text');` and really get it wrong. Interestingly, when you do `htp.bold('My Text');` you get `My Text`, which is not XHTML compliant either, it should be `My Text`. And this leads to the strength of the htp procedures. One change in the core code will generate results for the correct syntax for all programs in the database, whereas if you manually call it, then you need to check every statement for compliance.

Note: Access to the core code is done by modifying the `$ORACLE_HOME/rdbms/admin/privht.sql` package and recompiling as sys. As usual be careful doing this.

Performance wise, htp.p is faster than htp.bold. This has been tested and proven (see results below). The performance difference isn't great, but if you are trying to support thousands of users doing large, complex htp calls then that difference might be significant.

The Mod PL/SQL package has been in use since 1995 and different programming styles have emerged as the package grew. Some programmers insisted on using all the supplied functions, such as `htp.bold` and `htp.frameset`, whilst others decided to encode raw html in the htp.p procedure.

For those who are not familiar with the htp package, when looking at the raw code all htp.x procedures eventually boil down to a function call then a htp.p call which creates the raw html. There are advantages and disadvantages to each and they are not going to be covered here. What we are looking at is which is fastest?

The following test was done from a Linux server and a laptop running windows, so it's not fair to compare Linux to windows. When looking at the results you need to compare the calls.

	Iterations	Windows (seconds)	Linux (seconds)
<code>htp.p('<TD...</code>	1,000,000	13.1	17.5
<code>htp.tabledata('...</code>	1,000,000	15.1	20.6

So the `htp.p` call is faster because it doesn't have to make a function call to the `htf` function. Based on these results, I wouldn't stop using `htp.tabledata` or any standard `htp` call. If I was building an application that needed to support thousands of concurrent users on a minimally configured server, then I might be tempted to insist on using the `htp.p` calls. In this case one is sacrificing maintainability for performance. Otherwise think maintainability and use the `htp` and `htf` calls that ensure compliance, and make the code easier to manage.