Structuring an APEX application

Developing Oracle database applications with Oracle Application Express (APEX) can be really easy. Even if you have a basic understanding of SQL and PL/SQL you can create an application really fast. Utilising all the wizards for form pages, reports, calendars or graphs will help you to get started with APEX really fast and easy.

And that's where it starts to hurt.

If you are new to APEX it is very tempting to use SELECT statements in reports, this is what the wizards asks for. Form pages ask you for which database table the form should be based upon, again what the wizards ask for. SQL statements get spread all over the application in this fashion, and it's not always clear where the statements end up.

Changing the underlying data model could end up as a big search party. Which pages are affected by the change, and in what way?

Because all pages are rendered dynamically based on the metadata, errors will only occur at runtime - while the application is used.

It is very well possible that you might miss something, but the enduser won't. Usually the enduser finds these error within second after deployment, making you look bad and having to create a patch immediately.

This is a big difference when compared to the Forms-era: pages had to be compiled before they could be used and errors would surface during the compilation.

Another disadvantage is the dynamic nature of the APEX pages: every SQL statement, every PL/SQL call will be executed by Dynamic SQL. This could potentially affect performance of the application.

Given the above it is advisable to have a methodology for developing APEX applications. In this article I will share my methodology when I create a new APEX application.

The Foundation

The data model is essential. Not only for storing the data, implementing data related business rules, but also for performance reasons. In this article I will not cover how to create and implement the perfect data model, there are other resources that cover that.

Just like the APEX application itself, every page will also get a so-called Page Alias. The format for these aliases that I use is: the alias of the application followed by the page number. Suppose the alias of the APEX application is STRUCT, the Page Alias for page number 1234 will become STRUCT1234.

By placing the Page Alias on the page itself also gives you a way to communicate with the enduser. When an enduser encounters an error or has a question regarding a certain page, ask for this Page Alias (for example always locate it on the right lower corner of the page). Or when the enduser sends you a screenshot it will be immediately clear for the developer which page it is.

Numbering the pages is also part of the whole scheme. For the whole application different ranges of page numbers are used depending on the functionality within the application. Of course it depends on the size of your application how you define these pages ranges .

0	999	Algemene Pagina's; bv. de Global Page
1000	6999	Main Application Pages
7000	8999	Reporting Pages
9000	…	Maintenance Pages

The home page for each section, e.g. a dashboard page, will get the thousand number. Detail pages (of this home page) will get the hundred number. Detail pages of these pages will get the ten number. Detail pages of these pages will get the single digit.

Using the page numbering like this will create a hierarchal relation between the pages making it obvious which page belongs to which section and how it relates to other pages..

Modules within the Application

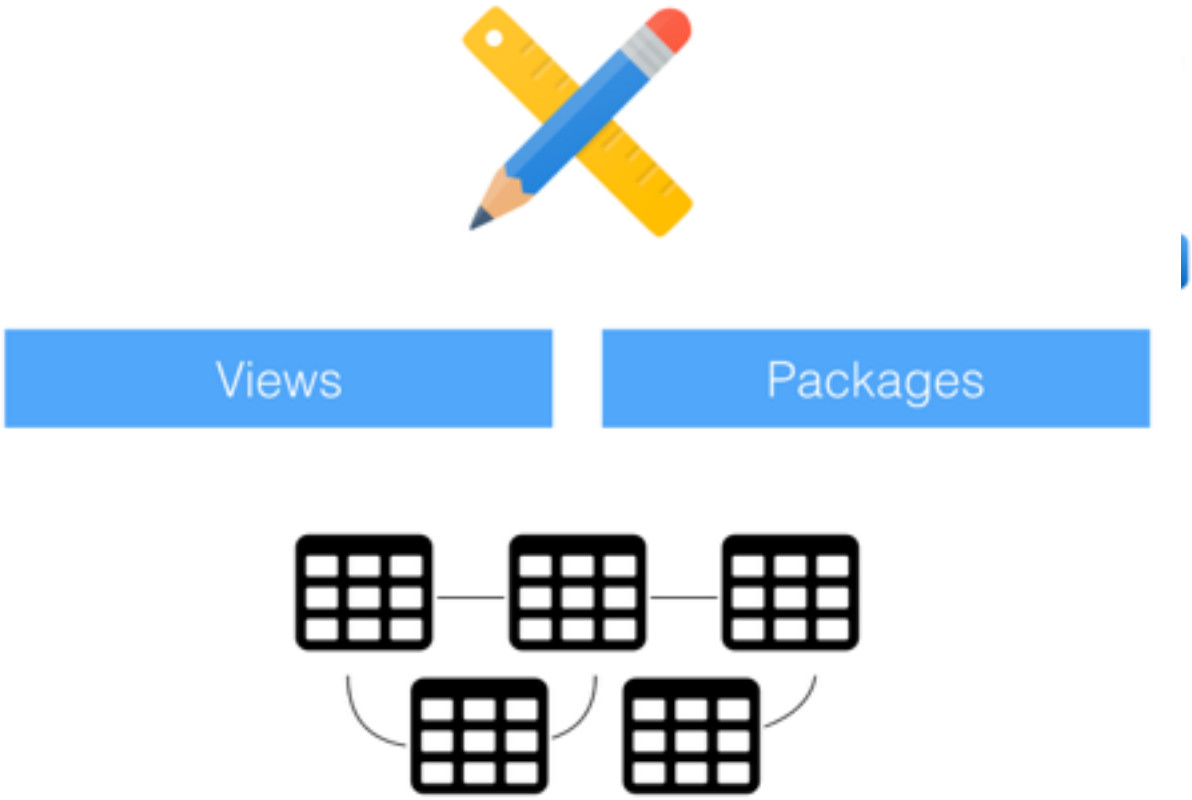
The page ranges are often used as a logical structure of the application on which the navigation menu can be based. Utilising Page Group can aid with enforcing this structure. When you are working on a certain module within the application, use the filter in the Application Builder to only show the pages that belong to the module.

Using the APEX-repository it is trivial to highlight the correct navigation menu based on the Page Group that the page belongs to.

No Table Access in the Application

My rule of thumb is: "A page will never access a Table directly, but will be use a View instead"

Every page will get its own view with the same name like the Page Alias, prefixed with "v_". Page 1010 will have the Page Alias STRUCT1010 and the view that is used will be named V_STRUCT1010. Page 1020 will have the Page Alias STRUCT1020 and the corresponding view will be named V_STRUCT1020. If a certain page requires multiple views to display the information then the view will get the suffix "_2".



The advantage of this method is that every change in the database will be directly correlated to the views and thus the related pages in the APEX application. Now it will be immediately clear which pages are affected limiting your search through the application.

For reporting pages using a view is trivial; in the wizard for the report page simply point to the view and you're done. When the page that you are creating is a Form page, it could be more complex. Now the DML processing will have to go through the View.

Personally I have some bad experiences with Instead-Of-triggers on view in combination with APEX, so I replace the generated DML processing in the APEX page with my own custom PL/SQL code (which is of course in a database package).

It is very important to keep in mind that APEX uses optimistic locking to check for changes that might have taken place in the database after the data is shown on the screen in a page. When you don't take this into account, you might run into a situation that is called the Lost Update. A Lost Update occurs when one transaction overwrites the changes that another transaction has made. Because all transactions happen independently and there is no pessimistic locking, previous updates can be overwritten. This is something you will want to prevent at all costs.

One way to implement Optimistic Locking in APEX is to calculate an MD5 hash based on all data which is shown on the page. When the data is sent back to the database again the MD5 hash is determined based on the values stored in the database. Both of the MD5 hashes (one from the page and the other based on the database values) are compared. When they are the same, you can go ahead and continue with the Update. If they are different, some data changes must have happened on the database in the mean time, you should raise an error informing the enduser that someone else made changes.

Database packages always have the preference over standalone procedures or functions. Using packages will have a positive effect on performance, but will also encourage a modular design of the application. The implementation of the functionality will be hidden in the package body.

The pages don't exclusively make calls to DML processes. It makes more sense to call a PL/SQL program unit to perform a certain task which is related to a business functionality, like "process an order" . In this "Transactional API" you can also implement Business Rules.

Conclusion

At a high level, this is the way I structure APEX applications. Hopefully you will find it useful and will get you started with creating your own standards. Perhaps you have different ideas regarding the way an APEX application should be structured. If this is the case, I invite you to share your thoughts with me. My contact details can be found at the end of this article.

Alex Nuijten is an independent consultant (allAPEX), specializing in Oracle database development with PL/SQL and Oracle Application Express (APEX) and member of the Smart4APEX Guild.

Besides his consultancy work, he conducts training classes, mainly in APEX, SQL and PL/SQL. Alex has been a speaker at numerous international conferences, and received several Best Speaker awards.

He wrote many articles in Oracle related magazines, and at regular intervals he writes about Oracle Application Express and Oracle database development on his blog "Notes on Oracle" (nuijten.blogspot.com). Alex is co-author of the following books "Oracle APEX Best Practices" (published by Packt Publishers) and "Real World SQL and PL/SQL" (published by Oracle Press).

Because of his contributions to the Oracle community, Alex was awarded the Oracle ACE Director membership in August 2010.

You can contact Alex via email: alex@allapex.nl