

Oracle PL/SQL and APEX

Best Practices For PL/SQL Development in Oracle Application Express

Steven Feuerstein

steven@stevenfeuerstein.com

PL/SQL Evangelist

Quest Software

Resources for PL/SQL Developers



www.plsqlchallenge.com

Daily PL/SQL quiz, weekly SQL, APEX and logic quizzes. Prizes!



www.plsqlchannel.com

27+ hours of detailed video training on Oracle PL/SQL



www.stevenfeuerstein.com

Sign up for monthly PL/SQL newsletter



Steven Feuerstein's
PL/SQL Obsession

www.toadworld.com/SF

Quest Software-sponsored portal for PL/SQL developers; download powerpoints and demo.zip

Best practices all happen in here:



- A little obsessive...
- A little compulsive...
- Is probably an OK thing for a developer.

The Big Picture on NOT Best Practices



PL/SQL and APEX: the best *and* worst of friends

- Point and click interface to define your application
 - *Way* better than writing it all in *code*, right?
- One language (PL/SQL) for backend and frontend development.
 - Of course, it sure helps to know Javascript, jQuery, HTML, CSS, etc.
- But how well do APEX and PL/SQL play (together)?

Some Issues We Face

- Repetition and hard-coding
 - APEX doesn't make it particularly easy to reuse application elements, especially code.
- When code is written inside that oh-so-cool UI environment, we tend to pay less attention to it.
 - Irritating interruption from the usual point-and-click.
 - Less than minimal editing capabilities.
- The custom-written source code in your APEX application will present the biggest challenge in terms of debugging, maintenance, support.

What's an APEX developer to do?

- Set clear rules on...
 - Where code goes, and
 - Where and how SQL statements are written.
- Don't repeat anything – hide everything.
- Soft-code as much as you can.
 - Fully leverage APEX's close connection to Oracle tables.
- Prepare for (assume) the worst.

Set Some Clear Rules

1. If the PL/SQL code does not contain references to UI elements, move to packages.
2. If the PL/SQL code contains UI elements, move *as much code as possible* to packages.
 - Leave behind a minimal "footprint" in APEX code.
3. The only SQL you should write inside APEX are queries to populate tables.
 - And even then you should query from views.
 - No updates, inserts, deletes - unless generated and performed automatically by APEX.

Don't repeat anything – hide everything.

- This is Golden Rule of Programming.
 - Repetition = hard coding = maintenance nightmare.
 - You end up with brittle, hard-to-change apps.
- Instead, aim for a Single Point of Definition (SPOD) for every aspect of your application.
 - That way, when you have to change it, you change it in one place
- The hard part is recognizing all the sorts of hard-codings that can occur.

Common Hard Codings in APEX

- Literals that are defined in PL/SQL as constants, but cannot be referenced in queries.
- WHERE clauses, validations, conditions
- Entire pages of similar functionality.
- Be ruthless and disciplined.
 - Single, shared pages are more complex, but in the long run easier to maintain.

Hide with PL/SQL Packages

- Key building block for any PL/SQL-based application.
- Move as much code as possible into packages, avoid schema-level functions and procedures.
- Create lots of small, narrowly focused packages.
- Build a toolbox of handy utilities. Examples from the PL/SQL Challenge code base:
 - Application configuration/constants (qdb_config)
 - URL management (qdb_utilities.apex_url)
 - Date manipulation (qdb_utilities)
 - Template processing (qdb_template_mgr)
 - Practice management (qdb_practice_mgr)

Hide Your SQL

- We PL/SQL developers take SQL completely for granted. *It's so easy to write!*
- But what do we *know* about SQL statements?
 - They change constantly, forcing us to change code; cause most of the performance problems; cause many *runtime errors* in an application.
- Seems to me we should very careful about how, when and where we write SQL statements.
 - Don't take SQL for granted!

Compile Time vs Run Time

- One big advantage of putting your SQL in PL/SQL subprograms or even views is that any errors are detected at compile time.
- If you "dump" the SQL into APEX, then you (or your users) don't see the error until run time.
- This is especially dangerous with upgrades, and consequent incomplete testing.

Step 1: add new columns to table X. Has same name as column in table it joins with.

Step 2: upgrade all packages to support new column. All looks OK, but...

Problem: getting "ambiguous column" errors in application – but *only* if and when you actually exercise that page.

In the context of APEX....

- Your APEX-based, custom-written source code should *never contain SQL statements*.
 - Except for "simple" queries against *views*.
- Replace validation and branch queries with functions.
- Create or generate table APIs or transaction APIs to *hide* the SQL.
 - It's always more complicated than it initially appears.

Prepare for the Worst

- Bad things will happen on your website.
- How prepared are you (and your application code) to deal with them?
- Every well-designed application should have:
 - Execution tracing: enable with "flip of switch"
 - Error logging: make it easy for developers to log and raise errors.
 - Dashboard to easily check status
 - Notifications of problems: email, text, etc.

What we do at PL/SQL Challenge

- Application tracing
 - The "usual" for PL/SQL code and also "in-line" trace calls to add to expressions in APEX.
- Maintain system log for critical information that we don't want lost in the regular trace.
- Integrated test case definition and results tracking

And Finally Some Little Things

- Put all PL/SQL code inside BEGIN-END blocks.
 - Then you can copy into Toad (or your IDE) and reformat it, then paste it back.
- More generally, use your IDE to format all the SQL and PL/SQL statements so you can read it more easily.
- All application items are strings!
 - Watch out for overloadings of string and number.
 - Watch out for comparisons of "date" items.

PL/SQL and APEX: a match made in Oracle

- It's not surprising that it's easy to work with PL/SQL from within APEX.
- But as with SQL, just because it's easy doesn't mean you should treat it casually.
- Write your PL/SQL code with care.
 - Move as much as possible to PL/SQL packages.
 - Decide on clear standards for SQL construction, error management, naming conventions.
 - Build tracing into your code.
 - Test it thoroughly.

REGISTER TODAY!

JW MARRIOTT.
SAN ANTONIO HILL COUNTRY



ODTUG
Kscope12



SAN ANTONIO, TEXAS * JUNE 24-28

Application Express * Database * Developer's Toolbox
Business Intelligence * Essbase * Hyperion Applications
Hyperion Business Content * Fusion Middleware

www.kscope12.com