Enterprise-Grade PL/SQL: Doing it Right the First Time

Or…
Making database engineering easy and fun

by adding rigor to your development process

- Speaker at RMOUG, IOUG, ODTUG and UTOUG since 2001
- With Church of Jesus Christ of Latter Day Saints since 2007
- Consultant and data/database architect in telecom and utility industry (San Francisco, Denver, Houston) from 1995 to 2007

Contacts Info
Website: dbartisans.com and dbsherpa.com
Twitter: @billcoulam
Email: bcoulam@yahoo.com
LinkedIn: billcoulam

Passionate about programming & design practices…
…that make our craft fun and fast!

- *Note! Where technical detail seems to be missing, it is because each slide is worthy of an hour or two of discussion.*

- Indeed I have an 8 hour course covering many of these topics, and individual 1 to 2-hour courses covering Debugging, Instrumentation, Best Practices, Data Layer Decisions, Data Modeling, PL/SQL Application Frameworks, etc. These can all be found at the aforementioned DBArtisans.com or dbsherpa.com website. Look for the Presentations and Papers page.

- This presentation is meant to be an overview of the software products, principles and practices a great Oracle development team uses, providing links to the software and papers that can be perused later for further detail.

Agenda

- Habits of Great Database Developers
- Foundations
- Design
- Code
- Test
- DevOps
- Maintenance

Does this resemble your typical day?

Or this?

Great database developers habitually…

- Simplify / Eliminate Waste / Do Not Duplicate (DRY)
- Take Pride in their Work
- Learn their Craft
- Learn their Tools
- Stand on the Shoulders of Giants
- Get Another Pair of Eyes
- Test
- Document & Instrument

Agenda

- Habits of Great Database Developers
- Foundations
- Design
- Code
- Test
- DevOps
- Maintenance

Foundations

Foundations are tools and technologies that must be decided upon, configured and fully tested before beginning work.

If changing or introducing a technology is impossible or far too costly to consider in the middle of the project, then it is foundational.

Foundations

- For example…
  - Application technology stack
  - Place to store everything created
  - Development methodology
  - Standards and conventions
  - Design, Development and Testing tools

Foundations: Fundamental Questions

- What are you building?
  - UI, server-side, ETL, validation, pub-sub, message-oriented, fat vs. thin, stateful or stateless, services, event-driven, automation, etc.
- What is the audience like?
  - Location, size, concurrency, language, etc.
- Budget?
- Performance?
- Security?
- Growth?
- Quality vs. Time-to-market

Foundations: Technology Stack

- Answers to the fundamental questions help guide:
  - Which UI, app server and database tech will be used
  - Which OS will be used by developers, designers and DBAs
  - Which language will be used to develop the server side code that interacts with the database
  - Whether or not the data services will be kept in the middle tier or in the database
  - Which client and server tools can be used

Foundations: Technology Stack

- As this is a class on enterprise-grade PL/SQL, we'll focus on the backend and assume that your stack involves at least:
  - Modern app server on Windows, linux or the proverbial "cloud" platform
    - Services written in Python, java or C#
    - Data services use JDBC to issue SQL, call ORDS or PL/SQL APIs
    - Data services use JSON to communicate with the front-end
  - Oracle SE or EE Database on Unix or linux
  - Robust, tested, monitored, easily maintained, packaged, framework-driven PL/SQL

Foundations: Everything needs a home

1. Establish a directory structure
2. Install and configure a version control system

Foundations: Directory Structure

- Consider:
  - Organizational structure
  - Project and product structure
  - Team structure
  - Nature of the artifacts produced
    - Organization-wide vs. project-specific

Foundations: Directory Structure

Projects	Exempted from corporate virus checker
project1	One project folder per application
project2	
project3	
dbproject	Separate project folder for all DB work
_Documentation	"_" sorts to top so standards are easy to find
Backup and Recovery	
Data Governance	
Monitoring	
Refreshes	
Security	
Standards	
Builds	Build infrastructure kept here AND on build box
DBA	Stuff that spans databases or schemas
sandbox	Each DBA has their own subfolder here
scripts	
{more}	As you see fit.
Models	One subfolder for all modeling artifacts
{databasename}	One per database name
db	Enforced by our build tool
schema	Enforced by our build tool
schema1	Good to have one subfolder per app schema
schema2	
schema3	
archive	Old "build stream" files moved here
future	Stuff that gets postponed
hotfix	Scripts that do approved prod data fixes
src	"Golden" copy or "tip" of latest source code here. One subfolder per Oracle object type.
test	Unit test cases, scripts and results kept here
utils	Catch-all for useful schema-specific scripts
work	Subfolder per ticket, or per DBA
1803290010-TKT-0010-create_table1.fwd.sql	Example of first DB script in "build stream"
1803290020-TKT-0020-create_view1.fwd.sql	Second script in build stream
...	And so on
always-post.sql	Cleanup, grants, test data, etc. Run by build
schema4	

Foundations: Version Control System

- Many viable products
- I recommend Subversion or Git
  - Also TortoiseSVN
    - Nice integration with Windows Explorer
  - Or TortoiseGit

Foundations: Standards and Conventions

- Technically can be added later in a project, but…
- Management pays for functionality and critical fixes, not to correct ugly, fragile, unmanageable code. They expect that kind of professionalism from the start.
- Far better to have standards in place, and tools to make compliance effortless, BEFORE work begins

Foundation: Standards and Conventions

- Data Design Standard
- Database Development Standard
- Naming Standard
- Build Artifact Standard
- Development Methodology
  - Expectations of analysis, design, coding, testing and release phases
- Team Culture and Workspace

Foundations: Standards and Conventions

- No need to write your own
- Google "PL/SQL Standards"
  - William Robertson
  - Trivadis
  - Steven Feuerstein
  - Coulam
- Adopt one
  - Customize it to your liking
- Ease adoption and enforce use with templates, formatters and peer review.

Foundations: Design Tools

- I recommend ER/Studio and Oracle SQL Developer Data Modeler
  - Oracle's SDDM is now strong enough that it ought to be considered first (especially since it is free)
  - Erwin may be making a comeback now that CA has sold it

Foundations: Development Tools

- Oracle SQL Developer or Allround Automations PL/SQL Developer
- CompareIt or WinMerge for comparing code
- TextCrawler for searching code
- Automated code formatter for beautifying code and complying with standards w/o effort…

Foundations: Formatting Tools

- Should be configurable to match your team's standards and conventions
- Formatting tool found in every PL/SQL IDE. Strongest I've found are:
  - Instant SQL Formatter website
  - SQL Developer's SQL/Oracle Formatter
  - TOAD's Formatter - Great, flexible formatter
  - PL/SQL Developer's Beautifier - 98% there
  - SQL Detective's Code Analyzer - formatter + more
  - DDL Wizard - for messy DDL

Foundations: Testing Tools

- PL/SQL Developer's Test Manager and Data Generator
- SQL Developer's Unit Testing features and Repository
- TOAD's Code Tester
- utPLSQL – enjoying a rebirth and wide use of late

Foundations: Other Tools

- Cygwin - linux terminal power on Windows
- Ditto - (clipboard on steroids)
- WinSCP - open-source SFTP, FTP, WebDAV, Amazon S3 and SCP client
- FreeCommander XE - (two-pane file explorer on steroids)
- KeePass - (or similar password keeper)
- MobaXterm/PuTTY - *nix terminal manager
- Notepad++ - (or similarly powerful text editor)
- OneNote/EverNote – multi-OS, do-everything notekeeper

Milestones for Coding Readiness

How is Your Craftsmanship Perceived by Others?

or

Agenda

- Habits of Great Database Developers
- Foundations
- Design
- Code
- Test
- DevOps
- Maintenance

Design Practices

- Follow basic relational data modeling principles:
  - Create and verify the conceptual model
  - Create and verify the logical model
  - Create and verify the physical model
    - Modeling tool should ensure changes in one flow into the others (if applicable)
  - Document and describe every entity, attribute and relationship
  - Enforce integrity (datatype, relationships, defaults and cardinality)
  - Eliminate duplication wherever found
  - Strive for consistent, simple, flexible models
  - Version, publish and discuss models with all concerned parties, including sponsors

Design Practices

- First make the model correct, efficient, normalized…
- THEN make it friendly
  - Abstract complex queries behind views and PL/SQL APIs that return ref cursors
  - RESTful JSON APIs use the views and PL/SQL APIs
  - Denormalizations like materialized views and rollup/tally tables for performance goals
  - Virtual columns, user-defined types, updateable views and other Oracle goodies to meet system requirements as needed

Design Practices

- Always start design and change design from modeling tool, then forward engineer into the database.
  - Allows changes to the model to be versioned
  - Allows modeling mistakes to be easily seen with visual cues
- Publish designs on the intranet and on paper
  - Distribute to team, management and sponsors
  - Bring them to meetings when discussing enhancements and issues

Agenda

- Habits of Great Database Developers
- Foundations
- Design
- Code
- Test
- DevOps
- Maintenance

Coding Practices: Use the Version Control System

- Ensure the version control system is used religiously.
  - Every database object that begins with the keyword "CREATE" gets its own source file stored and versioned in your directory structure
  - Never modify the database object directly
  - Development and maintenance activities **always** begin with the source file
    - Some PL/SQL IDEs default database object browsing to read-only mode. Use this feature to enforce the above.
  - Once compiled and tested, check in changed file with good comments
  - If development will take a long time, check it in periodically to preserve work

Coding Practices: Use Frameworks and Libraries

- No need to re-invent the wheel.
- Use pre-built, pre-tested PL/SQL frameworks and libraries to save months of coding effort.
  - Used to be a handful of full frameworks. Now we're down to some of Feuerstein's older stuff and mine.
  - I open-sourced my "PL/SQL Starter" framework in 2008 which includes a number of robust libraries, including packages for logging, debugging, emailing, auditing and performance view tagging.
  - One experiment done for a prior presentation on frameworks found that a complex reporting and emailing PL/SQL proc, coded side-by-side with and without a framework, took only 25% as much time and yielded 3X better code using libraries.
- "Alexandria Project" is a list of PL/SQL libraries and utilities

Coding Practices: Routine Design and Structure

- Use PL/SQL packages to group related functionality
- Each routine should do one thing and one thing well
- Keep each routine short, easily read and understood in a few minutes
- Use packaged constants for immutable literals
- Use table-driven parameters for mutable limits, ranges, thresholds and business-rule values
- Break enormous, generic packages apart into separate packages that group related functionality.

Coding Practices: Documentation

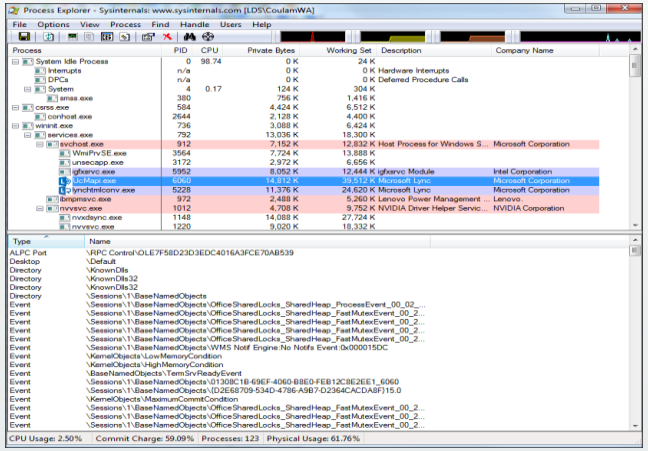
- Documentation is critical to high-quality work products
- Each packaged routine, trigger, view and job should have a detailed comment
- Encourage good descriptions through templates that have comment placeholders

Coding Practices: Documentation

- Fully document each packaged routine in its comment block
  - Comment block in pkg spec if public routine, in body if private routine. NOT BOTH!
  - Focus on "tribal knowledge" that can't be discovered by reading the code:
    - Who wrote it? When? Why? For whom or what system? What was the intended purpose? Were obvious alternatives rejected and why? Caveats, instructions and warnings.
  - Assumptions and expectations about input parameters
  - Return values, exceptions handled and errors raised
  - Usage example if not easy/obvious
- Also document the body of each routine with pseudo-code
  - As the body is built, convert the pseudo-code into debug or log messages and add valuable context

Coding Practices: Instrumentation

Think about some real-world complex software, like Oracle's Enterprise Manager, a modern car, Windows Process Explorer, an aircraft, or a network operations center.



They all have interfaces that offer real-time, valuable insight into the inner workings, performance and historical behavior of their system.

Coding Practices: Instrumentation

And yet this is how much insight and metrics most database engineers build into their enterprise-grade code:





None! You get to fly blind.

Coding Practices: Instrumentation

Instrumentation is the act of adorning systems with code that directs runtime context to a destination where it can be useful.

Types of instrumentation:

- Logging
- Debugging
- Notifying
- Auditing

Coding Practices: Instrumentation

Useful runtime context includes:

- Who: Name of called routine, metadata of caller, etc.
- What: Parameters passed, variable values, iteration values, etc.
- When: DATE or TIMESTAMP
- What changed with old/new values (audit trail)
- Metrics (timings, counters and sums)
- Exceptions, warnings, errors
- Landmarks and breadcrumbs (great for complex or long-running routines)

Coding Practices: Instrumentation

Useful destinations include:

- Logging table
- Email
- SMS Text
- Web service

Other destinations include:

- DBMS_PIPE, DBMS_AQ, DBMS_ALERT, DBMS_SYSTEM, UTL_FILE, DBMS_OUTPUT, DBMS_APPLICATION_INFO, DBMS_SESSION, ftp

Coding Practices: Instrumentation

- Should not have to build your own
- There are many free and open-source instrumentation libraries
  - My "PL/SQL Starter" framework includes packages for logging, debugging, emailing, auditing and performance view tagging.
  - Tyler Muth's forked "Logger" now found on github, is quite versatile
    - Local Oracle Advocate, Blaine Carter, has presentations and videos on using Logger
- Adopt a good library that is simple to use and meets your needs
- Can be added incrementally or all at once

Coding Practices: Performance & Resilience

- If it can be done in a single SQL statement, do so
- If not, use bulk PL/SQL features
  - BULK COLLECT, FORALL, etc.
- If it must be done row-by-row, use record-based PL/SQL
  - Records or user-defined objects and input parameter type
  - Record-based inserts and updates
  - FORALL can be used on collections of type RECORD

Coding Practices: Exceptions

- Ban the use of WHEN OTHERS
  - The only exception is when the error must be hidden
- Write handlers only for **anticipated** exceptions.
  - Use a standard way of logging and re-raising handled exceptions
- Allow PL/SQL's default exception model to raise and rollback every unanticipated exception
- Use pre-defined and user-defined exceptions.
  - Internally-defined exceptions should be abstracted with user-defined named exceptions

Coding Practices: Exceptions

Internally Defined

```
CREATE OR REPLACE FUNCTION table_analyzed(
   i_tbl_nm IN VARCHAR2
) RETURN DATE IS
   l_last_analyzed DATE;
   l_tbl_nm        VARCHAR2(30);
BEGIN
   l_tbl_nm := UPPER(i_tbl_nm);

   BEGIN
      SELECT last_analyzed
        INTO l_last_analyzed
        FROM user_tables
       WHERE table_name = l_tbl_nm;
   EXCEPTION
      WHEN OTHERS THEN
         IF SQLCODE = -1403 THEN
            dbms_output.put_line('ERROR(-1403): Table '||
               l_tbl_nm || ' not found.');
            RAISE;
         END IF;
         IF SQLCODE = 100 THEN
            dbms_output.put_line('ERROR(100): Table '||
               l_tbl_nm || ' not found.');
            RAISE;
         END IF;
   END;
   RETURN l_last_analyzed;
END;
```

Bad

Bad but correct

Predefined

```
CREATE OR REPLACE FUNCTION table_analyzed(
   i_tbl_nm IN VARCHAR2
) RETURN DATE IS
   l_last_analyzed DATE;
   l_tbl_nm        VARCHAR2(30);
BEGIN
   l_tbl_nm := UPPER(i_tbl_nm);

   BEGIN
      SELECT last_analyzed
        INTO l_last_analyzed
        FROM user_tables
       WHERE table_name = l_tbl_nm;
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         dbms_output.put_line('ERROR: Table ' ||
            l_tbl_nm || ' not found.');
   END;
   RETURN l_last_analyzed;
END;
```

Better

User Defined

```
CREATE OR REPLACE FUNCTION table_analyzed(
   i_tbl_nm IN VARCHAR2
) RETURN DATE IS
   l_last_analyzed  DATE;
   l_tbl_nm         VARCHAR2(30);
   lx_table_missing EXCEPTION;
   PRAGMA EXCEPTION_INIT(lx_table_missing, 100);
BEGIN
   l_tbl_nm := UPPER(i_tbl_nm);

   BEGIN
      SELECT last_analyzed
        INTO l_last_analyzed
        FROM user_tables
       WHERE table_name = l_tbl_nm;
   EXCEPTION
      WHEN lx_table_missing THEN
         dbms_output.put_line('ERROR: Table ' ||
            l_tbl_nm || ' not found.');
   END;
   RETURN l_last_analyzed;
END;
```

Best

Coding Practices: Transactions

- The caller is in charge of transaction commit/rollback decisions
  - This is typically a java class, so **most PL/SQL should never commit or rollback on its own**
  - If the caller is a database job, the PL/SQL block driving the job makes the decision to commit or rollback

Agenda

- Habits of Great Database Developers
- Foundations
- Design
- Code
- Test
- DevOps
- Maintenance

Testing Practices

- Best testing occurs while documenting the interface
  - Write assertions in the body that test the validity of those assumptions (known as code-by-contract)
- Test-Driven Development
  - Write tests of the interface before writing the implementation
  - Write body, re-testing all cases as you code to reach requirements
  - Left with nice suite of re-usable tests
    - Re-run the test suites whenever the code changes – able to quickly prove the change hasn't adversely affected anything

Testing Practices

- Use testing frameworks to help automate tests and the creation of test suites
  - utPLSQL is probably your best option as a framework
    - Various videos, webinars and tutorials available on how to use it
    - CodeTalk Series: Unit Testing PL SQL Code in the Real World
    - March 20th, 2018 ProHuddle webinar on utPLSQL by Jacek Gebal

Testing Practices: The Conundrum

- Most enterprise PL/SQL routines use a lot of complex, interrelated data or do a lot of complex stuff.
- Writing re-usable tests involves controlled data conditions that may not be present the next time the test is used. So one must write "setup" and "tear-down" scripts that create test data specifically for a given test.
- Writing data setup and tear-down scripts can easily occupy 60 to 80% of the development effort.
- So re-usable test suites are often skipped
- Nirvana: Someday, using tools like Delphix, the entire database state will be saved in a repository for single or multiple test conditions and that state can be instantiated in seconds.

Agenda

- Habits of Great Database Developers
- Foundations
- Design
- Code
- Test
- DevOps
- Maintenance

DevOps

- What is DevOps? What does it include?
- Build (compile or run SQL statements against target database)
  - SQL should be scripted. Can include DCL, DDL or DML
- Deploy/Release (pre-scripts/tasks, build to Prod, post-scripts/tasks)
- Manage Issues and Enhancements
  - We use Jira and Jama. There are many, many others.
- Enterprise-grade DevOps use automation to handle these tasks
  - We use a custom build system using Maven and CruiseControl that is awesome
  - Liquibase and FlywayDB embody many of the same features and principles as our system.

DevOps: Lessons Learned

- Folder per object-owning schema
- Hook to notify everyone of new check-ins
- Prefix all files with YYYYMMDD####-TICKET#-description prefix
- The build stream is incremental. Each script is a new "version" of the DB.
  - Don't change scripts once checked in. Instead fix them with another script that will run later.
- Ensure each script is re-entrant
  - Aka re-runnable or [buzzword alert!] "idempotent"

Agenda

- Habits of Great Database Developers
- Foundations
- Design
- Code
- Test
- DevOps
- Maintenance

Maintenance

- Traditional maintenance has mostly been folded into DevOps, where the same developers are responsible for the system front-to-back, from design to build to deployment to issue identification, documentation and resolution.
- When things go wrong in Production, switch on debug statements for PL/SQL unit, authenticated user, Oracle session or schema
  - Should never have to recompile code in Production to get runtime context
  - Shouldn't take more than a few minutes to identify root cause of issue
- Agree upon and use tool to track and manage bugs, change requests and release bundles
  - We use Jama and Jira

Maintenance: Lessons Learned

- Refresh production frequently, nightly if possible to continuous build DB box
- Run proposed production data fixes on the refreshed copy first
- Communicate well about future downtime for releases and PL/SQL compilation “hiccups”
- Ensure each DB script intended for release or hotfix is tagged in some way to associate with the change management tool holding the problem description.
- Write lots of proactive monitoring scripts and email/SMS DBAs when nasty errors are detected. DBAs should be aware of the problem before the customers are. Seek budget to improve the issues that waste the most time.

Day-in-the-Life Demos

- <time permitting>

What Will Be Your Design & Build Legacy?

OR

Contact and Further Info

Contact: bcoulam@yahoo.com

Papers and Code:

- http://www.dbartisans.com
- http://www.dbsherpa.com

Framework:

- http://sourceforge.net/projects/plsqlframestart
- http://github.org/plsqlstarter