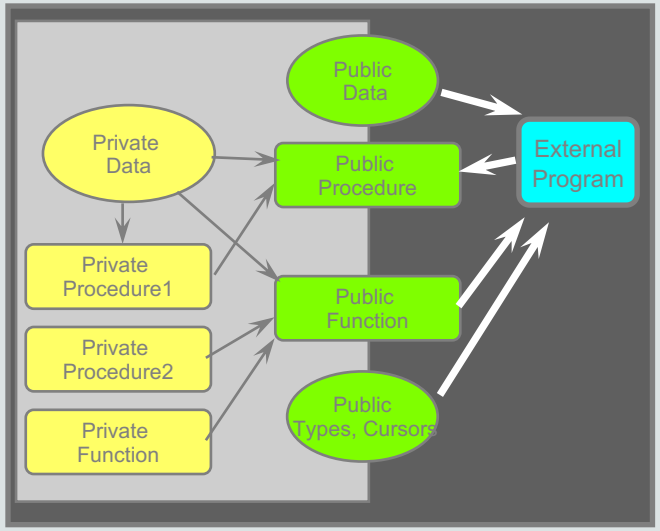
Building APIs in PL/SQL



Steven Feuerstein
Developer Advocate for PL/SQL
Oracle Corporation

Email: steven.feuerstein@oracle.com
Twitter: @sfonplsql
Blog: stevenfeuersteinonplsql.blogspot.com
YouTube: Practically Perfect PL/SQL

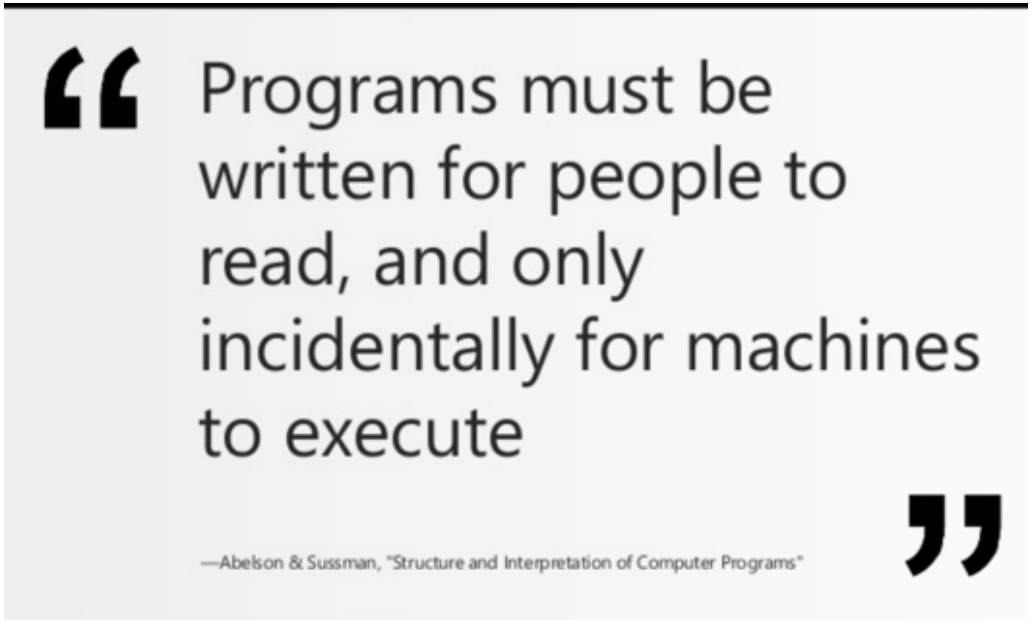
Resources for Oracle Database Developers

- Official homes of SQL and PL/SQL - oracle.com/sql oracle.com/plsql
- Dev Gym: quizzes, workouts and classes - devgym.oracle.com
- Ask Tom - asktom.oracle.com – 'nuff said (**+ new: *Office Hours*!**)
- LiveSQL - livesql.oracle.com – script repository and 24/7 18c database
- SQL-PL/SQL discussion forum on OTN https://community.oracle.com/community/database/developer-tools/sql_and_pl_sql
- oracle-base.com – "best of the web" content from Tim Hall
- oracle-developer.net - great content from Adrian Billington
- And never underestimate the greatest resource and motivator of them all….

API: Application Programming Interface

- Defines what is possible to do in a given subject area of an application
- Provides a set of procedures or functions to perform operations
- Hides the underlying details (implementation) from users of the API

Why Build APIs?



http://bit.ly/lukasederapi

Why Build APIs?

- You almost don't really have a choice.
  - Distributed, serverless, microservice computing is in full swing
- Make it easier for others to use your code
- Increase the reliability of the application
- Provide access to your data without compromising security or performance
- Manage business logic more effectively

Building APIs for UI Developers

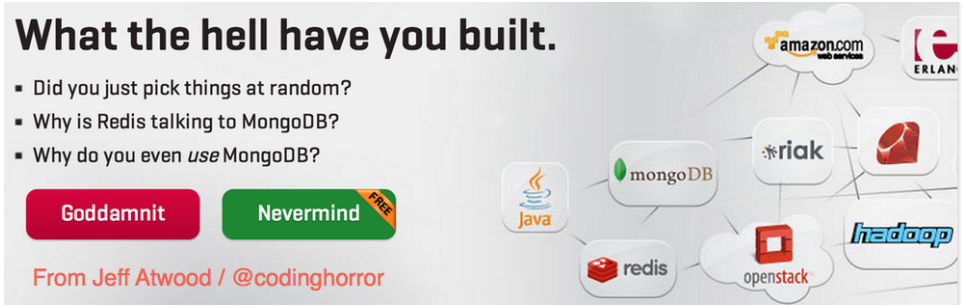
We all want the same thing, right? A successful app!

- It's not the 1990s anymore.
- So much has changed in the world of software – and how we deliver it.
  - **Consumers are now MVU – The Most Valuable User.**
  - Apps are delivered on tiny screens with YUGE resolution.
  - Users get to *choose* their apps.
  - Training? Fuggedaboudit! It just needs to work/make sense.
  - If it's not free, I'm not paying for it.
  - *Eventually Consistent* is good enough for me.

We've Got It (Relatively) Easy

There's a reason for the Framework Insanity of JavaScript

- User interfaces are tied directly and tightly to culture. Uh oh.
- Lots and lots of code (compared to, say, APEX)
- Microservices, bots, containers, asynchronous communication....
- Endless demand for changes to UIs, since we need to hide all that ever-increasing complexity



So how do we communicate with our fellow developers?

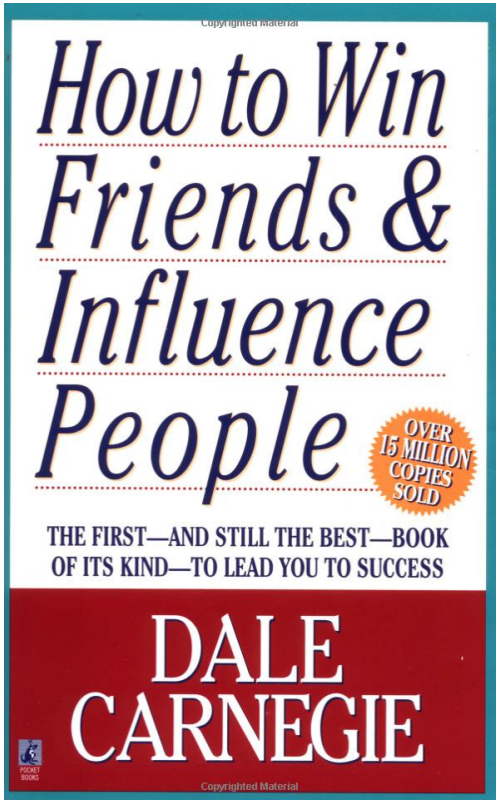
We treat them like they're dumb and we are wise.

I’ve been doing this for 20 years.
I've seen firsthand the consequences of
your ignorance and impatience.
Five years from now,
your life is going to be pure agony
as you wrestle
with data integrity and consistency issues.

Change the Message to *How can I help?*

What do you have to lose? ☺

- Don't be oppositional. Never say "No".
- Don't point out where others are *wrong.*
  - Better to admit *you* are wrong.
- Find developer pain points. These come to mind:
  - Performance of DB access
  - Headaches wrestling with SQL
  - Needs JSON-based APIs
- Then offer solutions, of which you have lots.



We can help UI developers – a LOT.

And 12.2 makes it easier than ever before.

- You want APIs? We've got the best data APIs!
  - PL/SQL is *the* best performing, most secure and productive language for creating APIs to the database, through packages.
- You want JSON?
  - Oracle Database offers native JSON support via SQL and PL/SQL, and it gets better with each release.
- You will only talk REST?
  - No problem.
  - Easy, secure REST APIs (often generated) through Oracle REST Data Services

Types of APIs

- Transactional business function API (XAPI)
- Query-only business function API (QAPI)
- Table-level APIs for basic CRUD operations (TAPI)
- Consolidated rule/ business logic encapsulation (CRAPI)
- REST APIs
- UI APIs (JavaScript, Python, etc.)

How to Build APIs in PL/SQL

- Write packages.

- Yes, you could also use object types, but I prefer and suggest packages.

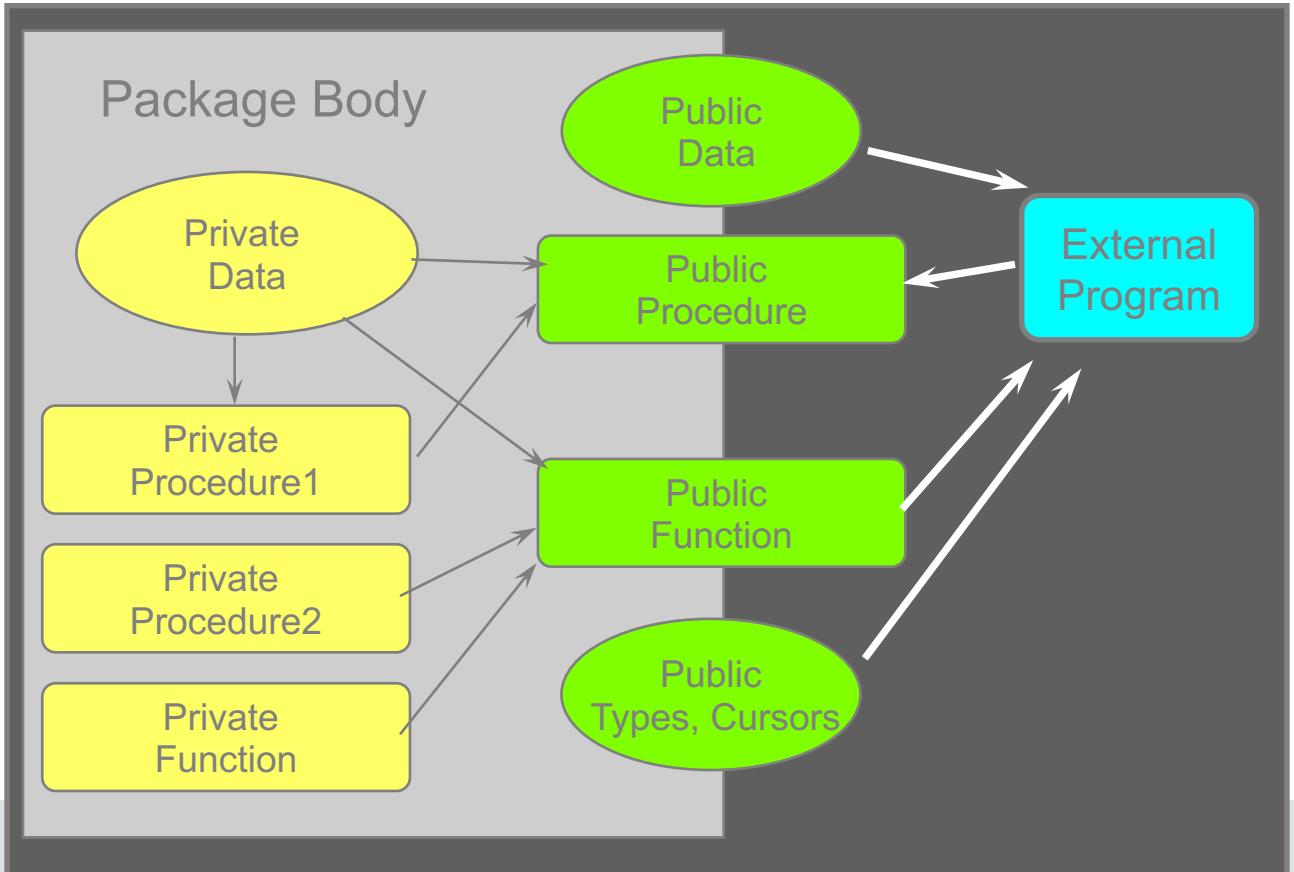
- Let's explore the why and how.

Why Packages

- Easier code management
  - Group related functionality (namespace), grant privileges to package
- Hide implementations
  - Too much information can be a really bad thing for developers
- Minimize unit invalidations
  - Schema-level procedures and functions, not so much
- Overload subprograms
  - One of the best "clean API" features

A Package in Pictures

- Specification is public
- Body is private



Overloading

- Multiple subprograms with same name
  - Different parameter list, different subprogram type
- Moves the burden of knowledge from API user to API author
  - "Do X" and we "do the right thing"
- Any declaration section can have overloadings
  - Used mostly in packages
- Careful implementation critical to avoid a big mess
  - DRY, SPOD, Code normalization

Quiz!

- What's another name for overloading?

Dynamic Polymorphism

Interface Inheritance

Multiple Monomorphism

Static Polymorphism

Quiz!

```
DBMS_OUTPUT.PUT_LINE ('Hello World');
DBMS_OUTPUT.PUT_LINE (SYSDATE);
DBMS_OUTPUT.PUT_LINE (3.14);
DBMS_OUTPUT.PUT_LINE (SYSTIMESTAMP);
```

- How many overloadings of PUT_LINE are defined in the DBMS_OUTPUT package?

1 **2** **3** **4**

How Overloading Works

- PL/SQL resolves references at *compile-time*; there are two:
  - When the package itself is compiled
  - When you try to *use* the overloaded subprogram
- At either point, the compiler could raise one of the following:

What the Compiler Needs

- Good to go
  - Formal parameters of overloaded modules must differ in number, order or datatype family
  - The programs are of different types: procedure and function.
- Unacceptable
  - Functions differ only in their RETURN datatype
  - Arguments differ only in their mode (IN, OUT, IN OUT)
  - Formal parameters differ only in datatypes that are in the same family

Quiz!

- Is it possible to write a package that contains 2 or more subprograms so that it compiles successfully but cannot be used?

Yes **No**

Tips for Overloading

- Make sure the differences between overloadings are obvious
  - Just because it compiles, doesn't mean it's OK
- Do not require named notation to avoid ambiguity
  - Even if named notation is a recommended style
- Avoid duplication of code
  - The names are the same – in most cases the implementations are at least very similar.

Careful, Consistent Error Handling

- Decide on a single strategy for error handling, logging and re-raising
  - Never allow an exception to propagate unhandled out of the database?
  - Handle exceptions deep in the call stack or only at top level?
- Use a single error logging API (package!) across all your packages
  - Check out oraopensource Logger if you don't already have one

Writing High Quality APIs

- Consistency
  - Naming conventions, error handling, transaction management (autonomous transactions)
- Overload for native PL/SQL and SQL use
  - No Booleans in SQL!
- Tight focus, manageable size
  - Who wants to wait a minute to recompile?


ORACLE®