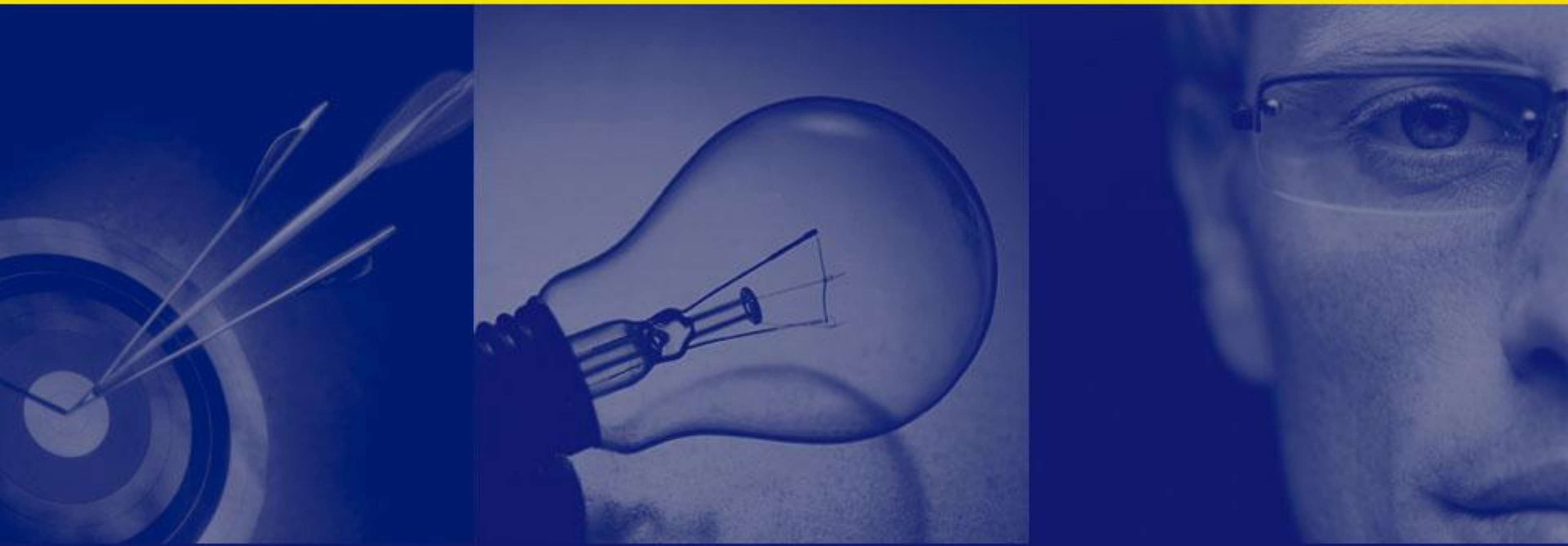
Best Practices for Writing SQL in PL/SQL



Steven Feuerstein
PL/SQL Evangelist, Quest Software
www.ToadWorld.com/SF
www.plsqlchallenge.com
steven.feuerstein@quest.com

How to benefit most from this session

- Watch, listen, *ask questions*. Then afterwards....
- Download and use any of my the training materials, available at my "cyber home" on Toad World, a portal for Toad Users and PL/SQL developers:

PL/SQL Obsession **http://www.ToadWorld.com/SF**

- Download and use any of my scripts (examples, performance scripts, reusable code) from the demo.zip, available from the same place.

filename_from_demo_zip.sql

- You have my permission to use *all* these materials to do internal trainings and build your own applications.
  - But they should not considered production ready.
  - You must test them and modify them to fit your needs.

And some other incredibly fantastic and entertaining websites for PL/SQL

Covers Versions Through Oracle Database 11g Release 2
5th Edition
Oracle PL/SQL
Programming
O'REILLY
Steven Feuerstein with Bill Pribyl

Best Practices for Writing SQL in PL/SQL

- Set standards and guidelines for writing SQL.
- Take full advantage of the SQL language.
- Hide SQL statements behind an interface.
- Hide all tables in schemas users cannot access.
- Qualify every identifier in SQL statements.
- Dot-qualify references to Oracle-supplied objects.
- Use SELECT INTO for single row fetches.
- Always BULK COLLECT with LIMIT clause.
- Always use FORALL for multi-row DML.
- Use collection and TABLE operator for IN clauses of indeterminate count.
- Avoid implicit conversions.
- Key dynamic SQL best practices

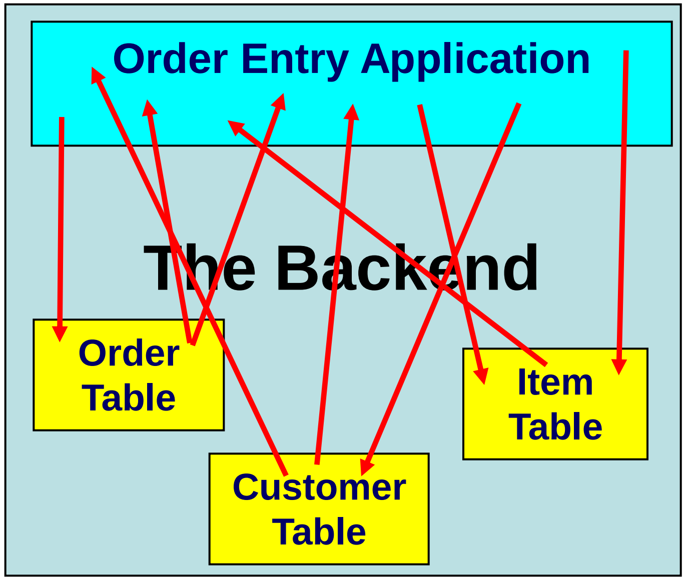
Set standards and guidelines for SQL

- **Many organizations have coding standards.**
  - How to format code, how to name programs and variables, etc.
- **Very few development teams have standards for how, when and where to write SQL statements.**
  - We just all take SQL for granted.
- **This is very strange and very dangerous.**

Why lack of standards for SQL is dangerous

- **SQL statements are among the most critical elements of our applications.**
- **SQL statements reflect our business model.**
  - And those models are always changing.
- **SQL statements cause most of the performance problems in our applications.**
  - Tuning SQL and the way that SQL is called in PL/SQL overwhelms all other considerations.
- **Many runtime errors result from integrity and check constraints on tables.**

The fundamental problem with SQL in PL/SQL



- We take it entirely for granted.
  - Why not? It's so easy to write SQL in PL/SQL!
- As a result, our application code is packed full of SQL statements, with many repetitions and variations.
  - Worst of all: SQL in .Net and Java!
- This makes it very difficult to optimize and maintain the application code.

So set some SQL standards!

- **At a minimum, before starting next application, ask yourselves explicitly:**
  - Do we want standards or should we just do whatever we want, whenever we want?
  - That way, you are making a conscious decision.
- **This presentation as a whole forms a reasonable foundation for such standards.**
- **Another excellent resource (and source for this presentation):**

Doing SQL from PL/SQL: Best and Worst Practices
by Bryn Llewellyn, PL/SQL Product Manager

http://www.oracle.com/technology/tech/pl_sql/pdf/doing_sql_from_plsql.pdf

Fully leverage SQL in your PL/SQL code

- **Oracle continually adds significant new functionality to the SQL language.**
- **If you don't keep up with SQL capabilities, you will write slower, more complicated PL/SQL code than is necessary.**
  - I am actually a good example of what you *don't* want to do or how to be.
- **So take the time to refresh your understanding of Oracle SQL in 10g and 11g.**

Some exciting recently added SQL features

- **Courtesy of Lucas Jellama of AMIS Consulting**
- **Analytical Functions**
  - Primarily LAG and LEAD; these allow to look to previous and following rows to calculate differences)
- **WITH clause (subquery factoring)**
  - Allows the definition of 'views' inside a query that can be used and reused; they allow procedural top-down logic inside a query
- **Flashback query**
  - No more need for journal tables, history tables, etc.
- **ANSI JOIN syntax**
  - Replaces the (+) operator and introduces FULL OUTER JOIN
- **SYS_CONNECT_BY_PATH and CONNECT_BY_ROOT for hierarchical queries**
- **Scalar subquery**
  - Adds a subquery to a query like a function call.

```
select d.deptno
     , (select count(*)
          from emp e where
e.deptno = d.deptno)
number_staff from dept
```

Hide SQL statements behind an interface

- **You, of course, need to write SQL in Oracle applications.**
- **And PL/SQL is the best place to write and store the SQL statements.**
- **But we must stop writing SQL statements *all over* the application code base.**
  - Repetition of SQL is a real nightmare.
- **The best way to understand this is to accept a harsh reality:**

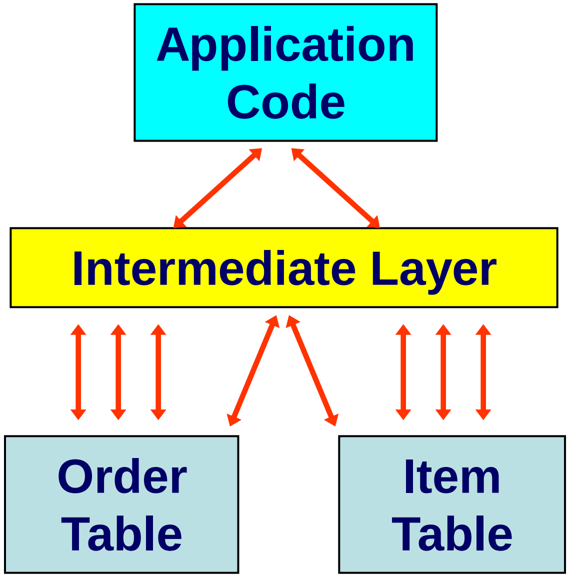
Every SQL statement you write is a hard-coding that is *worse* than a hard-coded literal.

SQL as Hard-Coding....huh?

- **We all agree that hard-coding "magic values" is a bad idea.**
  - When value changes (and it will), you must find all occurrences and update them.
- **But SQL statements suffer from the same problem.**
  - When you write SQL, you are saying "Today, *at this moment*, this is the complex code needed to describe this dataset."
- **There is no logical distinction between a magic value and a "magic query."**
  - Both will change, both should not be repeated.

SQL as a Service

- **Think of SQL as a *service* that is provided to you, not something you write.**
  - Or if you write it, you put it somewhere so that it can be easily found, reused, and maintained.
- **This service consists of *programs* defined in the data access layer.**
  - Known as table APIs, transaction APIs, or data encapsulation, these programs contain all the intelligence about business transactions and underlying tables.



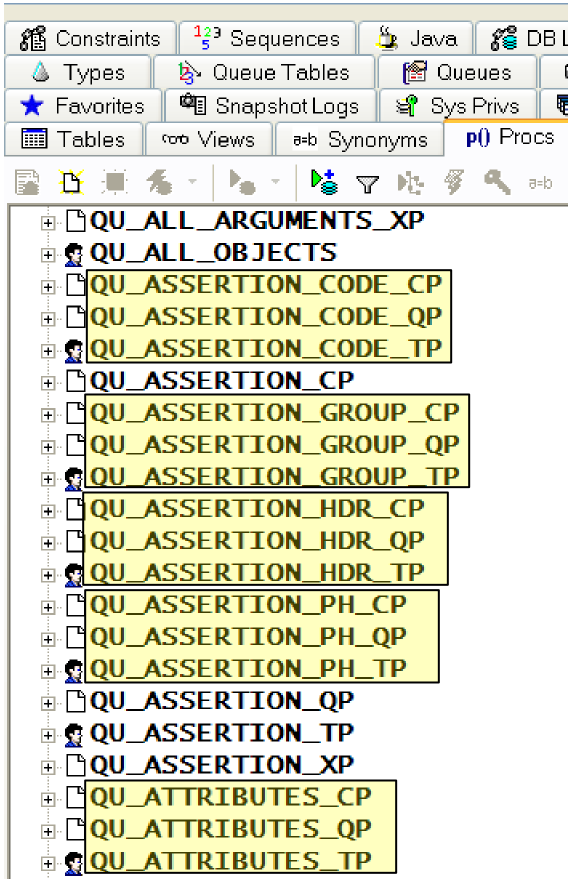
With encapsulated SQL I can...

- **Change/improve my implementation with minimal impact on my application code.**
  - – The underlying data model is constantly changing.
  - – We can depend on Oracle to add new features.
  - – We learn new ways to take advantage of PL/SQL.
- **Vastly improve my SQL-related error handling.**
  - – Do you handle dup_val_on_index for INSERTs, too_many_rows for SELECT INTOs, etc?
- **Greatly increase my productivity**
  - – I want to spend as much time as possible implementing business requirements.

11g_emplu.pkg

Example: Quest Code Tester backend

- **For each table, we have three generated packages:**
  - <table>_CP for DML
  - <table>_QP for queries
  - <table>_TP for types
- **And usually an "extra stuff" package (_XP) with custom SQL logic and related code.**
  - You can't generate everything.



QU_ALL_ARGUMENTS_XP
QU_ALL_OBJECTS
QU_ASSERTION_CODE_CP
QU_ASSERTION_CODE_QP
QU_ASSERTION_CODE_TP
QU_ASSERTION_CP
QU_ASSERTION_GROUP_CP
QU_ASSERTION_GROUP_QP
QU_ASSERTION_GROUP_TP
QU_ASSERTION_HDR_CP
QU_ASSERTION_HDR_QP
QU_ASSERTION_HDR_TP
QU_ASSERTION_PH_CP
QU_ASSERTION_PH_QP
QU_ASSERTION_PH_TP
QU_ASSERTION_QP
QU_ASSERTION_TP
QU_ASSERTION_XP
QU_ATTRIBUTES_CP
QU_ATTRIBUTES_QP
QU_ATTRIBUTES_TP

How to implement data encapsulation
(After all, I did promise to be practical!)

- **It must be very consistent, well-designed and efficient - or it will not be used.**
- **Best solution: generate as much of the code as possible.**
  - And any custom SQL statements should be written once and placed in a standard *container* (usually a package).
- **One option for generating table APIs for use with PL/SQL is the freeware Quest CodeGen Utility, available at PL/SQL Obsession:**

www.ToadWorld.com/SF

Encapsulating Data Retrieval

- **Simplify query access with views**
  - Another form of encapsulation
- **Hide queries behind functions**
  - Return cursor variable to non-PL/SQL host environment.
  - Return collection or record to other PL/SQL programs
- **Use table functions to encapsulate complex data transformations**

Encapsulating DML statements

- **Hiding inserts are relatively straightforward**
  - Insert by record, collection, individual columns
- **Encapsulating updates is more challenging.**
  - Many variations
  - Some choose a blend of dynamic and static SQL
  - Others use a parallel "indicator" argument to specify which of the columns should be included in the update.
  - You *will* write your own custom encapsulators.

Before you start your next application...

- **Sit down as a team and decide what you are going to do about SQL.**

Choice #1. Keep doing what you've been doing (everyone writes SQL wherever and whenever they want.

Choice #2. Full encapsulation: no directly granted privileges on tables, only access path is through API.

Choice #3. Encapsulate most important tables and run validations against code to identify violations of the API.

Choice #4. Encapsulate queries to prepare for upgrade to Oracle11g and the function result cache.

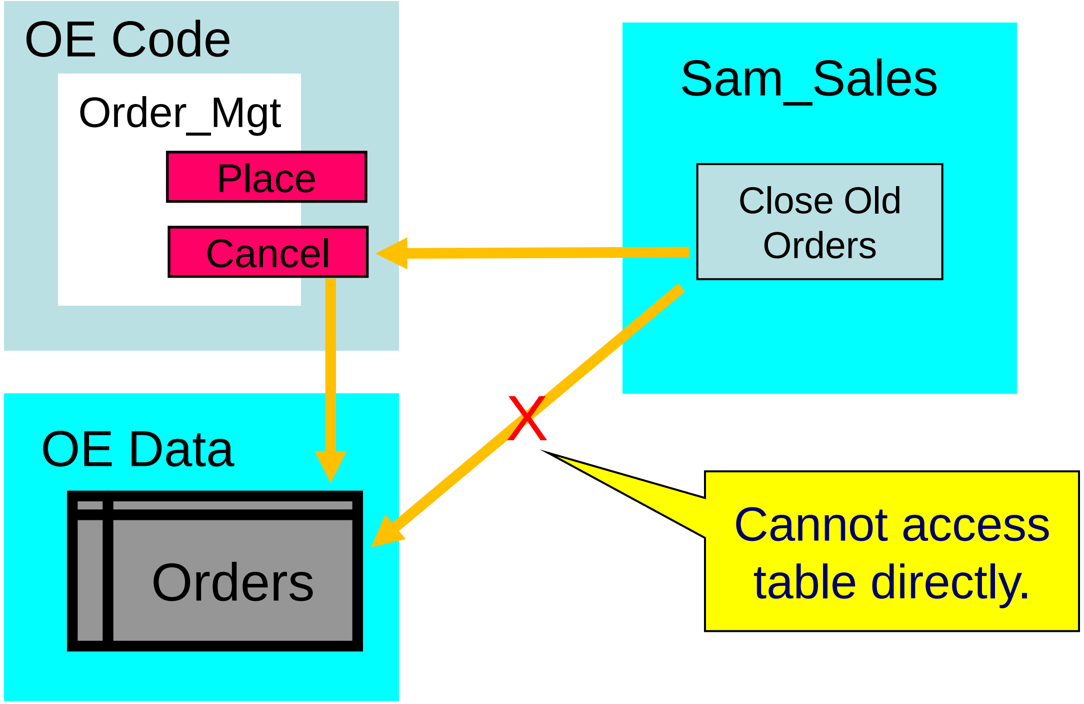
code_referencing_tables.sql

Hide all tables in schemas users cannot access.

- **A fundamental issue of control and security.**
- **Do not allow users to connect to any schema that contains tables.**
  - Simply too risky.
- **Define tables in other schemas.**
- **Grant access via privileges, mostly via EXECUTE on packages that maintain the tables.**

Architecture with inaccessible schema for data

- **The OE Data schemas own all tables.**
- **The OE Code schema owns the code and has directly granted privileges on the tables.**
- **User schemas have execute authority granted on the code.**



Cannot access table directly.

Copyright © 2010 Steven Feuerstein

Qualify every column and identifier in the SQL statement.

- **Improves readability.**
- **Avoids potential bugs when variable names match column names.**
- **Minimizes invalidation of dependent program units in Oracle11g.**

11g_fgd*.sql

Instead of this....

```
PROCEDURE abc (...)
IS
BEGIN
   SELECT last_name
     INTO l_name
     FROM employees
    WHERE employee_id = employee_id_in;
```

Write this....

```
PROCEDURE abc (...)
IS
BEGIN
  SELECT e.last_name
    INTO l_name
    FROM employees e
   WHERE e.employee_id = abc.emp_id_in;
```

Dot-qualify all references to Oracle-supplied objects with "SYS."

- **Another annoying, but incontestable recommendation.**
- **If you don't prefix calls to all supplied packages with "SYS.", you are more vulnerable to *injection*.**

```
BEGIN
   run_dynamic_plsql_block
      (append_this_in =>
         'employee_id=101; EXECUTE IMMEDIATE
         ''CREATE OR REPLACE PACKAGE DBMS_OUTPUT ... '''
      );
END;
```

Use SELECT INTO for single row fetches.

- **Long ago, Oracle "gurus" warned against SELECT INTO (implicit query) and pushed explicit cursors for single row fetches.**
- **Then Oracle optimized SELECT INTO, so that implicits are generally faster than explicits.**
  - In Oracle11, the difference is small.
- **The most important thing to do, however, is to *hide* your query inside a function.**
  - So when Oracle changes the picture again, you only have to adjust your code in one place (for each query).

expl_vs_impl.sql

Always BULK COLLECT with LIMIT clause.

- **First, always use BULK COLLECT to retrieve multiple rows of data.**
  - Note: "Read only" (no DML) cursor FOR loops are automatically optimized to array performance.
- **With BULK COLLECT into a varray if you *know* there is a maximum limit on the number of rows retrieved.**
- **Otherwise, use the LIMIT clause with your BULK COLLECT statement.**
  - Avoid hard-coding: LIMIT can be a variable or parameter.

bulk*coll*.sql

Always use FORALL for multi-row DML.

- **Convert all loops containing DML statements into FORALL statements.**
  - Incredible boost in performance.
- **The conversion process can be tricky and complicated.**
  - Use SAVE EXCEPTIONS or LOG ERRORS to continue past errors.
  - Use INDICES OF and VALUES OF with sparse collections

forall*.sql

Use collection and TABLE operator for IN clauses of indeterminate count.

- **The IN clause may contain no more than 1000 elements.**
- **Several options for "dynamic" IN clause:**
  - Dynamic SQL
  - IN clause with TABLE operator
  - MEMBER OF
- **Using a collection with the TABLE operator offers best flexibility and performance.**
  - Must be declared at schema level.

in_clause*.sql

Avoid implicit conversions

- **Oracle is very forgiving.**
  - If it can implicitly convert a value from one datatype to another, it will do it without complaint.
- **There is, however, a price to pay.**
  - Implicit conversions can affect optimization of SQL statements.
  - There is overhead to the conversion that is best avoided.
- **So whenever possible....**
  - Use correct datatypes to avoid need to convert.
  - Rely on explicit rather than implicit conversions.

explicit_implicit.sql

Key Dynamic SQL Best Practices

- **Always EXECUTE IMMEDIATE a variable.**
  - Otherwise it will be very difficult to debug those complicated strings.
- **Stored programs with dynamic SQL should be AUTHID CURRENT_USER.**
  - Make sure the right DB objects are affected.
- **Dynamic DDL programs should be autonomous transactions.**
  - Watch out for those implicit commits!
- **Minimize the possibility of SQL injection.**

dropwhatever.sp

SQL (code) Injection

- **"Injection" means that unintended and often malicious code is inserted into a dynamic SQL statement.**
  - Biggest risk occurs with dynamic PL/SQL, but it is also possible to subvert SQL statements.
- **Best ways to avoid injection:**
  - Restrict privileges tightly on user schemas.
  - Use bind variables whenever possible.
  - Check dynamic text for dangerous text.
  - Use DBMS_ASSERT to validate object names, like tables and views.

code_injection.sql
sql_guard.*
dbms_assert_demo.sql

usebinding.sp
toomuchbinding.sp
useconcat*.*
ultrabind.*

Best Practices for Writing SQL in PL/SQL

- **Stop taking SQL for granted.**
  - The most important part of your application code base.
- **Fully utilize the SQL language.**
  - If you can do it in SQL, don't complicate with PL/SQL.
- **Avoid repetition of SQL statements.**
  - At a minimum, hide queries inside functions to prepare for the function result cache.
- **Take advantage of collections for flexibility and performance .**
  - FORALL and BULK COLLECT
  - IN clause flexibility