PL/SQL 101: 5+ Basic Tips for Highly-performant Code

Peter Koletzke

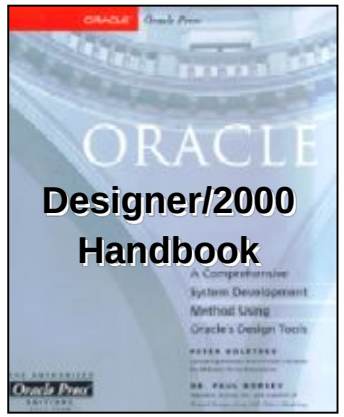
Technical Director & Principal Instructor

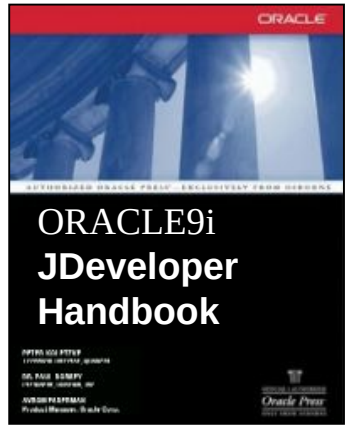
Oracle ACE*x?*

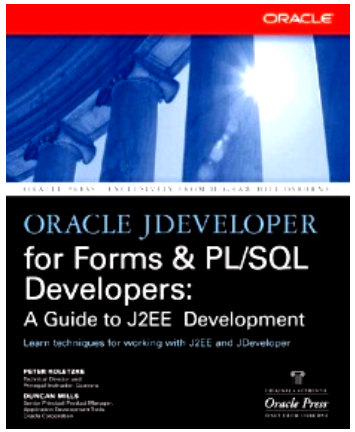
NYOUG

CV

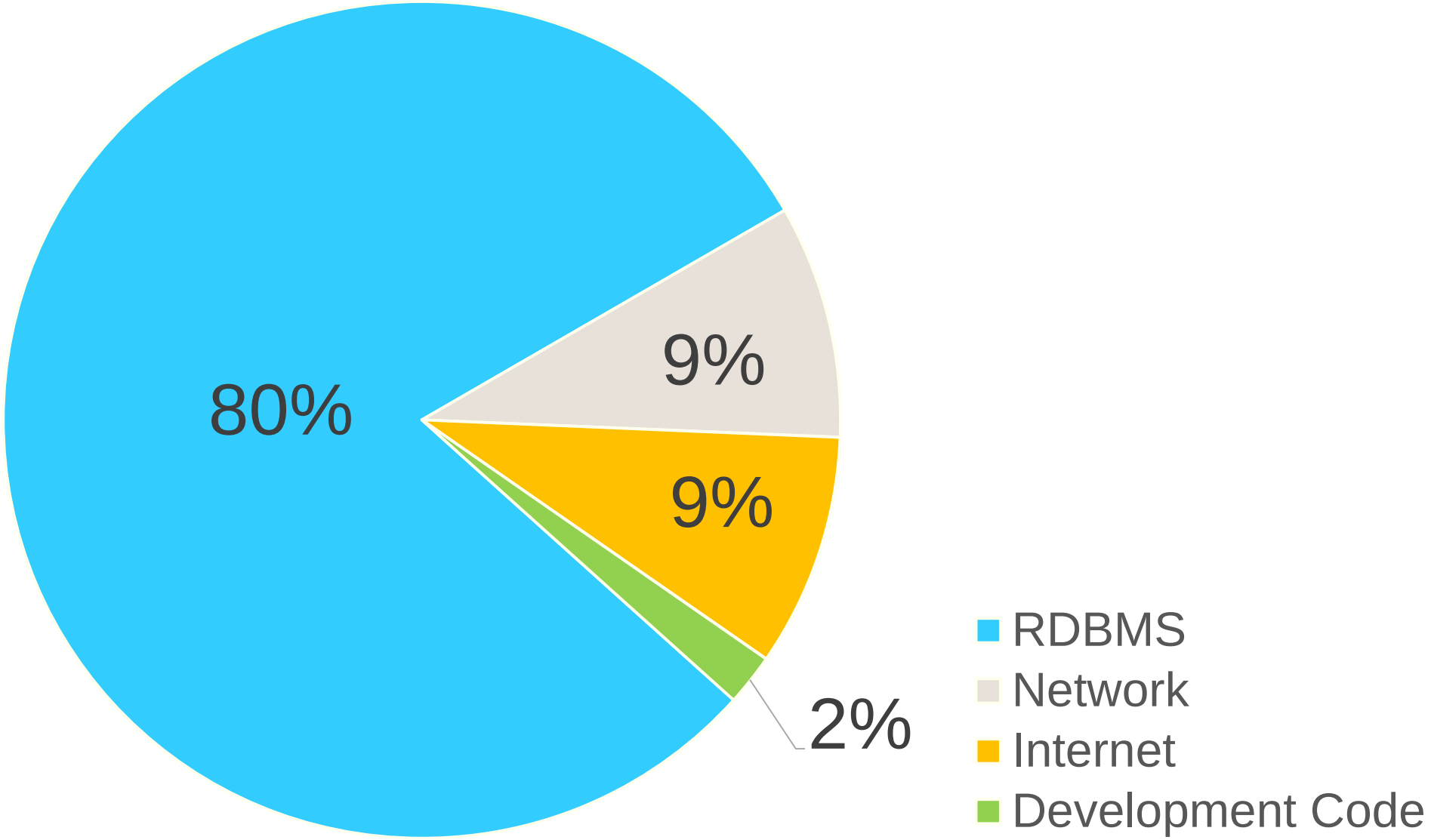
- 38 yrs. database industry
- 31 yrs. consulting in Oracle arena
  - Since Oracle 5.1C, SQL*Forms 2.3
- 41 yrs. as trainer/presenter
- User groups
  - 390+ presentations, 12 awards
  - 11+ yrs. total on boards of directors
    - IOUG(-A), NYOUG, UTOUG
- Oracle ACE Director 2005-?
  - 2022 qualification is in process
- Oracle Certified Master
  - At program inception in Dec. 2001
- 8 Oracle Press books coauthored
  - 6262 pages total



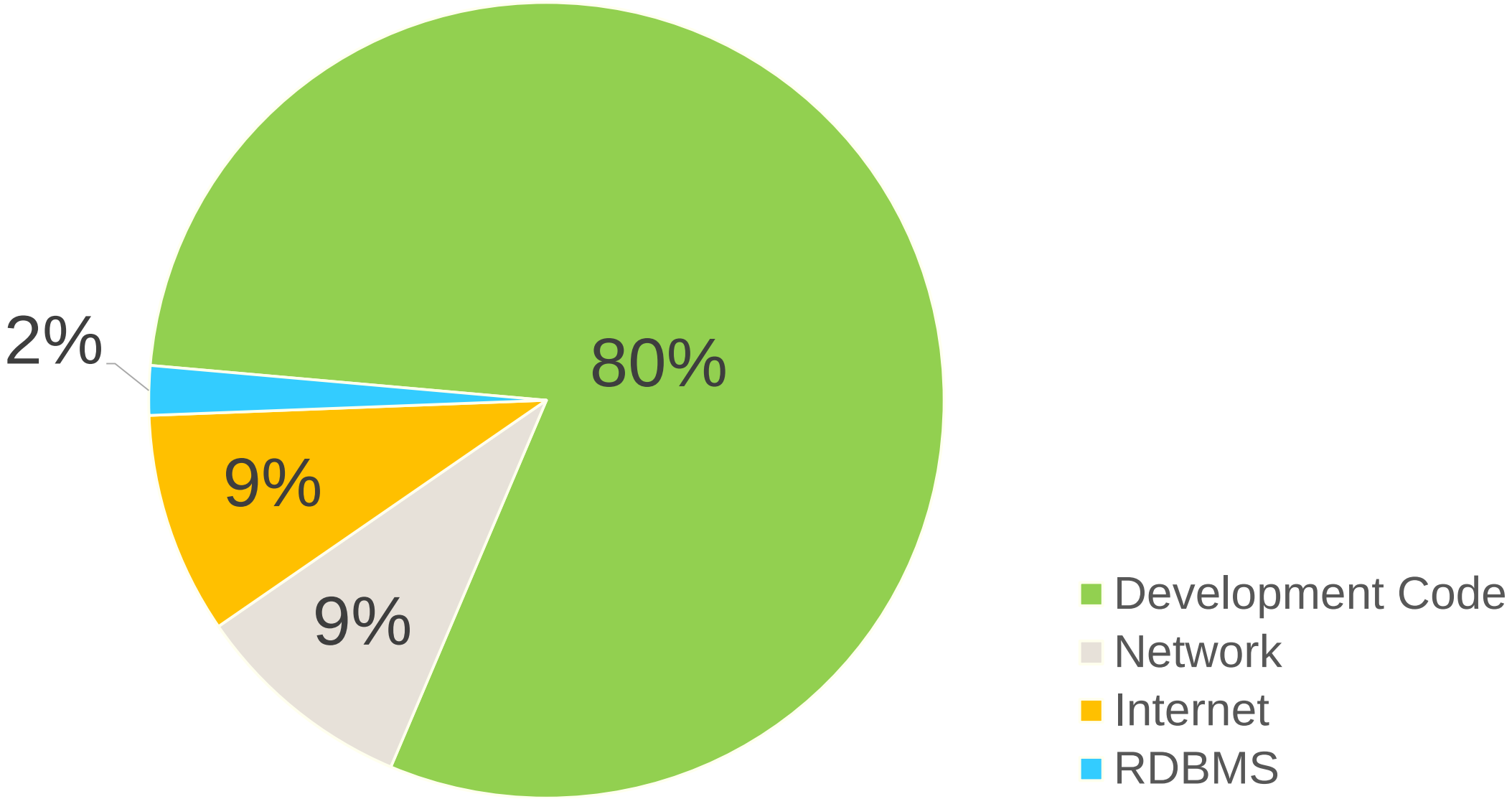




Performance Issues – Developer's View



Performance Issues – DBA's View



Skip explanation and quote

The Reality

- Application performance issues are **rarely** due to the RDBMS
  - Now: the Autonomous Database tunes itself!
- PL/SQL is already very mature and efficient – 30+ years old
- Reasons developers give for slow performance:
  - User misuse
  - "The network"
  - Windows upgrade
  - "Herman" (developer, now retired)
  - "Phases of the moon," "Mercury in retrograde"

Something to Consider

Sometimes it pays
to stay in bed on Monday,
rather than spending the
rest of the week
debugging Monday's code.

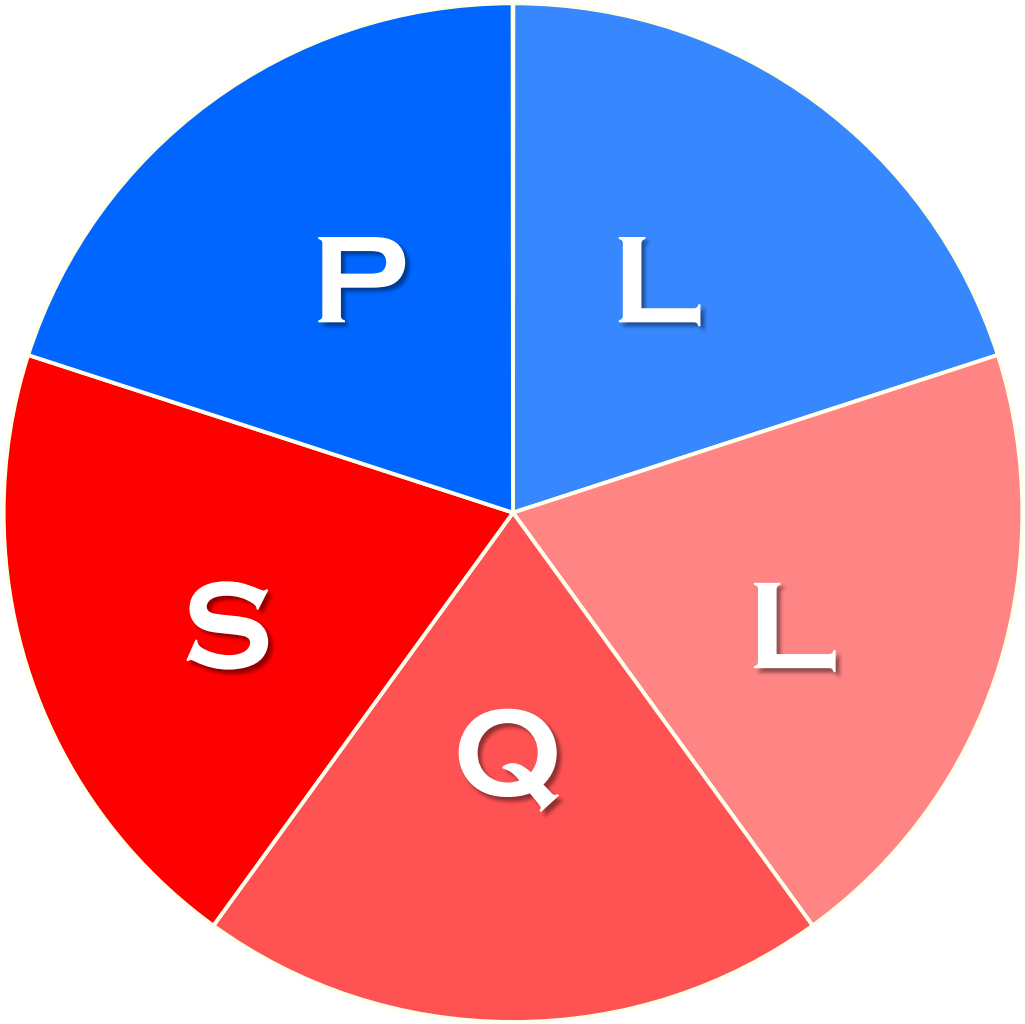
—Dan Salomon *(1940-)*

Top Tips

- Tune the SQL
- Use packages
- Tune the loops
- Cache the code/results
- Use datatypes correctly
- Plus

Recording will be on www.nyoug.org

60% of PL/SQL is SQL



Use Indexes

- Be sure indexes are being used
  - Full table scans are **not** bad for small tables
  - Examine costs as well as timings
  - Consider index-only tables
- Turn indexes off for bulk load
  - Drop and recreate
- 19*c*+ has automatic indexing

Remember

- Index will not be used if a function is applied to that column in the query
  - E.g., `WHERE UPPER(department_name) = 'SALES'`
  - Index on **department_name** will not be used
  - Instead use a function-based index
    - `CREATE INDEX dept_name_ix ON departments (UPPER(dept_name));`
    - Function-based index will be used

Watch the Optimizer

- Don’t trust the optimizer
  - Run Explain Plan to ensure efficient plans
  - Analyze tables first
- Trust the optimizer
  - Hints should be a last resort
  - **Reason**: if data volume or other factors change, the optimizer should be allowed to change the plan

INSERT, UPDATE, and DELETE use query syntax and must be tuned, too.

Tune Your Queries

- Don’t forget to tune views as well as raw SELECTs
- Use a tool to get hint suggestions
  - SQL Developer, TOAD, SQL Navigator, SQL Analyze (OEM)
  - Some can run multiple versions with different hints
- Break up large UNION or other complex statements
  - Put results together yourself with PL/SQL or a global temporary table

Use Materialized Views

- FKA, “snapshots”
  - Data stored for a SELECT statement
- Need to decide refresh frequency
  - Whenever
  - “Fast Refresh” on commits
  - Consider “enable query rewrite” clause
- For multi-table queries or complex WHERE clauses
- Use for high frequency queries
- Much faster data access for complex queries

PL/SQL Within SQL

Pre-19.7

- Avoid PL/SQL function calls within SQL
  - E.g., **`SELECT my_function(x,y) FROM table`**
  - Better
    - **`SELECT x, y INTO var FROM table`**
    - Then loop through var and apply **`my_function()`**
- Watch this: “Calling PL/SQL Functions from SQL”
  - https://www.youtube.com/watch?v=YuotfdaxOoM
  - Scalar subquery caching, DETERMINISTIC keyword, function result cache, PRAGMA UDF, virtual column (11g+)
  - Query PL/SQL tables with TABLE(CAST())

PL/SQL Within SQL – SQL Macros

19.7, 21.3+

- Embed a PL/SQL function inside SQL **without** the context switching overhead
- SCALAR or TABLE return types
  - **`CREATE FUNCTION f_calc`**
    `RETURN VARCHAR2 SQL_MACRO(SCALAR)...`
- Lots of examples at Tim Hall's legendary Oracle Base site:
  - https://oracle-base.com/articles/21c/sql-macros-21c
- Also check out the 21c docs

Query PL/SQL Tables

- Can help performance of complex queries

Example

1. Create an object and a type

```
CREATE TYPE dept_rec_typ AS OBJECT (
  department_id   NUMBER(4),
  department_name VARCHAR2(30),
  location_id     NUMBER(4)
);
--
CREATE TYPE dept_tab_typ
   AS TABLE OF dept_rec_typ;
```

Query PL/SQL Tables

2. Create package (body shown here)

```
CREATE OR REPLACE PACKAGE BODY dept_pkg
AS
   FUNCTION secure_dept_list (
      p_max_dept_id    PLS_INTEGER)
      RETURN dept_tab_typ
   IS
      v_dept_tab  dept_tab_typ;
   BEGIN
       -- Some complex logic to determine secure departments
       -- This example assigns hard coded values
       -- using the table and object constructors.
       v_dept_tab := dept_tab_typ(
           dept_rec_typ('10', 'Administration', '1700'),
           dept_rec_typ('20', 'Marketing', '1100'));
       --
      RETURN v_dept_tab;
   END;
END dept_pkg;
```

Query PL/SQL Tables

3. Use the function as a data source

```
SELECT department_id, department_name
FROM   TABLE ( CAST (
       dept_pkg.secure_dept_list(30)
       AS dept_tab_typ)
       );

DEPARTMENT_ID DEPARTMENT_NAME
------------- ------------------------------
           10 Administration
           20 Marketing

2 rows selected.
```

- **TABLE(CAST())** is the data set

8i+

Other SQL Tuning Tips

- Keep statistics up to date (ANALYZE)
- INSERT or UPDATE on large number of records may need to process sets
  - Again, use PL/SQL instead of SQL
  - Commit frequently (500-1000 rows)
  - Better: use BULK COLLECT (more later)
- Trigger actions must be tuned, too
  - These may execute for each row or statement in an INSERT, UPDATE, or DELETE

Even More SQL Tuning Tips

- Update using ROWID
  - –Can change between statements
  - –Use SELECT … FOR UPDATE
- Use analytic functions
- Let SQL not PL/SQL filter rows

Top Tips

Tune the SQL

Use packages

Tune the loops

Cache the code/results

Use datatypes correctly

Plus

Why Use Packages?

- Loaded into memory upon first use
- Hide implementation details
  - Grants to package offer access only to the spec, not the body
- Source of “global” session variables
  - Package specification variable values persist in the session
- Organize program units into logical groups
- Grants are on the package level, not function level
- Implement a Table API

Strong Use Case: Table API

- A PL/SQL package per table or business entity
  - All data modification ("DML") is accomplished through procedures
    - Single-row: INS(), UPD(), DEL(), LCK()
    - Multi-row: INS_ALL(), UPD_ALL(), DEL_ALL(), LCK_ALL()
    - Single-row samples here: https://bit.ly/tapi_code
  - Procedures call business rules validation code
- Call procedures from application code OR
- Call procedures from view trigger code

https://bit.ly/tapi_code

Table API Benefits

- Places the logic on the database side
  - Saves network messaging
  - Easier maintenance for logic changes
- Simplifies the user interface code
  - Business rules logic is in the database not the UI
- Promotes good security practices
  - Access to package only, not tables
  - Logic is hidden

JOBS_PKG Spec Excerpt

```
CREATE OR REPLACE PACKAGE jobs_tapi
AUTHID DEFINER
IS
   -- insert
   PROCEDURE ins (
      p_job_id        IN jobs.job_id%TYPE,
      p_min_salary    IN jobs.min_salary%TYPE DEFAULT NULL,
      p_job_title     IN jobs.job_title%TYPE,
      p_max_salary    IN jobs.max_salary%TYPE DEFAULT NULL
   );
   -- update
   PROCEDURE upd (
      p_job_id        IN jobs.job_id%TYPE,
      p_min_salary    IN jobs.min_salary%TYPE DEFAULT NULL,
      p_job_title     IN jobs.job_title%TYPE,
      p_max_salary    IN jobs.max_salary%TYPE DEFAULT NULL
   );
...
END jobs_tapi;
```

JOBS_PKG Body Excerpt

```
CREATE OR REPLACE PACKAGE BODY jobs_tapi
IS
   -- insert
   PROCEDURE ins (
      p_job_id        IN jobs.job_id%TYPE,
      p_min_salary    IN jobs.min_salary%TYPE DEFAULT NULL,
      p_job_title     IN jobs.job_title%TYPE,
      p_max_salary    IN jobs.max_salary%TYPE DEFAULT NULL
   );
   IS
      -- TODO: implement business rules check here
      INSERT INTO jobs (
            job_id, min_salary,
            job_title, max_salary
         ) VALUES (
            p_job_id, p_min_salary,
            p_job_title, p_max_salary
         );
   END ins;
...
END jobs_tapi;
```

Top Tips

Tune the SQL

Use packages

Tune the loops

Cache the code/results

Use datatypes correctly

Plus

Always Question Nested Loops

- Normal PL/SQL is highly optimized
- Actions inside loops (e.g., queries) can degrade performance
- Nested loops may not be necessary

```
FOR v_dept_rec IN (
    SELECT department_id
    FROM   departments
    WHERE  location_id = p_location_id)
LOOP
    FOR v_emp_rec IN (
        SELECT first_name, last_name
        FROM   employees
        WHERE  department_id =
               v_dept_rec.department_id)
    LOOP
        -- Some processing
    END LOOP;
END LOOP;
```

Better: Combine Queries, A Single Loop

```
FOR v_emp_rec IN (
    SELECT first_name, last_name
    FROM   departments dept, employees emp
    WHERE  dept.department_id = emp.department_id
    AND    dept.location_id = p_location_id)
LOOP
    -- Some processing
END LOOP;
```

- Fewer context switches between PL/SQL and SQL
- Fewer cursor openings and closings

BULK COLLECT

- Array processing
- Another way to avoid loops

```
FUNCTION load_summary(
   p_company_no  co_trans.company_no%TYPE)
   RETURN VARCHAR2
IS
   v_return_message      VARCHAR2(2000);
   --
   CURSOR c_trans(
      p_comp co_trans.company_no%TYPE)
   IS
      SELECT trans_id, company_no,
             trans_date, trans_amount
      FROM   co_transaction
      WHERE  company_no = p_comp;
   --
   TYPE trans_tab_typ IS TABLE OF c_trans%ROWTYPE;
   t_trans  trans_tab_typ;
BEGIN
   OPEN c_trans(p_company_no);
   LOOP
```

BULK COLLECT

- Query into an array
- INSERT using the array

```
      -- Get records
      FETCH c_trans
      BULK COLLECT INTO t_trans LIMIT 500;
      -- Insert records, ID column only
      FORALL i IN 1 .. t_trans.COUNT
          INSERT INTO transaction_summary
          VALUES t_trans(i);
      --
      EXIT when c_trans%NOTFOUND;
   END LOOP;
   --
   CLOSE c_trans;
   COMMIT;
   --
   RETURN v_return_message;
EXCEPTION
   WHEN OTHERS THEN
      IF c_trans%ISOPEN THEN CLOSE c_trans; END IF;
      RETURN 'Error in LOAD _SUMMARY:' || SQLERRM;
END;
```

Top Tips

Tune the SQL

Use packages

Tune the loops

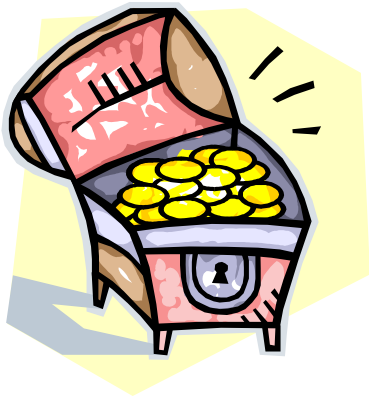
Cache the code/results

Use datatypes correctly

Plus

Package Variable Magic

- Load PL/SQL tables into associative arrays* and query the array – saves db queries
  - Row in table can be accessed by textual index
- Load data into PL/SQL package table variable
  - Startup section of package body. Check if the table is already loaded.
  - Query that data instead of the table
- Or use an application context
  - Linked to a package for a session



* http://www.dba-oracle.com/plsql/t_plsql_object_types.htm

Package

- Load the array in the package body initialization section

Demo

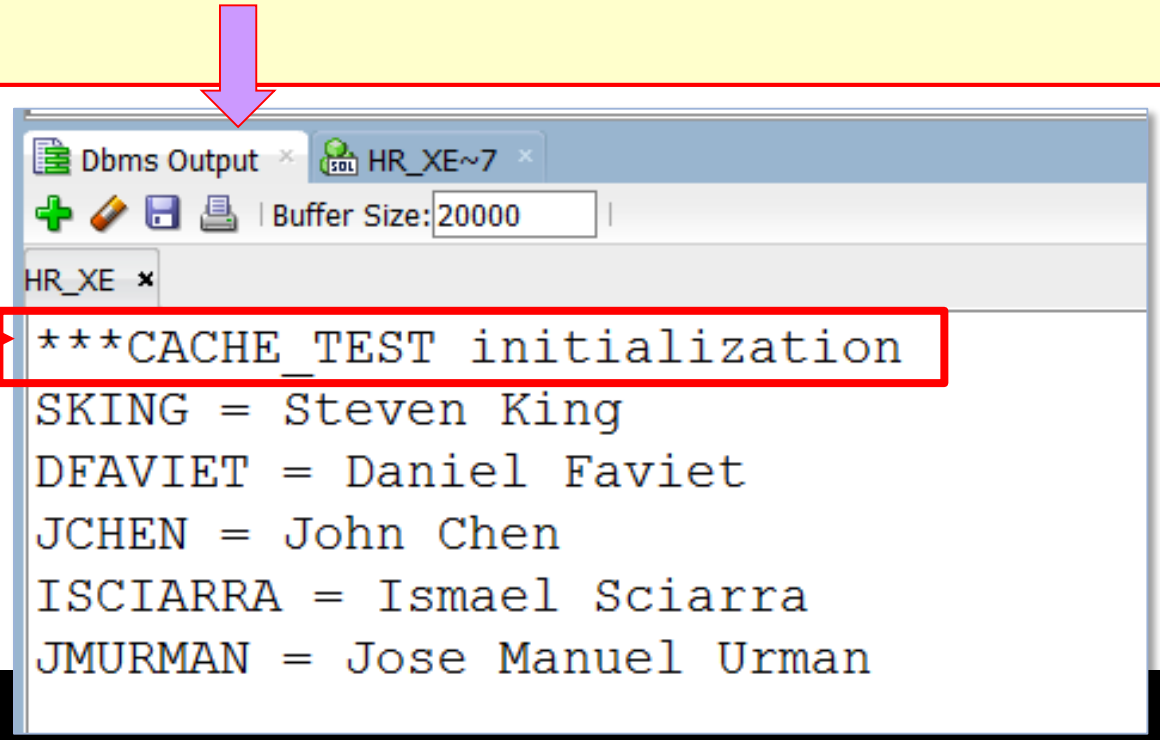
```
CREATE OR REPLACE PACKAGE cache_test
AS
   TYPE employee_typ is TABLE OF VARCHAR2(50)
      INDEX BY VARCHAR2(25);
   emp employee_typ;
END cache_test;
```

```
CREATE OR REPLACE PACKAGE BODY cache_test
AS
  x NUMBER;
-- Package initialization section
BEGIN
   DBMS_OUTPUT.PUT_LINE('***CACHE_TEST initialization');
   FOR c_emp IN (
         SELECT first_name||' '||last_name ename, email
         FROM employees)
   LOOP
      emp(c_emp.email) := c_emp.ename;
   END LOOP;
END cache_test;
```

Test Run 1

```
BEGIN
   DBMS_OUTPUT.PUT_LINE('SKING = '||cache_test.emp('SKING'));
   DBMS_OUTPUT.PUT_LINE('DFAVIET = '||cache_test.emp('DFAVIET'));
   DBMS_OUTPUT.PUT_LINE('JCHEN = '||cache_test.emp('JCHEN'));
   DBMS_OUTPUT.PUT_LINE('ISCIARRA = '||cache_test.emp('ISCIARRA'));
   DBMS_OUTPUT.PUT_LINE('JMURMAN = '||cache_test.emp('JMURMAN'));
END;
```

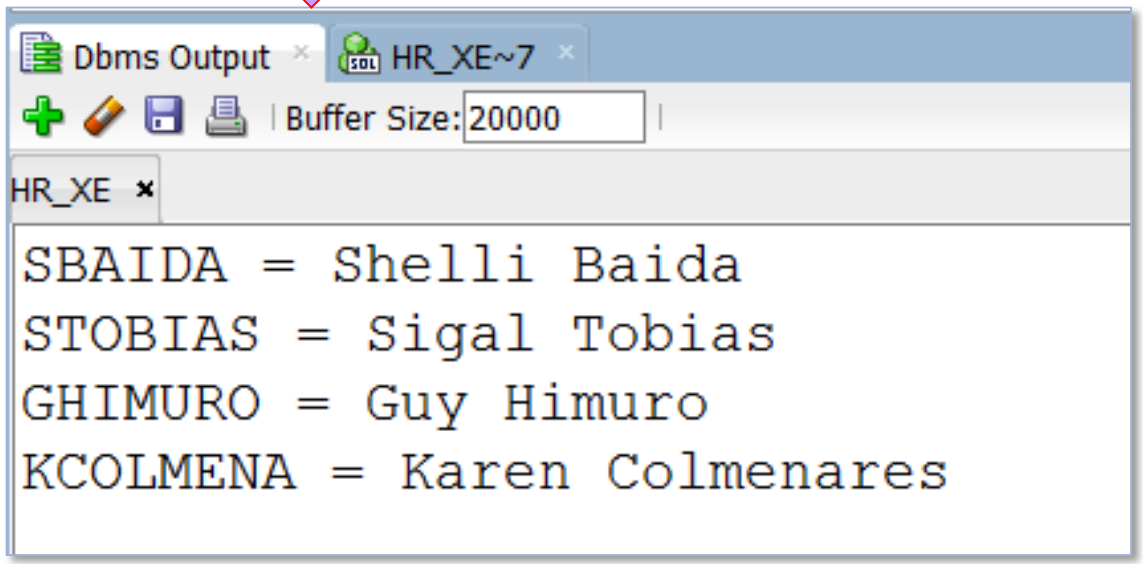
Only once per session



Test Run 2

```
BEGIN
   dbms_output.put_line('SBAIDA = '||cache_test.emp('SBAIDA'));
   dbms_output.put_line('STOBIAS = '||cache_test.emp('STOBIAS'));
   dbms_output.put_line('GHIMURO = '||cache_test.emp('GHIMURO'));
   dbms_output.put_line('KCOLMENA = '||cache_test.emp('KCOLMENA'));
END;
```

- Subsequent calls use the cached array
- No re-initialization

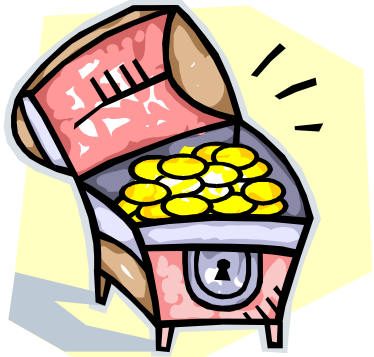


More Caching

- PL/SQL Function Result Cache * 11g+
  - If a function is called again with the same parameters, it doesn't run the logic, it just returns the previous value.
  - Syntax:

```
CREATE FUNCTION my_function()
RETURN VARCHAR2 RESULT_CACHE AS
```

- Pin packages in the shared pool
  - **DBMS_SHARED_POOL.KEEP()**



* https://blogs.oracle.com/oraclemagazine/on-the-plsql-function-result-cache

Top Tips

Tune the SQL

Use packages

Tune the loops

Cache the code/results

Use datatypes correctly

Plus

Need an Integer?

- Use PLS_INTEGER or SIMPLE_INTEGER
- 100M arithmetic operations *

Datatype	Execution Time	
NUMBER	536 hsec	
INTEGER	973 hsec	only for NOT NULL
PLS_INTEGER	162 hsec	
BINARY_INTEGER	150 hsec	
SIMPLE_INTEGER	138 hsec	11g+ only for NOT NULL

Considered "identical" for 10g+

Demo

- Need a floating point?
  - Use BINARY_FLOAT and BINARY_DOUBLE

* https://oracle-base.com/articles/misc/performance-of-numeric-data-types-in-plsql

https://blogs.oracle.com/oraclemagazine/working-with-numbers-in-plsql

Avoid Implicit Datatype Conversions

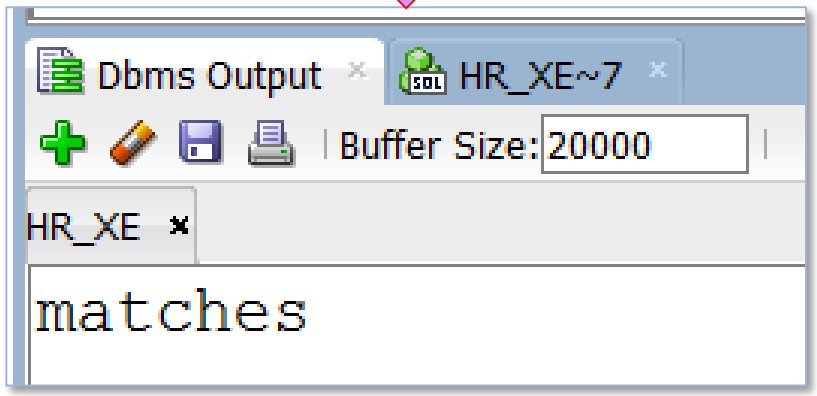
```
DECLARE
   i PLS_INTEGER;
BEGIN
   i := '1';
END;
```

```
DECLARE
   d DATE;
BEGIN
   d := '02/09/2021';
   DBMS_OUTPUT.PUT_LINE(d);
END;
```

- Not an error, just not as efficient
- Ensure same datatypes in comparisons
  - Especially in embedded SQL

Dissimilar Length String is Padded

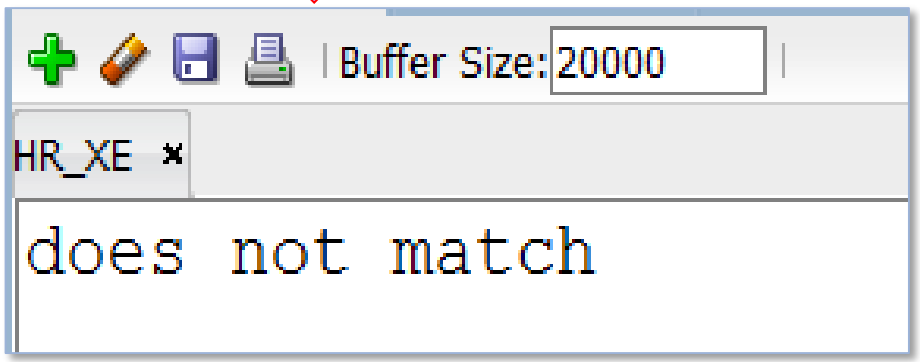
```
BEGIN
   IF '1234' = '1234  '
   THEN DBMS_OUTPUT.PUT_LINE('matches');
   ELSE DBMS_OUTPUT.PUT_LINE('does not match');
   END IF;
END;
```



But Yet...

```
DECLARE
   v_var1 VARCHAR2(6) := '1234';
   v_var2 VARCHAR2(6) := '1234  ';
BEGIN
   IF v_var1 = v_var2
   THEN DBMS_OUTPUT.PUT_LINE('matches');
   ELSE DBMS_OUTPUT.PUT_LINE('does not match');
   END IF;
END;
```

- Only explicit strings pad values




Plus

6. Profiling

- “Profiling” = analyzing the code
- Simple version: time each section of code
  - Write to a message table in an autonomous transaction
  - Find and tune the longest-running sections
  - DBMS_UTILITY.GET_TIME to obtain the timing

```
v_start_time := DBMS_UTILITY.GET_TIME;

-- run the procedure

v_stop_time := DBMS_UTILITY.GET_TIME;

v_elapsed_seconds := (v_stop_time –
   v_start_time)/100;

DBMS_OUTPUT.PUT_LINE('The procedure took '||
   v_elapsed_seconds ||' seconds');
```

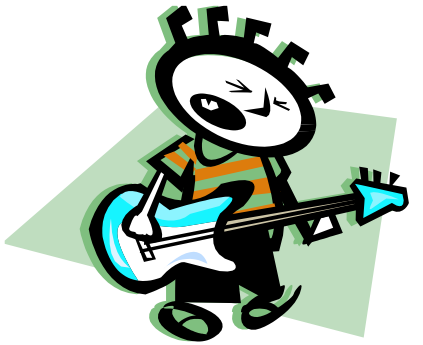
Better: Oracle Profiling Packages

- DBMS_HPROF
  - Hierarchical profiler (11g+)
  - Use it to determine problem program units
- DBMS_PROFILER (8i+ profiler)
- Simple version can also use DBMS_APPLICATION_INFO
  - Add lines to inject values into session
    - Include timing information
  - Periodically query V$SESSION (ACTION and CLIENT_INFO columns)

Bonus Tips

7. Use SELECT INTO
   - No longer less efficient than OPEN-FETCH-CLOSE
8. Avoid excessive context switches
   - Embedding PL/SQL inside SQL, or vice versa
   - Use BULK COLLECT and FORALL– mentioned before
9. Implicit vs explicit cursors – implicit faster
   - Explicit = OPEN-LOOP-FETCH-CLOSE
10. Slowdowns
    - Variable declarations inside loops vs outside
      - See Tim Hall's book excerpt *
    - Function call within loop vs. function logic inside loop

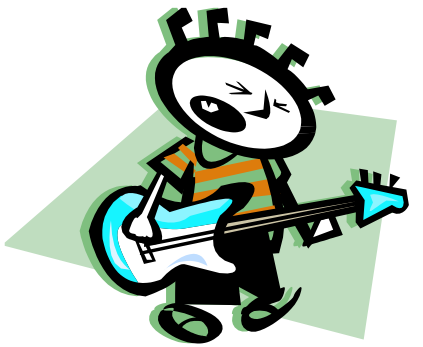
* http://www.dba-oracle.com/plsql/t_plsql_declarations.htm



11. Advanced: Subprogram Inlining

- If a function or procedure is called in a loop, overhead of calling it can add up
- Solution:
  - `PRAGMA INLINE (proc_name, 'YES');`
  - This embeds the code from proc_name in the loop
- Read the docs for when this can be applied
- Configure with database parameter PLSQL_OPTIMIZE_LEVEL
  - Value of 2 relies on PRAGMA INLINE
  - Value of 3 may inline automatically

11g+



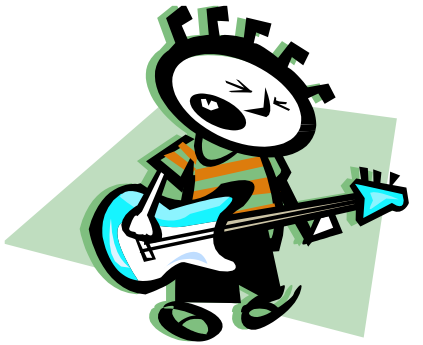
12. Conditional Compilation

10gR2+

```
$IF debug_pkg.do_debug
$THEN DBMS_OUTPUT.PUT_LINE('DEBUG point 4: i='||i);
$END IF;
```

- Then set the **do_debug** variable (CONSTANT)
- Recompile DEBUG_PKG which recompiles all code
- $ markers signal to the compiler to ignore the code when the **do_debug** variable is FALSE

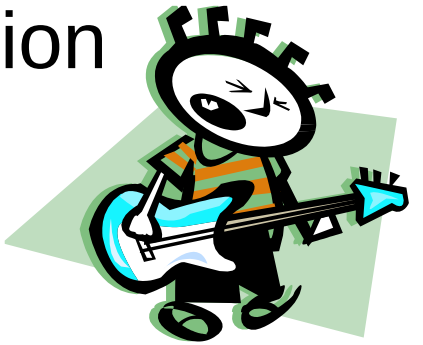
```
CREATE OR REPLACE PACKAGE debug_pkg
AS
   do_debug CONSTANT BOOLEAN DEFAULT TRUE;
END debug_pkg;
```



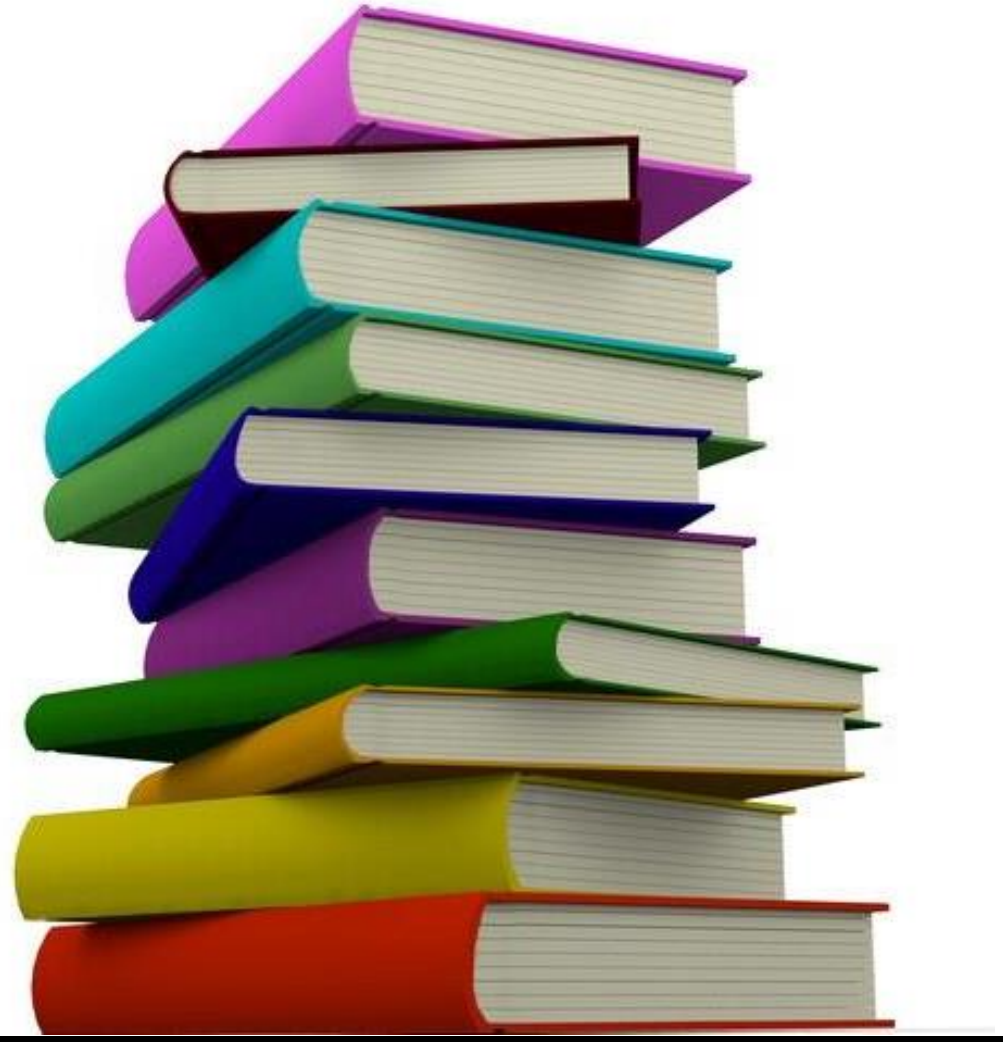
More Bonus Tips

13. Use bind variables in dynamic SQL
    - Not hard-coded strings
    - Allows statement parsing to be reused
14. Pay attention to the order of conditions
    - IF...THEN…ELSIF and also CASE…WHEN structures
    - ***Fail Fast*** in a Boolean AND:
      - **`(FALSE) AND (TRUE)`** ← Not tested

      Least expensive/quickest condition
15. Regex to check format vs. hand-crafted parsing function
16. Long-running process can use advanced queuing for async operation: DBMS_AQADM

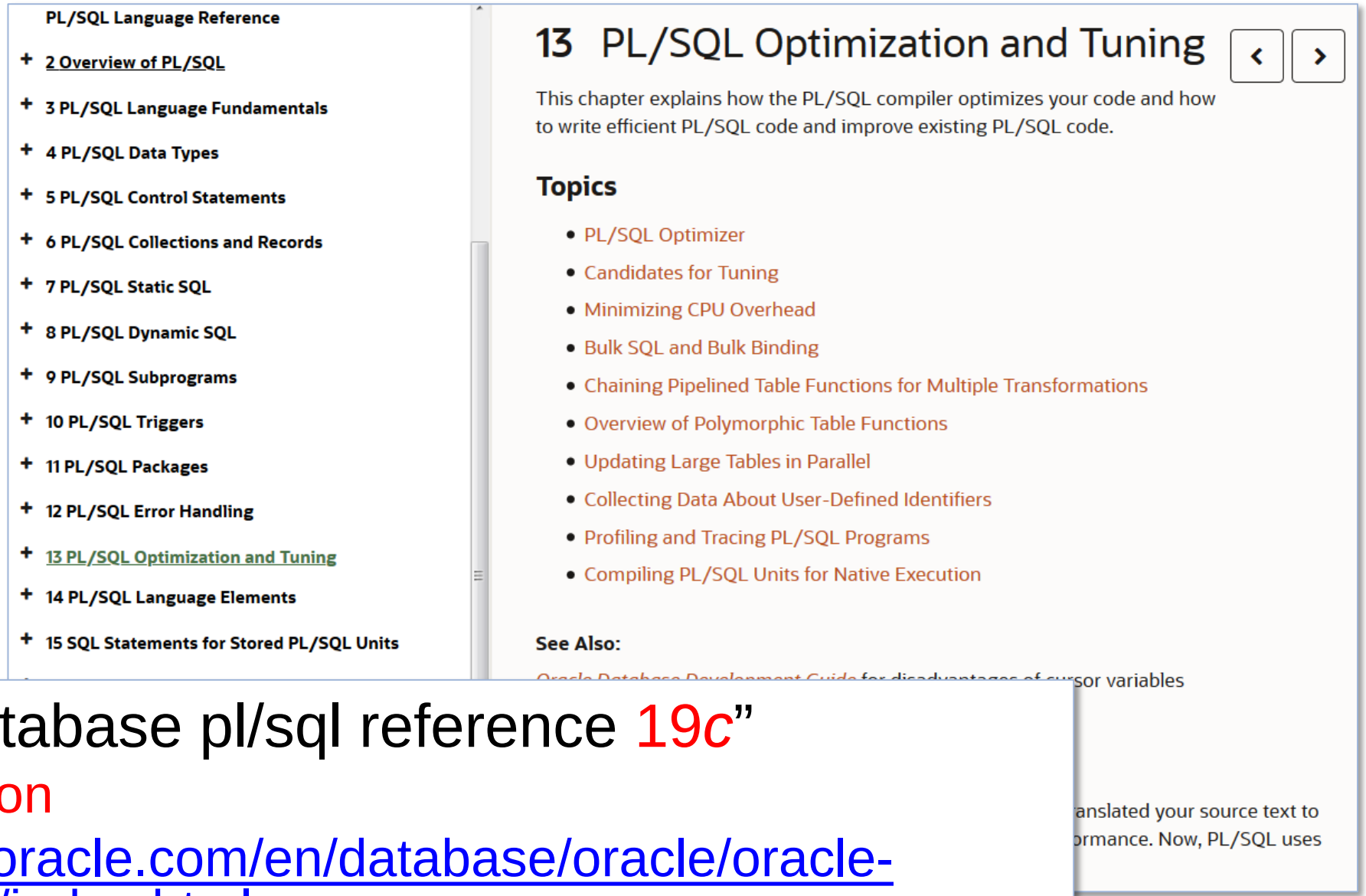


17. Follow Up Reading



Oracle Docs!

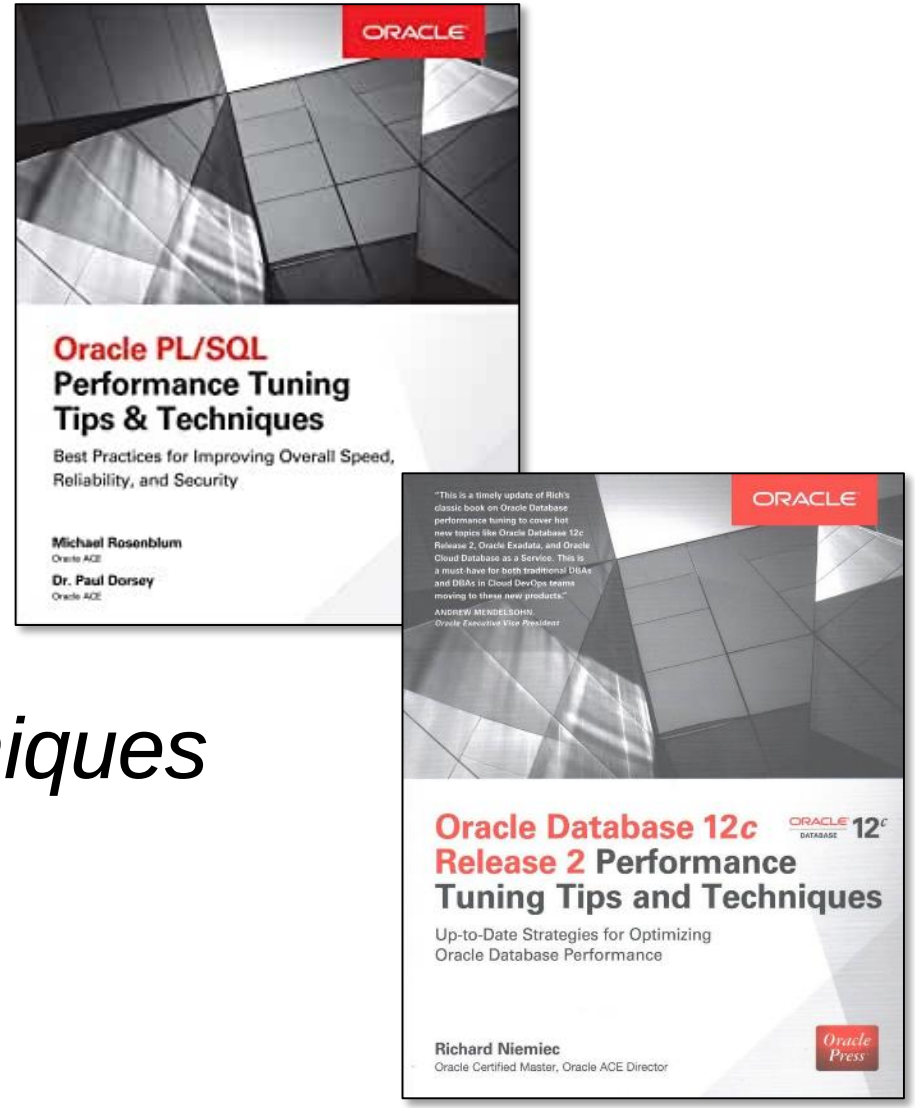
- Contains good tips on basics with links to specific features



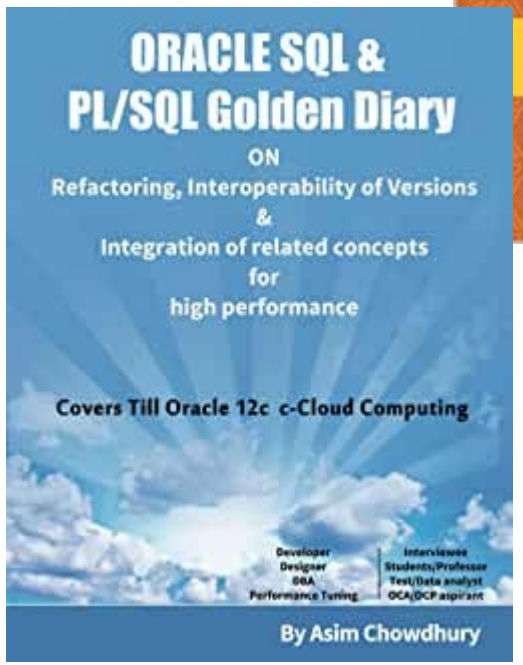
- Google “oracle database pl/sql reference 19*c*”
  - Or whatever version
  - E.g., https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/index.html

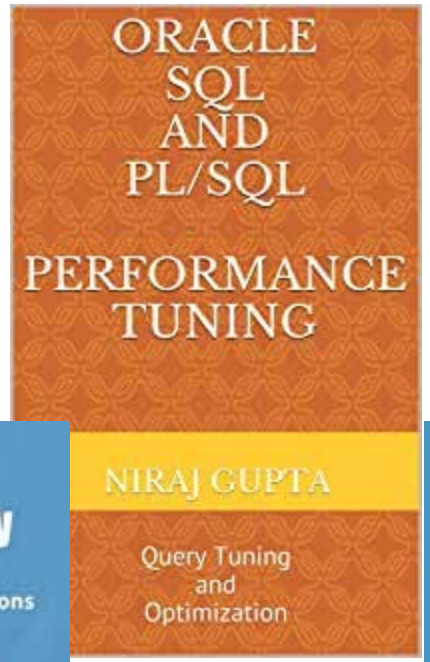
PL/SQL 201: Deeper Tuning

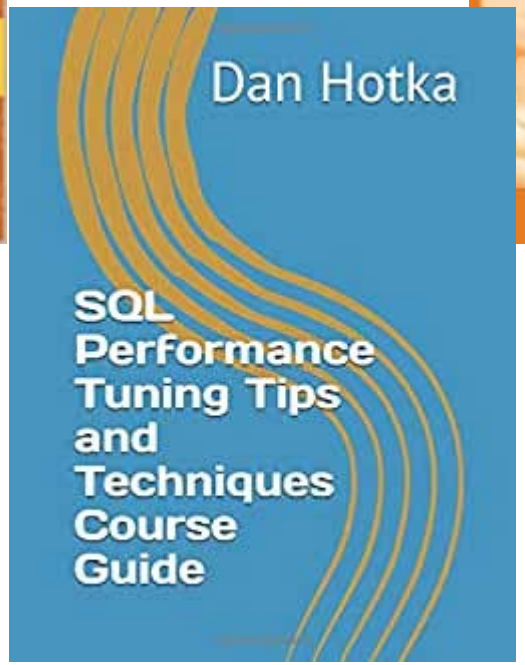
- *Oracle PL/SQL Performance Tuning Tips & Techniques*
  - Rosenblum and Dorsey
  - Oracle Press, 2014
  - Depth on hierarchical profiler and tuning details
- *Oracle Database 12c Release 2 Performance Tuning Tips & Techniques* (Oracle Press)
  - Rich Niemiec
  - Index and SQL tuning

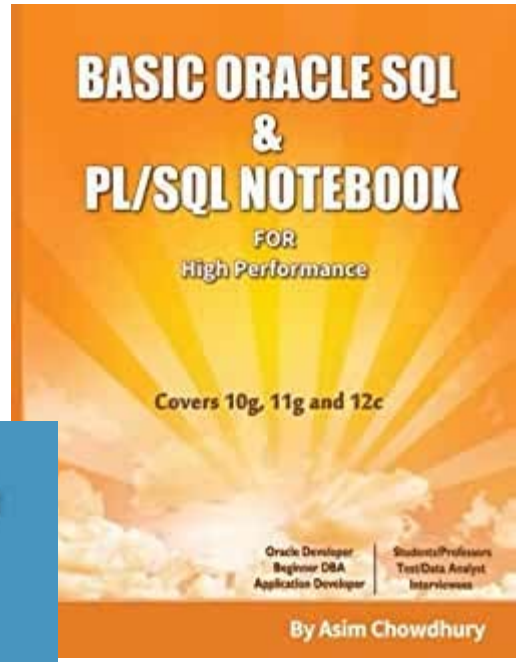


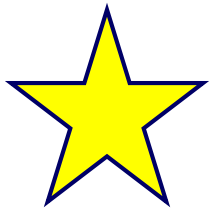
Other Books



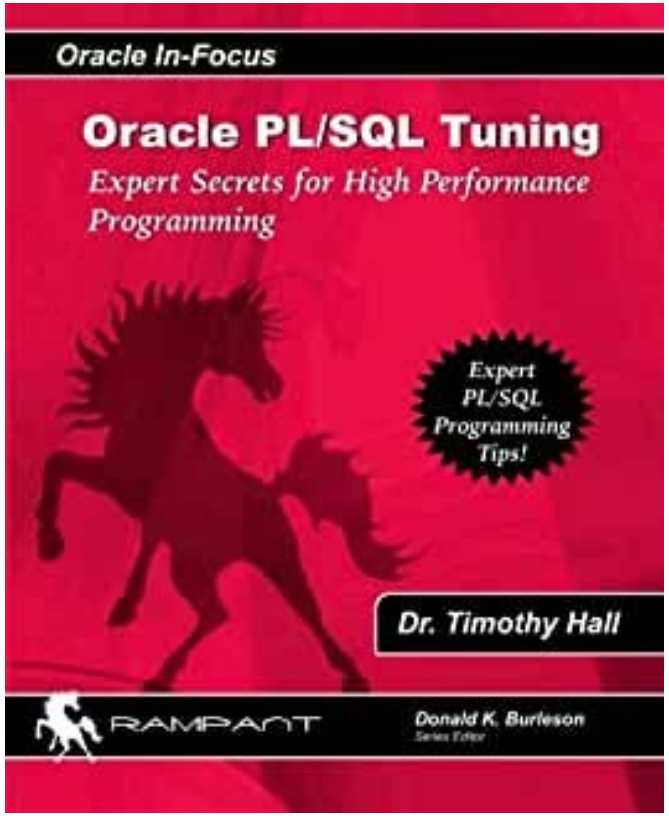






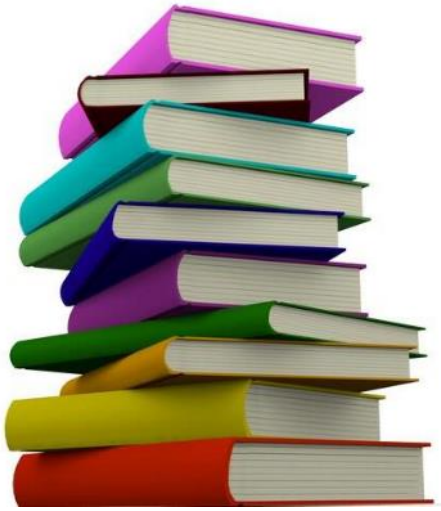


http://www.dba-oracle.com/plsql/



More Reading

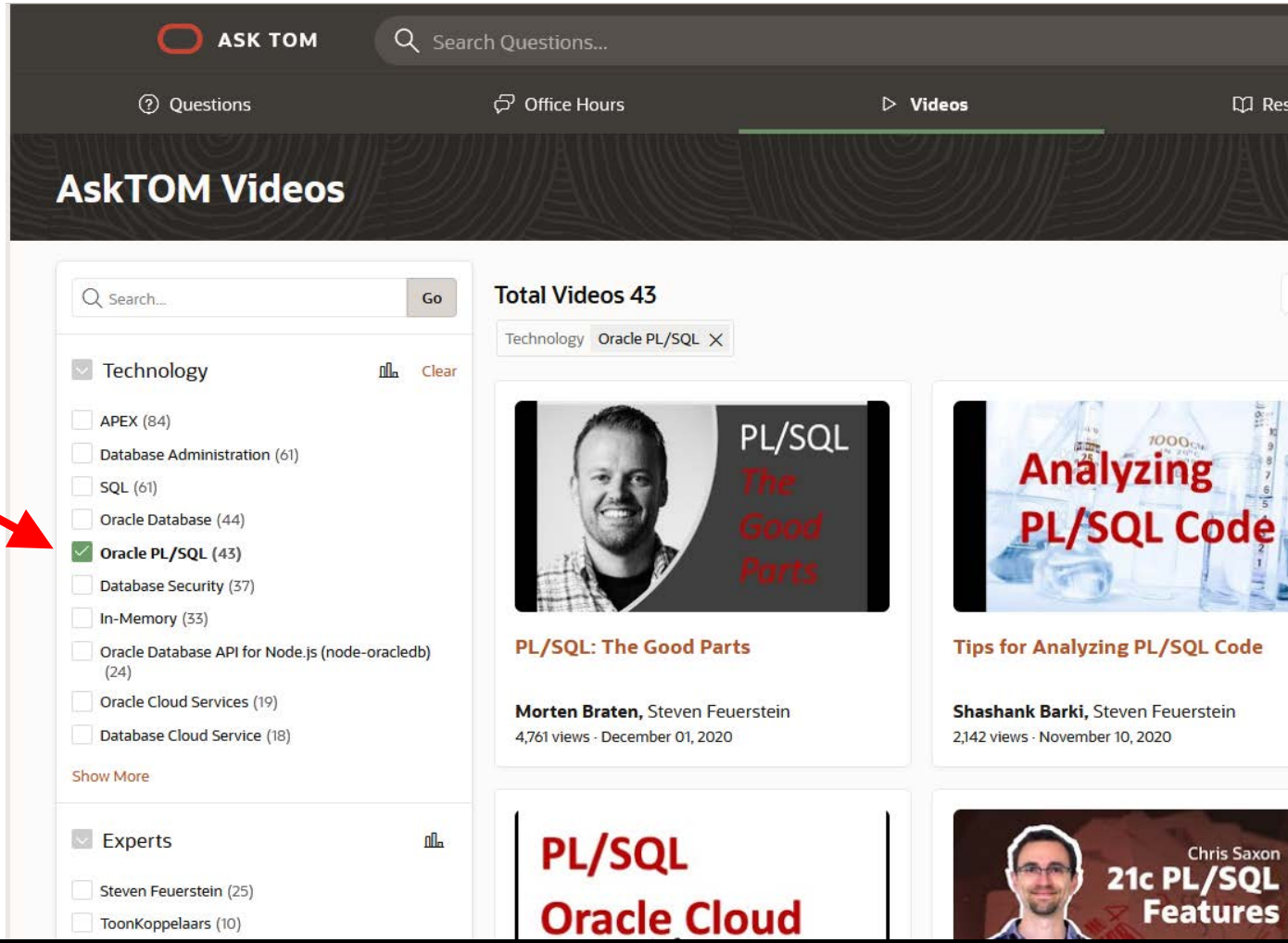
- For bulk data loading
  - *Ultra-High Performance SQL and PL/SQL in Batch Processing*, by Dr. Paul Dorsey
  - Google ‘nyoug "ultra-high performance sql”’ with double quotes
- *How to Tune PL/SQL Performance with Oracle SQL Profiler*
  - https://www.devart.com/dbforge/oracle/studio/performance-tuning.html



Ask Tom PL/SQL Office Hours

- https://asktom.oracle.com/officehours
- No more PL/SQL sessions
- **C**lick “View Recordings” on the right

  - Then filter by PL/SQL
  - E.g., “Modularization Controversies!” Aug. 2019 session discussed “functions vs. procedures” for 21 minutes!




QUIZ

1. When do you tune PL/SQL?

Answer: The earlier in the process the better.

2. How could this be better?

```
FOR item IN (
   SELECT DISTINCT(SQRT(department_id)) col_alias
   FROM   employees
   ORDER BY col_alias
 )
 LOOP
   DBMS_OUTPUT.PUT_LINE('SQRT of dept. ID = ' ||
      item.col_alias);
 END LOOP;
```

From "Oracle PL/SQL Language Reference"

Better

```
FOR item IN (
   SELECT SQRT(department_id) col_alias
   FROM (SELECT DISTINCT department_id
         FROM   employees)
   ORDER BY col_alias
 )
 LOOP
   IF item.col_alias IS NOT NULL
   THEN
     DBMS_OUTPUT.PUT_LINE('SQRT of dept. ID = ' ||
        item.col_alias);
   END IF;
 END LOOP;
```

Reason: SQRT() applied only to DISTINCT IDs, not to all IDs.

3. True or False?

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

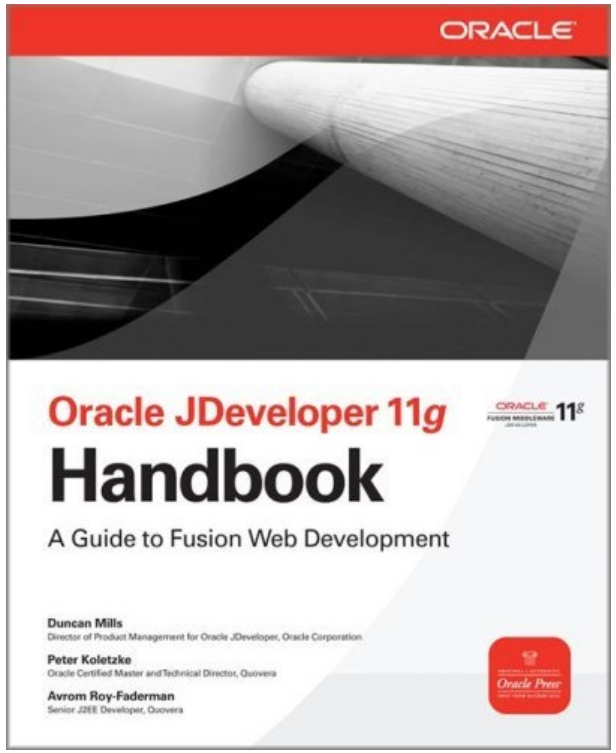
—*Brian Kernighan*

```
isSet = (i >= x ? (wasSet ? true :
    false) : (notSet ? false : true));
```

Summary

- Tune the SQL
- Use packages
- Tune the loops
- Cache the code/results
- Use datatypes correctly

+ Keep reading



- Thank you for attending
- Recording will be available at www.nyoug.org