Achieving PL/SQL Excellence

Top 200 Oracle PL/SQL Tips for Tuning

Steven Feuerstein

Me - www.StevenFeuerstein.com

PL/Solutions - www.plsolutions.com

RevealNet - www.revealnet.com

Starbelly.com – www.starbelly.com

One Hour, 200 Tips, No Problem!

Improve the performance of your application *1000 fold* for only $19.95 a month!*

***Plus shipping and handling and technical support. All performance degradation the responsibility of the user.**

Resources for PL/SQL Tuning

- Interested in "Oracle tuning"? The world is your oyster:
  - Oracle documentation
  - Numerous tuning books and web sites
  - Many, many tools
- But PL/SQL tuning? Slim pickings...
  - PL/SQL books and general Oracle tuning books offer some coverage, but it is minimal and piecemeal.
  - Code Complete by Steve McConnell (Microsoft Press)
    - » Many tuning tips are not language-specific. This book offers an excellent treatment of tuning philosophies and issues you have to address in any programming language.
- PL/SQL tuning is *tough*, compared to SQL tuning.
  - Optimize algorithms, write SQL in PL/SQL properly, tune PL/SQL execution in the SGA.

Putting Tuning in Context

OPTIMAL PATH TO OPTIMAL CODE

Write well-structured, readable code following established best practices.

- Performance is just one aspect of high-quality software and usually not the most important.
- You can't improve performance without quantitative analysis.
  - Where are the bottlenecks?
  - How much did my code's performance improve?
- The 80/20 Rule: most programs spend most of their time in a small portion of the code.
  - Ah, but which portion? This is hard to predict.

plvtmr.pkg
tmr81.ot
ovrhead.sql

Possible Tuning Topics for PL/SQL

- Analyze Performance of Your Application
- Optimize SQL inside Your PL/SQL
- Manage Code in the Database and SGA
- Optimize Algorithms
- Use Data Structures Efficiently

- All source code examples downloadable from the RevealNet PL/SQL Pipeline Archive…
  - Under Miscellaneous, PL/SQL Seminar Files
  - demo.zip

plvtmr.pkg
tmr81.ot

PL/SQL Tuning and Best Practices

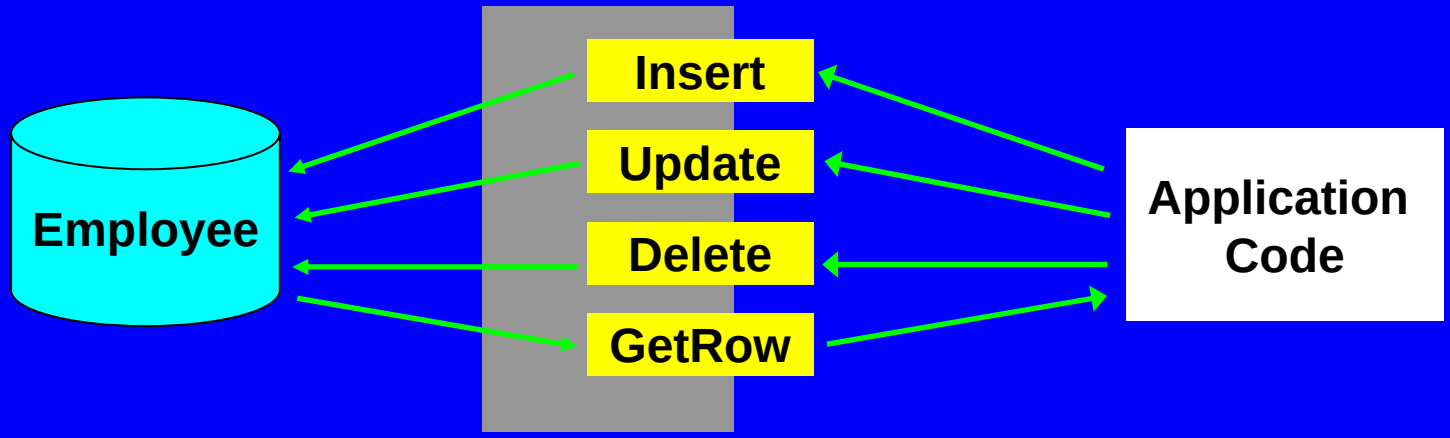
Writing SQL in PL/SQL

- What's the Big Deal?
- Some Rules to Follow
- Synchronize Code with Data Structures
- Avoid Repetition of SQL
- Optimize the PL/SQL we write in SQL

"SQL in PL/SQL" Rules to Follow

- Rule #1: Write your code so that it adapts automatically (with nothing more than a compile) to changes in the underlying data structures.
  - Anchor declarations with %TYPE and %ROWTYPE.
  - Fetch into records, not variables.
- Rule #2: Never repeat any of the SQL (inserts, updates, deletes, queries, DDL) in your code.
  - Build layers of code around your data structures (table encapsulation packages).

Build SQL Encapsulation Packages

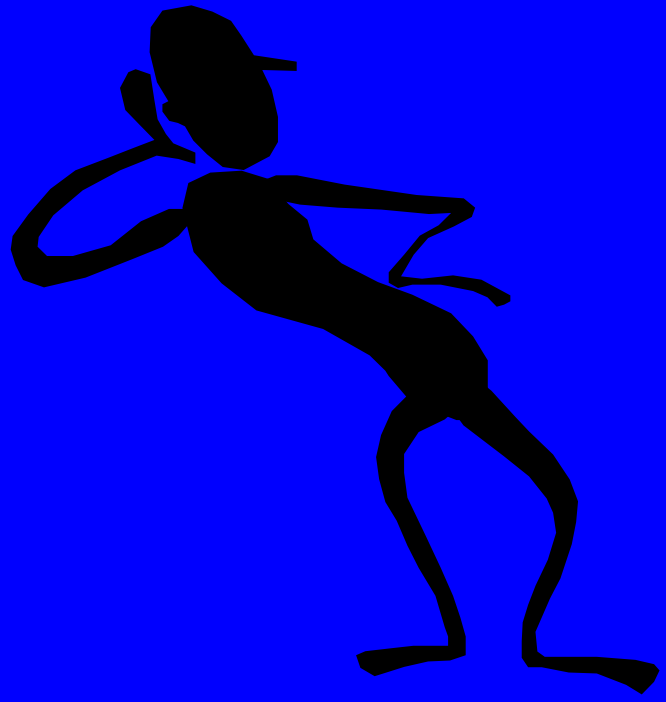


- Store all of your SQL inside packages: one per table or "business object".
  - All DML statements written by an expert, behind a procedural interface, with standardized exception handling.
  - Commonly-needed cursors and functions to return variety of data (by primary key, foreign key, etc.).
  - If the encapsulation package doesn't have what you need, add the new element, so that everyone can take advantage of it.
  - Could create separate packages for query-only and change-related functionality.

te_employee.*

Answer the Question Being Asked

- Are you a good listener? Listening to what other people is an excellent skill to have and develop -- and it applies to programming as well.
- All too often, we don't listen or read carefully enough to the requirement -- and we answer the wrong question.



If We Have At Least One…

```
CREATE OR REPLACE PROCEDURE drop_dept
   (deptno_in IN NUMBER, reassign_deptno_in IN NUMBER)
IS
   temp_emp_count  NUMBER;
BEGIN
   -- Do we have any employees in this department to transfer?
   SELECT COUNT(*)
     INTO temp_emp_count
     FROM emp WHERE deptno = deptno_in;

   -- Reassign any employees
   IF temp_emp_count >0
   THEN
      UPDATE emp
         SET deptno = reassign_deptno_in
       WHERE deptno = deptno_in;
   END IF;

   DELETE FROM dept WHERE deptno = deptno_in;
   COMMIT;
END drop_dept;
```

- ***How much*** **is wrong with this code?**

The Minimalist Approach

- At least one row?

```
BEGIN
   OPEN cur;
   FETCH cur INTO rec;
   IF cur%FOUND
   THEN
      ...
```

Use an explicit cursor, fetch once and then check the status.

- More than one row?

```
BEGIN
   OPEN cur;
   FETCH cur INTO rec;
   IF cur%FOUND
   THEN
      FETCH cur INTO rec;
      IF cur%FOUND
      THEN
         ...
```

Use an explicit cursor, fetch once and then fetch *again*. "Two times" is the charm.

atleastone.sql

Give W/One Hand, Take W/the Other

- Oracle has a habit of offering improvements in ways that can make it very difficult for us to take advantage of them.
- When are SQL statements the same and yet different?

```
UPDATE ceo_compensation
   SET stock_options = 1000000,
       salary = salary * 2.0
 WHERE layoffs > 10000;
```

```
begin update ceo_compensation
   set stock_options = 1000000,
       salary = salary * 2.0
 where layoffs > 10000; end;
```

```
UPDATE ceo_compensation
   SET stock_options = 1000000,
       salary = salary * 2
 WHERE layoffs > 10000;
```

```
BEGIN
UPDATE CEO_COMPENSATION
   SET STOCK_OPTIONS = 1000000,
       SALARY = SALARY * 2.0
 WHERE LAYOFFS > 10000;
END;
```

```
begin UPDATE ceo_compensation
SET stock_options = 1000000,
salary = salary * 2 WHERE
layoffs > 10000; end;
```

All these statements are executed at 10 AM.
How many times does Oracle parse?

SQL Cursors in the SGA

- As of Version 7, all parsed cursors (SQL statements as well as PL/SQL blocks) are cached in the SGA.
- Every time you request a parse (again: SQL and PL/SQL blocks), Oracle hashes the string.
  - If it finds an *exact, physical* match already in the SGA, then it uses that pre-parsed cursor.
  - This can lead to significant performance improvements.
  - But also a problem: we live, work and breathe at the *logical* level. Now we have to be aware of the physical form of our code!
- Some things to keep in mind:
  - White space counts -- unless the SQL statement is executed inside a PL/SQL block.
  - In this case, the PL/SQL engine does some pre-formatting -- all upper case, no extra white space.

Analyzing SGA-Cached Cursors

- The best way to understand the requirements and activity of the PL/SQL code in the SGA is to look at the SGA.
- Oracle offers a variety of data structures to get this information:
  - V$ROWCACHE: check for data dictionary cache hits/misses
  - V$LIBRARYCACHE: check for object access hits/misses
  - V$SQLAREA: statistics on shared SQL area, one row per SQL string (cursor or PL/SQL block)
  - V$DB_OBJECT_CACHE: displays info on database objects that are cached in the library cache.

```
SQL> exec insga.show_similar
*** Possible Statement Redundancy:
     begin fix_me (1); end;
     begin fix_me(1); end;
*** Possible Statement Redundancy:
     select * from EMP
     select * from emp
```

grantv$.sql
insga.pkg
similar.sql

Best Way to Code Single Row Query?

- Let's end the debate over implicit vs. explicit cursors. There are pluses and minuses for each approach...

Reuse: There is no way to reuse an implicit cursor, except by calling the program in which the cursor is executed. A cursor declared in a package specification can be used in multiple programs.

Performance: Implicits in 7.3 and above can be faster than explicits. Implicits are more likely, however, to be coded repetitively. Explicit cursors improve chance of using pre-parsed SQL in the SGA

Programmatic control: With explicits, you're not forced into the exception section when various data conditions arise.

Developer productivity: Why lose time trying to decide which way to code each single-row queries? Give yourself one less thing to think about.

- The real question is how can we make sure that our queries are always encapsulated?

explimpl.sql
explimpl.pkg

Don't Dither – Encapsulate!

- Whichever way you go, put the logic in a function.

```
FUNCTION i_empname (
   employee_id_in IN

employee.employee_id%TYPE)
   RETURN fullname_t
IS
   retval fullname_t;
BEGIN
   SELECT last_name
     INTO retval
     FROM employee
    WHERE employee_id =
             employee_id_in;
   RETURN retval;
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN RETURN NULL;
   WHEN TOO_MANY_ROWS
   THEN log_error; RAISE;
END;
```

```
FUNCTION e_empname (
   employee_id_in IN
      employee.employee_id%TYPE)
   RETURN fullname_t
IS
   rec allcols_cur%ROWTYPE;
BEGIN
   OPEN allcols_cur (employee_id_in);
   FETCH allcols_cur INTO rec;
   CLOSE allcols_cur;

   IF rec.employee_id IS NOT NULL
   THEN
      RETURN rec.last_name)
   ELSE
      RETURN NULL;
   END IF;
END;
```

explimpl.pkg

Ah, the Wonders of Dynamic SQL!

```
CREATE OR REPLACE PROCEDURE updnumval (
   col_in IN VARCHAR2,
   ename_in IN emp.ename%TYPE,
   val_in IN NUMBER)
IS
   cur PLS_INTEGER := DBMS_SQL.OPEN_CURSOR;
   fdbk PLS_INTEGER;
   dmlstr PLV.dbmaxvc2 :=
   'UPDATE emp SET ' || col_in || ' = ' || val_in ||
      ' WHERE ename LIKE UPPER (''' || ename_in || ''')';
BEGIN
   DBMS_SQL.PARSE (cur, dmlstr, DBMS_SQL.NATIVE);

   fdbk := DBMS_SQL.EXECUTE (cur);

   DBMS_OUTPUT.PUT_LINE (
       'Rows updated: ' || TO_CHAR (fdbk));

   DBMS_SQL.CLOSE_CURSOR (cur);
END;
/
```

Just throw a bunch of strings together and off you go!
Well, maybe not...

DynSQL: Bind Whenever Possible

- You *can* concatenate rather than bind, but binding is almost always preferable. Two key reasons:
  - Simpler code to build and maintain
  - Improved application performance
- Simpler code to build and maintain
  - Concatenation results in much more complicated and error-prone code unless you are doing a very simple operation.
- Improved application performance
  - Concatenates requires an additional call to DBMS_SQL.PARSE and also increases the likelihood that the SQL statement will be physically different, requiring an actual re-parsing and unnecessary SGA utilization.
- Note: you cannot bind schema elements, like table names.

effdsql1.sql
effdsql.tst
updnval2.sp
updnval3.sp

PL/SQL Tuning & Best Paractices

Optimize Algorithms

- Avoid Unnecessary Code Execution
- Answer the Question Being Asked
- Do Lots of Stuff At the Same Time
- Avoid the Heavy Lifting

Do No Unnecessary Thing - 1

- What's wrong with this code?

```
DECLARE
   CURSOR emp_cur
   IS
      SELECT last_name, TO_CHAR (SYSDATE, 'MM/DD/YYYY') today
        FROM employee;
BEGIN
   FOR rec IN emp_cur
   LOOP
      IF LENGTH (rec.last_name) > 20
      THEN
         rec.last_name := SUBSTR (rec.last_name, 20);
      END IF;
      process_employee_history (rec.last_name, today);
   END LOOP;
END;
/
```

slowalg_q1.sql
slowalg_a1.sql

Do No Unnecessary Thing - 2

- This program is running slowly. How can I improve it?
  - This is a test of analyzing algorithms for unnecessary and/or slow program performance, *and* tuning of DBMS_SQL code.

```
CREATE OR REPLACE PROCEDURE insert_many_emps
IS
   cur INTEGER := DBMS_SQL.open_cursor;
   rows_inserted INTEGER;

BEGIN
   DBMS_SQL.parse (cur,
      'INSERT INTO emp (empno, deptno, ename)
         VALUES (:empno, :deptno, :ename)',
      DBMS_SQL.native);

   FOR rowind IN 1 .. 1000
   LOOP
      DBMS_SQL.bind_variable (cur, 'empno', rowind);
      DBMS_SQL.bind_variable (cur, 'deptno', 40 * rowind);
      DBMS_SQL.bind_variable (cur, 'ename', 'Steven' || rowind);
      rows_inserted := DBMS_SQL.execute (cur);
   END LOOP;

   DBMS_SQL.close_cursor (cur);
END;
```

slowsql_q2.sql
slowsql_a2.sql
slowsql_a2.tst
loadlots*.*

IF There Are Too Many IFs...

- How can I optimize this code?

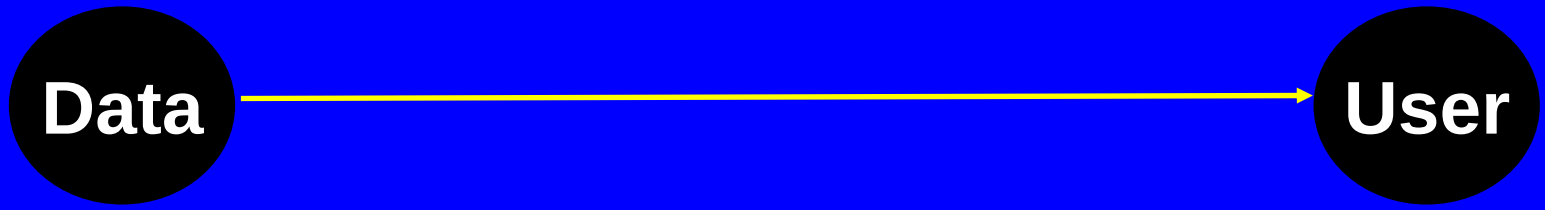
```
PROCEDURE exec_line_proc (line IN INTEGER)
IS
BEGIN
   IF line = 1 THEN exec_line1; END IF;
   IF line = 2 THEN exec_line2; END IF;
   IF line = 3 THEN exec_line3; END IF;
   ...
   IF line = 2045 THEN exec_line2045; END IF;
END;
```

slowalg_q2.sql
slowalg_a2.sql

PL/SQL Tuning and Best Practices

Use Data Structures Efficiently

Shortest Path Between Two Points?



- What's the shortest/fastest way to connect these two?
  - Keep the data as close as possible to the user/program that needs the data.
- Packages offer an ideal caching mechanism.
  - Any data structure defined at the package level (whether in specification or body) serves as a persistent, global structure.
  - Remember: separate copy for each connection to Oracle

Cache Session-static Information

- Great example: the USER function.
  - The value returned by USER never changes in a session.
  - Each call to USER is in reality a SELECT FROM dual.
  - So why do it more than once?

```
CREATE OR REPLACE PACKAGE thisuser
IS
   FUNCTION name RETURN VARCHAR2;
END;

CREATE OR REPLACE PACKAGE BODY thisuser
IS
   /* Persistent "global" variable */
   g_user VARCHAR2(30) := USER;

   FUNCTION name RETURN VARCHAR2 IS
   BEGIN
      RETURN g_user;
   END;
END;
```

- Hide package data!
  - If exposed, you cannot guarantee integrity of data.
  - Build "get and set" programs around it.

thisuser.pkg
thisuser.tst
emplu.*

Leverage Oracle Hashing

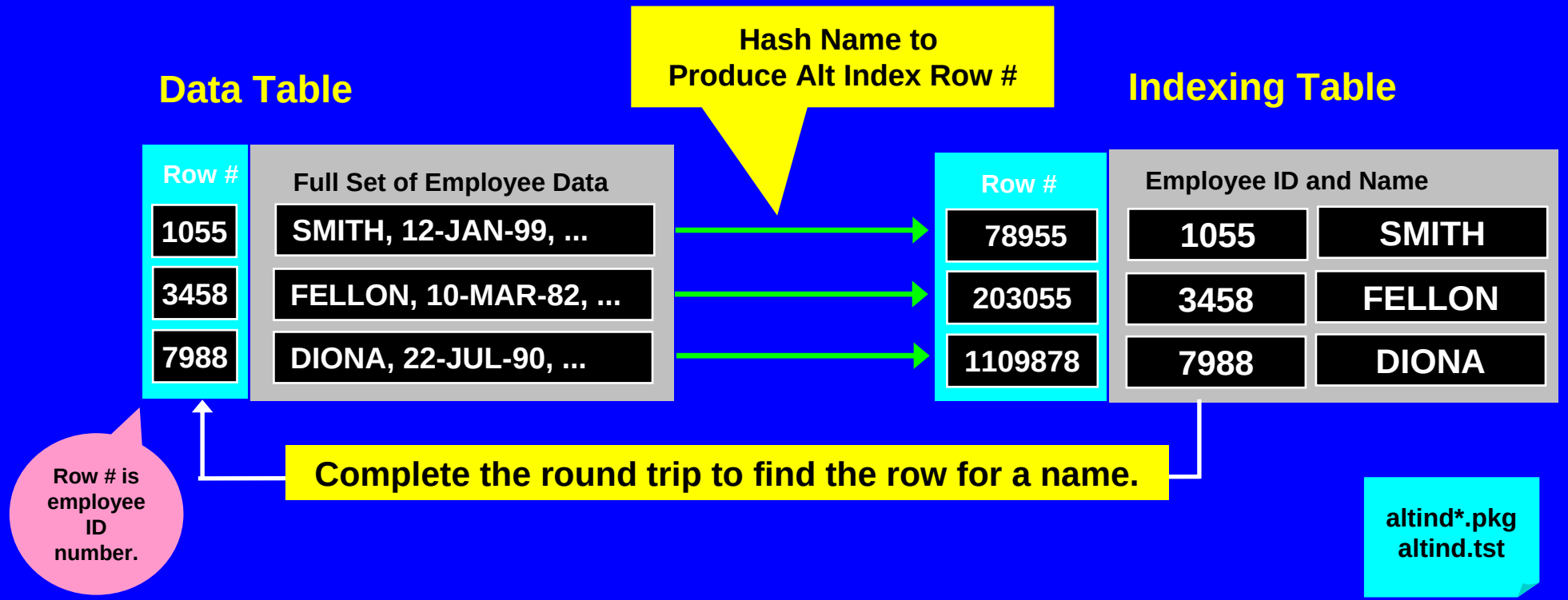
- Hashing algorithms transform strings to numbers.
  - Standard usage: generate unique values for distinct strings.

```
FUNCTION DBMS_UTILITY.GET_HASH_VALUE
   (name IN VARCHAR2,
    base IN NUMBER,
    hash_size IN NUMBER)
RETURN NUMBER;
```

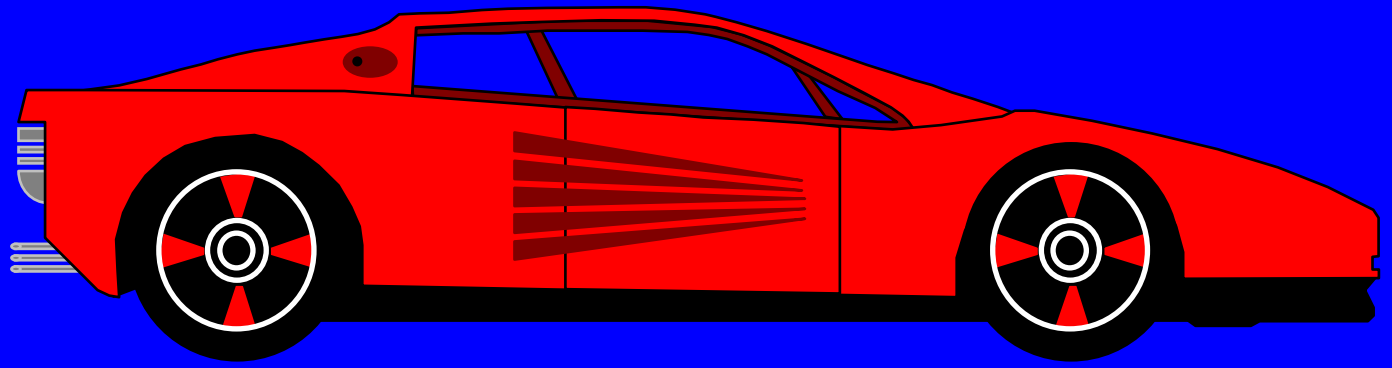
- Provide the string, the base or starting point, and the hash size (total number of possible return values).
- Tips for hashing:
  - You must use the same base and hash size to obtain consistent hash values.
  - Maximum hash size is upper limit of BINARY_INTEGER: 2**31-1.
  - No guarantee that two different strings will *not* hash to the same number. Check for and resolve conflicts.

Hashing for an Alternative Index

- Index-by tables allow only a single index -- the row number.
  - So to locate the row in which a particular string is located, you have to do a "full table scan" -- or do you?
- Use the hash function to build an alternative index to the contents of the PL/SQL table.



Yes, *You* Can Write Blazing Fast PL/SQL!



- VROOOM, VROOM!
  - (This may be the closest you get to driving around in the types of cars preferred by Our Most Exalted Larry Ellison)
- Tuning PL/SQL code is an iterative and incremental process.
  - You are unlikely to uncover a "silver bullet" that is *not* related to some SQL statement.
  - You can, however, have a substantial impact on the performance of your and others' code.

Closing Comments

- Write code with efficiency in mind, but save intensive tuning until entire components are complete and you can perform benchmarking.
- ***MOST IMPORTANT!*** Avoid repetition and dispersion of SQL statements throughout your application.
- Be especially careful to analyze code executed within loops (including SQL statements).
- PL/SQL code is executed from shared memory. You *must* tune the shared pool to avoid excessive swapping of code.

Visit the PL/SQL Pipeline (www.revealnet.com/plsql-pipeline) to share what you learn about tuning and to get your questions answered.

PL/SQL Happy Hour!

- Sponsored by O'Reilly and Associates and the Oracle PL/SQL Development Team.
- Drinks, snacks, discussion and "news you can use" from:
  - Chris Racicot, Senior Manager, PL/SQL and Precompilers, Oracle Corporation
  - Steve Muench, author of Building Oracle XML Applications, and Lead XML Evangelist & Consulting Product Manager BC4J & XSQL Servlet DevelopmentTeams
  - Bill Pribyl, author of Oracle PL/SQL Language Pocket Reference, andco-author of Oracle PL/SQL Language Pocket Reference
  - And I'll throw in a few words on utPLSQL, a fantastic new utility for unit testing of PL/SQL code.

Monday, October 2, 2000
San Francisco Marriott Hotel, 55 Fourth Street
6:30 pm - 8:30 pm