PL/SQL Practicum:
On Fetching and Explaining SQL from the SGA

John Beresniewicz
Technology Manager
Precise Software Solutions
Design by Contract

Design by Contract is a powerful metaphor for software construction. It leads developers and managers to view the construction of a software system as consisting of a large number of contract decisions, large and small, between modules cooperating toward a common goal.

Bertrand Meyer, Object Success
Design by Contract

- **Preconditions**: what must be true upon entering the module?
- **Postconditions**: what does the module promise will be true upon exit?
- **Invariants**: what does the module promise not to change?
SQL in the SGA

- Oracle caches SQL in library cache
  - Query plan and other execution info
- Pointed to by hash_value and address
- Pointers located in several (V$) places
  - Full text available in V$SQLTEXT
V$ pointers to cached SQL

- **V$SQLAREA**
  - Aggregate resource metrics across all child cursors
  - Expensive due to use of group functions
  - Problem: Only 1000 characters of SQL text

- **V$SESSION**
  - Current and previous SQL statement for session

- **V$SQL**
  - Same as V$SQLAREA but not aggregated over children, less expensive to query
Piecing SQL text together

- **V$SQLTEXT**
  - Full SQL statement cut into ordered 64 byte pieces
  - Concatenate SQL_TEXT column value for successive pieces, ordered by PIECE column
  - Address and Hash_value are the key

- How about a PL/SQL function to do this?
  - Accept address/hash and return SQL text
Function SQLtxt (v1)

```sql
FUNCTION SQLtxt
    (hash_IN IN v$sqltext.hash_value%TYPE,
     addr_IN IN v$sqltext.address%TYPE)
RETURN VARCHAR2
```

- Can be used in SQL statements
- Anchored parameter declarations
- Returns PL/SQL varchar2 (up to 32767 chars)
- Cursor FOR loop driven by module parameters
## The driving cursor

<table>
<thead>
<tr>
<th>CURSOR sqlpiece_cur</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
</tr>
<tr>
<td>select piece, sql_text</td>
</tr>
<tr>
<td>from v$sqltext</td>
</tr>
<tr>
<td>where hash_value = hash_IN</td>
</tr>
<tr>
<td>and address = addr_IN</td>
</tr>
<tr>
<td>order by piece;</td>
</tr>
</tbody>
</table>

- Cursor-for loop on this does cursor does it all
SQLtxt contract elements

• Return full text of SQL identified by IN parameters (or NULL)
• Callable from SQL statements
  – WNDS purity inside pre-8i packages
• Not raising exceptions is an extremely valuable invariant
  – Exceptions change the “state” of the system
The “problem” that isn’t…yet

• SQL text could exceed 32K in length
  – Could it really?

• Function SQLtxt could raise exception
  – Should we catch and process?
  – Should we process and not raise?

• Calling programs may need to handle
  – Exception decreases module usability
1\textsuperscript{st} strategy: quick elimination

```
EXCEPTION
    WHEN OTHERS THEN
        temp_sqltxt := SQLERRM(SQLCODE);
        RETURN temp_sqltxt;
```

- Error messages will fit into VARCHAR2
- Function will not raise an exception
- However, we have violated contract
Better exception externalization

```sql
EXCEPTION
    WHEN OTHERS THEN
        temp_msg := SQLERRM(SQLCODE);
        debug(SYSDATE,temp_msg);
        RETURN NULL; -- or temp_sqltxt;
```

- Debug procedure logs time-stamped errors
- Returning NULL is within contract
- However, module dependence is introduced
Avoiding the exception

- Catch exception and return whatever we have to that point
- Allow caller to control how much text to return up to 32K

- Both solutions can return truncated SQL
- Cannot explain truncated SQL, problem?
- Should we return NULL instead?
Function SQLtxt (v2)

FUNCTION SQLtxt
  (hash_IN IN sys.v_$sqltext.hash_value%TYPE,
   addr_IN IN sys.v_$sqltext.address%TYPE,
   maxlen_IN IN IN INTEGER := 32767)
RETURN VARCHAR2

• New parameter maxlen
  – Caller controls size of return varchar2
  – Defaults to existing (v1) signature and behavior
• Eliminate synonym-based variable anchoring
New variables = new contract

- Precondition: Maxlength_IN should be NOT NULL integer between 0 and 32767
- Do not want a possible new exception
  - Force the precondition to be true

```sql
-- force maxlen between 0 and 32767
maxlength := GREATEST(
    LEAST(NVL(maxlength_IN,0),32767),0);
```
SQLtxt v2 Function Body

• Open cursor-for loop on sqlpiece_cur
  – Add whole piece and track total length if this will not exceed maxlen
  – Otherwise add substring until maxlen reached

• RETURN entire SQL text or maxlen size substring of it

• Brute-force implementation, not elegant
  – But… it meets design criteria
Problems with v2

• Boundary analysis: maxlength=1 and SQL size 32767?
  – Loop is run 32766 times too many!

• Cursor-for is the wrong loop
  – It was right for v1, where it originated

• Solution: simple loop with explicit exit when SQL reaches maxlength size
Function SQLtxt (v3)

- Simple LOOP on sqlpiece_cur
- Exit condition reads clearly
  - No more pieces OR textsize = maxlength
- Lesson: added features may induce changes to trusted code sections
- The IF…END IF text size tracking is ugly
Function SQLtxt (v4)

- Replace ugly IF…END IF with simple assignment
- Local function `cur_length` returns size of text so far
  - Used in exit condition and assignment statement
  - It also protects us against NULL mistakes
- May be(?) less efficient but much more elegant
Explaining cached SQL

- Use a dedicated PLAN_TABLE…play nice
- Explain as the correct user
  - Parsing_schema_id in V$SQLAREA
  - ALTER SESSION SET CURRENT SCHEMA
- Explain full text of SQL using SQLTx
- Collect SQL text and parse users first, then explain iteratively
XplnAll.SQL

- Loads full SQL select stmt text and parse users into PL/SQL index-by tables
  - Indexed by hash_value (note potential problem)
- Loop through tables using FIRST and NEXT
  - Set current_schema in session
  - Explain the SQL
  - Native dynamic SQL makes it easy!
- Report on PLAN_TABLE
  - See Burleson article in Oracle Magazine
Objectives

• Learn where V$ SQL pointers can be found
• Implement full SQL text retrieval in a SQL-callable PL/SQL function
• Engage best practice considerations for developing more bullet-proof modules
• Use Oracle8i PL/SQL features
• Appreciate the usefulness and power of server-side PL/SQL modules and packages
Resources

Contact info

- jberesni@precise.com
- www.precise.com