Dynamic SQL in the 11g World

Michael Rosenblum
www.dulcian.com

June 5, 2012
Who Am I? – “Misha”

- Oracle ACE
- Co-author of 2 books
  - PL/SQL for Dummies
  - Expert PL/SQL Practices
- Won ODTUG 2009 Speaker of the Year
- Known for:
  - SQL and PL/SQL tuning
  - Complex functionality
    - Code generators
    - Repository-based development
Why am I giving this presentation? – Because…

- Dynamic SQL is my favorite toy – and I like to talk about it!
  - ODTUG 2007 – Dynamic SQL in Dynamic World
  - ODTUG 2008 – Dynamic SQL: The Sequel

- Developers are still uncomfortable with Dynamic SQL.
  - There are many myths and misunderstandings floating around.

- Majority of people are (finally!) using 11g
  - So, it is time to give an 11g-only presentation.
Just to make sure that we are on the same page...
Dynamic SQL:

- Makes it possible to build and process complete SQL and PL/SQL statements as strings at runtime.
About 90% of Dynamic SQL is covered by a single command (with variations):

```sql
declare
    v_var varchar2(N)|CLOB;
begin
    v_var:='whatever_you_want';
    EXECUTE IMMEDIATE v_var;
end;
OR
begin
    EXECUTE IMMEDIATE 'whatever_you_want';
end;
```
Dynamic Cursors

**Syntax**

```sql
declare
    v_cur SYS_REFCURSOR;
    v_sql varchar2(N) | CLOB := ...;
    v_rec ...%rowtype; -- or record type
begin
    open v_cur for v_sql_tx;
    fetch v_cur into v_rec;
    close v_cur;
end;
```

**Most common use:**

- Processing large datasets with unknown structure
Predecessor of Native Dynamic SQL

**Pros:**
- Separates PARSE and EXECUTE
  - The same query can be reused with different bind variables.
- Works with unknown number/type of INPUT/OUTPUT values

**Cons:**
- Significantly slower
- More difficult to use
11g-Only Features
Agenda

- Support of CLOBs in Native Dynamic SQL
- Cursor transformation
- No datatype restrictions for DBMS_SQL
  - DBMS_SQL now is a complete superset of Native Dynamic SQL
- DBMS_ASSERT is official now.
- Nice-to-have:
  - Better integration
  - On-the-fly wrapper
CLOB as Input

◆ Good news:
  ➢ EXECUTE IMMEDIATE can take CLOB.

◆ Bad news:
  ➢ You need to know how to properly use CLOBs.
    ▪ Concatenation is not such a good idea!
  ➢ If you need to build an on-the-fly PL/SQL object > 32K there may be something wrong with this picture…
    ▪ But it may also be OK (I personally have used these!)
function f_BuildQuery_CL (in_tab_tx varchar2) return CLOB is
  v_out_cl CLOB;
  v_break_tx varchar2(4):='<BR>';
  v_hasErrors_yn varchar2(1):='N';

  v_buffer_tx varchar2(32767);

  procedure p_flush is
  begin
    dbms_lob.writeappend
      (v_out_cl,length(v_buffer_tx), v_buffer_tx);
    v_buffer_tx:=null;
  end;

  procedure p_addToClob (in_tx varchar2) is
  begin
    if length(in_tx)+length(v_buffer_tx)>32767 then
      p_flush;
    end if;
    v_buffer_tx:= v_buffer_tx||in_tx;
  end;
Begin
    dbms_lob.createtemporary
        (v_out_cl,true,dbms_lob.Call);

    p_addToClob(
        'create or replace view V_'||in_tab_tx||
        ' as select ROWNUM num_rows');

    for rec_rep in (select * from my_tab_col
                      where table_name = in_tab_tx)
        loop
        p_addToClob(','||rec_rep.column_tx);
        end loop;

    p_addToClob(' from '||in_tab_tx);

    p_flush; -- write leftovers

    return v_out_cl;
end;
Dynamic Wrapper

- **DBMS_DDL package**
  - **WRAP**
  - **CREATE_WRAPPED (wrap + Execute Immediate)**

- **What does it do?**
  - Objects are created and wrapped on the fly.
    - This is a really good idea for repository-based systems.
    - This feature was introduced long ago, but I only recently discovered it.
  - Unfortunately it does not accept CLOB as input (only Varchar2/Array of Varchar2)
Sample

declare
    v_wrap_tx  varchar2(32767);
    v_ddl_tx   varchar2(32767);
begin
    v_ddl_tx:='create or replace procedure ' ||
              'p_emp is begin null; end;';

    v_wrap_tx:=dbms_ddl.wrap(v_ddl_tx);
    execute immediate v_wrap_tx;
    or

    dbms_ddl.create_wrapped(v_ddl_tx);
end;
Starting with 11g, DBMS_SQL can operate with user-defined object types and collections.

- Exception: BIND_ARRAY – no support of user-defined collections
- Workaround: PL/SQL block with OUT-variable of a needed type
Object Type Case

◆ Setting
  ➢ There is a universal value list builder that returns an object collection.

◆ Task
  ➢ Input:
    ▪ SQL query with value list object collection as bind variable
    ▪ Object collection
  ➢ Output:
    ▪ Detailed description of the query
    ▪ Opened SQL cursor
  ➢ Usage:
    ▪ Applications with on-the-fly query builders
create type lov_othy as object
    (id_nr number, display_tx varchar2(4000));
create type lov_nt is table of lov_othy;

Create function f_getlov_nt
    (i_table_tx varchar2,
     i_id_tx varchar2,
     i_display_tx varchar2,
     i_order_tx varchar2)
return lov_nt is
    v_out_nt lov_nt := lov_nt();
begin
    execute immediate
        'select lov_othy(' || i_id_tx || 
        ',' || i_display_tx || ')
        from ' || i_table_tx || 
        ' order by ' || i_order_tx
    bulk collect into v_out_nt;
    return v_out_nt;
end;
PROCEDURE p_prepareSQL
   (i_sql_tx in varchar2,
    i_lov_nt in lov_nt,
    o_cur OUT SYS_REFCURSOR,
    o_structure_tx OUT varchar2)
is
   v_cur INTEGER;
   v_result_nr integer;

   v_cols_nr number := 0;
   v_cols_tt dbms_sql.desc_tab;
begin
   v_cur:=dbms_sql.open_cursor;
   dbms_sql.parse(v_cur, i_sql_tx, dbms_sql.native);

   dbms_sql.describe_columns(v_cur, v_cols_nr, v_cols_tt);

   for i in 1 .. v_cols_nr loop
      o_structure_tx:=o_structure_tx||'|'|v_cols_tt (i).col_name;
   end loop;

   dbms_sql.bind_variable(v_cur, 'NT1',i_lov_nt);
   v_result_nr:=dbms_sql.execute(v_cur);

   o_cur:=dbms_sql.to_refcursor(v_cur);
end;
SQL> declare
2    v_ref_cur SYS_REFCURSOR;
3    v_columnList_tx varchar2(32767);
4    v_lov_nt lov_nt:=
5        f_getlov_nt('DEPT','DEPTNO','DNAME','DEPTNO');
6    v_sql_tx varchar2(32767):=
7        'select * '||chr(10)||
8        'from emp '||chr(10)||
9        'where deptno in ('||chr(10)||
10       '    select id_nr'||chr(10)||
11       '    from table(cast (:NT1 as lov_nt))'||chr(10)||
12       '    )';
13  begin
14      p_prepareSQL(v_sql_tx,
15      v_lov_nt,
16      v_ref_cur,
17      v_columnList_tx);
18      dbms_output.put_line('Columns:'||v_columnList_tx);
19      if v_ref_cur%isopen then
20          dbms_output.put_line('Valid Cursor!');
21      end if;
22  end;
23  /
Columns: |EMPNO|ENAME|JOB|MGR|HIREDATE|SAL|COMM|DEPTNO
Valid Cursor!
PL/SQL procedure successfully completed.
SQL>
Transformation of Cursors

Conversion between REF_CURSOR and DBMS_SQL cursor

- **DBMS_SQL.TO_CURSOR_NUMBER**
  - From: REF CURSOR
  - To: DBMS_SQL cursor
  - Condition: REF CURSOR must be opened

- **DBMS_SQL.TO_REFCURSOR**
  - From: DBMS_SQL cursor
  - To: REF CURSPR
  - Condition: DBMS_SQL cursor must be opened, parsed and executed
Operational Details

- **Fetching:**
  - Fetching pointer is preserved.
    - It does not matter which side is doing the FETCH.
  - Transformation to REF CURSOR is allowed only if previous FETCH returned something.
    - i.e. CUR%NOTFOUND is still TRUE

- **Accessibility:**
  - Cursors that are being transformed (FROM-side) are automatically closed during the transformation and no longer accessible afterwards.
The problem:

- A lot of REF Cursors in the system with no clear way of figuring out what exactly they are

The solution:

- Generic routine to describe REF Cursor with minimal impact on the system
Explaining REF Cursor (2)

procedure p_expCursor
  (io_ref_cur IN OUT SYS_REFCURSOR) is
    v_cur integer;
    v_cols_nr number := 0;
    v_cols_tt dbms_sql.desc_tab;
begin
  v_cur := dbms_sql.to_cursor_number (io_ref_cur);
  DBMS_SQL.describe_columns
    (v_cur, v_cols_nr, v_cols_tt);
  for i in 1 .. v_cols_nr loop
    dbms_output.put_line
      ('*'|v_cols_tt (i).col_name);
  end loop;
  io_ref_cur := dbms_sql.to_refcursor (v_cur);
end;
declare
    v_ref SYS_REFCURSOR;
    v_emp_rec emp%rowtype;
begin
    open v_ref for
        select * from emp order by ename;

    -- fetch once
    fetch v_ref into v_emp_rec;
    dbms_output.put_line
        (v_emp_rec.empno||'-'||v_emp_rec.ename);
    p_expCursor(v_ref);

    -- fetch twice
    fetch v_ref into v_emp_rec;
    dbms_output.put_line
        (v_emp_rec.empno||'-'||v_emp_rec.ename);

    close v_ref;
end;
SQL> /
7876-ADAMS
*EMPNO
*ENAME
*JOB
*MGR
*HIREDATE
*SAL
*COMM
*DEPTNO
7499-ALLEN

PL/SQL procedure successfully completed.

SQL>
PROCEDURE p_expCursor
  (io_ref_cur IN OUT SYS_REFCURSOR) is
  v_cur integer;
  v_cols_nr number := 0;
  v_cols_tt dbms_sql.desc_tab;
  
  v_sample_tx varchar2(4000);
  
  v_fetch_nr integer;
begin
  v_cur := dbms_sql.to_cursor_number(io_ref_cur);
  DBMS_SQL.describe_columns
    (v_cur, v_cols_nr, v_cols_tt);
  for i in 1 .. v_cols_nr loop
    dbms_output.put_line
      ('*' || v_cols_tt (i).col_name);
  end loop;

  ------------ fetch on this side too --------------
  dbms_sql.define_column(v_cur,2,v_sample_tx,4000);
  v_fetch_nr := dbms_sql.fetch_rows(v_cur);
  dbms_sql.column_value(v_cur,2,v_sample_tx);
  dbms_output.put_line('Fetch>>>'||v_sample_tx);
  
  io_ref_cur := dbms_sql.to_refcursor(v_cur);
end;
SQL> /
7876–ADAMS
*EMPNO
*ENAME
*JOB
*MGR
*HIREDATE
*SAL
*COMM
*DEPTNO
Fetch>>>ALLEN
7698–BLAKE

PL/SQL procedure successfully completed.

SQL>
DECLARE
  v_ref SYS_REFCURSOR;
  v_emp_rec emp%rowtype;
BEGIN
  OPEN v_ref FOR
    SELECT * FROM emp
    WHERE ROWNUM < 1
    ORDER BY ename;
  FETCH v_ref INTO v_emp_rec;
  DBMS_OUTPUT.PUT_LINE (v_emp_rec.empno || '-' || v_emp_rec.ename);
  P_EXP_CURSOR(v_ref);
  CLOSE v_ref;
END;

Now
V_REF%NOTFOUND = TRUE

Will fail with
"ORA-01001: invalid cursor"
calling
DBMS_SQL.TOFRECURSOR
Fighting Code Injections

◆ Old techniques:
  ➢ USE BIND VARIABLES!!!
    ▪ Because bind variables are injection-proof
  ➢ Use repositories for structural elements.
    ▪ Although, repository-based systems are a bit hard to sell nowadays…

◆ New technique:
  ➢ DBMS_ASSERT package
    ▪ If you cannot use repositories but still want to make sure that your code is protected
Validating strings:

- SQL_OBJECT_NAME (string) – checks whether or not string is a valid object
- SIMPLE_SQL_NAME – checks whether or not string is a valid simple SQL name
- SCHEMA_NAME – validates that passed string is a valid schema
- ENQUOTE_NAME – adds a second quote to every instance in the name (and double quotes around)
- ENQUOTE_LITERAL – adds single quotes
function F_GET_col_TX
    (i_table_tx, i_showcol_tx, i_pk_tx, i_pkValue_nr)
return varchar2 is
    v_out_tx varchar2(4000);
    v_sql_tx varchar2(32000);
Begin
    v_sql_tx:='select to_char('||
        dbms_assert.simple_sql_name(i_showcol_tx)||
        ')' from '||
    dbms_assert.simple_sql_name(i_table_tx)||
    ' where '||
    dbms_assert.simple_sql_name(i_pk_tx)||
    '=:v01';
    EXECUTE IMMEDIATE v_sql_tx INTO v_out_tx
    USING i_pkValue_nr;
    return v_out_tx;
end;
More DBMS_ASSERT - 1

◆ You can check syntax yourself 😊, but some things you cannot find in docs:

➢ If DB-link is specified, only syntax is checked:

```sql
SQL> select DBMS_ASSERT.SQL_OBJECT_NAME('DBMS_ASSERT@DUMMY_DBLINK') check_yn 2 from dual;
CHECK_YN
---------------------------
DBMS_ASSERT@DUMMY_DBLINK
```

➢ Schema name is case-sensitive only sometimes!

```sql
SQL> select DBMS_ASSERT.SCHEMA_NAME('Scott') from dual;
select DBMS_ASSERT.SCHEMA_NAME('Scott') from dual *
ERROR at line 1:
ORA-44001: invalid schema
ORA-06512: at "SYS.DBMS_ASSERT", line 266
SQL> select DBMS_ASSERT.SQL_OBJECT_NAME('Scott.emp') check_yn 2 from dual;
Check_yn
-----------------------
Scott.emp
```
Otherwise object names are case-sensitive only if wrapped in double-quotes

```sql
SQL> select DBMS_ASSERT.SQL_OBJECT_NAME('DBMS_ASSERT') check_yn from dual;

CHECK_YN
-------------------------
DBMS_ASSERT

SQL> select DBMS_ASSERT.SQL_OBJECT_NAME('"DBMS_ASSERT"') check_yn from dual;

CHECK_YN
-------------------------
"DBMS_ASSERT"

SQL> select DBMS_ASSERT.SQL_OBJECT_NAME('"dbms_assert"') check_yn from dual;
select DBMS_ASSERT.SQL_OBJECT_NAME('"dbms_assert"') check_yn from dual

* ERROR at line 1:
ORA-44002: invalid object name
ORA-06512: at "SYS.DBMS_ASSERT", line 316

SQL>
```
Integration with Other Features

- PL/SQL team really tries to make Dynamic SQL better integrated with other features
  - RESULT CACHE auto-detects objects referenced via Dynamic SQL call [11.2 only!]
-- Original function
create or replace function f_getCount_nr
  (i_tab_tx varchar2)
return number
result_cache
is
  v_sql_tx varchar2(256);
  v_out_nr number;
begin
  execute immediate
    'select count(*) from ' || i_tab_tx into v_out_nr;
  return v_out_nr;
end;
SQL> select f_getCount_nr('EMP') from dual;
F_GETCOUNT_NR('EMP')
---------------------
          14
SQL> select ro.id, ro.name, do.object_name
2 from v$result_cache_objects ro,
3 v$result_cache_dependency rd,
4 dba_objects do
5 where ro.id = rd.result_id
6 and rd.object_no = do.object_id;
ID | NAME | OBJECT_NAME
---|------|-----------------
1  | "SCOTT"."F_GETCOUNT_NR":8."F_GETCOUNT_NR"#8440831613f0f5d3 | EMP
1  | "SCOTT"."F_GETCOUNT_NR":8."F_GETCOUNT_NR"#8440831613f0f5d3 | F_GETCOUNT_NR

SQL> select f_getCount_nr('EMP') from dual;
F_GETCOUNT_NR('EMP')
---------------------
          14
SQL> select *
2 from v$result_cache_statistics
3 where name in ('Create Count Success','Find Count');
ID | NAME                  | VALUE
---|-----------------------|------
5  | Create Count Success  | 1    
7  | Find Count            | 1    
SQL>
SQL> insert into emp(empno) values (100);
1 row created.
SQL> commit;
Commit complete.
SQL> select f_getCount_nr('EMP') from dual;
F_GETCOUNT_NR('EMP')-------------------
15
SQL> select id, name, value
    2  from v$result_cache_cache_statistics
    3  where name in ('Create Count Success',
    4    'Find Count','Invalidation Count');

   ID   NAME                VALUE
    ---  ---------------------  ------
     5 Create Count Success  2
     7     Find Count        1
     8    Invalidation Count  1

SQL>

Cache was invalidated and rebuilt
Result Cache - 4

SQL> select f_getCount_nr('DEPT') from dual;
F_GETCOUNT_NR('DEPT')---------------------
           4
SQL> select id, name, object_name
2      from v$result_cache_objects  ro,
3            v$result_cache_dependency rd,
4        dba_objects  do
5      where ro.id = rd.result_id
6       and rd.object_no = do.object_id;

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>OBJECT_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>&quot;SCOTT&quot;.&quot;F_GETCOUNT_NR&quot;::8.&quot;F_GETCOUNT_NR&quot;#8440831613f0f5d3</td>
<td>DEPT</td>
</tr>
<tr>
<td>3</td>
<td>&quot;SCOTT&quot;.&quot;F_GETCOUNT_NR&quot;::8.&quot;F_GETCOUNT_NR&quot;#8440831613f0f5d3</td>
<td>EMP</td>
</tr>
<tr>
<td>3</td>
<td>&quot;SCOTT&quot;.&quot;F_GETCOUNT_NR&quot;::8.&quot;F_GETCOUNT_NR&quot;#8440831613f0f5d3</td>
<td>F_GETCOUNT_NR</td>
</tr>
<tr>
<td>4</td>
<td>&quot;SCOTT&quot;.&quot;F_GETCOUNT_NR&quot;::8.&quot;F_GETCOUNT_NR&quot;#8440831613f0f5d3</td>
<td>F_GETCOUNT_NR</td>
</tr>
</tbody>
</table>

SQL> select id, name, value
2      from v$result_cache_statistics
3       where name in ('Create Count Success',
4            'Find Count','Invalidation Count');

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Create Count Success</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>Find Count</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>Invalidation Count</td>
<td>1</td>
</tr>
</tbody>
</table>

New cache with new dependency
Summary

- Dynamic SQL continues to evolve.
- Starting with 11g, DBMS_SQL has all functionality (kind of) available to Native Dynamic SQL.
- It is an advanced feature. Please, be careful while using it.
- Check all examples BEFORE using in production. 😊
Contact Information

- Michael Rosenblum – mrosenblum@dulcian.com
- Blog – wonderingmisha.blogspot.com
- Website – www.dulcian.com

Available now:

*Expert PL/SQL Practices*

for Oracle Developers and DBAs

Deep insights into database programming with PL/SQL

Author’s Web site includes all code samples

Dr. Paul Dorsey
Michael Rosenblum

Apress*