Effective Utilization of the Database in Web Development

Dr. Paul Dorsey
Dulcian, Inc.

NYOUG 25th Anniversary/
NYC Metro Area Oracle Users Group Meeting
December 8, 2009
Background

- Fusion technology stack is large and complex.
- Hard to make the transition into the J2EE environment.
- Host of different tools, programming languages, architectures, and technologies
- Projects often have the illusion of progress.
- Building functioning, scalable production software often becomes an impossible task.
Why do OO people avoid the database?

- Culture?
- Lack of knowledge?
- Clinical pathology?
“Frameworkaphobia”

Definition:
- An irrational avoidance of frameworks (particularly non-open source)

Diagnostic Indications:
- Desire to build everything him/herself
- “If I don’t build it, it must stink.”
- “If Oracle built it, it must really stink.”
- Irrational avoidance of Application Development Framework – Business Components (ADF BC)

Symptoms:
- Higher than expected project cost
- Project failure

Treatment
- No known cure
- Some success with short leashes and large bats

Related conditions
- Megalomania
- Paranoid delusions
“Database Avoidance Syndrome”

Definition:
- An aversion to placing any logic in the database

Diagnostic Indications:
- “We should be database-independent.”
- “Databases are old fashioned. Everyone is coding this way.”

Symptoms:
- Twice as much code as is necessary
- Performance is 10 times slower.
- Network traffic is 100 times as great.
- Four times the load on the database server
- Three times the development time

Treatment
- Direct application of logic (restraints probably required)

Related conditions
- Technical conformity
“SOAphilia”

Definition:
- Irrational desire to refactor small systems to use web services and BPEL

Diagnostic Indications:
- Ownership of 72 BPEL books
- Desire to use BPEL for data-centric processes

Symptoms:
- Projects only succeed with excessive time and funding.

Treatment
- Load testing
- Limit funding

Related conditions
- Herd mentality
“Thick Database” Defined (1)

- Micro-Service-Oriented-Architecture (M-SOA) approach
- Service Component Architecture (SCA)
- Division between the database and user interface (UI) portions.

- Two key features involved in "thick database thinking":
  - Nothing in the UI ever directly interacts with a database table. All interaction is accomplished through database views or APIs.
  - Nearly all application behavior (including screen navigation) is handled in the database.

- Thick database does not simply mean stuffing everything into the database and hoping for the best.
“Thick Database” Defined (2)

- Creating a thick database makes your application UI technology-independent.
  - Creates reusable, UI technology-independent views and APIs.
  - Reduces the complexity of UI development.
  - Database provides needed objects.
  - Reduces the burden on the UI developer
Thick Database Benefits

◆ Minimizes development risk
◆ Helps build working applications that scale well.
◆ Benefit Metrics:
  - Better performance (10X)
  - Less network traffic (100X)
  - Less code (2X)
  - Fewer application servers (3X)
  - Fewer database resources (2X)
  - Faster development (2X)
Easier to Refactor

- UI technology stack changes are common.
- The .Net vs. Java EE battle rages on.
- Web architecture is more volatile than the database platform.
- Defense against the chaos of a rapidly evolving standard.
- Test: What is the probability that your web UI standards will be the same in 18 months?

Answer 0%
How Thick is too Thick?

- What would happen if 100% of all UI logic were placed in the database?
  - Tabbing out of a field
  - LOV populated from database
  - Page navigation

- Pathologically complete way to implement the thick database approach.

- A system built this way would be sub-optimal.
  - But it works
How Thin is too Thin?

◆ Can a skilled team successfully build applications that are 100% database “thin”?
  ➢ Requires a highly skilled team.
  ➢ Minimize round trips
  ➢ ANY middle tier technology (e.g. BPEL) can also be a performance killer.

◆ Possible but difficult
Thick Database Development Process

- Two portions of an application can be coded independently
  - Teams can work in isolation until substantive portions are working.

- First version of the UI is built within a few days
  - Use as testing environment for the database team
  - Feedback can be received from users.

- Use Agile process
  - Minimal design work done to produce a partially working system.
  - Additional functionality created in an iterative design process.
Interface Stubbing

- Stub out the code for the views and APIs.
  - `select <values> from dual`
  - APIs = functions that return a correct value (usually hard-coded).

- Interfaces will change as the application matures.
De-Normalized Views

The idea:
- Convert relational data into something that will make user interface development easier.
- Easiest way to separate data representation in the front-end from the real model.

The solution:
- Use a view with a set of INSTEAD-OF triggers
create or replace view v_customer
as
select c.cust_id,
    c.name_tx,
    a.addr_id,
    a.street_tx,
    a.state_cd,
    a.postal_cd
from customer c
left outer join address a
    on c.cust_id = a.cust_id
create or replace trigger v_customer_ii
instead of insert on v_customer
declare
  v_cust_id customer.cust_id%rowtype;
begin
  if :new.name_tx is not null then
    insert into customer (cust_id, name_tx)
    values (object_seq.nextval, :new.name_tx)
    returning cust_id into v_cust_id;
  if :new.street_tx is not null then
    insert into address (addr_id, street_tx,
                        state_cd, postal_cd, cust_id)
    values (object_seq.nextval, :new.street_tx,
            :new.state_cd, :new.postal_cd, v_cust_id);
  end if;
end;
Function-Based Views: Collections
Using Function-Based Views

- Sometimes it is just not possible to represent all required functionality in a single SQL statement.
- Denormalized view cannot be built.
- Oracle provides a different mechanism:
  - Collections allow you to hide the data separation, as well as all of the transformation logic.
What is a collection?

✦ Definition:
  ➢ An ordered group of elements, all of the same type, addressed by a unique subscript.

✦ Implementation:
  ➢ Since all collections represent data, they are defined as data types.
Collections: Pros & Cons

Three types:
1. Nested tables
2. Associative arrays
3. Variable-size arrays (V-Arrays)

◆ Good news
  ➢ Usually faster
  ➢ Cleaner code
  ➢ Great for UI views

◆ Bad news
  ➢ Not always faster
  ➢ Somewhat annoying syntax
Why use collections?

◆ Logical reason:
  ➢ Collections allow you to articulate and manipulate sets of data.

◆ Technical reason:
  ➢ Processing data in sets is “usually” faster than doing so one element at a time.

◆ Physical reason:
  ➢ Manipulating sets in memory is “usually” 100 times faster than manipulating sets on the storage device.
Possible Issues

- **Technical problem:**
  - Amount of memory is limited (especially in 32-bit architecture)

- **Economic problem:**
  - Storage is cheap – memory is NOT.

- **Learning curve:**
  - People who are used to old habits of processing one row at a time (since COBOL days) will have problems working with sets.
Nested Tables
Nested tables – arbitrary group of elements of the same type with sequential numbers as a subscript

- Undefined number of elements (added/removed on the fly)
- Available in SQL and PL/SQL
- Very useful in PL/SQL! (but not in tables)
Definition:

declare

    type NestedTable is
table of ElementType;

...

create or replace type NestedTable
    is table of ElementType;
Nested Tables (3)

- Nested tables are NOT dense:
  - You can remove objects from inside of the array.
  - Size of the nested table MAY OR MAY NOT equal the subscript of the last element
  - Built-in NEXT and PREVIOUS can go over the gap
declare

    type month_nt is table of VARCHAR2(20);
    v_month_nt month_nt:=month_nt();
    i number;

begin

    v_month_nt.extend(3);
    v_month_nt(1):='January';
    v_month_nt(2):='February';
    v_month_nt(3):='March';
    v_month_nt.delete(2);
    DBMS_OUTPUT.put_line('Count:' || v_month_nt.count);
    DBMS_OUTPUT.put_line('Last:' || v_month_nt.last);
    i:=v_month_nt.first;
    loop
        DBMS_OUTPUT.put_line(v_month_nt(i));
        i:=v_month_nt.next(i);
        if i is null then exit;
    end if;
    end loop;
end;
Nested tables can be used in SQL queries with the special operator: TABLE

- Allows hiding of complex procedural logic “under the hood”
- Nested table type must be declared as a user-defined type (CREATE OR REPLACE TYPE...)

More About Nested Tables
Nested Tables – Example 2a

Specify exactly what is needed as output and declare the corresponding collection:

Create type lov_oty is object
   (id_nr NUMBER,
    display_tx VARCHAR2(256));

Create type lov_nt
   as table of lov_oty;
Nested Tables - Example 2b

◆ Write a PL/SQL function to hide all required logic

```sql
function f_getLov_nt
  (i_table_tx, i_id_tx, i_display_tx, i_order_tx)
return lov_nt is
  v_out_nt lov_nt := lov_nt();
begin
  execute immediate
    'select lov_oty('
      || 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;
```
Nested Tables - Example 2c

Test SQL statement with the following code:

```sql
select id_nr, display_tx
from table(
    cast (f_getLov_nt
    ('emp',
      'empno',
      'ename'||''-'''||job',
      'ename')
    as lov_nt)
)
```
Create a VIEW on the top of the SQL statement.

- Completely hides the underlying logic from the UI
- INSTEAD-OF triggers make logic bi-directional
- Minor problem: There is still no way of passing parameters into the view other than some kind of global.

Create or replace view v_generic_lov as

```sql
select id_nr, display_tx
from table(cast(f_getLov_nt
(GV_pkg.f_getCurTable,
GV_pkg.f_getPK(GV_pkg.f_getCurTable),
GV_pkg.f_getDSP(GV_pkg.f_getCurTable),
GV_pkg.f_getSORT(GV_pkg.f_getCurTable))
as lov_nt)
)```
Optimizing Database Processing
**Associative Arrays (1)**

- **An associative array** is a collection of elements that uses arbitrary numbers and strings for subscript values.
  - PL/SQL only
  - Still useful

<table>
<thead>
<tr>
<th>Year</th>
<th>Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990</td>
<td>December</td>
</tr>
<tr>
<td>1995</td>
<td>June</td>
</tr>
<tr>
<td>2000</td>
<td>April</td>
</tr>
</tbody>
</table>

*Table of varchar2(30) Index by binary_integer*
Definition:

```
declare
    type NestedTable is
        table of ElementType
        index by Varchar2([N]);
...
    type NestedTable is
        table of ElementType
        index by binary_integer;
```
declare

  type dept_rty is record
    (deptNo number, extra_tx VARCHAR2(2000));

  type dept_aa is table of dept_rty
    index by binary_integer;

  v_dept_aa dept_aa;

begin

  for r_d in (select deptno from dept) loop
    v_dept_aa(r_d.deptno).deptNo := r_d.deptno;
  end loop;

  for r_emp in (select ename, deptno from emp) loop
    v_dept_aa(r_emp.deptNo).extra_tx :=
      v_dept_aa(r_emp.deptNo).extra_tx ||
      ' ' || r_emp.eName;
  end loop;

end;
More About Associative Arrays

- Index by VARCHAR2 instead of by BINARY_INTEGER
  - Cannot be used in a FOR-loop
  - Allow creation of simple composite keys with direct access to the row in memory
Prepare memory structure

declare

    type list_aa is table of VARCHAR2(2000)
        index by VARCHAR2(256);

    v_list_aa list_aa;
    cursor c_emp is
        select ename, deptno, to_char(hiredate, 'q') q_nr
        from emp;
    v_key_txt VARCHAR2(256);
begin
    for r_d in (select deptno from dept order by 1) loop
        v_list_aa(r_d.deptno||'|1') :=
            'Q1 Dept#' || r_d.deptno || ':';
        v_list_aa(r_d.deptno||'|2') :=
            'Q2 Dept#' || r_d.deptno || ':';
        ...
end loop;
Process data and present results

... for r_emp in c_emp loop
    v_list_aa(r_emp.deptno||''||r_emp.q_nr):= list_aa(r_emp.deptno||''||r_emp.q_nr)||'
        r_emp.ename;
    end loop;

v_key_tx:=v_list_aa.first;
loop
    DBMS_OUTPUT.put_line(v_list_aa(v_key_tx));
    v_key_tx:=v_list_aa.next(v_key_tx);
exit when v_key_tx is null;
end loop;
end;
Bulk Operations
Bulk operations

◆ Operations on SETs
  ➢ BULK loading into the memory
  ➢ BULK processing
BULK COLLECT (1)

◆ BULK COLLECT clause

➢ The idea:
  ▪ Fetch a group of rows all at once to the collection
  ▪ Control a number of fetched rows (LIMIT)

➢ Risks:
  ▪ Does not raise NO_DATA_FOUND
  ▪ Could run out of memory
BULK COLLECT (2)

- Syntax:

  ```sql
  select ... 
bulk collect into Collection
  from Table;
  
  update ... 
  returning ... bulk collect into Collection;
  
  fetch Cursor 
bulk collect into Collection;
  ```
BULK COLLECT example

declare
    type emp_nt is table of emp%rowtype;
    v_emp_nt emp_nt;

cursor c_emp is select * from emp;

begin
    open c_emp;
    loop
        fetch c_emp
        bulk collect into v_emp_nt limit 100;
        p_proccess_row (v_emp_nt);
        exit when c_emp%NOTFOUND;
    end loop;
    close c_emp;
end;
FORALL (1)

◆ FORALL command

➢ The idea:
  ▪ Apply the same action for all elements in the collection.
  ▪ Have only one context switch between SQL and PL/SQL

➢ Risks:
  ▪ Special care is required if only some actions from the set succeeded
FORALL (2)

◆ Syntax:

forall Index in lower..upper
  update ... set ... where id = Collection(i)
  ...
forall Index in lower..upper
  execute immediate '...' using Collection(i);
FORALL (3)

◆ Restrictions:

- Only a single command can be executed.
- Must reference at least one collection inside the loop.
- All subscripts between lower and upper limits must exist.
- Cannot work with associative array INDEX BY VARCHAR2.
- Cannot use the same collection in SET and WHERE.
- Cannot refer to the individual column on the object/record (only the whole object).
DECLARE

TYPE number_nt IS TABLE OF NUMBER;

v_deptNo_nt number_nt := number_nt(10,20);

BEGIN

FORALL i IN v_deptNo_nt.first()..v_deptNo_nt.last()
    UPDATE emp
    SET sal = sal + 10
    WHERE deptNo = v_deptNo_nt(i);

END;
Conclusions

- The #1 critical success factor for any web development is effective utilization of the database.
- PL/SQL is not irrelevant (and it continues to improve).
- Code that needs to access the database is faster if it is placed in the database.
- Database independence is irrelevant
  - UI technology independence is more important.
- Just because everyone is moving logic to the middle tier, does not make it a smart idea.
Share your Knowledge:
Call for Articles/Presentations

**IOUG – The SELECT Journal**
- select@ioug.org
- 500-1,000 words long with a specific focus

**ODTUG – Technical Journal**
- pubs@odtug.com
Sign up for a free Associate Membership and access great technical content

Oracle Development Tools User Group

www.odtug.com

A Real World User Group For Real World Developers
OPP2009
PL/SQL TRAINING
Featuring Steven Feuerstein, Quest Software

www.odtugopp.com

APEXPOSED2009!
APPLICATION EXPRESS TRAINING
Featuring Scott Spendolini, Sumner Technologies

www.odtugapextraining.com

The PL/SQL & APEX Experts Converge!

NOVEMBER 10-11, 2009
Sheraton Gateway Atlanta Airport, Atlanta

TWO TRAINING EVENTS IN ONE!
Dulcian’s BRIM® Environment

- Full business rules-based development environment
- For Demo
  - Write “BRIM” on business card