

Customizing Domain Processing

Leslie M. Tierstein,
W R Systems, Ltd.

Designer/2000 domains are a powerful design and development feature. They allow analysts to document, as early as the logical design phase, code values to be used to validate data entry. At build time, the same domain specifications can be generated into database check constraints and/or online lists of values, radio buttons, or other user interface objects.

However, domains have certain limitations, centered on performance; and user maintainability (or lack thereof); and referential integrity (or lack thereof). This paper presents an approach that has been used at W R Systems to overcome these limitations.

Performance Concerns

The Domains should not be defined for highly volatile codes and/or descriptions. Instead, more traditional code tables should be used to maintain these values. Further, the Oracle documentation warns that domains should not be defined for code tables which will hold more than 250 values. In such cases, the List of Values processing which would be generated for domains would result in unsatisfactory performance.

User Maintainability

Even using the above criteria, most codes could be defined as domains. However, users want to be able, at a minimum, to view the available values, and potentially to add new values, delete unused values, or alter the descriptions associated with existing values? The Designer/2000 toolkit does not include a means for performing domain maintenance in a production environment. Application implementors need to supply this.

Referential Integrity

Once the design team decides that users need to add or delete domain values, developers can no longer generate check constraints based on domains: any change to a domain value would necessitate running an ALTER TABLE command to redefine the check constraint.

Consequently, you are left with client-side only domain validation. Designer/2000 does an adequate job of generating code to do this in WHEN-VALIDATE-ITEM triggers applied to items with associated domains. However, this addresses only half the problem. There is no foreign key constraint between the values of a particular domain and columns validated against those values. Once you allow users to maintain domain values, how can you ensure that referential integrity will be maintained, that they will not be able to delete a domain value that is still being used by records in the database?

Requirements

A analysis of our applications revealed that domain maintenance needed to be meet the following requirements:

- ?? In a multiple-application system, users should be able to view and/or maintain only those domains which are owned by the application(s) to which they have access rights.
- ?? The users' ability to change the contents of some domains – those whose values are directly referenced by procedural code – needs to be restricted.
- ?? In domains over which users have free reign, users still shouldn't be able to delete a value that is currently being used in a column in the database.

Maintaining Domains

To support these requirements, the development team needed to:

- ?? Add elements to the database to allow storing information to support the additional capability
- ?? Wrote SQL scripts to populate the additional database objects
- ?? Provide a form to allow users to view and/or maintain domain values

Design the Database

Designer/2000 creates the REF_CODES table (which stores domain values in a runtime environment) containing the following columns:

Column	Usage
rv_low_value	Discrete domain value to be validated, or the low value of a range.
rv_high_value	High value of the range
rv_abbreviation	Abbreviation assigned to the domain
rv_domain	Domain name
rv_meaning	Meaning assigned to the domain
rv_type	Set to "CG" (CASE Generator) by the Utility to Populate the Reference Codes Table

To support the domain maintenance requirements, we added three columns to the REF_CODES table:

Column	Usage
subsystem_cd	Subsystem (Designer/2000 application) which owns the domain. Our deployed application consisted of multiple functional areas, developed as separate Designer/2000 applications.

Column	Usage
update_ind	Indicator which specifies whether users will be able to update the abbreviation and/or meaning of the domain; the domain value is never updateable
delete_ind	Indicator which specifies whether users will be able to delete the current value from the domain

A crucial requirement was maintaining referential integrity via the equivalent of a RESTRICTED DELETE: if the domain value was in use in any associated column in the database, the value could not be deleted from the domain. To track domain usage in the runtime environment, we designed a DOMAIN_USAGE table which would track every column which had an associated domain:

```
CREATE TABLE DOMAIN_USAGE (
  DOMAIN_NM  VARCHAR2(30) NOT NULL
, TABLE_NM  VARCHAR2(21) NOT NULL
, COLUMN_NM  VARCHAR2(21) NOT NULL
```

The column lengths reflect the maximums allowed by our development standards. The primary key was a composite of all three columns. A program would need to be written, and procedures put in place, to populate this table.

We also added a view to the database, consisting of the name of each domain and the subsystem which owned it. This would let us design a user-friendly interface to the domain maintenance form.

```
CREATE OR REPLACE VIEW DOMAIN_NAME_VW(
  DOMAIN_NM
, SUBSYSTEM_CD)
AS
SELECT DISTINCT RV_DOMAIN,
  SUBSYSTEM_CD
FROM CG_REF_CODES
/
```

Our standards specified that any non-updateable view should end with the _VW suffix.

Implement the Program

The domain maintenance program was developed in Designer/2000 and enhanced in Developer/2000. It required the following work:

1. Enter the revised definition of the REF_CODES table in Designer/2000. The table is actually named CG_REF_CODES if the DVTABL preference is set to "General", so that domains for multiple applications are stored in the same table.

2. Let Designer/2000 create the CG_REF_CODES table. Execute an ALTER TABLE command so the table format fits the revised Designer/2000 definition:

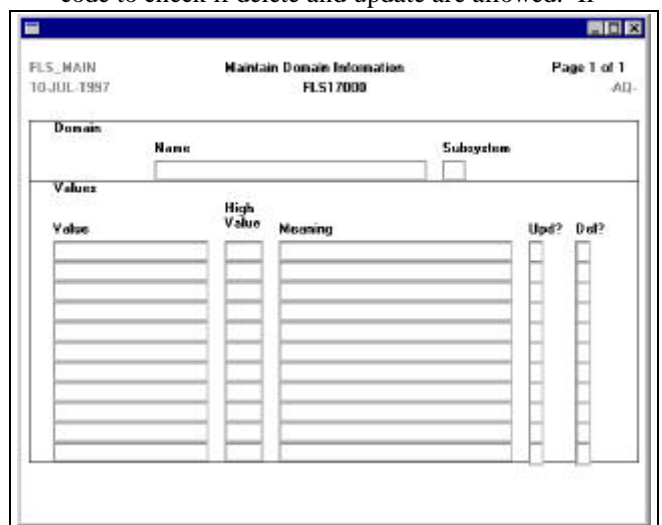
```
ALTER TABLE cg_ref_codes
ADD (subsystem_cd VARCHAR2(3),
  delete_ind VARCHAR2(1),
  update_ind VARCHAR2(1))
```

3. Enter the definition for the DOMAIN_USAGE table, which maintains a list of the columns which use the domain for validation. Generate the DDL and create the table.
4. Enter the definition for the DOMAIN_NAME view which contains distinct domain names and the subsystem which owns the domain. Generate the DDL and create the view.
5. Define a module which allows users to maintain the domains. Base tables are:

- ?? DOMAIN_NAME_VW, which is query-only. The WHERE clause restricts the display to those domains owned by the application which is currently running, and allows users to select the domain they want to maintain
- ?? CG_REF_CODES, which allows SELECT, UPDATE, INSERT, and DELETE.

Generate the form.

6. Customize the form's behavior. Write PL/SQL code to check if delete and update are allowed. If



delete is, in fact, allowed, perform a referential integrity check to ensure that the domain value to be deleted is not currently being used.

Delete Processing

In order to implement the desired behavior if the user attempted to delete a domain value, the following code was written:

- ?? Additional code was inserted before the generated code in the KEY_DELREC trigger for the block based on the CG_REF_CODES table. If the deletion indicator did not allow deleting the current domain value, the user was so informed; otherwise, the form performed its “referential integrity” checks.
- ?? The Forms PL/SQL unit *chk_domain_delete_allowed* was invoked if deleting a domain value was, in principle, permitted. This procedure found the “foreign key references” to the domain by building a cursor from

the DOMAIN_USAGE table consisting of the table.column references associated with the current domain.

- ?? The database function *chk_domain_val* used dynamic SQL to check every table.column reference found for the domain against the value the user was trying to delete. If the value was used anywhere, the deletion was not allowed, and an appropriate alert issued. The function was defined in the Module Logic Diagrammer and returned one value, a Boolean indicating whether or not the deletion was valid.

The PL/SQL code is given below.

```
/* block CGREF, trigger KEY-DELREC */
IF :cgref.delete_ind = 'N' THEN
  msg_alert('This domain value may not be deleted.', 'E', TRUE);
ELSE;
  chk_domain_delete_allowed(:dvn.domain_nm, :cgref.rv_low_value);
END IF;
/* generated code goes here */

/* PL/SQL unit chk_domain_delete_allowed */
PROCEDURE chk_domain_delete_allowed(
  p_domain_name_in IN VARCHAR2,
  p_domain_value_in IN VARCHAR2) IS
  table_nm          CHAR(30);
  column_nm         CHAR(30);
  delete_allowed   BOOLEAN;
  row_cnt           NUMBER(3);
BEGIN
  DECLARE
    CURSOR c_domain_usage IS
  /* Tables/columns which reference the domain */
    SELECT table_nm, column_nm
      FROM domain_usage
     WHERE domain_nm = p_domain_name_in;
  BEGIN
    OPEN c_domain_usage;
    LOOP
      FETCH c_domain_usage
        INTO table_nm, column_nm;
      IF c_domain_usage%NOTFOUND THEN
        EXIT;
      ELSE
        delete_allowed := chk_domain_val(table_nm, column_nm, p_domain_value_in);
        IF NOT delete_allowed THEN
          CLOSE c_domain_usage;
          msg_alert('This value can not be deleted.' || 'It is referenced by ' ||
            UPPER(table_nm) || '.' || UPPER(column_nm) || '.', 'E', TRUE);
        END IF;
      END IF;
    END LOOP;
    CLOSE c_domain_usage;
  EXCEPTION
    WHEN OTHERS THEN
      CGTE$OTHER_EXCEPTIONS;
  END;
END;
```

```

/* Database function chk_domain_val */
cursor_handle    INTEGER;
row_cnt          INTEGER;
bind_variable    VARCHAR2(20);
delete_allowed   BOOLEAN;
BEGIN
  cursor_handle := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(cursor_handle,
    'SELECT 1 ' ||
    ' FROM ' || p_table_nm_in ||
    ' WHERE ' || p_column_nm_in ||
    ' = :col_value' || ' AND rownum = 1',
    DBMS_SQL.V7);
  DBMS_SQL.BIND_VARIABLE(cursor_handle, 'col_value', p_column_value_in);
  row_cnt := DBMS_SQL.EXECUTE (cursor_handle);
  row_cnt := DBMS_SQL.FETCH_ROWS (cursor_handle);
  IF row_cnt = 1 THEN
    delete_allowed := FALSE;
  ELSE
    delete_allowed := TRUE;
  END IF;
  DBMS_SQL.CLOSE_CURSOR(cursor_handle);
  RETURN(delete_allowed);
END;

```

Supporting Software

Two SQL scripts needed to be written and run as part of system deployment:

Script	Description
popdusg.sql	Populate the DOMAIN_USAGE table with the table and column to which each domain is applied.
poprcr.sql	Populate the three additional columns in CG_REF_CODES (the subsystem which owns the domain, the update indicator, and the delete indicator) with the correct values.

Domain Usage Table

The script to populate the domain usage table must be run whenever a developer changes domain usage, for example, by removing domain validation from a column; adding it to a column; or deleting a column which used a domain. In reality, we ran the script at regularly scheduled intervals during the development and testing phases. The script takes one parameter – the current application version; the application names are hard-coded.

```

DELETE FROM domain_usage;
INSERT INTO domain_usage
  (domain_nm, table_nm, column_nm)
SELECT d.name, t.name, c.name
  FROM ci_domains d,
       ci_table_definitions t,
       ci_columns c
 WHERE t.table_type = 'TABLE'

```

```

AND d.application_system_owned_by =
(SELECT id FROM ci_application_systems
 WHERE version = &&APP_VER
 AND name = &&APP_NAME)
AND c.table_reference = t.id
AND t.application_system_owned_by =
(SELECT id FROM ci_application_systems
 WHERE version = &&APP_VER
 AND name = &&APP_NAME)
AND c.domain_reference = dom.id
/

```

Reference Codes Table

We started out with a basic script to update the additional columns in the reference codes table. It set the subsystem (SUBSYSTEM_CD) to the subsystem which owns the domain, and set both the update and delete indicators to “No”. This allowed users with access rights to a particular subsystem to maintain any of the domains owned by that subsystem.

```

UPDATE cg_ref_codes
SET subsystem_cd =
  (SELECT SUBSTR(app.name,1,3)
   FROM ci_application_systems a,
        ci_domains d
   WHERE d.name = rv_domain
        AND d.application_system_owned_by
          = a.id
        AND a.version = &&APP_VER
        AND a.name = &&APP_NAME)
 WHERE subsystem_cd IS NULL
/
UPDATE cg_ref_codes
SET update_ind = 'Y'
 WHERE update_ind IS NULL
/

```

```

UPDATE cg_ref_codes
  SET delete_ind = 'Y'
  WHERE delete_ind IS NULL
/

```

Then, a memo was distributed to all developers: they needed to let the Technical Manager know which domain values the users should not be allowed to update or delete. The guidelines were:

- ?? Specific domains or domain values which trigger different processing options should not be allow deletions.
- ?? Domains should not allow updates if the meaning appears prominently on printed output or official documents. Updates are okay if the meaning and/or abbreviation is used strictly in the on-line user interface.
- ?? Most domains -- those that were purely descriptive -- could allow both updates and deletions.

Code to restrict deletions or updates for the specified domains was appended to the *popcrc* script. For example,

```

UPDATE cg_ref_codes
  SET delete_ind = 'N',
      update_ind = 'N'
  WHERE rv_domain = 'ACCT_TYPE_CD'
/
UPDATE cg_ref_codes
  SET delete_ind = 'N'
  WHERE rv_domain = 'DOC_TYPE_CD'
      AND rv_low_value IN
      ('DUNLTR', 'OVDUE')
/

```

In these examples, users would not be able to delete or modify any account type codes (ACCT_TYPE_CD). They would be able to modify the descriptions associated with any document types (DOC_TYPE_CD), but not delete those corresponding to dunning letters (DUNLTR) and overdue notices (OVDUE).

We had a minor hiccup to this approach when the applications entered integration testing. At that time, ownership of all database objects, including domains, was transferred to one master application, to facilitate configuration management and change control. This meant that we could no longer use the APPLICATION_SYSTEM_OWNED_BY column to populate the SUBSYSTEM_CD column, since all domains would be owned by the same application. We took a cheap way out to fix this: Rather than extending the repository to include the logical domain owner, we used the API to copy this value to the UNIT_OF_MEASURE column, which we weren't using, and changed the *popdusg* script to refer to this column. Then, we transferred table ownership – the domains get transferred automatically with the tables.

Retrofitting Designer/2000 Domain Specifications

Once the applications go into production, and users start maintaining their domains, the CG_REF_CODES table will contain entries that no longer match the contents of the Designer/2000 repository view ci_attribute_values. These entries can be differentiated from domain values written to CG_REF_CODES by the Designer-supplied utility, since the utility assigns the value “CG” to the CG_REF_CODES.RV_TYPE column for all rows which it populates. The user-added domain values can be queried for discrepancies from the repository and/or an API utility written to update the repository.