WSRC-MS-99-00408

Oracle Database Y2K Protection Triggers Generators

Carl Cribbs
Westinghouse Savannah River Company
Aiken, SC 29808

This document was prepared in conjunction with work accomplished under Contract No. DE-AC09-96SR18500 with the U. S. Department of Energy.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

This report has been reproduced directly from the best available copy.

Available to DOE and DOE Contractors from the Office of Scientific and Technical Information, P. O. Box 62 Oak Ridge, TN 37831; prices available from (423) 576-8401.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161.

Administrative

This document is for distribution internally and externally (for off site public domain).

Text enclosed between "<" and ">" are meant to represent generic information.
For example: <date column> is meant to represent date columns in general- NOT THE SPECIFIED "DATE COLUMN."

Synopsis

I developed PL/SQL code that generates or modifies PL/SQL "BEFORE EACH ROW" triggers to protect database date columns from Y2K non-compliant date input (from all sources) into the database. A function is imbedded in the triggers that uses the "RR" year formatted date conversion. For each table with at least one date column and with INSERT/UPDATE/DELETE trigger(s), my code inserts date conversion code into the existing trigger(s). For INSERT/UPDATE not in a trigger(s), my code creates a trigger for the absent DML command(s).

Designed to be: Transferable to other servers with minimum effort; A uniform and consistent problem solution with easy implementation, testing, and configuration management.

No need to manually code and edit SQL trigger files: Modifies existing triggers; Creates needed triggers; Self documented (output comments with code); SQL files configuration management ready. Can customize the: Date conversion function; Code modifications for the trigger; Universal lookup/key; …

Problem

Due to legacy software, we needed to protect database dates columns from Y2K non-compliant dates. That is, to control date data coming into the database. With legacy software of SQL*FORMS 3.0, SQLPLUS, and Pro*FORTRAN, the solution had to be applicable for multiple input sources. Next, it would have to work for future unknown application, upgrades, etc. Additionally, we needed uniform and consistent problem solution for ease of implementation, testing, and configuration management. Then, it would have to be transferable to other servers with minimum effort. With a four month Y2K conversion deadline for our entire Laboratory Information Management System (LIMS), the need was immediate. All this had to be resolved in our legacy software environment.

Software Environment and Software Restrictions

Our in-house LIMS uses OpenVMS V6.2 and VT terminals (or VT emulators) for the system platform. The Oracle software consists of: Oracle7 Server Release 7.1.5.2.3; SQL*Plus Release 3.1.3.5.1; PL/SQL Release 2.1.5.2.0. All our dates are greater than 1980 but less than 2010. Therefore, the most encompassing solution pointed to was triggers.

Triggers and Tables with Dates

To give complete coverage for all date fields within our software context, the approach I used was to both modify existing triggers and create new triggers to contain date conversion code. Since can only change a column's value in "BEFORE" triggers and want the Y2K conversion to affect each record inserted or updated, all triggers upgraded or created are/will be "BEFORE EACH ROW" triggers associated with tables that have at least one date field. Specifically, generates code like:

```
:NEW.timestamp := rr_date_conversion(:NEW.timestamp);
```

Using PL/SQL code that generates PL/SQL triggers that run from SQL command files, I developed a module to upgrade existing triggers and an another module to create needed triggers. This gave full coverage for each table with at least one date field:

Existing BEFORE/EACH ROW Trigger(s) to Upgrade	BEFORE/EACH ROW New Trigger Generated
DELETE	None
INSERT OR UPDATE	None
INSERT OR UPDATE OR DELETE	None
INSERT	UPDATE
INSERT OR DELETE	UPDATE
UPDATE	INSERT
UPDATE OR DELETE	INSERT
None	INSERT OR UPDATE

NOTE: By creating temporary triggers for every combination of INSERT/UPDATE/UPDATE OF/DELETE, all possibilities were successfully tested.

50% of the triggers are "BEFORE EACH ROW" triggers. My pre-Y2K conversion trigger break out base on triggering events is:

i. 10% contain "INSERT OR UPDATE" (most are "INSERT OR UPDATE OR DELETE")
ii. 10% contain "INSERT" (most are "INSERT OR DELETE")
iii. 30% contain "UPDATE" or "UPDATE OF" (most are "UPDATE OR DELETE")

These triggers were upgraded. All created or upgraded triggers are configured to call the date conversion function.

Solution

The created and upgraded triggers use the standard date conversion function "rr_date_conversion." Using the "RR" date format, the date conversion function is based on the last two digits of the year. Each year with its last two digits less than 50 is converted to the 21st century. Each year with its last two digits greater than or equal to 50 is converted to the 20th century. For all new and modified triggers, the date conversion syntax is " (also see Attachment Number 1):

```
<owner>.rr_date_conversion
    (
    yydate IN DATE    /* date to convert to Y2K compliance */
    )
RETURN DATE IS
     rrdate      DATE;   /* Y2K compliant date to return */
```

```
BEGIN
      SELECT
            TO_DATE
            (
                  TO_CHAR (yydate,
                              'DD-MM-YY HH:MI:SS' ),
                  'DD-MM-RR HH:MI:SS'
            )
          INTO
            rrdate
          FROM DUAL;
  RETURN (rrdate);
END;
```

All the triggers call “rr_date_conversion” using the following structure (referenced in this document as “<Date conversion code>”):

```
--
-- Convert date coming into database to be Y2K complaint.
--
      IF :NEW.<date column> IS NOT NULL
      THEN
        :NEW.<date column> :=
          rr_date_conversion( :NEW.<date column> );
      END IF;
       <repeats IF THEN block for each date column>
```

The file “Y2K_TRIGGER_GENERATOR.SQL” (see Attachment Number 2) contains the PL/SQL “Generator” code. By spooling created triggers to “Y2K_DATE_COLUMN_NEW_TRIG_CODE.SQL,” a command file of new/created triggers is created. Additionally, “DROP TRIGGER” commands are spool to Y2K_DROP_RR_TRIGGERS.SQL.” (As shown in the above table, triggers are not generated for existing "BEFORE INSERT…" and/or " BEFORE UPDATE…" triggers.) The basic structure of the created trigger is:

```
<Header information/comments>
BEGIN
        <Date conversion code>
 END;
```

Additionally, all new/created trigger names consists of “RR_” concatenated to the beginning of the trigger’s table name. The creating of triggers methodology is basically a subset of the upgrade of existing triggers.

The file “Y2K_EXISTING_TRIGGER_UPGRADE.SQL” (see Attachment Number 3) contains the PL/SQL code for “Upgrade.” By spooling upgraded triggers to “Y2K_TRIGGER_UPGRADE_CODE.SQL,” it generated the code that inserts date conversion code into existing triggers. The basic structure of the original syntax in “SYS.DBA_TRIGGERS.TRIGGER_BODY” is:

```
BEGIN
        <Original code- less BEGIN/END>
END;
```

The basic structure of the upgraded code is:

```
BEGIN
        <Header information/comments>
        BEGIN
                <Date conversion code>
        END;
```

```
        BEGIN
                <Original code- less BEGIN/END>
        END;
    END;
```

Having the generic solution detailed, the basic logic of the create and upgrade SQL files is outlined next.

Basics of Code Logic

The basic logic used to generate the new triggers (PL/SQL code "Generator") is:

a. Creates list of tables with date column(s).
b. Builds list of existing (if any) "BEFORE EACH ROW" triggers associated with specified table:
   i. Develops new trigger based on triggering events NOT in list of existing "BEFORE EACH ROW" triggers.
   ii. If UPDATE and INSERT triggering events exists, no new trigger.
c. Constructs trigger's code (triggering event,…)
d. Create list of the table's date columns to insert into the date conversion code (represented above by "<Date conversion code>")
e. "DBMS_OUTPUT.PUT_LINE" commands output trigger code and comments to build each trigger.
f. Executes spooled SQL file of built triggers.
g. Creates the SQL command file to "DROP TRIGGER" for all triggers with "RR_" prefixing the trigger name.

The basic logic used to update/upgrade old triggers (PL/SQL code for "Upgrade") is:

a. Creates list of tables with date column(s).
b. Creates list of existing (if any) "BEFORE EACH ROW" triggers associated with specified table (excludes triggers with "RR_" prefix).
c. Reconstructs trigger's code (triggering event, WHEN clause, …).
d. Create list of table columns for code to reconstruct UPDATE OF clause.
h. Create list of date columns for date conversion code (represented above with "<Date conversion code>").
i. "DBMS_OUTPUT.PUT_LINE" commands output code and comments to build each trigger .
e. Must execute the spooled trigger file separately.

This reasonably simple approach yielded the desired results and more.

Benefits

The process of protecting date fields has been automated. Since the SQL files can be run from the SQL*Plus prompt, no special tools are required- nothing to buy. Moreover, there is no need to edit SQL trigger files: Modifies existing triggers; Creates needed triggers; Self documented (output comments with code); SQL files configuration management ready. Available now requiring minimal modification for usage.

Required Customization/Modification

Since this was developed with PL/SQL Release 2.1.5.2.0 and SQL*Plus Release 3.1.3.5.1, each line is prefixed with a tab. Since the original code and the Y2K conversion are imbedded into two separate inner blocks and leading blanks are suppressed/trimmed when using "DBMS_OUTPUT.PUT_LINE" commands, the tab prefix maintains a semblance of the original indentation. This "tab" code can be removed, commented out, or replaced with spacing when using later PL/SQL versions. Next, if already used with objects, may want to replace the "RR_" (newly created trigger) prefix.

Finally, for the provided SQL files to work, the trigger owner must be inserted in various WHERE clauses:

a. Replace "<owner>" in WHERE clauses with desired owner/schema
   or
b. Remove "AND OWNER = <owner>" from the WHERE clauses. Naturally, this implies you create/upgrade triggers for all schemas.

Having to make required changes demonstrates that general customization is feasible.

Possible Alternatives and Enhancements

With your unique date requirement, you can customize "RR_DATE_CONVERSION" for the desired date handling (e.g. for the years 1910 through 2009). If you can determine uniform keying information, the concept of creating triggers/modifying existing triggers can be expanded. (For me it was: SYS.DBA_TAB_COLUMNS.DATA_TYPE = "DATE.") "WHERE" logic can be set up for fields with the same/similar names (e.g. tables with field names of "USER_ID"). In addition to any type of triggering event and/or trigger type, you can look for values in the columns of the SYS.DBA_TRIGGERS or SYS.DBA_TRIGGER_COLS. Also, occurrences of an object in a trigger (e.g. a synonym in SYS.DBA_TRIGGERS "WHEN_CLAUSE" field). Additionally, the "WHEN" clause is a prime candidate to control data coming into the database. If you have something consistent to search for in all the needed triggers, you can make uniform modifications to the trigger codes (found in: SYS.DBA_TRIGGERS.TRIGGER_BODY).

Any one want to integrate these concepts into a GUI product?

Conclusion

Database now has blanket input Y2K protection that meets our requirements. Without extensive editing work by developers, all or selected triggers can be "electronically" modified. Additionally, "electronically" created triggers when the need existed. The concept can be expanded well beyond Y2K protection.

Think in terms of
automating problem solutions

Attachment Number 1
RR_DATE_CONVERSION.SQL

```
/*_____________________________________________________________________________

Function       : rr_date_conversion
Author         : Carl Cribbs
Date Created   : 02-Mar-1999

Description    : Converts dates to be Y2K compliant
Restrictions   : Any year with it last two digits less than 50 is treated as
                 a 21st century date. Any year with it last two less greater
                 than or equal to 50 is treated as a 20th century date.
Parameters     : yydate - the date to convert
Return Value   : rrdate - Y2K compliant date to return
_____________________________________________________________________________*/
CREATE OR REPLACE FUNCTION
   ops$accesslims.rr_date_conversion
      (
      yydate IN DATE   /* date to convert to Y2K compliance */
      )
RETURN DATE IS
   rrdate      DATE;   /* Y2K compliant date to return */

/*_____________________________________________________________________________

  Name  /  Date   /   Comment
-----------------------------------------------------------------------------
        /         /
        /         /
_____________________________________________________________________________*/
BEGIN
/*
    If last two digits of date to convert is less than 50, converts
    to a 21st century date. Otherwise, converts to a 20th century date.
*/
      SELECT
         TO_DATE ( TO_CHAR ( yydate, 'DD-MM-YY HH:MI:SS' ),'DD-MM-RR HH:MI:SS' )
      INTO
         rrdate
      FROM DUAL;

   RETURN (rrdate);
/*
_____________________________________________________________________________
*/
END rr_date_conversion;
/
/*
Make this function accessible by all schemas
*/
CREATE PUBLIC SYNONYM rr_date_conversion FOR ops$accesslims.rr_date_conversion;
```

Attachment Number 2
Y2K_TRIGGER_GENERATOR.SQL

```
SET ECHO OFF
SET FEEDBACK OFF
SET HEADING OFF
SET LINESIZE 80
SET SERVEROUTPUT On SIZE 1000000
SET SHOWMODE OFF
SET TERM OFF
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--
-- File         : Y2K_TRIGGER_GENERATOR.SQL
-- Purpose      : Each table's trigger will convert date data (for each
--                date field) to the year range of 1950 through 2049.
-- Description  : PL/SQL code generates each table's trigger.
-- Author       : Carl Cribbs
-- Date Created : 01-APR-1999
-- Restrictions : a. Only for tables with date fields.
--                b. The date conversion function that is called,
--                   "rr_date_conversion," converts:
--                   i.  All years with it last two digits less than 50 to
--                       a 21st century date.
--                   ii. All years with it last two digits greater than or
--                       equal to 50 to a 20th century date.
--                c. Will not generate triggers for existing "BEFORE INSERT"
--                   and/or "BEFORE UPDATE" triggers:
--                 +----------------------------+-------------------------+
--                 |Existing  BEFORE/EACH ROW   | Type of BEFORE/EACH ROW |
--                 |Trigger for a Table         | Trigger Generated       |
--                 |----------------------------+-------------------------|
--                 | INSERT OR UPDATE           |    None                 |
--                 | INSERT OR UPDATE OR DELETE |    None                 |
--                 | INSERT                     |    UPDATE               |
--                 | INSERT OR DELETE           |    UPDATE               |
--                 | UPDATE                     |    INSERT               |
--                 | UPDATE OR DELETE           |    INSERT               |
--                 | None                       |    INSERT OR UPDATE     |
--                 +----------------------------+-------------------------+
--
--------------------------------------------------------------------------------
--
SPOOL y2k_date_column_new_trig_code.sql
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--
DECLARE

--------------------------------------------------------------------------------
--
-- List of tables with date column(s)
--
-- Each "BEFORE EACH ROW" trigger with both "INSERT" and "UPDATE"
-- triggering events preclude its associated table from being listed (and
-- having a trigger generated).
--
--------------------------------------------------------------------------------

   CURSOR date_tables IS
      SELECT DISTINCT
         owner, table_name
      FROM
         sys.dba_tab_columns
      WHERE
         owner     = 'OPS$ACCESSLIMS'
```

```
         AND
         data_type = 'DATE'
   MINUS
      SELECT
         table_owner, table_name
      FROM
         sys.dba_triggers
      WHERE
         table_owner  = 'OPS$ACCESSLIMS'
         AND
         trigger_type = 'BEFORE EACH ROW'
         AND
         triggering_event LIKE '%UPDATE%INSERT%'
   MINUS
      SELECT
         table_owner, table_name
      FROM
         sys.dba_triggers
      WHERE
         table_owner  = 'OPS$ACCESSLIMS'
         AND
         trigger_type = 'BEFORE EACH ROW'
         AND
         triggering_event LIKE '%INSERT%UPDATE%'
      ORDER BY 1, 2;

--------------------------------------------------------------------------------
--
-- Gets a list of triggers associated with the specified table.
--
--------------------------------------------------------------------------------

   CURSOR trigger_for_table
      (
         table_with_date sys.dba_triggers.table_name%TYPE
      ) IS
      SELECT
         triggering_event
      FROM
         sys.dba_triggers
      WHERE
         table_name   = table_with_date
         AND
         table_owner  = 'OPS$ACCESSLIMS'
         AND
         owner        = table_owner
         AND
         trigger_type = 'BEFORE EACH ROW';

--------------------------------------------------------------------------------
--
-- List of dates for each table
--
--------------------------------------------------------------------------------

   CURSOR build_date_columns
      (
         table_date_columns sys.dba_tab_columns.table_name%TYPE
      ) IS
      SELECT
         column_name
      FROM
         sys.dba_tab_columns
      WHERE
         owner      = 'OPS$ACCESSLIMS'
         AND
         table_name = table_date_columns
         AND
         data_type  = 'DATE'
      ORDER BY column_name;
```

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--
-- Variables
--
--------------------------------------------------------------------------------
--
-- var_filtered_triggering_event
--     Triggering event information string. Receives a trigger's
--     triggering event(s) text. Then has everything removed except
--     "INSERT" and/or "UPDATE."
--
-- var_triggering_event_string
--     After being filtered, determines if can build an INSERT trigger or
--     an UPDATE trigger or cannot build a trigger.
--
--------------------------------------------------------------------------------

   var_filtered_triggering_event
          sys.dba_triggers.triggering_event%TYPE;

   var_triggering_event_string
          VARCHAR2(12);

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--
-- Trigger generation for each table with at least one date column
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

BEGIN

--------------------------------------------------------------------------------
--
-- LOOP once for each table to make a trigger if possible.
-- (Each table has at least one date column.)
--
--------------------------------------------------------------------------------

   FOR  trig_build
   IN   date_tables
   LOOP

--------------------------------------------------------------------------------
-----
----- Determine the kind of triggering event(s), if any, that can be generated
-----
--------------------------------------------------------------------------------
-----
----- What is left ("INSERT" and/or "UPDATE") after filtering, if anything,
----- will be the generated trigger's triggering event(s)
-----
      var_triggering_event_string := 'INSERTUPDATE';
-----
----- Determine the triggering events, if any, of existing triggers for the
----- specified table (trig_build.table_name)
-----
      FOR  triggering_event_check
      IN   trigger_for_table (trig_build.table_name)
      LOOP
--------
-------- To be left with "INSERT" or "UPDATE" or NULL, remove "OR's,"
-------- "DELETE," and spaces from the information string
--------
         var_filtered_triggering_event :=
            REPLACE ( triggering_event_check.triggering_event, 'OR' );
         var_filtered_triggering_event :=
```

```
            REPLACE ( var_filtered_triggering_event, 'DELETE' );
         var_filtered_triggering_event :=
            REPLACE ( var_filtered_triggering_event, ' ' );
--------
-------- To avoid error conflicts with already existing "UPDATE" or
-------- "INSERT" triggers, remove any triggering event from the
-------- triggering event string
--------
         IF var_filtered_triggering_event IS NOT NULL
         THEN
            var_triggering_event_string :=
               REPLACE
                  (
                  var_triggering_event_string       ,
                  var_filtered_triggering_event
                  );
         END IF;
      END LOOP;
-----
----- If the triggering event string is not null, then can build a trigger
-----
      IF var_triggering_event_string IS NOT NULL
      THEN

--------------------------------------------------------------------------------
--------
-------- Output trigger creation command, headers, comments, etc.
--------
--------------------------------------------------------------------------------
         DBMS_OUTPUT.PUT_LINE (
            'CREATE OR REPLACE TRIGGER ' || trig_build.owner || '.RR_'
            || LOWER( REPLACE( trig_build.table_name,'DWPF_') ) );
--------
-------- What ever is left in the triggering event string is the type(s)
-------- of triggering events the trigger will be
--------
         IF var_triggering_event_string = 'INSERTUPDATE'
         THEN
            DBMS_OUTPUT.PUT_LINE ( 'BEFORE INSERT OR UPDATE ' );
         ELSE
            DBMS_OUTPUT.PUT_LINE ( 'BEFORE ' || var_triggering_event_string );
         END IF;

         DBMS_OUTPUT.PUT_LINE (
            'ON                        ' || trig_build.owner || '.'
            || LOWER(trig_build.table_name) );
         DBMS_OUTPUT.PUT_LINE ( 'FOR EACH ROW' );
         DBMS_OUTPUT.PUT_LINE ( '/*' );
         DBMS_OUTPUT.PUT_LINE (
            '____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE ( '|' );
         DBMS_OUTPUT.PUT_LINE (
            '| Trigger     : RR_' || REPLACE( trig_build.table_name, 'DWPF_' ));
         DBMS_OUTPUT.PUT_LINE (
            '| Author      : Carl Cribbs' );
         DBMS_OUTPUT.PUT_LINE (
            '| Date Created: ' || to_char( SYSDATE, 'DD-MON-YYYY' ) );
         DBMS_OUTPUT.PUT_LINE (
            '| Description : For table ' || trig_build.table_name || '.' );
         DBMS_OUTPUT.PUT_LINE (
            '|               When an insert occurs on a record or an update,' );
         DBMS_OUTPUT.PUT_LINE (
            '|               occurs on the date column, converts the date ' );
         DBMS_OUTPUT.PUT_LINE (
            '|               to be Y2K compliant.' );
         DBMS_OUTPUT.PUT_LINE (
            '| Restrictions: Any year with it last two digits less than 50 is');
         DBMS_OUTPUT.PUT_LINE (
            '|               converted to a 21st century date. '
```

```
            || 'Any year with it' );
         DBMS_OUTPUT.PUT_LINE (
            '|                last two digits greater than or equal to 50 is' );
         DBMS_OUTPUT.PUT_LINE (
            '|                converted to a 20th century date.' );
         DBMS_OUTPUT.PUT_LINE ( '|' );
         DBMS_OUTPUT.PUT_LINE (
            '_____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE (
            '_____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE (
            '| Name   | Date        | Comment' );
         DBMS_OUTPUT.PUT_LINE (
            '_____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE (
            '|        |             |' );
         DBMS_OUTPUT.PUT_LINE (
            '|        |             |' );
         DBMS_OUTPUT.PUT_LINE (
            '|        |             |' );
         DBMS_OUTPUT.PUT_LINE (
            '_____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE ( '*/' );
         DBMS_OUTPUT.PUT_LINE ( 'BEGIN' );
         DBMS_OUTPUT.PUT_LINE ( '--' );
         DBMS_OUTPUT.PUT_LINE (
            '-- Convert date coming into database to be Y2K complaint.' );
         DBMS_OUTPUT.PUT_LINE ( '--' );
--------------------------------------------------------------------------------
--------
-------- LOOP for a specified table's date columns.
-------- Each date column has a Y2K compliant function call built for it.
--------
--------------------------------------------------------------------------------
         FOR   date_cols
         IN    build_date_columns ( trig_build.table_name )
         LOOP
            DBMS_OUTPUT.PUT_LINE
            (
               CHR(9) || 'IF :NEW.' || LOWER( date_cols.column_name )
               || ' IS NOT NULL '
            );
            DBMS_OUTPUT.PUT_LINE ( CHR(9) || 'THEN' );
            DBMS_OUTPUT.PUT_LINE
            (
               CHR(9) || '   :NEW.' || LOWER( date_cols.column_name )
                  || ' :='
            );
            DBMS_OUTPUT.PUT_LINE
            (
               CHR(9) || '      rr_date_conversion( :NEW.'
                  || LOWER( date_cols.column_name ) || ' );'
            );
            DBMS_OUTPUT.PUT_LINE ( CHR(9) || 'END IF;' );
            DBMS_OUTPUT.PUT_LINE ( CHR(9) );
         END LOOP;
--------------------------------------------------------------------------------
--------
-------- Trigger termination code
--------
--------------------------------------------------------------------------------
         DBMS_OUTPUT.PUT_LINE ( 'END;' );
         DBMS_OUTPUT.PUT_LINE ( '/*' );
         DBMS_OUTPUT.PUT_LINE (
            '_____________________________________________________________'
            || '_________________________' );
```

```
         DBMS_OUTPUT.PUT_LINE ( '*/' );
         DBMS_OUTPUT.PUT_LINE ( '/' );

      ROLLBACK;

      END IF;

   END LOOP;

END;
.
/
--
--
SPOOL OFF
--
-- Execute the generated code to create the triggers
--
@y2k_date_column_new_trig_code.sql
--
--
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--
-- Remove/drop Y2K protection triggers
-- (DROP TRIGGER commands save to y2k_drop_rr_triggers.sql
--
-------------------------------------------------------------------------------
--
SPOOL y2k_drop_rr_triggers.sql
--
--
SELECT DISTINCT
   'DROP TRIGGER ' || owner || '.' || trigger_name || ';'
FROM
   sys.dba_triggers
WHERE
   owner     = 'OPS$ACCESSLIMS'
   AND
   trigger_name LIKE 'RR_%'
ORDER BY
   'DROP TRIGGER ' || owner || '.' || trigger_name || ';';
--
--
SPOOL OFF
--
--
-------------------------------------------------------------------------------
--  F O R   D E B U G G I N G-
--  cannot create triggers generator file if table already has triggers
--
-- @y2k_drop_rr_triggers.sql
-------------------------------------------------------------------------------
--
--
SET ECHO On
SET FEEDBACK On
SET SERVEROUTPUT OFF
SET SHOWMODE On
SET TERM On
SET HEADING On
SET LONG 80
```

Attachment Number 3
Y2K_EXISTING_TRIGGER_UPGRADE.SQL

```
SET ECHO OFF
SET LINESIZE 132
SET SERVEROUTPUT On SIZE 1000000
SET FEEDBACK OFF
SET SHOWMODE OFF
SET TERM OFF
SET HEADING OFF
SET LONG 100000
SET ARRAYSIZE 10

--
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--
-- File         : Y2K_EXISTING_TRIGGER_UPGRADE.SQL
-- Purpose      : Each table's trigger will convert date data (for each
--                date field) to the year range of 1950 through 2049.
-- Description  : PL/SQL code generates each table's trigger.
-- Author       : Carl Cribbs
-- Date Created : 15-APR-1999
-- Restrictions : a. Only for existing "BEFORE" triggers that have tables
--                   with date fields.
--                b. The date conversion function that is called,
--                   "rr_date_conversion," converts:
--                   i.  All years with its last two digits less than 50 to
--                       a 21st century date.
--                   ii. All years with its last two digits greater than or
--                       equal to 50 to a 20th century date.
--                c. Since this was developed with PL/SQL Release 2.1.5.2.0 and
--                   SQL*Plus Release 3.1.3.5.1, each line is prefixed with
--                   a tab. (Since the original code and the Y2K conversion
--                   are made into two separate inner blocks and leading
--                   blanks are suppressed/trimmed, this maintains a
--                   semblance of the original indentation. The logic
--                   can be commented out or prefixed with spacing for
--                   later PL/SQL versions.)
--
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

--
--
spool y2k_trigger_upgrade_code.sql
--
--
DECLARE
--
-------------------------------------------------------------------------------
--
-- List of date columns for a specified table.
-- Builds the statement (for each date column) that calls the
-- date conversion function.
--
-------------------------------------------------------------------------------

   CURSOR build_date_columns
      (
         table_date_columns sys.dba_tab_columns.table_name%TYPE
      ) IS
      SELECT
         column_name
      FROM
         sys.dba_tab_columns
      WHERE
         owner      = 'OPS$ACCESSLIMS'
```

```
         AND
         table_name = table_date_columns
         AND
         data_type  = 'DATE';

-------------------------------------------------------------------------------
--
-- List of any columns a SQL" UPDATE" occurs for.
-- Used to rebuild the "UPDATE OF" clause.
--
-------------------------------------------------------------------------------

   CURSOR build_update_columns_string
      (
         table_update_columns    sys.dba_trigger_cols.table_name%TYPE,
         update_of_trigger_name  sys.dba_trigger_cols.trigger_name%TYPE
      ) IS
      SELECT
         column_name
      FROM
         sys.dba_trigger_cols
      WHERE
         table_name    = table_update_columns
         AND
         trigger_name  = update_of_trigger_name
         AND
         trigger_owner = 'OPS$ACCESSLIMS'
         AND
         table_owner   = trigger_owner
         AND
         column_list  = 'YES';

-------------------------------------------------------------------------------
--
-- List of triggers (for a specified table with at least one date column).
-- CURSOR ignores triggers with 'RR_' prefix since these were created
-- exclusively for converting the date fields.
--
-------------------------------------------------------------------------------

   CURSOR existing_tiggers
      (
         table_with_date    sys.dba_tab_columns.table_name%TYPE
      ) IS
      SELECT
         table_name,
         owner,
         trigger_name,
         trigger_type,
         triggering_event,
         table_owner,
         referencing_names,
         when_clause,
         status,
         description,
         trigger_body
      FROM
         sys.dba_triggers
      WHERE
         table_name        = table_with_date
         AND
         owner             = 'OPS$ACCESSLIMS'
         AND
         trigger_type      = 'BEFORE EACH ROW'
         AND
         trigger_name NOT LIKE 'RR_%'
       ORDER BY table_name;

-------------------------------------------------------------------------------
--
```

```
-- List of tables that have date column(s) .
--
-------------------------------------------------------------------------------

   CURSOR tables_with_dates IS
      SELECT DISTINCT
         table_name
      FROM
         sys.dba_tab_columns
      WHERE
         owner     = 'OPS$ACCESSLIMS'
         AND
         data_type = 'DATE';

-------------------------------------------------------------------------------
-----
----- Variables
-----
-------------------------------------------------------------------------------
-----
----- var_beginning_of_line
-----    The character number (within a trigger's body) of a line's
-----    first character (the beginning of the line). (This plus
-----    var_end_of_line are used to edit each trigger's line.)
-----
----- var_end_of_line
-----    The character number (within a trigger's body) of a line's
-----    last character (the end of line). (This plus var_beginning_of_line
-----    are used to edit each trigger's line.)
-----
----- var_length_trigger_string
-----    Number of characters in a trigger's body
-----
----- var_str_column
-----    Text string of columns that appeared after UPDATE FOR plus
-----    date columns.
-----
----- var_str_row
-----    Text string of a row of the trigger's original code.
-----
----- var_trigger_type
-----    Processes/edits the trigger type value.
-----
-------------------------------------------------------------------------------

       var_beginning_of_line       INTEGER;

       var_end_of_line             INTEGER;

       var_length_trigger_string   INTEGER;

       var_str_column              VARCHAR2(1000);

       var_str_row                 VARCHAR2(132);

       var_trigger_type            VARCHAR2(100);

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--
-- LOOP once for each table with a date
--
-------------------------------------------------------------------------------

BEGIN
   FOR table_loop
   IN tables_with_dates
   LOOP
-------------------------------------------------------------------------------
-----
```

```
----- LOOP once for each trigger to modify (add date protection).
-----
-------------------------------------------------------------------------------
      FOR modification_loop
      IN  existing_tiggers ( table_loop.table_name )
      LOOP

-------------------------------------------------------------------------------
--------
-------- Trigger command syntax
--------
-------------------------------------------------------------------------------
         DBMS_OUTPUT.PUT_LINE (
            'CREATE OR REPLACE TRIGGER '|| modification_loop.owner || '.'
             || modification_loop.trigger_name );
--------
-------- Since only "BEFORE" triggers types can modify "NEW" data values
-------- and want each row/record to have its date column affected,
-------- only modifying "BEFORE EACH ROW" triggers. (If a "BEFORE EACH ROW"
-------- trigger does not exist for "UPDATE" and/or "INSERT,"
-------- "Y2K_TRIGGER_GENERATOR.SQL" creates the Y2K conversion trigger.)
--------
         var_trigger_type := 'BEFORE';

-------------------------------------------------------------------------------
--------
-------- Reconstruct the triggering event (UPDATE and/or INSERT and/or DELETE)
--------
-------------------------------------------------------------------------------
         IF modification_loop.triggering_event IS NOT NULL
         THEN
-----------
----------- Combine trigger type and triggering event
-----------
            var_trigger_type := var_trigger_type || ' '
               || modification_loop.triggering_event;
----------- Initialize column list
            SELECT NULL INTO var_str_column FROM DUAL;
-----------
----------- For an UPDATE with column list ("UPDATE OF"), rebuild the list.
-----------
            FOR update_cols
            IN  build_update_columns_string
            (
               modification_loop.table_name,
               modification_loop.trigger_name
            )
            LOOP
               var_str_column :=
                  var_str_column || update_cols.column_name || ', ';
            END LOOP;
-------------------------------------------------------------------------------
-----------
----------- If any columns are in the column ("UPDATE OF") list,
----------- edit column list for proper syntax.
----------- If no columns in the column list, no action required.
-----------
-------------------------------------------------------------------------------
            IF var_str_column IS NOT NULL
            THEN
--------------
-------------- Remove trailing comma from non-null column list
--------------
               var_str_column := RTRIM ( var_str_column, ', ');
--------------
-------------- If have a column list, combine trigger type with
-------------- "UPDATE" changed to "UPDATE OF" and the column list.
--------------
               var_trigger_type := REPLACE ( var_trigger_type,
                  'UPDATE',  'UPDATE OF ' || var_str_column );
```

```
            END IF;
         END IF;

-------- Output triggering type concatenated with the triggering event (if any)
         DBMS_OUTPUT.PUT_LINE ( var_trigger_type );

--------------------------------------------------------------------------------
-------- End of reconstructing the triggering event
--------------------------------------------------------------------------------

-------- Output trigger's owner
         DBMS_OUTPUT.PUT_LINE (
            'ON ' || modification_loop.table_owner || '.'
            ||  modification_loop.table_name );

-------- Output any "REFERENCING" statement (e.g. "NEW AS NEW OLD AS OLD")
         IF modification_loop.referencing_names IS NOT NULL
         THEN
            DBMS_OUTPUT.PUT_LINE ( modification_loop.referencing_names );
         END IF;

-------- "FOR EACH ROW" part (only modifying "BEFORE EACH ROW" triggers)
         DBMS_OUTPUT.PUT_LINE (  'FOR EACH ROW' );

-------- If there is a "WHEN" clause, recreate it with the logic
-------- NOTE: No formatting of "WHEN" clause. (Displayed/output as retrieved
--------       in sys.dba_triggers.when_clause
         IF modification_loop.when_clause IS NOT NULL
         THEN
            DBMS_OUTPUT.PUT_LINE ( 'WHEN' );
            DBMS_OUTPUT.PUT_LINE ( '(' );
            DBMS_OUTPUT.PUT_LINE ( modification_loop.when_clause );
            DBMS_OUTPUT.PUT_LINE ( ')' );
         END IF;

--------------------------------------------------------------------------------
--------
-------- Change header and associated comments
--------
--------------------------------------------------------------------------------

         DBMS_OUTPUT.PUT_LINE ( '/*' );
         DBMS_OUTPUT.PUT_LINE (
            '_____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE ( '| Name   |  Date      |   Comment' );
         DBMS_OUTPUT.PUT_LINE (
            '_____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE (
            '| Cribbs | ' || TO_CHAR( SYSDATE, 'DD-MON-YYYY')
            || ' | Added Y2K protection to date fields' );
         DBMS_OUTPUT.PUT_LINE (
               '_____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE ( '*/' );

--------------------------------------------------------------------------------
--------
-------- BEGINNING OF TRIGGER'S CODE
--------
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------
-------- Beginning of outer block.
-------- Added to encompass new/added date conversion block and
-------- original code.
--------
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
         DBMS_OUTPUT.PUT_LINE ( '/*' );     -- Outer block comment text
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE (
            'Beginning of the new outer block to encompass the added Y2K' );
         DBMS_OUTPUT.PUT_LINE ( 'conversion code and the original code.' );
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE ( '*/' );
         DBMS_OUTPUT.PUT_LINE ( 'BEGIN' );  -- Beginning of outer block
         DBMS_OUTPUT.PUT_LINE ( '/*' );     -- Outer block comment text
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE ( 'Inserted Y2K conversion code.' );
         DBMS_OUTPUT.PUT_LINE (
            'Convert "NEW" date value to be Y2K complaint.' );
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE ( '*/' );

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--------
-------- BEGINNING OF THE Y2K CONVERSION BLOCK
--------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

         DBMS_OUTPUT.PUT_LINE ( CHR (9) || 'BEGIN' ); -- Beginning of
                                                      -- conversion block
         DBMS_OUTPUT.PUT_LINE ( CHR (9) );            -- Blank line spacing

-------------------------------------------------------------------------------
--------
-------- LOOP for a specified table's date columns.
-------- Each date column has a Y2K compliant function call built for it.
--------
-------------------------------------------------------------------------------

         FOR date_cols IN build_date_columns ( modification_loop.table_name )
         LOOP
            DBMS_OUTPUT.PUT_LINE
               (
               CHR (9) || '   IF :NEW.' || LOWER( date_cols.column_name )
               || ' IS NOT NULL '
               );
            DBMS_OUTPUT.PUT_LINE ( CHR (9) || '   THEN' );
            DBMS_OUTPUT.PUT_LINE
               (
               CHR (9) || '      :NEW.' || LOWER( date_cols.column_name )
               || ' := ' || 'rr_date_conversion( :NEW.'
               || LOWER( date_cols.column_name ) || ' );'
```

```
               );
            DBMS_OUTPUT.PUT_LINE ( CHR (9) || '   END IF;' );
            DBMS_OUTPUT.PUT_LINE ( '--' );
         END LOOP;

--------
-------- Termination of the Y2K conversion block
--------
         DBMS_OUTPUT.PUT_LINE ( CHR (9) || 'END;' );
         DBMS_OUTPUT.PUT_LINE ( '/*' );
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '__________________________' );
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '__________________________' );
         DBMS_OUTPUT.PUT_LINE ( 'Beginning of original code.' );
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '__________________________' );
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '__________________________' );
         DBMS_OUTPUT.PUT_LINE ( '*/' );

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--------
-------- ORIGINAL TRIGGER CODE
--------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

         var_beginning_of_line   := 1;
         var_end_of_line         := 0;
         var_length_trigger_string := LENGTH ( modification_loop.trigger_body );

         WHILE var_end_of_line  < var_length_trigger_string
         LOOP
-----------
----------- Find the next end of line (each line terminated with a line feed)
-----------
            var_end_of_line :=  INSTR (
                                       modification_loop.trigger_body,
                                       CHR( 10 ),
                                       var_beginning_of_line
                                       );
-----------
----------- Extract the line of existing trigger code
-----------
            var_str_row := SUBSTR (
                                  modification_loop.trigger_body,
                                  var_beginning_of_line,
                                  var_end_of_line - var_beginning_of_line
                                  );

-------------------------------------------------------------------------------
-----------
----------- SINCE THIS WAS DEVELOPED WITH PL/SQL Release 2.1.5.2.0 and
----------- SQL*Plus: Release 3.1.3.5.1, THE BELOW LOGIC IS REQUIRED
-----------
-------------------------------------------------------------------------------
-----------
----------- Due to leading blanks being suppressed/trimmed, prefix each line
----------- with a tab- "CHR(9)." (This is to indent inner blocks and to
----------- keep from totally losing the existing indentations.
-----------
----------- Id desirable, can replace "CHR(9)" with spacing or remove totally.
-----------
-------------------------------------------------------------------------------
```

```
----------- Do not tab left justified comments
            IF SUBSTR( var_str_row, 1, 1 ) != '/'
               AND
               SUBSTR( var_str_row, 1, 1 ) != '*'
               AND
               SUBSTR( var_str_row, 1, 1 ) !=  '-'
            THEN
               var_str_row := CHR(9) || var_str_row;
            END IF;

-------------------------------------------------------------------------------
-----------
----------- END OF REQUIRED LOGIC FOR PL/SQL Release 2.1.5.2.0 and
----------- SQL*Plus: Release 3.1.3.5.1
-----------
-------------------------------------------------------------------------------

-----------
----------- Output trigger code and reset or update variables
-----------
            DBMS_OUTPUT.PUT_LINE ( var_str_row );

            SELECT NULL INTO var_str_row FROM DUAL;

            var_beginning_of_line := var_end_of_line + 1;

         END LOOP;

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--------
-------- TERMINATION OF THE NEW/ADDED OUTER BLOCK
--------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
         DBMS_OUTPUT.PUT_LINE ( '/*' );
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE (
            'End of the new outer block to encompass the inserted Y2K' );
         DBMS_OUTPUT.PUT_LINE ( 'date conversion code and the original code.' );
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE (
               '____________________________________________________________'
            || '_________________________' );
         DBMS_OUTPUT.PUT_LINE ( '*/' );
         DBMS_OUTPUT.PUT_LINE ( 'END;' );
         DBMS_OUTPUT.PUT_LINE ( '/' );     -- Trigger's terminating slash
      END LOOP;

   END LOOP;

   ROLLBACK;

END;
.
/
--
--
SPOOL OFF
--
SET ARRAYSIZE 15
SET LONG 80
```

```
SET SERVEROUTPUT OFF
SET FEEDBACK On
SET SHOWMODE On
SET TERM On
SET HEADING On
SET ECHO On
```