Title: Representing Group/Members Using a Temp-Table Approach

Author: Hansdip Singh Bindra

Business Application programming using Relational Data Structures in all environments – GUI, Web and CHUI, almost always involve the representation of Groups and its Members. These Group/Member relationships could be a Group of Customers by Sales Region, the Users in a particular Security Group, the States that belong to a Country, etc. There are various approaches to presenting this association between Groups and its Members, depending on the nature of the application, the programming and UI environment, and other user requirements.

One of the most common presentation methods, used under GUI, Web and CHUI environments is known as the “Left-to-Right” method. Under this presentation methodology, the user is presented with a list of all potential Members for a Group on the “Left” and the current selected Members in the Group on the “Right”. This article provides an easy to use Temp-Table approach to the “Left-to-Right” method, whereby the developer can simply replace the custom fields in the Temp-Table (in the example code) to satisfy the needs of his/her application.

In order to further explain the Temp-Table approach to Group/Member representation using the “Left-to-Right” method, we use a Customer Groups example, in a GUI client-server environment. The business requirement is that Customers can belong to one or more Customer Groups. These Customer Groups could be for Sales, Reporting, Security, or any other purposes. The example code is based on the Customer Table in the Sports Database supplied by Progress with the addition of two new Tables – CustomerGroup and CustomerGroupMember. The schema for the new Tables is provided.

First, we use the Customer Group Maintenance to setup Customer Groups. Next we assign Members to these Groups using Customer Group Member Maintenance. This application uses the “Left-to-Right” method using a Temp-Table. The list of all Customers is presented on the “Left”, whereas the members in a particular Group are on the “Right”. The user can choose to add/remove Members from the Group by clicking on the right/left arrow Buttons in the middle of the Window. Once the user has made all the updates to the list of Members in the Group on the Window, he/she clicks on the Save Button to commit the update to the Database.
Representing Group/Members Using a Temp-Table Approach.
Published in *Progressions* – Winter 2000, Number 40.

“Left-to-Right” Group/Member Window

The two lists are presented using Browse widgets. The “Left” Browse is tied to the Customer Table, whereas the “Right” browse is tied to the Temp-Table – TempMember. The TempMember Temp-Table is similar to the CustomerGroupMember Table which stores the Customers associated with a particular Customer Group. Whenever the user selects a Group on this Window, the “Right” list is built from the TempMember Temp-Table which in turn is built from the CustomerGroupMember Table for that CustomerGroup.

The TempMember Temp-Table has two mandatory (or non-customizable Fields) – Removed and MemberOfGroup, and two mandatory Indexes associated with these Fields with the same names as the Fields. The rest of the Fields in the Temp-Table is specific to the application being developed. The usage of these two mandatory Fields and the Indexes provide a simplified framework for presenting the Group/Member using the “Left-to-Right”, and other extensions of this method. In this example the TempMember Temp-Table has Customer specific information, such as the Cust-Num Field. If one were dealing with States in a Country, this would be replaced with the unique State Code.

Having described the data structure and the business use of this example, it would be appropriate to explain the main code constructs behind this approach. The MemberOfGroup mandatory Logical Field indicates whether the Customer is an existing Member of the Group (YES) or not (NO), and the Removed mandatory Logical Field indicates whether the existing Member was removed (YES) or not (NO) from the Group during the current user session on the Window.

When the user selects a Customer Group, the Members of that Group are filled into the TempMember Temp-Table with the MemberOfGroup set to YES and Removed set to NO, as follows:

```plaintext
PROCEDURE TempMemberFillProc :
/*------------------------------------------------------------------------------
Purpose: Fill TempMember from CustomerGroupMember on selecting a Customer Group
Parameters:  <none>
Notes:
--------------------------------------------------------------------------*/
```

© 1999-2005 – Innov8 Computer Solutions, LLC.
Representing Group/Members Using a Temp-Table Approach.
Published in Progressions – Winter 2000, Number 40.

FOR EACH CustomerGroupMember WHERE CustomerGroupMember.CustomerGroup = CustomerGroup.CustomerGroup
NO-LOCK USE-INDEX CustomerGroup:

FIND FIRST bufferCustomer WHERE bufferCustomer.Cust-Num = CustomerGroupMember.Cust-Num
USE-INDEX Cust-Num
NO-LOCK NO-ERROR.

CREATE bufferTempMember.

BUFFER-COPY CustomerGroupMember
EXCEPT CustomerGroupMember.CustomerGroup
TO bufferTempMember.

ASSIGN bufferTempMember.Name = IF AVAILABLE bufferCustomer THEN bufferCustomer.Name ELSE "".

/** ADD EXISTING MEMBER – DO NOT MODIFY **/
ASSIGN bufferTempMember.Removed = NO
/**
***/

RELEASE bufferTempMember.
END.
END PROCEDURE.

Now, the “Right” list contains all the existing Members of the Group. If the user decides to add/remove Members in the Update Mode on the Window, using the right (add) or the left (remove) arrow Buttons, or double-clicking on the appropriate lists, the Temp-Table is manipulated accordingly. If the user decides to add a Customer to the Group, a new record is Created in the Temp-Table with the MemberOfGroup set to NO (as it not an existing Member), and Removed set to NO. This is visually represented in the example with the Row in the “Right” list having a yellow background color. Existing Members Rows have a white background color. If the user has removes an existing Member and then re-adds it to the “Right” list, we set the Removed Field from YES to NO. Appropriate validations (if any) should be done before adding the Member to the Group. In this example, we do not allow a Customer to be represented more than once in the Group. Please refer to ItemAddProc Internal Procedure in CustomerGroupMemberMaintenance.w, for the code that adds a Row (Member) to TempMember Temp-Table.

The removal of a Customer from the “Right” list (TempMember) involved two scenarios. The first is the removal of a Customer (Member) who was added in the present session (a Customer that does not currently exist in the Group). This is simple, as all we do is Delete the Row from the Temp-Table. The second scenario involves the removal of an existing Customer (Member). In this case we set the Removed Field to YES and refresh the “Right” list, which only shows TempMember Rows with Removed set to NO. Thus, the removed row is visually removed from the Browse widget. The code for removing a Row (Member) from TempMember Temp-Table is as follows:

PROCEDURE ItemRemoveProc :
ستراتيجي
Purpose: Remove Row (Member) from TempMember
Parameters: inMemberOfGroup – TempMember.MemberOfGroup
Notes:
******************************************************************************
DEFINE INPUT PARAMETER inMemberOfGroup AS LOGICAL NO-UNDO.
******************************************************************************

© 1999-2005 – Innov8 Computer Solutions, LLC.
DO WITH FRAME {&FRAME-NAME}:
IF inMemberOfGroup = YES THEN
  DO:
    /** Existing Member **/
    IF SESSION:SET-WAIT-STATE("GENERAL":U) THEN.
      DO TRANSACTION:
        FIND CURRENT TempMember EXCLUSIVE-LOCK NO-ERROR.
        IF AVAILABLE TempMember THEN
          DO:
            ASSIGN TempMember.Removed = YES.
          END.
        END. /* TRANSACTION */
    END. /* TRANSACTION */
    IF SESSION:SET-WAIT-STATE(""") THEN.
      RUN TempMemberQueryProc(INPUT ?).
    END.
  ELSE
    DO:
      /** Member added during present session **/
      DO TRANSACTION:
        FIND CURRENT TempMember EXCLUSIVE-LOCK NO-ERROR.
        DELETE TempMember.
      END. /* TRANSACTION */
      IF browseTempMember:DELETE-CURRENT-ROW() THEN.
      END.
  END.
END PROCEDURE.

Once the user has decided on the Members for the Group, he/she clicks on the Save Button to commit the updated “Right” list to the Database. At this stage, any further validations (if needed) should be done on the Members in the “Right” list. If the validations are successful, the application should do the appropriate locking on the Tables and commit the Members in the TempMember Temp-Table to the CustomerGroupMember Table. The core of the commit process is based on the two mandatory Fields in the Temp-Table.

<table>
<thead>
<tr>
<th>ACTION</th>
<th>MemberOfGroup</th>
<th>Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW Member</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>EXISTING (Updated) Member</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>DELETED Member</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

The section in UpdateProc Internal Procedure in CustomerGroupMemberMaintenance.w that handles this commit process is as follows:

```*/ ** NEW MEMBER - DO NOT MODIFY ***/
FOR EACH bufferTempMember WHERE
  bufferTempMember.MemberOfGroup = NO
  USE-INDEX MemberOfGroup NO-LOCK:
    ASSIGN numTotalUpdateMember = numTotalUpdateMember + 1.
    RUN GroupMemberCreateProc.
END.

```*/ ** EXISTING MEMBER - DO NOT MODIFY ***/
FOR EACH bufferTempMember WHERE
  bufferTempMember.MemberOfGroup = YES AND
  bufferTempMember.Removed = NO
```
Representing Group/Members Using a Temp-Table Approach.
Published in *Progressions* – Winter 2000, Number 40.

```plaintext
USE-INDEX MemberOfGroup NO-LOCK:
    ASSIGN numTotalUpdateMember = numTotalUpdateMember + 1.
    RUN GroupMemberUpdateProc(INPUT "Update":U).
END.

/** DELETE MEMBER - DO NOT MODIFY **/
FOR EACH bufferTempMember WHERE
    bufferTempMember.MemberOfGroup = YES AND
    bufferTempMember.Removed = YES
USE-INDEX MemberOfGroup NO-LOCK:
    ASSIGN numTotalDeleteMember = numTotalDeleteMember + 1.
    RUN GroupMemberUpdateProc(INPUT "Delete":U).
END.
```

These commit loops are scoped to the main lock on the Customer Table and depending on the action, the referential integrity checks are done against the Database. The Existing Member Action, which is mute in this example, is shown because in other more complicated applications could be used to Update other Fields in the Member Table, for example the State Name for a State associated with a Country. This Update Action should check whether the existing Row in the CustomerGroupMember Table has been updated by another user, and if so should not be updated through this commit process. The `numTotalUpdateMember` (the total number of Members to update/commit from TempMember) and `numUpdateMember` (the total number of Members that are updated/committed from TempMember) counters keep track of such referential integrity checks. Similar counters are provided for the New and Delete Actions, and could be examined in `GroupMemberCreateProc` and `GroupMemberUpdateProc` Internal Procedures in `CustomerGroupMemberMaintenance.w`.

As stated in the beginning of the article, the Temp-Table approach with the two mandatory Fields provides an efficient and easy implementation to the Group/Member presentation using the “Left-to-Right” (and other similar) methodology. The code supplied with this article is extremely easy to read, as I have made the Widget names self-explanatory and provided comments in the programs. Please use the “ini” file supplied with the code example in order to get the correct “look and feel” for the Group/Member applications.