APEX_ITEM and Beyond

Greg Jarmiowlowski
SQLPrompt LLC
Agenda

- Built-in Tabular Form Limitations
- Tabular Form Post Processing Built-ins
- Building Forms with APEX_ITEM
- Global Arrays
- Custom Post processing
- APEX_COLLECTIONS
- Validation
- Locking and Concurrency
- AJAX with Tabular Forms
What Are Tabular Forms

- Forms for editing multiple records at once
- Attribute pairs/many to many list management
- Questions and answers
- The detail for a master detail relationship
Survey Roles

<table>
<thead>
<tr>
<th></th>
<th>Survey Role Name</th>
<th>Created On</th>
<th>Created By</th>
<th>Updated On</th>
<th>Updated By</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EVERYONE</td>
<td>06-OCT-08</td>
<td>HR_USER</td>
<td>(null)</td>
<td>(null)</td>
</tr>
<tr>
<td>2</td>
<td>SUPERVISOR</td>
<td>06-OCT-08</td>
<td>HR_USER</td>
<td>08-OCT-08</td>
<td>GREG</td>
</tr>
</tbody>
</table>

1 row(s) updated, 0 row(s) inserted.
How To Create A Tabular Form

• Wizard
• “Tabular Form Element” attributes of the report columns
  SQL Query (updatable report)
• SQL report using APEX_ITEM
• PL/SQL region using APEX_ITEM
Built-in Tabular Form Limitations

• Reasons for using the APEX_ITEM API
• Item types
  – Checkboxes
  – Radio Buttons
  – Popup Key
  – Very large list(s) of values
• Runtime item type determination
• Multiple tabular forms per page
APEX_ITEM Package

- Functions that return HTML input items
- Procedure for updating multiple records
- Checkbox
- Checksum
- Date Popup
- Display only
- Hidden
- Popup LOV
- Radiogroup
- Select List
- Text
- Textarea
How to Use APEX_ITEM

SELECT APEX_ITEM.DATE_POPUP(2,
    , NULL
    , date_entered
    , 'MM/DD/YYYY')

...
Post Processing

- **APEX_ITEM.MULTI_ROW_UPDATE**
- Takes an "MRU" string in the format:
  OWNER:TABLE:pk_col1,pk_idx:pk_col2,p_idx2|col,idx:col:idx..
  - Col is column name
  - Idx is the number used in APEX_ITEM
- Can manipulate the posted values before processing
THE REPORT SQL
SELECT apex_item.hidden(1, emp_id) emp_id,
      apex_item.text(2, emp_fname) emp_fname,
      apex_item.text(3, emp_lname) emp_lname
FROM emp

THE SUBMIT PROCESS
BEGIN
  APEX_ITEM.MULTI_ROW_UPDATE(
    'SCOTT:EMP:EMP_ID,1|EMP_FNAME,2:EMP_LNAME,3');
END;

• Simple
• All or nothing
APEX Global Collections

- Each form element has an HTML name attribute
- Posted values are added to Nested Tables
- Called Global Arrays in some docs
  
  ```sql
  type vc_arr2 is table of varchar2(32767)
  index by binary_integer;
  ```
- “Arrays” are in package apex_application
- Named g_f01 to g_f50 plus g_fcs
- Correspond to g_ and the name attribute of the form element
- Helper utility prints matrix of posted arrays
  
  [http://jornica.blogspot.com/2008/02/apexglobalarrays.html](http://jornica.blogspot.com/2008/02/apexglobalarrays.html)
SELECT apex_item.hidden(50, ROWNUM) AS row_number
, apex_item.hidden(49, pk_column) AS pk
, apex_item.radiogroup(ROWNUM, radio_value, selected_value, value_label) as radiobutton
FROM wherever
WHERE whatever
BEGIN

FOR i IN 1 .. apex_application.g_f01.COUNT LOOP
    -- shorten the submitted first name to 20 characters
    apex_application.g_f02(i) :=
        SUBSTR(apex_application.g_f02(i),1,20);

    -- shorten the submitted last name to 20 characters
    apex_application.g_f03(i) :=
        SUBSTR(apex_application.g_f03(i),1,20);

END LOOP;

APEX_ITEM.MULTI_ROW_UPDATE(
    'SCOTT:EMP:EMP_ID,1|EMP_FNAME,2:EMP_LNAME,3');

END;
Global Collection Basics

<table>
<thead>
<tr>
<th>Idx</th>
<th>g_f01</th>
<th>g_f02</th>
<th>g_f03</th>
<th>g_f50</th>
<th>g_fcs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13432</td>
<td>4</td>
<td>1</td>
<td></td>
<td>..B2449..</td>
</tr>
<tr>
<td>2</td>
<td>14567</td>
<td>1</td>
<td></td>
<td></td>
<td>..A3609..</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>..Z8134..</td>
</tr>
</tbody>
</table>

- Tabular form elements are posted to server and then stored in the apex_application arrays
- The name attribute of the item corresponds with the array name
- G_fcs is populated by md5_checksum function
- Non selected checkboxes and radio buttons (and disabled elements) are *not* posted
Radio Button and Checkbox Issues

• List type form elements may be single or multiselect
• Select lists from APEX_ITEM are only single select
• Multi-select items are commonly stored as children of the main record
• APEX makes it simple to use multi-selects stored as colon delimited strings
### Radio Button and Checkbox Issues

<table>
<thead>
<tr>
<th>FIRST_NAME</th>
<th>LAST_NAME</th>
<th>ID</th>
<th>RED</th>
<th>GREEN</th>
<th>BLUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tester</td>
<td>One</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tester</td>
<td>Two</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Tester</td>
<td>Three</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Unselected form elements are not posted to the server

Global array positions (idx) will not match other elements in the row
Checkbox Considerations

• Checkboxes and radiobuttons present unique challenges
• If checkbox values are consistent across all rows you can use the row key for the value
• Otherwise you need the value and a way to associate it with the row key
Radiogroup Considerations

- Radio buttons present even more challenges
- HTML radio input types are grouped by NAME
- Only one radio button per group can be selected
Matching Keys and Values

Three main methods
• Store the PK in the value attribute of the element and associate the p_idx with a database value
• Store the PK and the database value in the value attribute and use any p_idx (Compound Key)
• Store the database value in the value attribute using a rownum derived p_idx and then store that rownum in a hidden item (Triangulation)
## Matching Keys and Values

<table>
<thead>
<tr>
<th>Element</th>
<th>ID</th>
<th>Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ID</td>
<td>Name</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
</tbody>
</table>
| Element       | Hidden | Display Only | Radio | Radio | Radio |}
| P_idx ->      | 1 | 2 | 3 | 4 | 5 |}
| Value         | Primary Key | Database Value | Primary Key | Primary Key | Primary Key |}

Store PK in value of element and use the p_idx to get the value for database
Matching Keys and Values

--delete using pk
FORALL i IN 1 .. g_f01.COUNT
  DELETE mytable
    WHERE id = apex_application.g_f01(i);

--insert using posted values
FORALL red IN 1 .. g_f03.COUNT
  INSERT INTO mytable(ID, color)
    VALUES (apex_application.g_f03(red)
          , 'RED');
## Compound Keys

<table>
<thead>
<tr>
<th>Element</th>
<th>ID</th>
<th>Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>Hidden</td>
<td>Display Only</td>
<td>Radio</td>
<td>Radio</td>
<td>Radio</td>
</tr>
<tr>
<td>P_idx -&gt;</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Value</td>
<td>Primary Key (PK)</td>
<td>Database Value</td>
<td>PK</td>
<td>RED</td>
<td>PK</td>
</tr>
</tbody>
</table>

Store PK and DB value in value attribute of element
DECLARE
    l_key_arr     wwv_flow_global.vc_arr2;
BEGIN
    FORALL i IN 1 .. apex_application.g_f01.COUNT
        DELETE mytable
        WHERE ID = apex_application.g_f01(i);

    FOR x IN 1 .. apex_application.g_f03.COUNT
        LOOP
            l_key_arr :=
                apex_util.string_to_table
                (apex_application.g_f03(x), '|');

            INSERT INTO mytable(ID, color)
                VALUES (l_key_arr(1), l_key_arr(2));
        END LOOP;
END;
Triangulation Keys

- Simple to create key structure
- Flattens the checkbox or radio button arrays to one array each

```sql
SELECT apex_item.hidden(50, ROWNUM) AS row_number,
     apex_item.hidden(49, pk_column) AS pk,
     apex_item.radiogroup(ROWNUM, radio_value, selected_value, value_label) as radiobutton
FROM wherever
WHERE whatever
```
### “Triangulation” Keys

<table>
<thead>
<tr>
<th>idx</th>
<th>g_f01</th>
<th>g_f02</th>
<th>g_f03</th>
<th>g_f04</th>
<th>g_f05</th>
<th>g_f06</th>
<th>g_f07</th>
<th>g_f08</th>
<th>g_f09</th>
<th>g_f10</th>
<th>g_f11</th>
<th>g_f12</th>
<th>g_f13</th>
<th>g_f14</th>
<th>g_f15</th>
<th>g_f49</th>
<th>g_f50</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>96</td>
<td>02</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>97</td>
<td>03</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>98</td>
<td>04</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99</td>
<td>05</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>06</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>101</td>
<td>07</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>102</td>
<td>08</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>103</td>
<td>09</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>104</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>105</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>106</td>
<td>12</td>
</tr>
</tbody>
</table>

- `g_f49` is primary key
- `g_f50` points to the array with the value
- All item values are in index 1 of arrays 1 to 48 (except for checkboxes and multiselects)
DECLARE
    l_arrnum         VARCHAR2(50);
    l_stmt           VARCHAR2(200);
    l_arrval         VARCHAR2(100);
BEGIN
    FOR i IN 1 .. apex_application.g_f49.COUNT
    LOOP
        -- build string to find the array number stored in array 50
        l_arrnum := 'apex_application.g_f' || LPAD(LTRIM(RTRIM(apex_application.g_f50(i))),2, '0');

        -- build pl/sql block to extract value(s) as delimited string
        l_stmt := 'BEGIN ' || ' :l_arrval := apex_util.table_to_string(' || l_arrnum || ', '':''); END;';
        -- run block using out bind value
        EXECUTE IMMEDIATE l_stmt USING OUT l_arrval;
        save_response(:g_resp_id, apex_application.g_f49(i), l_arrval);
    END LOOP;
END;
APEX_ITEM.CHECKBOX

Arguments:
p_idx = The form element name, e.g. 1 equals f01, 2 equals f02, etc.

Typically the p_idx argument is constant for a given column.
p_value = When checked return this value
p_attributes = Custom HTML arguments added to the HTML input type of checkbox.
p_checked_values = Colon (by default delimited list of values
p_checked_values_delimiter = Defaults to a colon "::" and is used to pass multiple values in one string.
p_item_id = Will set the ID of the item to this value (id="..."). Must be unique! Try concatenating some string with rownum. Required for 508 compliance
p_item_label = Creates an invisible label for an item. Used for Section 508 Compliance. Class is hideMe508.
Multiple Checkbox Function

- Most examples of checkboxes show one “idx” per column
- Not easy to do this in SQL with a varying number of columns (case range $\geq x$)
- `SELECT_LIST_FROM_LOV` but no `CHECKBOX_FROM_QUERY` or similar
Multiple Checkbox Function

SELECT multi_checkbox_from_list(10, 'Green;1,Red;2, Blue;3', '1:2') FROM DUAL

☑️ Green ☑️ Red ☐️ Blue

SELECT multi_checkbox_from_list(10, 'Green;1,Red;2, Blue;3', '2') FROM DUAL

☐ Green ☑️ Red ☐️ Blue
CREATE OR REPLACE FUNCTION multi_checkbox_from_list(
    p_idx              IN   NUMBER,
    p_value_list       IN   VARCHAR2,
    p_checked_values   IN   VARCHAR2
) RETURN VARCHAR2 IS
    l_list_arr    wwv_flow_global.vc_arr2;
    l_list_item   wwv_flow_global.vc_arr2;
    l_return      VARCHAR2(32000); BEGIN
    l_list_arr    := apex_util.string_to_table(p_value_list, ',');

    FOR i IN 1 .. l_list_arr.COUNT loop
        l_list_item   := apex_util.string_to_table(l_list_arr(i), ';');
        l_return      := l_return || '<label>' || apex_item.checkbox(p_idx => p_idx,
                                                                   p_value => l_list_item(2),
                                                                   p_checked_values => p_checked_values);
        l_return      := l_return || l_list_item(1) || '</label>';  
    END LOOP;

    RETURN l_return;
END multi_checkbox_from_list;
Some Limitations

When working with tabular forms

• Submitted data is not stored in session state

• No easy way to validate

• MRU Checksum Error
  – If you use the built in processing
APEX Collections To The Rescue

- APEX collections are a view and an API to work with the data in the underlying tables.
- Each collection instance belongs to and can only be viewed by the owning session.
- Has a name, row index, 50 varchar columns, one LOB column and a checksum column.
Not All Collections Are The Same

- APEX_COLLECTION
data stored in tables with an API to read and write
- APEX Global Collections
  VARCHAR array package variables
  PL/SQL types – not SQL types
How To Create And Populate

- `create_collection` (delete_collection)
- `create_or_truncate_collection`
- `add_member`
- `update_member`
- `create_collection_from_query`
- `create_collection_from_query_b`
Create Collection From Query

- Cannot pass query into function with :BIND syntax
- Convert :BIND to `v('BIND')`
  ```sql
  REGEXP_REPLACE(sql, ':[a-zA-Z0-9_]*', 'v(\'1\') ')'
  ```
- `p_generate_md5` parameter creates checksum of columns in SQL
- `create_collection_from_query_b` does not offer checksum
Querying Your Collection

• Works just like a regular table or view
• Multiple collections can be joined or unioned

```
SELECT c001, c002, ..., c050
FROM apex_collections
WHERE collection_name = :mycoll
```
APEX Collections Checksum

- API for collections can compute checksums for stored columns
- When creating the members from a query the checksum process locks the row to read it
- The checksum can be used for lost update detection and to prevent tampering with read only elements
Post Processing With Collections

- **CREATE_OR_TRUNCATE_COLLECTION**

- Loop over values from APEX_APPLICATION collections and use ADD_MEMBER
  or
  - Dump all your APEX_APPLICATION collections into it with ADD_MEMBERS

- Generating a checksum here allows you to determine what records changed
SELECT apex_item.hidden(50, ROWNUM) AS row_number
, apex_item.hidden(49, pk_column) AS pk
, apex_item.radiogroup(ROWNUM, radio_value, selected_value, value_label) as radiobutton
FROM wwv_flow_collections
WHERE collection_name = :l_collection
Validation Errors

- Report query outer joined to collection
- Submit process to CREATE_OR_TRUNCATE_COLLECTION
- Submit process finds validation errors and stores information in collection
- Submit process to update data for records without validation error
- Branch to same page and report query will access any error messages in collection (or just limit it to those with errors if errors are present)
Taking it a step further

- Extract core report query to On Load process that creates collection from the query
- Report query selects from the collection instead of your tables
- On Submit process saves submitted values in another collection
- Joining collections by key and comparing MD5_ORIGINAL column shows “dirty” records
- Take this approach as far as you want – add error messages; decide what and when to process; etc
Combining AJAX and Tabular Forms

• Basic AJAX functionality requires three components
  1. Trigger for form element
  2. Javascript function to make AJAX call
  3. On Demand Process in APEX
Combining AJAX and Tabular Forms

- Trigger for form element options
- Attribute of the element

```html
<input type="text"
onchange="SetPrioritySeq(391120, this.value )"
value="" name="f07"/>
```

- Register event on page load
Combining AJAX and Tabular Forms

Registering event handlers on page load

```javascript
if (window.addEventListener) {
    window.addEventListener("load", s_registerClicks, true);
} else if (window.attachEvent) {
    window.attachEvent("onload", s_registerClicks);
}

function s_registerClicks() {
    var answerArr = new Array();
    answerArr = document.getElementsByName('f03');
    for (var i = 0; i < answerArr.length; i++) {
        answerArr[i].onchange = function() {
            return s_changedElement(this);
        }
    }
}
```

`s_changedElement` is the actual handler of the change
function SetPrioritySeq(prec, pval)
{
    var get = new htmldb_Get(null,$v('pFlowId'),'APPLICATION_PROCESS=UpdatePriSeq',0);
    get.addParam('x01',prec); // the primary key
    get.addParam('x02',pval);  // the update value
    gReturn = get.get();
    get = null;
    gReturn = (!gReturn)?'null':gReturn;
}
DECLARE
  l_grid          NUMBER;
  l_priority_seq  grievance.arb_priority_seq%TYPE;
BEGIN
  l_grid            := wwv_flow.g_x01;   -- the primary key
  l_priority_seq   := wwv_flow.g_x02;   -- the value to update
  UPDATE grievance
  SET arb_priority_seq = l_priority_seq
  WHERE grid = l_grid;
END;
Useful Helper Functions

• Find the nth element in delimited array
  \texttt{REGEXP\_SUBSTR (:PX,'[^:]+' ,1,n)}

• Count elements in delimited array
  \texttt{REGEXP\_COUNT (:PX,'[^:]+' )}
Useful Helper Functions

When using checkboxes for list management instead of deleting and reinserting you can find the intersect

```sql
-- old minus new equals those we need to delete
l_delete_arr := l_old_arr MULTISET EXCEPT l_new_arr;
-- new minus old equals those we need to insert
l_insert_arr := l_new_arr MULTISET EXCEPT l_old_arr;
```

* Works only on table types declared in SQL