Tabular Forms

This chapter will cover the Tabular Forms feature of APEX. Tabular Forms are one of the most important features of APEX. They were there in the very first version of the tool. However, they remained almost unchanged until the version 4.0. A Tabular Form will provide us with a way to display, create, edit and delete multiple records using a grid. The main goal is to edit and change multiple records at once, without having to go back and forth like in a normal single record form.

# Introduction

How did Tabular Forms evolve? As we already mentioned, this feature was a part of APEX (previously called HTML\_DB) from the very beginning. Until version 4.0 there were no (major) changes in the way Tabular Forms operate. There was a wizard to lead us through creation of a Tabular Form. This wizard would create a simple editable report for us. Beside the report, it would create couple of buttons required for saving or discarding of our changes and four processes for create, update and delete of our data. That was basically it. If we needed more, we had to create our own code.

# APEX 4.0 and Tabular Forms

With APEX 4.0 we got the first major change in Tabular Forms:

* New item types (single checkbox, jQuery date picker, radio group, popup key LOV)
* Client side “Add Row” capability
* Validations for Tabular Form columns
* Lost update protection – finally it is possible to validate a tabular form and show the error message on the same screen without losing the updates.
* Reduced number of processes required for a Tabular Form (two instead of four)
* Some other features we will mention later in this chapter

# Constraints

APEX and it’s wwv\_flows package provides 50 predifined PL/SQL arrays for Tabular Form operations. We can reference these arrays using the following syntax:

wwv\_flow.g\_f01 … wwv\_flow.g\_f50 or

apex\_application.g\_f01 … apex\_application.g\_f50

Every updatable column in our Tabular Form will have a unique ID in a sequential order (our SQL statement) mapped to one of these arrays:

f01\_0001 … f01\_n

This is also a major constraint if we talk about Tabular Forms. A Tabular Form will currently allow us to update or create maximum of 50 columns per page. The maximum number of columns an Oracle table can have is 1024. Trying to exceed that number of 50 updatable columns will result in an error. We will receive the same error if we try building manual tabular forms and index our column outside of the specified range (this error is similar to the error we get if we try to create a simple form with more than 100 items per page).



Figure 3.1 : Typical error message after referencing a non existing array

CAUTION: Keep in mind that new item types available in APEX 4 (simple checkbox and popup key LOV will require two of these ID’s). Therefore the limitation of 50 updatable columns per page may vary depending on how many of those elements we have.

# Tabular Forms and Excel Spreadsheets (Confusion)

One of the most frequent misunderstandings related to Tabular Forms is that we could use a Tabular Form as a replacement for Excel spreadsheets. At least one third of the questions posted in the forum and related to Tabular Forms are touching this issue or basically expressing this requirement. The fact is that we could modify Tabular Forms up to a certain extent in order to simulate the behaviour of an Excel sheet. We can add some javascript and Ajax to it and make it behave like an excel grid. However very soon we will realize that there are many limitations and that our code is exponentialy growing for every functionality we add. What we definitelly do not want is to later support and debug that code. Even if it is our own. Tabular Forms are not meant to be an Excel replacement – they have a completely different purpose.

# Tabular Forms and their Purpose

The main purpose of a Tabular form can be described by two cases:

* Maintaining smaller sets of data
* Maintaining parent – child relations

A typical example for the first case would be a page in our application where we maintain our lists of values. Normally a list of values will contain couple of records (options) and we could display this set of data on one page. The reason we would choose this method is that we can quickly edit and save our records without having to drill down, paginate or switch between the pages.

An example for the second case would be an application for the order management. An order would be a parent record (master) and ordered items would be child records (detail). In the most of the cases there is a limited (small) amount of details for one master record. Normally we would want to maintain that relation between the master and the detail on one page. In that case Tabular Forms are way to go.

This is of course only a recommendation on where or when to use Tabular Forms. Sure we can use them to update even thousands of records with up to 50 columns but in that case we will face a couple of issues:

* Our application performance will go down – APEX will call the apex\_item package for each row and each column we display on our page (this will be even worse if we build manual tabular forms since this will happen not only for displayed rows but for the whole result set). The time required to render a page after an update process will dramatically increase.
* The quality of our data will suffer and our users will lose their overview and would eventually need to remember their changes since they need to scroll the screen in all directions.
* The safety of our data will be poor. It is quite easy to accidentally delete records without even noticing it.

# Tabular Forms – New Features in APEX 4

As mentioned earlier, the first major change of Tabular Forms happened in the release 4 of APEX. A couple of very important features were added or enhanced. We will try to cover them all.

## New Item Types

In the releases prior to the release 4 of APEX, Tabular Forms were somewhat limited compared to the single row Forms. A couple of important items were missing. Release 4 corrected that issue by introducing the following item types:

* single checkbox
* jQuery date picker
* radio group
* popup key LOV

### Single Checkbox

In some cases our Tabular Form will need to provide a column of type checkbox to give our users a possibility to “flag” a record. Usually it would be “Yes” or “Y” for an activated checkbox and “No” or “N” for the not checked (empty) state. In the earlier versions of APEX such a requirement would cause a lot of coding just to create a workaround for a simple problem. The checkbox item was (is) different than the other items in APEX – its value will not be submitted to the server for the empty state. This means that the array (g\_f01) will contain the values for the checked items only and there was no really good way to go around this problem.

APEX 4 changed this in a positive way by introducing the Single Checkbox item. This item can be used as any other item and it is capable of storing either a single value for a checked or NULL for the empty state.

In order to demonstrate how this feature works, we will need to do couple of preparation steps first:

* create a new workspace (we need to make sure that our instance settings allow creation of the demo application - Workspace: INTERNAL Home > Manage Instance > Feature Configuration – “Create demonstration objects in new workspaces” needs to be set to “Yes”)
* this will create the required tables we need for this demonstration (EMP and DEPT)
* change our EMP table by adding an additional column and changing one of the columns – use the code below

ALTER TABLE emp MODIFY (ename VARCHAR2(40) NOT NULL);

ALTER TABLE emp ADD (valid VARCHAR2(1));

CREATE TABLE emp\_bkp AS SELECT \* FROM emp;

CREATE TABLE dept\_bkp AS SELECT \* FROM dept;

CREATE SEQUENCE dept\_seq START WITH 50 INCREMENT BY 10 NOCACHE;

CREATE SEQUENCE emp\_seq START WITH 7950 INCREMENT BY 1 NOCACHE;

CREATE OR REPLACE TRIGGER dept\_tr

BEFORE INSERT

ON dept

FOR EACH ROW

BEGIN

IF :NEW.deptno IS NULL

THEN

SELECT dept\_seq.NEXTVAL

INTO :NEW.deptno

FROM DUAL;

END IF;

END;

/

CREATE OR REPLACE TRIGGER emp\_tr

BEFORE INSERT

ON emp

FOR EACH ROW

BEGIN

IF :NEW.empno IS NULL

THEN

SELECT emp\_seq.NEXTVAL

INTO :NEW.empno

FROM DUAL;

END IF;

END;

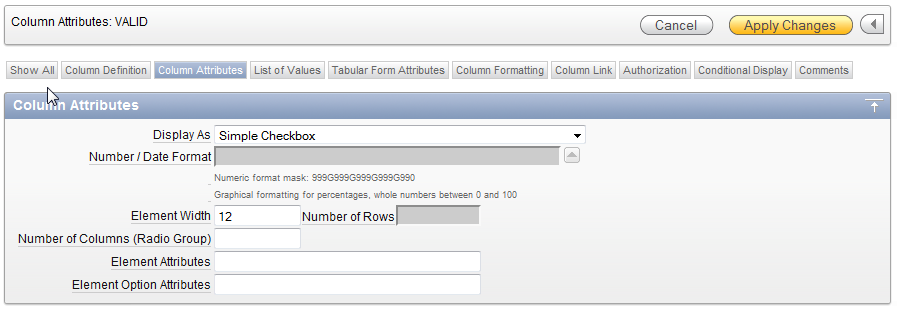
/

Snippet 3.1 : Extending EMP Table

* create a tabular form using wizard based on the EMP table and include all the columns

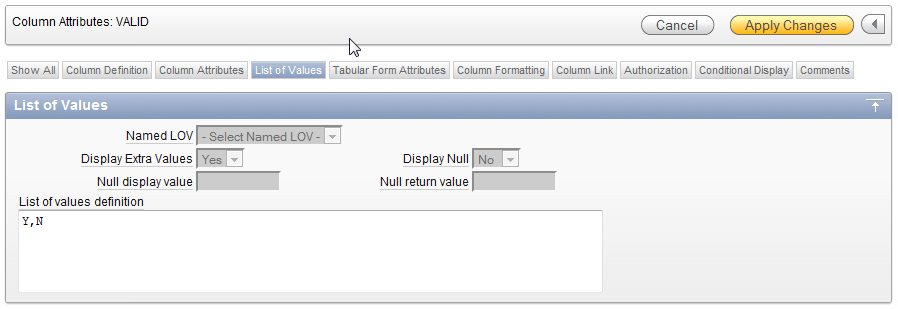
CAUTION: In this book we will not provide step by step instructions on how to create some basic functionality. Out assumption is that we already know how to accomplish that without further help.

Now, editing the Report Attributes and the newly created column properties for the column “VALID” we can change “Display As” to “Simple Checkbox” as shown in the figure 3.2



Using the “List of Values” tab we will need to enter the required static LOV for a simple checkbox:

Figure 3.2 : Using Simple Checkbox Item Type



TIP: If we require only one value we will need to change the list of values definition to “Y" instead of “Y,N”.

Figure 3.3 : Simple Checkbox LOV

After applying the changes, running our application and opening the page with the tabular form we should see a similar result:

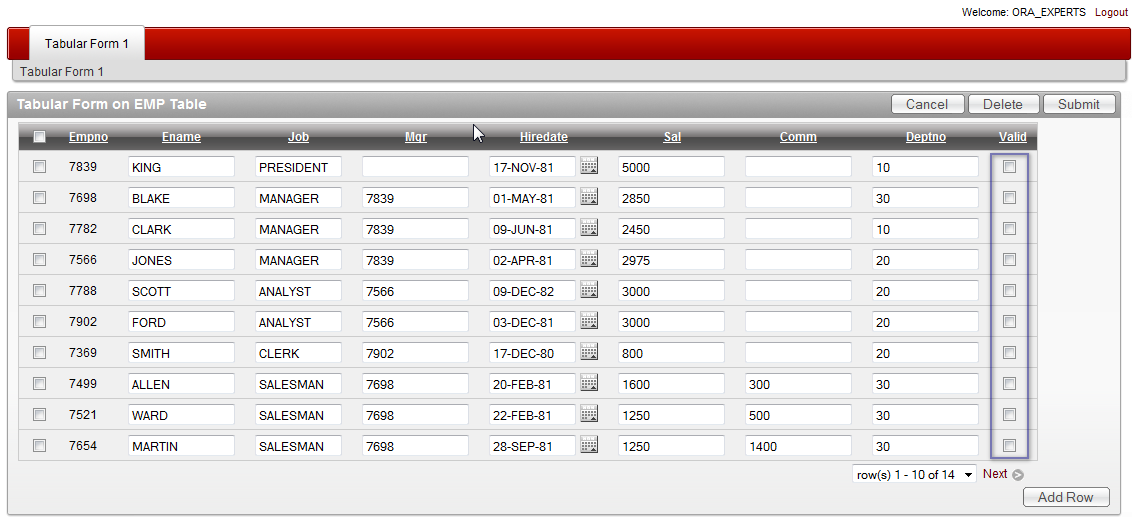


Figure 3.4 : Simple Checkbox in a Tabular Form

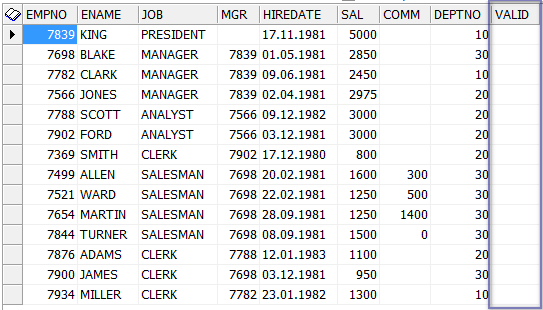


Figure 3.5 : EMP Table

CAUTION: All checkboxes are empty since we initially created a new empty column.

If we now activate couple of checkboxes and submit the changes Tabular Form will update the column to the expected values for the displayed set of rows only.

### jQuery Date Picker

APEX 4 wizard for creation of Tabular Forms will do some additional work for us and save us some time. If we have columns of type DATE or TIMESTAMP in our table and include those in a tabular form, APEX will automatically set it up as a Date Picker column using the new jQuery calendar:

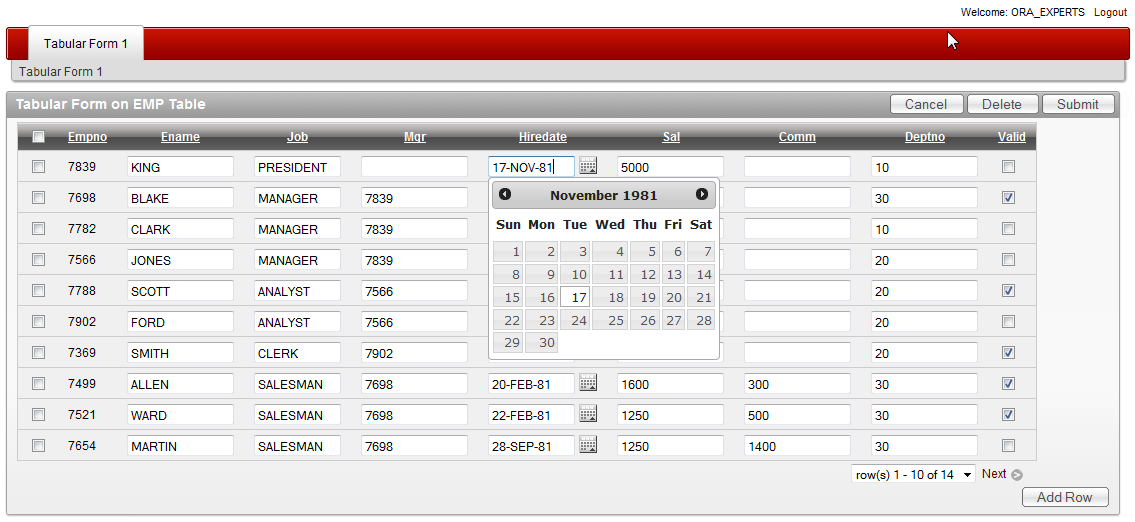


Figure 3.6: jQuery Date Picker in a Tabular Form

Unfortunately, we cannot extend that calendar feature the way we can do that in simple forms and specify number of months or add a year range to it. Hopefully this will be included in one of the next releases.

### Radio Group

Radio Group is the next item type which came with release 4 of APEX. In earlier versions we were able to manually create that item type by using apex\_item.radiogroup packaged function. The disadvantage was that APEX would take one array per entry in the radio group and we needed to write our own code in order to handle that problem while inserting or updating the record.

To show how this feature works we will now change the item type for the column VALID to “Radio Group (Static LOV)”, change the “List of values definition” to

STATIC:Yes;Y,No;N

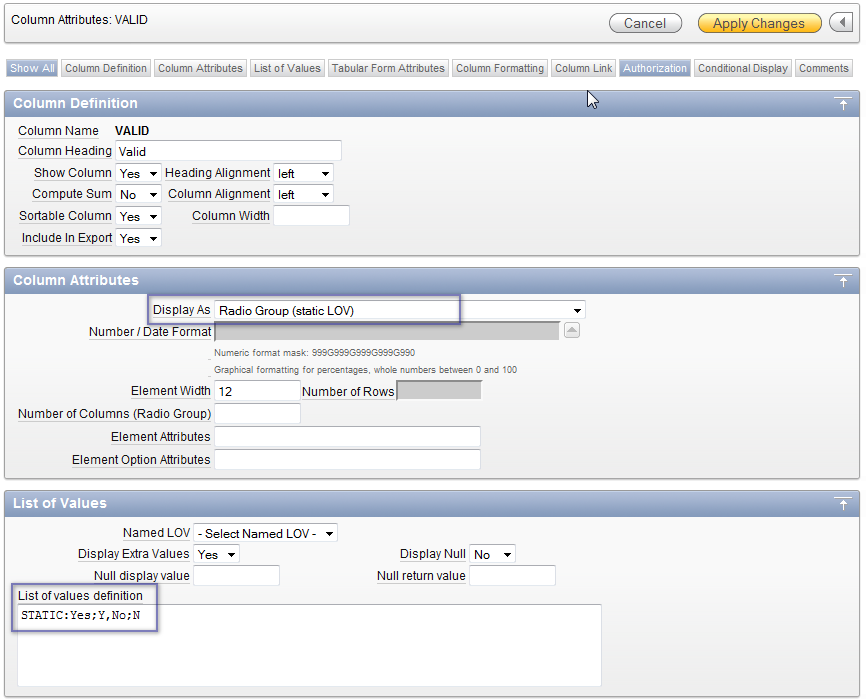


Figure 3.7: Changing Colum Attributes to Radio Group

After applying the changes, running our application and opening the page with the tabular form we should see a similar result:

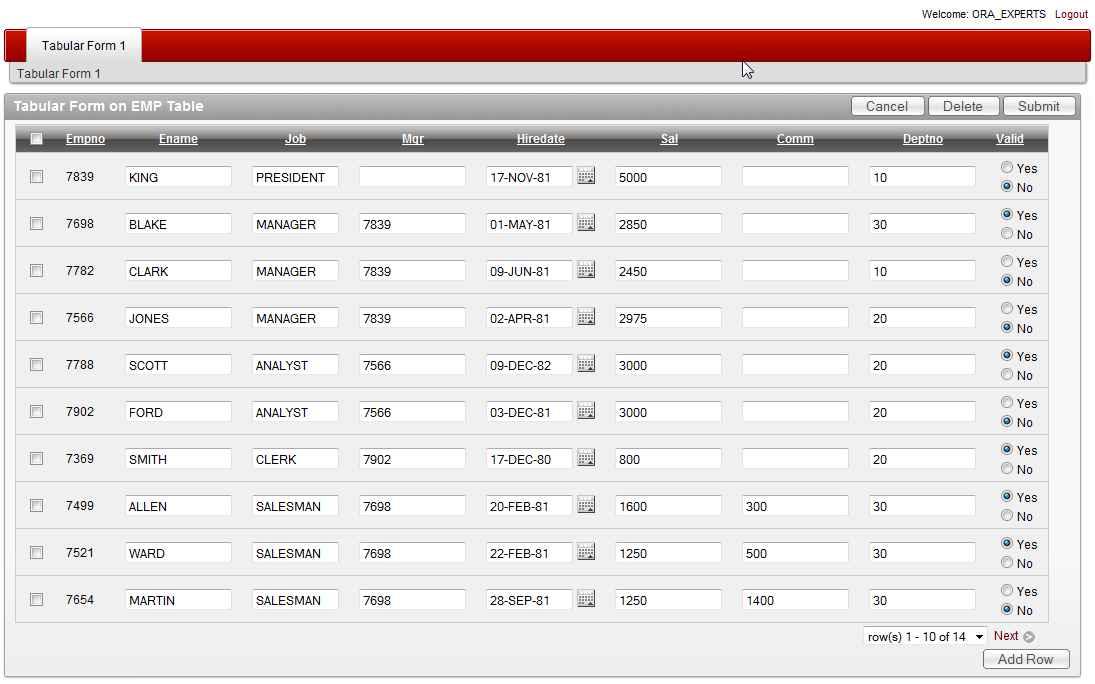


Figure 3.8: Tabular Form with Radio Group Column

### Popup Key LOV

The limitation for the select list item type regarding the number of entries we could use was one of the biggest problems in earlier releases of APEX. As soon as our list would grow over a certain size (a combination of the number and size of the available options) we would receive an error saying the character string buffer is too small:



For the simple forms the workaround was to use a popup LOV returning key value but for tabular forms this simply wasn’t possible. APEX 4 changed this by including this item type for Tabular Forms.

Figure 3.9: Select List Error

To test this feature we will need to modify our Tabular Form by editing the column DEPTNO and changing it to a “Popup Key LOV (query based LOV)” and adding the following query to the “List of values definition”:

SELECT dname d, deptno r

FROM dept

If we now apply the changes, run our application and sort the Tabular Form using the DEPTNO column, it should show the popup key LOV for the that column as in the following figure:

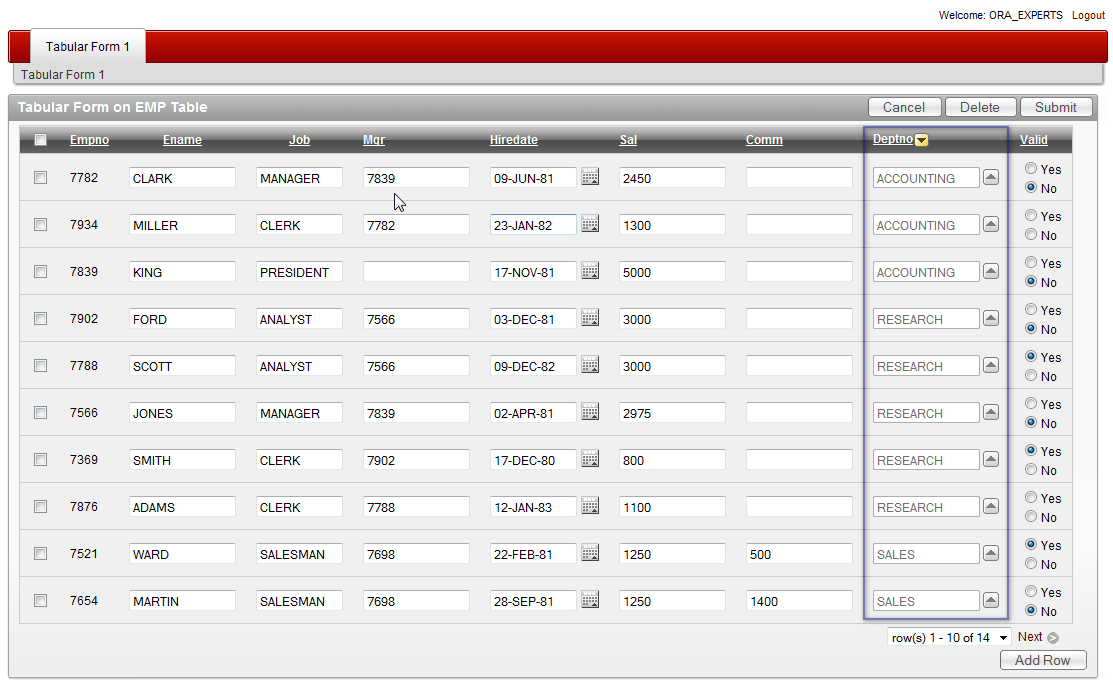


Figure 3.10: Tabular Form with a Popup Key LOV

One additional change was added to the select list item type in general. In earlier version of APEX the sorting on a column containing a select list would sort the values on the return value. Many times this was not desired. Since APEX 4 this behavior changed and the sorting is done on the displayed value as we can see in the Figure 3.10.

## Declarative Validations

APEX 4 introduced declarative validations for Tabular Forms. This feature is quite important since it saves some time while building our applications. Declarative validations can be created on a single column and are grouped in:

* NOT NULL validations
* Column String Comparison validations

Column String Comparison validations are several and we can view them in the figure 3.11:

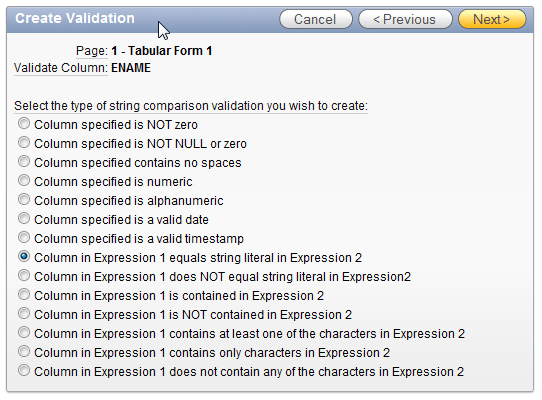


Figure 3.11: Tabular Form Validations – Column String Comparison

The current release of APEX does not provide declarative validations out of the scope of a single column. That type of validation is planned for one of the next releases. If such validations are needed, we will have to write our own code. Later on we will show how to that.

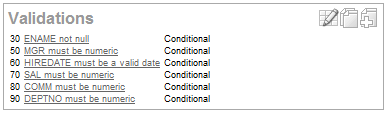
### Validations created by Tabular Form wizard

After creating a Tabular Form, we will find couple of validations created by the wizard. APEX wizard will use the table definition and sort out the constraints we defined for our table. It will create a validation for each single column containing a:

* NOT NULL definition (unless the column is a primary key used for the DML processes)

or for columns:

* of type numeric and
* columns of type date



As we can see in the figure 3.12, APEX Tabular Form wizard created several validations automatically. If we remember the changes we made in the snippet 3.1 we will notice that this change resulted in a validation checking the ENAME column (Validation with sequence 30 – ENAME IS NOT NULL). The other validations are related to the data type of all existing numeric and date columns included in our Tabular Form.

Figure 3.12: Tabular Form Validations – Created by Wizard

### Highlighting of validation errors

If we want to test the validations created for by the wizard, we can do couple of changes in our tabular for to see how those work:

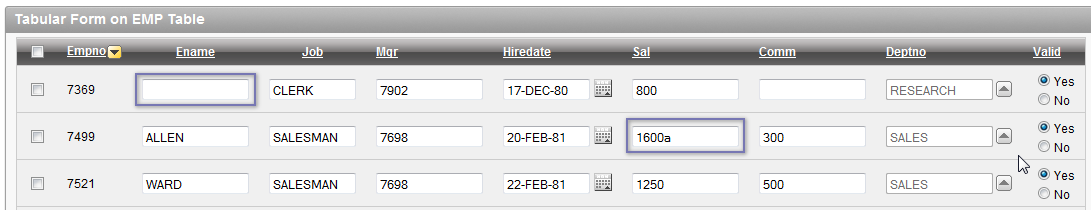


Figure 3.13: Tabular Form Validations

CAUTION: Prior to doing these changes and submitting those, we should change the sorting of our Tabular Form from DEPTNO to EMPNO. In the Version 4.0.2 of APEX there is a bug described here: http://forums.oracle.com/forums/thread.jspa?threadID=2139640

We changed the entry for the column ENAME to NULL and modified the entry for the column SALARY to an alphanumeric character. If we try to save that, we should receive an error like in the figure 3.14

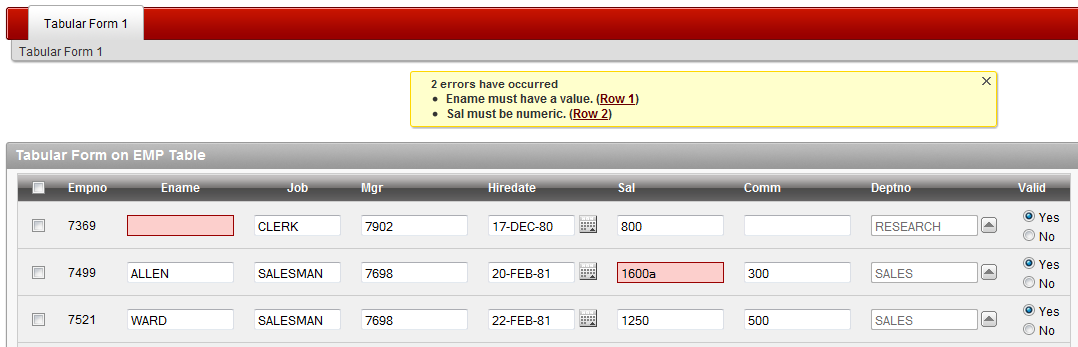


Figure 3.14: Tabular Form Validations – Error Highlighting

If we remember how such validations worked before the version 4, we will notice a couple of changes:

* APEX is now highlighting the cells in which there are validation errors found.
* APEX will tell us the name of the column affected by the error message.
* It will provide us with a direct link for setting focus to the affected cell as a part of the error message and.
* The most important difference is that we didn’t lose the changes we made.

In the earlier versions of APEX a lot of custom coding was required to get a similar functionality. In order to not lose the changes we made after an unsuccessful validation, it was required to display the validation errors on an error page and press the back button.

One more important thing needs to be mentioned. If we edit one of the validations created by the wizard, we will notice that there is a new substitution string in the “Error Message”. We can use #COLUMN\_HEADER# for the heading of the associated tabular form column in order to display the header name as a part of the error message:

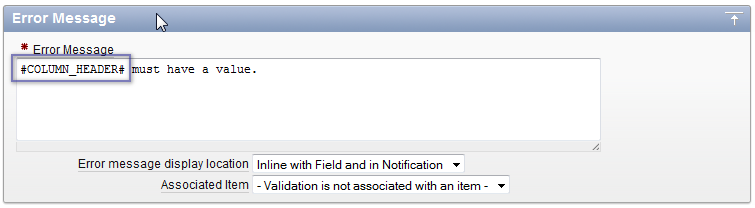


Figure 3.15: Tabular Form Validations – Substitution String for Column Names

## Other Features

APEX 4 also introduces couple of other neat features.

### Lost Update Protection

APEX 4 will inform us about the changes we made and about the risk to lose our changes if we try to:

* Change the sorting of the Tabular Form
* Try to paginate through the Tabular Form

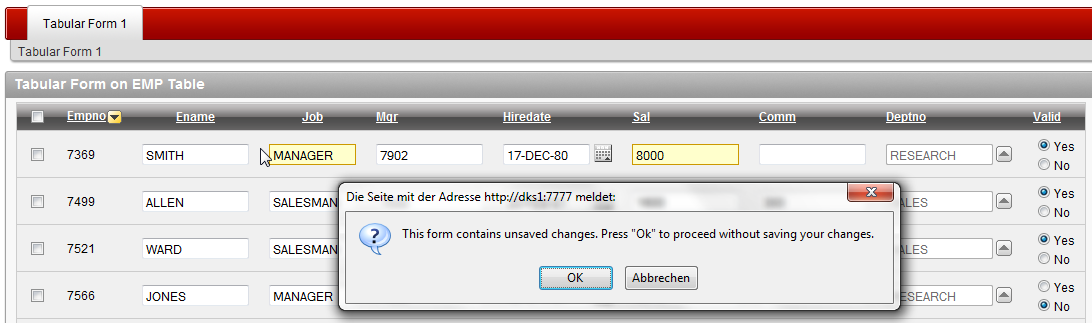


Figure 3.16: Tabular Form – Lost Update Protection

### Client side Add Row

In the earlier versions the process for adding new rows to a Tabular Form would require to submit and load our page once per new row. This process had to validate and save our changes each time we were adding a new row. The add row functionality is done by using new java script function

javascript:addRow();

which we can find in the ADD button – URL Target.

How does that work?

After the initial rendering of a Tabular Form APEX will remove that new line from the DOM and put it in a java script variable. After pressing the ADD button java script will replace couple of substitution strings in that variable (index, names, etc.) and add that row again to the end of the table. If we want to add multiple rows to a tabular form, the only thing we need to do is to press that button again.

### Reduced number of processes

Solving the problem by adding the client side add row functionality resulted also in a reduced number of processes for DML. In the earlier versions of APEX we would get four on submit processes required for DML:

* Apply MRU process for saving data on submit
* Apply MRU process for saving data after adding a new row
* Apply MRD process for deleting of data
* AddRow process for adding one or multiple rows

In APEX 4 we will find only two of those processes. One MRU process isn’t required any more since adding of rows is now on the client side The AddRow process became redundant and replaced by the same javascript function as well.

## Limitations (or features we are still waiting for)

The new Tabular Form feature of APEX 4 still has couple of limitations. Hopefully the most of them will be solved in one of the future releases of APEX (APEX 4.1).

These are:

* Declarative validations out of the single column scope.
* Multiple tabular forms per page.
* Item type settings equal to those for page items.
* Autocomplete and Autocomplete Returning Key Value for Tabular Forms.
* Dynamic Actions for Tabular Forms.

On the following pages we will try to describe some workarounds regarding these issues.

# Tabular Forms – Custom Coding

As we already said, we will need to know how to write our own code in case we have some kind of a validation we couldn’t cover using column string comparison or we need to do some DML.

## Hitting the right row

Assuming we have to write our custom process for deleting rows from a Tabular Form. Having in mind what we said about check box item (in the section Single Checkbox) for Tabular Forms we would need to do some special coding to be able to delete the row we marked. If we code it simply as:

BEGIN

FOR i IN 1 .. apex\_application.g\_f01.COUNT

LOOP

DELETE FROM emp

WHERE empno = apex\_application.g\_f02 (i);

-- g\_f02 is the hidden column containing

-- the primary key of the EMP table (empno)

END LOOP;

END;

Snippet 3.2 : Deleting Checked Rows – Common Mistake

we will delete the right number of rows but not those we checked. This code will actually delete the rows starting from the first row in the table and ending with *n* (representing the number of rows we checked).

Let us show how the right code should look like. Since we are going to create more of the processes like that it would make sense to put all those in a package related to this topic. We can then simply call this package in our processes or validations. We will also create couple of small procedures within this package in order to make the whole processing easier and not overload our page and application with loose PL/SQL blocks:

CREATE OR REPLACE PACKAGE BODY tab\_form\_emp\_pkg

AS

PROCEDURE disable\_foreign\_constraints

IS

BEGIN

FOR c IN (SELECT constraint\_name, table\_name

FROM user\_constraints

WHERE table\_name IN ('EMP', 'DEPT')

AND constraint\_type = 'R'

ORDER BY table\_name)

LOOP

EXECUTE IMMEDIATE 'ALTER TABLE '

|| c.table\_name

|| ' DISABLE CONSTRAINT '

|| c.constraint\_name;

END LOOP;

END disable\_foreign\_constraints;

PROCEDURE enable\_foreign\_constraints

IS

BEGIN

FOR c IN (SELECT constraint\_name, table\_name

FROM user\_constraints

WHERE table\_name IN ('EMP', 'DEPT')

AND constraint\_type = 'R'

ORDER BY table\_name)

LOOP

EXECUTE IMMEDIATE 'ALTER TABLE '

|| c.table\_name

|| ' ENABLE CONSTRAINT '

|| c.constraint\_name;

END LOOP;

END enable\_foreign\_constraints;

PROCEDURE restore\_tables

IS

-- We will use this process to restore our date after testing.

BEGIN

FOR c IN (SELECT constraint\_name, table\_name

FROM user\_constraints

WHERE table\_name IN ('EMP', 'DEPT')

AND constraint\_type = 'R'

ORDER BY table\_name)

LOOP

EXECUTE IMMEDIATE 'ALTER TABLE '

|| c.table\_name

|| ' DISABLE CONSTRAINT '

|| c.constraint\_name;

END LOOP;

EXECUTE IMMEDIATE 'TRUNCATE TABLE dept DROP STORAGE';

EXECUTE IMMEDIATE 'TRUNCATE TABLE emp DROP STORAGE';

INSERT INTO dept

SELECT \*

FROM dept\_bkp;

INSERT INTO emp

SELECT \*

FROM emp\_bkp;

COMMIT;

FOR c IN (SELECT constraint\_name, table\_name

FROM user\_constraints

WHERE table\_name IN ('EMP', 'DEPT')

AND constraint\_type = 'R'

ORDER BY table\_name)

LOOP

EXECUTE IMMEDIATE 'ALTER TABLE '

|| c.table\_name

|| ' ENABLE CONSTRAINT '

|| c.constraint\_name;

END LOOP;

END restore\_tables;

PROCEDURE delete\_emp\_row (p\_message OUT VARCHAR2)

IS

v\_row INTEGER;

v\_count INTEGER := 0;

BEGIN

FOR i IN 1 .. apex\_application.g\_f01.COUNT

LOOP

v\_row := apex\_application.g\_f01 (i);

DELETE FROM emp

WHERE empno = apex\_application.g\_f02 (v\_row);

-- g\_f02 is the hidden column containing

-- the primary key of the EMP table (empno)

v\_count := v\_count + 1;

END LOOP;

p\_message := v\_count || ' row(s) deleted.';

END delete\_emp\_row;

END tab\_form\_emp\_pkg;

/

Snippet 3.3 : Deleting Checked Rows – Procedure

Now, let us show how this will work. We need to do the following preparations for a test:

* Create an application item T\_MESSAGE and set the session state protection to “Restricted – may not be set from browser”. This application item we will use to display messages. In a case of an “ApplyMRD” process we have the substitution strings like #MRD\_COUNT# or #MRI\_COUNT# or #MRU\_COUNT#. For a custom process we need out own variable.
* Set the existing “ApplyMRD” process to Conditional > Never so it doesn’t run.
* Create an own On Submit Process > PL/SQL anonymous block > “ApplyMRD Manual” and use the following code.

BEGIN

tab\_form\_emp\_pkg.disable\_foreign\_constraints;

tab\_form\_emp\_pkg.delete\_emp\_row (:t\_message);

END;

* Use the substitution String for the success message

&T\_MESSAGE.

* Make the process conditional so it runs on condition type PL/SQL Expression.

:REQUEST IN ('MULTI\_ROW\_DELETE')

* Confirm that and create a process.

CAUTION: As we can notice, we used some additional procedures to disable constraints for referential integrity. This is for testing purposes only because we are talking here about an isolated case and we shouldn’t do that in practice. In the real life, we would create a validation checking if such constraints exist. We can use the restore procedure restore\_tables provide in the package in order to bring the objects and the data in the original state.

We can now test this process and try to delete couple of rows to see what we get.

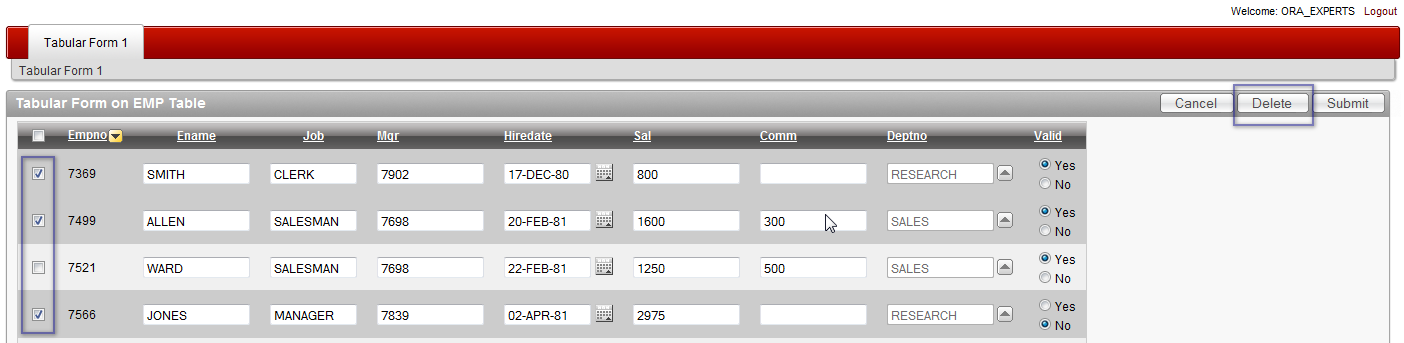


Figure 3.17: Tabular Form – Deleting Records using Custom Prozess

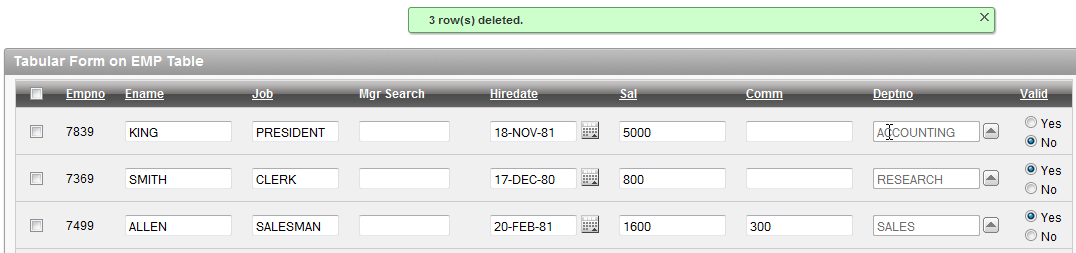


Figure 3.18: Tabular Form – Deleting Records - Results

Run the following block of code in SQL Workshop in order to restore the EMP table:

BEGIN

tab\_form\_emp\_pkg.restore\_tables;

END;

## Data Integrity

As we already mentioned, there is much more work to do if we write our own custom processes. Automatic processes created using Tabular Form wizard are taking care about data integrity and we would need to do the same kind of thing manually, once we start to write our own code. Automatic processes may display awkward errors but they are secure. It is not easy to code all of that code ourselves since there are many things we need to think off. Our goal is now to show how to do that and make you aware of the most important things you should keep in mind.

### Checksum

APEX 4 Tabular Form wizard will create a hidden checksum item for every Tabular Form we create. This checksum will be used for later automatic MRU and MRD processes. Using firebug extension of Firefox, we can see that hidden item if we investigate the generated HMTL:

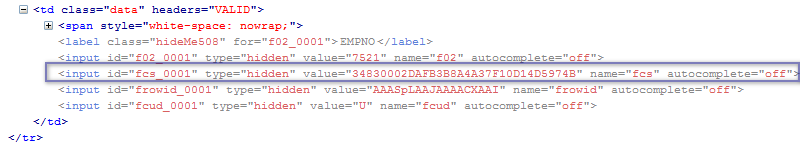


Figure 3.19: Tabular Form – Checksum

If we speak about checksum and data integrity, we need to keep in mind several issues:

* If we want to run our custom code, we will need to do a check and find out if the data in a row has changed – it doesn’t make sense to do an update on a row which hasn’t changed.
* If the data has changed, we will need to compare the checksum of the original data we loaded while rendering the page and the checksum of the current data in the table for each row.
* If a row has changed (old checksum in the g\_fcs array <> new checksum of the tabular form rows we calculate), we will do an update.
* However, we should be able to update only then if the original data in the table hasn’t been changed since the last fetch (old checksum in the g\_fcs array <> new checksum of the row in the table we need to calculate). Otherwise there should be an error displayed and we should stop the processing.

### Validations

Following these rules, we can start extending our package by adding couple of functions:

* The first function will compare the original checksum with the new generated checksum for each row in the Tabular Form. It will return a BOOLEAN.
* The second function will compare the original checksum with the checksum of the data in the table for each row where the first function returns FALSE. It will return a BOOLEAN as well.
* Finally, the third function will be a validation function returning an error message in the case some of the rows we are trying to update have been changed by other users – the second function returns FALSE.

Let’s extend the package

tab\_form\_emp\_pkg

by adding the first function we described:

FUNCTION compare\_checksum\_change (p\_array IN NUMBER)

RETURN BOOLEAN;

FUNCTION compare\_checksum\_change (p\_array IN NUMBER)

RETURN BOOLEAN

IS

BEGIN

IF apex\_application.g\_f02 (p\_array) IS NOT NULL

THEN

IF apex\_application.g\_fcs (p\_array) <>

wwv\_flow\_item.md5 (apex\_application.g\_f02 (p\_array),

apex\_application.g\_f03 (p\_array),

apex\_application.g\_f04 (p\_array),

apex\_application.g\_f05 (p\_array),

apex\_application.g\_f06 (p\_array),

apex\_application.g\_f07 (p\_array),

apex\_application.g\_f08 (p\_array),

apex\_application.g\_f09 (p\_array),

apex\_application.g\_f11 (p\_array)

)

THEN

RETURN FALSE;

ELSE

RETURN TRUE;

END IF;

ELSE

RETURN TRUE;

END IF;

END compare\_checksum\_change;

Snippet 3.4 : Checksum – Function I

CAUTION: We mentioned before that the items of type simple checkbox or popup key LOV will reserve two arrays for one item. This is the reason for the gap between the g\_f09 and g\_f11 arrays.

After that we will add the second function:

FUNCTION compare\_checksum\_table (p\_array IN NUMBER)

RETURN BOOLEAN;

FUNCTION compare\_checksum\_table (p\_array IN NUMBER)

RETURN BOOLEAN

IS

v\_empno NUMBER;

v\_emp\_checksum VARCHAR2 (40);

BEGIN

IF apex\_application.g\_f02 (p\_array) IS NOT NULL

THEN

v\_empno := apex\_application.g\_f02 (p\_array);

SELECT wwv\_flow\_item.md5 (empno,

ename,

job,

mgr,

hiredate,

sal,

comm,

deptno,

valid

)

INTO v\_emp\_checksum

FROM emp

WHERE empno = v\_empno;

IF apex\_application.g\_fcs (p\_array) <> v\_emp\_checksum

THEN

RETURN FALSE;

ELSE

RETURN TRUE;

END IF;

ELSE

RETURN TRUE;

END IF;

END compare\_checksum\_table;

Snippet 3.5: Checksum – Function II

As the last function we can now create a validation function returning VARCHAR2 which we will then call on the page:

FUNCTION validate\_data\_integrity

RETURN VARCHAR2;

FUNCTION validate\_data\_integrity

RETURN VARCHAR2

IS

v\_error VARCHAR2 (4000);

BEGIN

FOR i IN 1 .. apex\_application.g\_f02.COUNT

LOOP

IF NOT compare\_checksum\_change (i)

–- we changed the row

AND NOT compare\_checksum\_table (i)

-– however the table data has changed

THEN

v\_error :=

v\_error

|| '<br/>'

|| 'Row '

|| i

|| ': The version of the data in the '

|| 'table has been change since the last page '

|| 'rendering. Click <a href="f?p='

|| v ('APP\_ID')

|| ':'

|| v ('APP\_PAGE\_ID')

|| ':'

|| v ('APP\_SESSION')

|| '">here</a> to reload the page.';

END IF;

END LOOP;

v\_error := LTRIM(v\_error, '<br/>');

RETURN v\_error;

END validate\_data\_integrity;

Snippet 3.6: Checksum – Function III

We can test this code by

* Creating a page validation of type PL/SQL Function Returning Error Message > using sequence number 1 for the process > naming the process “Check Data Integrity” > process is unconditional > entering there the required call to the package procedure

BEGIN

RETURN tab\_form\_emp\_pkg.validate\_data\_integrity;

END;

* Changing one of the records using some other tools (SQL Workshop) and committing the changes.
* Going to the Tabular Form, changing there the same record and submitting the change.

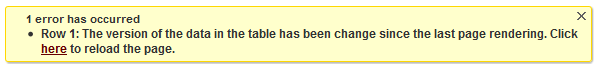


Figure 3.20: Tabular Form – Data Integrity Validation Error

INFO: We don’t need to use this code in a standard Tabular Form. A similar message will be generated automatically. We need this code only in a manually generated Tabular Form.

## Manual Tabular Forms

As long as APEX doesn’t allow creation of multiple standard Tabular Forms per page, we will be faced with a requirement to create a workaround for some complex forms where we need to manage relations between a master and multiple detail tables. Eventually one of the next versions of APEX will introduce this useful feature. Even then there will always be some cases where we need to (at least partially) use this technique. Let’s show how we write the code for a manual Tabular Form. We will use

* apex\_item.hidden
* apex\_item.checkbox and
* apex\_item.text

only and will not deal with other item types.

### SQL

The SQL created for a standard Tabular Form definitely looks much nicer than a code we write for a manual Tabular Form. For a manual Tabular Form we will need to use apex\_item package and parse a couple of parameters for

* column array and
* column value
* column size
* column max length

Let us create a second page (page 2) and call it Tabular Form 2. After that we will create a report region for our manual Tabular Form. We will need to modify our SQL and add a sub query. In the sub query we will create one empty row first, to be able to enter new rows upon request. We will also need to generate a checksum for the rows which we will use later on to check which rows have changed and run an update for those. This is how the query will look like:

SELECT apex\_item.checkbox (1, '#ROWNUM#') empno,

apex\_item.hidden (2, empno)

|| apex\_item.text (3, ename, 20, 20) ename,

apex\_item.text (4, job, 10, 10) job,

apex\_item.text (5, mgr, 5, 5) mgr,

apex\_item.text (6, hiredate, 12, 12) hiredate,

apex\_item.text (7, sal, 6, 6) sal,

apex\_item.text (8, comm, 6, 6) comm,

apex\_item.text (9, deptno, 4, 4)

|| apex\_item.hidden (10, checksum) deptno

FROM (SELECT NULL empno, NULL ename, NULL job,

NULL mgr, NULL hiredate,

NULL sal, NULL comm, NULL deptno,

NULL checksum

FROM DUAL

WHERE :request IN ('ADD')

UNION ALL

SELECT empno, ename, job, mgr, hiredate,

sal, comm, deptno,

wwv\_flow\_item.md5 (empno,

ename,

job,

mgr,

hiredate,

sal,

comm,

deptno

) checksum

FROM emp)

Snippet 3.7: Manual Tabular Form - SQL

The select statement for the empty column will run only if the REQUEST is set to the specified value.

The next step is to create four buttons required for:

* saving changes – button SUBMIT. Submitting the page.
* deleting rows – button MULTI\_ROW\_DELETE. The target of this button will be URL

javascript:apex.confirm(htmldb\_delete\_message,'MULTI\_ROW\_DELETE');

* adding rows – button ADD. This button will submit the page.
* canceling – button CANCEL. This button will redirect to the same page.

and position those in our report region.

We will also need to create at least two branching processes:

* On Submit – After Processing > Conditional > When Button Pressed > ADD > Page 2 > include process success message > Sequence 5
* On Submit – After Processing > Unconditional > Page 2 > include process success message > Sequence 10

The first branching will submit the page and redirect to the same page adding a REQUEST “ADD”.

We are now ready to write the code we will need for processing. We will create one process and one validation:

* ApplyMRU Manual – process for updating of existing and adding of new rows. This process will be conditional using PL/SQL Expression

:REQUEST IN ('ADD', 'SUBMIT')

* Validate Commission – validation which will make sure we can enter a commission value only for the department SALES (30). This validation will be unconditional.

For deleting of rows we can use the procedure we already created (Snippet 3.3 – Deleting Checked Rows) and copy the process “ApplyMRD Manual” from the page 1. Since we already know how to use checksum to ensure data integrity, we will not repeat that part in this example.

Let’s include the procedure for updating and the validation in our package.

PROCEDURE save\_emp\_custom (p\_message OUT VARCHAR2);

FUNCTION validate\_emp\_comm

RETURN VARCHAR2;

PROCEDURE save\_emp\_custom (p\_message OUT VARCHAR2)

IS

v\_ins\_count INTEGER := 0;

v\_upd\_count INTEGER := 0;

BEGIN

FOR i IN 1 .. apex\_application.g\_f02.COUNT

LOOP

BEGIN

IF apex\_application.g\_f02 (i) IS NOT NULL

THEN

IF apex\_application.g\_f10 (i) <>

wwv\_flow\_item.md5 (apex\_application.g\_f02 (i),

apex\_application.g\_f03 (i),

apex\_application.g\_f04 (i),

apex\_application.g\_f05 (i),

apex\_application.g\_f06 (i),

apex\_application.g\_f07 (i),

apex\_application.g\_f08 (i),

apex\_application.g\_f09 (i)

)

THEN

UPDATE emp

SET ename = apex\_application.g\_f03 (i),

job = apex\_application.g\_f04 (i),

mgr = apex\_application.g\_f05 (i),

hiredate = apex\_application.g\_f06 (i),

sal = apex\_application.g\_f07 (i),

comm = apex\_application.g\_f08 (i),

deptno = apex\_application.g\_f09 (i)

WHERE empno = apex\_application.g\_f02 (i);

v\_upd\_count := v\_upd\_count + 1;

END IF;

ELSE

INSERT INTO emp

(ename,

job,

mgr,

hiredate,

sal,

comm,

deptno

)

VALUES (apex\_application.g\_f03 (i),

apex\_application.g\_f04 (i),

apex\_application.g\_f05 (i),

apex\_application.g\_f06 (i),

apex\_application.g\_f07 (i),

apex\_application.g\_f08 (i),

apex\_application.g\_f09 (i)

);

v\_ins\_count := v\_ins\_count + 1;

END IF;

EXCEPTION

WHEN OTHERS

THEN

p\_message := p\_message || SQLERRM;

END;

END LOOP;

IF v\_ins\_count > 0 OR v\_upd\_count > 0

THEN

p\_message :=

p\_message

|| v\_ins\_count

|| ' row(s) inserted. '

|| v\_upd\_count

|| ' row(s) updated.';

END IF;

EXCEPTION

WHEN OTHERS

THEN

p\_message := SQLERRM;

END save\_emp\_custom;

FUNCTION validate\_emp\_comm

RETURN VARCHAR2

IS

v\_message VARCHAR2 (4000);

BEGIN

FOR i IN 1 .. apex\_application.g\_f02.COUNT

LOOP

IF apex\_application.g\_f09 (i) <> 30

AND apex\_application.g\_f08 (i) IS NOT NULL

THEN

v\_message :=

v\_message

|| '<br/>'

|| 'Commission is allowed for the sales department only.'

|| ' (Row '

|| i

|| ')';

END IF;

END LOOP;

v\_message := LTRIM (v\_message, '<br/>');

RETURN v\_message;

END validate\_emp\_comm;

Snippet 3.8: Manual Tabular Form – Update and Validation Processes

The update / insert process will loop through the array of EMPNO (g\_f02) and for all rows containing the primary key compare the original checksum with the new calculated checksum. If there is a difference it will update the corresponding rows and update the counter. For the added rows where the array is NULL it will insert a new row. We will use this PL/SQL block to start that process:

BEGIN

tab\_form\_emp\_pkg.save\_emp\_custom (:t\_message);

END;

and put our

&T\_MESSAGE.

in the it in the Process Success Message section.

The validation process is fairly simple. It will also loop through the array of EMPNO (g\_f02) and check if there is an entry for COMM column (g\_f08) where DEPTNO (g\_f09) is different than SALES (value 30). We will use this PL/SQL block for the validation:

BEGIN

RETURN tab\_form\_emp\_pkg.validate\_emp\_comm;

END;

We can now test our form to confirm that it works as expected.

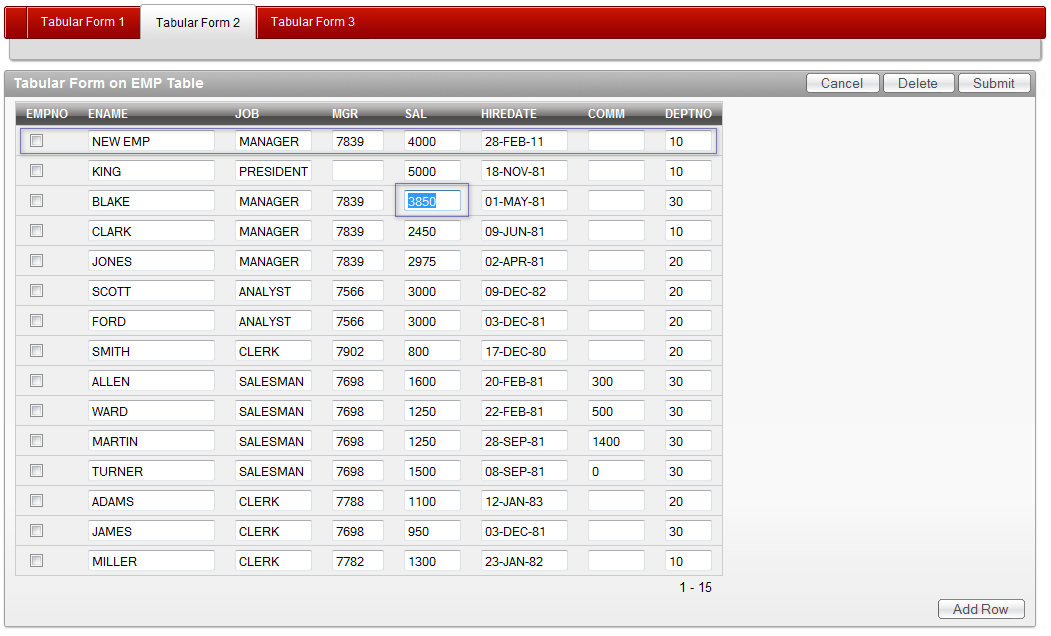
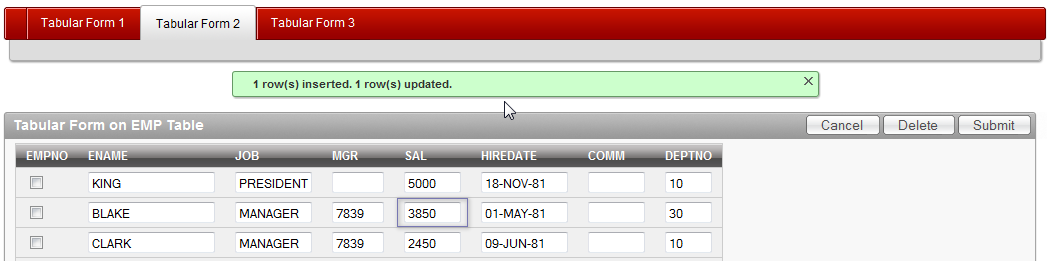


Figure 3.21: Manual Tabular Form



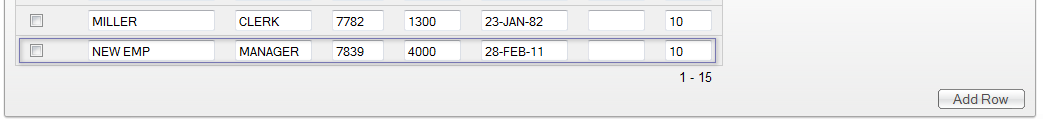
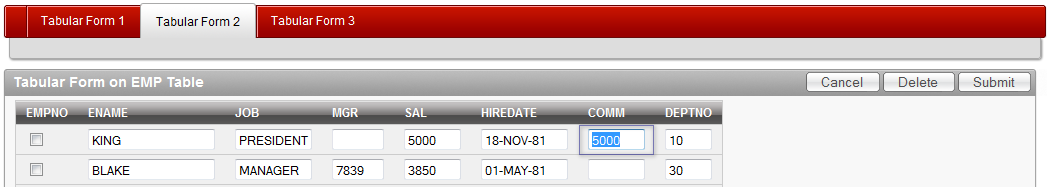


Figure 3.22 / 3.23: Manual Tabular Form – Success Message



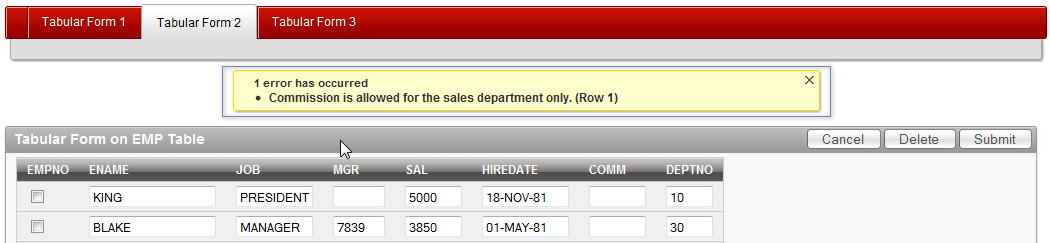


Figure 3.24 / 3.25: Manual Tabular Form – Validation Error Message

CAUTION: If we use a standard tabular form, you will not lose our changes after a validation error stops the page processing. In a manual tabular form the changes will be lost. If you want to keep the changes you will need to display the validation error on an error page or create a workaround using collections.

## Tabular Forms and Collections

Collection is one of the greatest features of APEX. APEX Collection is a set of tables, packaged procedures and functions for maintaining of session related data. We can use it to create, modify and delete our own datasets without having to touch the original source. Once the processing is done, we can decide either to save it back to the source (insert, update or delete) or to discard our changes. If we log off or somehow lose our session, we will not be able to retrieve that data again.

If there is a need to use a collection, we will probably talk about a small portion of data or about a window.

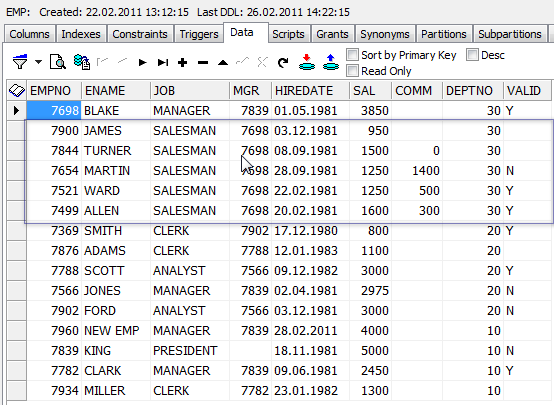


Figure 3.26: APEX Collection Data

Although a collection can accept almost an unlimited amount of data, it wouldn’t make sense to use it for loading hundreds or even thousands of records. It is designed to serve smaller datasets and it does it very well.

Our goal is to show how to work with collections using Tabular Forms. We will demonstrate a couple of possible ways and techniques. This exercise will not cover all the details and the techniques we showed in the previous examples we will not repeat here (validations, data integrity, etc.). In this exercise we will:

* Create a collection containing all employees of one department
* Create a couple of alternate processes we could use for updating collections from a Tabular Form
* Create a process for updating the original source using collection data.

We will continue with the methods we used before and will keep all of our code in a package. We will use the APEX regions and processes only to make a call to that code.

First, we will create a view and a procedure:

* Procedure for creating a collection containing all employees of one department.
* Create a view upon that collection.

PROCEDURE create\_emp\_collection (p\_deptno IN NUMBER,

p\_message OUT VARCHAR2);

PROCEDURE create\_emp\_collection (p\_deptno IN NUMBER,

p\_message OUT VARCHAR2)

IS

v\_collection VARCHAR2 (40) := 'EMP\_DEPT';

BEGIN

IF apex\_collection.collection\_exists (v\_collection)

THEN

apex\_collection.delete\_collection (v\_collection);

p\_message := 'Collection deleted.';

END IF;

apex\_collection.create\_collection\_from\_query

(v\_collection,

'SELECT a.\*, wwv\_flow\_item.md5(empno, ename, job, '

|| 'mgr, hiredate, sal, comm, deptno, valid) '

|| 'FROM EMP a WHERE deptno = '

|| p\_deptno

);

p\_message := p\_message || '<br/>' || 'Collection created.';

p\_message := LTRIM (p\_message, '<br/>');

END create\_emp\_collection;

CREATE OR REPLACE VIEW emp\_coll\_v

AS

SELECT seq\_id, c001 empno, c002 ename,

c003 job, c004 mgr, c005 hiredate,

c006 sal, c007 comm, c008 deptno,

c009 valid, c010 checksum, c011 delete\_flag

FROM apex\_collections

WHERE collection\_name = 'EMP\_DEPT';

Snippet 3.9: Tabular Form – Collection – Procedure and View

The process will check if the collection exists. If yes, it will delete it and create a new one based on the input. If it doesn’t exist, it will only create it.

The view will make it easier to deal with the collection. We will not need to remember the member number in order to insert, update or delete a row

We can now start creating a new page, page 3 and use a standard Tabular Form based on the view emp\_coll\_v. We will include all the column. Primary key will be the SEQ\_ID and EMPNO column. Editable columns are all columns. Tabular Form region title will be “Tabular Form Collection”. After the creation process, we will change the item type for the columns CHECKSUM and DELETE\_FLAG to “Hidden”.

We will also need to edit the generated SQL for the Tabular form and add there a condition:

WHERE delete\_flag IS NULL

This will exclude those records from the collection, we would mark as deleted.

We will create a select list within the Tabular Form region – P3\_DEPT. This select list will show a list of available departments based on the SQL query:

SELECT dname, deptno

FROM dept

We will also need a button displayed after the select list (Create a button displayed among this region's items) which we will use to trigger our process. Let’s name this button P3\_GO and assign it a request GO.

Now we can start creating a page process (Create Collection) on submit for creating a collection. We will use the following PL/SQL block for that:

BEGIN

tab\_form\_emp\_pkg.create\_emp\_collection (:p3\_deptno, :t\_message);

END;

and make it conditional to run based on a PL/SQL Expression

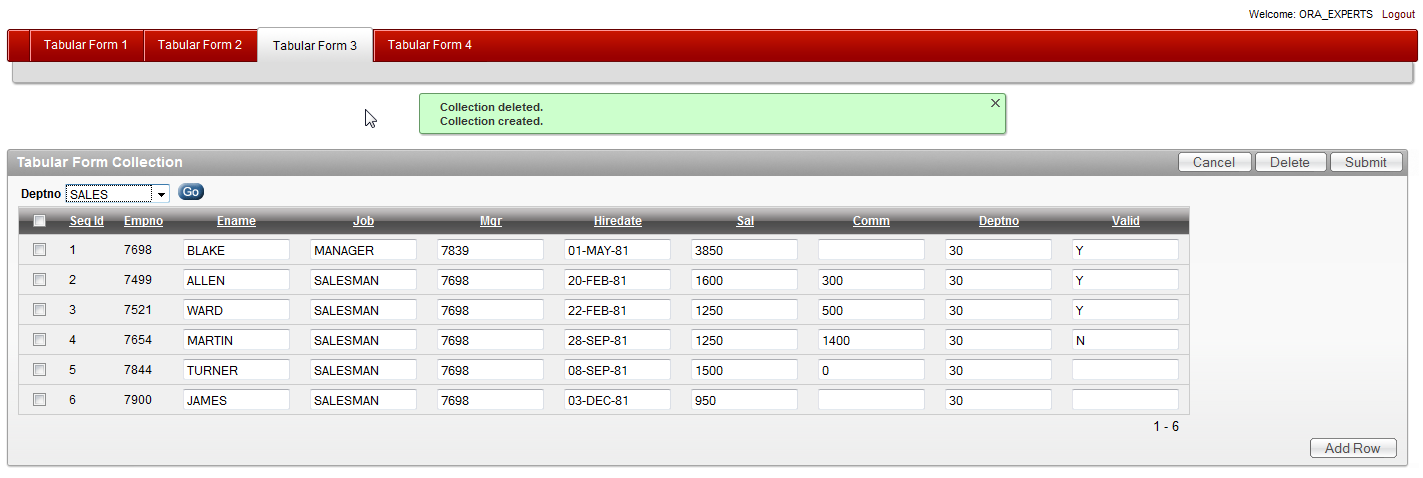
:REQUEST IN ('GO')

We shouldn’t forget to put

&T\_MESSAGE.

in the Success Message of the process.

Finally, we have everything set for testing of what we have done. If we select the sales department from the list and press the GO button, we should get a result similar to the one showed in the next figure:



The question now is: How do we update our collection?

Figure 3.27: Tabular Form on APEX Collection

There are a couple of possible ways to do that. We will examine three different methods.

### Instead of Trigger

The ideal way would be to create three instead of triggers on the view for updating, deleting and inserting rows. Thinking further, we could imagine writing a package which would automatically create all of the code for us (I also saw some examples on the internet). We would just need to provide our collection query and the package would generate all of the code for use. After that we would just place that code in our application and it would run with the automatic DML process of Tabular Forms… But this is obviously too good to be true. Currently, you can’t get the instead of triggers to work with collections. If you try doing that you will receive all kinds of strange looking error messages.

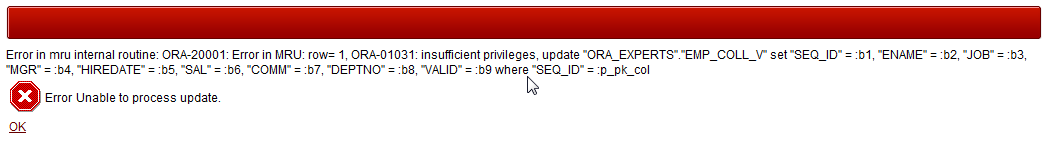


Figure 3.28: Tabular Form on APEX Collection – Instead of Trigger Error

We can immediately notice that it has to do something with the privileges. We would need to grant access privileges on WWV\_FLOW\_COLLECTION to our schema and this is definitely not a way to go.

We will delete the automatic DML processes wizard created for us on the page 3.

### Writing packaged Update Procedures

We will need to create two procedures in our package. These procedures are similar to those we wrote for the manual Tabular Forms. Let’s add those to our package:

PROCEDURE save\_emp\_coll\_custom (p\_message OUT VARCHAR2);

PROCEDURE delete\_emp\_coll\_custom (p\_message OUT VARCHAR2);

PROCEDURE save\_emp\_coll\_custom (p\_message OUT VARCHAR2)

IS

v\_ins\_count INTEGER := 0;

v\_upd\_count INTEGER := 0;

v\_collection VARCHAR2 (40) := 'EMP\_DEPT';

BEGIN

FOR i IN 1 .. apex\_application.g\_f02.COUNT

LOOP

BEGIN

IF apex\_application.g\_f02 (i) IS NOT NULL

THEN

IF apex\_application.g\_f12 (i) <>

wwv\_flow\_item.md5 (apex\_application.g\_f03 (i),

apex\_application.g\_f04 (i),

apex\_application.g\_f05 (i),

apex\_application.g\_f06 (i),

apex\_application.g\_f07 (i),

apex\_application.g\_f08 (i),

apex\_application.g\_f09 (i),

apex\_application.g\_f10 (i),

apex\_application.g\_f11 (i)

)

THEN

apex\_collection.update\_member

(p\_collection\_name => v\_collection,

p\_seq => apex\_application.g\_f02(i),

p\_c001 => apex\_application.g\_f03(i),

p\_c002 => apex\_application.g\_f04(i),

p\_c003 => apex\_application.g\_f05(i),

p\_c004 => TO\_NUMBER(apex\_application.g\_f06(i)),

p\_c005 => TO\_DATE(apex\_application.g\_f07(i)),

p\_c006 => TO\_NUMBER(apex\_application.g\_f08(i)),

p\_c007 => TO\_NUMBER(apex\_application.g\_f09(i)),

p\_c008 => TO\_NUMBER(apex\_application.g\_f10(i)),

p\_c009 => apex\_application.g\_f11(i),

p\_c010 => apex\_application.g\_f12(i),

p\_c011 => apex\_application.g\_f13(i));

v\_upd\_count := v\_upd\_count + 1;

END IF;

ELSE

apex\_collection.add\_member

(p\_collection\_name => v\_collection,

p\_c001 => emp\_seq.NEXTVAL,

p\_c002 => apex\_application.g\_f04(i),

p\_c003 => apex\_application.g\_f05(i),

p\_c004 => TO\_NUMBER(apex\_application.g\_f06(i)),

p\_c005 => TO\_DATE(apex\_application.g\_f07(i)),

p\_c006 => TO\_NUMBER(apex\_application.g\_f08(i)),

p\_c007 => TO\_NUMBER(apex\_application.g\_f09(i)),

p\_c008 => TO\_NUMBER(apex\_application.g\_f10(i)),

p\_c009 => apex\_application.g\_f11(i)

);

v\_ins\_count := v\_ins\_count + 1;

END IF;

EXCEPTION

WHEN OTHERS

THEN

p\_message := p\_message

|| '<br/>'

|| 'Row: '

|| i

|| ' > '

|| SQLERRM;

p\_message := LTRIM (p\_message, '<br/>');

END;

END LOOP;

IF v\_ins\_count > 0 OR v\_upd\_count > 0

THEN

p\_message :=

p\_message || '<br/>'

|| v\_ins\_count

|| ' row(s) inserted. '

|| v\_upd\_count

|| ' row(s) updated.';

END IF;

p\_message := LTRIM (p\_message, '<br/>');

EXCEPTION

WHEN OTHERS

THEN p\_message := SQLERRM;

END save\_emp\_coll\_custom;

PROCEDURE delete\_emp\_coll\_custom (p\_message OUT VARCHAR2)

IS

v\_row INTEGER;

v\_count INTEGER := 0;

v\_collection VARCHAR2 (40) := 'EMP\_DEPT';

BEGIN

FOR i IN 1 .. apex\_application.g\_f01.COUNT

LOOP

v\_row := apex\_application.g\_f01 (i);

apex\_collection.update\_member

(p\_collection\_name => v\_collection,

p\_seq => apex\_application.g\_f02(v\_row),

p\_c011 => 'Y');

v\_count := v\_count + 1;

END LOOP;

p\_message := v\_count || ' row(s) deleted.';

END delete\_emp\_coll\_custom;

Snippet 3.10: Tabular Form – Collection – Update and Delete Procedures

Before we can start testing this code, we will create two on submit processes on our page 3:

* Update Collection – process for updating of existing and adding of new rows. This process will be conditional using PL/SQL Expression

:REQUEST IN ('ADD', 'SUBMIT')

* Delete Collection Member – process for flagging of deleted records to “Y” This process will be conditional using PL/SQL Expression

:REQUEST IN ('MULTI\_ROW\_DELETE')

We are going to use the following PL/SQL blocks to run those processes:

BEGIN

tab\_form\_emp\_pkg.save\_emp\_coll\_custom (:t\_message);

END;

BEGIN

tab\_form\_emp\_pkg.delete\_emp\_coll\_custom (:t\_message);

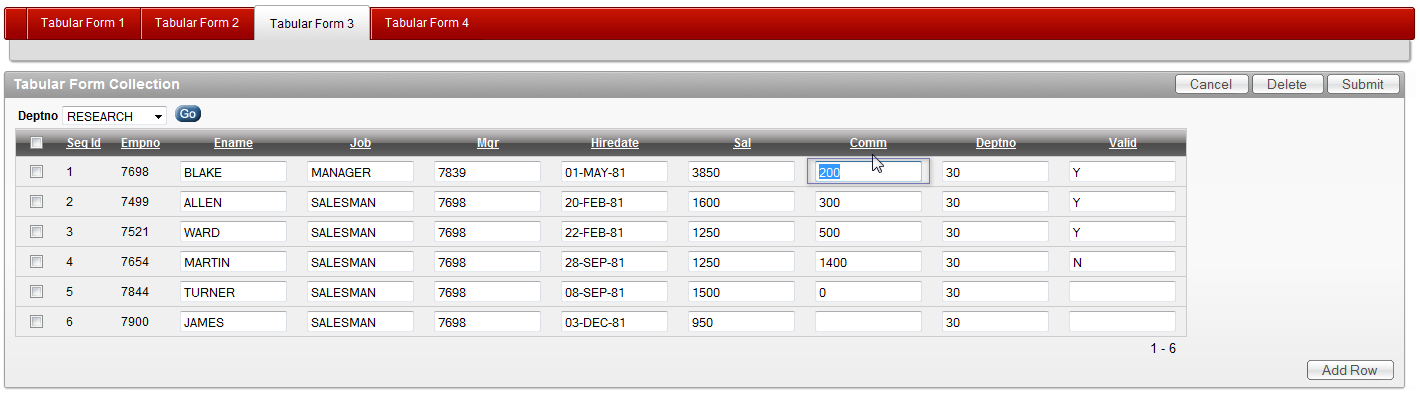
END;

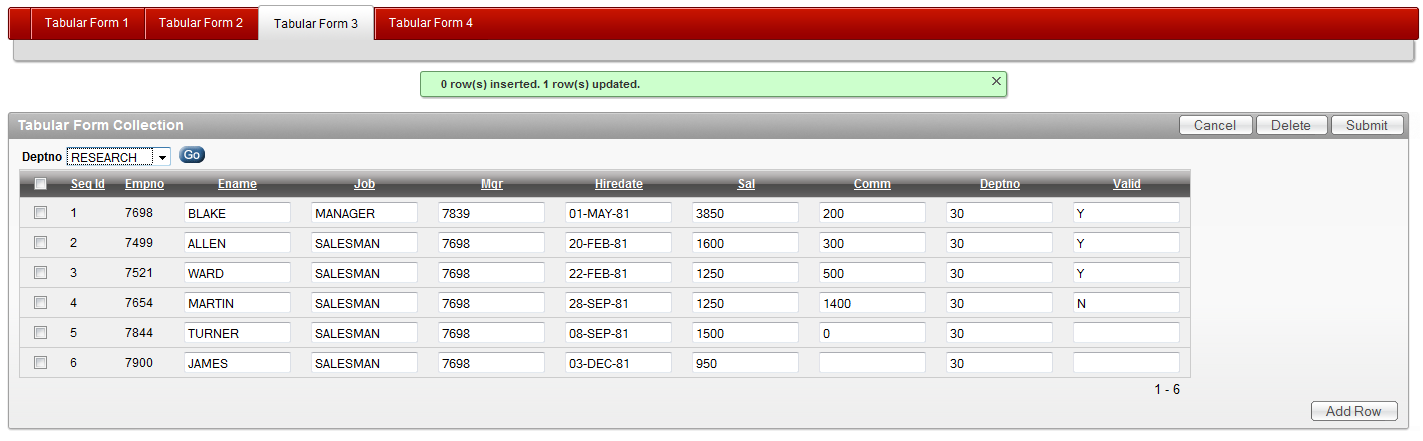
and put our

&T\_MESSAGE.

in the Process Success Message section.

Now, let’s test the page 3 and change one of the records in our Tabular Form. We should get a similar result to the one shown in the figure 3.209





Figures 3.29 and 3.30: Tabular Form on APEX Collection – Update Process

### Updating Collection using On Demand Process

Since collections are only a snapshot of the original data, we can use a different approach for an update. We can create some javascript code and combine this with an on demand process to update the collection on change of a single Tabular Form item. The validation of the collection values will be done only if we decide to save the collection back to the source. We will include only some basic validations for numeric and date columns in this code.

The code we need to write for this demonstration consists of:

* One procedure for updating of a collection member.
* Two small javascript functions.
* Three page items we will use for parsing parameters

T\_COL\_VAL\_ITEM, T\_COL\_SEQ\_ITEM, T\_COL\_ATTR\_ITEM

* One on demand process to call our procedure.

The procedure code we will add to our package is as follows:

PROCEDURE update\_emp\_coll\_member (

p\_seq\_id IN NUMBER,

p\_attribute\_number IN NUMBER,

p\_attribute\_value IN VARCHAR2

);

PROCEDURE update\_emp\_coll\_member (

p\_seq\_id IN NUMBER,

p\_attribute\_number IN NUMBER,

p\_attribute\_value IN VARCHAR2

)

IS

v\_collection VARCHAR2 (40) := 'EMP\_DEPT';

v\_number NUMBER;

v\_date DATE;

v\_message VARCHAR2 (4000);

BEGIN

IF p\_seq\_id IS NOT NULL

THEN

IF p\_attribute\_number IN (4, 6, 7, 8)

THEN

v\_number := TO\_NUMBER (p\_attribute\_value);

ELSIF p\_attribute\_number IN (5)

THEN

v\_date := TO\_DATE (p\_attribute\_value);

END IF;

apex\_collection.update\_member\_attribute

(p\_collection\_name => v\_collection,

p\_seq => p\_seq\_id,

p\_attr\_number => p\_attribute\_number,

p\_attr\_value => p\_attribute\_value

);

END IF;

EXCEPTION

WHEN OTHERS

THEN

v\_message := SQLERRM;

HTP.p (v\_message);

END update\_emp\_coll\_member;

Snippet 3.11: Tabular Form – Collection – Update on Demand

Javascript code, added to the page header of the page 3 will call the on demand process and check the return value of that process for errors:

<script language="JavaScript" type="text/javascript">

function LPad(ContentToSize,PadLength,PadChar)

{

var PaddedString=ContentToSize.toString();

for(i=ContentToSize.length+1;i<=PadLength;i++)

{PaddedString=PadChar+PaddedString;}

return PaddedString;

}

function f\_update\_emp\_coll\_member(pThis,pRownum,pAttribNo){

var v\_seq\_array = 'f02\_' + LPad(pRownum, 4, "0");

var SeqID = $x(v\_seq\_array).value;

var get = new htmldb\_Get(null,$x('pFlowId').value,

'APPLICATION\_PROCESS=update\_emp\_coll\_member',0);

get.add('T\_COL\_VAL\_ITEM',pThis);

get.add('T\_COL\_SEQ\_ITEM',SeqID);

get.add('T\_COL\_ATTR\_ITEM',pAttribNo);

gReturn = get.get();

if(gReturn) {alert(gReturn)}

get = null;

}

</script>

Snippet 3.12: Tabular Form – Collection – Ajax

On Demand process “update\_emp\_coll\_member” will be a simple PL/SQL block:

BEGIN

tab\_form\_emp\_pkg.update\_emp\_coll\_member (:t\_col\_seq\_item,

:t\_col\_attr\_item,

:t\_col\_val\_item);

END;

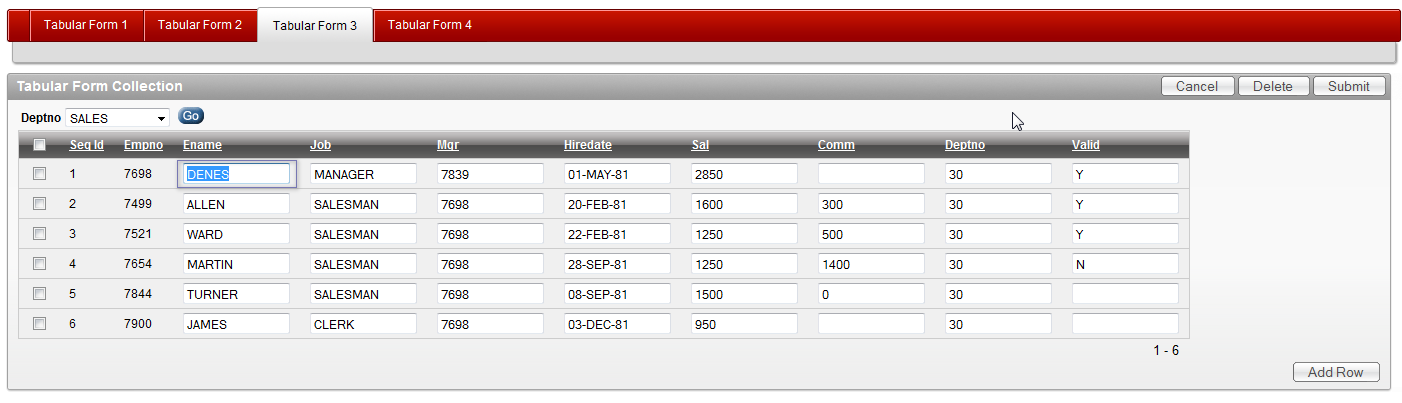
The last thing we need to do is to call this code from the Tabular Form. We will do that by entering the following in the ENAME column properties > Element Attributes

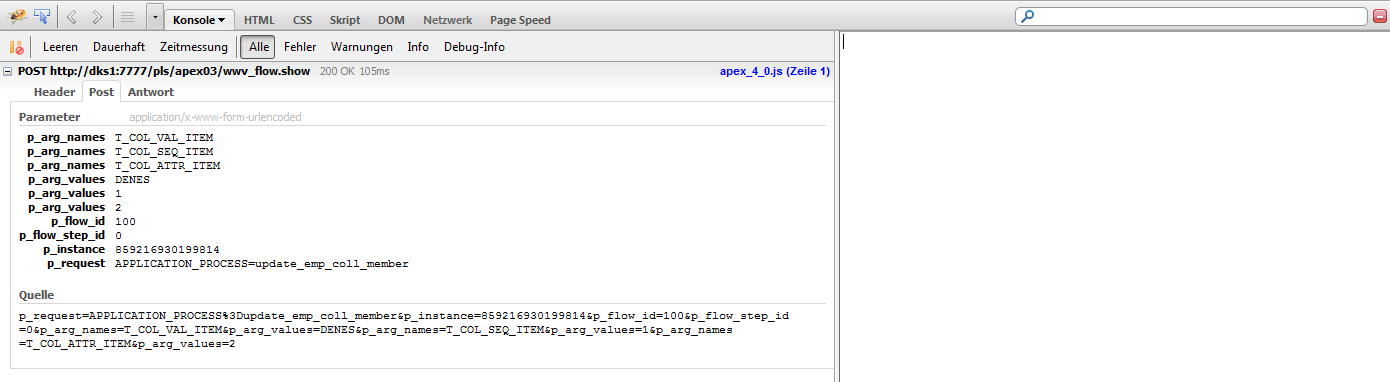
onchange="f\_update\_emp\_coll\_member(this.value,'#ROWNUM#',2);"

This needs to be done for all the other columns accordingly.

CAUTION: The third parameter of the javascript function expects the attribute number of the column. We need to be careful with that and adjust this value to the right one. The mapping we have in the view code can help us with that.

If we now run the page 3 and try to update some of the rows in the Tabular Form, we can see what happens if we use Firebug extension for Firefox.





Figures 3.31 and 3.32: Tabular Form on APEX Collection – Updating Collection using On Demand Process

The final step in this exercise is to save the collection data back to the table. For that we will create:

* One validation function which will take care of the data integrity.
* One procedure which will save the data back to the table.

FUNCTION validate\_collection\_data

RETURN VARCHAR2;

PROCEDURE update\_table\_from\_collection (

p\_deptno IN NUMBER,

p\_message OUT VARCHAR2

);

FUNCTION validate\_collection\_data

RETURN VARCHAR2

IS

v\_message VARCHAR2 (4000);

v\_checksum VARCHAR2 (400);

BEGIN

FOR c IN (SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno,

valid, checksum, delete\_flag

FROM emp\_coll\_v

WHERE checksum IS NOT NULL)

LOOP

SELECT wwv\_flow\_item.md5 (empno,

ename,

job,

mgr,

hiredate,

sal,

comm,

deptno,

valid

)

INTO v\_checksum

FROM emp

WHERE empno = c.empno;

IF c.checksum <> v\_checksum

THEN

v\_message :=

v\_message

|| '<br/>'

|| 'Empno: '

|| c.empno

|| ': Snapshot too old.';

END IF;

END LOOP;

v\_message := LTRIM (v\_message, '<br/>');

RETURN v\_message;

END validate\_collection\_data;

PROCEDURE update\_table\_from\_collection (

p\_deptno IN NUMBER,

p\_message OUT VARCHAR2

)

IS

v\_ins\_count INTEGER := 0;

v\_upd\_count INTEGER := 0;

v\_del\_count INTEGER := 0;

v\_message VARCHAR2 (4000);

BEGIN

FOR c IN (SELECT empno, ename, job, mgr, hiredate,

sal, comm, deptno,

valid, checksum, delete\_flag

FROM emp\_coll\_v)

LOOP

IF c.delete\_flag IS NULL AND c.checksum IS NOT NULL

THEN

IF c.checksum <>

wwv\_flow\_item.md5 (c.empno,

c.ename,

c.job,

c.mgr,

c.hiredate,

c.sal,

c.comm,

c.deptno,

c.valid

)

THEN

UPDATE emp

SET ename = c.ename,

job = c.job,

mgr = c.mgr,

hiredate = c.hiredate,

sal = c.sal,

comm = c.comm,

deptno = c.deptno

WHERE empno = c.empno;

v\_upd\_count := v\_upd\_count + 1;

END IF;

ELSIF c.delete\_flag IS NULL AND c.checksum IS NULL

THEN

INSERT INTO emp

(empno, ename, job, mgr,

hiredate, sal,

comm, deptno, valid

)

VALUES (c.empno, c.ename, c.job,

c.mgr, c.hiredate, c.sal,

c.comm, c.deptno, c.valid

);

v\_ins\_count := v\_ins\_count + 1;

ELSIF c.delete\_flag IS NOT NULL AND c.checksum IS NOT NULL

THEN

DELETE FROM emp

WHERE empno = c.empno;

v\_ins\_count := v\_ins\_count + 1;

END IF;

END LOOP;

p\_message :=

p\_message

|| '<br/>'

|| v\_ins\_count

|| ' row(s) inserted. '

|| v\_upd\_count

|| ' row(s) updated. '

|| v\_del\_count

|| ' row(s) deleted.';

create\_emp\_collection (p\_deptno, v\_message);

p\_message :=

p\_message

|| '<br/>' || v\_message;

p\_message := LTRIM (p\_message, '<br/>');

EXCEPTION

WHEN OTHERS

THEN

p\_message := SQLERRM;

END update\_table\_from\_collection;

To be able to do an update of the table, we will need to create a button on our page 3 and name it

Snippet 3.13: Tabular Form – Saving Collection Data back to the Source

SAVE\_DATA

We can now call the validation function and the update process on our page 3. We will create a validation of type PL/SQL Function Returning Error Message – Validate Collection Data and make it conditional upon the new button we created (SAVE\_DATA)

BEGIN

RETURN tab\_form\_emp\_pkg.validate\_collection\_data;

END;

The process – Save Collection to Table – will be on submit and it will be conditional when button pressed (SAVE\_DATA). We will also enter

&T\_MESSAGE.

in the Success Message of the process

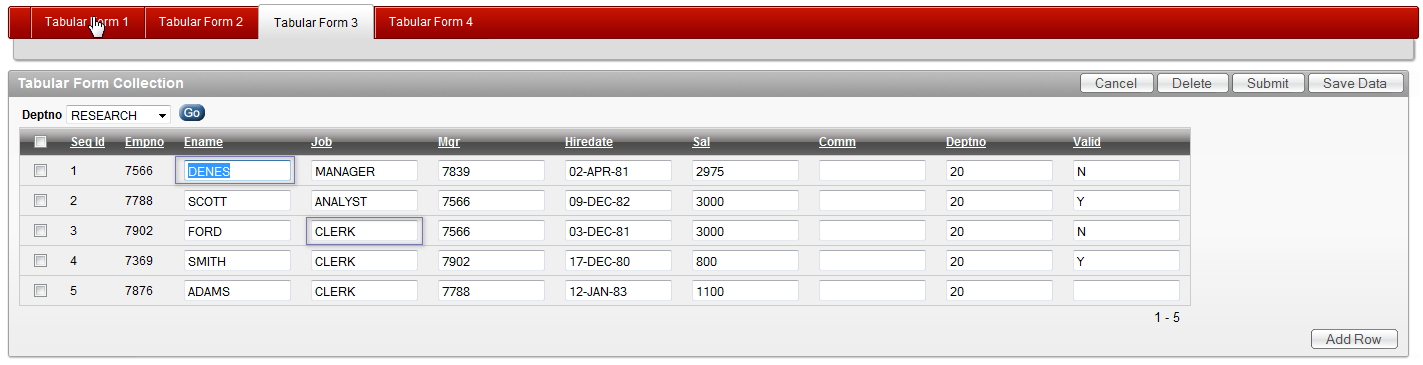
BEGIN

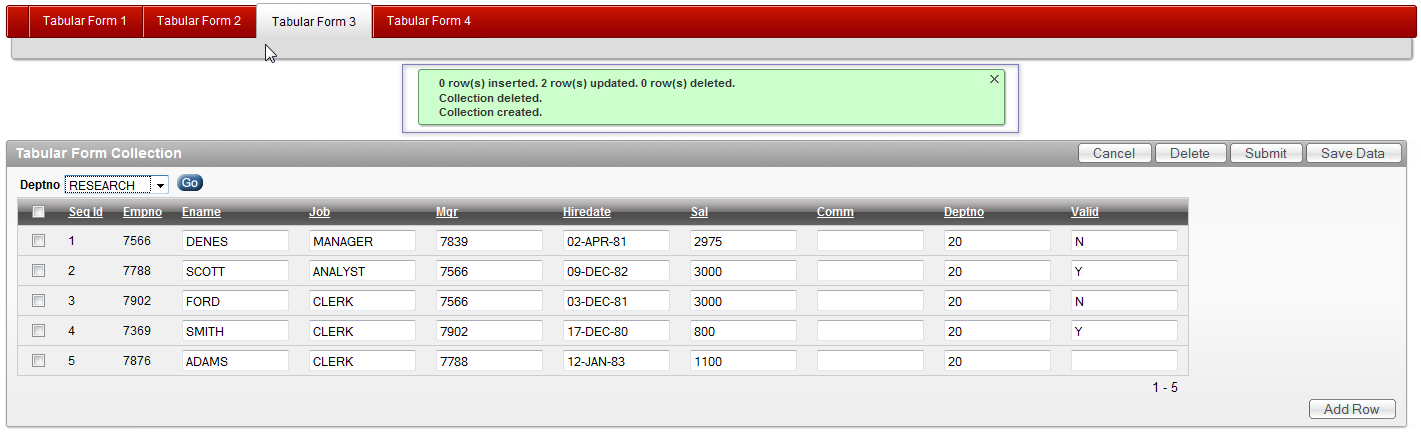
tab\_form\_emp\_pkg.update\_table\_from\_collection

(:p3\_deptno,:t\_message);

END;

Let’s now do couple of changes in our collection and confirm the code works as expected.





Figures 3.33 / 3.34: Tabular Form on APEX Collection – Saving Changes to the Source Table

CAUTION: The examples we show here are not complete and they are focusing single functionalities. If you want to use them in your applications, you would need to complete those. In the example for updating a collection using an on demand process, you would still need to take care of the new added rows. The code we showed doesn’t do that. However, that should be fairly easy to add.

# Interesting Tips

In this part we will cover couple of useful tips we can program ourselves using some existing tips from the internet. These will be

* Simple Autocomplete for Tabular Forms
* Autocomplete returning key value for Tabular Forms
* Clone Row functionality

## Simple Autocomplete

APEX 4 introduced a new item type called “Text Field with autocomplete”. This is a simple select list generated using jQuery libraries. This type of select list allows you editing and will shows the results as you type. The only thing you need to do is to define a SQL query like this:

SELECT ename

FROM emp

and you are all set and ready to go.

This item type is not available for Tabular Forms. However, there are some examples on the internet from the time before APEX 4. One of those examples I used in this chapter. Tyler Muth posted an example in his blog:

http://tylermuth.wordpress.com/2010/03/16/jquery-autocomplete-for-apex/

We will use some of his code and create an autocomplete select list for the column JOB in our example of wizard generated Tabular Form. Before we do that, we will need to do some preparation work and:

* Download the jquery.autocompleteApex1.2.js available in Tyler’s blog.
* Upload it to our workspace
* Reference it on our page

<script type="text/javascript" src="#WORKSPACE\_IMAGES#jquery.autocompleteApex1.2.js"></script>

Now, we can create some code in our package

tab\_form\_emp\_pkg

We will add the following functions and procedures to our package:

* Procedure for creating a list of jobs based on the EMP table used for the On Demand process.
* Function for generating javascript.

PROCEDURE get\_job;

FUNCTION get\_autocomplete (

p\_item IN VARCHAR2,

p\_rownum IN VARCHAR2,

p\_width IN NUMBER,

p\_process IN VARCHAR2

)

RETURN VARCHAR2;

PROCEDURE get\_job

IS

v\_search emp.job%TYPE;

BEGIN

EXECUTE IMMEDIATE 'alter session set NLS\_SORT=BINARY\_CI';

EXECUTE IMMEDIATE 'alter session set NLS\_COMP=LINGUISTIC';

v\_search := REPLACE (wwv\_flow.g\_x01, '\*', '%');

FOR i IN (SELECT DISTINCT job

FROM emp

WHERE job LIKE '%' || v\_search || '%')

LOOP

HTP.p (i.job);

END LOOP;

END get\_job;

FUNCTION get\_autocomplete (

p\_item IN VARCHAR2,

p\_rownum IN VARCHAR2,

p\_width IN NUMBER,

p\_process IN VARCHAR2

)

RETURN VARCHAR2

IS

v\_rownum VARCHAR2 (20);

v\_item VARCHAR2 (20);

v\_script VARCHAR2 (1000);

BEGIN

v\_rownum := p\_rownum;

v\_item := p\_item || '\_' || v\_rownum;

v\_script :=

'<script type="text/javascript">'

|| '$(document).ready( function() {'

|| '$("#'

|| v\_item

|| '").autocomplete(''APEX'', {'

|| 'apexProcess: '''

|| p\_process

|| ''','

|| 'width: '

|| p\_width

|| ','

|| 'multiple: false,'

|| 'matchContains: true,'

|| 'cacheLength: 1,'

|| 'max: 100,'

|| 'delay: 150,'

|| 'minChars: 1,'

|| 'matchSubset: false'

|| '});'

|| '});'

|| '</script>';

RETURN v\_script;

END get\_autocomplete;

Snippet 3.14: Tabular Form - Autocomple Functions and Procedures

There are only three more things we need to add (change) to get that autocomplete list working:

* We will modify our SQL created for the Tabular Form and include a call to our function

get\_autocomplete

* Create an On Demand process in the Application Processes

get\_job

* Modify the HTML Expression for the affected column JOB

The modified SQL will be:

SELECT empno,

empno

|| tab\_form\_emp\_pkg.get\_autocomplete ('f04', '#ROWNUM#', 85, 'get\_job') empno\_display, ename, job, mgr,

hiredate, sal, comm, deptno, valid

FROM emp

Snippet 3.15: Tabular Form – SQL for Simple Autocomplete

In this SQL we concatenate the return value of the get\_autocomplete function with an existing display only column – EMPNO\_DISPLAY. This function will have four input parameters for:

* The referenced item for which we create this autocomplete list

f04

* Row number for the current row.
* The width in pixel for our select list.
* The name of the corresponding On Demand process.

For the On Demand process – get\_job, we will use a simple PL/SQL block:

BEGIN

tab\_form\_emp\_pkg.get\_job;

END;

Finally, let’s modify the HTML Expression of the Tabular Form column JOB to:

<input type="text" id="f04\_#ROWNUM#" value="#JOB#" maxlength="2000" size="12" name="f04" autocomplete="off" class="ac\_input">

We will also need to change the Column Properties for the column EMPNO\_DISPLAY from “Display as Text…” to “Standard Report Column”.

If we now run the page, edit the JOB column and start typing, we should get a similar result to the one in the following figure:

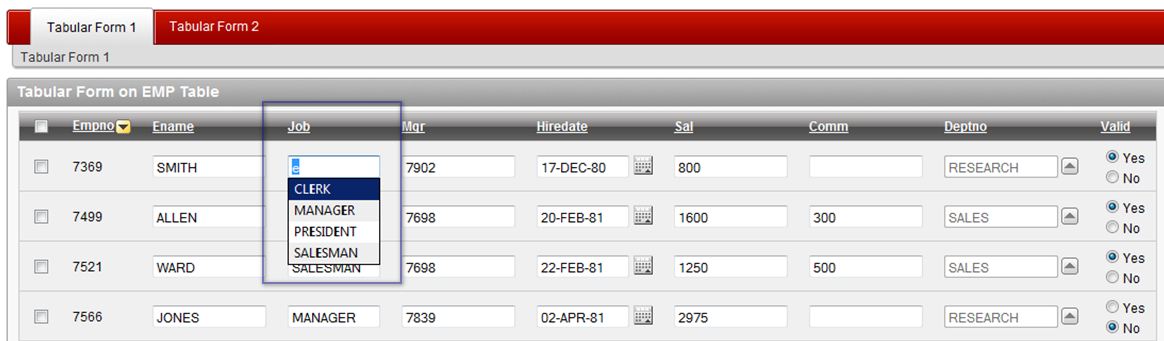


Figure 3.35: Tabular Form – Autocomplete on JOB column

## Autocomplete Returning Key Value

The following example is based on what we examined in the previous one with one addition – we will need to show an autocomplete select list and upon selection return a key value into another column. EMP table has a column MGR showing the manager’s ID. We will create an additional column showing manager names (MGR SEARCH ) and parse the selected manager’s ID into the MGR column.

For this, we will need two additional functions and a procedure in our package:

* Procedure for creating a list of managers based on the EMP table and the selected manager used for the On Demand process.
* Function for generating javascript. This function will also create an additional text item used for the autocomplete list. We will assign the array 50 to it.
* Additional function for returning ENAME from the EMP table for the existing MGR entries in the EMP table.

FUNCTION get\_manager\_for\_id (p\_mgr IN NUMBER)

RETURN VARCHAR2;

PROCEDURE get\_manager;

FUNCTION get\_autocomplete\_key (

p\_value IN VARCHAR2,

p\_key\_item IN VARCHAR2,

p\_rownum IN VARCHAR2,

p\_width IN NUMBER,

p\_process IN VARCHAR2,

p\_max\_length IN NUMBER DEFAULT 80,

p\_size IN NUMBER DEFAULT 80

)

RETURN VARCHAR2;

FUNCTION get\_manager\_for\_id (p\_mgr IN NUMBER)

RETURN VARCHAR2

IS

v\_ename emp.ename%TYPE;

BEGIN

FOR c IN (SELECT ename

FROM emp

WHERE empno = p\_mgr)

LOOP

v\_ename := c.ename;

END LOOP;

RETURN v\_ename;

END get\_manager\_for\_id;

PROCEDURE get\_manager

IS

v\_search VARCHAR2 (255);

v\_output VARCHAR2 (400);

BEGIN

EXECUTE IMMEDIATE 'alter session set NLS\_SORT=BINARY\_CI';

EXECUTE IMMEDIATE 'alter session set NLS\_COMP=LINGUISTIC';

v\_search := REPLACE (wwv\_flow.g\_x01, '\*', '%');

FOR i IN (SELECT ename, empno

FROM emp

WHERE ename LIKE '%' || v\_search || '%'

AND empno IN (SELECT mgr FROM emp))

LOOP

v\_output := i.ename || '|' || i.empno;

HTP.p (v\_output);

END LOOP;

END get\_manager;

FUNCTION get\_autocomplete\_key (

p\_value IN VARCHAR2,

p\_key\_item IN VARCHAR2,

p\_rownum IN VARCHAR2,

p\_width IN NUMBER,

p\_process IN VARCHAR2,

p\_max\_length IN NUMBER DEFAULT 80,

p\_size IN NUMBER DEFAULT 80

)

RETURN VARCHAR2

IS

v\_rownum VARCHAR2 (20);

v\_name VARCHAR2 (5) := 'f50';

v\_item VARCHAR2 (20);

v\_item\_html VARCHAR2 (3000);

v\_key\_item VARCHAR2 (20);

v\_script VARCHAR2 (1000);

v\_output\_item VARCHAR2 (4000);

a\_rownum VARCHAR2 (400);

BEGIN

v\_rownum := p\_rownum;

v\_item := v\_name || '\_' || v\_rownum;

v\_item\_html :=

'<input type="text" id="'

|| v\_item

|| '" value="'

|| p\_value

|| '" maxlength="'

|| p\_max\_length

|| '" size="'

|| p\_size

|| '" name="'

|| v\_name

|| '" '

|| 'autocomplete="off" class="ac\_input"'

|| 'style="width:'

|| p\_width

|| 'px">';

v\_key\_item := p\_key\_item || '\_' || v\_rownum;

v\_script :=

'<script type="text/javascript">'

|| '$(document).ready( function() {'

|| '$("#'

|| v\_item

|| '").autocomplete(''APEX'', {'

|| 'apexProcess: '''

|| p\_process

|| ''','

|| 'width: '

|| p\_width

|| ','

|| 'multiple: false,'

|| 'matchContains: true,'

|| 'cacheLength: 1,'

|| 'max: 100,'

|| 'delay: 150,'

|| 'minChars: 1,'

|| 'matchSubset: false,'

|| 'x02: ''foo'','

|| 'x03: $(''#'

|| v\_key\_item

|| ''').val()'

|| '});'

|| '$("#'

|| v\_item

|| '").result(function(event, data, formatted) {'

|| 'if (data){'

|| '$("#'

|| v\_key\_item

|| '").val(data[1]);'

|| '}'

|| '});'

|| '});'

|| '</script>';

v\_output\_item := v\_item\_html || v\_script;

RETURN v\_output\_item;

END get\_autocomplete\_key;

Snippet 3.16: Tabular Form – SQL for Autocomplete Returning Key Value

After adding this code to our package we can start modifying our Tabular Form SQL. The modified SQL will be:

SELECT empno,

empno

|| tab\_form\_emp\_pkg.get\_autocomplete ('f04', '#ROWNUM#', 85, 'get\_job') empno\_display,

ename, job,

**tab\_form\_emp\_pkg.get\_autocomplete\_key**

**(tab\_form\_emp\_pkg.get\_manager\_for\_id (mgr),**

**'f05',**

**'#ROWNUM#',**

**85,**

**'get\_manager'**

**) mgr\_search,**

mgr, hiredate, sal, comm, deptno, valid

FROM emp

Snippet 3.17: Tabular Form – SQL for Autocomplete Returning Key Value

In this SQL we add the get\_autocomplete\_key function between the columns JOB and MGR. This function will have four input parameters for:

* ENAME of the existing manager returned by the function

get\_manager\_for\_id

* The referenced item in which we will pars the key value

f05

* Row number for the current row.
* The width in pixel for our select list.
* The name of the corresponding On Demand process.

To get the newly created column MGR SEARCH into the right order, we will edit the Tabular Form properties and move the column between the columns JOB and MGR.

For the On Demand process – get\_manager, we will use a simple PL/SQL block:

BEGIN

tab\_form\_emp\_pkg.get\_manager;

END;

Finally, let’s modify the HTML Expression of the Tabular Form column MGR to:

<input type="text" id="f05\_#ROWNUM#" value="#MGR#" maxlength="2000" size="16" name="f05" autocomplete="off">

We will also need to change the Column Properties from “Display as Text…” to “Standard Report Column”.

If we now run the page, edit the MGR SEARCH column and start typing, we should get the list of the managers. If we select one of them, the entry in the column MGR should change to the corresponding EMPNO of the selected ENAME. The result should look similar to the one in the following figure:

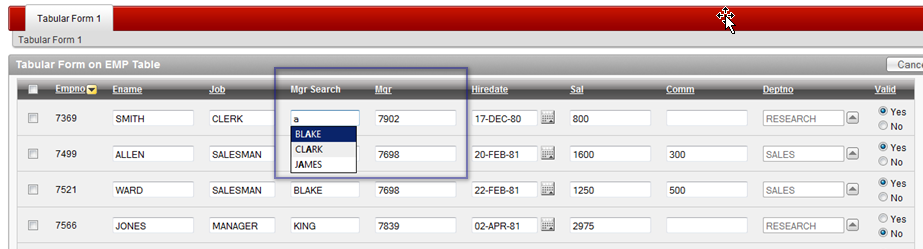


Figure 3.36: Tabular Form – Autocomplete on MGR column – Selecting Value from the List

If we now change the value and select another manager, the MGR column value should change correspondingly:

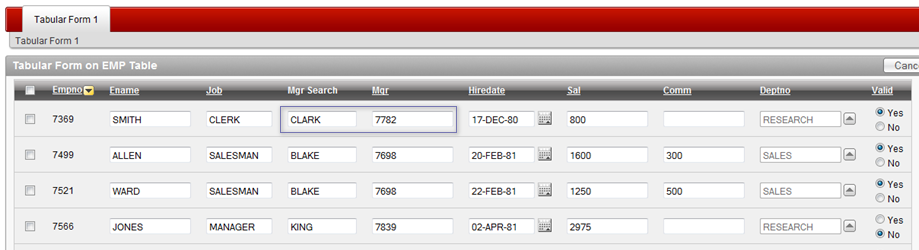


Figure 3.37: Tabular Form – Autocomplete on MGR column – Returned Key Value

We can now hide the MGR column by changing the HTML Expression from type text to type hidden:

<input type="hidden" id="f05\_#ROWNUM#" value="#MGR#" maxlength="2000" size="16" name="f05" autocomplete="off">

and remove the column heading and in the Report Attributes.

CAUTION: Moving the column to the end of the report or simply changing it’s position will cause an error while trying to submit the page. The arrays will change while changing positions. If you still want to do that, undo the changes in the HTML Expression, change the parameters in the function used in the SQL, check your processes, procedures and functions in order to make sure you are referencing the right array and then move the column.

## Clone Rows (in the case you really, really need this)

In one of my recent projects customer had a requirement to easily clone rows in a tabular form. With help of my colleague and friend Linh Dinh

*http://www.dinh.de/wordpress/*

we developed a small piece of code which does that. The goal was to clone one or more rows, change and save them. The idea behind that is quite interesting. Eventually the existing functionality for adding rows could be extended in the way that adding of new rows could be possible at any position in a table and not always at the end. In some cases this could speed up the process of creating of new records. Let us show how that works.

We will need to do couple of things and prepare a new page for a demonstration of this small feature:

* Copy the page 1 to page 4 and remove all of the custom coding we created.
* Change the SQL of the Tabular Form report.
* Add a small javascript to the page header.
* Modify the column holding the image / link for this process.

New (old) SQL for the Tabular Form report on the new page would be:

SELECT empno, NULL clone, empno empno\_display, ename, job, mgr,

hiredate, sal, comm, deptno, valid

FROM emp

We will create a new column CLONE, to hold the image for cloning of the selected row. We will need to move that new column right after the checkbox. We will edit the column properties and put a link in the HTML Expression

<a href="#" onclick="javascript:fn\_CloneRow(this);">

<img src="#IMAGE\_PREFIX#copy.gif" alt=""></a>

We will also change the Column Attributes > Display As > Standard Report Column.

Finaly, we will add a small javascript to the page header:

<script type="text/javascript">

function fn\_delete(pThis)

{

var l\_tr=$x\_UpTill(pThis,'TR');

l\_tr.parentNode.removeChild(l\_tr);

}

function fn\_CloneRow(pThis) {

$(pThis).parent().parent().clone(true,false).insertAfter($(pThis).parent().parent());

newRow = $(pThis).parent().parent().next();

newRow.find('[type=hidden]').val('');

newRow.find('[name=fcs]').val('Z');

newRow.find('[type=checkbox]').remove();

html\_RowHighlight($(newRow).get(0),"#9E0200");

// Delete Cloned Row

newLink = $(newRow).find('img')[0];

newLink.src = '/i/del.gif';

$(newLink).parent().removeAttr('onclick');

$(newLink).parent().get(0).onclick = function() {

fn\_delete(this)};

}

</script>

Now, let’s look at the results.

Snippet 3.18: Tabular Form – Javascript for Cloning Rows

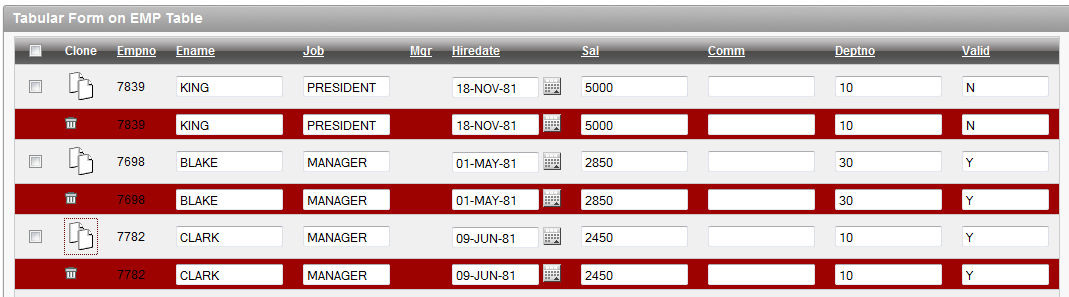


Figure 3.38: Tabular Form – Cloning Rows

This script could easily be extended to create empty rows as well. We could also change the ID’s of the new items (currently those are copied). Please, use this with caution. It is suitable for an Excel-like data management only, where you have a plane grid of text only items and would like to easily multiply the rows without taking much care of the data integrity. I could imagine using it in a combination with collections.

# Wrap Up

Tabular Forms is a one of the greatest features in APEX. If APEX wouldn’t provide this functionality it simply wouldn’t be as successful as it is. There is still a lot of potential to improve this feature and make it even better:

* Multiple Tabular Forms per page.
* Dynamic Actions for Tabular Form elements.
* Increasing the number of arrays.
* More options for adding / copying of rows.
* Item options equal to those of page items

I hope that in this chapter we managed to show some of the options we have when we need to use Tabular Forms and maybe this gave you an idea on how you could improve your applications.

If you would like to inform you further on Tabular Forms, you may have a look at my Demo Application at

http://apex.oracle.com/pls/otn/f?p=31517:1

In the Section VI there are several examples dealing with that topic.