ApexNinjas Whitepaper

Checkboxes in Interactive Reports

By George Bara

WWW.APEXNINJAS.COM

12/2/2011

This is a guide on how to develop applications with checkboxes in APEX 3.x versions

Document history

Date	Author	Version	Modifications
12/02/2011	George Bara	1.0	Created this document

Contents

Initial Considerations

One of the minuses of the APEX framework (at least until Apex 4.0) is the lack of adding checkboxes to reports, whether SQL or Interactive Reports (further called IR). Even if there is a workaround to this problem, it becomes serious when working with a large amount of data and even more serious when trying to implement checkboxes in interactive reports.

Simple checkbox creation with APEX_ITEM

Fortunately, APEX provides us with an assist package: APEX_ITEM. This package provides us with an API for creating almost every kind of objects: select lists, radio buttons, radiogroup, textareas, date pickers and checkboxes.

Let's create a simple table:

```
create table CHK_TEST(
 TEST_ID number(3),
 TEST_INFO varchar2(100),
 TEST_DT   date,
 SELECTED number(1));
```

Creating a checkbox is possible using the CHECKBOX function:

```
APEX_ITEM.CHECKBOX(
    p_idx                       IN    NUMBER,
    p_value                     IN    VARCHAR2 DEFAULT,
    p_attributes                IN    VARCHAR2 DEFAULT,
    p_checked_values            IN    VARCHAR2 DEFAULT,
    p_checked_values_delimiter  IN    VARCHAR2 DEFAULT)
    RETURN VARCHAR2;
```

The usage of this function can be like in the following example, in a SQL Query Report Region:

```
select
 apex_item.checkbox(1,TEST_ID,'UNCHECKED') " ",
 TEST_ID,
 TEST_INFO,
 TEST_DT,
 SELECTED
from CHK_TEST
```

This will create a column displaying an unchecked checkbox, having the value of the TEST_ID column. This is the easiest way a checkbox column is created, but it lacks the most basic of functionalities:

1. Loading the checkboxes at page refresh, using the SELECTED column
2. Marking the table row as checked or unchecked (SELECTED column 1 or 0)
3. Widgets to select or de-select of the rows

Checkbox functionalities

Loading the checkboxes state (checked/unchecked)

Previously we created a SQL Report region with a checkbox, as in the image:

Test Report

	TEST_ID	TEST_INFO	TEST_DT	SELECTED
☐	1	test 1	01.12.2010	0
☐	2	test doi	02.12.2010	0
☐	3	test tr3i	03.12.2010	0
☐	4	jane	04.12.2010	0
☐	5	joe	05.12.2010	0
☐	6	jack	14.12.2010	0
☐	7	dick	17.12.2010	0
☐	8	dan	25.12.2010	0
☐	9	diane	20.12.2010	0
☐	10	arnold	20.12.2010	0
☐	-	-	20.12.2010	0

1 - 11

The current issue is that we created the checkboxes with a value (TEST_ID), but unchecked and with no link to the SELECTED column, which was created exactly for this purpose.

There are two ways of doing this:

1. **By using the p_atributes parameter from the `APEX_ITEM.CHECKBOX` function:**

```
select
 apex_item.checkbox(1,TEST_ID,
                    decode(SELECTED,1,'checked','unchecked')
                   )
 TEST_ID,
 TEST_INFO,
 TEST_DT,
 SELECTED
from CHK_TEST
```

   Basically, we just added an HTML checkbox attribute in a dynamic way, using a decode on the SELECTED column value. This way, the checkbox item will always be checked the SELECT=1 and unchecked otherwise.

   This is a clean-cut solution and it's usage is recommended especially when we have a large number of rows in the table (more tha 500, 1000, etc). Still, the process of saving the checkbox values in the table will prove more difficult when using Interactive Reports.

2. **By using an APEX built-in functionality, of storing the checkbox values "the APEX way", in a colon separated string.**

Even when creating checkboxes or radiobutton groups as discting items on a APEX page, their values will be stored like this: ***itemvalue1:itemvalue2:itemvalue4****. The values checked will always be part of this string, the values missing indicating the fact that it was not checked. Example:*

APEX has built this functionality inside `APEX_ITEM.CHECKBOX` function:

```
APEX_ITEM.CHECKBOX(
        p_idx                       IN    NUMBER,
        p_value                     IN    VARCHAR2 DEFAULT,
        p_attributes                IN    VARCHAR2 DEFAULT,
        p_checked_values            IN    VARCHAR2 DEFAULT,
        p_checked_values_delimiter  IN    VARCHAR2 DEFAULT)
RETURN VARCHAR2;
```

We can store the rows IDs that were checked in the p_checked_values parameters and the delimiter in the p_checked_values_delimiter, like this:

a. Create an application item named F_TEST_LIST
   Write the SQL Report source like this:

```
select
 apex_item.checkbox(1,TEST_ID,
                    'onchange="spCheckChange(this);"',
                    :F_TEST_LIST,
                    ':'
                   )
 TEST_ID,
 TEST_INFO,
 TEST_DT,
 SELECTED
from CHK_TEST
```

This version will load the checkbox status from the F_TEST_LIST item, which contains the selected rows' TEST_ID delimited by colons: *3:5:6:10.* The loading of this application item will be done from the *spCheckChange* JavaScript function and will be discussed later.
The loading of the the F_TEST_LIST item can be done via an "On load – Before Header" process:

```
Begin
Select stragg(TEST_ID)
Into :F_TEST_LIST
```

```
From CHK_TEST
Where SELECTED = 1;
End;
```

It was presumed that you are familiar with Tom Kytes rows-to-string **stragg** function and that this function was modified/customized to use the ":" delimiter.

The advantage of this version is that the selected rows will already be stored inside the F_TEST_LIST item, for further processing (saving etc). The disadvantages of this version are lots:

- **A custom process to load the values from the table into the F_TEST_LIST item**
- **Custom on-demand process to remove the unchecked rows and add the checked rows into F_TEST_LIST**
- **If there are too many selected rows in the table, the F_TEST_LIST will become larger that 4000 character, resulting in serious errors.**

Test Report

TEST_ID	TEST_INFO	TEST_DT	SELECTED
☐	test 1	01.12.2010	0
☐	test doi	02.12.2010	0
☑	test tr3i	03.12.2010	1
☐	jane	04.12.2010	0
☑	joe	05.12.2010	1
☑	jack	14.12.2010	1
☐	dick	17.12.2010	0
☐	dan	25.12.2010	0
☐	diane	20.12.2010	0
☑	arnold	20.12.2010	1
☐	-	20.12.2010	0

1 - 11

Saving the checkbox state #1

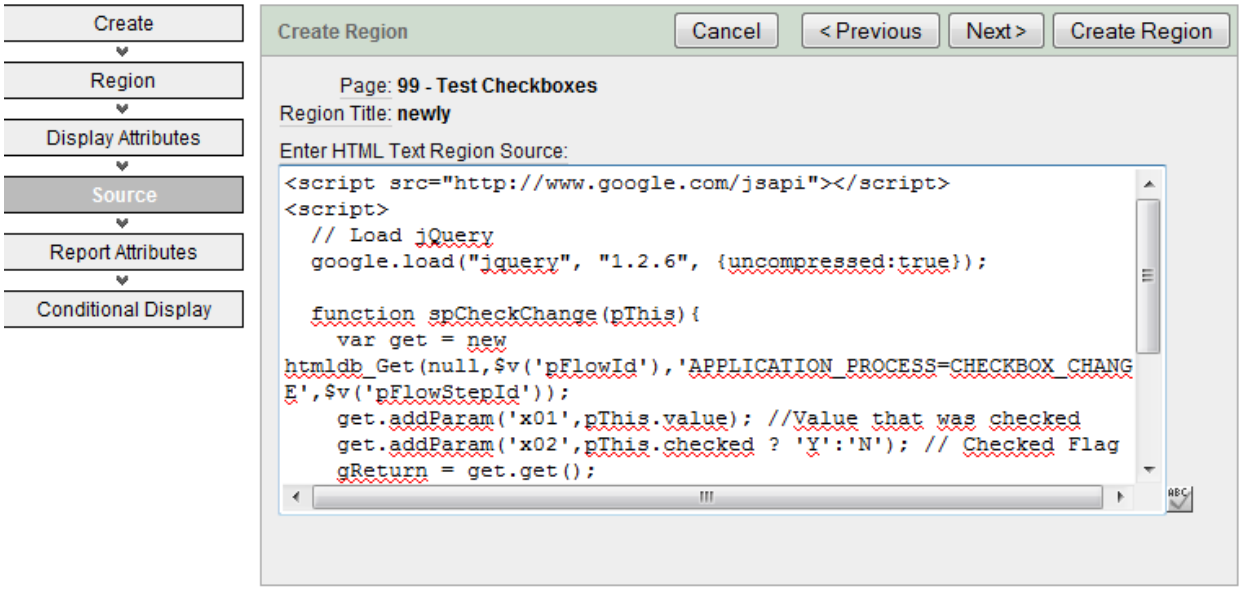
We will first demonstrate the second version for checkbox functionalities, using an application item to store the IDs of the selected rows. This demonstration is done purely for detailing the pros and cons of this approach and the things we can learn from it.

In the previous chapter, we demonstrated the checkboxes state load from the application item. But how do we store the IDs inside the item?

```
select apex_item.checkbox(1,TEST_ID,
```

```
                                'onchange="spCheckChange(this);"',
                                :F_TEST_LIST,
                                ':'
from CHK_TEST;
```

The *onchange* statement will trigger the javascript function called *spCheckChange*. The code of this javascript function will be added in the region source of a newly created HTML region:




```
<script src="http://www.google.com/jsapi"></script>
<script>
  // Load jQuery
  google.load("jquery", "1.2.6", {uncompressed:true});

  function spCheckChange(pThis){
    var get = new
htmldb_Get(null,$v('pFlowId'),'APPLICATION_PROCESS=CHECKBOX_CHANGE',$v('pFlow
StepId'));
    get.addParam('x01',pThis.value); //Value that was checked
    get.addParam('x02',pThis.checked ? 'Y':'N'); // Checked Flag
    gReturn = get.get();

    $x('checkListDisp').innerHTML=gReturn;
  }
</script>
CHECKBOX List:
<div id="checkListDisp">&F_TEST_LIST.</div>
```

This JavaScript code will do the following:

1. It defines the function **spCheckChange**, which is called on the checkbox definition (onchange=”spCheckChange(this)”)
2. It creates a HTML area where the application item **F_TEST_LIST** is displayed.

These two characteristics generate a dynamic region, where every change of a checkbox status (check/uncheck) will trigger the javascript function which alters the content of the application item F_TEST_LIST (which stores the checkboxes' ID separated by colon) and displays it on the screen as soon as the value is changed, without the need of a page refresh.

The JS function SpCheckChange calls an **on demand application process** called CHECKBOX_CHANGE:

```
-- Application Process: CHECKBOX_CHANGE
-- On Demand...

DECLARE
  v_item_val NUMBER := apex_application.g_x01;
  v_checked_flag VARCHAR2 (1) := apex_application.g_x02;
BEGIN
  IF v_checked_flag = 'Y' THEN
    -- Add to the list
    IF :F_TEST_LIST IS NULL THEN
      :F_TEST_LIST:= ':' || v_item_val || ':';
    ELSE
      :F_TEST_LIST:= :F_TEST_LIST || v_item_val || ':';
    END IF;
  ELSE
    -- Remove from the list
    :F_TEST_LIST:= REPLACE (:F_TEST_LIST, ':' || v_item_val || ':', ':');
  END IF;

  -- Just for testing
  HTP.p (:F_TEST_LIST);
END;
```

This on-demand process is called by the javascript function **spCheckChange** every time a checkbox is ticked. It simply adds or removes the checkbox ID from the F_TEST_LIST list. This application item is the base for the checkbox status, so every time the list is modified, the status of every checkbox is saved. Finally, the F_TEST_LIST item is used to save the status of the checkboxes in the CHK_TEST table, column SELECTED.

A simple PLSQL submit process can do this:

```
Declare
 vOccurences number;
Begin
   Update CHK_TEST set SELECTED = 0 where SELECTED = 1; // a trigger that
checks different old - new values could prove a plus

vOccurences := occurencesinString(F_TEST_LIST,':'); //get the number of
items in list

for I in 1.. vOccurences loop
  vCheckBoxId := getItemValuebyPosition(F_TEST_LIST,i);
  update CHK_TEST set SELECTED = 1 where TEST_ID = vCheckBoxId;
```

```
end loop;
End;
```

This approach has three benefits:.

First, the function spCheckChange can be used to dynamically print values from the table (sums or other calculations), as ticking on the checkboxes, with no refresh to the page.

Second, when saving the status of the checkboxes, we already have stored the checked values in the application item.

Third, when using Interactive Reports, it's much safer to make the table updates with the T_TEST_LIST string, instead of using the default *APEX_APPLICATION.G_F01* item, which does basically the same thing. **The key is when filtering the interactive report: our T_TEST_LIST will continue to store the correct checkboxes ID's, while APEX's `APEX_APPLICATION.G_F01` will store only the selected checkboxes on the filtered report, making non-visible information subject of wrong updates!**

But, the *cons* is significant: if having a checkbox ID made up of 10 digits, we care easily make T_TEST_LIST item larger than 4000 characters, resulting in an error.

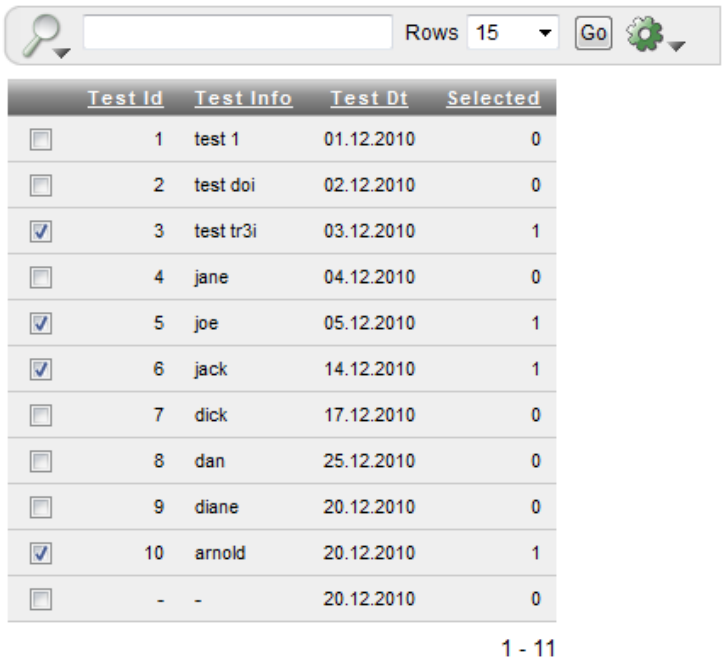
Saving the checkbox state #2

The more robust approach of managing the checkbox states and saving this information in the source table is using the **APEX_APPLICATION.G_F01** item. By default, using the APEX_ITEM package, one can define up to 50 custom created items. When we created our checkbox, we assigned it an APEX array called G_F01:

```
apex_item.checkbox(1,TEST_ID,'UNCHECKED') " ",
```

The first parameter creates this array, naming it G_F01. This array will store all the checkboxes that are selected, by default. Every time we tick a checkbox, the item value is added or removed from the APEX_APPLICATION.G_F01 array. APEX does automatically what we have done manually above with the T_TEST_LIST application item. But there is a serious problem when using this with interactive reports. Consider the following interactive report:

```
select
 apex_item.checkbox(1,TEST_ID,
                    decode(SELECTED,1,'CHECKED','UNCHECKED')
                   ) " ",
 TEST_ID,
 TEST_INFO,
 TEST_DT,
 SELECTED
from CHK_TEST
```



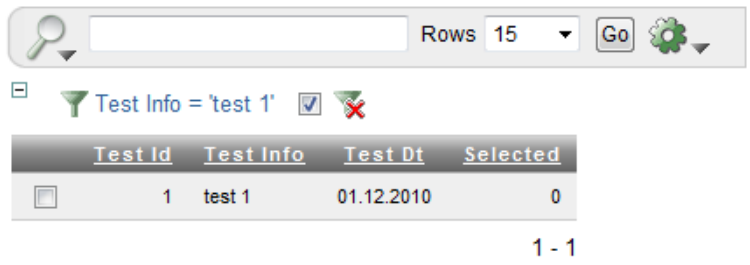
Without any filter applied to the IR, the APEX_APPLICATION.G_F01 has the values:

APEX_APPLICATION.G_F01 (1) = 3
APEX_APPLICATION.G_F01 (2) = 5
APEX_APPLICATION.G_F01 (3) = 6
APEX_APPLICATION.G_F01 (4) = 9

So, we can use these values to save the changes in the CHK_TEST table, updating every line to 0 (SELECTED = 0) and then updating the lines from the APEX_APPLICATION.G_F01 array to 1 (SELECTED = 1), like this:

```
Update CHK_TEST set SELECTED = 0 where SELECTED = 1;
For I in 1.. APEX_APPLICATION.G_F01.COUNT
  loop
       UPDATE CHK_TEST set SELECTED = 1
       where TEST_ID = APEX_APPLICATION.G_F01(i);
  End loop;
```

But, if we filter the IR like this:



Although we didn't change the checkboxes status, the APEX_APPLICATION.G_F01 array will be blank, because it relates to the filtered IR, not to all the rows in the IR default query! So, an update procedure like the one above will not work, deselecting all the rows from the table.

The only information that is missing is the IR filter. This can be easily obtained using the APEX_IR_QUERY package designed by Stewart Stryker (listed at the end of this document) or making a direct query:

```
SELECT condition_column_name,
                              condition_operator,
                              condition_expression,
                              condition_expression2
FROM apex_application_page_ir_cond cond
JOIN apex_application_page_ir_rpt r ON r.application_id = cond.application_id
                                   AND r.page_id = cond.page_id
                                   AND r.report_id = cond.report_id
WHERE cond.application_id = app_id_in
       AND cond.page_id = page_id_in
       AND cond.condition_type = 'Filter'
       AND cond.condition_enabled = 'Yes'
       AND r.base_report_id = base_report_id_in
       AND r.session_id = session_id_in
```

This will get the where clause of the current Interactive Report. The information necessary for obtaining the filter is:

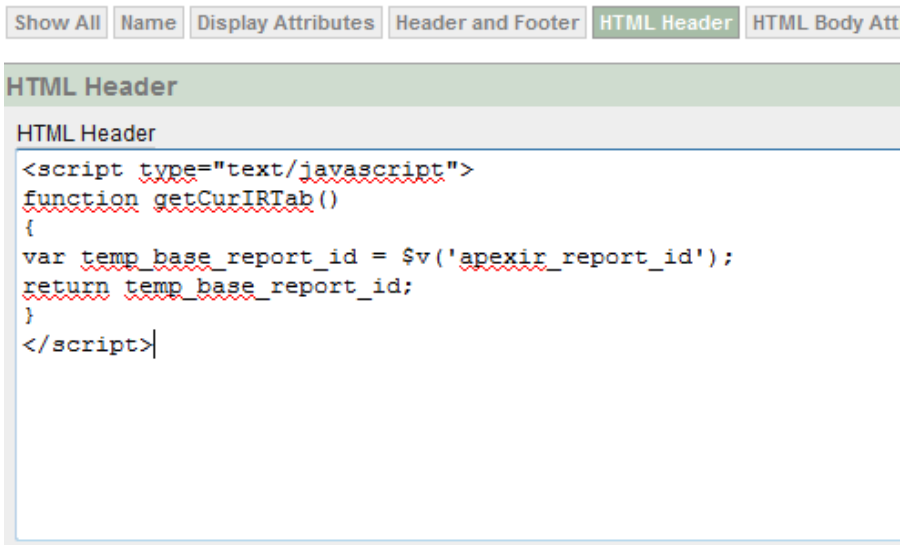
1. The application ID
2. The Page ID
3. The Session ID
4. The Base Report ID

The first 3 are easy to get and quite straightforward. The Base Report ID information is a bit tricky! This is how we get it:

Each APEX Interactive report is assigned a unique id and stores it in a hidden item called **apexir_report_id**. There is only one item of this king inside an APEX page, because there can be only one IR per page. The value of this item is needed to get the current user filter on the IR.

To get the value, use the Javascript code (pasted in the page HTML header):

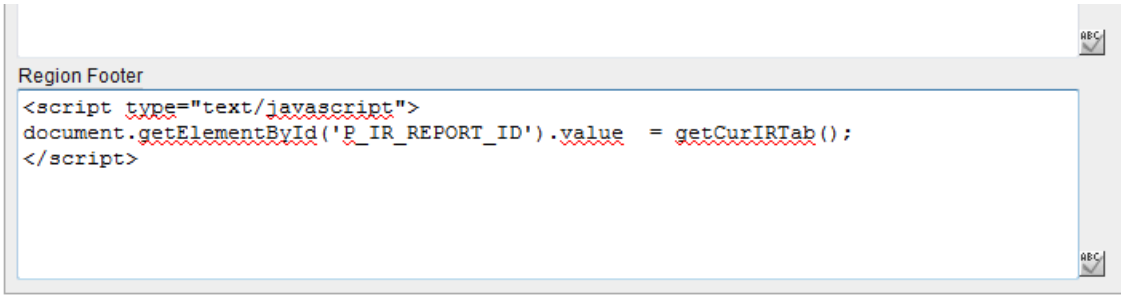
```
<script type="text/javascript">
function getCurIRTab()
{
var temp_base_report_id = $v('apexir_report_id');
return temp_base_report_id;
}
</script>
```



Now, create a hidded (just hidden, not protected!) item called P_IR_REPORT_ ID inside a HTML region. **Important! This region has to be rendered <u>after</u> the Interactive Report, so create it in the "after footer" display point!**

In this regions's footer paste the following JavaScript code, that will assign the hidden item P_IR_REPORT_ID the value of the Interactive Report ID.

```
<script type="text/javascript">
document.getElementById('P_IR_REPORT_ID').value  = getCurIRTab();
</script>
```



The only thing left is to save the checkboxes' state inside the table. Using the APEX_IR_QUERY package, we can get the where clause for the IR filters. The code:

```
apex_ir_query.ir_query_where(:APP_ID,  -- application id
                             99, --page id
                             :SESSION, --sesion ID
                             :P_IR_REPORT_ID -interactive report ID
                             );
```

Could return something like "*where TEST_INFO='test1'*" and can be appended to the SQL statement.

The process that does the table update could look like this:

```
declare
  whereStmt varchar2(2000);
  sqlStmt varchar2(2000);
begin
```

```
whereStmt := apex_ir_query.ir_query_where(:APP_ID,
                              99,
                              :SESSION,
                              :P_IR_REPORT_ID);

sqlStmt := 'update CHK_TEST set SELECTED = 0 where SELECTED = 1 '||
whereStmt;

execute immediate sqlStmt;

if APEX_APPLICATION.G_F01.COUNT>0 then
      FOR i in 1..APEX_APPLICATION.G_F01.COUNT LOOP
            update CHK_TEST
            set VALID_LINE = 1
            where SELECTED_ID = APEX_APPLICATION.G_F01(i)
                  and VALID_LINE = 0;

      END LOOP;
end if;

end;
```

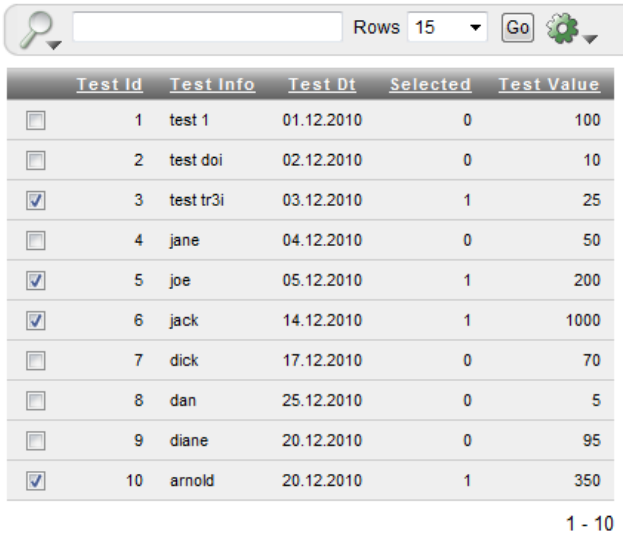
This way of updating the source table with the checkbox state has the big advantage of storing the ID's of the checked lines inside the default APEX array and is not limited in dimension like a standard application item. It could be that, due to many filters applied on the IR, the string that stores the where clause of the filters can become bigger that 4000 character and result in errors, but this is not likely.

Dynamic sum on checkbox change. Test scenario

Let's add a value column to our table:

```
alter table CHK_TEST add TEST_VALUE number(4);
```

and also add some values:



It would be very nice to have the following functionalities on this Interactive Report:

1. Full checkboxes functionalities (load checkbox state depending on SELECTED column, save the state inside the table even if IR is filtered)
2. Dynamically display the sum of TEXT_VALUE column when we check/uncheck the checkboxes.

The first point was solved above, in the Saving the checkbox state #2 paragraph. For the second point, we will use some of the code described in the the Saving the checkbox state #1 paragraph.

Step 1

Rewrite the IR SQL statement:

```
select
 apex_item.checkbox(1,TEST_ID,
                    'onclick="spCheckChange(this);"
'||decode(t.SELECTED,1,'checked','unchecked')
                   ) " ",
 TEST_ID,
 TEST_INFO,
 TEST_DT,
 SELECTED,
 TEST_VALUE
from CHK_TEST;
```

This will keep the checkboxes selected/unselected depending of the SELECTED column value, and also call the **spCheckChange** javascript function.

Step 2

```
Create a HTML region where the sum will be displayed and paste the following
code inside the region source:
<script src="http://www.google.com/jsapi"></script>
<script>
  // Load jQuery
  google.load("jquery", "1.2.6", {uncompressed:true});

  function spCheckChange(pThis){
    var get = new
htmldb_Get(null,$v('pFlowId'),'APPLICATION_PROCESS=CHECKBOX_CHANGE',$v('pFlow
StepId'));
    get.addParam('x01',pThis.value); //Value that was checked
    get.addParam('x02',pThis.checked ? 'Y':'N'); // Checked Flag
    gReturn = get.get();

    $x('checkListDisp').innerHTML=gReturn;
  }
</script>
The sum:
<div id="checkListDisp">&F_TEST_SUM.</div>
```

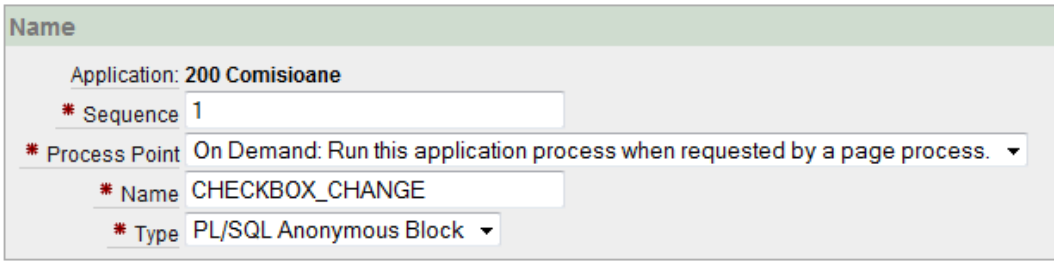
Create a “before header” process that loads the sum of the checked lines in F_TEST_SUM:

```
begin
select sum(TEST_VALUE)
   into :F_TEST_SUM
```

```
   from CHK_TEST
   where SELECTED = 1;
end;
```

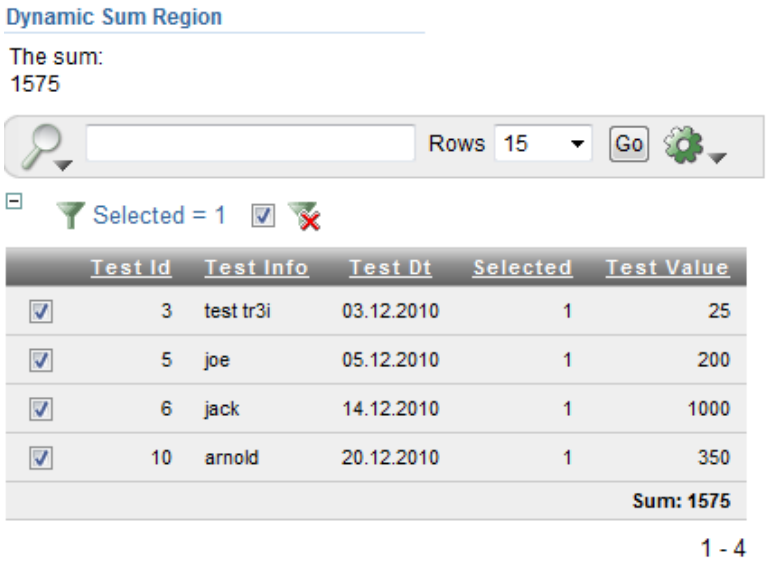
Step 3

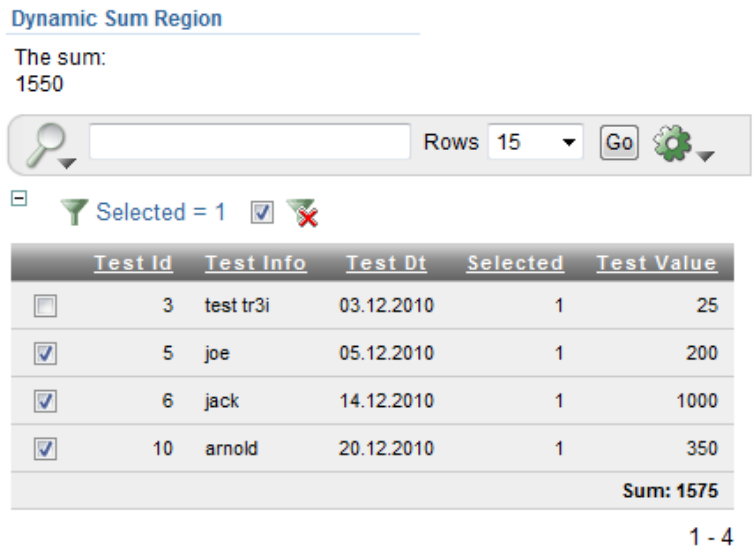
Create the **F_TEST_SUM** application item and the **CHECKBOX_CHANGE** "on demand" application process:




```
-- Application Process: CHECKBOX_CHANGE
-- On Demand...

DECLARE
  v_item_val NUMBER := apex_application.g_x01;
  v_checked_flag VARCHAR2 (1) := apex_application.g_x02;
  v_ttlsum number;
  v_sum number;
BEGIN


   select TEST_VALUE
   into v_sum
   from CHK_TEST
   where TEST_ID = v_item_val;


  IF v_checked_flag = 'Y' THEN
    :F_TEST_SUM := :F_TEST_SUM + v_sum;
  ELSE
   :F_TEST_SUM := :F_TEST_SUM - v_sum;

  END IF;

  HTP.p (:F_TEST_SUM);

END;
```

This process will be called every time a checkbox is ticked (by spCheckChange javascript function) and will return the sum of the selected lines. The region defined at Step 2 will load the sum (selecting TEST_VAL for SELECTED = 1) at page refresh from CHK_TEST and then will dynamically update the sum, when the checkboxes are ticked/unticked.



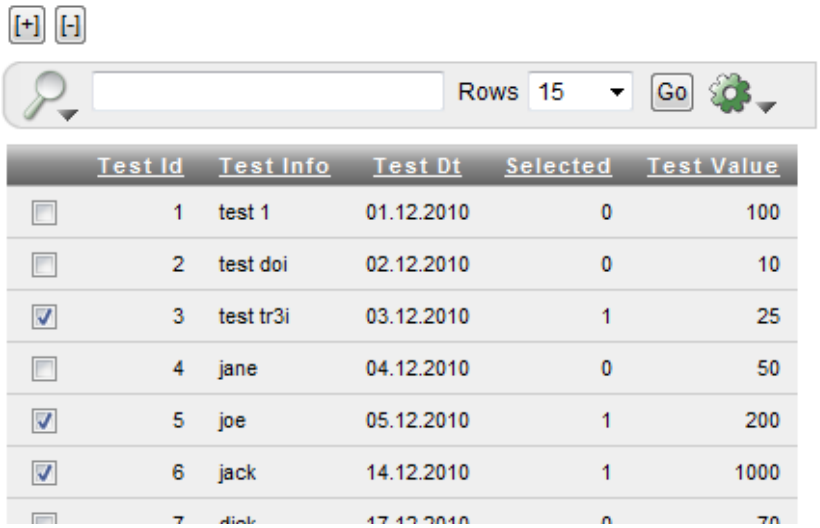


Select/unselect all

Another functionality that is a must, especially when working with many rows, is a select/unselect all. This can be done by creating an item situated in the IR region, located above the region. Create the item as **display as text**,without any label. Then, in the **Pre Element Text** region, paste this code:

```
<input type="button" name="CheckAll" value="[+]"
onClick="checkAll(f01)">
<input type="button" name="CheckAll" value="[-]"
onClick="uncheckAll(f01)">
```

This will add the two buttons on your screen:



	Test Id	Test Info	Test Dt	Selected	Test Value
☐	1	test 1	01.12.2010	0	100
☐	2	test doi	02.12.2010	0	10
☑	3	test tr3i	03.12.2010	1	25
☐	4	jane	04.12.2010	0	50
☑	5	joe	05.12.2010	1	200
☑	6	jack	14.12.2010	1	1000

Now, add the two javascript functions inside you page HTML Header :

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Begin
function checkAll(field)
{
for (i = 0; i < field.length; i++)
      {field[i].checked = true;}
}

function uncheckAll(field)
{
for (i = 0; i < field.length; i++)
      {field[i].checked = false ;}
}
</script>
```

These functions will be called when clicking the two buttons created before and will check/uncheck all of the checkboxes from the IR.

Check/uncheck all and calculate sum

If you want also to dynamically calculate the sum when checked/unchecked all, modify the functions like this:

```
<SCRIPT LANGUAGE="JavaScript">
<!-- Begin
function checkAll(field)
{
for (i = 0; i < field.length; i++)
      {field[i].checked = true;}

var get = new
htmldb_Get(null,$v('pFlowId'),'APPLICATION_PROCESS=CheckAll',$v('pFlowStepId'
));
```

```
gReturn = get.get();
$x('checkListDisp').innerHTML=gReturn;
}

function uncheckAll(field)
{
for (i = 0; i < field.length; i++)
      {field[i].checked = false ;}

var get = new
htmldb_Get(null,$v('pFlowId'),'APPLICATION_PROCESS=UnCheckAll',$v('pFlowStepI
d'));
gReturn = get.get();
$x('checkListDisp').innerHTML=gReturn;
}
```

Then, create the CheckAll and UncheckAll application processes, with the following code:

```
-- Application Process: CheckAll
-- On Demand...
DECLARE
  v_item_val NUMBER := apex_application.g_x01;
  v_checked_flag VARCHAR2 (1) := apex_application.g_x02;
  v_sum number;
BEGIN

   select sum(TEST_VAL)
   into v_sum
   from CHK_TEST;


   :F_TEST_SUM:= v_sum;

  HTP.p (:F_TEST_SUM);

END;
```

```
-- Application Process: UnCheckAll
-- On Demand...
DECLARE
BEGIN

 :F_TEST_SUM:= 0;
 HTP.p (:F_TEST_SUM);

END;
```

APEX_IR_QUERY package

Spec

```
CREATE OR REPLACE PACKAGE apex_ir_query IS

  -- Author  : STEWART_L_STRYKER
  -- Created : 2/13/2009 4:00:45 PM
  -- Purpose : Method of collecting user query from Interactive Report

  -- GRANT EXECUTE ON apex_ir_query to <apex_schema>;

  /* ----------- PUBLIC FUNCTION AND PROCEDURE DECLARATIONS ----------- */

    /* Setup:

       Add your own version of the following Javascript function to the application page
       that contains the button to run your custom report.
       In this example, I'm popping up a new page (#7) to display a PDF report.  The important
       thing is to pass the apexir_report_id value to a page item via a parameter.

        <script language="JavaScript" type="text/javascript">
        function SaveAndRunReport() {
             popUp2('f?p=&APP_ID.:7:&SESSION.:new_request:NO::'+
                    'P7_IR_REPORT_ID:'+$v('apexir_report_id'), 1024, 768);
        }
</script>

        I calculate this WHERE clause, then add it to a fixed query string that our interactive
        reports run against.

        It supports Filters, Searches and Saved reports.
        It DOES NOT support saving the Sort order.

    */
    -- Generate IR query string
    FUNCTION ir_query_where(app_id_in IN NUMBER,
                            page_id_in IN NUMBER,
                            session_id_in IN NUMBER,
                            base_report_id_in IN VARCHAR2) RETURN VARCHAR2;

END apex_ir_query;
```

Body

```
CREATE OR REPLACE PACKAGE BODY apex_ir_query IS

  /* ----------- PRIVATE FUNCTION AND PROCEDURE DECLARATIONS ----------- */
    FUNCTION string_is_valid_date(string_in IN VARCHAR2, format_in IN VARCHAR2 DEFAULT
'MM/DD/YYYY')
        RETURN BOOLEAN IS
        v_date DATE;
    -- From Kai-Joachim Kamrath @ columbia.edu
    BEGIN
        IF string_in IS NULL
        THEN
            RETURN FALSE;
        ELSE
            v_date := to_date(string_in, format_in);
```

```
        RETURN TRUE;
    END IF;
 EXCEPTION
    WHEN OTHERS THEN
        RETURN FALSE;
 END string_is_valid_date;

FUNCTION string_is_valid_number (string_in IN VARCHAR2)
   RETURN BOOLEAN
IS
   l_dummy       NUMBER;
   l_is_number   BOOLEAN DEFAULT FALSE;
BEGIN
   IF string_in IS NOT NULL
   THEN
      l_dummy := TO_NUMBER (string_in);
      l_is_number := TRUE;
   END IF;

   RETURN l_is_number;
EXCEPTION
   WHEN OTHERS
   THEN
      RETURN FALSE;
END string_is_valid_number;

 -- Generic routine to log usage to your favorite log table
 -- You'll need to write your own version, as this is simply a place-holder for the
 -- installation-specific version I call in production
 PROCEDURE log_apex_access(app_name_in IN VARCHAR2,
                           app_user_in IN VARCHAR2,
                           msg_in IN VARCHAR2) IS
 BEGIN
    NULL;
 END log_apex_access;

 -- Convert Apex Interactive Report filter values to query clause
 FUNCTION ir_query_parse_filter(col_name IN VARCHAR2,
                          col_operator IN VARCHAR2,
                          exp1 IN VARCHAR2,
                          exp2 IN VARCHAR2) RETURN VARCHAR2 IS
    result_clause VARCHAR2(512);
    l_col_name VARCHAR2(32) := TRIM(col_name);
    l_col_operator VARCHAR2(32) := LOWER(TRIM(col_operator));
    l_expr1 VARCHAR2(512) := REPLACE(TRIM(exp1), '''', '''''');
    l_expr2 VARCHAR2(512) := REPLACE(TRIM(exp2), '''', '''''');
    in_list VARCHAR2(512);
    in_table wwv_flow_global.vc_arr2;

    FUNCTION wrap_expr(expr_in IN VARCHAR2) RETURN VARCHAR2 IS
        c_date_fmt CONSTANT VARCHAR2(32) := 'YYYYMMDD';
        l_expr VARCHAR2(512) := TRIM(expr_in);
        l_short_date VARCHAR2(8) := SUBSTR(l_expr, 1, 8);
    BEGIN
        IF string_is_valid_date(l_short_date, c_date_fmt)
        THEN
            RETURN 'TO_DATE(''' || l_short_date || ''', ''' || c_date_fmt || ''')';
        ELSIF string_is_valid_number(l_expr)
        THEN
            RETURN l_expr;
        ELSE
            RETURN '''' || l_expr || '''';
```

```
            END IF;
        END wrap_expr;

        -- For "in the last/next" date comparisons
        FUNCTION calc_time_diff(operator_in IN VARCHAR2,
                                value_in IN VARCHAR2,
                                diff_expr IN VARCHAR2) RETURN VARCHAR2 IS
            ret_value VARCHAR2(60);
            factor VARCHAR2(32);
        BEGIN
            factor := CASE LOWER(diff_expr)
                          WHEN 'minutes' THEN value_in || ' / 1440'
                          WHEN 'hours' THEN value_in || ' / 24'
                          WHEN 'days' THEN value_in
                          WHEN 'weeks' THEN value_in || ' * 7'
                          WHEN 'months' THEN value_in || ' * 30'
                          WHEN 'years' THEN value_in || ' * 365'
                          END;
            ret_value := CASE operator_in
                          WHEN 'is in the last' THEN '>= SYSDATE - (' || factor || ')'
                          WHEN 'is in the next' THEN '>= SYSDATE + (' || factor || ')'
                          WHEN 'is not in the last'
                               THEN 'NOT BETWEEN (SYSDATE - ' || factor || ') AND SYSDATE'
                          WHEN 'is not in the next'
                               THEN 'NOT BETWEEN SYSDATE AND (SYSDATE + ' || factor || ')'
                          END;
            RETURN ret_value;
        END calc_time_diff;

    BEGIN
        CASE
            WHEN l_col_operator IN ('not in', 'in') THEN
                in_table := apex_util.string_to_table(l_expr1, ',');

                IF in_table.COUNT > 0
                THEN
                    FOR i IN 1 .. in_table.COUNT
                    LOOP
                        in_table(i) := TRIM(in_table(i));

                        in_table(i) := wrap_expr(in_table(i));
                    END LOOP;
                END IF;
                in_list := apex_util.table_to_string(in_table, ',');
                result_clause := l_col_name || ' ' || l_col_operator || ' (' || in_list || ')';
            WHEN l_col_operator IN ('is null', 'is not null') THEN
                result_clause := l_col_name || ' ' || l_col_operator;
            WHEN l_col_operator IN ('not like',
                                    'like',
                                    '=',
                                    '>=',
                                    '<=',
                                    '<',
                                    '>',
                                    '!=') THEN
                result_clause := l_col_name || ' ' || l_col_operator || ' ' ||
wrap_expr(l_expr1);
            WHEN l_col_operator IN ('regexp_like') THEN
                result_clause := l_col_operator || '(' || l_col_name || ', ' ||
wrap_expr(l_expr1) || ')';
            WHEN l_col_operator = 'contains' THEN
                result_clause := 'INSTR(UPPER(' || l_col_name || '), UPPER(''' || l_expr1 ||
```

```
''')) > 0';
            WHEN l_col_operator = 'does not contain' THEN
                result_clause := 'INSTR(UPPER(' || l_col_name || '), UPPER(''' || l_expr1 ||
''')) = 0';
            WHEN l_col_operator = 'between' THEN
                result_clause := l_col_name || ' ' || l_col_operator || ' ' || wrap_expr(l_expr1)
||
                                 ' AND ' || wrap_expr(l_expr2);
            WHEN l_col_operator LIKE 'is %in the %' THEN
                result_clause := l_col_name || ' ' ||
                                 calc_time_diff(l_col_operator, l_expr1, l_expr2);
            ELSE
                DBMS_OUTPUT.PUT_LINE('Unknown operator: ' || l_col_operator || ']');
        END CASE;

        RETURN result_clause;
    END ir_query_parse_filter;

    -- Convert Apex Interactive Report filter values to query clause
    FUNCTION ir_query_parse_search(col_name_list IN VARCHAR2,
                                   expr_in IN VARCHAR2,
                                   app_id_in IN NUMBER,
                                   ir_id_in IN VARCHAR2 DEFAULT NULL) RETURN VARCHAR2 IS
        crlf CONSTANT VARCHAR2(2) := CHR(10);

        result_clause VARCHAR2(2000);
        l_expr VARCHAR2(512);
        col_name_table wwv_flow_global.vc_arr2;
        col_count PLS_INTEGER := 0;
        match_count PLS_INTEGER := 0;

        FUNCTION wrap_expr(exp_in IN VARCHAR2) RETURN VARCHAR2 IS
            l_expr VARCHAR2(512) := TRIM(exp_in);
        BEGIN
            RETURN '''' || l_expr || '''';
        END wrap_expr;

        -- limit searches to String and numbers and exclude bogus columns
        -- which can be generated by Apex
        FUNCTION right_col_type(col_name_in IN VARCHAR2) RETURN BOOLEAN IS
            found_col VARCHAR2(32);
        BEGIN
            SELECT col.column_type
              INTO found_col
              FROM apex_application_page_ir_col col
             WHERE col.application_id = app_id_in
           --    AND col.page_id = page_id_in
               AND col.interactive_report_id = ir_id_in
               AND col.allow_filtering = 'Yes'
               AND col.column_type IN ('STRING', 'NUMBER')
               AND col.column_alias = col_name_in
               AND rownum <= 1;

           RETURN found_col IS NOT NULL;

        EXCEPTION WHEN OTHERS THEN
            RETURN FALSE;
        END;

    BEGIN
        -- Simplistic protection against user entering huge match value
        IF length(REPLACE(TRIM(expr_in), '''', '''''')) > 512
```

```
        THEN
            RETURN NULL;
        END IF;

        l_expr := REPLACE(TRIM(expr_in), '''', '''''');
        col_name_table := apex_util.string_to_table(col_name_list);
        col_count := col_name_table.COUNT;

        IF col_count > 0
           AND l_expr IS NOT NULL
        THEN
            result_clause := '(';

            FOR i IN 1 .. col_count
            LOOP
                IF result_clause > '(' AND right_col_type(col_name_table(i))
                THEN
                    result_clause := result_clause || ' or ';
                END IF;

                IF right_col_type(col_name_table(i))
                THEN
                    match_count := match_count + 1;
                    result_clause := result_clause || 'instr(upper(' || col_name_table(i) ||
                                     '), upper(' || wrap_expr(l_expr) || ')) > 0' || crlf;
                END IF;
            END LOOP;
        END IF;

        result_clause := result_clause || ')';
        RETURN result_clause;
    EXCEPTION
    WHEN OTHERS THEN
            log_apex_access(app_name_in => 'ir_query_parse_search',
                                           app_user_in => v('APP_USER'),
                                           msg_in      => 'EXCEPTION: ' || SQLERRM ||
                                                          '.  column count: ' || col_count ||
                                                          '.  IR Report id: ' || ir_id_in ||
                                                          '.  l_expr length: ' ||
LENGTH(l_expr));
    END ir_query_parse_search;

    -- Generate IR query string for a user from their filters and searches
    FUNCTION ir_query_where(app_id_in IN NUMBER,
                            page_id_in IN NUMBER,
                            session_id_in IN NUMBER,
                            base_report_id_in IN VARCHAR2) RETURN VARCHAR2 IS
        /*
            Parameters:     base_report_id_in - User's currently-displayed report (including
saved)

            Returns:        ANDed WHERE clause to be run against base view

            Author:         STEWART_L_STRYKER
            Created:        2/12/2009 5:16:51 PM

            Usage:          RETURN apex_ir_query.ir_query_where(app_id_in => :APP_ID,
                                       page_id_in => 2,
                                       session_id_in => :APP_SESSION);

            CS-RCS Modification History: (Do NOT edit manually)
```

```
        $Log: $
    */
    query_string VARCHAR2(32500);
    test_val VARCHAR2(80);
    search_count PLS_INTEGER := 0;
    clause VARCHAR2(32000);

    query_too_long EXCEPTION;
    PRAGMA EXCEPTION_INIT(query_too_long, -24381);
BEGIN
    FOR filters IN (SELECT condition_column_name,
                           condition_operator,
                           condition_expression,
                           condition_expression2
                      FROM apex_application_page_ir_cond cond
                      JOIN apex_application_page_ir_rpt r ON r.application_id =
                                                             cond.application_id
                                                         AND r.page_id = cond.page_id
                                                         AND r.report_id = cond.report_id
                     WHERE cond.application_id = app_id_in
                       AND cond.page_id = page_id_in
                       AND cond.condition_type = 'Filter'
                       AND cond.condition_enabled = 'Yes'
                       AND r.base_report_id = base_report_id_in
                       AND r.session_id = session_id_in)
    LOOP
        clause := ir_query_parse_filter(filters.condition_column_name,
                                        filters.condition_operator,
                                        filters.condition_expression,
                                        filters.condition_expression2);
        IF LENGTH(clause) + LENGTH(query_string) > 32500
        THEN
            RAISE query_too_long;
        END IF;

        query_string := query_string || ' AND ' || clause;
    END LOOP;


    FOR searches IN (SELECT r.report_columns,
                            cond.condition_expression,
                            to_char(r.interactive_report_id) AS interactive_report_id
                      FROM apex_application_page_ir_cond cond
                      JOIN apex_application_page_ir_rpt r ON r.application_id =
                                                             cond.application_id
                                                         AND r.page_id = cond.page_id
                                                         AND r.report_id = cond.report_id
                     WHERE cond.application_id = app_id_in
                       AND cond.page_id = page_id_in
                       AND cond.condition_type = 'Search'
                       AND cond.condition_enabled = 'Yes'
                       AND r.base_report_id = base_report_id_in
                       AND r.session_id = session_id_in)
    LOOP
        search_count := search_count + 1;
        test_val := NVL(searches.interactive_report_id, 'null');
        clause := ir_query_parse_search(searches.report_columns,
                                        searches.condition_expression,
                                        app_id_in,
                                        searches.interactive_report_id);

        IF LENGTH(clause) + LENGTH(query_string) > 32500
```

```
            THEN
                RAISE query_too_long;
            END IF;

            query_string := query_string || ' AND ' || clause;

        END LOOP;

        log_apex_access(app_name_in => app_id_in,
                                       app_user_in => v('APP_USER'),
                                       msg_in      => 'Searches: ' || search_count ||
                                                      '.  base_report_id_in: ' ||
nvl(base_report_id_in, 'null')
                                                      || '.  Session: ' || session_id_in);
        RETURN query_string;
    EXCEPTION
        WHEN query_too_long THEN
            log_apex_access(app_name_in => app_id_in,
                                          app_user_in => v('APP_USER'),
                                          msg_in      => 'Generated query string would have
been > 32k');

            RETURN query_string;
        WHEN no_data_found THEN
            log_apex_access(app_name_in => app_id_in,
                                          app_user_in => v('APP_USER'),
                                          msg_in      => 'NDF. Searches: ' || search_count ||
                                                         '.  IR Report id: ' || nvl(test_val,
'null'));

            RETURN query_string;
        WHEN OTHERS THEN
            log_apex_access(app_name_in => app_id_in,
                                          app_user_in => v('APP_USER'),
                                          msg_in      => 'EXCEPTION: ' || SQLERRM ||
                                                         '.  Searches: ' || search_count ||
                                                         '.  IR Report id: ' || nvl(test_val,
'null'));

            RETURN query_string;
    END ir_query_where;

END apex_ir_query;
```