Dynamic SQL and Multi-Parameter Input Criteria

Providing Services for Oracle and other platforms

Overview

This document discusses the use of refcursors and associative arrays to build dynamic conditional logic. The resulting logic is capable of simple and complex scenarios. We will provide two versions of the api with this whitepaper. Version#1 will utilize a refcursor for the parameter criteria grid and the other will use an associative array to perform the same. The latter is more than likely far more cross platform, since the use of refcursors requires persistent connections.

Note : <PARMLIST> indicator will apply to both use of REFCURSOR (`dynsql_engine.q_demo_pg_current_snap_c1`) and Associative Array (`dynsql_engine.q_demo_pg_current_snap_c2`)

Essentially it provides the following key features :

- Selective Column Return using two methods
  - Dynamic selection
  - No bind variables required
  - Table driven

- Pagination or Chunking
  - Define page/chunk size dynamically
  - Very few bind variables required, and same regardless of call
  - Collect a page/chunk at a time
    - Improves Web Page performance
    - Decreases impact of data pulls on overall system performance
    - Permits data-stream processing

- Variable Parameter Input
  - Instead of hardcode api parameters or limiting to an array containing multiple values for a single parameter type, uses an associative array to build a Criteria Grid of all parameters to applied to SQL execution.
  - No bind variables required

- Logic to filter out potential SQL Injection routines

Selective Column Return

There are key tables that exist which support the identification of columns to include for a given *feed*. See QCURSOR_DATASET_COLUMN_MSTR and QCURSOR_DATASET_COLUMN_DETAIL. Access to the identified columns is via the use of key indicators called PROCESS and SUB_PROCESS. These will represent specific feeds or family of feed (e.g. same Process descriptor, but different Sub Process descriptor).

Optionally, we can leverage dynamic column selection. If the column name is not recognized, then it will simply ignore the column. The key advantages of this feature can be found in testing and situations where you want to be able to dynamically hide/show data columns or on popup search boxes where the columns returned could be potentially different from one app to another thus enabling the end user to select the desired record they were looking for.

We also identify whether selected columns are considered as HR Secure data-points. HR Secure data-points require special signoff and approval before proceeding with development. This approval is between the requestor and HR. In all other cases, we need only notify HR that the feed is being developed and will include stated columns. This is purely optional, however a good idea with respect to data privacy.

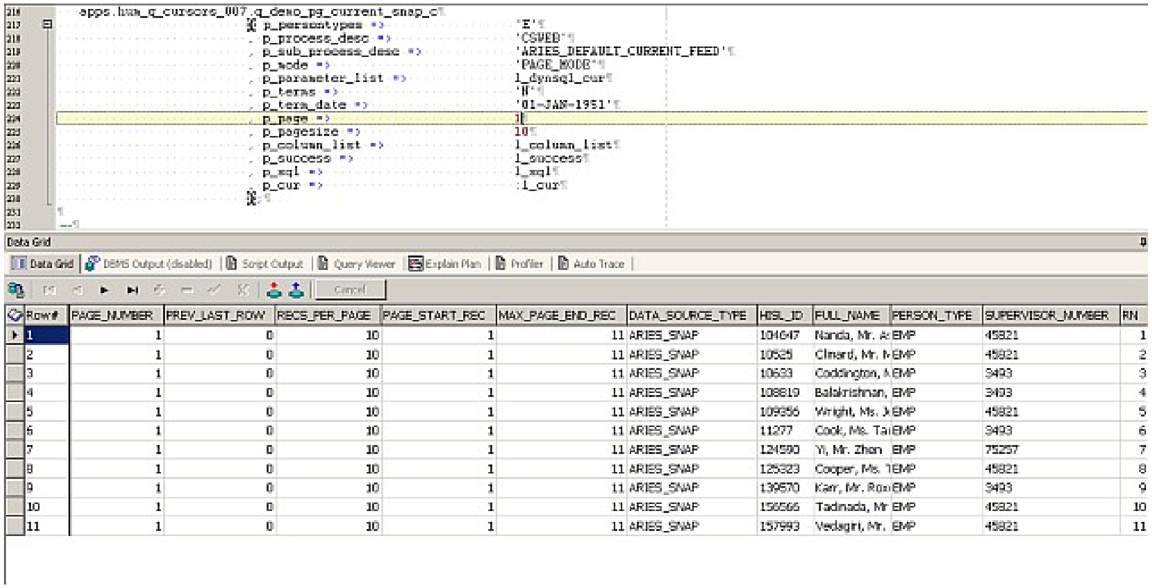
Pagination or Chunking

On occasions you may find yourself requesting large sums of data. This can be quite taxing on the system. One route to remove that issue, would be to size the data into chunks. The new api call introduces the ability to specify page size (or chunk size depending on how you want to look at it) and also what page you are requesting. The resulting SQL is optimized by using the optimizer hint /*+ FIRST_ROWS(###) */ where the number of rows is the same as your defined page/chunk size.

The data returned contains the requested number of rows plus 1 extra. The extra is an indicator that more records exist. When you request the next page, you will get that extra record again as the first record in the result set. This feature enables you to turn off web navigation as needed or let a process know that it has to go out and get more data. Additionally, there are other data-points return that describes what row number is the record in the full dataset or page details such as current page and starting record number.

Pagination/Chunk mode is controlled by a p_MODE parameter which should by default be NORMAL_MODE vs PAGE_MODE. Default page/chunk size is set to 25, but can be any value desired.

Here is a screen shot of requesting pages where the page size is 10. Notice that we return 11 records so as to indicate that there are more records to retrieve.

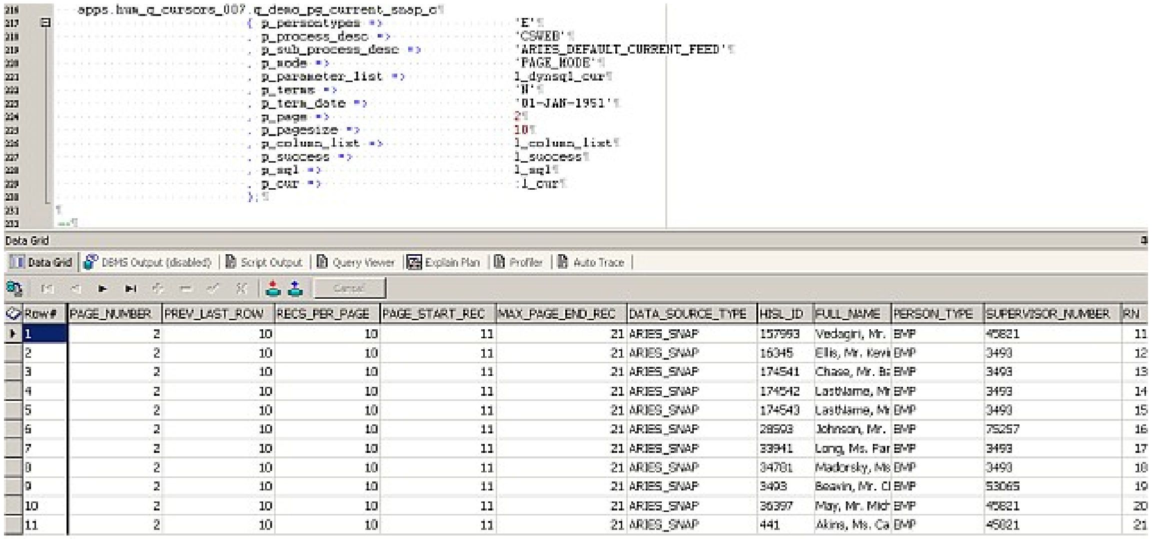


```
    apps.hum_q_cursors_007.q_demo_pg_current_snap_c
                              ( p_persontypes =>              'E'
                              , p_process_desc =>             'CSWEB'
                              , p_sub_process_desc =>         'ARIES_DEFAULT_CURRENT_FEED'
                              , p_mode =>                     'PAGE_MODE'
                              , p_parameter_list =>           l_dynsql_cur
                              , p_terms =>                    'N'
                              , p_term_date =>                '01-JAN-1951'
                              , p_page =>                     1
                              , p_pagesize =>                 10
                              , p_column_list =>              l_column_list
                              , p_success =>                  l_success
                              , p_sql =>                      l_sql
                              , p_cur =>                      :l_cur
                              );

--
```

Row#	PAGE_NUMBER	PREV_LAST_ROW	RECS_PER_PAGE	PAGE_START_REC	MAX_PAGE_END_REC	DATA_SOURCE_TYPE	HISL_ID	FULL_NAME	PERSON_TYPE	SUPERVISOR_NUMBER	RN
1	1	0	10	1	11	ARIES_SNAP	104647	Nanda, Mr. A	EMP	45821	1
2	1	0	10	1	11	ARIES_SNAP	10525	Clinard, Mr. M	EMP	45821	2
3	1	0	10	1	11	ARIES_SNAP	10633	Coddington, M	EMP	3493	3
4	1	0	10	1	11	ARIES_SNAP	108819	Balakrishnan,	EMP	3493	4
5	1	0	10	1	11	ARIES_SNAP	109356	Wright, Ms. J	EMP	45821	5
6	1	0	10	1	11	ARIES_SNAP	11277	Cook, Ms. Ta	EMP	3493	6
7	1	0	10	1	11	ARIES_SNAP	124590	Yi, Mr. Zhen	EMP	75257	7
8	1	0	10	1	11	ARIES_SNAP	125323	Cooper, Ms. T	EMP	45821	8
9	1	0	10	1	11	ARIES_SNAP	139570	Karr, Mr. Ron	EMP	3493	9
10	1	0	10	1	11	ARIES_SNAP	156566	Tadinada, Mr	EMP	45821	10
11	1	0	10	1	11	ARIES_SNAP	157993	Vedagiri, Mr.	EMP	45821	11

Request page #2, notice the change in page_number, prev_last_row, recs_per_page, page_start_rec, and the max_page_end_rec possible for this dataset.

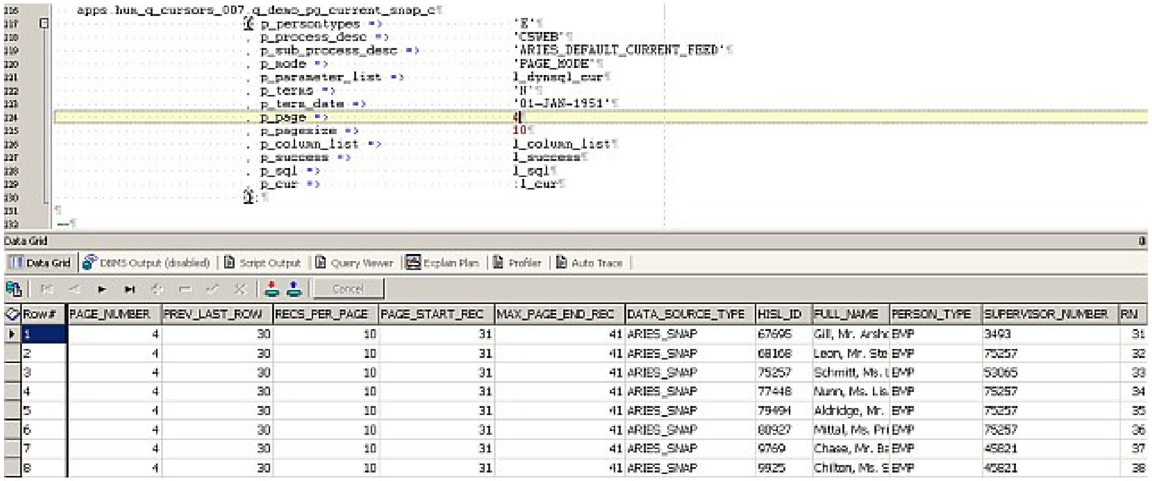


```
  apps.hum_q_cursors_007.q_demo_pg_current_snap_c
                          ( p_persontypes =>                      'E'
                          , p_process_desc =>                     'CSWEB'
                          , p_sub_process_desc =>                 'ARIES_DEFAULT_CURRENT_FEED'
                          , p_mode =>                             'PAGE_MODE'
                          , p_parameter_list =>                   l_dynsql_cur
                          , p_terms =>                            'N'
                          , p_term_date =>                        '01-JAN-1951'
                          , p_page =>                             2
                          , p_pagesize =>                         10
                          , p_column_list =>                      l_column_list
                          , p_success =>                          l_success
                          , p_sql =>                              l_sql
                          , p_cur =>                              :l_cur
                          );

--
```

Row#	PAGE_NUMBER	PREV_LAST_ROW	RECS_PER_PAGE	PAGE_START_REC	MAX_PAGE_END_REC	DATA_SOURCE_TYPE	HISL_ID	FULL_NAME	PERSON_TYPE	SUPERVISOR_NUMBER	RN
1	2	10	10	11	21	ARIES_SNAP	157993	Vedagiri, Mr.	EMP	45821	11
2	2	10	10	11	21	ARIES_SNAP	16345	Ellis, Mr. Kevi	EMP	3493	12
3	2	10	10	11	21	ARIES_SNAP	174541	Chase, Mr. B	EMP	3493	13
4	2	10	10	11	21	ARIES_SNAP	174542	LastName, Mr	EMP	3493	14
5	2	10	10	11	21	ARIES_SNAP	174543	LastName, Mr	EMP	3493	15
6	2	10	10	11	21	ARIES_SNAP	28593	Johnson, Mr.	EMP	75257	16
7	2	10	10	11	21	ARIES_SNAP	33941	Long, Ms. Par	EMP	3493	17
8	2	10	10	11	21	ARIES_SNAP	34781	Madorsky, Ms	EMP	3493	18
9	2	10	10	11	21	ARIES_SNAP	3493	Beavin, Mr. C	EMP	53065	19
10	2	10	10	11	21	ARIES_SNAP	36397	May, Mr. Mic	EMP	45821	20
11	2	10	10	11	21	ARIES_SNAP	441	Akins, Ms. Ca	EMP	45821	21

We repeat the api for each page we wish to retrieve. At page 4 we have retrieved our final
set. We know this because our page size is 10, but we only returned 8 rows. If there had been more there would have been 11 rows returned. You can also see that our maximum rows returned were 38 rows. This could have just as easily been chunk 1,2,3… of sizes 1000 till completed.



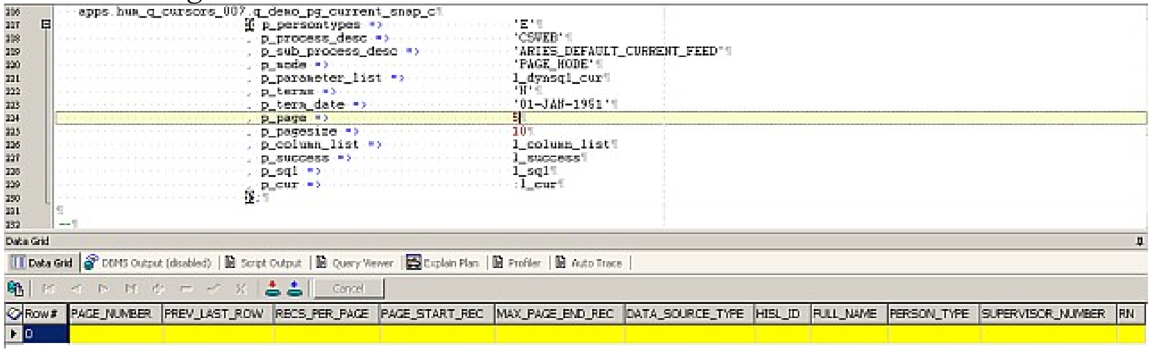
```
   apps.hum_q_cursors_007.q_demo_pg_current_snap_c
                              ( p_persontypes =>             'E'
                              , p_process_desc =>            'CSWEB'
                              , p_sub_process_desc =>        'ARIES_DEFAULT_CURRENT_FEED'
                              , p_mode =>                    'PAGE_MODE'
                              , p_parameter_list =>          l_dynsql_cur
                              , p_terms =>                   'N'
                              , p_term_date =>               '01-JAN-1951'
                              , p_page =>                    4
                              , p_pagesize =>                10
                              , p_column_list =>             l_column_list
                              , p_success =>                 l_success
                              , p_sql =>                     l_sql
                              , p_cur =>                     :l_cur
                              );

--
```

Row#	PAGE_NUMBER	PREV_LAST_ROW	RECS_PER_PAGE	PAGE_START_REC	MAX_PAGE_END_REC	DATA_SOURCE_TYPE	HISL_ID	FULL_NAME	PERSON_TYPE	SUPERVISOR_NUMBER	RN
1	4	30	10	31	41	ARIES_SNAP	67695	Gill, Mr. Arshc	EMP	3493	31
2	4	30	10	31	41	ARIES_SNAP	68108	Leon, Mr. Ste	EMP	75257	32
3	4	30	10	31	41	ARIES_SNAP	75257	Schmitt, Ms. L	EMP	53065	33
4	4	30	10	31	41	ARIES_SNAP	77448	Nunn, Ms. Lis	EMP	75257	34
5	4	30	10	31	41	ARIES_SNAP	79494	Aldridge, Mr.	EMP	75257	35
6	4	30	10	31	41	ARIES_SNAP	80927	Mittal, Ms. Pri	EMP	75257	36
7	4	30	10	31	41	ARIES_SNAP	9769	Chase, Mr. Br	EMP	45821	37
8	4	30	10	31	41	ARIES_SNAP	9925	Chilton, Ms. S	EMP	45821	38

If we accidentally tried to retrieve the next page, which does not exist because there is no more data to be found… we get an empty result set as seen here :

Screenshot Pagination #4



```
   apps.hum_q_cursors_007.q_demo_pg_current_snap_c
                              ( p_persontypes =>             'E'
                              , p_process_desc =>            'CSWEB'
                              , p_sub_process_desc =>        'ARIES_DEFAULT_CURRENT_FEED'
                              , p_mode =>                    'PAGE_MODE'
                              , p_parameter_list =>          l_dynsql_cur
                              , p_terms =>                   'N'
                              , p_term_date =>               '01-JAN-1951'
                              , p_page =>                    5
                              , p_pagesize =>                10
                              , p_column_list =>             l_column_list
                              , p_success =>                 l_success
                              , p_sql =>                     l_sql
                              , p_cur =>                     :l_cur
                              );

--
```

Row#	PAGE_NUMBER	PREV_LAST_ROW	RECS_PER_PAGE	PAGE_START_REC	MAX_PAGE_END_REC	DATA_SOURCE_TYPE	HISL_ID	FULL_NAME	PERSON_TYPE	SUPERVISOR_NUMBER	RN
0											

Another advantage is that a requestor can perform data-stream processing. Meaning that they can collect a 1000 records, and insert into a staging table with a process flag marked as unprocessed, then trigger a process to begin picking up that data and process same. While that is occurring return and request another 1000 records and drop into the queue for processing. Repeat the process till completed.

The triggered process always checks at the end to see if there are more records to process and repeats its cycle till completed. So instead of waiting for the hour it took to grab the data and then trigger the process to work the data, it begins to process the data in parallel to the actual data retrieval.

Further more you can leverage this feature to identify a requestor when additional criteria may be required. This is accomplished by establishing some tolerance, say 1000 records. If we set the page size to 1000 and get 1001 records back, using the feature of always getting plus 1 back to signify more records to go get. Then prompt the requestor to refine their result set.

Variable Parameter Input

Variable Parameter Input is satisfied by way of a Criteria Grid, which is passed as another associative array to the api. This grid is validated and then subsequently processed to identify the conditional logic to be applied to requested data. It permits both inclusion and exclusion logic.

e.g. Example of Condition Addon Blocks

```
AND (BLOCK 1
       |
       |__>>> SET 1
         |    |____>> MEMBER 1
         |    |____>> MEMBER 2
         |
         |_>>> SET 2
                 |
                 |___>> MEMBER 1
      )
```

```
AND (BLOCK 2
       |
       |__>>> SET 1
         |    |____>> MEMBER 1
         |    |____>> MEMBER 2
         |
         |_>>> SET 2
                 |
                 |___>> MEMBER 1
                 |___>> MEMBER 2
      )
```

As you can see you can see we build a construct that contains blocks which contain sets which contain members. Although we permit dynamic parameter selections, we enforce a series of rules around this logic.

Criteria Grid Rules

```
The following rules permit highly flexible queries to be generated dynamically.
The VALIDATE_DEMO_PG_CRITERIA API call will be used to enforce these rules.

0.  NON RULE VIOLATION ENCOUNTERED, BUT CAUSED FAILURE IN VALIDATION ROUTINE (e.g. Parameters
    invalid)
1.  CONDITION MEMBERS ARE SEPARATED BY CONDITION OPERATORS OF 'OR' and 'AND' WHEN MORE THAN ONE
    CONDITION MEMBER.
2.  WHEN ONE CONDITION MEMBER ONLY, THEN CONDITION OPERATOR SHOULD BE 'AND'.
3.  CONDITION OPERATORS APPLY THE CONDITION SET AS A WHOLE AND USED AS A SEPARATOR BETWEEN
    CONDITION MEMBERS.
4.  CONDITION SETS CAN ONLY CONTAIN ONE TYPE OF CONDITION OPERATOR. NO MIXING OF OPERATORS.
5.  ANY GIVEN CONDITION SET CAN CONTAIN ANY NUMBER OF ONE OR MORE CONDITION MEMBERS.
6.  ANY GIVEN CONDITION BLOCK CAN CONTAIN ANY NUMBER OF ONE OR MORE CONDITION SETS.
7.  MUST AT LEAST CONTAIN ONE CONDITION BLOCK
8.  CONDITIONS SETS WITHIN A BLOCK ARE ALWAYS SEPARATED BY 'OR' OPERATORS. MEANING
    ANY ONE OF THE CONDITION SETS CAN BE TRUE, AND DOES REQUIRE ALL CONDITION SETS
    TO BE TRUE TO SATISFY THE QUERY CONDITION LOGIC.
9.  CONDITION BLOCKS ARE ALWAYS SEPARATED BY 'AND' OPERATORS. MEANING THAT EACH BLOCK MUST BE
    TRUE TO SATISFY QUERY CONDITION LOGIC.
10. ANY NUMBER OF ONE OR MORE CONDITION BLOCKS CAN BE DEFINED.
11. CONDITION BLOCK ID's are UNIQUE. CONDITION SET IDs ARE UNIQUE WITHIN A GIVEN CONDITION BLOCK.
    CONDITION MEMBER IDs ARE UNIQUE WITHIN A GIVEN CONDITION SET.
12. ONLY ONE ORGTREE SEARCH PER PARAMETER_ENTRY_ID.
13. KEYWORDS NOT AUTHORIZED IN PARAMETER VALUES
        - DBA_
        - DBMS_SQL
        - GRANT
        - CREATE
        - ALTER
        - REVOKE
        - EXECUTE IMMEDIATE
        - SELECT
```

The requestor (whether it be external or internal Oracle stored procedure call) must build the criteria grid into the form of a <PARMLIST>. The record structure of the <PARMLIST> is :

```
{PARAMETER_ENTRY_ID}|{PARAMETER_NAME}|{VDATA_TEXT}|{VDATA_DATE}|{VDATA_NUMBER}|{INCL_EXCL_FLAG}|
{CONDITION_BLOCK_ID}|{CONDITION_SET_ID}|{CONDITION_MEMBER_ID}|{CONDITION_OPERATOR}
```

e.g.

```
l_parameter_list(1) := 1|ORGTREE|139245|NULL|NULL|I|1|1|1|OR
l_parameter_list(2) := 2|HISL_ID|139245|NULL|NULL|I|1|1|2|OR
l_parameter_list(3) := 3|BASE_ROLE|Applications Consultant|NULL|NULL|I|2|1|1|AND
..
..
and so on.

Or refcursor where the data structure matches the GTT.
```

Parameter Criteria Grid Layout

When using the refcursor method for the <PARMLIST>

```
  PARAMETER_ENTRY_ID    NUMBER,
  PARAMETER_NAME        VARCHAR2(100 BYTE),
  VDATA_TEXT            VARCHAR2(4000 BYTE),
  VDATA_DATE            DATE,
  VDATA_NUMBER          NUMBER,
  INCL_EXCL_FLAG        VARCHAR2(10 BYTE),
  CONDITION_BLOCK_ID   NUMBER,
  CONDITION_SET_ID     NUMBER,
  CONDITION_MEMBER_ID  NUMBER,
  CONDITION_OPERATOR   VARCHAR2(10 BYTE)
```

When using the associative array method for the <PARMLIST>

- NULL fields are reperesented by a literal text string stating NULL
- Since the array is a single string, as such the individual fields separated by pipes (|) also represent text strings
- Pipes (|) should contain any extra spaces around them, unless required by the physical value itself contained within that defined field area
- VDATA_DATE must be passed as YYYY/MM/DD HH24:MI:SS, as this will be the only recognized format. This is the standard Oracle text representation of dates. Internally, we will convert same to a true date and process accordingly

All fields are required fields, including the VDATA* fields. VDATA fields that do not contain data should contain the respective NULL entry. The PARAMETER_NAME should be passed in as UPPERCASE. The INCL_EXCL_FLAG (Inclusions/Exclusions) should be passed as either I or E respectively. There exist a check constraint to prevent anything but those values. A similar check constraint exists on CONDITION_OPERATOR and only permits values of AND or OR. Condition BLOCK, SET, and MEMBER IDs will contain positional numbers. There must always be a CONDITION_BLOCK_ID of value 1. In any given BLOCK there must always exist a CONDITION_SET_ID of value 1 and CONDITION_MEMBER_ID of value 1.

The parameter_entry_id is used to collapse rows into single conditional statements. Best way to describe is :

You want to send over a list of 1000 Employee Numbers. So you know that your dataset is going to be 1000 rows long. What we don't want is 1000 EMPLOYEE_NUMBER = x added to the query… what we want is EMPLOYEE_NUMBER in (Select value from table where parameter_name = EMPLOYEE_NUMBER) more or less. By sharing the same parameter_entry_id, condition_block_id, condition_set_id, etc., we are saying its all part of the same parameter value set. Using the shared parameter_entry_id is our best way to represent that request.

Available search criteria comes from the QCURSOR_SEARCH_COLUMNS, where more can easily be established. The use of this table results in fewer if any bind variable requirements for the api, which at best can get quite complex in maintaining code wise.

Note
To determine what vdata to use, look up the search column in the QCURSOR_SEARCH_COLUMN table and take note of the data type indicated.*

Building Criteria Grid

Here is an example criteria grid submission :

Objective :

ORGTREE of Supervisor # 45821 or ORGTREE of Supervisor # 53605 or Employee # 45821 or Employee# 53065, but do not include ORGTREE of Supervisor # 3493 who is a direct report to Supervisor 53605.

Example

parameter_entry_id	=>	1	parameter_entry_id	=>	3
parameter_name	=>	ORGTREE	parameter_name	=>	HISL_ID
vdata_text	=>	45821	vdata_text	=>	45821
vdata_date	=>	NULL	vdata_date	=>	NULL
vdata_number	=>	NULL	vdata_number	=>	NULL
incl_excl_flag	=>	I	incl_excl_flag	=>	I
condition_block_id	=>	1	condition_block_id	=>	1
condition_set_id	=>	1	condition_set_id	=>	1
condition_member_id	=>	1	condition_member_id	=>	3
condition_operator	=>	OR	condition_operator	=>	OR
parameter_entry_id	=>	2	parameter_entry_id	=>	3
parameter_name	=>	ORGTREE	parameter_name	=>	HISL_ID
vdata_text	=>	53605	vdata_text	=>	53065
vdata_date	=>	NULL	vdata_date	=>	NULL
vdata_number	=>	NULL	vdata_number	=>	NULL
incl_excl_flag	=>	I	incl_excl_flag	=>	I
condition_block_id	=>	1	condition_block_id	=>	1
condition_set_id	=>	1	condition_set_id	=>	1

condition_member_id => 2
condition_operator => OR
	condition_member_id => 3
	condition_operator => OR

parameter_entry_id => 4
parameter_name => ORGTREE
vdata_text => 3493
vdata_date => NULL
vdata_number => NULL
incl_excl_flag => E
condition_block_id => 2
condition_set_id => 1
condition_member_id => 1
condition_operator => AND	

This information would be passed over within the <PARMLIST> to the demographic api call. From there it will pass through a validation routine to ensure that none of the validation rules are violated. It will fetch through the criteria grid and store into a Global Temp Table (GTT). The purpose of using a GTT is that they offer session activity separate from other sessions, and once the session ends, so does the data stored there.

It is critically important that the session not be reused. To handle this potential event, unique indexes have been applied to the GTT where we store the criteria grid. Should the index constraints be violated due to a session being reused, the request will fail. This prevents the undesirable event from the beginning.

The necessary SQL will not be built to select the desired return columns, any special conditions that may apply and final the specific criteria grid items that were submitted. The criteria is evaluated and a representative condition is built to be appended to the SQL. At this point the query is the executed and the resulting result set is passed back to the calling system as a refcursor for display or additional processing.

An example test script is provided with this documentation which is capable of providing some debug output data and generated SQL in the appendix. You will additionally find other example test cases beyond the one provided above.

With respect to the generated SQL, the below snipped represents the condition built from the submitted criteria grid. Notice that are parameter_entry_id 3 (EMPLOYEE_NUMBER) is only listed once in the condition because we rolled them together. This is accomplished by taking the requested parameter_name and concatenating the parameter_entry_id requested to the end of the parameter_name and storing in the GTT. We then later pull all records with a parameter_name equal to **EMPLOYEE_NUMBER [3]**.

```
AND ( ( /* ORGTREE */    ( hds.employee_number IN (
                          SELECT     hdsx. employee_number
                               FROM demo_snap hdsx
                              WHERE 1 = 1 AND LEVEL > 1
                          CONNECT BY PRIOR hdsx. employee_number =
                                           hdsx.supervisor_number
                          START WITH hdsx.hisl_id IN (
                                      SELECT vdata_text
                                        FROMq_cursor_parms_global_temp
                                       WHERE 1 = 1
                                         AND parameter_name =
                                                'ORGTREE' || ' [1]'
                                         AND incl_excl_flag = 'I' ))
                     )
                    OR /* ORGTREE */ ( hds. employee_number IN (
                                      SELECT     hdsx. employee_number
                                           FROM demo_snap hdsx
                                          WHERE 1 = 1 AND LEVEL > 1
                                      CONNECT BY PRIOR hdsx. employee_number =
                                                 hdsx.supervisor_number
                                      START WITH hdsx. employee_number IN (
                                                 SELECT vdata_text
                                                   FROMq_cursor_parms_global_temp
                                                  WHERE 1 = 1
                                                    AND parameter_name =
                                                           'ORGTREE'
                                                        || ' [2]'
                                                    AND incl_excl_flag =
                                                           'I' ))
                                 )
                    OR /* EMPLOYEE_NUMBER */ ( hds. employee_number IN (
                                          SELECT hdsx. employee_number
                                            FROM demo_snap hdsx
                                           WHERE 1 = 1
                                             AND EXISTS (
                                                   SELECT 'x'
                                                     FROMq_cursor_parms_global_temp
                                                    WHERE 1 = 1
                                                      AND parameter_name =
                                                               ‘EMPLOYEE_NUMBER’
                                                          || ' [3]'
                                                      AND incl_excl_flag =
                                                                  'I'
                                                      AND hdsx. employee_number =
                                                             vdata_text ))
                                      )
     )
    )
  AND ( ( /* ORGTREE */ ( hds. employee_number NOT IN (
                          SELECT     hdsx. employee_number
                               FROM demo_snap hdsx
                              WHERE 1 = 1 AND LEVEL > 1
                          CONNECT BY PRIOR hdsx. employee_number =
                                             hdsx.supervisor_number
                          START WITH hdsx. employee_number IN (
                                      SELECT vdata_text
                                        FROMq_cursor_parms_global_temp
                                       WHERE 1 = 1
                                         AND parameter_name =
                                                'ORGTREE' || ' [4]'
                                         AND incl_excl_flag = 'E' ))
                     )
     )
    )
```

Submitting the same criteria, but in page_mode, essentially generates the same query with a wrapper piece around it which provides the paging effect. An example is provided in the Appendix as well.

As evidence here, the capabilities of this mechanism can go from simple to very complex. The real identifiable effort is the building of the user interface so that it is simple but effective.

Deep Dive…

If all you do is run the required objects and compile the package header/body, you should be able to run the test script right off the bat (assuming you are using TOAD). However, lets take some time to go deeper into the code and run step by step whats really happening. Given that both the c1 and c2 api calls (refcursor and associative array respectively) are similar albeit the one difference, we will only review in detail the associative array entry since it is more cross-platform compliant.

Supporting Objects, API Calls, Variable Declarations, etc.

A number of procedures and functions have been added to the DYNSQL_ENGINE package as support utilities for the c1 and c2 api calls. Those calls are :

PARSE UTILITIES

- a_delimiter
    - Returns TRUE if the character passed into the function is found in the list of delimiters
- string_length
    - Returns the length of the string passed
- next_atom_loc
    - Returns the location in the string of the starting point of the next atomic (from the start location)
- increment_counter
    - Used by nth_atomic and number_of_atomics to add to the count of atomics

- display_atomics
    - Dumps contents of PL/SQL table
- parse_string (overloaded)
    - Stores list of atomics in a PL/SQL table
    - Writes the atomics out to a packed list in the format |A|,|C|
- number_of_atomics
    - Counts number of atomics in the string
- nth_atomic
    - Find and Return the nth atomic in a string

DBMS_OUTPUT UTILITIES

- print_output
    - Simple utility to dump to the dbms serveroutput to screen in chunks of 250 to a maximum of 1000.

VALIDATION UTILITIES

- validate_demo_pg_criteria
    - Utility to process submitted criteria through some basic sanity checks before continuing. More checks can be added as needed within this routine, or enhanced so that it is more table driven.

DEBUG MODE

- Debug mode
- An l_debug declaration has been added to the package specification. By using the appropriate statement in your test scripts, you will be able to produce additional serveroutput which could enable insight to how the process works as well as performing any necessary analysis.

  Dynsql_engine.l_debug := 'Y';

OBJECT TYPES

- QCURSOR_COL_RECTYPE
  - Used for defining QCURSOR_COL_TBL record layout, which represents a container for list of column names to display
- QCURSOR_COL_TBL
  - Container for list of column names to display

TABLES

- QCURSOR_DATASET_COLUMN_MSTR
  - Table that contains master listing of columns and related SQL representation of which are available for column display
  - Columns are identified as three types : TEXT, DATE, or NUMBER
  - Column Order is provided so that columns are consistently displayed in same order regardless of feed. This provides standardization and common grouping of columns
  - Secure Column indicator for where data privacy is an issue, can be specifically identified
  - Detail column which contains the SQL representation of column display, which could be a physical column in the respective table or a function call
- QCURSOR_DATASET_COLUMN_DETAIL
  - Table that contains list of columns on a per feed basis as identified by Process and Sub Process
  - Same columns as master table, however it also includes PROCESS and SUB_PROCESS identifiers, and also an ENABLED column
- Q_CURSOR_DYNCOL_GLOBAL_TEMP
  - This tables stores the list of columns to be displayed in the event requestor chooses to pass list of columns to display as an associative array, as opposed to selecting from the QCURSOR_DATASET_COLUMN_DETAIL
  - Global Temp Tables are good for the session or until commit is issued
- Q_CURSOR_PARMS_GLOBAL_TEMP
  - This table is utilized to store the parameter criteria grid submitted
  - Global Temp Tables are good for the session or until commit is issued
  - Check Constraints have been added for the INCL_EXCL_FLAG and CONDITION_OPERATOR columns
  - No unique indexes assigned, however there is a single BTree index

- QCURSOR_SEARCH_COLUMNS
  - This table is used to store specific search conditions relating to data being queried
  - Searches contained within here are used as a component to building necessary parameter criteria grids
- DEMO_SNAP
  - This is a test table that contains demographic data on associates
  - Concept is that you might have a nightly job that polls your system tables and collates all the related demographic data into a single table. This table can then be used to source data from as opposed to using live system tables. Use of the latter could creation potential performance issues for the system as a whole
  - Although this whitepaper has been built around the premise of a demographic search utility, the concept could just as easy be applied to inventory, geographic, etc. data sets.

API :: By the Numbers

The api looks as follows :

API CALL

```
PROCEDURE q_demo_pg_current_snap_c2 (
   p_process_desc IN VARCHAR2 DEFAULT 'GENERIC'
 , p_sub_process_desc IN VARCHAR2 DEFAULT 'GENERIC'
 , p_mode IN VARCHAR2 DEFAULT 'NORMAL_MODE'
 , p_parameter_list IN dynsql_engine.vc4000_table
 , p_terms IN VARCHAR2 DEFAULT 'N'
 , p_term_date IN DATE DEFAULT TRUNC ( SYSDATE )
 , p_page IN NUMBER DEFAULT 1
 , p_pagesize IN NUMBER DEFAULT NULL
 , p_column_list IN dynsql_engine.vc4000_table
 , p_success OUT VARCHAR2
 , p_sql OUT VARCHAR2
 , p_cur OUT dynsql_engine.c_cursor
);
```

The parameters in detail :

PARAMETER DETAILS

- p_process_desc
  - Input Parameter
  - Narrative text that identifies a specific feed to retrieve data for
  - Process and Sub Process are found in the dataset master table
- p_sub_process_desc
  - Input Parameter
  - Narrative text that identifies same as above, but for the Sub Process instead
- p_mode
  - Input Parameter
  - States if in NORMAL_MODE or PAGE_MODE
  - Page mode is also known as Chunk
  - Used for either getting data in pieces or all at once, depending on design and size of data
- p_parameter_list
  - Input Parameter
  - Represents the parameter criteria grid of records to be searched (search conditions)
- p_terms
  - Input Parameter
  - This is one of three values
  - Y => Yes, include only terminated records
  - N => No, do not include terminated records
  - ALL => Include both, terminated and active records
- p_term_date
  - Input Parameter
  - This is regarded as the termination date cutoff, meaning how far back to search for terms
  - The further back you search the slower potentially the query

- p_page
  - Input Parameter
  - When in PAGE_MODE, identifies page/chunk to be retrieved
- p_pagesize
  - Input Parameter
  - When in PAGE_MODE, identifies how many records per page to be displayed
  - This impacts the optimization hint for retrieving data to where it uses FIRST_ROWS
- p_column_list
  - Input Parameter
  - This associative array identifies which columns are being requeseted
  - If empty, then api will fall back to the use of the QCURSOR_DATASET_COLUMN_DETAIL
- p_success
  - Output parameter
  - If the api ends without error, then OK is returned
  - If error occurs, then p_success is prefixed with the verbage ERROR and the error narration to follow
- p_sql
  - Output parameter
  - This contains the generated SQL that was executed
  - Excellent for debugging
  - Consider writing to a table to log queries executed or email SQL to email address for analysis
- p_cur
  - Output Parameter
  - REFCURSOR output

DEFINED EXCEPTIONS

- err_noparms_passed
  - Indicates missing parameters
- err_badparm_combo
  - Parameter combination detected that is invalid

    - Details will following in message
- err_invalid_parameter
    - Specific parameter input has been identified as invalid on the parameter criteria grid
- err_placeholder
    - Unhandled errors should go to the WHEN OTHERS exception, however to handle final case situation we need this
- err_dataset_not_defined
    - Indicates that there are no records existing for stated Process / Sub Process in the QCURSOR_DATASET_COLUMN_DETAIL table
- err_parmlistvalidate
    - The parameter criteria grid submitted has violated a validation rule
    - Rule Id will be provided on message

DEFINED CURSOR :: C_DATASET

C_DATASET cursor is defined that will provide list columns to display and their associated SQL text for usage. Depending upon if identified as table driven or dynamically driven, it will source appropriately from the QCURSOR_DATASET_COLUMN_MSTR or the QCURSOR_DATASET_COLUMN_DETAIL table.

SETTING UP ENVIRONMENT

At the initial stages of the api call, we establish some basic environmental settings such as whether we are in Normal mode or Page mode. To support the latter, we also determine what the optimization hint needs to be for pagination/chunking.

We then take a quick count of the atomics in the associative array *p_column_list* . If there are entries identified, we then populate the global tempt table (GTT) called Q_CURSOR_DYNCOL_GLOBAL_TEMP with those values.

Once our columns have been identified, we then load up our bulk collection array called *ardataset* . The bulk collection array is used to build our columns to be displayed. A column alias name is utilized when the column to be displayed is either a function or different than the standard column alias name to be utilized. During this process we have to also capture the NULL data representation of columns so that we can build our empty sql query. The empty sql query is used during exception handling so that an output REFCURSOR is *always* produced, even though it may be an empty set. The requestor has the responsibility of evaluating the p_success parameter to determine if it is OK and there simply are no data matches to the request, or if the p_success is actually identified as ERROR and that is why we did not return data.

PARAMETER VALIDATION

We now go through a series of parameter validation routines that check for bad combinations, unknown parameter names, invalid criteria grid builds, disabled search column options, etc. The criteria grid is submitted in the c2 api call as an Associative Array, where the single string is actually a concatenated list of values separated by pipes. This is intended to represent a record layout.

Using the parse string (atomics calls at lead of package sourcecode) we are able to break out the individual values, and the subsequently check them and store into the Q_CURSOR_PARMS_GLOBAL_TEMP table. One of the key things we do is modify the parameter name to include a concatenated parameter_entry_id. This id provides uniques to the parameter name but also enables the stacking of the same parameter name or slamming the parameter together with similar parameter criteria grid entries. After they have been loaded into our GTT, we then run through the *validate_demo_pg_criteria* api call.

Throughout the code there are various points where we check against the *l_debug* variable in the package specification, to see if we are in debug mode. If so, we display a variety of serveroutput messages to aid in process understanding and problem resolution.

ACTIVE VERSUS TERMINATION SEARCH

The *p_terms* and *p_term_date* parameters come into play now as we build our termination search criteria. We maintain these separate from the criteria grid, since they can be more generically applied to individuals. Ideally, we should keep our *p_term_date* relatively close to current data when *p_terms is set to Y or ALL*. Anything else will potentially decrease performance due to full scans into the past.

BUILDING BASIC SQL

We have now reached a point in our api call where we have the basics of our SQL statement :

```
      l_sql_emp :=
            ' SELECT '
         || l_hint_text_emp
         || l_emp_dataset_cols
         || ' FROM  demo_snap hds '
         || ' WHERE 1 = 1 '
         || l_terms_emp;
```

The only thing missing is the special conditions / search criteria.

CRITERIA GRID :: BUILDING SEARCH CONDITIONS

Building the criteria grid involves the use of SQL, though by appearances complex, simply uses LAG and LEAD analytical functions to flag data accordingly. What we mean by flag data accordingly is that since we are trying to build a where clause, we need to understand how to place all of our SQL syntax :

```
FROM  demo_snap hds
WHERE 1 = 1
  AND (/* BLOCK#1 */
            (/* SET#1 OF BLOCK#1 */
                  {MEMBER1 OF SET#1 OF BLOCK#1} AND {MEMBER2 OF SET#1 OF BLOCK#1})
                  OR
                  (
                  /* SET#2 OF BLOCK#1 */
```

```
                {MEMBER1 OF SET#2 OF BLOCK#1})
          )
  AND (
      /* BLOCK#2 */
          (
          /* SET#1 OF BLOCK#2 */
          {MEMBER1 OF SET#1 OF BLOCK#2} OR {MEMBER2 OF SET#1 OF BLOCK#2})
          )
```

The real trick to determining how to leverage the LEAD/LAG functions, is being able to visualize the query and its associated syntax.

IDENTIFYING EXCLUSION VERSUS INCLUSION ENTRIES

One of the unique features of this process is the ability to have exclusion logic as well as inclusion logic… dynamically. All of our search columns contained with the QCURSOR_SEARCH_COLUMNS table are essentially prepared as inclusion statements. Through the parsing of the parameter criteria grid we are able to recognize when certain criteria is intended to be exclusion entries. When this occurs, we exchange *INCL_EXCL_FLAG* column entries from *I* to *E* to represent the criteria as Exclusion, as well as changing the *EXISTS* to *NOT EXISTS*. OrgTree related search have some additional handling to switch to Exclusion mode. Finally, as we prepare to build the necessary SQL syntax, we swap out *xxxparameter_entry_idxxx* contained within the search column entries with the appropriate parameter entry id. Doing so, enables the use of the unique parameter name structures we built earlier in the process.

Beyond that, presuming that the search column entries were prepared consistently, we need only respond to the construct instructions to building the necessary SQL.

Note : The resulting SQL where clauses will have comments embedded within them that displays what search column was being added to the SQL. Also, if in debug mode, we are permitted to see what the parameter entry id was before and after the current parameter. These are offered as part of the analysis output to resolve possible issues with SQL building and use of the api.

PAGINATION MODE

Pagination Mode or also known as data chunking, provides a useful and necessary way of controlling data flow. So often, we run into scenarios where the data coming back is very large. If we had a way to extract and process in sizable chunks… would that not be great ?! Needless say, what a benefit that would be to the end user on a web page who does not want to wait for 2000 entries to display on their web page, but instead would be happy to review same 100 at a time.

It is important that you include PERSON_TYPE and EMPLOYEE_NUMBER as part of column returns, as these are used as a component of the ranking logic that provides the pagination effect.

This is the one scenario is this whitepaper where we introduce bind variables back into the equation. However, the only requirement of the api is that it told the page to be viewed and what the page size should be. The default page size is 25.

OUTBOUND PARAMETER :: REFCURSOR

We have finally made it to the end. With our SQL built, we simply open the cursor using our newly created SQL, and return same via the output parameter *p_cur* .

Possible areas of improvement

- Store search criteria in a table for later retrieval
  - Associated to Process / Sub Process as seeded predefined search conditions
  - Shared search parameters for other processes to utilize
  - Data analysis and troubleshooting
  - Etc.

- Graphical frontend to interface and create necessary parameter criterion

FINAL STATEMENT

As stated earlier, to include verbal conversations that I have had with many, this process lends itself to being very powerful. However, with that power comes great responsibility… sounds familiar doesn't it. Well what we are leading to is that since we are providing a means to customize search conditions on the fly, in theory the resulting query could look very different from one execution to another. That difference means that Oracle has to reparse the query as fresh for each variation. So pick wisely where you want to permit custom selection on the fly type functionality.

The best applications are those that are predefined, where you wish to define your *business rules* within a table structure instead of hardcoded into the query. Where you desire a means to maintain it via a table update and not through change of code every time. Where if you want to add a column… you add a column to a table. Where if you wish to create a new search option, you simply add the search option to a table… never touching code.

In the event you do however expose the free execution of custom conditional logic to a larger populace, I encourage you to restrict how much you are willing to let them customize their search conditions. I have been able to produce some very complex search conditions which return the data promptly, but I have also created conditions that were bad combinations which filtered each other out which very quickly return no data. This can be confusing to an end user who may not understand what they were just asking for.

So in the end, understand :

- Your Database
- Your Data
- Your Goal
- Your Customer

This information is posted at my website, http://www.myoracleportal.com, and is available for free download. If you have specific questions, I can be reached there as well. Thank you for your time and interest. I hope this document serves you well, if only that it may have proved thought provoking.

Appendix A :: Test Script

```
DECLARE
/*

Script is optimized for execution in TOAD.

Input Parameters :
    l_dynamic_column_list   =>  [STRING]    If Y, it will read column_list array submitted
                                            for columns to return, otherwise it will read from table
                                            the assigned columns to the feed
    l_user_refcursor        =>  [STRING]    If Y, it will read refcursor info otherwise uses associative array entries
    l_email_flag            =>  [STRING]    Required. Y/N to gen email containing SQL executed.
    l_smtp_server           =>  [STRING]    Required if l_email_flag = Y. SMTP Server name.
    l_domain                =>  [STRING]    Required "  ". Server Domain.
    l_from_email            =>  [STRING]    Required "  ". Sender email address.
    l_from_name             =>  [STRING]    Required "  ". Text Name of Sender e.g. Barry Chase.
    l_to_email              =>  [STRING]    Required "  ". Recipient email address.
    l_test_case             =>  [STRING]    Required. Test Case to execute.
    l_mode                  =>  [STRING]    Required. NORMAL_MODE or PAGE_MODE.
    l_terms                 =>  [STRING]    Required. Y, N, or ALL (includes Active and Termed).
    l_term_date_boundary    =>  [DATE]      Required only if l_terms = Y or ALL.
                                            Date to limit term search by. Must be less than TRUNC(SYSDATE).
    l_page                  =>  [STRING]    Required only if l_mode = PAGE_MODE.
                                            Page Number to view when in PAGE_MODE.
    l_pagesize              =>  [STRING]    Required only if l_mode = PAGE_MODE.
                                            Page/Chunk Size. If result is +1 than requested,
                                             increment page and execute again.
    l_cur                   =>  [CURSOR]    Output Parameter. No input value required.
                                            Returns REFCURSOR to data grid.

Additionally, set dbms_output on to see generated debug information.

*/

   --
   lbok                         BOOLEAN;
   l_sql                        VARCHAR2 ( 32767 );
   l_parm_sql                   VARCHAR2 ( 32767 );
   l_success                    VARCHAR2 ( 32000 );
   l_dynsql_error               VARCHAR2 ( 32000 );
   l_dynsql_cur                 dynsql_engine.parameter_cur_type;
   l_parameter_list_data        q_cursor_parms_global_temp%ROWTYPE;
   l_cur                         dynsql_engine.c_cursor;
   l_column_list_empty           dynsql_engine.vc4000_table;
   l_column_list                 dynsql_engine.vc4000_table
                                                       := l_column_list_empty;
   l_parameter_list_empty        dynsql_engine.vc4000_table;
   l_parameter_list              dynsql_engine.vc4000_table
                                                    := l_parameter_list_empty;
--
-- Change email address local variables as needed.
--
-- e.g. bchase@xxxx.com or
--       bchase@xxxx.com,fjohnson@xxxxx.com,staylor@xxxx.com
--
--
   l_from_email_address          VARCHAR2 ( 250 ) := :l_from_email;
   l_from_email_address_name     VARCHAR2 ( 250 ) := :l_from_name;
   l_to_email_address            VARCHAR2 ( 250 ) := :l_to_email;
   l_email_flag                  CHAR ( 1 ) := NVL ( :l_email_flag, 'N' );
   l_smtp_server                 VARCHAR2 ( 100 ) := :l_smtp_server;
   l_domain                      VARCHAR2 ( 100 ) := :l_domain;
--
   v_line                        VARCHAR2 ( 32767 );
--
   pos                           PLS_INTEGER := 1;
   bytes_o_data         CONSTANT PLS_INTEGER := 32767;
   offset                        PLS_INTEGER := bytes_o_data;
   msg_length                    PLS_INTEGER;
   i                             BINARY_INTEGER;
   pos_dbms                      PLS_INTEGER := 1;
   bytes_o_data_dbms    CONSTANT PLS_INTEGER := 1000;
   offset_dbms                   PLS_INTEGER := bytes_o_data_dbms;
   msg_length_dbms               PLS_INTEGER;
--
--
   l_test_case                   PLS_INTEGER := NVL ( :l_test_case, 0 );
--
   err_no_test_selected          EXCEPTION;
   err_dynsql_error              EXCEPTION;

--
   TYPE varchar2_table IS TABLE OF VARCHAR2 ( 1000 )
      INDEX BY BINARY_INTEGER;

--
   c                             UTL_SMTP.connection;

   PROCEDURE send_header ( NAME IN VARCHAR2, header IN VARCHAR2 )
   AS
   BEGIN
      UTL_SMTP.write_data ( c, NAME || ': ' || header || UTL_TCP.crlf );
   END;

   PROCEDURE dynsql (
      p_sql IN VARCHAR2
    , p_error OUT VARCHAR2
    , p_cur OUT dynsql_engine.c_cursor
   )
   IS
      stmt                              VARCHAR2 ( 32767 );
      c_process            CONSTANT VARCHAR2 ( 30 ) := 'ANONYMOUS_DYNSQL';
   BEGIN
      stmt := p_sql;

      OPEN p_cur FOR stmt;

      p_error := NULL;
   EXCEPTION
      WHEN OTHERS THEN
         p_error :=
            'ERROR IN ' || c_process || ' :: ' || SQLCODE || ' - ' || SQLERRM;
   END;
--
--
BEGIN
   dynsql_engine.print_output ( '.' );
   dynsql_engine.print_output ( '.' );
   dynsql_engine.print_output ( '.' );
   dynsql_engine.print_output ( '.' );
   dynsql_engine.print_output ( '**********************************' );
   dynsql_engine.print_output ( '**********************************' );
   dynsql_engine.print_output ( 'TEST CASE [ ' || l_test_case || ' ]' );
   dynsql_engine.print_output ( '**********************************' );
   dynsql_engine.print_output ( '**********************************' );
   dynsql_engine.print_output ( '.' );
--
-- Columns to include. Modify as needed
--
-- NOTE ** Must be defined in the HUMCUST.QCURSOR_DATASET_COLUMN_MSTR table.
--
-- If l_dynamic_column_list is N then it will read from table the assigned columns
-- HUMCUST.QCURSOR_DATASET_COLUMN_DETAIL
--
   l_column_list.DELETE;
   IF NVL ( :l_dynamic_column_list, 'N' ) = 'Y' THEN
      l_column_list ( 1 ) := 'EMPLOYEE_NUMBER';
      l_column_list ( 2 ) := 'FIRST_NAME';
      l_column_list ( 3 ) := 'LAST_NAME';
      l_column_list ( 4 ) := 'PERSON_TYPE';
      l_column_list ( 5 ) := 'COID';
      l_column_list ( 6 ) := 'UDN';
      l_column_list ( 7 ) := 'SUPERVISOR_NUMBER';
      l_column_list ( 8 ) := 'BASE_ROLE';
      l_column_list ( 9 ) := 'TERMINATION_DATE';
      l_column_list ( 10 ) := 'MIDDLE_NAMES';
      l_column_list ( 11 ) := 'SUFFIX';
      l_column_list ( 12 ) := 'DIRECT_REPORT_EXISTS';
      l_column_list ( 13 ) := 'KNOWN_AS';
      l_column_list ( 14 ) := 'DEPARTMENT';
      l_column_list ( 15 ) := 'HIRE_DATE';
   END IF;

--
-- TEST CASES
-- Modify as needed
--
   CASE
      WHEN NVL ( l_test_case, 0 ) = 0 THEN
         RAISE err_no_test_selected;
      WHEN l_test_case = 1 THEN
         dynsql_engine.print_output ( 'SEARCH FOR A SPECIFIC EMPLOYEE NUMBER' );
         dynsql_engine.print_output ( '.' );
--
--
         l_parameter_list ( 1 ) :=
                              '1|EMPLOYEE_NUMBER|NULL|NULL|10125|I|1|1|1|AND';
--
--
         l_parm_sql :=
               ' SELECT 1 parameter_entry_id, ''EMPLOYEE_NUMBER'' parameter_name, to_char(NULL) vdata_text, '
            || ' TO_DATE(NULL) vdata_date, 10125 vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 1 condition_member_id, ''AND'' condition_operator FROM DUAL '
            || '';
      WHEN l_test_case = 2 THEN
         dynsql_engine.print_output
            ( 'INCLUDE ORGTREE SUPERVISOR 8163 BUT ONLY THOSE THAT MATCH ROLE'
            );
         dynsql_engine.print_output ( '.' );
--
--
         l_parameter_list ( 1 ) := '1|ORGTREE|NULL|NULL|8163|I|1|1|1|AND';
         l_parameter_list ( 2 ) :=
                   '2|BASE_ROLE|Applications Consultant|NULL|NULL|I|1|1|2|AND';
--
--
         l_parm_sql :=
               ' SELECT 1 parameter_entry_id, ''ORGTREE'' parameter_name, to_char(NULL) vdata_text, '
            || ' TO_DATE(NULL) vdata_date, 8163 vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 1 condition_member_id, ''AND'' condition_operator FROM DUAL '
            || ' UNION ALL '
            || ' SELECT 2 parameter_entry_id, ''BASE_ROLE'' parameter_name, ''Applications Consultant'' vdata_text, '
            || ' TO_DATE(NULL) vdata_date, to_number(NULL) vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 2 condition_member_id, ''AND'' condition_operator FROM DUAL '
            || '';
      WHEN l_test_case = 3 THEN
         dynsql_engine.print_output
                      ( 'INCLUDE ORGTREE SUPERVISOR 8163 NOT INCLUDING 8163.' );
         dynsql_engine.print_output ( '.' );
--
--
         l_parameter_list ( 1 ) := '1|ORGTREE|NULL|NULL|8163|I|1|1|1|AND';
--
--
         l_parm_sql :=
               ' SELECT 1 parameter_entry_id, ''ORGTREE'' parameter_name,  to_char(NULL) vdata_text, '
            || ' TO_DATE(NULL) vdata_date, 8163 vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 1 condition_member_id, ''AND'' condition_operator FROM DUAL '
            || '';
      WHEN l_test_case = 4 THEN
         dynsql_engine.print_output
                          ( 'INCLUDE ORGTREE SUPERVISOR 8163 INCLUDING 8163.' );
         dynsql_engine.print_output ( '.' );
--
--
         l_parameter_list ( 1 ) := '1|ORGTREE|NULL|NULL|8163|I|1|1|1|OR';
         l_parameter_list ( 2 ) :=
                                '2|EMPLOYEE_NUMBER|NULL|NULL|8163|I|1|1|2|OR';
--
--
         l_parm_sql :=
               ' SELECT 1 parameter_entry_id, ''ORGTREE'' parameter_name,  to_char(NULL) vdata_text, '
            || ' TO_DATE(NULL) vdata_date, 8163 vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 1 condition_member_id, ''OR'' condition_operator FROM DUAL '
            || ' UNION ALL '
            || ' SELECT 2 parameter_entry_id, ''EMPLOYEE_NUMBER'' parameter_name,  to_char(NULL) vdata_text, '
            || ' TO_DATE(NULL) vdata_date, 8163 vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 2 condition_member_id, ''OR'' condition_operator FROM DUAL '
            || '';
      WHEN l_test_case = 5 THEN
         dynsql_engine.print_output
            ( 'INCLUDE NAMES LIKE xxxx . NOTICE SAME PARAMETER_ENTRY_ID IS USED WHICH ROLLS SIMILAR VALUESETS INTO ONE CONDITION'
            );
         dynsql_engine.print_output ( '.' );
--
--
         l_parameter_list ( 1 ) :=
                        '1|LAST_AND_FIRST_NAMES|BECKER#%|NULL|NULL|I|1|1|1|OR';
         l_parameter_list ( 2 ) :=
                       '1|LAST_AND_FIRST_NAMES|JOHNSON#%|NULL|NULL|I|1|1|1|OR';
--
--
         l_parm_sql :=
               ' SELECT 1 parameter_entry_id, ''LAST_AND_FIRST_NAMES'' parameter_name, ''BECKER#%'' vdata_text, '
            || ' to_date(NULL) vdata_date, to_number(NULL) vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 1 condition_member_id, ''OR'' condition_operator FROM DUAL '
            || ' UNION ALL '
            || ' SELECT 1 parameter_entry_id, ''LAST_AND_FIRST_NAMES'' parameter_name, ''JOHNSON#%'' vdata_text, '
            || ' to_date(NULL) vdata_date, to_number(NULL) vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 1 condition_member_id, ''OR'' condition_operator FROM DUAL '
            || '';
      WHEN l_test_case = 6 THEN
         dynsql_engine.print_output
                       ( 'INCLUDE EITHER FROM ONE ORGTREE OR THE OTHER OTHER' );
         dynsql_engine.print_output ( '.' );
--
--
         l_parameter_list ( 1 ) := '1|ORGTREE|NULL|NULL|537|I|1|1|1|OR';
         l_parameter_list ( 2 ) := '2|ORGTREE|NULL|NULL|11578|I|1|1|2|OR';
--
--
         l_parm_sql :=
               ' SELECT 1 parameter_entry_id, ''ORGTREE'' parameter_name,  to_char(NULL) vdata_text, '
            || ' TO_DATE(NULL) vdata_date, 537 vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 1 condition_member_id, ''OR'' condition_operator FROM DUAL '
            || ' UNION ALL '
            || ' SELECT 2 parameter_entry_id, ''ORGTREE'' parameter_name,  to_char(NULL) vdata_text, '
            || ' TO_DATE(NULL) vdata_date, 11578 vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 2 condition_member_id, ''OR'' condition_operator FROM DUAL '
            || '';
      WHEN l_test_case = 7 THEN
         dynsql_engine.print_output
                  ( 'INCLUDE ORGTREE BUT EXCLUDE ONE OF THE SUPERVISOR NODES' );
         dynsql_engine.print_output ( '.' );
--
--
```

```
         l_parameter_list ( 1 ) := '1|ORGTREE|NULL|NULL|8163|I|1|1|1|AND';
         l_parameter_list ( 2 ) :=
                             '2|SUPEVISOR_NUMBER|NULL|NULL|11578|E|2|1|1|AND';
--
--
         l_parm_sql :=
               ' SELECT 1 parameter_entry_id, ''ORGTREE'' parameter_name,  to_char(NULL) vdata_text, '
            || ' TO_DATE(NULL) vdata_date, 8163 vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 1 condition_member_id, ''AND'' condition_operator FROM DUAL '
            || ' UNION ALL '
            || ' SELECT 2 parameter_entry_id, ''SUPERVISOR_NUMBER'' parameter_name,  to_char(NULL) vdata_text, '
            || ' TO_DATE(NULL) vdata_date, 11578 vdata_number, ''E'' incl_excl_flag, '
            || ' 2 condition_block_id,1 condition_set_id, 1 condition_member_id, ''AND'' condition_operator FROM DUAL '
            || '';
      WHEN l_test_case = 8 THEN
         dynsql_engine.print_output
            (    'SAME AS BEFORE BUT WE LIMITED THE RESULT SET BY MAKING'
              || ' IT FALL INTO SEPERATE BLOCKS FORCING THE AND CONDITION TO BE APPLIED'
            );
         dynsql_engine.print_output ( '.' );
--
--
         l_parameter_list ( 1 ) := '1|ORGTREE|NULL|NULL|8163|I|1|1|1|AND';
         l_parameter_list ( 2 ) := '2|ORGTREE|NULL|NULL|11578|I|2|1|1|AND';
--
--
         l_parm_sql :=
               ' SELECT 1 parameter_entry_id, ''ORGTREE'' parameter_name,  to_char(NULL) vdata_text, '
            || ' TO_DATE(NULL) vdata_date, 8163 vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 1 condition_member_id, ''AND'' condition_operator FROM DUAL '
            || ' UNION ALL '
            || ' SELECT 2 parameter_entry_id, ''ORGTREE'' parameter_name,  to_char(NULL) vdata_text, '
            || ' TO_DATE(NULL) vdata_date, 11578 vdata_number, ''I'' incl_excl_flag, '
            || ' 2 condition_block_id,1 condition_set_id, 1 condition_member_id, ''AND'' condition_operator FROM DUAL '
            || '';
      WHEN l_test_case = 9 THEN
         dynsql_engine.print_output
            (    'NAMES LIKE xxxx ROLLING INTO SAME PARAMETER_ENTRY_ID'
              || ' (INCLUDING CHECKING THE KNOWN AS FIELD FOR FIRST NAME CHECKS)'
            );
         dynsql_engine.print_output ( '.' );
--
--
         l_parameter_list ( 1 ) :=
                         '1|LAST_AND_FIRST_NAMES|%#ROCKY|NULL|NULL|I|1|1|1|OR';
         l_parameter_list ( 2 ) :=
                       '1|LAST_AND_FIRST_NAMES|JOHNSON#%|NULL|NULL|I|1|1|1|OR';
```

```
--
--
         l_parm_sql :=
               ' SELECT 1 parameter_entry_id, ''LAST_AND_FIRST_NAMES'' parameter_name, ''%#ROCKY'' vdata_text, '
            || ' to_date(NULL) vdata_date, to_number(NULL) vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 1 condition_member_id, ''OR'' condition_operator FROM DUAL '
            || ' UNION ALL '
            || ' SELECT 1 parameter_entry_id, ''LAST_AND_FIRST_NAMES'' parameter_name, ''JOHNSON#%'' vdata_text, '
            || ' to_date(NULL) vdata_date, to_number(NULL) vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 1 condition_member_id, ''OR'' condition_operator FROM DUAL '
            || '';
      WHEN l_test_case = 10 THEN
         dynsql_engine.print_output
            ( 'NAMES LIKE xxx AND (EXCLUDE NAMES LIKE yyyy) AND ONLY THOSE IN DEPARTMENT zzzz'
            );
         dynsql_engine.print_output ( '.' );
--
--
         l_parameter_list ( 1 ) :=
                         '1|LAST_AND_FIRST_NAMES|%#ROCKY|NULL|NULL|I|1|1|1|OR';
         l_parameter_list ( 2 ) :=
                       '1|LAST_AND_FIRST_NAMES|JOHNSON#%|NULL|NULL|I|1|1|1|OR';
         l_parameter_list ( 3 ) :=
                         '1|LAST_AND_FIRST_NAMES|CHASE#%|NULL|NULL|I|1|1|1|OR';
         l_parameter_list ( 4 ) :=
                   '2|LAST_AND_FIRST_NAMES|JOHNSON#OLGA|NULL|NULL|E|2|1|1|AND';
         l_parameter_list ( 5 ) :=
                        '2|LAST_AND_FIRST_NAMES|ELLIS#%|NULL|NULL|E|2|1|1|AND';
         l_parameter_list ( 6 ) :=
                             '3|DEPARTMENT_LIKE|Payroll|NULL|NULL|I|3|1|1|AND';
--
--
         l_parm_sql :=
               ' SELECT 1 parameter_entry_id, ''LAST_AND_FIRST_NAMES'' parameter_name, ''%#ROCKY'' vdata_text, '
            || ' to_date(NULL) vdata_date, to_number(NULL) vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 1 condition_member_id, ''OR'' condition_operator FROM DUAL '
            || ' UNION ALL '
            || ' SELECT 1 parameter_entry_id, ''LAST_AND_FIRST_NAMES'' parameter_name, ''JOHNSON#%'' vdata_text, '
            || ' to_date(NULL) vdata_date, to_number(NULL) vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 1 condition_member_id, ''OR'' condition_operator FROM DUAL '
            || ' UNION ALL '
            || ' SELECT 1 parameter_entry_id, ''LAST_AND_FIRST_NAMES'' parameter_name, ''CHASE#%'' vdata_text, '
            || ' to_date(NULL) vdata_date, to_number(NULL) vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 1 condition_member_id, ''OR'' condition_operator FROM DUAL '
            || ' UNION ALL '
            || ' SELECT 2 parameter_entry_id, ''LAST_AND_FIRST_NAMES'' parameter_name, ''JOHNSON#OLGA'' vdata_text, '
            || ' to_date(NULL) vdata_date, to_number(NULL) vdata_number, ''E'' incl_excl_flag, '
            || ' 2 condition_block_id,1 condition_set_id, 1 condition_member_id, ''AND'' condition_operator FROM DUAL '
            || ' UNION ALL '
            || ' SELECT 2 parameter_entry_id, ''LAST_AND_FIRST_NAMES'' parameter_name, ''ELLIS#%'' vdata_text, '
            || ' to_date(NULL) vdata_date, to_number(NULL) vdata_number, ''E'' incl_excl_flag, '
            || ' 2 condition_block_id,1 condition_set_id, 1 condition_member_id, ''AND'' condition_operator FROM DUAL '
            || ' UNION ALL '
            || ' SELECT 3 parameter_entry_id, ''DEPARTMENT_LIKE'' parameter_name, ''Payroll'' vdata_text, '
            || ' to_date(NULL) vdata_date, to_number(NULL) vdata_number, ''I'' incl_excl_flag, '
            || ' 3 condition_block_id,1 condition_set_id, 1 condition_member_id, ''AND'' condition_operator FROM DUAL '
            || '';
      WHEN l_test_case = 11 THEN
         dynsql_engine.print_output
            (    'INCLUDE THOSE WITH HIRE_DATE BETWEEN xxxx and yyyy'
              || ' OR THOSE WITH HIRE_DATE BETWEEN aaaa and bbbb'
              || ' NOTE :: CHANGE TERMS SEARCH TO ALL AND THE TERM_DATE'
              || ' LIMITER BACK TO 01/01/2000 AND YOU WILL PULL UP THE'
              || ' INDIVIDUALS OF HIRE DATE BETWEEN 01-NOV-2000 and 30-NOV-2000'
            );
         dynsql_engine.print_output ( '.' );
--
--
         l_parameter_list ( 1 ) :=
            '1|HIRE_DATE_BTWN_DT1_DT2|01-SEP-2005#30-SEP-2005|NULL|NULL|I|1|1|1|OR';
         l_parameter_list ( 2 ) :=
            '2|HIRE_DATE_BTWN_DT1_DT2|01-NOV-2000#30-NOV-2000|NULL|NULL|I|1|1|2|OR';
--
--
         l_parm_sql :=
               ' SELECT 1 parameter_entry_id,''HIRE_DATE_BTWN_DT1_DT2'' parameter_name, ''01-SEP-2005#30-SEP-2005'' vdata_text, '
            || ' TO_DATE(NULL) vdata_date, TO_NUMBER(NULL) vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 1 condition_member_id, ''OR'' condition_operator FROM DUAL '
            || ' UNION ALL '
            || ' SELECT 2 parameter_entry_id,''HIRE_DATE_BTWN_DT1_DT2'' parameter_name, ''01-NOV-2000#30-NOV-2000'' vdata_text, '
            || ' TO_DATE(NULL) vdata_date, TO_NUMBER(NULL) vdata_number, ''I'' incl_excl_flag, '
            || ' 1 condition_block_id,1 condition_set_id, 2 condition_member_id, ''OR'' condition_operator FROM DUAL '
            || '';
      ELSE
         RAISE err_no_test_selected;
   END CASE;

--
--
   IF NVL ( :l_use_refcursor, 'N' ) = 'Y' THEN
      dynsql ( p_sql =>                         l_parm_sql
             , p_error =>                       l_dynsql_error
             , p_cur =>                         l_dynsql_cur
             );
```

```
--
--
      IF l_dynsql_error IS NOT NULL THEN
         dynsql_engine.print_output ( l_dynsql_error );
         RAISE err_dynsql_error;
      END IF;
   END IF;

--
-- Check global temp table for stored recs. Important to issue a COMMIT
-- before you run script again in order to clear out global temp table
--
   dynsql_engine.l_debug := 'Y';

--
--
   IF NVL ( :l_use_refcursor, 'N' ) = 'Y' THEN
      dynsql_engine.q_demo_pg_current_snap_c1
                                       ( p_process_desc =>                'GENERIC'
                                       , p_sub_process_desc =>            'GENERIC'
                                       , p_mode =>                        :l_mode
                                       , p_parameter_list =>              l_dynsql_cur
                                       , p_terms =>                       :l_terms
                                       , p_term_date =>                   :l_term_date_boundary
                                       , p_page =>                        :l_page
                                       , p_pagesize =>                    :l_pagesize
                                       , p_column_list =>                 l_column_list
                                       , p_success =>                     l_success
                                       , p_sql =>                         l_sql
                                       , p_cur =>                         :l_cur
                                       );
   ELSE
      dynsql_engine.q_demo_pg_current_snap_c2
                                      ( p_process_desc =>                'GENERIC'
                                      , p_sub_process_desc =>            'GENERIC'
                                      , p_mode =>                        :l_mode
                                      , p_parameter_list =>              l_parameter_list
                                      , p_terms =>                       :l_terms
                                      , p_term_date =>                   :l_term_date_boundary
                                      , p_page =>                        :l_page
                                      , p_pagesize =>                    :l_pagesize
                                      , p_column_list =>                 l_column_list
                                      , p_success =>                     l_success
                                      , p_sql =>                         l_sql
                                      , p_cur =>                         :l_cur
                                      );
   END IF;

   --
   IF UPPER ( NVL ( l_email_flag, 'N' )) = 'Y' THEN
      msg_length := DBMS_LOB.getlength ( l_sql );
      c := UTL_SMTP.open_connection ( l_smtp_server );
      UTL_SMTP.helo ( c, l_domain );
      UTL_SMTP.mail ( c, l_from_email_address );
      UTL_SMTP.rcpt ( c, l_to_email_address );
      UTL_SMTP.open_data ( c );
      send_header ( 'From'
                  ,    '"'
                    || l_from_email_address_name
                    || '" <'
                    || l_from_email_address
                    || '>'
                  );
      send_header ( 'To', l_to_email_address );
      send_header ( 'Subject'
                  , 'TESTING EMAIL ONLY :: TEST CASE [ ' || l_test_case
                    || ' ]'
                  );
--
      UTL_SMTP.write_data ( c, UTL_TCP.crlf );

--
      WHILE pos < msg_length LOOP
         UTL_SMTP.write_data ( c, DBMS_LOB.SUBSTR ( l_sql, offset, pos ));
         pos := pos + offset;
         offset := LEAST ( bytes_o_data, msg_length - offset );
      END LOOP;

--
      UTL_SMTP.close_data ( c );
      UTL_SMTP.quit ( c );
   END IF;

   dynsql_engine.print_output ( l_success );
--
--
-- DISPLAY SQL ON OUTPUT IF YOU CANNOT EMAIL SOMEWHERE
-- REMEMBER IT DOES NOT LOGICALLY CUTOFF AFTER COMPLETE WORDS
-- SO YOU MAY HAVE TO ADJUST THE RESULTING OUTPUT A LITTLE
-- BEFORE ATTEMPTING TO EXECUTE OR ANALYZE FURTHER
--
--   dynsql_engine.print_output ( '.' );
--   dynsql_engine.print_output ( '.' );
--   dynsql_engine.print_output ( '.' );
--   msg_length_dbms := DBMS_LOB.getlength ( l_sql );

--   WHILE pos_dbms < msg_length_dbms LOOP
--      dynsql_engine.print_output ( DBMS_LOB.SUBSTR ( l_sql
--                                                    , offset_dbms
--                                                    , pos_dbms
--                                                    ));
--      pos_dbms := pos_dbms + offset_dbms;
--      offset_dbms :=
--                    LEAST ( bytes_o_data_dbms, msg_length_dbms - offset_dbms );
--   END LOOP;
--
EXCEPTION
   WHEN err_dynsql_error THEN
      raise_application_error
                     ( -20000
                     , 'DYNSQL Error occurred. Check your test criteria SQL.'
                     );
   WHEN err_no_test_selected THEN
      raise_application_error
         ( -20000
         , 'Invalid Test Case Selected. Choose a valid l_test_case in script and rerun.'
         );
   WHEN UTL_SMTP.transient_error OR UTL_SMTP.permanent_error THEN
      BEGIN
         UTL_SMTP.quit ( c );
      EXCEPTION
         WHEN UTL_SMTP.transient_error OR UTL_SMTP.permanent_error THEN
            NULL;    -- When the SMTP server is down or unavailable, we don't
                     -- have a connection to the server. The quit call will
                     -- raise an exception that we can ignore.
      END;

      raise_application_error
                       ( -20000
                       ,    'Failed to send mail due to the following error: '
                         || SQLERRM
                       );
END;
--
-- TEST QUERIES
--
   /*
--
-- DISPLAY SUBMITTED CRITERIA GRID AS LONG AS COMMIT HAS NOT BEEN ISSUED.
--
```

```
SELECT NVL(RTRIM ( SUBSTR ( parameter_name
                                        , 1
                                        , INSTR ( parameter_name, '[' )
                                          - 1
                                        )
                               ),parameter_name) clipped,x.*
  FROM q_cursor_parms_global_temp x

--
-- DISPLAY CONDITIONAL LOGIC BUILD
--

SELECT   pgt.parameter_name parm_name
--       , ( CASE
--              WHEN TO_CHAR ( pgt.vdata_date, 'YYYY/MM/DD HH24:MI:SS' ) IS NOT NULL THEN TO_CHAR
--                                                                                           ( pgt.vdata_date
--                                                                                           , 'YYYY/MM/DD HH24:MI:SS'
--                                                                                           )
--              WHEN TO_CHAR ( pgt.vdata_number ) IS NOT NULL THEN TO_CHAR
--                                                                    ( pgt.vdata_number
--                                                                    )
--              ELSE pgt.vdata_text
--           END
--         ) parm_value
       , ( CASE
              WHEN (     NVL ( pgt.condition_set_id, '0' ) = 1
                     AND ( NVL
                              ( LEAD ( pgt.condition_block_id, 1 ) OVER ( ORDER BY pgt.condition_block_id
                               , pgt.condition_set_id
                               , pgt.condition_member_id
                               , pgt.parameter_entry_id
                               , qsc.column_name )
                              , '0'
                              ) <> NVL ( pgt.condition_block_id, '0' )
                         )
                     AND ( NVL
                              ( LAG ( pgt.condition_block_id, 1 ) OVER ( ORDER BY pgt.condition_block_id
                               , pgt.condition_set_id
                               , pgt.condition_member_id
                               , pgt.parameter_entry_id
                               , qsc.column_name )
                              , '0'
                              ) <> NVL ( pgt.condition_block_id, '0' )
                         )
                   )
               OR
```

```
(     NVL ( pgt.condition_set_id, '0' ) = 1
                     AND ( NVL
                              ( LEAD ( pgt.condition_block_id, 1 ) OVER ( ORDER BY pgt.condition_block_id
                               , pgt.condition_set_id
                               , pgt.condition_member_id
                               , pgt.parameter_entry_id
                               , qsc.column_name )
                              , '0'
                              ) = NVL ( pgt.condition_block_id, '0' )
                         )
                     AND ( NVL
                              ( LAG ( pgt.condition_block_id, 1 ) OVER ( ORDER BY pgt.condition_block_id
                               , pgt.condition_set_id
                               , pgt.condition_member_id
                               , pgt.parameter_entry_id
                               , qsc.column_name )
                              , '0'
                              ) <> NVL ( pgt.condition_block_id, '0' )
                         )
                     AND ( NVL
                              ( LEAD ( pgt.condition_set_id, 1 ) OVER ( ORDER BY pgt.condition_block_id
                               , pgt.condition_set_id
                               , pgt.condition_member_id
                               , pgt.parameter_entry_id
                               , qsc.column_name )
                              , '0'
                              ) <> NVL ( pgt.condition_set_id, '0' )
                         )
                   )
               OR
(     NVL ( pgt.condition_set_id, '0' ) > 1
                     AND ( NVL
                              ( LAG ( pgt.condition_block_id, 1 ) OVER ( ORDER BY pgt.condition_block_id
                               , pgt.condition_set_id
                               , pgt.condition_member_id
                               , pgt.parameter_entry_id
                               , qsc.column_name )
                              , '0'
                              ) = NVL ( pgt.condition_block_id, '0' )
                         )
                     AND pgt.condition_member_id = 1
                     AND ( NVL
                              ( LEAD ( pgt.condition_set_id, 1 ) OVER ( ORDER BY pgt.condition_block_id
                               , pgt.condition_set_id
                               , pgt.condition_member_id
                               , pgt.parameter_entry_id
                               , qsc.column_name )
```

```
                , '0'
                ) <> NVL ( pgt.condition_set_id, '0' )
           )
     )
     THEN 'OPEN_N_CLOSE_PARENTHESIS'
WHEN ( NVL
          ( LEAD ( pgt.condition_block_id, 1 ) OVER ( ORDER BY pgt.condition_block_id
           , pgt.condition_set_id
           , pgt.condition_member_id
           , pgt.parameter_entry_id
           , qsc.column_name )
          , '0'
          ) <> NVL ( pgt.condition_block_id, '0' )
     ) THEN 'CLOSE_PARENTHESIS'
WHEN ( NVL
          ( LEAD ( pgt.condition_block_id, 1 ) OVER ( ORDER BY pgt.condition_block_id
           , pgt.condition_set_id
           , pgt.condition_member_id
           , pgt.parameter_entry_id
           , qsc.column_name )
          , '0'
          ) = NVL ( pgt.condition_block_id, '0' )
     )
AND ( NVL
         ( LEAD ( pgt.condition_set_id, 1 ) OVER ( ORDER BY pgt.condition_block_id
          , pgt.condition_set_id
          , pgt.condition_member_id
          , pgt.parameter_entry_id
          , qsc.column_name )
         , '0'
         ) <> NVL ( pgt.condition_set_id, '0' )
    ) THEN 'CLOSE_PARENTHESIS'
WHEN NVL ( pgt.condition_member_id, '0' ) = 1 THEN 'OPEN_PARENTHESIS'
WHEN (     NVL
              ( LEAD ( pgt.parameter_entry_id, 1 ) OVER ( ORDER BY pgt.condition_block_id
               , pgt.condition_set_id
               , pgt.condition_member_id
               , pgt.parameter_entry_id
               , qsc.column_name )
              , '0'
              ) = NVL ( pgt.parameter_entry_id, '0' )
       AND NVL
              ( LAG ( pgt.parameter_entry_id, 1 ) OVER ( ORDER BY pgt.condition_block_id
               , pgt.condition_set_id
               , pgt.condition_member_id
               , pgt.parameter_entry_id
               , qsc.column_name )
```

```
                            , '0'
                            ) <> NVL ( pgt.parameter_entry_id, '0' )
                   )
              AND ( NVL
                       ( LAG ( pgt.condition_block_id, 1 ) OVER ( ORDER BY pgt.condition_block_id
                        , pgt.condition_set_id
                        , pgt.condition_member_id
                        , pgt.parameter_entry_id
                        , qsc.column_name )
                       , '0'
                       ) = NVL ( pgt.condition_block_id, '0' )
                  ) THEN 'OPEN_PARENTHESIS'
              ELSE 'MIDDLE'
           END
         ) row_flag
         ,(CASE WHEN  NVL
                       ( LEAD ( pgt.condition_block_id, 1 ) OVER ( ORDER BY pgt.condition_block_id
                        , pgt.condition_set_id
                        , pgt.condition_member_id
                        , pgt.parameter_entry_id
                        , qsc.column_name )
                       , '0'
                       ) = 0
                   AND (SELECT COUNT(distinct pgt1.condition_set_id) FROM humcust.q_cursor_parms_global_temp pgt1
                  WHERE pgt1.condition_block_id = pgt.condition_block_id) > 1 THEN
                  'Y'
                  ELSE
                  'N'
         END) last_in_multiset_block
       , pgt.condition_block_id block_id, pgt.condition_set_id set_id
       , pgt.condition_member_id member_id, pgt.condition_operator OPERATOR
       , DECODE ( pgt.incl_excl_flag
                , 'I', 'INCLUDE'
                , 'EXCLUDE'
                ) incl_excl_flag
                ,pgt.parameter_entry_id
    FROM qcursor_search_columns qsc
       , (select
parameter_entry_id,parameter_name,incl_excl_flag,condition_block_id,condition_set_id,condition_member_id,condition_operator
       from q_cursor_parms_global_temp group by
parameter_entry_id,parameter_name,incl_excl_flag,condition_block_id,condition_set_id,condition_member_id,condition_operator) pgt
   WHERE 1 = 1
     AND qsc.enabled_flag = 'Y'
     AND qsc.global_table = 'Q_CURSOR_PARMS_GLOBAL_TEMP'
     AND qsc.column_name =
            NVL ( RTRIM ( SUBSTR ( pgt.parameter_name
                                 , 1
                                 , INSTR ( pgt.parameter_name, '[' ) - 1
                                 )
                        )
                , pgt.parameter_name
                )
ORDER BY pgt.condition_block_id
       , pgt.condition_set_id
       , pgt.condition_member_id
       , pgt.parameter_entry_id
       , qsc.column_name;


   */
```

End of Document