

# Building Reports Using Oracle's Flexfield Support API

By Debra Addeo

*Editor's Note: One quick way to become the "most popular developer" on your Oracle® project is to automate mundane and repetitive tasks. Properly utilized, Flexfield Support API is one of the Oracle developer's most powerful tools. But as expert Debra Addeo has found, this tool does require an organized thought process in order to master. One key element behind the use of APIs is lexical parameters. If that last sentence made sense to you, then you're really going to enjoy this developer's guide to creating and testing Oracle Reports using this functionality.*

## Introduction

Reports that use cost codes, accounting segments, or any flexfield segment as a parameter can be more challenging to develop without the use of the FLEXSQL API. This API makes use of user exits that can create WHERE statements for SQL, select columns in a SQL statement, or be used for displaying segment values as prompts. The flexfield data can easily be displayed as segment values or descriptions with output restricted based on high - low value ranges.

**A report using the API for flexfield segments can decrease SQL.**

```
select    segment1, segment2, segment3
from      table
where segment1 >= (:p_input1_start)
and segment1 <= (:p_input1_end)
and segment2 >= (:p_input2_start)
and segment2 <= (:p_input2_end)
and segment3 >= (:p_input3_start)
and segment3 <= (:p_input3_end)
order by segment1, segment2, segment3
```

Figure 1: Sample SQL Query

```
select &lexical1 alias
from table
where &lexical2
order by &lexical3
```

Figure 2: Sample SQL Query Using API Logic

## Report API Setup

A report using the API for flexfield segments can decrease SQL.

Without the API, a query using flex segments would look something like Figure 1.

With the API, the same script uses a lexical parameter to display the information as shown in Figure 2.

The lexical parameter is designated by the "&" before the parameter name and is used as a place holder that will fill in the values at run time with information passed to the report or information that is static. A lexical parameter can appear after a SELECT, WHERE, FROM, ORDER BY, HAVING, CONNECT BY, or START WITH keyword in a SQL statement.

**Note:** A lexical cannot be used inside a PL/SQL block.

**After the API is called:**

- lexical1 could look like segment1, segment2, segment3 or segment1||separator||segment2||separator||segment3
- lexical2 could look like segment1 = 123 and segment2 between 123 and 456
- lexical3 could look like segment2, segment3, segment1.

**Note:** Parameters that are used with lexical values should have default or initial values to allow the queries to work properly.

Another option to using a lexical for the SELECT part of the statement is to use the field concatenated segments from the table, if available.

To use the API, the following triggers first need to be populated:

- after parameter form
- after report

- before report

The after parameter form trigger should look like Figure 3.

The after report trigger should look like Figure 4.

Most of the code for the call goes into the before report trigger and should look like Figure 5.

The first user exit gathers the names of the columns for the SELECT statement and places the information into the lexical parameter. The second user exit in Figure 5 puts together the WHERE clause for the SELECT statement and places the information into the lexical parameter.

If the after report code is placed into a different report trigger, there is a possibility that the user exit will not function properly and the lexical will not be filled in properly. The srw.message lines puts the information into the log file from the concurrent manager so that a log can be kept for reference if there are any problems and can be used for debugging the code.

Before the code can be compiled, define the following parameters:

```
function AfterPForm return boolean is
begin
  srw.user_exit('FND SRWINIT');
  return (TRUE);
end;
```

Figure 3: After Parameter Form Trigger

```
function AfterReport return boolean is
begin
  srw.user_exit('FND SRWEXIT');
  return (TRUE);
end;
```

Figure 4: After Report Trigger

```
function BeforeReport return boolean is
begin
  | srw.message(10,'start set-up of fnd flexsql');

  srw.reference(:P_STRUCT_NUM);

  srw.user_exit('FND FLEXSQL CODE="COST" NUM=:P_STRUCT_NUM"
              APPL_SHORT_NAME="PAY"
              OUTPUT=:P_SELECT_FLEX_SEGMENTS"
              TABLEALIAS="FLEX"
              MODE="SELECT" DISPLAY="ALL"');
  srw.message(10,'Flex segs: '||:P_select_flex_segments);

  srw.reference(:P_STRUCT_NUM);
  srw.reference(:P_MIN_FLEX);
  srw.reference(:P_MAX_FLEX);
  srw.user_exit('FND FLEXSQL CODE="COST" NUM=:P_STRUCT_NUM"
              APPL_SHORT_NAME="PAY"
              OUTPUT=:P_WHERE_FLEX"
              TABLEALIAS="FLEX"
              MODE="WHERE" DISPLAY="ALL"
              OPERATOR="BETWEEN"
              OPERAND1=:P_MIN_FLEX"
              OPERAND2=:P_MAX_FLEX"');

  srw.message(11,'Where: '||:P_where_flex);

  return (TRUE);

EXCEPTION
  WHEN NO DATA FOUND THEN
    return(FALSE);
  WHEN OTHERS THEN
    srw.message(?,sqlerrm);
    RETURN NULL;
end;
```

Figure 5: BeforeReport Trigger

- p\_struct\_num parameter – provides the id for the structure of the flexfields. 101 is used for the flexfield structure in the example.
- p\_where\_flex parameter – defines the conditions for the lexical placed for the WHERE field in the query. This should default to 1=1 in the event none of the segments are chosen as parameters.
- p\_min\_flex and p\_max\_flex – used as user parameters to input a range of values in the query as shown in the p\_where\_flex.

The FND FLEXSQL script is shown in Figure 6.

### FLEXSQL Script Glossary Code:

Figure 7 is a partial list of flexfield codes.

#### APPL\_Short\_Name:

This is the short name of the application that owns the flexfield. This is found in the FND\_Application table. For the payroll costing flexfield, the application short name is PAY; for GL, it is SQLGL.

#### Output:

Output is the parameter name that is used in the lexical parameter. This will go into the WHERE clause of the query.

```

FND FLEXSQL
CODE="flexfield code"
APPL_SHORT_NAME="application short name"
OUTPUT=":output lexical parameter name"
MODE="{ SELECT | WHERE | HAVING | ORDER BY}"
[DISPLAY="{ALL | flexfield qualifier | segment
number}"]
Reporting on Flexfields Data 8 - 23
[SHOWDEPSEG="{Y | N}"]
[NUM=":structure defining lexical" |
MULTINUM="{Y | N}"]
[TABLEALIAS="code combination table alias"]
[OPERATOR="{ = | < | > | <= | >= | != | "||" |
BETWEEN | QBE}"]
[OPERAND1=":input parameter or value"]
[OPERAND2=":input parameter or value"]
    
```

Figure 6: FLEXSQL Script

### Mode:

SELECT retrieves the segment values in a non-displayable format. If a select flexfield qualifier is chosen and that flexfield segment is dependent, then the flexfield automatically selects both the parent and child segment. Additionally, the lexical used for the SELECT statement can be used in the GROUP BY clause.

WHERE is used to restrict the query by specifying constraints on the flexfield columns.

HAVING has the same functionality as the WHERE clause.

ORDER BY is the list of fields, separated by a comma, to be used in the ORDER BY clause in the SQL statement.

### Display:

The display parameter allows the specification of which segments are needed. The value "all" will return all the segments. The display parameter can be used multiple times for "all" and then return other lines that would exclude segments.

For example DISPLAY="ALL"  
 DISPLAY="1" DISPLAY="2"  
 would display all but the first two segments.

### Showdepseg:

If this is set to "N", then it disables the automatic addition of dependent upon segment to the order criteria. The default is "Y" and is only valid for mode ORDER BY.

### Num or Multinum:

Specifies the name of the lexical or source column that contains the flexfield structure information.

### TableAlias:

This is the alias that is used by the table that contains the segments located in the SQL query.

Owner	Name	Code
Oracle Assets	Asset Key Flexfield	KEY#
Oracle Assets	Category Flexfield	CAT#
Oracle Assets	Location Flexfield	LOC#
Oracle General Ledger	Accounting Flexfield	GL#
Oracle Human Resources	Grade Flexfield	GRD
Oracle Human Resources	Job Flexfield	JOB
Oracle Human Resources	Personal Analysis Flexfield	PEA
Oracle Human Resources	Position Flexfield	POS
Oracle Human Resources	Soft Coded KeyFlexfield	SCL
Oracle Inventory	Account Aliases	MDSP
Oracle Inventory	Item Catalogs	MICG
Oracle Inventory	Item Categories	MCAT
Oracle Inventory	Sales Orders	RLOC
Oracle Inventory	Stock Locators	MTLL
Oracle Inventory	System Items	MSTK
Oracle Payroll	Bank Details KeyFlexField	BANK
Oracle Payroll	Cost Allocation Flexfield	COST
Oracle Payroll	People Group Flexfield	GRP
Oracle Receivables	Sales Tax Location Flexfield	MKTS
Oracle Receivables	Territory Flexfield	CT#
Oracle Service	Oracle Service Item Flexfield	SERV
Oracle Training Administration	Training Resources	RES

Figure 7: Flexfield Codes

### Operator:

Specifies which operator to use in the WHERE clause.

### Operand 1:

Used in the WHERE clause for a min value for a BETWEEN or, if there is no max value, then just the operand to use.

### Operand 2:

Used in the BETWEEN clause with the operator between.

If the report needs to display the description of the segment, this can be accomplished using another user exit.

Create formula columns `c_flexfield` and `c_desc_all` for this example. Additional columns may be used if the report requires it. These formula columns should correspond to the values and descriptions displayed in the report.

To retrieve the concatenated flexfield segment values and description to incorporate into the flexfields user exit, the above formula fields need to be defined. Pass the concatenated segments along with other information to the user exit; the user exit populates the concatenated values in this column as specified by the VALUE token. This would populate the segment values.

An example would look similar to Figure 8.

```
SRW.REFERENCE(:P_STRUCT_NUM);
SRW.REFERENCE(:C_FLEXDATA);
SRW.USER_EXIT('FND FLEXIDVAL
CODE="GL#"
NUM=":P_STRUCT_NUM"
APPL_SHORT_NAME="SQLGL"
DATA=":C_FLEXDATA"
VALUE=":C_FLEXFIELD"
DISPLAY="ALL");
RETURN(:C_FLEXFIELD);
```

Figure 8: Sample Query to Retrieve Descriptions

```
SRW.REFERENCE(:P_STRUCT_NUM);
SRW.REFERENCE(:P_FLEXDATA);

SRW.USER_EXIT('FND FLEXIDVAL
CODE="COST"
NUM=":P_STRUCT_NUM"
APPL_SHORT_NAME="PAY"
DATA=":C_FLEXDATA"
DESCRIPTION=":C_DESC_ALL"
DISPLAY="ALL");
RETURN(:C_DESC_ALL);
```

Figure 9: Sample Query FND FLEXIDVAL

The user exit FND FLEXIDVAL could also use descriptions. The code for that would look as in Figure 9.

The display information would be placed into `c_desc_all`.

After the report is defined, the wizard can be used to create the report layout, and the report added to the concurrent manager.

Once created, execute from the machine that has the Apps loaded. The user exits normally cannot be run on a client machine. It can be run through the concurrent manager for testing and using a flexfield pair to create the parameter screen needed to call the report.

### Concurrent Manager Setup

The concurrent manager is used to run the report. To use the segments of the flexfield for a parameter as input to the report, an application validation set needs to be defined.

If using a flex segment as a range of information (for example, a high and low value for a cost code or GL segment), then a pair would need to be set up to accomplish this.

***If the report needs to display the description of the segment, this can be accomplished using another user exit.***

To define a value set with validation type "pair", use the System Administrator responsibility and the navigation path: Application > Validation > Set (Figure 10).

The Validation Type would be a pair and the List Type would be a list of values. A pair is used when there would be a high and low range for a value.



Figure 10: Value Sets

Click on the Edit Information box and add the following:

On Edit and Validate, the following are set up for the cost code.

Once the above information is set up, then the parameters to run the report can be set up in the concurrent manager.

### Running the Report

When the report is run with a pair through the concurrent manager, the flexfield parameters will look like Figure 13. Once the pair is set up, then all that needs to be done is to assign it to the parameter, and it will use the segments to create the popup.

Once the parameters are filled in, then the lexical is created for which-ever parameters are filled in. For example, if Fund is filled in as 10 for high and low and Locations are filled in for a range between 10 and 20, then the lexical for the WHERE clause would look like:

Where segment1 = 10 and segment2 between 10 and 20

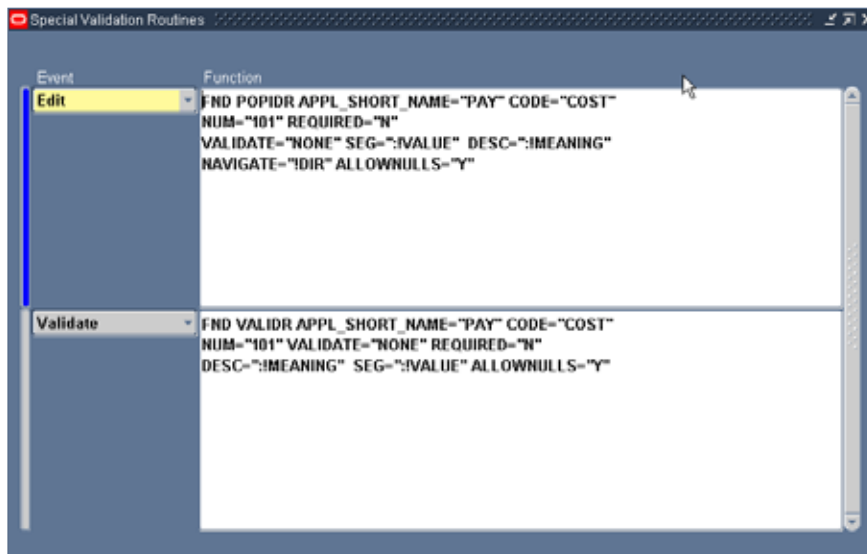


Figure 11: Edit Button from Value Sets

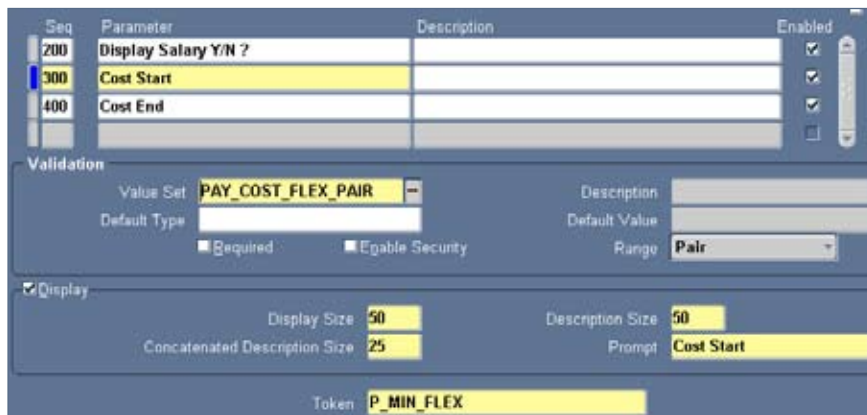



Figure 12: Concurrent Manager Parameters for the Report

### Conclusion

FLEXSQL can be used to replace multiple segments in a select statement in Oracle Reports. Once set up, this method allows reports to be created faster with lower maintenance.

**Debra Addeo, Douglas County Public Schools** – Debra has been an application developer and database administrator working primarily with Oracle for the past 15 years. Her experience includes development on UNIX and Windows NT, using various programming languages and tools. She is skilled at various aspects of Oracle development, performance tuning, and troubleshooting, as well as database administration and Web application development. Debra may be contacted at [Debra.Addeo@ERPtips.com](mailto:Debra.Addeo@ERPtips.com). 

### References

1. Oracle Applications Flexfields Guide
2. Oracle MetaLink ([metalink.oracle.com](http://metalink.oracle.com))

# **ORAtips** *Journal*

*The information on our website and in our publications is the copyrighted work of Klee Associates, Inc. and is owned by Klee Associates, Inc. NO WARRANTY: This documentation is delivered as is, and Klee Associates, Inc. makes no warranty as to its accuracy or use. Any use of this documentation is at the risk of the user. Although we make every good faith effort to ensure accuracy, this document may include technical or other inaccuracies or typographical errors. Klee Associates, Inc. reserves the right to make changes without prior notice. NO AFFILIATION: Klee Associates, Inc. and this publication are not affiliated with or endorsed by Oracle Corporation. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Klee Associates, Inc. is a member of the Oracle Partner Network*

**This article was originally published by Klee Associates, Inc., publishers of JDEtips and SAPtips. For training, consulting, and articles on JD Edwards or SAP, please visit our websites: [www.JDEtips.com](http://www.JDEtips.com) and [www.SAPtips.com](http://www.SAPtips.com).**