

# Oracle® Fusion Middleware

## Data Modeling Guide for Oracle Business Intelligence Publisher



12c (12.2.1.3.0)

E80603-02

November 2017

The Oracle logo, consisting of a solid red square with the word "ORACLE" in white, uppercase, sans-serif font centered within it.

ORACLE®

Oracle Fusion Middleware Data Modeling Guide for Oracle Business Intelligence Publisher, 12c (12.2.1.3.0)

E80603-02

Copyright © 2015, 2017, Oracle and/or its affiliates. All rights reserved.

Primary Author: Hemala Vivek

Contributing Authors: Suzanne Gill, Leslie Studdard, Reena Titus

Contributors: Oracle Business Intelligence Publisher development, quality assurance, and product management teams.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

# Contents

## Preface

---

Audience	ix
Documentation Accessibility	ix
Related Documentation and Other Resources	x
Conventions	x

## New Features for Data Model Designers

---

New Features and Changes for Oracle BI Publisher 12c (12.2.1.1.0)	xi
New Features and Changes for Oracle BI Publisher 12c (12.2.1)	xi

## 1 Using the Data Model Editor

---

What Is a Data Model?	1-1
Components of a Data Model	1-1
Features of the Data Model Editor	1-2
About the Data Source Options	1-2
Process Overview for Creating a Data Model	1-4
Launching the Data Model Editor	1-4
About the Data Model Editor Interface	1-5
Setting Data Model Properties	1-6
XML Output Options	1-8
Adding Attachments to the Data Model	1-8
Attaching Sample Data	1-9
Attaching Schema	1-9
Data Files	1-9
Managing Private Data Sources	1-9

## 2 Creating Data Sets

---

Overview of Creating Data Sets	2-1
Editing an Existing Data Set	2-2
Creating Data Sets Using SQL Queries	2-3

Entering SQL Queries	2-3
Creating Non-Standard SQL Data Sets	2-5
Using the SQL Query Builder	2-8
Overview of the Query Builder	2-8
Understanding the Query Builder Process	2-9
Supported Column Types	2-9
Adding Objects to the Design Pane	2-10
Removing or Hiding Objects in the Design Pane	2-10
Specifying Query Conditions	2-10
Creating Relationships Between Objects	2-12
Saving a Query	2-13
Editing a Saved Query	2-14
Adding a Bind Variable to a Query	2-15
Adding a Bind Variable Using a Text Editor	2-15
Adding Lexical References to SQL Queries	2-16
Defining SQL Queries Against the Oracle BI Server	2-18
Notes for Queries Against Oracle Fusion Applications Tables	2-19
Creating a Data Set Using a MDX Query Against an OLAP Data Source	2-20
Creating a Data Set Using a MDX Query	2-20
Using MDX Query Builder	2-21
Understanding the MDX Query Builder Process	2-21
Using the Select Cube Dialog	2-22
Selecting Dimensions and Measures	2-22
Adding Dimension Members to the Slicer/POV Axis	2-23
Performing MDX Query Actions	2-23
Applying MDX Query Filters	2-24
Selecting MDX Query Options and Saving MDX Queries	2-24
Creating a Data Set Using an Oracle BI Analysis	2-26
Additional Notes on Oracle BI Analysis Data Sets	2-27
Creating a Data Set Using a View Object	2-27
Additional Notes on View Object Data Sets	2-28
Creating a Data Set Using a Web Service	2-28
Web Service Data Source Options	2-29
Creating a Data Set Using a Simple Web Service	2-29
Creating a Data Set Using a Complex Web Service	2-32
Additional Information on Web Service Data Sets	2-33
Creating a Data Set Using an LDAP Query	2-33
Creating a Data Set Using a XML File	2-35
About Supported XML Files	2-35
Using a XML File Stored in a File Directory Data Source	2-35
Uploading a XML File Stored Locally	2-36

Refreshing and Deleting an Uploaded XML File	2-37
Creating a Data Set Using a Content Server	2-38
Creating a Data Set Using a Microsoft Excel File	2-38
Options for Uploading Excel Files to BI Publisher	2-39
About Supported Excel Files	2-39
Accessing Multiple Tables per Sheet	2-40
Using a Microsoft Excel File Stored in a File Directory Data Source	2-41
Uploading a Microsoft Excel File Stored Locally	2-42
Refreshing and Deleting an Uploaded Excel File	2-43
Creating a Data Set Using a CSV File	2-44
About Supported CSV Files	2-45
Creating a Data Set from a Centrally Stored CSV File	2-45
Uploading a CSV File Stored Locally	2-46
Editing the Data Type	2-47
Refreshing and Deleting an Uploaded CSV File	2-47
Creating a Data Set from an HTTP XML Feed	2-48
Creating a Data Set from an HTTP XML Data Set	2-48
Using Data Stored as a Character Large Object (CLOB) in a Data Model	2-49
How the Data Is Returned	2-50
Additional Notes on Data Sets Using CLOB Column Data	2-52
Handling XHTML Data Stored in a CLOB Column	2-52
Retrieving XHTML Data Wrapped in CDATA	2-52
Wrapping the XHTML Data in CDATA in the Query	2-53
Testing Data Models and Generating Sample Data	2-53
Including User Information Stored in System Variables in Your Report Data	2-54
Adding the User System Variables as Elements	2-55
Sample Use Case: Limit the Returned Data Set by User ID	2-55
Creating Bind Variables from LDAP User Attribute Values	2-56

### 3 Structuring Data

---

Working with Data Models	3-1
About Multipart Unrelated Data Sets	3-1
About Multipart Related Data Sets	3-3
Guidelines for Working with Data Sets	3-5
Features of the Data Model Editor	3-6
About the Interface	3-7
Creating Links Between Data Sets	3-10
About Element-Level Links	3-10
About Group-Level Links	3-11
Creating Element-Level Links	3-11

Deleting Element-Level Links	3-11
Creating Group-Level Links	3-12
Deleting Group-Level Links	3-14
Creating Subgroups	3-14
Moving an Element Between a Parent Group and a Child Group	3-16
Creating Group-Level Aggregate Elements	3-17
Creating Group Filters	3-22
Performing Element-Level Functions	3-24
Setting Element Properties	3-25
Sorting Data	3-26
Performing Group-Level Functions	3-26
The Group Action Menu	3-27
Editing the Data Set	3-28
Removing Elements from the Group	3-28
Editing the Group Properties	3-28
Performing Global-Level Functions	3-29
Adding a Global-Level Aggregate Function	3-30
Adding a Group-Level or Global-Level Element by Expression	3-31
Adding a Global-Level Element by PL/SQL	3-32
Using the Structure View to Edit Your Data Structure	3-33
Renaming Elements	3-34
Adding Value for Null Elements	3-34
Function Reference	3-34

## 4 Adding Parameters and Lists of Values

---

About Parameters	4-1
Adding a New Parameter	4-2
Creating a Text Parameter	4-3
Creating a Menu Parameter	4-4
Customizing the Display of Menu Parameters	4-6
Defining a Date Parameter	4-7
About Lists of Values	4-8
Adding Lists of Values	4-8
Creating a List from a SQL Query	4-9
Creating a List from a Fixed Data Set	4-11
Adding Flexfield Parameters	4-11
Prerequisites for Using Flexfields	4-12
Adding a Flexfield Parameter and List of Values	4-12
Adding the Flexfield List of Values	4-12
Adding the Menu Parameter for the Flexfield List of Values	4-13

Using the Flexfield Parameter to Pass Values to a Flexfield Defined in the Data Model	4-14
Referencing the Flexfield in the SQL Query	4-15
Passing a Range of Values	4-16

## 5 Adding Event Triggers

---

About Triggers	5-1
Adding Before Data and After Data Triggers	5-1
Order of Execution	5-2
Creating Schedule Triggers	5-3

## 6 Adding Flexfields

---

About Flexfields	6-1
Using Flexfields in Your Data Model	6-2
Adding Key Flexfields	6-2
Entering Flexfield Details	6-3
Adding Descriptive Flexfields	6-7
Including Descriptive Flexfield Reference in SQL Queries	6-8

## 7 Adding Bursting Definitions

---

About Bursting	7-1
What is the Bursting Definition?	7-1
Adding a Bursting Definition to Your Data Model	7-2
Attaching PDF to Reports using Bursting Engine	7-3
Defining the Query for the Delivery XML	7-4
Passing a Parameter to the Bursting Query	7-8
Defining the Split By and Deliver By Elements for a CLOB/XML Data Set	7-10
Configuring a Report to Use a Bursting Definition	7-12
Sample Bursting Query	7-12
Creating a Table to Use as a Delivery Data Source	7-13

## 8 Adding Custom Metadata for Oracle WebCenter Content Server

---

About Custom Metadata Mapping	8-1
Prerequisites	8-1
Mapping Data Fields to Custom Metadata Fields	8-1
Deleting Unused Metadata Fields	8-4

## 9 Performance Best Practices

---

Know Oracle WebLogic Server Default Time Out Setting	9-1
Best Practices for SQL Data Sets	9-1
Only Return the Data You Need	9-2
Use Column Aliases to Shorten XML File Length	9-2
Avoid Using Group Filters by Enhancing Your Query	9-2
Avoid PL/SQL Calls in WHERE Clauses	9-3
Avoid Use of the System Dual Table	9-3
Avoid PL/SQL Calls at the Element Level	9-3
Avoid Including Multiple Data Sets	9-4
Avoid Nested Data Sets	9-4
Avoid In-Line Queries as Summary Columns	9-5
Avoid Excessive Parameter Bind Values	9-5
Tips for Multi-value Parameters	9-6
Group Break and Sorting Data	9-6
Limit Lists of Values	9-8
Working with Lexicals/Flexfields	9-8
Working with Date Parameters	9-10
Run Report Online/Offline (Schedule)	9-10
Setting Data Model Properties to Prevent Memory Errors	9-10
Query Time Out	9-11
DB Fetch Size	9-11
Scalable Mode	9-12
SQL Pruning	9-13
SQL Query Tuning	9-13
Generate Explain Plan	9-14
Explain Plan for a Single Query	9-14
Explaining Plan for Reports	9-14
Guidelines for Tuning Queries	9-16
Tips for Database Tuning	9-16



# Preface

Welcome to Release 12c (12.2.1.3.0) of the *Data Modeling Guide for Oracle Business Intelligence Publisher*.

Oracle BI Publisher is an enterprise reporting solution for authoring, managing, and delivering all your highly formatted documents, such as operational reports, electronic funds transfer documents, government PDF forms, shipping labels, checks, sales and marketing letters, and much more.

## Audience

This guide describes how report authors use BI Publisher's data model editor to fetch and structure the data for use in the many different types of report layouts that BI Publisher supports.

The following table provides more information about using BI Publisher for other business roles.

Role	Sample Tasks	Guide
Administrator	Configuring Security Configuring System Settings Diagnosing and Monitoring System Processes	<i>Administrator's Guide for Oracle Business Intelligence Publisher</i>
Application developer or integrator	Integrating BI Publisher into existing applications using the application programming interfaces	<i>Developer's Guide for Oracle Business Intelligence Publisher</i>
Report consumer	Viewing reports Scheduling report jobs Managing report jobs	<i>User's Guide for Oracle Business Intelligence Publisher</i>
Report designer	Creating report definitions Designing layouts	<i>Report Designer's Guide for Oracle Business Intelligence Publisher</i>

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

### Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/>

[lookup?ctx=acc&id=info](#) or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

## Related Documentation and Other Resources

Oracle Business Intelligence features a variety of documentation and user assistance resources.

See the Oracle Business Intelligence documentation library for a list of related Oracle Business Intelligence documents.

In addition:

- Go to the Oracle Learning Library for Oracle Business Intelligence-related online training resources.
- Go to the Product Information Center Support note (Article ID 1338762.1) on My Oracle Support at <https://support.oracle.com>.

## System Requirements and Certification

Refer to the system requirements and certification documentation for information about hardware and software requirements, platforms, databases, and other information. Both of these documents are available on Oracle Technology Network (OTN).

The system requirements document covers information such as hardware and software requirements, minimum disk space and memory requirements, and required system libraries, packages, or patches:

<http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-requirements-100147.html>

The certification document covers supported installation types, platforms, operating systems, databases, JDKs, and third-party products:

<http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html>

## Conventions

This document uses text conventions, summarized below.

Convention	Meaning
<b>boldface</b>	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

# New Features for Data Model Designers

This preface describes changes to the Oracle BI Publisher data model designer.

- See [New Features and Changes for Oracle BI Publisher 12c \(12.2.1.1.0\)](#).
- See [New Features and Changes for Oracle BI Publisher 12c \(12.2.1\)](#).

## New Features and Changes for Oracle BI Publisher 12c (12.2.1.1.0)

This section includes new features and changes for Oracle BI Publisher 12c (12.2.1.1.0).

### **Content Server as a Data Source**

You can set up connections to the Content Server data source on the Administration page and then use in multiple data models. See [Creating a Data Set Using a Content Server](#).

### **BI Publisher REST API Support**

Use the BI Publisher REST API to programmatically run and manage reports. Click REST API in the BI Publisher page of Oracle Help Center.

## New Features and Changes for Oracle BI Publisher 12c (12.2.1)

New features and changes for Oracle BI Publisher 12c Release 1 (12.2.1) include:

- Generate Explain Plan from SQL Data Set
- Best Practices Information

### **Generate Explain Plan from SQL Data Set**

For data sets generated by SQL queries issued against the Oracle Database, you can now generate an Explain Plan to provide valuable information about the efficiency of your query. See [Generate Explain Plan](#).

### **Best Practices Information**

Poorly constructed data models can result in out-of-memory exceptions. Use these best practices guidelines to help you tune your data models for more efficient memory usage. See [Setting Data Model Properties to Prevent Memory Errors](#).

# 1

## Using the Data Model Editor

This topic describes the components and features supported by the data model editor.

### Topics:

- [What Is a Data Model?](#)
- [Components of a Data Model](#)
- [Features of the Data Model Editor](#)
- [About the Data Source Options](#)
- [Process Overview for Creating a Data Model](#)
- [Launching the Data Model Editor](#)
- [Setting Data Model Properties](#)
- [Managing Private Data Sources](#)

## What Is a Data Model?

A data model is an object that contains a set of instructions to retrieve and structure data for a pixel-perfect report. Data models reside as separate objects in the catalog.

A data model can be simple with one data set retrieved from a single data source (for example, the data returned from the columns in the employees table) or can be complex with parameters, triggers, and bursting definitions and using multiple data sets.

Use the data model editor to build a data model.

## Components of a Data Model

A data model supports the following components:

- Data set

A data set contains the logic to retrieve data from a single data source. A data set can retrieve data from a variety of data sources (for example, a database, an existing data file, a Web service call to another application, or a URL/URI to an external data provider). A data model can have multiple data sets from multiple sources.

- Event triggers

A trigger checks for an event. When the event occurs the trigger runs the PL/SQL code associated with it. The data model editor supports before data and after data triggers as well as schedule triggers. Before data and after data triggers consist of a call to execute a set of functions defined in a PL/SQL package stored in an Oracle database. A schedule trigger is executed for scheduled reports and tests for a condition that determines whether or not to run a scheduled report job.

- Flexfields  
A flexfield is a structure specific to Oracle Applications. The data model editor supports retrieving data from flexfield structures defined in your Oracle Application database tables.
- Lists of values  
A list of values is a menu of values from which report consumers can select parameter values to pass to the report.
- Parameters  
A parameter is a variable whose value can be set at runtime. The data model editor supports several parameter types.
- Bursting Definitions  
Bursting is a process of splitting data into blocks, generating documents for each data block, and delivering the documents to one or more destinations. A single bursting definition provides the instructions for splitting the report data, generating the document, and delivering the output to its specified destinations.
- Custom Metadata (for Web Content Servers)  
If you have configured a Web content server as a delivery destination and enabled custom metadata, the Custom Metadata component displays in the data model editor. Use this component to map data fields from your data model to the custom metadata fields set up for a set of rules defined in a Content Profile.

## Features of the Data Model Editor

Use the data model editor to combine data from multiple data sets from different data sources such as SQL, Excel files, Web services, HTTP feeds, and other applications into a single XML data structure. You can use data sets that are unrelated or establish a relationship between the data sets using a data link.

The data model editor enables you to perform the following tasks:

- Link data - Define master-detail links between data sets to build a hierarchical data model.
- Aggregate data - Create group level totals and subtotals.
- Transform data - Modify source data to conform to business terms and reporting requirements.
- Create calculations - Compute data values that are required for your report that are not available in the underlying data sources.

## About the Data Source Options

Data source types supported for creating data sets can be categorized into three general types.

### **Data set types that can use the full range of data model editor functions**

The full range of data model editor functions are supported for these data set types:

- SQL queries submitted against Oracle BI Server, an Oracle Database, or other supported databases. BI Publisher can retrieve the metadata information from these SQL queries.

See [Creating Data Sets Using SQL Queries](#).

For information on supported databases, see [System Requirements and Certification](#).

- Multidimensional (MDX) queries against an OLAP data source

See [Creating a Data Set Using a MDX Query Against an OLAP Data Source](#).

- Queries against your LDAP repository to retrieve user data

You can report on this data directly, or join this to data retrieved from other sources. See [Creating a Data Set Using an LDAP Query](#).

- Microsoft Excel spreadsheet data sources

You can store the Excel spreadsheet in a file directory set up as a data source by your administrator, or you can upload it directly from a local source to the data model. See [Creating a Data Set Using a Microsoft Excel File](#).

- XML data file data sources

You can store the XML file stored in a file directory set up as a data source by your administrator, or you can upload it directly from a local source to the data model. See [Creating a Data Set Using a XML File](#).

- CSV (comma separated value) file data sources

You can store the CSV file in a file directory set up as a data source by your administrator, or you can upload it directly from a local source to the data model. See [Creating a Data Set Using a CSV File](#).

#### **Data set types that can use partial data model editor functions**

BI Publisher can retrieve the column names and data type information from the data source of these data set types, but it cannot process or structure the data. Only a subset of the full range of data model editor functions are supported for data set types:

- Oracle BI Analysis  
See [Creating a Data Set Using an Oracle BI Analysis](#).
- View objects created using Oracle Application Development Framework (ADF).  
See [Creating a Data Set Using a View Object](#).

#### **Data set types that cannot be modified in the data model editor**

For these data set types, BI Publisher can retrieve the data generated and structured at the source. You cannot apply additional modifications in the data model editor for these data set types:

- HTTP XML feeds off the Web  
See [Creating a Data Set from an HTTP XML Feed](#).
- Web services  
See [Creating a Data Set Using a Web Service](#).

To use a Web service to return data for the report, supply the Web service WSDL to BI Publisher and then define the parameters in BI Publisher.

## Process Overview for Creating a Data Model

Follow the steps below to create a data table.

Step	Reference
Launch the data model editor.	<a href="#">Launching the Data Model Editor</a>
Set properties for the data model. (Optional)	<a href="#">Setting Data Model Properties</a>
Create the data sets for the data model.	<a href="#">Creating Data Sets</a>
Define the data output structure. (Optional)	<a href="#">Structuring Data</a>
Define the parameters to pass to the query, and define lists of values for users to select parameter values. (Optional)	<a href="#">Adding Parameters and Lists of Values</a>
Define Event Triggers. (Optional)	<a href="#">About Triggers</a>
(Oracle Applications Only) Define Flexfields. (Optional)	<a href="#">Adding Flexfields</a>
Test your data model and add sample data.	<a href="#">Testing Data Models and Generating Sample Data</a>
Add a bursting definition. (Optional)	<a href="#">Adding Bursting Definitions</a>
Map Custom Metadata for documents to be delivered to Web Content Servers (Optional)	<a href="#">Adding Custom Metadata for Oracle WebCenter Content Server</a>

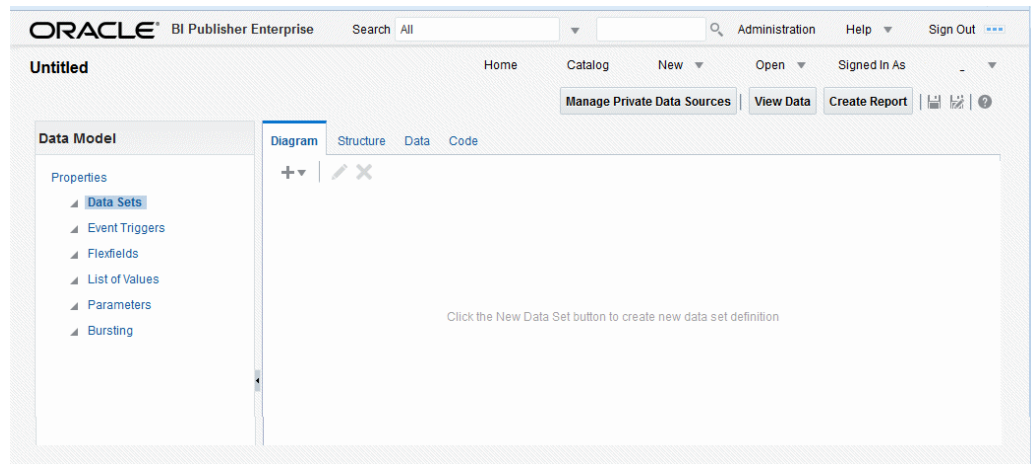
## Launching the Data Model Editor

Launch the data model editor from the header or from the Home page.

To launch the Data Model Editor:

- Use one of these ways:
  - Click **New** and then click **Data Model**.
  - Under the **Create** region, click **Data Model**.

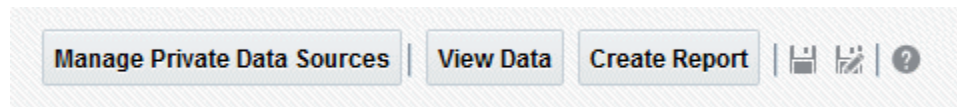
The Data Sets page displays as shown below.



## About the Data Model Editor Interface

The data model editor is designed with a component pane on the left and work pane on the right. Selecting a component on the left pane launches the appropriate fields for the component in the work area.

The data model editor toolbar, shown below, provides the following functions:



- **Manage Private Data Sources** — Connect to private data sources for your personal use that do not require setup by an administrator.
- **View Data** — Display the Data tab where you view and generate sample data.
- **Create Report** — Create a new report with this data model.
- **Save / Save As** — Select **Save** to save your work in progress to the existing data model object or select **Save As** to save the data model as a new object in the catalog.

### Note:

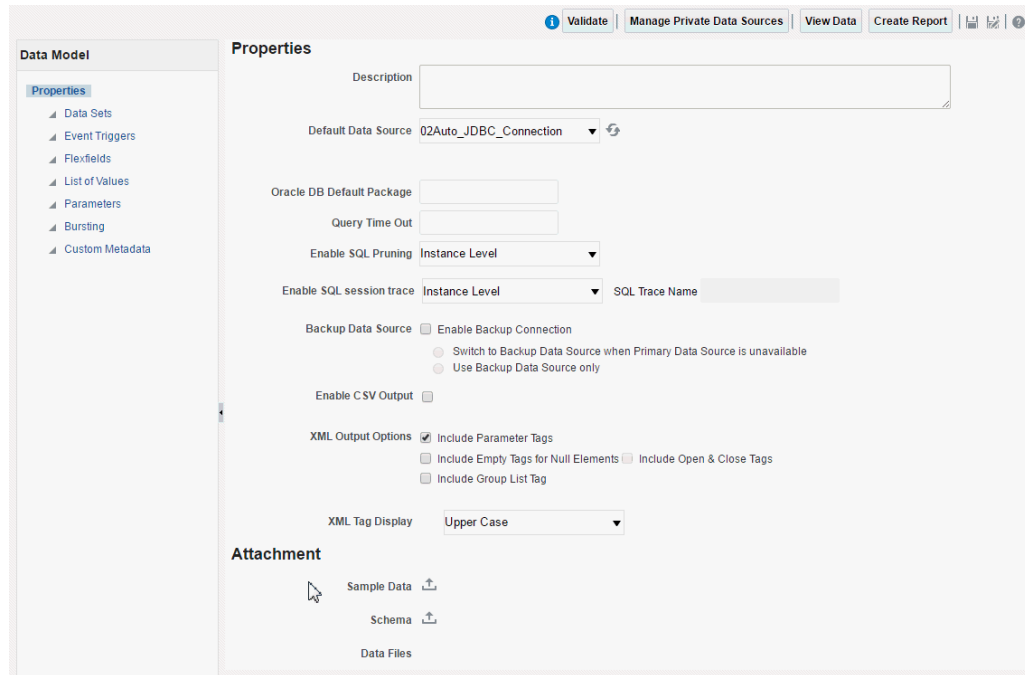
If you create a data model and then navigate out of the data model editor without saving it, a draft or temporary data model entry may display in the Recents section of the Home page. These entries cannot be manually deleted, but are automatically deleted after 24 hours.

- **Help** — View online help.



# Setting Data Model Properties

You can access the Data Model Properties page when you click **Properties** in the components pane.



Enter the following properties for the data model:

Property	Description
<b>Description</b>	The description that you enter here displays in the catalog. This description is translatable.
<b>Default Data Source</b>	Select the data source from the list. Data models can include multiple data sets from one or more data sources. The default data source you select here is presented as the default for each new SQL data set you define. Select <b>Refresh Data Source List</b> to see any new data sources added since your session was initiated.
<b>Oracle DB Default Package</b>	If you define a query against an Oracle Database, then you can include before or after data triggers (event triggers) in your data model. Event triggers make use of PL/SQL packages to execute RDBMS level functions. For data models that include event triggers or a PL/SQL group filter, you must enter a default PL/SQL package here. The package must exist on the default data source.
<b>Database Fetch Size</b>	Sets the number of rows fetched at a time through the JDBC connection. This value overrides the value set in the system properties. If neither this value nor the server setting is defined, then the server default value of 20 is used. If the server property <b>Enable Auto DB fetch size mode</b> is set to <b>True</b> , this value is ignored.

Property	Description
<b>Query Time Out</b>	Applies to SQL query-based data models. If the SQL query is still processing when the time out value is met, the error <code>Failed to retrieve data xml.</code> is returned. Enter a value in seconds. If you do not enter a value for this data model, the server property value is used.
<b>Scalable Mode</b>	<p>Processing large data sets requires the use of large amounts of RAM. To prevent running out of memory, activate scalable mode for the data engine. In scalable mode, the data engine takes advantage of disk space when it processes the data. Setting this to <b>On</b> will impact performance, but guard against out of memory errors.</p> <p>Note that <b>Enable Data Model Scalable Mode</b> is also a server-level property therefore by default the data model-level property is set to Instance Level to inherit the server or instance level setting. To turn scalable mode on or off for this particular data model, select <b>On</b> or <b>Off</b> from the list.</p>
<b>Enable SQL Pruning</b>	<p>Applies to Oracle Database queries only that use standard SQL. If your query returns many columns but only a subset are used by your report template, SQL pruning returns only those columns required by the template. Setting this property enhances processing time and reduces memory usage.</p> <p>Note that <b>Enable SQL Pruning</b> is also a server-level property therefore by default the data model-level property is set to Instance Level to inherit the server or instance level setting. To turn SQL pruning on or off for this particular data model, select <b>On</b> or <b>Off</b> from the list.</p> <p>SQL pruning is not applicable for PDF, Excel, and E-text template types.</p>
<b>Enable SQL Session Trace</b>	<p>Applies to Oracle Database queries that use standard SQL. If you enable trace, for each SQL statement, the trace contains:</p> <ul style="list-style-type: none"><li>• Parse, execute, and fetch counts</li><li>• CPU time and elapsed time</li><li>• Physical reads and logical reads</li><li>• Number of rows processed</li><li>• Library cache failures</li><li>• User name for which each parse occurred</li><li>• Each commit and rollback</li></ul>
<b>SQL Trace Name</b>	Name for the SQL trace.

Property	Description
<b>Backup Data Source</b>	<p>If you have set up a backup database for this data source, select <b>Enable Backup Connection</b> to enable the option; then select it when you want Oracle BI Publisher to use the backup.</p> <ul style="list-style-type: none"> <li>To use the backup data source only when the primary is down, select <b>Switch to Backup Data Source when Primary Data Source is unavailable</b>. Note that when the primary data source is down, the data engine must wait for a response before switching to the backup.</li> <li>To always use the backup data source when executing this data model, select <b>Use Backup Data Source Only</b>. Using the backup database may enhance performance.</li> </ul> <p>You must enable a backup for the data source.</p> <p>See Setting Data Engine Properties and About Backup Databases in <i>Administrator's Guide for Oracle Business Intelligence Publisher</i>.</p>
<b>Enable CSV Output</b>	Select this option to generate report output only in a CSV file.

## XML Output Options

These options define the characteristics of the XML data structure. Any changes to these options can impact layouts that are built on the data model.

- Include Parameter Tags** — If you define parameters for your data model, select this option to include the parameter values in the XML output file. See [Adding Parameters and Lists of Values](#) for adding parameters to your data model. Enable this option when you want to use the parameter value in the report.
- Include Empty Tags for Null Elements** — Select this option to include elements with null values in your output XML data. When you include a null element, then a requested element that contains no data in your data source is included in your XML output as an empty XML tag as follows: <ELEMENT\_ID\>. For example, if the element MANAGER\_ID contained no data and you chose to include null elements, it would appear in your data as follows: <MANAGER\_ID />. If you do not select this option, no entry is displayed for MANAGER\_ID.
- Include Open & Close Tags** — Select this option to include the open and close tags in your output XML data.
- Include Group List Tag** — (This property is for 10g backward compatibility and Oracle Report migration.) Select this option to include the rowset tags in your output XML data. If you include the group list tags, then the group list displays as another hierarchy within your data.
- XML Tag Display** — Select this option to generate the XML data tags in uppercase, in lowercase, or to preserve the definition you supplied in the data structure.

## Adding Attachments to the Data Model

The Attachment region of the page displays data files that you have uploaded or attached to the data model.

## Attaching Sample Data

After you build your data model, you must attach a small, but representative set of sample data generated from your data model. The sample data is used by BI Publisher's layout editing tools. Using a small sample file helps improve performance during the layout design phase.

The data model editor provides an option to generate and attach the sample data. See [Testing Data Models and Generating Sample Data](#).

The administrator can set a limit to the size of the sample data file. See [Setting Data Engine Properties](#) in *Administrator's Guide for Oracle Business Intelligence Publisher*.

## Attaching Schema

The data model editor enables you to attach sample schema to the data model definition.

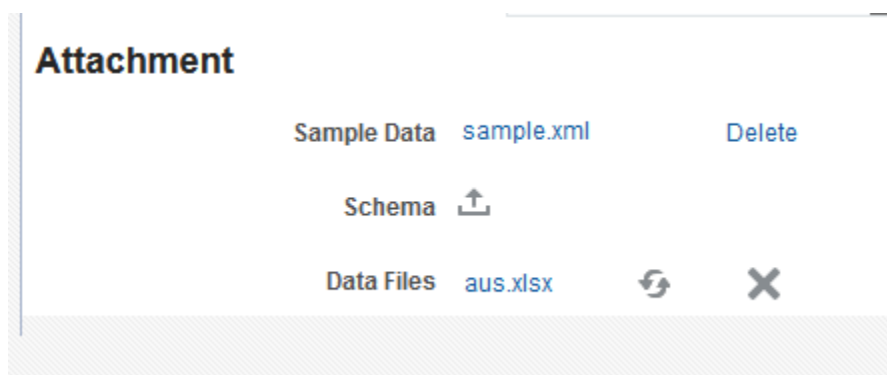
Oracle BI Publisher does not use the schema file. However, you can attach the schema for developer reference. The data model editor does not support schema generation.

## Data Files

If you have uploaded a local Microsoft Excel, CSV, or XML file as a data source for this report, the file displays here

Use the refresh button to refresh this file from the local source. For information on uploading files to use as data sources, see [Creating Data Sets](#).

The figure below shows the Attachments region with sample data and data files attached:



## Managing Private Data Sources

Data model developers can create and manage private JDBC, OLAP, Web service, and HTTP data source connections without having to depend on an Administrator

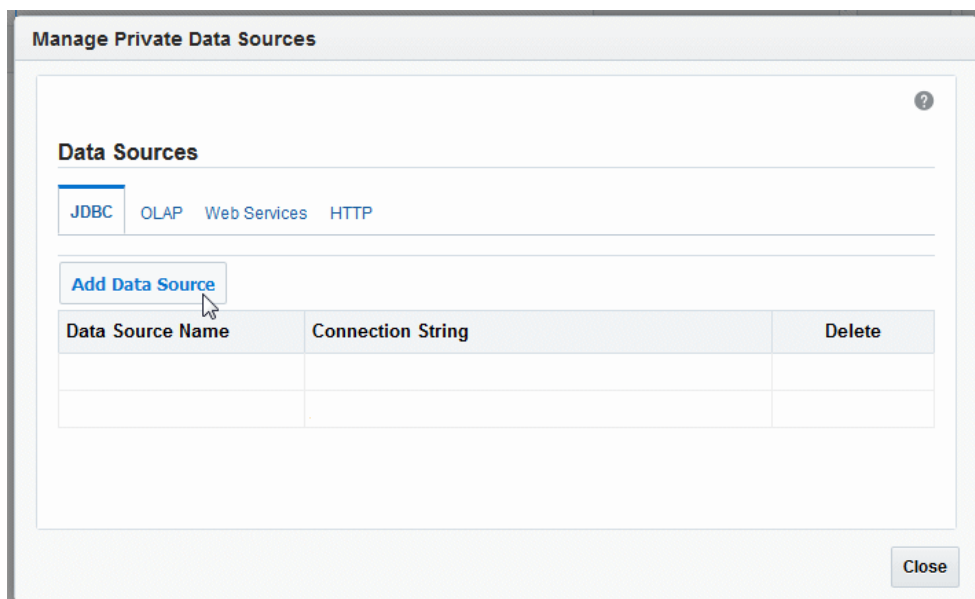
user. However, Administrator users can still view, modify, and delete private data source connections, if needed.

Private data source connections are identified by the word (Private) appended to the end of the data source name. For example, if you create a private JDBC connection called My JDBC Connection, it is displayed as My JDBC Connection (Private) in the data source drop-down lists.

If your user has the Administrator role, you can only create public data sources, even if you create the data source from the Manage Private Data Sources page. See About Private Data Source Connections and Setting Up Data Sources in *Administrator's Guide for Oracle Business Intelligence Publisher*.

To create a private data source connection:

1. From the data model editor toolbar, click **Manage Private Data Sources**.
2. Select the connection type tab, and click **Add Data Source** as shown below.



 **Note:**

If you are logged in as an Administrator, all data source connections will display for you in this dialog; however, you can only create or modify JDBC, OLAP, HTTP, and web service data sources from this dialog.

3. Enter the private connection name, and the connection information.
4. Click **Test Connection**. A confirmation is displayed.
5. Click **Apply**. The private data source connection is now available for use in your data sets.

# 2

## Creating Data Sets

This topic describes creating data sets, testing data models, and saving sample data.

### Topics:

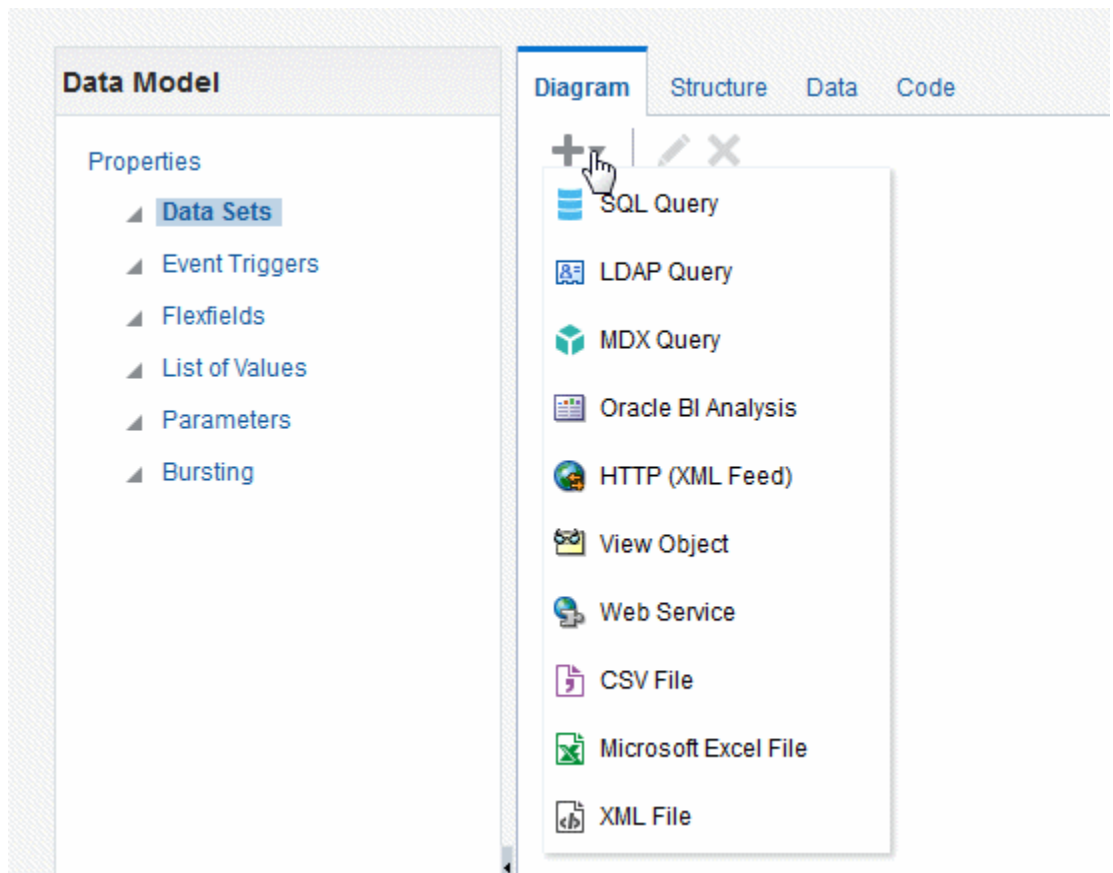
- [Overview of Creating Data Sets](#)
- [Editing an Existing Data Set](#)
- [Creating Data Sets Using SQL Queries](#)
- [Creating a Data Set Using a MDX Query Against an OLAP Data Source](#)
- [Using MDX Query Builder](#)
- [Creating a Data Set Using an Oracle BI Analysis](#)
- [Creating a Data Set Using a View Object](#)
- [Creating a Data Set Using a Web Service](#)
- [Creating a Data Set Using an LDAP Query](#)
- [Creating a Data Set Using a XML File](#)
- [Creating a Data Set Using a Microsoft Excel File](#)
- [Creating a Data Set Using a CSV File](#)
- [Creating a Data Set from an HTTP XML Feed](#)
- [Using Data Stored as a Character Large Object \(CLOB\) in a Data Model](#)
- [Testing Data Models and Generating Sample Data](#)
- [Including User Information Stored in System Variables in Your Report Data](#)

## Overview of Creating Data Sets

Oracle BI Publisher can retrieve data from multiple types of data sources.

To create a data set:

1. On the component pane of the data model editor, click **New Data Set** and select your source data set type.



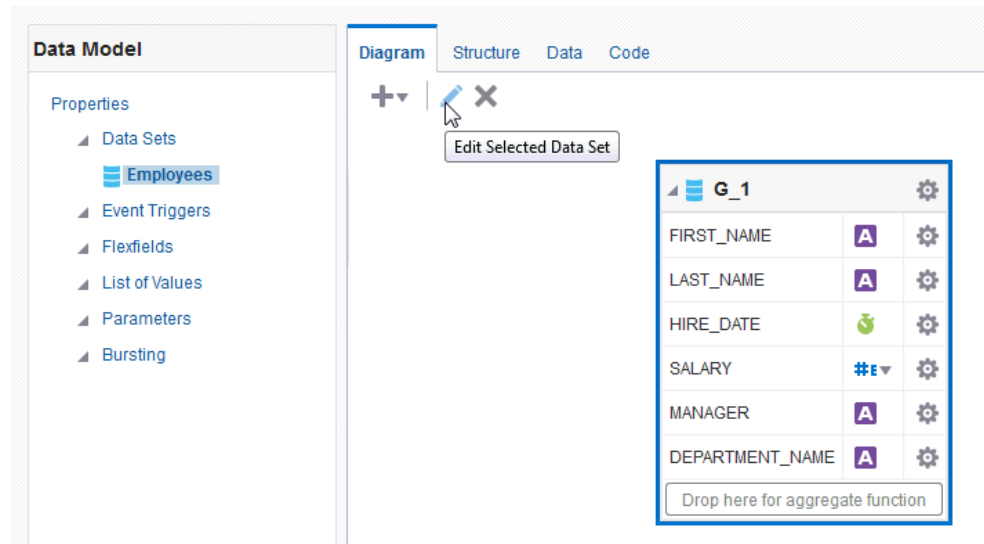
2. Complete the required fields. See the corresponding section:
  - [Creating Data Sets Using SQL Queries](#)
  - [Creating a Data Set Using a MDX Query Against an OLAP Data Source](#)
  - [Creating a Data Set Using an Oracle BI Analysis](#)
  - [Creating a Data Set Using a View Object](#)
  - [Creating a Data Set Using a Web Service](#)
  - [Creating a Data Set Using an LDAP Query](#)
  - [Creating a Data Set Using a XML File](#)
  - [Creating a Data Set Using a Microsoft Excel File](#)
  - [Creating a Data Set Using a CSV File](#)
  - [Creating a Data Set from an HTTP XML Feed](#)
  - [Using Data Stored as a Character Large Object \(CLOB\) in a Data Model](#)

## Editing an Existing Data Set

Modify data models by editing any of the data sets they contain.

1. On the component pane of the data model editor click **Data Sets**. All data sets for this data model display in the working pane.
2. Click the data set that you want to edit.

3. Click **Edit Selected Data Set**. The dialog for the data set opens. For information about each type of data set, see the corresponding section in this chapter.



4. Make changes to the data set and click **OK**.
5. Save the data model.
6. Test your edited data model and add new sample data. See [Testing Data Models and Generating Sample Data](#) for more information about testing and generating sample data.

## Creating Data Sets Using SQL Queries

These topics explain how to create data sets using SQL queries.

- [Entering SQL Queries](#)
- [Creating Non-Standard SQL Data Sets](#)
- [Using the SQL Query Builder](#)
- [Adding a Bind Variable to a Query](#)
- [Adding Lexical References to SQL Queries](#)
- [Defining SQL Queries Against the Oracle BI Server](#)

## Entering SQL Queries

Use these steps to enter SQL queries.

For information about optimizing your SQL Queries and the **Generate Explain Plan** option, see [Best Practices for SQL Data Sets](#).

1. Click **New Data Set** and then click **SQL Query**. The Create Data Set - SQL dialog opens, as shown below.



**New Data Set - SQL Query**

\* Name: Employees

\* Data Source: demo (Default)

\* Type of SQL: Standard SQL

\* SQL Query

```
select  "EMPLOYEES"."FIRST_NAME" as "FIRST_NAME",
        "EMPLOYEES"."LAST_NAME" as "LAST_NAME",
        "EMPLOYEES"."HIRE_DATE" as "HIRE_DATE",
        "EMPLOYEES"."SALARY" as "SALARY",
        "EMPLOYEES_1"."LAST_NAME" as "MANAGER",
        "DEPARTMENTS"."DEPARTMENT_NAME" as "DEPARTMENT_NAME"
from    "OE"."EMPLOYEES" "EMPLOYEES_1",
        "OE"."DEPARTMENTS" "DEPARTMENTS",
        "OE"."EMPLOYEES" "EMPLOYEES"
where  "EMPLOYEES"."MANAGER_ID"="EMPLOYEES_1"."MANAGER_ID"
and    "EMPLOYEES"."DEPARTMENT_ID"="DEPARTMENTS"."DEPARTMENT_ID"
```

Buttons: Generate Explain Plan, OK, Cancel

2. Enter a name for the data set.
3. The data source defaults to the default data source that you selected on the Properties page. If you are not using the default data source for this data set, select the **Data Source** from the list.

You can also use your private data source connections as data sources for SQL query data sets. See [Managing Private Data Sources](#) for information about private data source connections.

4. The SQL type defaults to **Standard SQL** used for normal SELECT statements interpreted to understand database schema. See [Creating Non-Standard SQL Data Sets](#) for information on using other types of SQL.
5. Enter the SQL query or click **Query Builder** to launch the Query Builder page. See [Using the SQL Query Builder](#) for more information about the Query Builder utility.
6. If you are using Flexfields, bind variables, or other special processing in your query, edit the SQL returned by the Query Builder to include the required statements.

**Note:**

If you include lexical references for text that you embed in a SELECT statement, then you must substitute values to get a valid SQL statement.

- After entering the query, click **OK** to save. For Standard SQL queries the data model editor validates the query.

If your query includes a bind variable, you are prompted to create the bind parameter. Click **OK** to have the data model editor create the bind parameter.

See [Adding Parameters and Lists of Values](#) for more information on editing parameters.

## Creating Non-Standard SQL Data Sets

In addition to creating data sets using basic SQL commands, you can create data sets using more complex commands.

### Procedure Call

Use this query type to call a database procedure. For example, Oracle PL/SQL statements start with `BEGIN`. When you use this SQL data type, no metadata is displayed on the data model structure tab, therefore you cannot modify the data structure or data fields. To construct your SQL with a procedure call enter the code directly in the text box or copy and paste from another SQL editor. You cannot use the Query Builder to modify or build these types of queries.

### Non-standard SQL

Use this query type to issue SQL statements that can include the following:

- Cursor statements that return nested results sets

For example:

```
Ex:SELECT TO_CHAR(sysdate,'MM-DD-YYYY') CURRENT_DATE ,
CURSOR
  (SELECT d.order_id department_id,
        d.order_mode department_name ,
        CURSOR
          (SELECT e.cust_first_name first_name,
                e.cust_last_name last_name,
                e.customer_id employee_id,
                e.date_of_birth hire_date
          FROM customers e
          WHERE e.customer_id IN (101,102)
          ) emp_cur
        FROM orders d
        WHERE d.customer_id IN (101,102)
        ) DEPT_CUR FROM dual
```

- Functions returning `REF` cursors

For example:

```
create or replace PACKAGE REF_CURSOR_TEST AS
  TYPE refcursor IS REF CURSOR;
  pCountry VARCHAR2(10);
  pState VARCHAR2(20);
  FUNCTION GET( pCountry IN VARCHAR2, pState IN VARCHAR2) RETURN
  REF_CURSOR_TEST.refcursor;
END;
```

```

create or replace PACKAGE BODY REF_CURSOR_TEST AS
FUNCTION GET(
  pCountry IN VARCHAR2,
  pState   IN VARCHAR2)
RETURN REF_CURSOR_TEST.refcursor
IS
l_cursor REF_CURSOR_TEST.refcursor;
BEGIN
  IF ( pCountry = 'US' ) THEN
    OPEN l_cursor FOR
      SELECT TO_CHAR(sysdate,'MM-DD-YYYY') CURRENT_DATE ,
             d.order_id department_id,
             d.order_mode department_name
      FROM orders d
      WHERE d.customer_id IN (101,102);
  ELSE
    OPEN l_cursor FOR
      SELECT * FROM EMPLOYEES;
  END IF;
  RETURN l_cursor;
END GET;
END REF_CURSOR_TEST;

```

To use REF cursor in Oracle BI Publisher:

```

create SQL dataset with query as SELECT REF_CURSOR_TEST.GET(:PCNTRY,:PSTATE) AS
CURDATA FROM DUAL

```

- Anonymous blocks/Stored procedures

BI Publisher supports executing PL/SQL anonymous blocks. You can perform calculations in the PL/SQL block and return the result set. BI Publisher uses callable statements to execute anonymous blocks.

The requirements are:

- The PL/SQL block must return a result set of type REF cursor
- You must declare the out variable with the name, `xdo_cursor`. If you do not declare the name properly, the first bind variable is treated as an out variable type and binds with REF cursor
- Declare the data model parameter with name `xdo_cursor`. This name is reserved for out variable type for procedure/anonymous blocks.

Example:

```

DECLARE
  type refcursor is REF CURSOR;
  xdo_cursor refcursor;
  empno number;
BEGIN
  OPEN :xdo_cursor FOR
    SELECT *
    FROM EMPLOYEES E
    WHERE E.EMPLOYEE_ID = :P2;
  COMMIT;
END;

```

- Conditional queries can be executed if you use an if-else expression. You can define multiple SQL queries in a single data set, but only one query executes at

run time depending on the expression value. The expression validates and returns a boolean value. If the value is true, executes that section of the SQL query.

The limitations are:

- The following syntax is supported to evaluate expressions: `$if{, $elseif{, $else{`
- The expression must return true, false
- Only the following operators are supported:  
`== <= >= < >`

Example:

```
create sql dataset with following query
$if{ (:P_MODE == PRODUCT) }$
    SELECT PRODUCT_ID
       ,PRODUCT_NAME
       ,CATEGORY_ID
       ,SUPPLIER_ID
       ,PRODUCT_STATUS
       ,LIST_PRICE
    FROM PRODUCT_INFORMATION
    WHERE ROWNUM < 5
$elseif{(:P_MODE == ORDER )}$
    SELECT ORDER_ID
       ,ORDER_DATE
       ,ORDER_MODE
       ,CUSTOMER_ID
       ,ORDER_TOTAL
       ,SALES_REP_ID
    FROM ORDERS
    WHERE ROWNUM < 5
$else{
    SELECT PRODUCT_ID
       , WAREHOUSE_ID
       ,QUANTITY_ON_HAND
    FROM INVENTORIES
    WHERE ROWNUM < 5
    }$
$endif$
```

When your data set is created using non-standard SQL statements, no metadata is displayed on the data model structure tab, therefore you cannot modify the data structure or data fields. You cannot use Query Builder to modify or build these types of queries.

To define XML row tag for non-standard SQL dataset:

Use `xmlRowTagName=""` in data model definition to define xml row tag for non-standard SQL query dataset. This allows you to enter a valid tag name. If the attribute is empty, it defaults to ROW at runtime.

Dataset definition:

```
<dataSet name="Q1" type="simple">
  <sql dataSourceRef="bipdev4-demo" nsQuery="true" xmlRowTagName="">
    ''
  </sql>
</dataset>
```

## Using the SQL Query Builder

Use the Query Builder to build SQL queries without coding. The Query Builder enables you to search and filter database objects, select objects and columns, create relationships between objects, and view formatted query results with minimal SQL knowledge.

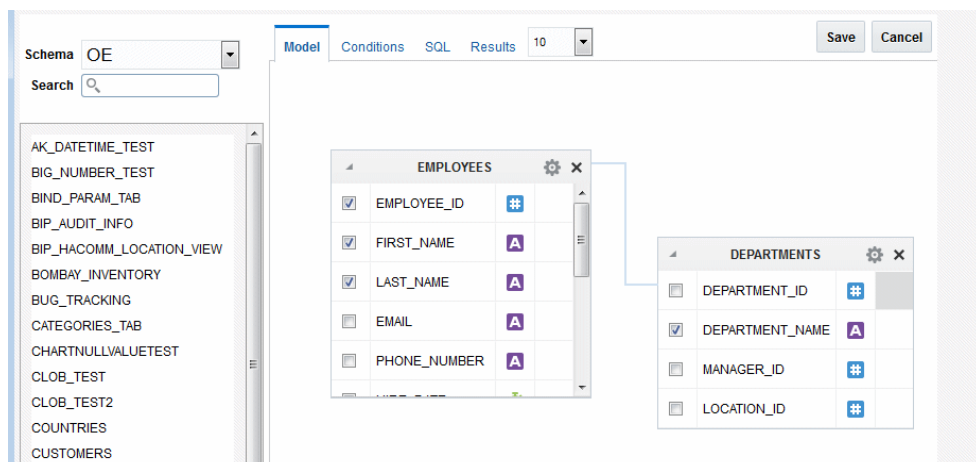
This section describes how to use the Query Builder and includes the following topics:

- [Overview of the Query Builder](#)
- [Understanding the Query Builder Process](#)
- [Supported Column Types](#)
- [Adding Objects to the Design Pane](#)
- [Removing or Hiding Objects in the Design Pane](#)
- [Specifying Query Conditions](#)
- [Creating Relationships Between Objects](#)
- [Saving a Query](#)
- [Editing a Saved Query](#)

### Overview of the Query Builder

The Query Builder page is divided into an Object Selection pane and a design and output pane.

- Object Selection pane contains a list of objects from which you can build queries. Only objects in the current schema display.
- Design and output pane consists of four tabs:
  - **Model** — Displays selected objects from the Object Selection pane.
  - **Conditions** — Enables you to apply conditions to your selected columns.
  - **SQL** — Displays the query.
  - **Results** — Displays the results of the query.



## Understanding the Query Builder Process

Perform these steps to build a query.

1. Select a schema.

The Schema list contains all the available schema in the data source. You might not have access to all that are listed.

2. Add objects to the Design pane and select columns.

The Object Selection pane lists the tables, views, and materialized views from the selected schema, for Oracle Database, synonyms are also listed. Select the object from the list and it displays on the Design pane. Use the Design pane to identify how the selected objects are used in the query.

You might need to use the Search field to enter a search string. If more than 100 tables are present in the data source, you must use the Search feature to locate and select the desired objects.

3. (Optional) Establish relationships between objects.
4. Add a unique alias name for any duplicate column.
5. (Optional) Create query conditions.
6. Execute the query and view results.

## Supported Column Types

Columns of all types display as objects in the Design pane.

The column restrictions are:

- You can select no more than 60 columns for each query.
- You can select the following column types:
  - VARCHAR2, CHAR
  - NUMBER
  - DATE, TIMESTAMP

### Note:

The `TIMESTAMP WITH LOCAL TIMEZONE` data type is not supported.

- Binary Large Object (BLOB)

### Note:

The BLOB must be an image. When you execute the query in the Query Builder, the BLOB does not display in the Results pane; however, the query is constructed correctly when saved to the data model editor. BI Publisher does not support querying of BLOB columns in an Oracle BI EE data source.

- Character Large Object (CLOB)



**Note:**

BI Publisher does not support querying of CLOB columns in an Oracle BI EE data source.

See [Using Data Stored as a Character Large Object \(CLOB\) in a Data Model](#) for more information about working with CLOB data in the data model.

## Adding Objects to the Design Pane

Select each object you want to add to the Design pane.

- When you add an object, an icon representing the data type displays next to each column name.
  - When you select a column, it appears on the **Conditions** tab. The **Show** check box on the **Conditions** tab controls whether a column is included in query results. By default, this check box is selected.
  - To select the first twenty columns, click the small icon in the upper left corner of the object and then select **Check All**.
  - You can also execute a query by pressing the **CTRL + ENTER** keys.
1. Select an object.
  2. Select the check box for each column to include in your query.
  3. To execute the query and view results, select **Results**.

## Removing or Hiding Objects in the Design Pane

You can remove or hide objects in the Design pane.

To remove an object:

1. Click **Remove** in the upper right corner of the object.

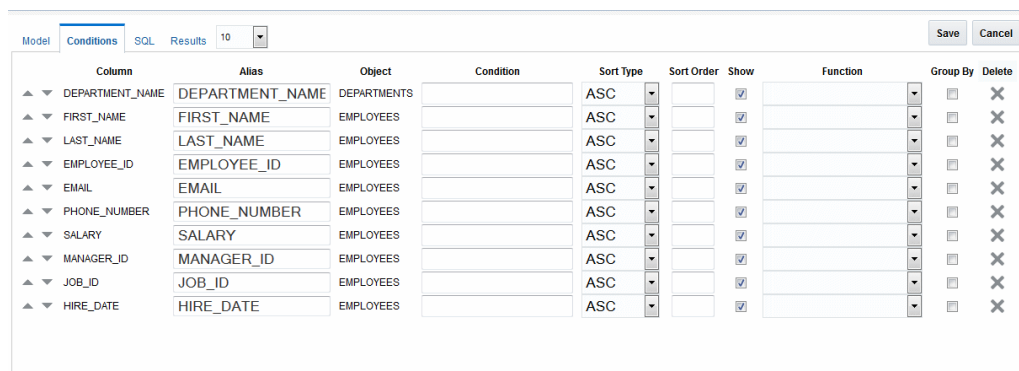
To temporarily hide the columns within an object:

1. Click **Show/Hide Columns**.

## Specifying Query Conditions

Conditions enable you to filter and identify the data you want to work with.

As you select columns within an object, you can specify conditions on the Conditions tab (shown below). You can use these attributes to modify the column alias, apply column conditions, sort columns, or apply functions.



The table below describes the attributes available on the Conditions tab.

Condition Attribute	Description
Up and Down Arrows	Controls the display order of the columns in the resulting query.
Column	Displays the column name.
Alias	Specify an optional column alias. An alias is an alternative column name. Aliases are used to make a column name more descriptive, to shorten the column name, or prevent possible ambiguous references. Note that multibyte characters are not supported in the alias name.
Object	Displays the object name.
Condition	The condition modifies the query's WHERE clause. When specifying a column condition, you must include the appropriate operator and operand. All standard SQL conditions are supported. For example: $\geq 10$ $= 'VA'$ IN (SELECT dept_no FROM dept) BETWEEN SYSDATE AND SYSDATE + 15
Sort Type	Select ASC (Ascending) or DESC (Descending).
Sort Order	Enter a number (1, 2, 3, and so on) to specify the order in which selected columns should display.
Show	Select this check box to include the column in your query results. You do not need to select Show to add a column to the query for filtering only. For example, to create following query: <pre>SELECT ename FROM emp WHERE deptno = 10</pre> <p><b>To create this query in Query Builder:</b></p> <ol style="list-style-type: none"> <li>From the Object list, select EMP.</li> <li>In the Design Pane, select ename and deptno.</li> <li>For the deptno column, in Condition enter <math>=10</math> and clear the <b>Show</b> check box.</li> </ol>



Condition Attribute	Description
Function	Available argument functions include: <ul style="list-style-type: none"> <li>• <b>Number columns</b> — COUNT, COUNT DISTINCT, AVG, MAXIMUM, MINIMUM, SUM</li> <li>• <b>VARCHAR2, CHAR columns</b> — COUNT, COUNT DISTINCT, INITCAP, LENGTH, LOWER, LTRIM, RTRIM, TRIM, UPPER</li> <li>• <b>DATE, TIMESTAMP columns</b>- COUNT, COUNT DISTINCT</li> </ul>
Group By	Specify columns to be used for grouping when an aggregate function is used. Only applicable for columns included in output.
Delete	Deselect the column, excluding it from the query.

As you select columns and define conditions, Query Builder writes the SQL for you.

To view the underlying SQL:

- Select the **SQL** tab.

## Creating Relationships Between Objects

You can create relationships between objects by creating a join. A join identifies a relationship between two or more tables, views, or materialized views.

- [About Join Conditions](#)
- [Joining Objects Manually](#)

### About Join Conditions

When you write a join query, you specify a condition that conveys a relationship between two objects. This condition is called a join condition.

A join condition specifies how the rows from one object combine with the rows from another object.

Query Builder supports inner, outer, left, and right joins.

- An inner join, also called a simple join, returns the rows that satisfy the join condition.
- An outer join extends the result of a simple join.

An outer join returns all rows that satisfy the join condition and returns some or all of those rows from one table for which no rows from the other satisfy the join condition.

### Joining Objects Manually

Create a join manually by selecting the Join column in the Design pane.

To join objects manually:

1. From the Object Selection pane, select the objects you want to join.
2. Identify the columns you want to join.

You create a join by selecting the **Join** column adjacent to the column name. The **Join** column displays to the right of the data type. When your cursor is in the appropriate position, the following help tip displays:

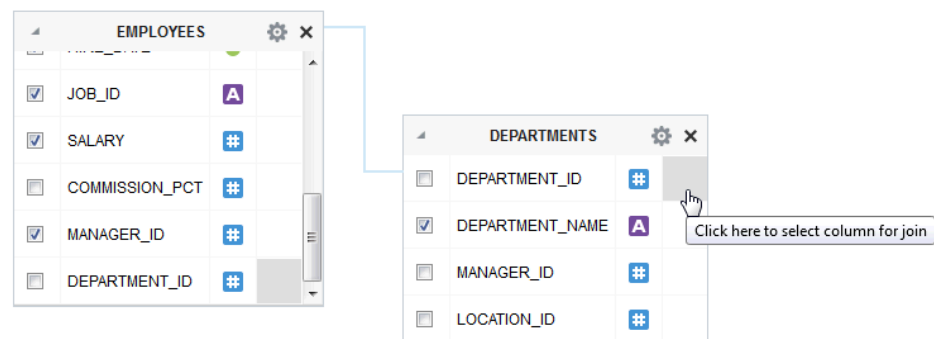
Click here to select column for join

3. Select the appropriate **Join** column for the first object.

When selected, the **Join** column is darkened. To deselect a **Join** column, simply select it again or press **ESC**.

4. Select the appropriate **Join** column for the second object.

When joined, line connects the two columns. An example is shown below.

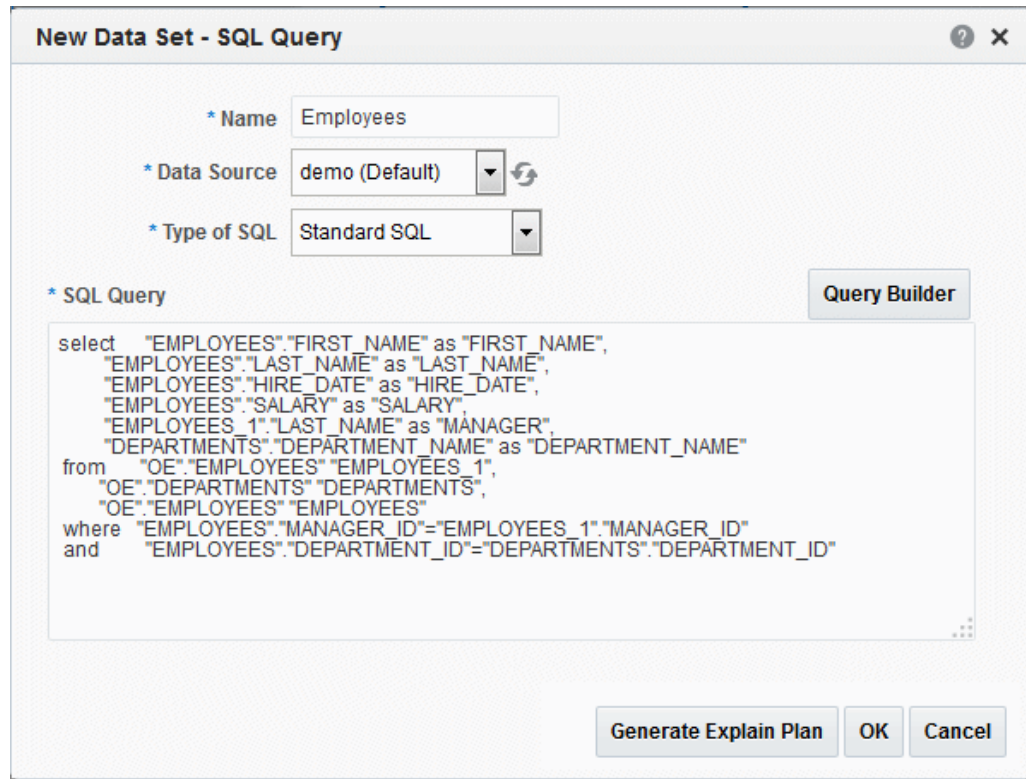


5. Select the columns to be included in your query. You can view the SQL statement resulting from the join by positioning the cursor over the join line.
6. Click **Results** to execute the query.

## Saving a Query

Save queries after building them.

Once you have built the query, click **Save** to return to the data model editor. The query appears in the SQL Query box. Click **OK** to save the data set.



## Editing a Saved Query

When you have saved the query from the Query Builder to the data model editor, you can also use the Query Builder to edit the query.

If you have made modifications to the query, or did not use the Query Builder to construct it, you may receive an error when launching the Query Builder for editing the query. If the Query Builder cannot parse the query, you can edit the statements directly in the text box.

### Note:

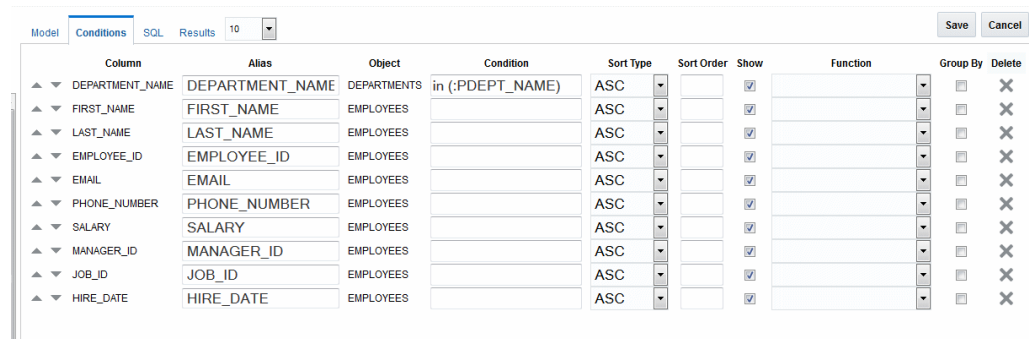
You cannot edit a customized or an advanced query by using Query Builder.

1. Select the SQL data set.
2. On the toolbar, click **Edit Selected Data Set** to launch the **Edit Data Set** dialog.
3. Click **Query Builder** to load the query to the Query Builder.
4. Edit the query and click **Save**.

## Adding a Bind Variable to a Query

After creating a query, you may want users to be able to pass a parameter to the query to limit the results. For example, in the employee listing, you want users to be able to choose a specific department.

The image shows the columns in the department table.



- In the Query Builder Conditions tab, add the bind variable for the column using the following:

```
in (:PARAMETER_NAME)
```

Where PARAMETER\_NAME is the name you choose for the parameter.

### Note:

After manually editing the query, the Query Builder can no longer parse it. You must make any additional edits manually.

## Adding a Bind Variable Using a Text Editor

Use the Data Model Editor to update a SQL query.

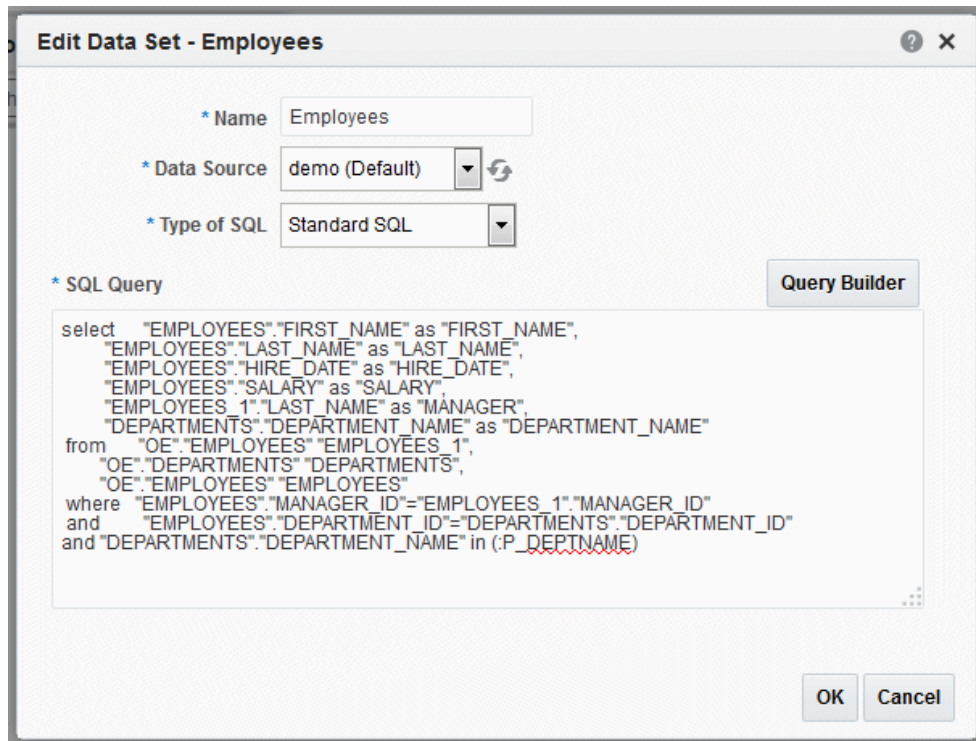
- In the Edit Data Set text box, update the SQL query by adding the following after the where clause in your query:

```
and "COLUMN_NAME" in (:PARAMETER_NAME)
```

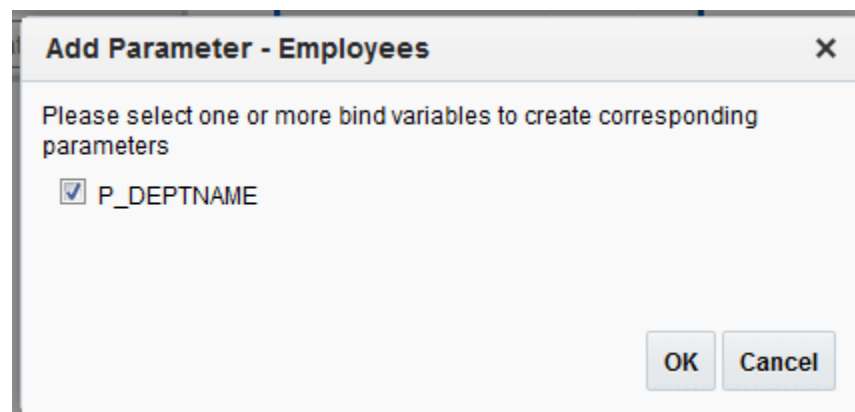
for example:

```
and "DEPARTMENT_NAME" in (:P_DEPTNAME)
```

where P\_DEPTNAME is the name you choose for the parameter, as shown below.



2. Click Save.
3. In the data model editor, create the parameter that you entered with the bind variable syntax as shown in the image.



4. Select the parameter, and click **OK** to enable the data model editor create the parameter entry for you.

See [Adding Parameters and Lists of Values](#) for more information on defining parameter properties.

## Adding Lexical References to SQL Queries

You can use lexical references to replace the clauses appearing after **SELECT**, **FROM**, **WHERE**, **GROUP BY**, **ORDER BY**, or **HAVING**.

Use a lexical reference when you want the parameter to replace multiple values at runtime. Lexical references are supported in queries against Oracle applications only.

Create a lexical reference in the SQL query using the following syntax:

```
&parametername
```

 **Note:**

You also use lexical references to include flexfields in your query. For more information about using flexfields, see [Adding Flexfields](#).

1. Before creating your query, define a parameter in the PL/SQL default package for each lexical reference in the query. The data engine uses these values to replace the lexical parameters.
2. In the data model editor, on the Properties page, specify the **Oracle DB Default Package**.
3. In the data model editor, create a **Before Data** event trigger to call the PL/SQL package. See [Adding Before Data and After Data Triggers](#) for more information about procedures.
4. Create your SQL query containing lexical references.
5. When you click **OK** to close your SQL query, you are prompted to enter the parameter.

For example, create a package called `employee`. In the `employee` package, define a parameter called `where_clause`:

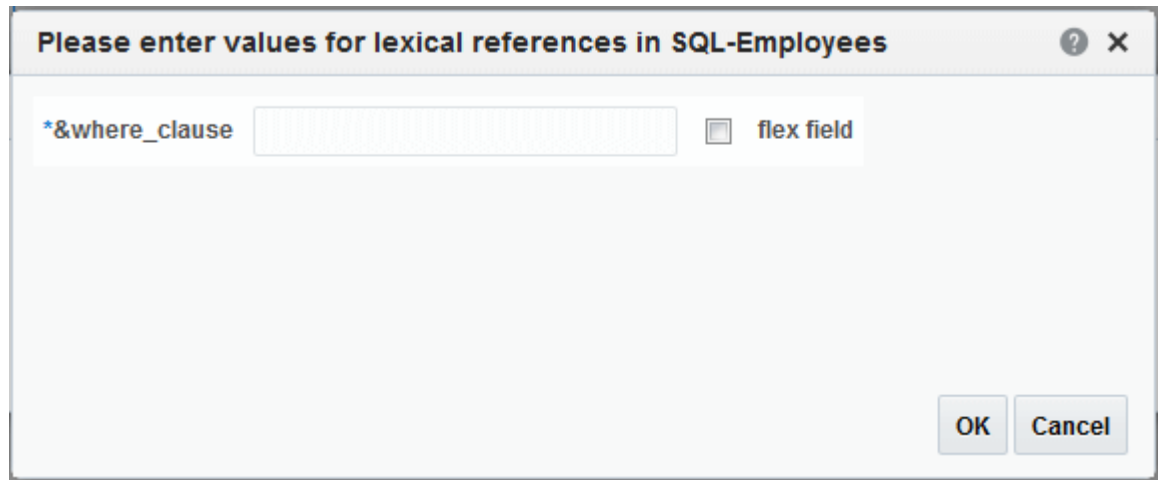
```
Package employee
AS
  where_clause varchar2(1000);
  ....

Package body employee
AS
  ....
where_clause := 'where DEPARTMENT_ID=10';
  ....
```

Reference the lexical parameter in the SQL query where you want the parameter to be replaced by the code defined in the package, for example:

```
select      "EMPLOYEES"."EMPLOYEE_ID" as "EMPLOYEE_ID",
           "EMPLOYEES"."FIRST_NAME" as "FIRST_NAME",
           "EMPLOYEES"."LAST_NAME" as "LAST_NAME",
           "EMPLOYEES"."SALARY" as "SALARY",
from        "OE"."EMPLOYEES" "EMPLOYEES"
&where_clause
```

When you click **OK** on the Create SQL Data Set dialog box, the lexical reference dialog box prompts you to enter a value for lexical references you entered in the SQL query, as shown in the image that follows. Enter the value of the lexical reference as it is defined in the PL/SQL package.



At runtime, the data engine replaces `&where_clause` with the contents of `where_clause` defined in the package.

## Defining SQL Queries Against the Oracle BI Server

When you launch the Query Builder against the Oracle BI Server, the Query Builder displays the subject areas from the catalog. You can drag the subject areas to the Query Builder workspace to display the columns. Select the columns to include in your data model.

Keep the following points in mind when creating a data set against the Oracle BI Server:

- When you create a SQL query against the Oracle BI Server using the SQL Data Editor or the Query Builder, logical SQL is generated, not physical SQL like other database sources.
- Hierarchical columns are not supported. The highest level is always returned.
- Within a subject area, the join conditions between tables are already created; it is therefore not necessary to create joins in the Query Builder. The Query Builder does not expose the primary key.

It is possible to link data sets using the data model editor's **Create Link** function. See [Creating Element-Level Links](#). For data sets created from the Oracle BI Server, there is a limit of two element-level links for a single data model.

- In the Query Builder, the functions **Sort Order** and **Group By** shown on the Conditions tab are not supported for queries against the Oracle BI Server. If you enter a Sort Order or select the Group By check box, the Query Builder constructs the SQL, and write it to the BI Publisher SQL Query text box, but when you attempt to close the Data Set dialog, the query fails validation.

To apply grouping to the data retrieved by the SQL query, you can use the data model editor's **Group by** function instead. See [Creating Subgroups](#).

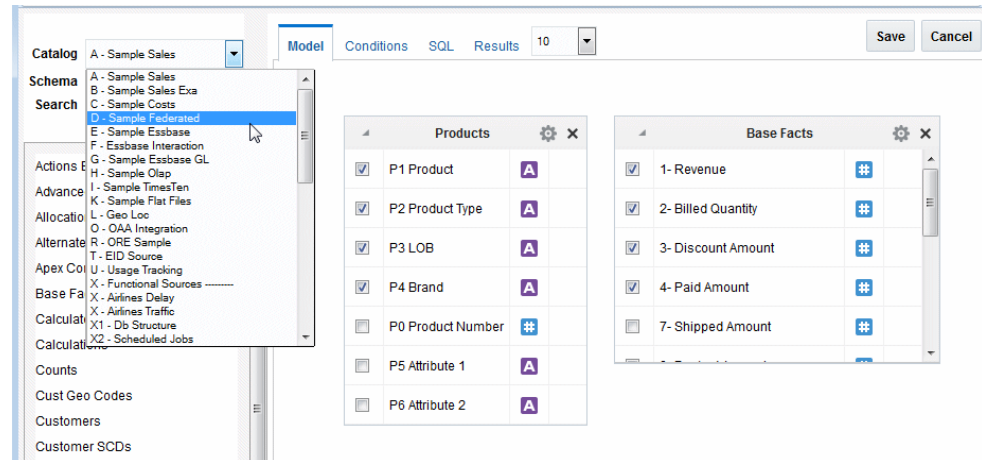
- If you pass parameters to the Oracle BI Server and you choose Null Value Passed for Can Select All, you must ensure that you handle the null value in your query.

1. In the data model editor, click **New Data Set** and then click **SQL Query**.
2. Enter a name for the data set.

- From the Data Source list, select the Oracle BI Server connection, usually shown as Oracle BI EE.
- Click **Query Builder** to launch the Query Builder page. See [Using the SQL Query Builder](#) for more information about the Query Builder utility.

You can also enter the SQL syntax manually in the SQL Query text box, however you must use the Logical SQL syntax used by the Oracle BI Server.

- From the Catalog drop-down list, select a subject area as shown below. The list displays the subject areas defined in the Oracle BI Server.



- Select tables and columns for the query.
- Click **Save**.
- Click **OK** to return to the data model editor. The SQL that is generated is Logical SQL that follows a star schema. It is not physical SQL.
- Save your changes to the data model.

## Notes for Queries Against Oracle Fusion Applications Tables

Special considerations for Oracle Fusion Applications customers apply when writing queries against the Oracle Fusion Applications tables

- You cannot return month name from `sysdate` using `to_char(sysdate, "mon")`. This function returns the month number. To display month name, use one of the following solutions:
  - Format the date field in your layout using the following syntax: `<? format_date:fieldname;MASK)?>`  
See *Formatting Dates in Report Designer's Guide for Oracle Business Intelligence Publisher*.
  - To display month name based on month number, use the following syntax in your layout:  
`<?xdoxslt:month_name(month, [abbreviate?], $_XDOLOCALE)?>`  
where `month` is the numeric value of the month (January = 1) and `[abbreviate?]` is the value 0 for do not abbreviate or 1 for abbreviate.  
For example:



```
<?xdoxslt:month_name(1, 0, $_XDOLocale)?>
```

returns January

- To add an expression in the data model, use the following expression:

```
Format_date(date, format_String)
```

For example:

```
SUBSTRING(FORMAT_DATE(G_1.SYSDATE, MEDIUM), 0, 3)
```

returns Nov (when the current SYSDATE is November)

## Creating a Data Set Using a MDX Query Against an OLAP Data Source

BI Publisher supports Multidimensional Expressions (MDX) queries against OLAP data sources.

MDX enables you to query multidimensional objects, such as Essbase cubes, and return multidimensional cell sets that contain the cube's data. You create MDX queries by manually entering the MDX query or by using MDX Query Builder to build the query.

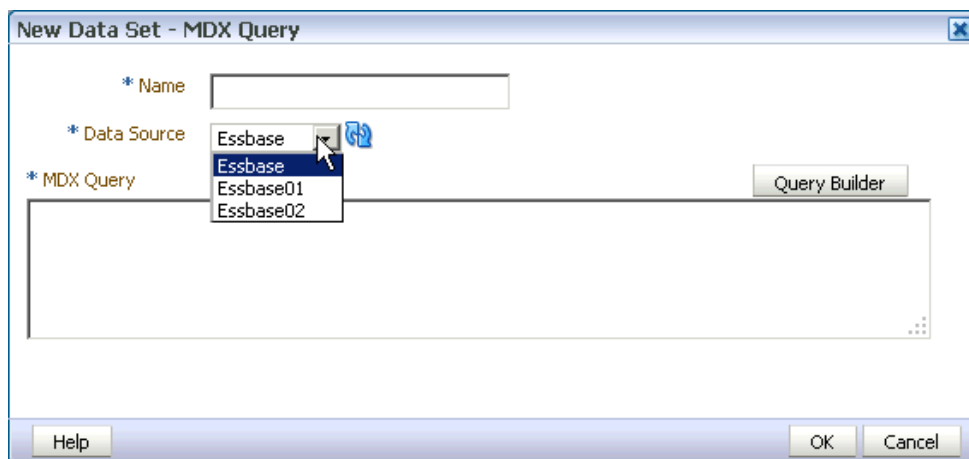
- [Creating a Data Set Using a MDX Query](#)
- [Using MDX Query Builder](#)

### Creating a Data Set Using a MDX Query

You create MDX queries either by manually entering the MDX query or by using MDX Query Builder to build the query.

1. On the toolbar, click **New Data Set** and then select **MDX Query**.

The New Data Set - MDX Query dialog is shown below.



2. Enter a name for the data set.
3. Select the data source for the data set. Only the data sources defined as OLAP connections are displayed in the list.

Any private OLAP data source connections that you created will also be available in the Data Source drop-down list. For more information on creating private data source connections, see [Managing Private Data Sources](#).

4. Enter the MDX query or click **Query Builder**. See [Using MDX Query Builder](#).
5. Click **OK** to save. The data model editor validates the query.

 **Note:**

Ensure that in your OLAP data source that you do not use Unicode characters from the range U+F900 to U+FFFE to define any metadata attributes such as column names or table names. This Unicode range includes half-width Japanese Katakana and full-width ASCII variants. Using these characters results in errors when generating the XML data for a BI Publisher report.

For more information on writing MDX queries, see *Writing MDX Queries* in the *Oracle Essbase Database Administrators Guide* which can be found here: [Oracle Essbase for BI documentation](#).

## Using MDX Query Builder

Use MDX Query Builder to build MDX basic queries without having to code them. MDX Query Builder enables you to add dimensions to columns, rows, pages, and point of view axes and preview the query results.

- [Understanding the MDX Query Builder Process](#)
- [Using the Select Cube Dialog](#)
- [Selecting Dimensions and Measures](#)
- [Performing MDX Query Actions](#)
- [Applying MDX Query Filters](#)
- [Selecting MDX Query Options and Saving MDX Queries](#)

 **Note:**

MDX Query Builder only enables you to build data sets against Essbase data sources. For all other OLAP data sources, you must manually create the query.

## Understanding the MDX Query Builder Process

You create MDX queries either by manually entering the MDX query or by using MDX Query Builder to build the query.

To use MDX Query Builder to build a MDX query:

1. On the toolbar, click **New Data Set** and then select **MDX Query** to launch the New Data Set - MDX Query dialog.

2. Enter a name for the data set.
3. Select a data source.
4. Launch MDX Query Builder.
5. Select an Essbase cube for the query.
6. Select dimensions and measures by dragging and dropping them to the Columns, Rows, Slicer/POV, and the Pages axes.
7. (Optional) Use actions to modify the query.
8. (Optional) Apply filters.
9. Set the query options and save the query.

## Using the Select Cube Dialog

In the Select Cube dialog, select the Essbase cube that you want to use to build the MDX query.

The MDX data source connection that you selected previously drives which Essbase cubes are available for selection.

## Selecting Dimensions and Measures

You build MDX queries by selecting dimensions for the Columns, Rows, Slicer/POV, and Pages axes.

Account dimension members are listed individually by member name. All other dimension members are represented by generation name as shown below.

You can drag dimension generations and individual measures from the Account dimension to the Columns, Rows, Slicer/POV, and Pages axes.

The screenshot shows the MDX Query Builder interface. On the left is a tree view of dimensions including Market, Product, Scenario, Year, and Accounts. The main area shows the query configuration:

- Columns:** Gen2,Market
- Rows:** Gen3,Product, Margin, Sales, Total\_Expenses
- Slicer/POV:** Scenario (Budget)
- Pages:** Gen3,Year

The MDX query text is:

```
SELECT
NON EMPTY Hierarchize([Market].Generations(2).Members) ON Axis(0),
NON EMPTY CROSSJOIN(Hierarchize([Product].Generations(3).Members),{[Accounts].[Margin],[Accounts].[Sales],[Accounts].[Total_Expenses]}) ON Axis(1),NON EMPTY [Year].Generations(3).Members ON Axis(2)
FROM Demo.Basic
WHERE ([Budget])
```

Below the query is a data table for the month of January:

		East	West	South
Stereo	Margin	1,239.00	1,680.00	
	Sales	2,950.00	4,000.00	
	Total_Expenses	1,145.00	1,580.00	
Compact_Disc	Margin	2,139.00	2,914.00	
	Sales	3,450.00	4,700.00	
	Total_Expenses	1,040.00	1,250.00	
Television	Margin	2,880.00	3,060.00	2,040.00
	Sales	4,800.00	5,100.00	3,400.00
	Total_Expenses	1,640.00	1,900.00	1,220.00
VCR	Margin	2,646.00	2,851.00	2,142.00
	Sales	4,200.00	4,650.00	3,400.00
	Total_Expenses	1,180.00	1,415.00	895.00
Camera	Margin	2,023.00	2,874.50	1,707.00
	Sales	2,850.00	4,050.00	2,400.00
	Total_Expenses	1,495.00	1,856.00	1,301.00

Build the query by dragging dimension members or measures from the Dimensions panel to one of the following axes areas:

- **Columns**— Axis (0) of the query
- **Rows** — Axis (1) of the query
- **Slicer/POV**— The slicer axis enables you to limit a query to only a specific slice of the Essbase cube. This represents the optional WHERE clause of a query.
- **Axis** — Axis (2) of the query

You can nest dimension members in the Columns and Rows axes, but you can only add a single dimension to the Slicer/POV axis.

## Adding Dimension Members to the Slicer/POV Axis

When you add a dimension to the Slicer/POV axis, the Member Selection dialog launches.

You can only select one dimension member for this axis. Simply select the dimension in the Member Selection dialog, and then click **OK**.

The Member Selection dialog does not display if you add a measure to the Slicer/POV axis.

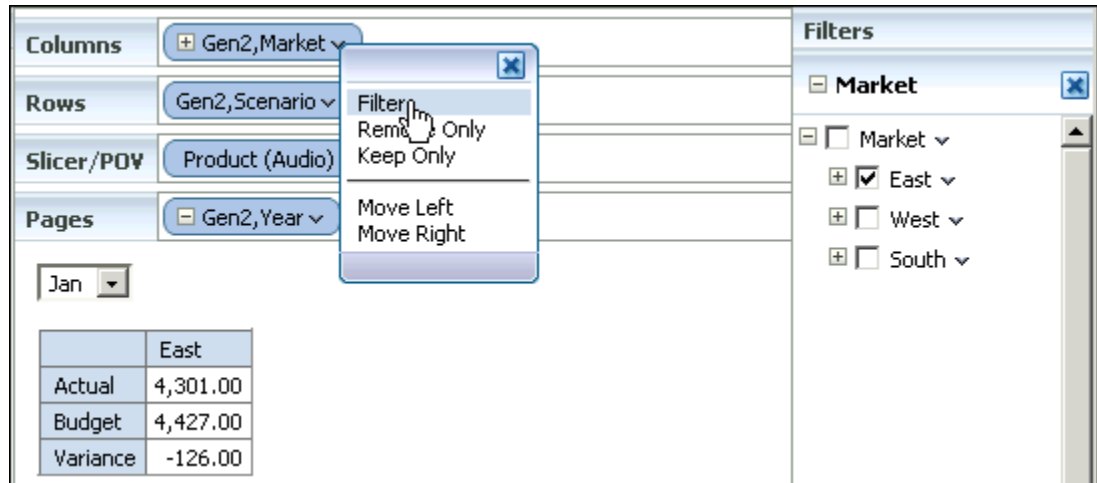
## Performing MDX Query Actions

The MDX Query Builder toolbar contains the following buttons for modifying the MDX query:

- Click **Swap Rows and Columns** to flip dimensions between columns and rows.
- Click **Actions** to display the following menu items for selection:
  - **Select Cube** - Selects a different Essbase cube for the query.
  - **Set Alias Table** - Selects the alias table used for dimension display names. Alias names are used for display only and are not used in the query.
  - **Auto Refresh** - Displays the results as dimension members are placed in the Columns, Rows, Slicer/POV, and Pages axes and automatically refreshes the MDX query syntax.
  - **Clear Results** - Clears the results and removes member selections from all of the axes and any filters added to the query.
  - **Show Empty Columns** - Displays columns that do not contain data.
  - **Show Empty Rows** - Displays rows that do not contain data.
  - **Show Query** - Displays the MDX query syntax resulting from how the dimensions are placed in the Columns, Rows, Slicer/POV, and Pages axes.

## Applying MDX Query Filters

You can create filters for dimensions on the Columns, Rows, and Pages axes in MDX Query Builder to further streamline your MDX query.



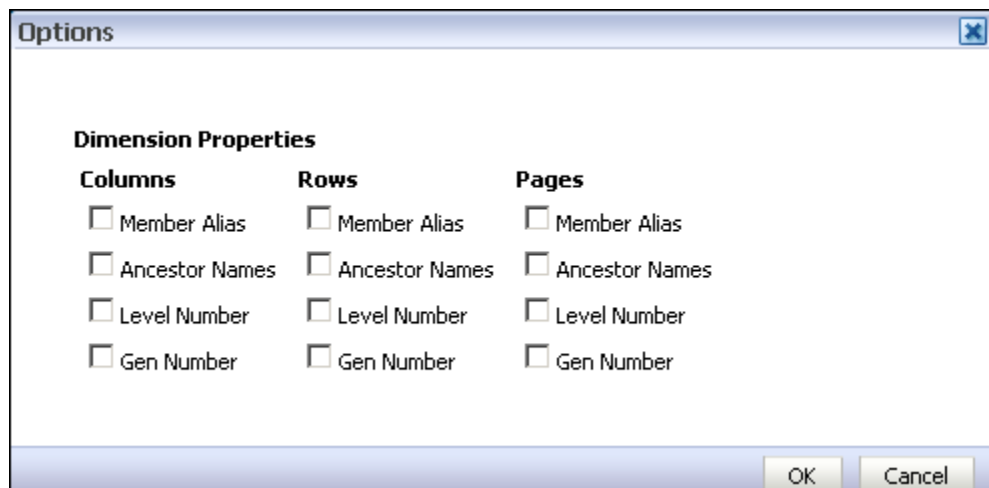
You can create multiple filters for a query, but you can only create one filter for each Columns, Rows, or Pages axis.

- To create a filter, click the down-arrow button to the right of a dimension in the Columns, Rows, or Pages axes to display it in the Filters area. You create the filter by selecting the desired dimension member as shown below.

## Selecting MDX Query Options and Saving MDX Queries

Use the Options dialog to select the dimension properties to include in the query for each of the dimensions in the Columns, Rows, and Pages axes.

Once you have built the query, click **Save** to display the Options dialog as shown below.



By default, none of the properties are selected.

The dimension properties are as follows:

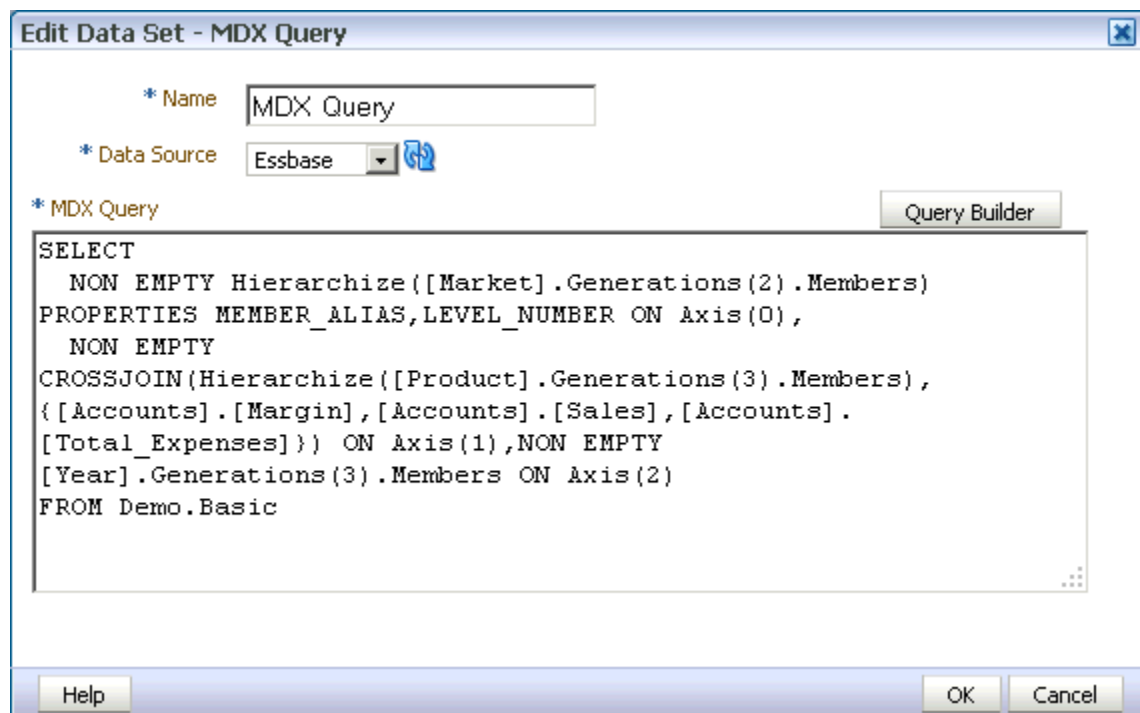
- **Member Alias** — Dimension member alias names as listed in the Essbase outline.
- **Ancestor Names** — Ancestor dimension names as listed in the Essbase outline.
- **Level Number** — Dimension level numbers as listed in the Essbase outline.
- **Gen Number** — Generation number of the dimensions as listed in the Essbase outline.

For example, if you select the Member Alias and Level Number properties for Columns, the MDX query results are as follows:

```
SELECT
NON EMPTY Hierarchize([Market].Generations(2).Members)
PROPERTIES MEMBER_ALIAS,LEVEL_NUMBER ON Axis(0),
NON EMPTY CROSSJOIN(Hierarchize([Product].Generations(3).Members),
{[Accounts].[Margin],[Accounts].[Sales],[Accounts].[Total_Expenses]})ON Axis(1),
NON EMPTY [Year].Generations(3).Members ON Axis(2) FROM Demo.Basic
```

For more information on Essbase dimension properties, see: *Oracle Essbase Database Administrator's Guide* which can be found here: [Oracle Essbase for BI documentation](#).

After you select options for the MDX query, click **OK** to return to the New Data Set - MDX Query dialog and review the MDX query output as shown below.



Click **OK** to return to the data model editor, and save your changes.

 **Note:**

If you modify a MDX query after you save it in BI Publisher, Oracle recommends that you manually change the syntax and not use MDX Query Builder to do so.

## Creating a Data Set Using an Oracle BI Analysis

If you have enabled integration with Oracle Business Intelligence, then you can access the Oracle Business Intelligence Presentation catalog to select an Oracle BI analysis as a data source.

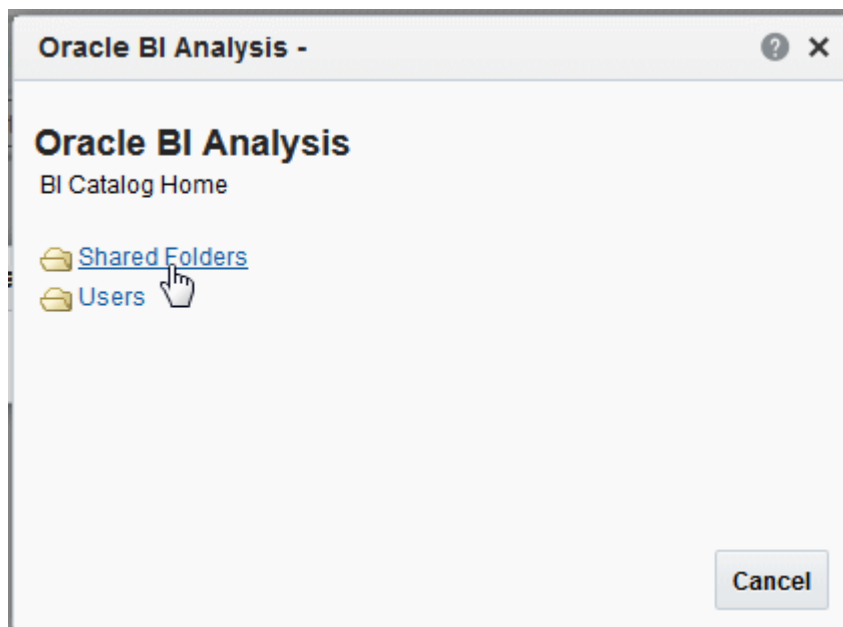
An analysis is a query against an organization's data that provides answers to business questions. A query contains the underlying SQL statements that are issued to the Oracle BI Server.

See *User's Guide for Oracle Business Intelligence Publisher*.

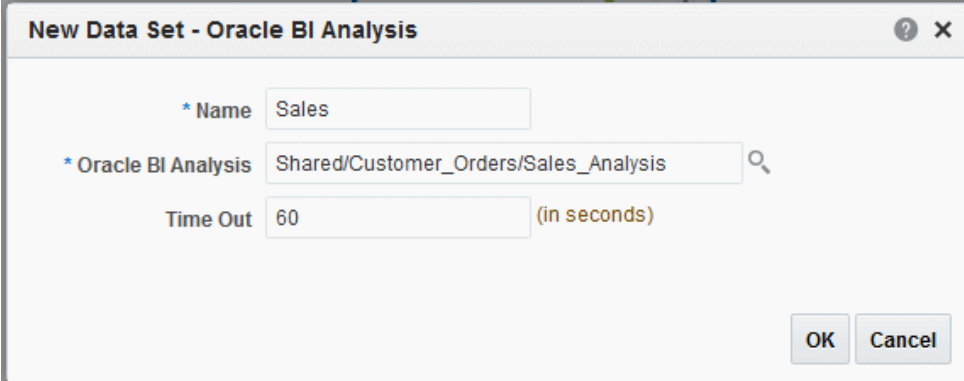
 **Note:**

Hierarchical columns are not supported in BI Publisher data models.

1. Click the **New Data Set** toolbar button and select **Oracle BI Analysis**.
2. In the New Data Set - Oracle BI Analysis dialog, enter a name for this data set.
3. Click the browse icon to connect to the Oracle Business Intelligence Presentation catalog, as shown below.



4. When the catalog connection dialog launches, navigate through the folders to select the Oracle BI analysis to use as the data set for the report.
5. Enter a **Time Out** value in seconds, as shown below. If BI Publisher has not received the analysis data after the time specified in the time out value has elapsed, then BI Publisher stops attempting to retrieve the analysis data.



The screenshot shows a dialog box titled "New Data Set - Oracle BI Analysis". It contains three input fields: "\* Name" with the value "Sales", "\* Oracle BI Analysis" with the value "Shared/Customr\_Orders/Sales\_Analysis", and "Time Out" with the value "60" and the text "(in seconds)" to its right. There are "OK" and "Cancel" buttons at the bottom right.

6. Click **OK**.

## Additional Notes on Oracle BI Analysis Data Sets

Parameters and list of values are inherited from the BI analysis and they display at run time.

The BI Analysis must have default values defined for filter variables. If the analysis contains presentation variables with no default values, it is not supported as a data source by BI Publisher.

If you want to structure the data based on Oracle BI Analysis Data Sets, the group breaks, group filters, data links and group-level functions are not supported.

The following are supported:

- Global level functions
- Setting the value for elements if null

For more information about the above supported features, see [Structuring Data](#).

## Creating a Data Set Using a View Object

BI Publisher enables you to connect to your custom applications built with Oracle Application Development Framework and use view objects in your applications as data sources for reports.

Performance of the query execution is better as the SQL is executed directly against the database.

Before you can create an Oracle BI Publisher data model using a view object, you must first create the view object in your application following the guidelines in *Developer's Guide for Oracle Business Intelligence Publisher*.

1. Click the **New Data Set** toolbar button and select **View Object**.



2. In the New Data Set - View Object dialog, enter a name for this data set.
3. Select the Data Source from the list. The data sources that you defined in the `providers.xml` file display.
4. Do one of the following:
  - Select **Yes** for **Execute as SQL** to extract the SQL query from the View Object and execute it on the Oracle Business Intelligence domain.
  - Select **No** to execute the view object on the Oracle Applications domain through the ADF layer. The XML data is then streamed to the Oracle Business Intelligence domain in chunks. This method results in poorer performance, but enables execution on the Applications domain. You can invoke some service interface layers to allow custom data manipulation.
5. Enter the fully qualified name of the application module, for example, `example.apps.pa.entity.applicationModule.AppModuleAM`.
6. Click **Load View Objects**.  
BI Publisher calls the application module to load the view object list.
7. Select the **View Object**.  
Any bind variables defined are retrieved.
8. Create a parameter to map to this bind variable, see [Adding Parameters and Lists of Values](#).
9. Click **OK** to save your data set.

## Additional Notes on View Object Data Sets

To structure data based on view object data sets, the group breaks, data links and group-level functions are not supported.

The following is supported: Setting the value for elements if null.

For more information about this supported feature, see [Structuring Data](#).

## Creating a Data Set Using a Web Service

BI Publisher supports data sets that use simple and complex Web service data sources to return valid XML data.

To include parameters for Web service methods, it is recommended that you define the parameters first, so that the methods are available for selection when setting up the data source. See [Adding Parameters and Lists of Values](#).

Multiple parameters are supported. Ensure the method name is correct and the order of the parameters matches the order in the method. To call a method in the Web service that accepts two parameters, you must map two parameters defined in the report to the two parameters in the method. Note that only parameters of simple type are supported, for example, string and integer.

 **Note:**

Only document/literal Web services are supported.

To specify a parameter, click **Add Parameter** and select the parameter from the drop-down list.

 **Note:**

The parameters must be set up in the Parameters section of the report definition. For more information, see [Adding Parameters and Lists of Values](#).

## Web Service Data Source Options

Web service data sources can be set up on the Administration page or as a data source.

- On the Administration page  
Connections to Web service data sources can be set up on the Administration page and then used in multiple data models. See *Setting Up a Connection to a Web Service* in *Administrator's Guide for Oracle Business Intelligence Publisher*.
- As a private data source  
You can also set up a private connection accessible only to you. See [Managing Private Data Sources](#) for information about private data source connections.

You must set up the connection before you create the data model.

BI Publisher supports data sets that use simple and complex Web service data sources to return valid XML data:

- [Creating a Data Set Using a Simple Web Service](#)
- [Creating a Data Set Using a Complex Web Service](#)

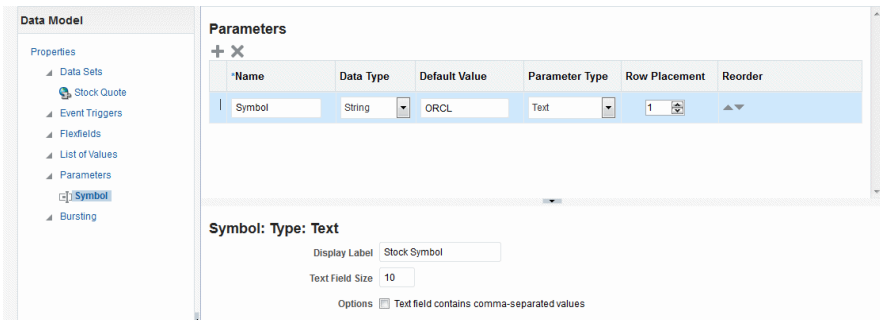
## Creating a Data Set Using a Simple Web Service

If you are not familiar with the available methods and parameters in the Web service to call, you can open the URL in a browser to view them.

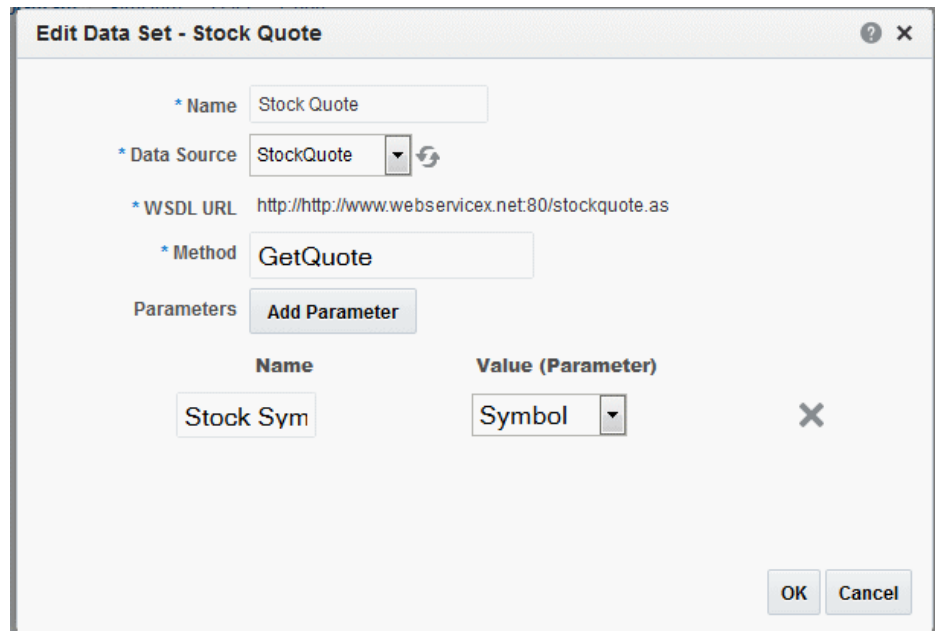
1. Click the **New Data Set** toolbar button, and then select **Web Service**.

The New Data Set - Web Service dialog is shown below.

2. Enter the data set name.
3. Select the Data Source.  
The WSDL URL, Web Service, and Method fields are automatically populated from the Web service data source.
4. Enter the **Method**.
5. Click **OK**.
6. Define the parameters to make them available to the Web service data set as follows:
  - Select **Parameters** on the Data Model pane, click **Create New Parameter**, and then enter the following attributes:
    - **Name** - Enter an internal identifier for the parameter, for example, *Symbol*.
    - **Data Type** - Select String.
    - **Default Value** - Enter a default for the parameter, for example, *ORCL*.
    - **Parameter Type** - Select Text.
    - **Row Placement** - Select a row placement value. The default is 1.
  - In the **New\_Parameter\_1:Type: Text** region, enter the following attributes:
    - **Display Label** - Enter the label you want displayed for your parameter, for example, *Stock Symbol*.
    - **Text Field Size** - Enter the size for the text entry field in characters.



- (Optional) Select the following parameter options:
    - **Text field contains comma-separated values** - Enables the user to enter multiple comma-delimited values for this parameter.
    - **Refresh other parameters on change** - Performs a partial page refresh to refresh any other parameters whose values are dependent on the value of this one.
7. Return to the Web service data set and add the parameter as follows:
- Click the data set name.
  - On the toolbar, click **Edit Selected Data Set** to launch the Edit Data Set dialog.
  - In the **Edit Data Set** dialog, click **Add Parameter**.
  - Name the parameter, and then click **OK** to close the Edit Data Set dialog.



8. Click **Save**.

## Creating a Data Set Using a Complex Web Service

BI Publisher supports data sets that use complex Web service data sources to return valid XML data. A complex Web service type internally uses soapRequest / soapEnvelope to pass the parameter values to the destination host.

When a data set uses a complex Web service as a data source, the data model editor displays the WSDL URL, available Web service, and operations associated with the complex Web service. For each selected operation, the data model editor displays the structure of the required input parameters. If you choose **Show optional parameters**, all optional parameters as displayed as well.

If you are not familiar with the available methods and parameters in the Web service, open the WSDL URL in a browser to view them.

1. Enter the data set information as follows:

- Enter the data set name.
- Select the data source. The WSDL URL and Web service fields are automatically populated from the complex Web service data source.
- Select the **Method**.

The Methods available for selection are based on the complex Web service data source. When you select a method, the **Parameters** are displayed. To view optional parameters, select **Show optional parameters**.

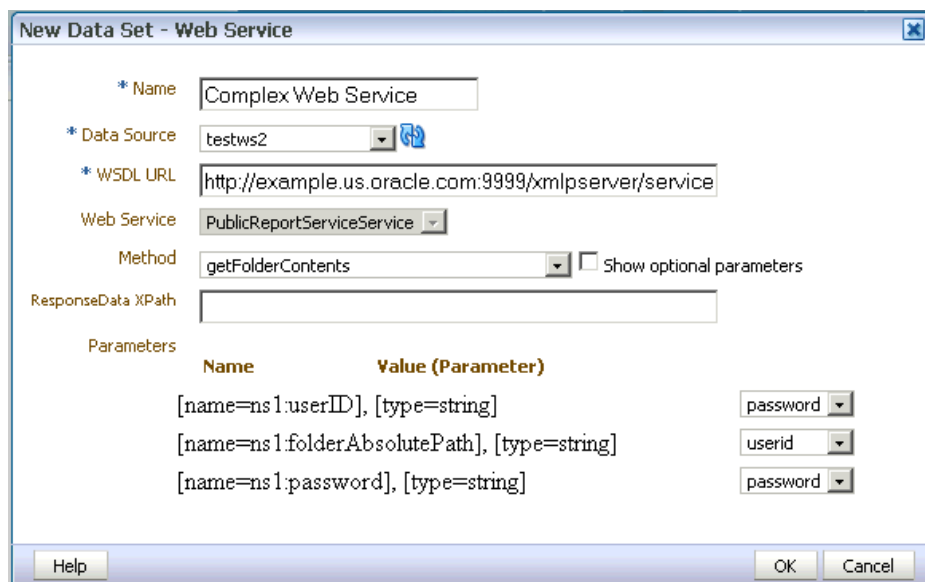
- **Response Data XPath** - If the start of the XML data for the report is deeply embedded in the response XML generated by the Web service request, use this field to specify the path to the data to use in the Oracle BI Publisher report.

2. Define the parameter to make it available to the Web service data set as follows:

- Select **Parameters** on the **Report** definition pane, and then click **New** to create a parameter.
- Define the following parameter attributes:
  - **Name** - Enter an internal identifier for the parameter.
  - **Data Type** - Select the appropriate data type for the parameter.
  - **Default Value** - Enter a default value for the parameter.
  - **Parameter Type** - Select the appropriate parameter type.
  - **Display label** - Enter the label you want displayed for your parameter.
  - **Text Field Size** - Enter the size for the text entry field in characters.

3. Return to the Web Service data set and add the parameter.

- Select the Web service data set, and then click **Edit Selected Data Set** to launch the Edit Data Set dialog.
- Select the parameters as shown below.



- To test the Web service, see [Testing Data Models and Generating Sample Data](#).

## Additional Information on Web Service Data Sets

There is no metadata available from Web service data sets, therefore grouping and linking are not supported.

## Creating a Data Set Using an LDAP Query

BI Publisher supports queries against Lightweight Directory Access protocol (LDAP) data sources.

You can query user information stored in LDAP directories and then use the data model editor to link the user information with data retrieved from other data sources.

For example, to generate a report that lists employee salary information that is stored in the database application and also include employee e-mail addresses that are stored in the LDAP directory in the report, you can create a query against each and then link the two in the data model editor to display the information in a single report. The figure below shows a sample LDAP query.

The screenshot shows a dialog box titled "New Data Set - LDAP Query". It has a standard Windows-style title bar with a close button. The dialog contains the following fields and controls:

- Name:** A text box containing "LDAP Data Set".
- Data Source:** A dropdown menu showing "ldap11" and a refresh icon.
- Search Base:** An empty text box.
- Attributes:** A text box containing "mail, cn, givenName".
- Filter:** A text box containing "(objectclass=person)".
- Buttons:** "Help", "OK", and "Cancel" buttons at the bottom.

1. Click the **New Data Set** toolbar button and select **LDAP Query**.
2. In the New Data Set - LDAP Query dialog, enter a name for this data set.
3. Select the **Data Source** for this data set.
4. In the **Search Base** field, enter the starting point for the search in the directory tree.

 **Note:**

Search Base is required when the LDAP provider is Microsoft Active Directory. The Search Base defines the starting point of the search in the directory tree. For example, if you want to query the entire directory, specify the root.

To specify the starting point, enter each hierarchical object separated by a comma, starting with the lowest level in the hierarchy, for example, to search the Sales container in the *mycompany.com* domain, enter:

```
ou=Sales,dc=mycompany,dc=com
```

5. In the **Attributes** entry box, enter the attributes whose values you want to fetch from the LDAP data source.
6. To filter the query, enter the appropriate syntax in the **Filter** entry box. The syntax is as follows:

```
(Operator (Filter)through(Filter))
```

For example:

(objectclass=person)

LDAP search filters are defined in the Internet Engineering Task Force (IETF) Request for Comments document 2254, [The String Representation of LDAP Search Filters](#) (RFC 2254) on the IETF Web site.

7. Link the data from this query to the data from other queries or modify the output structure. For instructions on completing this step, see [Structuring Data](#).

## Creating a Data Set Using a XML File

You can use an XML file to create a data source.

Do one of the following:

- Place the XML file in a directory that your administrator has set up as a data source. See [Setting Up a Connection to a File Data Source](#) in *Administrator's Guide for Oracle Business Intelligence Publisher*.
- Upload the XML file to the data model from a local directory.

### Note:

To use BI Publisher's layout editor and interactive viewer, sample data from the XML file source must be saved to the data model.

## About Supported XML Files

Support of XML files as a data set type in BI Publisher follows certain guidelines.

- The XML files that you use as input to the BI Publisher data engine must be UTF-8 encoded.
- Do not use the following characters in XML tag names: ~, !, #, \$, %, ^, &, \*, +, `, |, :, \", \\, <, >, ?, ,, /. If your data source file contains any of these characters, use the data model editor Structure tab to change the tag names to an acceptable one.
- Use valid XML files. Oracle provides many utilities and methods for validating XML files.
- There is no metadata available from XML file data sets, therefore grouping and linking are not supported.

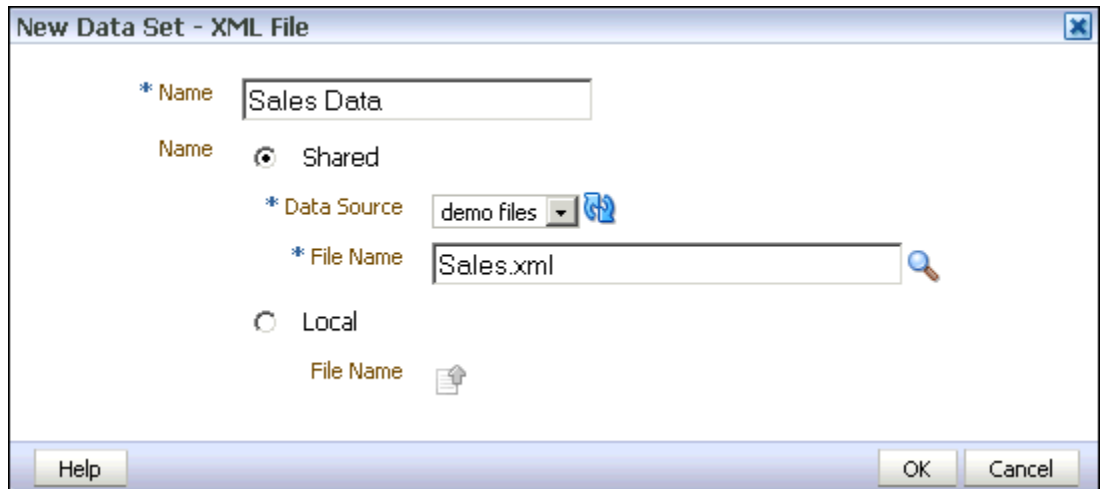
## Using a XML File Stored in a File Directory Data Source

Create data sets using XML files stored in file directories.

To create a data set using a XML file from a file directory data source:

1. On the toolbar, click **New Data Set** and select **XML File**. The New Data Set - XML File dialog launches, as shown below.





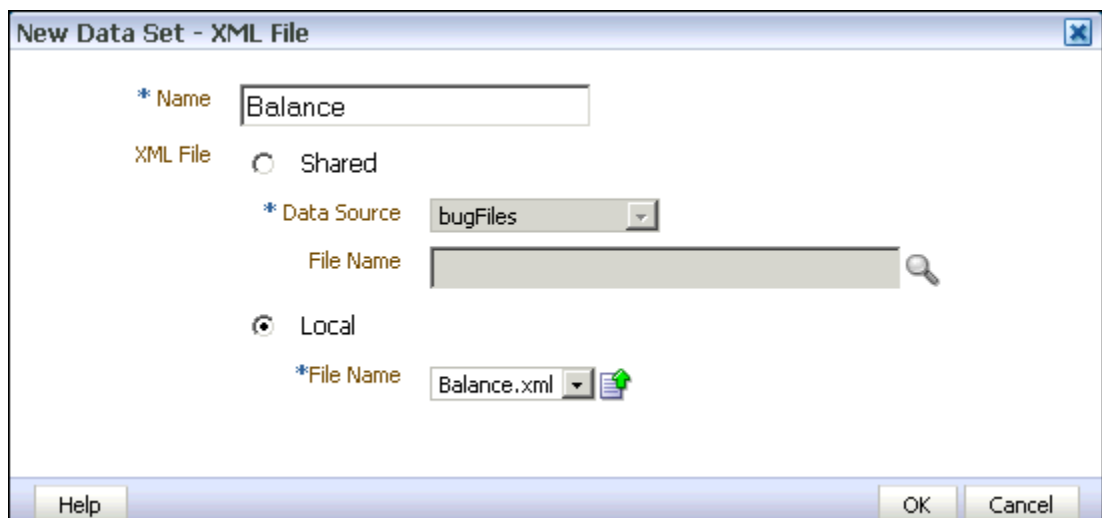
2. Enter a name for the data set.
3. Click **Shared** to enable the **Data Source** list. This is the default selected option.
4. Select the **Data Source** where the XML file resides. The list is populated from the configured File Data Source connections.
5. To the right of **File Name**, click **Browse** to connect to the data source and browse the available directories. Select the file.
6. Click **OK**.
7. (Required) Save sample data to the data model. See [Testing Data Models and Generating Sample Data](#).

## Uploading a XML File Stored Locally

You can create data sets using locally stored XML files.

To create a data set using a XML file stored locally:

1. On the toolbar, click **New Data Set** and select **XML File**. The New Data Set - XML File dialog launches, as shown below.

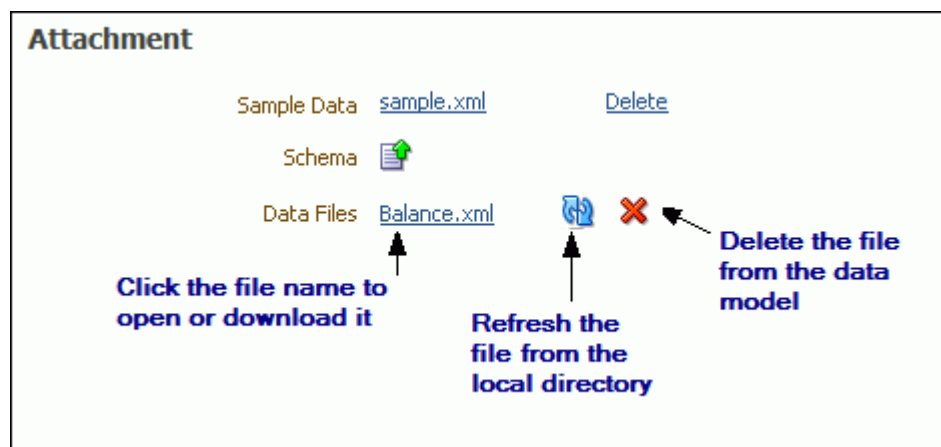


2. Enter a name for this data set.
3. Select **Local** to enable the Upload button.
4. Click **Upload** to browse for and upload the XML file from a local directory. If the file has been uploaded to the data model, then it is available for selection in the File Name List.
5. Click **Upload**.
6. Click **OK**.
7. (Required) Save sample data to the data model. See [Testing Data Models and Generating Sample Data](#).

## Refreshing and Deleting an Uploaded XML File

You can refresh and delete uploaded local XML files.

After uploading the file, it is displayed on the **Properties** pane of the data model under the **Attachments** region, as shown below.



See [Setting Data Model Properties](#) for more information about the Properties pane.

To refresh the local file in the data model:

1. In the component pane, click **Data Model** to view the Properties page.
2. In the **Attachment** region of the page, locate the file in the **Data Files** list.
3. Click **Refresh**.
4. In the Upload dialog, browse for and upload the latest version of the file. The file must have the same name or it will not replace the older version.
5. Save the data model.

To delete the local file:

1. In the component pane, click **Data Model** to view the Properties page.
2. In the **Attachment** region of the page, locate the file in the Data Files list.
3. Click **Delete**.

4. Click **OK** to confirm.
5. Save the data model.

## Creating a Data Set Using a Content Server

You can set up connections to Content Server data source on the Administration page and then use that in multiple data models.

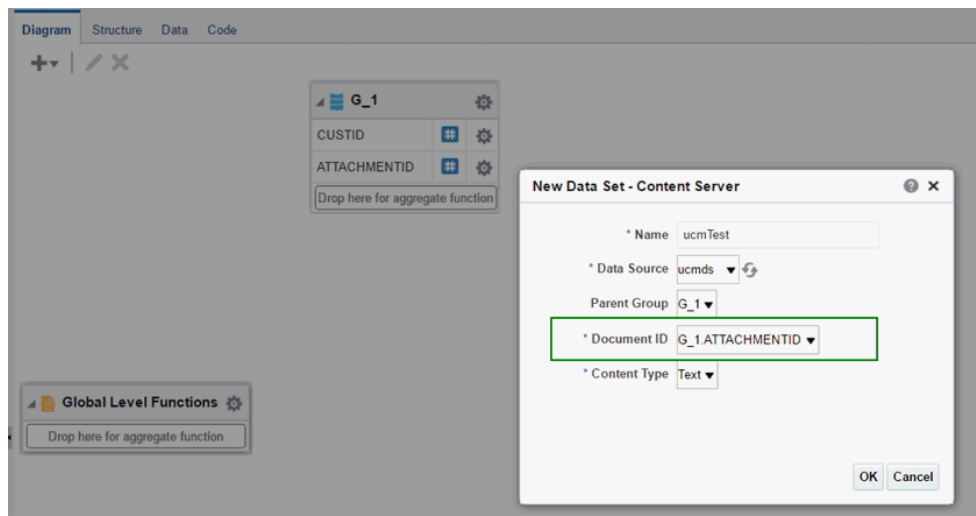
For more information, see *Setting Up a Connection to a Content Server in Administrator's Guide for Oracle Business Intelligence Publisher*.

You must set up the connection before you create a data model. Create a data model by creating the SQL Query data set (required) first and then create the Content Server data set.

1. Click the New Data Set toolbar button and select **Content Server**.

In the New Data Set, Content Server dialog do the following:

2. Enter a name for the data set in the **Name** field.
3. Select the content server data source in the **Data Source** field.
4. Select the **Parent Group** from the LOV.
5. Select the **Document ID** from the LOV.
6. Select the **Content Type** from the LOV.



7. Click **OK**.

## Creating a Data Set Using a Microsoft Excel File

These topics describe requirements, options, and procedures for using Microsoft Excel files as a data source.

- [Options for Uploading Excel Files to BI Publisher](#)
- [About Supported Excel Files](#)
- [Accessing Multiple Tables per Sheet](#)

- [Using a Microsoft Excel File Stored in a File Directory Data Source](#)
- [Uploading a Microsoft Excel File Stored Locally](#)

## Options for Uploading Excel Files to BI Publisher

To use a Microsoft Excel file as a data source, you have the two options for providing the file to BI Publisher.

- Place the file in a directory that your administrator has set up as a data source. See *Setting Up a Connection to a File Data Source* in *Administrator's Guide for Oracle Business Intelligence Publisher*.
- Upload the file to the data model from a local directory.

## About Supported Excel Files

Support of Microsoft Excel files as a data set type in Oracle BI Publisher follows certain guidelines.

- Save Microsoft Excel files in the Excel 97-2003 Workbook (\*.xls) format by Microsoft Excel. Files created by a third party application or library are not supported.
- The source Excel file can contain a single sheet or multiple sheets.
- Each worksheet can contain one or multiple tables. A table is a block of data that is located in the continuous rows and columns of a sheet.

In each table, Oracle BI Publisher always considers the first row to be the heading row for the table.

- The first row under the heading row must not be empty and is used to determine the column type of the table. The data type of the data in the table may be number, text, or date/time.
- If multiple tables exist in a single worksheet, the tables must be identified with a name for BI Publisher to recognize each one. See [Accessing Multiple Tables per Sheet](#).
- If all tables in the Excel file are not named, only the data in the first table is recognized and fetched.
- When the data set is created, BI Publisher truncates all trailing zeros after the decimal point for numbers in all cases. To preserve the trailing zeros in your final report, you must apply a format mask in your template to display the zeroes. See *Formatting Numbers, Dates, and Currencies* in *Report Designer's Guide for Oracle Business Intelligence Publisher*.
- Single value parameters are supported, but multiple value parameters are not supported.

## Accessing Multiple Tables per Sheet

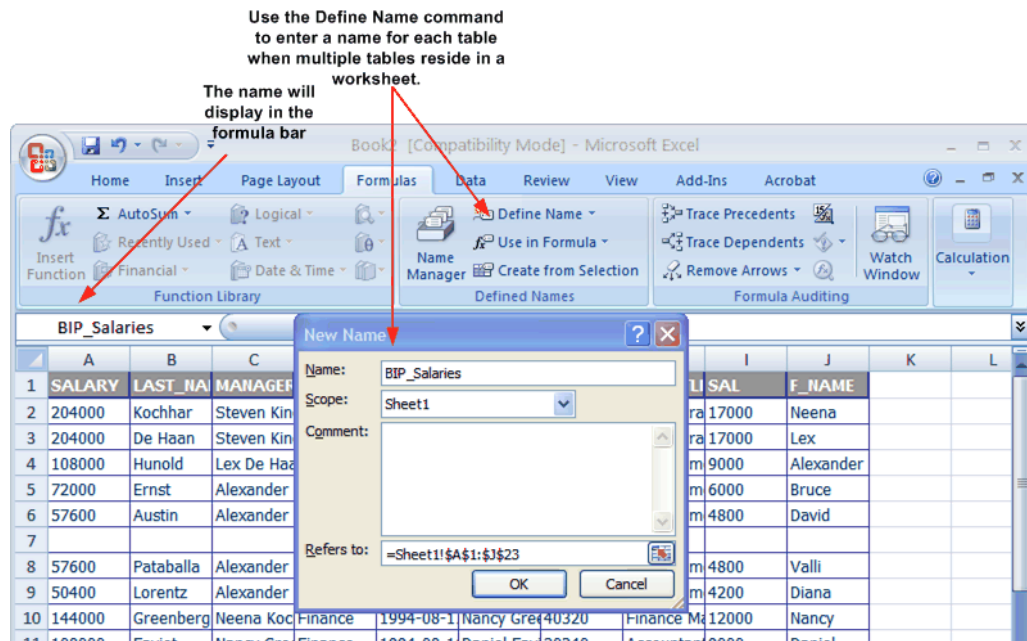
If the Excel worksheet contains multiple tables that you want to include as data sources, then you must define a name for each table in Excel.

 **Note:**

The name that you define must begin with the prefix: *BIP\_*, for example, *BIP\_SALARIES*.

1. Insert the table in Excel.
2. Do one of the following:
  - Using Excel 2003, select the table. On the **Insert** menu, click **Name** and then **Define**. Enter a name that is prefixed with *BIP\_*.
  - Using Excel 2007, select the table. On the Formulas tab, in the **Defined Names** group, click **Define Name**, then enter the name in the **Name** field. The name you enter appears on the Formula bar

For example, you could use the `Define Name` command in Microsoft Excel 2007 to name a table *BIP\_Salaries*.



## Using a Microsoft Excel File Stored in a File Directory Data Source

Create data sets using Microsoft Excel files stored in file directories.

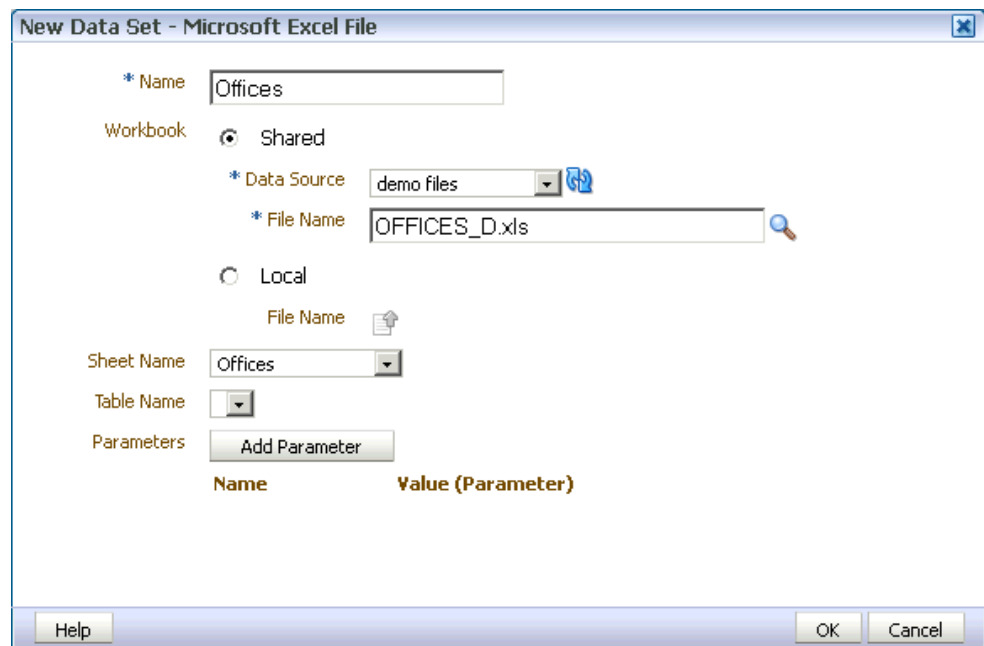
Note that to include parameters for your data set, you must define the parameters first, so that they are available for selection when defining the data set. See [Adding Parameters and Lists of Values](#).

 **Note:**

The Excel data set type supports one value per parameter. It does not support multiple selection for parameters.

To create a data set using a Microsoft Excel file from a file directory data source:

1. Click the **New Data Set** toolbar button and select **Microsoft Excel File**. The New Data Set - Microsoft Excel File dialog launches.
2. Enter a name for this data set.
3. Click **Shared** to enable the Data Source list.
4. Select the data source where the Microsoft Excel File resides.
5. To the right of the **File Name** field, click the browse icon to browse for the Microsoft Excel file in the data source directories. Select the file.
6. If the Excel file contains multiple sheets or tables, select the appropriate **Sheet Name** and **Table Name** for this data set, as shown below.



7. If you added parameters for this data set, click **Add Parameter**. Enter the **Name** and select the **Value**. The Value list is populated by the parameter Name defined

in the Parameters section. Only single value parameters are supported. See [Adding Parameters and Lists of Values](#).

8. Click **OK**.
9. Link the data from this query to the data from other queries or modify the output structure. For more information on linking queries, see [Structuring Data](#).

## Uploading a Microsoft Excel File Stored Locally

To use a local Microsoft Excel file as a data source, you must first upload it.

Note that to include parameters for the data set, you must define the parameters first, so that they are available for selection when defining the data set. See [Adding Parameters and Lists of Values](#).

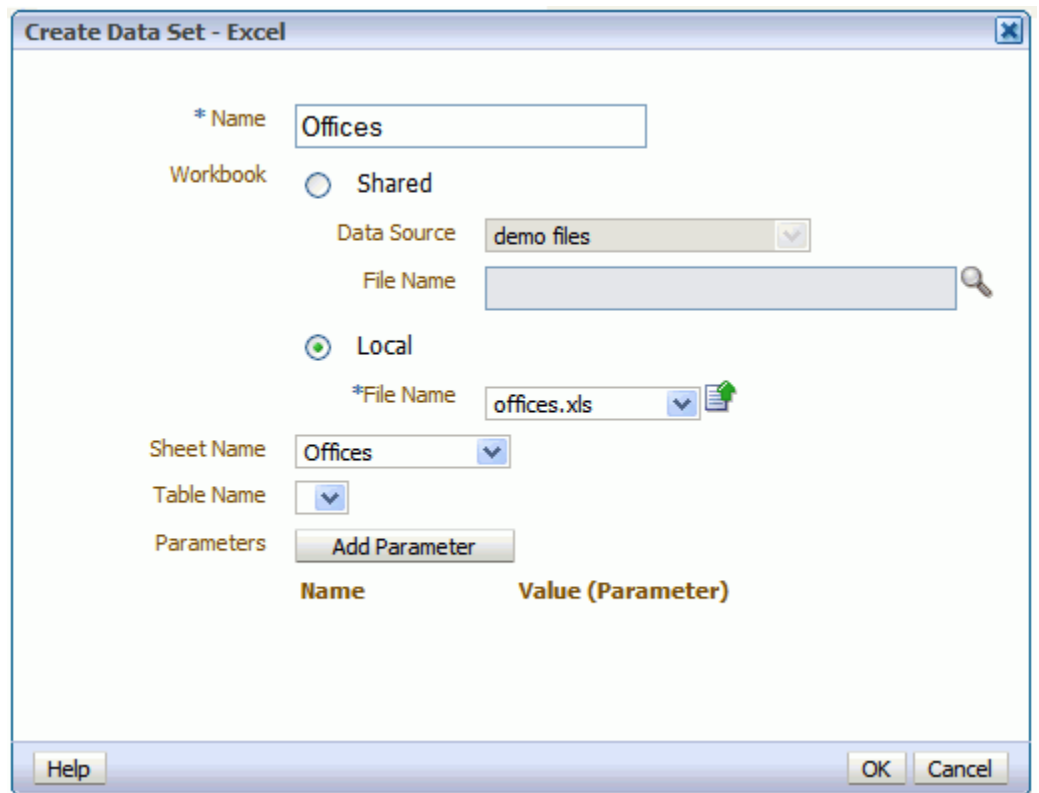


### Note:

The Excel data set type supports one value per parameter. It does not support multiple selection for parameters.

To create a data set using a Microsoft Excel file stored locally:

1. Click the **New Data Set** toolbar button and select Microsoft Excel File. The Create Data Set - Excel dialog launches.
2. Enter a name for this data set.
3. Select **Local** to enable the upload button.
4. Click the **Upload** icon to browse for and upload the Microsoft Excel file from a local directory. If the file has been uploaded to the data model, then it is available for selection in the **File Name** list.
5. If the Excel file contains multiple sheets or tables, select the appropriate **Sheet Name** and **Table Name** for this data set, as shown below.



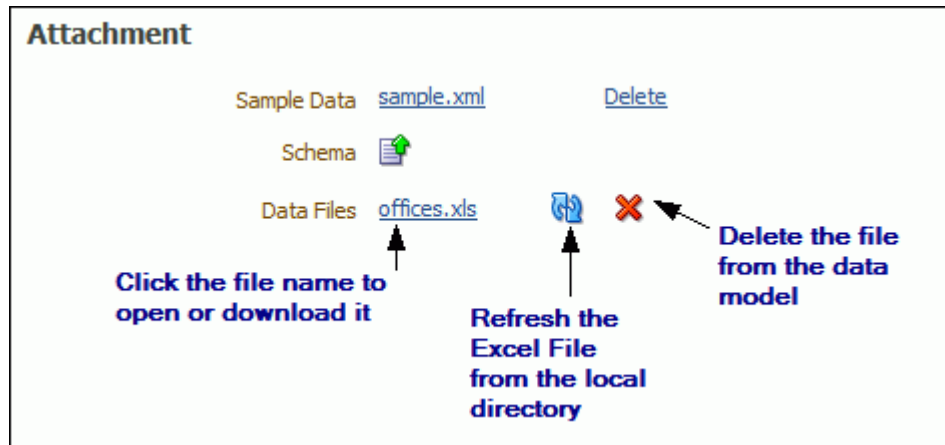
6. If you added parameters for this data set, click **Add Parameter**. Enter the **Name** and select the **Value**. The **Value** list is populated by the parameter **Name** defined in the **Parameters** section. Only single value parameters are supported. See [Adding Parameters and Lists of Values](#).
7. Click **OK**.
8. Link the data from this query to the data from other queries or modify the output structure. For more information on linking queries, see [Structuring Data](#).

## Refreshing and Deleting an Uploaded Excel File

You can refresh and delete uploaded local Excel files.

After uploading the file, it displays on the **Properties** pane of the data model under the **Attachments** region, as shown below.





See [Setting Data Model Properties](#) for information about the **Properties** page.

To refresh the local file in the data model:

1. Click **Data Model** in the component pane to view the **Properties** page.
2. In the **Attachment** region of the page, locate the file in the **Data Files** list.
3. Click **Refresh**.
4. In the **Upload** dialog, browse for and upload the latest version of the file. The file must have the same name or it will not replace the older version.
5. Save the data model.

To delete the local file:

1. Click **Data Model** in the component pane to view the **Properties** page.
2. In the **Attachment** region of the page, locate the file in the **Data Files** list.
3. Click **Delete**.
4. Click **OK** to confirm.
5. Save the data model.

## Creating a Data Set Using a CSV File

BI Publisher supports data sets that use CSV file data sources to return valid XML data.

The following topics describe using requirements and procedures for using a CSV as a data source:

- [About Supported CSV Files](#)
- [Creating a Data Set from a Centrally Stored CSV File](#)
- [Uploading a CSV File Stored Locally](#)

## About Supported CSV Files

Support of CSV files as a data set type in BI Publisher follow certain guidelines.

- You can use a CSV file that is located in a directory that your administrator has set up as a data source. See *Setting Up a Connection to a File Data Source in Administrator's Guide for Oracle Business Intelligence Publisher*.

You can upload a file from a local directory.

- The supported CSV file delimiters are Comma, Pipe, Semicolon, and Tab.
- If your CSV file contains headers, the header names are used as the XML tag names. The following characters are not supported in XML tag names: ~, !, #, \$, %, ^, &, \*, +, `, |, :, \", \\, <, >, ?, ,, /. If your data source file contains any of these characters in a header name, use the data model editor Structure tab to edit the tag names.
- CSV data sets support editing the data type assigned by the data model editor. See [Editing the Data Type](#). If you update the data type for an element in the data set, you must ensure that the data in the file is compliant with the data type that you selected.
- The CSV files that you use as input to the Oracle BI Publisher data engine must be UTF-8 encoded and cannot contain empty column headers.
- Group breaks, data links, expression and group-level functions are not supported.
- Data fields in CSV files must use the canonical ISO date format for mapped date elements, for example, 2012-01-01T10:30:00-07:00, and #####.## for mapped number elements.
- Data validation is not provided for CSV files.

## Creating a Data Set from a Centrally Stored CSV File

You can use a CSV file from a file directory to create a data set.

1. On the data model editor toolbar, click **New Data Set** and select **CSV File**. The New Data Set - CSV File dialog launches.

The screenshot shows the 'New Data Set - CSV File' dialog box. The dialog is titled 'New Data Set - CSV File' and has a close button in the top right corner. The main content area contains the following elements:

- A text box labeled '\* Name' containing the text 'Employees'.
- A radio button labeled 'Shared' which is selected, and a radio button labeled 'Local' which is unselected.
- A dropdown menu labeled '\* Data Source' with 'demo files' selected.
- A text box labeled '\* File Name' containing 'Emps\_WithHdrRow.csv' and a search icon to its right.
- A checkbox labeled 'The first row is a column header' which is unchecked.
- A dropdown menu labeled '\* CSV delimiter' with 'Comma(,)' selected.
- Buttons for 'Help', 'OK', and 'Cancel' at the bottom of the dialog.

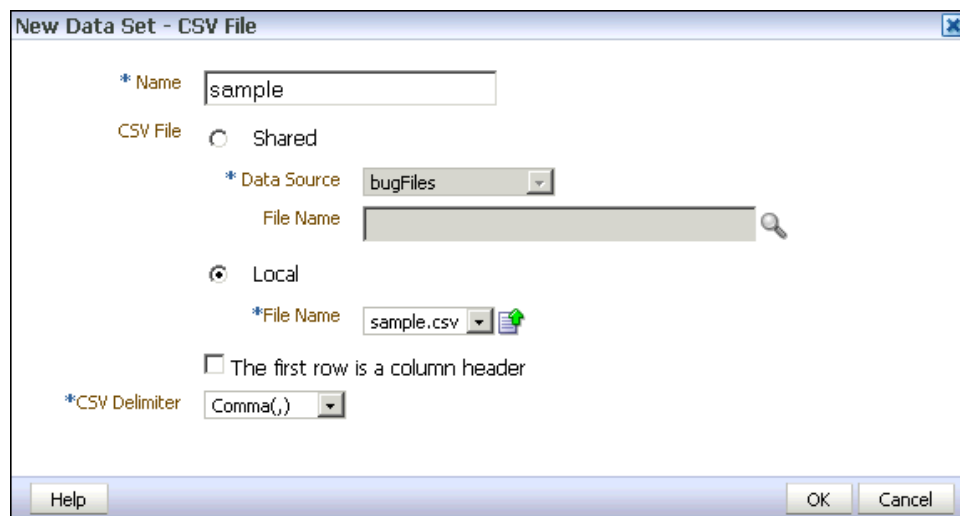
2. Enter a name for this data set.
3. Click **Shared** to enable the Data Source list.
4. Select the **Data Source** where the CSV file resides.  
The list is populated from the configured File Data Source connections.
5. Click **Browse** to connect to the data source, browse the available directories, and select the file.
6. Select **The first row a column header** to specify if the first row in the file contains column names.  
If you do not select this option, the columns are assigned a generic name, for example, *Column1*, *Column2*. You can edit the XML tag names and display names in the data model editor Structure tab.
7. Select the **CSV delimiter** used in the file.  
The default selection is Comma (.).
8. Click **OK**.

## Uploading a CSV File Stored Locally

Create data sets using CSV files stored in local file directories.

To create a data set using a CSV file stored locally:

1. On the toolbar, click **New Data Set** and select **CSV File**. The New Data Set - CSV File dialog launches, as shown below.



2. Enter a name for this data set.
3. Select **Local** to enable the Upload button.
4. Click **Upload** to browse for and upload the CSV file from a local directory. If the file has been uploaded to the data model, then it is available for selection in the File Name List.
5. (Optional) Select **The first row a column header** to specify if the first row in the file contains column names. If you do not select this option, the columns are

assigned a generic name, for example, Column1, Column2. The XML tag names and display names assigned can be edited in the data model editor Structure tab.

6. Select the **CSV Delimiter** used in the file. The default selection is Comma (.).
7. Click **OK**.

## Editing the Data Type

After uploading a CSV file data type, you can edit it as needed.

To edit the data type for a CSV file element, click the data type icon or update it from the element Properties dialog.

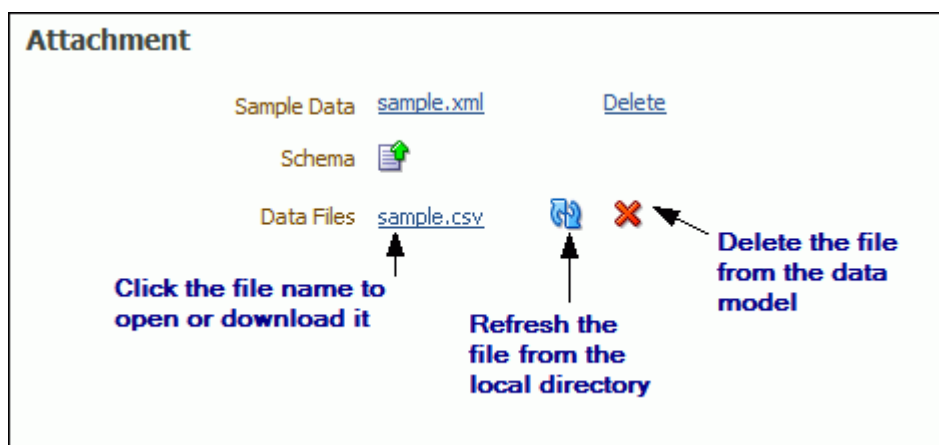
The data for an element must be compliant with the data type that you assign. The user interface does not validate the data when you update the data type. If the data does not match, for example, a string value is present for an element you defined as Integer, errors may occur in the layout editing tools and or at runtime.

You can only update the data types for CSV file data sources.

## Refreshing and Deleting an Uploaded CSV File

You can refresh and delete uploaded local CSV files.

After uploading the file, it is displayed on the Properties pane of the data model under the Attachments region, as shown below.



See [Setting Data Model Properties](#) for more information about the Properties pane.

To refresh the local file in the data model:

1. In the component pane, click **Data Model** to view the Properties page.
2. In the Attachment region of the page, locate the file in the Data Files list.
3. Click **Refresh**.
4. In the Upload dialog, browse for and upload the latest version of the file. The file must have the same name or it will not replace the older version.
5. Save the data model.

To delete the local file:

1. In the component pane, click **Data Model** to view the Properties page.
2. In the Attachment region of the page, locate the file in the Data Files list.
3. Click **Delete**.
4. Click **OK** to confirm.
5. Save the data model.

## Creating a Data Set from an HTTP XML Feed

Using the HTTP (XML Feed) data set type, you can create data models from RSS and XML feeds over the Web by retrieving data through the HTTP GET method.



### Note:

You might require additional configuration to access external data source feeds depending on your system's security. If the RSS feed is protected by Secure Sockets Layer (SSL), see *Configuring BI Publisher for Secure Socket Layer (SSL) Communication* in *Administrator's Guide for Oracle Business Intelligence Publisher*.

To include parameters for the data set, it is recommended that you define the parameters first, so that they are available for selection when defining the data set. See [Adding Parameters and Lists of Values](#).

There is no metadata available from HTTP XML feed data sets, therefore grouping and linking are not supported.

## Creating a Data Set from an HTTP XML Data Set

You can set up an HTTP (XML Feed) data sources in two different ways.

- On the Administration page:  
Connections to HTTP data sources can be set up on the Administration page and then used in multiple data models. See *Setting Up a Connection to a HTTP XML Feed* in *Administrator's Guide for Oracle Business Intelligence Publisher*.
  - As a private data source:  
You can also set up a private connection accessible only to you. See [Managing Private Data Sources](#) for information about private data source connections.
1. On the toolbar, click **New Data Set** and select **HTTP (XML Feed)**. The New Data Set - HTTP (XML Feed) dialog launches, as shown below.

**New Data Set - HTTP (XML Feed)**

\* Name: News

\* Data Source: EMTest

\* URL Suffix: http://rss.news.yahoo.com/rss/topstories

Method: GET

Parameters: Add Parameter

Name	Value (Parameter)
------	-------------------

OK Cancel

2. Enter a name for this data set.
3. Select a data source.
4. Enter the URL Suffix for the source of the RSS or XML feed.
5. Select the Method: GET.
6. To add a parameter, click **Add Parameter**. Enter the **Name** and select the **Value**. The **Value** list is populated by the parameter **Name** defined in the **Parameters** section.  
See [Adding Parameters and Lists of Values](#).
7. Click **OK** to close the data set dialog.

## Using Data Stored as a Character Large Object (CLOB) in a Data Model

BI Publisher supports using data stored as a character large object (CLOB) data type in your data models. This feature enables you to use XML data generated by a separate process and stored in your database as input to a BI Publisher data model.

Use the Query Builder to retrieve the column in your SQL query, then use the data model editor to specify how you want the data structured. When the data model is executed, the data engine can structure the data either as:

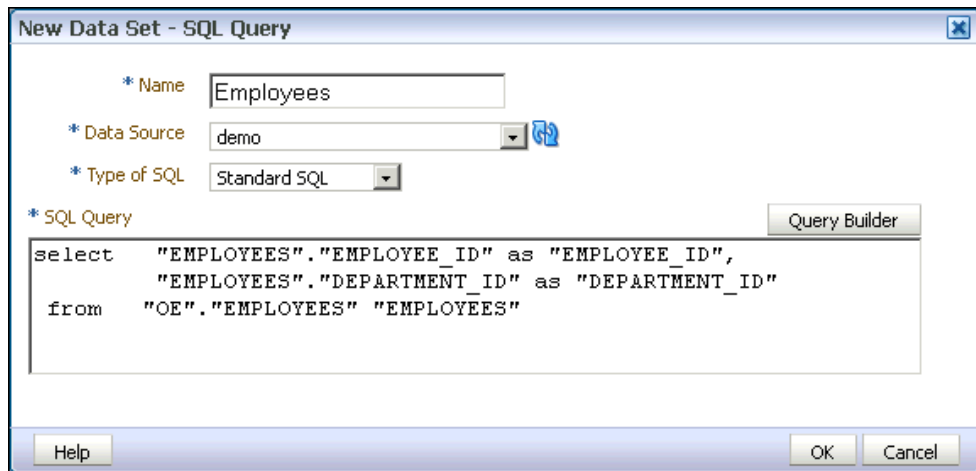
- A plain character set within an XML tag name that can be displayed in a report (for example, an Item Description)
- Structured XML

### **Note:**

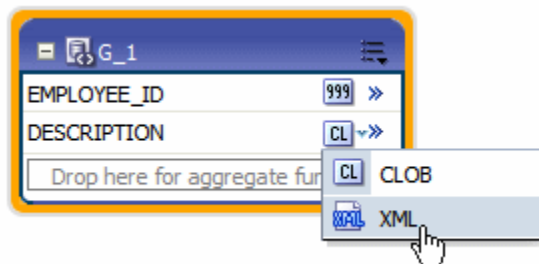
Ensure that your data does not include line feeds or carriage returns. Line feeds and carriage returns in your data may not render as expected in BI Publisher report layouts.

To create a data set from data stored as a CLOB:

1. On the toolbar, click **New Data Set** and then select **SQL Query**. The New Data Set - SQL Query dialog launches.
2. Enter a name for the data set.
3. If you are not using the default data source for this data set, select the **Data Source** from the list.
4. Enter the SQL query or use the **Query Builder** to construct your query to retrieve the CLOB data column. See [Using the SQL Query Builder](#) for information on the Query Builder utility. For example, you could create a query in which the CLOB data is stored in a column named "DESCRIPTION".



5. After entering the query, click **OK** to save. BI Publisher validates the query.
6. By default, the data model editor assigns the CLOB column the "CLOB" data type. To change the data type to XML, click the data type icon and select XML.



## How the Data Is Returned

When you execute the query, if the CLOB column contains well-formed XML, and you select the XML data type, the data engine returns the XML data, structured within the CLOB column tag name.

### Example output when data type is XML:

Note the <DESCRIPTION> element contains the XML data stored in the CLOB column, as shown below.

```

- <DATA_DS>
  - <G_1>
    <EMPLOYEE_ID>102</EMPLOYEE_ID>
  - <DESCRIPTION>
    - <DATA_DS>
      - <G_Q1>
        <DEPTNO>10</DEPTNO>
        <DNAME>PURCHASE</DNAME>
        <LOC>HQ</LOC>
      - <G_Q2>
        <DEPTNO_1>10</DEPTNO_1>
        <EMPNO>10001</EMPNO>
        <ENAME>SCOTT</ENAME>
        <SAL>5000</SAL>
      </G_Q2>
    + <G_Q2></G_Q2>
    </G_Q1>
  - <G_Q1>
    <DEPTNO>20</DEPTNO>
    <DNAME>FINANCE</DNAME>
    <LOC>HQ</LOC>

```

#### Example output when data type is CLOB:

If you select to return the data as the CLOB data type, the returned data is structured as shown below.

```

- <DATA_DS>
  - <G_1>
    <EMPLOYEE_ID>102</EMPLOYEE_ID>
  - <DESCRIPTION>
    <DATA_DS> <G_Q1> <DEPTNO>10</DEPTNO>
    <DNAME>PURCHASE</DNAME> <LOC>HQ</LOC> <G_Q2> <DEPTNO_1>10</DEPTNO_1>
    <EMPNO>10001</EMPNO> <ENAME>SCOTT</ENAME> <SAL>5000</SAL> </G_Q2> <G_Q2>
    <DEPTNO_1>10</DEPTNO_1> <EMPNO>10002</EMPNO> <ENAME>SMITH</ENAME>
    <SAL>3000</SAL> </G_Q2> </G_Q1> <G_Q1> <DEPTNO>20</DEPTNO>
    <DNAME>FINANCE</DNAME> <LOC>HQ</LOC> <G_Q2> <DEPTNO_1>20</DEPTNO_1>
    <EMPNO>10003</EMPNO> <ENAME>AMY</ENAME> <SAL>5500</SAL> </G_Q2> <G_Q2>
    <DEPTNO_1>20</DEPTNO_1> <EMPNO>10004</EMPNO> <ENAME>MARLIN</ENAME>
    <SAL>4000</SAL> </G_Q2> </G_Q1> <G_Q1> <DEPTNO>30</DEPTNO>
    <DNAME>CORPORATE</DNAME> <LOC>HQ</LOC> </G_Q1> </DATA_DS>
  </DESCRIPTION>
  </G_1>
</DATA_DS>

```



## Additional Notes on Data Sets Using CLOB Column Data

More information is available on CLOB column data.

For specific notes on using CLOB column data in a bursting query, see [Adding a Bursting Definition to Your Data Model](#).

## Handling XHTML Data Stored in a CLOB Column

BI Publisher can retrieve data stored in the form of XHTML documents stored in a database CLOB column and render the markup in the generated report.

To enable the BI Publisher report rendering engine to handle the markup tags, you must wrap the XHTML data in a CDATA section within the XML report data that is passed by the data engine.

It is recommended that you store the data in the database wrapped with the CDATA section. You can then use a simple select statement to extract the data. If the data is not wrapped in the CDATA section, then you must include in your SQL statement instructions to wrap it.

The following sections describe how to extract XHTML data in each case:

- [Retrieving XHTML Data Wrapped in CDATA](#)
- [Wrapping the XHTML Data in CDATA in the Query](#)

To display the markup in a report, you must use the syntax described in Rendering HTML Formatted Data in a Report in *Report Designer's Guide for Oracle Business Intelligence Publisher*. This section also describes the supported HTML formats. Rendering the HTML markup in a report is supported for RTF templates only.

## Retrieving XHTML Data Wrapped in CDATA

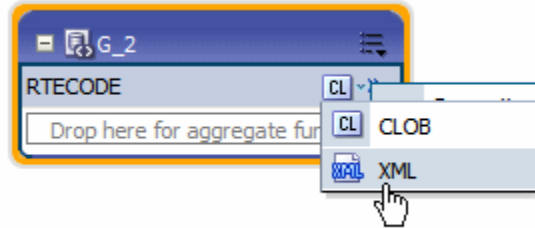
This exercise assumes you have the following data stored in a database column called "CLOB\_DATA".

```
<![CDATA[  
<p><font style="font-style: italic; font-weight: bold;" size="3">  
<a href="http://www.oracle.com">oracle</a></font> </p>  
<p><font size="6"><a href="http://docs.oracle.com/">Oracle Documentation</a>  
</font></p>  
]]>
```

Retrieve the column data using a simple SQL statement, for example:

```
select CLOB_DATA as "RTECODE" from MYTABLE
```

In the data model editor, set the data type of the **RTECODE** column to XML, as shown below.



## Wrapping the XHTML Data in CDATA in the Query

This exercise assumes you have the following data stored in a database column called "CLOB\_DATA".

```
<p><font style="font-style: italic; font-weight: bold;" size="3">  
<a href="http://www.oracle.com">oracle</a></font> </p>  
<p><font size="6"><a href="http://docs.oracle.com/">Oracle Documentation</a>  
</font></p>
```

Use the following syntax in your SQL query to retrieve it and wrap it in the CDATA section:

```
select '<![CDATA' || '[' || CLOB_DATA || ']' || '>' as "RTECODE" from MYTABLE
```

In the data model editor, set the data type of the **RTECODE** column to XML.

## Testing Data Models and Generating Sample Data

The data model editor enables you to test your data model and view the output to ensure your results are as expected.

After running a successful test, you can choose to save the test output as sample data for your data model. You can also use the Export feature to export sample data to a file. If your data model fails to run, you can view the data engine log.

To test your data model:

1. In the data model editor, select the **Data** tab, as shown below.



2. For SQL Query, Oracle BI Analysis, and View Object data sets: On the Data tab, select the number of rows to return. If you included parameters, enter the desired values for the test.
3. Click **View** to display the XML that is returned by the data model.
4. Select one of the following options to display the sample data:
  - Use **Tree View** to view the sample data in a data hierarchy. This is the default display option.

- Use **Table View** to view the sample data in a formatted table like you see in BI Publisher reports.

You can create a report based on this data model.

To save the test data set as sample data for the data model:

1. After the data model has successfully run, click **Save as Sample Data**. The sample data is saved to the data model. See [Adding Attachments to the Data Model](#) for more information.

To export the test data:

1. For SQL Query, Oracle BI Analysis, and View Object data sets: On the Data tab, select the number of rows to return.
2. After the data model has successfully run, click **Export**. You are prompted to open or save the file to a local directory.

To view the data engine log:

1. Click **View Data Engine Log**. You are prompted to open or save the file to a local directory. The data engine log file is an XML file.

To test UCM dataset:

For Content Server, based on the document ID and the content type the document content is retrieved from the content (UCM) server.



#### Note:

If the Document ID is empty or null, then the document content will be empty.

## Including User Information Stored in System Variables in Your Report Data

BI Publisher stores information about the current user that can be accessed by your report data model.

The user information is stored in system variables as described below.

System Variable	Description
xdo_user_name	User ID of the user submitting the report. For example: Administrator
xdo_user_roles	Roles assigned to the user submitting the report. For example: XMLP_ADMIN, XMLP_SCHEDULER
xdo_user_report_oracle_language	Report language from the user's account preferences. For example: ZHS
xdo_user_report_locale	Report locale from the user's account preferences. For example: en-US
xdo_user_ui_oracle_lang	User interface language from the user's account preferences. For example: US

System Variable	Description
xdo_user_ui_locale	User interface locale from the user's account preferences. For example: en-US

## Adding the User System Variables as Elements

To add the user information to the data model, you can define the variables as parameters and then define the parameter value as an element in your data model.

You can also simply add the variables as parameters then reference the parameter values in your report.

The following query:

```
select
:xdo_user_name as USER_ID,
:xdo_user_roles as USER_ROLES,
:xdo_user_report_oracle_lang as REPORT_LANGUAGE,
:xdo_user_report_locale as REPORT_LOCALE,
:xdo_user_ui_oracle_lang as UI_LANGUAGE,
:xdo_user_ui_locale as UI_LOCALE
from dual
```

returns the following results:

```
<?xml version="1.0" encoding="UTF-8"?>
<! - Generated by Oracle BI Publisher - >
<DATA_DS>
<G_1>
<USER_ROLES>XMLP_TEMPLATE_DESIGNER, XMLP_DEVELOPER, XMLP_ANALYZER_EXCEL, XMLP_ADMIN,
XMLP_ANALYZER_ONLINE, XMLP_SCHEDULER </USER_ROLES>
<REPORT_LANGUAGE>US</REPORT_LANGUAGE>
<REPORT_LOCALE>en_US</REPORT_LOCALE>
<UI_LANGUAGE>US</UI_LANGUAGE>
<UI_LOCALE>en_US</UI_LOCALE>
<USER_ID>administrator</USER_ID>
</G_1>
</DATA_DS>
```

## Sample Use Case: Limit the Returned Data Set by User ID

The following example limits the data returned by the user ID.

```
select EMPLOYEES.LAST_NAME as LAST_NAME,
EMPLOYEES.PHONE_NUMBER as PHONE_NUMBER,
EMPLOYEES.HIRE_DATE as HIRE_DATE,
:xdo_user_name as USERID
from HR.EMPLOYEES EMPLOYEES
where lower(EMPLOYEES.LAST_NAME) = :xdo_user_name
```

Notice the use of the lower() function, the xdo\_user\_name is always be in lowercase format. BI Publisher does not have a USERID so you must use the user name and either use it directly in the query; or alternatively you could query against a lookup table to find a user id.

## Creating Bind Variables from LDAP User Attribute Values

To bind user attribute values stored in your LDAP directory to a data query you can define the attribute names to BI Publisher to create the bind variables required.

### Prerequisite

The attributes that can be used to create bind variables must be defined in the Security Configuration page by an administrator.

The attributes are defined in the Attribute Names for Data Query Bind Variables field of the LDAP Security Model definition. See *Configuring BI Publisher to Use an LDAP Provider for Authentication and Authorization in Administrator's Guide for Oracle Business Intelligence Publisher*. Any attribute defined for users can be used (for example: memberOf, sAMAccountName, primaryGroupID, mail).

### How BI Publisher Constructs the Bind Variable

You can reference the attribute names that you enter in the **Attribute Names for Data Query Bind Variables** field of the LDAP Security Model definition in the query.

The following shows how bind variables are constructed:

```
xdo_<attribute name>
```

Assume that you have entered the sample attributes: memberOf, sAMAccountName, primaryGroupID, mail. These can then be used in a query as the following bind variables:

```
xdo_memberof  
xdo_SAMACCOUNTNAME  
xdo_primaryGroupID  
xdo_mail
```

Note that the case of the attribute is ignored; however, the "xdo\_" prefix must be lowercase.

Use these in a data model as follows:

```
SELECT  
:xdo_user_name AS USER_NAME,  
:xdo_user_roles AS USER_ROLES,  
:xdo_user_ui_oracle_lang AS USER_UI_LANG,  
:xdo_user_report_oracle_lang AS USER_REPORT_LANG,  
:xdo_user_ui_locale AS USER_UI_LOCALE,  
:xdo_user_report_locale AS USER_REPORT_LOCALE,  
:xdo_SAMACCOUNTNAME AS SAMACCOUNTNAME,  
:xdo_memberof as MEMBER_OF,  
:xdo_primaryGroupID as PRIMARY_GROUP_ID,  
:xdo_mail as MAIL  
FROM DUAL
```

The LDAP bind variables return the values stored in the LDAP directory for the user that is logged in.

# 3

## Structuring Data

This topic describes techniques for structuring the data that is returned by BI Publisher's data engine, including grouping, linking, group filters, and group-level and global-level functions.

### Topics:

- [Working with Data Models](#)
- [Features of the Data Model Editor](#)
- [About the Interface](#)
- [Creating Links Between Data Sets](#)
- [Creating Element-Level Links](#)
- [Creating Group-Level Links](#)
- [Creating Subgroups](#)
- [Moving an Element Between a Parent Group and a Child Group](#)
- [Creating Group-Level Aggregate Elements](#)
- [Creating Group Filters](#)
- [Performing Element-Level Functions](#)
- [Setting Element Properties](#)
- [Sorting Data](#)
- [Performing Group-Level Functions](#)
- [Performing Global-Level Functions](#)
- [Using the Structure View to Edit Your Data Structure](#)
- [Function Reference](#)

## Working with Data Models

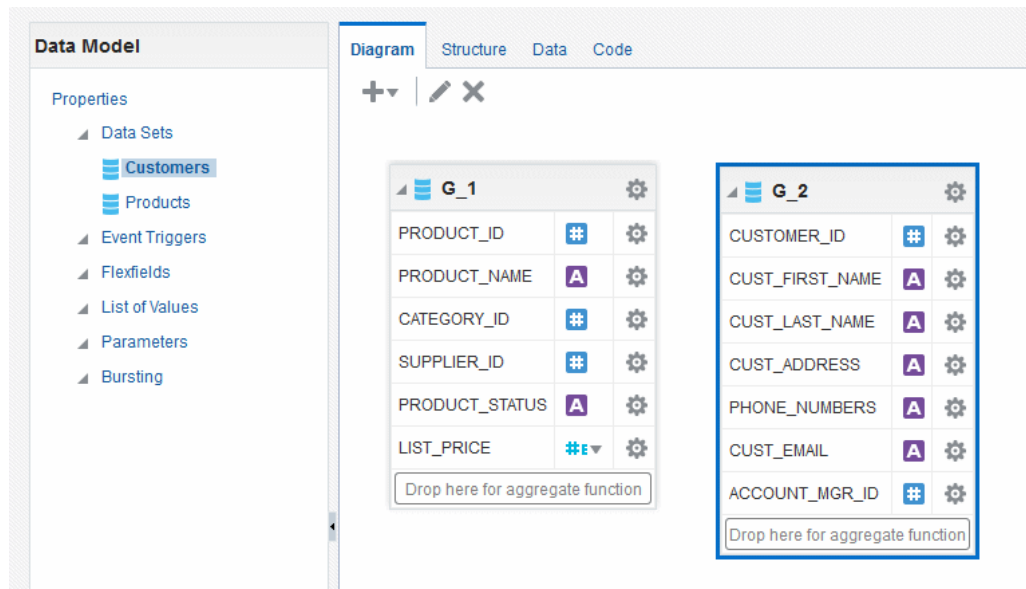
The Data Model diagram helps you to quickly and easily define data sets, break groups, and totals for a report based on multiple data sets.

- [About Multipart Unrelated Data Sets](#)
- [About Multipart Related Data Sets](#)
- [Guidelines for Working with Data Sets](#)

## About Multipart Unrelated Data Sets

If you do not link the data sets (or queries), the data engine produces a multipart unrelated query data set.

For example, in the data model, image shown below, one query selects *products* and another selects *customers*. There is no relationship between the products and customers.



The result is shown in the data structure as depicted in the following image.

The screenshot shows the Oracle Data Modeler interface with the 'Structure' tab selected. Under 'Table View', the 'Output' view is active. It displays two columns: 'XML output' and 'Descriptive XML output'. Both columns show a hierarchical tree structure for a data set named 'DATA\_DS'. The tree is expanded to show two groups, 'G\_1' and 'G\_2'. Group 'G\_1' contains attributes: PRODUCT\_ID, PRODUCT\_NAME, CATEGORY\_ID, SUPPLIER\_ID, PRODUCT\_STATUS, and LIST\_PRICE. Group 'G\_2' contains attributes: CUSTOMER\_ID, CUST\_FIRST\_NAME, CUST\_LAST\_NAME, CUST\_ADDRESS, PHONE\_NUMBERS, CUST\_EMAIL, and ACCOUNT\_MGR\_ID.

XML output	Descriptive XML output
▲ DATA_DS	▲ DATA_DS
▲ G_1	▲ G_1
PRODUCT_ID	PRODUCT_ID
PRODUCT_NAME	PRODUCT_NAME
CATEGORY_ID	CATEGORY_ID
SUPPLIER_ID	SUPPLIER_ID
PRODUCT_STATUS	PRODUCT_STATUS
LIST_PRICE	LIST_PRICE
▲ G_2	▲ G_2
CUSTOMER_ID	CUSTOMER_ID
CUST_FIRST_NAME	CUST_FIRST_NAME
CUST_LAST_NAME	CUST_LAST_NAME
CUST_ADDRESS	CUST_ADDRESS
PHONE_NUMBERS	PHONE_NUMBERS
CUST_EMAIL	CUST_EMAIL
ACCOUNT_MGR_ID	ACCOUNT_MGR_ID

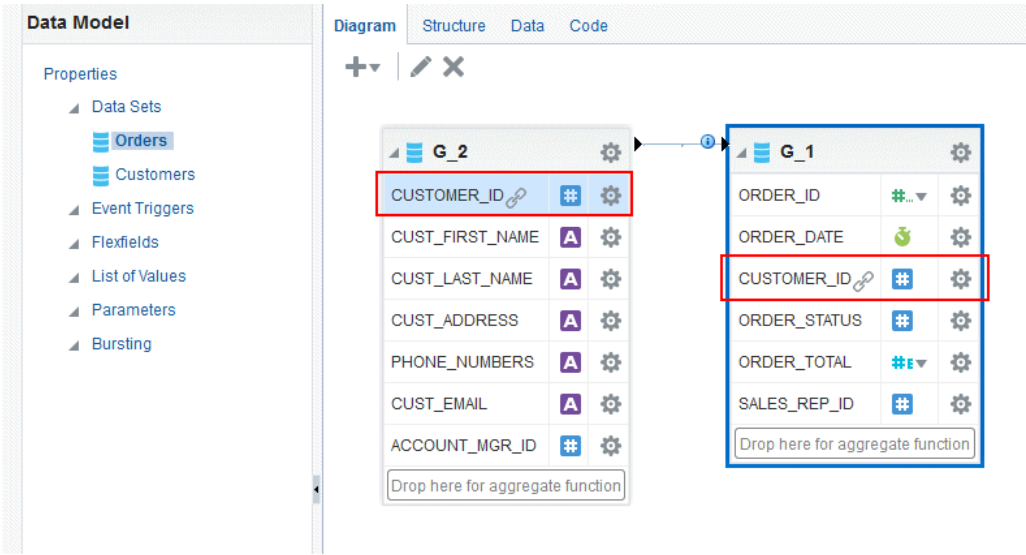
## About Multipart Related Data Sets

It is possible that the data fetched for one part of the data set or query is determined by the data fetched for another part. The result is often called a *master/detail*, or *parent/child* relationship that is defined with a data link between two data sets or queries.

When you run a master/detail data model, each row of the master (or parent) query executes a query against the detail (or child) to retrieve only matching rows.

In the example, image below, two data sets are linked by the element Customer ID. The Orders data set a child of the Customers data set.





The example produces the data structure shown in the following image.

The screenshot shows a software interface with four tabs: Diagram, Structure, Data, and Code. The 'Structure' tab is active. Below the tabs, there are two sub-tabs: 'Table View' and 'Output', with 'Output' selected. The main content area is divided into two columns: 'XML output' and 'Descriptive XML output'. Both columns show a hierarchical tree structure. Under 'XML output', there is a root node 'DATA\_DS' with a child 'G\_2' containing attributes: CUSTOMER\_ID, CUST\_FIRST\_NAME, CUST\_LAST\_NAME, CUST\_ADDRESS, PHONE\_NUMBERS, CUST\_EMAIL, and ACCOUNT\_MGR\_ID. Below 'G\_2' is another child 'G\_1' with attributes: ORDER\_ID, ORDER\_DATE, CUSTOMER\_ID, ORDER\_STATUS, ORDER\_TOTAL, and SALES\_REP\_ID. The 'Descriptive XML output' column shows an identical tree structure.

## Guidelines for Working with Data Sets

Certain guidelines are recommended for building data models.

- Reduce the number of data sets or queries in your data model as much as possible. In general, the fewer data sets and queries you have, the faster your data model will run. While multi-query data models are often easier to understand, single-query data models tend to execute more quickly. It is important to understand that in parent-child queries, for every parent, the child query is executed.
- You should only use multi-query data models in the following scenarios:
  - To perform functions that the query type, such as a SQL query, does not support directly.

- To support complex views, for example, distributed queries or GROUP BY queries.
- To simulate a view when you do not have or want to use a view.

## Features of the Data Model Editor

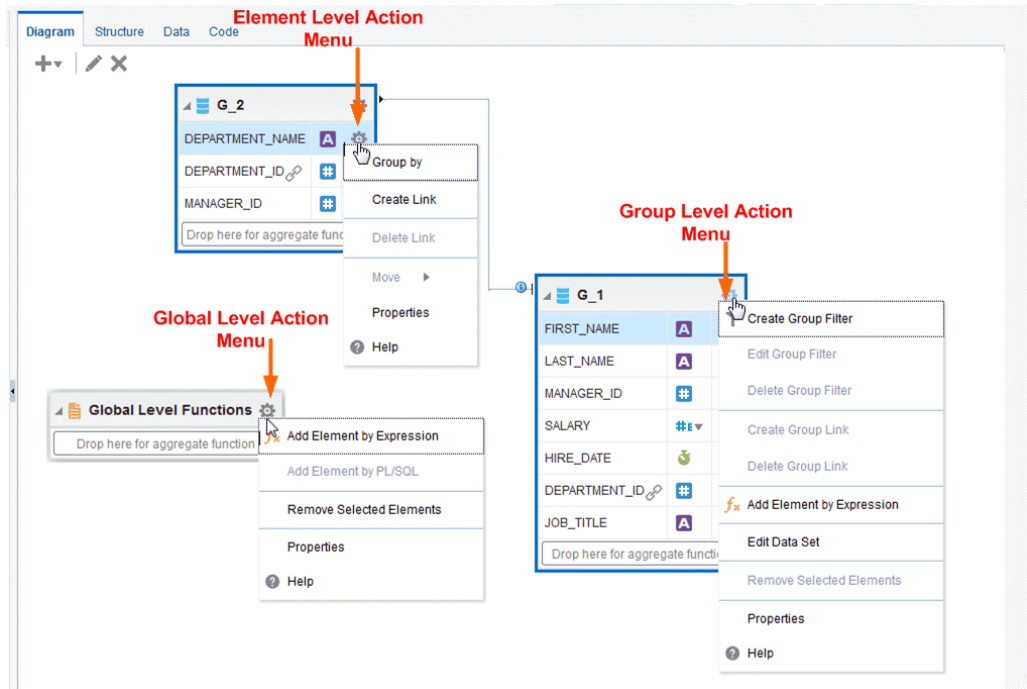
The data model editor enables you to combine data from multiple data sets into a single XML data structure.

Data sets from multiple data sources can be merged either as sequential XML or at line-level to create a single combined hierarchical XML. Using the data model editor you can easily combine data from the following data set types: SQL query, OLAP (MDX query), LDAP, and Microsoft Excel.

The data model editor supports the following

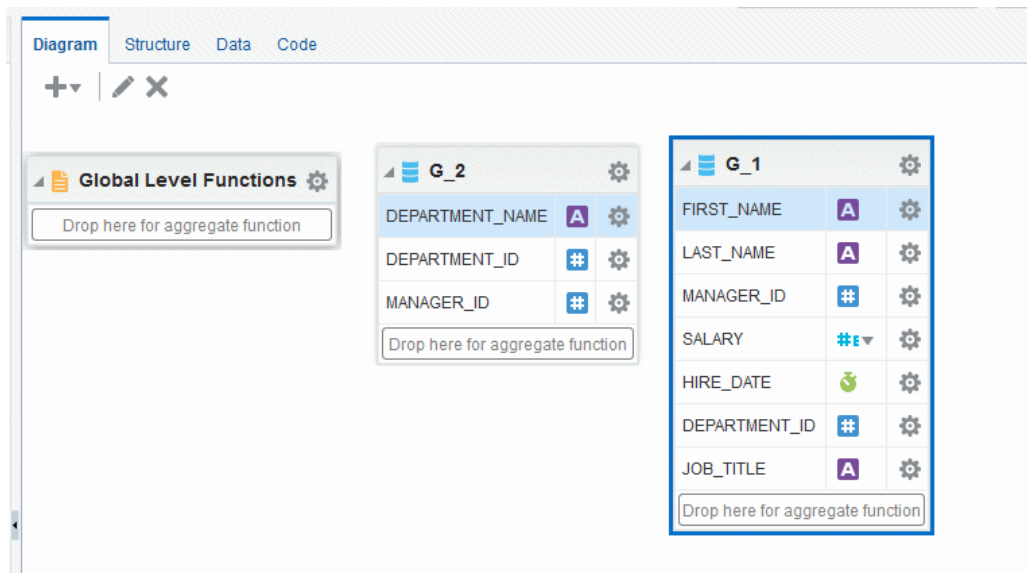
- **Group data** - Create groups to organize the columns in your report. Groups can do two things: separate a query's data into sets, and filter a query's data.  
  
When you create a query, the data engine creates a group that contains the columns selected by the query; you can create groups to modify the hierarchy of the data appearing in a data model. Groups are used primarily when you want to treat some columns differently than others. For example, you create groups to produce subtotals or create breaks.
- **Link data** - Define master-detail links between data sets to group data at multiple levels.
- **Aggregate data** - Create group level totals and subtotals.
- **Transform data** - Modify source data to conform to business terms and reporting requirements.
- **Create calculations** - Compute data values that are required for your report that are not available in the underlying data sources.

The data model editor provides functions at the element level, the group level, and the global level. Note that not all data set types support all functions. See the Important Notes section that accompanies your data set type for limitations. The figure below highlights some of the features and actions available in the data model editor.



## About the Interface

By default, the data sets that you created are shown in the Diagram View as separate objects.



The data set structure builder has three views:

- **Diagram View** - The **Diagram View** displays data sets and enables graphically creating links and filters, adding elements based on expressions, adding aggregate functions and global-level functions, editing element properties, and

deleting elements. The Diagram View is typically the view you use to build your data structure.

- **Structure View** - The **Structure View** has two modes:

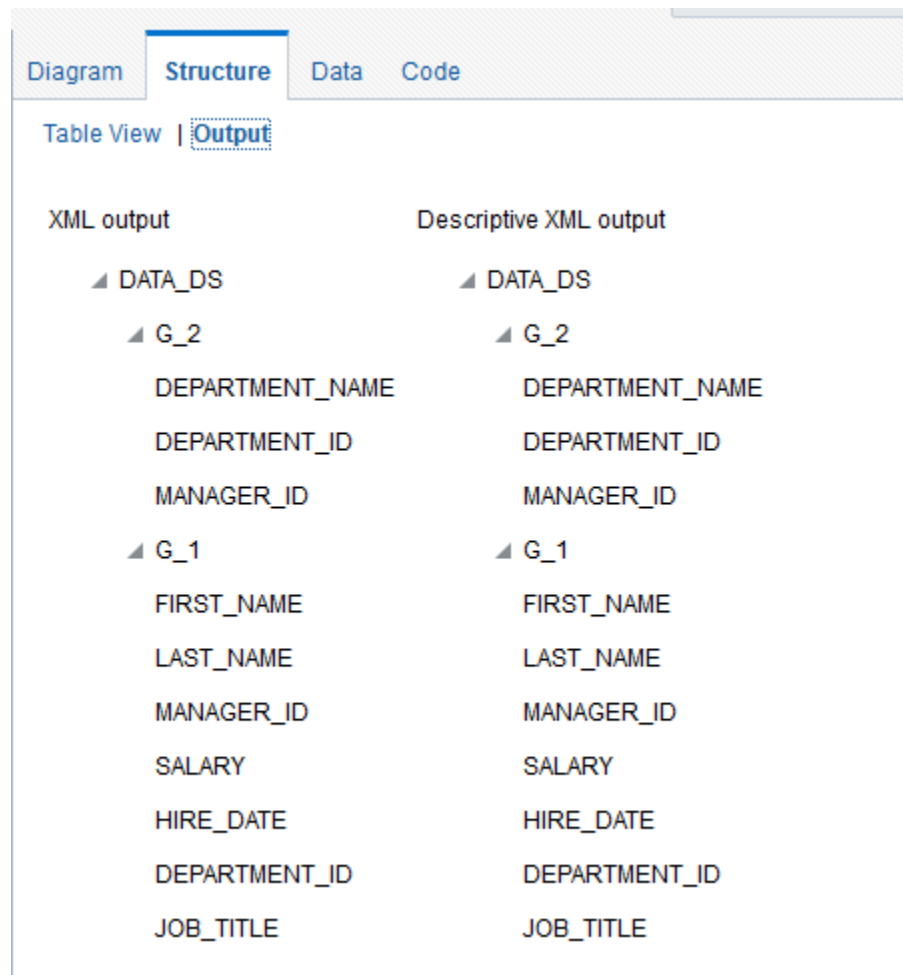
Table View and Output

The table view displays element properties in a table and enables updating XML element alias names, presentation names of the elements, sorting, null values, and reset options. The image below shows the structure Table View.

The screenshot shows a software interface with tabs for 'Diagram', 'Structure', 'Data', and 'Code'. The 'Structure' tab is active, and the 'Table View' is selected. The table below displays the configuration for XML and Business views.

Data Source	XML View			Business View	
	XML Tag Name	Sorting	Value If Null	Display Name	Data Type
Report Data					
Data Structure	DATA_DS				
Departments	G_2				
DEPARTMENT_NAME	DEPARTMENT_NAME			DEPARTMENT_NAME	A
DEPARTMENT_ID	DEPARTMENT_ID			DEPARTMENT_ID	#
MANAGER_ID	MANAGER_ID			MANAGER_ID	#
Employees	G_1				
FIRST_NAME	FIRST_NAME			FIRST_NAME	A
LAST_NAME	LAST_NAME			LAST_NAME	A
MANAGER_ID	MANAGER_ID			MANAGER_ID	#
SALARY	SALARY			SALARY	##
HIRE_DATE	HIRE_DATE			HIRE_DATE	🕒
DEPARTMENT_ID	DEPARTMENT_ID			DEPARTMENT_ID	#
JOB_TITLE	JOB_TITLE			JOB_TITLE	A

The **Output** view provides a clear view of the XML structure that is generated. The Output view cannot be updated. The figure shows the Output view.



- **Code View** - The **Code View** displays the data structure code created by the data structure builder that is read by the data engine. You can update the content in code view. The figure shows the code view.

```

Diagram  Structure  Data  Code
<output rootName="DATA_DS" uniqueRowName="false">
<nodeList name="data-structure">
<dataStructure tagName="DATA_DS">
<group name="G_2" label="G_2" source="Departments">
<element name="DEPARTMENT_NAME" value="DEPARTMENT_NAME" label="DEPARTMENT_NAME"
dataType="xsd:string" breakOrder="" fieldOrder="1"/>
<element name="DEPARTMENT_ID" value="DEPARTMENT_ID" label="DEPARTMENT_ID"
dataType="xsd:integer" breakOrder="" fieldOrder="2"/>
<element name="MANAGER_ID" value="MANAGER_ID" label="MANAGER_ID" dataType="xsd:integer"
breakOrder="" fieldOrder="3"/>
</group>
<group name="G_1" label="G_1" source="Employees">
<element name="FIRST_NAME" value="FIRST_NAME" label="FIRST_NAME" dataType="xsd:string"
breakOrder="" fieldOrder="1"/>
<element name="LAST_NAME" value="LAST_NAME" label="LAST_NAME" dataType="xsd:string"
breakOrder="" fieldOrder="2"/>
<element name="MANAGER_ID" value="MANAGER_ID" label="MANAGER_ID" dataType="xsd:integer"
breakOrder="" fieldOrder="3"/>
<element name="SALARY" value="SALARY" label="SALARY" dataType="xsd:double" breakOrder=""
fieldOrder="4"/>
<element name="HIRE_DATE" value="HIRE_DATE" label="HIRE_DATE" dataType="xsd:date"
breakOrder="" fieldOrder="5" formatMask=""/>
<element name="DEPARTMENT_ID" value="DEPARTMENT_ID" label="DEPARTMENT_ID"
dataType="xsd:integer" breakOrder="" fieldOrder="6"/>
<element name="JOB_TITLE" value="JOB_TITLE" label="JOB_TITLE" dataType="xsd:string"
breakOrder="" fieldOrder="7"/>
</group>
</dataStructure>
</nodeList>
</output>

```

## Creating Links Between Data Sets

You can use the BI Publisher data engine to combine and structure data after you extract it from the data source.

Joining and structuring data at the source into one combined data set is sometimes not possible. For example, you cannot join data at the source when data resides in disparate sources such as Microsoft SQL Server and an Oracle Database. Even if your data is coming from the same source, if you are creating large reports or documents with potentially hundreds of thousands of rows or pages, structuring your data so that it matches the intended layout optimizes document generation.

Create a link to define a master-detail or parent-child relationship between two data sets. You can create links as element-level links or group-level links. The resulting, hierarchical XML data is the same. Creating links as element-level links is the preferred method. Group-level links are provided for backward compatibility with data templates from earlier versions of Oracle BI Publisher.

A data link or parent-child relationship relates the results of multiple queries. A data link can establish these relationships:

- Between one query's column and another query's column.
- Between one query's group and another query's group, useful when you want the child query to know about its parent's data.

## About Element-Level Links

Element-level links create a bind (join) between two data sets and define a master-detail (parent-child) relationship between them.

Create element-level links, the preferred method, to define master detail relationships between data sets. When you use element-level links to link data sets, you do not need to code a join between the two data sets through a bind variable.

## About Group-Level Links

Group-level links determine how data sets are structured as hierarchical XML, but lack the join information that the data engine needs to execute the master and detail queries.

When you define a group-level link, you must update your query with a link between the two data sets through a unique bind variable.

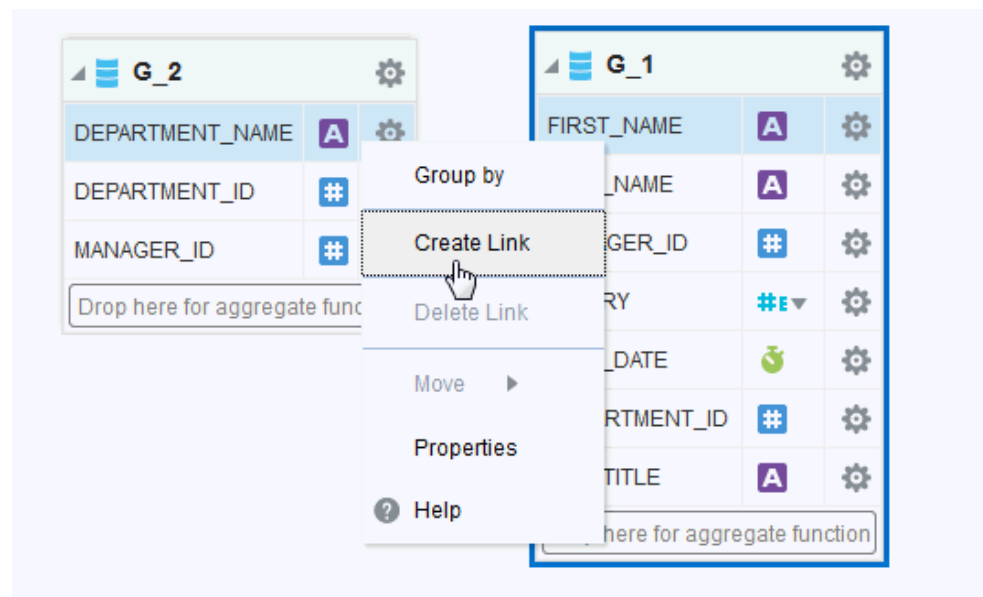
## Creating Element-Level Links

Link data sets to define a master-detail or parent-child relationship between two data sets.

Defining an element-level link enables you to establish the binding between the elements of the master and detail data sets.

1. Open the element action menu and click **Create Link**.
2. In the Create Link dialog, choose the element, and click **OK** to create the link.

The Create Link dialog is shown below.



## Deleting Element-Level Links

You can delete both group-level and element-level links between data sets.

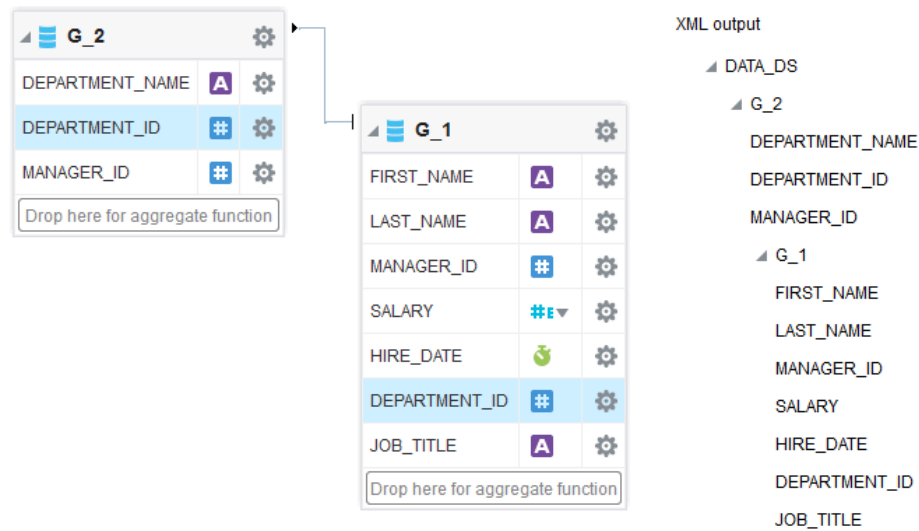
- Do one of the following:
  - Open the element action menu for either element and click **Delete Link**
  - Select the element connector to display the linked element names and click the delete button.



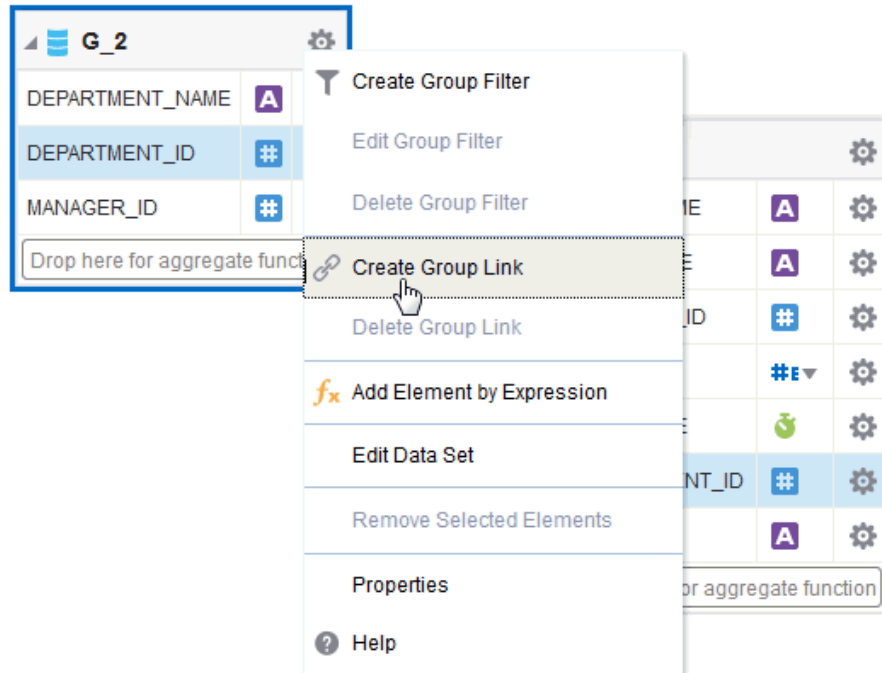
## Creating Group-Level Links

A group-level link defines a master-detail relationship between two data sets.

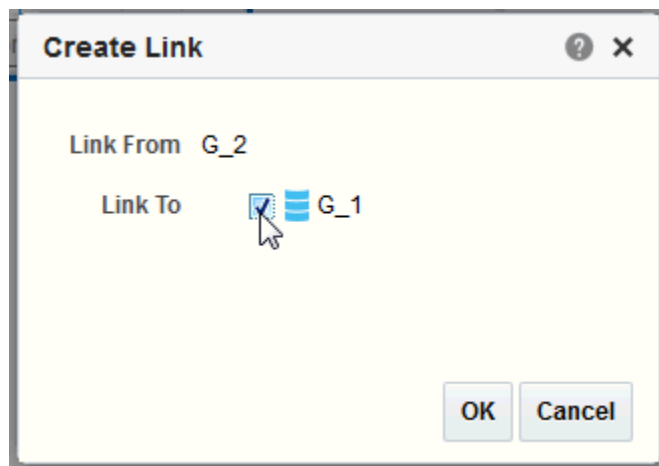
The following figure shows two data sets with a group-level link defined. Next to the data sets the resulting XML data structure is shown.



1. In the parent group, click **Menu**.
2. Click **Create Group Link** as shown below.



3. In the **Create Group Link** dialog, select the child group and click **OK**. The **Create Link** dialog is shown below.



4. Click **Menu** and then click **Edit Data Set** to add the bind variables to your query. An example is shown below.

Data Set: DEPT	Data Set: EMP
<pre>Select DEPT.DEPTNO as DEPTID,        DEPT.DNAME as DNAME,        DEPT.LOC as LOC from   OE.DEPT DEPT</pre>	<pre>Select EMP.EMPNO as EMPNO,        EMP.ENAME as ENAME,        EMP.JOB as JOB,        EMP.MGR as MGR,        EMP.HIREDATE as HIREDATE,        EMP.SAL as SAL,        EMP.COMM as COMM,        EMP.DEPTNO as DEPTNO from   OE.EMP EMP where  DEPTNO=:DEPTID</pre>

**Note:**

You must define a unique bind variable in the child query.

## Deleting Group-Level Links

You can delete both group-level and element-level links between data sets.

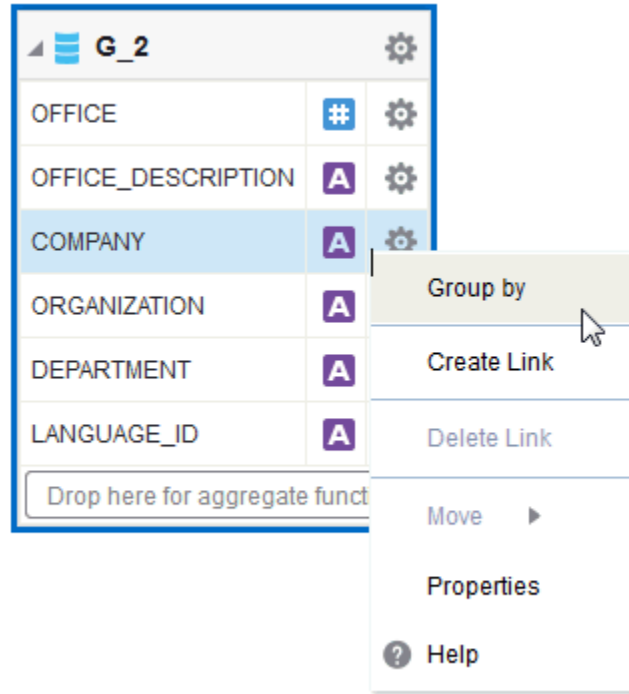
1. In the parent group, click **Menu**.
2. Click **Delete Group Link**.
3. In the **Delete Group Link** dialog, select the **Child Group** from the list and click **OK**.

## Creating Subgroups

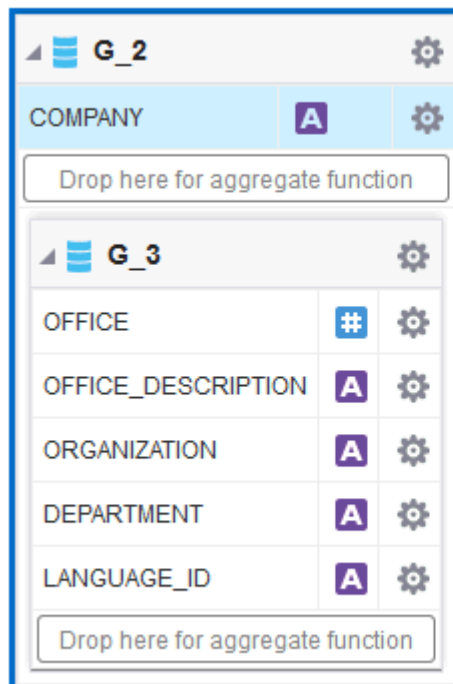
In addition to creating parent-child structures by linking two data sets, you can also group elements in the same data set by other elements.

Creating subgroups might be helpful if your query returns data that has header data repeated for each detail row. By creating a subgroup you can shape the XML data for better, more efficient document generation.

1. Select the element to group with the other elements in the data set.
2. Click the element action menu icon to open the menu and select **Group by** as shown.



This creates a new group within the displayed data set. The following figure shows the G\_2 data set grouped by the element COMPANY. This creates a new group called G\_3 that contains the other five elements in the data set. The following figure shows how the grouped data set is displayed in the Diagram View along with the structure.



XML output

```

└─ DATA_DS
  └─ G_2
    └─ COMPANY
      └─ G_3
        └─ OFFICE
          └─ OFFICE_DESCRIPTION
            └─ ORGANIZATION
              └─ DEPARTMENT
                └─ LANGUAGE_ID
  
```

You can perform any of the group actions on the group you have created.

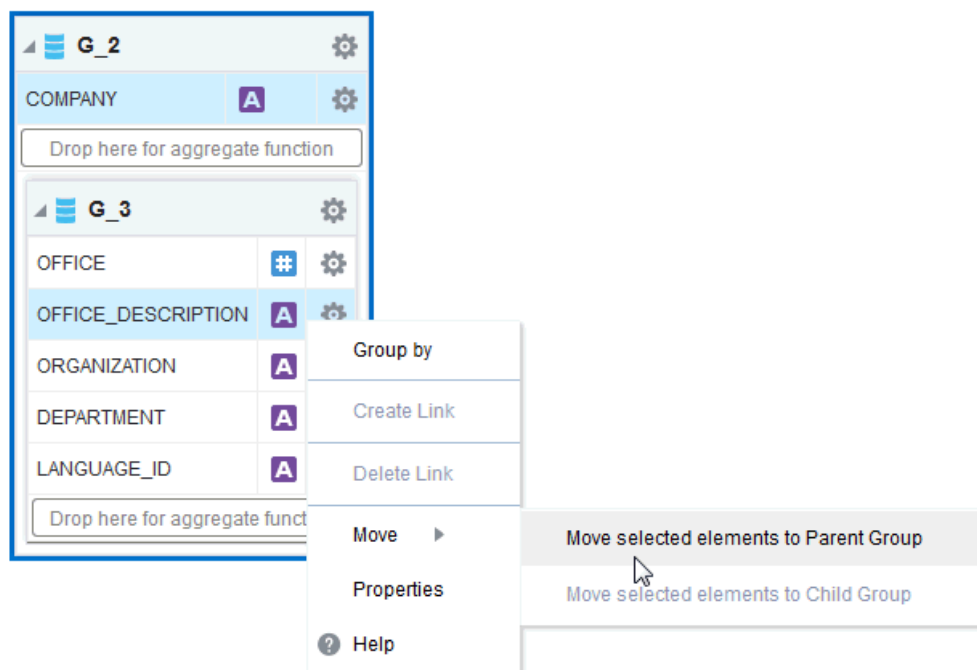
- To ungroup, click **Menu** on the group's title bar, and then click **Ungroup**.

## Moving an Element Between a Parent Group and a Child Group

Once you have created a group within your data set, two new options display on the element action menu that enable you to move elements between the parent and child groups.

For the element that you want to move, click the element action icon to open the menu. If the element is in the parent group and you want to move it to the child group, and then **Move this element to Child Group**.

If the element is in the child group and you want to move it to the parent group, select and then **Move this element to Parent Group**. In the figure below, the element action menu for OFFICE\_DSC displays the option to move the element to the parent group.



### Note:

Before moving an element be aware of any dependencies on other elements.

## Creating Group-Level Aggregate Elements

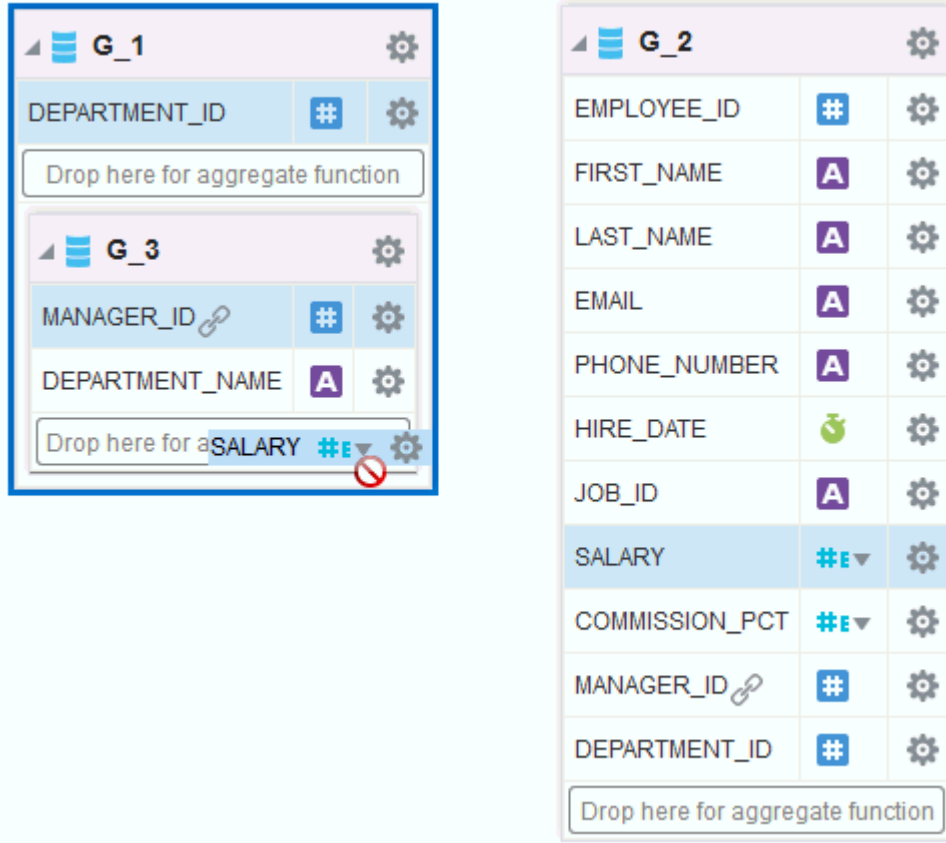
You can use the data model editor to aggregate data at the group or report level.

For example, if you group sales data by Customer Name, you can aggregate sales to get a subtotal for each customer's sales. You can only aggregate data for at the parent level for a child element.

The aggregate functions are:

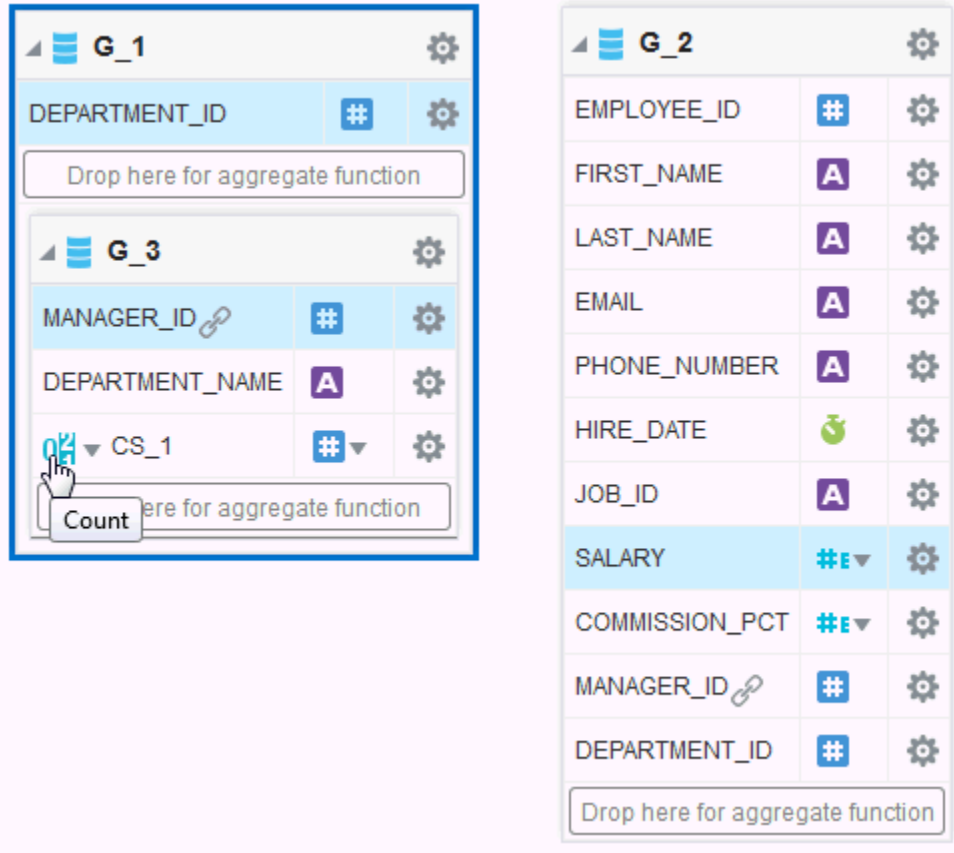
- **Average** - Calculates the average of all the occurrences of an element.
  - **Count** - Counts the number of occurrences of an element.
  - **First** - Displays the value of the first occurrence of an element in the group.
  - **Last** - Displays the value of the last occurrence of an element in the group.
  - **Maximum** - Displays the highest value of all occurrences of an element in the group.
  - **Minimum** - Displays the lowest value of all occurrences of an element in a group.
  - **Summary** - Sums the value of all occurrences of an element in the group.
1. Drag the element to the **Drop here for aggregate function** field in the parent group.

The figure below shows creating a group-level aggregate function in the G\_DEPT based on the SALARY element.



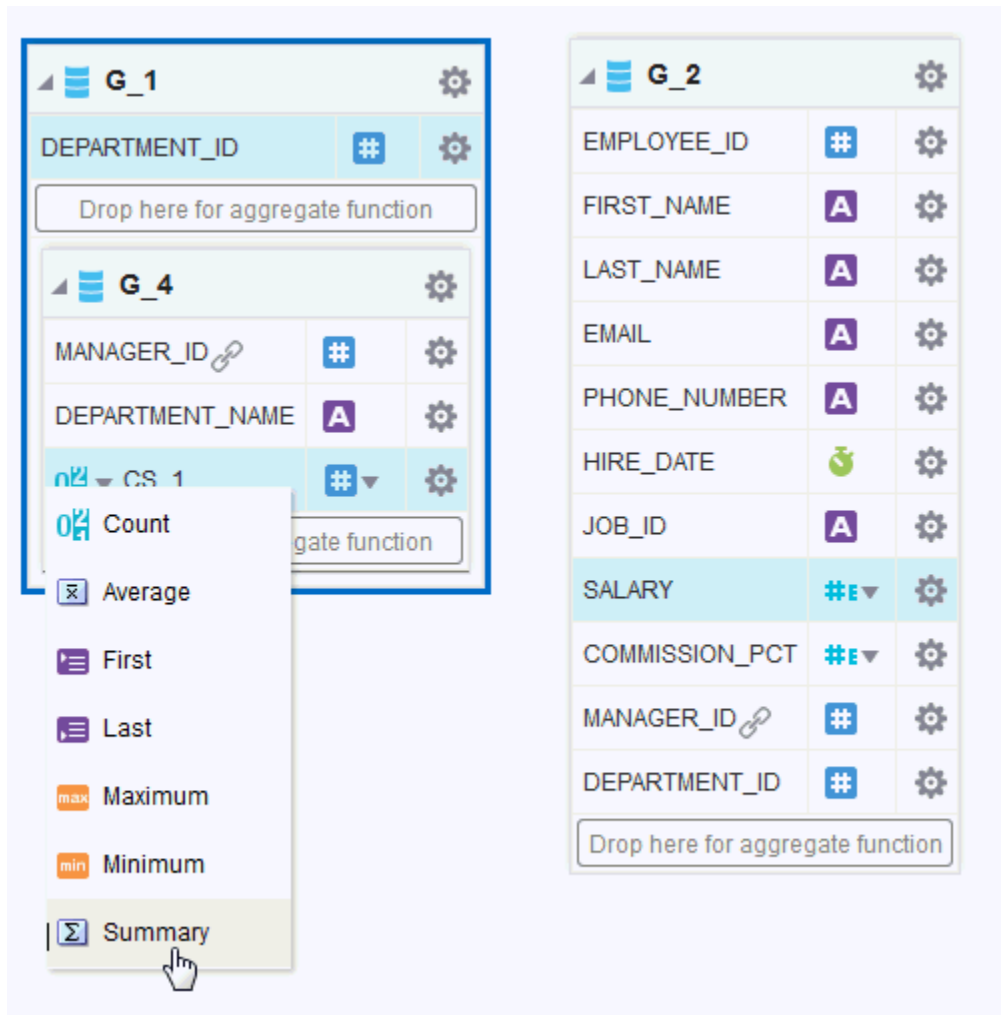
Once you drop the element, a new element is created in the parent group. By default, the Count function is applied. The icon next to the name of the new aggregate element indicates the function. Pause your cursor over the icon to display the function.

The figure below shows the new aggregate element, CS\_1, with the default Count function defined.

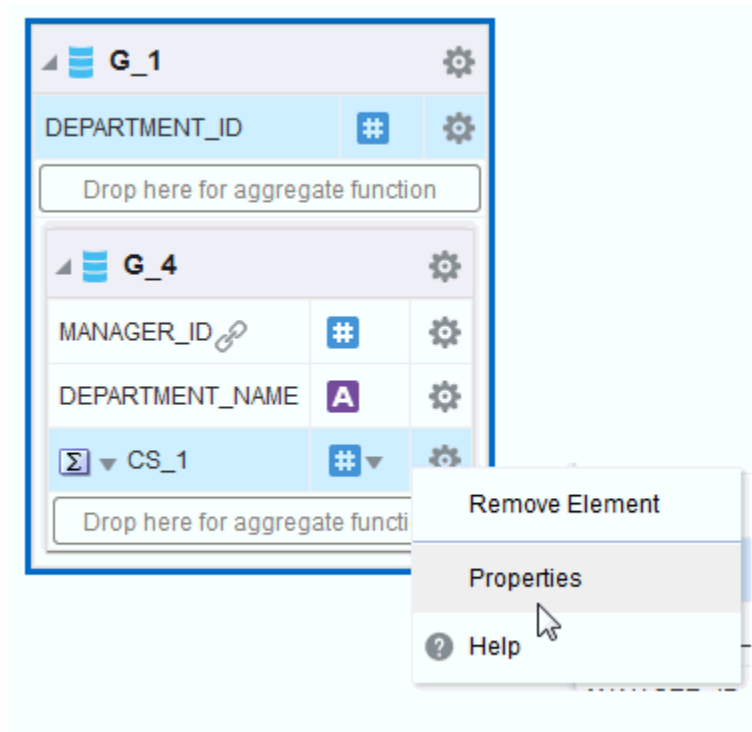


2. To change the function, click the function icon to view a list of available functions and choose from the list, as shown below.

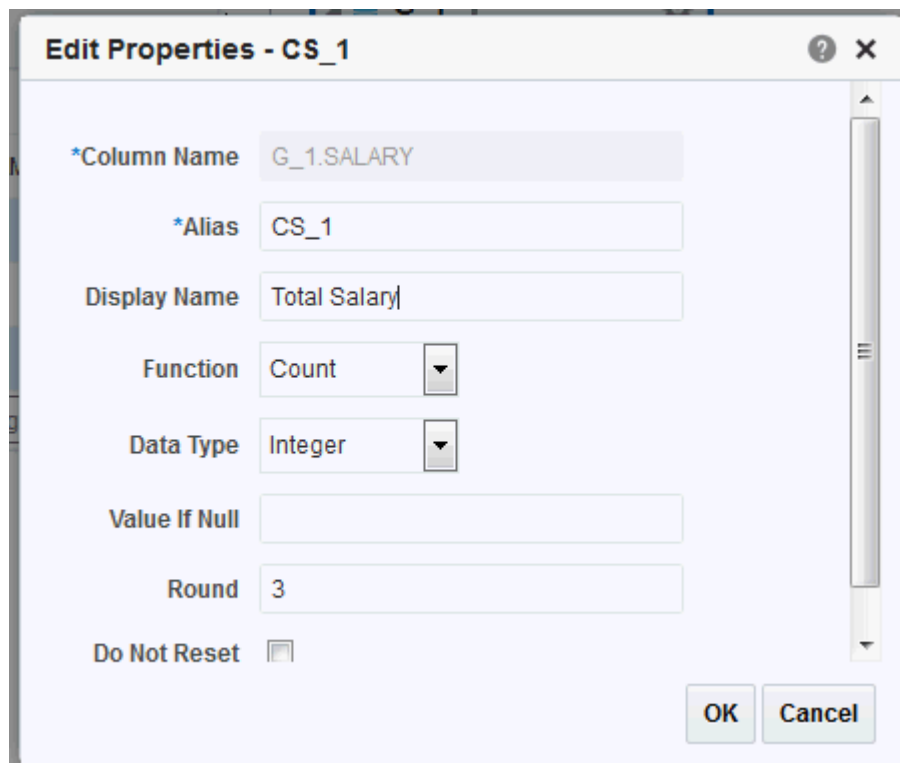




3. To rename the element or update other properties, click the element's **Action** menu icon.



On the menu, click **Properties**. The Properties dialog is shown below.



 **Note:**

Be careful when renaming an element as it can have dependency on other elements.

Set the properties described in the table below as needed.

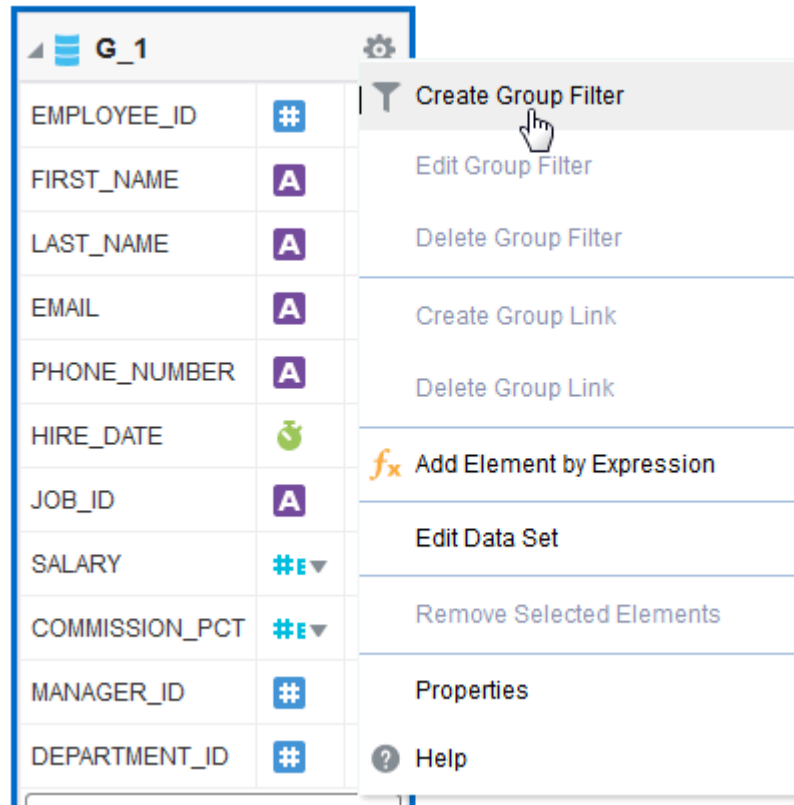
Property	Description
Column Name	The internal name assigned to the element by the BI Publisher data model editor. This name cannot be updated.
Alias (XML Tag Name)	Oracle BI Publisher assigns a default tag name for the element in the XML data file. You can update this tag name to assign a more user-friendly name within the data file.
Display Name	The Display Name appears in the report design tools. Update this name to be meaningful to your business users.
Function	If you have not already selected the desired function, then you can select it from the list here.
Data Type	BI Publisher assigns a default data type of Integer or Double depending on the function. Some functions also provide the option of Float.
Value if Null	If the value returned from the function is null, you can supply a default value here to prevent having a null in your data.
Round	By default, the value is rounded to the nearest third decimal. You can change the round value, if needed.
Do Not Reset	By default, the function resets at the group level. For example, if your data set is grouped by DEPARTMENT_ID, and you have defined a sum function for SALARY, then the sum is reset for each group of DEPARTMENT_ID data, giving you the sum of SALARY for that department only. If instead you want the function to reset only at the global level, and not at the group level, select <b>Do Not Reset</b> . This creates a running total of SALARY for all departments. This property is for group level functions only.

## Creating Group Filters

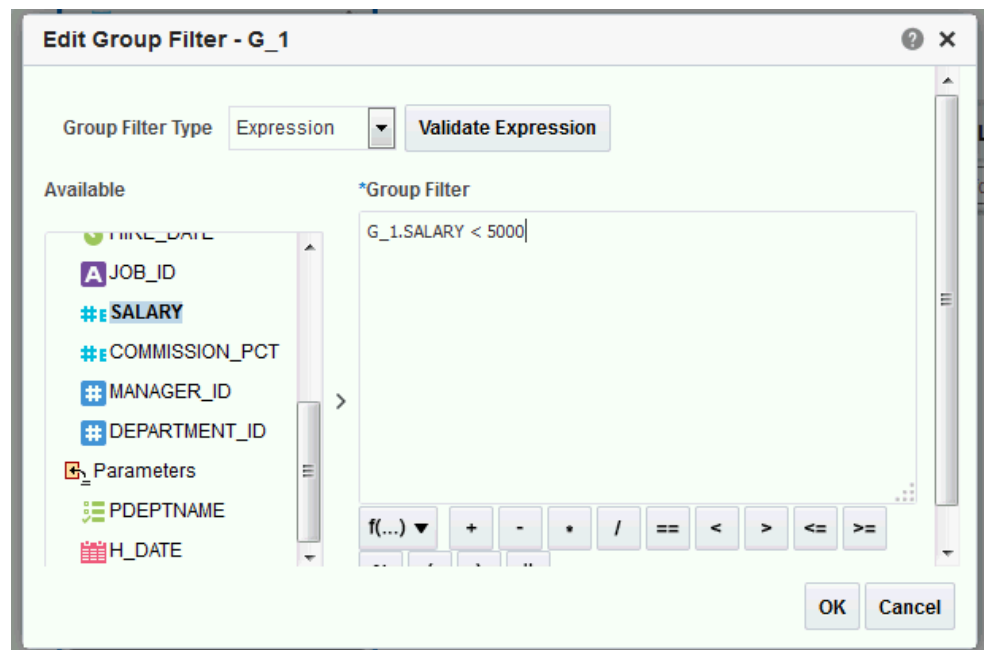
Filters enable you to conditionally remove records selected by your queries.

Groups can have two types of filters:

- Expression — Create an expression using predefined functions and operators
  - PL/SQL Function — Create a custom filter
1. Click **Menu**, and then select **Create Group Filter**.



The Edit Group Filter dialog is displayed as shown below.



2. Select the Group Filter Type: **Expression** or **PL/SQL**.

 **Note:**

For PL/SQL filters, you must first specify the PL/SQL Package as the **Oracle DB Default Package** in the data model properties. See [Setting Data Model Properties](#).

## 3. Enter the Filter:

- To enter an expression, select the elements and click the shuttle button to move the element to the Group Filter definition box. Click the predefined functions and operators to insert them in the Group Filter box.

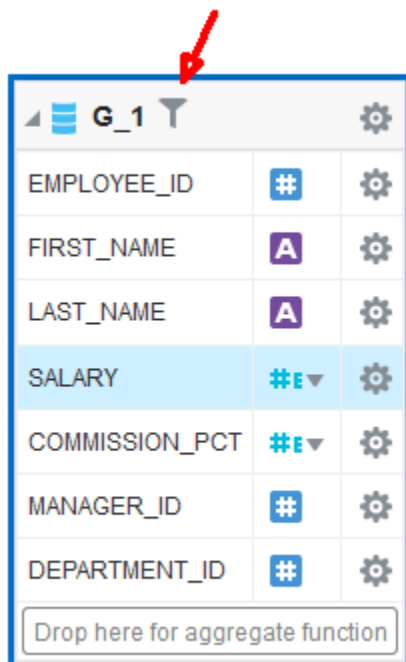
See [Function Reference](#) for a description of the available functions.

Click **Validate Expression** to ensure that the entry is valid.

- To enter a PL/SQL function, select the PL/SQL package from the Available box and click the shuttle button to move the function to the Group Filter box.

Your PL/SQL function in the default package must return a Boolean type.

After you have added the group filter, the data set object displays the filter indicator, as shown.



## Performing Element-Level Functions

You can perform various functions at the element level.

- Group by an element to create a subgroup, as described in [Creating Subgroups](#)
- Create element-level links between data sets, as described in [Creating Element-Level Links](#)

- Set element properties, as described in [Setting Element Properties](#)

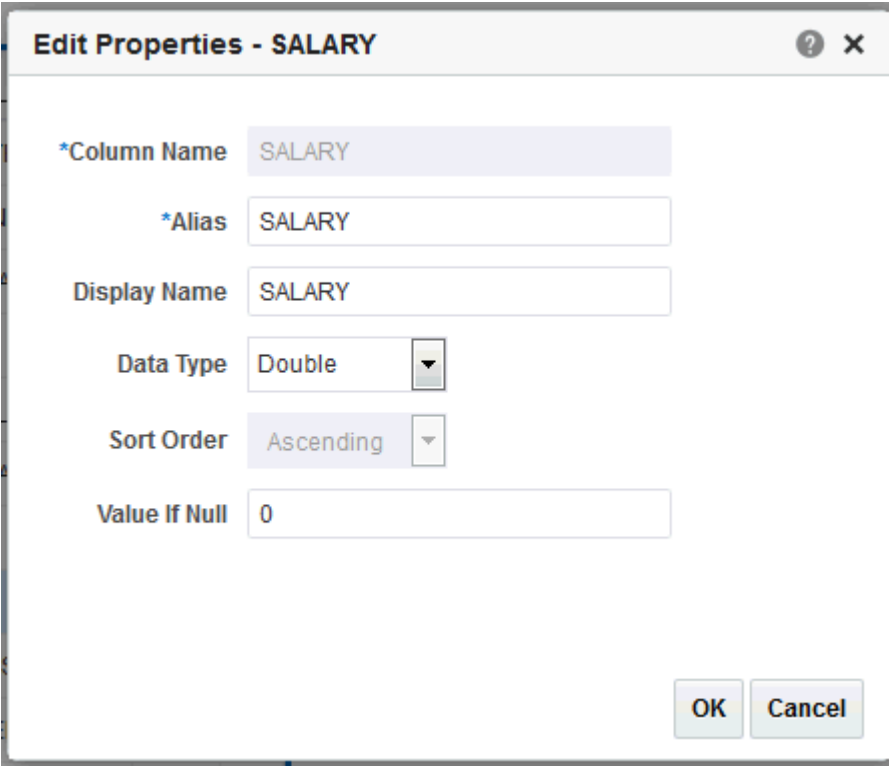
## Setting Element Properties

You can set properties for individual elements.

Note that these properties are also editable from the Structure View. If you need to update multiple element properties, it may be more efficient to use the Structure View. See [Using the Structure View to Edit Your Data Structure](#).

To set element-level properties using the element dialog:

1. Click the element's action menu icon. From the menu, select **Properties**. The Properties dialog is shown below.



The screenshot shows a dialog box titled "Edit Properties - SALARY". It contains the following fields and controls:

- \*Column Name: SALARY
- \*Alias: SALARY
- Display Name: SALARY
- Data Type: Double (dropdown menu)
- Sort Order: Ascending (dropdown menu)
- Value If Null: 0
- Buttons: OK, Cancel

2. Set the properties as needed.

Property	Description
Alias	BI Publisher assigns a default tag name to the element in the XML data file. You can update this tag name to assign a more user-friendly name within the data file.
Display Name	The Display Name appears in the report design tools and the column name in reports. Update this name to be meaningful to your business users.
Data Type	BI Publisher assigns a default data type. Valid values are String, Date, Integer, Double, Float.

Property	Description
Sort Order	You can sort XML data in a group by one or more elements. For example, if in a data set employees are grouped by department and manager, you can sort the XML data by department. Within each department you can group and sort data by manager, and within each manager subgroup, employees can be sorted by salary. If the element is not in a parent group, the <b>Sort Order</b> property is not available.
Value if Null	If the value of an occurrence of the element is null, you can supply a default value here to prevent having a null in your data.

## Sorting Data

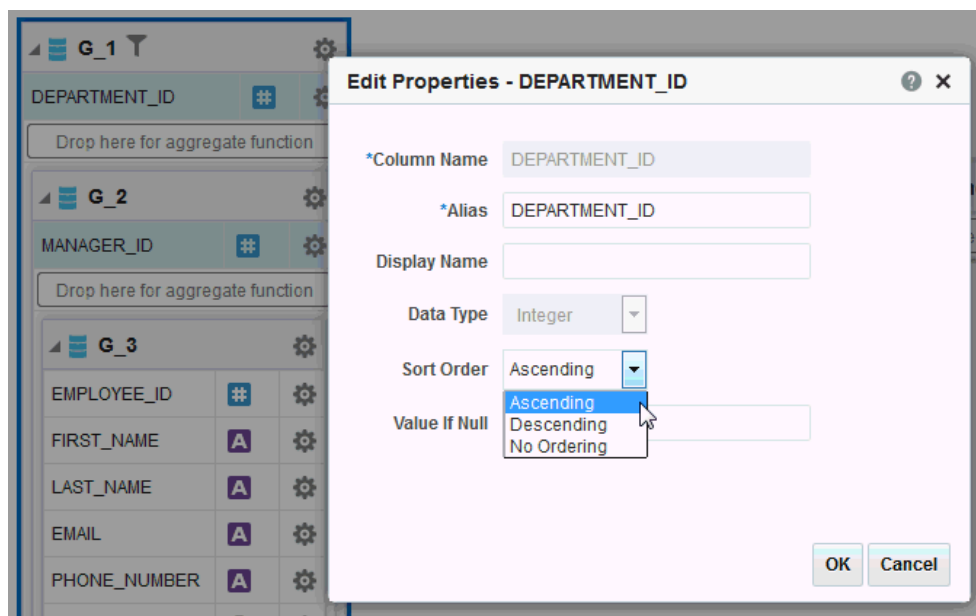
Sorting is supported for parent group break columns only.

For example, if a data set of employees is grouped by department and manager, you can sort the XML data by department. Within each department you can group and sort data by manager. If you know how the data should be sorted in the final report, you specify sorting at data generation time to optimize document generation.

To apply a sort order to a group:

1. Click the action menu icon of the element you want to sort by. From the menu, select **Properties**.
2. Select the **Sort Order**.

The figure below shows the Properties dialog for the DEPARTMENT\_ID element with the Sort Order list displayed.



## Performing Group-Level Functions

This section describes how to perform group-functions.

Topics include:

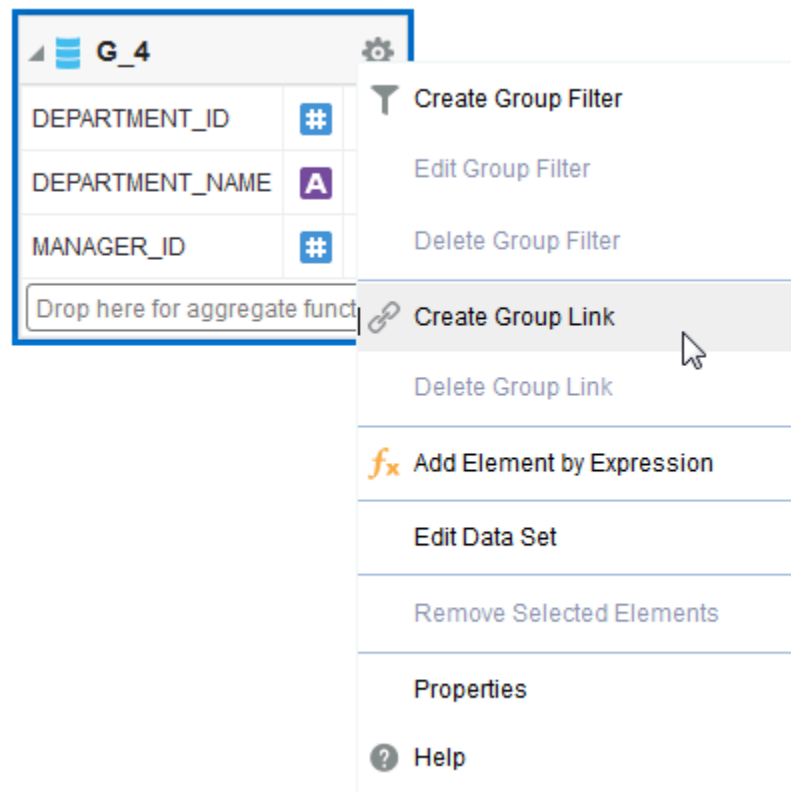
- [The Group Action Menu](#)
- [Editing the Data Set](#)
- [Removing Elements from the Group](#)
- [Editing the Group Properties](#)

## The Group Action Menu

The **Menu** button is available at the group level and enables to perform various functions.

- Create and delete group links, as described in [Creating Group-Level Links](#)
- Create, edit, and delete group filters, as described in [Creating Group Filters](#)
- Add an element to the group based on an expression, as described in [Adding a Group-Level or Global-Level Element by Expression](#)
- Edit the data set, as described in [Editing the Data Set](#)
- Remove elements from the group, as described in [Removing Elements from the Group](#)
- Edit group properties, as described in [Editing the Group Properties](#)

The group-level **Menu** button is shown below.





## Editing the Data Set

Launch the data set editor to modify properties of selected data sets.

- Click **Edit Data Set** to launch the data set editor.

See the appropriate section for the data set type in [Creating Data Sets](#) for more information.

## Removing Elements from the Group

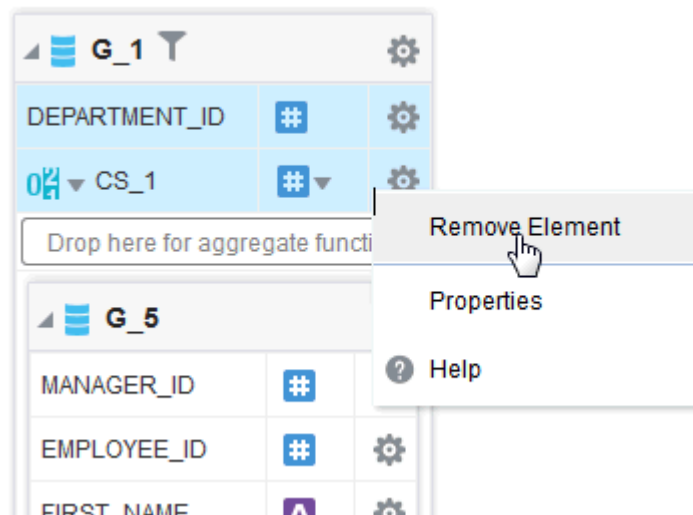
You can remove elements from groups as needed.

To remove an element from the group:

- On the element row, click the menu and then click **Remove Element**. An example is shown below.

### Note:

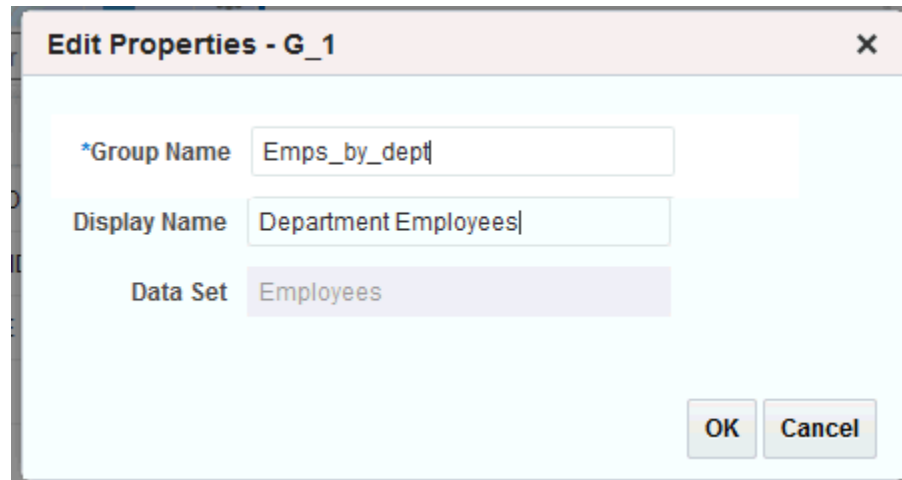
You can only remove elements added as a group function (sum, count, and so on) or added as an expression.



## Editing the Group Properties

Edit the properties of a group as needed.

1. Click **Menu** and select **Properties**.
2. Edit the **Group Name** or **Display Name** and click **OK**, as shown below.



## Performing Global-Level Functions

The Global Level Functions object enables you to add elements to your report data set at the top report level.

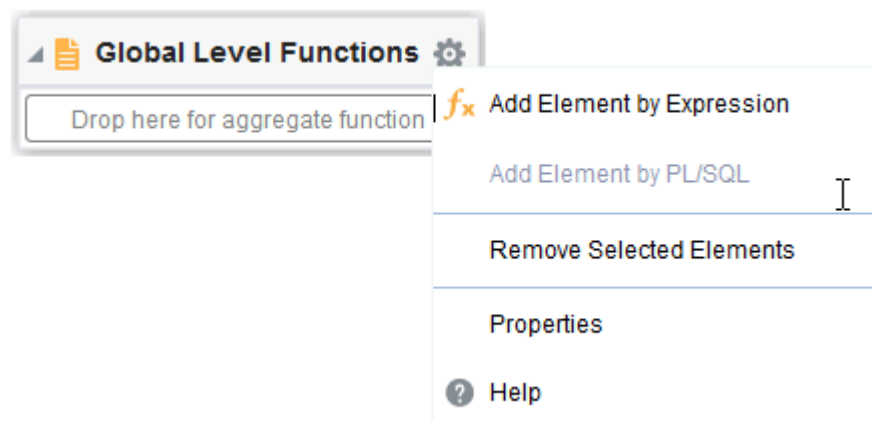
You can add the following types of elements as top-level data:

- Elements based on aggregate functions
- Elements based on expressions
- Elements based on PL/SQL statements (for Oracle Database data sources)

### Note:

If you select a data type of Integer for any calculated element and the expression returns a fraction, the data is not truncated.

The **Global Level Functions** object is shown below. To add elements based on aggregate functions, drag the element to the "Drop here for aggregate function" space of the object. To add an element based on an expression or PL/SQL, click **Menu**, and choose the appropriate action.



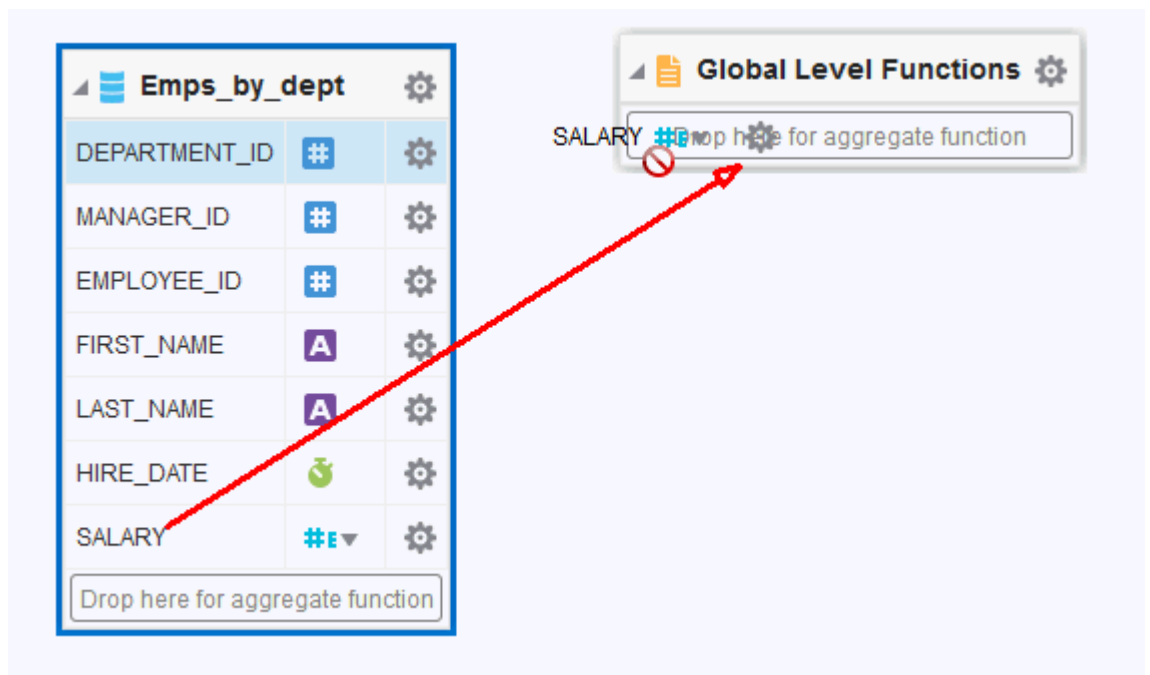
## Adding a Global-Level Aggregate Function

You can add global-level aggregate functions based on selected elements.

Available functions are as follows:

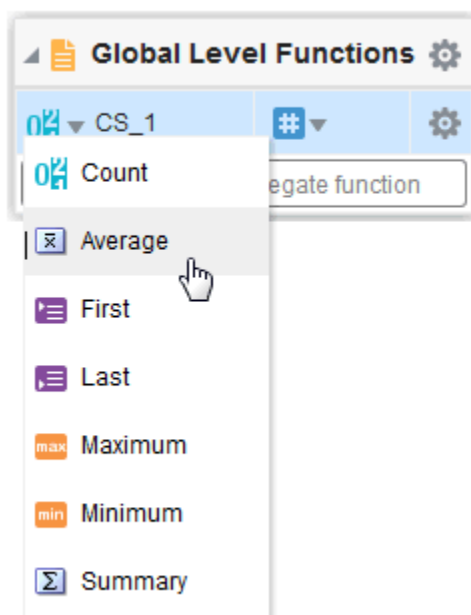
- Count
  - Average
  - First
  - Last
  - Maximum
  - Minimum
  - Summary
1. Drag and drop the data element from the data set to the **Drop here for aggregate function** area of the Global Level Functions object.

For example, the image below shows creating a global level aggregate function based on the Salary element.



2. When you release the mouse, the data model editor assigns a default name to the aggregate element and assigns Count as the default function.

The figure below shows the function for the new global level element CS\_1 being modified from Count to Average.

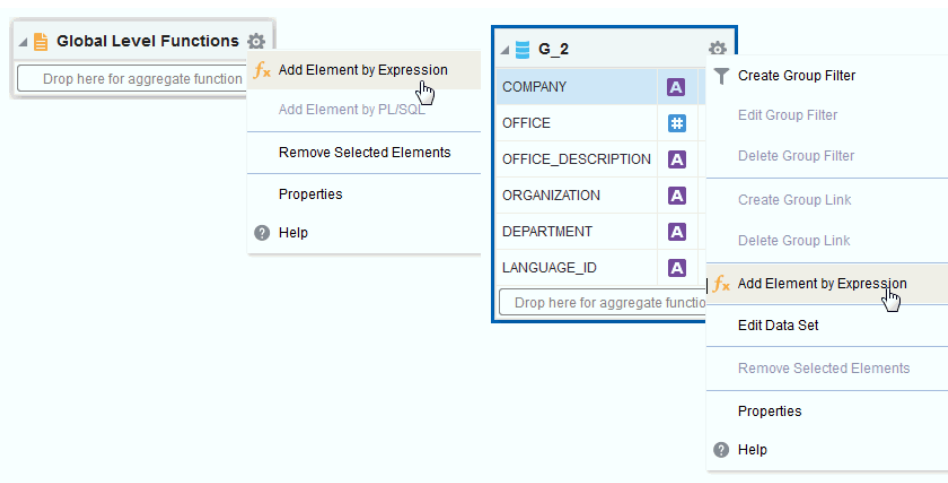


3. Click the function icon to the left of the new element name and choose the function from the list.
4. To change the default name, click the actions icon to the right of the element name and click **Properties** to launch the **Edit Properties** dialog. See [Setting Element Properties](#) for more about the properties available on this dialog.

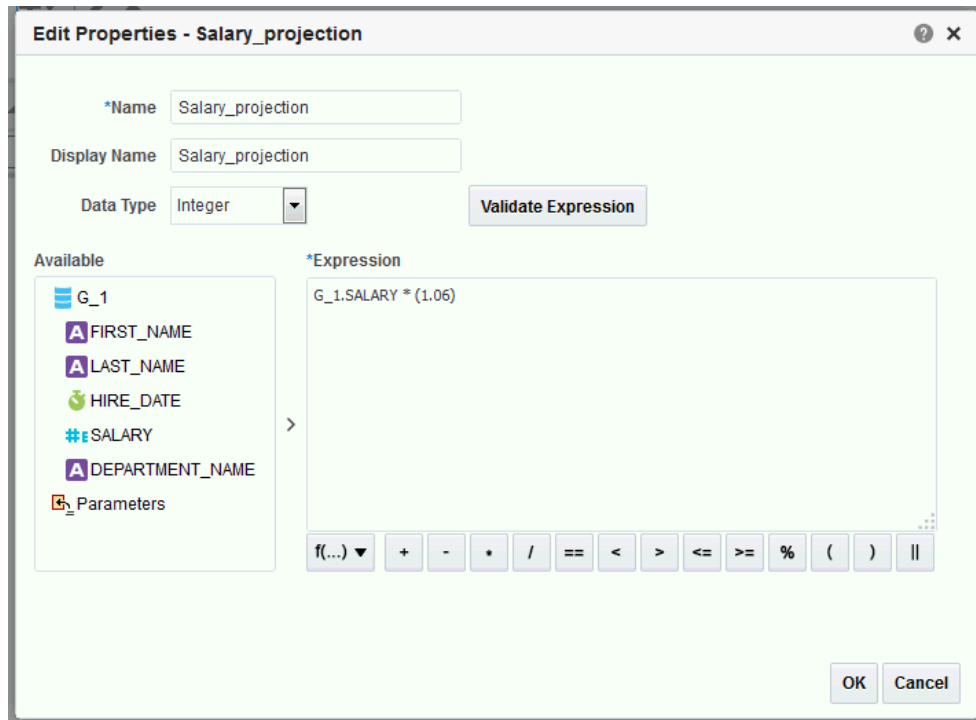
## Adding a Group-Level or Global-Level Element by Expression

You can add group-level or global-level aggregate functions by expressions.

1. To add a group-level element, on the **Group** object, click **Menu** and select **Add Element by Expression**.



2. In the **Add Element by Expression** dialog, enter fields and operators.

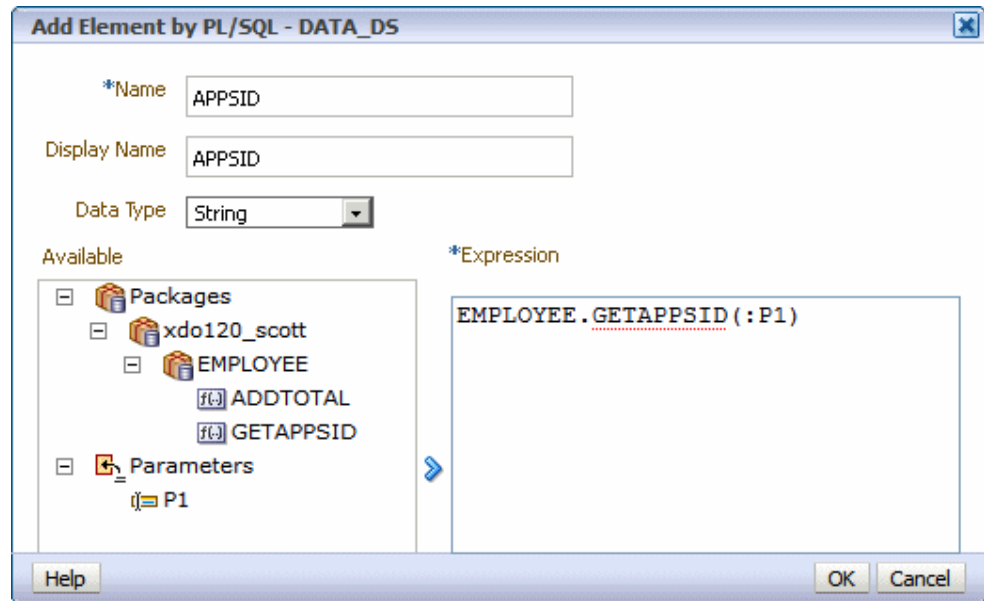


3. In the **Display Name** field, enter a name that is meaningful to your business users.
4. (Optional) Select a data type.
5. Use the shuttle arrow to move the data elements required for the expression from the **Available** box to the **Expression** box.
6. Click an operator to insert it in the **Expression** box, or choose from the function list.
7. Click **Validate Expression** to validate the expression.

## Adding a Global-Level Element by PL/SQL

The PL/SQL function must return a VARCHAR data type.

1. On the Properties page, specify the PL/SQL Package as the **Oracle DB Default Package** in the data model properties. .
2. On the **Global Level Functions** object, click **Menu**, and then click **Add Element by PL/SQL**.
3. In the **Add Element by PL/SQL** dialog, enter the following fields:
  - **Name** - Enter a meaningful name for the element.
  - **Display Name** - Enter a display name. This appears in the report design tools. Enter a name that is meaningful to your business users.
  - **Data Type** - Select **String**.



4. Select the PL/SQL package from the Available box and click the shuttle button to move the function to the Group Filter box.

## Using the Structure View to Edit Your Data Structure

The Structure view enables you to preview the structure of your data model.

The Data Source column displays the date elements in a hierarchical tree that you can collapse and expand. Use this view to verify the accuracy of the data model structure and to perform the following edits:

- [Renaming Elements](#)
- [Adding Value for Null Elements](#)

The Structure view is shown below.

Diagram   Structure   Data   Code						
Table View   Output						
Data Source	XML View			Business View		
	XML Tag Name	Sorting	Value If Null	Display Name	Data Type	
Report Data						
Data Structure	DATA_DS					
Employees	G_1					
DEPARTMENT_NAME	DEPARTMENT_NAME			DEPARTMENT_NAME	A	
Salary_projection	Salary_projection			Salary_projection	#	
Departments	G_2					
DEPARTMENT_NAME	DEPARTMENT_NAME			DEPARTMENT_NAME	A	
MANAGER_ID	MANAGER_ID			MANAGER_ID	#	
FIRST_NAME	FIRST_NAME			FIRST_NAME	A	
LAST_NAME	LAST_NAME			LAST_NAME	A	
PHONE_NUMBER	PHONE_NUMBER			PHONE_NUMBER	A	

## Renaming Elements

Use the Structure page to define user-friendly names for elements in the data model.

You can rename both the XML element tag name (XML View) and the name that displays in the report layout tools (Business View). The figure below shows renaming the Data Source elements to friendlier Business View names.

Data Source	XML View			Business View	
	XML Tag Name	Sorting	Value If Null	Display Name	Data Type
Report Data					
Data Structure	DATA_DS				
Employees	G_1				
DEPARTMENT_NAME	DEPARTMENT_NAME			Department	A
Salary_projection	Salary_projection			Salary Projection	#
Departments	G_2				
DEPARTMENT_NAME	DEPARTMENT_NAME			Department	A
MANAGER_ID	MANAGER_ID			Manager	#
FIRST_NAME	FIRST_NAME			First	A
LAST_NAME	LAST_NAME			Last	A
PHONE_NUMBER	PHONE_NUMBER			Phone	A

## Adding Value for Null Elements

The Structure also enables you to enter a value to use for an element if the data model returns a null value for the element.

- Enter the value to use in the **Value if Null** field for the element.

## Function Reference

The table below describes the usage of supported functions available from the Add Element by Expression dialog and the Edit Group Filter dialog.

Function	Description	Syntax	Example
IF	Logical IF operator Evaluates boolean_expr, and returns true_return if boolean_expr is true, and false_return if boolean_expr is false.	IF (boolean_expr, true_return, false_return)	IF (G_1.DEPARTMENT_ID == 10, 'PASSED', 'FAIL') returns 'PASSED' if DEPARTMENT_ID = 10, otherwise returns 'FAIL'

Function	Description	Syntax	Example
NOT	Logical NOT operator Evaluates <code>boolean_expr</code> , and returns true if <code>boolean_expr</code> is false.	<code>STRING(NOT(boolean_expr))</code>	<code>STRING(NOT(G_1.JOB_ID == 'MANAGER'))</code> returns 'TRUE' if <code>JOB_ID = MANAGER</code> , otherwise returns 'FALSE'
AND	Logical AND operator Evaluates <code>boolean_expr1</code> and <code>boolean_expr2</code> , and returns true if both boolean expressions are true, otherwise returns false.	<code>STRING(AND(boolean_expr1, boolean_expr2, ...))</code>	<code>STRING(AND(G_1.JOB_ID == 'MANAGER', G_1.DEPARTMENT_ID == 10))</code> returns 'TRUE' if both <code>JOB_ID = MANAGER</code> and <code>DEPARTMENT_ID = 10</code> , otherwise returns 'FALSE'
&&	Logical AND operator Evaluates <code>boolean_expr1</code> and <code>boolean_expr2</code> , and returns true if both boolean expressions are true, otherwise returns false.	<code>STRING(boolean_expr1 &amp;&amp; boolean_expr2)</code>	<code>STRING(G_1.JOB_ID == 'MANAGER' &amp;&amp; G_1.DEPARTMENT_ID == 10)</code>  returns 'TRUE' if both <code>JOB_ID = MANAGER</code> and <code>DEPARTMENT_ID = 10</code> , otherwise returns 'FALSE'
	Logical OR operator Evaluates <code>boolean_expr1</code> and <code>boolean_expr2</code> and returns true if both boolean expressions are true, otherwise returns false.	<code>STRING(OR(boolean_expr1, boolean_expr2))</code>	<code>STRING(OR(G_1.JOB_ID == 'MANAGER', G_1.DEPARTMENT_ID == 10))</code>  returns 'TRUE' if either <code>JOB_ID = MANAGER</code> or <code>DEPARTMENT_ID = 10</code> , otherwise returns 'FALSE'
MAX	Returns the maximum value of the element in the set.	<code>MAX(expr1, expr2, expr3, ...)</code>	<code>MAX(G1_Salary, 10000)</code> returns max of salary or 10000
MIN	Returns the minimum value of the element in the set.	<code>MIN(expr1, expr2, expr3, ...)</code>	<code>MIN(G1_Salary, 5000)</code> returns min of salary or 5000
ROUND	Returns a number rounded to the integer places right of the decimal point.	<code>ROUND(number[,integer])</code> If integer is omitted, number is rounded to 0 places. Integer can be negative to round off digits left of the decimal point. Integer must be an integer.	<code>ROUND(2.777)</code> returns 3 <code>ROUND(2.777, 2)</code> returns 2.78
FLOOR	Returns the smallest integer equal to or less than n.	<code>FLOOR(n)</code>	<code>FLOOR(2.777)</code> returns 2



Function	Description	Syntax	Example
CEILING	Returns the largest integer greater than or equal to n.	CEILING(n)	CEILING(2.777) returns 3
ABS	Returns the absolute value of n.	ABS(n)	ABS(-3) returns 3
AVG	Returns the average value of the expression.	AVG(expr1, expr2, expr3, ...)	AVG(G_1.SALARY,G_1.COMMISSION_PCT*G_1.SALARY) returns the average of SALARY and COMMISSION For example, if SALARY = 14000 and COMMISSION_PCT = .4, the expression evaluates to 9800.0
LENGTH	Returns the length of an array. The LENGTH function calculates the length using characters as defined by the input character set. If char is null, the function returns null. If char is an array, it returns the length of the array.	LENGTH(expr)	Example to return the length of an array: LENGTH{1, 2, 4, 4}) returns 4 Example to return the length of a string: LENGTH('countries') returns 9
SUM	Returns the sum of the value of the expression.	SUM(expr1, expr2, ...)	SUM (G_1.SALARY, G_1.COMMISSION_PCT*G_1.SALARY) returns sum of salary and commission For example, if SALARY = 14000 and COMMISSION_PCT =.4, the expression evaluates to 19,600.0
NVL	Replaces null (returned as a blank) with a string in the results of a query.	NVL(expr1, expr2) If expr1 is null, then NVL returns expr2. If expr1 is not null, then NVL returns expr1.	NVL(G_1.COMMISSION_PCT, .3) returns .3 when G_1.COMMISSION_PCT is null
CONCAT	Returns char1 concatenated with char2.	CONCAT(char1, char2)	CONCAT(CONCAT(First_Name, ' '), Last_Name) where First_Name = Joe and Last_Name = Smith returns Joe Smith
STRING	Returns char as a string data type.	STRING(expr)	STRING(G1_SALARY) where salary = 4400 returns 4400 as a string

Function	Description	Syntax	Example
SUBSTRING	Extracts a substring from a string.	SUBSTRING(string, start_pos, end_pos) string is the source string. start_pos is the position to start the extraction. end_pos is the end position of the string to extract (optional).	SUBSTRING('this is a test', 5, 7) returns "is" (that is, characters 6 through 7) SUBSTRING('this is a test', 5) returns "is a test"
INSTR	Returns the position/location of the first character of a substring in a string.	INSTR(string1, string2) string1 is the string to search. string2 is the substring to search for in string1.	INSTR('this is a test', 'is a') returns 5
DATE	Converts a valid Java date string to a date data type in canonical format.	DATE(char, format_string) where (1) char is any valid Java date string (for example, 13-JAN-2013)(2) format_string is the Java date format of the input string (for example, dd-MMM-yyyy) The input and format strings must be a valid Java date format string.	DATE('01-Jan-2013','dd-MMM-yyyy') returns 2013-01-01T08:00:00.000+00:00
FORMAT_DATE	Converts a date argument in the Java date format to a formatted string.	FORMAT_DATE(date, format_string)	FORMAT_DATE(SYSDATE, 'dd-MMM-yyyy') where the value of SYSDATE = 2013-01-24T16:32:45.000-08:00 returns 24-Jan-2013
FORMAT_NUMBER	Converts a number or numeric string to a string in the specified number format.	FORMAT_NUMBER(number, format_string)	FORMAT_NUMBER(SOME_NUMBER, '\$9,999.00') where the value of SOME_NUMBER = 12345.678 returns \$12,345.68

---

Function	Description	Syntax	Example
DECODE	Replaces the value of an expression with another value based on the specified search and replace criteria.	DECODE(expr, search, result [, search, result]... [, default])	DECODE(PROD_FAMILY_CODE,100,'Colas', 200,'Root Beer', 300,'Cream Sodas', 400,'Fruit Sodas','Other')returns(1) 'Colas' if PROD_FAMILY_CODE = 100(2) 'Root Beer' if PROD_FAMILY_CODE = 200(3) 'Cream Sodas' if PROD_FAMILY_CODE = 300(4) 'Fruit Sodas' if PROD_FAMILY_CODE = 400(5) 'Other' if PROD_FAMILY_CODE is any other value
REPLACE	Replaces a sequence of characters in a string with another set of characters.	REPLACE(expr,string1, string2) where string1 is the string to search for and string2 is the string to replace.	REPLACE(G_1.FIRST_NAME,'B','L') where G_1.FIRST_NAME = Barry returns Larry

---

# 4

## Adding Parameters and Lists of Values

This topic describes how to add parameters and lists of values to a data model.

### Topics:

- [About Parameters](#)
- [Adding a New Parameter](#)
- [About Lists of Values](#)
- [Adding Lists of Values](#)
- [Adding Flexfield Parameters](#)

### About Parameters

Adding parameters to a data model enables users to interact with data when they view reports.

Oracle BI Publisher supports the following parameter types:

- *Text* - Enables entering a text string to pass as the parameter.
- *Menu* - Enables making selections from a list of values. A list of values can contain fixed data that you specify or a list created using a SQL query that is executed against any of the defined data sources. This option supports multiple selection, a **Select All** option, and partial page refresh for cascading parameters.

To create a menu type parameter, define the list of values, and then define the parameter and associate it to the list of values. See [Adding Lists of Values](#).

- *Date* - Enables the user to select a date as a parameter. You must use the data type *Date* and the Java date format.

After defining the parameters in the data model, you can configure how the parameters are displayed in the report as a report-level setting. See *Configuring Parameter Settings for the Report* in *Report Designer's Guide for Oracle Business Intelligence Publisher*.

Support for parameters varies based on the data set type. SQL Query data sets support the full set of available parameter features. Other types of data sets might support all, none, or a subset of these features. The table below summarizes the support for each data set type.

Data Set Type	Parameter Support	Multiple Selection	Can Select All	Refresh Other Parameters on Change
SQL Query	Yes	Yes	Yes	Yes
MDX Query	No	No	No	No

Data Set Type	Parameter Support	Multiple Selection	Can Select All	Refresh Other Parameters on Change
Oracle BI Analysis	Inherited from Oracle BI Analysis	Yes (using Oracle BI Dashboards)	Yes (using Oracle BI Dashboards)	Yes (using Oracle BI Dashboards)
View Object	Yes, provided that the view object supports and is designed for it	No	No	Yes (view object parameters only)
Web Service	Yes	No	No	No
LDAP Query	Yes	No	No	No
XML File	No	No	No	No
Microsoft Excel File	Yes	No	No	No
CSV File	No	No	No	No
HTTP (XML Feed)	Yes	No	No	No

## Adding a New Parameter

Create a new parameter by assigning it a name and other properties.

The parameter name you choose must not exceed the maximum length allowed for an identifier by your database. Refer to your database documentation for identifier length limitations.

Supported types are:

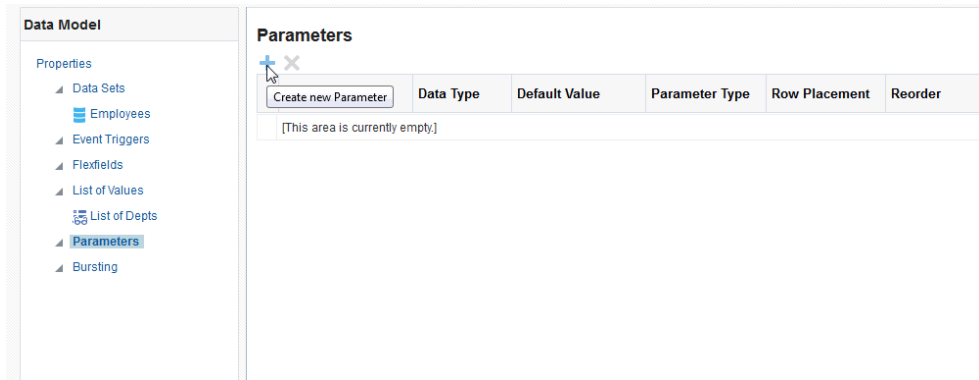
- **Text** - Allows the user to enter a text entry to pass as the parameter. See [Creating a Text Parameter](#).
- **Menu** - Presents a list of values to the user. See [Creating a Menu Parameter](#).
- **Date** - Passes a date parameter. The **Data Type** must also be Date. See [Defining a Date Parameter](#).

Default parameter values are also used to preview the report output when you design report layouts using Oracle BI Publisher Layout Editor.

BI Publisher supports parameters that are of type text entry or menu (list of values) but not both. You cannot define a combination parameter that enables a user to enter a text value or choose from a menu list of values.

You can configure row placement at the report level. The report definition supports additional display options for parameters. See Configuring Parameter Settings for the Report in *Report Designer's Guide for Oracle Business Intelligence Publisher*.

1. On the Data Model components pane, click **Parameters** and then click **Create new Parameter**.



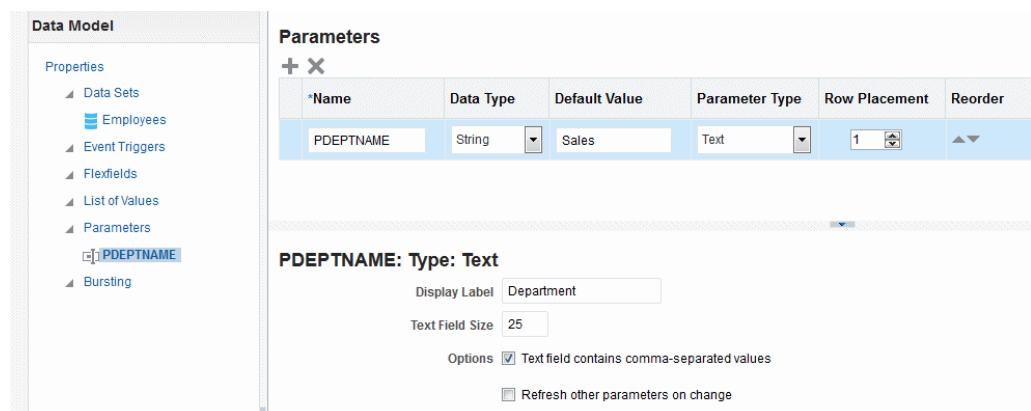
2. Enter a **Name** for the parameter.  
The name must match any references to this parameter in the data set.
3. Select the **Data Type** from the list. A **Date** data type only supports a **Date Parameter Type**.
4. Enter a **Default Value** for the parameter. This is recommended to prevent long running queries.
5. Select the **Parameter Type**.
6. In the **Row Placement** setting configure the number of rows for displaying the parameters and in which row to place each parameter.

For example, if your report has six parameters, you can assign each parameter to a separate row, 1 - 6, with one being the top row; or, you can assign two parameters each to rows 1, 2, 3. By default, all parameters are assigned to row 1.

## Creating a Text Parameter

The **Text** type parameter provides a text box to prompt the user to enter a text entry to pass as the parameter to the data source.

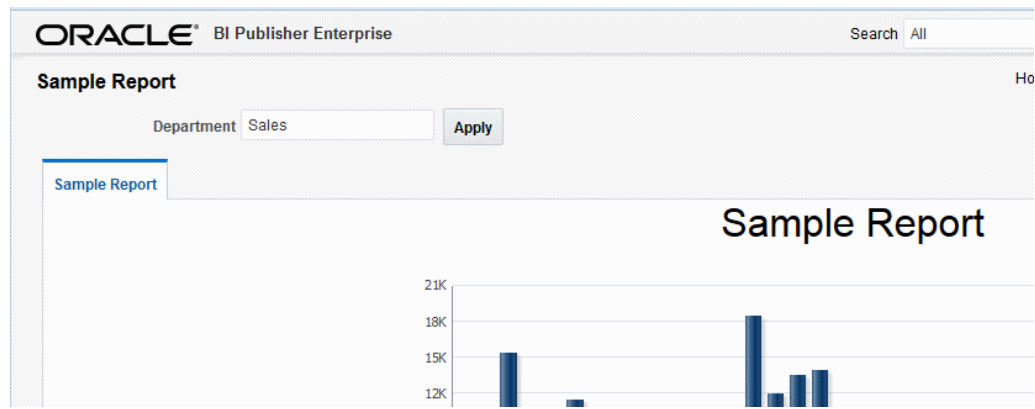
The figure below shows a text parameter definition.



1. Select Text from the Parameter Type list.
2. Enter the **Display Label**. For example, Department.

3. Enter the **Text Field Size** as an integer. This field determines the size (width) of the field, but does not limit the number of characters that the user can enter into the text box.
4. Enable the following **Options** if required:
  - **Text field contains comma-separated values** - Enables the user to enter multiple comma-delimited values for this parameter. The parameter in your data source must be defined to support multiple values.
  - **Refresh other parameters on change** - Performs a partial page refresh to refresh any other parameters whose values are dependent on the value of this one.

The figure below shows how the Department parameter displays to the report consumer.



## Creating a Menu Parameter

A Menu type parameter presents a list of values to the user.

You must define the list of values first. See [Adding Lists of Values](#). The Menu type parameter supports the data types of String and Integer only.

The Menu parameter definition includes various options, as shown in the figure below.

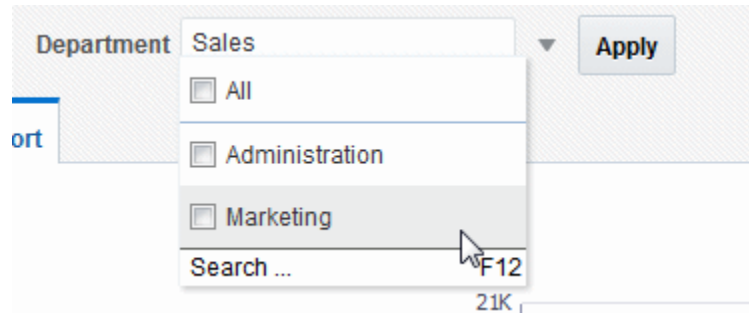
The screenshot shows the 'Parameters' configuration window in Oracle BI Publisher. On the left, there is a 'Data Model' tree with 'Properties' expanded, showing a list of data sets including 'Employees', 'Event Triggers', 'Flexfields', 'List of Values', 'List of Depts', 'Parameters', 'PDEPTNAME', and 'Bursting'. The 'Parameters' section is active, showing a table with the following details:

*Name	Data Type	Default Value	Parameter Type	Row Placement	Reorder
PDEPTNAME	String	Sales	Menu	1	

Below the table, the configuration for the 'PDEPTNAME: Type: Menu' parameter is shown:

- Display Label: Department
- List of Values: List of Depts
- Number of Values to Display in List: 100
- Options:
  - Multiple Selection
  - Can select all
  - NULL Value Passed
  - All Values Passed
  - Refresh other parameters on change

1. Select **Menu** from the **Parameter Type** list. The lower pane displays the appropriate fields.
2. In **Data Type**, select *String* or *Integer*.
3. Enter the **Display Label**. The display label is the label that displays to users when they view the report. For example: Department.
4. Enter the **Number of Values to Display in List**. If the number of values in the list exceeds the entry in this field, the user must click **Search** to find a value not displayed, as shown in the figure below. This field defaults to 100.



5. Select the **List of Values** that you defined for this parameter.
6. Enable the following **Options** if required:
  - **Multiple Selection** - Allows the user to select multiple entries from the list. Your data source must be able to support multiple values for the parameter. The display of a menu parameter that supports multiple selection differs. See the two figures below.
  - **Can select all** - Inserts an *All* option in the list.

When the user selects *All* from the list of values, you have the option of passing a null value for the parameter or all list values. Choose **NULL Value Passed** or **All Values Passed**.

 **Note:**

Using \* passes a null, so you must handle the null in your data source. A method to handle the null would be the standard Oracle NVL command, for example:

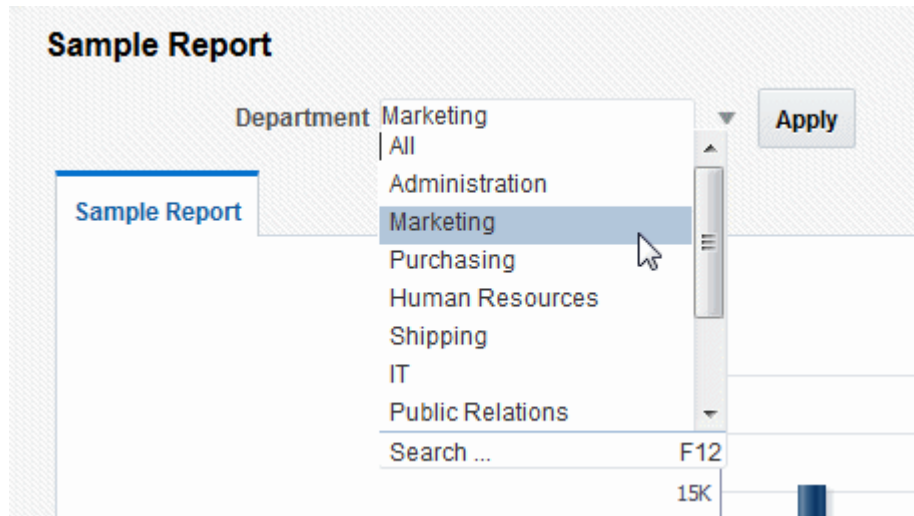
```
where customer_id = nvl(:cstid, customer_id)
```

where *cstid* is a value passed from the list of values, and when the user selects *All* it passes a null value.

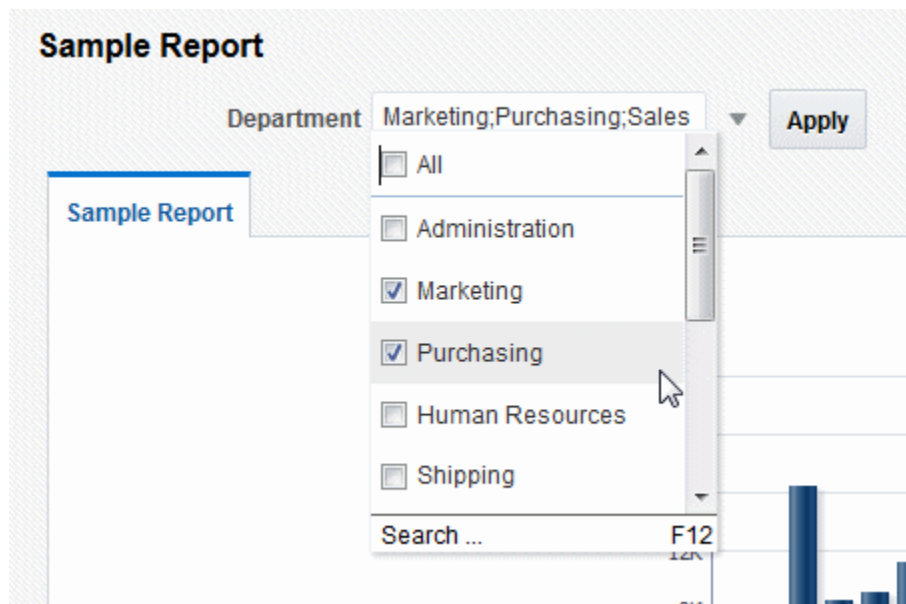
- **Refresh other parameters on change** — Performs a partial page refresh to refresh any other parameters whose values are dependent on the value of this one.

The figure below shows how the Department menu type parameter displays to the report consumer when multiple selection is not enabled.





The figure below shows how the Department menu type parameter displays to the report consumer when multiple selection is enabled.



## Customizing the Display of Menu Parameters

The display of menu parameters in the report can be further customized in the report definition.

Menu type parameters support the additional display option as a static list of checkboxes or radio buttons. See *Configuring Parameter Settings for the Report in Report Designer's Guide for Oracle Business Intelligence Publisher*.

## Defining a Date Parameter

The Date type parameter provides a date picker to prompt the user to enter a date to pass as the parameter to the data source.

The figure shows the date parameter definition.

The screenshot shows the 'Parameters' configuration window. On the left is a 'Data Model' tree with 'Properties' expanded, showing a list of data sets including 'H\_DATE'. The main area is titled 'Parameters' and contains a table with the following data:

Name	Data Type	Default Value	Parameter Type	Row Placement	Reorder
H_DATE	Date		Date	1	

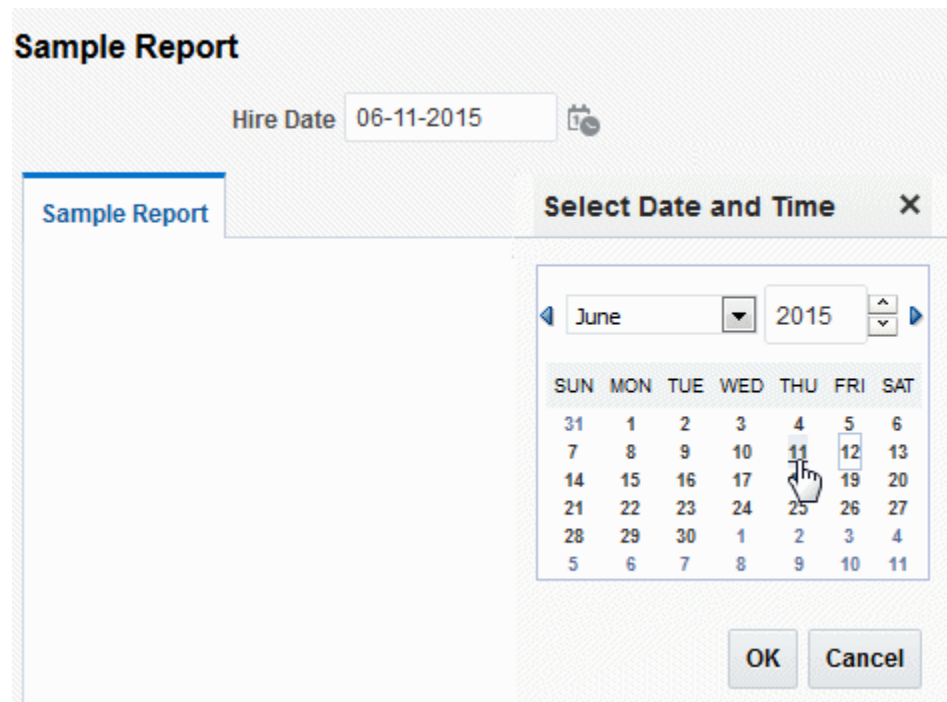
Below the table, the configuration for the selected parameter 'H\_DATE' is shown:

**H\_DATE: Type: Date**

- Display Label:
- Text Field Size:
- Date Format String:  (must be Java date format, e.g., MM-dd-yyyy)
- Date From:
- Date To:

1. Select Date from the Parameter Type list. The lower pane displays the appropriate fields for your selection.
2. Enter the **Display Label**. The display label is the label that displays to users when they view the report. For example: Hire Date.
3. Enter the **Text Field Size** as an integer. This field determines the number of characters that the user can enter into the text box for the date entry. For example: 10.
4. Enter the **Date Format String**. The format must be a Java date format, for example, *MM-dd-yyyy*.
5. Optionally, enter a **Date From** and **Date To**. The dates entered here define the date range that are presented to the user by the date picker. For example if you enter the **Date From** as 01-01-1990, the date picker does not allow the user to select a date before 01-01-1990. Leave the **Date To** blank to enable all future dates.

The figure shows how the Hire Date parameter displays to the report consumer.



## About Lists of Values

A list of values is a defined set of values that a report consumer can select from to pass a parameter value to your data source.

If you define a menu type parameter, the list of values provides the menu of choices. You must define the list of values before you define the menu parameter.

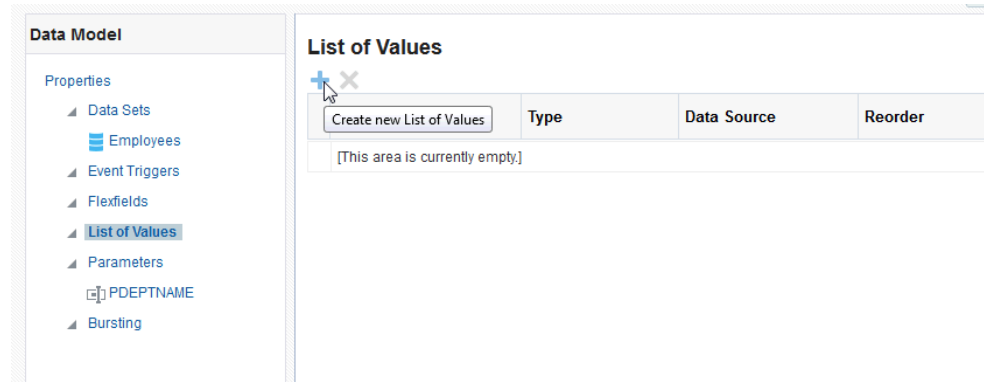
Populate the list using one of the following methods:

- *Fixed Data*  
Manually enter the list of values.
- *SQL Query*  
Retrieves the values from a database using a SQL query.
- *Flexfield*  
Retrieves the values from a key flexfield defined in Oracle E-Business Suite. The flexfield option is only available when Oracle BI Publisher is using the Oracle E-Business Suite security model, see [Adding Flexfield Parameters](#).

## Adding Lists of Values

You can create lists of SQL Query or Fixed Data values .

1. In the Data Model components pane, click **List of Values** and then click **Create new List of Values**.



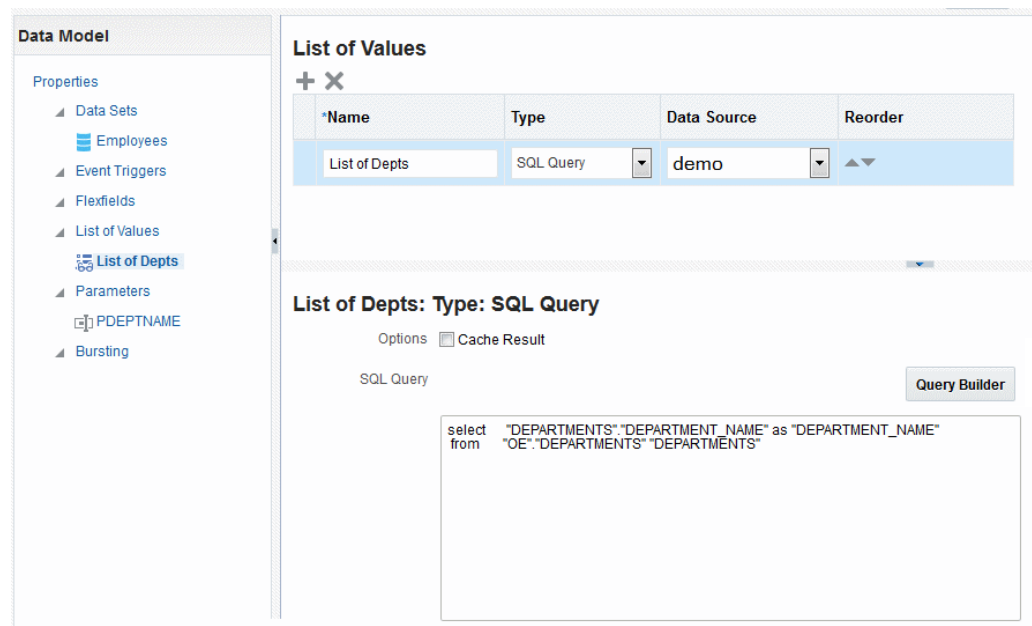
2. Enter a Name for the list and select a **Type**.

## Creating a List from a SQL Query

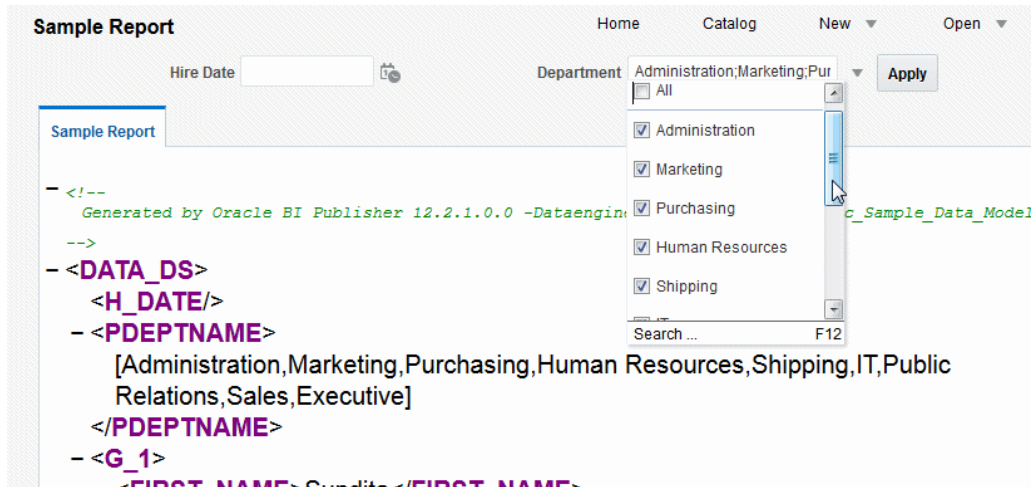
The data engine expects a (display) name-value pair from the list of values query. In the list of values select statement, the column listed first is used as the display name and the second is used for the value that is passed to the parameter in the data set query by the data engine.

If the query returns only one column, then the same column value is used both as the list of values display name shown to the user and as the value that is passed to the parameter.

1. Select a **Data Source** from the list.
2. In the lower pane, select **Cache Result** (recommended) if you want the results of the query cached for the report session.
3. Enter the SQL query or use the Query Builder. See [Using the SQL Query Builder](#) for information on the Query Builder utility. The figure below shows a SQL query type list of values.



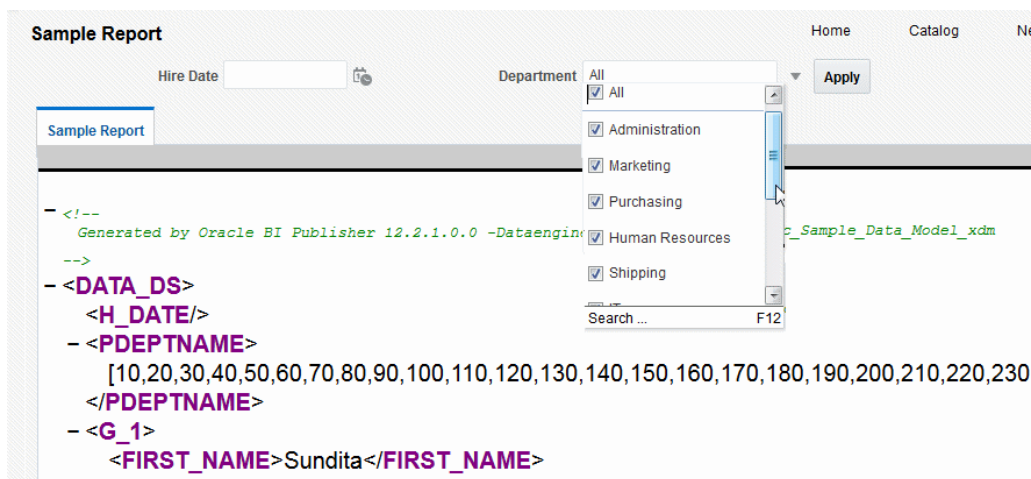
The SQL query shown below selects only the DEPARTMENT\_NAME column from the DEPARTMENTS table. In this case the list of values both displays the results of the query in the list and passes the same value to the parameter in the data set. The figure below shows the list of values display entries and the values passed to the data set. The menu items and the values shown for P\_DEPT are the DEPARTMENT\_NAME values.



If instead you wanted to pass the DEPARTMENT\_ID to the parameter in the data set, and display the DEPARTMENT\_NAME in the list, construct your SQL query as follows:

```
Select      "DEPARTMENTS"."DEPARTMENT_NAME" as "DEPARTMENT_NAME",
           "DEPARTMENTS"."DEPARTMENT_ID" as "DEPARTMENT_ID"
from        "DEMO"."DEPARTMENTS" "DEPARTMENTS"
```

The figure below shows the list of values display entries and the values passed to the data set. The menu lists the DEPARTMENT\_NAME while the values shown for P\_DEPT are the DEPARTMENT\_ID values.



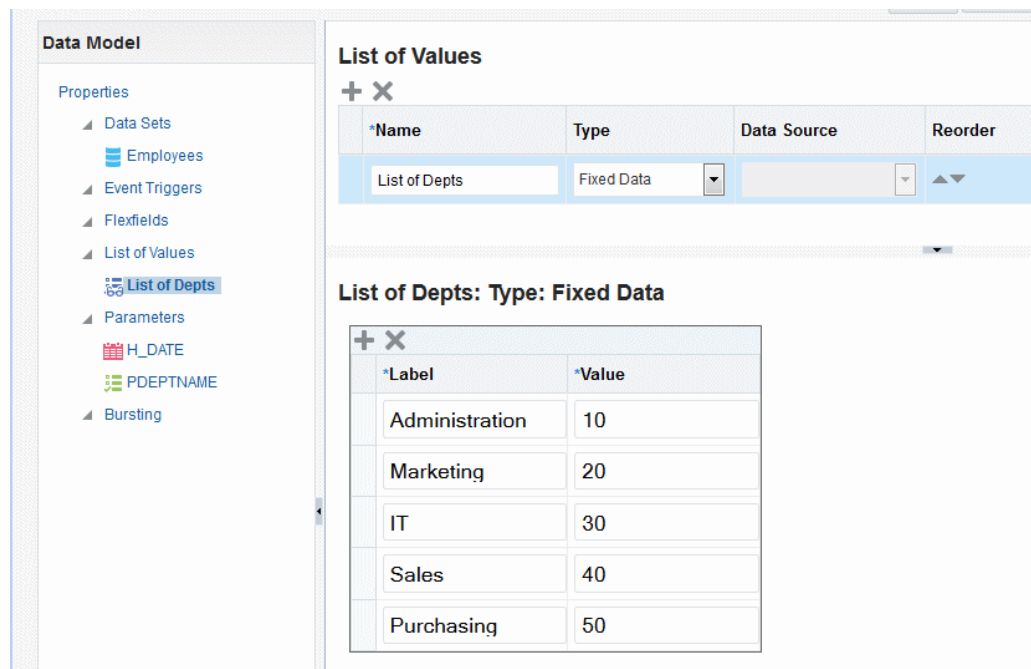
## Creating a List from a Fixed Data Set

Create a list from a fixed data set for each label-value pair required.

When you create a label-value pair, the label is displayed to the user in the list. The value is passed to the data engine.

1. In the lower pane, click the **Create new List of Values** icon to add a Label and Value pair.
2. Repeat for each label-value pair required.

The figure below shows fixed data type list of values.

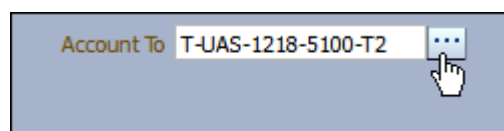


## Adding Flexfield Parameters

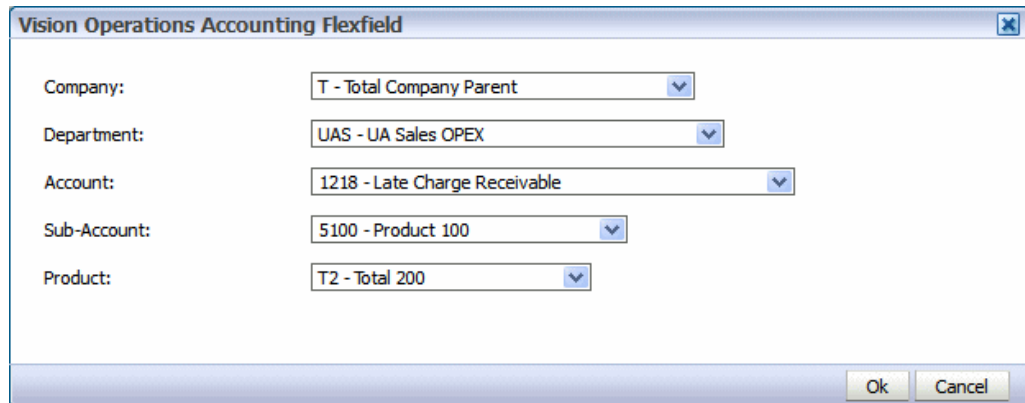
Oracle E-Business Suite customers who have configured BI Publisher to use E-Business Suite security can create reports that leverage key flexfields as parameters.

When you define a data model to pass a key flexfield as a parameter, BI Publisher presents a dialog to the report consumer to make selections for the flexfield segments to pass as parameters to the report, similar to the way flexfields are presented when running reports through the concurrent manager in the E-Business Suite.

The flexfield list of values displays in the report viewer as shown below.



The flexfield list of values displays as a dialog from which you select the segment values, as shown below.



## Prerequisites for Using Flexfields

When defining a list of values, E-Business Suite customers see a list Type called "Flexfield".

To enable the flexfield type list of values, BI Publisher must be configured to use E-Business Suite Security. The flexfield must already be defined in the E-Business Suite.

See *Oracle E-Business Suite Flexfields Guide* and *Integrating with Oracle E-Business Suite in Administrator's Guide for Oracle Business Intelligence Publisher*.

## Adding a Flexfield Parameter and List of Values

Add flexfield parameters by adding the list of values.

The flexfield type list of values retrieves the flexfield metadata definition to present the appropriate values for each segment in the flexfield list of values selection dialog. Use the flexfield parameter to pass values to the Flexfield defined in the Data Model.

At runtime the `&flexfield_name` reference is replaced with the lexical code constructed based on the values in the Flexfield component definition.

1. Add the flexfield list of values (LOV).
2. Add a parameter and associate it with the flexfield LOV by selecting your flexfield list of values as the source menu for the parameter.
3. Add the Flexfield component to the data model.
4. Reference the Flexfield in your SQL query using the `&flexfield_name` syntax.

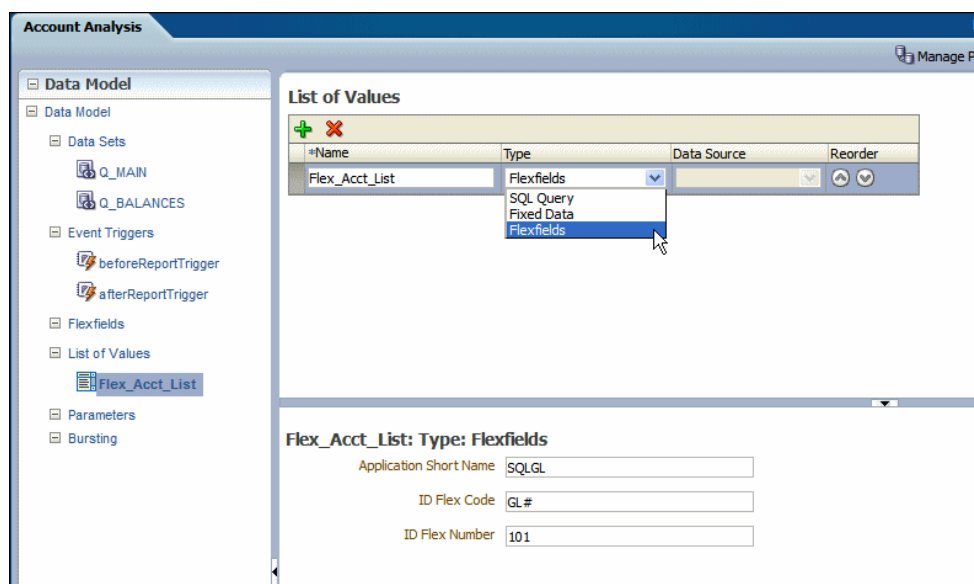
## Adding the Flexfield List of Values

Add a list of values retrieved from a flexfield definition.

When you choose Flexfields as the Type, the **Data Source** option is no longer editable. All flexfields type lists of values use the Oracle E-Business Suite as the data source.

1. On the Data Model components pane, click **List of Values** and then click **Create new List of Values**.
2. Enter a **Name** for the list and choose Flexfields as the **Type**.
3. In the Flex\_Acct\_List:Flexfields pane, enter the following:
  - **Application Short Name** - E-Business Suite application short name, for example: SQLGL.
  - **ID Flex Code** - Flexfield code defined for this flexfield in the Register Key Flexfield form, for example: GL#.
  - **ID Flex Number** - Name of the source column or parameter that contains the flexfield structure information, for example: 101 or :STRUCT\_NUM. If you use a parameter, ensure that you define the parameter in the data model.

The image shows a sample flexfield type, *LOV*.



## Adding the Menu Parameter for the Flexfield List of Values

Define the parameter to display the flexfield list of values and capture the values selected by the user.

The Flexfield type parameter definition includes an additional field called **Range** to support range flexfields. A range flexfield supports low and high values for each key segment rather than just single values. You can customize the default value of the flexfield and row placement in the report definition. The row placement determines where this parameter appears in the report viewer. To pass a range of flexfield segment values, see [Passing a Range of Values](#)

The following options are disabled for flexfield parameters: **Number of Values to Display in List**, **Multiple Selection**, **Can select all**, and **Refresh other parameters on change**.

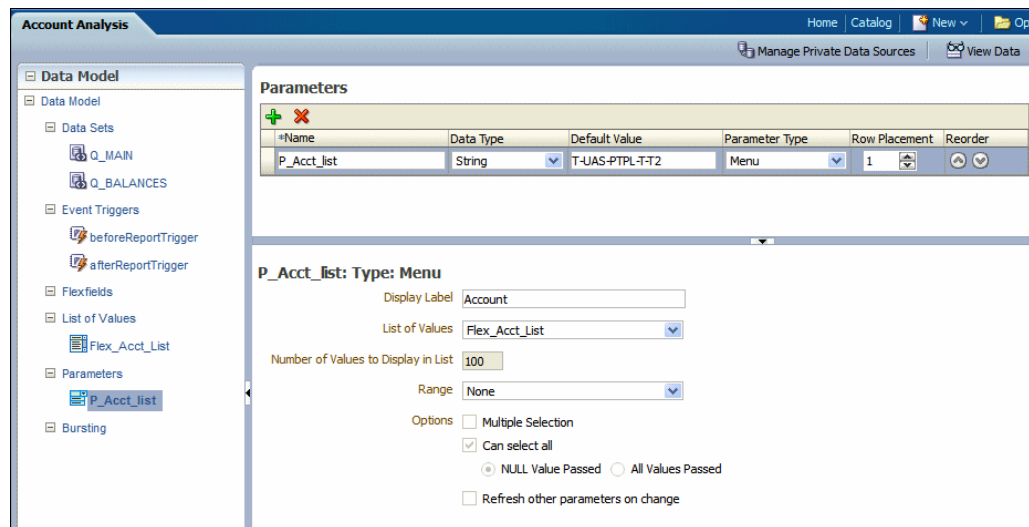
1. On the Data Model components pane, click **Parameters** and then click **Create new Parameter**.
2. Select **Menu** from the **Parameter Type** list.



3. Choose *String* or *Integer* as the **Data Type**.
4. Enter a **Default Value** for the flexfield parameter.
5. Enter the **Row Placement**.
6. Enter the **Display Label**. The display label is the label that displays to users when they view the report. For example: *Account From*.
7. Select the **List of Values** that you defined for this parameter.

When you select a list of values that is the Flexfield type, an additional field labeled **Range** displays.

The image shows a parameter definition for the flexfield list of values.



## Using the Flexfield Parameter to Pass Values to a Flexfield Defined in the Data Model

After adding the Menu parameter to the flexfield list of values, you can pass the parameter values to a flexfield component in the data model.

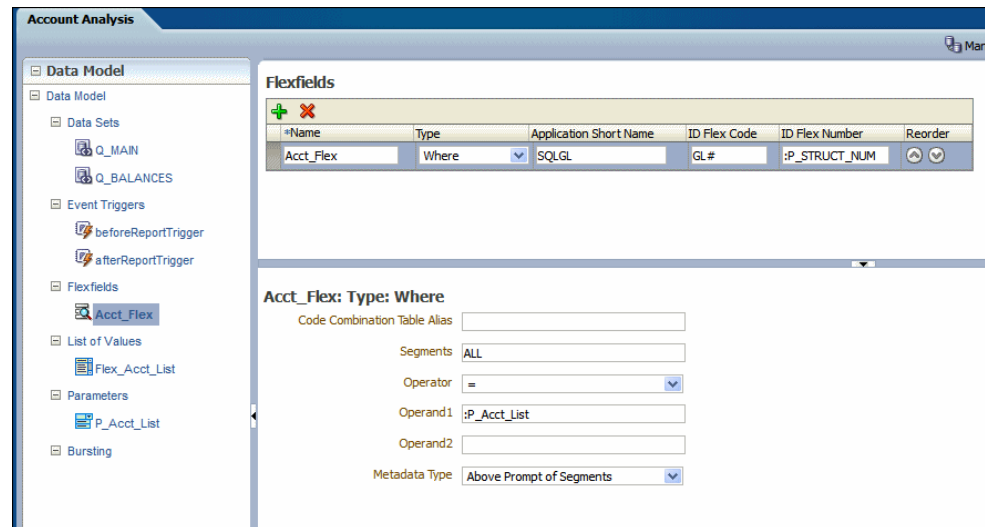
[Adding Flexfields](#) covers adding a flexfield component in detail. The simplified procedure is provided here to complete the example.

To define the Flexfield in the data model:

1. On the Data Model components pane, click **Flexfields** and then click **Create new Flexfield**.
2. Enter the following:
  - **Name** — Enter a name for the flexfield component.
  - **Type** — Select the flexfield type from the list. The type you select here determines the additional fields required. See [Entering Flexfield Details](#).
  - **Application Short Name** — Enter the short name of the Oracle Application that owns this flexfield (for example, GL).
  - **ID Flex Code** — Enter the flexfield code defined for this flexfield in the Register Key Flexfield form (for example, GL#).

- **ID Flex Number** — Enter the name of the source column or parameter that contains the flexfield structure information. For example: 101. To use a parameter, prefix the parameter name with a colon, for example, :PARAM\_STRUCT\_NUM.
3. In the lower region of the page, enter the details for the type of flexfield you selected. For the field that is to take the parameter value, enter the parameter name prefixed with a colon, for example, :P\_Acct\_List.

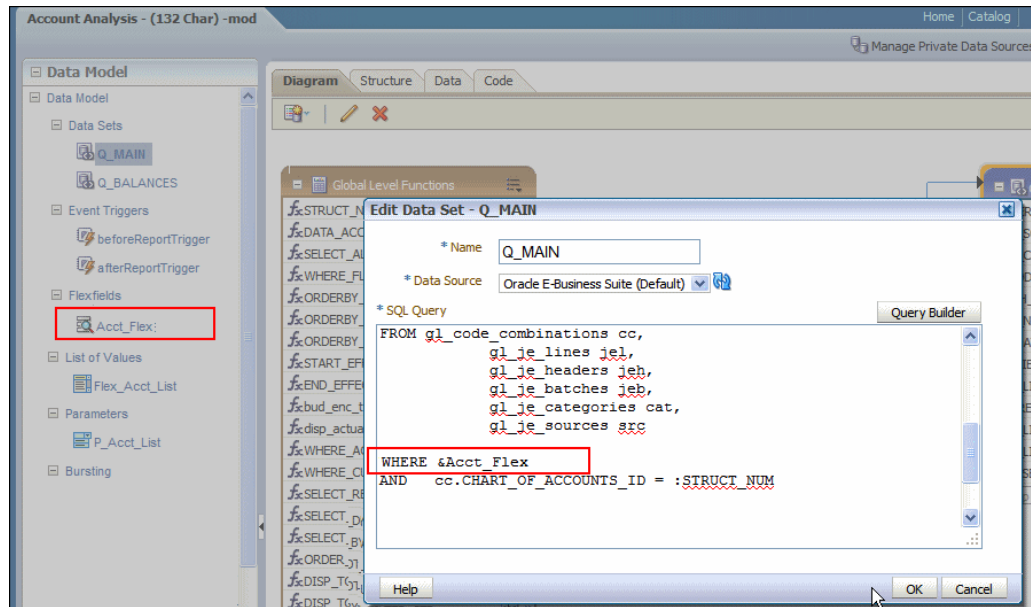
In the figure below the Flexfield component is defined as a "Where" **Type**. The parameter :P\_Acct\_List is entered in the Operand1 field. At runtime, values selected by the user for the parameter P\_Acct\_List will be used to create the where clause.



## Referencing the Flexfield in the SQL Query

Finally, create the SQL query against the E-Business Suite database.

Use the lexical syntax in the query as described in [Adding Key Flexfields](#). In the figure below &Acct\_Flex is the Flexfield lexical called in the where condition of the SQL query.

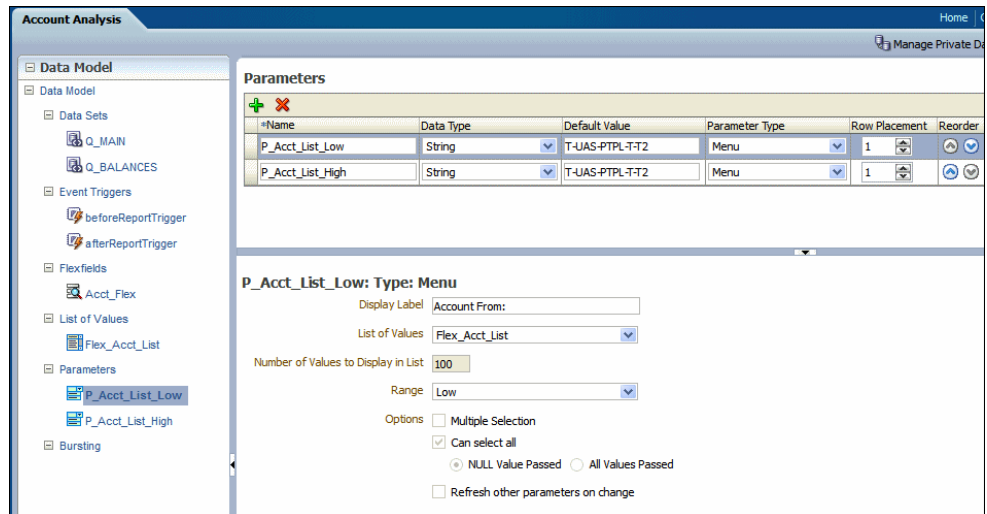


## Passing a Range of Values

To define the parameters for the flexfield lists of values when you want to pass a range of values you create two menu parameters that both reference the same flexfield LOV.

At runtime users choose a high value from the list of values and a low value from the same list of values. These two values are then passed as operands to the flexfield component of the data model.

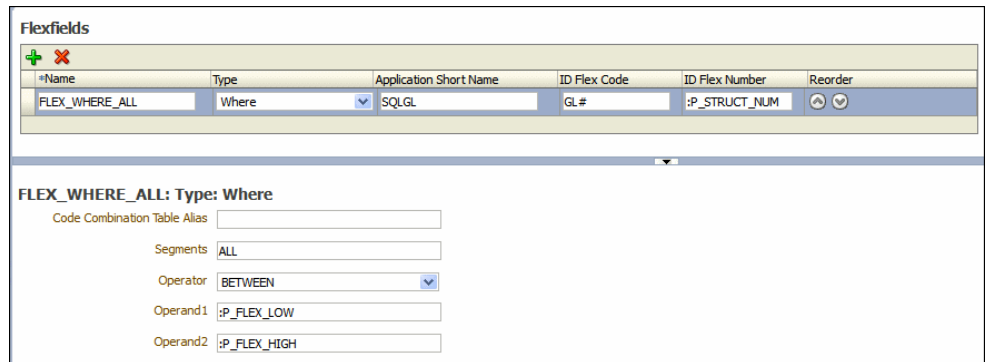
1. Create one flexfield LOV as described in [Adding the Flexfield List of Values](#).
2. Create the high range parameter following the steps in [Adding the Menu Parameter for the Flexfield List of Values](#). For the **Range** field, select **High** to designate this parameter as the high value.
3. Create the low range parameter following the steps in [Adding the Menu Parameter for the Flexfield List of Values](#). For the **Range** field, select **Low** to designate this parameter as the low value. Both parameters reference the flexfield list of values that you created in Step 1. The figure below shows creating the parameters to define the range.



4. Create the Flexfield in the data model, as described in [Using the Flexfield Parameter to Pass Values to a Flexfield Defined in the Data Model](#).

In the lower region of the page, enter the details for the type of flexfield you selected. Enter the parameter prefixed with a colon for example, :P\_Acct\_List.

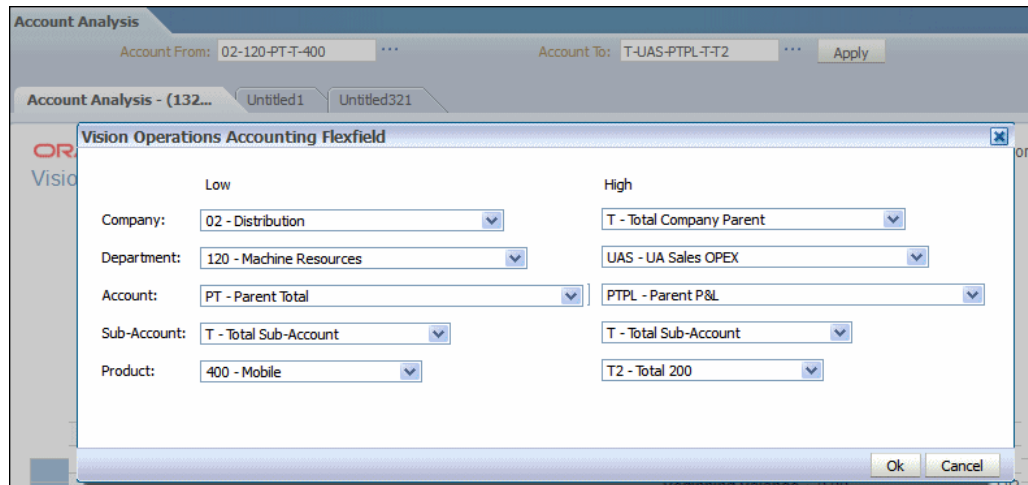
In the figure below the Flexfield component is defined as a "Where" **Type**. The parameters :P\_FLEX\_LOW and :P\_FLEX\_HIGH are entered in the Operand1 and Operand2 fields. At runtime, values selected by the user for the parameters P\_FLEX\_LOW and P\_FLEX\_HIGH will be used to create the where clause.



When the report associated with this data model is displayed in the report viewer, the report consumer sees the two flexfield parameters as shown below.



When the report consumer clicks either the high or low flexfield indicator (...), a dialog launches enabling input of both the high and low values as shown below.



The display characteristics in the report viewer of the range flexfield parameter resemble closely the presentation of range flexfields in the E-Business Suite.

# 5

## Adding Event Triggers

This topic describes how to define triggers in your data model. Data models support before data and after data event triggers and schedule triggers.

### Topics:

- [About Triggers](#)
- [Adding Before Data and After Data Triggers](#)
- [Creating Schedule Triggers](#)

## About Triggers

An event trigger checks for an event and when the event occurs, it runs the code associated with the trigger.

Oracle BI Publisher supports the following events: before data set is executed, after data set is executed and before a scheduled job is about to execute. There are three types of event triggers:

- **Before Data** - Fires right before the data set is executed.
- **After Data** - Fires right after the data engine executes all data sets and generates the XML.
- **Schedule Trigger** - Fires when a scheduled job is triggered and before it runs.

Before data and after data triggers execute a PL/SQL function stored in a PL/SQL package in your Oracle Database. The return data type for a PL/SQL function inside the package must be a Boolean type and the function must explicitly return TRUE or FALSE.

A schedule trigger is associated with a scheduled job. It is a SQL query that executes at the time a report job is scheduled to run. If the SQL returns any data, the report job runs. If the SQL query returns no data, the job instance is skipped.

Event triggers are not used to populate data used by the bursting definition. See [Adding Bursting Definitions](#).

## Adding Before Data and After Data Triggers

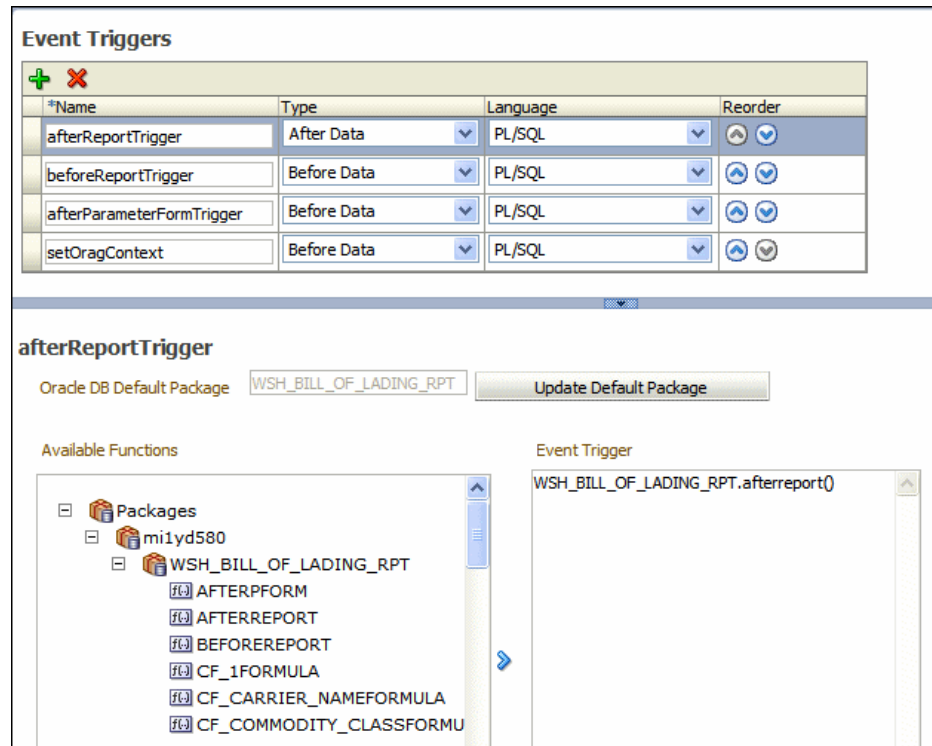
You can add event triggers that fire before and after data.

If you define a default package then you must define all parameters as a global PL/SQL variable in the PL/SQL package. You can then explicitly pass parameters to your PL/SQL function trigger or all parameters are available as a global PL/SQL variable, see [Setting Data Model Properties](#)

1. On the data model **Properties** pane, enter the **Oracle DB Default Package** that contains the PL/SQL function signature to execute when the trigger fires. .

2. From the task pane, click **Event Triggers**.
3. From the **Event Triggers** pane, click **Create New Event Trigger**.
4. Enter the following for the trigger:
  - **Name** - Name the trigger something meaningful.
  - **Type** - Select Before Data or After Data.
  - **Language** - Select PL/SQL.

The figure below shows an event trigger.



5. Select the package from the **Available Functions** box and click the arrow to move a function to the **Event Trigger** box.

The name appears as PL/SQL <package name>.<function name>.

## Order of Execution

If you define multiple triggers of the same type, they fire in the order that they appear in the table (from top to bottom).

To change the order of execution:

- Use the **Reorder** arrows to place the triggers in the correct order.

## Creating Schedule Triggers

A schedule trigger fires when a report job is scheduled to run. Schedule triggers are of type SQL Query.

When a report job is scheduled to run, the schedule trigger executes the SQL statement defined for the trigger. If data is returned, then the report job is submitted. If data is not returned from the trigger SQL query, the report job is skipped.

The schedule trigger that you associate with a report job can reside in any data model in the catalog. You do not need to create the schedule trigger in the data model of the report for which you wish to execute it. You can reuse schedule triggers across multiple report jobs.

1. In the data model editor task pane, click **Event Triggers**.
2. From the **Event Triggers** pane, click the **Create New** icon.
3. Enter the following for the trigger:
  - **Name** - Enter a name for the trigger.
  - **Type** - Select **Schedule**.
  - **Language** - Accept the default value, SQL Query.
4. In the lower pane, enter the following:
  - **Options** - Select this check box to cache the results of the trigger query.
  - **Data Source** - Select the data source for the trigger query.
  - **SQL Query** - Enter the query in the text area, or click **Query Builder** to use the utility to construct the SQL query, see [Using the SQL Query Builder](#).

You can include parameters in the trigger query. Define the parameter in the same data model as the trigger. Enter parameter values when you schedule the report job.

If the SQL query returns any results, the scheduled report job executes. The figure below shows a schedule trigger to test for inventory levels based on a parameter value that can be entered at runtime.

The screenshot displays the Oracle BI tool interface. On the left is the 'Data Model' pane with a tree view of properties including Data Sets, Employees, Event Triggers, Flexfields, List of Values, List of Depts, Parameters, H\_DATE, PQuantity, and Bursting. The 'Event Triggers' pane is active, showing a table with columns for Name, Type, Language, and Reorder. A row is added with Name 'Quantity', Type 'Schedule', and Language 'SQL Query'. Below this, the configuration for the 'Quantity' trigger is shown, including an unchecked 'Cache Result' option, a 'Data Source' dropdown set to 'demo', and a text area containing the following SQL query:

```
select 'true'
from "OE"."OC_INVENTORIES"."OC_INVENTORIES"
where "OC_INVENTORIES"."QUANTITY_ON_HAND" < :pQuantity
```



See Defining the Schedule for a Job in *User's Guide for Oracle Business Intelligence Publisher*.

# 6

## Adding Flexfields

This topic describes the support for flexfields in data models.

### Topics:

- [About Flexfields](#)
- [Adding Key Flexfields](#)
- [Adding Descriptive Flexfields](#)

## About Flexfields

A flexfield is a data field that your organization can customize to your business needs without programming.

Oracle applications (Oracle E-Business Suite and Oracle Fusion Applications) use two types of flexfields:

- key flexfields  
A key flexfield is a field you can customize to enter multi-segment values such as part numbers, account numbers, and so on.
- descriptive flexfields  
A descriptive flexfield is a field you customize to enter additional information for which your Oracle applications product has not already provided a field.

If you are reporting on data from Oracle applications, use the Flexfield component of the data model to retrieve flexfield data. When Oracle BI Publisher is integrated with Oracle Fusion Applications, both key flexfields and descriptive flexfields are supported. When BI Publisher is integrated with Oracle E-Business Suite, only key flexfields are supported.

The screenshot shows the 'Data Model' interface. On the left, a 'Properties' pane lists various components, with 'GL\_Flexfield' selected. The main area displays the 'Flexfields' configuration table.

Lexical Name	Flexfield Type	Lexical Type	Application Short Name	Flexfield Code	Reorder
GL_Flexfield	Key Flexfield	Where	GL	GL#	▲▼

Before including flexfields in your reports, you should understand flexfields in your applications. See your Oracle E-Business Suite or Oracle Fusion Applications documentation.

## Using Flexfields in Your Data Model

Use flexfields based on SQL SELECT statements in your data model.

To use flexfields in your SQL-based data model:

- Add the **Flexfield** component to the data model as described in this chapter.
- Define the SQL SELECT statement against the applications data tables.
- Within the SELECT statement, define each flexfield as a lexical. Use the &LEXICAL\_TAG to embed flexfield related lexicals into the SELECT statement.

## Adding Key Flexfields

You can use key flexfield references to replace the clauses appearing after SELECT, FROM, WHERE, ORDER BY, or HAVING.

Use a flexfield reference when you want the parameter to replace multiple values at runtime. The data model editor supports the following flexfield types:

- **Where** - This type of lexical is used in the WHERE section of the statement. It is used to modify the WHERE clause such that the SELECT statement can filter based on key flexfield segment data.
- **Order by** - This type of lexical is used in the ORDER BY section of the statement. It returns a list of column expressions so that the resulting output can be sorted by the flex segment values.
- **Select** - This type of lexical is used in the SELECT section of the statement. It is used to retrieve and process key flexfield (kff) code combination related data based on the lexical definition.
- **Filter** - This type of lexical is used in the WHERE section of the statement. It is used to modify the WHERE clause such that the SELECT statement can filter based on Filter ID passed from Oracle Enterprise Scheduling Service.
- **Segment Metadata** - This type of lexical is used to retrieve flexfield-related metadata. Using this lexical, you are not required to write PL/SQL code to retrieve this metadata. Instead, define a dummy SELECT statement, then use this lexical to get the metadata. This lexical should return a constant string.

After you set up the flexfield components of your data model, create a flexfield lexical reference in the SQL query using the following syntax:

```
&LEXICAL_TAG ALIAS_NAME
```

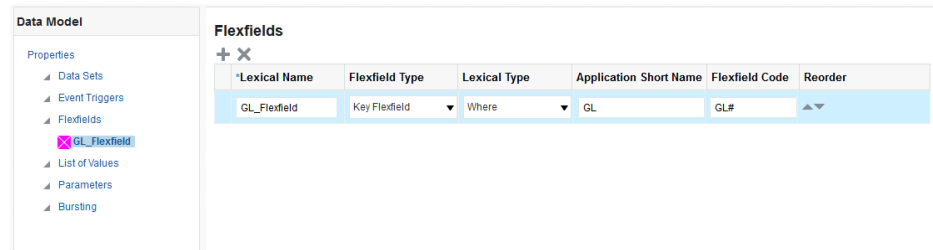
for example:

```
&FLEX_GL_BALANCING alias_gl_balancing
```

After entering the SQL query, when you click **OK**

- Enter the following:
  - **Lexical Name** - Enter a name for the flexfield component.
  - **Flexfield Type** - Select **Key Flexfield**.
  - **Lexical Type** - Select the type from the list. Your selection here determines the additional fields required. See [Entering Flexfield Details](#).

- **Application Short Name** - Enter the short name of the Oracle Application that owns this flexfield, for example, *GL*.
- **Flexfield Code** - Enter the flexfield code defined for this flexfield. In Oracle E-Business Suite this code is defined in the Register Key Flexfield form, for example, *GL#*.
- **ID Flex Number** - Enter the name of the source column or parameter that contains the flexfield structure information. For example: 101. To use a parameter, prefix the parameter name with a colon, for example, *:PARAM\_STRUCT\_NUM*.



## Entering Flexfield Details

The Details region displays appropriate fields depending on the Lexical Type you chose.

### Fields for Key Flexfield Type: Segment Metadata

The table describes the detail fields for segmented metadata.

Field	Description
Structure Instance Number	Enter the name of the source column or parameter that contains the flexfield structure information. For example: 101. To use a parameter, prefix the parameter name with a colon, for example, <i>:PARAM_STRUCT_NUM</i> .

Field	Description
Segments	(Optional) Identifies for which segments this data is requested. Default value is "ALL". See <i>Oracle E-Business Suite Developer's Guide</i> for syntax.
Show Parent Segments	Select this box to automatically display the parent segments of dependent segments even if it is specified as not displayed in the segments attribute.
Metadata Type	Select the type of metadata to return: Above Prompt of Segments — Above prompt of segment(s). Left Prompt of Segments — Left prompt of segment(s)

### Fields for Key Flexfield Type: Select

**KFF\_SELECT: Type: Select**

Enable Multiple Structure Instances

Code Combination Table Alias

Structure Instance Number

Segments

Show Parent Segments

Output Type

The table below shows the detail fields for the Select flexfield type.

Field	Description
Enable Multiple Structure Instances	Indicates whether this lexical supports multiple structures. Checking this box indicates all structures are potentially used for data reporting. The data engine uses <code>&lt;code_combination_table_alias&gt;.&lt;set_defining_column_name&gt;</code> to retrieve the structure number.
Code Combination Table Alias	Specify the table alias to prefix to the column names. Use TABLEALIAS if your SELECT joins to other flexfield tables or uses a self-join.
Structure Instance Number	Enter the name of the source column or parameter that contains the flexfield structure information. For example: 101. To use a parameter, prefix the parameter name with a colon, for example, :PARAM_STRUCT_NUM.
Segments	(Optional) Identifies for which segments this data is requested. Default value is "ALL". See <i>Oracle E-Business Suite Developer's Guide</i> for syntax.

Field	Description
Show Parent Segments	Select this box to automatically display the parent segments of dependent segments even if it is specified as not displayed in the segments attribute.
Output Type	Select from the following: <ul style="list-style-type: none"> <li>• <b>Value</b> — Segment value as it is displayed to user.</li> <li>• <b>Padded Value</b> — Padded segment value as it is displayed to user. Number type values are padded from the left. String type values are padded on the right.</li> <li>• <b>Description</b> — Segment value's description up to the description size defined in the segment definition.</li> <li>• <b>Full Description</b> — Segment value's description (full size).</li> <li>• <b>Security</b> — Returns Y if the current combination is secured against the current user, N otherwise.</li> </ul>

### Fields for Key Flexfield Type: Where

**KFF\_Where Type: Where**

Code Combination Table Alias

Structure Instance Number

Segments

Operator

Operand1

Operand2

Metadata Type

The table below shows the detail fields for the Where key flexfield type.

Field	Description
Code Combination Table Alias	Specify the table alias to prefix to the column names. You use TABLEALIAS if your SELECT joins to other flexfield tables or uses a self-join.
Structure Instance Number	Enter the name of the source column or parameter that contains the flexfield structure information. For example: 101. To use a parameter, prefix the parameter name with a colon, for example, :PARAM_STRUCT_NUM.
Segments	(Optional) Identifies for which segments this data is requested. Default value is "ALL". See <i>Oracle E-Business Suite Developer's Guide</i> for syntax.

Field	Description
Operator	Select the appropriate operator.
Operand1	Enter the value to use on the right side of the conditional operator.
Operand2	(Optional) High value for the BETWEEN operator.

### Fields for Key Flexfield Type: Order By

**KFF\_ORDER\_BY: Type: Order By**

Enable Multiple Structure Instances

Code Combination Table Alias

Structure Instance Number

Segments

Show Parent Segments

The table below shows the detail fields for the Order by flexfield type.

Field	Description
Enable Multiple Structure Instances	Indicates whether this lexical supports multiple structures. Selecting this box indicates all structures are potentially used for data reporting. The data engine uses <code>&lt;code_combination_table_alias&gt;.&lt;set_defining_column_name&gt;</code> to retrieve the structure number.
Structure Instance Number	Enter the name of the source column or parameter that contains the flexfield structure information. For example: 101. To use a parameter, prefix the parameter name with a colon, for example, :PARAM_STRUCT_NUM.
Code Combination Table Alias	Specify the table alias to prefix to the column names. You use TABLEALIAS if your SELECT joins to other flexfield tables or uses a self-join.
Segments	(Optional) Identifies for which segments this data is requested. Default value is "ALL". See <i>Oracle E-Business Suite Developer's Guide</i> for syntax.
Show Parent Segments	Select this box to automatically display the parent segments of dependent segments even if it is specified as not displayed in the segments attribute.

### Fields for Key Flexfield Type: Filter

**KFF\_FILTER: Type: Filter**

Code Combination Table Alias

Structure Instance Number

The table below shows the detail fields for the Filter flexfield type.

Field	Description
Code Combination Table Alias	Specify the table alias to prefix to the column names. You use TABLEALIAS if your SELECT joins to other flexfield tables or uses a self-join.
Structure Instance Number	Enter the name of the source column or parameter that contains the flexfield structure information. For example: 101. To use a parameter, prefix the parameter name with a colon, for example, :PARAM_STRUCT_NUM.

## Adding Descriptive Flexfields

Reporting on descriptive flexfields is supported only for Oracle Fusion Applications.

- Enter the basic flexfield information:
  - Name** - Enter a name for the flexfield component.
  - Flexfield Type** -Select **Descriptive Flexfield**.
  - Lexical Type** - Only **Select** is supported.
  - Application Short Name** - Enter the short name of the Oracle Application that owns this flexfield (for example, FND).
  - Flexfield Code** - Enter the flexfield code defined for this flexfield in the Register Descriptive Flexfield form, for example, FND\_DFF1.

**Flexfields**

+ X

*Lexical Name	Flexfield Type	Lexical Type	Application Short Name	Flexfield Code	Reorder
DFF_SELECT	Descriptive Flexfield ▼	Select ▼	FND	FLEX_DFF1	▲▼

- Enter the flexfield details:
  - Table Alias** -Specify the table alias to prefix to the column names. Use TABLEALIAS if your SELECT joins to other flexfield tables or uses a self-join.



- **Flexfield Usage Code** - (Optional) Identifies for which segments this data is requested. Default value is "ALL".
3. If your descriptive flexfield definition includes parameters, you can enter the parameters in the Parameters region.

To enter parameters, click + to add each parameter. Enter a **Label** and a **Value** for each parameter. The Label must match exactly the label in the descriptive flexfield definition.

**GL\_Flexfield: Type: Select**

Table Alias

Flexfield Usage Code

**Parameters**

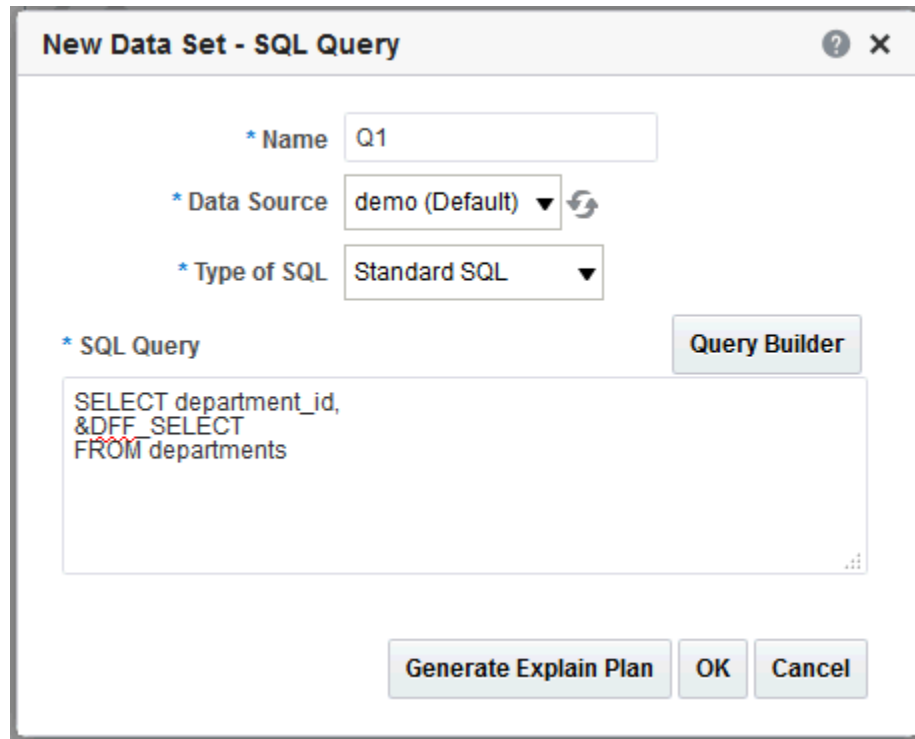
+ X

*Label	*Value
ALL SEGMENTS	3

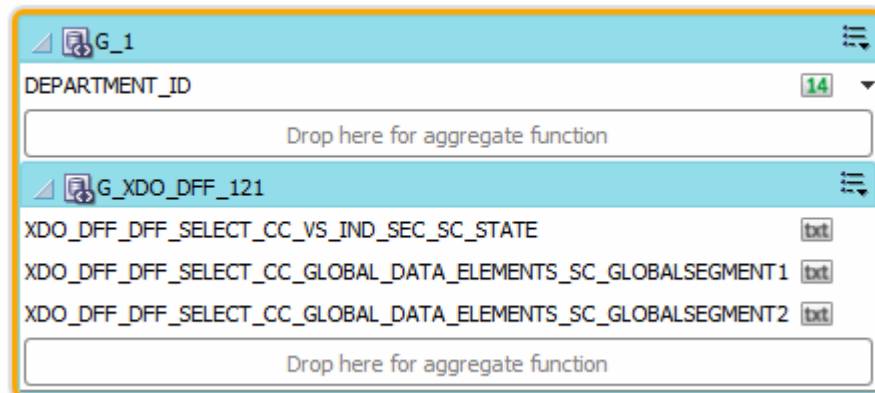
## Including Descriptive Flexfield Reference in SQL Queries

When you create the SQL data set, you can include the descriptive flexfield using the ampersand symbol.

For example, the figure below shows &DFF\_SELECT referencing of the descriptive flexfield.



When you click OK, the diagram of your data set shows the columns that are returned from your descriptive flexfield as shown below.



The columns that are returned from the key flexfield have the following limitations:

- Element properties are disabled
- In the data model Structure tab, you cannot edit the following fields: XML Tag Name, Value if Null, Display Name, Data Type
- Subgrouping of descriptive flexfield elements is not supported
- Element linking is not supported

# 7

## Adding Bursting Definitions

This topic describes the support for bursting reports and how to define a bursting definition in the data model to split and deliver your report to multiple recipients.

### Topics:

- [About Bursting](#)
- [What is the Bursting Definition?](#)
- [Adding a Bursting Definition to Your Data Model](#)
- [Defining the Query for the Delivery XML](#)
- [Passing a Parameter to the Bursting Query](#)
- [Defining the Split By and Deliver By Elements for a CLOB/XML Data Set](#)
- [Configuring a Report to Use a Bursting Definition](#)
- [Sample Bursting Query](#)
- [Creating a Table to Use as a Delivery Data Source](#)

## About Bursting

Bursting is a process of splitting data into blocks, generating documents for each block, and delivering the documents to one or more destinations.

The data for the report is generated by executing a query once and then splitting the data based on a *Key* value. For each block of the data, a separate document is generated and delivered.

Using Oracle BI Publisher bursting enables splitting a single report based on an element in the data model and deliver the report based on a second element in the data model. Driven by the delivery element, you can apply a different template, output format, delivery method, and locale to each split segment of the report. Example implementations include:

- Invoice generation and delivery based on customer-specific layouts and delivery preference.
- Financial reporting to generate a master report of all cost centers, splitting out individual cost center reports to the appropriate manager.
- Generation of pay slips to all employees based on one extract and delivered through e-mail.

## What is the Bursting Definition?

A bursting definition is a component of the data model. After you have defined the data sets for the data model, you can set up one or more bursting definitions.

When you set up a bursting definition, you define the following:

- The **Split By** element is an element from the data that governs how the data is split. For example, to split a batch of invoices by each invoice, you may use an element called CUSTOMER\_ID. The data set must be sorted or grouped by this element.
- The **Deliver By** element is the element from the data that governs how formatting and delivery options are applied. In the invoice example, it is likely that each invoice has delivery criteria determined by customer; therefore, the Deliver By element would also be CUSTOMER\_ID.
- The **Delivery Query** is a SQL query that you define for BI Publisher to construct the delivery XML data file. The query must return the formatting and delivery details.

## Adding a Bursting Definition to Your Data Model

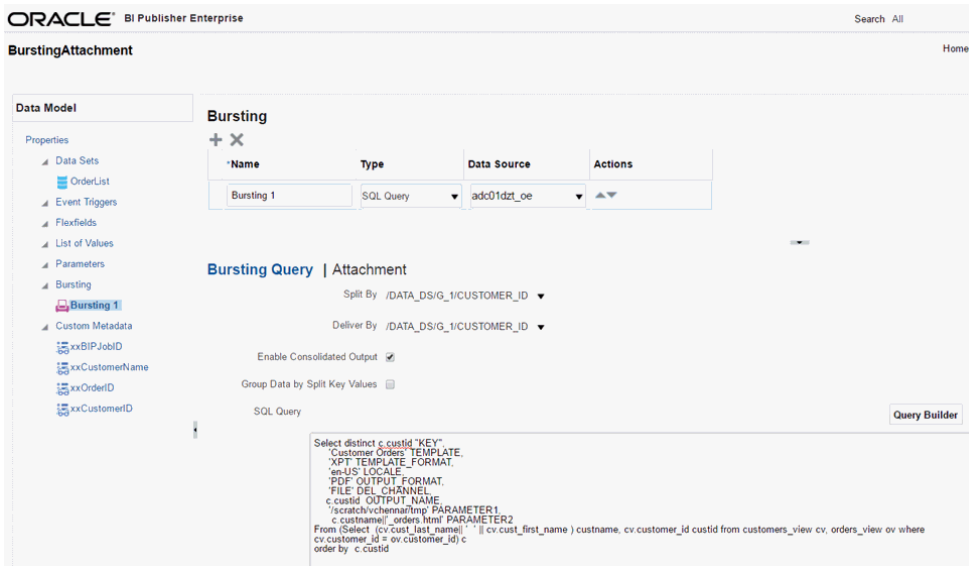
In the Bursting definition table, create a new bursting definition by specifying its name, type, data source, and other properties.

Prerequisites:

- You have defined the data set for this data model
- The data set is sorted or grouped by the element by which you want to split the data in your bursting definition
- The delivery and formatting information is available to Oracle BI Publisher. You can provide the information at runtime to BI Publisher in one of the following ways:
  - The information is stored in a database table available to BI Publisher for a dynamic delivery definition.
  - The information is hard coded in the delivery SQL for a static delivery definition.
- The report definition for this data model has been created and includes the layouts to be applied to the report data.

To add a bursting definition to the data model and enable bursting in your report:

1. On the component pane of the data model editor, click **Bursting** to create a SQL data set to create a bursting query.
2. On the Bursting definition table, click the **Create new Bursting** button.
3. Enter the following for this bursting definition:
  - Name** - Enter a name for the query. For example, *Burst to File*.
  - Type** - Select SQL Query. SQL Query is currently the only supported type.
  - Data Source** - Select the data source that contains the delivery information.



- In the lower region, enter the following for this bursting definition:

**Split By** - Select the element from the data set by which to split the data.

**Deliver By** - Select the element from the data set by which to format and deliver the data.

 **Note:**

If the **Split By** and **Deliver By** elements reside in an XML document stored as a CLOB in your database, you must enter the full XPATH in the **Split By** and **Deliver By** fields. For more information, see [Defining the Split By and Deliver By Elements for a CLOB/XML Data Set](#).

**Enable Consolidated Output** - Select the **Enable Consolidated Output** option to generate a single consolidated report.

**SQL Query** — Enter the query to construct the delivery XML. For information on how to construct the bursting query, see [Defining the Query for the Delivery XML](#).

**Attachment** — You can attach external PDF files to your bursted PDF output. See [Attaching PDF to Reports using Bursting Engine](#).

- In the Report Properties dialog, select **Enable Bursting** to enable bursting for a report.

## Attaching PDF to Reports using Bursting Engine

You may have a requirement to attach PDFs along with invoices for customers. You can now attach these documents as an attachment along with the invoice while bursting.

Once bursting query is defined, you can enter the attachment query in the **Attachment** tab. The attachment expects the repository source to be a WebCenter content, which can be defined as a data source by the Administrator.

- Click the Attachment tab.

2. Select the content server name from the **Attachment Repository** LOV.
3. Define the SQL query for the attachment in the Content Server.

The screenshot shows the Oracle BI Publisher Enterprise interface for configuring a bursting attachment. The main window is titled "BurstingAttachment". On the left, there is a "Data Model" sidebar with a tree view containing "Properties", "Data Sets", "Event Triggers", "Flexfields", "List of Values", "Parameters", and "Bursting". The "Bursting" section is active, showing a table with the following data:

Name	Type	Data Source	Actions
Bursting 1	SQL Query	adc01dzt_loe	▲▼

Below the table, the "Bursting Query" section is visible, showing the "Attachment Repository" dropdown set to "ucmcs". The "SQL Query" field contains the following text:

```
select "ATTACHMENT_MAPPING"."CUSTID" as "KEY",
"ATTACHMENT_MAPPING"."ATTACHMENTID" as "ATTACHMENT_ID"
from "OE"."ATTACHMENT_MAPPING" ATTACHMENT_MAPPING
```

**Note:**

The column alias names KEY and ATTACHMENT\_ID are mandatory.

- The SQL query uses the KEY and the ATTACHMENT\_ID (which map to the document id of the PDF in the content server) to create an association between the PDF report and the PDF attachment.
4. Click **Save** icon after you make changes to the data model.  
The PDF attachments are delivered to recipients along with the main report as a single PDF file.
  5. Click the **View Data** button.
  6. Click **View** to view the data.
  7. Save the data by clicking **Save As Sample Data**.
  8. To create a report based on the data model that you created, click **Create Report**.

If you have a requirement to save the entire PDF report along with the attachments as a single consolidated file, then the report author can check the option **Enable Consolidated Output** under bursting query. A user (with consumer role) who schedules the bursting report job and the Administrator will be able to view the consolidated output in the Job History Details page.

## Defining the Query for the Delivery XML

The bursting query is a SQL query that you define to provide the required information to format and deliver the report.

BI Publisher uses the results from the bursting query to create the delivery XML.

The BI Publisher bursting engine uses the delivery XML as a mapping table for each Deliver By element. The structure of the delivery XML required by BI Publisher is as follows:

```
<ROWSET>
  <ROW>
    <KEY></KEY>
    <TEMPLATE></TEMPLATE>
    <LOCALE></LOCALE>
    <OUTPUT_FORMAT></OUTPUT_FORMAT>
    <DEL_CHANNEL></DEL_CHANNEL>
    <TIMEZONE></TIMEZONE>
    <CALENDAR></CALENDAR>
    <OUTPUT_NAME></OUTPUT_NAME>
    <SAVE_OUTPUT></SAVE_OUTPUT>
    <PARAMETER1></PARAMETER1>
    <PARAMETER2></PARAMETER2>
    <PARAMETER3></PARAMETER3>
    <PARAMETER4></PARAMETER4>
    <PARAMETER5></PARAMETER5>
    <PARAMETER6></PARAMETER6>
    <PARAMETER7></PARAMETER7>
    <PARAMETER8></PARAMETER8>
    <PARAMETER9></PARAMETER9>
    <PARAMETER10></PARAMETER10>
  </ROW>
</ROWSET>
```

- **KEY** — The Delivery key and must match the **Deliver By** element. The bursting engine uses the key to link delivery criteria to a specific section of the burst data. Ensure that you use double quotes around "KEY" in the select statement, for example:

```
select d.department_name as "KEY",
```

- **TEMPLATE** — The name of the Layout to apply. Note that the value is the Layout name (for example, 'Customer Invoice'), not the template file name (for example, invoice.rtf).
- **LOCALE** — The template locale, for example, 'en-US'.
- **OUTPUT\_FORMAT** — The output format. For a description of each type, see *Selecting Output Formats in Report Designer's Guide for Oracle Business Intelligence Publisher*. The table below shows the valid values to enter for the bursting query.

Output Format	Value to Enter in Bursting Query	Template Types That Can Generate This Output Format
Interactive	N/A	Not supported for bursting
HTML	html	BI Publisher, RTF, XSL Stylesheet (FO)
PDF	pdf	BI Publisher, RTF, PDF, Flash, XSL Stylesheet (FO)
RTF	rtf	BI Publisher, RTF, XSL Stylesheet (FO)
Excel (mhtml)	excel	BI Publisher, RTF, Excel, XSL Stylesheet (FO)
Excel (html)	excel2000	BI Publisher, RTF, Excel, XSL Stylesheet (FO)
Excel (*.xlsx)	xlsx	BI Publisher, RTF, XSL Stylesheet (FO)

Output Format	Value to Enter in Bursting Query	Template Types That Can Generate This Output Format
PowerPoint (mhtml)	ppt	BI Publisher, RTF, XSL Stylesheet (FO)
PowerPoint (*.pptx)	pptx	BI Publisher, RTF, XSL Stylesheet (FO)
MHTML	mhtml	BI Publisher, RTF, Flash, XSL Stylesheet (FO)
PDF/A	pdfa	BI Publisher, RTF, XSL Stylesheet (FO)
PDF/X	pdfx	BI Publisher, RTF, XSL Stylesheet (FO)
Zipped PDFs	pdfz	BI Publisher, RTF, PDF, XSL Stylesheet (FO)
FO Formatted XML	xslfo	BI Publisher, RTF, XSL Stylesheet (FO)
Data (XML)	xml	BI Publisher, RTF, PDF, Excel, Flash, XSL Stylesheet (FO), Etext, XSL Stylesheet (HTML XML/Text)
Data (CSV)	csv	BI Publisher, RTF, PDF, Excel, Flash, XSL Stylesheet (FO), XSL Stylesheet (HTML XML/Text), Etext
XML	txml	XSL Stylesheet (HTML XML/Text)
Text	text	XSL Stylesheet (HTML XML/Text), Etext
Flash	flash	Flash

- **SAVE\_OUTPUT** — Indicates whether to save the output documents to BI Publisher history tables that the output can be viewed and downloaded from the Report Job History page.  
Valid values are 'true' (default) and 'false'. If this property is not set, the output is saved.
- **DEL\_CHANNEL** — The delivery method. Valid values are:
  - EMAIL
  - FAX
  - FILE
  - FTP
  - PRINT
  - WEBDAV
  - WCC
  - ODCS
- **TIMEZONE** — The time zone to use for the report. Values must be in the Java format, for example: 'America/Los\_Angeles'. If time zone is not provided, then the system default time zone is used to generate the report.
- **CALENDAR** — The calendar to use for the report. Valid values are:
  - GREGORIAN
  - ARABIC\_HIJRAH
  - ENGLISH\_HIJRAH
  - JAPANESE\_IMPERIAL
  - THAI\_BUDDHA



- ROC\_OFFICIAL (Taiwan)

If not provided, the value 'GREGORIAN' is used.

- **OUTPUT\_NAME** — The name to assign to the output file in the report job history.
- **Delivery parameters by channel** — The values required for the parameters depend on the delivery method chosen. The parameter values mappings for each method are shown in the table below. Not all delivery channels use all the parameters.

Delivery Channel	PARAMETER Values
Email	PARAMETER1: Email address PARAMETER2: cc PARAMETER3: From PARAMETER4: Subject PARAMETER5: Message body PARAMETER6: Attachment value ('true' or 'false'). If your output format is PDF, you must set this parameter to "true" to attach the PDF to the e-mail. PARAMETER7: Reply-To PARAMETER8: Bcc (PARAMETER 9-10 are not used)
Printer	PARAMETER1: Printer group PARAMETER2: Printer name or for a printer on CUPS, the printer URI, for example: <code>ipp://myserver.com:631/printers/printer1</code> PARAMETER3: Number of copies PARAMETER4: Sides. Valid values are: <ul style="list-style-type: none"> <li>• "d_single_sided" for single-sided</li> <li>• "d_double_sided_l" for duplex/long edge</li> <li>• "d_double_sided_s" for tumble/short edge</li> </ul> If the parameter is not specified, single-sided is used. PARAMETER5: Tray. Valid values are: <ul style="list-style-type: none"> <li>• "t1" for "Tray 1"</li> <li>• "t2" for "Tray 2"</li> <li>• "t3" for "Tray 3"</li> </ul> If not specified, the printer default is used. PARAMETER6: Print range. For example "3" prints page 3 only, "2-5" prints pages 2-5, "1,3-5" prints pages 1 and 3-5 (PARAMETER 7-10 are not used)
Fax	PARAMETER1: Fax Server Name PARAMETER2: Fax number (PARAMETER 3-10 are not used)
WebDAV	PARAMETER1: Server Name PARAMETER2: Username PARAMETER3: Password PARAMETER4: Remote Directory PARAMETER5: Remote Filename PARAMETER6: Authorization type, values are 'basic' or 'digest' (PARAMETER 7-10 are not used)

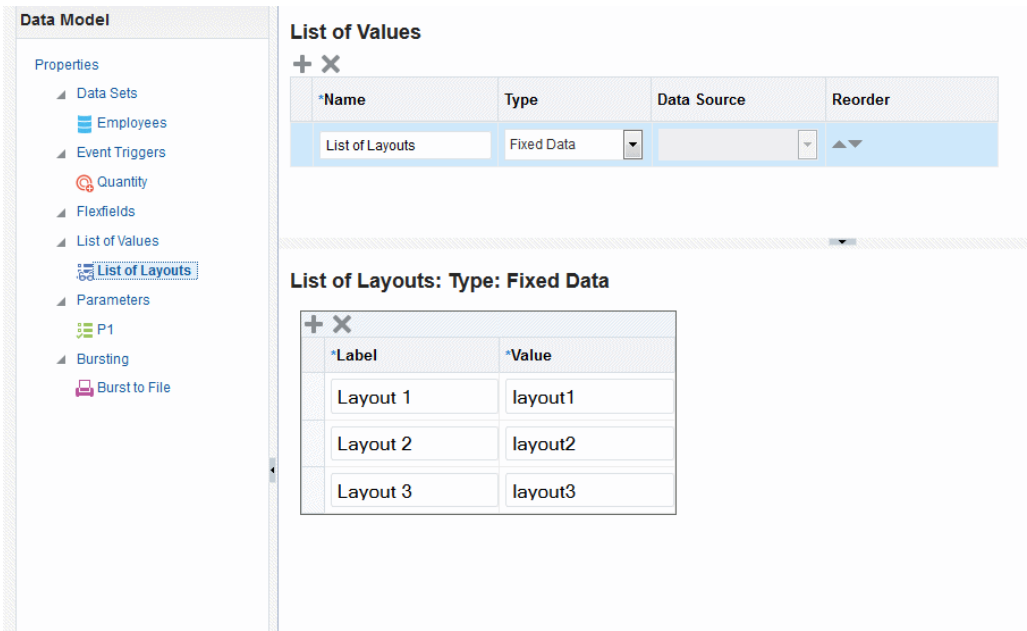
Delivery Channel	PARAMETER Values
File	PARAMETER1: Directory PARAMETER2: File Name (PARAMETER 3-10 are not used)
FTP and SFTP	PARAMETER1: Server Name PARAMETER2: Username PARAMETER3: Password PARAMETER4: Remote Directory PARAMETER5: Remote Filename PARAMETER6: Secure (set this value to 'true' to enable Secure FTP) (PARAMETER 7-10 are not used)
WCC	PARAMETER1: Server Name PARAMETER2: Security Group PARAMETER3: Author PARAMETER4: Account (Optional) PARAMETER5: Title PARAMETER6: Primary File (or File Name) PARAMETER7: Comments (Optional) PARAMETER8: Content ID (Optional. Content ID must be unique.) PARAMETER9: Custom Metadata
Document Cloud Services	PARAMETER1: Server Name PARAMETER2: Folder Name PARAMETER3: File Name

## Passing a Parameter to the Bursting Query

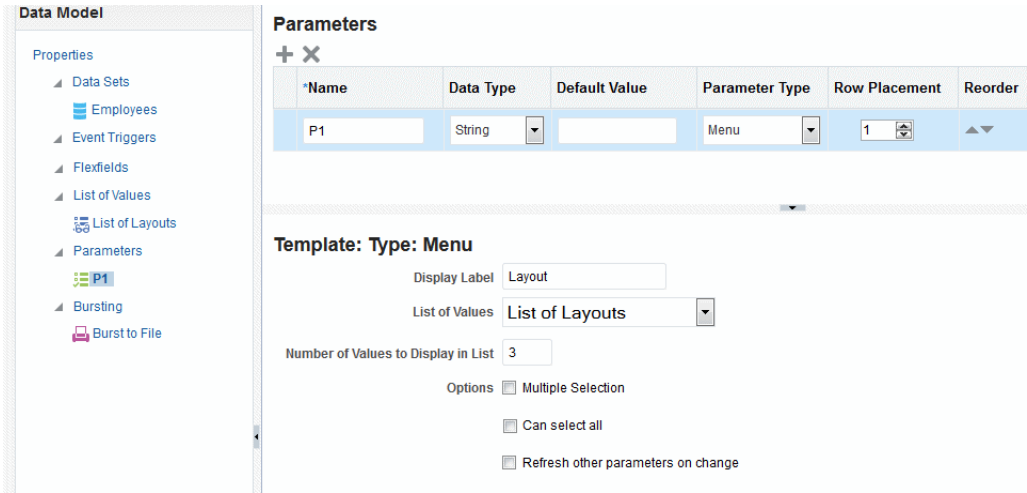
You can pass the value for an element of your bursting XML using a parameter defined in the data model.

For example, if you want to be able to select the template at the time of submission, you can define a parameter in the data model and use the `:parameter_name` syntax in your query. The following example demonstrates this use case of a parameter in a bursting query.

Assume your report definition includes three layouts: `layout1`, `layout2`, and `layout3`. At submission time you want to select the layout (or `TEMPLATE`, as defined in the bursting query) to use. In your data model, define a list of values with the layout names. The following figure shows a data model with the layout list of values:



Next create a menu type parameter, here named P1:



In the bursting query, pass the parameter value to the TEMPLATE field using :P1 as shown in the following figure:

**Data Model**

Properties

- Data Sets
  - Employees
- Event Triggers
- Flexfields
- List of Values
- List of Layouts
- Parameters
  - P1
- Bursting
  - Burst to File

**Bursting**

Name	Type	Data Source	Actions
Burst to File	SQL Query	demo	▲ ▼

**Burst to File**

Split By: /DATA\_DS/G\_1/DEPARTMEN

Deliver By: /DATA\_DS/G\_1/DEPARTMEN

SQL Query

```
select
d.department_name as "KEY"
:P1 as TEMPLATE,
'RTF' as TEMPLATE_FORMAT,
'en-us' as LOCALE,
'PDF' as OUTPUT_FORMAT,
'FILE' as DEL_CHANNEL,
'C:/Temp' as PARAMETER1,
d.department_name || '.pdf' PARAMETER2
from
departments d
```

Query Builder

## Defining the Split By and Deliver By Elements for a CLOB/XML Data Set

If the split-by and deliver-by elements required for your bursting definition reside in a data set retrieved from a CLOB column in a database, BI Publisher cannot parse the XML to present the elements in the **Split By** and **Deliver By** lists.

You therefore must manually enter the XPath to locate each element in the retrieved XML data set. To ensure that you enter the path correctly, use the data model editor's **Get XML Output** feature to view the XML that is generated by the data engine.

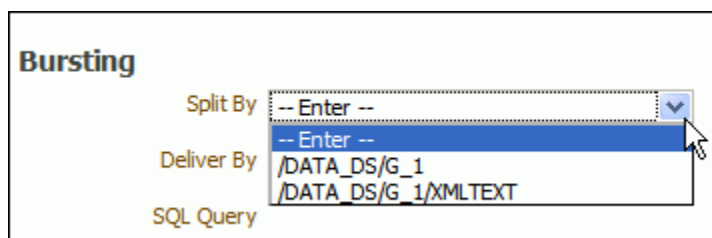
For example, the sample XML code, shown in the figure below, was stored in a CLOB column in the database called "XMLTEXT", and extracted as an XML data set:

```

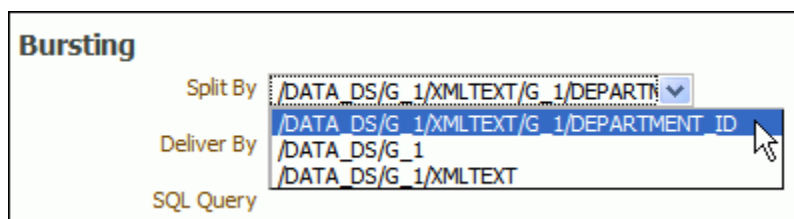
<!--Generated by Oracle BI Publisher -->
-<DATA_DS>
  -<G_1>
    -<XMLTEXT>
      -<DATA_DS>
        -<G_1>
          <DEPARTMENT_ID>10</DEPARTMENT_ID>
          <DEPARTMENT_NAME>Administration</DEPARTMENT_NAME>
          <MANAGER_ID>200</MANAGER_ID>
          <LOCATION_ID>1700</LOCATION_ID>
        -<G_2>
          <EMPLOYEE_ID>200</EMPLOYEE_ID>
          <FIRST_NAME>Jennifer</FIRST_NAME>
          <LAST_NAME>Whalen</LAST_NAME>
          <EMAIL>JWHALEN</EMAIL>
          <PHONE_NUMBER>515.123.4444</PHONE_NUMBER>
          <HIRE_DATE>1987-09-17T00:00:00.000-07:00</HIRE_DATE>
          <JOB_ID>AD_ASST</JOB_ID>
          <SALARY>4400</SALARY>
          <MANAGER_ID_1>101</MANAGER_ID_1>
          <DEPARTMENT_ID_1>10</DEPARTMENT_ID_1>
    
```

For this example, you want to add a bursting definition with split by and deliver by element based on the DEPARTMENT\_ID, which is an element within the CLOB/XML data set.

When you add the bursting definition, the Split By and Deliver By lists cannot parse the structure beneath the XMLTEXT element. Therefore, the list does not display the elements available beneath the XMLTEXT node, as shown in the figure below.



To use the DEPARTMENT\_ID element as the Split By element, manually type the XPath into the field as shown in the figure below.



## Configuring a Report to Use a Bursting Definition

Although you can define multiple bursting definitions for a single data model, you can enable only one for a report.

Enable a report to use a bursting definition on the Report Properties dialog of the report editor. See *Configuring Report Properties in Report Designer's Guide for Oracle Business Intelligence Publisher*.

After you configure the report to use the bursting definition, when you schedule a job for this report you can choose to use the bursting definition to format and deliver the report. See *Creating a Bursting Job in User's Guide for Oracle Business Intelligence Publisher*.

You can also opt not to use the bursting definition and choose your own output and destination as a regular scheduled report.

## Sample Bursting Query

This example of a bursting query is based on an invoice report. This report is to be delivered by CUSTOMER\_ID to each customer's individual e-mail address

This example assumes that the delivery and formatting preferences for each customer are contained in a database table named "CUSTOMERS". The CUSTOMERS table includes the following columns that will be retrieved to create the delivery XML dynamically at runtime:

- CST\_TEMPLATE
- CST\_LOCALE
- CST\_FORMAT
- CST\_EMAIL\_ADDRESS

The CUSTOMER\_ID will be used as the KEY and also to define the output file name.

The SQL code to generate the delivery data set for this example is as follows:

```
select distinct
CUSTOMER_ID as "KEY",
CST_TEMPLATE TEMPLATE,
CST_LOCALE LOCALE,
CST_FORMAT OUTPUT_FORMAT,
CUSTOMER_ID OUTPUT_NAME,
'EMAIL' DEL_CHANNEL,
CST_EMAIL_ADDRESS PARAMETER1,
'accounts.receivable@example.com' PARAMETER2,
'bip-collections@example.com' PARAMETER3,
'Your Invoices' PARAMETER4,
'Hi'||CUST_FIRST_NAME||':'|| 'Please find attached your
invoices.' PARAMETER5,
'true' PARAMETER6,
'donotreply@mycompany.com' PARAMETER7
from CUSTOMERS
```

## Creating a Table to Use as a Delivery Data Source

If the delivery information is not easily available in the existing data sources, then you can consider creating a table to use for the query to create the delivery XML.

Following is a sample:

### Note:

If the JDBC driver that you use does not support column alias, when you define the bursting control table, the columns must match exactly the control XML tag name. For example, the KEY column must be named *KEY*, upper case is required. PARAMETER1 must be named *PARAMETER1*, not *parameter1* nor *param1*, or any other non-matching name.

```
CREATE TABLE "XXX"."DELIVERY_CONTROL"
( "KEY" NUMBER,
  "TEMPLATE" VARCHAR2(20 BYTE),
  "LOCALE" VARCHAR2(20 BYTE),
  "OUTPUT_FORMAT" VARCHAR2(20 BYTE),
  "DEL_CHANNEL" VARCHAR2(20 BYTE),
  "PARAMETER1" VARCHAR2(100 BYTE),
  "PARAMETER2" VARCHAR2(100 BYTE),
  "PARAMETER3" VARCHAR2(100 BYTE),
  "PARAMETER4" VARCHAR2(100 BYTE),
  "PARAMETER5" VARCHAR2(100 BYTE),
  "PARAMETER6" VARCHAR2(100 BYTE),
  "PARAMETER7" VARCHAR2(100 BYTE),
  "PARAMETER8" VARCHAR2(100 BYTE),
  "PARAMETER9" VARCHAR2(100 BYTE),
  "PARAMETER10" VARCHAR2(100 BYTE),
  "OUTPUT_NAME" VARCHAR2(100 BYTE),
  "SAVE_OUTPUT" VARCHAR2(4 BYTE),
  "TIMEZONE" VARCHAR2(300 BYTE),
  "CALENDAR" VARCHAR2(300 BYTE)
) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
TABLESPACE "EXAMPLES";
```

Tips for creating a creating bursting delivery table:

- If the split data set does not contain a *DELIVERY\_KEY* value, then the document is neither delivered nor generated. For example, using the preceding example, if customer with ID 123 is not defined in the bursting delivery table, this customer's document is not generated.
- To enable a split data set to generate more than one document or deliver to more than one destination, duplicate the *DELIVERY\_KEY* value and provide different sets of *OUTPUT\_FORMAT*, *DEL\_CHANNEL*, or other parameters. For example, customer with ID 456 wants his document delivered to two e-mail addresses. To achieve this, insert two rows in the table, both with 456 as the *DELIVERY\_KEY* and each with its own e-mail address.

# 8

## Adding Custom Metadata for Oracle WebCenter Content Server

This topic describes how to use the data model editor to map fields from your data source to the custom metadata fields. When delivering reports to an Oracle WebCenter Content Server, BI Publisher can populate the custom metadata fields defined in your document profiles.

### Topics:

- [About Custom Metadata Mapping](#)
- [Mapping Data Fields to Custom Metadata Fields](#)
- [Deleting Unused Metadata Fields](#)

### About Custom Metadata Mapping

The Custom Metadata component of the data model enables mapping data fields, for example, invoice number or customer name, from your data model to the metadata fields defined in document profile rules configured on your Oracle WebCenter Content Server.

When you run the report and select an Oracle WebCenter Content server as the delivery destination, Oracle BI Publisher generates and stores the document on the content server with the metadata.

### Prerequisites

Certain requirements must be met to use this feature of the data model editor.

Prerequisites include:

- The content server must be configured as a delivery destination with custom metadata enabled.  
*See Adding a Content Server in Administrator's Guide for Oracle Business Intelligence Publisher.*
- To map the custom metadata fields to data fields from your data model data set, the data set must be of a type that the data model editor can retrieve the data structure, for example, SQL data sets and Excel data sets are supported; however, Web service data sets are not.

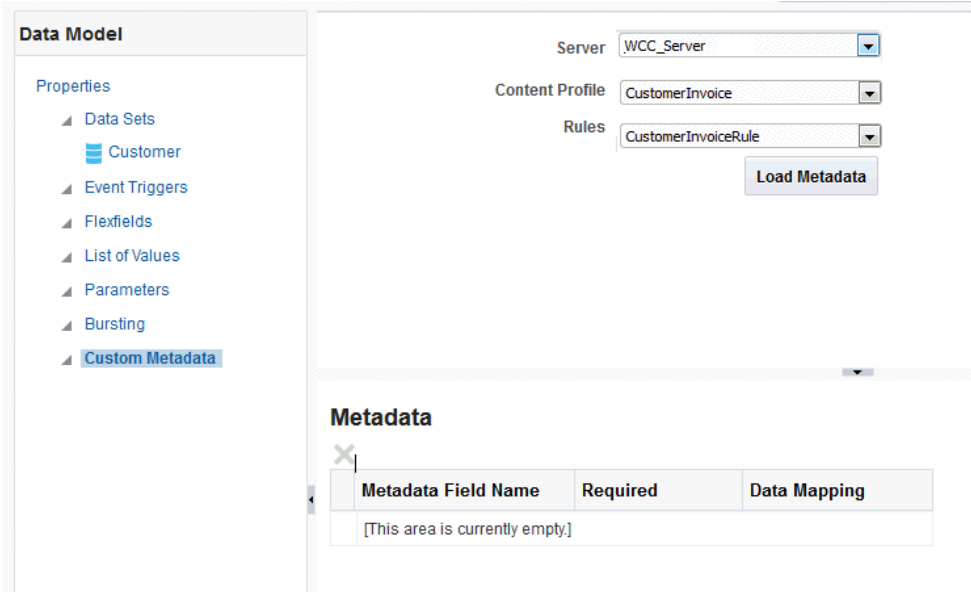
### Mapping Data Fields to Custom Metadata Fields

You can start metadata under a document profile.

To map custom metadata:

1. In the data model editor task pane, click **Custom Metadata**.





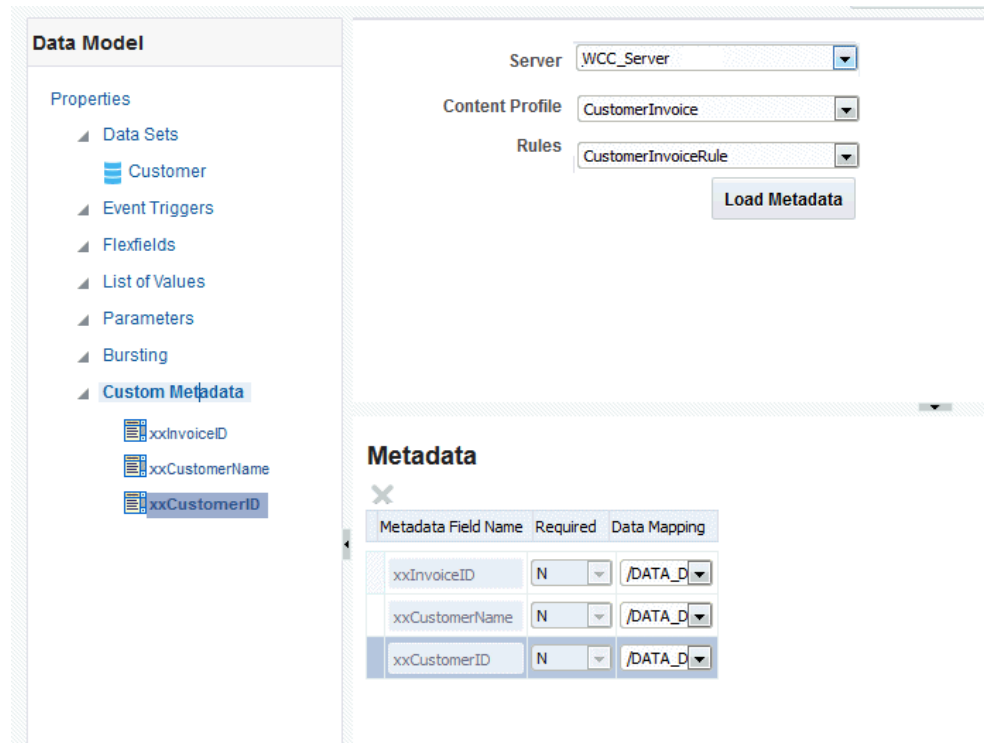
- Oracle WebCenter Content Server stores metadata under a document profile. A document profile is further nested into rules. To retrieve the metadata fields for mapping, you first select the WebCenter server, then the Content Profile, then the Rules set.

On the Custom Metadata header region, select the Rules as follows:

- **Server** — Select the Web content server where the content profile is defined.
- **Content Profile** — Select the content profile that includes the rules that define custom metadata fields.
- **Rules** — Select the Rules set that specifies the metadata fields.

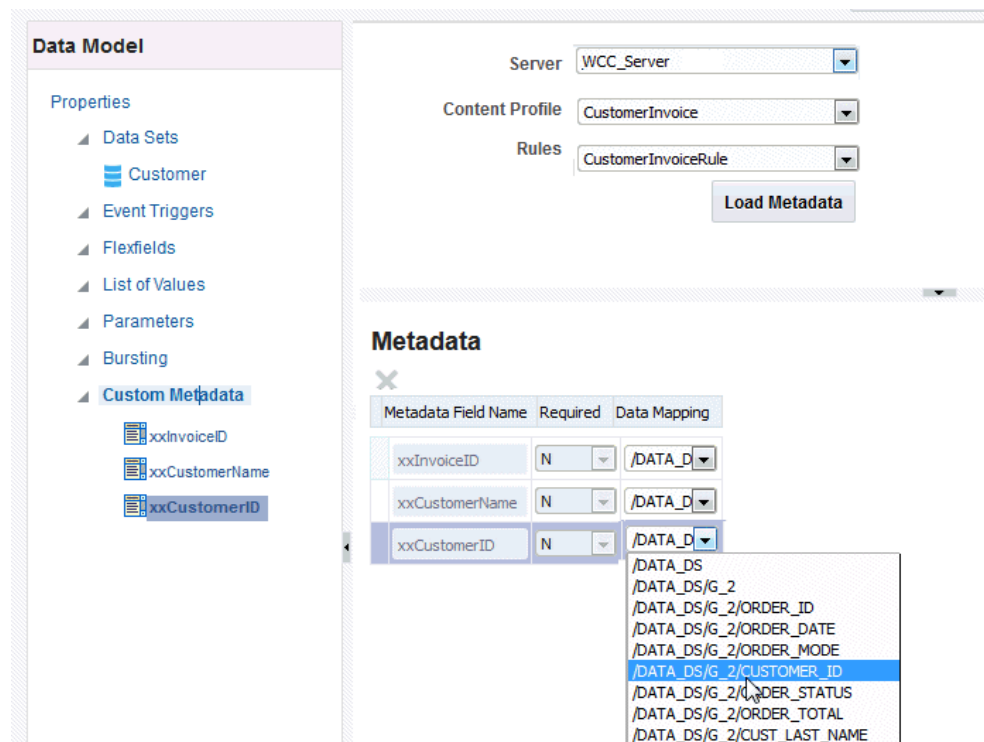
If you do not select a Rules set, then BI Publisher loads the metadata for all Rules under the Content Profile.

- Click **Load Metadata**. The lower pane displays the metadata fields defined in the **Rules** you selected.



- For each metadata field, map a data field from your data sets by selecting it from the **Data Mapping** list. The Data Mapping list displays all the data fields from your data sets.

If a metadata field is required a Y is displayed in the **Required** column.



- When you are done with mapping the metadata fields, click **Save**.

## Deleting Unused Metadata Fields

BI Publisher loads all the metadata fields defined for the Rules set that you select. You can delete unneeded custom metadata fields.

1. Select the metadata field, either by clicking the field name on the left pane or by clicking the selection column in the table.
2. Click the **Delete** button.

# 9

## Performance Best Practices

This topic provides tips for creating efficient data models for better performance.

### Topics:

- [Know Oracle WebLogic Server Default Time Out Setting](#)
- [Best Practices for SQL Data Sets](#)
- [Limit Lists of Values](#)
- [Working with Lexicals/Flexfields](#)
- [Working with Date Parameters](#)
- [Run Report Online/Offline \(Schedule\)](#)
- [Setting Data Model Properties to Prevent Memory Errors](#)
- [SQL Query Tuning](#)

## Know Oracle WebLogic Server Default Time Out Setting

WebLogic Server has a default time out of 600 seconds for each thread that spans for a request.

When the time exceeds 600 seconds, Oracle WebLogic Server marks the thread as *Stuck*. When the number of Stuck threads reaches 25, the server shuts down.

To avoid this problem, verify that your SQL execution time does not exceed the WebLogic Server setting.

## Best Practices for SQL Data Sets

Consider the following tips to help you create more efficient SQL data sets:

- [Only Return the Data You Need](#)
- [Use Column Aliases to Shorten XML File Length](#)
- [Avoid Using Group Filters by Enhancing Your Query](#)
- [Avoid PL/SQL Calls in WHERE Clauses](#)
- [Avoid Use of the System Dual Table](#)
- [Avoid PL/SQL Calls at the Element Level](#)
- [Avoid Including Multiple Data Sets](#)
- [Avoid Nested Data Sets](#)
- [Avoid In-Line Queries as Summary Columns](#)
- [Avoid Excessive Parameter Bind Values](#)

- [Tips for Multi-value Parameters](#)
- [Group Break and Sorting Data](#)

## Only Return the Data You Need

Ensure that your query returns only the data you need for your reports. Returning excessive data risks `OutOfMemory` exceptions.

For example, never simply return all columns as in:

```
SELECT * FROM EMPLOYEES;
```

Always avoid the use of `*`.

Two best practices for restricting the data returned are:

- Always select only the columns you need

For example:

```
SELECT DEPARTMENT_ID, DEPARTMENT_NAME FROM EMPLOYEES;
```

- Use a `WHERE` clause and bind parameters whenever possible to restrict the returned data more precisely.

This example selects only the columns needed and only those that match the value of the parameter:

```
SELECT DEPARTMENT_ID, DEPARTMENT_NAME  
FROM EMPLOYEES  
WHERE DEPARTMENT_ID IN (:P_DEPT_ID)
```

## Use Column Aliases to Shorten XML File Length

The shorter the column name, the smaller the resulting XML file; the smaller the XML file the faster the system parses it.

Shorten your column names using aliases to shorten I/O processing time and enhance report efficiency.

In this example, `DEPARTMENT_ID` is shortened to "id" and `DEPARTMENT_NAME` is shortened to "name":

```
SELECT DEPARTMENT_ID id, DEPARTMENT_NAME name  
FROM EMPLOYEES  
WHERE DEPARTMENT_ID IN (:P_DEPT_ID)
```

## Avoid Using Group Filters by Enhancing Your Query

Although the Data Model Group Filter feature enables you to remove records retrieved by your query, this process takes place in the middle tier, which is much less efficient than the database tier.

It is a better practice to remove unneeded records through your query using `WHERE` clause conditions instead.

## Avoid PL/SQL Calls in WHERE Clauses

PL/SQL function calls in the WHERE clause of the query can result in multiple executions.

These function calls execute for each row found in the database that matches. Moreover, this construction requires PL/SQL to SQL context switching, which is inefficient.

As a best practice, avoid PL/SQL calls in the WHERE clause; instead, join the base tables and add filters.

## Avoid Use of the System Dual Table

The use of the system DUAL table for returning the sysdate or other constants is inefficient. You should avoid using the system DUAL table when not required.

For example, instead of:

```
SELECT DEPARTMENT_ID ID, (SELECT SYSDATE FROM DUAL) TODAYS_DATE FROM DEPARTMENTS
WHERE DEPARTMENT_ID IN (:P_DEPT_ID)
```

Consider:

```
SELECT DEPARTMENT_ID ID, SYSDATE TODAYS_DATE FROM DEPARTMENTS WHERE DEPARTMENT_ID IN
(:P_DEPT_ID)
```

In the first example, DUAL is not required. You can access SYSDATE directly.

## Avoid PL/SQL Calls at the Element Level

Package function calls at the element, within the group or row level, are not allowed. You can include package function calls at the global element level because these functions are executed only once per data model execution request.

Example:

```
<dataStructure>
  <group name="G_order_short_text" dataType="xsd:string" source="Q_ORDER_ATTACH">
    <element name="order_attach_desc" dataType="xsd:string"
value="ORDER_ATTACH_DESC"/>
    <element name="order_attach_pk" dataType="xsd:string" value="ORDER_ATTACH_PK"/
  >
```

The following element is incorrect:

```
<element name="ORDER_TOTAL_FORMAT" dataType="xsd:string" value="
WSH_WSHRDPIK_XMLP_PKG.ORDER_TOTAL_FORMAT "/>
<!-- This is wrong should not be called within group.-->
</group>

  <element name="S_BATCH_COUNT" function="sum" dataType="xsd:double"
value="G_mo_number.pick_slip_number"/>
</dataStructure>
```

## Avoid Including Multiple Data Sets

It can seem desirable to create one data model with multiple data sets to serve multiple reports, but this practice results in very poor performance.

When a report runs, the data processor executes all data sets irrespective of whether the data is used in the final output.

For better report performance and memory efficiency, consider carefully before using a single data model to support multiple reports.

## Avoid Nested Data Sets

The data model provides a mechanism to create parent-child hierarchy by linking elements from one data set to another.

At run time, the data processor executes the parent query and for each row in the parent executes the child query. When a data model has many nested parent-child relationships slow processing can result.

A better approach to avoid nested data sets is to combine multiple data set queries into a single query using the WITH clause.

Following are some general tips about when to combine multiple data sets into one data set:

- When the parent and child have a 1-to-1 relationship; that is, each parent row has exactly one child row, then merge the parent and child data sets into a single query.
- When the parent query has many more rows compared to the child query. For example, an invoice distribution table linked to an invoice table where the distribution table has millions of rows compared to the invoice table. Although the execution of each child query takes less than a second, for each distribution hitting the child query can result in STUCK threads.

Example of when to use a WITH clause:

```
Query Q1:
SELECT DEPARTMENT_ID EDID,EMPLOYEE_ID EID,FIRST_NAME FNAME,LAST_NAME LNAME,SALARY
SAL,COMMISSION_PCT COMMFROM EMPLOYEES
```

```
Query Q2:
SELECT DEPARTMENT_ID DID,DEPARTMENT_NAME DNAME,LOCATION_ID LOCFROM DEPARTMENTS
```

Combine the these queries into one using WITH clause as follows:

```
WITH Q1 as (SELECT DEPARTMENT_ID DID,DEPARTMENT_NAME DNAME,LOCATION_ID LOC
FROM DEPARTMENTS),
Q2 as (SELECT DEPARTMENT_ID EDID,EMPLOYEE_ID EID,FIRST_NAME FNAME,LAST_NAME
LNAME,SALARY SAL,COMMISSION_PCT COMM
FROM EMPLOYEES)
SELECT Q1.*, Q2.*
FROM Q1 LEFT JOIN Q2
ON Q1.DID=Q2.EDID
```

## Avoid In-Line Queries as Summary Columns

In-line queries execute for each column for each row. For example, if a main query has 100 columns, and brings 1000 rows, then each column query executes 1000 times.

Altogether, it is 100 multiplied by 1000 times. This is not scalable and cannot perform well. Avoid using in-line sub queries whenever possible.

Avoid the following use of in-line queries. If this query returns only a few rows this approach may work satisfactorily. However, if the query returns 10000 rows, then each sub or in-line query executes 10000 times and the query would likely result in Stuck threads.

```
SELECT
NATIONAL_IDENTIFIERS,NATIONAL_IDENTIFIER,
PERSON_NUMBER,
PERSON_ID,
STATE_CODE
FROM
(select pprd.person_id,(select REPLACE(national_identifier_number,'-') from per_
national_identifiers pni where pni.person_id = pprd.person_id and rownum<2)
national_identifiers,(select national_identifier_number from per_national
identifiers pni where pni.person_id = pprd.person_id and rownum<2) national_
identifier,(select person_number from per_all_people_f ppf
where ppf.person_id = pprd.person_id
and :p_effective_start_date between ppf.effective_start_date and ppf.effective_
end_date) PERSON_NUMBER
(Select hg.geography_code from hz_geographies hg
where hg.GEOGRAPHY_NAME = paddr.region_2
and hg.geography_type = 'STATE') state_code
```

## Avoid Excessive Parameter Bind Values

Oracle Database allows bind maximum of 1000 values per parameter.

Binding a large number of parameter values is inefficient. Avoid binding more than 100 values to a parameter.

When you create a Menu type parameter and your list of values contains many values, ensure that you enable both the *Multiple Selection* and *Can Select All* options, then also select *NULL* value passed to ensure that too many values are not passed.



### New\_Parameter\_1: Type: Menu

Display Label

List of Values  ▼

Number of Values to Display in List

Options  Multiple Selection

Can select all

NULL Value Passed  All Values Passed

Refresh other parameters on change

## Tips for Multi-value Parameters

Report consumers often must run reports that support the certain conditions.

- If no parameter is selected (null), then return all.
- Allow selection of multiple parameter values

In these cases the use of NVL() does not work, you should therefore use

- COALESCE() for queries against Oracle Database
- CASE / WHEN for Oracle BI EE (logical) queries

Example:

```
SELECT EMPLOYEE_ID ID, FIRST_NAME FNAME, LAST_NAME LNAME FROM EMPLOYEES
WHERE DEPARTMENT_ID = NVL(:P_DEPT_ID, DEPARTMENT_ID)
```

The preceding query syntax is correct only when the value of P\_DEPT\_ID is a single value or null. This syntax does not work when you pass more than a single value.

To support multiple values, use the following syntax:

For Oracle Database:

```
SELECT EMPLOYEE_ID ID, FIRST_NAME FNAME, LAST_NAME LNAME FROM EMPLOYEES
WHERE (DEPARTMENT_ID IN (:P_DEPT_ID) OR COALESCE (:P_DEPT_ID, null) is NULL)
```

For Oracle BI EE data source:

```
(CASE WHEN ('null') in (:P_YEAR) THEN 1 END =1 OR "Time"."Per Name Year" in
(:P_YEAR))
```

For Oracle BI EE the parameter data type must be string. Number and date data types are not supported.

## Group Break and Sorting Data

The data model provides a feature to group breaks and sort data.

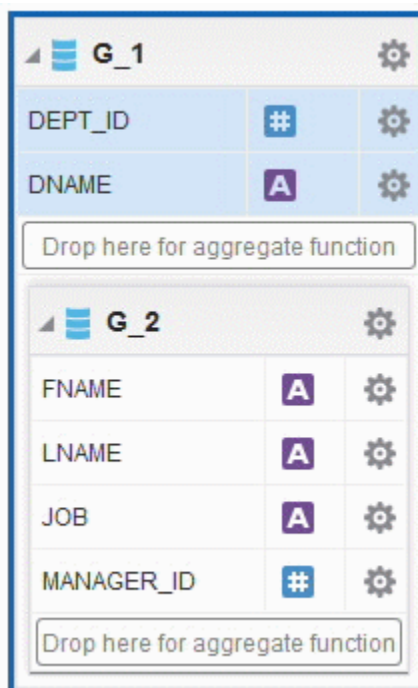
Sorting is supported for parent group break columns only. For example, if a data set of employees is grouped by department and manager, you can sort the XML data by department. If you know how the data should be sorted in the final report or template, you specify sorting at data generation time to optimize document generation. The column order specified in the SELECT clause must exactly match the element orders in the data structure. Otherwise group break and sort may not work. Due to complexity, multiple grouping with multiple sorts at different group levels is not allowed.

Example: In the example shown below, sort and group break are applied to the parent group only, that is, G\_1. Notice the column order in the query, data set dialog, and data structure. The SQL column order must exactly match the data structure element field order; otherwise, it may result in data corruption.

Example:

```
SELECT d.DEPARTMENT_ID DEPT_ID, d.DEPARTMENT_NAME DNAME,
       E.FIRST_NAME FNAME, E.LAST_NAME LNAME, E.JOB_ID JOB, E.MANAGER_ID
FROM EMPLOYEES E, DEPARTMENTS D
WHERE D.DEPARTMENT_ID = E.DEPARTMENT_ID
ORDER BY d.DEPARTMENT_ID, d.DEPARTMENT_NAME
```

Once you define the query, you can use the data model designer to select data elements and create group breaks as shown below.



The Data Structure with breaks is:

```
<output rootName="DATA_DS" uniqueRowName="false">
<nodeList name="data-structure"> <dataStructure tagName="DATA_DS">
<group name="G_1" label="G_1" source="q1">
  <element name="DEPT_ID" value="DEPT_ID" label="DEPT_ID" fieldOrder="1"/>
  <element name="DNAME" value="DNAME" label="DNAME" fieldOrder="2"/>
</group name="G_2" label="G_2" source="q1">
```

```
<element name="FNAME" value="FNAME" label="FNAME" fieldOrder="3"/>
<element name="LNAME" value="LNAME" label="LNAME" fieldOrder="4"/>
<element name="JOB" value="JOB" label="JOB" fieldOrder="5"/>
<element name="MANAGER_ID" value="MANAGER_ID" label="MANAGER_ID"
fieldOrder="6"/>
</group>
</group>
</dataStructure>
</nodeList>
</output>
```

## Limit Lists of Values

Lists of values based on SQL queries must be limited to 1000 rows.

Adding blind runaway queries in a list of values can cause OutOfMemory exceptions. Consider that the number of rows returned by an LOV is stored in memory, therefore the higher the number of rows the more memory usage.

## Working with Lexicals/Flexfields

Oracle BI Publisher supports lexical parameters for Oracle Fusion Applications and Oracle E-Business Suite. Lexical parameters enable you to create dynamic queries.

In BI Publisher, lexical parameters are defined as:

**Lexical** – PL/SQL packaged variable defined as a data model parameter.

**Key Flexfield (KFF)** – Lexical token in a data set query. KFF creates a "code" made up of meaningful segment values and stores a single value as a code combination id. Key Flexfields always return as a single column when used in SELECT / SEGMENT METADATA type or condition when used in WHERE clause. Key Flexfields execute at run time to extract the lexical definition and then are substituted in the SQL query.

**Descriptive Flexfields (DFF)** – Customizable expansion space to track additional information that is important and unique to the business. DFFs can be context sensitive, where the information stored in the application depends on the other values of the user input. Unlike Key Flexfields, Descriptive Flexfields can have multiple context-sensitive segments.

When you define any lexical, name the lexical to match the usage so that when the editor dialog pops up it will be easier to enter the default values for the SQL query. For example, if you are using a lexical in a SELECT clause, use "\_select" as a suffix. The default values must be valid to get metadata.

The following example demonstrates the usage of a lexical:

**Flexfields**

+ X

Lexical Name	Flexfield Type	Lexical Type	Application Short Name	Flexfield Code	Reorder
KFF_SELECT	Key Flexfield	Select	GL	GL#	▲▼

**KFF\_SELECT: Type: Select**

Enable Multiple Structure Instance

Code Combination Table Alias

Structure Instance Number

Segments

Show Parent Segments

Output Type

When you create the data set query for the select columns, specify column alias,

```
SELECT gcc.CODE_COMBINATION_ID,
GCC.ATTRIBUTE_CATEGORY,
gcc.segment1 seg1,
gcc.segment2 seg2,
gcc.segment3 seg3,
gcc.segment4 seg4,
gcc.segment5 seg5,
&KFF_SELECT account
FROM GL_CODE_COMBINATIONS GCC
WHERE gcc.CHART_OF_ACCOUNTS_ID = 101
AND &KFF_WHERE
```

When you save the query, a pop-up dialog prompts you for the default values. To get SQL metadata at design time you must specify the default values that can form a valid SQL query. For example,

- if the lexical usage is a SELECT clause then you could enter null
- if the lexical usage is a WHERE clause then you could enter 1 = 1 or 1 =2
- if the lexical usage is ORDER BY clause then you could enter 1

**Please enter values for lexical references in SQL-Q1** ? X

\*&KFF\_SELECT   flex field

\*&KFF\_WHERE   flex field

OK Cancel

## Working with Date Parameters

Oracle BI Publisher always binds date column or date parameter as a timestamp object.

To avoid timestamp conversion, define the parameter as a string and pass the value with formatting as 'DD-MON-YYYY' to match the RDBMS date format.

## Run Report Online/Offline (Schedule)

Running reports in interactive/online mode uses in-memory processing.

Use the following guidelines for deciding when a report is appropriate for running online.

For Online / Interactive mode:

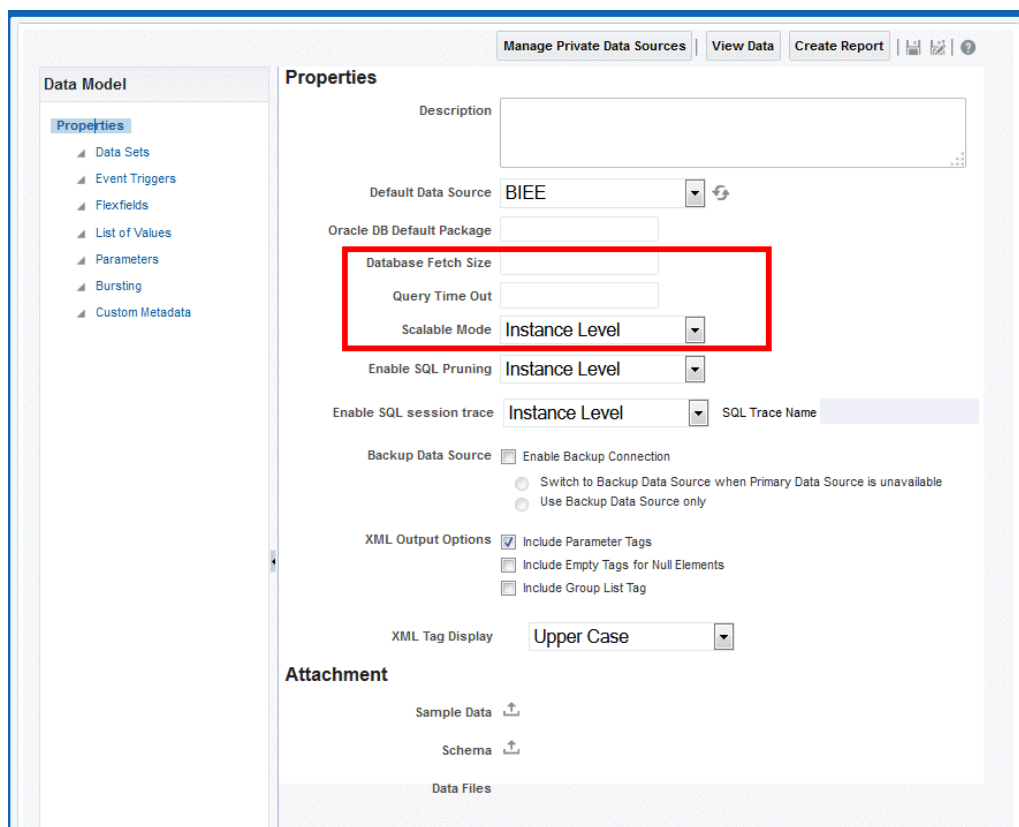
- When report output size is less than 50MB  
Browsers do not scale when loading large volumes of data. Loading more than 50MB in the browser will slow down or possibly crash your session.
- Data model SQL Query time out is less than 600 seconds  
Any SQL query execution that takes more than 600 seconds results in Stuck WebLogic Server threads. To avoid this condition, schedule long-running queries. The Scheduler process uses its own JVM threads instead of Weblogic server threads. It is more efficient to schedule reports than run reports online.
- Total number of elements in the data structure is less than 500  
When the data model data structure contains many data elements, the data processor must maintain the element values in memory; which may result in OutOfMemory exceptions. To avoid this condition, schedule these reports. For scheduled reports, the data processor uses temporary file system to store and process data.
- No CLOB or BLOB columns  
Online processing holds the entire CLOB or BLOB columns in memory. You should schedule reports that include CLOB or BLOB columns.

## Setting Data Model Properties to Prevent Memory Errors

You can use the different data model properties to help prevent memory errors in your system.

These properties include:

- [Query Time Out](#)
- [DB Fetch Size](#)
- [Scalable Mode](#)



## Query Time Out

The Query Time out property specifies the time limit in seconds within which the database must execute SQL statements.

BI Publisher provides a mechanism to set user preferred query time out at the data model level. The default value is 600 seconds.

Queries that cannot execute under 600 seconds are not well-optimized. Your DBA or a performance expert should analyze the query for further tuning.

Increasing the time out value risks Stuck WebLogic Server threads. Do not raise the value unless all other optimizations and alternatives have been utilized.

## DB Fetch Size

The Database Fetch Size property specifies the number of rows that are fetched from the database at a time.

This setting can be overridden at the data model level by setting the Database Fetch Size in the general properties of the data model.

Setting the value higher reduces the number of round trips to the database but consumes more memory. Consider the number of elements in the data model before changing this property.

BI Publisher recommends setting the property Auto DB fetch size to *true* so that the system calculates the fetch size at run time.

## Scalable Mode

When the Scalable mode property is on, BI Publisher uses the temp file system to generate data. The data processor uses the least amount of memory.

This scalable mode property can be set at the data model level and the instance level. The data model setting overrides the instance value.

You can set the instance value by expanding the nodes for **Administrator, Runtime Properties**, and then **Data Model**:

Data Model			
Maximum data size limit for data generation	5GB		500MB
Maximum sample data size limit	10MB		1MB
Enable Data Model scalable mode	True	▼	True
Enable Auto DB fetch size mode	True	▼	True
DB fetch size	20		20
SQL Query Timeout			600 (seconds)
Enable Data Model diagnostic	True	▼	False
Enable SQL session trace	False	▼	False
Enable SQL Pruning	False	▼	False

The instance value can be overridden by Data model setting shown here:

### Properties

Description

Default Data Source  ▼ ↻

Oracle DB Default Package

Database Fetch Size

Query Time Out

Scalable Mode  ▼

Enable SQL Pruning  ▼

Enable SQL session trace  ▼ SQL Trace Name

The following table details the expected results for the possible on/off settings at each level:

Scalable Mode Instance Value	Scalable Mode Data Model Value	Expected Result
On	Instance	On
Off	Instance	Off
On	On	On
On	Off	Off
Off	On	On
Off	Off	Off

## SQL Pruning

SQL pruning enhances performance by fetching only the columns that are used in the report layout/template.

Columns that are defined in the query but are not used in the report are not fetched. This improves query fetch size and reduces JDBC rowset memory.

Note that this feature does not alter the where clause but instead wraps the entire SQL with the columns specified in the layout.

To enable SQL pruning – On the Data Model Properties page, select **On** for the Enable SQL Pruning property.

### Properties

The screenshot shows the 'Properties' configuration page for a data model. The 'Enable SQL Pruning' dropdown menu is highlighted with a red box and is currently set to 'On'. Other visible properties include 'Default Data Source' (demo), 'Scalable Mode' (Instance Level), and 'Enable SQL session trace' (Off). The 'SQL Trace Name' field is also visible.

## SQL Query Tuning

Query tuning is the most important step to improve performance of any report.



Explain plan, SQL Monitoring, SQL Trace facility with TKPROF are the most basic performance diagnostic tools that can help to tune SQL statements in applications running against the Oracle Database.

Oracle BI Publisher provides a mechanism to generate the explain plan and SQL monitoring reports and to enable SQL session trace. This functionality is applicable to SQL statements executing against Oracle Database only. Logical queries against BI Server or any other type of database are not supported.

## Generate Explain Plan

You can generate an Explain plan at the data set level for a single query or at the report level for all queries in a report.

For more information about interpreting the explain plan, see *Oracle Database SQL Tuning Guide*.

## Explain Plan for a Single Query

From the SQL data set Edit dialog you can generate an explain plan before actually executing the query. This provides a best guess estimation of a plan. The query will be executed binding with null values.

Click **Generate Explain Plan** on the Edit SQL Query dialog. Open the generated document in a text editor like Notepad or WordPad.

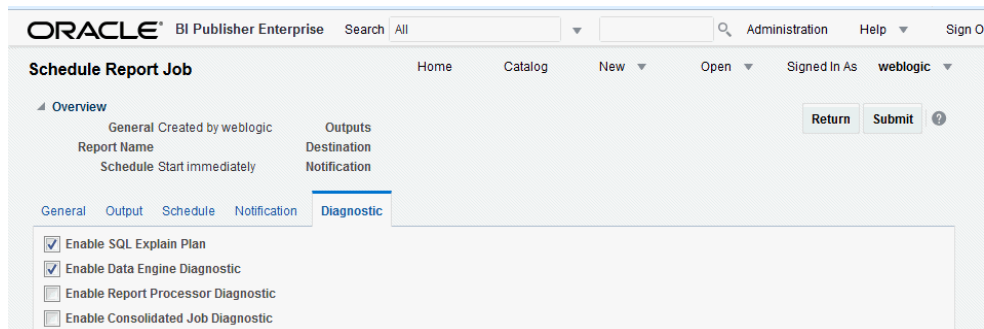
## Explaining Plan for Reports

To generate an explain plan for a report, run the report through the Scheduler.

1. From the **New** menu, select **Report Job**.
2. Select the report to schedule then click the **Diagnostics** tab.

You must have BI Administrator or BI Data Model Developer privileges to access the **Diagnostics** tab.

3. Select Enable SQL Explain Plan and Enable Data Engine Diagnostic.



- Enable SQL Explain Plan — Generates a diagnostic log with Explain plan/SQL monitor report information.
- Enable Data Engine Diagnostic — Generates a data processor log.

- Enable Report Processor Diagnostic — Generates FO (Formatting Options) and server related log information.
  - Enable Consolidated Job Diagnostic — Generates the entire log, which includes scheduler log, data processor log, FO and server log details.
4. Submit the report.
  5. From the Home page, under **Browse/Manage**, select **Report Job History**.
  6. Select the report to view the details. Under **Output & Delivery** click **Diagnostic Log** to download the explain plan output.

**Report Job History** Home Catalog New Open

Last Refreshed Mon Aug 03, 2015 09:28:13 AM Western European Summer Time

**General Information**

General Information		Report Job Execution Information	
Report Job ID	1004	Report Job Status	Success
Report Job Name	Orders Job	Start Processing Time	8/2/15 4:39:44 PM WEST
Owner		End Processing Time	8/2/15 4:39:54 PM WEST
Report Name	New Orders	Time Elapsed	10.655 seconds
Report Scope	Public		
Report Job Schedule	8/2/15 4:39:42 PM WEST		
Active Start Date			
Active End Date			
Trigger Data Model			
Trigger Name			
Trigger Retry Limit			
Trigger Pause Time			
Trigger Parameters			

No parameters available

**Output & Delivery**

XML Data Diagnostic Log

Status: All

Output Name	Template	Format	Locale	Time Zone	C
Output1	Orders	HTML	English (United States)	[GMT+00:00] Casablanca	Gr

Sample Explain plan:

```
SQLQuery:EXPLAIN PLAN SET STATEMENT_ID = 'dm_plan_Q2_150622_0249' FOR
select /* QUERY_SRC('datamodel: _Users_riyengar_XPLAN_Test1_xdm,dataset:Q2') */ *
from departments
WHERE :DEPARTMENT_ID=DEPARTMENT_ID
SQL Query Timeout: 600
Number of SQL Executionst: 108
PLAN_TABLE_OUTPUT
-----
Plan hash value: 4024094692
-----
| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
-----|-----|-----|-----|-----|-----|-----|
| 0 | SELECT STATEMENT | | 1 | 7 | 1 (0) | 00:00:01 |
| 1 | TABLE ACCESS BY INDEX ROWID | DEPARTMENTS | 1 | 7 | 1 (0) | 00:00:01 |
|* 2 | INDEX UNIQUE SCAN | DEPT_ID_PK | 1 | | 0 (0) | 00:00:01 |
-----
Predicate Information (identified by operation id):
-----
2 - access("DEPARTMENT_ID"=:DEPARTMENT_ID)
```

## Guidelines for Tuning Queries

Tune queries by following a set of guidelines.

- Analyze the explain plan and identify high impact SQL statements.
- Add required filter conditions and remove unwanted joins.
- Avoid and remove FTS (full table scans) on large tables. Note that in some cases, full table scans on small tables are faster and improve query fetch. Ensure that you use caching for small tables.
- Use SQL hints to force use of proper indexes.
- Avoid complex sub-queries and use Global Temporary Tables where necessary.
- Use Oracle SQL Analytical functions for multiple aggregation.
- Avoid too many sub-queries in where clauses if possible. Instead rewrite queries with outer joins.
- Avoid group functions like HAVING and IN / NOT IN where clause conditions.
- Use CASE statements and DECODE functions for complex aggregate functions.

## Tips for Database Tuning

Follow best practices when tuning a database.

- Work with your Database Administrator to gather statistics on the tables.
- If the server is very slow, analyze network / IO / Disk issues and optimize the server parameters.
- In some scenarios when you cannot avoid a large data fetch you may encounter PGA Heap size errors in the database. To resolve these issues, increase PGA heap size as a last resort. Use the following statement to increase heap size:

```
alter session set events '10261 trace name context forever, level 2097152'
```