One Design
One Server
One User Experience

Using Information Object Query Builder

Contains information proprietary to:
Actuate Corporation, 951 Mariners Island Boulevard, San Mateo, CA 94404

www.actuate.com

Document No. 130131-2-745303 January 23, 2013

Contents

About *Using Information Object Query Builder* iii

Chapter 1
Building an information object query 1
Examining Information Object Query Builder 2
Finding an information object 2
Creating an information object data source 2
Opening Information Object Query Builder 4
Creating a simple information object query 4
Previewing a query 5
Previewing query results 5

Chapter 2
Editing an information object query 7
About editing an information object query 8
Using the data set editor 8
Working with columns 9
Displaying column categories 9
Defining output columns 9
Using the expression builder 11
Setting column display properties 11
Specifying a join 12
About joins 12
Optimizing joins 14
Using a join algorithm 15
About dependent joins 15
About merge joins 15
About nested loop joins 16
Filtering data 16
Creating a filter condition 17
Selecting multiple values for a filter condition 20
Excluding data 21
Filtering empty or blank values 21
Specifying a date as a comparison value 22
Specifying a number as a comparison value 22
Comparing to a string pattern 22
Comparing to a value in another column 23
Using an expression in a filter condition 24

Creating multiple filter conditions 25
Adding a condition 25
Changing a condition 27
Providing values for a filter 28
Prompting for a single value 28
Prompting for multiple values 28
Using a predefined filter in a query 28
About dynamic filters 29
Creating a dynamic filter 30
Filtering on an aggregate column 33
Sorting a data set 34
Grouping data 35
Creating a GROUP BY clause 37
Creating a GROUP BY clause automatically 37
Creating a GROUP BY clause manually 37
Removing a column from the GROUP BY clause 38
Removing a GROUP BY column automatically 38
Removing a GROUP BY column manually 40
Defining parameters 40
Specifying prompt properties for a parameter 42
Setting parameter properties 44
Setting a source parameter 46
Using localized information objects 47
Creating a textual information object query 48
Displaying output columns 49
Displaying parameters 50
Index 51

About *Using Information Object Query Builder*

Using Information Object Query Builder provides information about using an information object as a data source in an Actuate BIRT design.

Using Information Object Query Builder includes the following chapters:

- *About Using Information Object Query Builder.* This chapter provides an overview of this guide.
- *Chapter 1. Building an information object query.*This chapter describes how to access data from information objects created using the Information Object (IO) Design perspective in Actuate BIRT Designer Professional.
- *Chapter 2. Editing an information object query.*This chapter describes procedures for refining an information object query to return specific data.

Chapter

1

Building an information object query

This chapter contains the following topics:

- Examining Information Object Query Builder
- Creating a simple information object query

Examining Information Object Query Builder

Information Object Query Builder supports defining a SQL query that returns a data set from an information object data source. A data modeler creates an information object using the IO Design perspective. To learn about creating an information object, read *Designing BIRT Information Objects*.

A report developer uses Information Object Query Builder to create a data set from an information object for use in a BIRT design. Examples in this chapter show how to use Information Object Query Builder to build a simple query from an information object. The query returns a data set for a BIRT design.

Finding an information object

Information Object Query Builder supports using an information object in the current project or a published information object as a data source. An information object in the current project is called a local information object. Information Object Query Builder supports building a query from a local information object or from an information object published in an Encyclopedia volume. In BIRT Designer Professional, use Navigator to find a project that contains information objects.

How to find an information object

1. In Navigator, expand the project folder to see the IO Designs and Report Designs folders.
2. Expand the IO Designs folder to view available information objects, as shown in Figure 1-1.

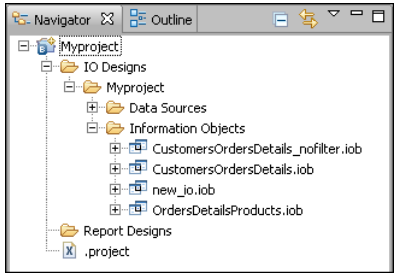


Figure 1-1 Locating information objects in a project

Creating an information object data source

Before defining a data set based on an information object, you must define an information object data source.

How to create an information object data source

1. In the Report Design perspective, create or open a report design.

2 In Data Explorer, right-click Data Sources. Then, choose New Data Source from the context menu.

3 In New Data Source, select Actuate Information Object Data Source from the list of data source types, as shown in Figure 1-2.

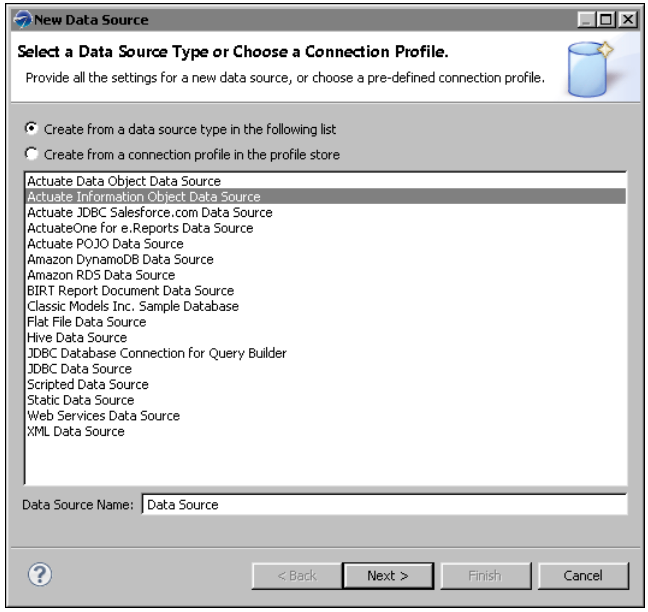


Figure 1-2 Selecting the Actuate Information Object data source type

4 In Data Source Name, type the name of the data source. Choose Next.

5 In New Actuate Information Object Connection Profile, select Use Local Information Objects, as shown in Figure 1-3. Choose Finish.

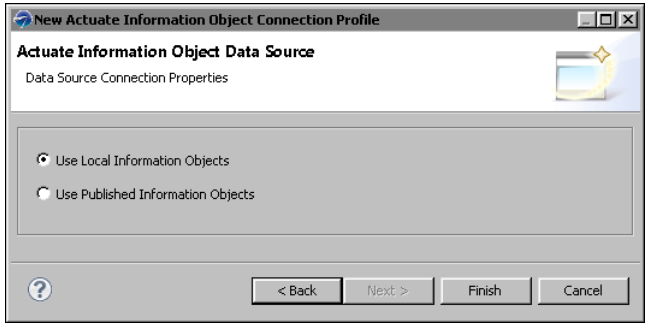


Figure 1-3 Connecting to a local information object data source

Opening Information Object Query Builder

To create a query that returns a data set from an information object, use Information Object Query Builder.

How to open Information Object Query Builder

1. In a report design, select Data Explorer.
2. Right-click Data Sets. Then, choose New Data Set.
3. In New Data Set, type the name of the data set in Data Set Name. Choose Next. Information Object Query Builder opens in New Data Set.

Creating a simple information object query

To create a SQL query using Information Object Query Builder, drag and drop the information object in the query editor. Information Object Query Builder creates a SQL query based on your selection.

How to create a simple query based on an information object

1. In Available Items, expand the project folders necessary to view information objects.
2. Drag one or more information objects from Available Items and drop it in the query editor, as shown in Figure 1-4.

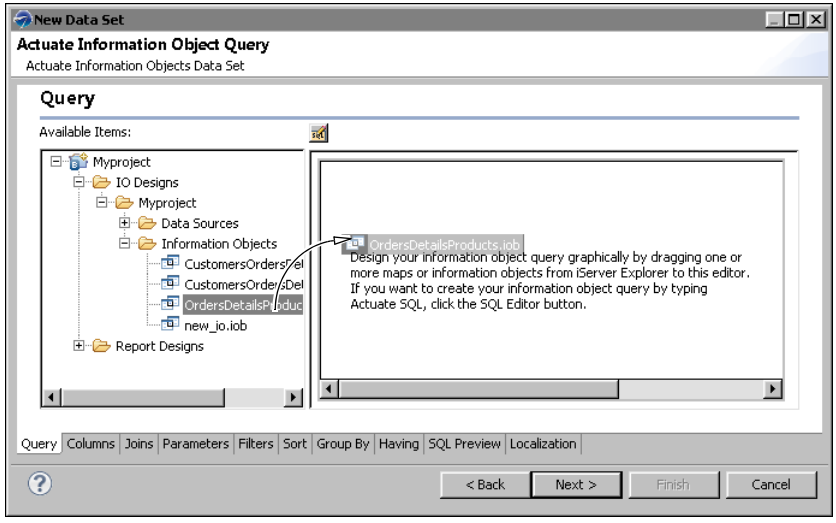


Figure 1-4 Creating a query based on an Actuate information object

3 If necessary, choose any of the tabs at the bottom of the query pane to customize the query.

4 Choose Next. The query editor checks the query syntax for errors.

5 If there are no errors, choose Finish.

Previewing a query

Information Object Query Builder supports previewing query syntax.

How to preview a query

1 In Information Object Query Builder, choose SQL Preview.

2 In SQL Preview, choose Refresh. SQL syntax created by the query builder appears, as shown in Figure 1-5.

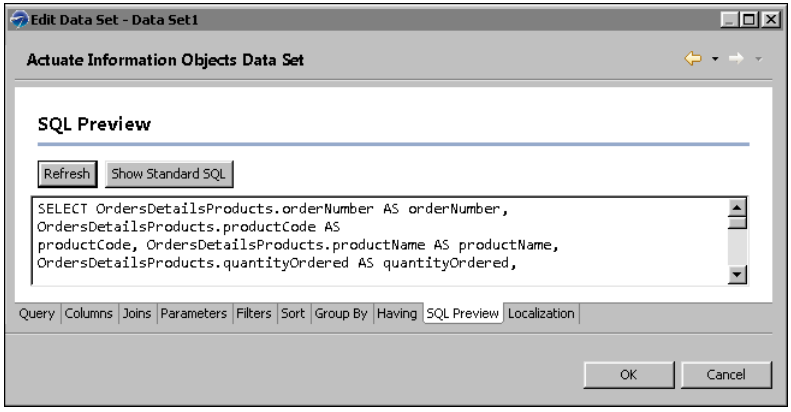


Figure 1-5 Previewing a SQL query based on an information object

Previewing query results

The data set editor supports previewing the results of a query from an information object. You can preview query results before associating the data set with a BIRT design.

How to preview the results of a query

1 In Edit Data Set, choose Preview Results.

A preview of the data rows that the query returns from the information object appears in Preview Results, as shown in Figure 1-6. For a query having parameters, the default value for each parameter appears.

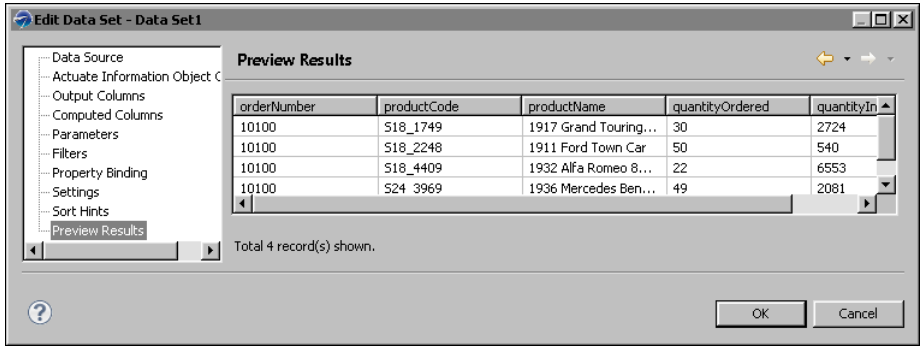


Figure 1-6 Previewing results of a query from an information object

2 Use the scroll bars to view all columns and displayed rows.

How to set the number of records that appear in Preview Results

1 Choose Window➛Preferences.

2 In Preferences, expand Report Design. Select Data Set Editor.

3 In Preview Page, in Number of rows to display, type a number.

Chapter

2

Editing an information object query

This chapter contains the following topics:

- About editing an information object query
- Working with columns
- Specifying a join
- Filtering data
- Sorting a data set
- Grouping data
- Defining parameters
- Using localized information objects
- Creating a textual information object query

About editing an information object query

After defining a data set based on one or more information objects, use Information Object Query Builder to edit the query. Editing a query limits the returned data set so that it contains only data that is useful for a report design.

To edit a query using Information Object Query Builder, perform any of the following tasks:

- Specify output columns for the query.
- For a query that includes more than one information object, define joins.
- Create parameters for which a user provides values when the query runs.
- Define filters that limit the data returned from individual columns.
- Specify a sort order for data rows.
- Select columns that you want the query to group on.
- Specify any aggregate columns that you want to filter in the query.
- Select any localized sources for the query.

The Information Object Query Builder tools appear in the Actuate Information Objects Data Set editor, as shown in Figure 2-1.

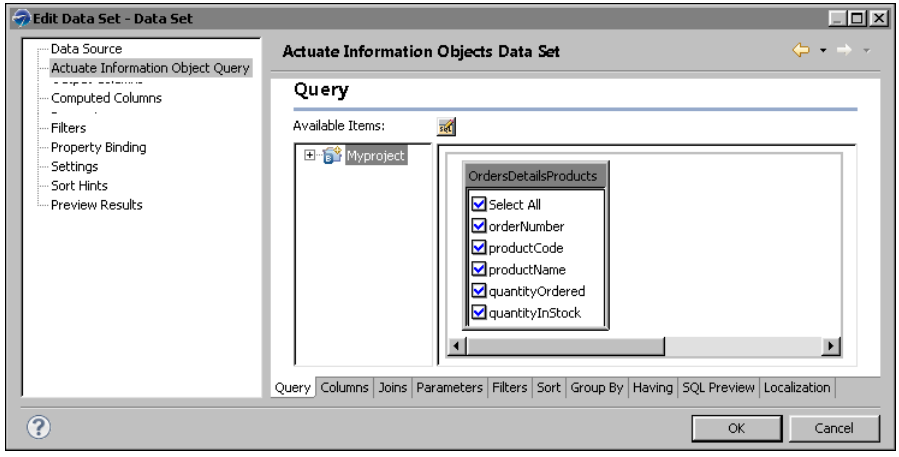


Figure 2-1 Viewing an information object query in the data set editor

Using the data set editor

To customize a query from an information object, open Information Object Query Builder in the data set editor.

How to open Information Object Query Builder in the data set editor

1. In the Report Design perspective, choose Data Explorer.
2. In Data Explorer, right-click the appropriate data set. Choose Edit Data Set.
3. In Edit Data Set, select Actuate Information Object Query.

Working with columns

Initially, an information object query includes all columns from the information object, as shown in Figure 2-1. A check mark appearing beside a column name in Query indicates that a reference to the column appears in the SELECT statement of the information object query.

To limit the data set that an information object query returns, deselect one or more column names. Deselecting a column name removes the column reference from the SELECT statement in the information object query.

Displaying column categories

To identify information object columns, a data modeler may use categories. For example, for an information object that returns customer data, a data modeler can create a Customer address category that contains the columns addressline1, addressline2, city, state, postalCode, and Country.

Column categories in an information object appear when you place the object in Query. To hide column categories in the query editor, choose Toggle categories, as shown in Figure 2-2. The information object on the left displays the Customer address category. The information object on the right does not display column categories.

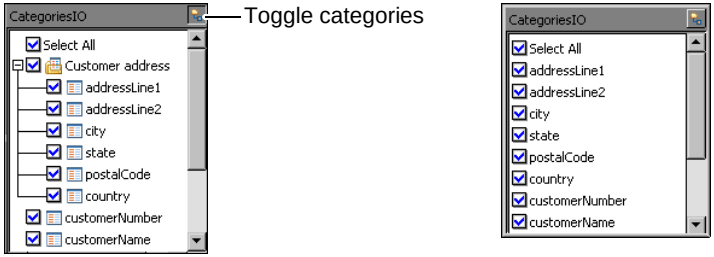


Figure 2-2 Toggling the display of categories in an information object

Defining output columns

To define the output columns for an information object query, use the Columns page in Information Object Query Builder. For example, using Columns, you can

create a SQL SELECT statement that returns employee names from the Employees table and their annual compensation. The SQL fragment is as follows:

```
SELECT ename AS employee, (salary * 12) AS annual_comp
FROM Employees
```

How to define an output column

1. In Information Object Query Builder, choose Columns. By default, the query includes all columns in an information object. Included columns appear in Specify output columns.
2. To modify the set of output columns, in Specify output columns, select a column to remove from the query. Then, choose Remove. To remove all columns from the query, choose Remove All.
3. In Columns:
   - To return only distinct rows, select Distinct values only. Some queries return duplicate rows. In a group of duplicate rows, each selected column contains the same value for all the rows in the group. To return only one row for each group of duplicate rows, select Distinct values only. This setting affects only rows in which all column values match. The query returns rows in which only some of the columns values match. If the Analysis Type property is set to Dimension or Attribute for all columns in an information object, the DISTINCT keyword is included in the query.
   - To change a column alias, type the new alias in Name. If a column alias contains a special character, such as a period or a space, enclose the alias in quotation marks. Column alias names are not case-sensitive. For example, do not use both status and STATUS as column alias names.

   - To enter an expression, select an item in Source column or expression. Then, type an expression. Alternatively, choose ellipsis, as shown in Figure 2-3, to open the expression builder.

   - To change the order of the columns, use the up and down arrows.

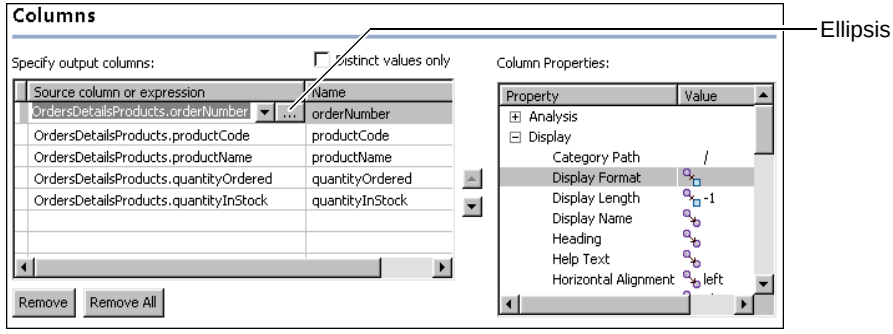


Figure 2-3 Defining output columns

4 To view the properties of a column, select a column row. In Column properties, expand each group to see the properties.

Using the expression builder

Many steps to build a query involve specifying a column or an expression. In a query, an Actuate SQL expression can specify a filter or join, create aggregate data, and so on. For example, the expression, officeID = 101, specifies that data returned by the query must have 101 in the officeID column.

You can add expressions to a query that you build using Information Object Query Builder. In the textual query editor, add an expression in the SQL SELECT statement. In the graphical query editor, you can type an expression or use the expression builder to develop an expression. The expression builder is a dialog box in which you develop an expression by selecting items such as column names, constants, functions, and operators from lists.

To build an expression, drag items from the left pane to the right pane or insert items by choosing the appropriate icon. If you select a function in the left pane, the function signature appears as shown in Figure 2-4.

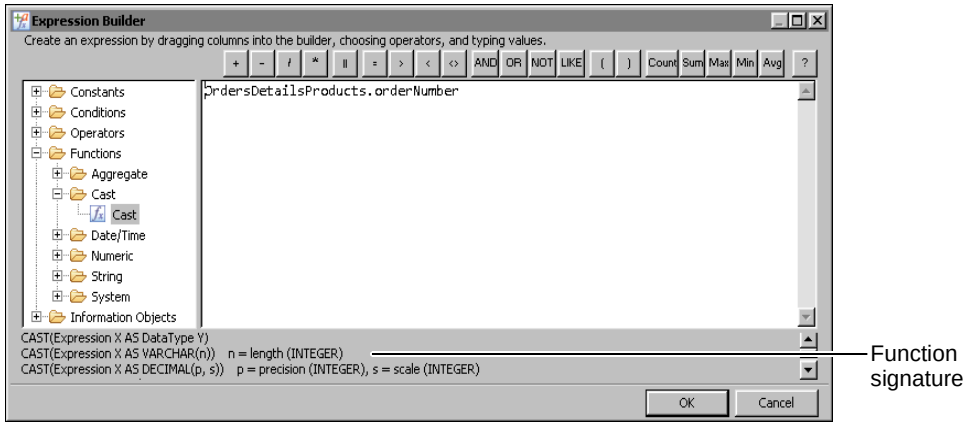


Figure 2-4 Using the expression builder to create expressions

How to open the expression builder

The expression builder opens from the Columns, Joins, Parameters, Filters, Sort, Group By, and Having pages of Information Object Query Builder. Choose ellipsis, appearing next to a column or a value that supports an expression as input, as shown in Figure 2-3.

Setting column display properties

Table 2-1 lists the display properties supported in Information Object Query Builder.

Table 2-1 Display properties

Column property	Description
Display Format	Format to apply to column values. Use a format pattern to specify the display format, for example mm/dd/yy.
Display Name	The column's display name in the Data Explorer view.
Horizontal Alignment	Horizontal alignment of column values: left, right, or center.

Specifying a join

Information Object Query Builder supports adding a join condition to a query based on information objects. To add a join condition to an information object query, drag a column name from one IO and drop it in another IO.

For example, dragging the customerNumber column from the Customers IO and dropping it in the customerNumber column in the Orders IO causes the following SQL fragment to appear in a SQL query:

```
FROM Customers
INNER JOIN Orders ON (Customers.customerNumber =
   Orders.customerNumber)
```

The join condition specifies that the value of the customerNumber column in the Customers IO must match the value of the customerNumber column in the Orders IO. The query returns no rows for customer records having no matching order records.

About joins

A join defines how a query returns data from two information objects. The information objects may be based on different data sources. A join consists of one or more conditions that must all be true. In the resulting SQL SELECT statement, join conditions are linked with AND.

A join can consist of multiple conditions in the following form:

```
columnA = columnB
```

A join can have only one condition that uses an operator other than equality (=) or an expression, for example:

```
columnA < columnB
```

Information Object Query Builder does not support right outer joins or full outer joins.

How to define a join condition

1 In Information Object Query Builder, choose Query.

2 In Query, drag a column from an information object, and drop it in a column in another information object.

A join symbol like the one shown in Figure 2-5 appears between the join columns.

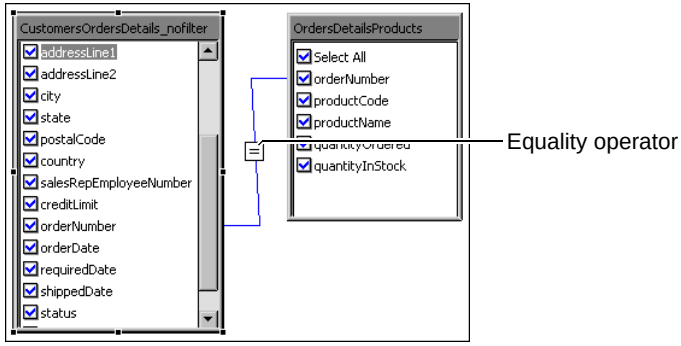


Figure 2-5 Examining a join in the graphical query editor

How to edit properties of a join condition

1 For a query that contains at least one join condition, select Joins.

2 In Joins, select the row that describes the join condition.

3 If necessary, select a different join condition operator from the drop-down list. By default, Information Object Query Builder uses the equality operator (=) to relate two columns.

4 To change a column name to an expression, select the column name, and type the expression. Alternatively, choose ellipsis to open the expression builder, as shown in Figure 2-6.

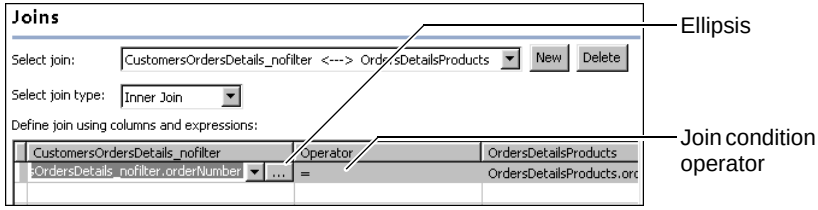


Figure 2-6 Defining a join condition

If the join has a condition that uses an operator other than = or an expression, the symbol shown in Figure 2-7 appears in Query.

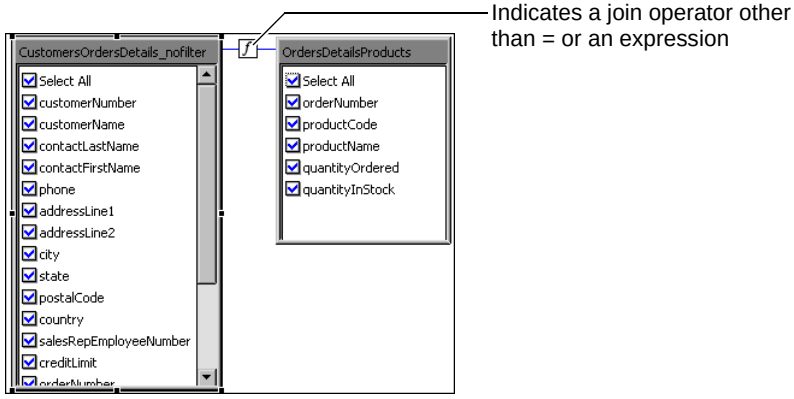


Figure 2-7 A join that uses an expression or an operator other than equality

5 If the join consists of more than one condition, repeat this procedure for the other conditions.

6 Choose one of the following join types:

- Inner join
- Left outer join

7 Optimize the join.

How to delete a join condition

To delete a join condition, select the join condition in Joins, and press Delete.

Optimizing joins

You can improve a query's performance by optimizing the joins. To optimize a join, you can specify the cardinality of the join. Specifying the cardinality of the join adds the CARDINALITY keyword to the Actuate SQL query. If your query is based on two or more information objects that are based on different data sources, you can also optimize the joins by specifying join algorithms.

Figure 2-8 shows how to specify the cardinality of a join and how to specify a join algorithm in Joins.

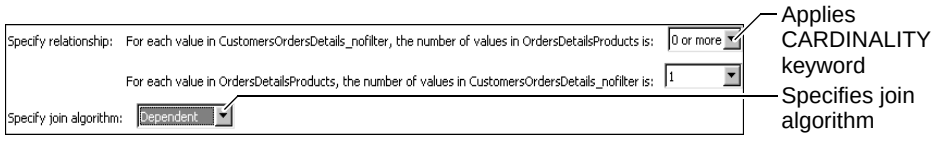


Figure 2-8 Optimizing a join

Using a join algorithm

When joining information objects built from different data sources, the Actuate SQL compiler chooses a join algorithm. If you have a good understanding of the size and distribution of the data, however, you can specify the join algorithm. Choosing the correct join algorithm can significantly reduce information object query execution time. Actuate SQL supports three join algorithms:

- Dependent
- Merge
- Nested loop

When you join information objects that are built from the same data source, specifying a join algorithm has no effect. The join is processed by the data source.

About dependent joins

A dependent join is processed in the following way:

- The left side of the join statement is executed, retrieving all the results. The results are then processed one at a time (pipelined).
- For each left side result, the right side of the join is executed, parameterized by the values provided by the current left side row.

A dependent join is advantageous when the cardinality of the left side is small, and the selectivity of the join criteria is both high and can be delegated to the data source. When the cardinality of the left side is high, a dependent join is relatively slow because it repeatedly executes the right side of the join. A dependent join can be used for any join criteria. Only join expressions that can be delegated to the right side's data source result in improved selectivity performance.

About merge joins

A merge join is processed in the following way:

- The left side of the join statement is executed, retrieving all the results sorted by the left side data source. The results are then processed one at a time (pipelined).
- The right side of the join statement is executed, retrieving all the results sorted by the right side data source. The results are then processed one at a time (pipelined).

A merge join supports only an equijoin. A merge join has much lower memory requirements than a nested loop join and can be much faster. A merge join is especially efficient if the data sources sort the rows.

About nested loop joins

A nested loop join is processed in the following way:

- The left side of the join statement is executed, retrieving all the results. The results are then processed one at a time (pipelined).
- The right side of the join statement is executed. The results are materialized in memory. For each row on the left side, the materialized results are scanned to find matches for the join criteria.

A nested loop join is advantageous when the cardinality of the right side is small. A nested loop join performs well when the join expression cannot be delegated to the data source. A nested loop join supports any join criteria, not just an equijoin.

A nested loop join is a poor choice when the cardinality of the right side is large or unknown, because it may encounter memory limitations. Increasing the memory available to the Integration service removes this limitation. The Integration service parameter Max memory per query specifies the maximum amount of memory to use for an Integration service query. For more information about this parameter, see *Configuring BIRT iHub.*

How to specify a join algorithm

In Joins, select the appropriate join. In Specify join algorithm, shown in Figure 2-9, choose one of the following join algorithms:

- Dependent
- Merge
- Nested loop

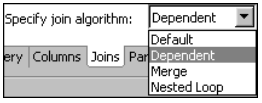


Figure 2-9 Specifying a join algorithm

Filtering data

If an information object query returns more data rows than you need, restrict the number of data rows by using a filter. For example, rather than list all customer sales, create a filter to select only the sales data for a particular week or only the sales data for a particular region.

Filtering data helps you work effectively with large amounts of data. It enables you to find the necessary pieces of information to answer specific business questions, such as which sales representatives generated the top ten sales accounts, which products generated the highest profit in the last quarter, which customers have not made a purchase in the past 90 days, and so on.

Filtering data can also have a positive effect on processing speed. Limiting the number of data rows can reduce the load on the databases because the information object query does not need to return all the rows every time it is run.

Creating a filter condition

When you create a filter, you define one or more conditions that specify which data rows to return. A filter condition is an If expression that must evaluate to true in order for a data row to be returned. For example:

```
If the order total is greater than 10000
If the sales office is San Francisco
If the order date is between 4/1/2012 and 6/30/2012
```

Filter conditions are appended to the information object query's WHERE clause, for example:

```
WHERE OrderTotal > 10000 AND SalesOffice LIKE 'San Francisco%' AND
   OrderDate BETWEEN TIMESTAMP '2012-04-01 00:00:00' AND TIMESTAMP
   '2012-06-30 00:00:00'
```

Figure 2-10 shows an example of a condition defined in Filter Conditions. Filter Conditions helps you define the condition by breaking it down into the following parts:

- The column to evaluate, such as credit limit
- The comparison operator that specifies the type of comparison test, such as > (greater than)
- The value to which all values in the column are compared, such as 10000

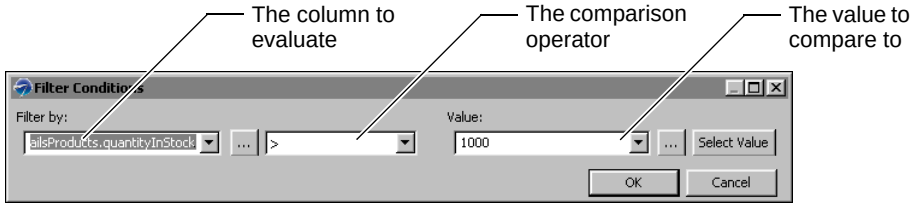


Figure 2-10 Filter Conditions displaying a filter condition

Table 2-2 lists the operators you can use when you create an expression for a filter condition.

Table 2-2 Operators in filter condition expressions

Operator	Use to test whether	Example
BETWEEN	A column value is between two specified values.	Profit BETWEEN 1000 AND 2000

(continues)

Table 2-2 Operators in filter condition expressions (continued)

Operator	Use to test whether	Example
= (Equal to)	A column value is equal to a specified value.	CreditLimit = 100000
> (Greater than)	A column value is greater than a specified value.	Total > 5000
>= (Greater than or equal to)	A column value is greater than or equal to a specified value.	Total >= 5000
IN	A column value is in the specified set of values.	Country IN ('USA', 'Canada', 'Mexico')
IS NOT NULL	A column value is not a null value. A null value means that no value is supplied.	CreditLimit IS NOT NULL
IS NULL	A column value is a null value.	CreditLimit IS NULL
< (Less than)	A column value is less than a specified value.	Total < 5000
<= (Less than or equal to)	A column value is less than or equal to a specified value.	Total <= 5000
LIKE	A column value matches a string pattern.	ProductName LIKE 'Ford%'
NOT BETWEEN	A column value is not between two specified values.	Profit NOT BETWEEN 1000 AND 2000
<> (Not equal to)	A column value is not equal to a specified value.	CreditLimit <> 100000
NOT IN	A column value is not in the specified set of values.	Country NOT IN ('USA', 'Canada', 'Mexico')
NOT LIKE	A column value does not match a string pattern.	ProductName NOT LIKE 'Ford%'

How to create a filter condition

1. In Information Object Query Builder, choose Filters.
2. In Filters, choose New.
3. In Filter Conditions, in Filter by, do one of the following:
   - Select a column from the drop-down list. The drop-down list contains the non-aggregate columns that you defined in Columns. To create a filter for an aggregate column, use Having.
   - Type an expression.
   - Use the expression builder to create an expression.

4 Select the comparison test, or operator, to apply to the selected column or expression. Depending on the operator you select, Filter Conditions displays one or two additional fields, or a completed filter condition.

5 If you selected an operator that requires a comparison value, specify the value in one of the following ways:

- Type the value or expression.
- If you selected a column in Filter by, choose Select Value to select from a list of values. Figure 2-11 shows the selection of Boston from a list of possible sales office values.

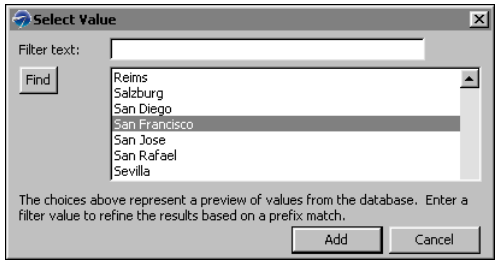


Figure 2-11 Select Values showing the values in the selected column

- Select a parameter or column from the drop-down list. You create parameters in Parameters.

- Open the expression builder to create an expression.

Figure 2-12 shows the completed filter condition.

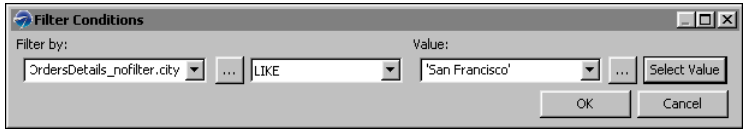


Figure 2-12 Filter Conditions displaying a completed filter condition

Choose OK. The filter condition appears in Filters as shown in Figure 2-13.

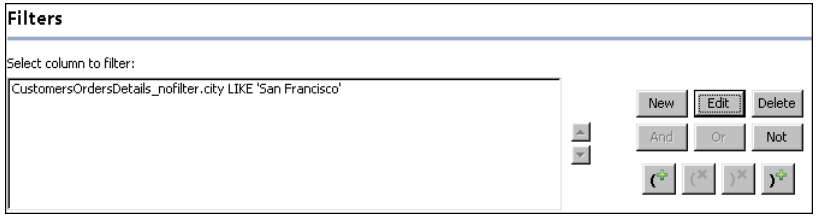


Figure 2-13 Examining a filter condition

6 Display the Actuate SQL query. Verify that the filter condition is appended to the WHERE clause and that the syntax is correct, for example:

```
WHERE SalesOffice LIKE 'Boston%'
```

How to create a filter condition using Actuate SQL

1 In Information Object Query Builder, choose Filters.

2 In Filters, complete the following tasks:

- Click in the text box.
- Type the filter condition using Actuate SQL, as shown in Figure 2-14. If a table or column identifier contains a special character, such as a space, enclose the identifier in double quotation marks ("). The filter condition is appended using AND operator, and is evaluated after other conditions.

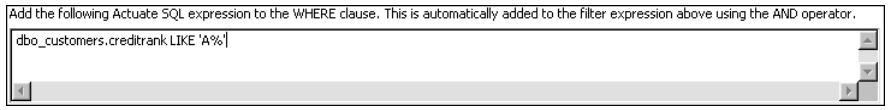


Figure 2-14 Using Actuate SQL to create a filter condition

Selecting multiple values for a filter condition

So far, the filter examples specify one comparison value. Sometimes you need to view more data, for example, sales details for several sales offices, not for only one office. To select more than one comparison value, select the IN operator, choose Select Values, then select the values. To select multiple values, press Ctrl as you select each value. To select contiguous values, select the first value, press Shift, and select the last value. This action selects the first and last values and all the values in between.

Figure 2-15 shows the selection of London and Paris from a list of sales office values.

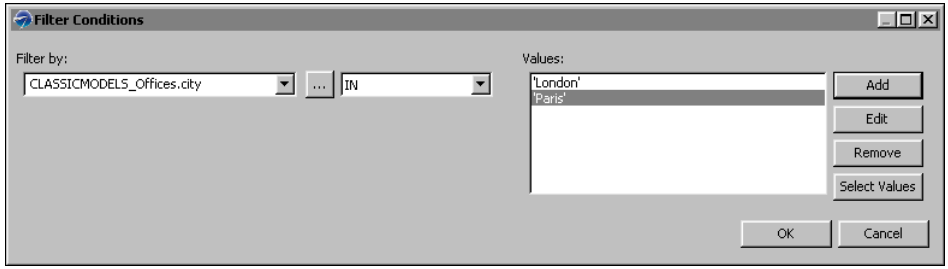


Figure 2-15 Filter Conditions showing the selection of multiple comparison values

Excluding data

You use comparison operators, such as = (equal to), > (greater than), or < (less than), to evaluate the filter condition to determine which data to include. Sometimes it is more efficient to specify a condition that excludes a small set of data. For example, you need sales data for all countries except USA. Instead of selecting all the available countries and listing them in the filter condition, simply use the NOT LIKE operator. Similarly, use NOT BETWEEN to exclude data in a specific range, and <> (not equal to) to exclude data that equals a particular value.

For example, the following filter condition excludes orders with amounts between 1000 and 5000:

```
OrderAmount NOT BETWEEN 1000 AND 5000
```

The filter condition in the next example excludes products with codes that start with MS:

```
ProductCode NOT LIKE 'MS%'
```

Filtering empty or blank values

Sometimes, rows display nothing for a particular column. For example, suppose a customer database table contains an e-mail field. Some customers, however, do not supply an e-mail address. In this case, the e-mail field might contain an empty value or a blank value. An empty value, also called a null value, means no value is supplied. A blank value is entered as '' (two single quotes without spaces) in the database table field. Blank values apply to string fields only. Null values apply to all data types.

You can create a filter to exclude data rows where a particular column has null or blank values. You use different operators to filter null and blank values.

When filtering to exclude null values, use the IS NOT NULL operator. If you want to view only rows that have null values in a particular column, use IS NULL. For example, the following filter condition excludes customer data where the e-mail column contains null values:

```
email IS NOT NULL
```

The following filter condition displays only rows where the e-mail column contains null values:

```
email IS NULL
```

When filtering blank values, use the NOT LIKE operator with '' (two single quotes without spaces) as the operand. For example, to exclude rows with blank values in an e-mail column, specify the following filter condition:

```
email NOT LIKE ''
```

Conversely, to display only rows where the e-mail column contains blank values, create the following condition:

```
email LIKE ''
```

In a report, you cannot distinguish between an empty value and a blank value in a string column. Both appear as missing values. If you want to filter all missing values whether they are null or blank, specify both filter conditions as follows:

```
email IS NOT NULL AND email NOT LIKE ''
```

Specifying a date as a comparison value

When you create a filter condition that compares the date-and-time values in a column to a specific date, the date value you supply must be in the following format regardless of your locale:

```
TIMESTAMP '2012-04-01 12:34:56'
```

Do not use locale-dependent formats such as 4/1/2012.

Specifying a number as a comparison value

When you create a filter condition that compares the numeric values in a column to a specific number, use . (period) as the decimal separator regardless of your locale, for example:

```
123456.78
```

Do not use , (comma).

Comparing to a string pattern

For a column that contains string data, you can create a filter condition that compares each value to a string pattern instead of to a specific value. For example, to display only customers whose names start with M, use the LIKE operator and specify the string pattern, M%, as shown in the following filter condition:

```
Customer LIKE 'M%'
```

You can also use the % character to ensure that the string pattern in the filter condition matches the string in the column even if the string in the column has trailing spaces. For example, use the filter condition:

```
Country LIKE 'USA%'
```

instead of the filter condition:

```
Country = 'USA'
```

The filter condition Country LIKE 'USA%' matches the following values:

```
'USA'
'USA   '
'USA     '
```

The filter condition Country = 'USA' matches only one value:

```
'USA'
```

You can use the following special characters in a string pattern:

- % matches zero or more characters. For example, %ace% matches any value that contains the string ace, such as Ace Corporation, Facebook, Kennedy Space Center, and MySpace.
- _ matches exactly one character. For example, t_n matches tan, ten, tin, and ton. It does not match teen or tn.

To match the percent sign (%) or the underscore character (_) in a string, precede those characters with a backslash character (\). For example, to match S_10, use the following string pattern:

```
S\_10
```

To match 50%, use the following string pattern:

```
50\%
```

Comparing to a value in another column

Use a filter condition to compare the values in one column with the values in another column. For example, in a report that displays products, sale prices, and MSRP (Manufacturer Suggested Retail Price), you can create a filter condition to compare the sale price and MSRP of each product, and display only rows where the sale price is greater than the MSRP.

How to compare to a value in another column

1. In Information Object Query Builder, choose Filters.
2. In Filters, choose New.
3. In Filter Conditions, in Filter by, select a column from the drop-down list.
4. Select the comparison test, or operator, to apply to the selected column.
5. In Value, select a column from the drop-down list. Figure 2-16 shows an example of a filter condition that compares the values in the priceEach column with the values in the MSRP column. Choose OK.

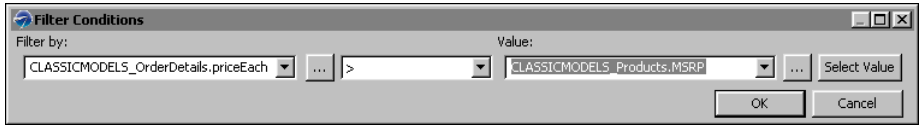


Figure 2-16 Comparing the values in priceEach with the values in MSRP

Using an expression in a filter condition

An expression is any combination of Actuate SQL constants, operators, functions, and information object columns. A filter condition supports using an expression in Filter by, Value, or both. To create the expression, use the expression builder.

For example, in an information object query that returns customer and order data, you want to see which orders shipped less than three days before the customer required them. You can use the DATEDIFF function to calculate the difference between the ship date and the required date:

```
DATEDIFF('d', shippedDate, requiredDate) < 3
```

Figure 2-17 shows this condition in Filter Conditions.

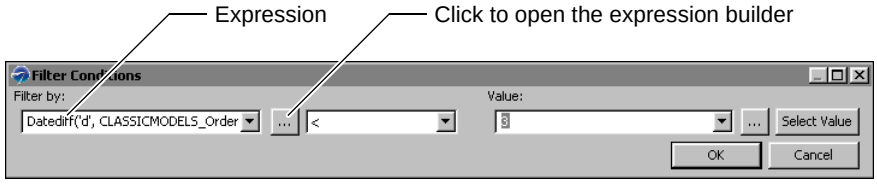


Figure 2-17 Filter Conditions with expression in Filter by

In an information object query that returns order data, you want to see which orders were placed today. You can use the CURRENT_DATE function to return today's date:

```
orderDate = CURRENT_DATE( )
```

Figure 2-18 shows this condition in Filter Conditions.

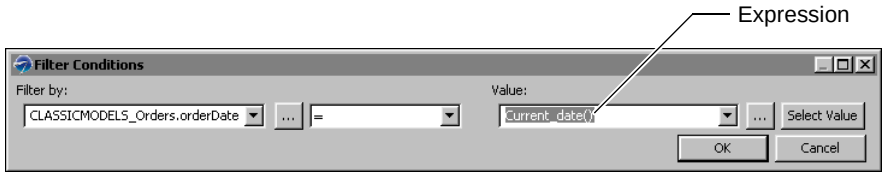


Figure 2-18 Filter Conditions with expression in Value

In an information object query that returns employee data, you want the information object query to return only data for the user who is currently logged in to the Encyclopedia volume. Use the LEFT function and the concatenation

operator (||) to construct the employee's user name, and the CURRENT_USER function to return the name of the user who is currently logged in:

```
LEFT(firstName, 1) || lastName = CURRENT_USER( )
```

Figure 2-19 shows this condition in Filter Conditions.

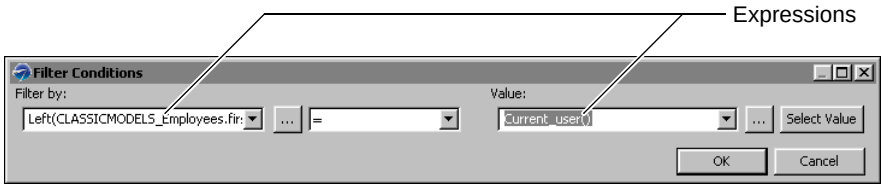


Figure 2-19 Filter Conditions with expressions in Filter by and Value

Creating multiple filter conditions

When you create a filter, you can define one or more filter conditions. Each condition you add narrows the scope of data further. For example, you can create a filter that returns rows where the customer's credit rank is either A or B and whose open orders total between $250,000 and $500,000. Each condition adds complexity to the filter. Design and test filters with multiple conditions carefully. If you create too many filter conditions, the information object query returns no data.

Adding a condition

You use Filters, shown in Figure 2-13, to create one or more filter conditions. To create a filter condition, you choose New and complete the Filter Conditions dialog, shown in Figure 2-12. When you create multiple filter conditions, Information Object Query Builder precedes the second and subsequent conditions with the logical operator AND, for example:

```
SalesOffice LIKE 'San Francisco%' AND
   ProductLine LIKE 'Vintage Cars%'
```

This filter returns only data rows that meet both conditions. Sometimes, you want to create a filter to return data rows when either condition is true, or you want to create a more complex filter. To accomplish either task, use the buttons on the right side of Filters, shown in Figure 2-20.

If you create more than two filter conditions and use different logical operators, use the parentheses buttons to group conditions to determine the order in which they are evaluated. Display the query output to verify the results.

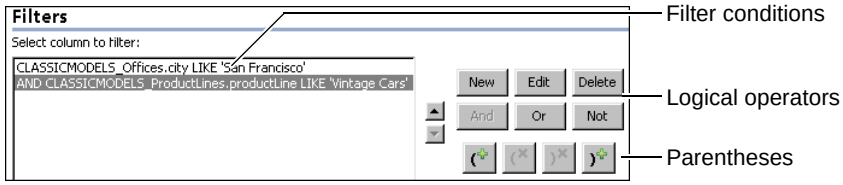


Figure 2-20 Working with multiple conditions in Filters

Selecting a logical operator

As you add each filter condition, the logical operator AND is inserted between filter conditions. You can change the operator to OR. The AND operator means both filter conditions must be true for a data row to be included in the data set. The OR operator means only one condition has to be true for a data row to be included. You can also add the NOT operator to either the AND or OR operators to exclude a small set of data. For example, the following filter conditions return only sales data for classic car items sold by the Boston office:

```
SalesOffice LIKE 'Boston%' AND ProductLine LIKE 'Classic Cars%'
```

The following filter conditions return all sales data for the San Francisco and Boston offices:

```
SalesOffice LIKE 'San Francisco%' OR SalesOffice LIKE 'Boston%'
```

The following filter conditions return sales data for all product lines, except classic cars, sold by the San Francisco office:

```
SalesOffice LIKE 'San Francisco%' AND
   NOT (Product Line LIKE 'Classic Cars%')
```

Specifying the evaluation order

Information Object Query Builder evaluates filter conditions in the order in which they appear. You can change the order by selecting a filter condition in Filters, shown in Figure 2-20, and moving it up or down using the arrow buttons. Filter conditions that you type in the Actuate SQL text box, shown in Figure 2-14, are preceded by AND and are evaluated last.

If you define more than two conditions, you can use parentheses to group conditions. For example, A AND B OR C is evaluated in that order, so A and B must be true or C must be true for a data row to be included. In A AND (B OR C), B OR C is evaluated first, so A must be true and B or C must be true for a data row to be included.

The following examples illustrate the difference a pair of parentheses makes.

The following filter contains three ungrouped conditions:

```
Country IN ('Australia', 'France', 'USA') AND
   SalesRepNumber = 1370 OR CreditLimit >= 100000
```

Figure 2-21 shows the first 10 data rows returned by the query. Although the filter specifies the countries Australia, France, and USA and sales rep 1370, the data rows display data for other countries and sales reps. Without any grouped conditions, the filter includes rows that meet either conditions 1 and 2 or just condition 3.

"Customer name"	Country	"Sales rep number"	"Credit limit"
Atelier graphique	France	1370	21000.0
Australian Collectors, Co.	Australia	1611	117300.0
La Rochelle Gifts	France	1370	118200.0
Mini Gifts Distributors Ltd.	USA	1165	210500.0
Land of Toys Inc.	USA	1323	114900.0
Euro+ Shopping Channel	Spain	1370	227600.0
Saveley & Henriot, Co.	France	1337	123900.0
Dragon Souveniers, Ltd.	Singapore	1621	103800.0
Muscle Machine Inc	USA	1286	138500.0
Diecast Classics Inc.	USA	1216	100600.0

Figure 2-21 Results of a complex filter without parentheses grouping

The following filter contains the same three conditions, but this time the second and third conditions are grouped:

```
Country IN ('Australia', 'France', 'USA') AND
   (SalesRepNumber = 1370 OR CreditLimit >= 100000)
```

Figure 2-22 shows the first 10 data rows returned by the query. The Country IN ('Australia', 'France', 'USA') condition must be true, then either the SalesRepNumber = 1370 condition or the CreditLimit >= 100000 condition is true.

"Customer name"	Country	"Sales rep number"	"Credit limit"
Atelier graphique	France	1370	21000.0
Australian Collectors, Co.	Australia	1611	117300.0
La Rochelle Gifts	France	1370	118200.0
Mini Gifts Distributors Ltd.	USA	1165	210500.0
Land of Toys Inc.	USA	1323	114900.0
Saveley & Henriot, Co.	France	1337	123900.0
Muscle Machine Inc	USA	1286	138500.0
Diecast Classics Inc.	USA	1216	100600.0
Daedalus Designs Imports	France	1370	82900.0
Mini Caravy	France	1370	53800.0

Figure 2-22 Results of a complex filter with parentheses grouping

Changing a condition

You can change any of the conditions in Filters.

How to change a filter condition

1. In Filters, shown in Figure 2-20, select the filter condition. Choose Edit.
2. In Filter Conditions, shown in Figure 2-12, modify the condition by changing the values in Filter by, Operator, or Value. Choose OK.

How to delete a filter condition

To delete a filter condition, in Filters, select the condition. Then, choose Delete. Verify that the remaining filter conditions still make sense.

Providing values for a filter

To prompt a report user for a filter value, use a parameter or a dynamic filter.

Prompting for a single value

Use an Actuate SQL parameter to prompt a report user for a single filter value. An Actuate SQL parameter enables the report user to restrict the data rows returned by the information object query. For example, for an information object query that returns sales data by sales office, you can create an Actuate SQL parameter called param_SalesOffice that prompts a report user to select an office, and then use the parameter in a filter condition.

The WHERE clause is modified as follows:

```
WHERE SalesOffice LIKE :param_SalesOffice
```

To create an Actuate SQL parameter for a data set and define prompt properties for the parameter, use Parameters. Prompt properties include the parameter's default value, a list of values for the user to choose from, and whether the parameter is required or optional. Parameters appear in the Value drop-down list in Filter Conditions.

The parameter name is preceded by a : (colon), as shown in Figure 2-23.

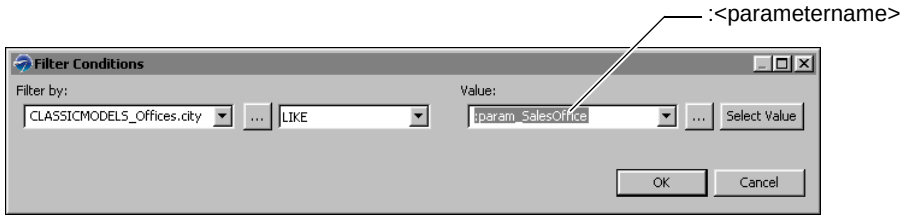


Figure 2-23 Filter Conditions with a parameter in the Value field

Prompting for multiple values

Use a dynamic filter to prompt a report user for multiple filter values. A dynamic filter can accept an operator and one or more operands. For example, to prompt a report user for the location of one or more sales offices, create a dynamic filter for the SalesOffice column. A user running the report selects the IN operator and the operands Boston and NYC to display data for the Boston and New York sales offices.

Using a predefined filter in a query

A predefined filter restricts the data returned by an information object. To create a filter on an information object, a data modeler sets the Filter property for a column to Predefined. Information Object Query Builder creates a dynamic filter for each column in an IO that has the Filter property set to Predefined. For more

information about creating a predefined filter, see *Designing BIRT Information Objects*.

About dynamic filters

A dynamic filter restricts the data returned by a query based on an information object. Modifications to the WHERE or HAVING clause occur at report run time, based on the values that a report user provides for the dynamic filter. A dynamic filter accepts an operator and zero, one, two, or more operands.

Table 2-3 lists operators available for a dynamic filter and the number of operands required by each operator.

Table 2-3 Operators and number of operands required for a dynamic filter

Operators	Number of operands required
Is Not Null, Is Null	0
Equal to, Greater than, Greater than or Equal, Less than, Less than or Equal, Like, Match, Not Equal to, Not Like, Not Match	1
Between, Not Between	2
In, Not In	1 or more

For example, you create a dynamic filter on the customerName column in CustomersOrdersDetails.iob. Responding to a report prompt, a report user selects the LIKE operator and the Atelier graphique operand. The query that returns the report data set is modified to include the following clause:

```
WHERE CustomerOrdersDetails.customerName LIKE 'Atelier
  graphique%'
```

Figure 2-24 shows a preview of the filter in Parameters.

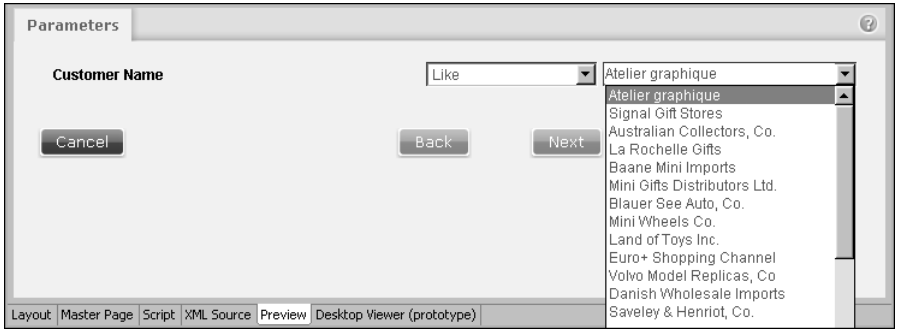


Figure 2-24 Selecting a customer name to filter a set of customer information

Prompt properties control how the dynamic filter prompt appears to a report user on the Parameters page. Prompt properties include the display control type, for example text box or combo box and a list of values. If a dynamic filter is auto-generated from a predefined filter, the dynamic filter inherits the prompt properties of the predefined filter. By default, a report user is not required to provide a value for a dynamic filter that is automatically created from a predefined filter.

Unless the display control type for a dynamic filter is text box, you must create a list of values for the dynamic filter. You create a list of values by typing the values or by creating an Actuate SQL query that retrieves the values.

The query must meet the following requirements:

- The query must retrieve one or two columns from an information object or map, for example:

  ```
  SELECT DISTINCT cust ID, CAST(customerName AS VARCHAR)
  FROM "IO Designs/MyProject/InformationObjects/
    MyInformationObject.iob" ORDER BY 2
  ```

  The first column, for example custID, contains the filter values. The second column, for example customerName, contains the values displayed to the report user. Use a relative path to reference the information object or map. The path must be relative to the resource root.

- The query must not contain a WITH clause.

Creating a dynamic filter

To create a dynamic filter for a data set, use the data set editor. For example, to filter a data set of customer information by customer name, edit the data set, adding a dynamic filter that uses only the customerName column, as shown in Figure 2-24.

How to create a dynamic filter

1. In Edit Data Set, choose Filters.
2. In Filters, choose New.
3. In New Filter Condition, shown in Figure 2-25, do the following tasks:
   1. Select Dynamic.
   2. In Column, choose the appropriate column name from the drop-down list.
   3. In Filter Parameter, choose <New Dynamic Filter Parameter...>.

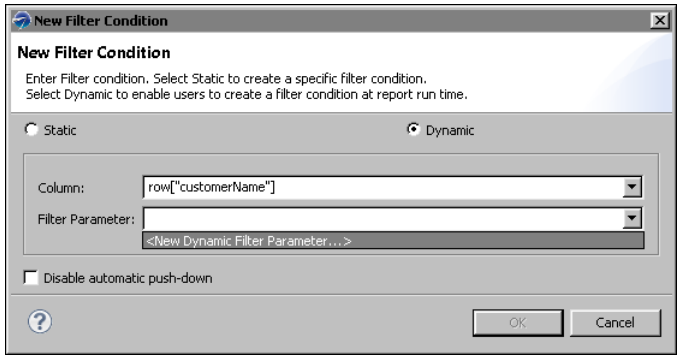


Figure 2-25 Creating a new dynamic filter

4 In New Dynamic Filter Parameter, do the following tasks:

1. In Name, edit the name of the filter.
2. In Prompt text, type the filter prompt.
3. In Data type, verify that the displayed data type matches the data type of the column to which the filter applies. If necessary, choose a different type.
4. In Display type, select a control type the report uses to prompt for values.

   For example, the selections shown in Figure 2-26 name a filter called customer name filter. The filter displays the text Customer Name and provides a combo/list box control that displays values of data type string.

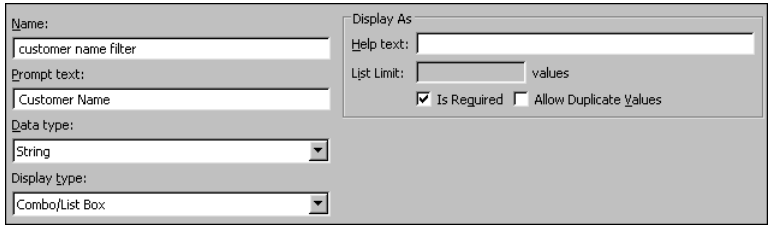


Figure 2-26 Providing basic definitions for a dynamic filter

5. In Dynamic Filter Condition, select the operators to display on the Parameter page, and, optionally, the default operator.

   For example, the selections shown in Figure 2-27 specify that the customer name filter displays:

   - Values from the customerName column
   - All operators that appear in the right pane
   - The operator Like as the default operator

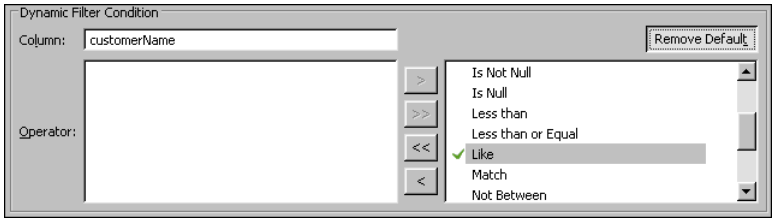


Figure 2-27 Defining the source column, available operators, and a default operator for a dynamic filter

6 In Selection list values, choose Static or Dynamic.

1 If you choose Static, provide values for the static list.

2 If you chose Dynamic, for Data Set, select a data set from the list, or choose Create New.

3 To create a new data set of values that appear in the filter prompt, create a query using the graphical or textual data set editor.

4 In Select value column, select the column from which to display values for the report parameter.

5 Optionally, in Sort, choose a column and order to sort the values.

For example, the selections shown in Figure 2-28 specify a sorted list of customer name values from which a report users chooses. The customer name Atelier graphique appears as the default customer name.

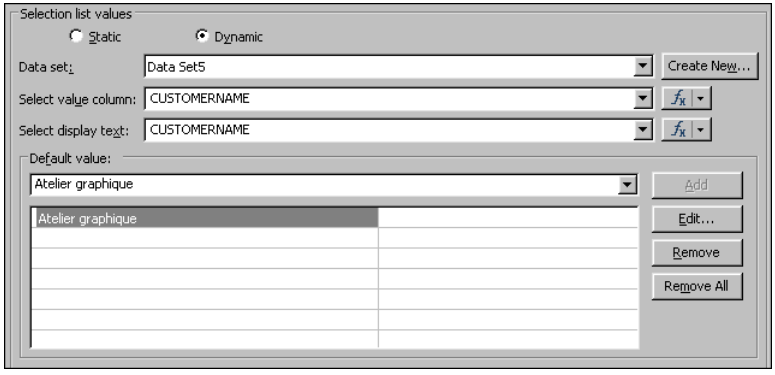


Figure 2-28 Defining selection list values for a dynamic filter

In New Dynamic Filter Parameter, choose OK.

In New Filter Condition, choose OK. The customer name filter appears in Filters for the customer info data set, as shown in Figure 2-29.

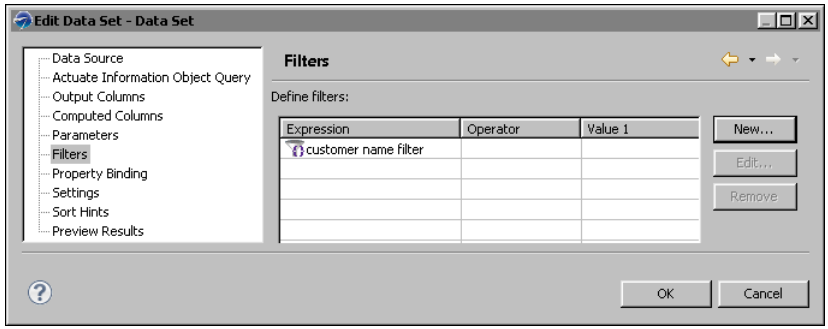


Figure 2-29 Examining a dynamic filter in the data set editor

In Data Explorer, the customer info and customer name data sets appear in Data Sets, and the customer name filter appears in Report Parameters, as shown in Figure 2-30.

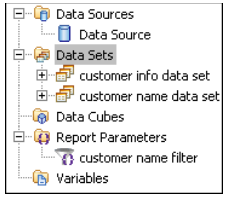


Figure 2-30 Examining a new dynamic filter and its data set of parameter values

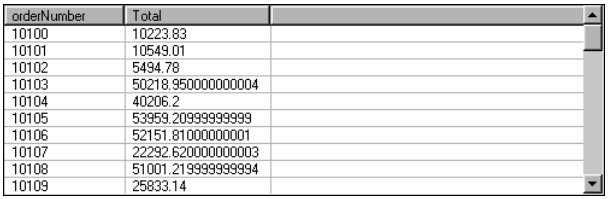
Filtering on an aggregate column

If an information object query includes a GROUP BY clause, you can restrict the data rows the query returns by adding a HAVING clause. The HAVING clause places a filter condition on one or more aggregate columns. An aggregate column is a computed column that uses an aggregate function such as AVG, COUNT, MAX, MIN, or SUM, for example SUM(quantityOrdered * priceEach).

For example, the following query returns order numbers and order totals. The Total column is an aggregate column. The data is grouped by order number and no filter condition is placed on the Total column.

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
```

Figure 2-31 shows the first 10 data rows returned by this information object query.



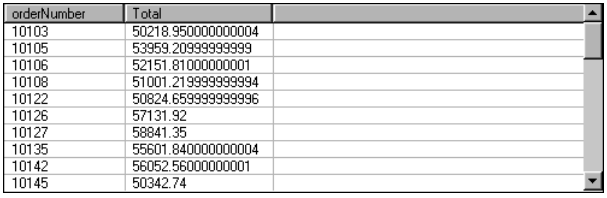
orderNumber	Total
10100	10223.83
10101	10549.01
10102	5494.78
10103	50218.950000000004
10104	40206.2
10105	53959.20999999999
10106	52151.81000000001
10107	22292.620000000003
10108	51001.219999999994
10109	25833.14

Figure 2-31 Data rows returned by query with GROUP BY clause

You can add a HAVING clause to this information object query to place a filter condition on the Total column. The following information object query returns only rows for which the order total is greater than or equal to 50000:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
HAVING SUM(quantityOrdered * priceEach) >= 50000
```

Figure 2-32 shows the first 10 data rows returned by this information object query.



orderNumber	Total
10103	50218.950000000004
10105	53959.20999999999
10106	52151.81000000001
10108	51001.219999999994
10122	50824.659999999996
10126	57131.92
10127	58841.35
10135	55601.840000000004
10142	56052.56000000001
10145	50342.74

Figure 2-32 Data rows returned by query with GROUP BY and HAVING clauses

The procedures for creating filter conditions for aggregate columns are identical to the procedures for creating filter conditions for other columns, except that you use Having instead of Filters. A query evaluates filter conditions that you create using Filters before it evaluates filter conditions that you create using Having. In other words, the query applies filter conditions in the WHERE clause before it applies filter conditions in the HAVING clause.

Sorting a data set

Sorting a data set orders the rows in a data set. A SQL query that includes an Order By clause returns a data set sorted using values in one or more columns included in the Order By clause. Sort supports sorting a data set by creating an Order By clause in a SQL query that you create or customize.

For example, consider a data set, containing customer order data, that includes a productName column. To sort the data set rows by product name in ascending order, add the following clause to the SQL query that returns the data set:

```
Order BY productName ASC
```

How to sort a data set

To sort a data set based on an information object, use Information Object Query Builder to add an Order By clause to the SQL query that returns the data set.

1. In Information Object Query Builder, choose Sort.
2. Expand Output columns that appear listed in Available.
3. Using the right arrow, move a column name to Selected.
4. Optionally, to change the sort order:
   1. Select Order. The Order type Asc, sorting rows in ascending order, appears by default.
   2. To sort rows in descending order, select the order type. Then, choose Desc.
5. Optionally, to sort by multiple columns, repeat step 3 to select additional column names.

The example in Figure 2-33 shows selections that add an Order By clause using the column productName to a SQL query.

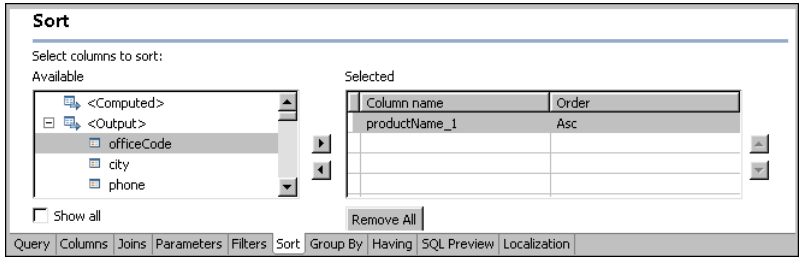


Figure 2-33 Adding an Order By clause to a SQL query

Grouping data

A GROUP BY clause groups data by column value. For example, consider the following query:

```
SELECT orderNumber
FROM OrderDetails
```

The first 10 data rows returned by this query are as follows:

```
orderNumber
10100
10100
10100
10100
10101
10101
10101
10101
10102
10102
```

Each order number appears more than once. For example, order number 10100 appears four times. If you add a GROUP BY clause to the query, you can group the data by order number so that each order number appears only once:

```
SELECT orderNumber
FROM OrderDetails
GROUP BY orderNumber
```

The first 10 data rows returned by this query are as follows:

```
orderNumber
10100
10101
10102
10103
10104
10105
10106
10107
10108
10109
```

Typically, you use a GROUP BY clause to perform an aggregation. For example, the following query returns order numbers and order totals. The Total column is an aggregate column. An aggregate column is a computed column that uses an aggregate function such as AVG, COUNT, MAX, MIN, or SUM.

```
SELECT orderNumber, (SUM(quantityOrdered*priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
```

Figure 2-31 shows the first 10 data rows returned by the information object query. The data is grouped by order number and the total for each order appears.

Creating a GROUP BY clause

By default, Information Object Query Builder creates a GROUP BY clause automatically. If you prefer, you can create a GROUP BY clause manually.

Creating a GROUP BY clause automatically

When an information object query's SELECT clause includes an aggregate column and one or more non-aggregate columns, the non-aggregate columns must appear in the GROUP BY clause. If the non-aggregate columns do not appear in the GROUP BY clause, Information Object Query Builder displays an error message. For example, consider the following information object query:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
```

When you attempt to compile the information object query, the error message shown in Figure 2-34 appears in the Problems view.

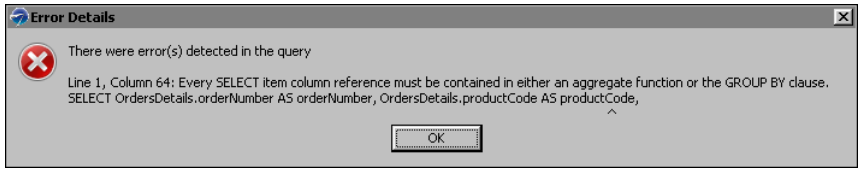


Figure 2-34 Information object query requires a GROUP BY clause

To avoid this problem, Information Object Query Builder automatically creates a GROUP BY clause:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
```

If more than one column appears in the GROUP BY clause, you can change the order of the columns using the up and down arrows in Group By.

Creating a GROUP BY clause manually

If automatic grouping does not generate the desired SQL query, create the GROUP BY clause manually. Create the GROUP BY clause manually if you want to group on a column that does not appear in the SELECT clause, for example:

```
SELECT (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
```

How to create a GROUP BY clause manually

1. In Information Object Query Builder, choose Group By.
2. In Group By, deselect Use Automatic Grouping.

3 In Available, expand the Computed and Output nodes to view the available columns.

   By default, Information Object Query Builder displays only output columns and non-aggregate computed fields. To group on a column that is not an output column, choose Show all.

4 In Available, select the appropriate column. Then, choose Select. The column name appears in Selected, as shown in Figure 2-35.

5 Repeat the previous step for each GROUP BY column.

6 To change the order of the GROUP BY columns, select a column in Selected, and use the up or down arrow.

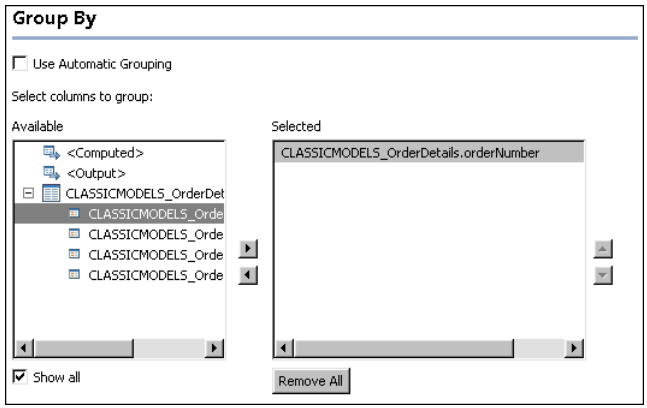


Figure 2-35 Selecting a GROUP BY column

Removing a column from the GROUP BY clause

By default, Information Object Query Builder removes GROUP BY columns automatically. If you disable automatic grouping, you must remove GROUP BY columns manually.

Removing a GROUP BY column automatically

Information Object Query Builder automatically removes a column from the GROUP BY clause when you complete the following actions:

- You remove the column from the SELECT clause. For example, consider the following information object query:

```
SELECT orderNumber, productCode, (SUM(quantityOrdered *
   priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber, productCode
```

- You remove the productCode column from the SELECT clause. Information Object Query Builder automatically removes productCode from the GROUP BY clause:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
```

- You manually add a column to the GROUP BY clause that does not appear in the SELECT clause and then enable automatic grouping. For example, consider the following information object query:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber, productCode
```

- The productCode column appears in the GROUP BY clause but not in the SELECT clause. You enable automatic grouping. Information Object Query Builder automatically removes productCode from the GROUP BY clause:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
```

Information Object Query Builder automatically removes the GROUP BY clause when:

- You remove all aggregate columns from the SELECT clause. For example, consider the following information object query:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
```

- You remove the aggregate column SUM(quantityOrdered * priceEach) from the SELECT clause. Information Object Query Builder automatically removes the GROUP BY clause:

```
SELECT orderNumber
FROM OrderDetails
```

- You remove all non-aggregate columns from the SELECT clause. For example, consider the following information object query:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
```

- You remove the orderNumber column from the SELECT clause. Information Object Query Builder automatically removes the GROUP BY clause:

```
SELECT (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
```

Removing a GROUP BY column manually

If you disable automatic grouping, you must manually remove columns from the GROUP BY clause.

How to remove a GROUP BY column manually

1. In Information Object Query Builder, choose Group By.
2. In Group By, complete one of the following tasks:

   - In Selected, select a column name. Then, choose Remove.
   - To remove all Group By columns, choose Remove All.

Defining parameters

An Actuate SQL parameter is a variable that is used in an information object query. When a user runs the report, the user provides a value for this variable on the Parameters page.

For example, the following Actuate SQL query uses the parameters lastname and firstname in the WHERE clause:

```
WITH ( lastname VARCHAR, firstname VARCHAR )
SELECT lname, fname, address, city, state, zip
FROM customerstable
WHERE (lname = :lastname) AND (fname = :firstname)
```

If an Actuate SQL query defines a parameter in a WITH clause but does not use the parameter, the query does not return any rows if no value is provided for the parameter when the report runs. For example, the following query does not return any rows if no values are provided for the lastname and firstname parameters when the report runs:

```
WITH ( lastname VARCHAR, firstname VARCHAR )
SELECT lname, fname, address, city, state, zip
FROM customerstable
```

How to define a parameter

1. In Information Object Query Builder, choose Parameters.
2. In Parameters, select the top, empty line, and complete the following tasks:
   - In Parameter, type the name of the parameter. If a parameter name contains a special character or a space, enclose the name in double quotation marks.
   - In Data type, select a data type from the drop-down list.

- In Default value, type the default value:
  - If Default value is a string, enclose the string in single quotation marks, as shown in the following example:

    ```
    'New York City'
    ```

  - If Default value is a timestamp, it must be of the following form:

    ```
    TIMESTAMP '2012-02-03 12:11:10'
    ```

  - If Default value is a number, use a period (.) as the decimal separator, as shown in the following example:

    ```
    123456.78
    ```

  NULL is not a valid parameter value.
- To change the order of the parameters, use the up or down arrow.
- To use Prompt editor to specify the parameter's prompt properties, choose Prompt editor, as shown in Figure 2-36.

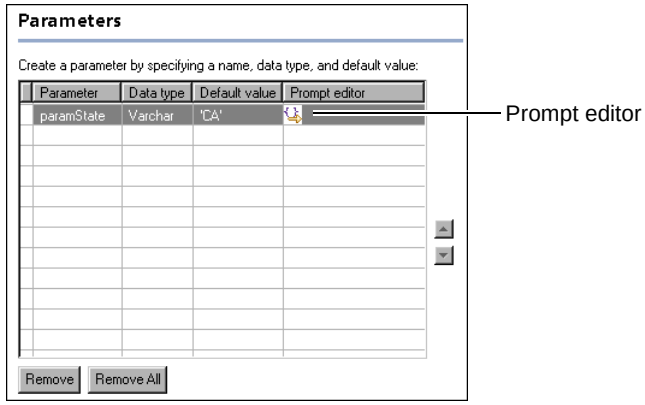


Figure 2-36 Choosing Prompt editor to specify a parameter's prompt properties

- To define other parameter properties, such as display name, select the parameter in Parameters, and define the properties in Properties.

How to delete a parameter

1. To delete a parameter, in Information Object Query Builder, choose Parameters.
2. In Parameters, complete one of the following tasks:
   - To delete an individual parameter, select the parameter, and choose Remove.
   - To delete all parameters, choose Remove All.

Specifying prompt properties for a parameter

Use Prompt editor to specify properties for a prompt that requests user entry of a parameter value. Prompt properties include display control type, list of values, and default value. You can specify the parameter values and, if desired, a corresponding set of display values that report users choose from. Create a list of values by typing the values or by typing an Actuate SQL query that retrieves the values. Use an absolute path to query a published information object.

The query must meet the following requirements:

- The query must retrieve one or two columns from an information object or map, as shown in the following example from a local information object:

  ```
  SELECT DISTINCT custID, customName
  FROM "IO Designs/MyProject/Information Objects/
     MyInformationObject.iob"
  ORDER BY 2
  ```

  The first column contains the parameter values. The second column contains the values that are displayed to the report user. You must use a relative path to resource root to reference the information object or map. If the information object or map defines a parameter, you must provide a value for the parameter, as shown in the following example from a local information object:

  ```
  SELECT DISTINCT custID, customName
  FROM "IO Designs/MyProject/Information Objects/
     MyInformationObject.iob" ['CA']
  ORDER BY 2
  ```

- The first column's data type must match the parameter's data type.
- The query must not contain a WITH clause.

How to specify prompt properties for a parameter

1. Locate the appropriate parameter in Parameters, and choose Prompt editor.
2. In Prompt editor, in Show as, select the method of prompting the user, as shown in Figure 2-37. If you select a prompt type other than text box, you can specify a list of prompt values from which the user may choose.

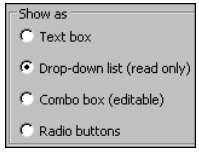


Figure 2-37 Selecting the prompt type

You can create a list of values by typing the parameter values and, optionally, the display names, as shown in Figure 2-38. If you do not provide display names, the parameter values are displayed to the user.

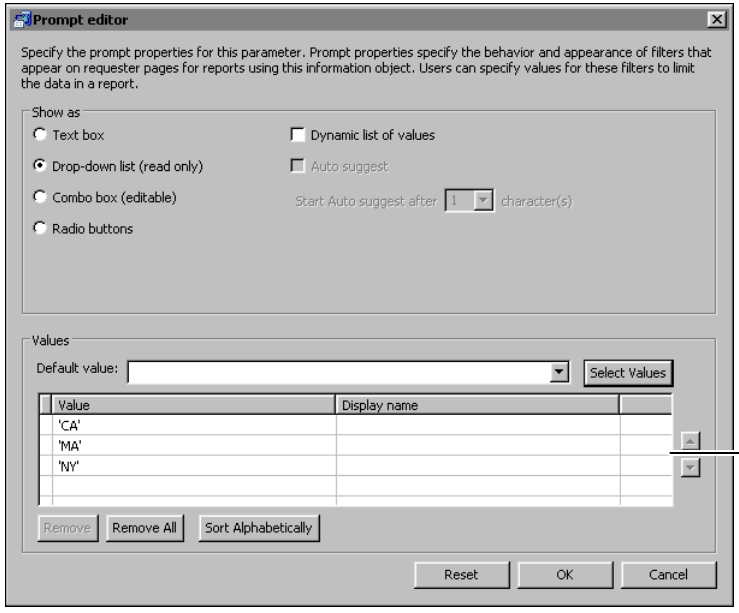


Figure 2-38 Typing a list of values and display names

You can create an Actuate SQL query that retrieves the parameter values or both the values and the corresponding display names. If the query has two columns, the values in the second column are used as the display names. To use a query to create the list of values, select Dynamic list of values, as shown in Figure 2-39, and type the query.

If you select Combo box (editable), Dynamic list of values, and Auto suggest, a drop-down list appears after the report user types the number of characters specified in Start Auto suggest after N character(s). The list contains values that begin with the characters the user typed. For example, if the user typed 'Atel and N=4, the list contains the value 'Atelier graphique'. In this case, the query that retrieves the values must select two columns, a value column and a display name column.

3 In Default value, specify the default value.

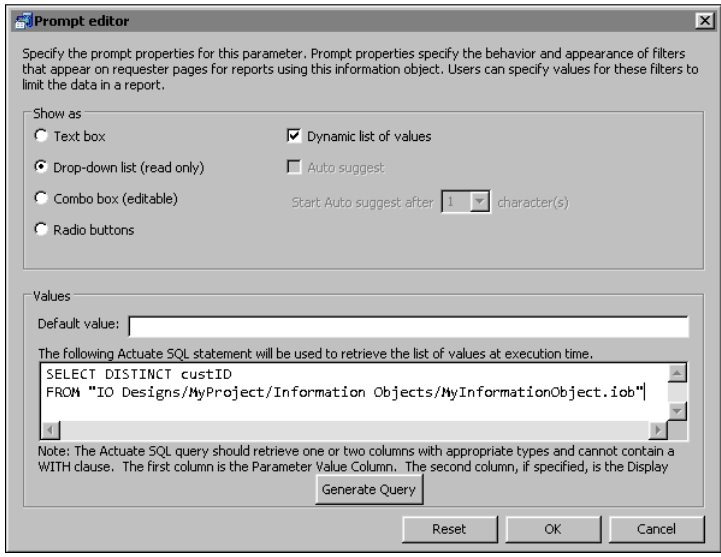


Figure 2-39 Specifying an Actuate SQL query to provide a dynamic list of values

4 You also can specify values for the following properties:

- Conceal value. Set to True or False to show or hide the parameter value.
- Do not prompt. Set to True or False to show or hide the parameter.
- Required. Set to True to require a value for the parameter.

When you finish specifying the property values for the prompt, choose OK.

Setting parameter properties

Table 2-4 lists parameter properties and provides a description of each property.

Table 2-4 Parameter properties

Parameter property	Can set?	Description
Conceal Value	Yes, in Prompt editor	Visibility of the value that the user provides for this parameter. To conceal the value, set to True. To display the value, set to False. This parameter property applies only to parameters with the varchar data type and the text box display type.
Data Type	Yes, on Parameters	Parameter's data type.

Table 2-4 Parameter properties

Parameter property	Can set?	Description
Default Value	Yes, in Prompt editor	Parameter's default value. If a parameter does not have a default value, and the Required property is set to False, the parameter takes one of the following values if the user does not provide a value: ■ 0 if the parameter is of type decimal, double, or integer ■ Empty string if the parameter is of the varchar data type ■ Current date and time if the parameter is of the timestamp data type
Description	Not used	Not used.
Display Control Type	Yes, in Prompt editor	Control type for this parameter. The options are text box, read-only drop-down list, editable drop-down list, or radio buttons.
Display Format	Not used	Not used.
Display Length	Not used	Not used.
Display Name	Yes	Prompt that appears on the Parameters page.
Do Not Prompt	Yes, in Prompt editor	Visibility of the parameter to the user. To hide this parameter, set to True. To display the parameter, set to False.
Heading	Not used	Not used.
Help Text	Not used	Not used.
Horizontal Alignment	Not used	Not used.
Name	Yes, on Parameters	Parameter name.
Parameter Mode	Yes	Setting for parameters that are in stored procedures and ODA data source queries to specify the input or output type of the parameter. The options are Input, Output, InputAndOutput, or ReturnValue. ReturnValue is used only for stored procedures and is equivalent to Output.
Required	Yes, in Prompt editor	Indicator of whether the parameter is required. To require a value for this parameter, set to True. Otherwise, set to False.
Size	Yes	The size of the parameter if the parameter data type is varchar. Otherwise, not used. Must be set if size is greater than 1300.

Setting a source parameter

A source parameter is a parameter that is defined in an information object from which you are building an information object query. If a query contains an information object with a parameter, Parameters has a Source parameter field.

You can set a source parameter to one of the following types of values:

- A single scalar value
- A local parameter in the information object query that you are creating

You cannot set a source parameter to a column reference, such as ORDERS.ORDERID, or an Actuate SQL expression.

To define a local parameter and set the value of the source parameter to the value of the local parameter in one step, drag the source parameter from Source parameter, and drop it in Parameter, as shown in Figure 2-40. To reset the value for a parameter to the default value, choose Reset values.

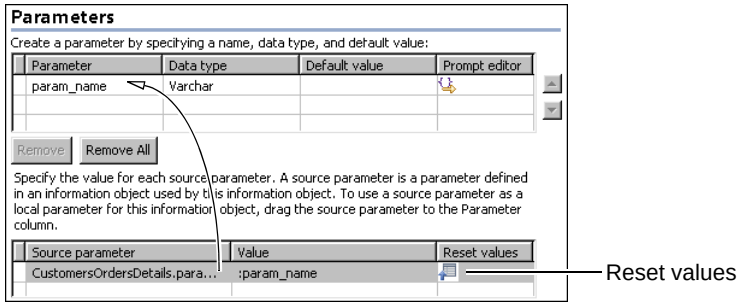


Figure 2-40 Setting the value of a source parameter to the value of a local parameter

When you set the value of a source parameter to the value of a local parameter, by dragging and dropping, the local parameter inherits the values of its prompt properties from the source parameter. The available prompt properties are Conceal Value, Default Value, Display Control Type, Do Not Prompt, and Required. If a local parameter is set to inherit its prompt property values from the source parameter, and prompt property values for the source parameter change, the changes propagate to the local parameter. For example, if the display control type for the source parameter changes from text box to read-only drop-down list, the display control type for the local parameter also changes from text box to read-only drop-down list.

If you change a prompt property value for a local parameter, it no longer inherits prompt property values from the source parameter. For example, if you change the display control type for a local parameter to editable drop-down list, and the display control type for its source parameter later changes to text box, the display control type for the local parameter remains editable drop-down list. To reinstate inheritance, choose Reset in Prompt editor for the local parameter. Choosing

Reset returns all prompt property values for the local parameter to inherited values. A reset local parameter inherits future changes made to prompt property values in the source parameter.

To set the value of a source parameter, use Parameters.

How to set the value of a source parameter

1. In Information Object Query Builder, choose Parameters.
2. In Parameters, complete the following tasks:
   - In Source parameter, select the appropriate parameter.
   - In Value, complete one of the following tasks:
     - Choose a parameter from the drop-down list. The drop-down list contains the local parameters for the information object query you are building.
     - Type a value, as shown in Figure 2-41:
       - If Value is a string, enclose the string in single quotation marks, as shown in the following example:

         ```
         'New York City'
         ```

       - If Value is a timestamp, it must be in the following form:

         ```
         TIMESTAMP '2001-02-03 12:11:10'
         ```

       - If Value is a number, use a period (.) as the decimal separator, as shown in the following example:

         ```
         123456.78
         ```

       - If Value is a parameter, precede the parameter name with a colon (:).

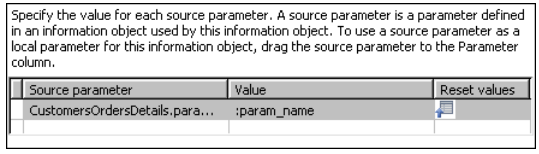


Figure 2-41 Providing a value for a source parameter

Using localized information objects

If the data modeler provides a localization properties file for a project, information object column properties, such as Heading, appear in the language the user selects when they log in to Information Console. The localization properties file appears as a shared resource in a project. Localization displays a

message indicating that a query is from a localized information object, as shown in Figure 2-42.

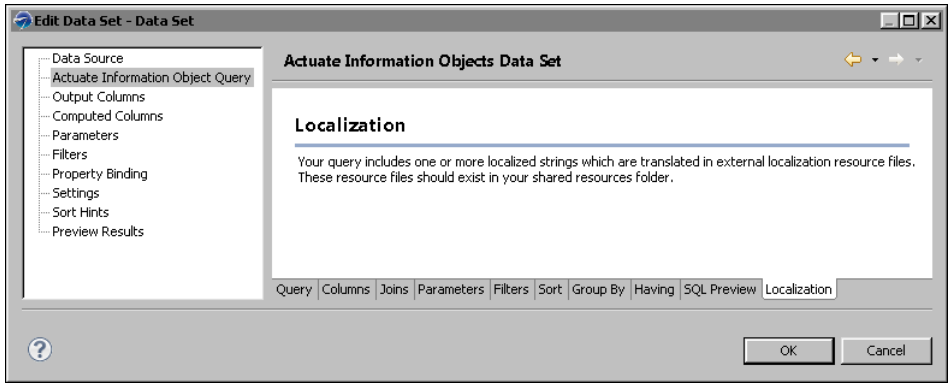


Figure 2-42 Determining whether a query is from a localized information object

Creating a textual information object query

Use the Actuate SQL text editor if either of the following conditions is true:

- Information Object Query Builder does not generate the desired Actuate SQL query, so you must edit the query. For example, a query that includes OR or UNION requires using the Actuate SQL text editor to edit the query.
- You can type or paste an Actuate SQL query and not use the graphical editor.

Saving a query in the Actuate SQL text editor, prevents further modifications to the query using the graphical editor.

To open the Actuate SQL text editor in Query, choose SQL editor, as shown in Figure 2-43.

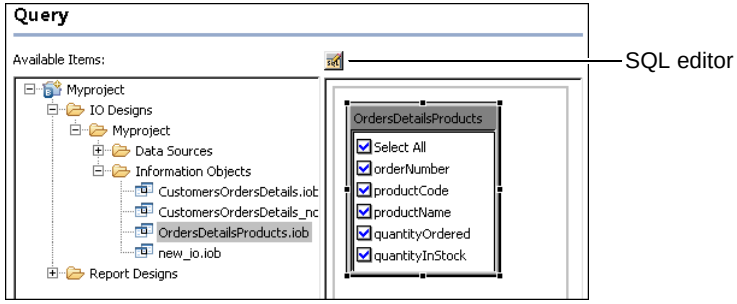


Figure 2-43 Opening the SQL text editor in Information Object Query Builder

The text editor appears in Query, as shown in Figure 2-44.

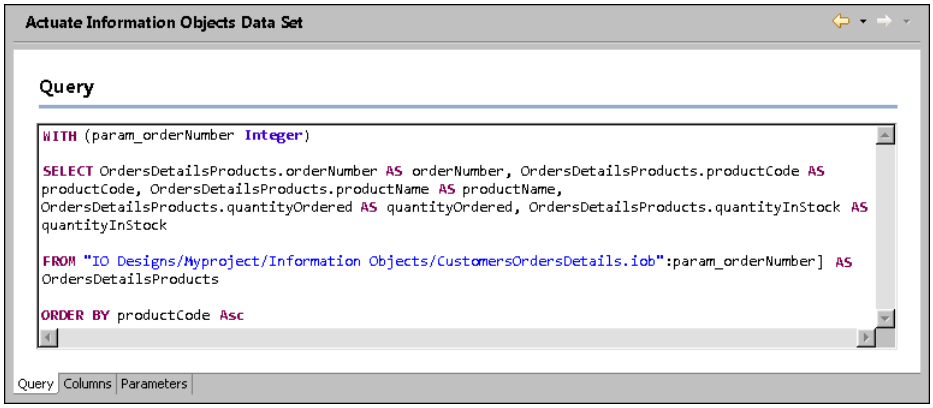


Figure 2-44 Editing an Actuate SQL query using the SQL text editor

You edit the query text that appears in Query. When editing a query in the SQL text editor, do not use table and column aliases that are identical except for case. For example, do not use both status and STATUS as column aliases.

Use paths that are relative to the resource root in the textual query editor.

Displaying output columns

In Columns, to display the query's output columns and the data type for each column, choose Describe Query, as shown in Figure 2-45.

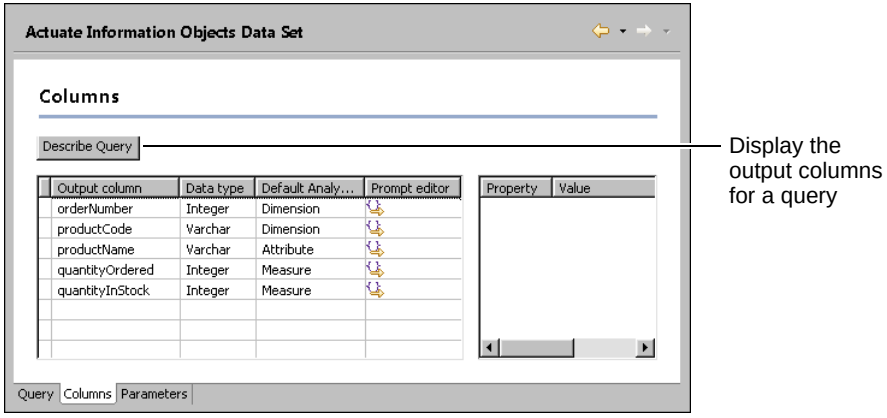


Figure 2-45 Using Describe Query to display output columns for a query

To specify column property values, select the column and specify the property values in Properties.

Displaying parameters

In Parameters, choose Describe Query to display the query's parameters and the data type for each parameter. You can type a default value for a parameter in Default value, as shown in Figure 2-46.

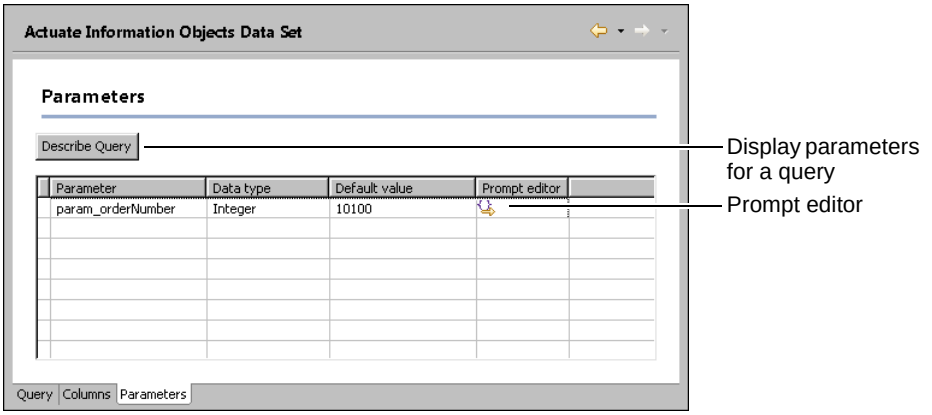


Figure 2-46 Using Describe Query to display parameters for a query

To set the prompt property values for a parameter, select Prompt editor for that parameter.

Index

Symbols

% character (pattern matching) 23
< operator 18
<> operator 18, 21
<= operator 18
= operator 12, 13, 18
> operator 18
>= operator 18
_ character (pattern matching) 23

A

accessing
  data set editor 9
  expression builder 10, 11
  information objects 2
  Prompt editor 41
  query editors 4, 9, 48
Actuate SQL expressions 11, 46
  See also SQL expressions
Actuate SQL parameters 28, 40
  See also parameters
Actuate SQL queries 15, 20
  See also queries
adding
  data sets 2, 4
  data sources 2
  filter conditions 17–27
  filters 17, 18, 20, 28, 30
  join conditions 12
  parameters to queries 40–41
aggregate columns 34, 36, 39
aggregate filters 18, 33
aggregate functions 36
aggregation 36
aliases 10, 49
alignment options 12
Analysis Type property 10
AND operator 12, 25, 26
Auto suggest property 43

B

BETWEEN operator 17
BIRT Designer Professional 2
blank values 21

C

cardinality (joins) 14, 15, 16
CARDINALITY keyword 14
changing
  column aliases 10
  filter conditions 27
  graphical queries 8, 48
  join conditions 13
  textual queries 49
characters
  column aliases and 10
  decimal separators and 22
  entering in lists 43
  filter conditions and 20, 22
  parameter names and 40
  pattern matching and 23
column aliases 10, 49
column identifiers 20
column names 9, 13
column references 9, 46
columns
  See also output columns
  adding to queries 9, 10, 42
  categorizing 9
  comparing values across 23
  defining source parameters and 46
  filtering missing values in 22
  filtering null values in 21
  grouping data and 35, 37, 38
  localized information objects and 47
  removing from queries 10, 38, 40
Columns page (Query Builder) 10, 49
comparisons
  date-and-time values 22
  filter conditions and 19, 20, 21, 23
  numeric values 22
  string values 22
computed columns 36
concatenation 24
Conceal Value property 44

control types (prompts) 30, 31, 42, 43, 45
creating
  data sets 2, 4
  filter conditions 17–27
  filters 17, 18, 20, 28, 30
  information object data sources 2
  information objects 2
  join conditions 12
  joins 12–16
  list of values 30, 32, 42
  queries 4, 48
  source parameters 46
  SQL parameters 28
CURRENT_DATE function 24
CURRENT_USER function 25
customizing queries 5, 8

D

data
  See also values
  aggregating 36
  filtering 16–34
  grouping 35–40
  limiting amount returned 8, 9, 16, 25
  retrieving 2, 12, 42
  sorting 34
data filters. *See* filters
data rows. *See* rows
data set editor. *See* Edit Data Set dialog
data sets 2, 4, 34
  See also data; result sets
data sources 2, 12, 45
  See also information objects
Data Type property 44
data types 31, 40, 44, 45, 49, 50
date values 22, 24
DATEDIFF function 24
decimal separators 22
Default Value property 45
default values 5, 41, 43, 45, 50
deleting
  column references 9
  columns 10, 38, 40
  filter conditions 27
  join conditions 14
  parameters 41
dependent joins 15
Describe Query button 49, 50
Display Control Type property 45
Display Name property 45
display names 42, 45
displaying
  column categories 9
  join conditions 13
  output columns 49
  query output 5, 6
  query syntax 5
  report parameters 50
DISTINCT keyword 10
distinct values 10
Do Not Prompt property 45
documentation iii
duplicate names 10
duplicate rows 10
dynamic data filters 28, 29, 30

E

Edit Data Set dialog
  accessing 9
  creating dynamic filters and 30
  editing queries and 8
  opening Query Builder from 9
  previewing result sets and 5, 6
editing queries 8, 48, 49
empty values 21
Equal to operator 18
equality operator 12, 13
equijoins 15, 16
errors 5
expression builder 10, 11
expressions 12, 24
  See also SQL expressions

F

fields. *See* columns
file paths 30, 42, 49
filter conditions 17–27
Filter Conditions dialog 17, 18, 25, 28
filter expressions 17, 18, 20, 25, 26, 29
filter operators 17, 19, 29
Filter property 28

filtering
data 16–34
date-and-time values 22, 24
string values 22, 24
filters
See also filter conditions; filter expressions
aggregating data and 18, 33
comparing string patterns with 22
comparing values with 19, 20, 21, 22, 23
creating 17, 18, 20, 28, 30
excluding empty or blank values with 21
excluding sets of values with 21, 26
literal characters and 23
prompting for values for 28, 31
returning missing values and 22
selecting multiple values for 20
testing 25
Filters tab (Query Builder) 18, 20, 25
formatting column values 12
full outer joins 12
function signatures 11
functions 11, 24, 36

G

graphical query editor. *See* Information Object Query Builder
Greater than operator 18
Greater than or equal to operator 18
GROUP BY clause 33, 35, 37–40
Group By tab (Query Builder) 37, 40
grouping
data 35–40
filter conditions 25, 26

H

HAVING clause 29, 33
hidden parameters 45

I

IN operator 18, 20
information object data sources 2
See also information objects
Information Object Query Builder
creating joins and 12, 13
creating queries and 2, 4, 11, 48
defining output columns and 10
defining parameters and 28, 40
editing queries and 8, 48
filtering data and 18, 20, 26, 28, 34
grouping data and 37, 38
hiding column categories for 9
opening 4, 9
previewing queries and 5
sorting data and 35
information objects
accessing 2
building queries for. *See* queries
categorizing columns in 9
creating 2
defining joins for 12–16
defining parameters in 46
filtering data in 16–34
limiting data returned from 8, 9, 16, 25
localizing 47
referencing 30, 42, 49
retrieving data from 2, 12, 42
retrieving distinct values from 10
selecting 2
inherited properties 30, 46
inner joins 14
input 28, 31, 42
input parameters 45
IS NOT NULL operator 18, 21
IS NULL operator 18, 21

J

join algorithms 14, 15, 16
join conditions 12, 13, 14
join operators 12, 13
join types 14
joins 12–16
Joins tab (Query Builder) 13

L

LEFT function 24
left outer joins 14
Less than operator 18
Less than or equal to operator 18
LIKE operator 18, 22
literal characters 23
local parameters 46

localization properties files 47
Localization tab (Query Editor) 47
logical operators 26

M

maps (information objects) 30, 42
matching character patterns 22
memory 15, 16
merge joins 15
missing values 22, 40
multiple filter conditions 25, 25–27
multiple join conditions 12, 14

N

Name property 45
naming
  output columns 10
  parameters 40, 45
nested loop joins 16
NOT BETWEEN operator 18, 21
Not equal to operator 18, 21
NOT IN operator 18
NOT LIKE operator 18, 21
NOT operator 26
null values 21, 41
numeric values 22, 41, 47
  See also values

O

ODA data sources 45
online documentation iii
opening
  data set editor 9
  expression builder 10, 11
  Information Object Query Builder 4, 9
  SQL text editor 48
operators. *See* SQL operators
optimizing queries 14, 15, 17
OR keyword 48
OR operator 26
Order By clause 34
outer joins 12, 14
output (query). *See* result sets
output columns
  defining 9–11
  displaying properties for 11
  formatting values for 12
  renaming 10
  setting order of 10
  setting properties for 11, 49
  viewing 49
output parameters 45

P

Parameter Mode property 45
parameter names 28, 40
parameterized queries 5, 40
parameters
  adding to queries 40–41
  assigning data types to 40, 44, 45
  assigning default values to 41, 43, 45, 50
  assigning null values to 41
  creating 28
  deleting 41
  filtering data and 19, 28
  hiding 45
  naming 40, 45
  previewing default values 5
  prompting for values and 42, 46
  returning missing values and 40
  setting properties for 28, 41, 42, 44
  setting source 46
  setting values for 44, 47
  specifying required 45
  viewing 50
Parameters tab (Query Builder) 28, 40, 41, 46, 50
paths 30, 42, 49
pattern matching 22
percentages 23
predefined filters 28
previewing
  query syntax 5
  result sets 5, 6
Prompt editor 41, 42
prompt properties 30, 42, 46
prompting for values 28, 31, 42
properties
  inheriting 30, 46
  join conditions 13
  output columns 11, 49

parameters 41, 42, 44
prompting for values and 28, 30, 42, 46

Q

queries
See also SQL expressions; SQL statements
aggregating data and 36
creating 4, 48
customizing 5, 8
defining output columns for 9–11
disabling automatic grouping for 40
editing 8, 48, 49
filtering data with 16–34
limiting data returned by 8, 9, 16, 25
optimizing 14, 15, 17
parameterized. *See* parameterized queries
previewing 5, 6
prompting for values and 30, 42, 43
retrieving data with 2, 9, 12
returning distinct values for 10
returning null values for 21
returning output for. *See* result sets
saving 48
validating 25
viewing output columns for 49
Query Builder. *See* Information Object Query Builder

R

removing. *See* deleting
Required property 45
result sets
See also queries
changing column order for 10
defining multiple conditions for 25, 25–27
limiting number of rows in 16, 21, 28, 33
missing values in 40
previewing 5
removing duplicate rows from 10
renaming columns in 10
setting number of records in 6
right outer joins 12
rows
excluding duplicate 10
restricting number returned 16, 21, 28, 33
setting filters for 17, 25

S

saving queries 48
scalar values 46
SELECT clause. *See* SELECT statements
SELECT statements 9, 11, 12, 37
See also SQL statements
Size property 45
Sort tab (Query Builder) 35
sorting data 34
source parameters 46
SQL expressions
See also SQL statements
adding 10, 11
changing column names to 13
entering source parameters in 46
entering special characters in 20
filtering data and 17, 18, 20, 25, 26, 29
previewing syntax 5
setting join algorithms in 14, 15, 16
setting join cardinality in 14, 15, 16
SQL functions. *See* functions
SQL operators
filter conditions 17, 19, 29
joins 12, 13
SQL parameters 28, 40
See also parameters
SQL Preview tab (Query Builder) 5
SQL statements
See also queries; SQL expressions
adding columns to 10, 42
adding parameters to 40–41
changing column aliases for 10
changing filter conditions in 27
changing join conditions in 13
comparing string values and 22
copying 48
displaying 5
entering manually 48
grouping data with 35–40
grouping filter conditions in 25, 26
referencing information objects in 30, 42, 49
removing column references from 9
removing columns from 10, 38, 40
removing filter conditions in 27
removing join conditions from 14

SQL statements *(continued)*
removing parameters from 41
setting filter conditions in 17–27
setting join conditions in 12–16
sorting data with 34
viewing column references in 9
viewing join conditions in 13
SQL text editor. *See* textual query editor
SQL text editor icon 48
stored procedures 45
strings
assigning to parameters 47
concatenating 24
matching exact characters in 23
matching patterns in 22
returning substrings in 23
setting default values for 41
setting filter conditions for 22, 24
summary data. *See* aggregation
syntax errors 5

T

table names 20, 49
text. *See* strings
textual queries
changing 49
creating 48–50
displaying output columns for 49
displaying parameters for 50
filtering data with 20, 26
grouping data and 37
removing columns from 40
saving 48
textual query editor 11, 48
time values 22
timestamps 41, 47
types. *See* data types

U

UNION keyword 48
unique values. *See* distinct values
user names 24

V

values
See also data
aligning 12
assigning default 41, 43, 45, 50
assigning to parameters 44, 47
comparing. *See* comparisons
creating list of 30, 32, 42
displaying default 5
filtering empty or blank 21
filtering on multiple 20, 23
hiding 44
previewing default parameter 5
prompting for 28, 31, 42
returning distinct 10
returning missing 22, 40
testing for null 21
viewing
column categories 9
join conditions 13
output columns 49
query output 5, 6
query syntax 5
report parameters 50

W

WHERE clause 17, 29, 40
WITH clause 40