

## Lesson 4: Working with Databases

*Lesson 4 introduces the datasource component, and leads you through the steps to create and run queries using this component. The completed project for this lesson is stored as project catalog1 in the tutorials component archive, Lesson4 folder. Sections in this lesson include:*



- 4.1 *The Problem*
- 4.2 *Creating the Catalog Document Types*
- 4.3 *Linking a Datasource Control to a Database*
- 4.4 *Mapping Types to Database Fields*
- 4.5 *Wiring and testing the catalogQuery Component*
- 4.6 *Restricting the Data Retrieved from the Database*
- 4.7 *Modifying Database Records*
- 4.8 *Review*
- 4.9 *Challenge Yourself*

One of the problems that faces businesses today is the inability to extract information from the huge volume of data stored in databases into a format that their systems can process. By virtue of its strength in handling user defined data types and performing data transformation, XML is a natural for exchanging information between valuable databases and the companies that want to access them. Component X Studio makes it easy to define the DTD's and build an XML application that supports the database exchange.

### 4.1 The Problem

---

In this lesson we'll be dealing with two database tables: The Products table contains a row for every product number a company's catalog. For every product that comes in different colors, there is a corresponding color row in the ProductColors table. Thus, there exists a parent/child relationship between the Products and ProductColors tables.

We want to create a component that can retrieve all or some of the product records, depending on the input we send it. We want to be able to fully express the parent/child relationship that exists between the tables.

Figure 4-1: The Products table contains a row for every product in a company's catalog.

| productNo | productName     | productDescription          | productPrice |
|-----------|-----------------|-----------------------------|--------------|
| 1         | Tennis Balls    | Vacuum sealed can of 3      | \$4.95       |
| 2         | Golf Glove      | Premium cabretta leather    | \$15.99      |
| 3         | Backpack        | 22" rolling canvas backpack | \$8.00       |
| 4         | Tent            | 2 person popup tent.        | \$120.00     |
| 5         | Basketball      | NBA Approved                | \$28.00      |
| 6         | Swim Goggles    | Anti-fog swim goggles       | \$18.00      |
| 7         | Gardening Tools | Trowel, hand rake and prun  | \$35.00      |
| 8         | Golf bag        | 9" circumference. Texture   | \$195.00     |

Figure 4-2: The ProductColor table contains a row for each color that a product comes in.

| productNo | productColor |
|-----------|--------------|
| 1         | White        |
| 1         | Yellow       |
| 2         | Tan          |
| 2         | White        |
| 3         | Blue         |
| 3         | Green        |
| 3         | Red          |
| 4         | Black        |
| 6         | Blue         |
| 6         | Red          |
| 8         | Black        |
| 8         | Blue         |
| 8         | Green        |
| 8         | Tan          |

We are interested in a *join* of these two tables. For every row in the products table we look for 0 or more matches in the productColors table on the productNo field, as shown in Figure 4-3.

Figure 4-3: The catalog query shows the join between the Products and ProductColor tables.

| productNo | productName  | productDescription                            | productPrice | productColor |
|-----------|--------------|---|--------------|--------------|
| 1         | Tennis Balls | Vacuum sealed can of 3                        | \$4.95       | White        |
|           |              |   |              | Yellow       |
| 2         | Golf Glove   | Premium cabretta leather                      | \$15.99      | White        |
|           |              |   |              | Tan          |
| 3         | Backpack     | 22" rolling canvas backpack with leather trim | \$58.00      | Blue         |

| productNo | productName  | productDescription                | productPrice | productColor |
|-----------|--------------|-----------------------------------|--------------|--------------|
|           |              |                                   |              | Green        |
|           |              |                                   |              | Red          |
| 4         | Tent         | 2 person popup tent.              | \$120.00     | Black        |
| 6         | Swim Goggles | Anti-fog swim goggles             | \$18.00      | Red          |
|           |              |                                   |              | Blue         |
| 8         | Golf bag     | 9" circumference. Textured vinyl. | \$195.00     | Black        |
|           |              |                                   |              | Blue         |
|           |              |                                   |              | Tan          |
|           |              |                                   |              | Green        |

## 4.2 Creating the Catalog Document Types

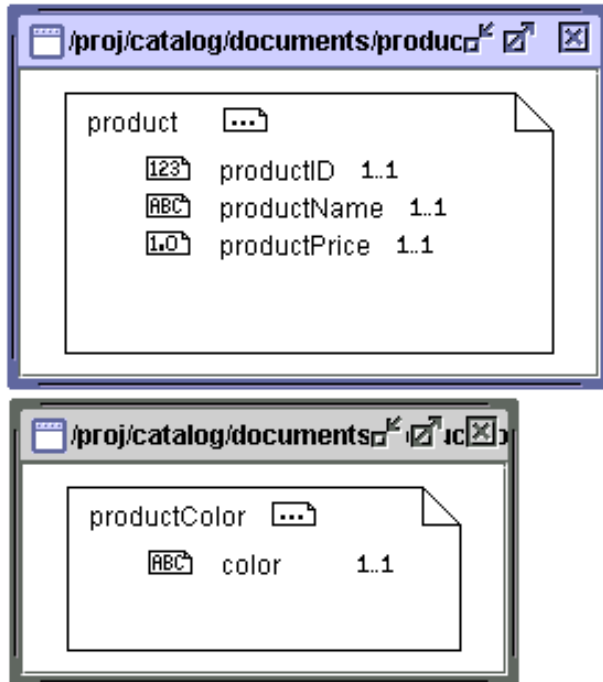
---

In this section you'll set up the project by creating the packages, type packages and types.



1. Open Component X Studio. By default you are in a new project.
2. Save the project as **catalog** in the Lesson4 folder of the tutorials archive using the **File | Save Project** menu item.
3. Add a catalog package to your project using the **New | New Package** menu item.
4. Add a new types package called documents to the catalog package.
5. Create a product type and a productColor type, as shown in Figure 4-4. The constraint and identity attributes for the elements are shown in Figure 4-5. If you are not familiar with how to create types, refer to Lesson 3.

**Figure 4-4: Create two composite types: product and productColor**

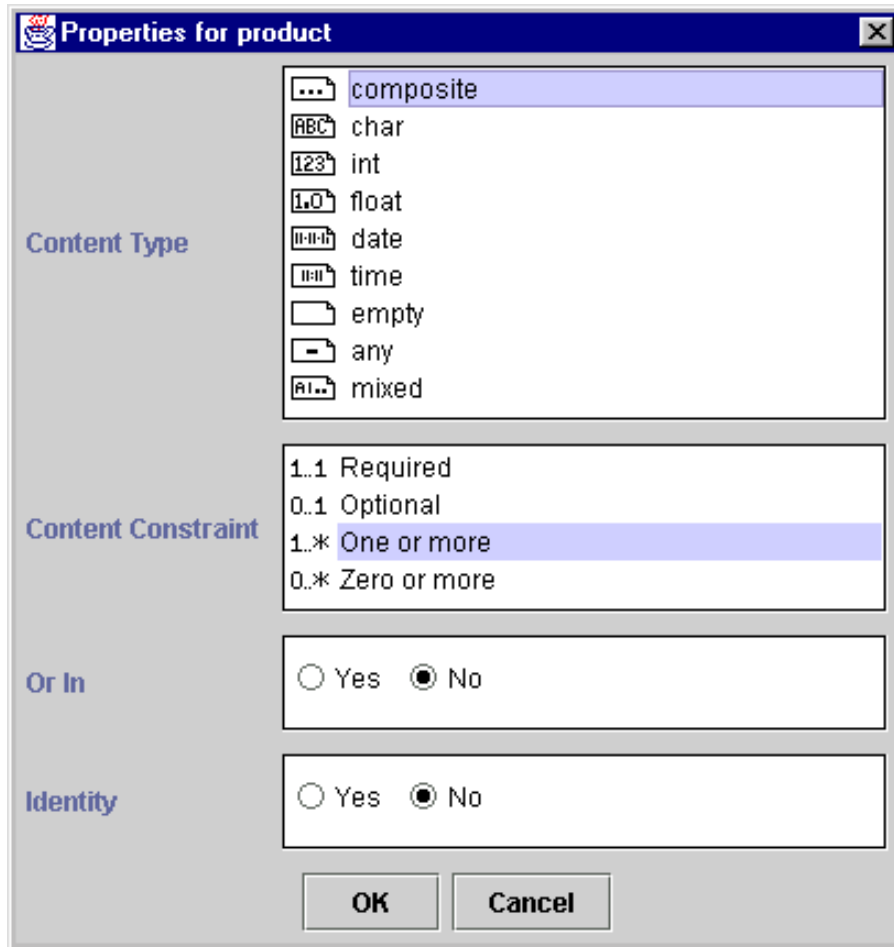


**Figure 4-5: Types for the catalog project.**

| TYPE         | Content Type | Content Constraint | Identity |
|--------------|--------------|--------------------|----------|
| Catalog      | Composite    |                    |          |
| Product      | Composite    |                    |          |
| ProductID    | Integer      | 1:1                | Yes      |
| productName  | Character    | 1:1                | No       |
| Price        | Float        | 1:1                | No       |
| productColor | Composite    |                    |          |
| Color        | Character    | 0 or more          | No       |

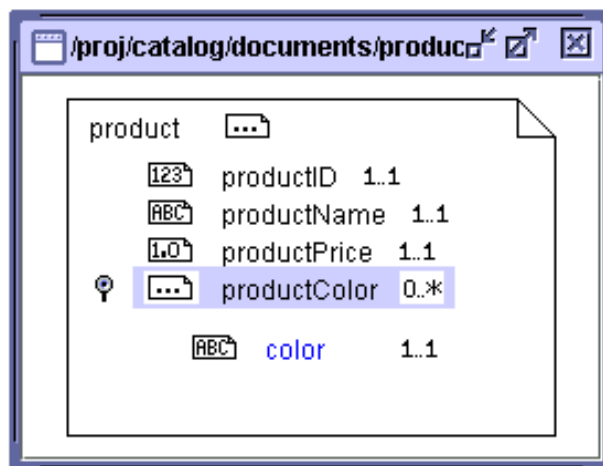
- Now drag the productColor type from the Component Tree to the product type. Open the properties dialog for the productColor type and set the content constraint to 0 or more as shown in Figure 4-6. This not only allows for products that don't have a color attribute, but also for those that have multiple color attributes.

Figure 4-6: Setting the product constraint to 1 or more within the catalog type.



7. When you are finished, the product type will appear as shown in Figure 4-7.

Figure 4-7: The completed product type.



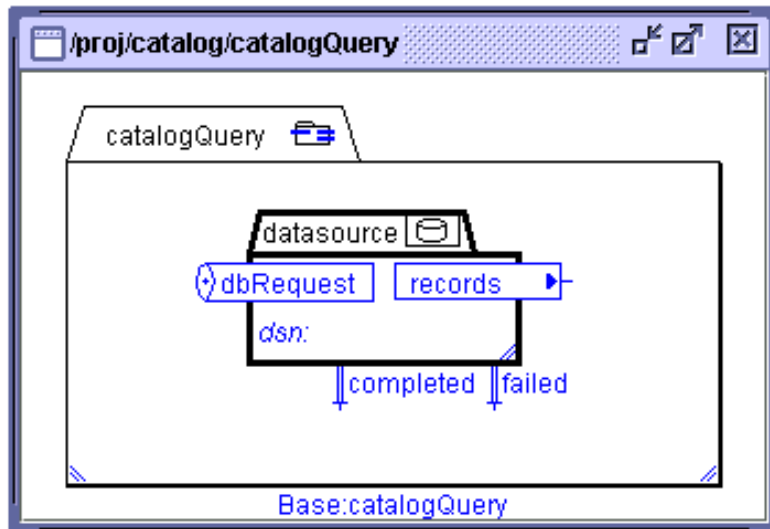
### 4.3 Linking a Datasource Control to a Database

Now that you have created the types used in this project, you are going to create catalog query component, and configure it to connect to an existing ODBC data source name.

- V

1. Add a new composite component called catalogQuery to the catalog package.
  2. Locate the datasource component in the database palette and add it to the catalogQuery component, as shown Figure 4-8.

**Figure 4-8: Adding the datasource component to the catalogQuery component.**



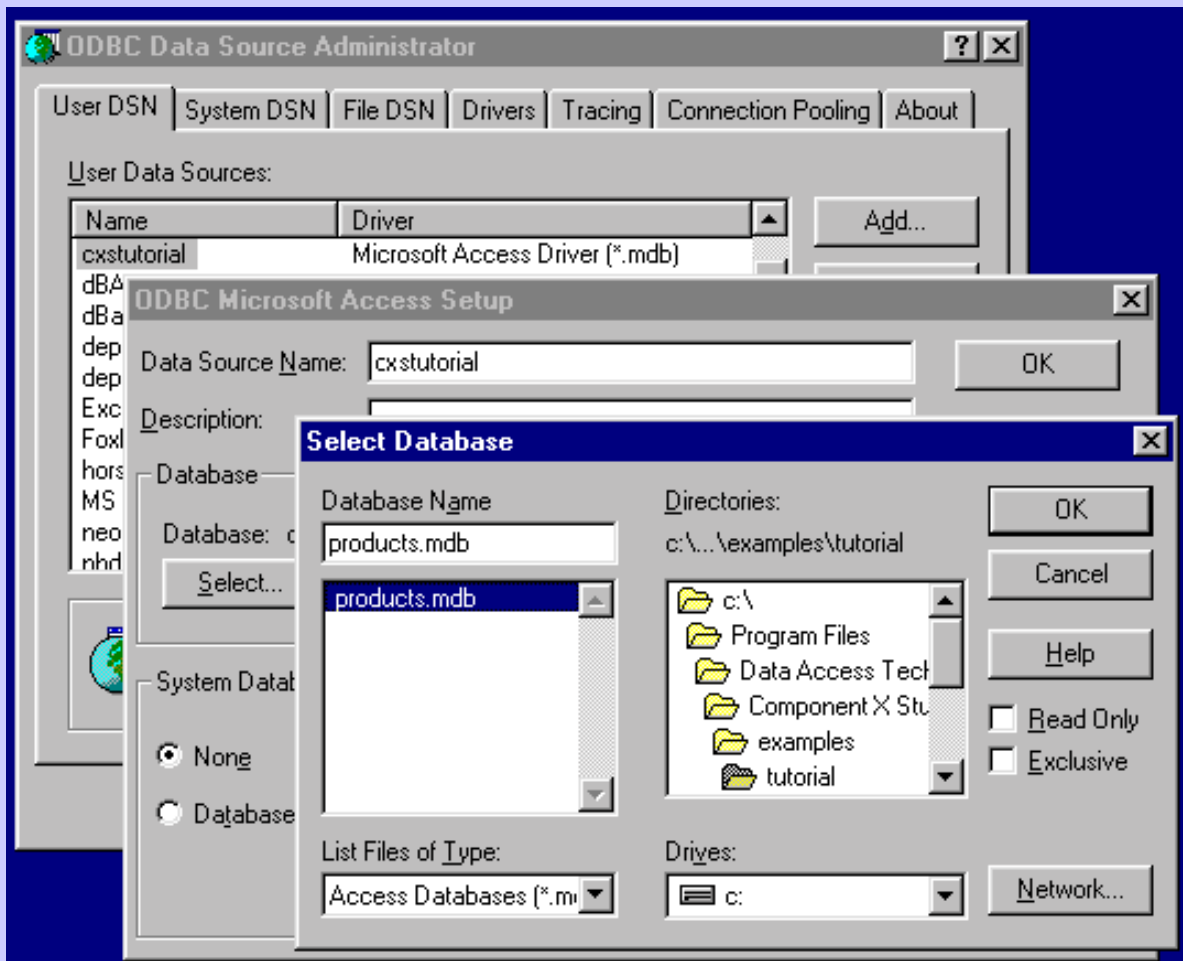


## Setting Up an ODBC Data Source Name

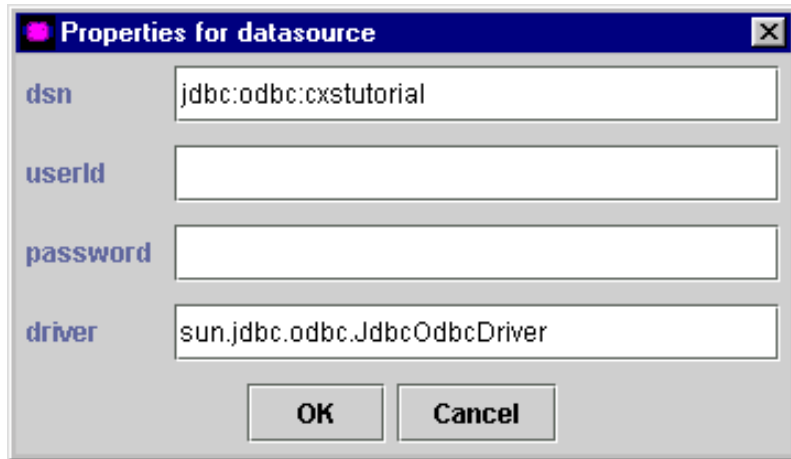
Note that this lesson assumes that you know how to set up a data source name (DSN) using ODBC or JDBC. You will need to create a dsn that points to the database used by the Tutorial lessons.

Briefly, if you are running on Windows or Windows NT, open the ODBC Administrator in the Control Panel and create a new data source name called cxstutorial that points to the Access database products.mdb, found in the Tutorial folder, as shown in Figure 4-9.

**Figure 4-9: Specifying an ODBC Data Source Name (DSN)**



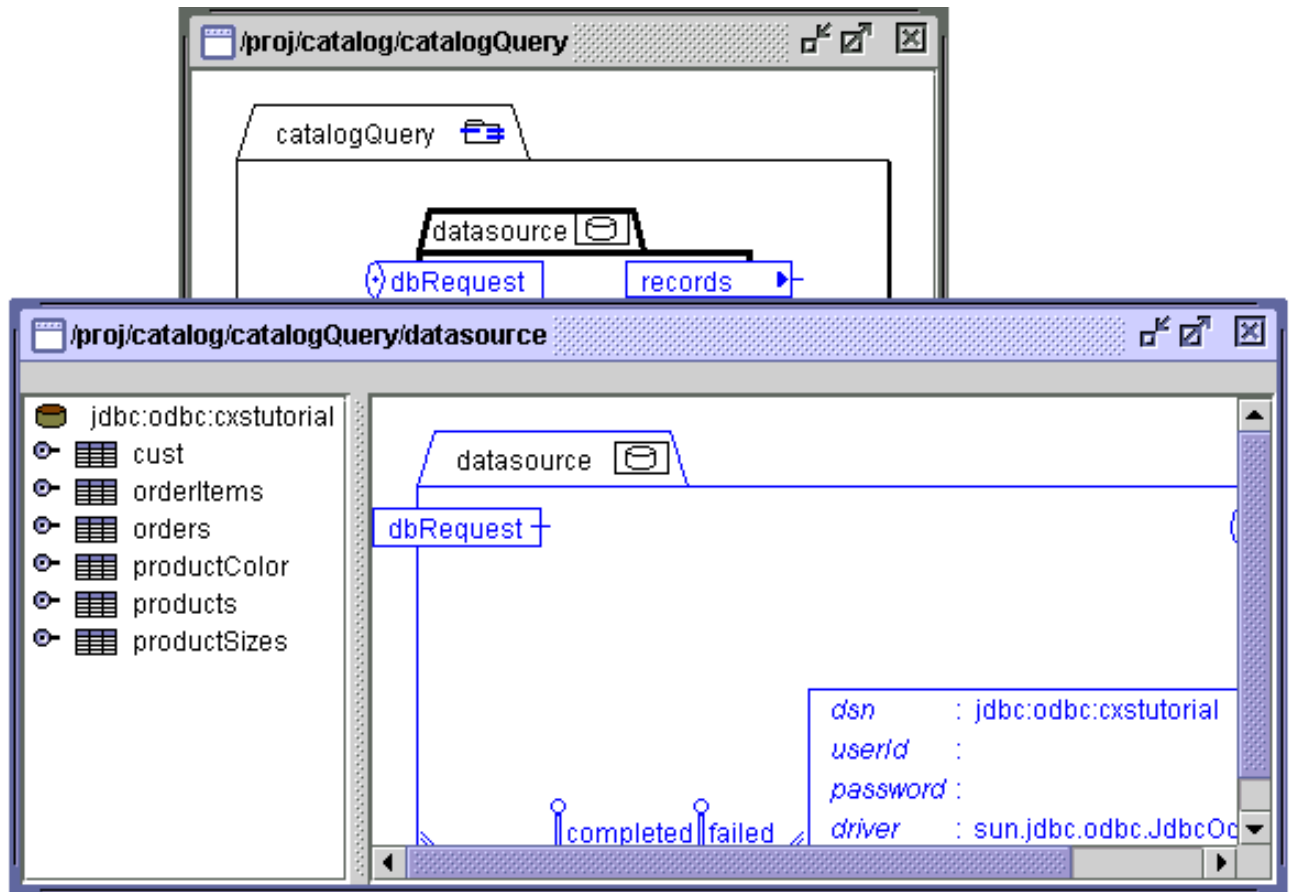
**Figure 4-10: Specifying the data source name (dsn) as jdbc:odbc:csxtutorial.**



3. Open the datasource component by double-clicking or right clicking on it. If you have correctly created and specified the dsn, you will see the database tables in tree on the left, as shown in Figure 4-11. If your dsn is not set up correctly, or you did not specify it correctly in the previous step, or if the database is not found, you will see a <not connected> message in the tree. In this case, consult the note below for additional information on creating a dsn.
4. Now open the datasource component by double-clicking or right clicking on it. If you have correctly created and specified the dsn, you will see the database tables in tree on the left, as shown in Figure 4-11.



Figure 4-11: If the JDBC/ODBC DSN is correctly configured, the tables will appear in the tree on the right of the datasource.

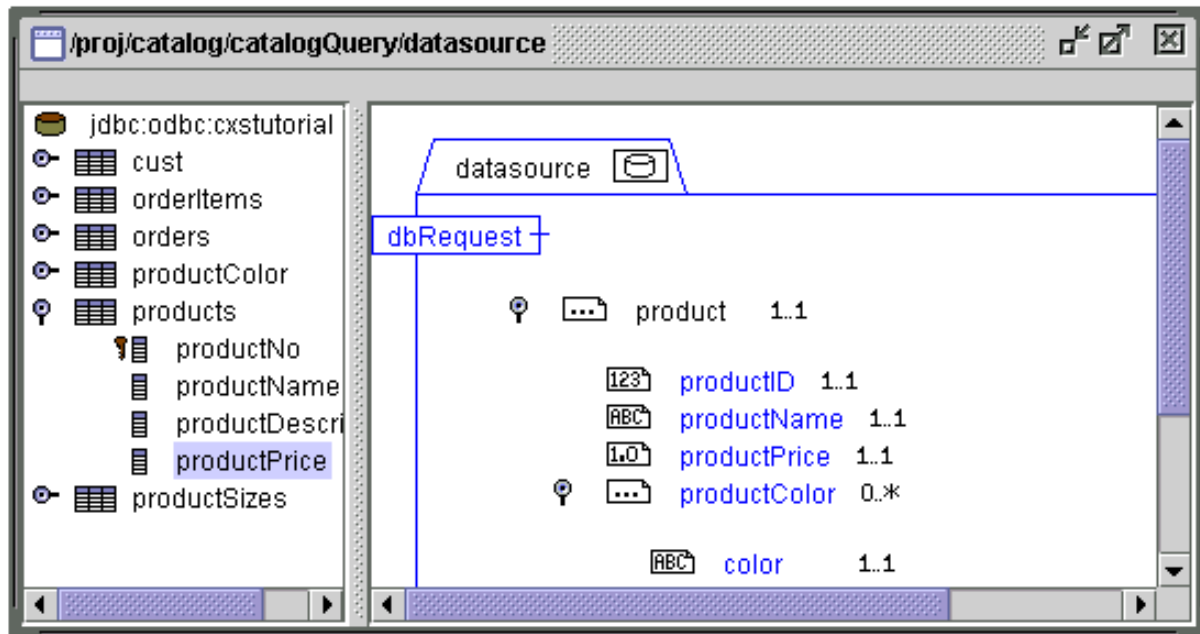


## 4.4 Mapping Types to Database Fields

In the previous section, you created a query component and linked it to an ODBC datasource. In this section, you will continue configuring the datasource component by mapping types to database fields.

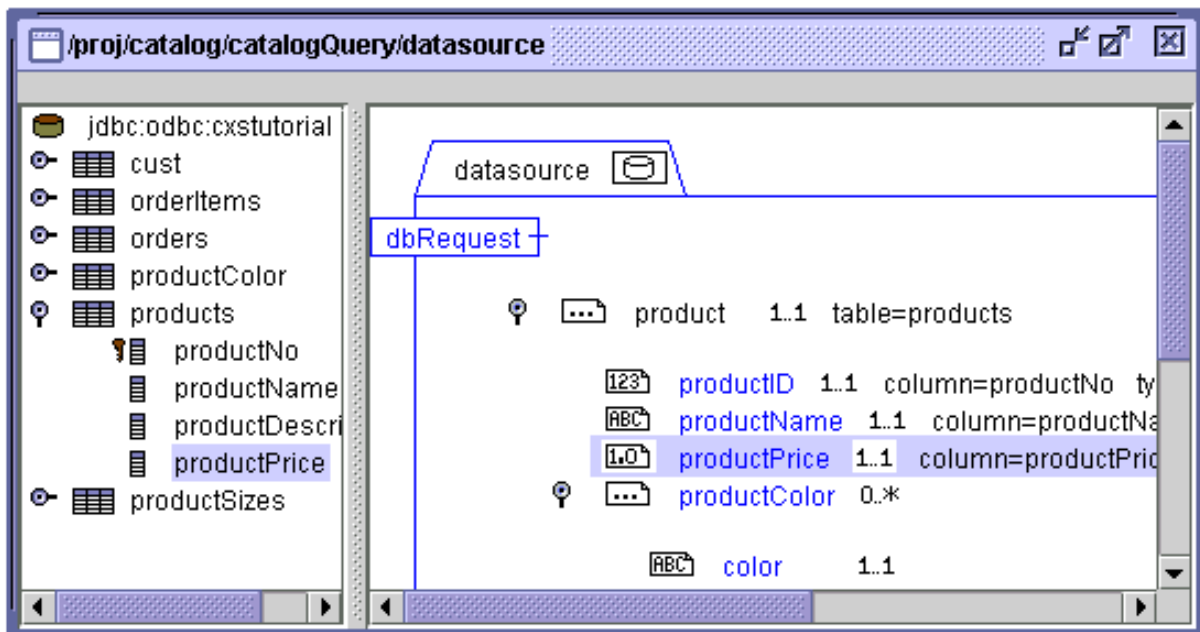
- V** Continuing on from step 6 of Section 4.3, with the catalogQuery datasource control open.
- Drag the product type from the documents type folder into the datasource control. The catalog type will be displayed in the datasource component, as shown in Figure 4-12

Figure 4-12: Drag the product type from the Component Tree to the datasource component.



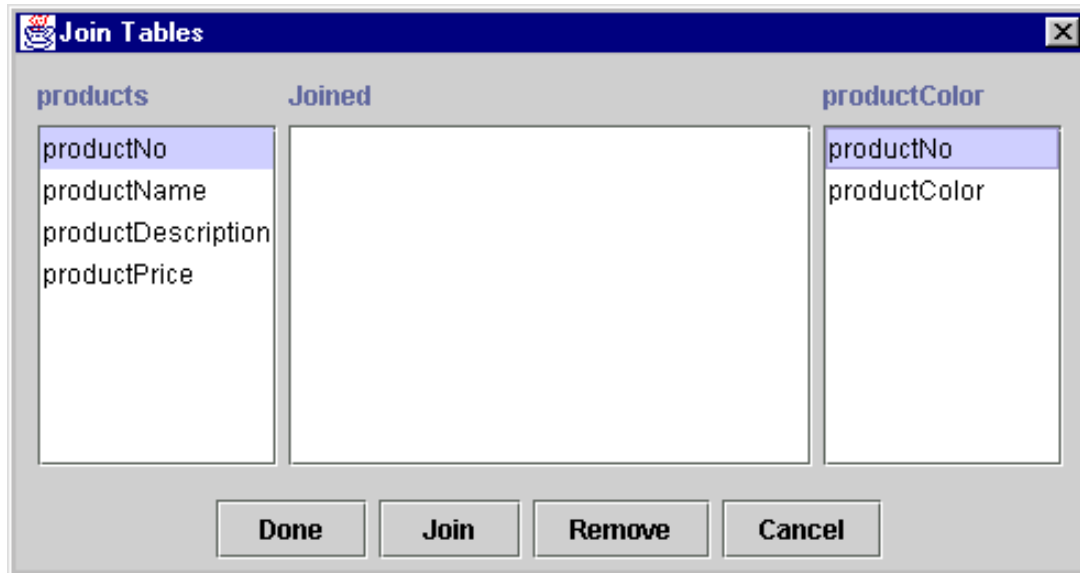
3. Now one by one, drag and drop each the fields from the Products table on the left to the corresponding type on the right, as shown in Figure 4-13. Note that the ProductDescription field does not appear in the product type, so you won't need to drag it.

Figure 4-13: Drag and drop the Product table fields from the left to the corresponding type on the right.



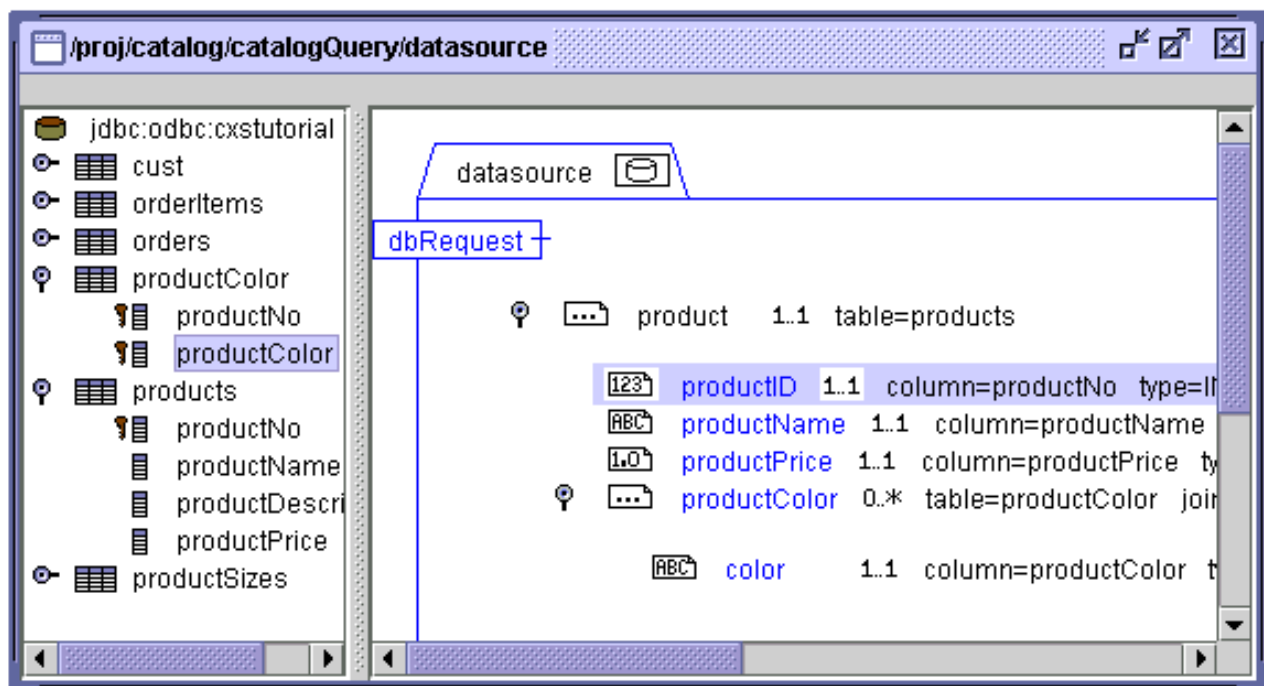
4. Next drag and drop the productColor.color field from the left to the color type on the right. A join dialog will appear. Fill it out by clicking on the productNo field in the left column, and the productNo field in the right column, and then clicking the Join button, as shown in Figure 4-14.

Figure 4-14: Specify the join between the Products and productColor tables.



5. Once you have specified the join, click the Done button. The mapping will appear as in Figure 4-15.

Figure 4-15: The datasource component with all of the fields mapped.

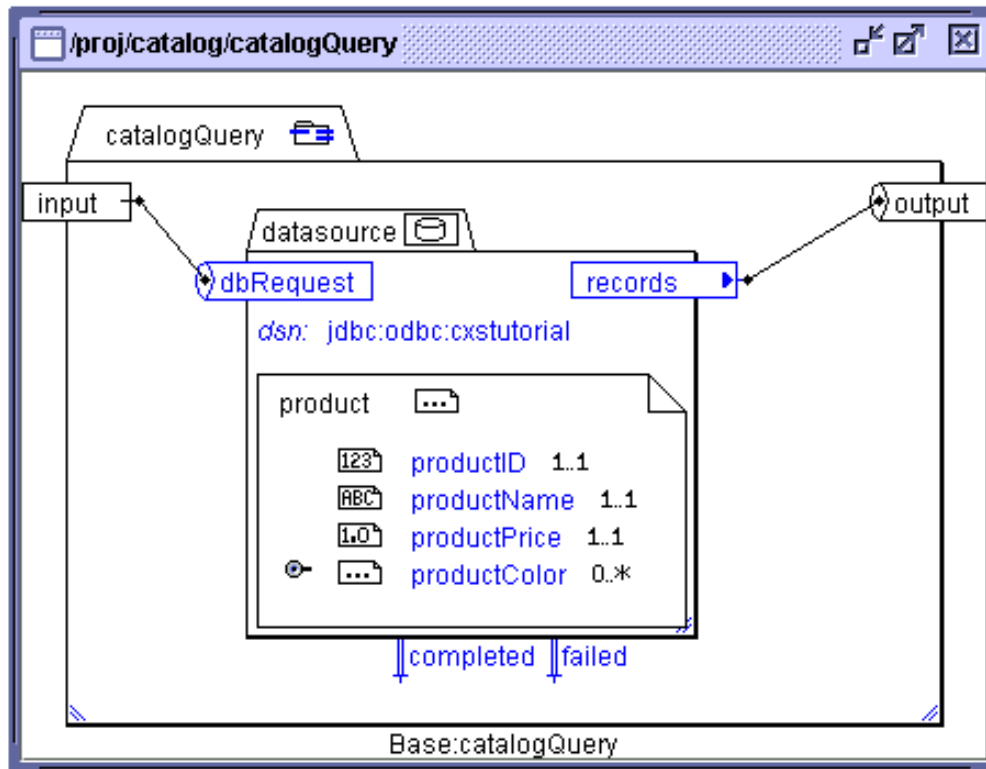


6. Close the datasource component.

## 4.5 Wiring and testing the catalogQuery Component

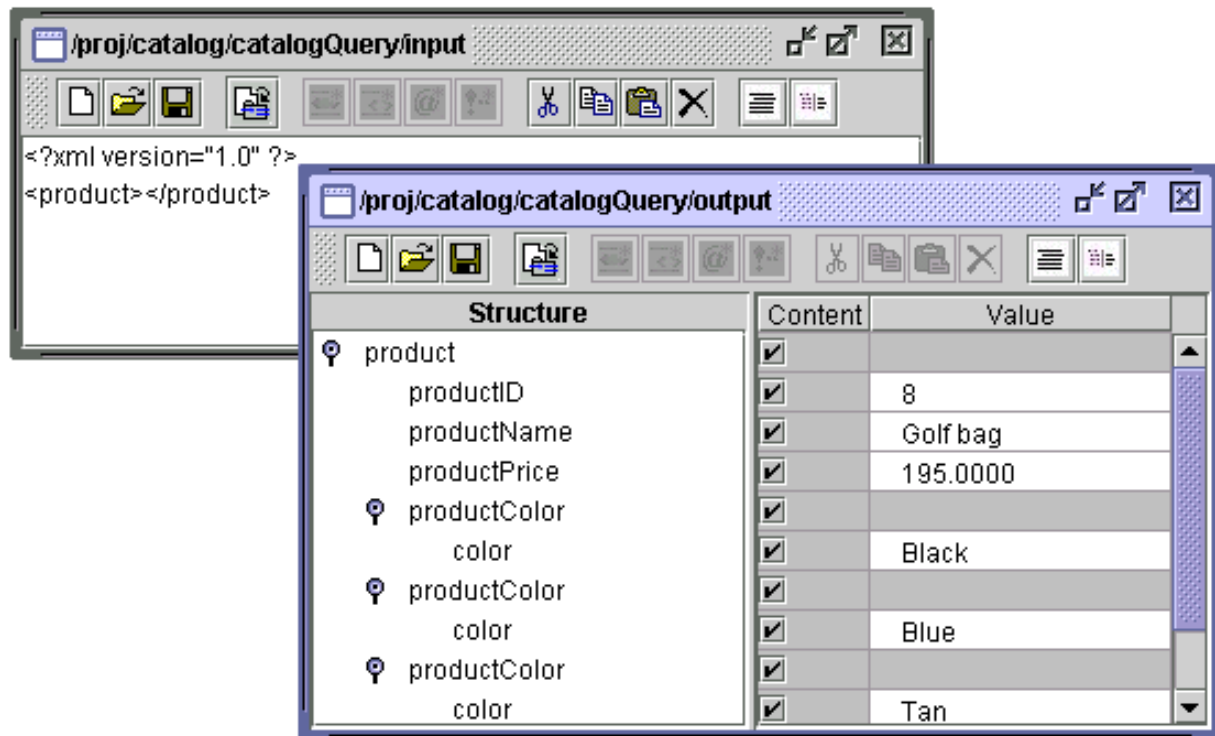
1. Add input and output ports to the catalogQuery component, and wire them ports as shown in Figure 4-16.

Figure 4-16: Add input and output ports to the catalogQuery component, and wire them as shown.



2. Open the input and output ports by double-clicking on them. The port dialogs will appear.
3. Run a query that retrieves all of the data in the Products/productsColor join by switching the input port to text mode and entering `<product/>`. Click the Send button.

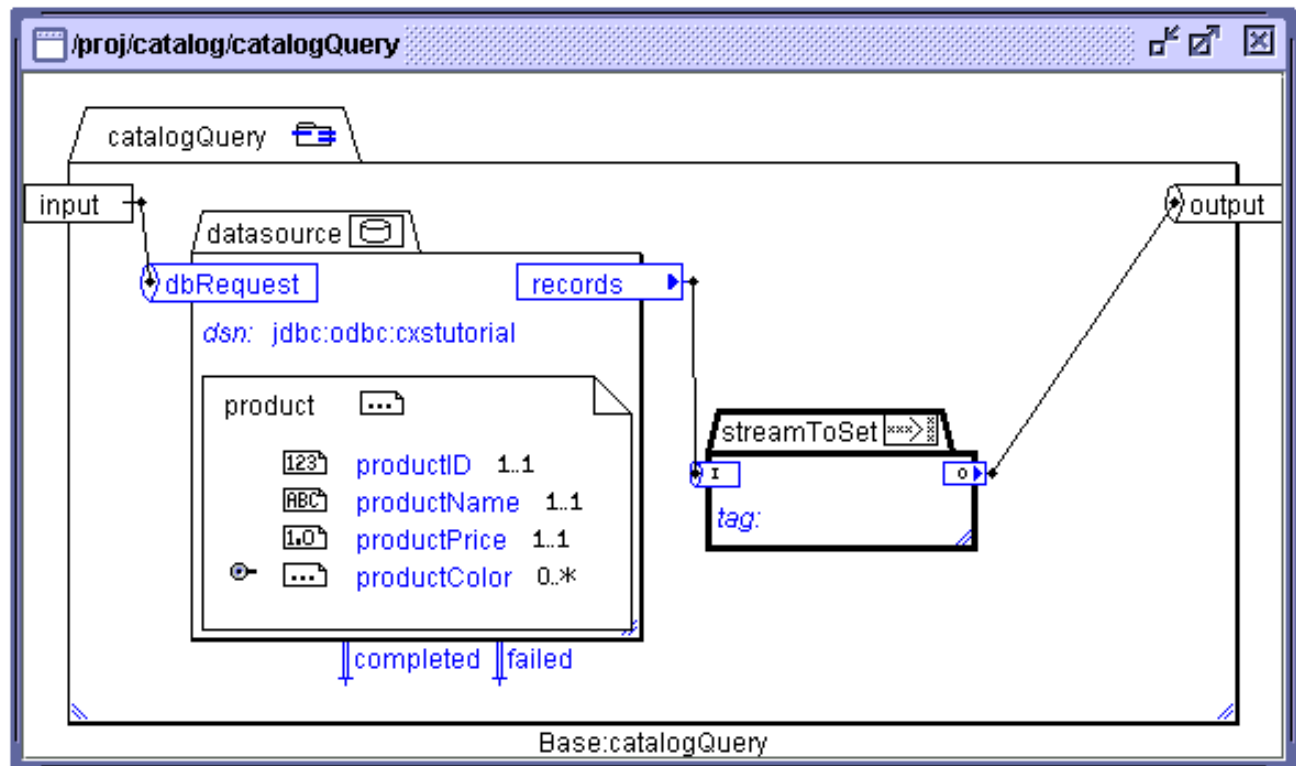
Figure 4-17: Test the data source by sending <product/> from the input port. The Output port shows only the last record retrieved.



4. Notice in **Error! Reference source not found.** that the output port shows only the last record retrieved. All of the records were sent through the output port as a stream, but only the last one is left in the port when it is displayed. In order to see all of the records, you need to add a *streamToSet* component, which converts a stream of data into a set that you can examine in its entirety in the output port

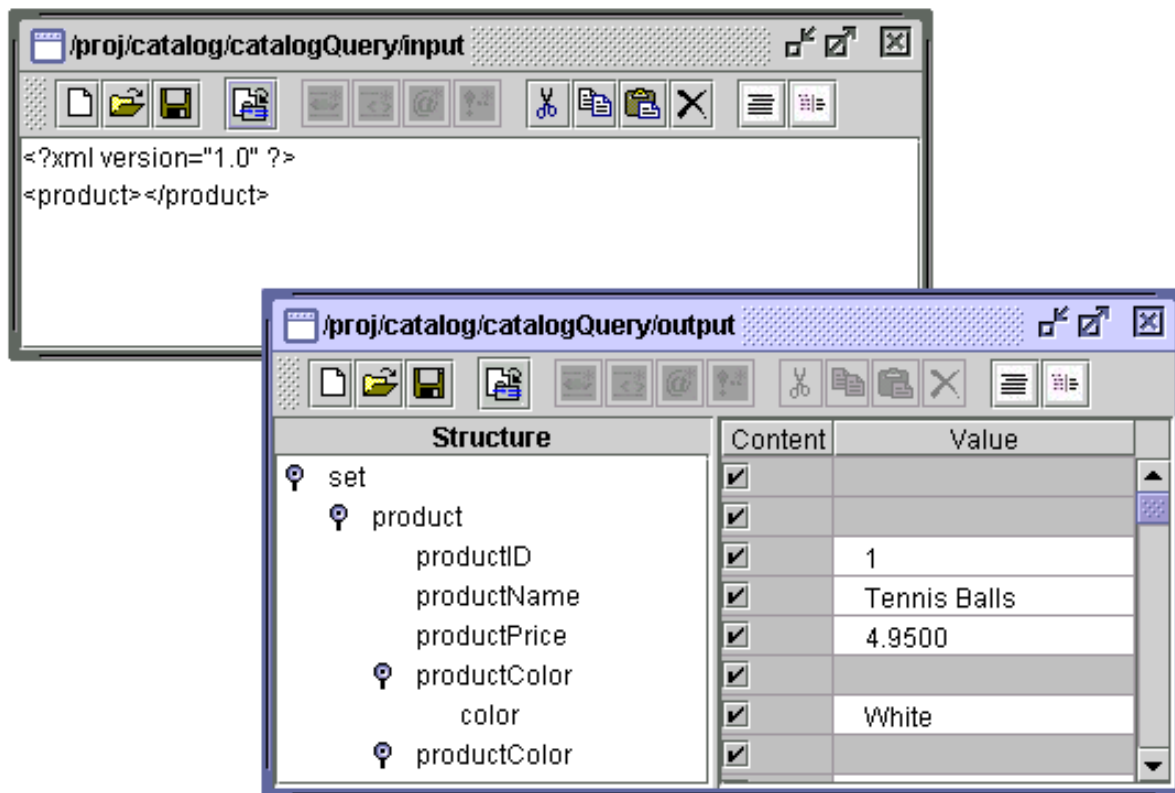
To add the *streamToSet* component, drag it from the filters palette to the *catalogQuery* component. Wire the Records port of the data source to the input port, and the output port to the *catalogQuery* output port, as shown in Figure 4-18.

**Figure 4-18:** Adding a streamToSet component allows you to examine all of the result records in the output port.



5. Now click the send button on the input port again. The result should be as shown in Figure 4-19.


**Figure 4-19:** Adding a streamToSet component causes all of the results to appear in the output port.



## 4.6 Restricting the Data Retrieved from the Database

One of the principal uses of database queries is to restrict the data retrieved by applying conditions to the values contained in fields in the database tables. In this section you will load and run some XML that contains query constraints, and examine the results.

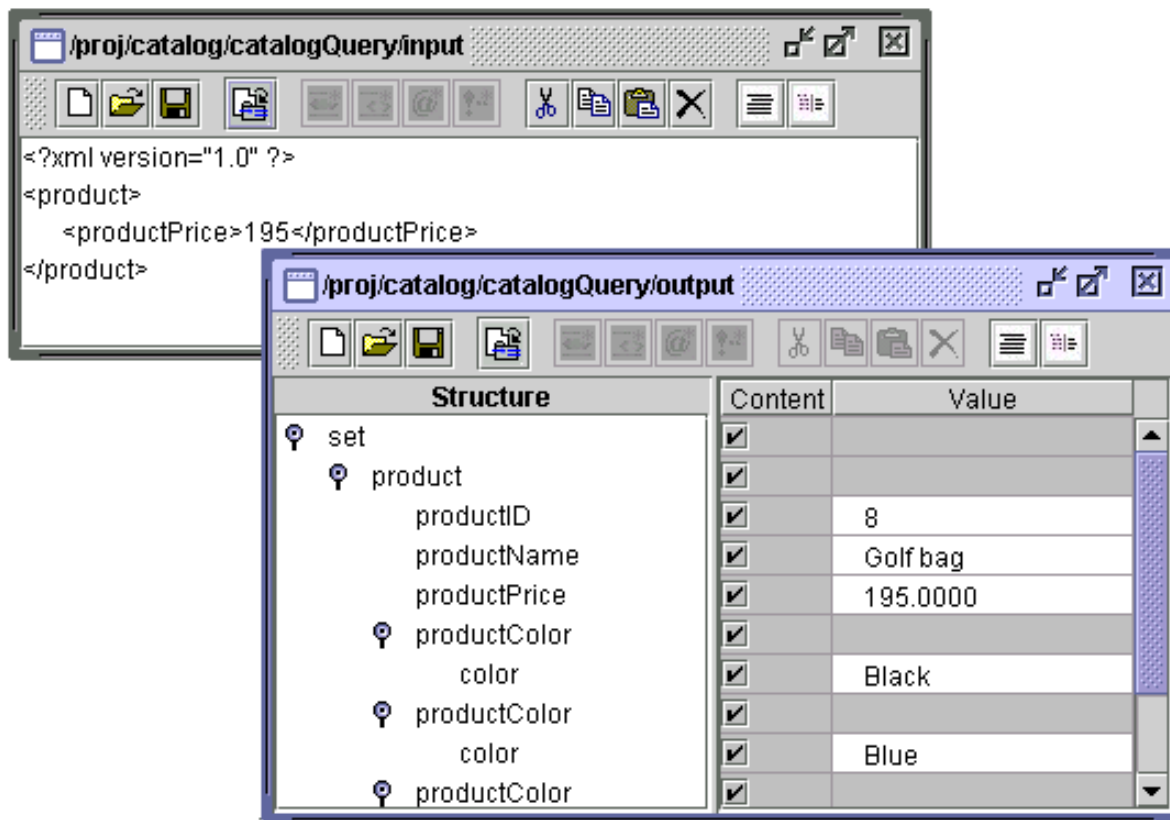


1. Open the Input port and click on the Open button (  ).
2. Locate and open the priceEQ195.xml file in the lesson4 folder. The following XML will be loaded:

```
<!-- Find all products where price = $195.00 -->
<product>
  <productPrice>195.00</productPrice>
</product>
```

3. Run the XML, and the query retrieves just the products for which the price is \$195.00. In this case there is exactly one product that satisfies the criteria -- a golf bag, but note that the product comes in several colors, as shown in Figure 4-20.

Figure 4-20: Retrieving products priced at \$195.00.



4. Locate and open the priceLT195.xml file in the lesson4 folder. The following XML will be loaded. Note the cx:comparison = "LT" attribute. The attribute value must appear in quotes.

```
<!-- Find all products where price < $195.00 -->
<product>
  <productPrice cx:comparison = "LT" >195.00</productPrice>
</product>
```

Figure 4-21: cx:comparison Attributes

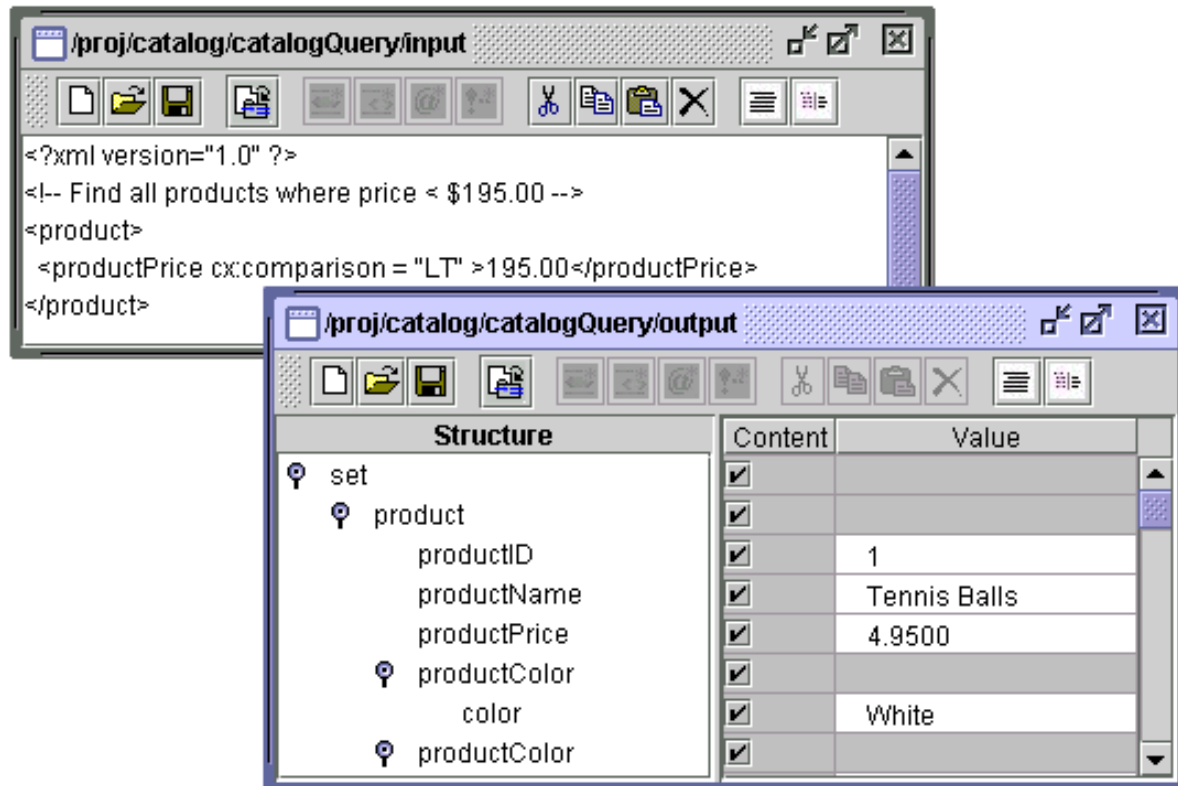
| cx:comparison attribute | Meaning  |
|-------------------------|--|
| EQ                      | Equal. (This is the default.)                                    |
| NE:                     | Not equal.   |
| ISNULL                  | Must be null   |
| ISNOTNULL               | Must not be NULL   |
| GT                      | Greater than   |
| GE                      | Greater than or equal  |
| LT                      | Less than  |
| LE                      | Less than or equal   |
| LIKE                    | Pattern matching. % matches any string, _ matches any character. |
| IFANY                   | At least one child must satisfy the nested constraints.          |
| IFEMPTY                 | No children satisfy the nested constraints.                      |
| IFALL                   | All children must satisfy the nested constraints.                |



Note when nesting a query, there are some combinations of actions that are not supported. Consult the Component X Studio datasource component documentation for details.

5. Run the XML, and the query retrieves just the products for which the price is less than \$195.00. In this case all products except the golf bag qualify.


Figure 4-22: Retrieving products priced less than \$195.00.



## 4.7 Modifying Database Records

The ability to update database records is another critical functionality. In this section you'll examine XML code that is used to add records to a database, and to update a database using the values in a document.




1. Open the Input port and click on the Open button (  ).
2. Locate and open the insertBirdcall.xml file in the lesson4 folder. The following XML will be loaded:

```

<!-- Insert related records into the products table and
prodColors table -->
<product cx:action="insert">
  <productID>9</productID>
  <productName>Birdcall</productName>
  <productPrice>12.50</productPrice>
  <productColor cx:action="insert">
    <color>red</color>
  </productColor>
  <productColor cx:action="insert">
    <color>blue</color>
  </productColor>
</product>

```

3. Note the cx:action attribute. In the previous XML, the default cx:action QUERY was used. Here we are using the cx:action="INSERT" attribute on both the product type and the prodColor type to direct the system to insert records into both of the associated tables.
4. Run the XML. Note that a non-Query cx:action does not retrieve data, so the output window will show an empty set ( <set/>.)
5. Clear the Input port and issue a command to see the record set (for example, <product />.) Notice that a new record has been added to the end, and that the birdcall comes in two colors.
6. Now we'll update the birdcall record to change the price to \$15.00. Open the Input port and click on the Open button (  ).
7. Locate and open the updateBirdcall.xml file in the lesson4 folder. The following XML will be loaded:

```

<!-- Modify the price of product 9 to $15.00 where product name
equals "Birdcall" -->
<product cx:action="update">
  <productName cx:action="constrain">Birdcall</productName>
  <productPrice cx:action="assign">15.00</productPrice>
</product>

```

8. Note the cx:action attributes. The UPDATE attribute tells the system that the document is to be updated. The CONSTRAIN attribute instructs the system to constrain the records that are to be updated, in this case, to restrict the records on which the update will be performed to just the record where productID equals 9. The ASSIGN action tells the system to assign the new value for the productPrice element.
9. Run the XML.

10. Clear the input port and send the command <product/>. Pay particular attention to the prices of the products. Note that only the price of the birdcall (product 9) was changed to \$15. Without the constrain action, all of the records would be updated!
11. You can delete the birdcall record(s) and restore the database to its original state by sending the following .XML, found in deleteBirdcall.xml in the Lesson4 folder. The related prodColor records are also deleted.

```
<!-- Delete product 9, the Birdcall -->
<product cx:action="delete">
  <productID cx:action="constrain">9</productID>
</product>
```

Figure 4-23 shows the cx:action attributes you can use to write your own datasource action commands. Also, there are more examples in the documentation, and in the .xml files in the Lesson4 folder.

**Figure 4-23: cx:action Attributes**

| cx:action attribute     | Meaning  |
|-------------------------|--|
| <b>QUERY</b>            | This is the default action. <ul style="list-style-type: none"> <li>❑ Applied to a FIELD, indicates the field will be used to constrain the parent document. This action is the default for fields nested within a CONSTRAIN, DELETE, QUERY, ASSIGN, or CONSTRAIN_PARENT document.</li> <li>❑ Applied to a document (type), outputs the set of records specified by the constraints contained within the document.,</li> </ul>  |
| <b>INSERT</b>           | Adds the document to the data source using whatever field information is supplied within the document.   |
| <b>UPDATE</b>           | Has two meanings depending on whether it is being applied to a field or a nested document: <ul style="list-style-type: none"> <li>❑ <b>Field:</b> indicates the field value will be set using the data in the field node. This action is the default for fields nested within a INSERT or UPDATE document.</li> <li>❑ <b>Document:</b> modifies the set of documents specified by the constraints contained within the document and using whatever field information is supplied within the document.</li> </ul> |
| <b>DELETE</b>           | Deletes the set of documents specified by the criteria contained within the document.  |
| <b>CONSTRAIN</b>        | Same as query except no records are output. This is used to limit a parent document set when operating on child documents  |
| <b>CONSTRAIN_PARENT</b> | Applies a constraint to a parent document based on child criteria. This is often used on a child document with a cx:comparison of IFEMPTY or IFANY, etc.   |
| <b>ASSIGN</b>           | Builds an associated between a child and parent document, and always applies by binding the child to the parent.   |

## 4.8 Review

---



You've covered a lot of material in this lesson, from creating a datasource component, to querying a related set of database tables and converting data streams to sets, to performing action commands like Inserts and Updates.

Before you can use the datasource component you must create a dsn. One way you can do this is by using the ODBC Administrator (found in the Control Panel in Windows.) Remember when you specify the dsn in the datasource component, you need to include the jdbc:odbc: prefix in front of the dsn name.

When working with datasource components that you need to create types (DTD's) that you will add to the datasource control. Then you complete the component by mapping the type elements to the corresponding database table fields.

When working with database tables that are related, you need to create a type for the child table as well as for the parent table. XML represents this relationship hierarchy using nesting.

Use a streamToSet component to translate a data stream into a data set that can be viewed in the output port.

You can restrict the data retrieved by a query using the cx:comparison attributes, such as EQ, LT, IFANY.

You can specify actions to be performed by the datasource using the cx:action attributes like QUERY, UPDATE, INSERT and CONSTRAIN. The QUERY action is the only one that retrieves a data stream.

In the next lesson you'll make use of this knowledge to improve the Market application by adding database interaction to it.

## 4.9 Challenge Yourself

---

Test what you learned in this lesson with the following:



1. From the input port, open and run each of the .xml files found in the Lesson4 folder. The comment at the top of the file describes the purpose of the XML.