Using IBM InfoSphere Warehouse Design Studio with pureXML data, Part 2: Create a control flow for multiple ETL jobs involving XML

Cynthia M. Saracco

April 01, 2010

Learn how to integrate business-critical XML data into your data warehouse using IBM® InfoSphere™ Warehouse Design Studio and DB2® 9.7 pureXML®. This two-part article series provides step-by-step instructions for using pureXML as both a source and target data source for extract, transform, and load (ETL) operations developed with InfoSphere Warehouse Design Studio. This article describes how to build a single control flow that calls multiple data flows that extract, transform, and load XML data in a specific sequence.

View more content in this series

Introduction

The first article in the series (Part 1) describes how InfoSphere Warehouse Design Studio can build and execute individual data flows that use DB2 pureXML operational data as input to relational or XML structures in a data warehouse. This article, the second in the series, describes how to build a single control flow that calls multiple data flows that extract, transform, and load XML data in a specific sequence. Using a control flow can enable you to incrementally develop, test, and execute complex operations that require a specific execution order.

The scenario described in this article uses the same development and runtime environment used in Part 1 of this series. The scenario is based on DB2 9.7 Enterprise Server Edition and IBM InfoSphere Warehouse Design Studio 9.7, both of which can be installed on a single Windows® system.

The source table and XML operational data are also the same as discussed in Part 1. The operational data includes information about financial accounts and their investment holdings. This data is derived from a subset of information used to populate customer account records used in the open-source Transaction Processing over XML (TPoX) benchmark. The target data warehouse tables are nearly identical as well, except a referential integrity constraint is now included in the database design.
Listing 1 defines the revised data warehouse tables. The new primary and foreign key constraints are capitalized.

**Listing 1. Definitions for target data warehouse tables**

```sql
-- DWADMIN.ACCT contains general information about accounts.
create table dwadmin.acct (
    id int PRIMARY KEY NOT NULL,
    title varchar(100),
    currency char(3),
    workingbalance int,
    officer varchar(50),
    datechanged date,
    timechanged time,
    totalholdings int,
    holdingtypes int,
    fullrecord xml
)

-- DWADMIN.HOLDINGS tracks specific investments (holdings) for a given account
create table dwadmin.holdings (
    id int REFERENCES DWADMIN.ACCT ON DELETE CASCADE,
    symbol varchar(10),
    type varchar(25),
    quantity decimal(12,3)
)
```

This referential integrity constraint fuels the need to alter the ETL work described in Part 1. In the Part 1 scenario, a single data flow extracts XML data from the INFO column of the source TPOXADMIN.ACCOUNT table and populates two target data warehouse tables: DWADMIN.ACCT and DWADMIN.HOLDINGS. Because no referential integrity constraints exist between those data warehouse tables, it does not matter which of the two target tables is populated first.

The scenario in this article changes the design of the target tables to include primary and foreign key constraints, which requires that the DWADMIN.ACCT table is populated first. To ensure that the DWADMIN.ACCT table is populated first, you need to split the ETL work into multiple data flows and use a control flow to specify the needed execution order for these data flows. Note that if DWADMIN.ACCT is empty, DB2 rejects attempts to insert rows into the DWADMIN.HOLDINGS table, which contains a foreign key referencing the DWADMIN.ACCT table.

To understand how the XML source data in the INFO column of TPOXADMIN.ACCOUNT is mapped to the various columns of the data warehouse tables, review Table 1 and Table 2 in Part 1.

**Understanding an overview of the control flow and data flows**

Before building a control flow, you need to create each data flow individually. Because each data flow is a subset of the overall data flow described in Part 1, the logic should be familiar to you.

**Designing the first flow**

The first flow extracts XML data from the source table and populates the DWADMIN.ACCT table, which contains a primary key, as shown in Figure 1.
The Table Source operator (shown in the upper left of Figure 1) identifies the TPOXADMIN.ACCOUNT table as the input source. XML data in this table feeds two XML Relational Mapping operators. One of these operators (shown on the lower branch in Figure 1) feeds a Group By operator that performs the aggregations required by two of the target table's columns. However, the target table also includes an XML column containing the original XML record. Because XML columns cannot participate in Group By clauses, you need to define a second XML Relational Mapping operator (shown on the upper branch of Figure 1). This operator extracts the account ID value from the XML data, and it preserves the full XML record. The account ID from this mapping operation can then be joined with the account ID included in the output of the Group By operation, creating a result set that populates the target ACCT table.

Designing the second flow

The design of the second data flow is simpler. It populates the DWADMIN.HOLDINGS table, which contains a foreign key that references the DWADMIN.ACCT table. This data flow simply reads the XML data from the source table and uses a single XML Relational Mapping operator to populate the target table directly, as shown in Figure 2.
Designing the control flow

The design of the control flow calls the two data flows just described to populate a hybrid XML/relational data warehouse, as shown in Figure 3.

When initialized, the control flow invokes a data flow that loads the target DWADMIN.ACCT table. Successful execution of this data flow causes the second data flow to be invoked, which loads the target DWADMIN.HOLDINGS table. If the first data flow terminates unsuccessfully, the control flow is programmed to issue an email notification and terminate. Figure 3 shows the control flow that you'll create in this article.
Step 1: Preparing the DB2 database environment

The Download section contains a DB2 script that creates the necessary operational (source) table and data warehouse (target) tables. It also populates the operational table with sample XML account information. This script is similar to the script for Part 1. However, it defines target data warehouse tables with appropriate primary and foreign keys.

Complete the following steps to prepare your DB2 environment.

1. Download the dwsetup2.zip file from the Resources section and extract the files.
2. Open a DB2 command window, and issue the command in Listing 2.

Listing 2. Invoking the DB2 script accompanying this article series

```
db2 -td@ -vf DWsetup2.db2
```

Note that this script is designed to support a DB2 9.7 server running on Windows. If you already have a database named TPOX on your server, this script drops the database and recreates it with new database objects.

Step 2: Preparing your Design Studio environment

Before you create your data flow, prepare the Design Studio environment by completing the steps from Part 1, Step 2: Preparing your Design Studio environment, if you haven’t already.

Step 3: Creating the data flow to extract account data

The first data flow needs to extract information from the source TPOXADMIN.ACCOUNT, transform it as required, and populate the DWADMIN.ACCT target table. Complete the following steps to create the data flow.

1. Copy the data flow you completed in Part 1, and give it a new name.
2. Edit this new data flow, deleting the Table Target operator for the DWADMIN.HOLDINGS table.
3. Edit the XML Relational Mapping operator that feeds the Group By operator to remove the column definitions for SYMBOL and QTY. The column definitions that should remain are id, title, currency, workingbalance, officer, datachanged, timechanged, and holdingtype, as shown in Figure 4.
**Figure 4. Column definitions extracted from XML nodes**

4. Edit the Result Columns for this operator to delete the SYMBOL and QTY columns, as shown in Figure 5.

**Figure 5. Result columns produced by the XML Relational Mapping operator**

5. Return to the graphic view of the data flow design.
6. Right-click the XML Relational Mapping operator that you just edited, and select Propagate Column Change.
7. When prompted, click OK.
8. Edit the Group By operator to verify that the Select List for Aggregation no longer contains expressions related to the SYMBOL and QTY columns, as shown in Figure 6.

Figure 6. Revised select list for Group By operator

9. Save your work. Your new data flow should appear similar to Figure 1.

Step 4: Creating the data flow to extract portfolio holdings data

The next data flow is simpler. It extracts XML information from the source TPOXADMIN.ACCOUNT table, maps specific nodes to relational columns, and populates the DWADMIN.HOLDINGS target table. Complete the following steps to create the data flow to extract portfolio holdings data.

1. Make another copy of the original data flow you completed in Part 1, and give it a new name.
2. Edit this new data flow, deleting all operators except the following:
   - The Table Source operator, which reads information from the TPOXADMIN.ACCOUNT table
   - The Table Target operator for the DWADMIN.HOLDINGS table
   - The XML Relational Mapping operator, which reads data from the source table and populates the DWADMIN.HOLDINGS table.
3. Edit the XML Relational Mapping operator, deleting all column definitions except those required by the target table. The column definitions that should remain are id, holdingtype, symbol and qty, as shown in Figure 7.
4. Delete all result set columns except those required by the target table. The result set columns that should remain are id, holdingtype, symbol, and qty, as shown in Figure 8.

5. Return to the graphic view of the data flow design.
6. Right-click the XML Relational Mapping operator that you just edited, and select Propagate Column Change.
7. When prompted, click OK.
8. Save your work. Your new data flow should appear similar to Figure 2.

**Step 5: Testing the data flows**

Before you build a control flow that invokes each of your data flows, you should verify that each data flow works correctly. Follow the standard Design Studio process as described in the steps below to test the first flow (for populating DWADMIN.ACCT) and then the second flow (for populating DWADMIN.HOLDINGS). You might find it helpful to optionally use the debugger for your initial test. Complete the following steps to test each flow.

1. If you haven't already done so, save your work.
2. Validate the data flow by right-clicking your data flow in the Data Project Explorer pane and selecting **Validate**. If any errors are detected, correct these before proceeding.
3. If desired, set breakpoints in your data flow by right-clicking a connection between operators in your data flow and selecting **Toggle Breakpoint**.
4. In the Data Project Explorer pane, right-click your flow and select **Debug** or **Execute**. If you debug the flow, you can monitor the output of each step in the data flow using the Resume icon at the top. Note that the debugger might generate warnings related to **DROP TABLE** statements, which you can ignore.

**Step 6: Creating your control flow**

With your two data flows developed and tested, you're ready to build a control flow that executes these in the correct order. To keep things simple, you'll create a basic control flow in this article by completing the following steps. More advanced features are available, but these are beyond the scope of this article series.

1. In your data warehousing project, right-click the **Control Flows** folder.
2. Create a new control flow, giving it the name of your choice.
3. From the Palette, select **Common operators > Data Flows**.
4. Drag an icon to the right of the Start operator that appears by default in your control flow design.
5. Right-click and select **Show Properties View**.
6. Click the **...** button to identify the appropriate data flow, as shown in Figure 9.
7. Navigate to the data flow you created earlier to populate the DWADMIN.ACCT table.
8. Optionally change the default name of this control flow operator to something more descriptive.
9. Repeat Steps 6-8 above to add the data flow that populates the DWADMIN.HOLDINGS table to this control flow.
10. Drag this operator to the right of the existing data flow operator.
11. From the Palette, select **Common operators > End**.
12. Place this to the right of the final data flow operator. You now have all the necessary operators to model a successful control flow.
13. Add operators to manage a runtime error condition by selecting **Common operators > Email**.
14. Place this beneath the first data flow operator.
15. Right-click the operator and select **Show Properties View** to specify information for this operator.
16. Identify the email address of the sender and recipient, as well as a subject and message. For this example, use fixed values (rather than variables) for each of these items, as shown in Figure 10. The icon beside each property indicates the use of fixed or variable values.

Figure 9. Edit the properties of the data flow operator to select the desired flow
Figure 10. Defining the Email operator's properties

17. Select **Common operators > Fail** and place this to the right of the Email operator.
18. Connect the Start operator to the input of the first data flow operator.
19. Connect the first data flow operator to the second using the green check mark in the upper right corner to make a conditional connection. A conditional connection executes the control flow only if this operation is successful. In this scenario, a conditional connection is appropriate because if the table containing primary key data cannot be populated, you want to avoid subsequent attempts to populate the table that references it.
20. Connect the second data flow operator to the End operator. For this example, define an unconditional connection using the blue arrow on the output port of the data flow operator.
21. Connect a failure of the first data flow (signified by the red x) to the Email operator below it.
22. Connect the Email operator to the Fail operator using an unconditional connection.
23. Verify that your control flow appears similar to Figure 3.

Step 7: Testing your control flow

After you create your control flow, test it following standard Design Studio procedures. Use the icons on the upper tool bar to validate the control flow and then debug or execute it as desired.

Conclusion

This article series explored how IBM InfoSphere Warehouse Design Studio and DB2 pureXML can be used together to support increased use of XML in data warehouses. XML has emerged as the preferred format for data exchange in many organizations. So, data architects are increasingly evaluating options for integrating XML into their data warehouses. Designing ETL jobs (or data flows) is an important aspect of integrating XML into data warehouse environments.

This article, the second in a two-part series, introduced the use of control flows to manage the execution of multiple data flows. Control flows are useful for complex operations. Control flows help promote code reuse, and they enable developers to specify the sequence of various activities.
Acknowledgments

Thanks to Matthias Nicola, Jing Shan, Lin Xu, and Shumin Wu for reviewing this article series.
Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample DB2 script and data (for Windows)</td>
<td>dwsetup2.zip</td>
<td>100KB</td>
</tr>
</tbody>
</table>

© Copyright IBM Corporation 2010

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)