Chapter 8.

August 2007

Welcome to the August 2007 edition of IBM Information Server Developer's Notebook. This month we answer the question;

> How do I normalize and/or de-normalize data with DataStage EE ?
>
> *While relational databases regularly model data in a form defined as normalized, non-relational data sources often model in a form defined as de-normalized. Moving data to and from a normalized or denormalized form may be a task one needs to complete. Even if you never need to achieve these distinct goals, many other concepts related to DataStage EE, and its various capabilities, are detailed below.*
>
> *To complete the examples below, you need slightly more than beginner level of capability with the DataStage EE product.*

In short, we are going discuss all of the relevant terms and technologies above, and provide examples that detail how to deliver this functionality using IBM Information Server.

Software Versions

All of these solutions were *developed and tested* on IBM Information Server version 8.01, on the Microsoft Windows XP/SP2 platform. IBM Information Server allows for a single, consistent, and accurate view of data across the full width of the corporate enterprise, be it relational or non-relational, staged or live data. As a reminder, the IBM Information Server product contains the following major components;

> WebSphere Business Glossary™, WebSphere Information Analyzer™, WebSphere Information Services Director™, WebSphere DataStage™, WebSphere QualityStage™, WebSphere Metadata Server and Metabridges™, WebSphere Metadata Workbench™, WebSphere Federation Serve™r, Classic Federation™, Event Publisher™, and Replication Server™, and Rational Data Architect™.

Obviously, IBM Information Server is a large and capable product, addressing many strategic needs across the enterprise, and supporting different roles and responsibilities. This month we give focus to the DataStage EE component of IBM Information Server.

8.1 Terms, terms, terms

While the term *databank* first appeared in the Washington Post circa 1966, the popular term used today, database, was first widely mentioned in Europe as early as 1968. A *database* is an organized collection of information.

There are many *database modelling theories*, including the early pioneering CODASYL IDMS *networked database model*, and the Rockwell and then IBM IMS *hierarchical database model*. The function and performance of early computing platforms, CPU and memory capacity, as well as DASD (hard disk) performance, greatly affected the design and function of early database models. Today, the defacto choice for new database applications support the *relational database model*. Relational database design is centered on the concept of *data normalization*, which we define by example below;

- In a relational model, a customer order is regularly modelled in two related tables, two distinct lists of records. (Records are comprised of rows and columns, much like a modern spreadsheet.) By example;
  - The OrderHeader table may contain attributes (columns) related to the date that the order was placed, the customer's identity, the method of payment, delivery address, and so on.
  - There is one OrderHeader record for every order that is received, and every order in the OrderHeader table is identified by a unique order number, also called the *primary key column(s)*.
  - The OrderLineItem table contains one record for every item that the order contains; one order with three distinct items equals 4 total records, one OrderHeader record, and three OrderLineItem records, one per item.
  - The OrderLineItem table may contain attributes related to the stock number for a given order line item, the quantity requested, and so on.
  - Every record in the OrderLineItem table also has primary key columns, so that each order line item may be uniquely identified; read, updated, deleted, etcetera. Records in the OrderLineItem table also have *foreign key column(s)*, which identify the specific order record these line items relate to.
- Relational databases have a formal set of stated modelling rules that are used to validate a given application, used to validate a given data model. These rules are called *normal forms*. There is *first normal form*, *second normal form*, *third normal form*, and so on. These forms are accumulative; meaning, to be in second normal form, you must first be in first normal form. Third normal form requires you to be in second and also first normal form.

When a data model follows at least the first three normal forms (it is in third normal form), that model is said to be *normalized*. If the model violates third normal form, it is said to be *de-normalized*.

– An example of the SQL syntax to create the OrderHeader and OrderLineItem tables, along with sample data, are listed in Example 8-1.

Example 8-1 OrderHeader and OrderLineItem tables, with sample data.

```
-- Example of normalized tables and data.

CREATE TABLE OrderHeader
    (
    OrderNumber INTEGER NOT NULL,
    CustomerName CHAR(20)
    );
ALTER TABLE OrderHeader ADD CONSTRAINT PRIMARY KEY
    ON (OrderNumber);

-- Sample data for OrderHeader table delimited by vertical bar symbol.

    101|Dave W|
    102|Emily M|
    103|Paul M|

CREATE TABLE OrderLineItem
    (
    OrderNumber INTEGER NOT NULL,
    LineItemNumber INTEGER NOT NULL,
    ItemName CHAR(20),
    ItemQuantity INTEGER
    );
ALTER TABLE OrderLineItem ADD CONSTRAINT PRIMARY KEY
    ON (OrderNumber, LineItemNumber);
ALTER TABLE OrderLineItem ADD CONSTRAINT FOREIGN KEY
    (OrderNumber) REFERENCES OrderHeader;

-- Sample data for OrderLineItem table delimited by vertical bar symbol.

    101|1|Blue Jeans|2|
    101|2|Socks|6|
    101|3Tie|1|
    101|4|Hat|3|
    102|1|Dress|1|
    103|1|Tuxedo|1|
    103|2|Ring|1|
    103|3|Airline tickets|2|
```


The table definitions (the data model) represented in Example 8-1 is in third normal form; the model and its accompanying data is normalized. These tables and data could also be modelled in a de-normalized form, example as shown in Example 8-2.

Example 8-2 De-normalized example of tables and data from Example 8-1.

```
-- Example of a denormalized table and data.

CREATE TABLE Order
    (
    OrderNumber INTEGER NOT NULL,
    CustomerName CHAR(20),
    ItemName1 CHAR(20),
    ItemQuantity1 INTEGER,
    ItemName2 CHAR(20),
    ItemQuantity2 INTEGER,
    ItemName3 CHAR(20),
    ItemQuantity3 INTEGER,
    ItemName4 CHAR(20),
    ItemQuantity4 INTEGER
    );
ALTER TABLE Order ADD CONSTRAINT PRIMARY KEY
    ON (OrderNumber);

-- Sample data for Order table delimited by vertical bar symbol.

    101|Dave W|Blue Jeans|2|Socks|6|Tie|1|Hat|3|
    102|Emily M|Dress|1|0|0|0|0|0|
    103|Paul M|Tuxedo|1|Ring|1|Airline tickets|2|0|
```

The Order table, its definition and accompanying data, is said to be de-normalized because it violates, in this case, first normal form. First normal form prohibits repeating column definitions; (Item name data is repeated within the table, ItemName1, ItemName2, and so on. A normalized table would have these as a single column, but then allow numerous items to be represented by numerous records in one table, as is shown in Example 8-1.)

Both normalized and de-normalized forms of data have advantages over one another, and it is not our place to debate such things here. What we will do now is demonstrate how to use the DataStage EE component of IBM Information Server to move data between these two forms.

8.2 Normalizing data using DataStage EE

First we are going to take de-normalized data and use DataStage EE to normalize it. De-normalized data is often received from a non-relational data source; be it an electronic switch, reading of a log file, a non-relational database table, a relational database table that is de-normalized, whatever. We read one input file and create two output files.

We are going to perform all of this activity inside DataStage EE, and the example table and data we are going to use comes from Example 8-2. To keep the example as simple as possible, we are going to read to and from ASCII text files. You could easily adjust this example to read to and from database tables.

1. Create an ASCII text file of data equal to what is shown in Example 8-2; three rows, ten columns each, vertical bar delimiters, and zeros for place holders on columns that are basically not in use.
2. Launch the DataStage EE Designer component to IBM Information Server, and create a new Parallel Job. Save the Parallel Job with a given name.
3. Inside the DataStage EE Designer, complete the following;

   From the Palette View, drag and drop the operators to the Parallel Canvas as shown in Figure 8-1.

   – InputFile1, OutputFile1, and OutputFile2 are all Sequential File objects from the File Drawer of the Palette View.

   – Every other object in Figure 8-1 is named per its source object in the Palette View; Copy, Transformer, Funnel and Filter.

   – Connectors are dragged between each object pair using a right-click of the mouse. There are four connectors between Transformer and Funnel, a condition we explain later.

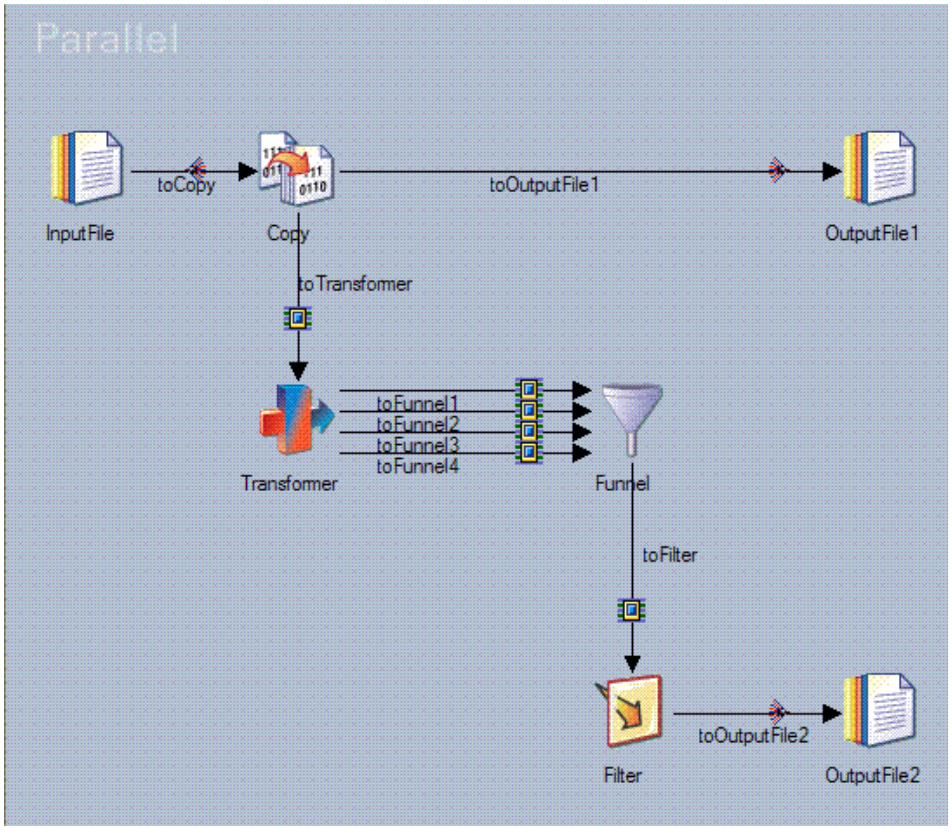


Figure 8-1 Initial drag and drop of operators from Palette View to Parallel Canvas.

Sequential File object, InputFile

4. Double-click the InputFile (Sequential File) object and complete the following;
   a. On the Output TAB -> Properties TAB -> Source -> File visual control, browse to the location of the ASCII text file referred to in Step-1 above.
   b. On the Output TAB -> Format TAB -> Record level -> Final delimiter visual control, set the delimiter to equal a vertical bar, (ASCII 127, "|").
   c. On the Output TAB -> Format TAB -> Field defaults -> Delimiter visual control, set the delimiter to equal a vertical bar.
   d. On the Output TAB -> Format TAB -> Field defaults -> Quote visual control, use the associated drop down list box to select, None.
   e. On the Output TAB -> Columns TAB, manage the display to equal what is displayed in Figure 8-2.

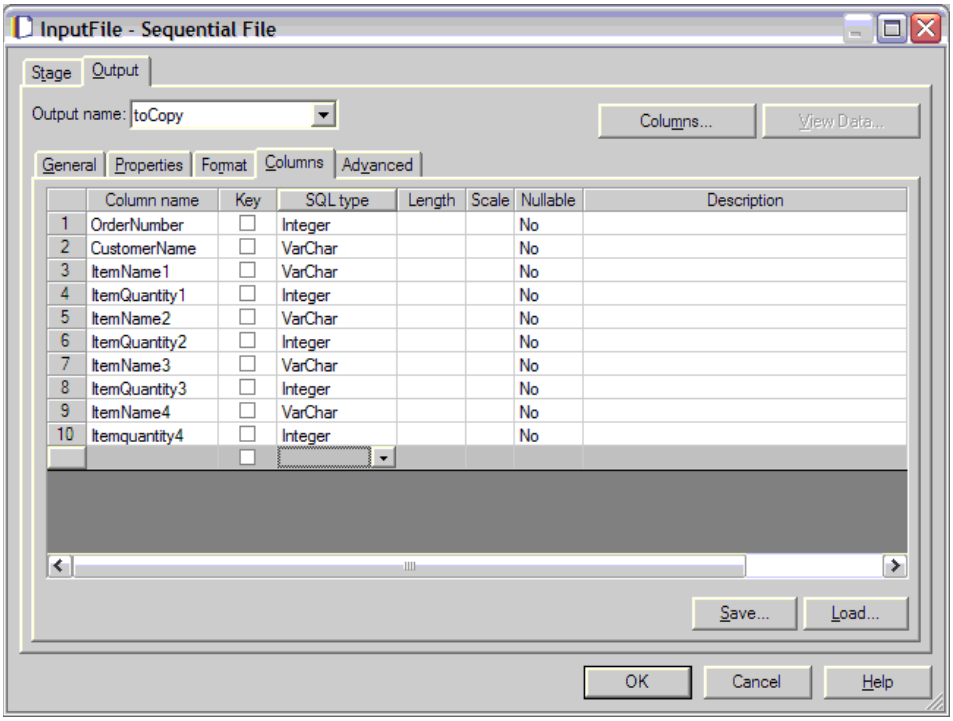


Figure 8-2 Input File -> Output TAB -> Columns TAB, defining input column properties.

f. When you are done creating Figure 8-2, save this table definition via the Save button, and follow the associated prompts. Call the saved table definition, Order.

Note: *We use our saved table definitions later many times to save steps.*

g. When you are done creating Figure 8-2, click the OK button to close the InputFile, Sequential File operator.

Copy operator

5. Double-click the Copy operator and complete the following;

   a. On the Output TAB -> Mapping TAB -> Output name "toOutputFile1" display, drag and drop the OrderNumber and CustomerName columns from the left side of the display to the right. Example as shown in Figure 8-3.

      OutputFile1 contains data destined for the OrderHeader table, which contains these two columns (OrderNumber and CustomerName) only. Per

the conditions in our source data file, no other operations are required to create the output data; no sort, no remove duplicates, etcetera.

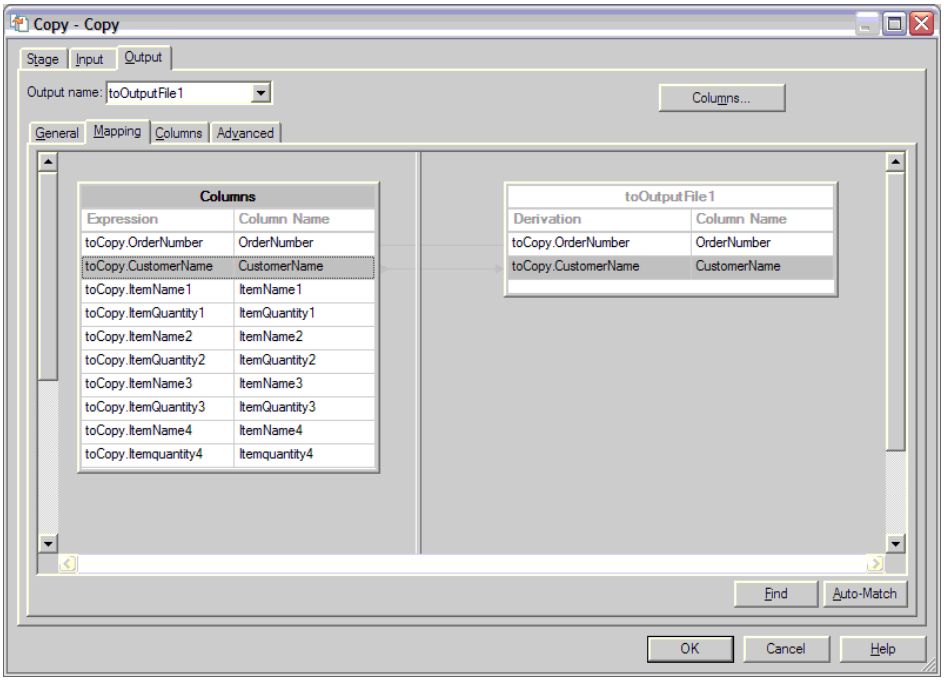


Figure 8-3 Copy operator -> Output TAB -> Mapping TAB -> toOutputFile1 display.

b. On the Output TAB-> Mapping TAB -> Output name "toTransformer" display, we want to **drag and drop all of the columns from the left side of the display minus one**; we don't want to send the CustomerName column to the Transformer operator.

   The quickest way to accomplish this goal is to drag and drop all columns, then delete the single undesired column. You can drag and drop all columns by dragging the word "Columns" from the table header on the left side of the display to the right. This will copy all columns.

   Then, on the right side of the display, right-click the CustomerName column and select, Delete Column -> Yes to All.

c. Click OK when you are done to close the Copy operator.

Sequential File object, OutputFile1

6. Double-click the OutputFile1 (Sequential File) object and complete the following;

a. On the Input TAB -> Properties TAB -> Target -> File visual control, browse to a directory location of your choosing, and give this output file a name. We used the same directory as Step-4.a above, and called the file, OutputFile01.txt.

b. On the Input TAB -> Format TAB, we want to set three properties; Record level -> Final delimiter, Field default -> Delimiter, and Field default -> Quote.

   We just set these same three parameters and values in Step-4.b through Step-4.d.

   We can save steps by clicking the Load button, and browsing to the Order table definition we created in Step-4.f.

c. On the Input TAB -> Columns TAB, there is no activity we need to perform; the runtime column propagation feature of DataStage EE populated these visual controls for us.

   We do however want to save this table definition to save us even more steps later.

   Click the Save button, and save this table definition with the name, OrderHeader.

d. Click the OK button to close the OutputFile1, Sequential File operator.

Transformer operator

7. Double-click the Transformer operator and complete the following;

   The dialog for the Transformer operator appears in Figure 8-4.

   - The upper left portion of the display in Figure 8-4 will automatically be filled with in with the input columns to this operator. This portion of the display gives us an area from which we can drag and drop.
   - The bottom left portion of the display gives more detail to these input columns; their data types and so on.
   - The top right portion of the display in Figure 8-4 has already had some activity done, and we will do this activity next; your display will appear slightly different at first.

     At first you should have at least four boxes, labelled whatever your four output links are from this operator. We called our output links; toFunnel1, toFunnel2, toFunnel3 and toFunnel4.

Note: *You can change the name of any link by right-clicking the link on the Parallel Canvas, and selecting, Rename. At this point you'd have to exit the dialog for the Transformer to return to the Parallel Canvas.*

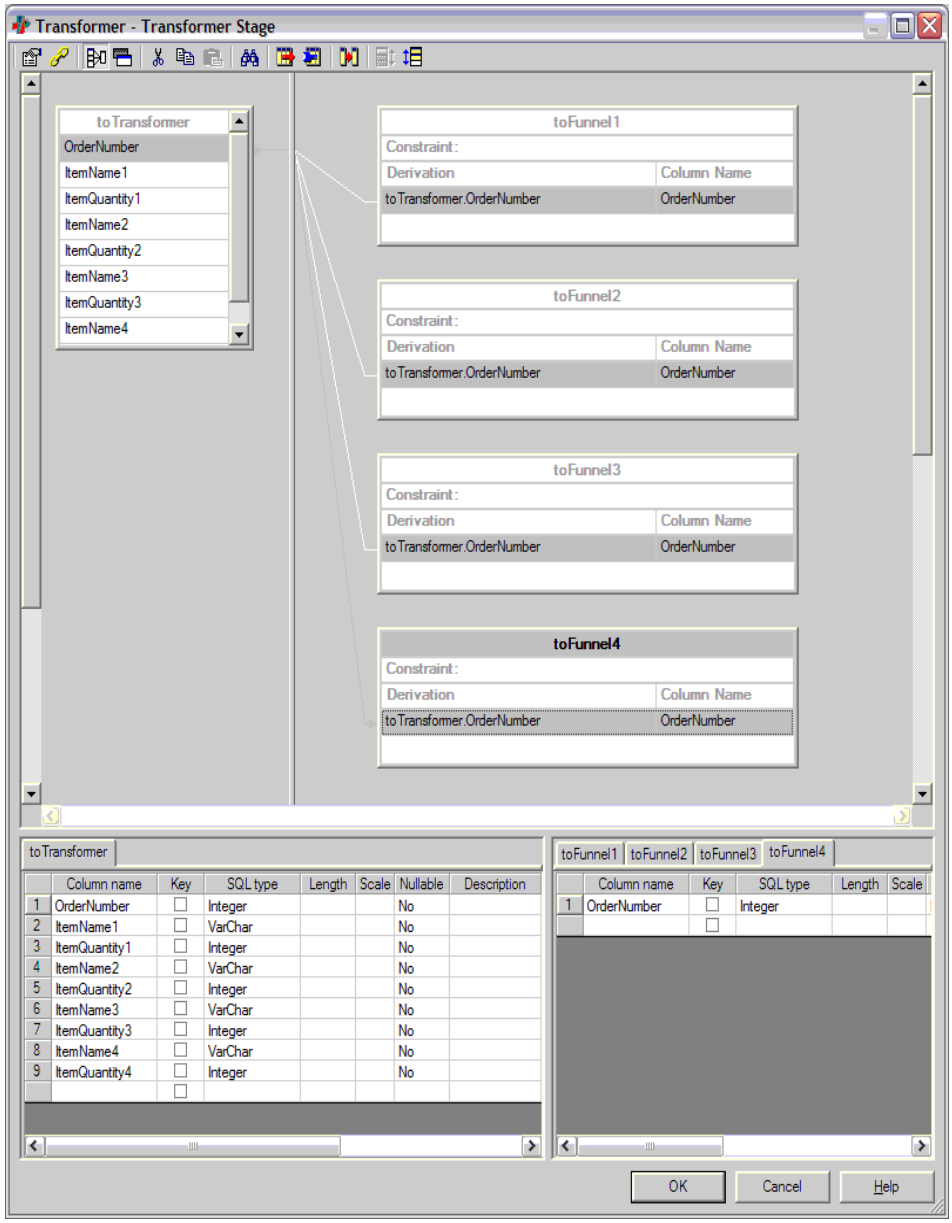


Figure 8-4 Transformer operator, Step 1 of 3.

- If you see an additional box in the upper right portion of the display entitled, Stage variables, no worries, we don't need this box. If you wish

to suppress display of this additional box, an icon in the toolbar can suppress its display.

- The bottom right portion of the display gives us more detail and control over what is displayed in the upper right portion of the display.

a. In Figure 8-4, drag and drop OrderNumber from the upper left portion of the display to each of the four boxes named, toFunnel1, toFunnel2, etcetera. You will need to drag and drop four times, one per box.

   A completed example appears in Figure 8-4.

b. Now we need to drag and drop several columns from the upper left portion of the display to the upper right.

   i. Drag and drop ItemName1 to the box entitled toFunnel1.

   ii. Drag and drop ItemQuantity1 to the box entitled toFunnel1.

   iii. Drag and drop ItemName2 and ItemQuantity2 to toFunnel2.

   iv. Complete this activity for the numbers 3 and 4 columns.

   A completed example appears in Figure 8-5. Give focus to the right side of the display, and the column names, location, and order in which they appear. Don't worry so much about the connecting lines, as they are merely for appearance.

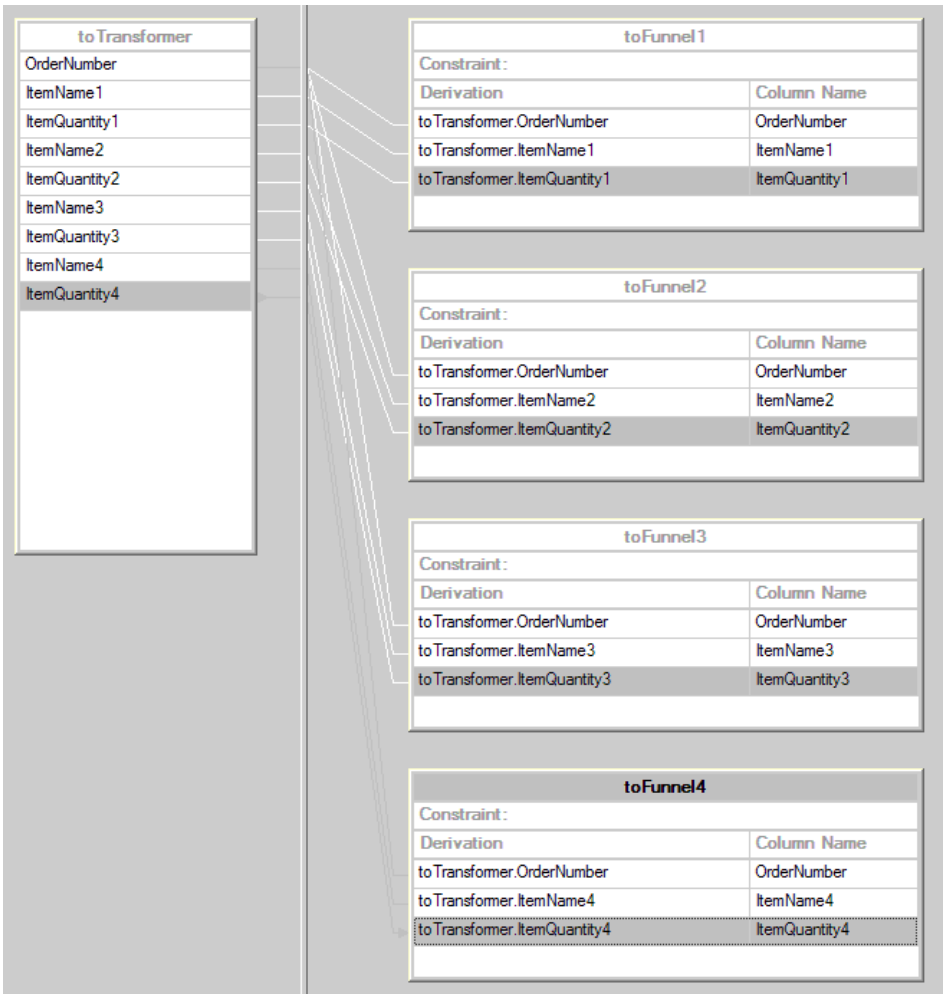


Figure 8-5 Transformer operator, Step 2 of 3.

Note: *There are two logical steps remaining in the completion of the Transformer operator, and after that, the remaining operators are easy.*

c. Now we want to add a new output column to each output link. (The output links are the boxes in the upper right portion of the display entitled, toFunnel1, toFunnel2, etcetera.) We will list detailed instructions for the first output link, and ask you to repeat these instructions for the remaining three output links;

i. In the upper right portion of the display, in the box entitled, toFunnel1, on the line entitled, ItemName1, right-click and select, Insert New Column.

   At first this newly inserted line will appear back lit in red because it is initially incomplete.

ii. On this red line which we just added, right-click on the left column. (The column header for this column is labelled, Derivation.

   When you right-click, you will get a context menu. Select, Edit Derivation.

   This action will open a small multi-line editor wherein you can edit the contents of this cell.

iii. Enter the literal single character 1 in this cell, and press the ENTER key on your keyboard.

iv. This newly added column received a default column name of New, and a default data type of character. These properties are changed in the bottom right portion of the display, which is TABBED, one TAB per output link.

   In the bottom right portion of the display, make the TAB for the toFunnel1 output link current.

v. In the bottom right portion of the display change the column named New, to equal LineItemNumber. This is done by right-clicking the given cell, and selecting, Edit Cell.

   Change the SQL Type for this column to type Integer. This is done via a single-click in the given cell, after which a drop down list box will appear.

   A completed example appears in Figure 8-6.

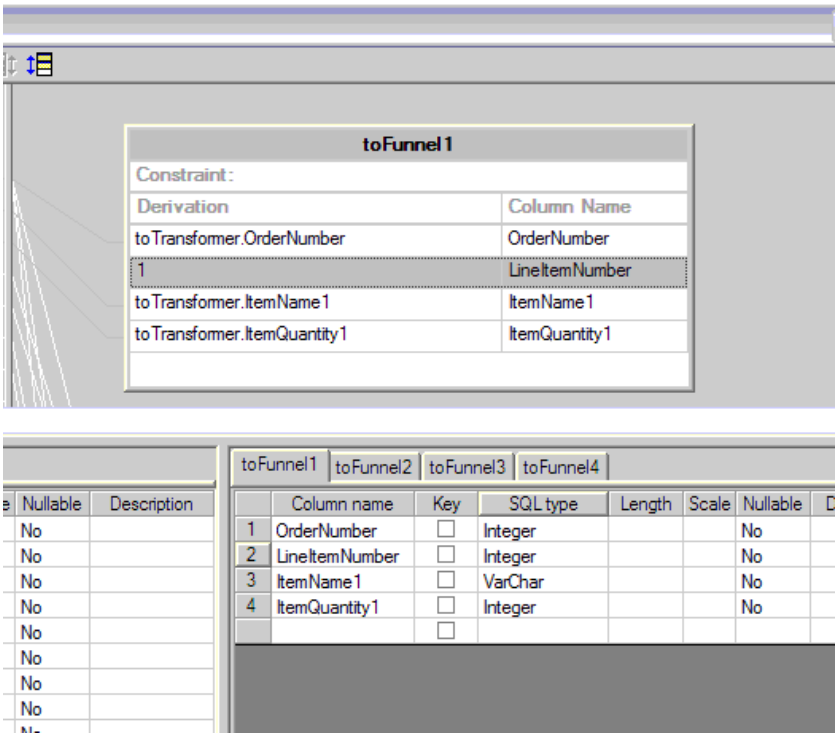


Figure 8-6 Transformer operator, Step 3 of 3.

d. In order for our next operator (the Funnel) to function correctly, we need to change the output column names as they appear in the bottom right portion of the display.

   In the bottom right portion of the display, change the column name ItemName1 to equal ItemName. Change ItemQuantity1 to equal ItemQuantity.

   You can change these column names by right-clicking on their given cell, and selecting, Edit Cell

e. Now we need to repeat Step-7.c and Step-7.d above for the remaining three output links, toFunnel2, toFunnel3, and toFunnel4. The following exceptions are listed;

   - In Step7-c.iii we entered the literal character 1 for the output link entitled, toFunnel1.

     Use the number 2 for toFunnel2, 3 for toFunnel3, and 4 for toFunnel4.

     These literal numbers are used to preserve line item column sort order in our newly created normalized data model.

- IN Step7-d, we changed columns name for ItemName1 and ItemQuantity1.

  As we repeat these steps for the remaining output links, the columns to change will be called ItemName2, ItemName3, etcetera.

  In all cases we are essentially dropping the trailing numeric from the column name.

  A completed example for the Transformer operator appears in Figure 8-7.

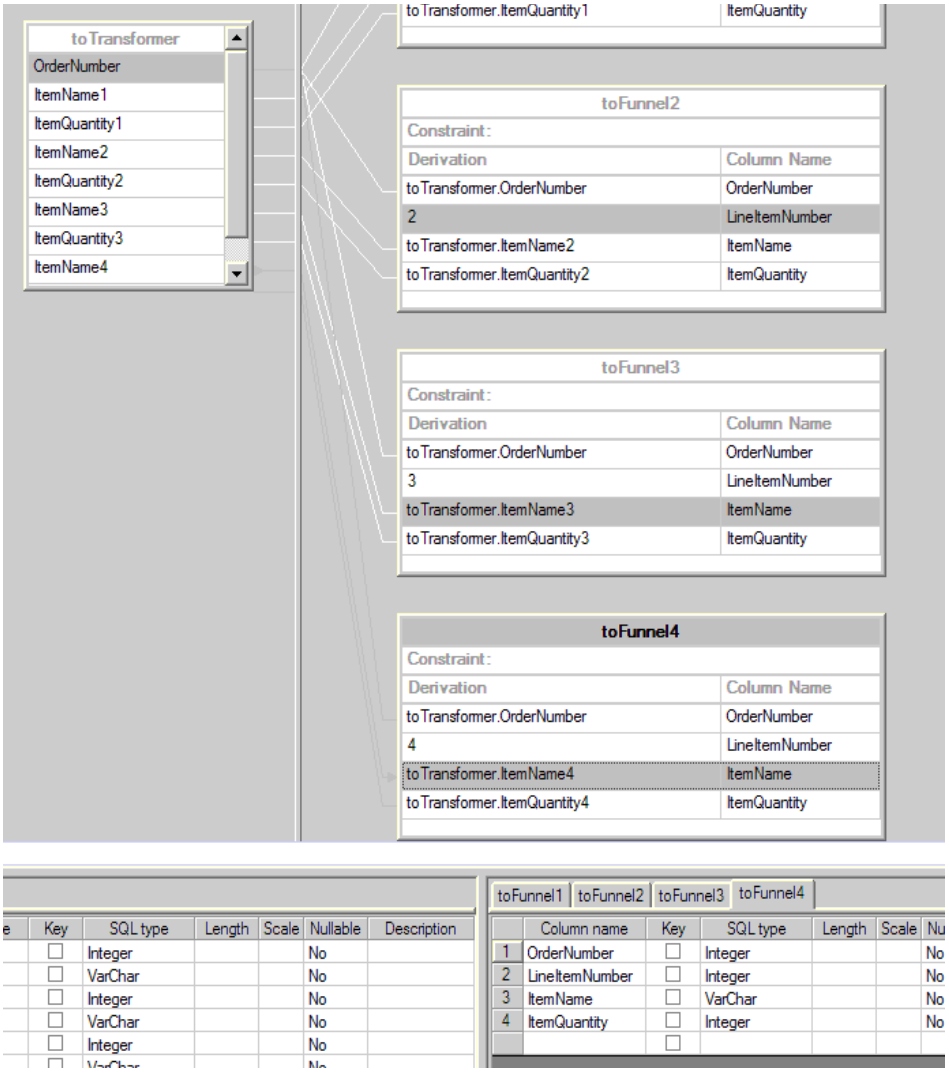


Figure 8-7 Transformer operator, when complete.

f. Check your activity before exiting the Transformer operator-

When you TAB through each of the four TABs in the bottom right portion of the display, each display should equal the other three. There should be no differences in this area of the display.

The main difference in the upper right portion of the display is the presence and locations of the literal numbers; 1, 2, 3 and 4.

If you have errors here, they will display themselves by giving you issues in configuring the next operator, the Funnel operator.

g. Close the Transformer operator with a click on the OK button.

Note: *In place of the Transformer operator here we could have used the Pivot operator. The Pivot operator is purpose built to perform the exact function we just did with a Transformer; a horizontal pivot of repeating columns into one or more records.*

The reason we choose Transformer over Pivot was record enumeration; meaning, we wanted to preserve the numeric sort order of the records we just produced so that, for example, line item one is followed by line number two, and so on. The Pivot operator does not have enumeration as a built in capability. We could achieve enumeration by linking a Pivot operator to a Column Generator operator or similar device, but using one Transformer operator versus a Pivot operator and a second support operator was our preference.

Lastly, while there is a (Horizontal) Pivot operator built into DataStage, there is not a built in reverse operator, something like a Vertical Pivot. That is understandable since there would be almost an endless variety of needs describing how to perform a vertical pivot. Better we should use the existing and robust Palette of operators that are already built into DataStage for this purpose. Our next example, Section 8.3. demonstrates how to perform a vertical pivot.

Funnel operator

8. Double-click the Funnel operator and complete the following;

a. On the Output TAB -> Mapping TAB, drag all columns from the left side of the display to the right. This is accomplished by dragging the table header entitled, Columns, from the left side of the display to the right.

If you are not able to perform this action, this indicates an error exists in Step-7 above.

A completed example appears in Figure 8-8.

b. Click the OK button to close the Funnel operator.

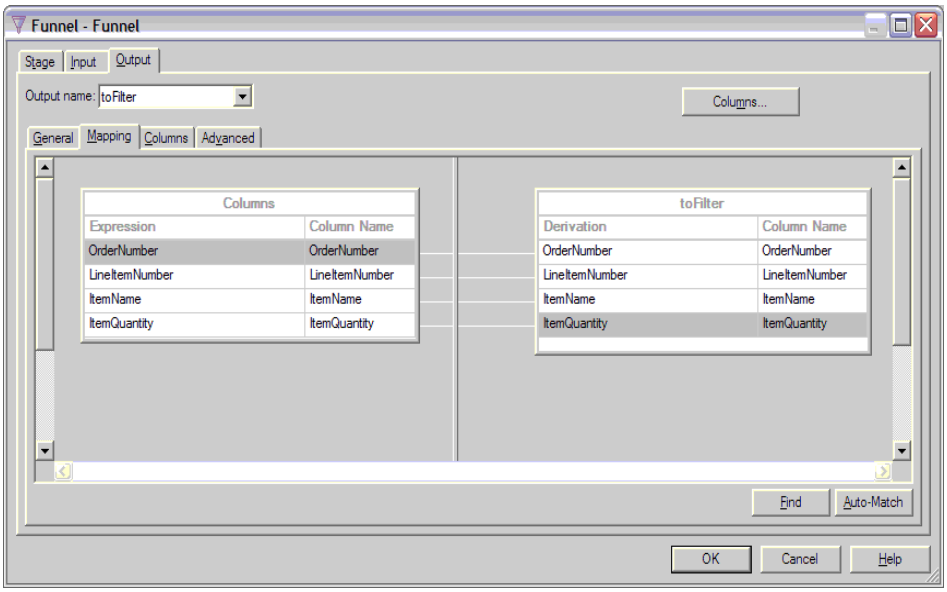


Figure 8-8 Completed Funnel operator.

Filter operator

9. Double-click the Filter operator and complete the following;

   a. On the Stage TAB -> Properties TAB -> Properties -> Where Clause visual control, edit the assigned value to equal, ItemQuantity > 0.

      Example as shown in Figure 8-9.

      De-normalized data has a fixed number of columns which may or may not be populated with real data; some of the allocated columns may be empty. Our source data was specifically created to demonstrate the condition of having empty data columns.

      This filter criteria will now remove empty data columns from our data stream.

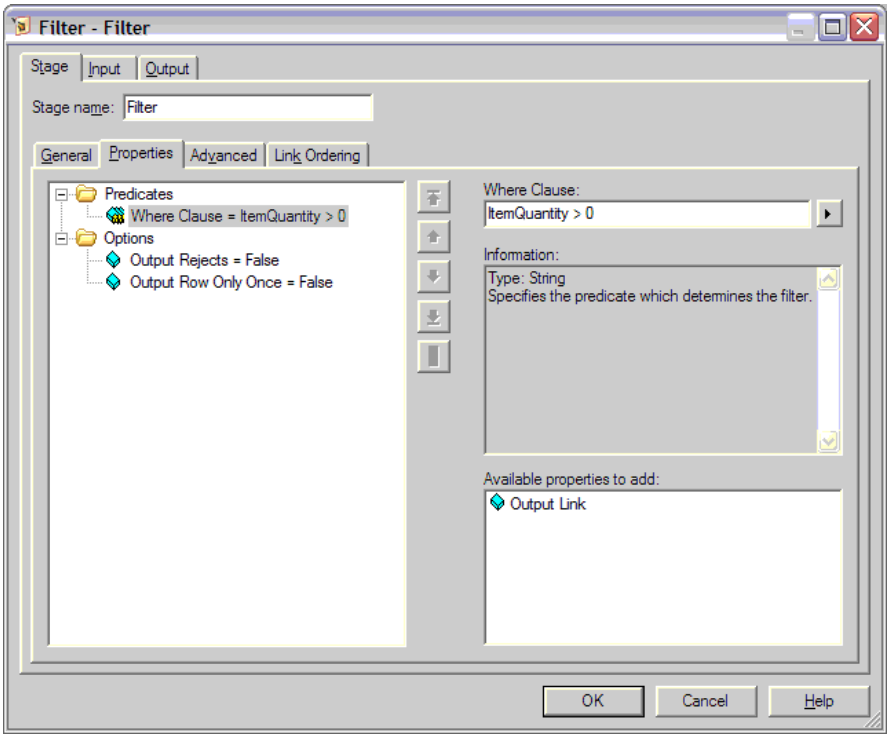


Figure 8-9 Filter operator, Step 1 of 2.

b. On the Output TAB -> Mapping TAB, drag and drop all columns from the left side of the display to the right.

   Example as shown in Figure 8-10.

c. Click the OK button to close the Filter operator.

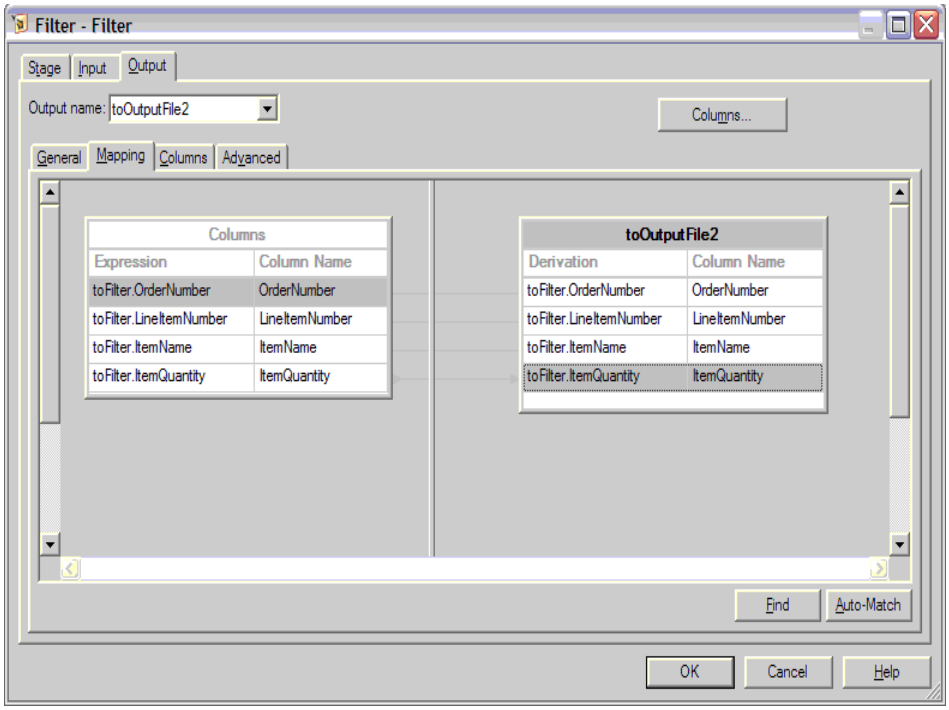


Figure 8-10 Filter operator, Step 2 of 2.

Sequential File object, OutputFile2

10. Double-click the OutputFile2 (Sequential File) operator and complete the following;

    a. On the Input TAB -> Properties TAB -> Target -> File visual control, manage this control as we did in Step-6.a. Name this output file, OutputFile02.txt.

    b. On the Input TAB -> Format TAB, complete the activity as we did in Step-6.b.

    c. On the Input TAB -> Columns TAB, again, all of our activity is done for us automatically by DataStage. We do again want to save our activity as we did in Step-6.c. Name this saved table definition, OrderLineItem.

    d. Close this operator by clicking on the OK button.

Save, compile and test

11. Finally it is time to save, compile and test our activities.

    a. Save, compile and run this Job.

b. To test your accomplishment, browse to and examine the two output data files. If everything is correct, these output data files should resemble the sample data from Example 8-1.

   If the record sort order of your data does not equal the sample data from Example 8-1, don't worry. We did not specify any output data sorting.

   We will prefer instead to sort the data in our next Job, the load, when the sort is minimally required.

   Lastly, we will use these two newly created output files as input for our next Job. If the next Job works correctly, we will have effectively round tripped the data and its format; a very good test as to the accuracy of our activities.

8.3 De-normalizing data using DataStage EE

In the last section, we took a single de-normalized data file and normalized it; outputting two new normalized data files. In this section we complete the round trip back to a de-normalized and single data file. If the new file we are about to output equals our original source file, we have proof as to the accuracy of our activities.

As a result of this document's design, this section has a dependency that you have previously completed section 8.2; we re-use the data files and table definitions.

12. Launch the DataStage EE Designer component to IBM Information Server, and create a new Parallel Job. Save the Parallel Job with a given name.
13. Inside the DataStage EE Designer, complete the following;

    From the Palette View, drag and drop operators to the Parallel Canvas as shown in Figure 8-11.

    – InputFile, lookupFile, and OutputFile are all Sequential File objects from the File Drawer of the Palette View.
    – Every other object in Figure 8-11 is named per its source object in the Palette View; Transformer, Aggregator and Lookup.
    – Connectors are dragged between each object pair using a right-click of the mouse.

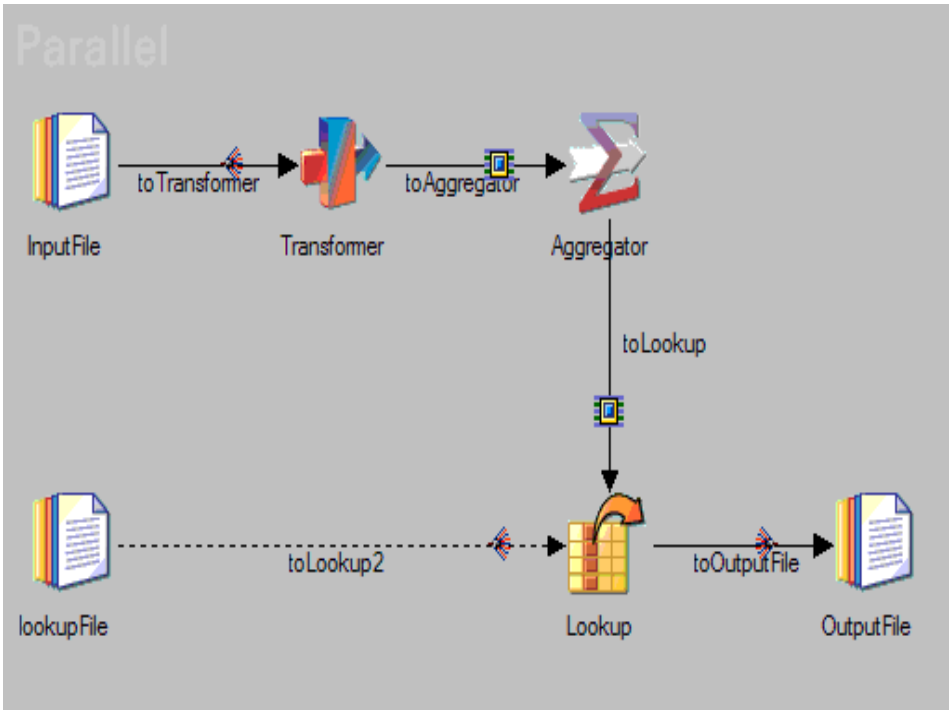


Figure 8-11 Initial drag and drop of operators from Palette View to Parallel Canvas.

Sequential File object, InputFile

14. Double-click the InputFile (Sequential File) object and complete the following;

    a. On the Output TAB -> Properties TAB -> Source -> File visual control, browse to the location of the ASCII text file we specified in Step-10.a, OutputFile02.txt.

       Our previously created output file is going to now serve as our input file.

    b. On the Output TAB -> Format TAB, click the Load button and browse for the previously saved, OrderLineItem table definition we saved in Step-10.c.

    c. On the Output TAB -> Columns TAB, again we are going to load previously saved work. Again, browse to the OrderLineItem table and load its previously saved column definitions.

       There are four columns that should load; OrderNumber, LineItemNumber, ItemName and ItemQuantity.

    d. Click the OK button to close the InputFile, Sequential File operator.

Transformer operator

15. Double-click the Transformer operator and complete the following;

    a. As we have seen before, the Transformer operator screen is organized into four regions of the screen.

       From the upper left portion of the screen, drag and drop the OrderNumber column to the table in the upper right portion of the screen.

    b. In the bottom right portion of the screen, add eight new columns after OrderNumber. Name these eight new columns; ItemName1, ItemQuantity1, ItemName2, and so on.

       Give these new columns data types, and other properties as shown in Figure 8-12.

       Initially these eight new columns will appear in red in the upper right portion of the display because their derivation is incomplete.

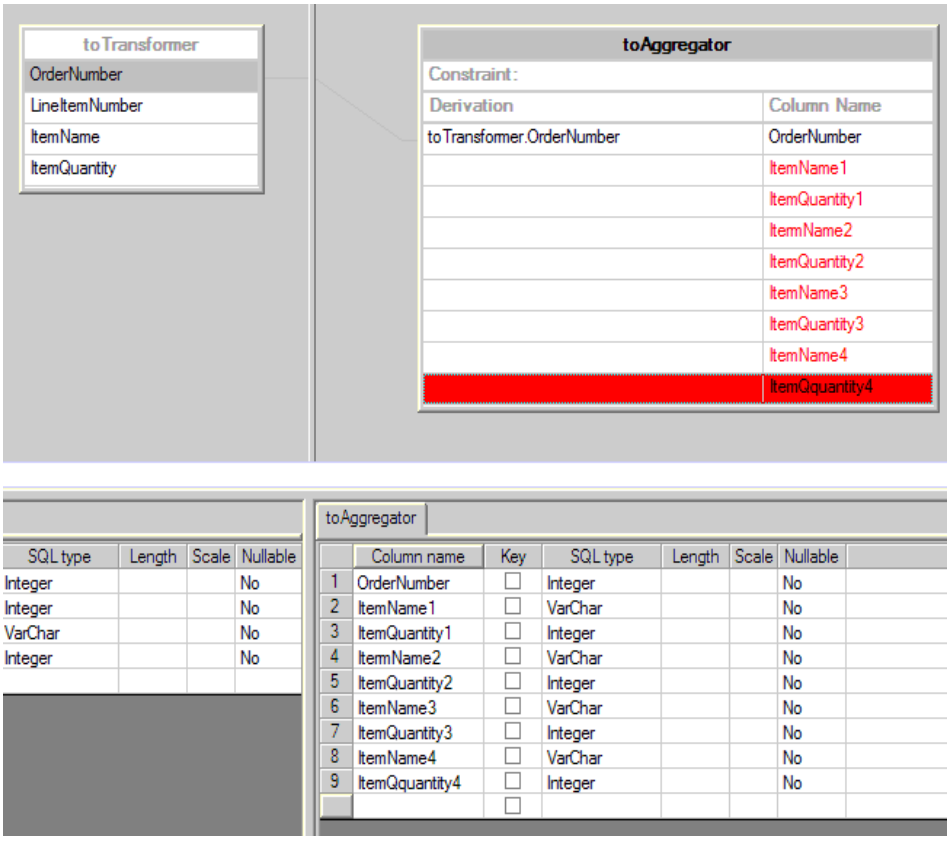
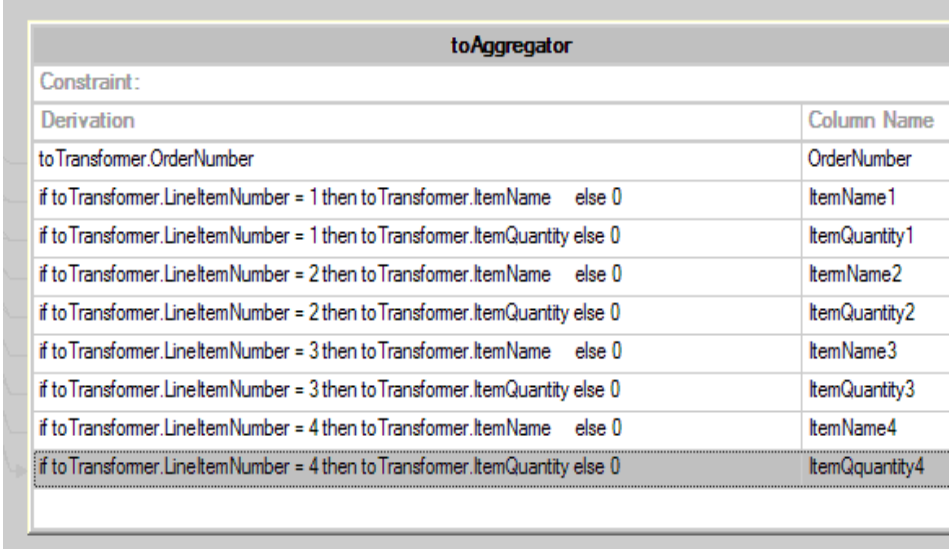


Figure 8-12 Transformer operator, Step 1 of 2.

c. Our last remaining task before completing the Transformer operator is made easier by using cut and paste; a reminder, Control-C in any cell copies, Control-V pastes.

   Manage the Transformer screen, upper right portion of the display, to equal what is displayed in Figure 8-13.



toAggregator

Constraint:

Derivation	Column Name
toTransformer.OrderNumber	OrderNumber
if toTransformer.LineItemNumber = 1 then toTransformer.ItemName else 0	ItemName1
if toTransformer.LineItemNumber = 1 then toTransformer.ItemQuantity else 0	ItemQuantity1
if toTransformer.LineItemNumber = 2 then toTransformer.ItemName else 0	ItemName2
if toTransformer.LineItemNumber = 2 then toTransformer.ItemQuantity else 0	ItemQuantity2
if toTransformer.LineItemNumber = 3 then toTransformer.ItemName else 0	ItemName3
if toTransformer.LineItemNumber = 3 then toTransformer.ItemQuantity else 0	ItemQuantity3
if toTransformer.LineItemNumber = 4 then toTransformer.ItemName else 0	ItemName4
if toTransformer.LineItemNumber = 4 then toTransformer.ItemQuantity else 0	ItemQquantity4

Figure 8-13 Transformer operator, Step 2 of 2.

Examining Figure 8-13 we see that we pasted nearly the same "if" statement 7 times, after creating it once for ItemName1. The expression you are pasting differs by only one numeric value, and then by the column name ItemName or ItemQuantity.

Again, make the upper right portion of the display equal to what is displayed in Figure 8-13. Use copy (Control-C) and paste (Control-V) to save time.

Note: *Here the Transformer is again acting as a pivot operator, although in this case it is acting as a vertical pivot operator. As each input record is processed with an Item Name and Item Quantity pair, four new column pairs of Item Name and Item Quantity are output. If the Line Item Number column is of the value we prefer, we output a real value, else we output a zero. The Aggregator operator that follows will roll up the real values, and suppress the presence of these place holder style zeros.*

d. Click the OK button to close the Transformer operator.

Aggregator operator

While the previous operator in our stream, the Transformer, did our pivot and output four sets of columns for every one source row, the source data still exists as many as four rows where we need to have only one. Now we use the Aggregator operator to collapse these numerous rows into one. We aggregate on (group on) Order Number and output the maximum value for any item quantity or item name.

16. Double-click the Aggregator operator and complete the following;

    a. Figure 8-14 displays what we need to perform on the Stage TAB -> Properties TAB.

       Set the Grouping Keys -> Group visual control value equal to the OrderNumber column.

    b. Under Aggregations you will be given one column to aggregate which you must set. Then you will need to add the remaining seven columns we wish to aggregate on.

       A right-click on Aggregations will allow you to select, Aggregate -> Add sub-property -> Column for Calculation, as well as other properties later.

       As an initial task, make your Aggregator operator, Stage TAB -> Properties TAB equal what is displayed in Figure 8-14.

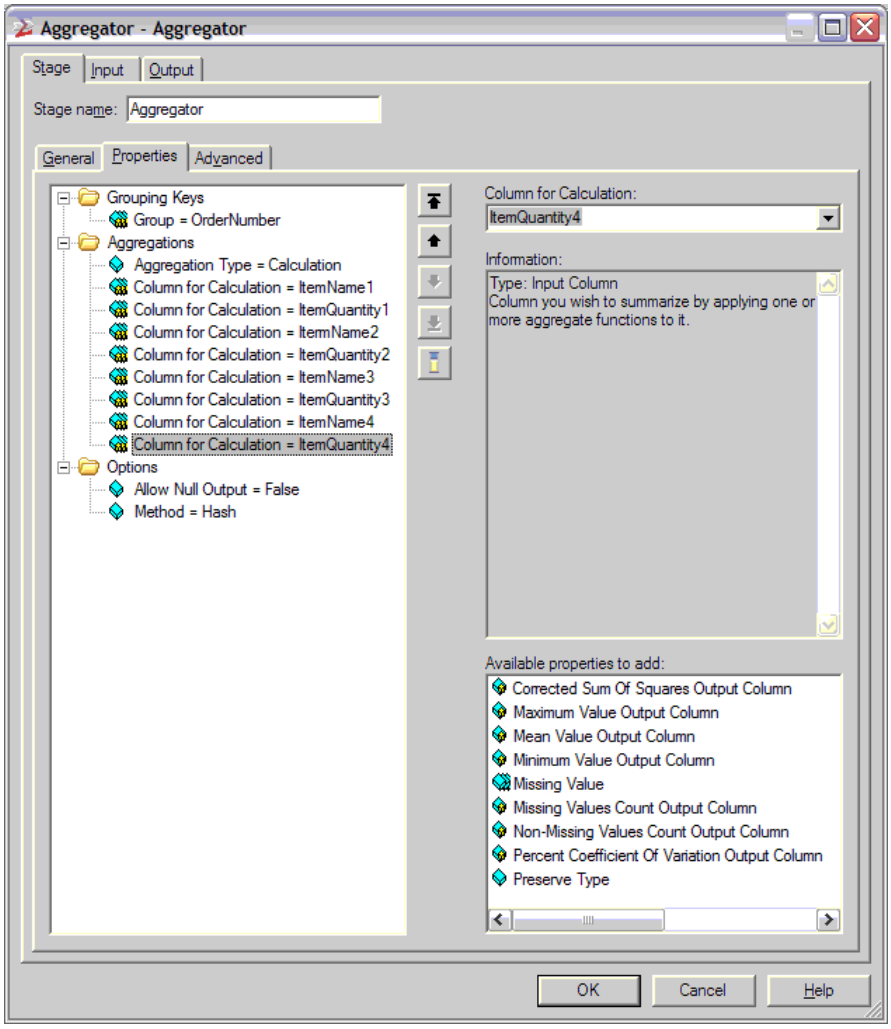


Figure 8-14 Aggregator operator, Step 1 of 4.

c. Continuing with the Aggregator operator, Stage TAB -> Properties TAB, add the remaining properties as displayed in Figure 8-15.

   In effect we previously added the column name to calculate. Now we are adding the specific calculation, and for the character type columns, one additional property.

   For each of the eight columns we added, specify a call to calculate the maximum value of the column. You can add this property via the "Available properties to add" pick list, or via a right-click on the "Column for Calculation" entry in the left pane of the display.

d. For each of the character type columns, we need to add a "Preserve (data) Type" property.

   We are calculating a maximum value for each column, which by default, outputs a numeric value. In the case of Item Name, we wish to output a character, not a numeric.

   You are finished on this TAB when your display equals what is shown in Figure 8-15.

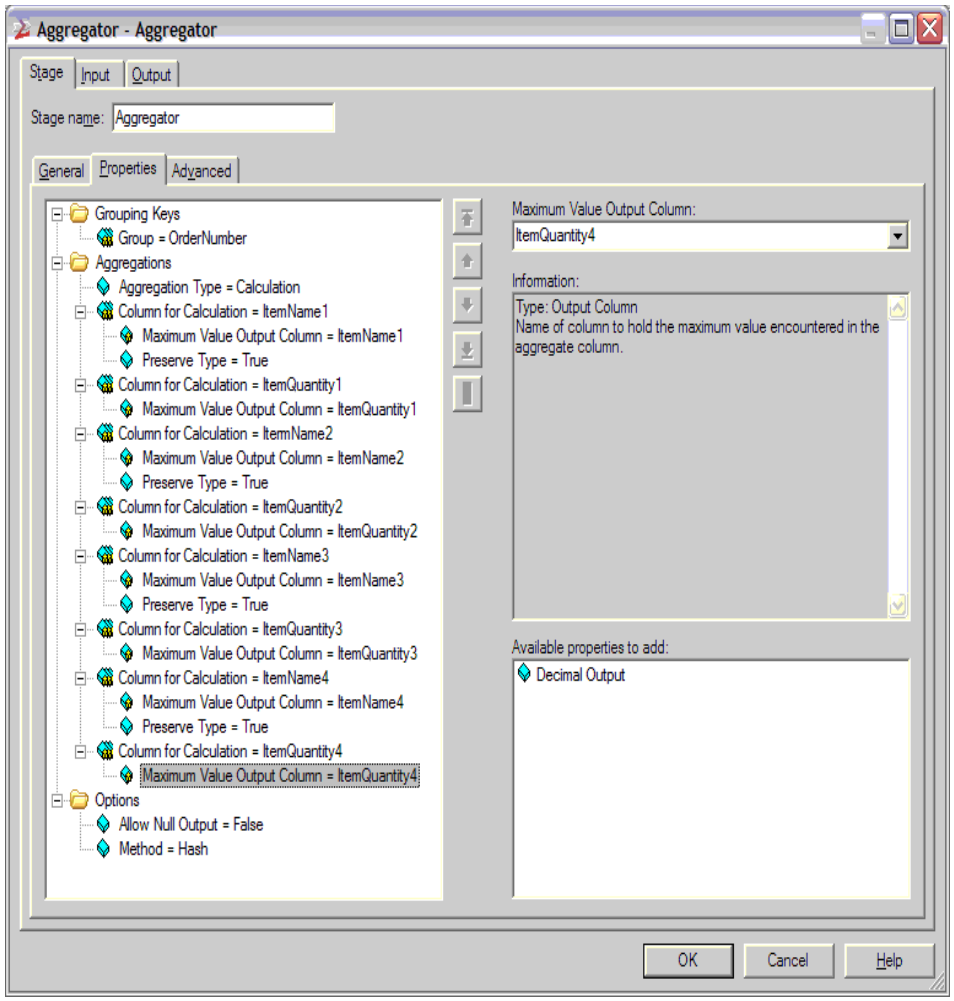


Figure 8-15 Aggregator operator, Step 2 of 4.

e. On the Aggregator operator, Output TAB -> Mapping TAB, drag and drop the Columns table header from the left side of the display to the right. Example as shown in Figure 8-16.

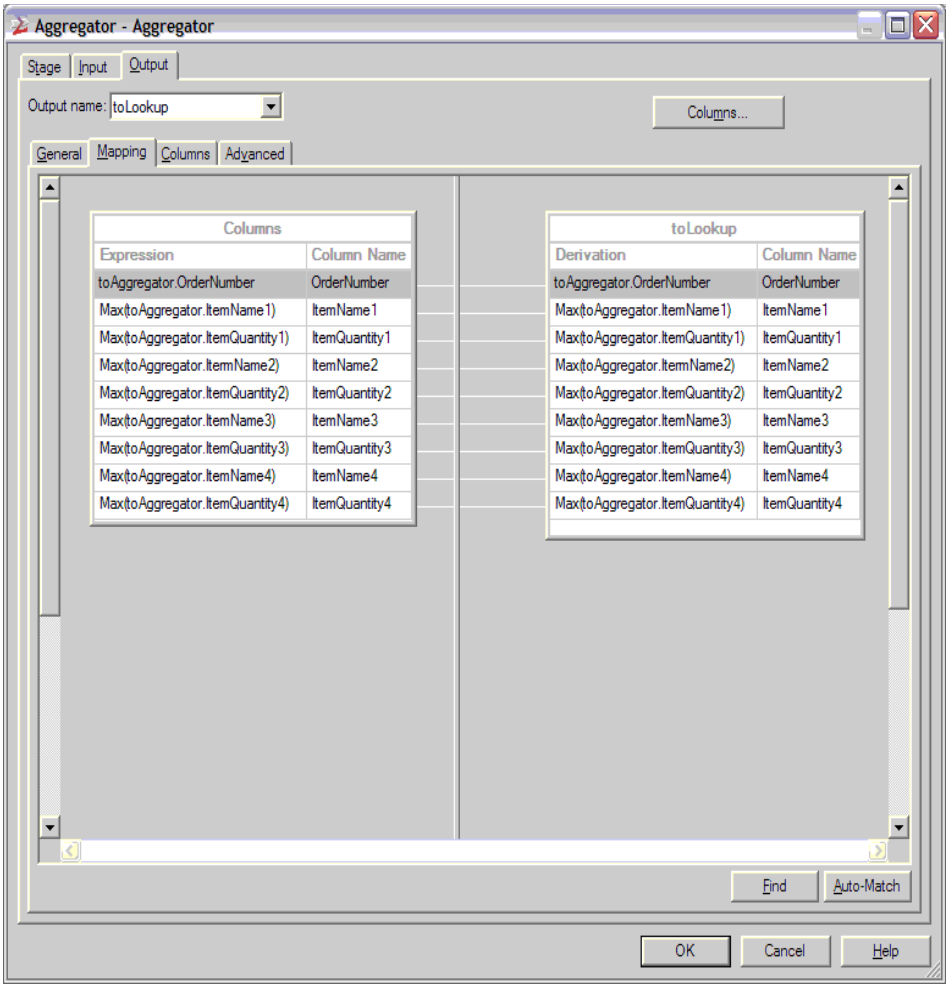


Figure 8-16 Aggregator operator, Step 3 of 4.

f. On the Aggregator operator, Output TAB -> Columns TAB, check to ensure that the output column data types equal what is displayed in Figure 8-17.

Depending on the actual sequence of steps you performed above, you may initially see a "double" data type for the calculated columns when we need VarChar and Integer, as displayed in Figure 8-17.

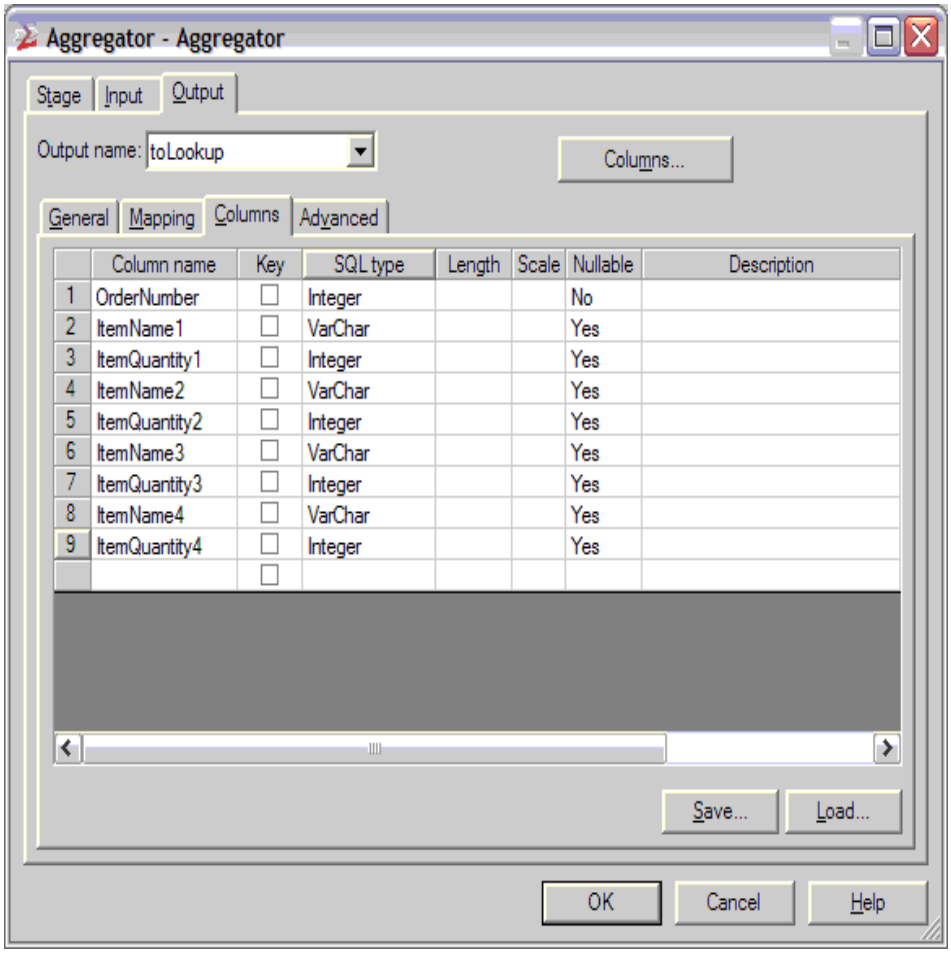


Figure 8-17 Aggregator operator, Step 4 of 4.

g. We are done with the Aggregator operator. Click the OK button to close this screen.

Note: *We are done with the majority of our tasks. If we output to a file now, all we would be missing is the Customer Name from our Order Header. The Lookup operator completes that task, and that is what we perform next.*

Sequential File operator (lookupFile), and Lookup operator

In general, the Lookup operator takes an input stream, and fills in missing values from a second, supporting input stream. The supporting input stream has a Link that appears as a dashed line. It makes sense to complete the supporting input stream first, the complete tasks on the Lookup operator.

17. Double-click the lookupFile (Sequential File) operator and complete the following.
    a. On the Output TAB -> Properties TAB -> Source -> File visual control, browse to the location of the table definition we created in Step-6.c, OutputFile01.txt.
    b. On the Output TAB -> Format TAB, click the Load button and browse to the previously saved, OrderHeader, table definition we saved in Step-6.c.
    c. On the Output TAB -> Columns TAB, click the Load button and browse to the previously saved, OrderHeader, table definition we saved in Step-6.c.
    d. Click the OK button to close the Sequential File operator.

    In effect, we just set the input file name for look up, delimiter properties, and column definitions.
18. Double-click the Lookup operator and complete the following;
    a. Much like other screens, the Lookup operator screen is organized into 4 areas. Figure 8-18 displays the Lookup operator screen, and has already had two tasks performed within it.

       First, drag and drop the table header entitled, "toLookup", from the upper left portion of the screen to the upper right.

       This action copies all of the input columns we receive to the output columns side.
    b. In the bottom right portion of the display, right-click on the column entry entitled, ItemName1, and select -> Insert Row.

       This action will add a new output column after the OrderNumber column, but before the ItemName1 output column.

       At first this new entry will appear back lit in red in the upper right portion of the display, because this column's derivation is not complete.
    c. Change the name of the newly created output column to equal, CustomerName, and change its data type to VarChar.

       This work is done in the bottom right portion of the display.

       A completed example appears in Figure 8-18.

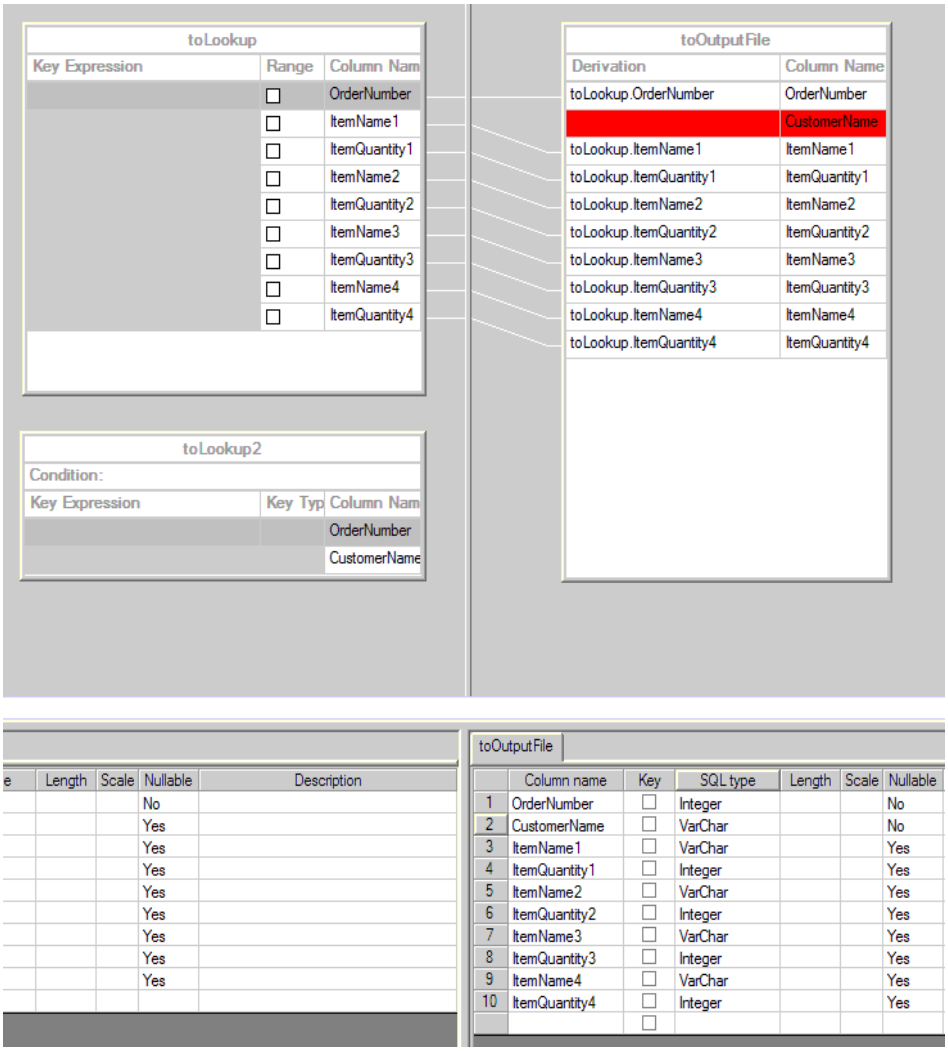


Figure 8-18 Lookup operator, Step 1 of 2.

Two more drag and drops, and we are done with the Lookup operator.

d. In the upper left portion of the display, drag and drop the toLookup -> Column Name entitled OrderNumber, to the toLookup2 -> Key Expression field, on the same row as OrderNumber.

You may receive a new dialog box saying that OrderNumber was not previously defined as a key field. Click the Yes to All button to clear this condition.

e. This second and final drag and drop, copies a column definition from the upper left portion of the display to the upper right.

   In the upper left portion of the display, drag and drop toLookup2 -> CustomerName, to the upper right portion of the display, CustomerName derivation column.

   A completed example appears in Figure 8-19.

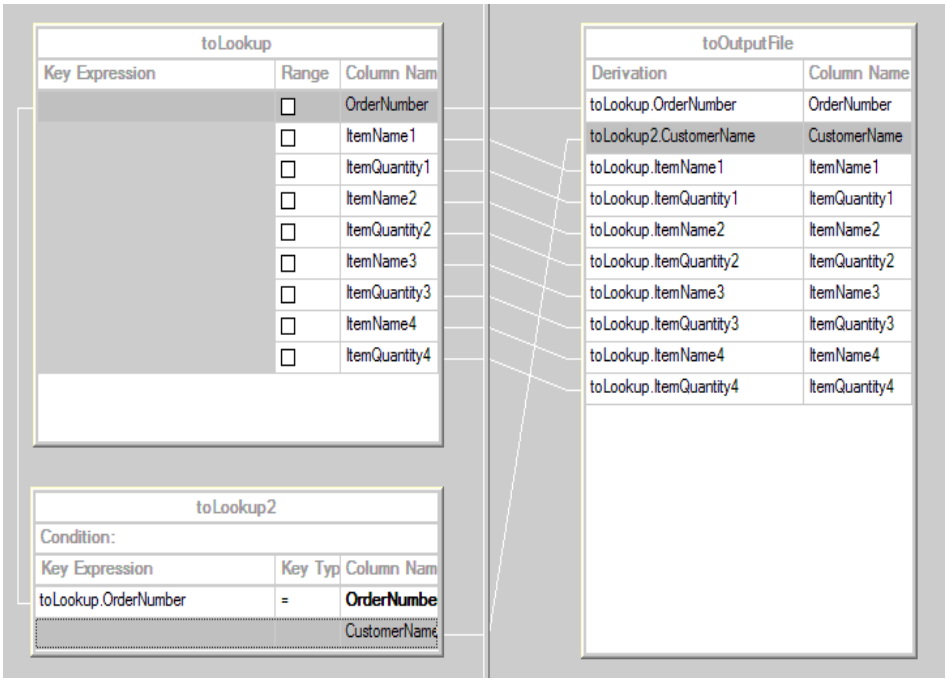


Figure 8-19 Lookup operator, Step 2 of 2.

f. You are done with the Lookup operator. Click the OK button to close this screen.

Sequential File object, OutputFile

19. Double-click the OutputFile (Sequential File) object and complete the following;

a. Set the Input TAB -> Properties TAB -> Target -> File property. We set out file name equal to, OutputFile03.txt, and put it in the same directory as every other file we created in this document.
b. Set the Input TAB -> Format TAB, delimiter and quote properties. These are most easily set by clicking the Load button, and copying these from a previously saved table definition.
c. Click the OK button to save changes and close the Sequential file object.

Save, compile and test

Save, compile and test this Job. If everything is working accurately, the output file from this Job will equal the input file from the very first Job we created in this document. With the two Jobs we created in this document, you have effectively normalized and de-normalized an input data set.

8.4 Summation

In this document, we reviewed or created:

All of the activity performed in this document was done inside the DataStage EE Designer client, a component to IBM Information Server. Starting with an ASCII text file of de-normalized data, we transformed this data into two representative normalized data sets. The terms normalized and de-normalized come from the relational database modelling arena. After creating normalized data, we completed the circuit and (re)created a denormalized data set.

Minus creating our original (test) input data set, 96% or more of the activities we performed were done via graphical control, drag and drop. And the Jobs we created a fully parallel aware, performance ready, etcetera.

Additional resources:

None.

Thank you to the persons who helped this month:

Alan Spayth, Neil Beckwith, and Patrick Owen.

Legal statements:

occurrence in this information with the appropriate symbol (® or ™), indicating trademarks that were owned by IBM at the time this information was published. A complete and current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

Other company, product or service names may be trademarks or service marks of others.

Special attributions:

The listed trademarks of the following companies require marking and attribution:

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Microsoft trademark guidelines

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Intel trademark information

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

Other company, product, or service names may be trademarks or service marks of others.

