Chapter 13. January 2008

Welcome to the January 2008 edition of IBM Information Server Developer's Notebook. This month we answer the question;

> How do I improve the performance of my IBM Information Server, DataStage Enterprise Edition Component, parallel Jobs?
>
> *Excellent question! While there are dozens of sub-topics that could be discussed here, we are going to give focus to use of 'in flight group marking (IFGM)'. IFGM is not an IBM Information Server related acronym or keyword. Instead, IFGM is a (DataStage Job) design pattern. Given an input data stream with sub-groupings of data records (rows), how does one reject all members of a sub-grouping of rows, even if only a single or smaller set of rows have issue? Better yet; How does one do this without landing data to disk?*

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 (IIS) version 8.01, FixPak 1A, using the Microsoft Windows XP/SP2 platform to support IIS client programs, and a RedHat Linux Advanced Server 4 (RHEL 4) FixPak U6 32 bit SMP server (Linux kernel version 2.6.9-67.EL-smp) to support the IIS server components.

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 Anywhere™, WebSphere Information Analyzer™, WebSphere Information Services Director™, WebSphere DataStage™, WebSphere QualityStage™, WebSphere Metadata Server and Metabridges™, WebSphere Metadata Workbench™, WebSphere Federation Server™, Classic Federation™, Event Publisher™, Replication Server™, Rational Data Architect™, DataMirror Transformation Server™, and others.

Obviously, IBM Information Server is a large and capable product, addressing many strategic needs across the enterprise, and supporting different roles and responsibilities.

13.1 Terms and core concepts

As stated above, this month's question related IBM Information Server (IIS) DataStage Enterprise Edition Component (DS/EE) Job tuning and performance, can invoke dozens of sub topics. There are the Resource Estimator and Performance Analysis dialogs; (Menu bar: File -> Estimate Resource, and Menu bar: File -> Performance Analysis). There is discussion of the DS/EE Configuration File resource and planning, ODBC driver use and tuning, including database parallelism and interface, and related. There is also a discussion of gating operators, parallelism, and Job design for maximum throughput, which is part of what we discuss this month.

In flight group marking (IFGM)

The primary concept we are giving focus to this month is; landing data to disk as part of Job processing, or performing the same work entirely in memory. For example;

- Imagine you have retail point of sale data. Each customer passes through the check out lane, and buys 1, 2, or more line items. Each customer, each sales transaction, has one or more data records in its grouping of records, which forms a single unit of work;
    - A single data record in this data stream, is a single item which was purchased. We'll call this a *line item*.
    - A sub grouping of data in this data stream, is the one or more items bought by just one customer. A customer may buy 1, 2, or more line items. This sub grouping of records for just 1 customer is a single sales transaction, a single unit of work. We'll call this a *transaction*.
    - The entire data stream of records represents all data from a given cash register for a period of time. We'll call this the input *data file*.
- What we wish to do in our DS/EE Job, is check each line item for accuracy. For example;
    - Does the given line item have a recognized product identifier? If not, reject the given line item.
    - Does a given line item sold by weight, have a reasonable weight? (Is the weight zero, or negative?) If not, reject the given line item.
    - And so on.
- If any line item within a given transaction is invalid, we wish to reject all line items for that transaction. And, we wish to accomplish this without landing (writing) data to disk, we wish to do this entirely in memory, within one input data stream processing. We call this DS/EE design pattern, *in flight group marking (IFGM)*.

Example: a less performant (bad) means to group mark

Figure 13-1 displays a number of DataStage Enterprise Edition (DS/EE) Jobs, and one Sequence (the DS/EE entity to run and control Jobs), *and serve as our less performant (bad) example;*

– Jobs 1, 2 and 3 basically receive the input data stream, the data file from a given cash register, and then reject individual line item records that are offensive.

– Job 4 collects all of the offensive line item records that have been output, and uses this information to the aggregate all of the transaction identifiers that have 1 or more offensive line items. Job 4 then filters out entire transactions based on these known (offensive) key values.

– The Sequence is used to run all 4 Jobs in order, and also to perform clean up of intermediary results.

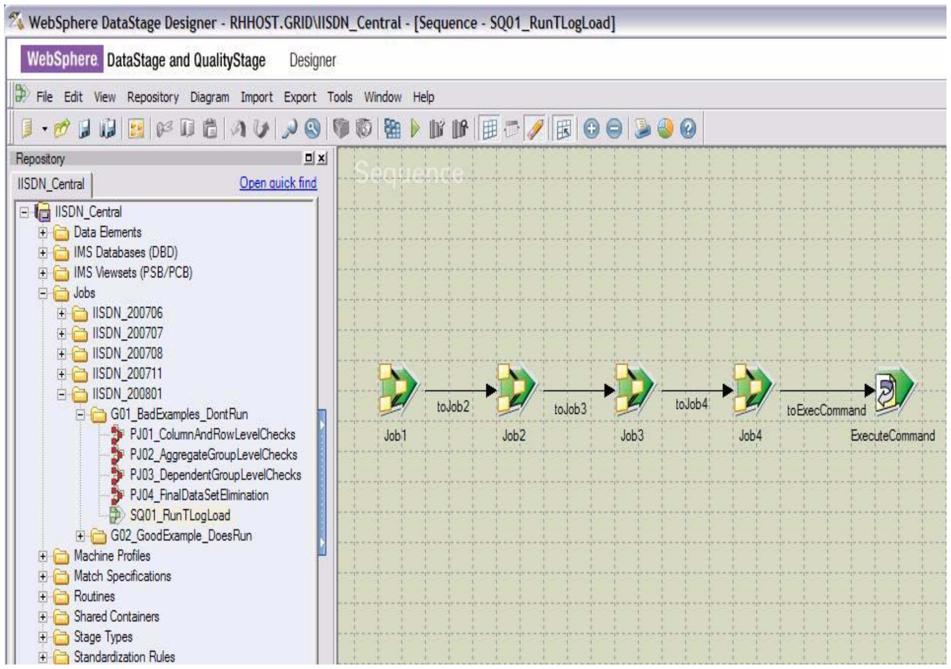


Figure 13-1 Bad example: Collection of Jobs to accomplish group marking.

Figure 13-2 displays a subset of Job 1. Reading from left to right, a DS/EE Lookup operator is used to validate line items as being valid or not valid. Offensive line items are sent to the first of several reject files for later processing. The second DS/EE Lookup operator performs a similar function, and is displayed to reinforce the fact that we have multiple output/reject files.

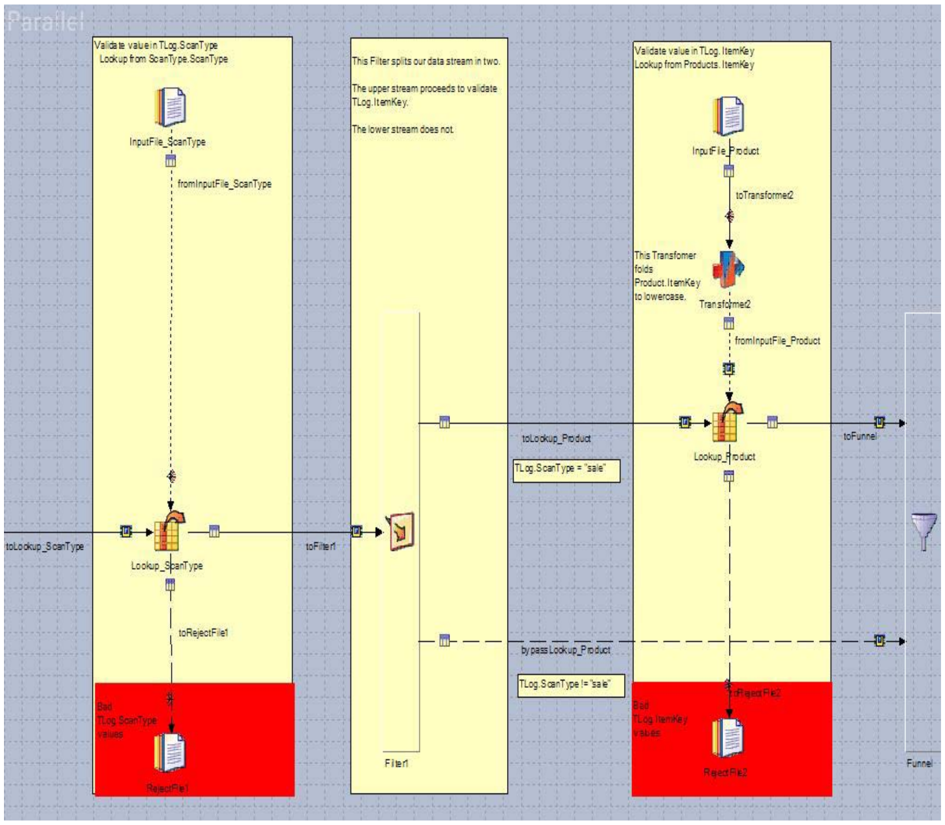


Figure 13-2 Bad example: subset of Job 1, sending offensive line items to reject files.

Figure 13-3 displays the second DS/EE Job in this group; between Figure 13-2 and Figure 13-3, and the presence of Filter (split a data stream) and Copy (duplicate an input stream) DS/EE operators, we see the business need to reject a line item outright (immediately reject it, remove it from the data stream), or record that it was rejected, but allow it to continue in the input stream due to a later dependency.

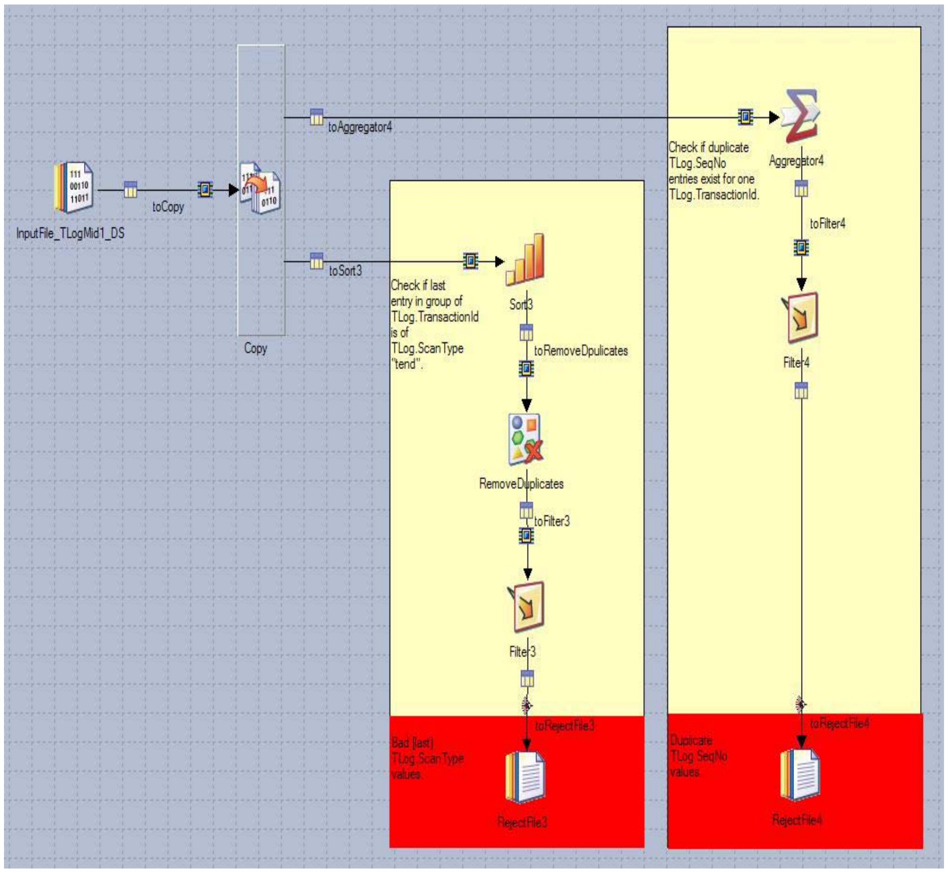

toAggregator4
Check if duplicate TLog.SeqNo entries exist for one TLog.TransactionId.
Aggregator4
toFilter4
Filter4
InputFile_TLogMid1_DS
toCopy
Copy
toSort3
Check if last entry in group of TLog.TransactionId is of TLog.ScanType "tend".
Sort3
toRemoveDpulicates
RemoveDuplicates
toFilter3
Filter3
toRejectFile3
Bad [last] TLog.ScanType values.
RejectFile3
toRejectFile4
Duplicate TLog.SeqNo values.
RejectFile4

Figure 13-3   Bad example: Job 2, sending offensive line items to reject files.

And then finally, Figure 13-4 displays how to gather all of the offensive (rejected) line item records together, aggregate the offensive transaction identifiers, and use this set of values to pull entire transactions out of the original input data file.

- From top to bottom, the (5) rejects files are sorted and aggregated, and used as a join key (Lookup operator) back to the original data file.
- From left to right, the original data file (actually cleaned a little bit in Job 1), is the primary data stream to the Lookup operator.
- Good (whole) transactions go to the green file, rejected (whole) transactions go red.

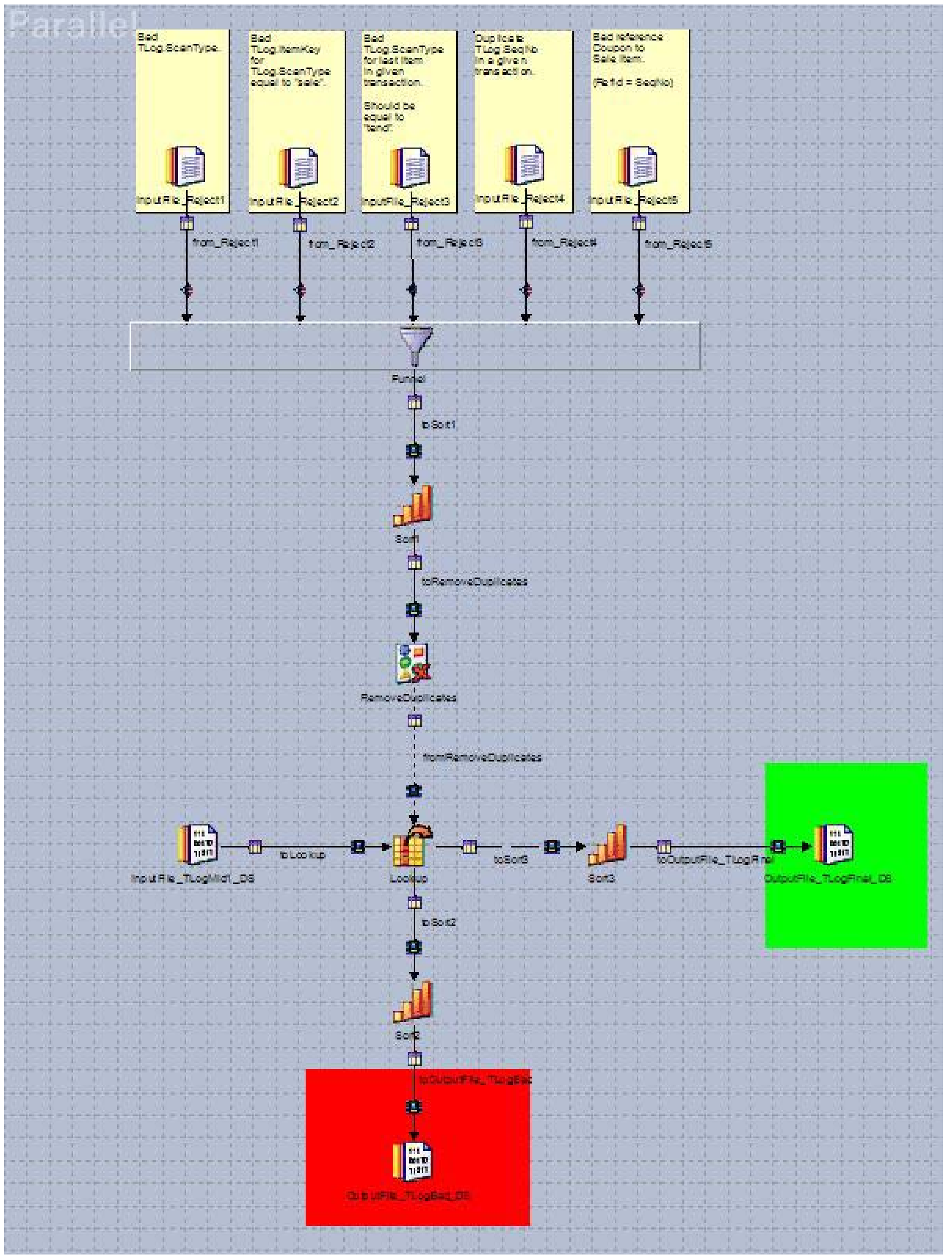


Figure 13-4   Bad example: pulling all of the rejected line items together.

How to do this faster, and with less resource

In the prior (bad) example, one sees that we land offensive line items to reject file (to disk), numerous times. We don't need these data files, other than to serve as temporary storage, a means to record offensive line items. Also, we run several Jobs (5 in the example as presented), when we really only needed 1 Job.

In the next section of this document, we demonstrate how to perform less disk I/O (not using reject files as temporary storage), consume less CPU (perform less processing), and likely consume less memory (less process memory being started and stopped, less unnecessary (re)loading of data sets).

13.2 Example, in flight group marking (IFGM)

The single IBM Information Server (IIS) DataStage Enterprise Edition Component (DS/EE) sample Job that we are about to create is visually very wide, and appears first in Figure 13-5, the first in a three diagram series. We create a DS/EE Job that validates individual line items of a retail point of sale transaction, and then rejects entire transactions that have a single or set of offending member rows. And we do all of this in flight, without landing data to disk unnecessarily.

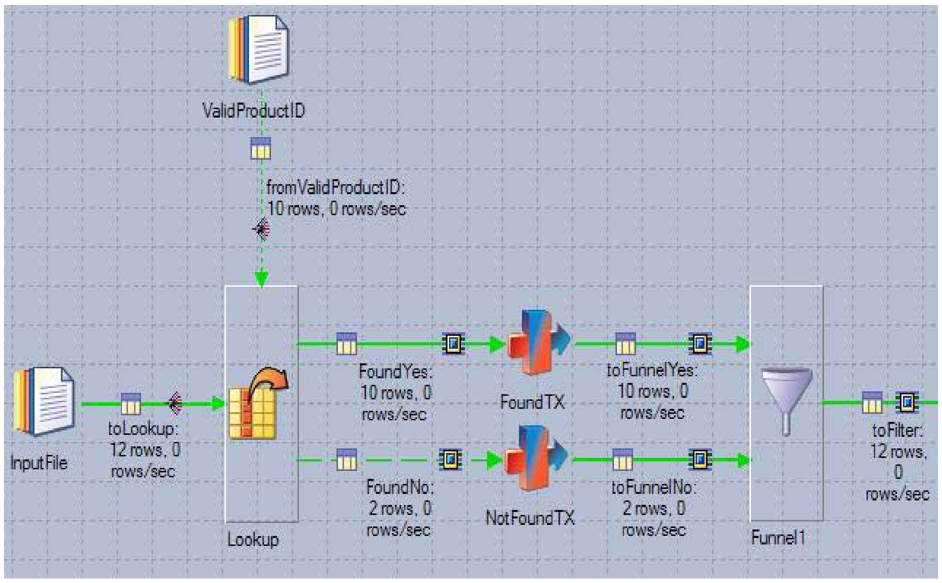

ValidProductID
fromValidProductID: 10 rows, 0 rows/sec
InputFile
toLookup: 12 rows, 0 rows/sec
Lookup
FoundYes: 10 rows, 0 rows/sec
FoundNo: 2 rows, 0 rows/sec
FoundTX
NotFoundTX
toFunnelYes: 10 rows, 0 rows/sec
toFunnelNo: 2 rows, 0 rows/sec
Funnel1
toFilter: 12 rows, 0 rows/sec

Figure 13-5 Good example, figure 1 of 3, read left to right.

Figure 13-6 continues the display of this sample Job, with the Funnel1 operator being repeated in its display.

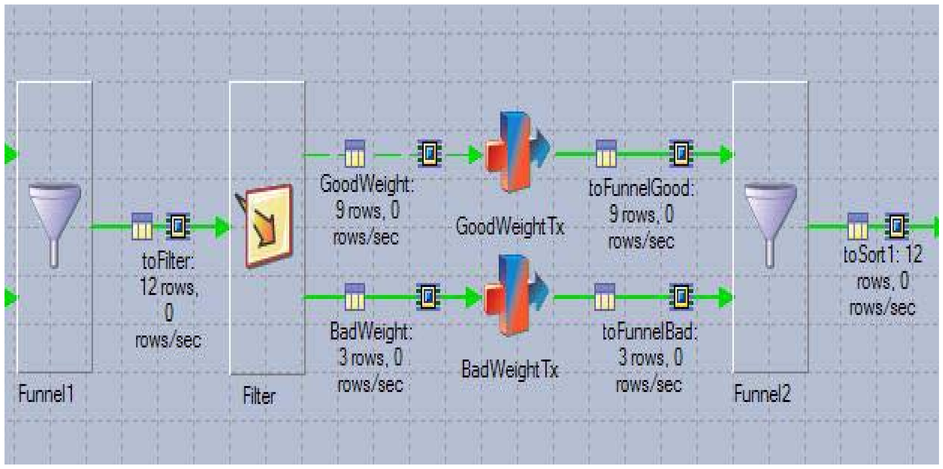

GoodWeight: 9 rows, 0 rows/sec
GoodWeightTx
toFunnelGood: 9 rows, 0 rows/sec
toFilter: 12 rows, 0 rows/sec
toSort1: 12 rows, 0 rows/sec
BadWeight: 3 rows, 0 rows/sec
BadWeightTx
toFunnelBad: 3 rows, 0 rows/sec
Funnel1
Filter
Funnel2

Figure 13-6   Good example, figure 2 of 3, read left to right.

And Figure 13-7 completes the display of this sample Job, with the Funnel 2 operator being repeated in its display.

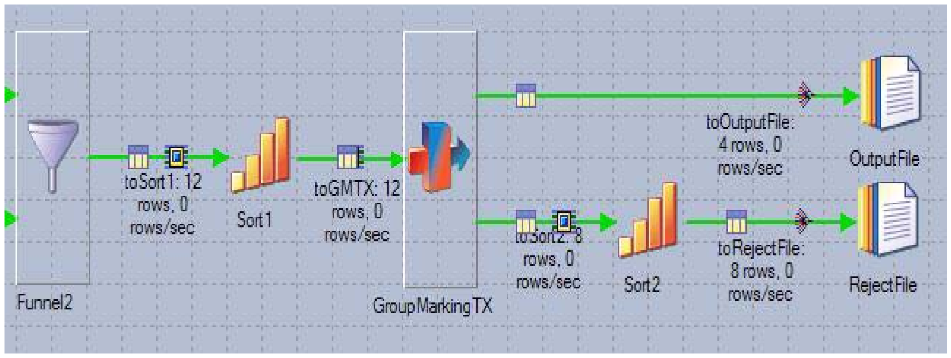

toOutputFile: 4 rows, 0 rows/sec
OutputFile
toSort1: 12 rows, 0 rows/sec
Sort1
toGMTX: 12 rows, 0 rows/sec
toSort2: 8 rows, 0 rows/sec
Sort2
toRejectFile: 8 rows, 0 rows/sec
RejectFile
Funnel2
GroupMarkingTX

Figure 13-7   Good example, figure 3 of 3, read left to right.

Business walk through of the sample Job above

Listed below, is an Operator by Operator description of the function of the Job displayed in Figure 13-5 through Figure 13-7;

– InputFile, is a DS/EE Sequential File Operator. We read a text file to start our input data stream.

You can rename Operators, and the Links that connect Operators. We named this Operator, InputFile.This data file has 5 columns, detailed below;

- TransactionID, the primary key identifier for a single sales event.

  One Transaction, and hence one Transaction ID, may have one or more line items, one or more things that were sold at one time.
- SequenceNumber, used to preserve line item ordering within a transaction.
- ProductID, is the name of the line item that was sold.

  This value is used later in a look up to a list of valid Product IDs, and thus validate the value in this column.
- IfIsWeighed, is used as a flag to indicate if the next column, the Weight, is to be validated.
- Weight, the weight of the line item. This value must pass a conditional test, later in the Job.

– Lookup, is a DS/EE Lookup Operator. The common use of this Operator is to determine if a standard abbreviation or code is valid. Further. this Operator, per our example, has 2 inputs and 2 outputs;

- toLookup is the *input stream* to this Operator, and supplies the data to be processed.
- fromValidProductID. is the *reference stream* to this Operator. Also an input stream, a reference stream indicates that this is the master side of the join pair, the value against which our input stream is validated.
- FoundYes, is the *output stream* of validated records. In our example, these are records with a valid ProductID.
- FoundNo, is the *reject stream*, records which where found not to have a matching value from the reference stream. In our example, these are records without a valid ProductID.

– ValidProductId, is a DS/EE Sequential file Operator, just like InputFile.

– FoundTx, and NotFoundTX, are DS/EE (Parallel) Transformer Operators.

- Transformers are the work horse of DS/EE, and can do almost anything. Often folks will use a Transformer in place of an existing (other) DS/EE operator just because they can.
- Here we use the Transformer to add a new column to our data stream. We call this new column, IfIsBadRow, and use it as a marker to indicate if the given record has failed a validation test.

– Funnel1, is a DS/EE Funnel Operator, used to merge two or more input data streams.

- Filter, is a DS/EE Filter Operator, and is used to direct the flow of an input data stream.
  - In this Filter, we first check if a given line item is supposed to be weighed (IfIsWeighed), then we check to see if its weight (Weight) is greater than zero.
  - Good records follow the *output stream* (GoodWeight), and bad records follow the *reject stream* (BadWeight).
- GoodWeightTX, and BadWeightTX are DS/EE (Parallel) Transformer Operators, just like FoundTX and NotFoundTX above. In our example, these Operators record how a given record performed its validation test; pass or fail.

Note: A given data record might fail more than one validation test. *Per our design*, we give good (non-failing) data records a value of zero.

Records that failed the first validation test (Lookup) are given a value of 1. In this validation test (Filter), we add 2 to the previous value.

So if a given record has a value of zero, it has passed all tests. A 1 means it failed the first test. A 2 means it failed only the second test. A value of 3 (1 + 2), means it failed both tests.

If we had another test, we'd add 4, a test after that, 8, and so on.

This is one means to record the history of all tests. another means would be bit mapping. E.g., "00101", for failed third and fifth tests, etcetera.

- Funnel2, is a DS/EE Funnel Operator, just like Funnel1.
- Sort1, is a DS/EE Sort Operator.

  Here we sort the input data stream by TransactionID, and then by IfIsBadRow. In this manner, the first bad record will appear first in the sub group of records. By having any bad record first in a sub group, we can set a flag to reject all rows in the sub group.
- GroupMarkingTX, is a DS/EE (Parallel) Transformer Operator.

  This is a really fun Transformer, worthy of study. The previous Transformers basically set a single column value. This Transformer actually has some meat to it. This Transformer demonstrates the following;
  - How to set and then use Transformer Variables.
  - How to use Transformer Variable to detect an after sub group condition.

- How to program multiple output Links from a Transformer.

Note: It is this Transformer, and the setup prior to this Operator, that allows us to do in flight group marking, the topic of this entire document.

Without this Operator, and how we use it, we'd only have the bad example offered previously, that lands data to disk.

– Sort2, is another DS/EE Sort Operator.

Note: Previously our example data was sorted by TransactionID, and then by IfIsBadRow. Here we sort by TransactionID, and then by SequenceNumber.

The Sort Operator can be configured to know that data was previously pre-sorted, even partially pre-sorted.

Sorting data here was done mostly for style; it wasn't required for our use case or demonstration.

The primary reason to include this Operator as configured, was to demonstrate use of pre-sorts; another sub topic related to improving DS/EE Job performance.

– And then lastly, OutputFile and RejectFile are DS/EE Sequential File Operators, just like Input File.

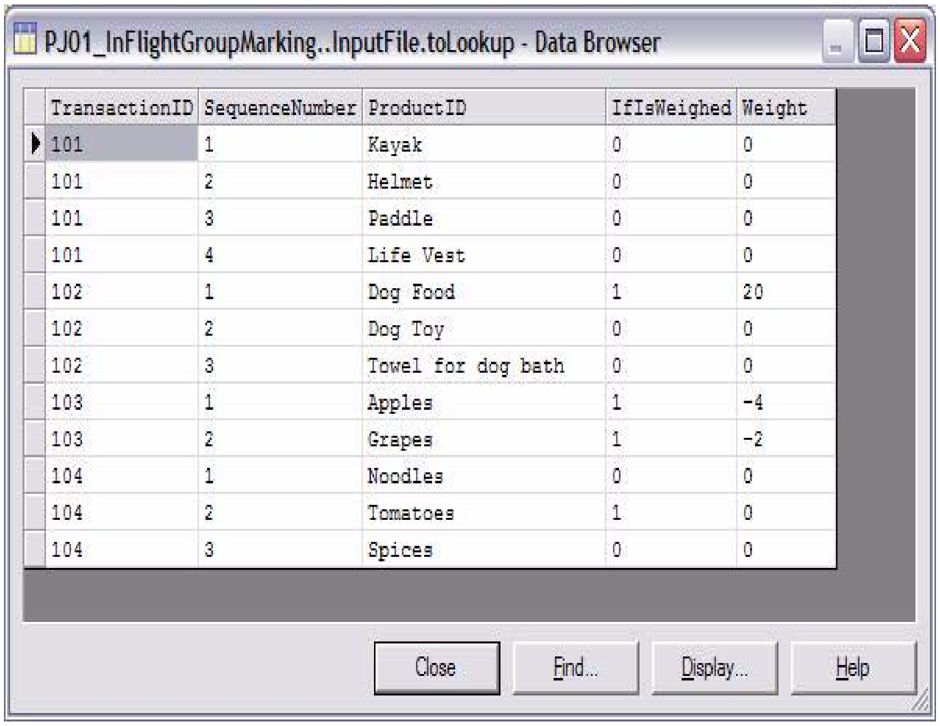
  OutputFile receives the data from output stream, whereas RejectFile receives the data from the bad data side of GroupMarkingTX.

How to create the DS/EE Job displayed above

Below is a step by step set of instructions that allow you to create the IBM Information Server (IIS) DataStage Enterprise Edition Component (DS/EE) Job displayed in Figure 13-5 through Figure 13-7.

1. Using your favorite program editor, create two ASCII text files.
   a. Name the first file InputFile.txt, and create its contents similar to what is shown in Figure 13-8.
      - Do not include the column headers; TransactionID, SequenceNumber, etcetera.
      - Separate each column with a vertical bar delimiter, and zero white space. For example, line 5 would read;

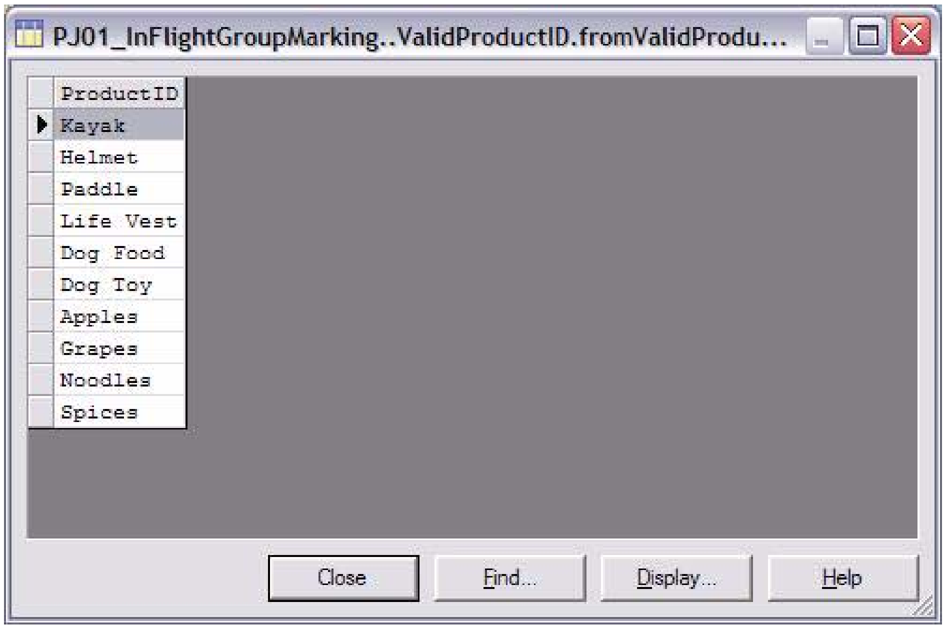
```
101|1|Dog Food|1|20|
```


PJ01_InFlightGroupMarking..InputFile.toLookup - Data Browser
Close
Find...
Display...
Help

TransactionID	SequenceNumber	ProductID	IfIsWeighed	Weight
101	1	Kayak	0	0
101	2	Helmet	0	0
101	3	Paddle	0	0
101	4	Life Vest	0	0
102	1	Dog Food	1	20
102	2	Dog Toy	0	0
102	3	Towel for dog bath	0	0
103	1	Apples	1	-4
103	2	Grapes	1	-2
104	1	Noodles	0	0
104	2	Tomatoes	1	0
104	3	Spices	0	0

Figure 13-8 Example InputFile.txt, contents. (Do not include column headers.)

b. Name the second file ValidProductID.txt, and create its contents similar to what is shown in Figure 13-9.



ProductID
Kayak
Helmet
Paddle
Life Vest
Dog Food
Dog Toy
Apples
Grapes
Noodles
Spices

Figure 13-9 Example ValidProductID.txt. (Do not include column headers.)

Specific conditions exist in InputFile.txt and ValidProductID.txt, so that we may generate offensive line item records later in our DS/EE Job. These conditions include;

- "Towel for dog bath" and "Tomatoes", are both ProductIDs in InputFile.txt, that are not found to exist inside ValidProductID.txt.
- Apples, Grapes and Tomatoes, are all line items that are marked as items which should be weighed, but they have invalid weights.

These bad data conditions will cause TransactionIDs 102, 103, and 104 to fall out in their entirety. Only TransactionID 101 contains a collection of valid line items that are entirely valid.

2. Log in to the IBM Information Server (IIS) DataStage Enterprise Edition (DS/EE) Designer Program.

   a. Close any optional dialog boxes that may present themselves.

   b. From the Menu Bar, select;

      File -> New -> Jobs -> Parallel Job, and then Click, OK.

   c. From the Menu Bar, select;

      File -> Save As.

      Complete the steps to save your Job with a given name and location.

Note: Remain inside the DataStage Designer program; all work is done inside this graphical program.

3. Inside the Palette view of the DS/EE Designer program, Drag and Drop several Operators onto the Parallel Canvas.
   a. Inside the File drawer of the Palette view, drag and drop 4 (count) Sequential File Operators onto the Parallel Canvas.
   b. Inside the Processing drawer of the Palette view, Drag and Drop;
      i. 1 (count) Lookup Operator.
      ii. 5 (count) Transformer Operators.
      iii. 2 (count) Funnel Operators.
      iv. 1 (count) Filter Operator.
      v. 2 (count) Sort Operators.
   c. Using a Click and Hold, move the Operators into position as shown in Figure 13-5 through Figure 13-7.
   d. Using a Right-Click on each Operator, Rename each Operator as shown in Figure 13-5 through Figure 13-7.
4. Linking the Operators on the Parallel Canvas.

   Links are the arrows that connect the Operators on the Parallel Canvas display, and determine properties such as the type of input or output that is desired. Four types of Links are displayed in Figure 13-5 through Figure 13-7. These include;

   - *Input Links*, standard input to an Operator, normally arriving from the standard output of the prior Operator.
   - *Reference Links*, a specific type of Input Link, meant to imply the parent or authoritative side of a join pair. Used generally to validate codes and abbreviations via use of a Lookup, Join, or Merge Operator.
   - *Output Links*, standard output from an Operator, generally leads to the input of the next Operator.
   - *Reject Links*, also an output Link, generally implies bad or erroneous data.

   What type of Link you create can be influenced by the order in which you attach Links to an Operator. You may also Right-Click a Link, to change its Link type. For now, we are going to influence our Link types by the order in which we create them.

Create the following Links, in order. A Link is created by a [Right-Click and Hold, Drag, and then Release], from one Operator to another.

Rename each Link after it is created. This is done via a Right-click on the Link itself, and then selecting, Rename.

a. InputFile to Lookup. Rename to toLookup.

   (This Link should appear as a solid line, indicating it is an Input Link.)

b. ValidProductID to Lookup. Rename to fromValidProductID

   (This Link should appear with a dashed line, indicating it is a Reference Link.)

c. Lookup to FoundTX. Rename to FoundYes.

d. Lookup to NotFoundTX. Rename to FoundNo.

   (This Link should appear with an elongated dashed line, indicating it is a Reject Link.)

e. FoundTX to Funnel1. Rename to toFunnelYes.

f. NotFoundTx to Funnel1. Rename to toFunnelNo.

g. Funnel1 to Filter. Rename to toFilter.

Note: Thus far the steps have been left to right, top to bottom. This next step does not follow that order.

h. Filter to BadWeightTX, Rename to BadWeight.

i. Filter to GoodWeightTX. Rename to GoodWeight.

   Right-Click this Link, Select, convert to Reject.

   (This Link should appear with an elongated dashed line, indicating it is a Reject Link.)

j. GoodWeightTX to Funnel2. Rename to toFunnelGood.

k. BadWeightTx to Funnel2, Rename to toFunnelBad.

l. Funnel2 to Sort1. Rename to toSort1.

m. Sort1 to GroupMarkingTX. Rename to toGMTX.

n. GroupMarkingTX to OutputFile. Rename to toOutputFile.

o. GroupMarkingTX to Ssort2. Rename to toSort2.

p. Sort2 to RejectFile. Rename to toRejectFile.

5. Configuring each Operator on the Parallel Canvas.

   At this point, our basic Job is displayed, including Operator types, and Linking. All that remains now, is to configure the Properties for each Operator, Compile and Test. To access the Properties dialog for each Operator, Double-Click each Operator in turn, as detailed below;

   a. InputFile (Sequential File Operator).

      i. On the Output TAB -> Properties TAB, set the Source -> File (name) value.

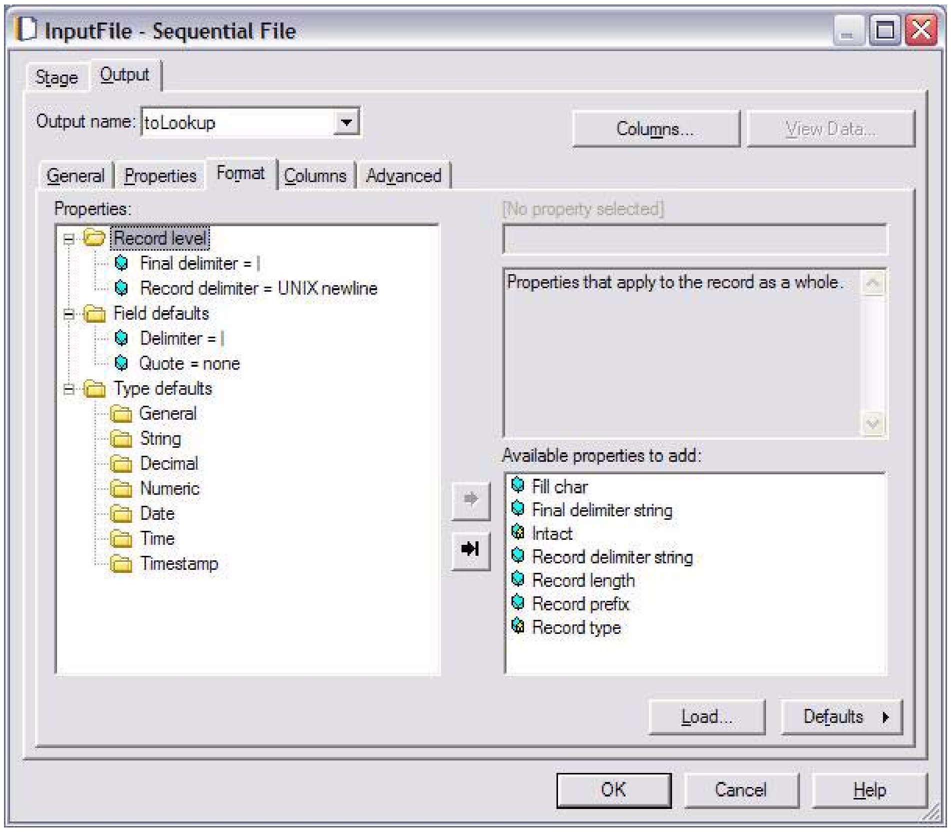
      When this control is highlighted (current), a text entry field will be available on the right. A Right-Arrow image button, to the right of this control, allows for a File Browse function.

      Set this property to the absolute pathname location of the input file you created in Step-1a, above.

      ii. On the Output TAB -> Format TAB, there are 4 changes we need to make. See also Figure 13-10.

      - Set Record level -> Final delimiter, to equal a vertical bar, | .
      - You will first have to add the Property of Record level -> Record delimiter, by highlighting the "Record level" group entry, and then picking the Record Delimiter from the, "Available properties to add" control.

        Then set Record level -> Record Delimiter, to equal a vertical bar, | .
      - Set Field defaults -> Delimiter, to equal a vertical bar, | .
      - Set Field defaults -> Quote, to none.



Output name: toLookup

Properties:

- Record level
  - Final delimiter = |
  - Record delimiter = UNIX newline
- Field defaults
  - Delimiter = |
  - Quote = none
- Type defaults
  - General
  - String
  - Decimal
  - Numeric
  - Date
  - Time
  - Timestamp

[No property selected]

Properties that apply to the record as a whole.

Available properties to add:

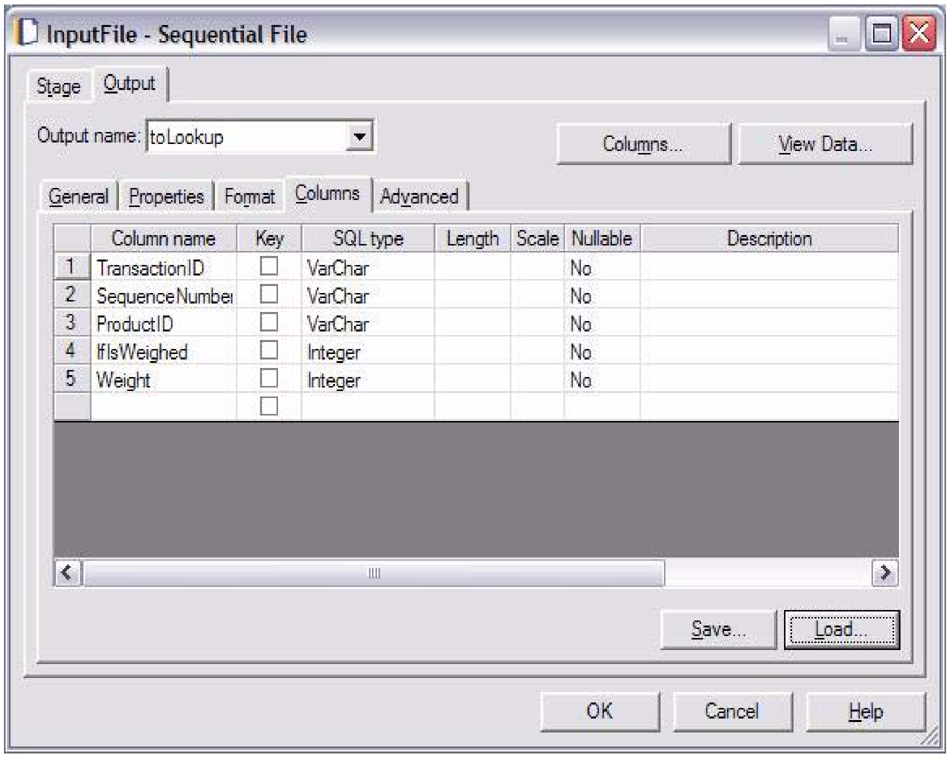
- Fill char
- Final delimiter string
- Intact
- Record delimiter string
- Record length
- Record prefix
- Record type

Figure 13-10 Sequential File Operator, Output TAB -> Format TAB.

iii. On the Output TAB -> Columns TAB, manage your display to equal what is shown in Figure 13-11.

To test the accuracy of your work, Click the View Data button, in the upper right portion of this dialog box. If you see the data from InputFile.txt, you are good to go.

iv. Click OK when you are done.



	Column name	Key	SQL type	Length	Scale	Nullable	Description
1	TransactionID	☐	VarChar			No	
2	SequenceNumbei	☐	VarChar			No	
3	ProductID	☐	VarChar			No	
4	IfIsWeighed	☐	Integer			No	
5	Weight	☐	Integer			No	
		☐					

Figure 13-11 Sequential File operator, Output TAB -> Columns TAB.

b. ValidProductID (Sequential File Operator).

As a Sequential File Operator, setting Properties for this control to our DS/EE Job is very similar to InputFile above, also a Sequential File Operator.

i. Similar to Step-5.a.i above, set the File (name) Property to the absolute pathname of the file you created in Step-1.b above.

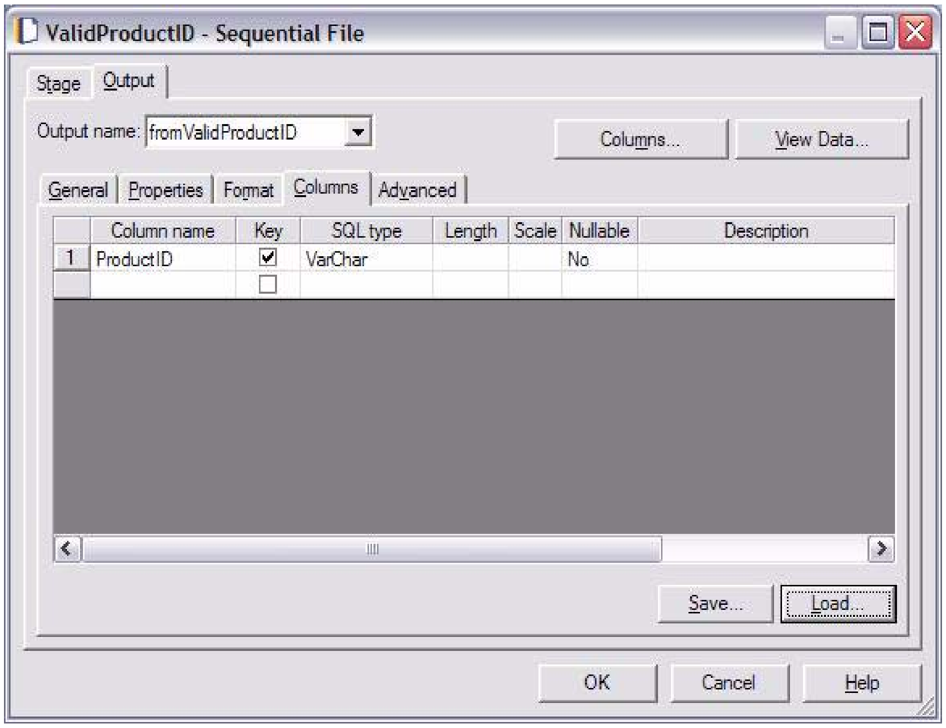
ii. Similar to Step-5.a.ii above, set the 4 Format Properties.

iii. On the Output TAB -> Columns TAB, manage your display to equal what is shown in Figure 13-12.

Notice that the Key (column) control is checked for ProductID.

To test the accuracy of your work, Click the View Data button, in the upper right portion of this dialog box. If you see the data from ValidProductID.txt, you are good to go.

iv. Click OK when you are done.



Output name: fromValidProductID

	Column name	Key	SQL type	Length	Scale	Nullable	Description
1	ProductID	☑	VarChar			No	
		☐					

Figure 13-12   Sequential File operator, Output TAB -> Columns TAB.

c. Lookup (Lookup Operator).

   i. Specifying the output columns.

      Figure 13-13 displays the Lookup Operator Properties dialog as it first appears. A common use of the Lookup Operator is to retrieve the State or Province Name from a control table, when one only has the State/Province abbreviation; the Lookup Operator performs something similar to a SQL Join to retrieve that information.

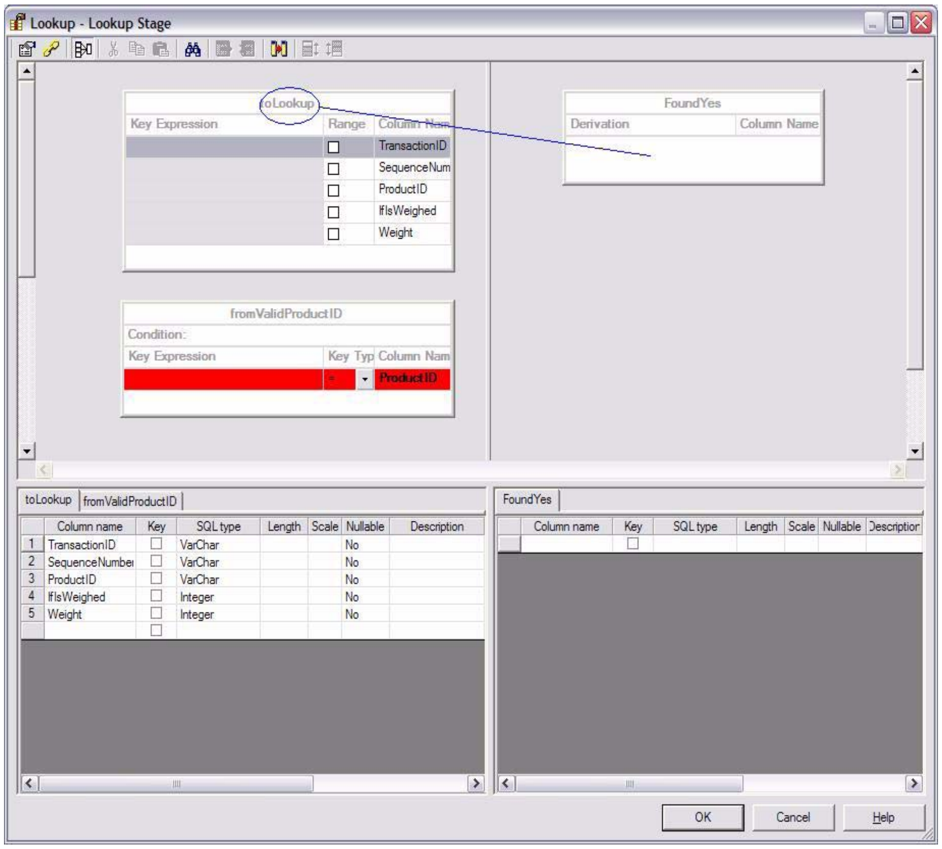
      In our use case, we are performing this Lookup, this Join, only to see if the (ProductID) we have in hand matches an entry in the control table; we are using this Lookup for data validation purposes, we aren't picking up any new columns.

Note: The area in red in Figure 13-13 displays a *required* setting for this Operator; one which must be set before the Operator has any chance to function.

The first task is to specify what this Operator should output to the next Operator in the stream. Since we aren't picking up new columns from the ValidProductID table, we need only output columns from the InputFile side. In any case, this is done via Drag and Drop.

As shown in blue in Figure 13-13, Click and Hold the word "toLookup" (the name of our Input Link), Drag, and Drop it in the whitespace towards the right.

This action will produce the display as shown in Figure 13-14.



toLookup

Key Expression	Range	Column Nam
	☐	TransactionID
	☐	SequenceNum
	☐	ProductID
	☐	IfIsWeighed
	☐	Weight

FoundYes

Derivation	Column Name

fromValidProductID

Condition:

Key Expression	Key Typ	Column Nam
	=	ProductID

toLookup | fromValidProductID

	Column name	Key	SQL type	Length	Scale	Nullable	Description
1	TransactionID	☐	VarChar			No	
2	SequenceNumbe	☐	VarChar			No	
3	ProductID	☐	VarChar			No	
4	IfIsWeighed	☐	Integer			No	
5	Weight	☐	Integer			No	
		☐					

FoundYes

	Column name	Key	SQL type	Length	Scale	Nullable	Descriptior
		☐					

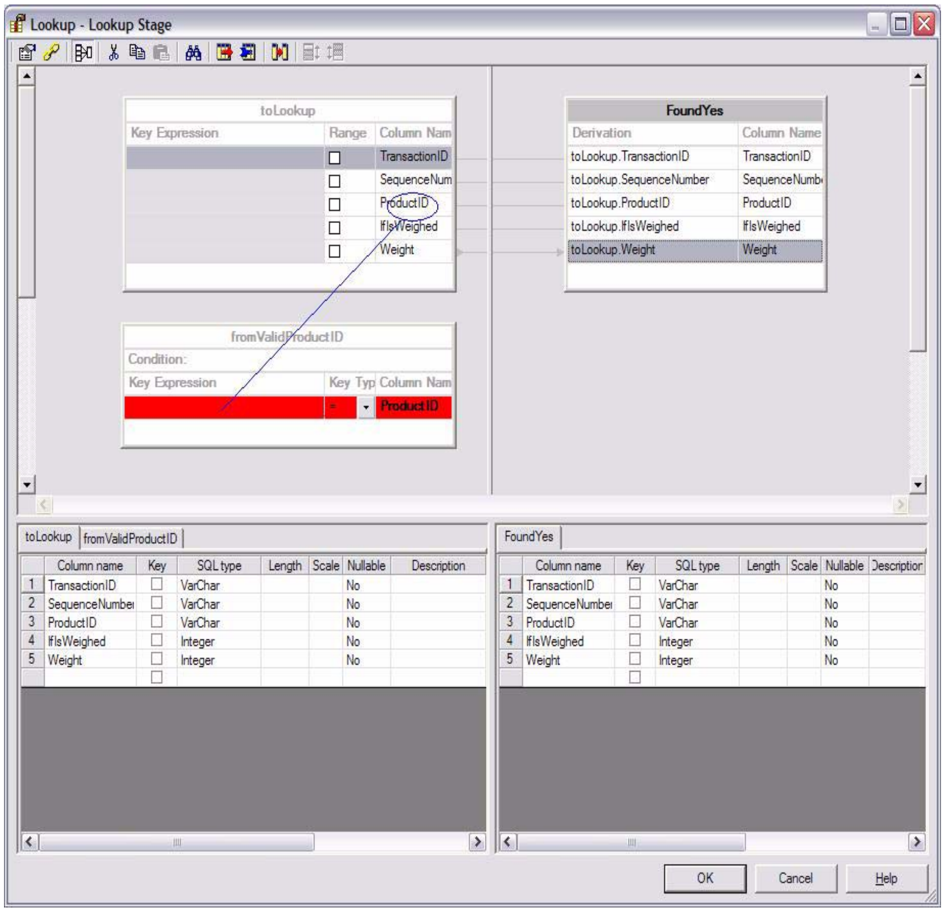
Figure 13-13 Lookup Operator upon first entering.

ii. Specifying the Join columns.

Figure 13-14 displays how to specify the Join columns between our Input Stream (toLookup) and our Reference Stream (fromValidProductID).

As shown in blue in Figure 13-14, Click and Hold the word “ProductID”, Drag, and Release in the white space under “Key Expression”.

This action will satisfy the required Property, which specifies the Join condition between these two input data streams. It also makes the red go away.



toLookup

Key Expression	Range	Column Nam
	☐	TransactionID
	☐	SequenceNum
	☐	ProductID
	☐	IfIsWeighed
	☐	Weight

FoundYes

Derivation	Column Name
toLookup.TransactionID	TransactionID
toLookup.SequenceNumber	SequenceNumb
toLookup.ProductID	ProductID
toLookup.IfIsWeighed	IfIsWeighed
toLookup.Weight	Weight

fromValidProductID

Condition:

Key Expression	Key Typ	Column Nam
	=	ProductID

toLookup | fromValidProductID

	Column name	Key	SQL type	Length	Scale	Nullable	Description
1	TransactionID	☐	VarChar			No	
2	SequenceNumber	☐	VarChar			No	
3	ProductID	☐	VarChar			No	
4	IfIsWeighed	☐	Integer			No	
5	Weight	☐	Integer			No	
		☐					

FoundYes

	Column name	Key	SQL type	Length	Scale	Nullable	Description
1	TransactionID	☐	VarChar			No	
2	SequenceNumber	☐	VarChar			No	
3	ProductID	☐	VarChar			No	
4	IfIsWeighed	☐	Integer			No	
5	Weight	☐	Integer			No	
		☐					

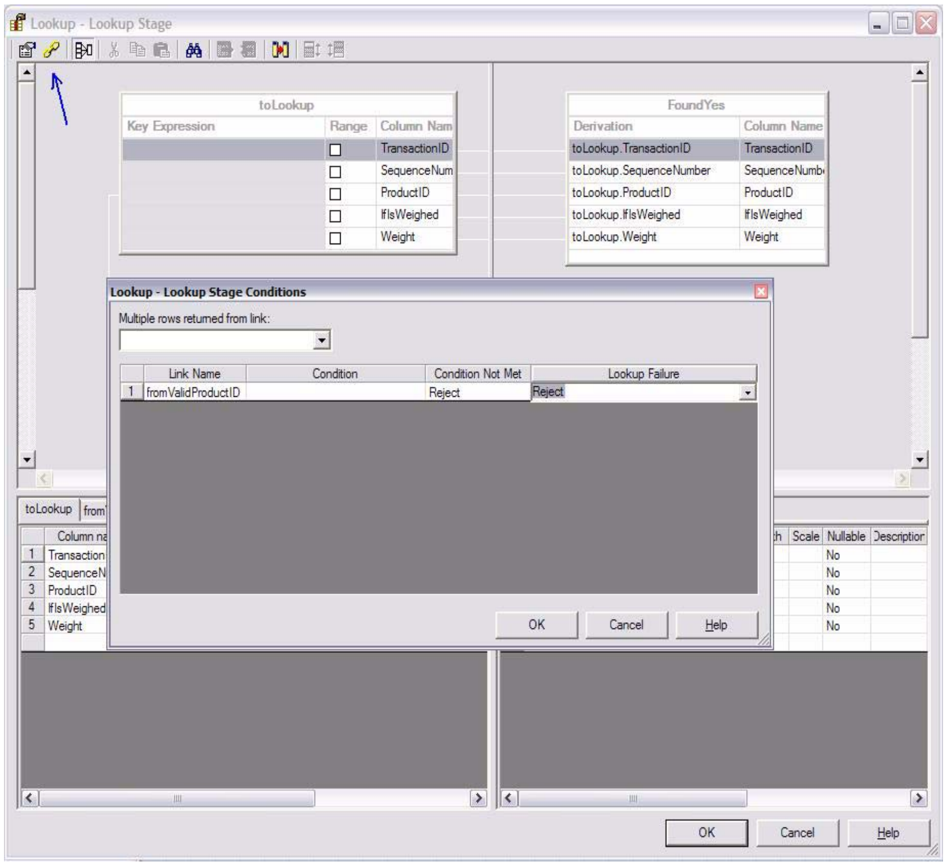
Figure 13-14   Lookup Operator, specifying the Join columns.

iii. Specifying Reject Link behavior.

In many use cases for a Lookup Operator, you would be done at this point. Because we want a Reject File capability, we have to specify that behavior.

On the Tool Bar to this dialog, click the Link icon. Then set the 2 Drop Down List Boxes to Reject. Example as shown in Figure 13-15.

(The Link Tool Bar icon is in the upper left, and highlighted per our image with a blue arrow.)


Lookup - Lookup Stage
Lookup - Lookup Stage Conditions
Multiple rows returned from link:
OK
Cancel
Help

toLookup

Key Expression	Range	Column Nam
	☐	TransactionID
	☐	SequenceNum
	☐	ProductID
	☐	IfIsWeighed
	☐	Weight

FoundYes

Derivation	Column Name
toLookup.TransactionID	TransactionID
toLookup.SequenceNumber	SequenceNumb
toLookup.ProductID	ProductID
toLookup.IfIsWeighed	IfIsWeighed
toLookup.Weight	Weight

	Link Name	Condition	Condition Not Met	Lookup Failure
1	fromValidProductID		Reject	Reject

	Column na	Scale	Nullable	Description
1	Transaction		No	
2	SequenceN		No	
3	ProductID		No	
4	IfIsWeighed		No	
5	Weight		No	

Figure 13-15   Specifying Reject Link behavior in a Lookup Operatorr.

iv. After completing Step-5.c.iii above per Figure 13-15, Click OK twice, to close and save all open dialogs.

d. FoundTX and NotFoundTX, (Transformer Operators).

Configuring Properties for these two Transformer Operators involve nearly the same steps. For that reason, we'll cover both sets of instructions at one time.

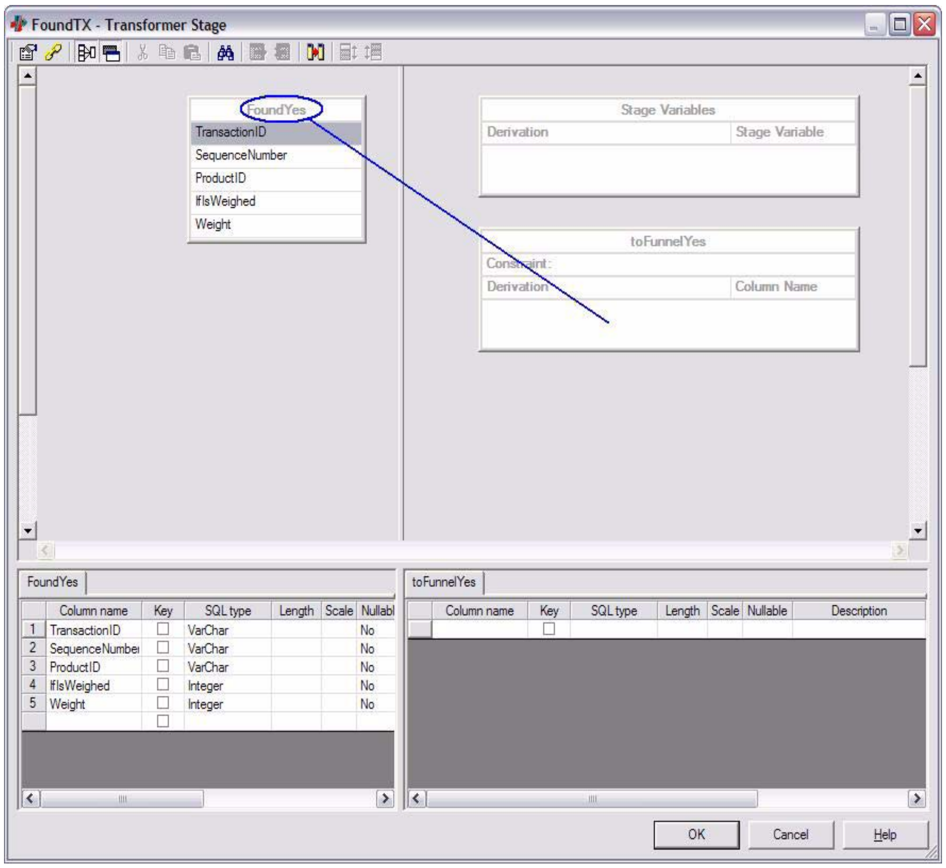
i. Specifying the output columns.

Double-click the FoundTX Transformer Operator to open its Properties dialog.

Click and Hold the word “FoundYes” (our Input Link), Drag, and Release in the white space under “toFunnelYes”. Example as shown in Figure 13-16 in blue.

Note: We’ve done this procedure at least once before. By grabbing the column header, we are grabbing all columns in the group. Alternately, we could perform this task column by column.

If we are using (outputting) all columns, it is faster to grab the column header.



FoundYes

	Column name	Key	SQL type	Length	Scale	Nullabl
1	TransactionID	☐	VarChar			No
2	SequenceNumber	☐	VarChar			No
3	ProductID	☐	VarChar			No
4	IfIsWeighed	☐	Integer			No
5	Weight	☐	Integer			No
		☐				

toFunnelYes

	Column name	Key	SQL type	Length	Scale	Nullable	Description
		☐					

Figure 13-16 Specifying output columns in Parallel Transformer Operator.

ii. Adding a new output column.

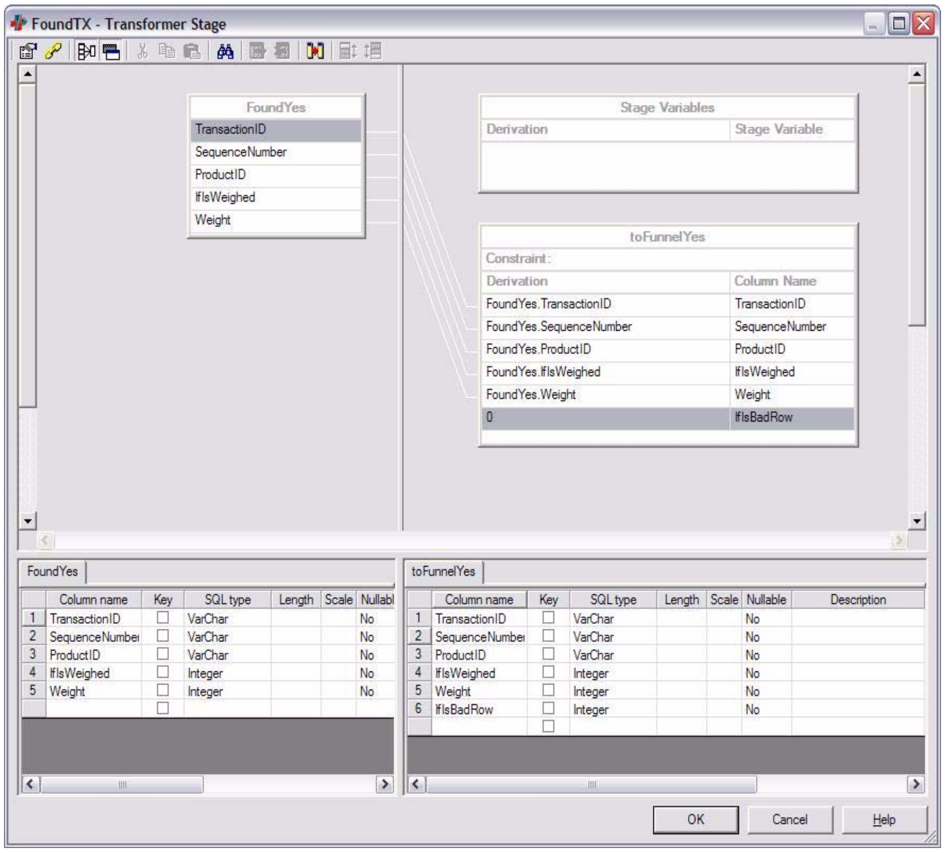
Per our use case, per our design, we wish to add a new output column. This column will track the status of the previous test; was our ProductID found or not found inside the ValidProductID table-

We will use the value of zero for no error, and a value greater than zero for an error of some kind. By using a non-zero value for error, we can record multiple and distinct error conditions.

There is more than one way to add a new output column, perhaps this way is easiest-

In the bottom right panel of the display, below the line that says, "(Line) 5, Weight '', Double-Click the empty (and possibly grey) line, so that you may enter a new value. You may also try a Right-Click on this line, and choose, "Edit Cell".

Complete the entry as shown in Figure 13-17.


FoundTX - Transformer Stage

FoundYes
TransactionID
SequenceNumber
ProductID
IfIsWeighed
Weight

Stage Variables

Derivation	Stage Variable

toFunnelYes

Constraint:

Derivation	Column Name
FoundYes.TransactionID	TransactionID
FoundYes.SequenceNumber	SequenceNumber
FoundYes.ProductID	ProductID
FoundYes.IfIsWeighed	IfIsWeighed
FoundYes.Weight	Weight
0	IfIsBadRow

FoundYes

	Column name	Key	SQL type	Length	Scale	Nullabl
1	TransactionID	☐	VarChar			No
2	SequenceNumbe	☐	VarChar			No
3	ProductID	☐	VarChar			No
4	IfIsWeighed	☐	Integer			No
5	Weight	☐	Integer			No
		☐				

toFunnelYes

	Column name	Key	SQL type	Length	Scale	Nullable	Description
1	TransactionID	☐	VarChar			No	
2	SequenceNumbe	☐	VarChar			No	
3	ProductID	☐	VarChar			No	
4	IfIsWeighed	☐	Integer			No	
5	Weight	☐	Integer			No	
6	IfIsBadRow	☐	Integer			No	
		☐					

Figure 13-17 Adding a new output column to a Transformer Operator.

iii. After you add the new output column entry (we called ours IfIsBadRow, of type Integer, as shown in Figure 13-17), click off of this line (click off of this control). This event causes a new line will appear in the upper right portion of the display.

   This line will appear in red, indicating an error. The error is that you have created this new column, but the Transformer Operator doesn't know what data value to put here.

   For the FoundTX, we wish to place the value of zero; we found a match, no error. For the NotFoundTX, we wish to put a value of 1.

   Per Figure 13-17, and in the upper right area of the display, place a zero in the cell for "Derivation", for the "IfIsBadRow" column entry.

iv. Click OK to complete your work, and close this dialog box.

v. Repeat Steps-5.d.i-iv for the NotFoundTX Transformer Operator.

   Where you had placed a zero for the "Derivation" in Step-5.d.iii, this time pace a 1.

   Click OK when you are done.

Note: The newly added column name created in Step-5.d.iii above (IfIsBadRow) has to match for both Transformer Operators, in order for our work on the next Operator, the Funnel Operator, to work as written below.

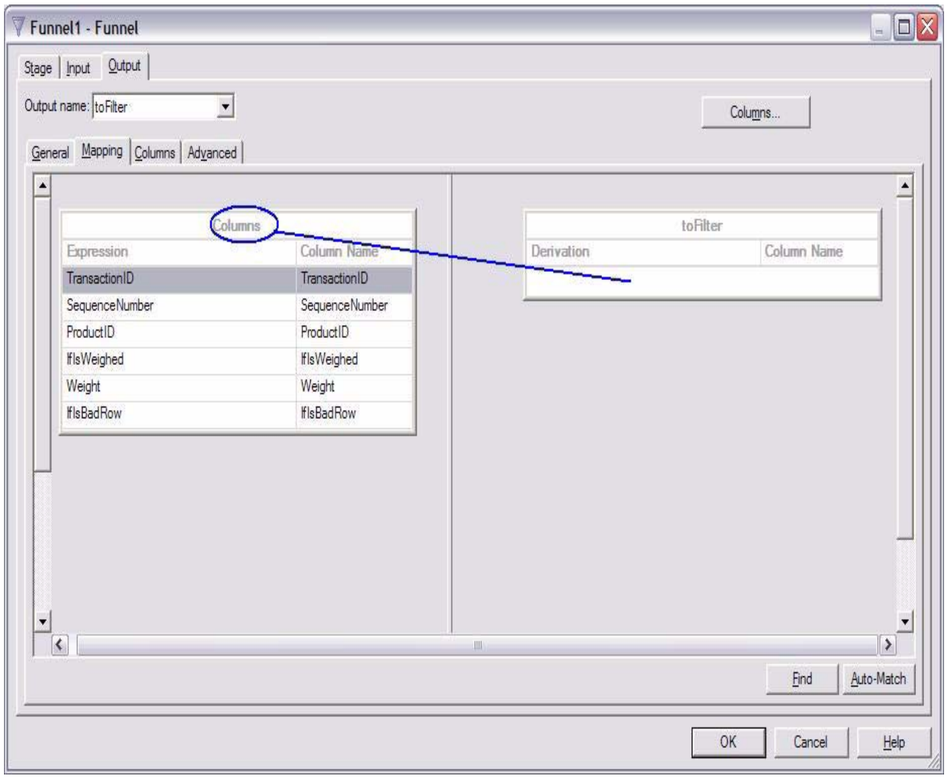
The newly added columns should share the same column name, and data type.

e. Funnel1, (Funnel Operator).

   Funnel Operators take two or more input data streams, and output them as one, merged, output data stream. Per our use case, we had split the data stream in two, so that we could mark data as found or not found (good or bad). For our use case, we need only make one change to the funnel Operator, and that is setting the output columns.

   In the Funnel Operator Properties dialog, Output TAB -> Mapping TAB, Click and Hold the "Columns" entry, Drag, and Release on the white space as shown with the blue line in Figure 13-18.

   Click OK when you are done.



Output name: toFilter

Columns	
Expression	Column Name
TransactionID	TransactionID
SequenceNumber	SequenceNumber
ProductID	ProductID
IfIsWeighed	IfIsWeighed
Weight	Weight
IfIsBadRow	IfIsBadRow

toFilter	
Derivation	Column Name

Figure 13-18   Funnel operator, Output TAB -> Mapping TAB.

f. Filter, (Filter Operator).

The Filter Operator is one of several 'switches' available on the DS/EE Parallel Canvas; a switch variably directs records from the Input Stream, to one or more output Links. We use this Filter Operator to check for the valid weight value our use case presents.

Whereas the previous Lookup Operator had to retrieve additional data to form the validation, a Filter is used here because we have every piece of data we need for a weight check.

i. Setting Filter criteria.

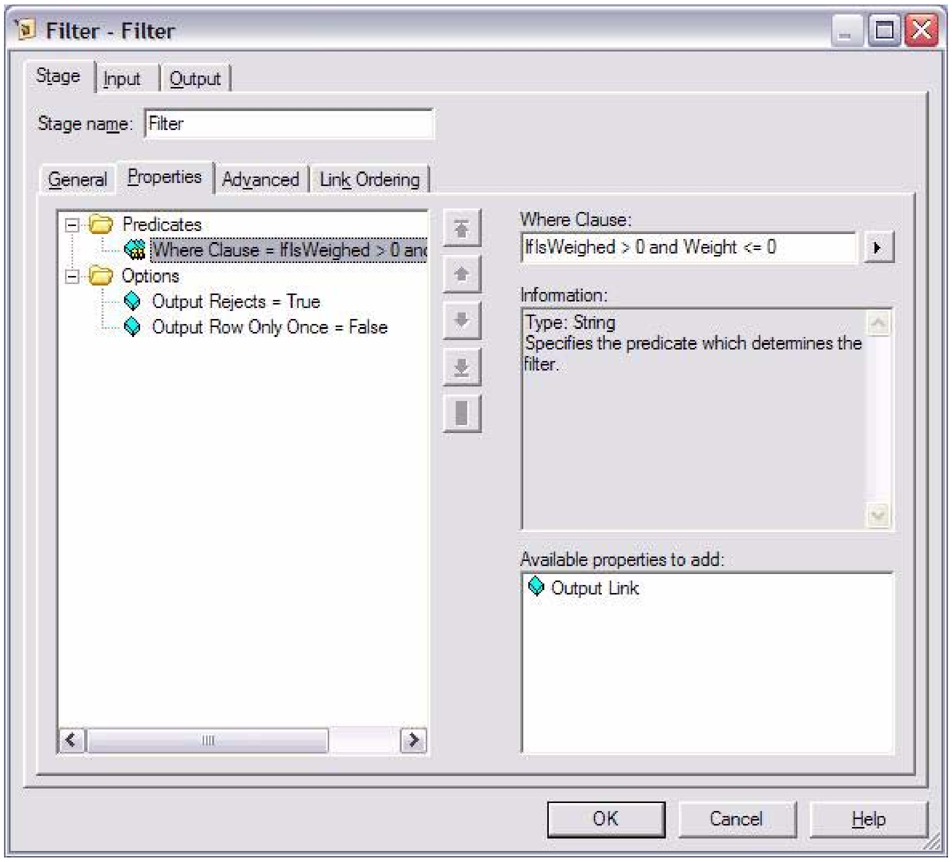
In the Filter Properties dialog box, Stage TAB -> Properties TAB, set the Predicates -> Where Clause value equal to,

```
IfIsWeighed > 0 and Weight <= 0
```

Similar to a SQL WHERE CLAUSE, this expression checks first to see if a given row should have its Weight value checked, and then checks the Weight value.

Then under Options -> Output Rejects, set this value equal to True.

Example as shown in Figure 13-19.



Stage name: Filter

- Predicates
  - Where Clause = IfIsWeighed > 0 and
- Options
  - Output Rejects = True
  - Output Row Only Once = False

Where Clause: IfIsWeighed > 0 and Weight <= 0

Information:
Type: String
Specifies the predicate which determines the filter.

Available properties to add:
Output Link

Figure 13-19 Filter Operator, setting Filter Where Clause.

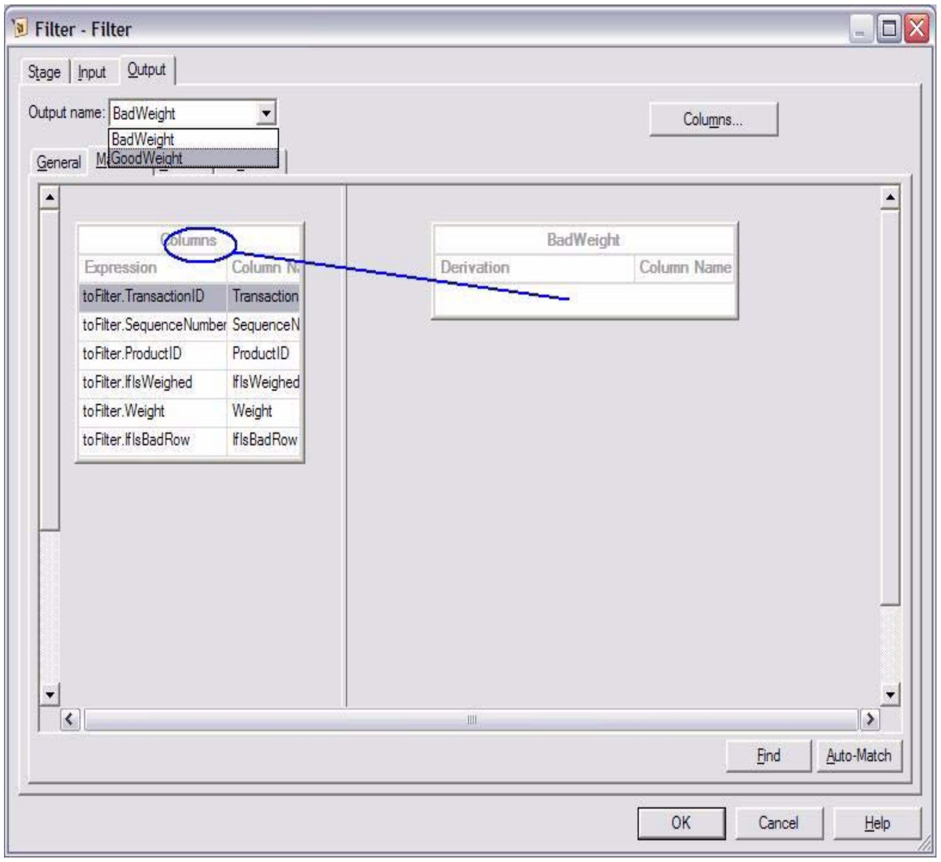
ii. Specifying the output columns.

Under the Output TAB -> Mapping TAB, Click and Hold the word “Columns”, Drag, and Release in the white space as shown by the blue line in Figure 13-20.

This action specifies our output columns for the Output Link.

As a convention, the Reject Link must output the same columns as an Output Link, so there is no need to specify columns for that output.

For future reference, multiple input or output Links are configured by managing a Drop Down List Box, similar to “Output name”, as shown in Figure 13-20.



Output name: BadWeight (BadWeight, GoodWeight)

Expression	Column Name
toFilter.TransactionID	Transaction
toFilter.SequenceNumber	SequenceN
toFilter.ProductID	ProductID
toFilter.IfIsWeighed	IfIsWeighed
toFilter.Weight	Weight
toFilter.IfIsBadRow	IfIsBadRow

BadWeight

Derivation	Column Name

Figure 13-20   Filter Operator, specifying output columns.

iii. Click OK when you are done.

g. GoodWeightTX, and BadWeightTX, (Transformer Operators).

Configuring Properties for these two Transformer Operators involve nearly the same steps. For that reason, we’ll cover both sets of instructions at one time.

Also, configuring these two Transformers is easier than configuring the two Transformers we did in Steps-5.d.1-v.

i. For the GoodWeightTX, simply specify the output columns as we did in Step-5.d.i.

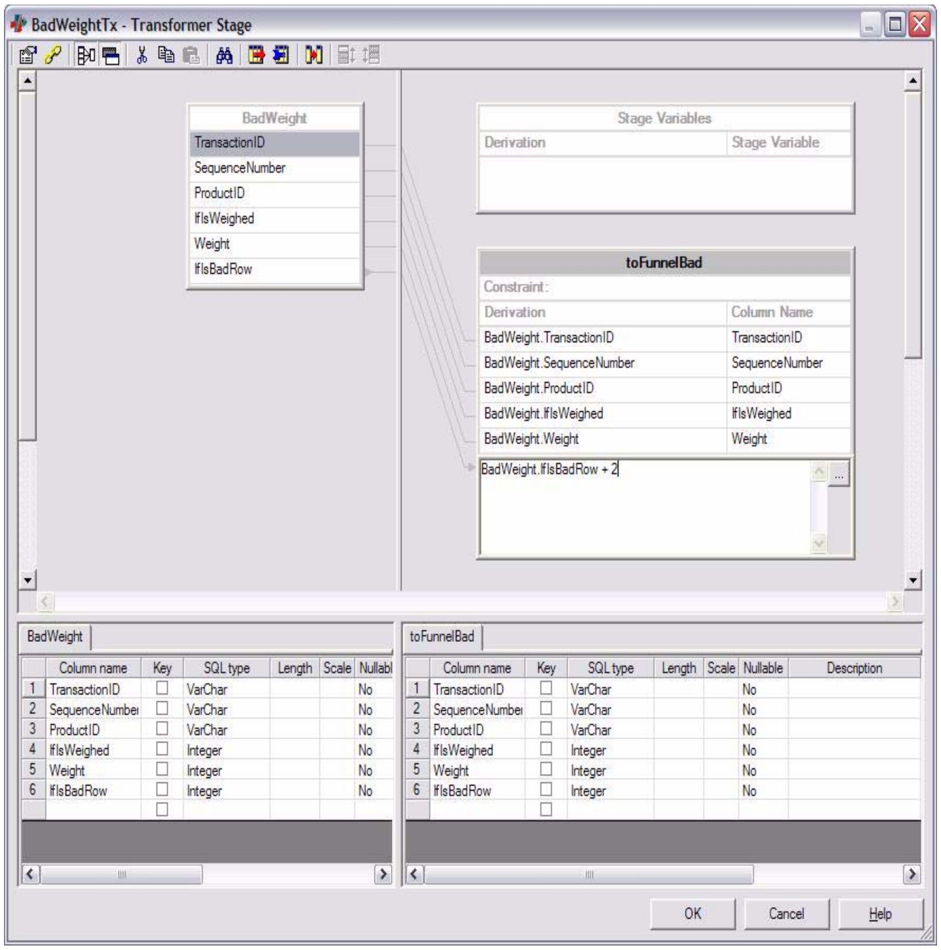
No new columns or other steps need to be done.

Click OK when you are done.

ii. For the BadWeightTX, we wish to specify the output columns as we did in Step-5.d.i, and, we wish to add a numeric 2 to the output column, named, IfIsBadRow.

As shown in Figure 13-21, edit the Derivation for IfIsBadRow, and add a “ + 2”, to the end of the existing expression. The finished expression for IfIsBadRow reads,

BadWeight.IfIsBadRow + 2



BadWeight
TransactionID
SequenceNumber
ProductID
IfIsWeighed
Weight
IfIsBadRow

Stage Variables

Derivation	Stage Variable

toFunnelBad

Constraint:

Derivation	Column Name
BadWeight.TransactionID	TransactionID
BadWeight.SequenceNumber	SequenceNumber
BadWeight.ProductID	ProductID
BadWeight.IfIsWeighed	IfIsWeighed
BadWeight.Weight	Weight

BadWeight.IfIsBadRow + 2

BadWeight

	Column name	Key	SQL type	Length	Scale	Nullabl
1	TransactionID	☐	VarChar			No
2	SequenceNumbe	☐	VarChar			No
3	ProductID	☐	VarChar			No
4	IfIsWeighed	☐	Integer			No
5	Weight	☐	Integer			No
6	IfIsBadRow	☐	Integer			No
		☐				

toFunnelBad

	Column name	Key	SQL type	Length	Scale	Nullable	Description
1	TransactionID	☐	VarChar			No	
2	SequenceNumbe	☐	VarChar			No	
3	ProductID	☐	VarChar			No	
4	IfIsWeighed	☐	Integer			No	
5	Weight	☐	Integer			No	
6	IfIsBadRow	☐	Integer			No	
		☐					

Figure 13-21 Transformer Operator, Setting the Derivation Expression for IfIsBadRow.

iii. Click OK when your display equals what is shown in Figure 13-21.

h. Funnel 2, (Funnel Operator).

Funnel 2 is configured in the exact same manner as Funnel1, above.

Complete the work outlined in Step-5.e, but on the Funnel2 Operator.

Click OK when you are done.

i. Sort1, (Sort Operator).

To configure the Properties for the Sort Operator, we need to do work on two of the Property TABs.

i. Setting the Sorting Keys.

Figure 13-22 displays the Sort Operator, Stage TAB -> Properties TAB.

Set the first Sorting Key -> Key to equal the input column, TransactionID. A pick list is available on the right side of the display when this visual control is current.

We also need to add a second sorting key, (a sub-key). This is done with a Right-Click on "Sorting Keys", and then selecting, "Add sub-property -> Key".

Make the second Sorting Key -> Key, "IfIsBadRow".

Only for "IfIsBadRow", set the Sorting Keys -> Key -> Sort Order to, "Descending".

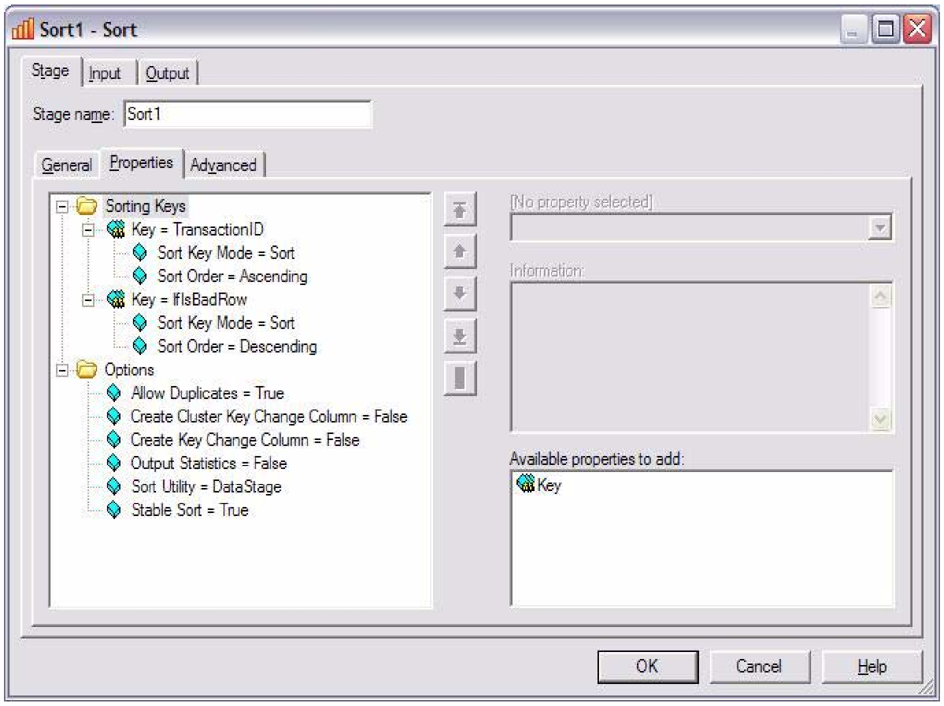
You are done when your display equals what is shown in Figure 13-22.

Note: We sort first on TransactionID, so that the input data stream sends records in groups (sub-groups) of whole transactions.

Then we sort (sub-sort) on IfIsBadRow in a *descending* sequence.

Good rows carry a value of zero, bad rows carry a value greater than zero. By sorting on IfIsBadRow *descending*, we are sending any bad records at the top (beginning) of a given transaction.

If a given transaction has *any* bad records, those records will appear first in the transaction sub-group, allowing us to mark the whole transaction bad, as we see in the next Operator, a Transformer.



Stage name: Sort1

- Sorting Keys
  - Key = TransactionID
    - Sort Key Mode = Sort
    - Sort Order = Ascending
  - Key = IfIsBadRow
    - Sort Key Mode = Sort
    - Sort Order = Descending
- Options
  - Allow Duplicates = True
  - Create Cluster Key Change Column = False
  - Create Key Change Column = False
  - Output Statistics = False
  - Sort Utility = DataStage
  - Stable Sort = True

Available properties to add:
Key

Figure 13-22   Sort Operator, Stage TAB -> Properties TAB.

ii. Specifying the output columns.

Under the Output TAB -> Mapping TAB, specify that all columns are to be output, like we have done many times, most recently in Step-5.f.ii.

iii. Click OK when you are done.

Note: We are nearly done creating this example. The next Operator, a Transformer, has the most complexity to it. This Transformer uses Stage Variables, which if you have never used, take a few minutes to understand.

We will say that (Transformer) Stage Variables operate much like procedural programming variables *from an interpretive language*, like the original BASIC programming language variables and their behavior, if that helps.

If you understand what we are doing with this Operator, you are well on your way to becoming a DS/EE uber-practitioner.

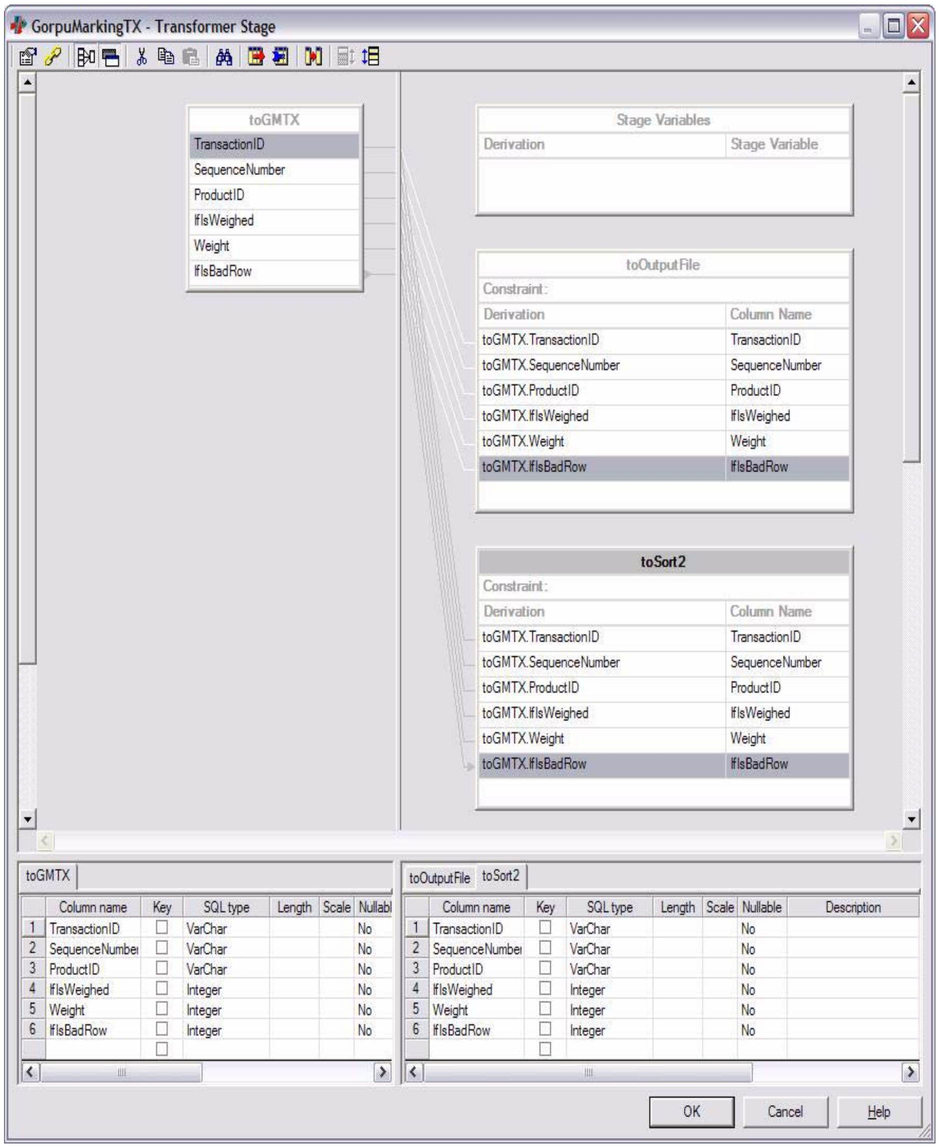
j. GroupMarkingTX, (Transformer Operator).

This is the money Operator in this whole DS/EE Job; the Operator that carries the most strategic functionality. First we’ll perform some of the tasks we already know how to perform from our earlier work.

i. Specify the output columns.

   This Operator has two Output Links, so we have the specify the output columns twice. This is done via two Drag and Drops, in a manner similar to what we did many times before.

   You are done specifying output columns when your display equals what is shown in Figure 13-23.



toGMTX

toGMTX
TransactionID
SequenceNumber
ProductID
IfIsWeighed
Weight
IfIsBadRow

Stage Variables

Derivation	Stage Variable

toOutputFile

Constraint:

Derivation	Column Name
toGMTX.TransactionID	TransactionID
toGMTX.SequenceNumber	SequenceNumber
toGMTX.ProductID	ProductID
toGMTX.IfIsWeighed	IfIsWeighed
toGMTX.Weight	Weight
toGMTX.IfIsBadRow	IfIsBadRow

toSort2

Constraint:

Derivation	Column Name
toGMTX.TransactionID	TransactionID
toGMTX.SequenceNumber	SequenceNumber
toGMTX.ProductID	ProductID
toGMTX.IfIsWeighed	IfIsWeighed
toGMTX.Weight	Weight
toGMTX.IfIsBadRow	IfIsBadRow

toGMTX

	Column name	Key	SQL type	Length	Scale	Nullabl
1	TransactionID	☐	VarChar			No
2	SequenceNumbe	☐	VarChar			No
3	ProductID	☐	VarChar			No
4	IfIsWeighed	☐	Integer			No
5	Weight	☐	Integer			No
6	IfIsBadRow	☐	Integer			No
		☐				

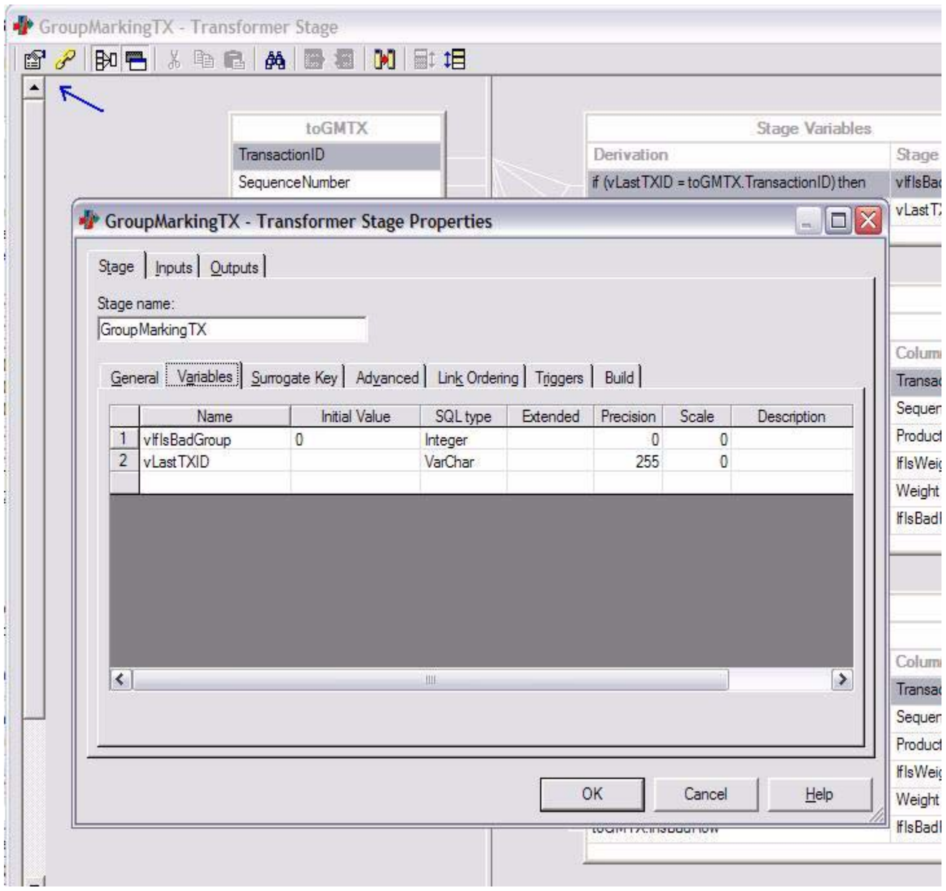
toOutputFile toSort2

	Column name	Key	SQL type	Length	Scale	Nullable	Description
1	TransactionID	☐	VarChar			No	
2	SequenceNumbe	☐	VarChar			No	
3	ProductID	☐	VarChar			No	
4	IfIsWeighed	☐	Integer			No	
5	Weight	☐	Integer			No	
6	IfIsBadRow	☐	Integer			No	
		☐					

Figure 13-23 Transformer Operator, specifying output columns.

ii. Declaring (Defining) Stage Variables.

The left most icon in the Tool Bar gives access to the Transformer Stage Properties dialog. Click that icon to produce the display as shown in Figure 13-24.


GroupMarkingTX - Transformer Stage Properties

Stage name: GroupMarkingTX

	Name	Initial Value	SQL type	Extended	Precision	Scale	Description
1	vIfIsBadGroup	0	Integer		0	0	
2	vLastTXID		VarChar		255	0	

Figure 13-24 Transformer Operator, Declaring Stage Variables.

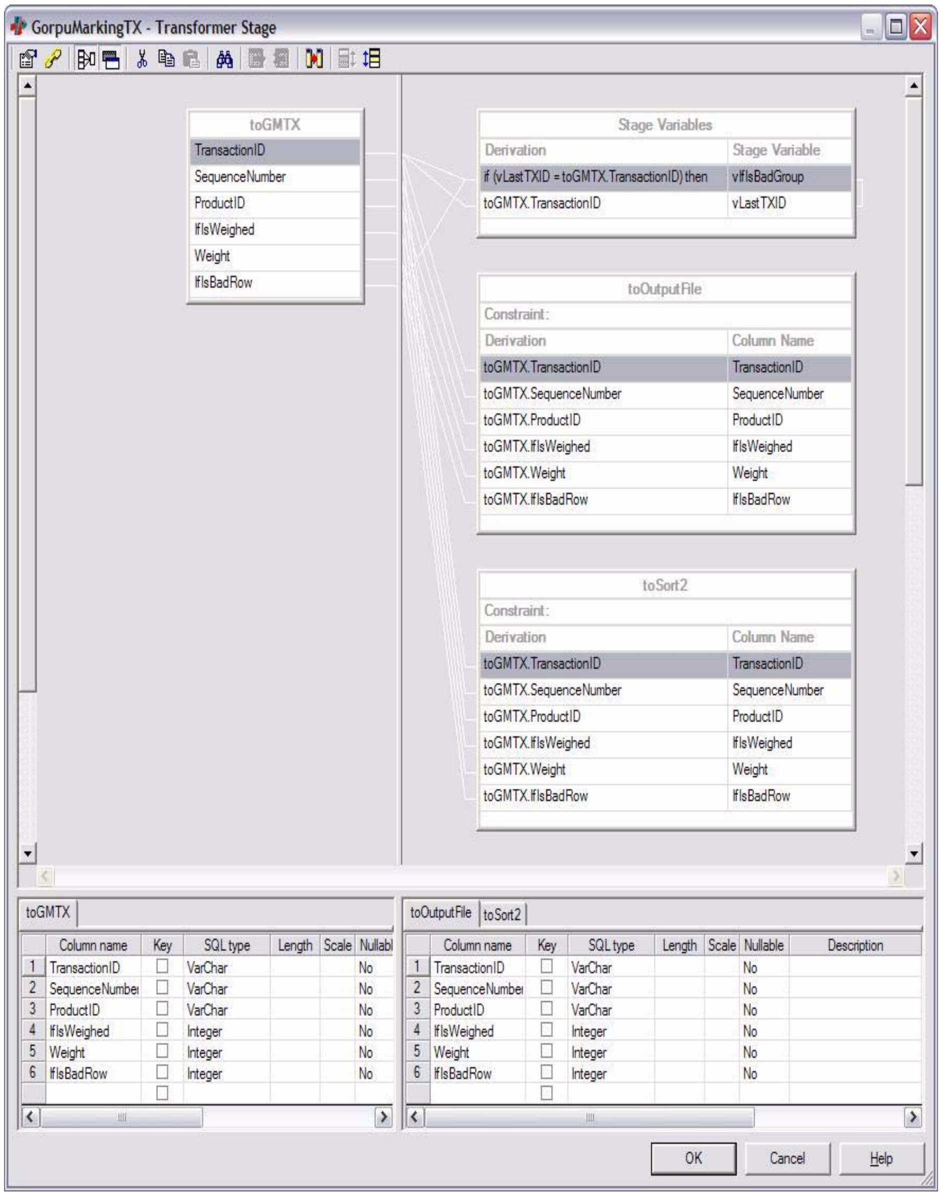
As shown in Figure 13-24, and on the Stage TAB -> Variables TAB, add two new Transformer Operator Stage Variables, as shown.

Click OK when you are done.

iii. Setting Stage Variable Derivation Expressions.

Thus far we have specified the output columns from this Operator, and we have declared (defined) two Stage Variables. Now it is time to set the Derivation Expressions for each of these two variables; define the values that they will hold.

Upon returning from the previous step, Step-5.j.ii, we see now that there are two new entries under "Stage Variables", as displayed in Figure 13-25.



toGMTX
TransactionID
SequenceNumber
ProductID
IfIsWeighed
Weight
IfIsBadRow

Stage Variables

Derivation	Stage Variable
if (vLastTXID = toGMTX.TransactionID) then	vIfIsBadGroup
toGMTX.TransactionID	vLastTXID

toOutputFile

Constraint:

Derivation	Column Name
toGMTX.TransactionID	TransactionID
toGMTX.SequenceNumber	SequenceNumber
toGMTX.ProductID	ProductID
toGMTX.IfIsWeighed	IfIsWeighed
toGMTX.Weight	Weight
toGMTX.IfIsBadRow	IfIsBadRow

toSort2

Constraint:

Derivation	Column Name
toGMTX.TransactionID	TransactionID
toGMTX.SequenceNumber	SequenceNumber
toGMTX.ProductID	ProductID
toGMTX.IfIsWeighed	IfIsWeighed
toGMTX.Weight	Weight
toGMTX.IfIsBadRow	IfIsBadRow

toGMTX

	Column name	Key	SQL type	Length	Scale	Nullabl
1	TransactionID	☐	VarChar			No
2	SequenceNumbe	☐	VarChar			No
3	ProductID	☐	VarChar			No
4	IfIsWeighed	☐	Integer			No
5	Weight	☐	Integer			No
6	IfIsBadRow	☐	Integer			No
		☐				

toOutputFile | toSort2

	Column name	Key	SQL type	Length	Scale	Nullable	Description
1	TransactionID	☐	VarChar			No	
2	SequenceNumbe	☐	VarChar			No	
3	ProductID	☐	VarChar			No	
4	IfIsWeighed	☐	Integer			No	
5	Weight	☐	Integer			No	
6	IfIsBadRow	☐	Integer			No	
		☐					

Figure 13-25 Transformer Operator, setting Stage Variable Derivation Expressions.

We need to edit the Derivation Expression to each of these two variables. The Derivation Expression for the first Stage Variable, vIfIsBadGroup, is more complex. And it is;

```
if (vLastTXID = toGMTX.TransactionID) then
    vIfIsBadGroup
  else
    if (toGMTX.IfIsBadRow > 0) then
      1
    else
      0
```

The Derivation Expression for the second Stage Variable, vLastTXID, is easier. And it is;

```
toGMTX.TransactionID
```

Edit the Derivation Expressions for each of these two Stage Variables to equal what is written above. Derivation Expressions are white space insensitive, you need not worry about extra spacing, blank lines, etcetera. (To enter a blank line inside the editor, press Control-M.) A code review explaining this text follows.

.

Note: Basically this Operator has (n) input columns, and it has two Stage Variables.

The input column values change upon the receipt of each new input row that is received. These column values equal whatever we receive from the input stream.

The Stage Variable values also change upon the receipt of each new input row that is received. These Stage Variable values equal whatever we say they should equal, per the Stage Variable’s associated Derivation Expression.

A key point, is that Stage Variables are set, one after another, in order, after the receipt of a new input stream record. What does that mean? As the first Stage Variable is evaluated/set, any subsequent Stage Variables equal their old value; a history of sorts.

The Stage Variable, vLastTXID, is used to detect an after group condition; that is, the time at which we have a new input record, and it is from a new TransactionID sub-group.

We need to know when we enter a new TransactionID sub-group, because we wish to mark all records in a Transaction sub-group as valid or invalid.

First this Stage Variable, vLastTXID, equals zero (per its declaration), then it equals the current TransactionID. As stated in the italicized comment above, any Stage Variable evaluated/set before this Stage Variable, before vLastTXID, can see the value from the previous TransactionID column, because of how we manage/set vLastTXID.

This wouldn’t work if vLastTXID were defined first. This works because vLastTXID is defined after the Stage Variable that reads it, in our case, vIfIsBadGroup.

Derivation Expressions (for each Stage Variable)

The Derivation Expression for vLastTXID is just,

toGMTX.TransactionID

(Set vLastTXID equal to the current input stream column value for TransactionID.)

Note: The Derivation Expression for vIfIsBadGroup is more complex. A code review with line numbers follows;

```
01 if ( vLastTXID = toGMTX.TransactionID ) then
02    vIfIsBadGroup
03 else
04    if ( toGMTX.IfIsBadRow > 0 ) then
05       1
06    else
07       0
```

The Stage Variable, vIfIsBadGroup, equals 1 (true) when a given TransactionID contains 1 or more bad line items, and zero (false) otherwise.

- Line 01 will be true, upon a continuing TransactionID being received; meaning, the second or subsequent line item for a given TransactionID. In this case, Let vIfIsBadGroup equal whatever it equalled before, as specified by Line 02.
- Line 03 is the 'else' to Line 01; meaning, we have the first line item of a new TransactionID, and we have work to do.
- The individual line items for a given TransactionID are sorted, bad rows first, as specified by the previous Operator, Sort1. Whether a given row is bad or not, is reflected in the column value, IfIsBadRow.
- Line 04 checks to see if IfIsBadRow is zero or non-zero. If its non-zero, we execute Line 05, else we execute Line 07.
- Line 05 sets vIfIsBadGroup to 1 (true).
- Line 07 sets vIfIsBadGroup to 0 (false).

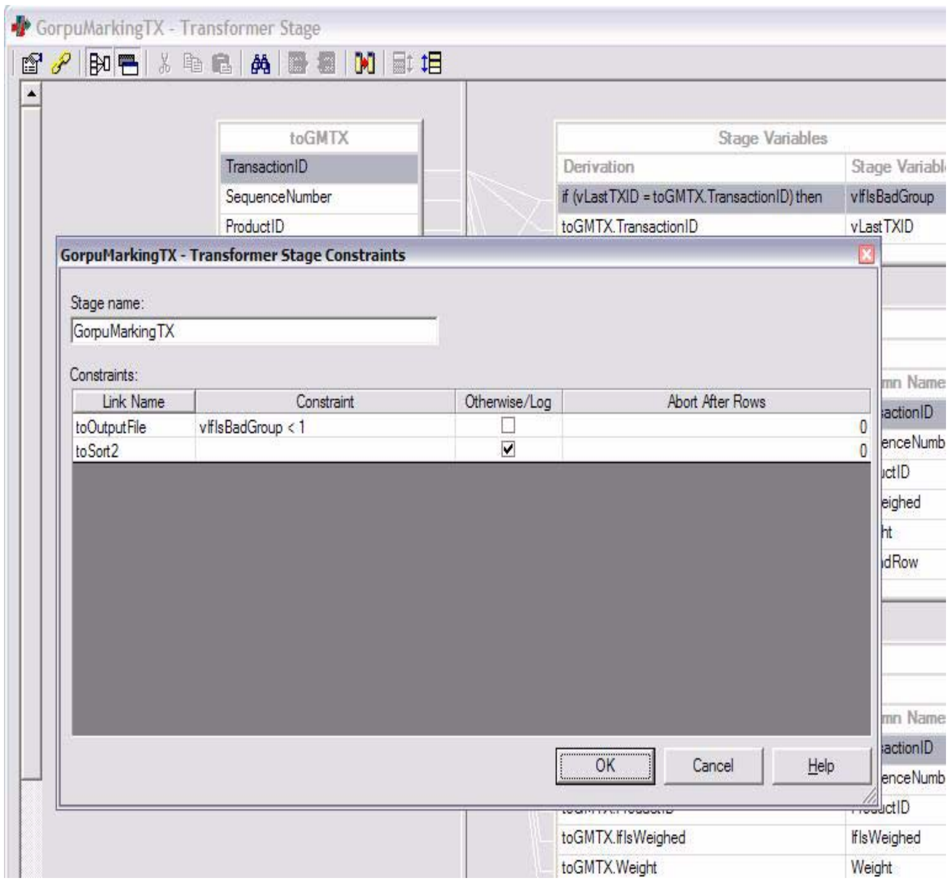
Why is vIfIsBadGroup the variable that is set ? Because this code is in the Derivation Expression for vIfIsBadGroup.

Why are we able to view the prior value of the TransactionID, as reflected in the Stage Variable, vLastTXID ? Because this Derivation Expression precedes setting vLastTXID to the current TransactionID.

iv. Setting the Output Link behavior.

Thus far we have set the output columns, declared two Stage Variables, and set the Derivation Expression for each of these two Stage Variables. Now it is time to set a condition on our two Output Links.

Figure 13-26 displays what we need to perform.



Stage name: GorpuMarkingTX

Constraints:

Link Name	Constraint	Otherwise/Log	Abort After Rows
toOutputFile	vIfIsBadGroup < 1	☐	0
toSort2		☑	0

Figure 13-26 Transformer Operator, setting Link Properties.

In the Tool Bar, Click the Link icon, which is the second icon, left to right. This action will produce the dialog box as displayed in Figure 13-26.

Enter the Condition for the first Link (toOutputFile) as shown,

```
vIfIsBadGroup < 1
```

This Condition specifies that this Link is only to receive input stream records which meet this criteria. This Condition represents whole TransactionID sub-groups without any participating bad line items.

The next Link is specified to be an Otherwise device; basically, any records which were not previously output above. This forms, in effect, our Reject Link.

Manage your display to equal what is shown in Figure 13-26.

Click OK twice to exit the Transformer Properties dialog.

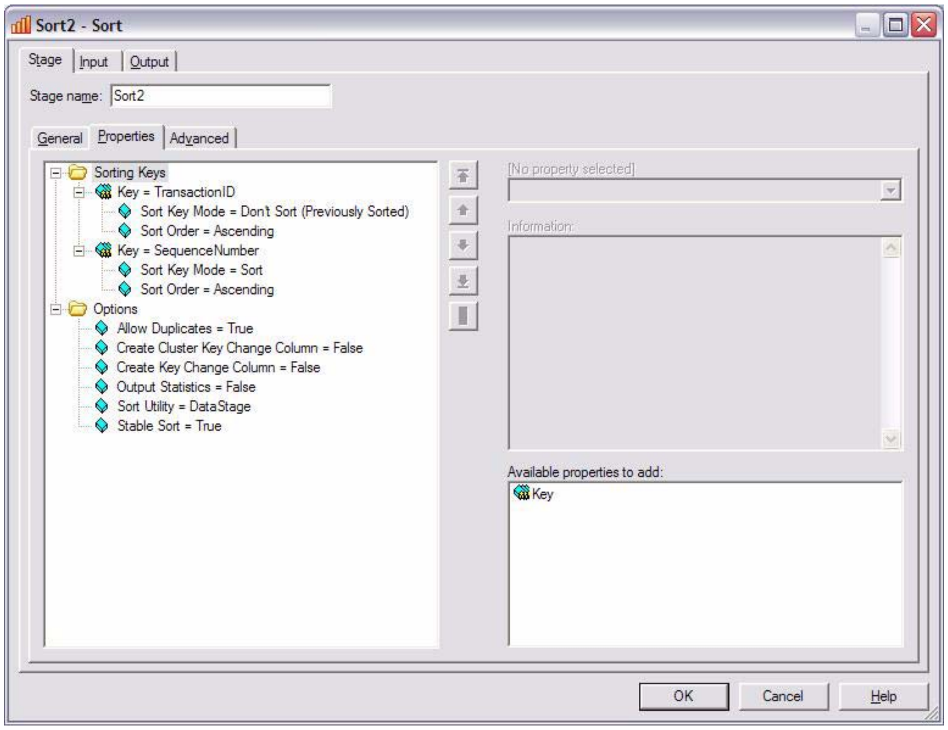
k. Sort2, (Sort Operator).

Per our use case, we didn't really need this Sort Operator. The primary reason to include it is to demonstrate a tuning technique that we were very adjacent to.

Currently data in the input stream is sorted by TransactionID, and then by IfIsBadRow. If we wish to output data now, sorted by TransactionID and then by SequenceNumber, *that would cause very little additional work, if we configure the Sort Operator correctly.*

Figure 13-27 displays how to specify a previously sorted TransactionID, and then a newly sorted SequenceNumber.

- You should set your dialog equal to what is displayed in Figure 13-27.
- You must also specify output columns, from the Output TAB -> Mapping TAB.
- Then Click OK to close this dialog.



Stage name: Sort2

```
Sorting Keys
    Key = TransactionID
        Sort Key Mode = Don't Sort (Previously Sorted)
        Sort Order = Ascending
    Key = SequenceNumber
        Sort Key Mode = Sort
        Sort Order = Ascending
Options
    Allow Duplicates = True
    Create Cluster Key Change Column = False
    Create Key Change Column = False
    Output Statistics = False
    Sort Utility = DataStage
    Stable Sort = True
```

Figure 13-27   Sort operator, taking advantage of a previously sorted data stream.

l. OutputFile, and RejectFile, (Sequential File Operators).

   Much like the InputFile in Step-5.a.i-ii, above, you must now specify the File (name) for each of these two output files.

   For style only, you may also wish to specify field delimiters for both files and the like.

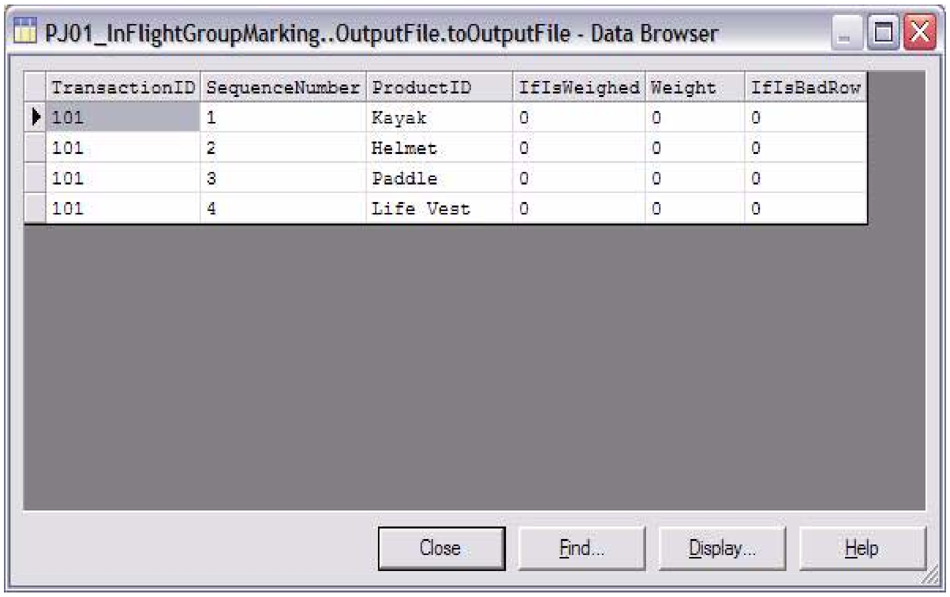
   When you are done, you ar ready to compile and test your DS/EE Job.

6. Compile and test your DS/EE Job.

   a. From the Menu Bar, select, File -> Compile, and then, File -> Run.

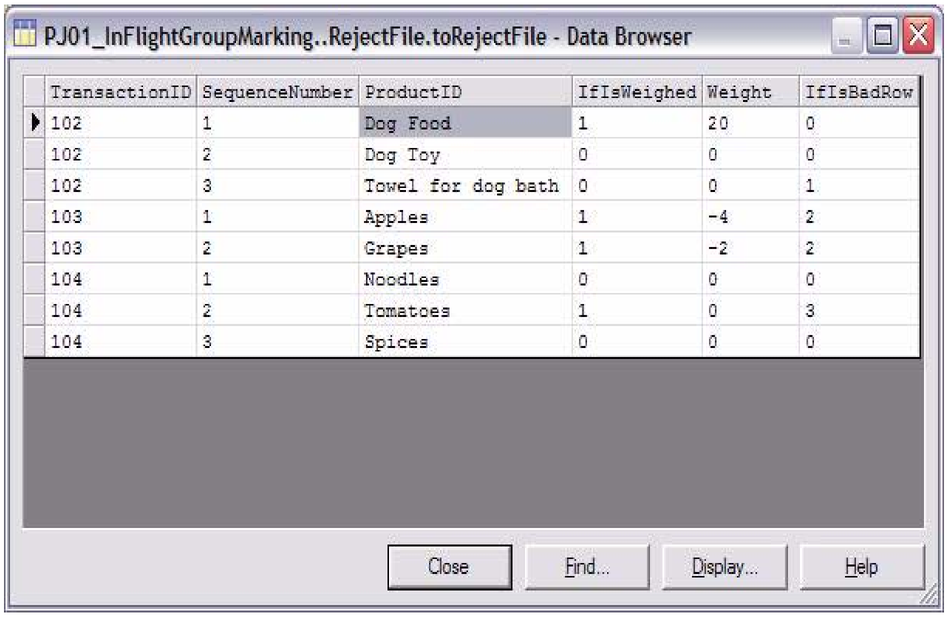
   b. You can view your results directly by a Right-Click on either the OutputFile or RejectFile, and selecting, View {object} data.

      Our sample results appear in Figure 13-28 and Figure 13-29.


PJ01_InFlightGroupMarking..OutputFile.toOutputFile - Data Browser
Close
Find...
Display...
Help

TransactionID	SequenceNumber	ProductID	IfIsWeighed	Weight	IfIsBadRow
101	1	Kayak	0	0	0
101	2	Helmet	0	0	0
101	3	Paddle	0	0	0
101	4	Life Vest	0	0	0

Figure 13-28   Output from OutputFile, the good /whole Transactions.


PJ01_InFlightGroupMarking..RejectFile.toRejectFile - Data Browser
Close
Find...
Display...
Help

TransactionID	SequenceNumber	ProductID	IfIsWeighed	Weight	IfIsBadRow
102	1	Dog Food	1	20	0
102	2	Dog Toy	0	0	0
102	3	Towel for dog bath	0	0	1
103	1	Apples	1	-4	2
103	2	Grapes	1	-2	2
104	1	Noodles	0	0	0
104	2	Tomatoes	1	0	3
104	3	Spices	0	0	0

Figure 13-29   Output from RejectFile, the bad/whole-or-in-part Transactions.

13.3 Summation

In this document, we reviewed or created:

We overviewed how to improve the performance of IBM Information Server (IIS) DataStage Enterprise Edition Component (DS/EE) Jobs, using a design pattern which this document refers to as, In Flight Group Marking (IFGM). IFGM is faster, and generally less resource consumptive as other means to process data; other means which often land data to disk unnecessarily.

Indirectly then, we detailed how to use DS/EE (Parallel) Transformer Operator Stage Variables, including how to use Stage Variables to detect after group (sub-group) conditions. And we detailed how to program multiple output Links from a Transformer Operator.

Persons who help this month.

Allen Spayth.

Additional resources:

None this month.

Legal statements:

Special attributions: