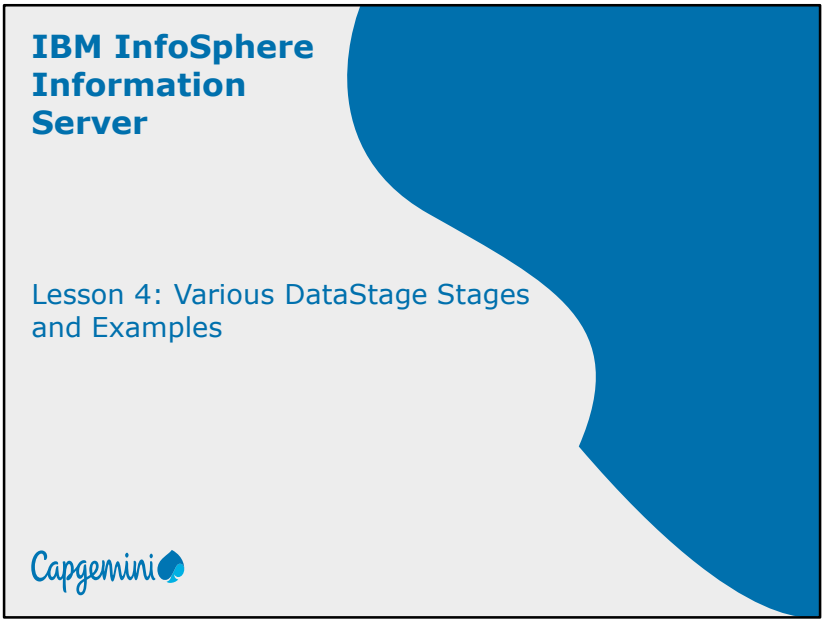

IBM InfoSphere
Information
Server
Lesson 4: Various DataStage Stages
and Examples
Capgemini

Lesson Objectives

- On completion of this lesson, you will be able to:
- Type of Stages
- Accessing Sequential Data
- Reading and Writing NULL Values to a Sequential File
- DataSet Stage
- Combining Data (Lookup,Join,Merge,Funnel)
- Sorting, Aggregating Data And Remove Duplicates(Sort,Aggregator,Remove Duplicates)
- Transforming Data
- Connectors Stages (Oracle, DB2)
- Building SQL to Write to a Table
- Change Capture Stage

Types Of Stages

Design Elements of Parallel Jobs

- Stages
  - Implemented as OSH operators (pre-built components)
  - Passive stages (E and L of ETL)
    - Read data
    - Write data
    - •E.g., Sequential File, DB2, Oracle, Peek stages
  - Processor (active) stages (T of ETL)
    - Transform data
    - Filter data
    - Aggregate data
    - Generate data
    - Split / Merge data
    - E.g., Transformer, Aggregator, Join, Sort stages
- Links
  - "Pipes" through which the data moves from stage to stage

Accessing Sequential Data

Unit objectives

- After completing this unit, you should be able to:
- Understand the stages for accessing different kinds of file data
- Sequential File stage
- Data Set stage
- Create jobs that read from and write to sequential files
- Create Reject links
- Work with NULLs in sequential files
- Read from multiple files using file patterns
- Use multiple readers

Types of File Data

- Sequential
  - –Fixed or variable length
- Data Set
- Complex flat file

Several stages handle sequential data. Each stage has both advantages and differences from the other stages that handle sequential data.

Sequential data can come in a variety of types -- including both fixed length and variable length.

How Sequential Data is Handled

- Import and export operators are generated
  - Stages get translated into operators during the compile
- Import operators convert data from the external format, as described by the Table Definition, to the framework internal format
  - Internally, the format of data is described by schemas
- Export operators reverse the process
- Messages in the job log use the "import" / "export" terminology
  - E.g., "100 records imported successfully; 2 rejected"
  - E.g., "100 records exported successfully; 0 rejected"
  - Records get rejected when they cannot be converted correctly during the import or export

Using the Sequential File Stage

Both import and export of sequential files (text, binary) can be performed by the SequentialFile Stage.

– Data import:  Internal format

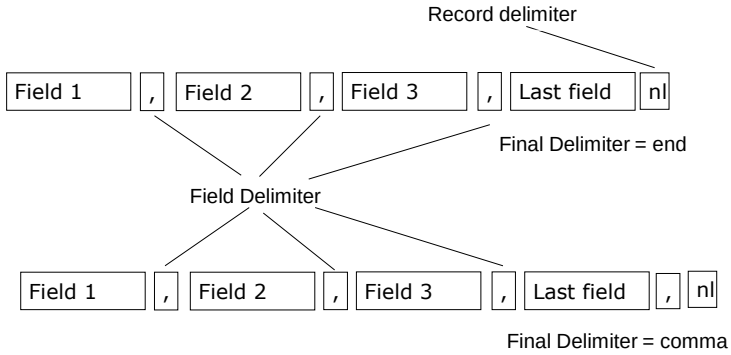
– Data export: Internal format 

When data is imported the imported operator translates that data into the framework internal format. The export operator performs the reverse action.

Features of Sequential File Stage

- Normally executes in sequential mode
- Executes in parallel when reading multiple files
- Can use multiple readers within a node
  - Reads chunks of a single file in parallel
- The stage needs to be told:
  - How file is divided into rows (record format)
  - How row is divided into columns (column format)

Sequential File Format Example

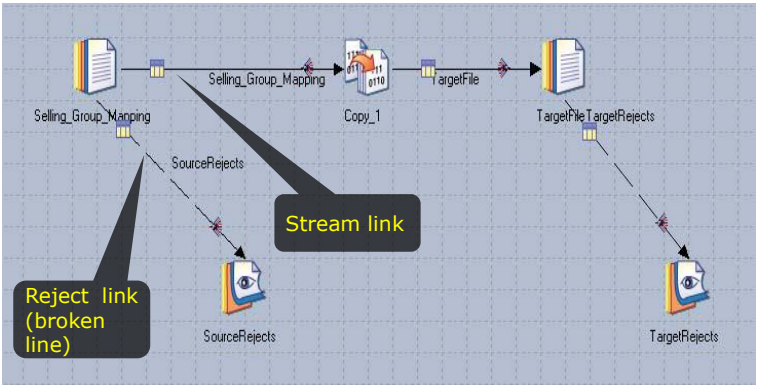


This shows the format of one type of sequential file. Delimiters separate columns. Fields all columns are defined my delimiters. Similarly, records are defined by terminating characters.

Sequential File Stage Rules

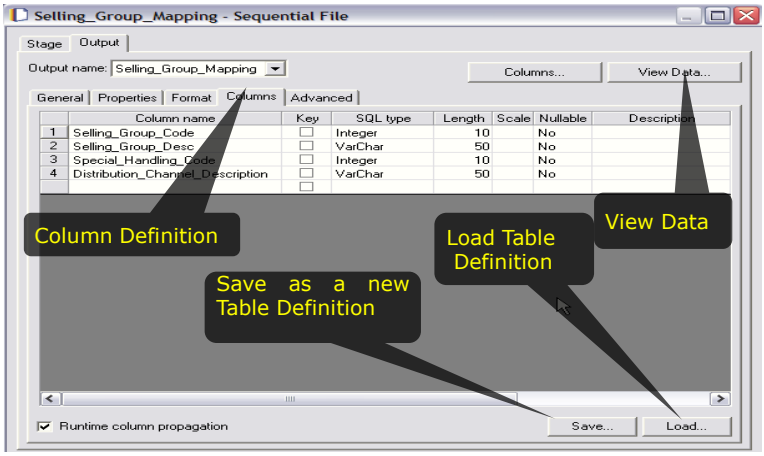
- One input link
- One stream output link
- Optionally, one reject link
  - Will reject any records not matching metadata in the column definitions
    - Example: You specify three columns separated by commas, but the row that's read had no commas in it
    - Example: The second column is designated a decimal in the Table Definition, but contains alphabetic characters

Job Design Using Sequential Stages



The source Sequential stage has a stream output link and a reject output link. The target Sequential stage has an input link and a reject, output link.

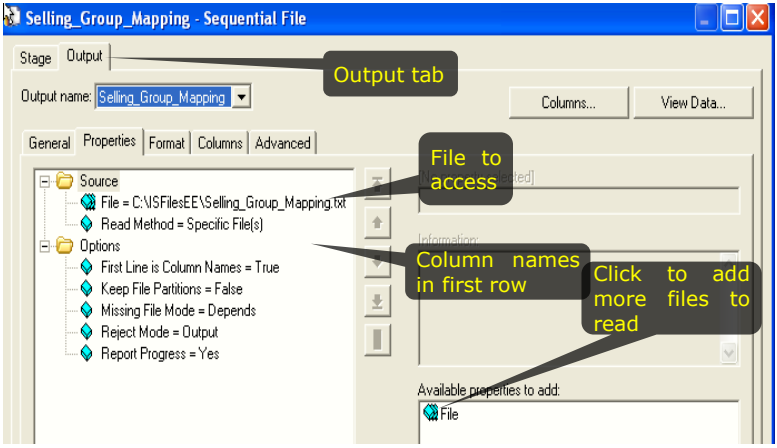
Sequential Source Columns Tab



Sequential Source Columns Tab

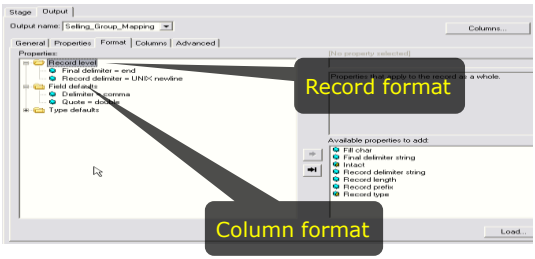
- The Columns tab has these features:
  - Column definitions,
  - View Data,
  - Load button.
  - Save button
    - The Save button is used to save the column definitions into the Repository as a new Table Definition.

Input Sequential Stage Properties

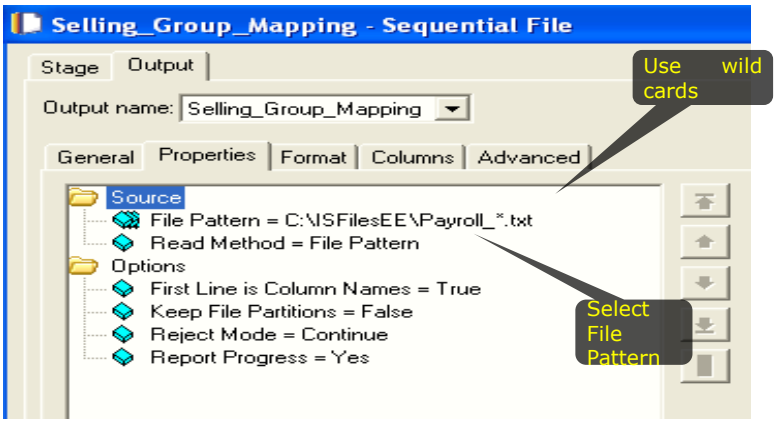


Format Tab

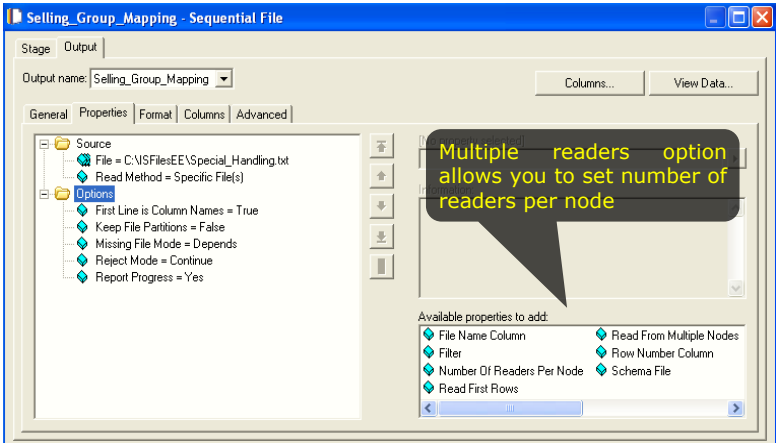
- DataStage needs to know:
  - How a file is divided into rows
  - How a row is divided into columns
- Column properties set on this tab are defaults for each column; they can be overridden at the column level (from columns tab).



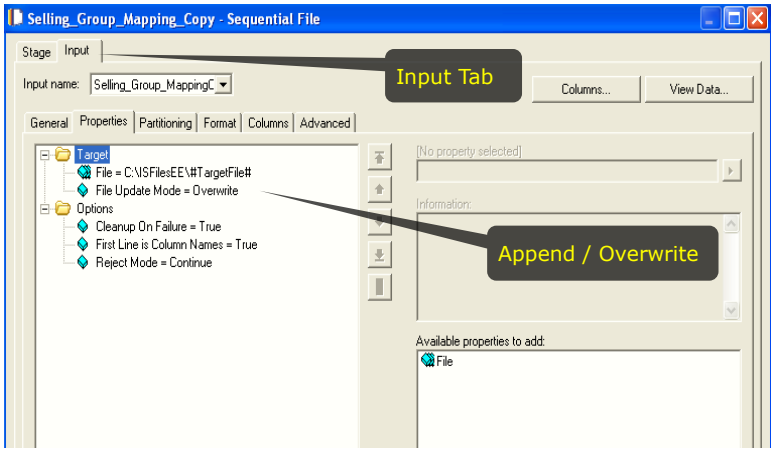
Reading Using a File Pattern



Properties - Multiple Readers

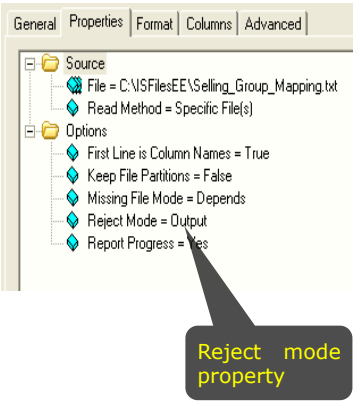


Sequential Stage As a Target



Reject Link

- Reject mode =
  - Continue: Continue reading records
  - Fail: Abort job
  - Output: Send down output link
- In a source stage
  - All records not matching the metadata (column definitions) are rejected
- In a target stage
  - All records that fail to be written for any reason
- Rejected records consist of one column, datatype = raw



Reject Link

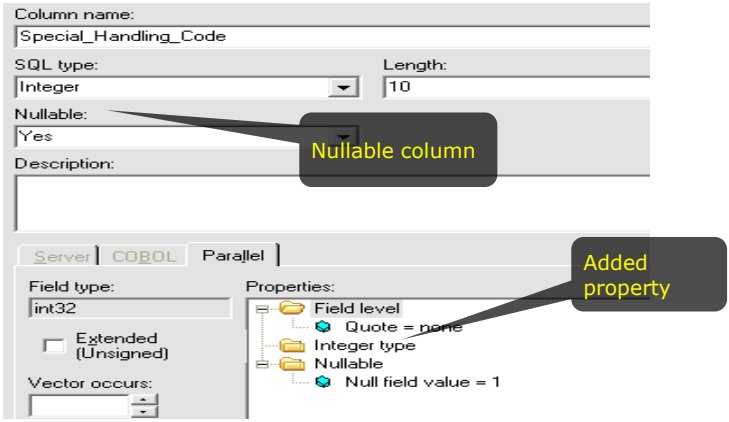
- The sequential stage can have a single reject link.
- This is typically used when you are writing to a file and provides a location where records that have failed to be written to a file for some reason can be sent.
- When you are reading files, you can use a reject link as a destination for rows that do not match the expected column definitions.

Reading and Writing Null Values

Working with NULLs

- Internally, NULL is represented by a special value outside the range of any existing, legitimate values
- If NULL is written to a non-nullable column, the job will abort
- Columns can be specified as nullable
  - NULLs can be written to nullable columns
- You must "handle" NULLs written to nullable columns in a Sequential File stage
  - You need to tell DataStage what value to write to the file
  - Unhandled rows are rejected
- In a Sequential File source stage, you can specify values you want DataStage to convert to NULLs

Specifying a Value for NULL



Dataset

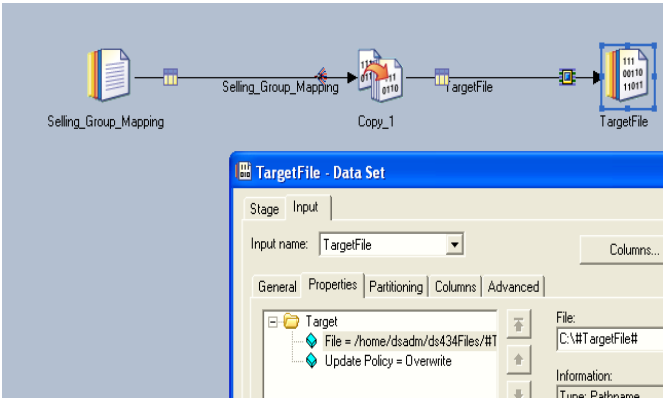
Data Set

- Binary data file
- Preserves partitioning
  - Component dataset files are written to each partition
- Suffixed by .ds
- Referred to by a header file
- Managed by Data Set Management utility from GUI (Manager, Designer, Director)
- Represents persistent data maintained in the internal format
- Key to good performance in set of linked jobs
  - No import / export conversions are needed
  - No repartitioning needed

Data Set

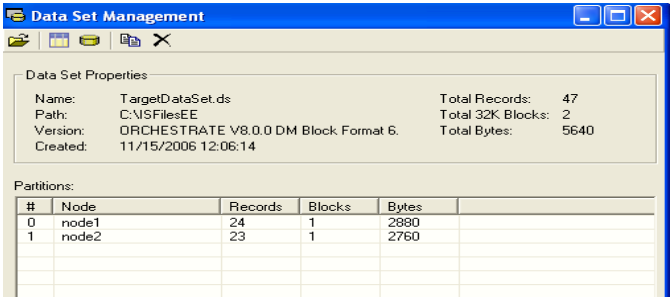
- Accessed using DataSet stage
- Implemented with two types of components:
  - Descriptor file:
- contains metadata, data location, but NOT the data itself
  - Data file(s)
    - contain the data
    - multiple files, one per partition (node)

Job With DataSet Stage

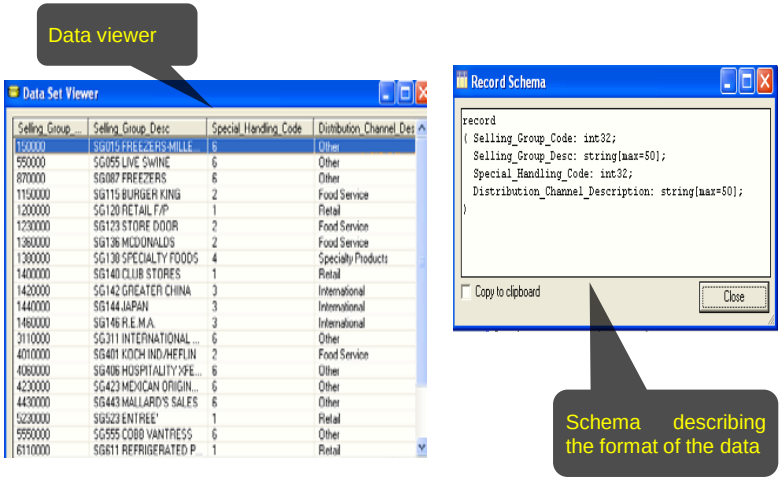


Data Set Management Utility

- The window is available (data sets management) from Designer and Director.
- In Designer, click Tools>Data Set Management to open this window.



Data and Schema Displayed



File Set Stage

- Use to read and write to filesets
- Files suffixed by .fs
- Files are similar to a dataset
  - Partitioned
  - Implemented with header file and data files
- How filesets differ from datasets
  - Data files are text files
    - Hence readable by external applications
  - Datasets have a proprietary data format which may change in future DataStage versions
- Number of raw data files depends on: the configuration file.

Q&A

- List three types of file data?
- What makes datasets perform better than other types of files in parallel jobs?
- What is the difference between a data set and a file set? ?

Q&A

- Sequential, dataset, complex flat files
- They are partitioned and they store data in the native parallel format
- Both are partitioned. Data sets store data in a binary format not readable by user applications. File sets are readable.

Dataset Stage summary

- Having completed this unit, you should be able to:
- Understand the stages for accessing different kinds of file data
- Sequential File stage
- Data Set stage
- Create jobs that read from and write to sequential files
- Create Reject links
- Work with NULLs in sequential files
- Read from multiple files using file patterns
- Use multiple readers

Combining Data

Unit objectives

- After completing this unit, you should be able to:
- Combine data using the Lookup stage
- Combine data using Merge stage
- Combine data using the Join stage
- Combine data using the Funnel stage

Combining Data

Ways to combine data:

- Horizontally:
  - Multiple input links
  - One output link made of columns from different input links.
  - Joins
  - Lookup
  - Merge
- Vertically:
  - One input link, one output link combining groups of related records into a single record
  - Aggregator
  - Remove Duplicates
- Funneling: Multiple input streams funneled into a single output stream
  - Funnel stage

Lookup, Merge, Join Stages

- These stages combine two or more input links
  - Data is combined by designated "key" column(s)
- These stages differ mainly in:
  - Memory usage
  - Treatment of rows with unmatched key values
  - Input requirements (sorted, de-duplicated)

Not all Links are Created Equal

- DataStage distinguishes between:
  - - The Primary input: (Framework port 0)
  - - Secondary inputs: in some cases "Reference" (other Framework ports)
- Conventions:

	Joins	Lookup	Merge
Primary Input: port 0	Left	Source	Master
Secondary Input(s): ports 1,…	Right	Lookup table(s)	Update(s)

- Tip: Check "Link Ordering" tab to make sure intended Primary is listed first

Lookup Stage

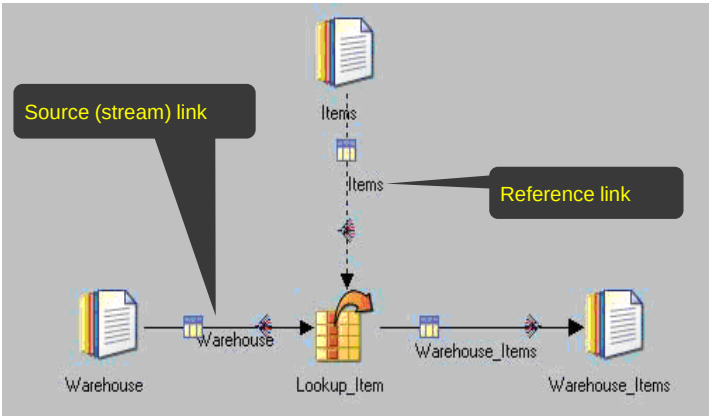
Lookup Features

- One stream input link (source link)
- Multiple reference links
- One output link
- Lookup failure options
  - –Continue, Drop, Fail, Reject
- Reject link
  - Only available with Reject lookup failure option
- Can return multiple matching rows
- Hash file is built in memory from the lookup files
  - Indexed by key
  - Should be small enough to fit into physical memory

Lookup Types

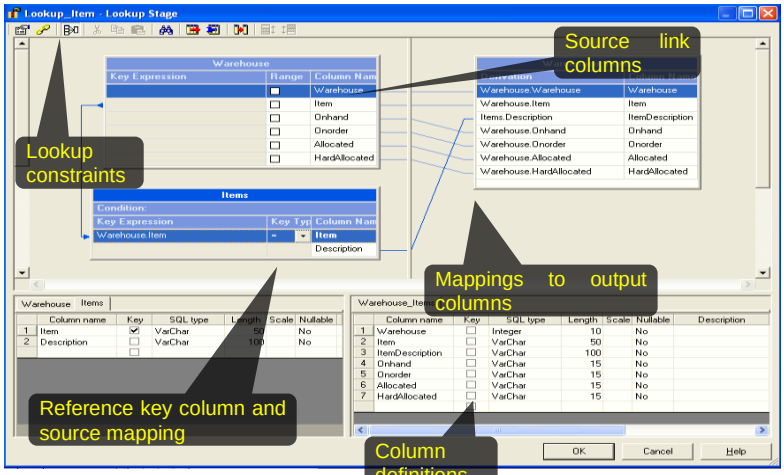
- Equality match
  - Match exactly values in the lookup key column of the reference link to selected values in the source row
  - Return row or rows (if multiple matches are to be returned) that match
- Caseless match
  - Like an equality match except that it's caseless
    - E.g., "abc" matches "AbC"
- Range on the reference link
  - Two columns on the reference link define the range
  - A match occurs when a selected value in the source row is within the range
- Range on the source link
  - Two columns on the source link define the range
  - A match occurs when a selected value in the reference link is within the range

Lookup Example



Capgemini Internal

Lookup Stage With an Equality Match

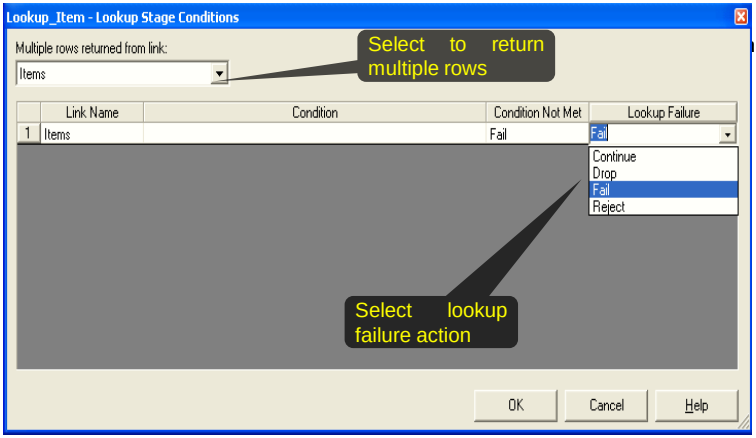


Lookup Failure Actions

If the lookup fails to find a matching key column, one of several actions can be taken:

- Fail (Default)
  - Stage reports an error and the job fails immediately
- Drop
  - Input row is dropped
- Continue
  - Input row is transferred to the output. Reference link columns are filled with null or default values
- Reject
  - Input row sent to a reject link
  - This requires that a reject link has been created for the stage

Specifying Lookup Failure Actions



Capgemini Internal

Lookup Stage Behavior

- We shall first use a simplest case, optimal input:
  - Two input links: "Source" as primary, "Look up" as secondary
  - sorted on key column (here "Citizen"),
- without duplicates on key

Source link (primary input)

Revolution	Citizen
1789	Lefty
1776	M_B_Dextrous

Lookup link (secondary input)

Citizen	Exchange
M_B_Dextrous	Nasdaq
Righty	NYSE

Lookup key column

Click the Constraints icon (top, second from left) to open this window.

Lookup Stage

- Output of Lookup with Continue option

Revolution	Citizen	Exchange
1789	Lefty	
1776	M_B_Dextrous	Nasdaq

- Output of Lookup with Drop option

Revolution	Citizen	Exchange
1776	M_B_Dextrous	Nasdaq

For the first source row (1789), the lookup fails to find a match.
Since Continue is the lookup failure option, the row is output.
The Exchange column is populated with NULL (if the column is nullable) or a default value, empty string (if the column is not nullable).
For the second source row (1776), the lookup finds a match, so the Exchange column gets a value from the lookup file.
When Drop is the lookup failure action, the first, unmatched row is dropped.

The Lookup Stage

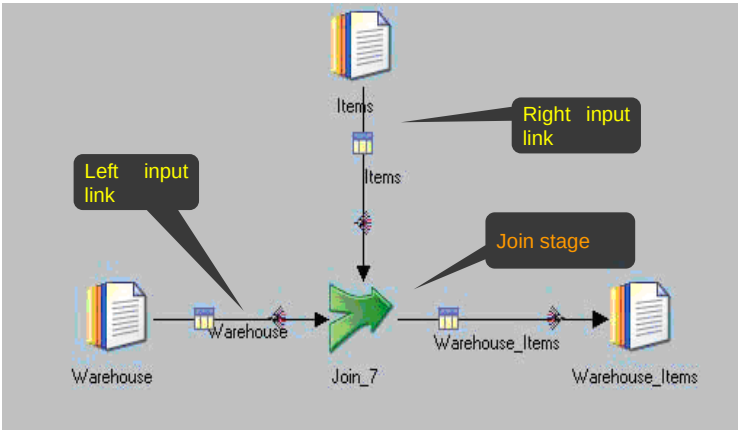
- Lookup Tables should be small enough to fit into physical memory
- On a MPP you should partition the lookup tables using entire partitioning method or partition them by the same hash key as the source link
  - Entire results in multiple copies (one for each partition)
- On a SMP, choose entire or accept the default (which is entire)
  - Entire does not result in multiple copies because memory is shared

Join Stage

Join Stage

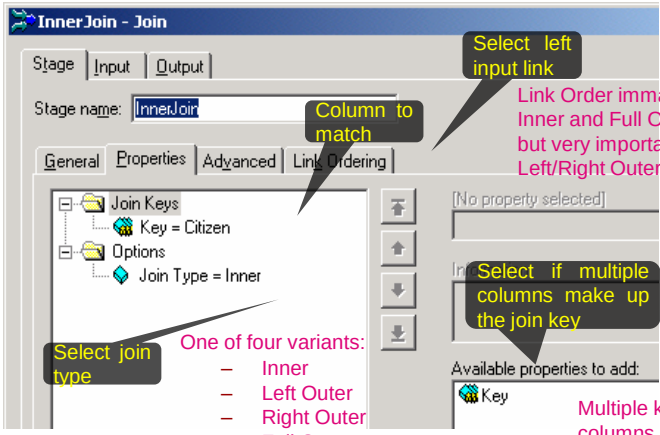
- Four types:
  - Inner
  - Left outer
  - Right outer
  - Full outer
- 2 or more *sorted* input links, 1 output link
  - "left" on primary input, "right" on secondary input
  - Pre-sort make joins "lightweight": few rows need to be in RAM
- Follow the RDBMS-style relational model
  - Cross-products in case of duplicates
  - Matching entries are reusable for multiple matches
  - Non-matching entries can be captured (Left, Right, Full)
- No fail/reject option for missed matches

Job With Join Stage



Capgemini Internal

Join Stage Editor



Join Stage Behavior

- We shall first use a simplest case, optimal input:
  - two input links: "left" as primary, "right" as secondary
  - sorted on key column (here "Citizen"),
  - **without** duplicates on key

Left link (primary input)

Revolution	Citizen
1789	Lefty
1776	M_B_Dextrous

Right link (secondary input)

Citizen	Exchange
M_B_Dextrous	Nasdaq
Righty	NYSE

Join key column

Inner Join

- Transfers rows from both data sets whose key columns have matching values
- Treats both inputs symmetrically

Output of inner join on key Citizen

Revolution	Citizen	Exchange
1776	M_B_Dextrous	Nasdaq

Left Outer Join

- Transfers all values from the left link and transfers values from the right link only where key columns match

Revolution	**(Left Record) Citizen Exchange**	
1789	Lefty	
1776	M_B_Dextrous	Nasdaq

Right Outer Join

- Transfers all values from the right link and transfers values from the left link only where key columns match.

Revolution	**(Right record) Citizen**	**Exchange**
1776	M_B_Dextrous	Nasdaq
	Righty	NYSE

Null or default value

Full Outer Join

- Transfers rows from both data sets, whose key columns contain equal values, to the output link.
- It *also* transfers rows, whose key columns contain unequal values, from both input links to the output link.
- Treats both input symmetrically.
- Creates new columns, with new column names!

Revolution	**(left record) Citizen Exchange**	**(right record) Citizen**	
1789	Lefty		
1776	M_B_Dextrous	M_B_Dextrous	Nasdaq
		Righty	NYSE

Merge Stage

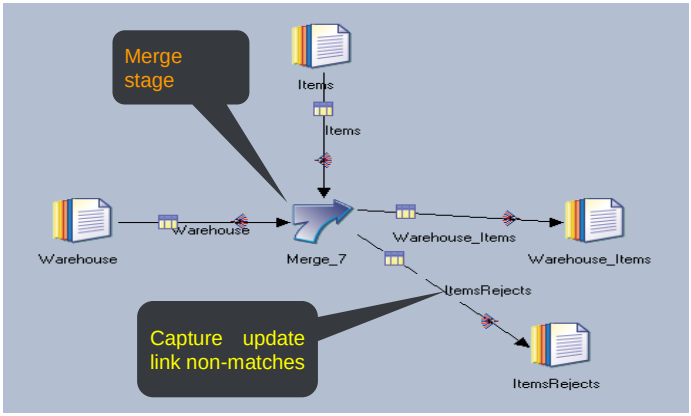
Merge Stage

- Similar to Join stage
- Input links must be sorted
  - Master link and one or more secondary links
  - Master must be duplicate-free
- Light-weight
  - Little memory required, because of the sort requirement
- Unmatched master rows can be kept or dropped
- Unmatched secondary links can be captured in a reject link

More on Merge Stage

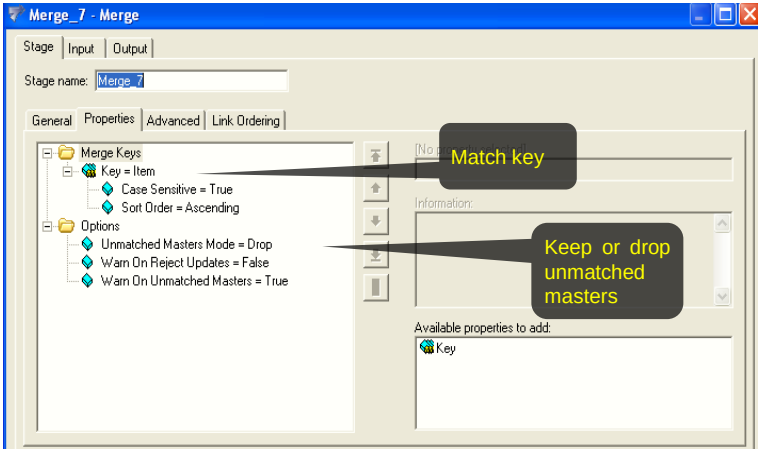
- Combines
  - one sorted, duplicate-free *master* (primary) link with
  - one or more sorted *update* (secondary) links.
  - Pre-sort makes merge "lightweight" for memory
  - Follows the Master-Update model:
  - Master row and one or more updates row are merged iff they have the same value in user-specified *key* column(s).
  - Unmatched ("Bad") master rows can be either
    - kept ([`-keepBadMasters`])
    - dropped (`-dropBadMasters`)
  - Unmatched ("Bad") update rows in input link *n* can be captured in a "reject" link in corresponding output link *n.*
  - Matched update rows are consumed

Merge Stage Job



Capgemini Internal

Merge Stage Properties



Unmatched updates are captured by adding additional reject links (one for each update link).
Unmatched Master rows
One of two options:

Keep [default]
Drop

(Capture in reject link is NOT an option)
Unmatched Update rows option:
Capture in reject link(s). Implemented by adding outgoing links

Merge Options

keepBadMaster

Revolution	Citizen	Exchange
1789	Lefty	
1776	M_B_Dextrous	Nasdaq

Same output as left outer join *and* lookup/continue

dropBadMaster

Revolution	Citizen	Exchange
1776	M_B_Dextrous	Nasdaq

Both options yield the same "reject" link of "bad" (unused) updates

Citizen	Exchange
Righty	NYSE

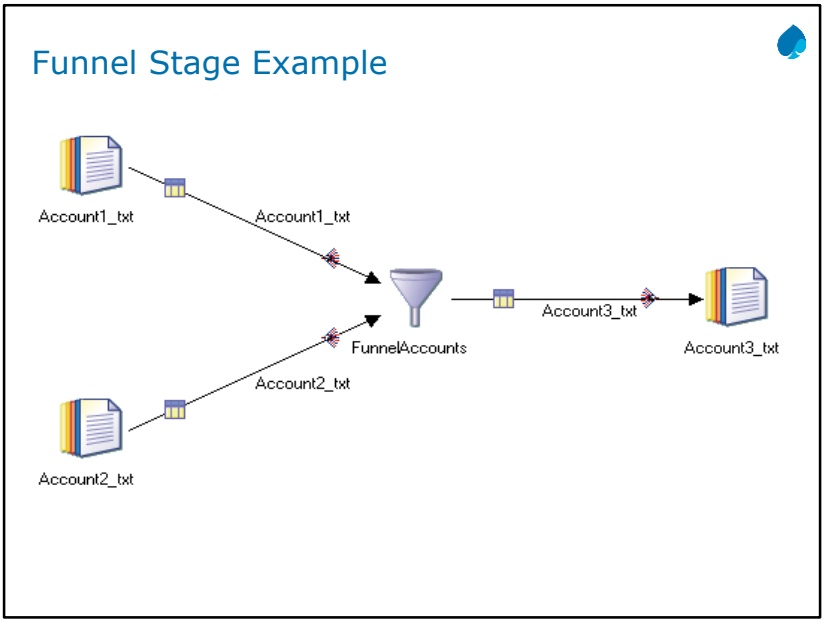
Comparison: Joins, Lookup, Merge

	Joins	Lookup	Merge
Model	RDBMS-style relational	Source - in RAM LU Table	Master -Update(s)
Memory usage	light	heavy	light
# and names of Inputs	2 or more: left, right	1 Source, *N* LU Tables	1 Master, *N* Update(s)
Mandatory Input Sort	all inputs	no	all inputs
Duplicates in primary input	OK (x-product)	OK	Warning!
Duplicates in secondary input(s)	OK (x-product)	Warning!	OK only when *N* = 1
Options on unmatched primary	Keep (left outer), Drop (Inner)	[fail] | continue | drop | reject	[keep] | drop
Options on unmatched secondary	Keep (right outer), Drop (Inner)	*NONE*	capture in reject set(s)
On match, secondary entries are	captured	captured	consumed
# Outputs	1	1 out, (1 reject)	1 out, (*N* rejects)
Captured in reject set(s)	*Nothing (N/A)*	unmatched primary entries	unmatched secondary entries

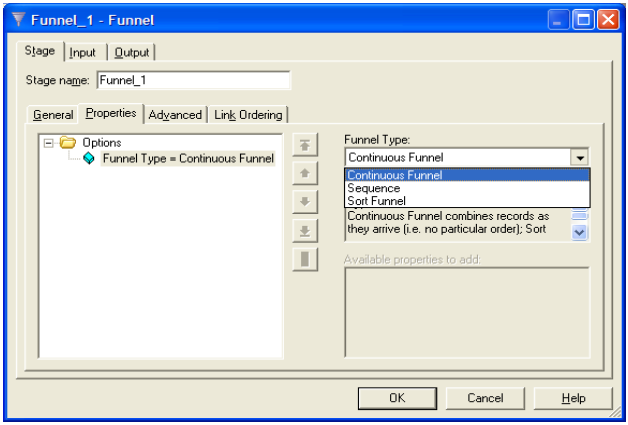
Funnel Stage

What is a Funnel Stage?

- Combines data from multiple input links to a single output link
- All sources must have identical meta data
- Three modes
  - Continuous
    Records are combined in no particular order
    First available record
    It takes one record from each input link in turn. If data is not available on an input link, the stage skips to the next link rather than waiting.
    Does not attempt to impose any order on the data it is processing Sort Funnel
    Combines the input records in the order defined by a key
    Produces sorted output if the input links are all sorted by the same key
  - Sequence:
    Outputs all records from the first input link, then all from the second input link, and so on


Funnel Stage Example
Account1_txt
Account1_txt
Account2_txt
Account2_txt
FunnelAccounts
Account3_txt
Account3_txt

Funnel Stage Properties



Q&A

- Name three stages that horizontally join data?
- Which stage uses the least amount of memory? Join or Lookup?
- Which stage requires that the input data is sorted? Join or Lookup?

Q&A

- Lookup, Merge, Join
- Lookup
- Join

Unit summary

- Having completed this unit, you should be able to:
- Combine data using the Lookup stage
- Define range lookups
- Combine data using Merge stage
- Combine data using the Join stage
- Combine data using the Funnel stage

Sorting and Aggregating Data

Unit objectives

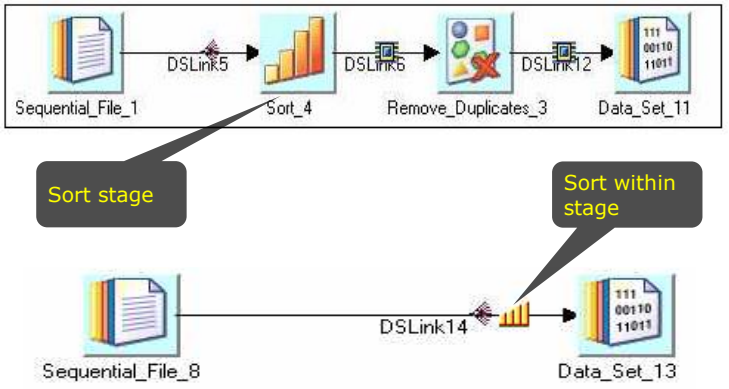
- After completing this unit, you should be able to:
  - Sort data using in-stage sorts and Sort stage
  - Combine data using Aggregator stage
  - Combine data Remove Duplicates stage

Sort Stage

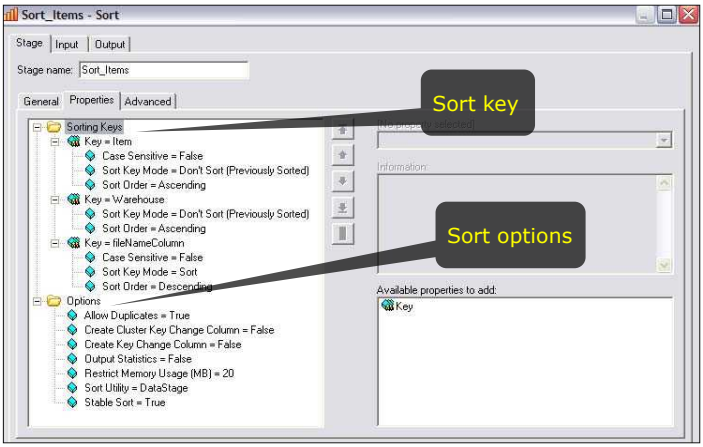
Sorting Data

- Uses
  - Some stages require sorted input
    - Join, merge stages require sorted input
  - Some stages use less memory with sorted input
    - E.g., Aggregator
- Sorts can be done:
  - Within stages
    - On input link Partitioning tab, set partitioning to anything other than Auto
  - In a separate Sort stage
    - Makes sort more visible on diagram
    - Has more options

Sorting Alternatives



Sort Stage


Sort_Items - Sort
Stage
Input
Output
Stage name:
Sort_Items
General
Properties
Advanced
Sort key
Sort options
Sorting Keys
Key = Item
Case Sensitive = False
Sort Key Mode = Don't Sort (Previously Sorted)
Sort Order = Ascending
Key = Warehouse
Sort Key Mode = Don't Sort (Previously Sorted)
Sort Order = Ascending
Key = fileNameColumn
Case Sensitive = False
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
Restrict Memory Usage (MB) = 20
Sort Utility = DataStage
Stable Sort = True
Information:
Available properties to add:
Key

Sort keys

- Add one or more keys
- Specify sort mode for each key
  - Sort: Sort by this key
  - Don't sort (previously sorted):
    - Assumes the data has already been sorted on this key
    - Continue sorting by any secondary keys
- Specify sort order: ascending / descending
- Specify case sensitive or not

Sort Options

- Sort Utility
  - DataStage – the default
  - Unix: Don't use. Slower than DataStage sort utility
- Stable
- Allow duplicates
- Memory usage
  - Sorting takes advantage of the available memory for increased performance
    - Uses disk if necessary
  - Increasing amount of memory can improve performance
- Create key change column
  - Add a column with a value of 1 / 0
  - 1 indicates that the key value has changed
  - 0 means that the key value hasn't changed
  - Useful for processing groups of rows in a Transformer

Partitioning V. Sorting Keys

- Partitioning keys are often different than Sorting keys
  - Keyed partitioning (e.g., Hash) is used to group related records into the same partition
  - Sort keys are used to establish order within each partition
- Example:
- Partition on HouseHoldID, sort on HouseHoldID, EntryDate
  - Partitioning on HouseHoldID ensures that the same ID will not be spread across multiple partitions
  - Sorting orders the records with the same ID by entry date
    - Useful for deciding which of a group of duplicate records with the same ID should be retained

Aggregator Stage

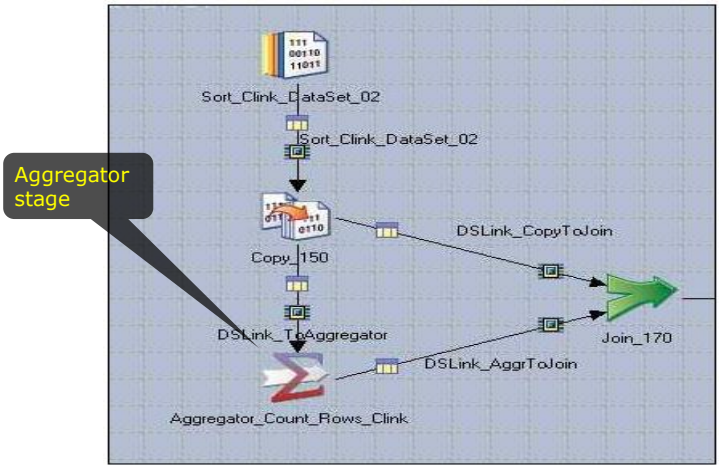
Aggregator Stage

Purpose: Perform data aggregations

Specify:

- One or more key columns that define the aggregation units (or groups)
- Columns to be aggregated
- Aggregation functions include, among many others:
  - count (nulls/non-nulls)
  - Sum
  - Max / Min
- The grouping method (hash table or pre-sort) is a
- performance issue

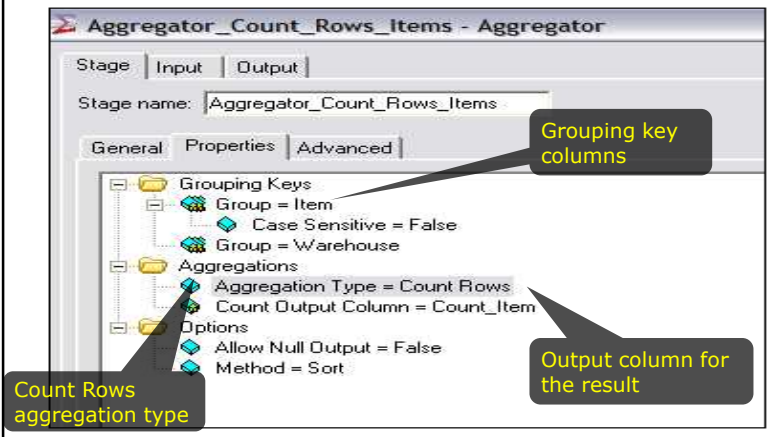
Job with Aggregator Stage



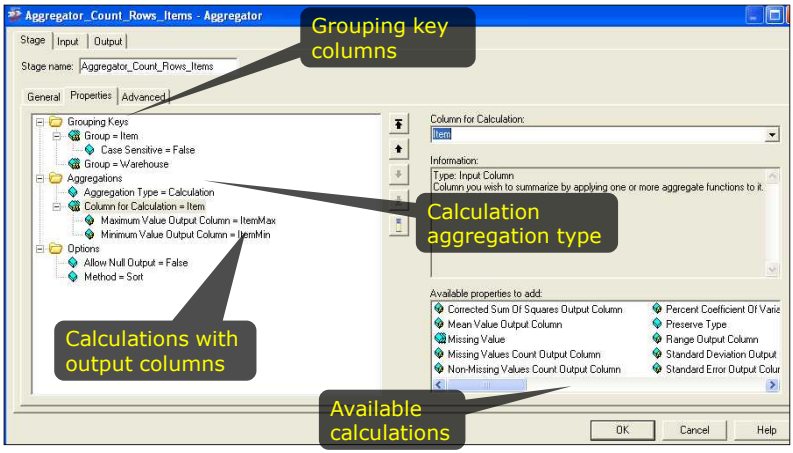
Aggregation Types

- Count rows
  - Count rows in each group
  - Put result in a specified output column
- Calculation
  - Select column
  - Put result of calculation in a specified output column
  - Calculations include:
    - Sum
    - Count
    - Min, max
    - Mean
    - Missing value count
    - Non-missing value count
    - Percent coefficient of variation

Count Rows Aggregator Properties



Calculation Type Aggregator Properties



Grouping Methods

- Hash (default)
  - Calculations are made for all groups and stored in memory
    - Hash table structure (hence the name)
  - Results are written out after all input has been processed
  - Input does not need to be sorted
  - Useful when the number of unique groups is small
    - Running tally for each group's aggregations needs to fit into memory
- Sort
  - Requires the input data to be sorted by grouping keys
    - Does not perform the sort! Expects the sort
  - Only a single aggregation group is kept in memory
    - When a new group is seen, the current group is written out
  - Can handle unlimited numbers of groups

Removing Duplicates

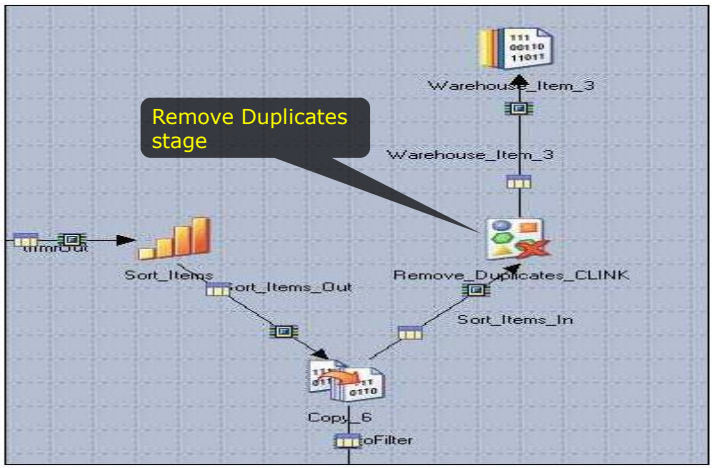
Removing Duplicates

- Can be done by Sort stage
  - Use unique option
    - No choice on which duplicate to keep
    - Stable sort always retains the first row in the group
    - Non-stable sort is indeterminate

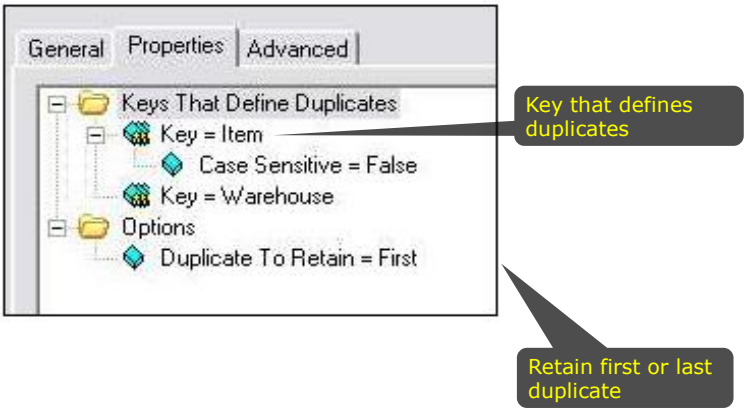
OR

- Remove Duplicates stage
  - Has more sophisticated ways to remove duplicates
    - Can choose to retain first or last

Remove Duplicates Stage Job



Remove Duplicates Stage Properties



Q&A

- What stage is used to perform calculations of column values grouped in specified ways?
- In what two ways can sorts be performed?
- What is a stable sort?

Q&A

- Aggregator stage
- Using the Sort stage. In-stage sorts.
- Stable sort preserves the order of non-key values.

Unit summary

- Having completed this unit, you should be able to:
  - Sort data using in-stage sorts and Sort stage
  - Combine data using Aggregator stage
  - Combine data Remove Duplicates stage

Transforming Data

Unit objectives

After completing this unit, you should be able to:

- Use the Transformer stage in parallel jobs
- Define constraints
- Define derivations
- Use stage variables
- Create a parameter set and use its parameters in constraints and derivations

Transformer Stage

Transformer Stage

- BASIC Transformer stage
  - The BASIC Transformer stage is a processing stage.
  - It appears under the processing category in the tool palette in the Transformer shortcut container.
  - The BASIC Transformer stage is similar in appearance and function to the Transformer stage described in Transformer stage. It gives access to BASIC transforms and functions (BASIC is the language supported by the server engine and available in server jobs).
  - You can only use BASIC transformer stages on SMP systems (not on MPP or cluster systems).
  - BASIC Transformer stages can have a single input and any number of outputs

In this module, we'll discuss two stages that can be used to perform derivations to implement business logic. Our main focus in on the Transformer stage. We'll also briefly discuss the Modify stage.

Transformer Stage

- Transformer stage
  - The Transformer stage is a processing stage.
  - It appears under the processing category in the tool palette.
  - Transformer stages allow you to create transformations to apply to your data.
  - These transformations can be simple or complex and can be applied to individual columns in your data.
  - Transformations are specified using a set of functions.
  - Transformer stages can have a single input and any number of outputs. It can also have a reject link that takes any rows which have not been written to any of the outputs links by reason of a write failure or expression evaluation failure.

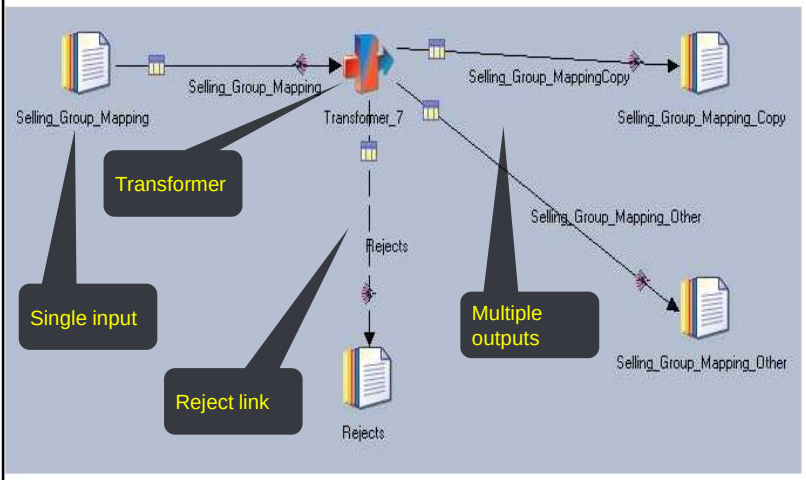
In this module, we'll discuss two stages that can be used to perform derivations to implement business logic. Our main focus in on the Transformer stage. We'll also briefly discuss the Modify stage.

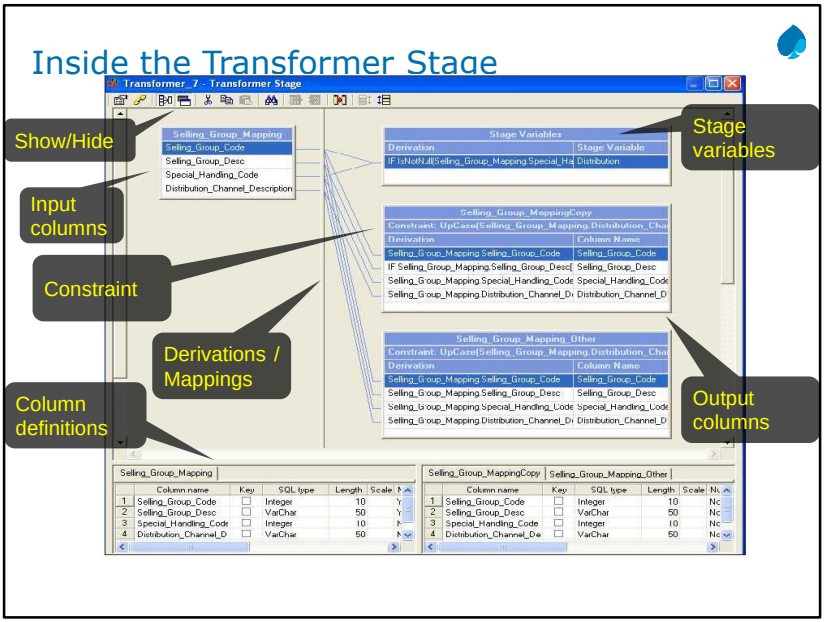
Transformer Stage

- Column mappings
- Derivations
  - Written in Basic
  - Final compiled code is C++ generated object code
- Constraints
  - Filter data
  - Direct data down different output links
    - For different processing or storage
- Expressions for constraints and derivations can reference
  - Input columns
  - Job parameters
  - Functions
  - System variables and constants
  - Stage variables
  - External routines

In this module, we'll discuss two stages that can be used to perform derivations to implement business logic. Our main focus in on the Transformer stage. We'll also briefly discuss the Modify stage.

Job With a Transformer Stage

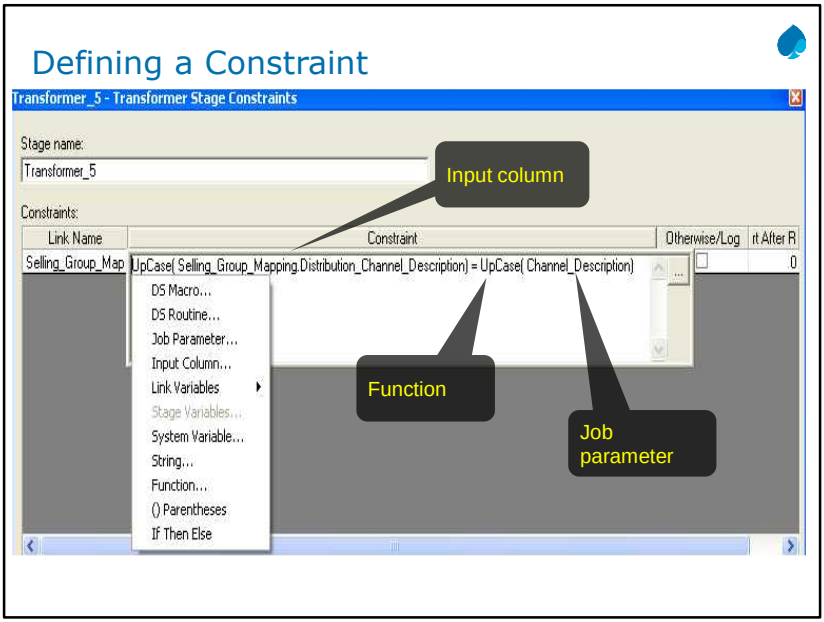




Link naming conventions are important because they identify appropriate links in the stage properties screen shown above.

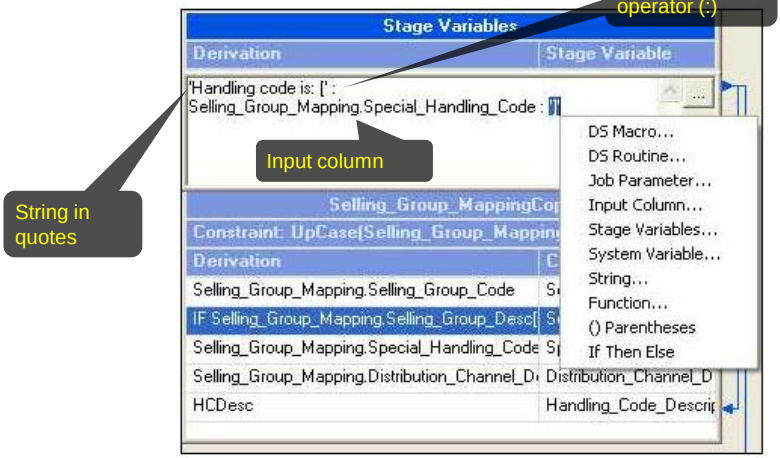
Four quadrants:
Incoming data link (one only)
Outgoing links (can have multiple)
Meta data for all incoming links
Meta data for all outgoing links – may have multiple tabs if there are multiple outgoing links

Note the constraints bar – if you double-click on any you will get screen for defining constraints for all outgoing links.
Double-click to the left of an output column or stage variable to define its derivation.



Click the Constraints icon at the top of the Transformer (yellow chain) to open the Constraints window.
Select items from the menu or type in items to build the expression.

Defining a Derivation



IF THEN ELSE Derivation

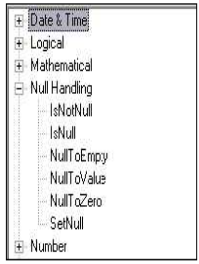
- Use IF THEN ELSE to conditionally derive a value
- Format:
  - IF <condition> THEN <expression1> ELSE <expression1>
  - If the condition evaluates to true then the result of expression1 will be copied to the target column or stage variable
  - If the condition evaluates to false then the result of expression2 will be copied to the target column or stage variable
- Example:
  - Suppose the source column is named In.OrderID and the target column is named Out.OrderID
  - Replace In.OrderID values of 3000 by 4000
  - IF In.OrderID = 3000 THEN 4000 ELSE Out.OrderID

String Functions and Operators

- Substring operator
  - Format: "String" [loc, length]
  - Example:
    - Suppose In.Description contains the string "Orange Juice"
    - InDescription[8,5] "Juice"
- UpCase(<string>) / DownCase(<string>)
  - Example: UpCase(In.Description) "ORANGE JUICE"
- Len(<string>)
  - Example: Len(In.Description) 12

Checking for NULLs

- Nulls can be introduced into the data flow from lookups
  - Mismatches (lookup failures) can produce nulls
- Can be handled in constraints, derivations, stage variables, or a combination of these
- NULL functions
  - Testing for NULL
    - IsNull(<column>)
    - IsNotNull(<column>)
  - Replace NULL with a value
    - NullToValue(<column>, <value>)
- Set to NULL: SetNull()
  - Example: IF In.Col = 5 THEN SetNull() ELSE In.Col



If you perform a lookup from a lookup stage and choose the continue option for a failed lookup, you have the possibility of nulls entering your data flow.

Important: Be sure to use the NULL functions in stage variable derivations rather than output column derivations. Reference the stage variable in column derivations rather than performing the NULL function in the column derivation. The reason for this is that records containing NULLs are rejected BEFORE column derivations are performed (but not before stage variable derivations are performed).

Transformer Functions

- Date & Time
- Logical
- Null Handling
- Number
- String
- Type Conversion

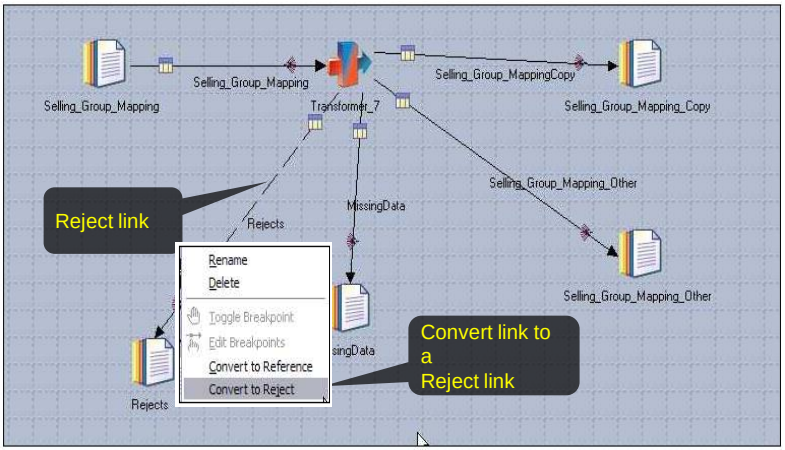
Transformer Execution Order

- Derivations in stage variables
- Constraints are executed before derivations
- Column derivations in earlier links are executed before later links
- Derivations in higher columns are executed before lower columns

Transformer Stage Variables

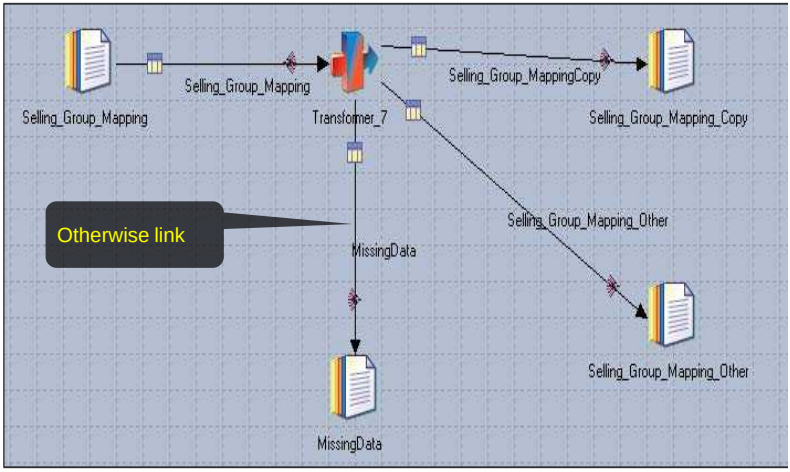
- Derivations execute in order from top to bottom
  - Later stage variables can reference earlier stage variables
  - Earlier stage variables can reference later stage variables
    - These variables will contain a value derived from the previous row that came into the Transformer
- Multi-purpose
  - Counters
  - Store values from previously read rows to make comparisons with the currently read row
  - Store derived values to be used in multiple target field derivations
  - Can be used to control execution of constraints

Transformer Reject Links



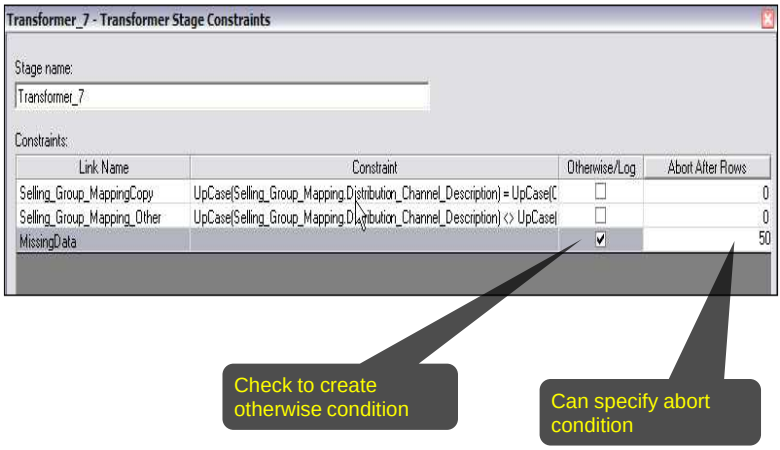
Rows are sent to the Reject link if a nullable column is used in a calculation or derivation and that column has a NULL value.
It is important to test for and replace NULLs in any derivations that reference nullable input columns.
Position these tests for and replaces of NULLs within stage variable derivations.

Otherwise Link



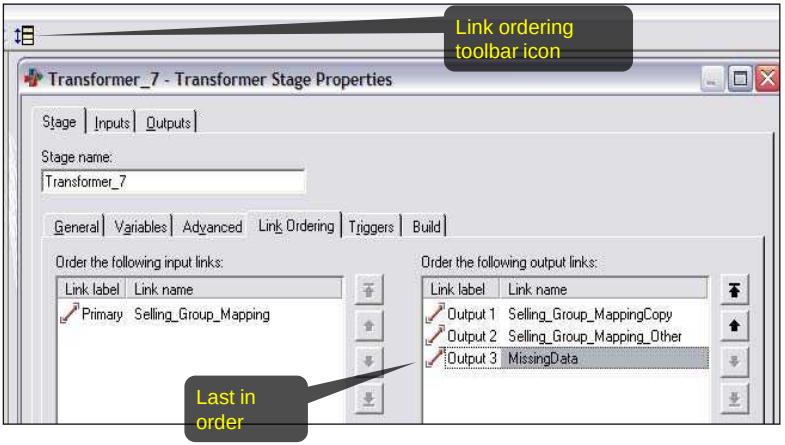
The otherwise link captures what doesn't go down any previous links.
It needs to come after the other links in the link ordering.
It's defined on the Constraints tab, as shown in the next slide.

Defining an Otherwise Link



The Otherwise link must come after the other links carrying the main constraints. You can change the ordering of links, as shown on the next slide. A warning is put in the log if any rows go down the otherwise link. You can specify an abort condition for any output link. The abort occurs after the specified number of rows occurs in one of the partitions. It is not based on the total number of rows, but on the number of rows in a single partition.

Specifying Link Ordering



The Link Ordering icon is at the last icon on the toolbar.

Q & A

1. What occurs first? Derivations or constraints?
2. Can stage variables be referenced in constraints?
3. Where should you test for NULLS within a Transformer? Stage Variable derivations or output column derivations?

Q & A

1. Constraints
2. Yes
3. Stage variable derivations. Reference the stage variables in the output column derivations.

Unit summary

Having completed this unit, you should be able to:

- Use the Transformer stage in parallel jobs
- Define constraints
- Define derivations
- Use stage variables
- Create a parameter set and use its parameters in constraints and derivations

Connectors Stages

Connector Stage Types

- ODBC
  - Conforms to ODBC 3.5 standard and is Level 3 compliant
  - Certified with Oracle, DB2 UDB, SQL Server, and others
  - DataDirect drivers
- DB2 UDB
  - 8.1 and 8.2
- WebSphere MQ
  - WSMQ 5.3 and 6.0 for Client / Sever
  - WSMB 5.0
- Teradata

Connector Stage Features

- Common stage editor
- Convenient drop-down lists to choose properties
- Job parameters can be inserted into any property
- Required properties are identified with a visual indicator
- Warning indicator for properties requiring attention
- Metadata retrieval
- Integrated SQL Builder

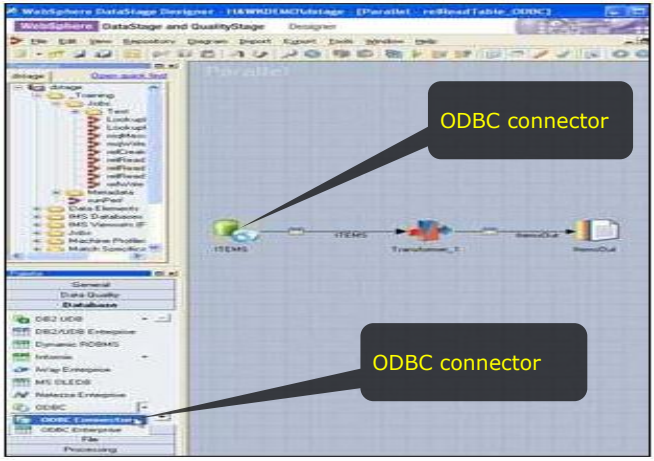
Connector Stage Features

- Parallel support
  - Read: parallel connections to the server and modified SQL queries for each connection
  - Write: parallel connections to the server
- Transaction isolation level support
  - Read Uncommitted
  - Read Committed
  - Repeatable Read
  - Serializable
- Before / After commands
  - Executed once before or after the job runs

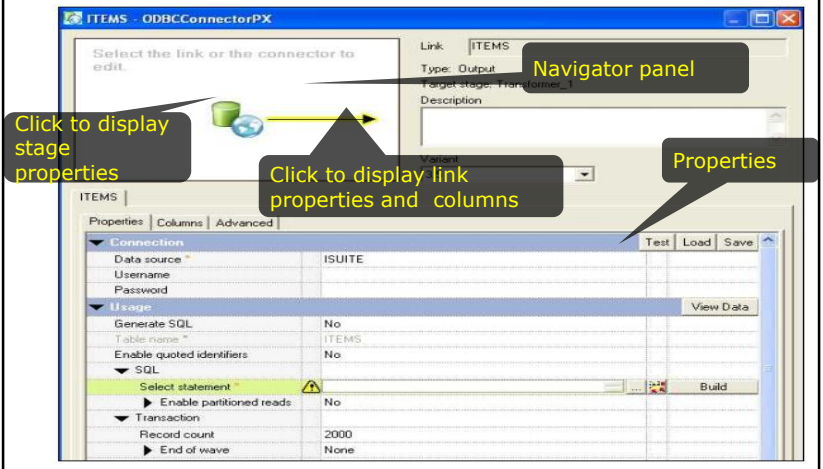
Metadata Retrieval

- List data sources
  - Can list both user and system DSNs
- List users defined for the given database
- Individual table or view metadata
  - Column data type, size, precision, etc.
- Database information
  - Name, vendor name, version number, etc.
- List of database schemas
- List tables that match the filter criteria
- List related tables
  - Those related by foreign key
  - Those whose foreign keys point to the primary key in the given table

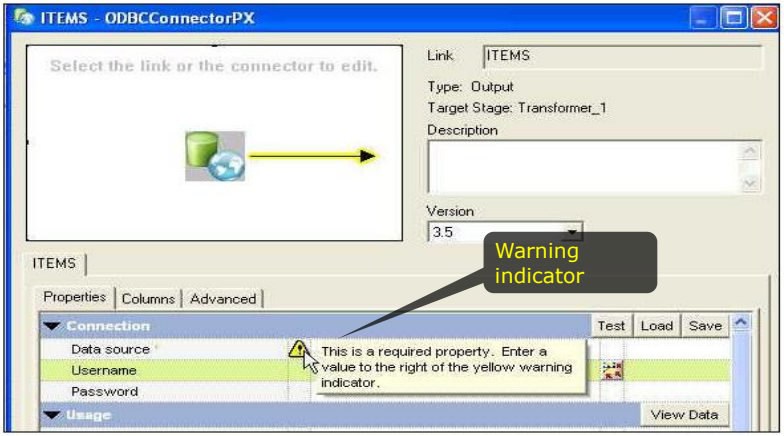
Connector Stages



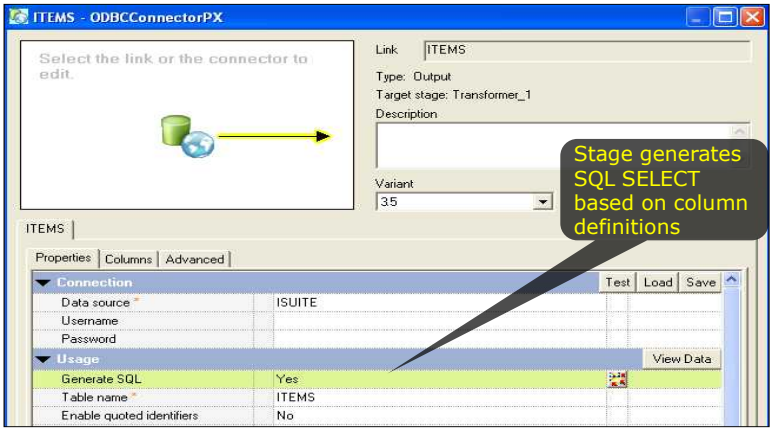
Stage Editor



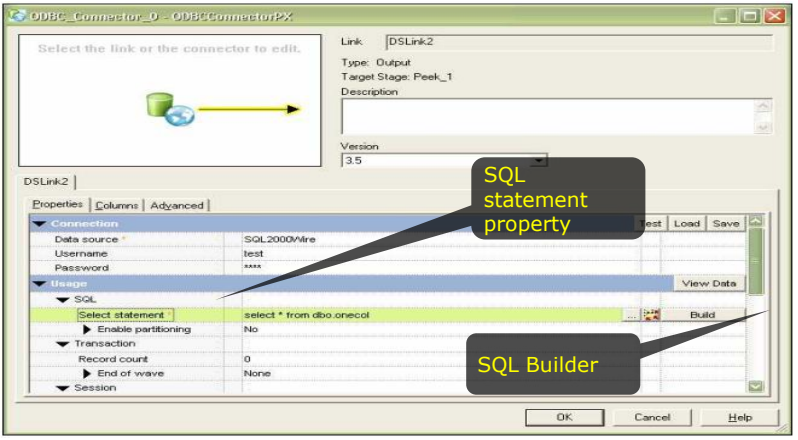
Property Warning Indicators



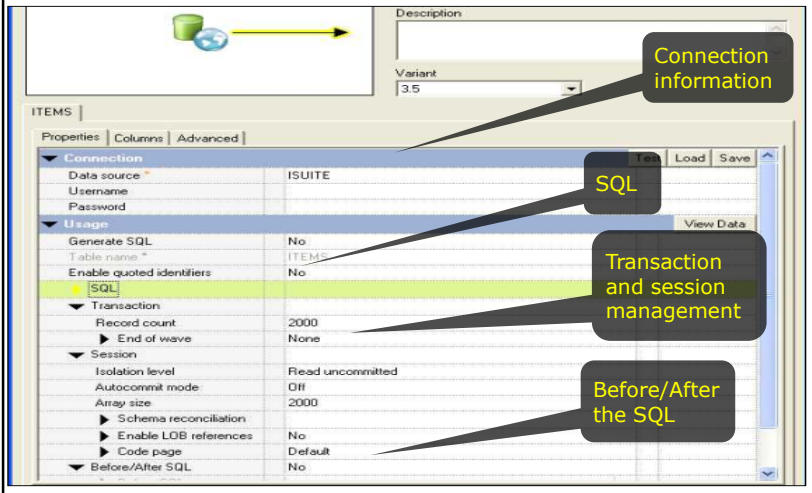
Auto-Generated SQL



Invoking SQL Builder

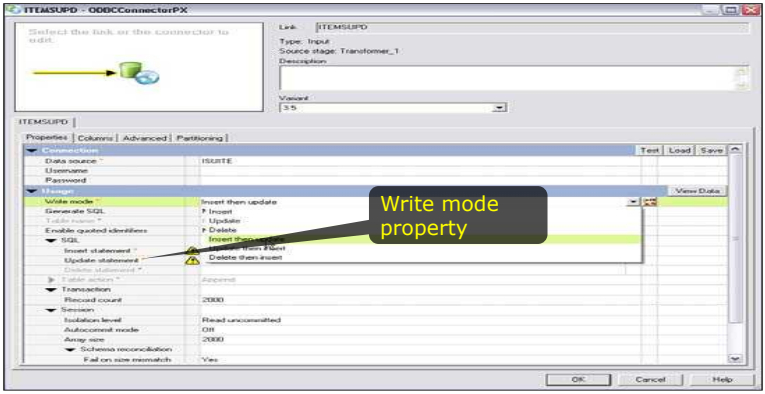


Connector Stage Properties

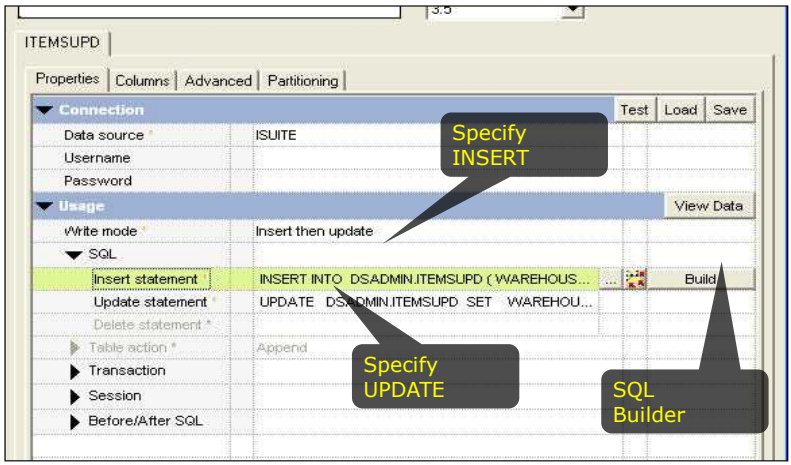


Building SQL to Write to a Table

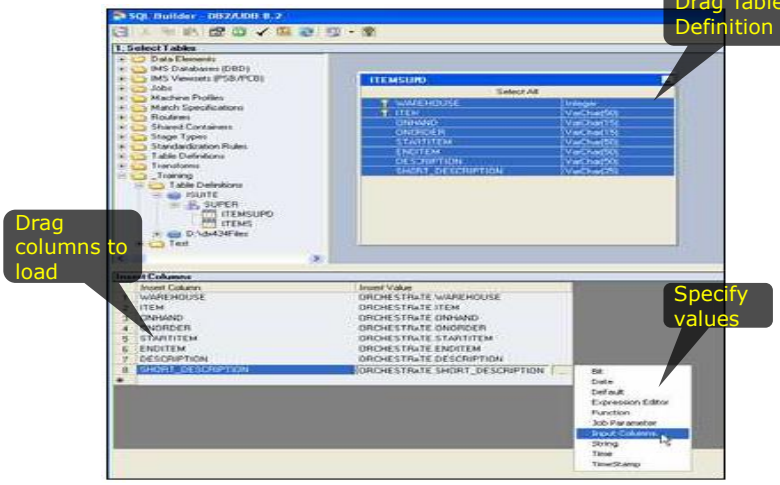
Select Type of Write



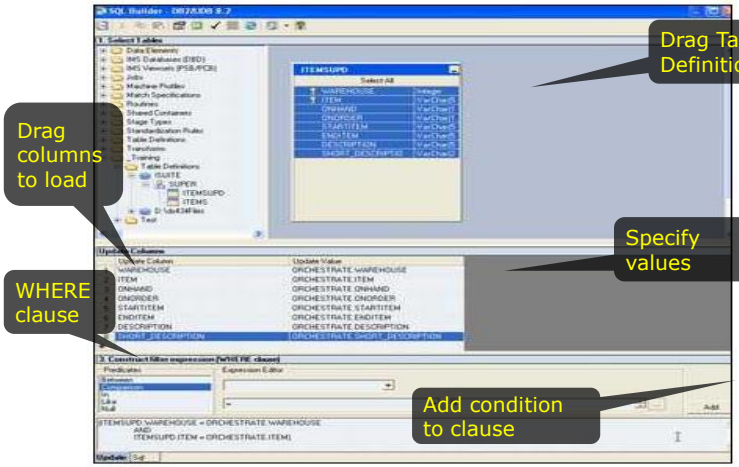
Invoking SQL Builder



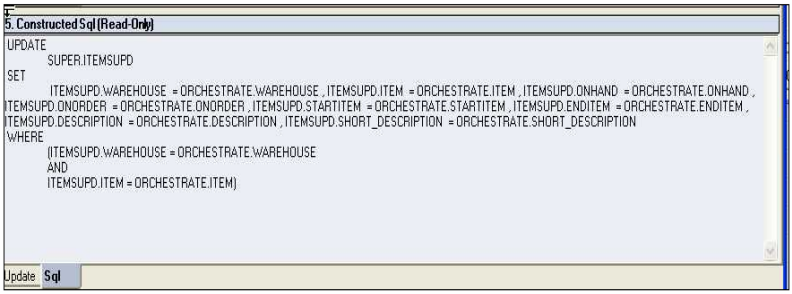
SQL Builder INSERT



SQL Builder Update



Viewing the SQL



Change Capture Stage

Change Capture Stage

- The Change Capture Stage is a processing stage.
- The stage compares two data sets and makes a record of the differences.
- The Change Capture stage takes two input data sets, denoted before and after, and outputs a single data set whose records represent the changes made to the before data set to obtain the after data set.
- The stage produces a change data set, whose table definition is transferred from the after data set's table definition with the addition of one column: a change code with values encoding the four actions: insert, delete, copy, and edit.
- The preserve-partitioning flag is set on the change data set.

Change Capture Stage

- The compare is based on a set a set of key columns, rows from the two data sets are assumed to be copies of one another if they have the same values in these key columns.
- Change Capture Stage is used to capture the changes between two sources based on the change code.

  Change Code is the Code generated when we capture the changes using Change Capture Stage.
- There are four change codes available:

0,1,2,3	
0 -	Copy
1 -	Insert
2 -	Delete
3 - Edit	