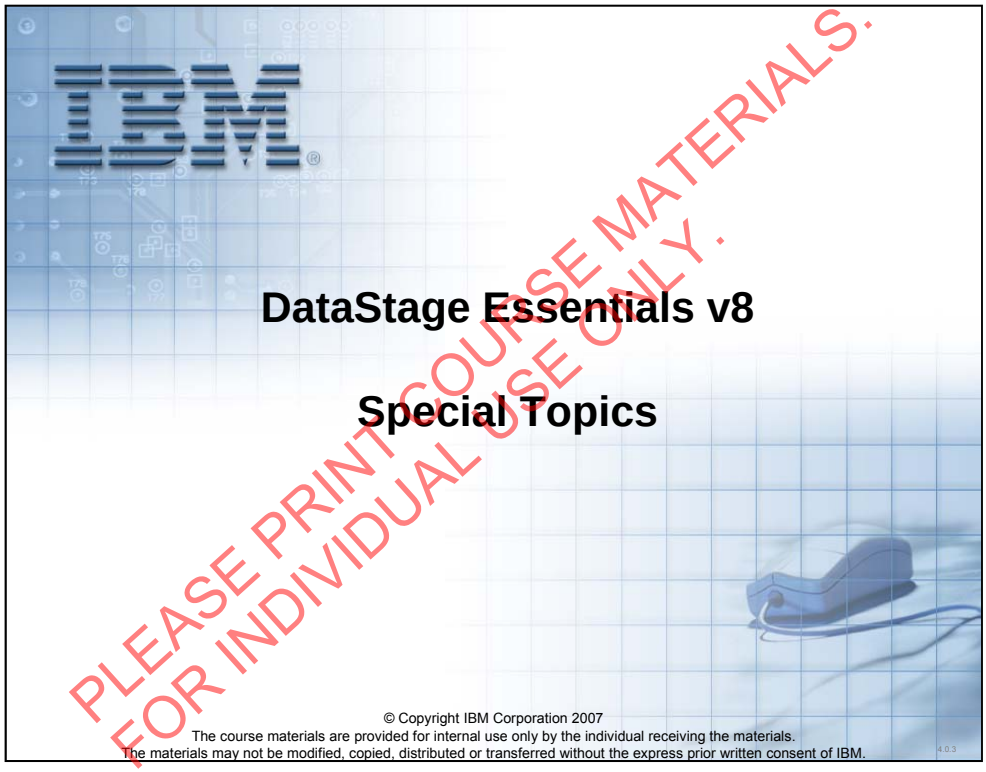
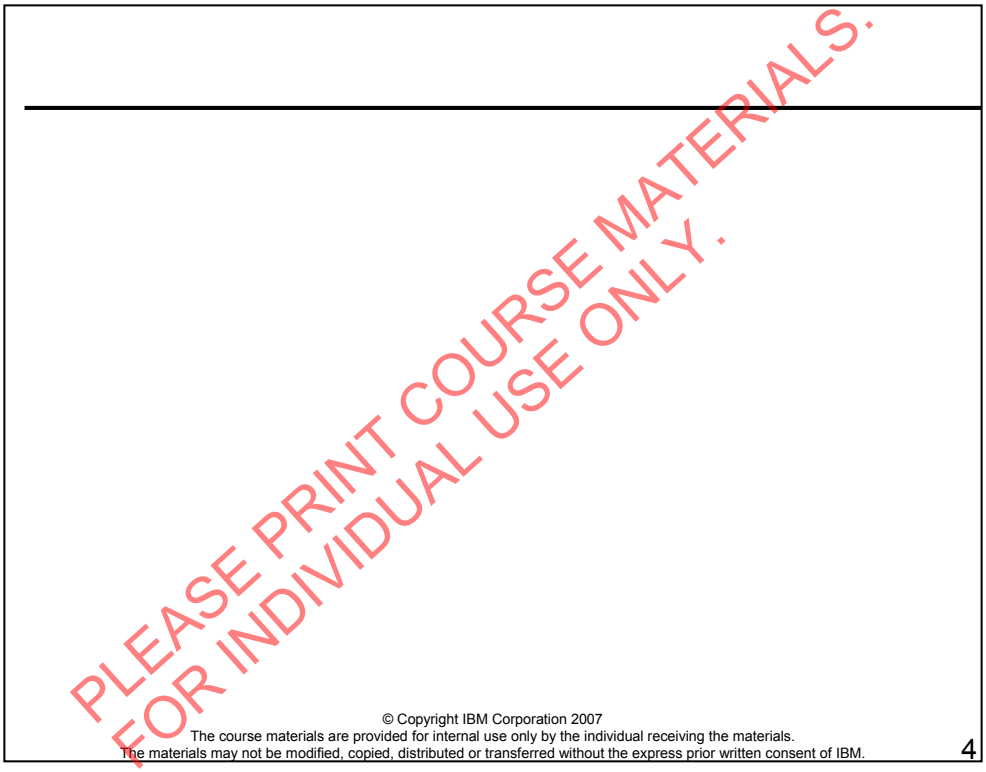
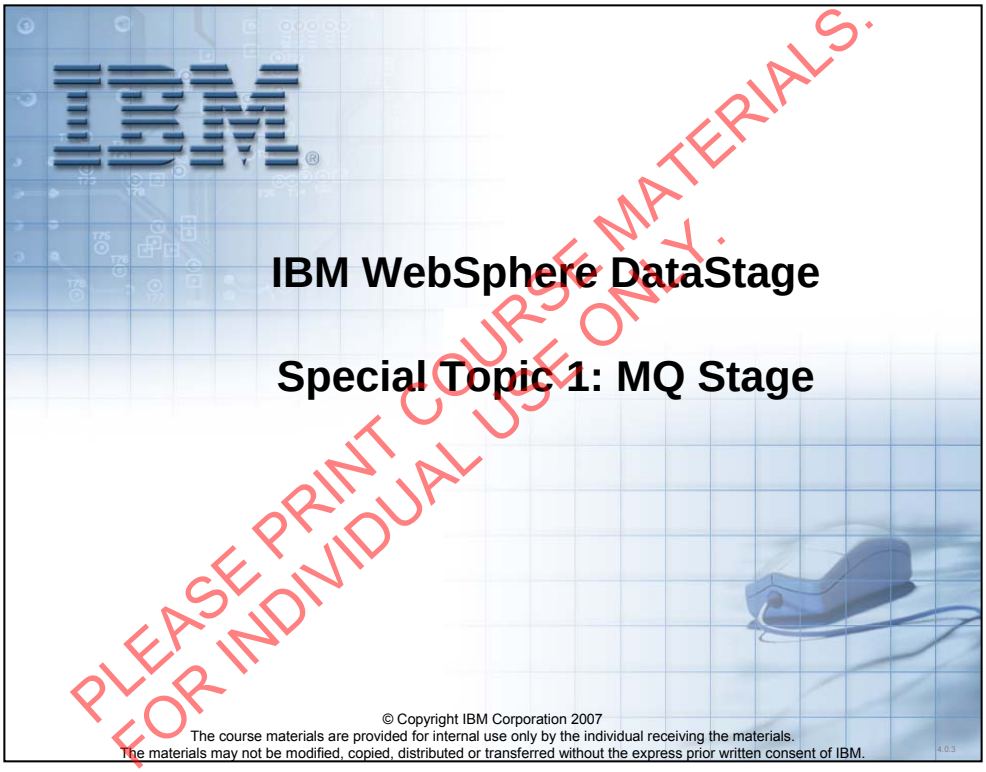

DataStage Essentials v8
Special Topics
PLEASE PRINT COURSE MATERIALS.
FOR INDIVIDUAL USE ONLY.
© Copyright IBM Corporation 2007
The course materials are provided for internal use only by the individual receiving the materials.
The materials may not be modified, copied, distributed or transferred without the express prior written consent of IBM.
4.0.3

IBM Software Group
One Rogers Street
Cambridge, MA 02142

Course Contents

3


PLEASE PRINT COURSE MATERIALS.
FOR INDIVIDUAL USE ONLY.
© Copyright IBM Corporation 2007
The course materials are provided for internal use only by the individual receiving the materials.
The materials may not be modified, copied, distributed or transferred without the express prior written consent of IBM.
4


IBM
IBM WebSphere DataStage
Special Topic 1: MQ Stage
PLEASE PRINT COURSE MATERIALS.
FOR INDIVIDUAL USE ONLY.
© Copyright IBM Corporation 2007
The course materials are provided for internal use only by the individual receiving the materials.
The materials may not be modified, copied, distributed or transferred without the express prior written consent of IBM.
4.0.3

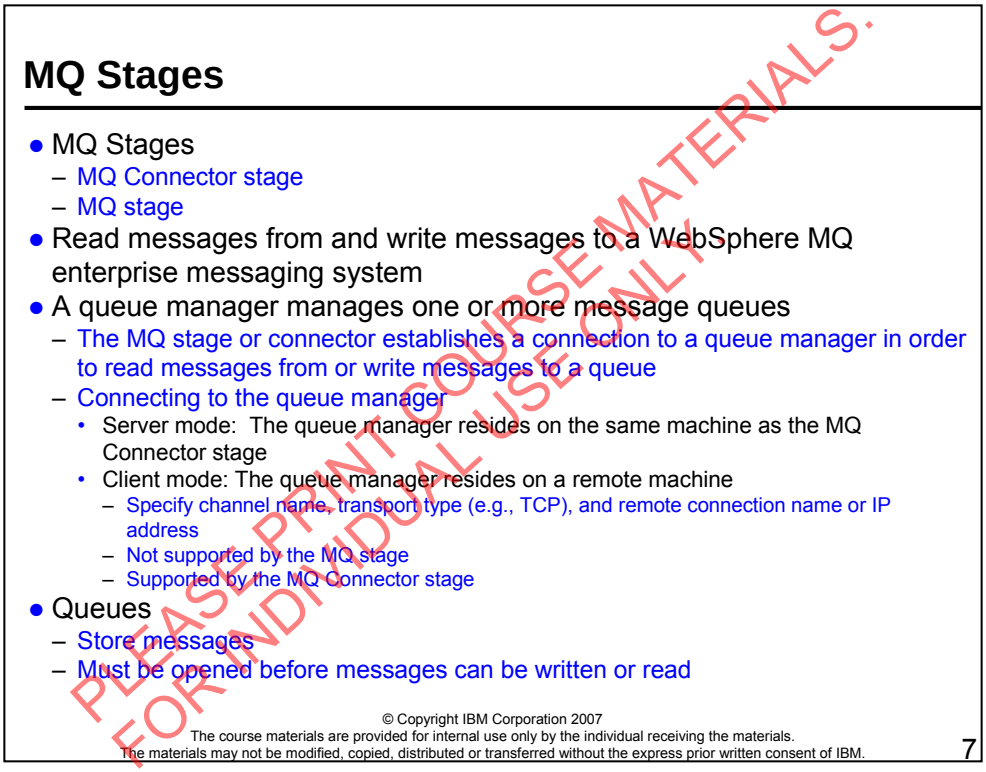
Unit objectives

After completing this unit, you should be able to:

- Connect an MQ queue manager
- Read messages from an MQ queue
- Write messages to an MQ queue
- Retrieve the message payload
- Specify header information to retrieve from a message

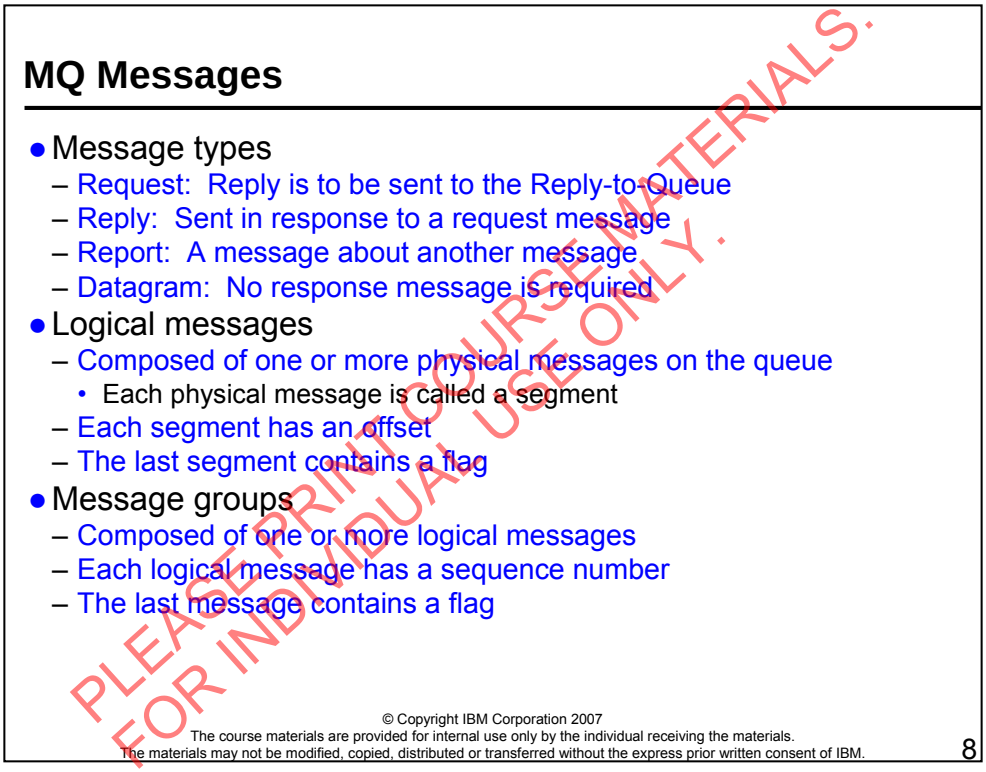
6

Notes:



There are two stages that can be used to read and write MQ messages: MQ Connector stage, which has the same GUI as the ODBC Connector stage, and the MQ stage. The MQ Connector stage is the latest technology.

The MQ Connector stage is not available in the first release of DataStage v8.



All record types are supported by the MQ stages. In this introductory course we will consider only the simplest type, datagram.

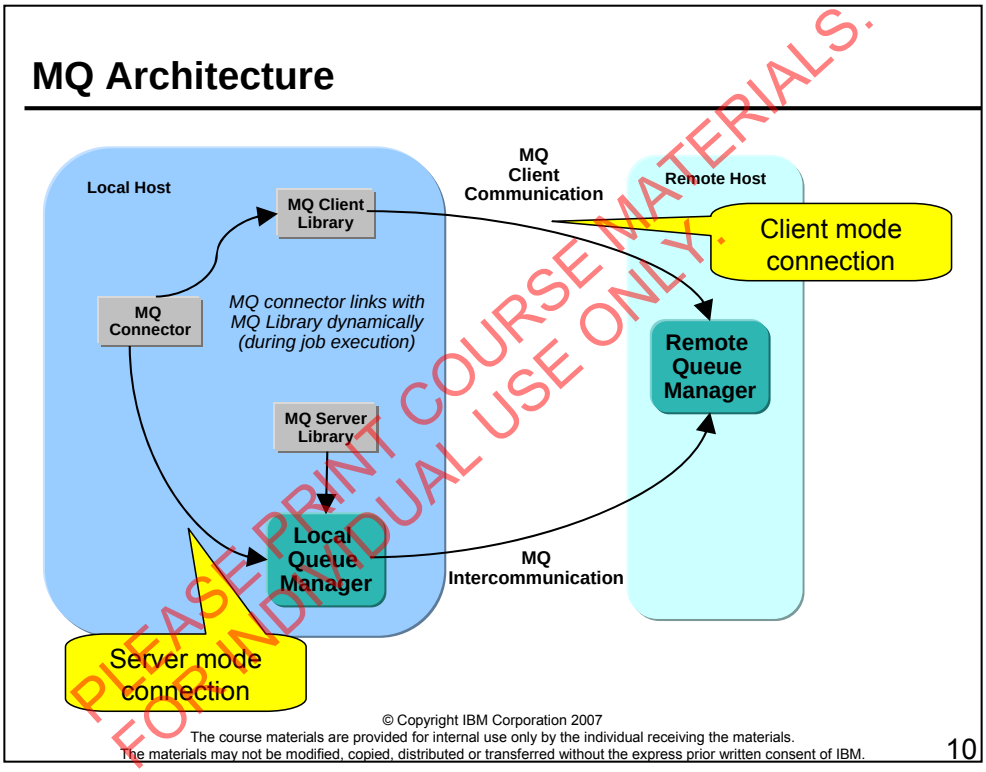
The user can specify the assembly of segments into a logical message.

The user can specify the assembly of messages into a message group.

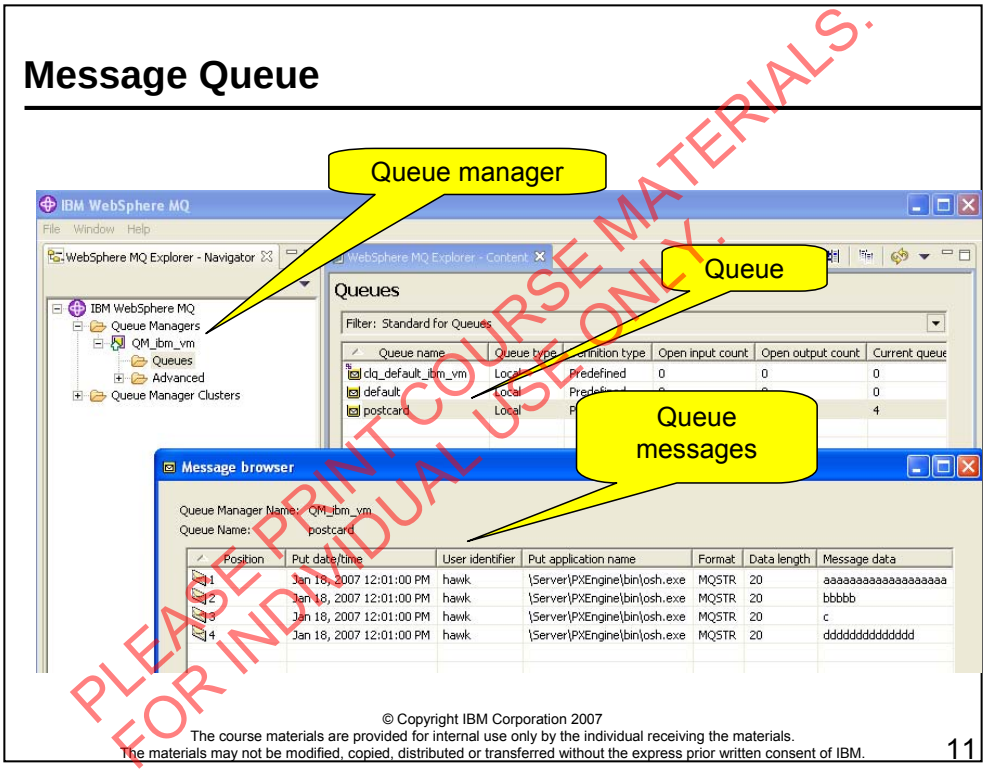
Message Structure

- Two or three parts
  - Header: Information about the content and structure of the data
  - Optional format header: Information about the message format
  - Payload: Message data
- Message schema
  - Defines the type and structure of the data

9

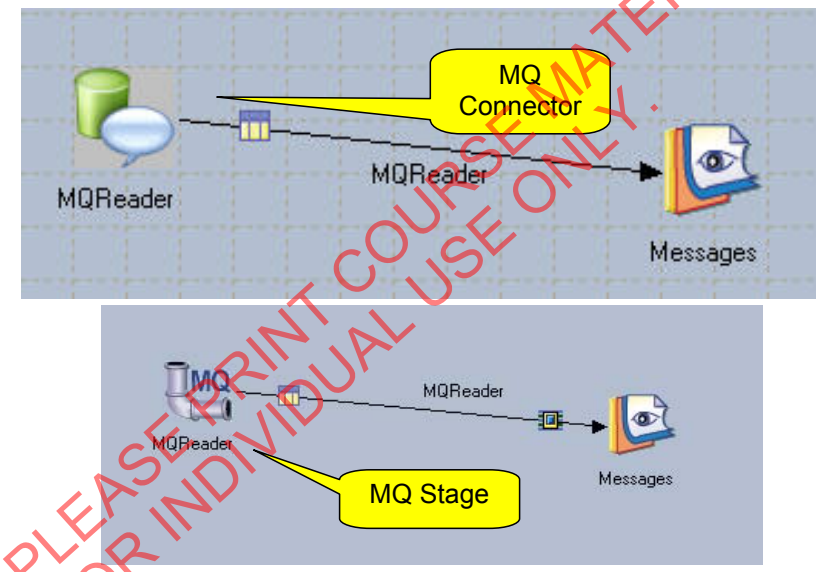


Client mode connection is only available in the MQ Connector.



This shows the WebSphere MQ Explorer application. A message queue named QUEUE1 is displayed with some messages it contains.

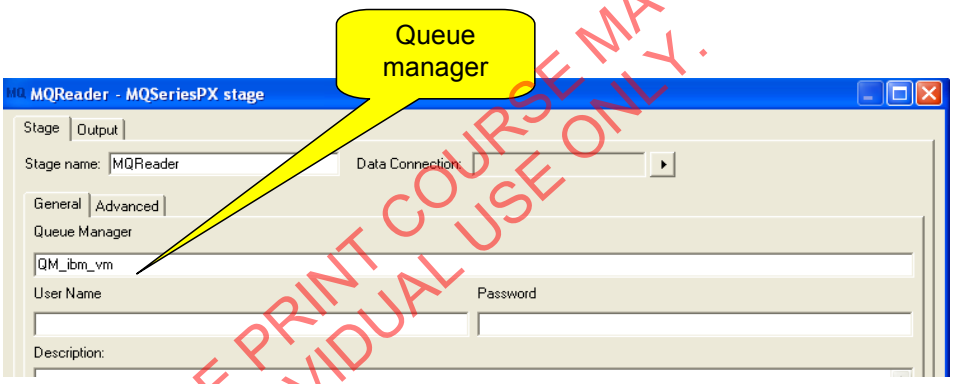
MQ Stage Readers



12

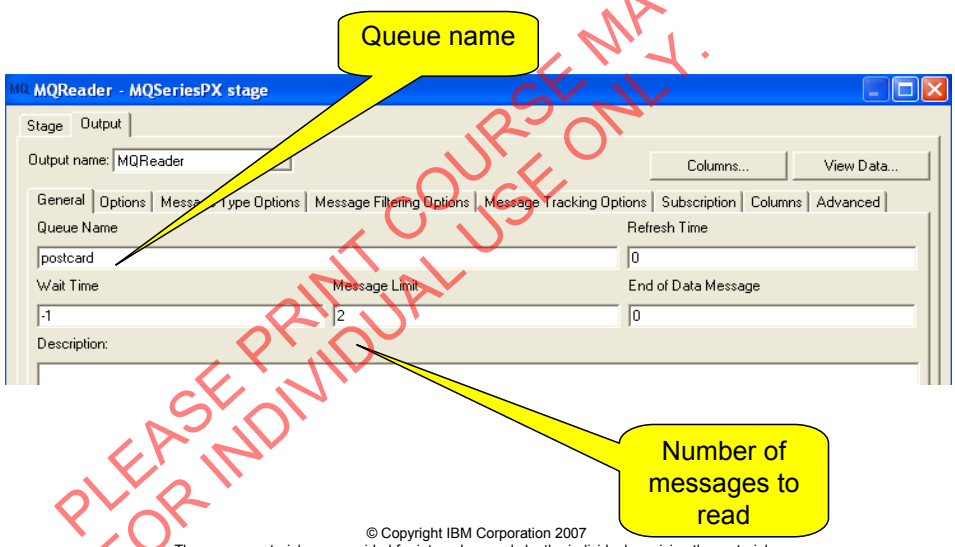
Here the MQ Connector and MQ stages used to read messages from a queue. The stage has a single output link. Messages read are sent to a Peek stage.

MQ Reader Properties



13

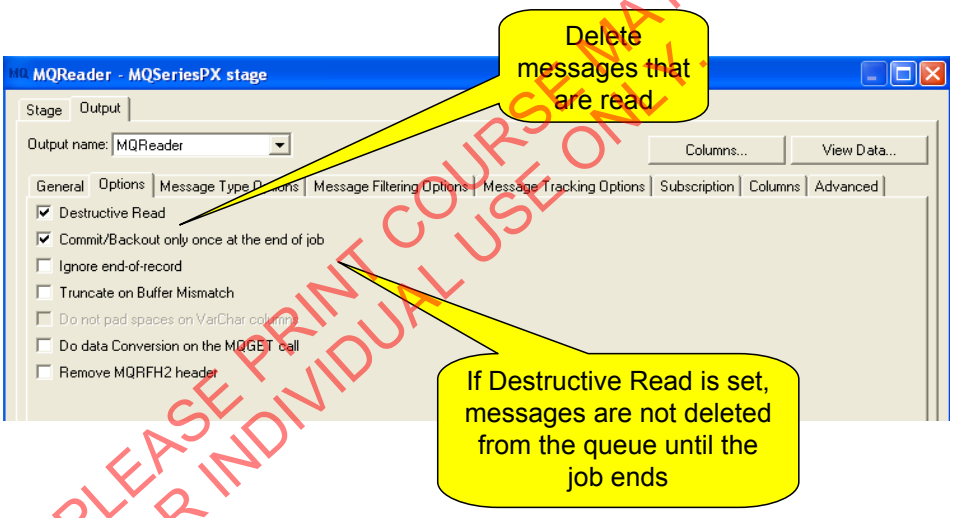
Output General Tab



14

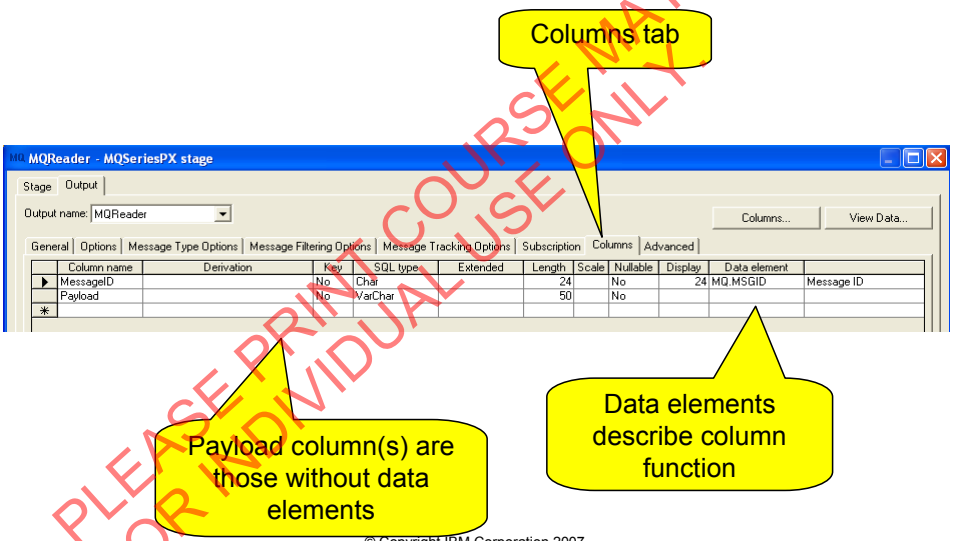
By default, the message limit is 0, meaning no limit. In that case the job will have no termination condition (unless another type is specified).

Options Tab



15

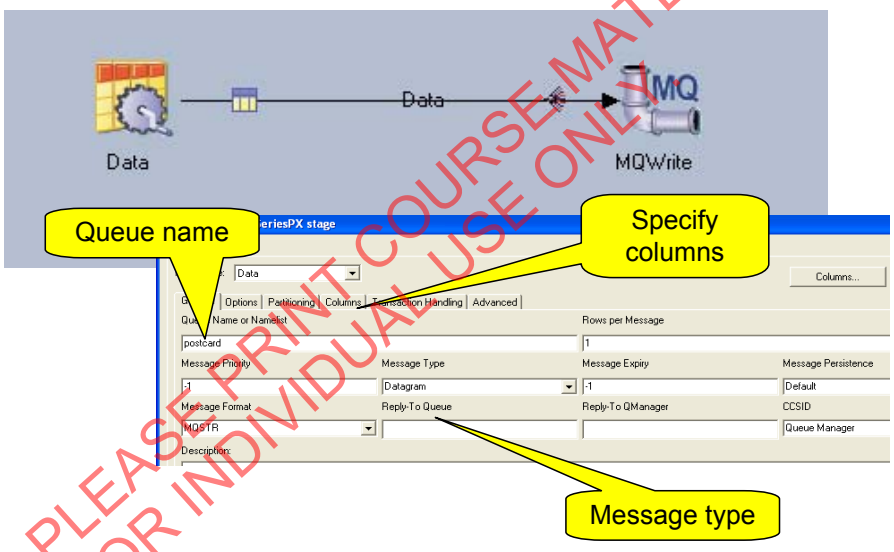
Columns Tab



16

There can be multiple payload columns. This allows the read data to be parsed for later processing.

Writing to a Queue

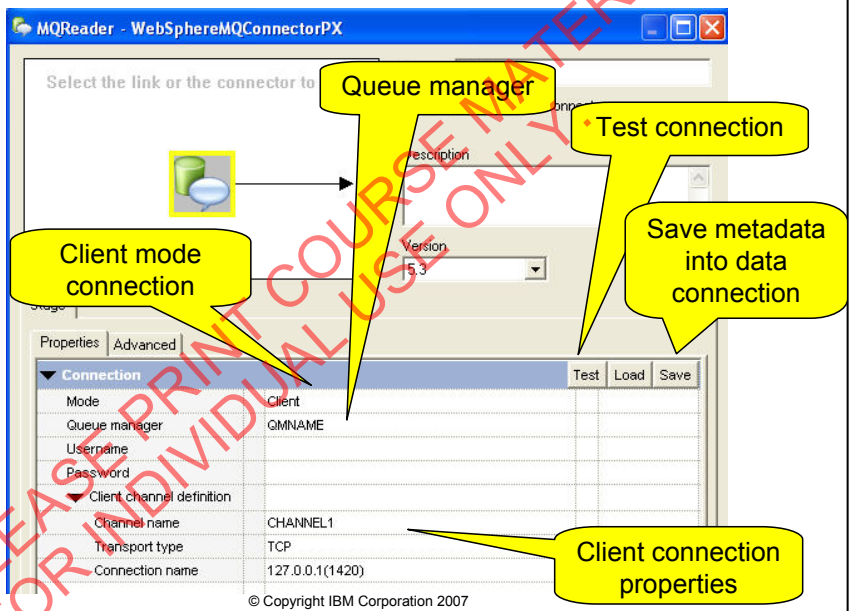


17

MQ Connector Stage

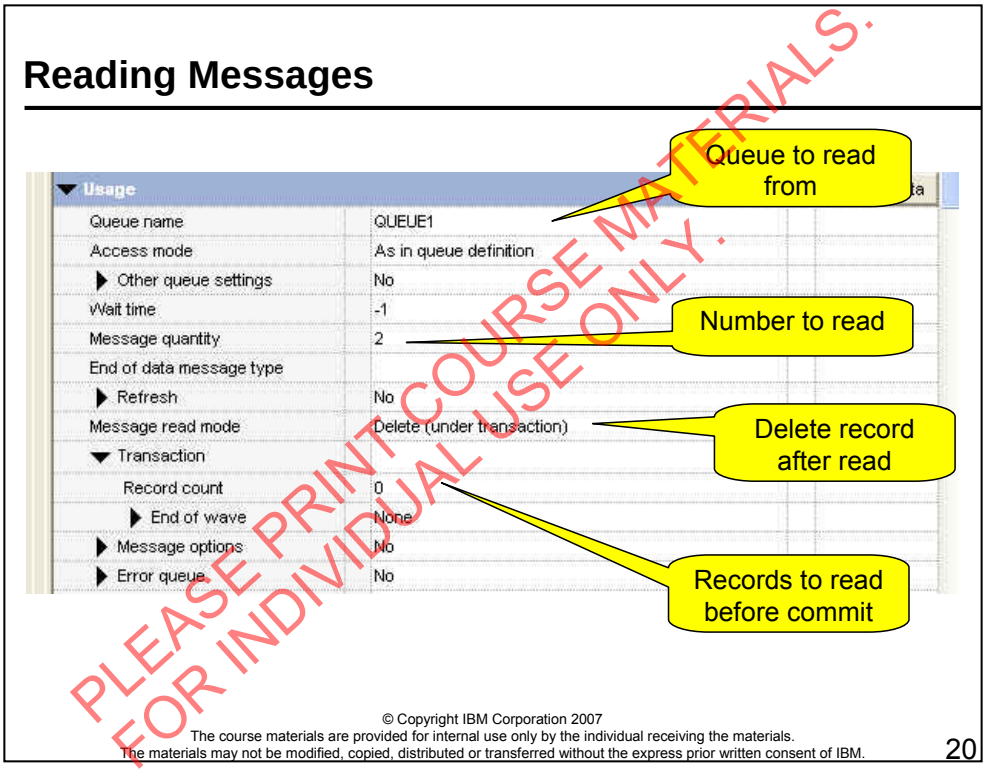
18

Connecting to the Queue Manager



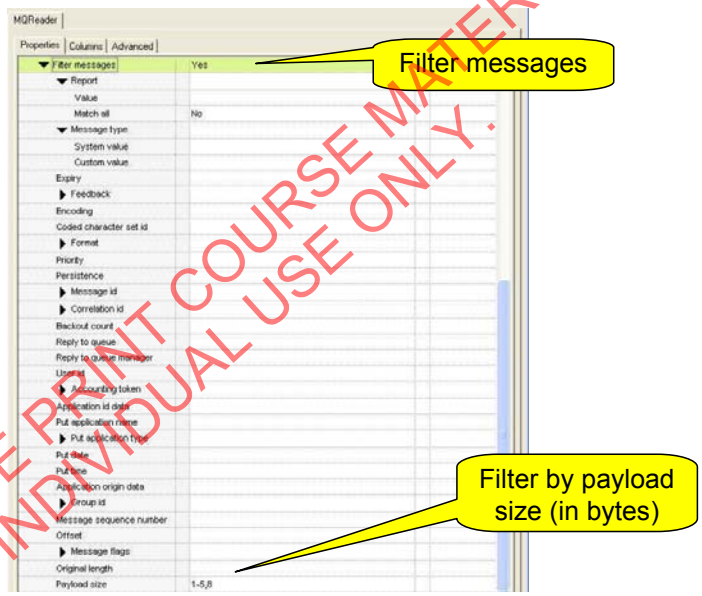
19

The queue manager can be selected from a list of discovered queue managers.



The queue can be selected from a list of discovered queues.

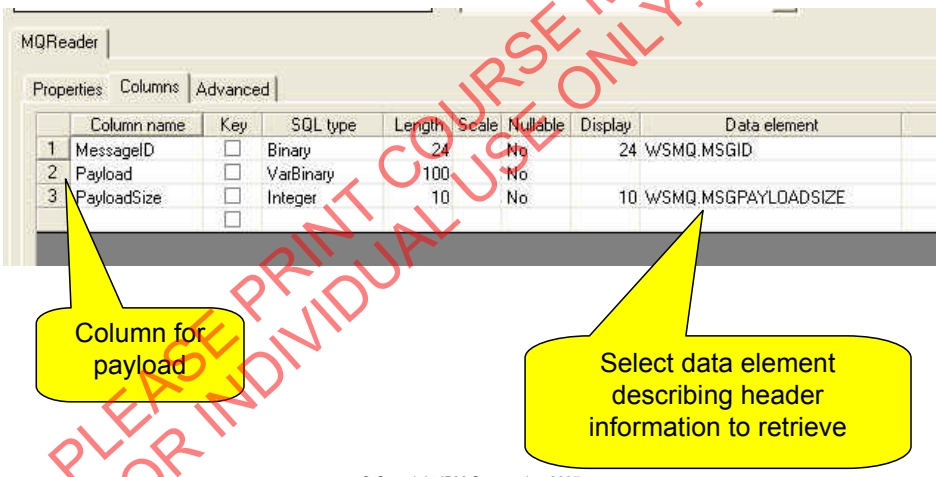
Filtering Read Messages



21

Here, we are filtering messages by payload (data) size. In this example, messages with between 1 and 5 bytes of data and messages with 8 bytes of data will be read (consumed). Other messages will be left in the queue.

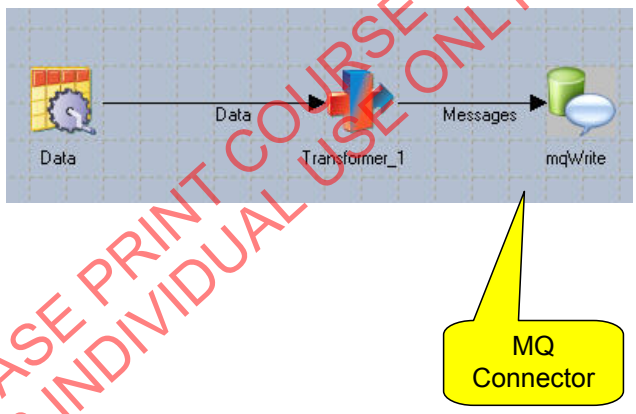
Selecting Message Information



22

Fields defined without specified data elements are treated as payload fields. Multiple payload fields can created. The first payload field will extract the first n characters of the payload, where n is the length of the field. The second will extract the next characters in the payload. And so on.

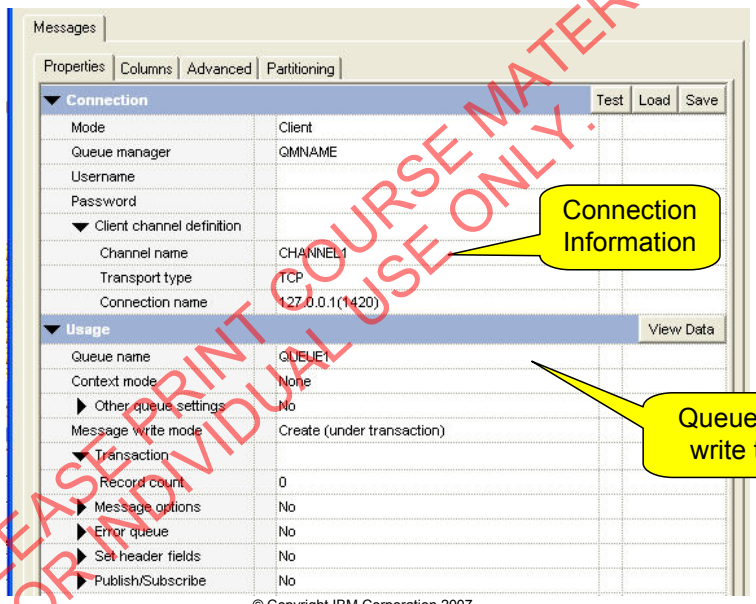
Writing Messages to a Queue



23

Here the MQ Connector is used to write messages to the queue. So it has a single input link. Messages are read from a Row Generator stage.

MQ Write Properties



24

Checkpoint

1. What two modes can be used to connect to a queue manager?
2. When reading a message from a queue will the MQ Connector stage use an input link, an output link, or both an input and output link?
3. When writing a message to a queue will the MQ Connector stage use an input link, an output link, or both an input and output link?
4. How do you specify that a column on the Columns tab is to retrieve a certain kind of header information, for example, the message type.

25

Notes:

Write down your answers here:

1.

2.

Checkpoint solutions

1. Server mode can be used to connect to a queue manager on the same machine as the MQ Connector stage. Client mode can be used to connect to a remote queue manager.
2. One output link.
3. One input link.
4. Select the data element that describes the type of information.

26

Unit summary

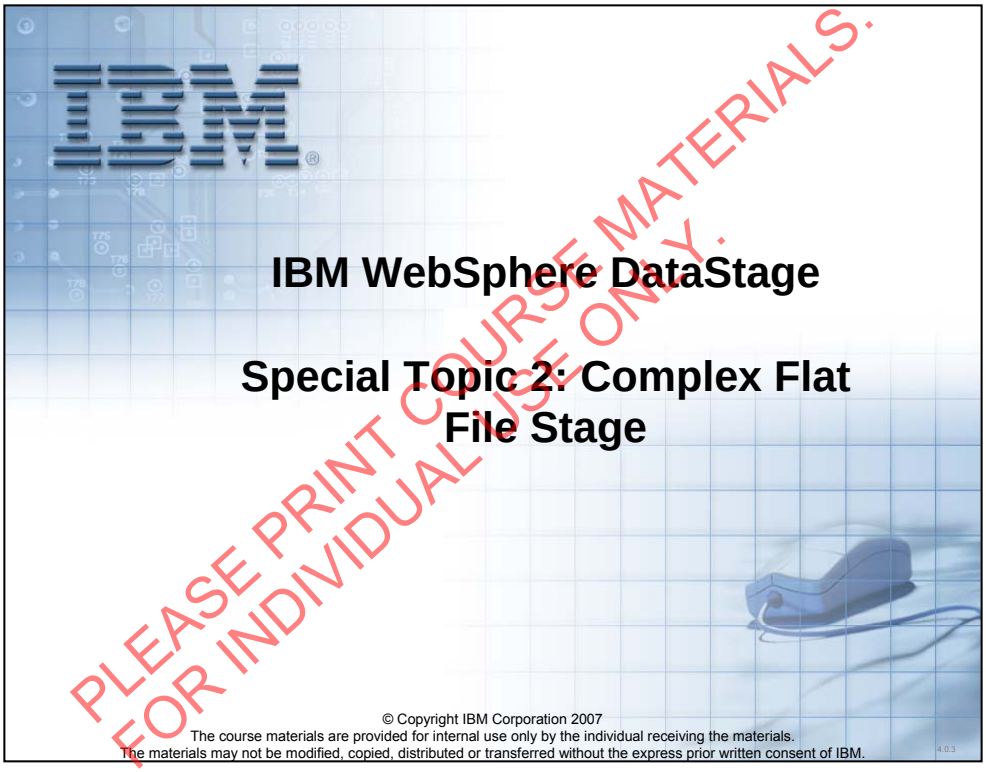
Having completed this unit, you should be able to:

- Connect remotely to an MQ queue
- Read messages from an MQ queue
- Write messages to an MQ queue
- Retrieve the message payload
- Specify header information to retrieve from a message

27

Notes:

28


IBM
IBM WebSphere DataStage
Special Topic 2: Complex Flat File Stage
PLEASE PRINT COURSE MATERIALS. FOR INDIVIDUAL USE ONLY.
© Copyright IBM Corporation 2007
The course materials are provided for internal use only by the individual receiving the materials.
The materials may not be modified, copied, distributed or transferred without the express prior written consent of IBM.
4.0.3

Unit objectives

After completing this unit, you should be able to:

- Import table definitions from a COBOL copybook
- Design a job that extracts data from a COBOL file containing multiple record types
- Specify in a Complex Flat File (CFF) stage the column layouts of each record type
- Specify in a CFF stage how to identify when a record is read of a specific type
- Select in a CFF stage which columns from the different records types are to be output from the stage

30

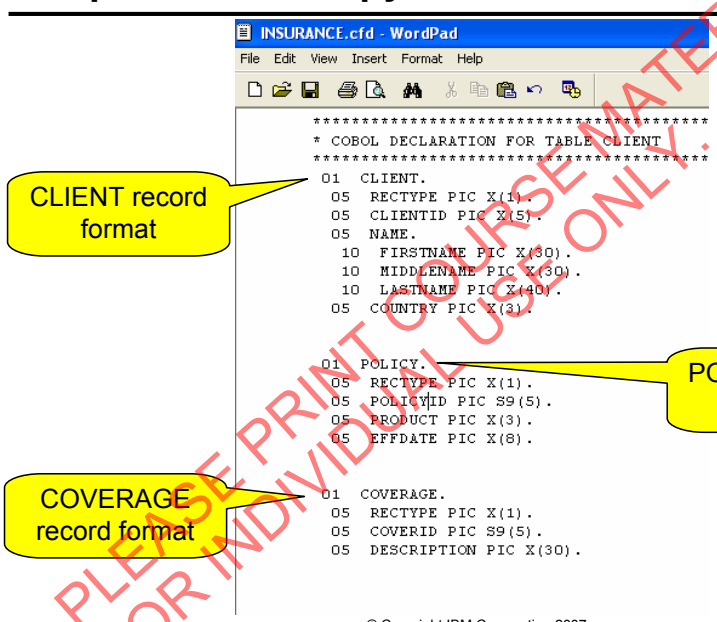
Notes:

Complex Flat File Stage

- Process data in a COBOL file
  - File is described by a COBOL file description
  - File can contain multiple record types
- COBOL copybooks with multiple record formats can be imported as COBOL File Definitions
  - Each format is stored as a separate DataStage table definition
- Columns can be loaded for each record type
- On the Records ID tab, you specify how to identify each type of record.
- Columns from any or all record types can be selected for output
  - This allows columns of data from multiple records of different types to be combined into a single output record

31

Sample COBOL Copybook



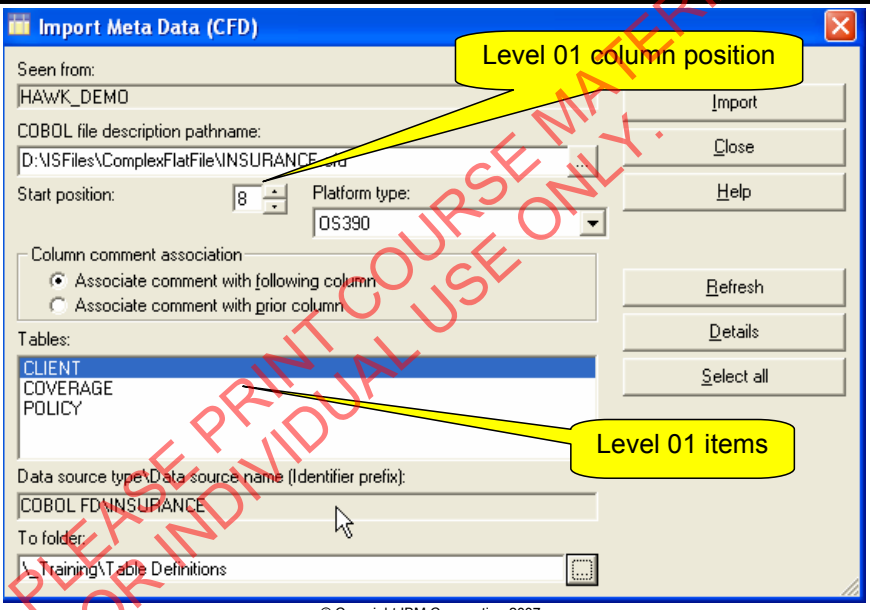
CLIENT record format

POLICY record format

COVERAGE record format

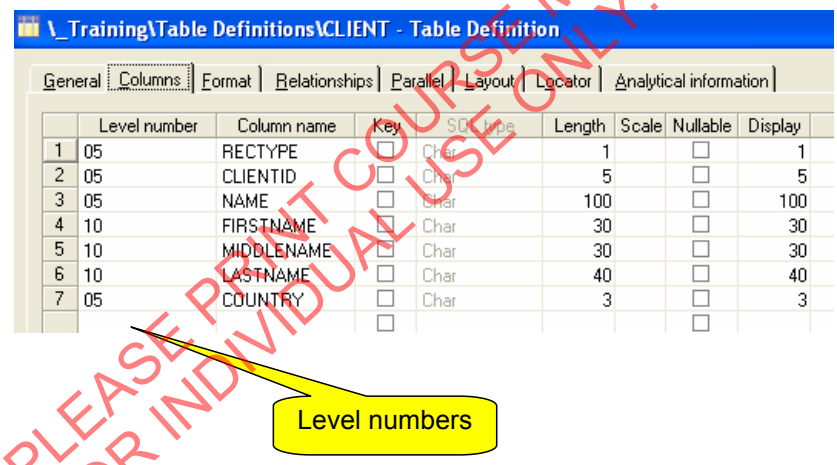
32

Importing a COBOL File Definition

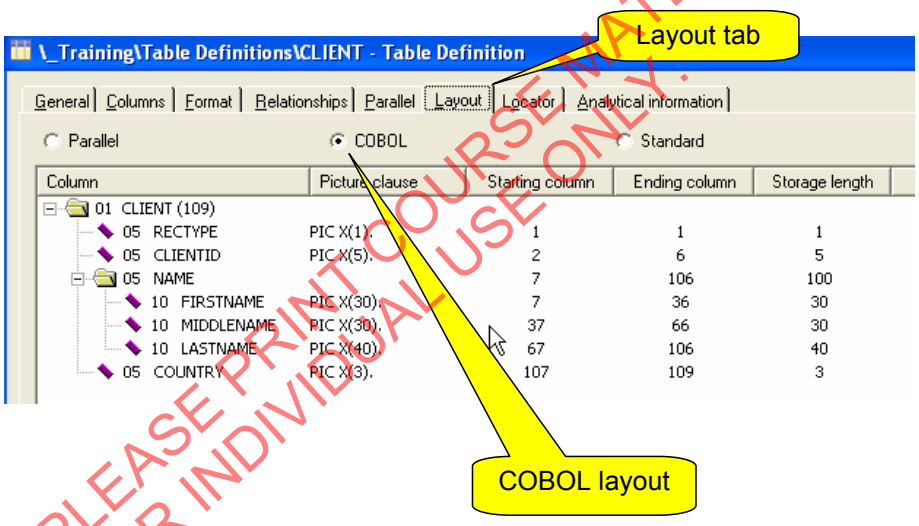


33

COBOL Table Definitions

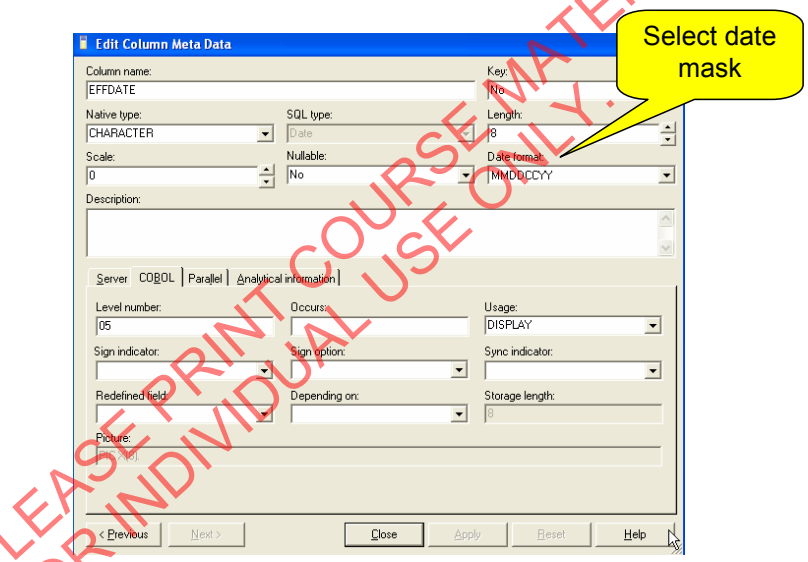


COBOL File Layout



35

Specifying a Data Mask

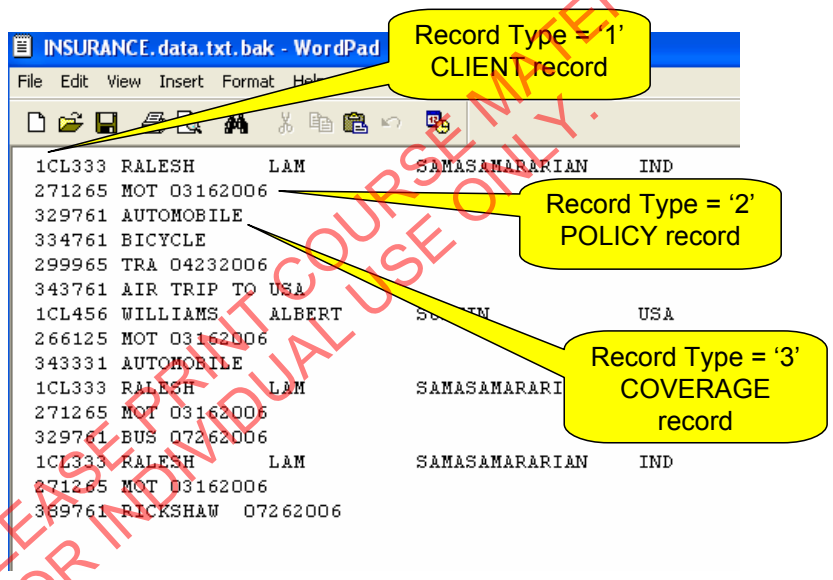


36

Double-click to the left of the column number on the Columns tab to open the Edit Column Meta Data window. Select a field that contains date values. Then select the date mask that describes the format of the date from the Date format list.

The SQL type is changed to Date. All dates are stored in a common format, which is described in project or job properties. By default, dates are stored in DB2 format.

Example Data File with Multiple Formats

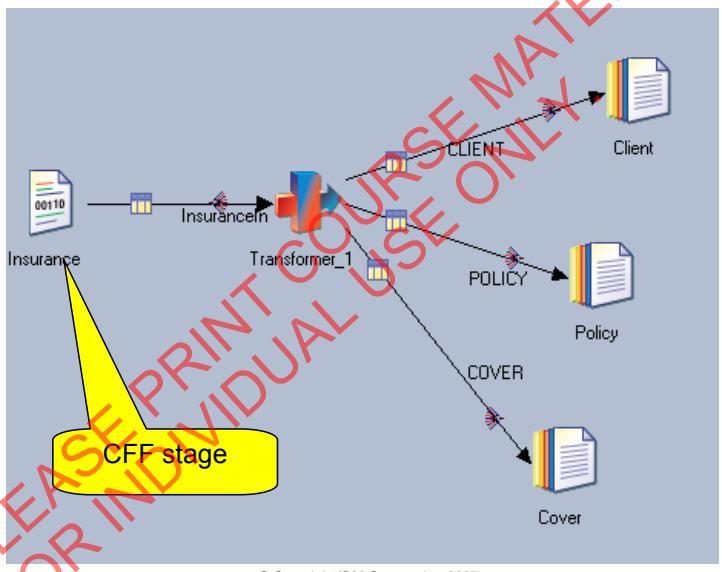


37

For clarity in this example, each record type has been placed on a separate line. Spaces have been added between fields. In practice, the records might follow each other immediately without being placed on a separate line. In the file used in the lab exercises, records follow each other immediately with a single record character, the pipe (|), separating them.

In this example, client information is stored as a group of three types of records: CLIENT, POLICY, COVERAGE. There is one CLIENT record type which is the first record of the group. This can be followed by one or more POLICY records. Each POLICY record is followed by one or more COVERAGE records. Client Ralesh has two insurance policies. The first is for motor vehicles (MOT). He has two coverages under this policy. The second policy is for travel (TRA). He has one coverage under this policy.

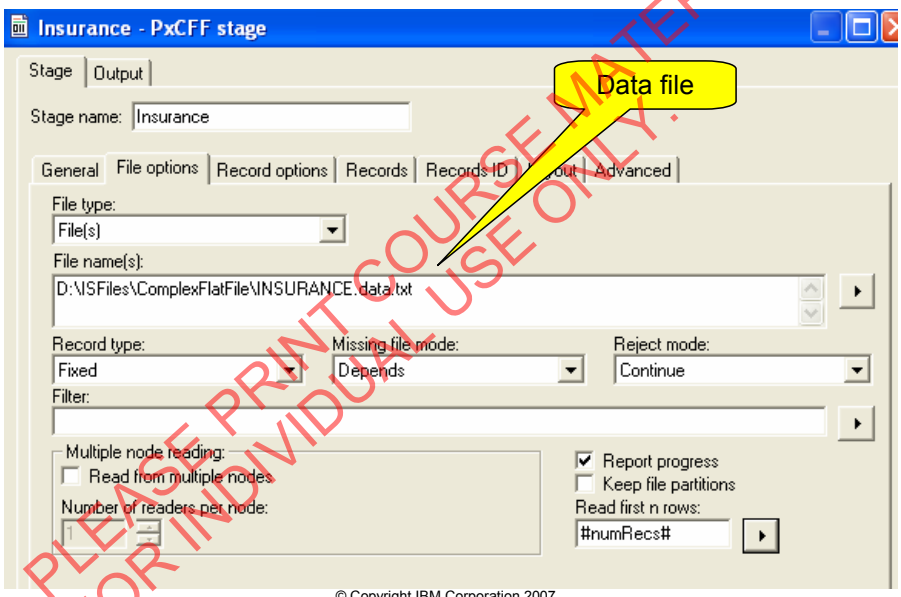
Sample Job With CFF Stage



38

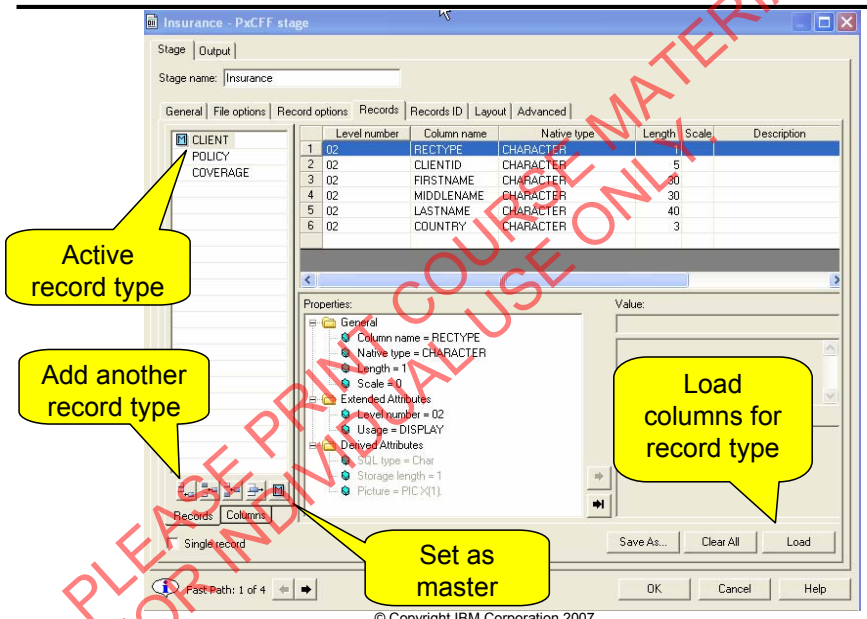
The Transformer in this job is used to split the data into multiple outputs streams. In the Transformer, a separate constraint is defined on each output link. Alternatively, the three output links with their constraints could have come directly from the CFF stage. The CFF stage supports multiple output links and constraints. A Transformer is required if derivations need to be performed on any columns of data.

File Options tab



39

Records Tab

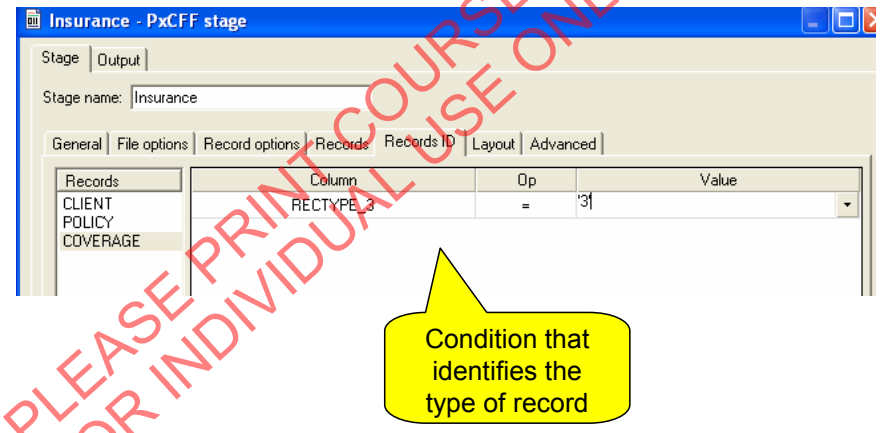


40

Define each record type on the Records tab. Here we see that three record types have been defined. For each type, click the Load button to load the table definition that defines the type.

To add another record type click a button at the bottom of the Records tab. Click the far right icon to set it as master. When a master record is read the output buffer will be emptied.

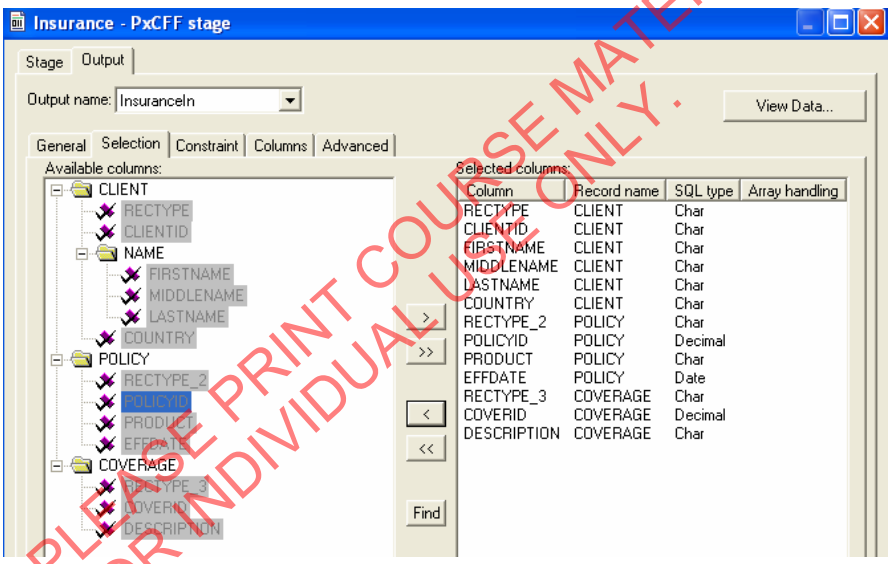
Record ID Tab



41

If the RECTYPE_3 field contains a '3', then the record is a COVERAGE record. A constraint must be defined for each record type.

Selection Tab



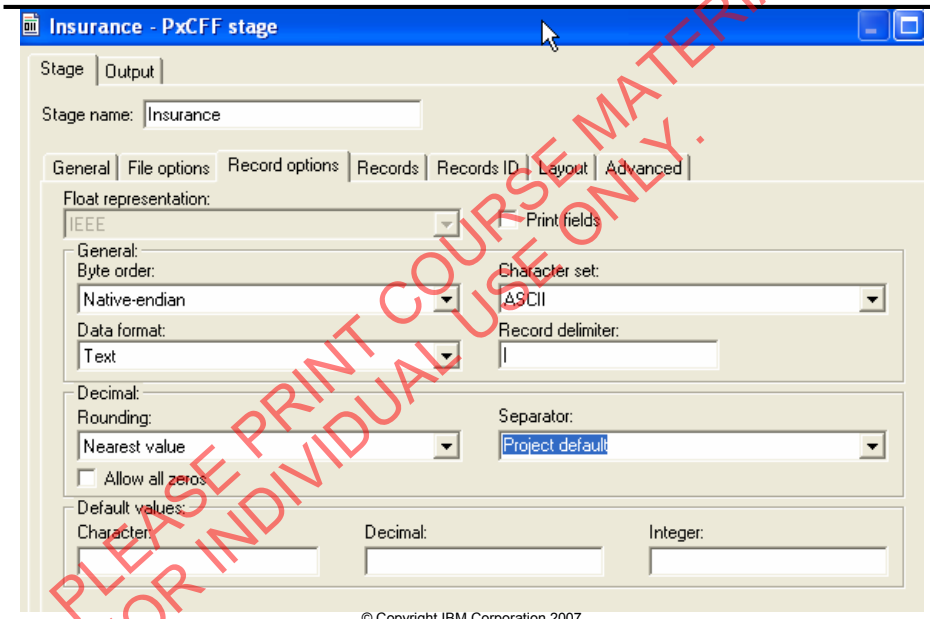
42

After each record is read a record will be sent out the output link. This tab is where you specify the columns of the output record. Notice that the output record can contain values from any or all of the record types.

Since only a single record type is read at a time, only some of the output columns (those which get their values from the current record type) will receive values. The other columns will retain whatever value they had before or they will be empty. Whenever the master record is read all columns are emptied before the new values are written.

It is crucial to be aware that although each output record has all of these columns, not all of these columns will necessarily have valid data. When you process these records, e.g., in a Transformer, you need to determine which fields contain valid data.

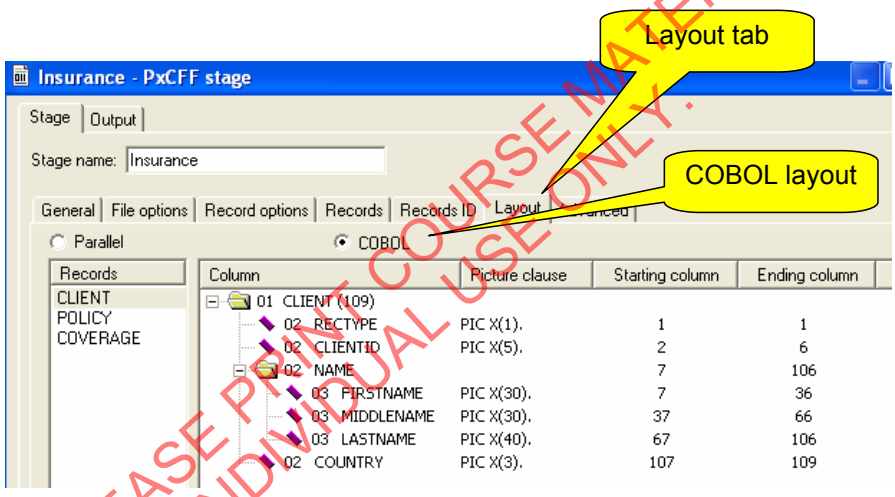
Record Options Tab



43

On the Records options tab, you specify format information about the file records. Here, the file is described as a text file (rather than binary), as an ASCII file (rather than EBCDIC), and a file with records separated by the pipe (|).

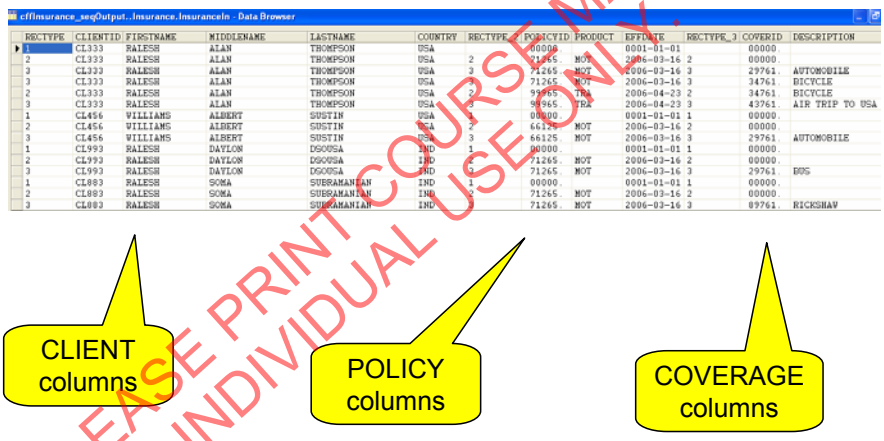
Layout Tab



44

The Layout tab is a very useful tab. It displays the length of the record (as described by the metadata) and the lengths and offsets of each column in the record. It is crucial that the metadata accurately describe the actual physical layout of the file. Otherwise errors will occur when the file is read.

View Data



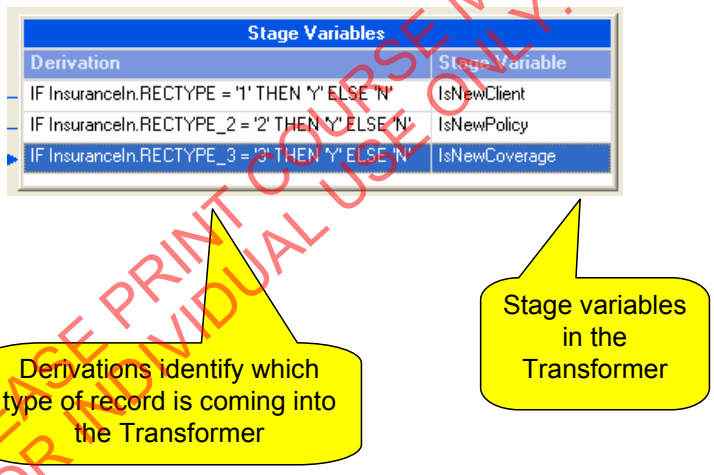
45

Click the View Data button to view the data. When you view the data, you are viewing the data in the output columns.

Notice here that after the first record is read, which we can tell by the RECTYPE field is a CLIENT record, only the CLIENT columns contain valid data. After the second record is read, which is POLICY record (as we can tell by the RECTYPE_2 field), the POLICY columns get populated. The CLIENT columns contain the values they contained after the previous record. After the third record is read, which is a COVERAGE record (as we can tell by the RECTYPE_3 field), the COVERAGE columns get populated.

Notice how the POLICY and COVERAGE fields are emptied when the next CLIENT record is read. These fields are emptied because CLIENT was designated the Master record.

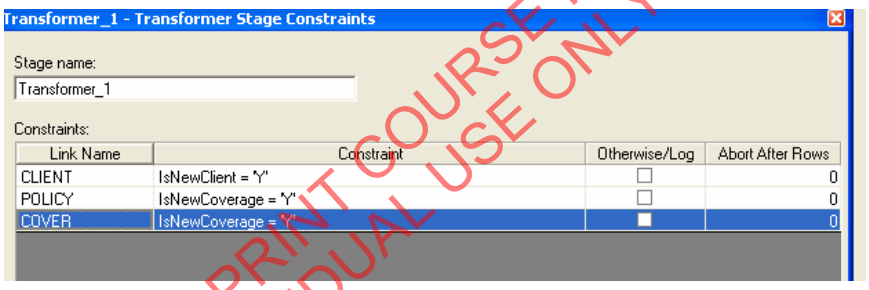
Processing Multi-Format Records



46

When the IsNewClient stage variable equals ‘Y’, then we know that the CLIENT columns contain valid data. When the IsNewPolicy stage variable equals ‘Y’, then we know that both the POLICY and the CLIENT columns contain valid data. When the IsNewCoverage stage variable equals ‘Y’, then we know that all the columns contain valid data.

Transformer Constraints



47

These constraints ensure that a record is written out the CLIENT output link only when the columns contain valid client information. And so on for the POLICY and COVERAGE output links.

Checkpoint

1. What type of files contain the metadata that is typically loaded into the CFF stage?
2. Does the CFF stage support variable length records?
3. How does DataStage know which type of record it is reading from a file containing records of different formats?
4. What does it accomplish to select a record type as a master?
5. How many record types can be designated Master?

48

Notes:

Write down your answers here:

1.

2.

Checkpoint solutions

1. COBOL copybooks or COBOL file definitions.
2. Yes, it can read files containing multiple file formats, each of a different physical length.
3. On the Records ID tab, you define constraints that identify the record type. These must reference fields common to all record formats.
4. When a master record is read, all output columns are emptied before the master record contents are written.
5. Only one.

Unit summary

Having completed this unit, you should be able to:

- Import table definitions from a COBOL copybook
- Design a job that extracts data from a COBOL file containing multiple record types
- Specify in a Complex Flat File (CFF) stage the column layouts of each record type
- Specify in a CFF stage how to identify when a record is read of a specific type
- Select in a CFF stage which columns from the different records types are to be output from the stage

50

Notes:


IBM WebSphere DataStage
Special Topic 3: Slowly Changing Dimensions
PLEASE PRINT COURSE MATERIALS.
FOR INDIVIDUAL USE ONLY.
© Copyright IBM Corporation 2007
The course materials are provided for internal use only by the individual receiving the materials.
The materials may not be modified, copied, distributed or transferred without the express prior written consent of IBM.
4.0.3

Unit objectives

After completing this unit, you should be able to:

- Design a job that creates a surrogate key source key file
- Design a job that updates a surrogate key source key file from a dimension table
- Design a job that processes a star schema database with Type 1 and Type 2 slowly changing dimensions

52

Notes:

Surrogate Key Generation Stage

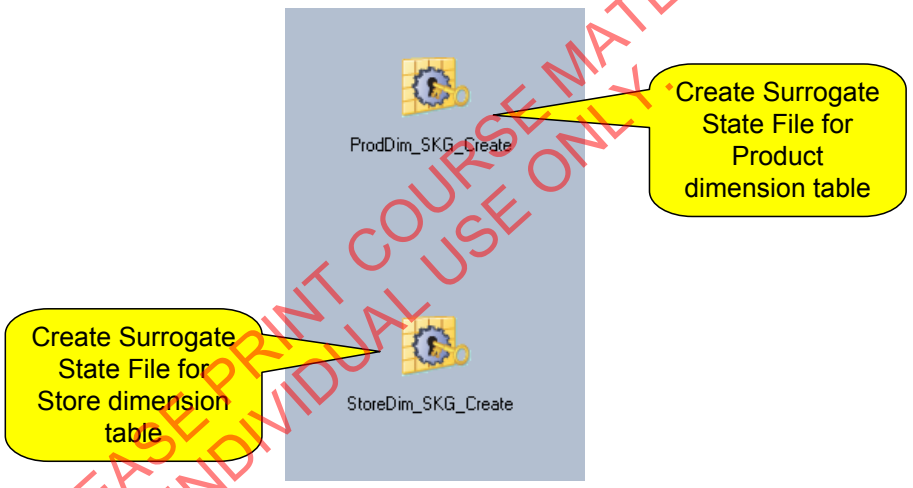
53

Surrogate Key Generator Stage

- Use to create and update the surrogate key state file
- Surrogate key state file
  - One file per dimension table
  - Stores the last used surrogate key integer for the dimension table
  - Binary file

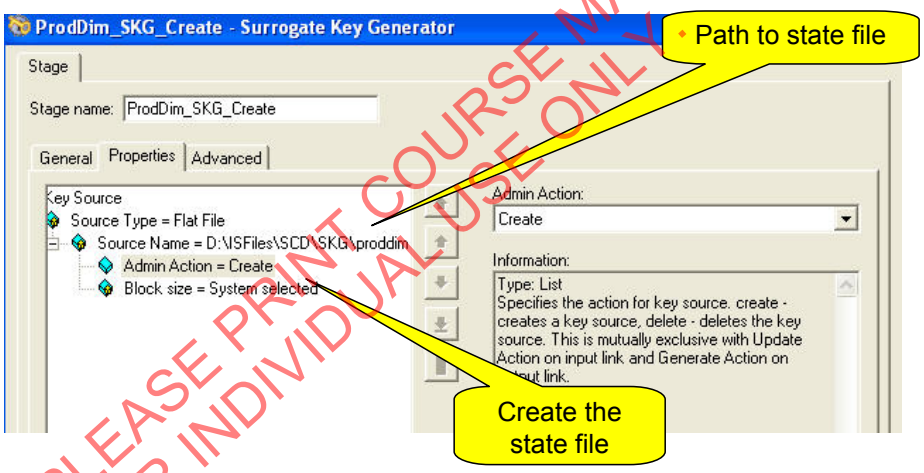
54

Example Job to Create Surrogate State Files



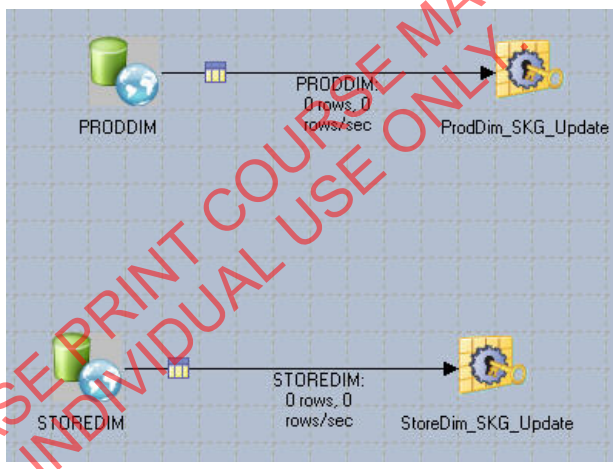
55

Editing the Surrogate Key Generator Stage



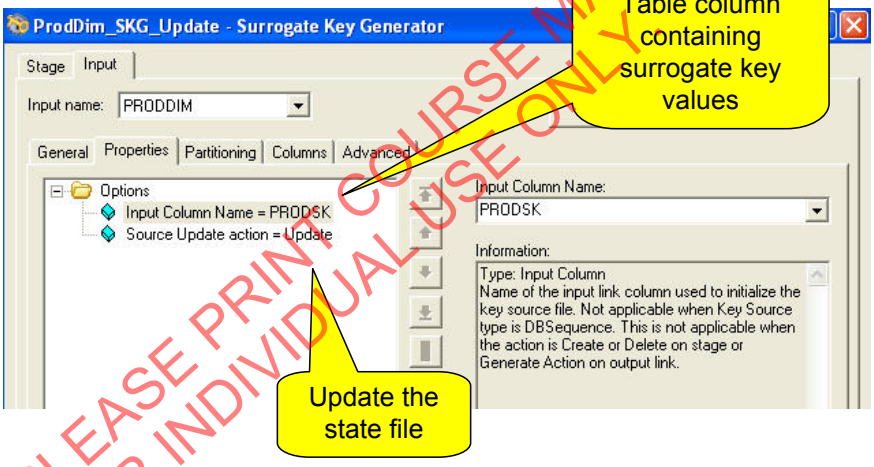
56

Example Job to Update the Surrogate State File



57

Specifying the Update Information



58

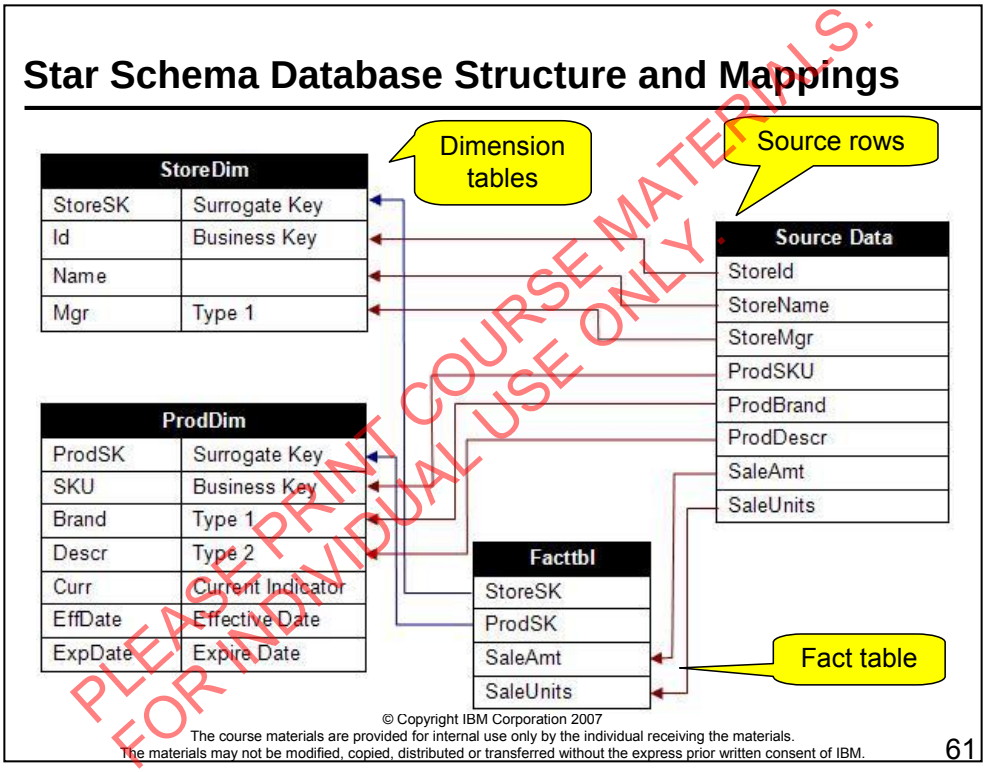
Slowly Changing Dimensions Stage

59

Slowly Changing Dimensions Stage

- Used for processing a star schema
- Performs a lookup into a star schema dimension table
  - Multiple SCD stages can be chained to process multiple dimension tables
- Inserts new rows into the dimension table as required
- Updates existing rows in the dimension table as required
  - Type 1 fields of a matching row are overwritten
  - Type 2 fields of a matching row are retained as history rows
    - A new record with the new field value is added to the dimension table and made the current record
- Generally used in conjunction with the Surrogate Key Generator stage
  - Creates a Surrogate Key state file that retains a list of the previously used surrogate keys

60



The fact table is the center of the Star Schema. It contains the numerical (factual) data that is aggregated over to produce analytical reports covering the different dimensions. Non-numerical (non-factual) information is stored in the dimension tables. This information is referenced by surrogate key values in the act table rows.

This example star schema database has two dimensions. The StoreDim table stores non-numerical information about stores. Each store has been assigned a unique surrogate key value (integer). Each row stores information about a single store, including its name, its manager, and its business identifier (a.k.a., natural key, business key). The ProdDim table stores non-numerical information about a single product, including its brand, its description, and its business identifier.

Each row in the fact table references a single store and a single product by means of their surrogate keys. Why are surrogate keys used rather than the business keys? There are two major reasons. First, surrogate keys can yield better performance because they are numbers rather than, possibly, long strings of characters. Secondly, it is possible for their to be duplicate business keys, coming from different source systems. For example, the business key X might refer to bananas in Australia, but tomato soup in Mexico.

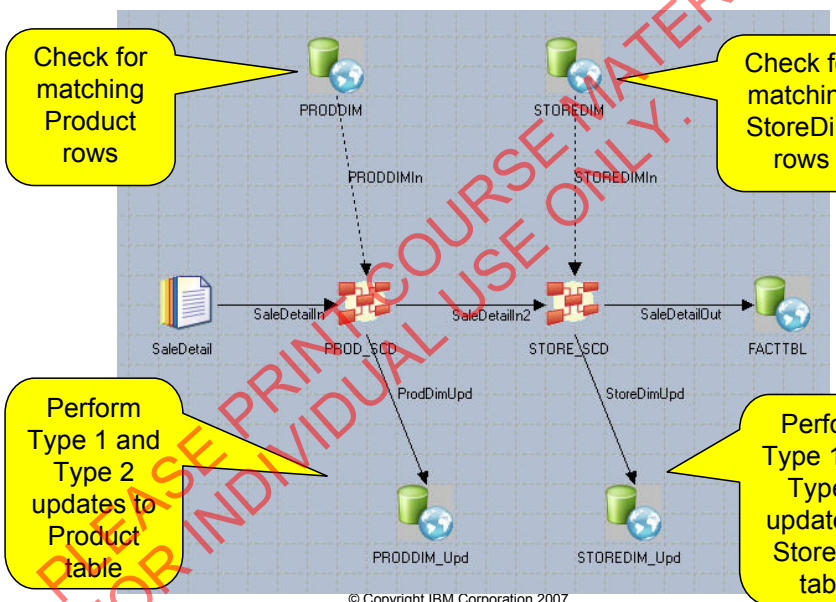
In this example, each row in the fact table contains a sales amount and units for a particular product sold by a particular store for some given period of time not shown in this example. For simplicity, the time dimension has been omitted.

A source record contains sales detail from a sales order. It includes information about the product sold and the store that sold the product. This information needs to be put into the star schema. The store information needs to go into the StoreDim table. The product information needs to go into the ProdDim table and the factual information needs to go into the Facttbl table. Moreover, the record put into the fact table must contain surrogate key references to the corresponding rows in the StoreDim and ProdDim tables.

In this example, the Mgr field in the StoreDim table is considered a Type 1 dimension table attribute. This means that if a source record that references a certain store lists a different manager, then this is to be considered a simple update of the record for that store. The value in the source data replaces the value in the existing store record by means of a simple update to the existing record. Similarly, Brand is a Type 1 dimension table attribute of the ProdDim table.

In this example, the Descr field is a Type 2 dimension table attriubute. Suppose a source data record contains a different product description for a given product than the current record for that product in the ProdDim table. The record in the ProdDim table is not simple updated with the new product description. The record is retained with the old product description but flagged as “non-current”. A new record is created for that product with the new product description. This record is flagged as “current”. The field that is used to flag a record as current or non-current is called the “Current Indicator” field. Two additional fields (called “Effective Date” and “Expire Date”) are used to specify the date-range that the description is applicable.

Example Slowly Changing Dimensions Job



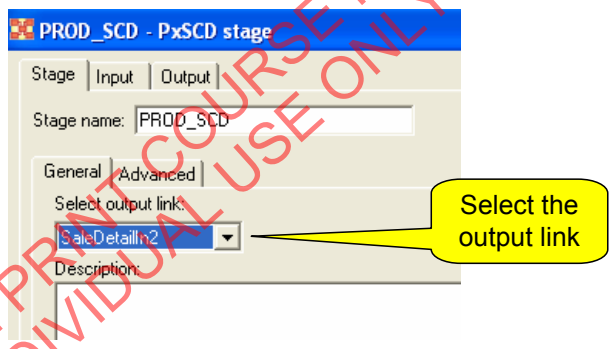
62

Working in the SCD Stage

- Five “Fast Path” pages to edit
- Select the Output link
  - This is the link coming out of the SCD stage that is not used to update the dimension table
- Specify the purpose codes
  - Fields to match by
    - Business key fields and the source fields to match to it
  - Surrogate key field
  - Type 1 fields
  - Type 2 fields
  - Current Indicator field for Type 2
  - Effective Date, Expire Date for Type 2
- Surrogate Key management
  - Location of State file
- Dimension update specification
- Output mappings

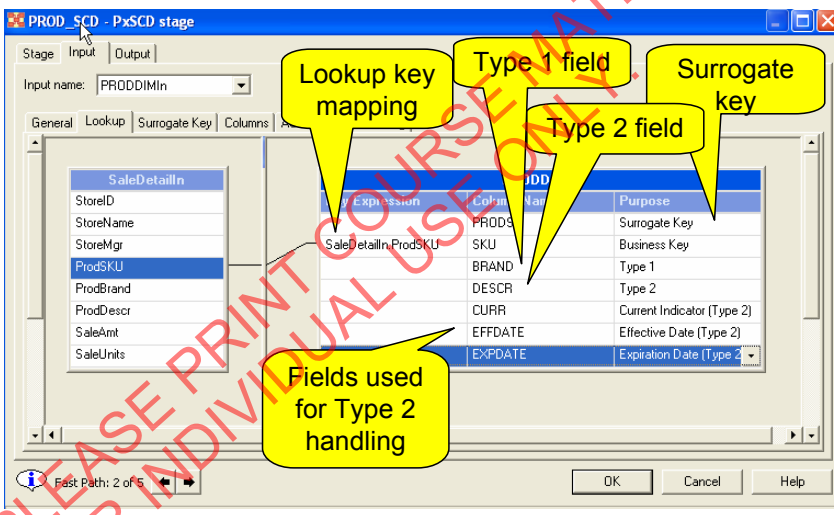
63

Selecting the Output Link



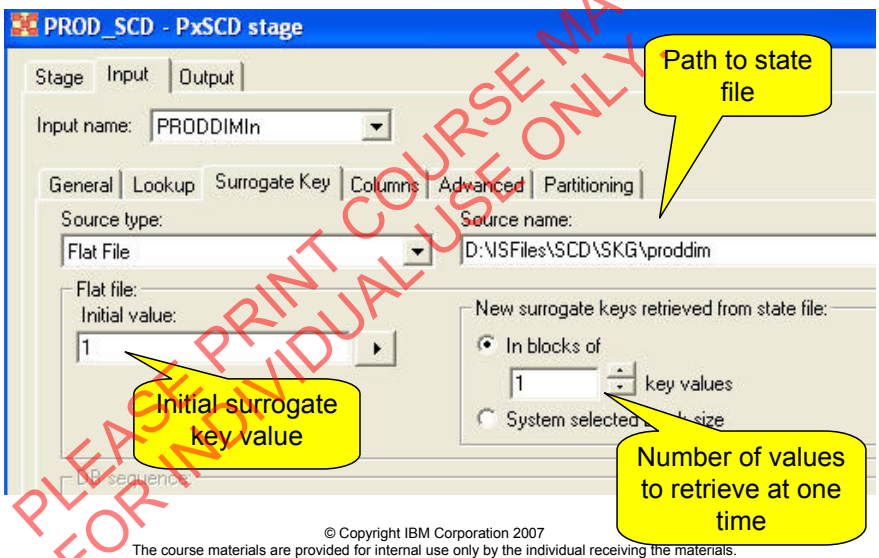
64

Specifying the Purpose Codes



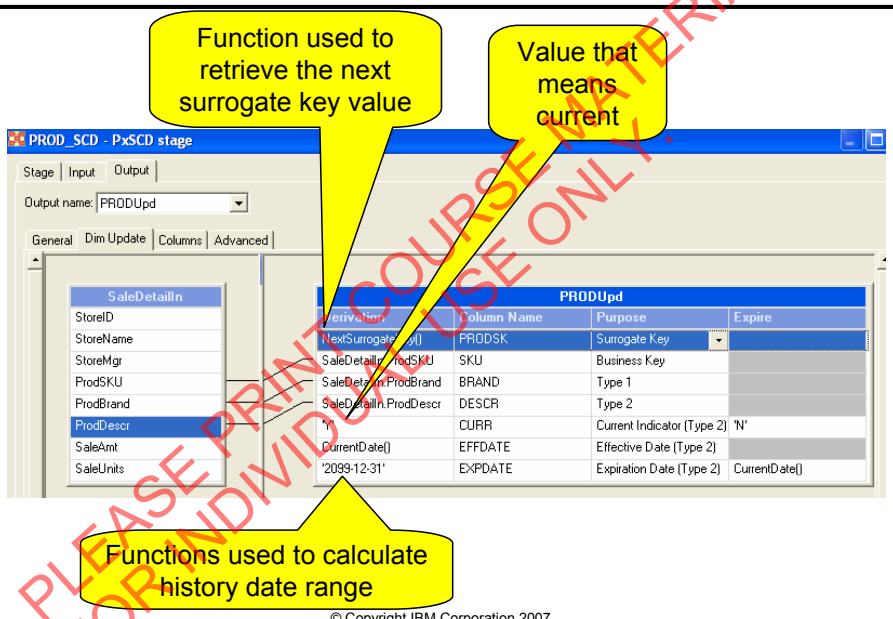
65

Surrogate Key Management



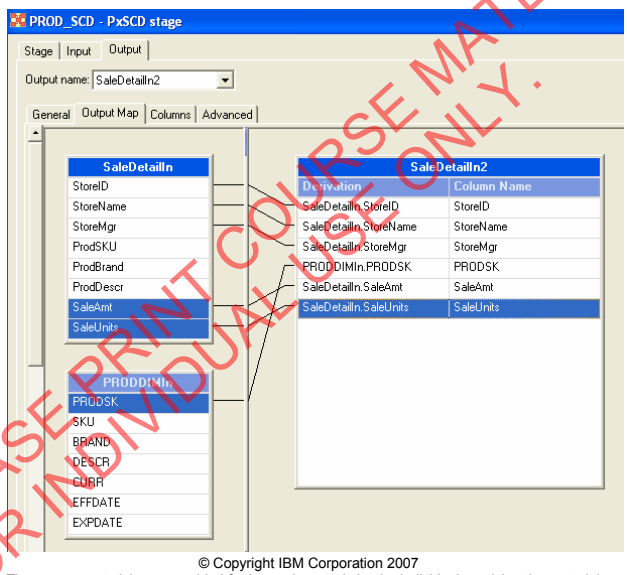
66

Dimension Update Specification



67

Output Mappings



68

Checkpoint

1. How many Slowly Changing Dimension stages are needed to process a star schema with 4 dimension tables?
2. How many Surrogate Key state files are needed to process a star schema with 4 dimension tables?
3. What's the difference between a Type 1 and a Type 2 dimension field attribute?
4. What additional fields are needed for handling a Type 2 slowly changing dimension field attribute?

69

Notes:

Write down your answers here:

1.

2.

Checkpoint solutions

1. Four SCD stages are needed. One for each dimension table. Each SCD stage does a lookup and update to the table.
2. Four Surrogate Key state files are needed. One for each dimension table. A separate state file is used for each.
3. Type 1 is a simple update. The value in the dimension record field is overwritten with the new value. Type two retains the value in a history record. A new record is created with the current value.
4. Three additional fields are needed: The Current Indicator is needed to flag whether a given record contains the current Type 2 value or an earlier value. The Effective Date and Expire Date fields are used to specify when the given record is applicable.

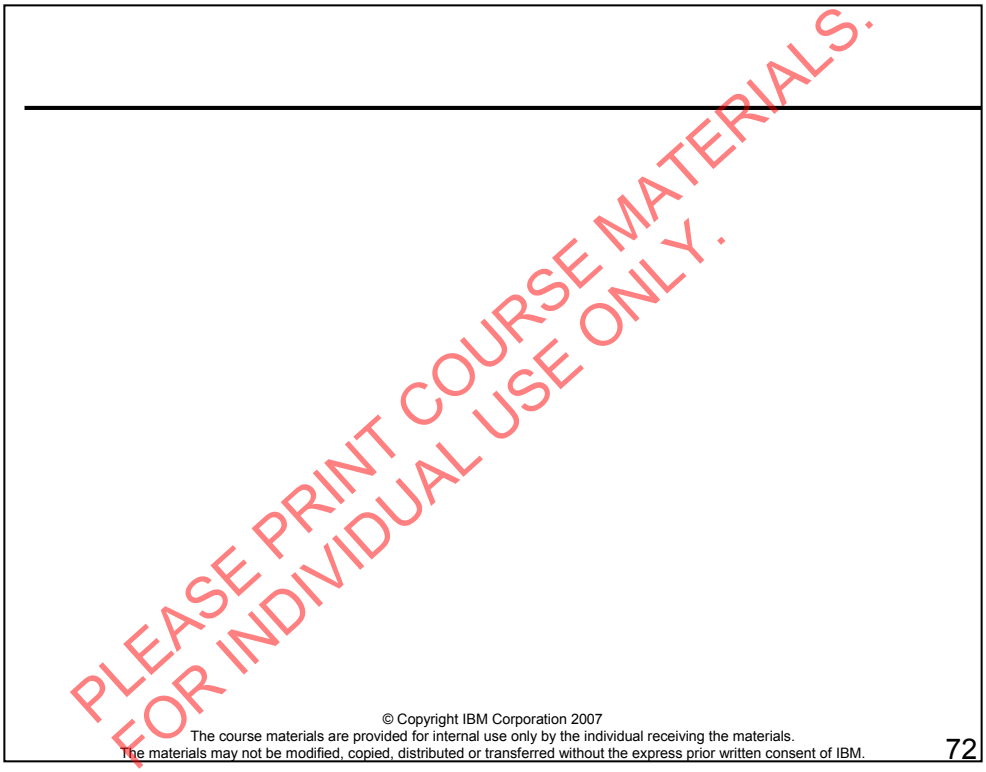
Unit summary

Having completed this unit, you should be able to:

- Design a job that creates a surrogate key source key file
- Design a job that updates a surrogate key source key file from a dimension table
- Design a job that processes a star schema database with Type 1 and Type 2 slowly changing dimensions

71

Notes:


PLEASE PRINT COURSE MATERIALS.
FOR INDIVIDUAL USE ONLY.
© Copyright IBM Corporation 2007
The course materials are provided for internal use only by the individual receiving the materials.
The materials may not be modified, copied, distributed or transferred without the express prior written consent of IBM.
72


IBM WebSphere DataStage
Special Topic 4: DataStage Utilities
PLEASE PRINT COURSE MATERIALS.
FOR INDIVIDUAL USE ONLY.
© Copyright IBM Corporation 2007
The course materials are provided for internal use only by the individual receiving the materials.
The materials may not be modified, copied, distributed or transferred without the express prior written consent of IBM.
4.0.3

Unit objectives

After completing this unit, you should be able to:

- Analyze the performance of a job
- Estimate the resources needed by a job

74

Notes:

Performance Analysis

75

Performance Analysis In the Past

- Use the Director monitor to watch the throughput (rows/sec) during a job run
- Compare job run durations
- Turn on APT_PM_PLAYER_TIMING and APT_PM_PLAYER_MEMORY to report player calls and memory allocation

How This Fails You…

- Long running jobs couldn't be watched for record throughput changes throughout the job run
- The job monitor didn't allow recording for playback
- Job monitor throughput rates included time waiting for data
- Couldn't determine what was happening on the machines

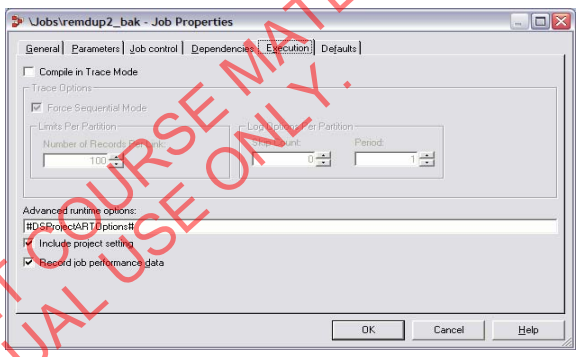
Performance Analyzer

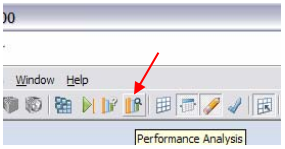
- Visualization tool that provides deeper insight into job runtime behavior
- Offers several categories of visualizations:
  - Record throughput (rows/sec)
  - CPU utilization
  - Job timing
  - Job memory utilization
  - Physical machine utilization
- Performance data to be visualized can be:
  - Filtered in selected ways, including
    - Hide startup processes
    - Hide license operators
    - Hide inserted operators
  - Isolated to selected stages (operators), partitions, and phases
- Charts can be saved and printed

77

Enabling Performance Data Recording

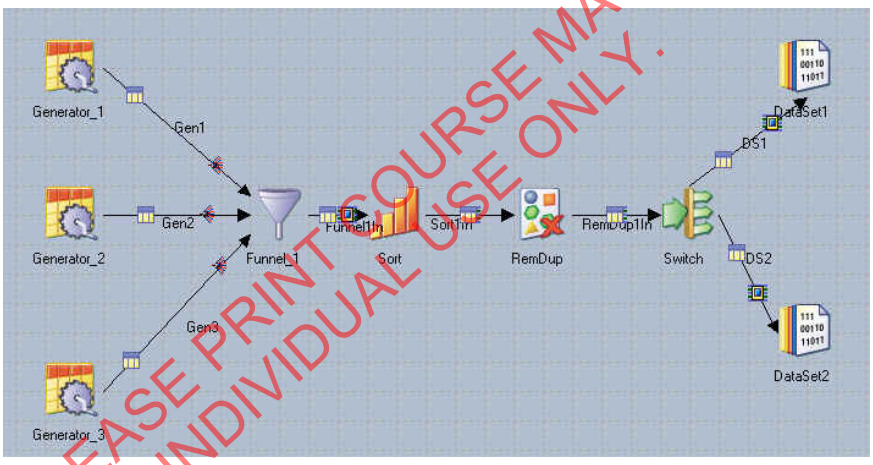
- Open the job in Designer
- Select Record job performance data in Job Properties
- Run your job. Performance collection has little impact on overall job performance
- To view the results, click the Performance Analysis icon in Designer





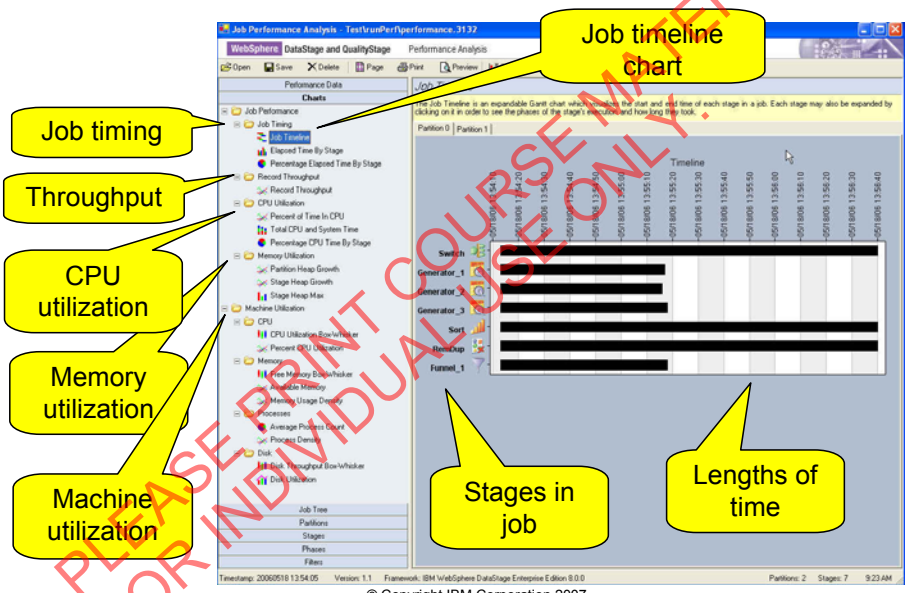
78

Example Job



79

Job Timeline Chart



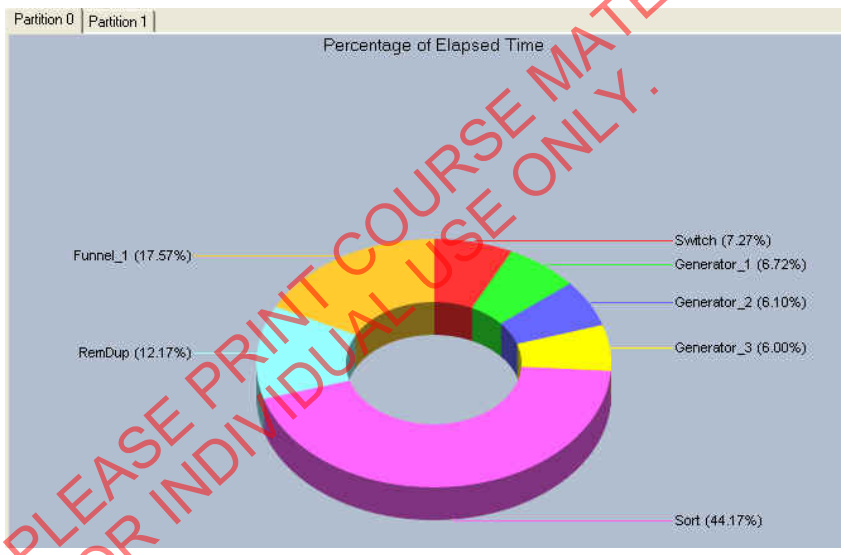
80

The Job Timeline chart breaks down the chart in terms of how long job processes take. Here we see how long the each player process takes. A player process is a process associated with an operator running on a node.

Expanding the Job Timeline Chart

81

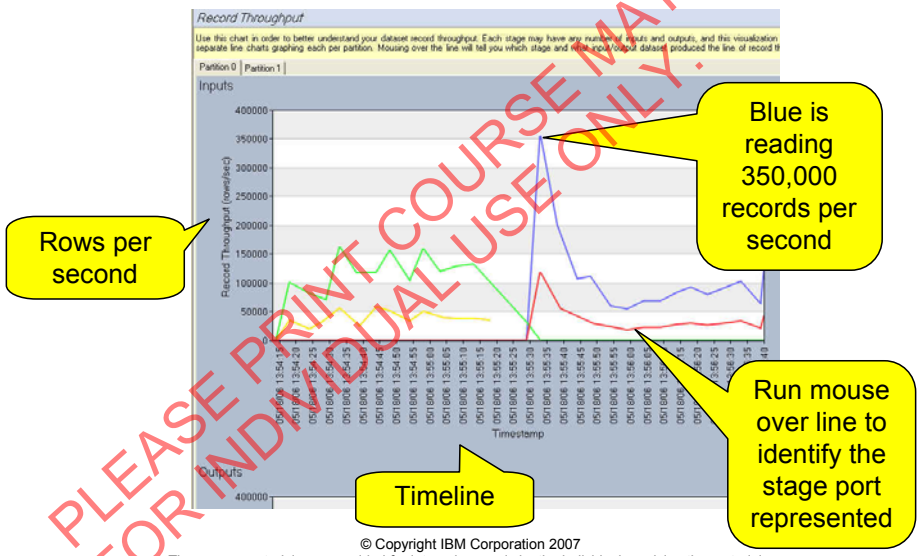
Another Job Timeline Chart



82

This shows the same information on a pie chart. This shows the amount of time of each process as a percentage of the total elapsed job time. Notice that most of the job time is spent sorting.

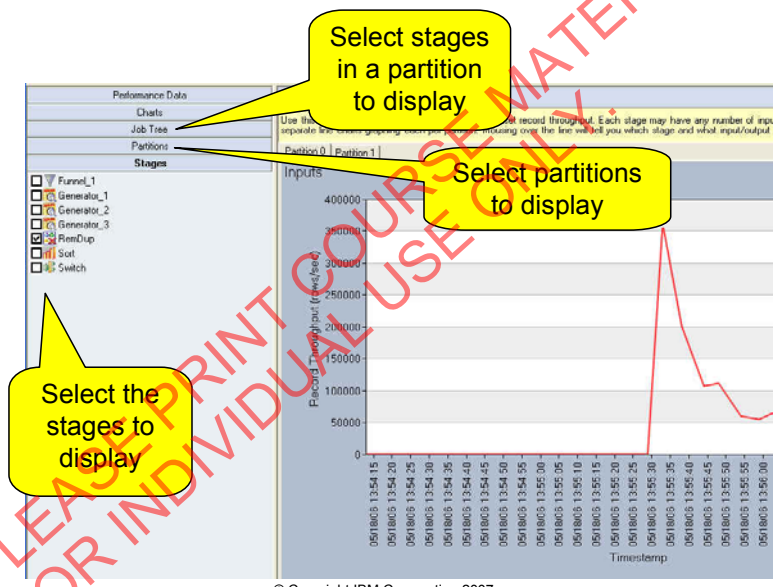
Record Throughput Chart



83

This chart shows the number of records read per second. Each line represents the throughput of a link (input or output) of a stage. For example, the blue line represents the first (and only) input link of the Remove Duplicates stage.

Displaying Selected Stages



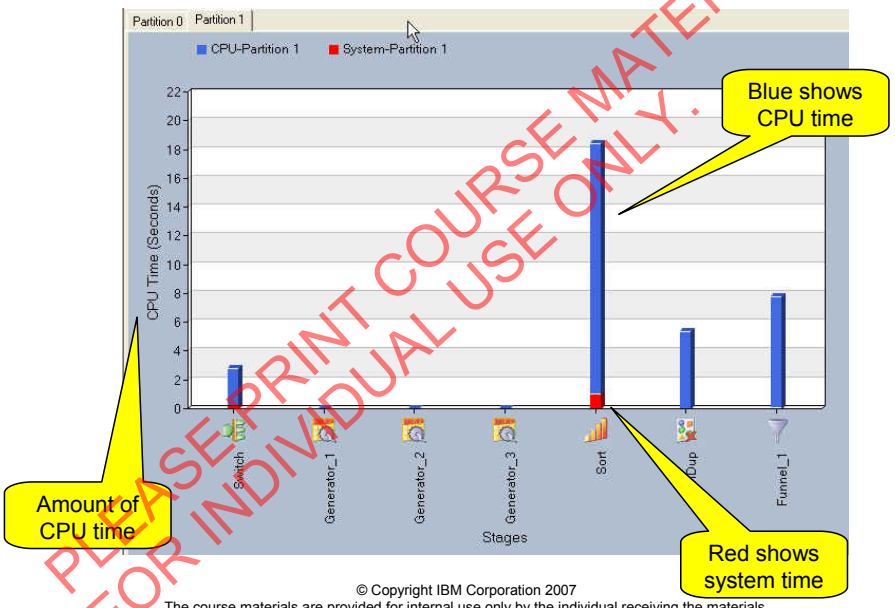
84

On the Stages tab you can select just the stages whose throughput you want to display. Here just the Remove Duplicates stage is displayed. Stage selection can be done for any chart. By default all stages are displayed.

You can also use the Job Tree and Partitions tab to select the results to display. The Job Tree tab allows you to select stages in partitions to display. The Partitions tab allows you to select partitions to display.

Similarly, the Phases tab (not shown) allows you to display what phases of a process to display: Initialization, RunLocally(), and Post processing.

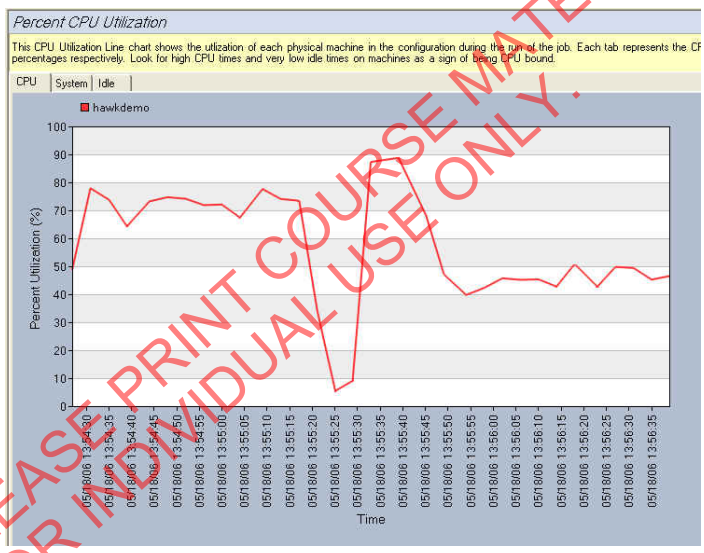
CPU Utilization Totals Chart



85

This chart is displaying the CPU time for Partition 1. Notice that the Generator stages are not using any CPU in this partition. This is because the Generator stages are running sequentially on Partition 0.

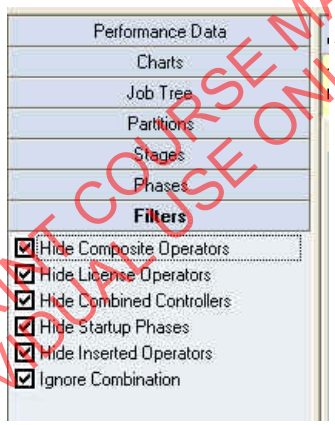
Machine Utilization - CPU



86

This chart shows the percentage of CPU usage through the job timeline.

Filters



87

By default all filters are entabled.

Resource Estimator

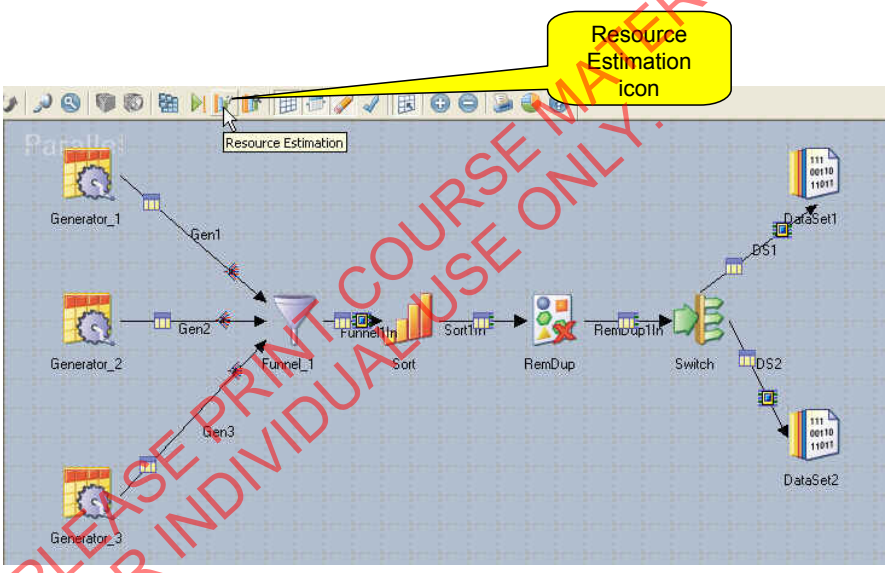
88

Resource Estimation

- To start:
  - Open job
  - Click the Resource Estimation icon in the Designer toolbar
  - Click Run to build statistics based on a job run
- Generate models
  - Static model: Computed worst-case scenarios of resource usage
  - Dynamic model: Computed from a sampling of the data
  - View resource estimates by stage
  - Compare model resource estimates
- Generate projections
  - View projection resource estimates

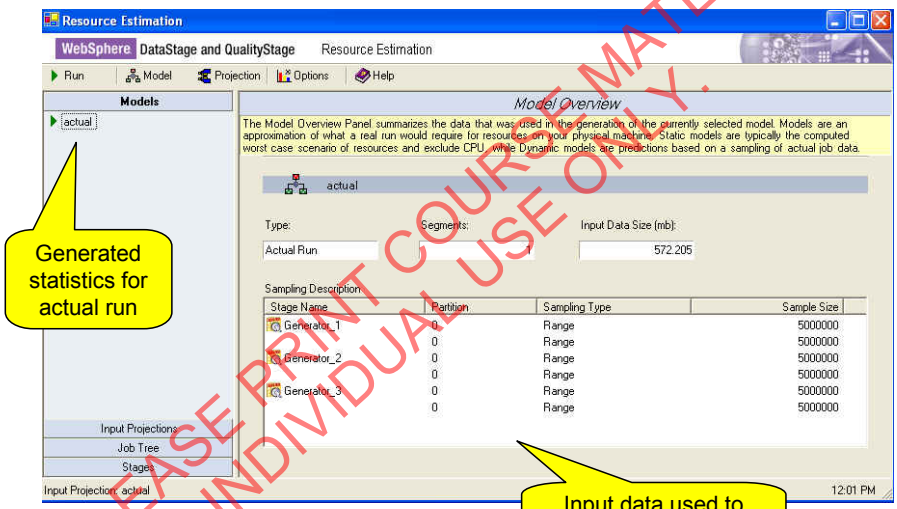
89

Example Job



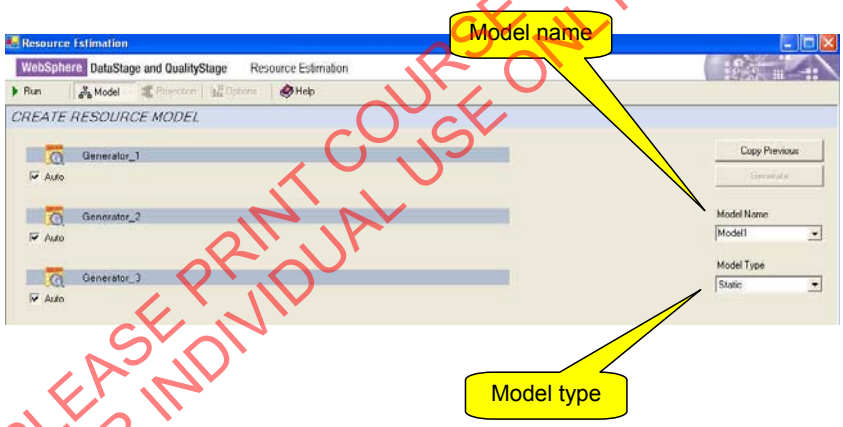
90

Resource Estimation Window



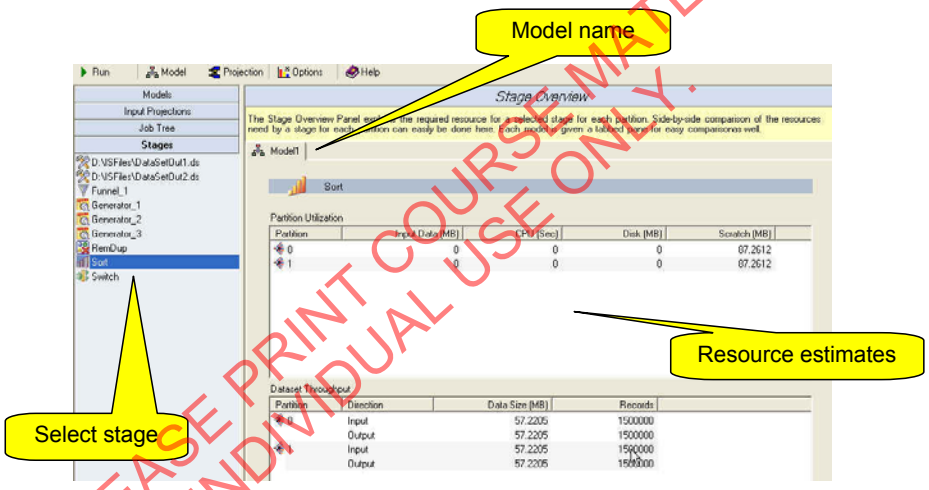
91

Model Tab



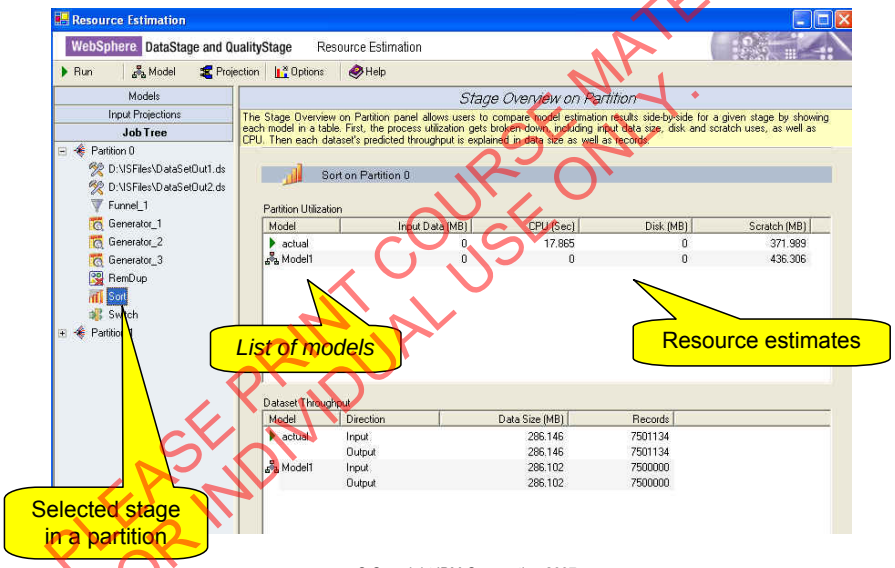
92

Model Estimates by Stage



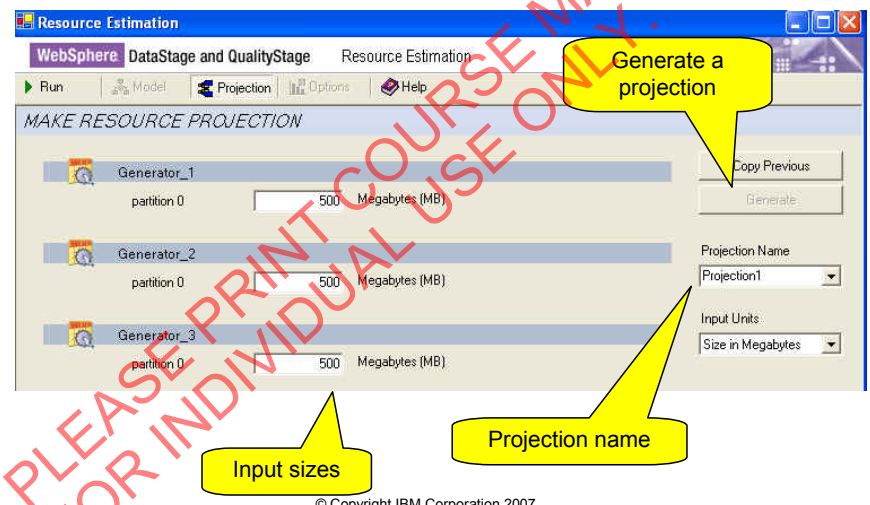
93

Comparing Model Estimates



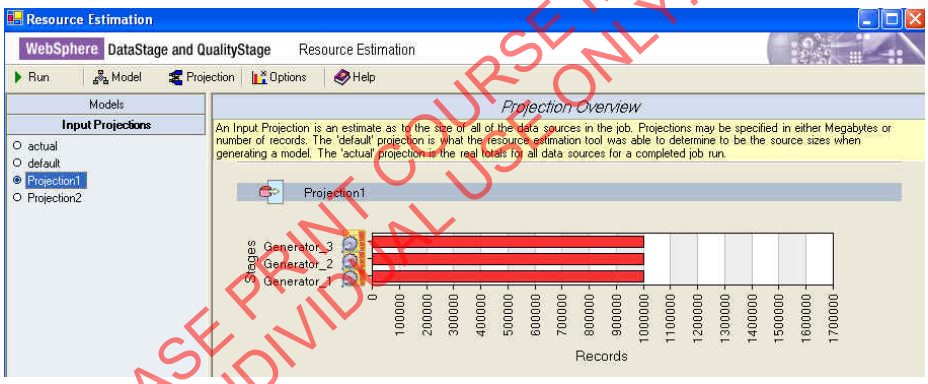
94

Projection Tab



95

Projection Estimates



96

Checkpoint

1. What are the five types of visualizations that can be created by the Performance Analyzer?
2. How do you enable the collection of performance data?
3. Describe the two types of models that can be generated by the Resource Estimator?

97

Notes:

Write down your answers here:

1.

2.

Checkpoint solutions

1. Record throughput (rows/sec). CPU utilization. Job timing. Job memory utilization. Physical machine utilization.
2. Click on the Execution tab in Job Properties. Select Record job performance data in Job Properties.
3. Static model: Computed worst-case scenarios of resource usage. Dynamic model: Computed from a sampling of the data

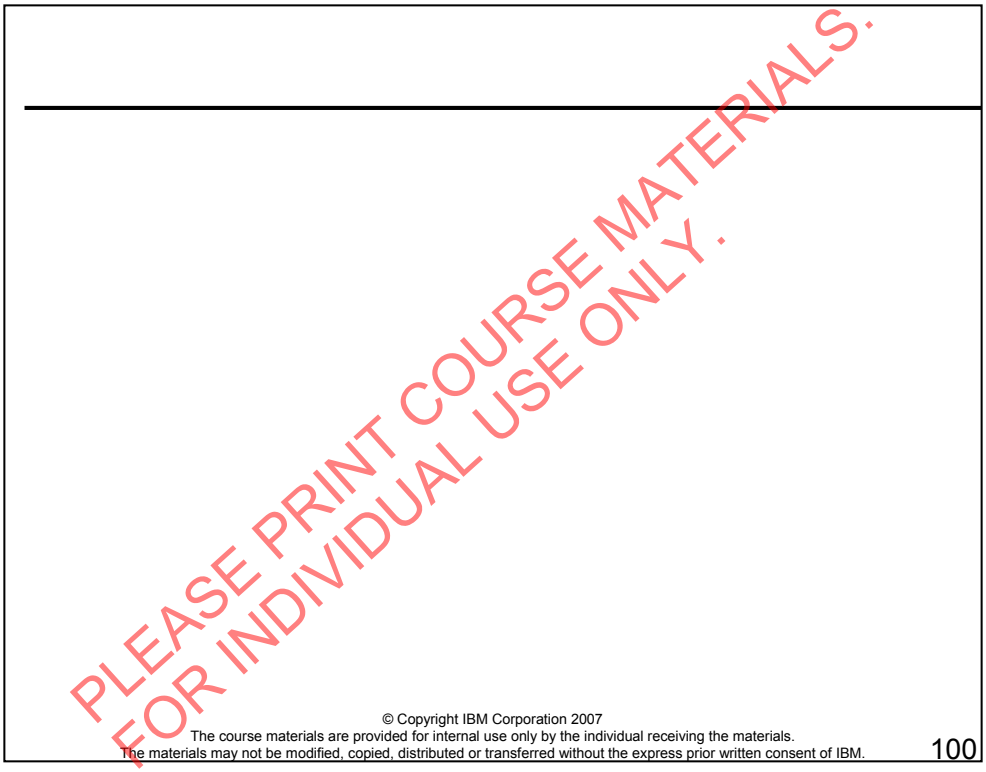
Unit summary

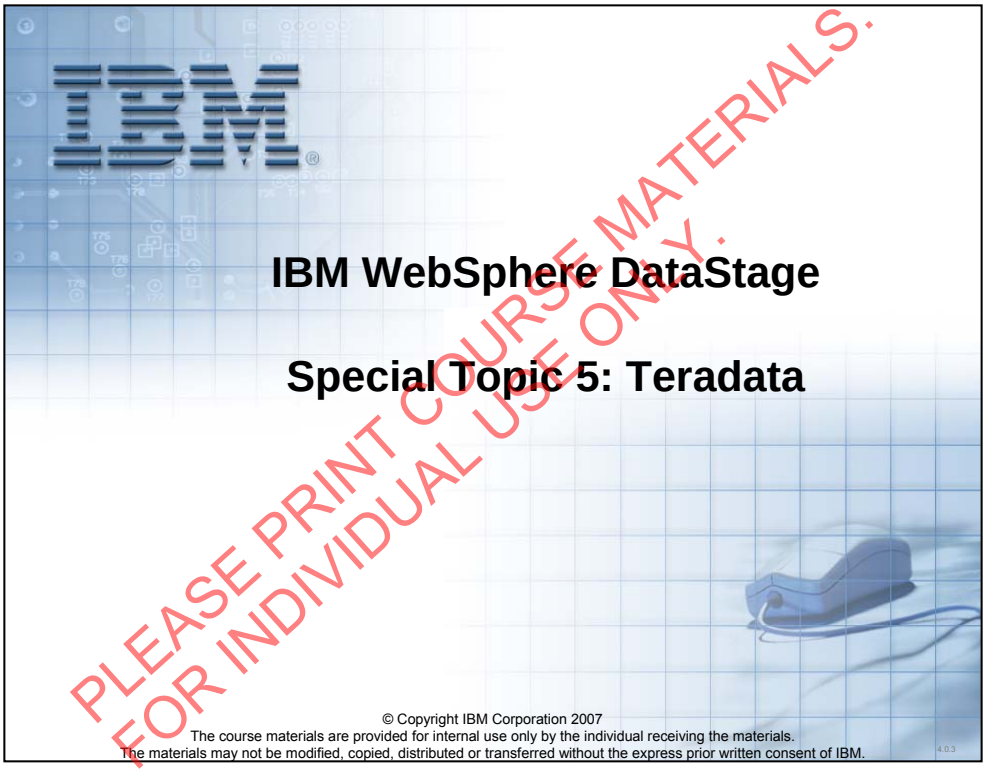
Having completed this unit, you should be able to:

- Analyze the performance of a job
- Estimate the resources needed by a job

99

Notes:


PLEASE PRINT COURSE MATERIALS.
FOR INDIVIDUAL USE ONLY.
© Copyright IBM Corporation 2007
The course materials are provided for internal use only by the individual receiving the materials.
The materials may not be modified, copied, distributed or transferred without the express prior written consent of IBM.
100


IBM WebSphere DataStage
Special Topic 5: Teradata
PLEASE PRINT COURSE MATERIALS.
FOR INDIVIDUAL USE ONLY.
© Copyright IBM Corporation 2007
The course materials are provided for internal use only by the individual receiving the materials.
The materials may not be modified, copied, distributed or transferred without the express prior written consent of IBM.
4.0.3

Unit objectives

After completing this unit, you should be able to:

- Explain what Teradata is
- Invoke Teradata utilities
- Use the Teradata stages in parallel jobs

102

Notes:

What is Teradata?

103

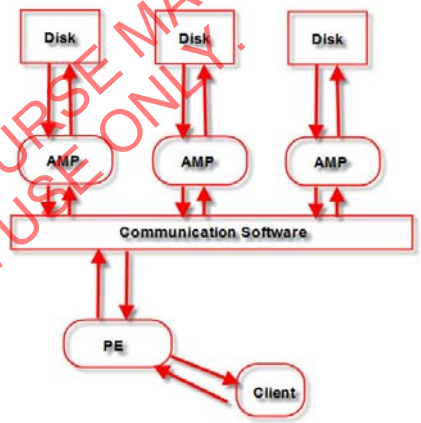
Teradata RDBMS

- Runs on SMP and MPP systems
- High volume database
- Parallel database processing
- Utilities for reading and writing to the RDBMS
  - Load utilities
    - FastLoad
    - MultiLoad
    - TPump
  - Read utility
    - FastExport

104

Teradata Architecture

- Disks contain database tables
  - Rows of tables are distributed across disks
- AMPs (Access Module Processors) are VPROCs (Virtual Processors) that manage the database
  - Locking
  - Joining
  - Read and Write functions
- PEs (Parsing Engines) are VPROCs that parse SQL sent to the Teradata Server by clients (e.g., DataStage)
  - Check syntax
  - Create optimized Parse Tree
  - Generate task Steps
  - Dispatch Steps to AMPs
- VPROCs run under PDE (Parallel Database Extensions)



105

The communication layer consists of BNET communications software supporting TCP/IP communication between the AMP processors. It supports broadcast, multi-cast, and point-to-point communication.

An **AMP**, or unit of parallelism, "owns" a portion of the database. Multiple AMPs reside on a single 2-CPU SMP. Therefore, the Teradata Database doesn't rely on the hardware platform for parallelism, scalability, reliability, or availability. These capabilities are inherent in the database architecture and are independent of the operating system and hardware configuration.

AMPs are one of two types of **virtual processors** (**VPROCs**). The second type of VPROC is **Parsing Engines (PE),** which break up a request or query into manageable pieces and distribute the work to the AMP VPROCs for processing. Multiple PEs can also exist on a single node. It's important to note that each PE has access to each AMP, which allows for complete parallel processing of each request.

PEs are similar in concept to a database optimizer.

Client Access to Teradata

- Utilities access
  - Teradata has a set number of slots (15 by default) in which the Teradata utilities can run concurrently
    - A DataStage job that invokes a utility when no slot is available will abort
      - The MultiLoad stage supports tenacity, which queues the process for retry when slots aren't available
  - Teradata utilities for loading tables
    - Vary in whether they take up a utility slot
    - Vary in the number of target tables they can load in a single run
    - Vary in their update and load capabilities
    - Vary in whether the table is locked for use by other users
    - Vary in their performance
  - Teradata utility for reading from tables
    - Only one, so no choice here
- Non-utilities access
  - ODBC: Access through ODBC driver
  - CLI (Call Level Interface)
    - Programmed access

106

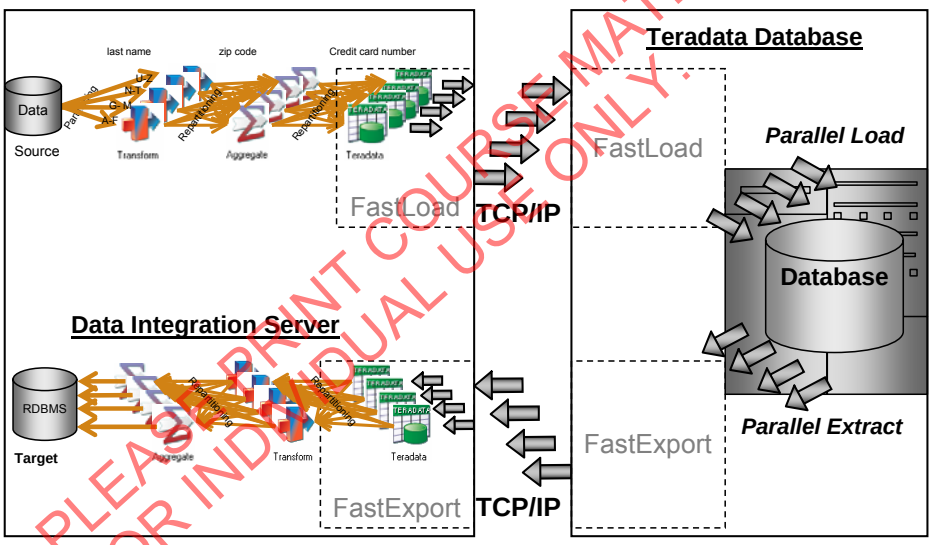
Teradata Utilities

- Utilities for loading tables
  - FastLoad (Takes a utility slot)
    - High performance loading to a single, empty table
    - No updates or upserts
    - Locks table
    - Supports checkpoint restart
  - MultiLoad (Takes a utility slot)
    - High performance inserts, updates, and deletes
    - Can update multiple tables per run
    - Table need not be empty
    - Locks table
    - Supports checkpoint restart
  - TPump (Does not take a utility slot; runs in the background)
    - Supports inserts, updates, upserts, and deletes
    - Can update multiple tables per run
    - "Trickle feed": Like a pump the data is fed to the target in a slow, steady stream that it can handle
      - Flow of data can be throttled to a specified number of updates per minute
    - Supports checkpoint restart
- Utilities for reading from tables
  - FastExport (Takes a utility slot)
    - Extracts large amounts of distributed data

107

Parallel Support For Teradata

- **EE drives FastLoad and FastExport in parallel**



108

DataStage Teradata Installation

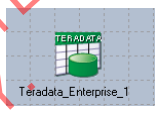
- Teradata Utilities Foundation must be installed on all nodes that will run DataStage parallel jobs
- Set up a Teradata database user
  - Must be able to create and delete tables, and to insert, and delete data
  - The database for which you create this account requires at least 100 MB of PERM space and 10 MB of SPOOL.

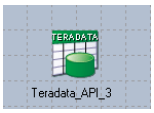
Teradata Stages

110

Teradata Stages

- Four Teradata stages
  - Teradata Enterprise
  - Teradata MultiLoad
  - Teradata API
  - Teradata Load
- All support one input link and/or one output link
  - Output link reads from database table(s) or load files
  - Input writes to database table(s)
- How they differ
  - Teradata utilities used / not used
  - Performance
  - Whether they take up a utility slot
  - Update (upsert) capabilities
  - Target one table or more than one per run
  - Whether table is locked

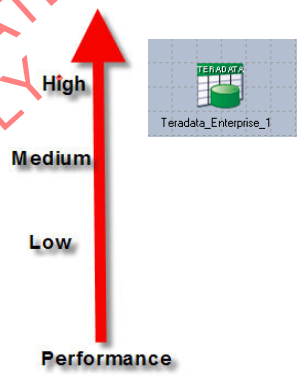




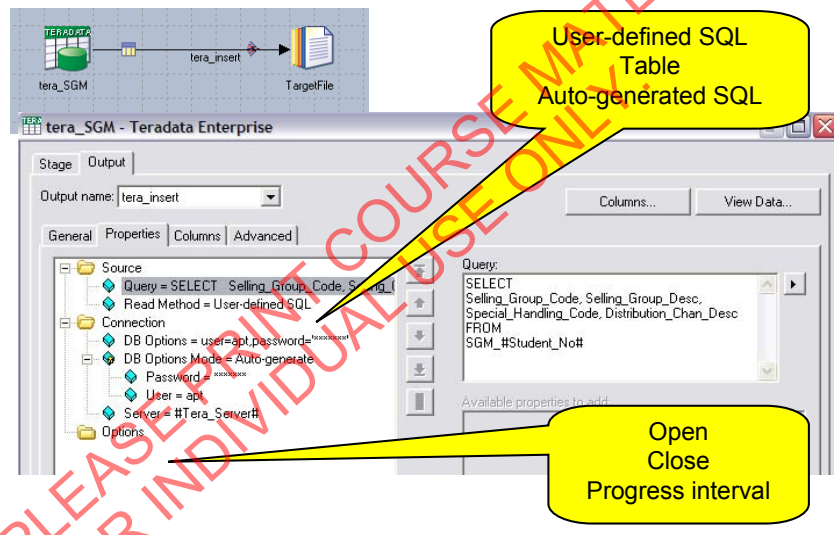
111

Teradata Enterprise Stage

- Optimized for high-volume, parallel reads and writes
- Reads using FastExport interface
- Writes using FastLoad interface
  - Can only insert or append rows into one table per stage
  - No update or upsert of existing rows
  - Locks table
- Takes up one utility slot when reading or writing
  - True no matter how many configuration file nodes the job is running on
- Supports OPEN and CLOSE commands
- Unlike FastLoad Teradata Utility, supports appending mode
  - Although FastLoad can only write to a single, empty target table, the Enterprise stage has been enhanced to support writing to a non-empty table



Teradata Enterprise Stage Read

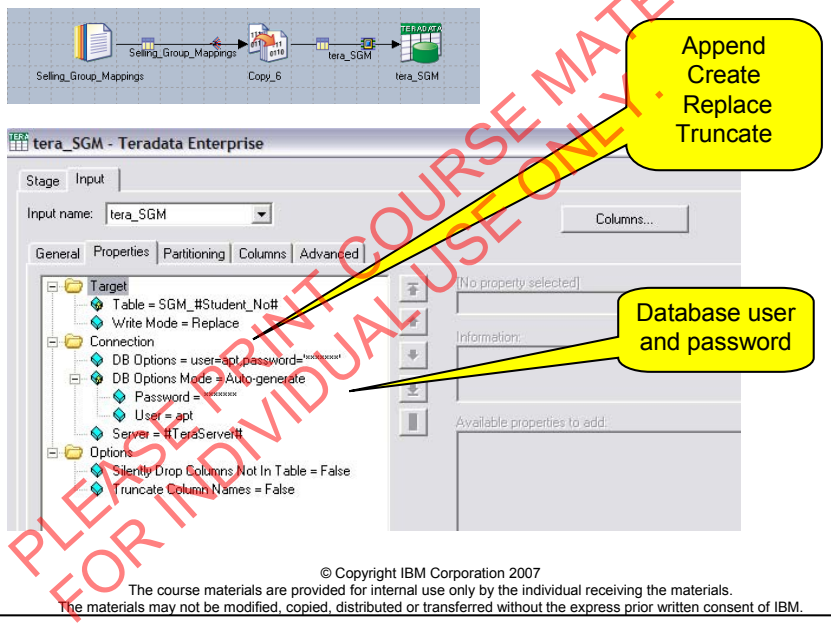


113

Progress interval is the number of rows per partition before progress is reported.

Open / Close are commands to executed before the table is opened / after processing on the table is done.

Teradata Enterprise Stage Write



114

Append adds rows to an existing table.

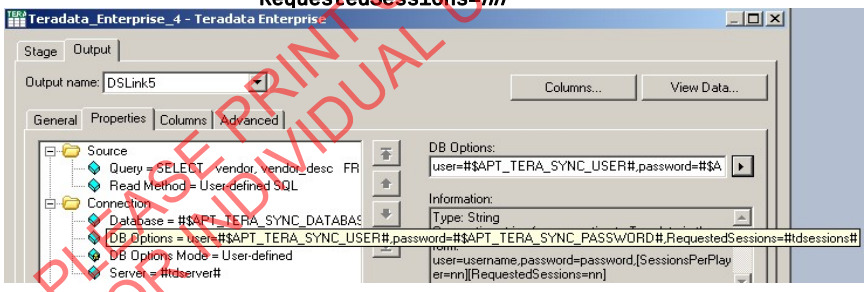
Create creates a new table before loading it. An error occurs if the table already exists.

Replace drops an existing table and then creates a new table. If the table doesn't exist, a new table is created.

Truncate empties the table before writing to it.

Teradata Enterprise Stage DB Options

- For Teradata instances with a large number of AMPs, it may be necessary to set the optional **SessionsPerPlayer** and **RequestedSessions** in the **DB Options** string
  - Good to parameterize these settings
  - Syntax is:
    - `user=[user],password=[password],SessionsPerPlayer=nn, RequestedSessions=nn`



115

Teradata Enterprise Sessions

- **RequestedSessions**
  - Total number of distributed connections to the Teradata database
    - Default equals the number of Teradata AMPs
  - Set between 1 and number of AMPs
- **SessionsPerPlayer**
  - Number of connections each player will have to Teradata.
    - Indirectly determines the number of players (degree of parallelism).
  - Default is 2 sessions / player
  - Number selected should be such that **SessionsPerPlayer * number of nodes * number of players per node = RequestedSessions**
  - Setting the value of *SessionsPerPlayer* too low on a large system can result in so many players that the job fails due to insufficient resources.

116

-These options provide a means for the developer to tune the Teradata stage such that the number of players matches the number of available AMPs.

-Find out how many AMPs are configured by checking with the DBA.

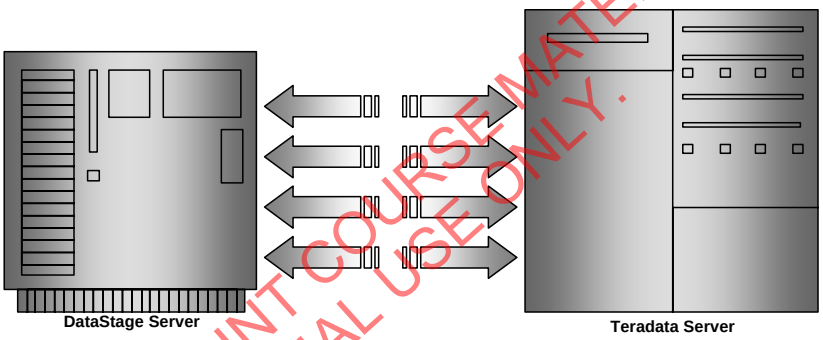
-These settings should be experimented with to obtain optimum performance.

-For example, sometimes 1 player per AMP will be optimal. Sometimes 2 players per AMP will be optimal.

-Several permission checks are performed at startup.

-To speed up the start time of the load process try setting the APT_TERA_NO_PERM_CHECKS environment variable to bypass permission checking on several system tables that need to be readable during the load process.

Teradata *SessionsPerPlayer* Example



- MPP with 4 TPA nodes
- 4 AMP's per TPA node

Example Settings

Configuration File	Sessions Per Player	Total Sessions
16 nodes	**1**	**16**
8 nodes	**2**	**16**
8 nodes	**1**	**8**
4 nodes	**4**	**16**

117

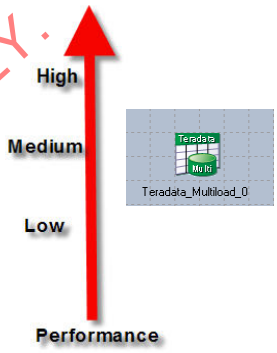
-TPA = Trusted Parallel Application

Partitioning and Teradata Enterprise

- If degree of parallelism chosen by the Teradata Enterprise stage does not match the degree of parallelism of the upstream operators, the job will abort
  - Teradata Enterprise degree of parallelism determined by VPROCs, RequestedSessions, SessionsPerPlayer
  - Upstream operators degree of parallelism set by $APT_CONFIG_FILE and/or node pool specification
- In general, do not specify “Auto” partitioning as the input to a Teradata Enterprise Stage
  - Instead, choose **Round Robin** to distribute rows evenly

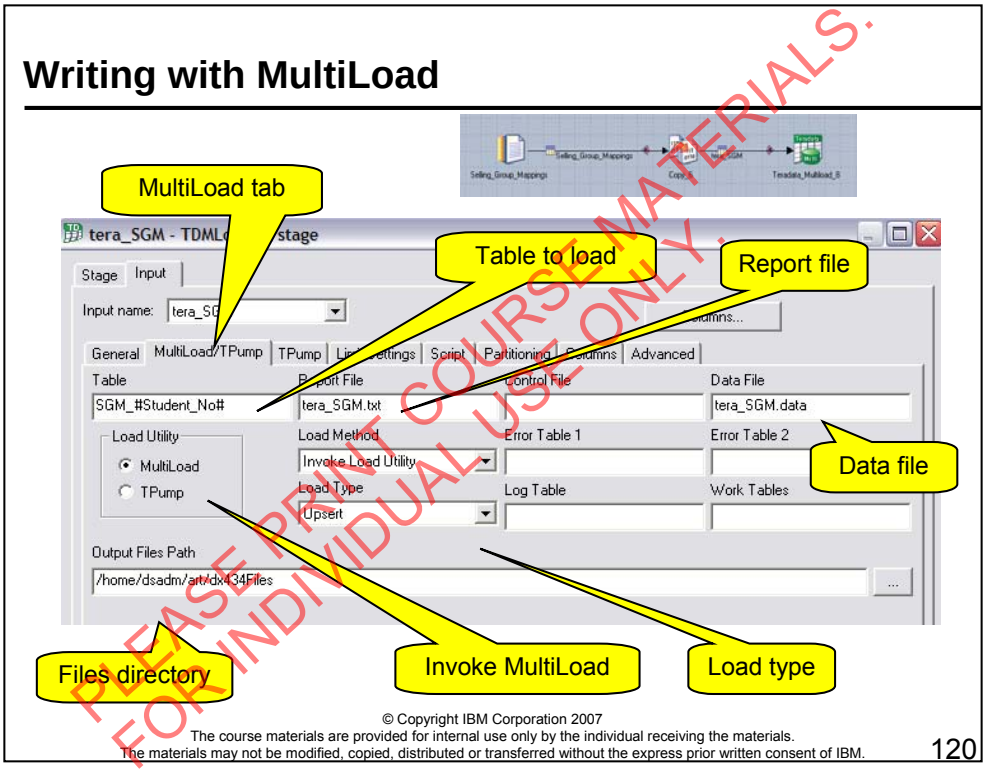
Teradata MultiLoad Stage Using MultiLoad

- Writes using MultiLoad interface
- Supports moderate volumes of data
- Must be run Sequentially
- Can target multiple tables in a single run
- Supports Checkpoint Restart
- Takes up one utility slot
  - Supports Tenacity (queues MultiLoad if no utility slots are available)
- Supports updates
- Takes full table locks
  - Tables are locked for other activity
  - This prevents parallel writes to the table
- Optionally writes to a data file before loading to the target
- Not intended for reads
  - Use Enterprise stage to read



119

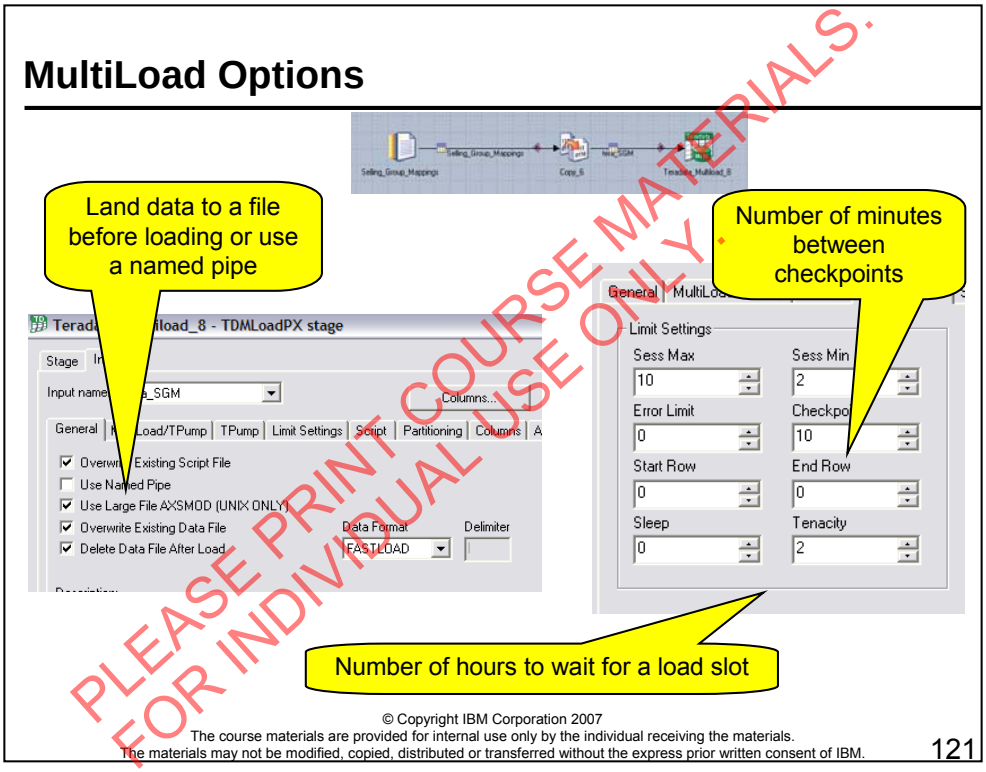
Although the MultiLoad stage can be used to read data using the FastExport interface, it is recommended that the Teradata Enterprise stage be used to read data. The Teradata Enterprise stage can read in parallel, but the MultiLoad stage cannot read in parallel.



Choose to write using either the MultiLoad utility or TPump. These are mutually exclusive.

Supported load types are: Insert, Update, Upsert, Delete, and Custom.

MultiLoad requires four additional special tables besides the target table to be loaded. Two error tables are needed to store conversion and constraint violations and the like; a work table is needed for receiving and sorting data; and a log table is needed to log checkpoints for restart. You have the option to name these tables, but if you do not, they have default names.



You can specify the minimum and maximum number of sessions to use. The maximum that can be specified is equal to the number of AMPs. If nothing is specified, then the maximum number is used.

Checkpoint is the number of minutes between checkpoints.

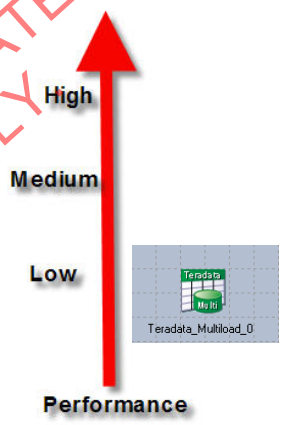
You can specify the maximum number or percentage of errors you will accept during the load.

Sleep tells MultiLoad how long to sleep before trying to log onto the system.

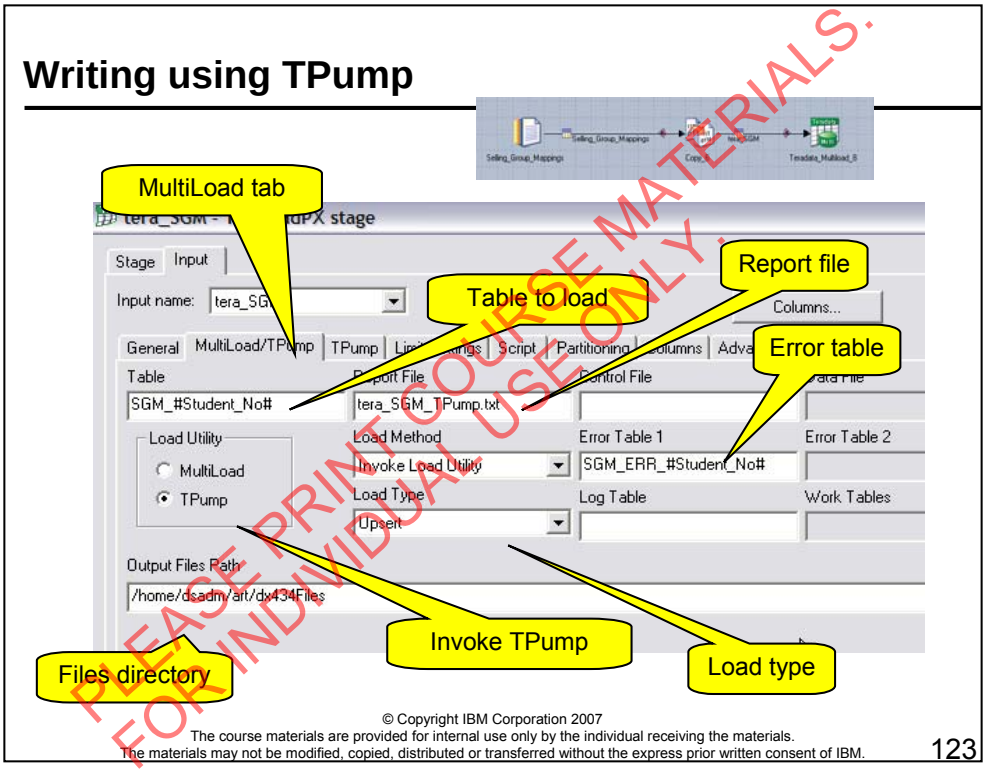
Tenacity is the number of hours to wait for a load slot.

Teradata MultiLoad Stage Using TPump

- Writes using TPump interface
- Does not take up a utility slot
- Can target only one table in a single run
- Supports Updates
- Takes row level locks
  - So other activity to table can occur concurrently
- Use when writing relatively small amounts of data



122

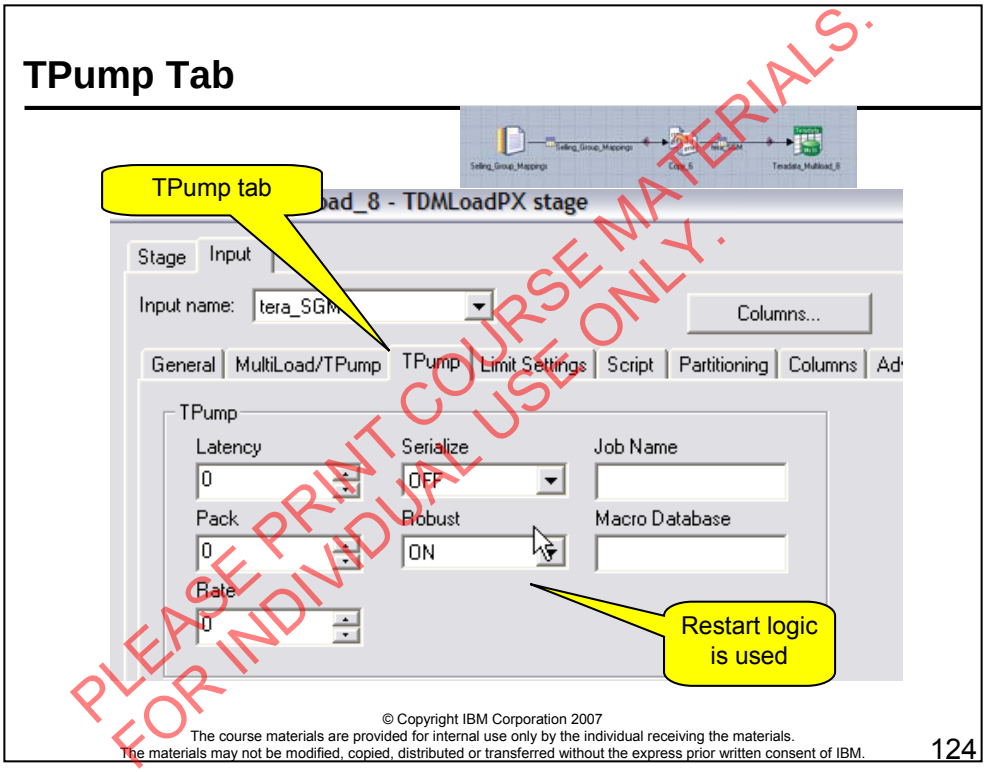


Choose to write using either the MultiLoad utility or TPump. These are mutually exclusive.

Supported load types are: Insert, Update, Upsert, Delete, and Custom.

Here, Data File is disabled because the Used Named Pipe option was selected on the General tab.

Unlike, MultiLoad only one error table is used by TPump.



The following properties are included on the **TPump** tab:

Latency: The maximum number of seconds that a record resides in the TPump buffers before the buffers are flushed. The default value of 0 indicates the Teradata default should be used.

Pack: The number of statements to pack into a multiple-statement request. The default value of 0 indicates the Teradata default should be used.

Rate: The number of statements per minute to be sent to the Teradata database. The default value of 0 indicates the Teradata default should be used.

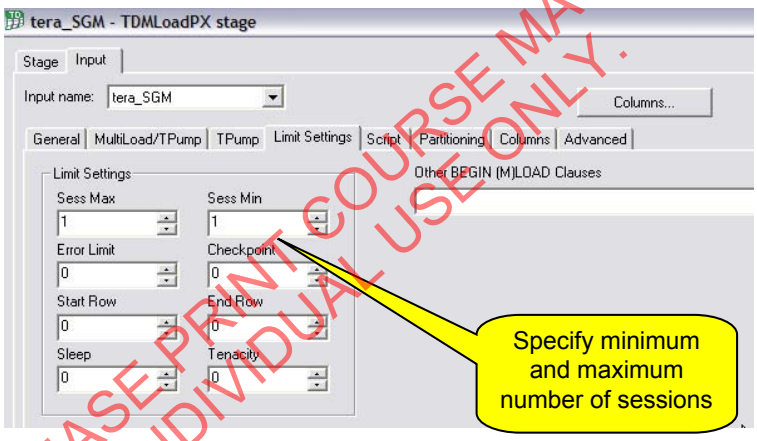
Serialize: The way multiple operations on a given row are guaranteed to occur. If **On** is selected, the multiple operations occur serially. The default is **Off** selected.

Robust: The restart logic to be used. If **Off** is selected, TPump uses simpler but less reliable restart logic. The default is **On** selected.

Job Name: A unique identifier assigned to the TPump environment variable SYSJOBNAME. TPump truncates the identifier to 16 characters. If not provided, TPump uses the Teradata default.

Macro Database: The name of the database that is to contain any macros built or used by TPump. If not provided, TPump uses the Teradata default.

Limit Settings



125

You can specify the minimum and maximum number of sessions to use. The maximum that can be specified is equal to the number of AMPs. If nothing is specified, then the maximum number is used.

Checkpoint is the number of minutes between checkpoints.

You can specify the maximum number or percentage of errors you will accept during the load.

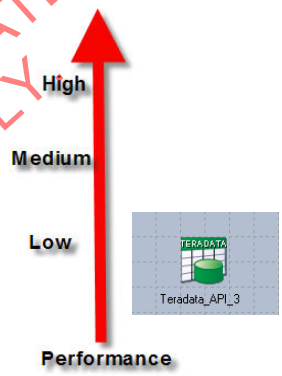
Sleep tells MultiLoad how long to sleep before trying to log onto the system.

Tenacity is the number of hours to wait for a load slot.

TPump has some LOAD options that are unique to TPump. These can be specified in the “Other BEGIN (M)LOAD Clauses” box. This includes RATE which enables the flow of data to be throttled. See the Teradata reference manual for details.

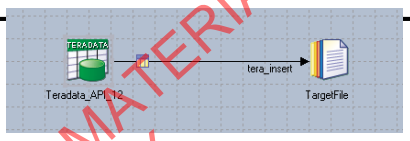
Teradata API Stage

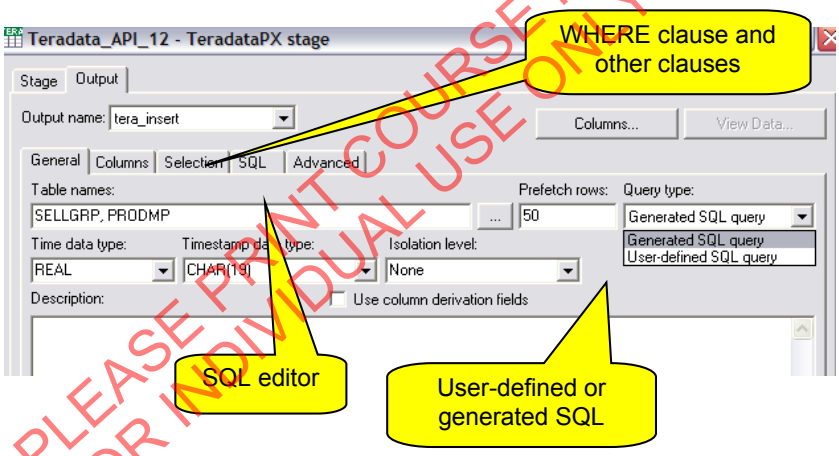
- Uses CLI interface
  - Same interface as BTEQ (Basic Teradata Query Utility)
- Input link writes to a Teradata table
- Output links read from Teradata tables
- Intended for low-volume reads and writes
  - Cannot read in parallel
- Doesn't take up a load utility slot
- Read from one or more tables using generated or user-defined SQL
- Can be useful for jobs with multiple lookups, because it doesn't impact the utility slot count



126

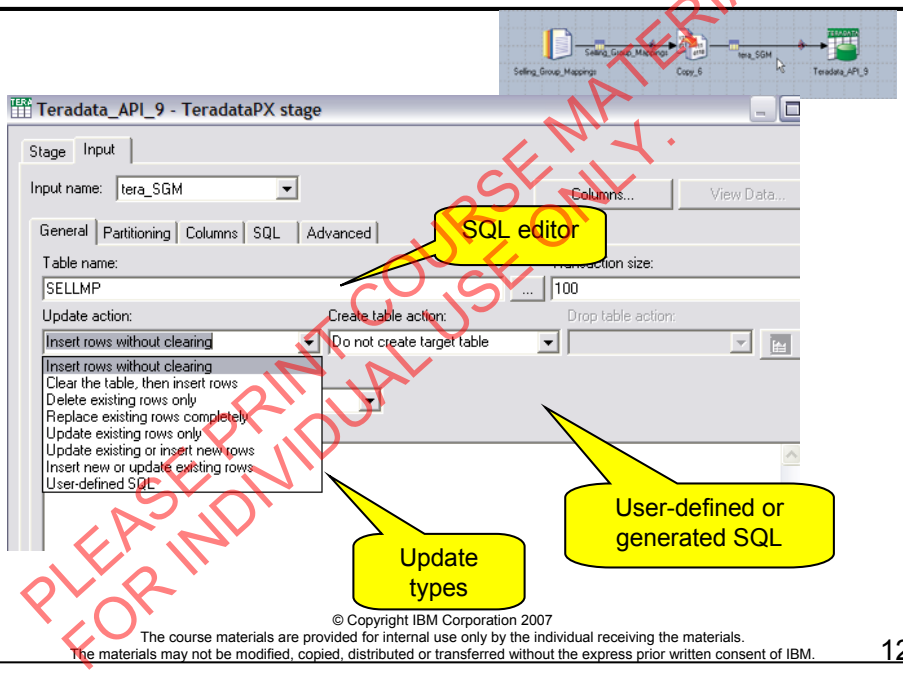
Teradata API Stage Read





127

Teradata API Stage Write



128

Teradata Environment Variables

- APT_TERA_NO_PERM_CHECKS
  - Set to bypass permission checking on several system tables that need to be readable during the load process.
  - Can speed up the load process
- APT_TERA_64K_BUFFERS
  - Setting this enables 64k buffer transfers (32k is the default). May improve performance in some cases.
- APT_TERA_SYNC_DATABASE
  - Used to set the database which is used for the terasync table.
- APT_TERA_SYNC_USER
  - Used to set the user that is used to create and write to the terasync table
- APT_TERA_SYNC_PASSWORD
  - The password for the user identified by APT_TERA_SYNC_USER
- APT_TERA_STALL_COUNT
  - The number of times a buffer read/write will be attempted after a stall condition before exiting with an error
- APT_TERA_STALL_DELAY
  - The number of seconds to wait between buffer reads/writes when a stall is detected

Checkpoint

1. List three Teradata utilities for loading tables.
2. List a Teradata utility used for reading from tables.

130

Notes:

Write down your answers here:

1.

2.

Checkpoint solutions

1. FastLoad, MultiLoad, TPump
2. FastExport

131

Unit summary

Having completed this unit, you should be able to:

- Explain what Teradata is
- Invoke Teradata utilities
- Use the Teradata stages in parallel jobs

132

Notes: