1.01 Aggregation using Sorted Input

Purpose

Aggregation using Sorted Input

Usage

Improve the performance of mappings that perform aggregation by using the "Sorted Input" option when source data can be sorted according to the Aggregator's "Group By" ports.

Download

m_AGG_Sorted_Input_v61.XML
m_1_01_AGG_Sorted_Input_v711.XML

Challenge Addressed
In a typical PowerCenter mapping that performs aggregation \textit{(without} the Sorted Input option), the Informatica server must read the entire data source before it begins performing calculations, in order to ensure that it has received every record that belongs to each unique group. While the server is optimized to perform aggregate calculations, the time required to read the entire data source can be a bottleneck in mappings that load very large amounts of data.

\textbf{Overview}

In a mapping that uses Sorted Input, the Informatica server assumes that all data entering an Aggregator transformation are already sorted into groups that correspond to the Aggregator’s "Group By" ports. As a result, the server does not have to read the entire data source before performing calculations. As soon as the Aggregator detects a new unique group, it performs all of the calculations required for the current group, and then passes the current group’s record on to the next transformation. Selecting Sorted Input often provides dramatic increases in Aggregator performance.

\textbf{Implementation Guidelines}

In order for Sorted Input to work, you must be able to sort the data in your source by the Aggregator’s Group By columns.

The key concepts illustrated in this mapping template can be found in two transformation objects, the Source Qualifier transformation (\texttt{SQ\_ORDER\_ITEMS}) and the Aggregator transformation (\texttt{agg\_CALC\_PROFIT\_and\_MARGIN}):

\begin{itemize}
  \item \texttt{SQ\_ORDER\_ITEMS} contains a SQL Override statement that pulls data from the \texttt{ORDER\_ITEMS} table. The select statement in this SQL Override contains an \texttt{ORDER BY} clause that orders the source data by the \texttt{ITEM\_ID} column. In addition, on the \textit{Properties} tab of \texttt{SQ\_ORDER\_ITEMS}, the "Number of Sorted Ports" option is set to "1." Configuring the Source Qualifier in this way prepares the data for Sorted Input aggregation.

  \item In \texttt{agg\_CALC\_PROFIT\_and\_MARGIN}, the \texttt{ITEM\_ID} port is selected as the "Group By" port in the transformation (the number of "Group By" ports must correspond to the "Number of Sorted Ports" indicated in the Source Qualifier object). In addition, on the \textit{Properties} tab of \texttt{agg\_CALC\_PROFIT\_and\_MARGIN}, the "Sorted Input" option is selected.
\end{itemize}
No other configuration is required in order for Sorted Input functionality to work in this mapping. When a session is created to implement this mapping, it will automatically detect that the Sorted Input option has been selected.

Please refer to the metadata descriptions in the mapping `m_AGG_Sorted_Input` for more details on the functionality provided by this mapping template.

**Pros**

- Improves performance of mappings that perform aggregation.
- Can be used with relational and file-based sources, as long as data can be sorted according to the Aggregator’s "Group By" ports.

**Cons**

- With relational sources, additional overhead is exerted on the database when "Order By" clauses are used.

### 1.03 Constraint-Based Loading

**Purpose**

Constraint-based load ordering

**Usage**

Using one mapping and session, load two tables that have a parent / child (primary key / foreign key) relationship.

**Download**

- `m_Constraint_Based>Loading_v61.XML`
- `m_1_03_Constraint_Based>Loading_v711.XML`
Challenge Addressed

Tables in the same dimension are frequently linked by a primary key / foreign key relationship, which requires that a record exist in the "parent" table before a related record can exist in a "child" table. Often these tables are populated by the same data source. One method of populating parent / child tables is to set up a separate mapping and session for each one. However, that requires two reads of the source. This mapping template illustrates PowerMart/ PowerCenter’s "Constraint Based Load Ordering" functionality, which allows developers to read the source once and populate parent and child tables in a single process.

Overview

In a mapping that uses Constraint Based Load Ordering, the Informatica server orders the target load on a row-by-row basis. For every row generated by an active source, the Informatica Server loads the corresponding transformed row first to the primary key table (the "parent" table), then to any foreign key tables (the "child" tables). An active source is the last active transformation in a data stream pipeline. The following transformations can be an active source within a mapping:

- Source Qualifier
- Normalizer (COBOL or flat file)
- Advanced External Procedure
- Aggregator
• Joiner
• Rank
• Filter
• Router
• Mapplet, if it contains one of the above transformations

Implementation Guidelines

Constraint-based load ordering is only implemented in a session for target tables that receive rows from the same active source. When target tables receive records from different active sources, the Informatica Server reverts to normal loading for those tables.

For example, a mapping contains three distinct data streams: the first two both contain a source, Source Qualifier, and target. Since these two targets receive data from different active sources, the Informatica Server reverts to normal loading for both targets. The third data stream contains a source, Normalizer, and two targets. Since these two targets share a single active source (the Normalizer), the Informatica Server performs constraint-based load ordering: loading the primary key table first, then the foreign key table.

When target tables have no key relationships, the Informatica Server does not perform constraint-based loading. Similarly, when target tables have circular key relationships, the Informatica Server reverts to a normal load. For example, you have one target containing a primary key and a foreign key related to the primary key in a second target. The second target also contains a foreign key that references the primary key in the first target. The Informatica Server cannot enforce constraint-based loading for these tables. It reverts to a normal load.

The key concepts illustrated in this mapping template can be found in the Router transformation (RTR_NEW_MANUFACTURERS_ALL_ITEMS) and the two target objects ITEMS and Manufacturers.

RTR_NEW_MANUFACTURERS_ALL_ITEMS is the last single active source of data for the two targets. The targets ITEMS and Manufacturers have a Primary Key – Foreign Key relationship. A manufacturer ID must exist in the Manufacturers table before a row containing the manufacturer ID, can be loaded into the ITEMS table.

Using a normal load for this mapping would result in a constraint error, as the PowerCenter server would attempt to load the tables in any order. In this example, this may result in attempt to load a row into the ITEMS table that does not have a corresponding manufacturer in the Manufacturers table.
Use constraint-based load ordering only when the session option Treat Rows As is set to "Insert." When you select a different Treat Rows As option and you configure the session for constraint-based loading, the Server Manager displays a warning. A session can be configured for constraint-based load ordering by selecting the "Constraint-based load ordering" check box on the Configuration Parameter window of the Session Properties sheet. The Configuration Parameter window is accessed by selecting the button "Advanced Options" from the General Tab of the session properties window.

1.04 Loading Incremental Updates

Purpose

Loading Incremental Changes

Usage

Uses PowerMart / PowerCenter Mapping Variables to process incremental changes in a table only, rather than re-processing all of the data in the source table.

Download

m_LOAD_INCREMENTAL_CHANGES_v61.XML
m_1.04_LOAD_INCREMENTAL_CHANGES_v711.XML
Challenge Addressed

When data in a source table is frequently updated, it is necessary to capture the updated information in the data warehouse. However, due to data volumes and load window considerations, it is often desirable to process only those records that have been updated, rather than re-reading the entire source into a mapping.

Overview

There are a few different methods of processing only the incremental changes that exist in a source table. This mapping template illustrates a method of using a PowerMart / PowerCenter mapping variable to process only those records that have changed since the last time the mapping was run.

Implementation Guidelines

Mapping variables add flexibility to mappings. Once a mapping variable has been declared for a mapping, it can be called by mapping logic at runtime. Unlike mapping parameters, the value of a mapping variable can change throughout the session. When a session begins, it takes the current value of the mapping variable from the repository and brings it into the mapping. When the session ends, it saves the new value of the mapping variable back to the repository, to be used the next time the session is implemented.

The mapping in this template uses a mapping variable called $$INCREMENT_TS. This variable is used in two places within the mapping:

In the Source Qualifier transformation SQ(Customer_Master), the mapping variable is included in the SQL Override;

In the Expression transformation exp_SET_INCREMENT_TS, the mapping variable is set by the port v_SET_INCREMENT_TS.

In this example, when the variable $$INCREMENT_TS was declared, it was given an initial value of "2000-01-01." So, the first time a session that implements this mapping runs, the value of $$INCREMENT_TS will be "2000-01-01." At runtime, the Informatica server will translate the WHERE clause in the SQL Override statement from:
WHERE
UPDATE_TS > '$$INCREMENT_TS'

To:

WHERE
UPDATE_TS > '2000-01-01'

Thus, the first time the mapping runs it will pull all records for which the "update timestamp" is greater than January 1, 2000.

Also, note that the SQL Override queries the database for the value of SYSDATE, and pulls that value into the mapping through the CURRENT_TIMESTAMP port. This brings the current system date from the source system into the mapping. This value will be used in the proceeding Expression transformation to set a new value for $$INCREMENT_TS.

The Expression transformation exp_SET_INCREMENT_TS sets a new value for $$INCREMENT_TS. To do this, it uses the SetVariable function to set the value of $$INCREMENT_TS to the value stored in the port CURRENT_TIMESTAMP, which is actually the value of SYSDATE from the source system. When the session implementing this mapping completes, this new value for $$INCREMENT_TS will be stored in the repository. As a result, the next time the mapping is run, only those records that have an UPDATE_TS greater than the SYSDATE for the previous session’s run will be processed.

**Pros**

- Process fewer records by eliminating static, unchanged records from the data flow.

**Cons**

- Relies on the existence of some kind of "update timestamp" in the source table.
1.05 Error Handling Strategy

Usage

Use this template as a guide for trapping errors in a mapping, sending errors to an error table so that they can be corrected, and reloading fixed errors from the error table into the target system.

Challenge Addressed
Developers routinely write mapping logic that filters records with data errors out of a data stream. However, capturing those error records so that they can be corrected and re-loaded into a target system can present a challenge. The mappings in this mapping template illustrate a process for trapping error records, assigning a severity level to each error, and sending the error rows – which include the complete source row plus additional columns for the error description and severity level – on to an error table.

**Overview**

This mapping template provides two mappings that, taken together, illustrate a simple approach to utilizing Informatica objects in handling known types of Errors. The essential objects shown and utilized are Expression transformations that provide error evaluation code, Lookup transformations that are used to compare or find prerequisite values, and a Router transformation that sends valid rows to the warehouse and error rows to an appropriate error table.

The key to the utilization of the error table in this example is to preserve the source record’s data structure for eventual correction and reprocessing. The first mapping that runs is **m_Customer_Load**. Each source row that is flagged as an error is loaded into the error table, along with an error description per row so that a subject matter expert can view, identify and correct errors. **M_Customer_Fixed_Error_Load** pulls fixed errors from the error table and loads them into the target system.

**Implementation Guidelines**

The error handling in this mapping template is looking at the following known issues with the source data: evaluating whether state and company columns are null or are of the correct length; ensuring email data is in a valid format; and validating that there are supporting rows in the Company and State lookup tables. There is also an assignment of severity level that is used to deliver rows to either the Warehouse or the Error table or both. Along with a severity level, an error description is assigned per error to assist in the correction process.

The key concepts illustrated by these mappings can be found in three transformation objects: a reusable Expression transformation (**exp_GENERAL_ERROR_CHECK**), the Expression transformation (**exp_Customer_Error_Checks**) and a Router transformation (**rtr_NEW_VALID_ROWS_and_ERROR_ROWS**):
exp\_GENERAL\_ERROR\_CHECK illustrates the concept that generic high level errors can be checked universally so as not to maintain and/or update all objects being used for error checking in a project. This particular object is evaluating strings for null data and this evaluates to an error flag of ‘Y’.

In exp\_Customer\_Error\_Checks, all the specific errors for this operation (Customer Load) are done, severity levels are assigned and descriptions of errors are assigned to pass to downstream objects.

The rtr\_NEW\_VALID\_ROWS\_and\_ERROR\_ROWS is used to take the severity levels and determine what rows go where. In this example there are low enough severity levels to warrant load to the warehouse where other levels are high enough to determine that rows only go to an Error table for correction and eventual reprocessing.

There are many options available for more complex error handling operations. Included in the mapping metadata is a UNION statement that can be added to the Source Qualifier that will allow for a single mapping to process new rows from the source and fixed rows from the error table at the same time. In addition, mapping parameters are commonly used to help assist in generalizing and simplifying error handling.

**Pros**

- This template can accommodate any number of data errors, and provides maximum flexibility in allowing for error correction

**Cons**

- Requires input from an external actor, such as a subject matter expert, to correct errors.

### 1.06 Flat File Creation with Headers and Footers

**Purpose**

Creating a flat file containing header, detail and trailer records.
Informatica Mappings

Usage

This mapping creates an output file containing header, detail and trailer records that can be used by other applications. The trailer record contains an extract date and row count that can be used for reconciliation purposes.

Download

m_HDR_DTL_TLR_v61.XML
m_1_06_HDR_DTL_TLR_v711.XML

Challenge Addressed

This mapping templates illustrates how to output a file that has header, trailer, and detail records from a source that contains only detail records. This is often a requirement of external applications that process files output by mappings. For example, applications often require a trailer record that summarizes the information contained in the detail records.

Overview

An aggregator transformation is used in the mapping to get the row count for the trailer record. Similar logic could be used to capture additional information from the detail
This mapping will create three files: MYCUSTOMERS_HEADER, MYCUSTOMERS_DETAIL and MYCUSTOMERS_TRAILER. Using post-session commands these three files are joined together to get the desired output file.

**Implementation Guidelines**

The three files created by the mapping have to be combined to create one output file. If the Informatica server is running on Windows platform, then use the TYPE command to combine the three files into one file. If the Informatica server is on UNIX, then use the CAT command.

The key concepts illustrated by the mapping can be found in the following transformation objects:

- The Expression transformation `exp_HDR_TLR` creates the information contained in the header and trailer records;
- The Aggregator transformation `agg_ROW_COUNT` contains the logic that calculates the number of rows used in the trailer record and removing duplicate records;
- The Aggregation transformation `agg_HDR_RECORD` removes duplicate rows and creates one header record.

Please refer to the metadata descriptions in the mapping `m_HDR_DTL_TLR` for more details on this mapping’s functionality.

**Pros**

- Easy process to create a full file containing header, detail and trailer information.
- The output file gives meaningful detailed information about the contents of the file.
- Using the same mapping, this information can be stored in a relational table (remove the post-session commands).

**Cons**

- Knowledge of some basic file management commands is required to join the three output files for this mapping.
1.07 Removing Duplicate Source Records

Purpose

Removing duplicate source records

Usage

Remove duplicate records from a source when the source records contain a functional key.

Challenge Addressed

Duplicate records are occasionally found in source data. Due to primary key constraints on a target database, only one version of a duplicate source record should be loaded into the target. This mapping illustrates one alternative for removing duplicate records when the source has a functional key that can be used for grouping.

Overview
The mapping **m_LoadNielsenData_RemoveDuplicates** illustrates the concept of using the functionality within an Aggregator transformation to remove duplicate records from a source and load this data into a target table of the same structure.

**Implementation Guidelines**

The PowerMart / PowerCenter Aggregator transformation can be used to "group" records based on a logical key. In this example mapping, the source data is Nielsen company data that resides in a relational table that does not contain a primary key (to allow duplicates). The functional key to each source record is the STATE_TAX_ID field.

The key concept illustrated by this mapping can be found in the following transformation objects:

The Source Qualifier object **SQ_NIELSEN** is used to retrieve data from the NIELSEN raw data table. The select statement orders the rows by STATE_TAX_ID (the group-by column in the Aggregator transformation), so that the "Sorted Ports" option can be used in the mapping.

In **Agg_RemoveDuplicateRecords**, the STATE_TAX_ID port is selected as the "Group By" port in the transformation (the number of "Group By" ports must correspond to the "Number of Sorted Ports" indicated in the Source Qualifier object). The Informatica server, by default, returns the last row in a group if no aggregate function is specified. If two records with the same value for STATE_TAX_ID enter the Aggregator, only one record will be returned by the Aggregator. As a result, duplicate source records are eliminated.

Note that the SORTED INPUT option is chosen in the aggregator so that only one group’s worth of data will be cached in memory by the Informatica server (dramatic increases in Aggregator performance).

Please refer to the metadata descriptions in the mapping **m_LoadNielsenData_RemoveDuplicates** for more details on this mapping’s functionality.

**Pros**

- Relatively simple method of eliminating duplicate records.
• At run-time, faster than allowing the database to fail records due to primary key constraints.
• Can group by multiple ports / functional keys.

Cons

• Records must contain a functional "group by" key.

1.08 Transforming One Record into Multiple Records

Purpose

Creating Multiple Target Rows from one Source Row

Usage

Use this template as a guide for creating multiple target rows from one source row.

Download

m_OneSourceRow_to_MultipleTargetRows_v61.XML
m_1_08_OneSourceRow_to_MultipleTargetRows_v711.XML
Challenge Addressed

Developers sometimes need to transform one source row into multiple target rows. For example, attributes that are stored in columns in one table or file may need to be broken out into multiple records – one record per attribute – upon being loaded into a data warehouse.

Overview

This mapping template demonstrates how to use a Normalizer transformation to split one source row into multiple target rows. Normalization is the process of organizing data. In database terms, this includes creating normalized tables and establishing relationships between those tables according to rules designed to both protect the data and make the database more flexible by eliminating redundancy and inconsistent dependencies.

The Normalizer transformation normalizes records from VSAM and relational sources, allowing you to organize the data according to your own needs. A Normalizer transformation can appear anywhere in a data flow when you normalize a relational source. Use a Normalizer transformation instead of the Source Qualifier transformation when you normalize a VSAM source.

The Normalizer transformation is primarily used with VSAM sources, which are often stored in a denormalized format. The OCCURS statement in a COBOL file, for example, nests multiple records of information in a single row. Using the Normalizer transformation, you can break out repeated data within a record into separate records. For each new record it creates, the Normalizer transformation generates a unique identifier. You can use this key value to join the normalized records.

You can also use the Normalizer transformation with relational and flat file sources to
create multiple rows from a single row of data. This is the way that we are using the Normalizer transformation in this Mapping Template.

**Implementation Guidelines**

Once you have created a Normalizer transformation in your mapping, you need to customize its properties. On the Normalizer transformation’s "Normalizer" tab, you can set the port names, data types, key fields, precision and an **Occurs** value for each port passing through the Normalizer.

The Occurs value corresponds to the number of attributes in each incoming row that need to be broken out into separate rows. In this example, each record from the source needs to be split into three records. Values stored in the source columns AREA, DEPARTMENT, and REGION will be stored in the column EMP_INFO in the target table.

To achieve this, we give the EMP_INFO port in the Normalizer transformation an **Occurs** value of three, indicating that EMP_INFO attributes appear three times in each source record. The Normalizer transformation automatically creates three input ports (EMP_INFO_in1, EMP_INFO_in2, EMP_INFO_in3), to which the AREA, DEPARTMENT, and REGION ports from the source can be mapped. As a result, each incoming source record is transformed into three separate target records, one of which contains AREA data in the EMP_INFO column, the second containing DEPARTMENT data, and the third containing REGION data.

**Tips:**

- You can use a single Normalizer transformation to handle multiple levels of denormalization in the same record. For example, a single record might contain two different detail record sets. Rather than using two Normalizer transformations to handle the two different detail record sets, you handle both normalizations in the same transformation.
- If you have denormalized data for which the Normalizer transformation has created key values, connect the ports representing the repeated data and the output port for the generated keys to a different portion of the data flow in the mapping. Ultimately, you may want to write these values to different targets.

**1.09 Processing VSAM Source Files**

**Purpose**
Processing VSAM files that contain OCCURS and REDEFINES

Usage

Extract multiple record types, such as Header, Detail, and Trailer records, from a VSAM source when the copybook contains OCCURS and REDEFINES statements.

Download

m_VSAM_Sales_Data_v61.XML
m_1_09_VSAM_Sales_Data_v711.XML

Challenge Addressed
VSAM sources often de-normalize data and compact the equivalent of separate table records into a single record. OCCURS statements, for example, define repeated information in the same record. REDEFINES statements build the description of one record based on the definition of another record. Processing VSAM sources with OCCURS and REDEFINES statements requires knowledge of how the PowerCenter Normalizer transformation object handles VSAM data.

Overview

The VSAM source in this example includes data about four different suppliers, each stored in the same record. The source file contains both OCCURS and REDEFINES statements, as shown below in bold:

```cobol
03  BODY-DATA.
  05  BODY-STORE-NAME   PIC X(30).
  05  BODY-STORE-ADDR1  PIC X(30).
  05  BODY-STORE-CITY   PIC X(30).

03  DETAIL-DATA REDEFINES BODY-DATA.
  05  DETAIL-ITEM       PIC 9(9).
  05  DETAIL-DESC       PIC X(30).
  05  DETAIL-PRICE      PIC 9(4)V99.
  05  DETAIL-QTY        PIC 9(5).

05  SUPPLIER-INFO OCCURS 4 TIMES.
  10  SUPPLIER-CODE     PIC XX.
  10  SUPPLIER-NAME     PIC X(8).
```

The mapping in this template illustrates how to use Normalizer and Router transformation objects to extract data from this VSAM file and send that data to multiple targets.

Implementation Guidelines

Part of the implementation for this mapping template requires an understanding of how the Source Analyzer and Warehouse Designer treat VSAM files.

When the Source Analyzer utility analyzes this VSAM file, it creates a different column for each OCCURS in the COBOL file. The OCCURS statements define repeated information in the same record. A Normalizer transformation will be used in the mapping to normalize this information.

Once the source definition is created, the Warehouse Designer utility does the following:
Informatica Mappings

- Creates one target table (SUPPLIER_INFO) for each OCCURS statement when you drag the COBOL source definition into the Warehouse Designer.
- Creates a primary-foreign key relationship between those tables.
- Creates a generated column ID (GCID).

The REDEFINES statement allows you to specify multiple PICTURE clauses for the sample physical data location. Therefore, Filter or Router transformations are required to separate data into the multiple tables created for REDEFINES statements.

For each REDEFINES statement, the Warehouse Designer utility does the following:

- Creates a target table (DETAIL_DATA) when the COBOL source definition is dragged into the Warehouse Designer.
- Creates one primary-foreign key relationship.

The following target tables show that the Warehouse Designer creates a target table called DETAIL_DATA for the REDEFINES in the source file, as well as a target table called SUPPLIER_INFO because of the OCCURS clause.

The mapping will contain logic for separating BODY_DATA, DETAIL_DATA and SUPPLIER_INFO into three target tables.

BODY_DATA contains store information such as store name, store address and store city. However, the REDEFINES clause indicates that detail information about merchandise is in the same physical location as BODY_DATA. Detail information about the store’s merchandise includes item, description, price and quantity. Finally, the OCCURS clause tells you that there can be four suppliers to each item.

Information about the store, the items and the suppliers are all in the same physical location in the data file. Therefore, a Router transformation is used to separate information about the store, the items and the suppliers.
Notice that at the beginning of the COBOL file, header information for a store is defined with a value of H and detail information about items is defined with a value of D. Detail information about items also includes supplier information, and supplier information has a supplier code that is not null. Using this information, the following logic is used in the Router transformation to separate records:

Records in the STORE_INFO_GRP are indicated by: HDR_REC_TYPE='H'

Records in the ITEM_INFO_GRP are indicated by: HDR_REC_TYPS='D'

Records in the SUPPLIER_INFO_GRP are indicated by: HDR_REC_TYPE='D' AND NOT ISNULL(SUPPLIER_CODE) AND SUPPLIER_CODE !=""

Please refer to the metadata descriptions in the mapping m_VSAM_Sales_Data for more details on this mapping’s functionality.

1.10 Processing Data from an XML Source

Purpose

Processing data from an XML source

Usage

This mapping template provides information about sourcing an XML document.

Download

m_XML_Source_v61.XML
m_1_10_XML_Source_v711.XML
Challenge Addressed

Overview

The mapping m_XML_Source illustrates the concept of using an XML file as a source. You can import source definitions from XML files, XML schemas, and DTDs.

When you import a source definition based on an XML file that does not have an associated DTD, Designer determines the types and occurrences of data based solely on how data is represented in the XML file. To ensure that Designer can provide an accurate definition of your data, import from an XML source file that represents the occurrence and type of data as accurately as possible. When you import a source definition from a DTD or an XML schema file, Designer can provide an accurate definition of the data based on the description provided in the DTD or XML schema file.

Designer represents the XML hierarchy from XML, DTD, or XML schema files as related logical groups with primary and foreign keys in the source definition. You can configure Designer to generate groups and primary keys, or you can create your own groups and specify keys.

Designer saves the XML hierarchy and group information to the repository. It also saves
the cardinality and datatype of each element in the hierarchy. It validates any change you make to groups against the XML hierarchy. It does not allow any change that is not consistent with the hierarchy or that breaks the rules for XML groups.

When you import an XML source definition, your ability to change the cardinality and datatype of the elements in the hierarchy depends on the type of file you are importing. You cannot change the cardinality of the root element since it only occurs once in an XML hierarchy. You also cannot change the cardinality of an attribute since it only occurs once within a group.

When you import an XML source definition, Designer disregards any encoding declaration in the XML file and uses the code page of the repository instead.

Implementation Guidelines

Tip: Your source file might contain elements that do not have associated data or it might have enclosure elements that contain only sub-elements. You can select not to include these empty elements as columns on the XML Import section of the Options window.

Tip: If you select the Generate Default Groups option, Designer lists the groups that it creates. You can modify the groups and columns if necessary. In the Columns section, Designer displays the columns for the current group, including the datatype, null constraint, and pivot attribute for each column. If you do not select the Generate Default Groups option, Designer does not display any groups. You can manually create groups, add columns and set column properties, and set primary and foreign keys to depict the relationship between custom groups.

Tip: In the XML Tree area, Designer highlights several things to emphasize the elements relevant to a selected group, including:

- The "Groups At" element for the group turns bright pink;
- All the elements corresponding to the columns in the group turn bold blue.
- The element corresponding to the foreign key column turns bold blue.
- The element corresponding to the column selected in the Columns section is underlined.

Tip: If you have PowerCenter 1.6 or 1.7, you can create a group by importing an existing XML view. To import an XML view, click Import XML View and select the metadata.xml file you want to import.
**Tip:** Once you create an XML source definition, you cannot change it to any other source type. Conversely, you cannot change other types of source definitions to XML.

### 1.11 Joining a Flat File with a Relational Table

**Purpose**

Joining Flat Files with Database Tables

**Usage**

Use this template as a guide for joining flat file sources with relational table sources.

**Download**

- [Join_FlatFile_with_Database_Table_v61.XML](#)
- [m_1_11_Join_FlatFile_with_Database_Table_v711.XML](#)
Challenge Addressed

A developer needs to join information from two heterogeneous sources: one database source and one flat file source.

Overview

The Joiner transformation can be used to join data from flat file sources with data from relational sources in a single mapping. In order to be used in a mapping, a Joiner transformation requires two input transformations from two separate data flows (an input transformation is any transformation connected to the input ports of the current transformation).

There are some limitations on the types of data flows that can be connected to a Joiner. You cannot use a Joiner in the following situations:

- Both pipelines begin with the same original data source.
- Both input pipelines originate from the same Source Qualifier transformation.
- Both input pipelines originate from the same Normalizer transformation.
- Both input pipelines originate from the same Joiner transformation.
- Either input pipeline contains an Update Strategy transformation.
- Either input pipeline contains a connected or unconnected Sequence Generator transformation.

Implementation Guidelines

When you insert a Joiner transformation into a mapping, there are several settings that need to be configured. For example, you need to indicate which source input is the Master source and which is the Detail source, you need to set the Join Type (see description of Join Types below), and you need to specify a valid join condition.

The Joiner transformation in this Mapping Template, jnr_JOIN_STORES_and_ORDERS, uses a Normal Join to join data from the STORES flat file source and the ORDERS table. The ORDERS table is defined as the Master source, and STORES file is the Detail. The join condition in jnr_JOIN_STORES_and_ORDERS is "Store_ID = Store_ID1." Records from
STORES will be joined to records from ORDERS when the Store_ID values from each source match, just as a simple join in a relational database occurs.

**Normal Join**

With a Normal join, the Informatica Server discards all rows of data from the master and detail source that do not match, based on the condition.

**Master Outer Join**

A Master Outer Join keeps all rows of data from the detail source and the matching rows from the master source. It discards the unmatched rows from the master source.

**Detail Outer Join**

A Detail Outer Join keeps all rows of data from the master source and the matching rows from the detail source. It discards the unmatched rows from the detail source.

**Full Outer Join**

A Full Outer Join keeps all rows of data from both the master and detail sources.

**Tips:**

- When a Joiner transformation occurs in a session, the Informatica Server first reads all the records from the Master source and builds Index and Data caches based on the Master source rows. Since the Joiner caches only the Master source records, the source with fewer records should be designated as the Master source so that the time required for caching is as little as possible.
- The Joiner transformation does not match null values. For example, if both Store_ID and Store_ID1 from the example above contain a row with a null value, the Informatica Server does not consider them a match and does not join the two rows. To join rows with null values, you can replace null input with default values, and then join on the default values.

**2.04 Aggregation Using Expression Transformation**

**Purpose**
Aggregating Records using an Expression Transformation

Usage

As a potential method of enhancing performance, this mapping uses PowerMart / PowerCenter Expression Transformation objects to aggregate data rather than using an Aggregator Transformation object.

Download

m_AGGREGATION_USING_EXPRESSION_TRANS_v61.XML
m_2_04_Aggregation_Using_Expression_Trans_v711.XML

Challenge Addressed

When you must aggregate records (sum, count, find the min or max values within a data set, etc.) within your mapping, PowerMart/PowerCenter provides for the use of an Aggregator Transformation object to do the aggregation. Using an Aggregator object, however, can sometimes use up significant server resources in order to actually perform
the aggregations and calculations. This is due partly to the fact that Aggregators must first cache records in memory and then perform functions and calculations.

**Overview**

One method of calculating aggregations without using an Aggregator Transformation object is to use an Expression Transformation in conjunction with variable logic to perform the calculation. An Expression Transformation does not cache records in memory, and therefore it can sometimes be more efficient than using an Aggregator. This mapping template illustrates a method of using an Expression object with variables to perform aggregations.

**Implementation Guidelines**

This template demonstrates reading from two source tables (ORDERS, ORDER_ITEMS), and calculates the sum of the total revenue (Price * Quantity) based off of the month and year in the DATE_ENTERED column. Therefore, we are trying to sum the revenue while grouping off of the month and year.

1. Select data from the source making sure to add an "order by" to the SQL Override on the column(s) that you need to group by. THE ORDER BY IS REQUIRED! Without the order by statement, the expression will not be able to properly sum the totals. *Note that the order by can either be done in the SQL Override of the Source Qualifier, or it can be done with the Sorter Transformation object. The SQL Override looks like this:

```sql
SELECT ORDER_ITEMS.QUANTITY, ORDER_ITEMS.PRICE, ORDERS.DATE_ENTERED
FROM ORDER_ITEMS, ORDERS
WHERE
ORDERS.ORDER_ID=ORDER_ITEMS.ORDER_ID
ORDER BY
ORDERS.DATE_ENTERED
```
2. For the mapping to work, you must be able to filter out every record except for the last record in each grouping (with that record holding the summation of total revenue). Assuming your data is ordered (see step 1), then you can determine the last record in each grouping by evaluating conditions in transformation variables.  
   Condition 1: If the current record starts a new Month/Year grouping but is not the first record read from the source, then the previous record was the last record in the grouping.  
   Condition 2: The current record is the last record read from the source. We will evaluate Condition 1 at a later time, but in order to evaluate Condition 2 (knowing if your current record is the last record from the source), you must know how many total records you are going to read, and you also must keep a record count of each record coming into the mapping. When your record count equals the total number of records from the source, then you know that the current record is the last record. The object `exp_Get_ORDER_ITEMS_Record_Count` accomplishes this task. First, a mapping variable `$Record_Count` is created. Then, the expression will call the unconnected lookup if the current record is the first record in the mapping (we will know that because the `$Record_Count` is initialized to '0'). We will send a dummy value of '0' to the lookup. The lookup SQL has been modified to the following:

```
SELECT COUNT(*) as ORDER_ID
FROM ORDER_ITEMS
```

The lookup adds the condition of 'where ORDER_ID > DUMMIE_IN'. Since we send in '0' for the dummy value, the full SQL executed internally for the lookup is as follows:

```
SELECT COUNT(*) as ORDER_ID
FROM ORDER_ITEMS
WHERE ORDER_ID > 0
ORDER BY ORDER_ID
```

The count is returned to the expression and then stored in the `v_SRC_TABLE_COUNT` port. Next, the `v_RECORD_COUNT` variable simply increments the `$Record_Count` mapping variable for every record that passes through the mapping. We now have the data we need to determine if the current record is the last record in the mapping. Also, we have the data we need to determine if the current record is the first record in the mapping, which is used later. Finally, two flags are set with the `o_FIRST_RECORD_FLAG` and the `o_LAST_RECORD_FLAG`. These flags are raised (i.e. set to 1) if the current
3. The object `exp_Sum_Revenue` is where the aggregation activities take place. This object must be able to:
   a. Calculate the total revenue of the current record by multiplying `price*quantity`
   b. Parse out the month and year from the `DATE_ENTERED` field
   c. Remember the running total revenue up to the previous record
   d. Remember the month and year of the previous record
   e. Determine if the current record starts a new aggregation grouping
   f. Add the running total revenue from the previous record with the revenue for the current record as long as the current record does not start a new aggregate grouping
   g. If the current record is the last record read from the source (based on the `o_LAST_RECORD_FLAG` set in the previous transformation), output the month, year, and running revenue of the current record, otherwise output the month, year, and total revenue up to the previous record

In order to accomplish part a) above, the `v_REVENUE` port was created to multiply the `price*quantity` for the current record. Two variables were created to accomplish part b): `v_MONTH` and `v_YEAR`

Part c) is done by creating two variable ports, ensuring that they are ordered correctly in the transformation. The `v_PREVIOUS_TOTAL` port must be first, and is set to evaluate to the value in the second port, `v_RUNNING_TOTAL`. It is important that `v_RUNNING_TOTAL` is at the end of the transformation, and `v_PREVIOUS_TOTAL` is at the beginning. When a record passes through this transformation, `v_RUNNING_TOTAL` is the last thing to be set, and it is set to add the revenue of the current record with itself if the record is a part of the current aggregate grouping. If the record is not a part of the current aggregate grouping, then it will simply evaluate to the revenue of the current record. `v_RUNNING_TOTAL` will remember how it evaluated in the previous record so when the next record comes in, it retains the previous value. Before that value is changed, the `v_PREVIOUS_TOTAL` variable stores whatever was in the `v_RUNNING_TOTAL`.

Similar to how we remembered the previous running total, we can accomplish d) with the same method. Two variable ports were created for the month and year ports. For example, before the `v_MONTH` port was evaluated to the current
record, the v_PREVIOUS_MONTH port would grab the value in v_MONTH before it could be changed.

Now that we have the month and year from the previous record, we can easily determine if the current record starts a new aggregate grouping (part e). The v_NEW_GROUPING_FLAG is set to 1 if the month concatenated with the year of the previous record do not equal the month concatenated with the year of the current record.

To accomplish part f), the v_RUNNING_TOTAL will first see if the current record starts a new grouping by checking the v_NEW_GROUPING_FLAG. If it is a new group, then it simply evaluates to the v_REVENUE value which is price*quantity for the current record. If the record falls within the same grouping, it will add the current records price*quantity with the running total from the previous record.

Finally, to accomplish part g), the three output ports were created to output the month, year, and revenue. Each port will output the previous records’ month, year, and running total UNLESS the current record is the last record read from the source. The o_LAST_RECORD_FLAG which was evaluated in the previous transformation is used to make this determination.

5. A filter is used to only pass the last record of each group, or the last record read from the source. All of the information needed to make this determination has been calculated in earlier steps. The check should be if a record that starts a new grouping (but is not the first record read from the source), or if the record is the last record read from the source, then pass through. Remember that as long as the current record is not the last record read from the source, the current record sitting in the filter actually holds the month, year, and revenue from the previous record. This is the trick to only inserting the total amounts for each grouping. The statement used is as follows:

\[(\text{o_NEW_GROUPING_FLAG}) \text{ and (NOT o_FIRST_RECORD_FLAG))}\]

or

\(\text{o_LAST_RECORD_FLAG}\)

6. Finally, a sequence generator is used to generate the primary key value for the target table.

**Pros**
• Does not cache records in memory thereby making the Expression object sometimes more efficient than an Aggregator.

Cons

• Does not allow for full Aggregator functionality and flexibility.
• Requires advanced development abilities to implement.

2.05 Dynamic Caching

Purpose

Dynamic Caching

Usage

This mapping uses Dynamic Lookup Cache functionality to detect duplicate records coming from the source.

Download

m_Dynamic_Caching_v61.XML
m_2_05_Dynamic_Caching_v711.XML
Challenge Addressed

Source data often contains duplicate records. It is typically desirable to insert only one instance of a unique record into the target. This mapping provides an alternative for detecting duplicate records from the source.

Overview

There are a few different methods of eliminating duplicate records that exist in a source table. This mapping template illustrates a method of using a PowerMart / PowerCenter dynamic lookup to check whether the record is new or if it already exists in the target. Dynamic lookups update the lookup cache "on the fly" and can determine if a record is an insert or update to the target. Unlike un-cached lookups, the cache file is updated each time a row is inserted/updated into the target table.

Implementation Guidelines

The mapping in this template uses a Lookup transformation object called **lkp_PRODUCT_DIMENSION**. The Lookup transformation uses a dynamic lookup cache. When the session starts, the Informatica Server builds the lookup cache from the target table, PRODUCT_DIMENSION. When **lkp_PRODUCT_DIMENSION** evaluates a row from the source that does not exist in the lookup cache, it inserts the row into cache and assigns the "NewLookupRow" output port the value of 1. When **lkp_PRODUCT_DIMENSION** evaluates a row from the source that does exist in the lookup cache, it does not insert the row into cache and assigns "NewLookupRow" output port the value of 0.
The filter in this mapping flt_DUPLICATE_ROW checks if the row is a duplicate or not by evaluating the NewLookupRow output port. If the value of the port is 0, the row is filtered out, as it is a duplicate row. If the value of the port is not equal to 0, then the row is passed out to the target table.

In this example you can test the mapping with 3 scenarios:

1. First time the mapping is run all records should be inserted into the target if every row is unique.
2. If the mapping is run again with the same source data and the target table with the data from the previous run no records will be inserted.
3. If you modify the source table to have three records that are all duplicates when the session is run the row will only be inserted once.

Pros

- Duplicates can be detected real-time during the session run.

Cons

- If the target table starts to grow very large the cache file will become very large, potentially increasing the amount of time required to create the cache.
2.08 Streamline a Mapping with a Mapplet

Purpose

Streamline a Mapping with a mapplet

Usage

Uses a PowerMart / PowerCenter mapplet to streamline lookup and calculation logic

Download

m_2_08_Streamlined_Mapping_w_Mapplet_v61.zip
m_2_08_Streamlined_Mapping_w_Mapplet_v711.XML
Challenge Addressed

Often a series of complex logic causes a mapping to become unwieldy and cumbersome. Debugging and documenting such a complex mapping can be difficult. It is also common to find that calculation and lookup procedures are used repetitively throughout the mapping.

Overview

This mapping template illustrates a method of using a PowerMart / PowerCenter mapplet to hide the complexity of certain logical procedures.
Implementation Guidelines

Mapplets can simplify and streamline complex and repetitive logic. The mapping in this template uses a mapplet to calculate and rank company sales by customer zip code. The mapplet itself does not have sources defined, and only expects two input ports via a mapplet input transformation. Two lookups are used to determine customer zip code and to return the necessary fields for the calculation of total sales. Total sales is calculated as 

\[(\text{PRICE} \times \text{QUANTITY}) - \text{DISCOUNT}.\]

After the calculation of TOTAL SALES, an Aggregator transformation sums TOTAL_SALES and groups by ZIP_CODE. Finally, a rank transformation is used to RANK the top 25 total sales amounts in descending order by zip code.

Most complex mappings can be simplified using mapplets. To remove sections of logic and build a mapplet, cut and paste the objects between the Mapping designer and the Mapplet designer. A mapplet can be active or passive depending on the transformations in the mapplet. Active mapplets contain one or more active transformation objects. Passive mapplets contain only passive transformation objects. When create a mapplet all transformation rules still apply. When adding transformations to a mapplet, keep the following restrictions in mind:

- You may only use a reusable Sequence Generator transformation if a sequence generator is needed.
- If a Stored Procedure transformation is needed, the Stored Procedure Type must be set to Normal.
- PowerMart 3.5-style LOOKUP functions cannot be used in a mapplet.
- The following objects cannot be used in a mapplet:
  - Normalizer transformations
  - COBOL sources
  - XML Source Qualifier transformations
  - XML sources
  - Target definitions
  - Other mapplets

Although reusable transformations and shortcuts can be used in a mapplet, it is recommended that a copy of a transformation be used. Reusable transformations and shortcuts automatically inherit changes to their original transformations. Thus, a change in the reusable object may invalidate the mapplet and the mappings that use the mapplet.

Pros
• Streamline large, complex mappings
• Maplet logic can be reused in other mappings

2.04 Aggregation Using Expression Transformation

Purpose

Aggregating Records using an Expression Transformation

Usage

As a potential method of enhancing performance, this mapping uses PowerMart / PowerCenter Expression Transformation objects to aggregate data rather than using an Aggregator Transformation object.

Download

m_AGREGATION_USING_EXPRESSION_TRANS_v61.XML
m_2_04_Aggregation_Using_Expression_Trans_v711.XML
Challenge Addressed

When you must aggregate records (sum, count, find the min or max values within a data set, etc.) within your mapping, PowerMart/PowerCenter provides for the use of an Aggregator Transformation object to do the aggregation. Using an Aggregator object, however, can sometimes use up significant server resources in order to actually perform the aggregations and calculations. This is due partly to the fact that Aggregators must first cache records in memory and then perform functions and calculations.

Overview

One method of calculating aggregations without using an Aggregator Transformation object is to use an Expression Transformation in conjunction with variable logic to perform the calculation. An Expression Transformation does not cache records in memory, and therefore it can sometimes be more efficient than using an Aggregator. This mapping template illustrates a method of using an Expression object with variables to perform aggregations.

Implementation Guidelines

This template demonstrates reading from two source tables (ORDERS,
ORDER_ITEMS), and calculates the sum of the total revenue (Price * Quantity) based off of the month and year in the DATE_ENTERED column. Therefore, we are trying to sum the revenue while grouping off of the month and year.

1. Select data from the source making sure to add an "order by" to the SQL Override on the column(s) that you need to group by. **THE ORDER BY IS REQUIRED!** Without the order by statement, the expression will not be able to properly sum the totals. *Note that the order by can either be done in the SQL Override of the Source Qualifier, or it can be done with the Sorter Transformation object. The SQL Override looks like this:

   ```sql
   SELECT ORDER_ITEMS.QUANTITY, ORDER_ITEMS.PRICE,
   ORDERS.DATE_ENTERED
   FROM ORDER_ITEMS, ORDERS
   WHERE
   ORDERS.ORDER_ID=ORDER_ITEMS.ORDER_ID
   ORDER BY
   ORDERS.DATE_ENTERED
   ```

2. For the mapping to work, you must be able to filter out every record except for the last record in each grouping (with that record holding the summation of total revenue). Assuming your data is ordered (see step 1), then you can determine the last record in each grouping by evaluating conditions in transformation variables. Condition 1: If the current record starts a new Month/Year grouping but is not the first record read from the source, then the previous record was the last record in the grouping. Condition 2: The current record is the last record read from the source. We will evaluate Condition 1 at a later time, but in order to evaluate Condition 2 (knowing if your current record is the last record from the source), you must know how many total records you are going to read, and you also must keep a record count of each record coming into the mapping. When your record count equals the total number of records from the source, then you know that the current record is the last record. The object `exp_Get_ORDER_ITEMS_Record_Count` accomplishes this task. First, a mapping variable $$Record_Count$$ is created. Then, the expression will call the unconnected lookup if the current record is the first record in the mapping (we will know that because the $$Record_Count$$ is initialized to '0'). We will send a dummy value of ‘0’ to the lookup. The lookup SQL has been modified to the following:
SELECT COUNT(*) as ORDER_ID
FROM ORDER_ITEMS

The lookup adds the condition of 'where ORDER_ID > DUMMY_IN'. Since we send in '0' for the dummy value, the full SQL executed internally for the lookup is as follows:

SELECT COUNT(*) as ORDER_ID
FROM ORDER_ITEMS
WHERE ORDER_ID > 0
ORDER BY ORDER_ID

The count is returned to the expression and then stored in the v_SRC_TABLE_COUNT port. Next, the v_RECORD_COUNT variable simply increments the $$Record_Count mapping variable for every record that passes through the mapping. We now have the data we need to determine if the current record is the last record in the mapping. Also, we have the data we need to determine if the current record is the first record in the mapping, which is used later. Finally, two flags are set with the o_FIRST_RECORD_FLAG and the o_LAST_RECORD_FLAG. These flags are raised (i.e. set to 1) if the current record is the first record or the last record in the mapping.

3. 4. The object **exp_Sum_Revenue** is where the aggregation activities take place. This object must be able to:
   a. Calculate the total revenue of the current record by multiplying price*quantity
   b. Parse out the month and year from the DATE_ENTERED field
   c. Remember the running total revenue up to the previous record
   d. Remember the month and year of the previous record
   e. Determine if the current record starts a new aggregation grouping
   f. Add the running total revenue from the previous record with the revenue for the current record as long as the current record does not start a new aggregate grouping
   g. If the current record is the last record read from the source (based on the o_LAST_RECORD_FLAG set in the previous transformation), output the
Informatica Mappings

month, year, and running revenue of the current record, otherwise output
the month, year, and total revenue up to the previous record

In order to accomplish part a) above, the v_REVENUE port was created to
multiply the price*quantity for the current record. Two variables were created to
accomplish part b): v_MONTH and v_YEAR.

Part c) is done by creating two variable ports, ensuring that they are ordered
correctly in the transformation. The v_PREVIOUS_TOTAL port must be first,
and is set to evaluate to the value in the second port, v_RUNNING_TOTAL. It is
important that v_RUNNING_TOTAL is at the end of the transformation, and
v_PREVIOUS_TOTAL is at the beginning. When a record passes through this
transformation, v_RUNNING_TOTAL is the last thing to be set, and it is set to
add the revenue of the current record with itself if the record is a part of the
current aggregate grouping. If the record is not a part of the current aggregate
grouping, then it will simply evaluate to the revenue of the current record.
v_RUNNING_TOTAL will remember how it evaluated in the previous record so
when the next record comes in, it retains the previous value. Before that value is
changed, the v_PREVIOUS_TOTAL variable stores whatever was in the
v_RUNNING_TOTAL.

Similar to how we remembered the previous running total, we can accomplish d)
with the same method. Two variable ports were created for the month and year
ports. For example, before the v_MONTH port was evaluated to the current
record, the v_PREVIOUS_MONTH port would grab the value in v_MONTH
before it could be changed.

Now that we have the month and year from the previous record, we can easily
determine if the current record starts a new aggregate grouping (part e). The
v_NEW_GROUPING_FLAG is set to 1 if the month concatenated with the year
of the previous record do not equal the month concatenated with the year of the
current record.

To accomplish part f), the v_RUNNING_TOTAL will first see if the current
record starts a new grouping by checking the v_NEW_GROUPING_FLAG. If it
is a new group, then it simply evaluates to the v_REVENUE value which is
price*quantity for the current record. If the record falls within the same grouping,
it will add the current records price*quantity with the running total from the
previous record.

Finally, to accomplish part g), the three output ports were created to output the
month, year, and revenue. Each port will output the previous records’ month,
year, and running total UNLESS the current record is the last record read from the
source. The `o_LAST_RECORD_FLAG` which was evaluated in the previous transformation is used to make this determination.

5. A filter is used to only pass the last record of each group, or the last record read from the source. All of the information needed to make this determination has been calculated in earlier steps. The check should be if a record that starts a new grouping (but is not the first record read from the source), or if the record is the last record read from the source, then pass through. Remember that as long as the current record is not the last record read from the source, the current record sitting in the filter actually holds the month, year, and revenue from the previous record. This is the trick to only inserting the total amounts for each grouping. The statement used is as follows:

\[
((o\_NEW\_GROUPING\_FLAG) \text{ and } (\text{NOT } o\_FIRST\_RECORD\_FLAG)) \text{ or } (o\_LAST\_RECORD\_FLAG)
\]

6. Finally, a sequence generator is used to generate the primary key value for the target table.

**Pros**

- Does not cache records in memory thereby making the Expression object sometimes more efficient than an Aggregator.

**Cons**

- Does not allow for full Aggregator functionality and flexibility.
- Requires advanced development abilities to implement.
2.06 Building a Parameter File

Purpose

Building a Parameter File

Usage

Uses PowerMart / PowerCenter to build a mapping that produces as a target file a parameter file (.parm) that can be referenced by other mappings and sessions.

Download

m_Build_Parameter_File_v61.XML
m_2_06_Build_Parameter_File_v711.XML

Challenge Addressed

When many mappings use the same parameter file it is desirable to be able to easily re-create the file when mapping parameters are changed or updated.

Overview
There are a few different methods of creating a parameter file with a mapping. This mapping template illustrates a method of using a PowerMart / PowerCenter mapping to source from a process table containing mapping parameters and create a parameter file.

**Implementation Guidelines**

The mapping sources a process table that tracks all mappings and the parameters they use.

When you configure the Informatica Server to move data in ASCII mode, CHR returns the ASCII character corresponding to the numeric value you pass to this function. ASCII values fall in the range 0 to 255. You can pass any integer to CHR, but only ASCII codes 32 to 126 are printable characters. In this example, the numeric value of 10 is passed in and returns the ASCII character for a carriage return.

In the example the following statement is created in the expression:

```
'['||RTRIM(FOLDER_NAME,' ')||' | | RTRIM(SESSION_NAME,' ')) | |chr(10)|| Line 1
'$$JobId= ||TO_CHAR(JOB_ID)||chr(10)|| Line 2
'$$SourceSysCodeId= ||TO_CHAR(SOURCE_SYSTEM_CODE_ID)||chr(10)|| Line 3
'$$TargetSysCodeId= ||TO_CHAR(TARGET_SYSTEM_CODE_ID)||chr(10) Line 4
```

Line 1 is equivalent to:

[FOLDER_NAME.SESSION_NAME]

Line 2 is equivalent to:

$JobId=

Line 3 is equivalent to:

$SourceSysCodeId=

Line 4 is equivalent to:

$TargetSysCodeId=
**Informatica Mappings**

**Pros**

- If many mappings use the same parameter file, changes can be made by updating the source table and recreating the file. Using this process is faster than manually updating the file line by line.

**Cons**

- Relies on all mapping parameters to be the same and in the same order in all of the mappings.

**2.07 Sequence Generator Alternative**

**Purpose**

Sequence Generator Alternative

**Usage**

Uses a PowerMart / PowerCenter expression object to create a series of sequence numbers rather than using the sequence generator object.

**Download**

- m_2_07_Sequenc Gen_Alternative_v61.zip
- m_2_07_Sequenc Gen_Alternative_v711.XML
Challenge Addressed

Many times data processing requires that new and updated records come in together in the same file or source system. In these instances, the sequence generator can become inefficient as numbers are generated whether or not a record is an insert or an update. In addition, the sequence generator can only produce numbers up to 2,147,483,647. There are times when there is a need to generate larger sequence numbers.

Overview

This mapping template illustrates a method of using PowerMart / PowerCenter expression objects to create a sequence generator that can intelligently apply sequence numbers to individual records.

Implementation Guidelines

The sequence generator mapplet is created to address the need to generate sequence numbers more efficiently, with fewer sequence number gaps, as well as allow the generation of a larger maximum sequence number. The mapplet allows the developer to intelligently generate the sequence number of an incoming record based on whether the record will be treated as a true update or as an insert.

The sequence generator mapplet must be used after the lookup object that determines the
maximum sequence id from the target table in all cases. The result of that lookup will be the starting point of the sequence numbers generated in the mapplet. If the developer chooses to create a mapping that performs true updates (such as Type 1 Slowly Changing Dimensions), then another lookup must be included that determines whether the record already exists in the target dimension and must be placed before the sequence generator mapplet. In addition, the developer must pass in a flag to set whether all records will be treated as inserts.

The key concepts illustrated by this mapplet can be found in the mapplet expression object, exp_GENERATE_SEQUENCE_NUMBER.

**exp_GENERATE_SEQUENCE_NUMBER**: This expression uses variable logic to determine the sequence number for the incoming record. Based on the highest sequence number from the target table, it determines the next sequence number for incoming records. The sequence number is incremented only when a record would be inserted (i.e. the LKP_SEQ_ID is null) or when the UPD_AS_INS flag is set to ‘Y’.

**Pros**

- This mapplet can generate sequence numbers without having gaps in sequence numbers.
- This mapplet can accommodate an update as update strategy and an update as insert strategy by passing in one flag.

**Cons**

- This sequence generator mapplet should not be used in sessions that are running concurrently to load the same target table, as there is no way to synchronize sequences between the sessions.
2.10 Best Build Logic

Purpose

Best Build Logic

Usage

Uses PowerMart / PowerCenter variable logic to collapse values from source records with duplicate keys into one master record.

Download

m_Best_Build_Logic_v61.XML
m_2_10_Best_Build_Logic_v711.XML

Challenge Addressed

Often it is necessary to 'build' a target record from several source records. One way to do this is to use variable logic to determine whether data in the incoming record should be added to data from previous records to form a "master" record.

Overview
There are a few different methods of building one record out of several. Often this can be accomplished by simply using an aggregator. Alternatively, variable logic can be utilized. This mapping template illustrates a method of using PowerMart / PowerCenter expression variables to build one record out of several.

In this example, the source table may provide several records with the same CUSTOMER_ID. This occurs when the length of a comment for a particular CUSTOMER_ID is longer than the source table's COMMENT field, and the source system must create several records for that CUSTOMER_ID to store the entire comment (a piece of the comment is stored in each record). This template illustrates functionality for concatenating all the appropriate COMMENT fields into one target record with a larger COMMENT field.

For example, source records:

CUSTOMER_ID, SEQUENCE_NUM, COMMENT
1,1,aaa
1,2,bbb
2,1,ccc
2,2,ddd
2,3,eee
3,1,fff

Would turn into target records:

CUSTOMER_ID, COUNT, COMMENT
1,2,aaabbb
2,3,ccccdddeee
3,1,fff

Implementation Guidelines
This mapping achieves its objectives by doing 3 things:

- Sorting the input data (with a sorter transformation) by CUSTOMER_ID and then SEQUENCE_NUM, both in ascending order.
- Using variable ports to determine whether the incoming CUSTOMER_ID is the same as the previous one. If so, then the COMMENT field from the current and previous records should be concatenated. Otherwise the COMMENT variable should be reset to the current COMMENT field.
- Using an aggregator transformation to count the number of records contributing to each outgoing record, and to return the final, "master" record (after all appropriate concatenations have taken place and the COMMENT field is complete).

**Pros**

- Using port variables allows for great flexibility on the condition on which the final record can be built. This is a very simple example, but the same idea can be used for a much more complex situation.

**Cons**

- The developer needs to be very careful of the order of evaluation of ports. The variable logic used here hinges on the fact that ports get evaluated in the following order:
  1. All Input ports (in the order they appear in the transformation);
  2. All Variable ports (in the order they appear in the transformation);
  3. All Output ports (in the order they appear in the transformation).

**2.11 Comparing Values Between Records**

**Purpose**
Comparing Values Between Records

**Usage**

Uses PowerMart / PowerCenter Local Variables to compare values between consecutive records.

**Download**

- m_Compare_Values_Between_Records_v61.XML
- m_2_11_Compare_Values_Between_Records_v711.XML

**Challenge Addressed**

When data in a source field is evaluated it is sometimes necessary to compare values in one record to values in a previous record.

**Overview**

There are several scenarios when this action is necessary. Sometimes all that is needed is an aggregator, however, our example is more complex.

**Implementation Guidelines**

Local variables add flexibility to transformations. They allow you to separate complex
logic into simpler pieces, allow the use of pass through variables to be used in other calculations, and hold their values over multiple rows of processing. It is this last property that we will be using. This example involves calculating the number of days since a customer’s last payment. Our example company wishes to track the number of days between customer payments for an aggregate metric.

The key points of this mapping will be the source qualifier SQ_DEBIT and the expression exp_Date_Compare. The source for the mapping will be the DEBIT table that holds all customer transactions for the last quarter. The target table is PAYMENTS. The source qualifier is important, as it is necessary to order the data on the payment date in ascending order. This will allow the variable to compare the appropriate values.

In the Source Qualifier transformation SQ_DEBIT, the SQL Override is as follows:

```
SELECT DEBIT.Acct_Num,  
DEBIT.Amount,  
DEBIT.Date_Received  
FROM DEBIT  
GROUP BY DEBIT.Acct_Num  
ORDER BY DEBIT.Date_Received
```

In the Expression transformation exp_Date_Compare, the local variable is used to compare dates between rows.

The Expression transformation exp_Date_Compare is passed the customer’s account number, payment amount, and the date the payment was received. The source qualifier has grouped all the account numbers together and ordered their date received in ascending order. The expression must do two things to make the proper calculation. First, it must determine when there is a change in the account number being evaluated. Second, it must store the previous transaction’s date within an account number. The initial row for a specific account must have Days_Between_Payments set to zero.

First the expression will receive its input data. This is held in Acct_Num, Amount, and Date_Received. At this point it is critical to understand the order of evaluation within ports for PowerMart / PowerCenter. Ports are evaluated according to the following rules:

- Input ports
- Variable ports
- Output ports
Within a port type (i.e. variable ports) ports are evaluated from top to bottom as they appear within the transformation.

With these rules in mind the variable ports are evaluated next from top to bottom. The port \texttt{v\_new\_date} will be set to the value of the input port \texttt{Date\_Received}. The port \texttt{v\_Days\_Between\_Payments} is now calculated using the expression:

\[
\text{IIF(Acct\_Num = v\_Acct\_Num, DATE\_DIFF(v\_new\_date, v\_prev\_date, 'DD'), 0)}
\]

In order to understand how this expression works we must first see what happens after it is evaluated. Next the port \texttt{v\_prev\_date} is set to the current row’s \texttt{Date\_Received}. Then the port \texttt{v\_Acct\_Num} is set to the current row’s \texttt{Acct\_Num}. Finally, the port \texttt{Days\_Between\_Payments} is populated with the result of the calculation in the port \texttt{v\_Days\_Between\_Payments}.

Keeping in mind the port processing order we can now revisit the date calculation above. The argument of the IIF statement compares the account number of the current row with that of the previous row. Remember, this expression is evaluated prior to the updating of the value in \texttt{v\_Acct\_Num}. So, at this point, it contains the value of the previously processed row. If they are not equal it indicates that this is the first row processed for a specific account. If they are equal the date comparison is performed. The port \texttt{v\_new\_date} has been updated to contain the current row’s date while \texttt{v\_prev\_date} still holds the date of the previous row. The function \texttt{DATE\_DIFF} returns the length of time, measured in the increment you specify (years, months, days, hours, minutes, or seconds), between two dates. The Informatica Server subtracts the second date from the first date and returns the difference.

The result is a field populated by a calculation using values across rows.

**Pros**

- Allows for added power and flexibility in implementing complex logic.

**Cons**

- Requires the ability to order the data in a proper manner for evaluation.
2.14 Pipeline Partitioning Mapping Template

Purpose

Pipeline Partitioning

Usage

Use the enhanced pipeline partitioning features in PowerCenter 6.0 to improve the performance of an individual mapping/workflow.

Download

m_2_14_Partitioning_Example_v61.zip
m_2_14_Partitioning_Example_v711.XML

Challenge Addressed

Performance tuning is an integral part of the data integration process. It is essential to optimize the performance of each individual piece of the application so that optimal performance can be achieved. After the application and databases have been tuned for maximum single partition performance, you may find that the system is under-utilized. PowerCenter 6.0 provides enhanced pipeline partitioning capabilities that when appropriately configured, can increase pipeline parallelism, resulting in greater throughput and a significant performance improvement.
Overview

The purpose of this template is to demonstrate how taking advantage of the pipeline partitioning functionality available in PowerCenter 6.0 can increase overall session performance. This individual template presents an excellent opportunity to see how increasing the number of partitions and modifying the type of partitioning schemes used and the location of partition points within the mapping can be an effective tuning technique. The mapping represented in this template reads student report card information from three flat files of varying sizes. It processes records for ‘Active’ students only, calculates their GPAs, and loads the data into a partitioned relational target.

Implementation Guidelines

In order to partition the pipeline to accommodate the three source files and read the data simultaneously, the number of partitions was increased from 1 to 3. By processing the data sets in parallel, true pipeline parallelism can be achieved.

The implementation guidelines in this template focus on session settings for the mapping illustrated above. The following screenshots illustrate session property sheets from the PowerMart/PowerCenter Workflow Manager utility.
Due to the varying size of the source files, the workloads will be unevenly distributed across the partitions. Setting a partition point at the Filter transformation and using round robin partitioning will balance the workloads going into the filter. Round Robin partitioning will force the Informatica Server to distribute the data evenly across the partitions.
As the data enters both the Sorter and Aggregator transformations, there is a potential for overlapping aggregate groups. To alleviate this problem, hash partitioning was used at the Sorter transformation partition point to route the data appropriately. This partition type will group the data based on the designated key designated in the Sorter transformation, keeping all of the data for each student together in the same partition, thus optimizing the sort and aggregation.

Since the target table itself is partitioned by key range, the partition type at the target instance was set to key range. The ‘OVERALL_GPA’ field was chosen as the partition key to mimic the physical partitions on the table. The Informatica Server uses this key to align the data with the physical partitions in the target table by routing it to the appropriate partition.

Pros

- Performance can be greatly enhanced by using Informatica’s partitioning.
Cons

- If the source and target databases are not fully optimized, then partitioning a mapping may not increase session performance.

### 2.18 Using Update Strategy to Delete Rows

**Purpose**

Deleting Rows using Update Strategy

**Usage**

Uses an Update Strategy Transformation to flag records for deletion from the target table.

**Download**

- [m_Delete_Rows_With_Update_Strategy_v61.XML](#)
- [m_2_18_Delete_Rows_With_Update_Strategy_v711.XML](#)

**Challenge Addressed**
When data in a target table is to be deleted, it is necessary to consider various delete strategies and adapt the best strategy for a business scenario. The delete strategy should be efficient, easy to implement, and flexible enough to allow for easy modification of the delete criteria in the future.

**Overview**

There are many methods of deleting records from a target table. This mapping template illustrates a method of using an Update Strategy transformation in a PowerMart / PowerCenter mapping to flag records that need to be deleted from the target table.

**Implementation Guidelines**

The mapping has an update strategy transformation just before the target to flag individual rows for deletion. In the Properties tab of the Update Strategy transformation, the ‘Update Strategy Expression’ property is set to DD_DELETE or the Numeric Value 2, to mark the rows for deletion. Using DD_DELETE makes it easier to debug mappings than using the numeric 2. The following are a few points to keep in mind while using update strategy to delete records:

- Make sure you select the ‘Delete’ option in the target properties in a session (This option is selected by default when you create a session).
- Rows can be deleted from a target table using various methods. For example, you can set the session property ‘Treat Rows As’ to ‘Delete’ to mark all rows for deletion. But this approach makes it impossible to flag only a few rows for deletion based on a criterion.
- You can also use a stored procedure to perform the deletion and call the stored procedure from within a PowerMart/PowerCenter mapping instead of using an Update Strategy transformation. However using an Update Strategy transformation will result in better performance over using stored procedures.
- Use a conditional expression to mark only selected rows for deletion. For example, you can use an expression such as IIF(STORE_ID < 100, DD_DELETE, DD_REJECT) to delete only the rows that have STORE_ID less than 100.
- If you use an Aggregator transformation after the update strategy transformation, beware of the aggregator behavior. For example, if the row marked for deletion is used in a ‘Sum’ operation, the aggregator actually subtracts the value appearing in this row.
Informatica Mappings

- Make sure that the user running this session has privileges to delete records from the target table in the database.
- Make sure that there are no constraints at the database level on this table that prevents records from being deleted.
- If you are deleting a large number of rows, then you should check to make sure that the rollback segment is big enough to hold that many rows.
- If the database does page level locking and your session tries to delete a row from a page that is already locked, the session will fail.

**Pros**

- Flexibility of using a conditional expression.
- Performance gain over a stored procedure for deleting records.

**Cons**

- The primary key constraint must exist in the target definition in the repository. If the primary key does not exist in the target table, then you have to create one in the designer so that the mapping has a where clause for the delete query (Note: it is not necessary to create the key in the underlying database).

**2.20 Loading Heterogenous Targets**

**Purpose**

Loading Heterogeneous Targets

**Usage**

Uses PowerMart / PowerCenter Mapping ability to load Heterogeneous Targets in one mapping.
Challenge Addressed

There are many occasions when the developer requires the ability to load targets that are from different systems. Examples of heterogenous targets include tables from different database management systems, and a flat file target and database target in the same mapping.

Overview

This mapping template illustrates a method of using a PowerMart / PowerCenter mapping ability to load different targets.

Implementation Guidelines

The mapping targets two tables. The first table resides on an Oracle schema and is the true target.

The second target table resides in a SQL Server environment and will contain the data validation errors.

The mapping in this template uses the Expression transformation `exp_FLAG_ERRORS` to validate each row of data that is passed. Should an error occur, the row will target the
CUSTOMERS_ERRORS table, otherwise the row will proceed to the CUSTOMERS_DIM table.

The Router transformation rtr_CUSTOMER_DIM_ERRORS uses the value from the error flag expression determined in exp_FLAG_ERRORS and will direct the row to the proper target.

While this mapping illustrates the ability to write to heterogenous targets, it is important to know that the locations of heterogenous targets can be defined at the session level.

Pros

- Developers are no longer limited to one target

Cons

- No cons for heterogeneous targeting.