Microsoft EDW Architecture, Guidance and Deployment Best Practices

# Chapter 3 - Data Integration

**By Microsoft Corporation**

**Acknowledgements:**

**Contributing writers from Solid Quality Mentors**: Larry Barnes, Erik Veerman

**Technical reviewers from Microsoft**: Ross LoForte, Benjamin Wright-Jones, Jose Munoz

**Contributing editors from Solid Quality Mentors**: Kathy Blomstrom

**Published:**

**Applies to:** SQL Server 2008 R2

[Chapter 3 - Data Integration 1](#_Toc274371604)

[Introduction 4](#_Toc274371605)

[Data Integration Overview 5](#_Toc274371606)

[Data Integration Patterns 6](#_Toc274371607)

[Which Pattern Should You Use? 8](#_Toc274371608)

[Roles and Responsibilities 9](#_Toc274371609)

[Data Integration Concepts 10](#_Toc274371610)

[Consolidation, Normalization, and Standardization 10](#_Toc274371611)

[Data Integration Paradigms (ETL and ELT) 16](#_Toc274371612)

[ETL Processing Categories 20](#_Toc274371613)

[Incremental Loads 21](#_Toc274371614)

[Detecting Net Changes 23](#_Toc274371615)

[Data Integration Management Concepts 25](#_Toc274371616)

[Batch Processing and the Enterprise ETL Schedule 31](#_Toc274371617)

[ETL Patterns 33](#_Toc274371618)

[Destination Load Patterns 33](#_Toc274371619)

[Versioned Insert Pattern 33](#_Toc274371620)

[Update Pattern 36](#_Toc274371621)

[Versioned Insert: Net Changes 37](#_Toc274371622)

[Data Quality 38](#_Toc274371623)

[Data Quality Scenario 39](#_Toc274371624)

[Data Exceptions 40](#_Toc274371625)

[Data Stewardship and Validation 42](#_Toc274371626)

[Data Profiling 44](#_Toc274371627)

[Data Cleansing 45](#_Toc274371628)

[Data Reconciliation 48](#_Toc274371629)

[Lineage 49](#_Toc274371630)

[ETL Frameworks 52](#_Toc274371631)

[ETL Framework Components 53](#_Toc274371632)

[Users and Interfaces 54](#_Toc274371633)

[Configurations 55](#_Toc274371634)

[Logging 56](#_Toc274371635)

[Master Package 60](#_Toc274371636)

[Execution Package 65](#_Toc274371637)

[Package Storage Options 67](#_Toc274371638)

[Backing Out Batches 69](#_Toc274371639)

[More ETL Framework Information 71](#_Toc274371640)

[Data Integration Best Practices 71](#_Toc274371641)

[Basic Data Flow Patterns 71](#_Toc274371642)

[Surrogate Keys 78](#_Toc274371643)

[Change Detection 84](#_Toc274371644)

[De-duping 89](#_Toc274371645)

[Dimension Patterns 92](#_Toc274371646)

[Fact Table Patterns 96](#_Toc274371647)

[Data Exception Patterns 99](#_Toc274371648)

[SSIS Best Practices 103](#_Toc274371649)

[The Power of Data Flow Scripting 103](#_Toc274371650)

[Destination Optimization (Efficient Inserts) 107](#_Toc274371651)

[Partition Management 109](#_Toc274371652)

[SSIS Scale and Performance 110](#_Toc274371653)

[Source Control 112](#_Toc274371654)

[Conclusion and Resources 112](#_Toc274371655)

[Resources 112](#_Toc274371656)

## Introduction

Data integration is responsible for the movement of data throughout the data warehouse and the transformation of that data as it flows from a source to its next destination.

Today’s reality is that a large percentage of a data warehouse’s total cost of ownership (TCO) is related to post development integration costs—that is, the ongoing costs of loading source data into the data warehouse and distributing data from the data warehouse to downstream data stores. The daily, and in some cases intraday, process of loading data and validating the results is a time-consuming and repetitive process.

The resources required to support this process increase over time due to:

* Increases in data volumes
* The growth of data warehouse integration processes and the long lifetime of the processes once the data warehouse is in production
* The lack of best software engineering practices when developing integration solutions
* The growing need for real-time or near real-time data

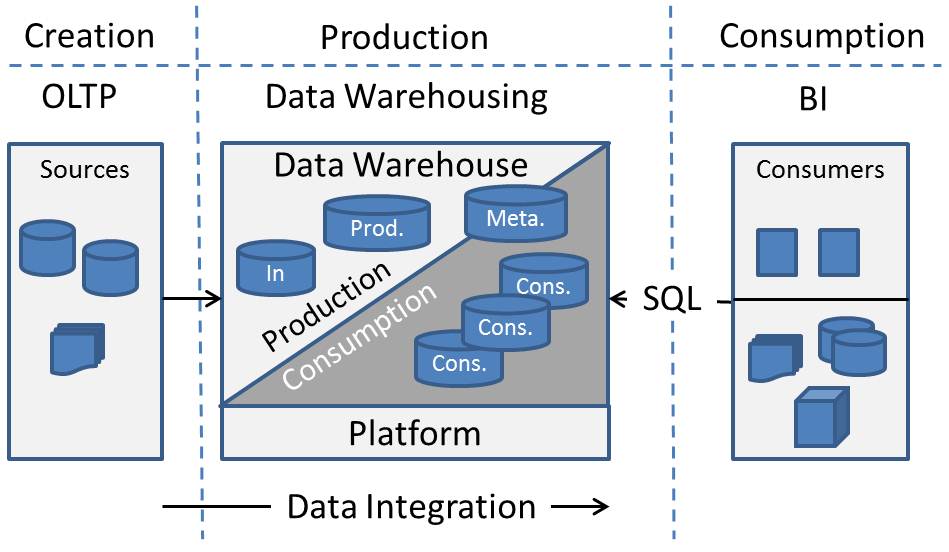
This chapter’s objective is to help reduce data integration TCO for data warehouses implemented on the Microsoft SQL Server platform by presenting a set of integration patterns and best practices found in successful Microsoft-centric data warehouse implementations today.

This chapter covers the following topics from the perspective of the noted intended audiences:

* Data integration overview and challenges
* ETL concepts and pattern (audience: data integration team)
* Data quality (audience: ETL operations, ETL developers and Data stewards)
* ETL Frameworks (audience: ETL developers and Data architects)
* Data Integration best practices (audience: ETL developers)
* SSIS best practices (audience: ETL developers)
* Conclusion and resources: Links to Web content

## Data Integration Overview

Data integration is responsible for moving, cleansing and transforming set-based data—often very large data sets—from source(s) into the Production data area and then into the Consumption data area as shown in Figure 3-1.



**Figure 3-1**: The role of data integration in a data warehouse project

The requirements for the data integration component include:

* **Trust** – Business consumers must be able to trust the results obtained from the data warehouse.
* **One version of the truth** – Consolidating disparate sources into an integrated view supports business consumers’ need for an enterprise-level view of data.
* **Current and historical views of data** – The ability to provide both a historical view of data as well as a recent view supports key business consumer activities such as trend analysis and predictive analysis.
* **Availability** – Data integration processes must not interfere with business consumers’ ability to get results from the data warehouse.

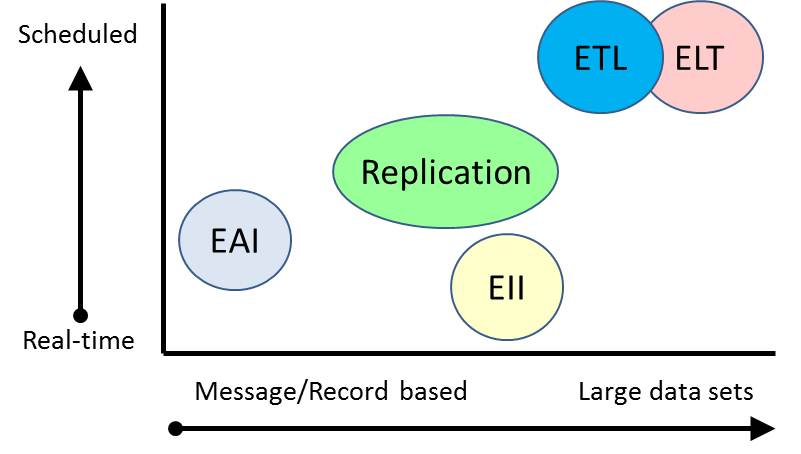
The challenges for the data integration team in support of these requirements include:

* **Data quality** – The data integration team must promote data quality to a first-class citizen.
* **Transparency and auditability** – Even high-quality results will be questioned by business consumers. Providing complete transparency into how the data results were produced will be necessary to allay business consumers’ concerns around data quality.
* **Tracking history** – The ability to correctly report results at a particular period in time is an ongoing challenge, particularly when there are adjustments to historical data.
* **Reducing processing times** – Efficiently processing very large volumes of data within ever shortening processing windows is an ongoing challenge for the data integration team.

### Data Integration Patterns

The industry has several well-known data integration patterns to meet these requirements and solve these challenges, and it’s important for data warehouse practitioners to use the correct pattern for their implementation. How do you determine which of these patterns you should use for your data integration needs?

Figure 3-2 positions the different integration options that are available.



**Figure 3-2**: Integration patterns

The two axes in Figure 3-2 represent the main characteristics for classifying an integration pattern:

* **Timing** – Data integration can be a real-time operation or can occur on a scheduled basis.
* **Volumes** – Data integration can process one record at a time or data sets.

The primary integration patterns are:

* **Enterprise Information Integration (EII)** – This pattern loosely couples multiple data stores by creating a semantic layer above the data stores and using industry-standard APIs such as ODBC, OLE-DB, and JDBC to access the data in real time.
* **Enterprise Application Integration (EAI)** – This pattern supports business processes and workflows that span multiple application systems. It typically works on a message-/event-based model and is not data-centric (i.e., it is parameter-based and does not pass more than one “record” at a time). Microsoft BizTalk is an example of an EAI product.
* **Extract, Transform, and Load (ETL)** – This pattern extracts data from sources, transforms the data in memory and then loads it into a destination. SQL Server Integration Services (SSIS) is an example of an ETL product.
* **Extract, Load, and Transform (ELT)** – This pattern first extracts data from sources and loads it into a relational database. The transformation is then performed within the relational database and not in memory. This term is newer than ETL but, in fact, was the method used in early data warehouses before ETL tools started to emerge in the 1990s.
* **Replication** – This is a relational database feature that detects changed records in a source and pushes the changed records to a destination or destinations. The destination is typically a mirror of the source, meaning that the data is not transformed en route from source to destination.

Data integration, which frequently deals with very large data sets, has traditionally been scheduled to run on a nightly basis during off hours. In this scenario, the following has held true for the different patterns:

* EII is not commonly used in data warehouses because of performance issues. The size and data volumes of data warehouses prohibit the real-time federation of diverse data stores, which is the technique employed by the EII pattern.
* EAI is not used in data warehouses because the volume of the data sets results in poor performance for message-/event-based applications.
* ETL is the most widely used integration pattern for data warehouses today.
* ELT is seen mostly in legacy data warehouse implementations and in very large data warehouse implementations where the data volumes exceed the memory required by the ETL pattern.
* Replication, used to extract data from sources, is used in conjunction with an ETL or ELT pattern for some data warehouse implementations.
  + The decision to use replication can be based on a variety of factors, including the lack of a last changed column or when direct access to source data is not allowed.

However, because of the growing need for real-time or near real-time reporting outside of the line of business (LOB) database, organizations are increasingly running some data integration processes more frequently—some close to real time. To efficiently capture net changes for near real-time data integration, more and more companies are turning to the following solutions:

* Replication to push data out for further processing in near real time when the consumer requires recent data (replication is also useful when the source system doesn’t have columns that the ETL or ELT tool can used to detect changed records)
* Relational databases’ additional capabilities to detect and store record changes, such as SQL Server 2008 Change Data Capture (CDC) which is based upon the same underlying technology used by replication.
* Incremental change logic within an ETL or ELT pattern (as long as the source table has a date or incrementing column that can be used to detect changes)

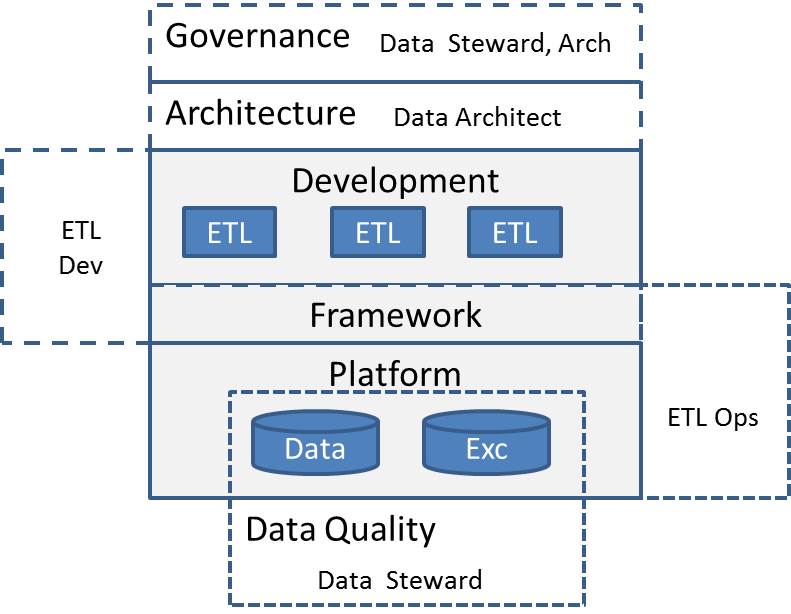
### Which Pattern Should You Use?

Typically, a data warehouse should use either ETL or ELT to meet its data integration needs. The costs of maintaining replication, especially when re-synchronizing the replication process is required, makes it a less attractive alternative for extracting data from sources. However, hybrid approaches such as ETL/ELT combined with source system net-change detection capabilities may be required for near real-time data.

Throughout the rest of this document, we will use ETL for most patterns and best practices and explicitly point out where ELT and source system net-change detection are applicable.

### Roles and Responsibilities

Chapter 1 outlined the team’s roles and responsibilities within the entire data warehouse effort. Figure 3-3 shows the responsibilities for the data steward, data architect, ETL developer, and ETL operations roles for a data warehouse’s data integration component.



**Figure 3-3**: Data Integration Team roles and responsibilities

The responsibilities of the different team roles are:

* **Governance** – Data stewards and data architects are members of the data warehouse Governance team.
* **Architecture** – The data architect is responsible for the data warehouse architecture, including but not limited to the platform architecture, best practices and design patterns, oversight of frameworks and templates, and creating naming conventions and coding standards.
* **Development** – ETL developers are responsible for designing and developing ETL packages and the underlying ETL framework. In addition, ETL developers are typically called when there’s an issue with the ETL processes (errors) or with the data results (exceptions).
* **ETL Operations** – The ETL operations team is responsible for deploying ETL solutions across the different environments (e.g., Dev, Test, QA, and Prod) and the day-to-day care and feeding of the ETL solutions once in production.
* **Data Quality** – Data stewards are responsible for data quality.

Note that the objective of this team setup is to minimize the TCO of daily data warehouse ETL activity. It’s important that ETL operations and data stewards have the necessary tools to diagnose errors and exceptions. Otherwise, the time required to diagnose and fix errors and exceptions increases, and ETL developers will be pulled into all error and exception activity, reducing the time they can spend on new applications. More important, this constant firefighting leads to burnout for all parties.

The rest of this chapter will expand on the patterns and best practices you can use to reduce data integration’s TCO, starting with key ETL concepts.

## Data Integration Concepts

This section introduces the following key Data integration concepts, which we will look at in more detail as we present best practices later in this chapter:

* Consolidation, normalization, and standardization
* Data integration paradigms (ETL and ELT)
* ETL processing categories
* Incremental loads
* Detecting net changes
* Data integration management concepts

### Consolidation, Normalization, and Standardization

Data integration processes typically have a long shelf life—it’s not uncommon for an ETL process to be in production for more than 10 years. These processes undergo many revisions over time, and the number of data processes grows over time as well. In addition, different development teams often work on these processes without coordination or communication.

The result is duplication of effort and having multiple ETL processes moving the same source data to different databases. Each developer often uses different approaches for common data integration patterns, error handling, and exception handling. Worse yet, the lack of error and exception handling can make diagnosing error and data exceptions very expensive. The absence of consistent development patterns and standards results in longer development cycles and increases the likelihood that the ETL code will contain bugs.

Longer development times, inconsistent error and exception handling, and buggy code all contribute to increasing data integration TCO. Well-run data integration shops have recognized that time spent up-front on ETL consistency is well worth the effort, reducing both maintenance and development costs.

ETL consistency is achieved through three practices—consolidation, normalization, and standardization:

* **Consolidation** is the practices of managing the breadth of processes and servers that handle ETL operations. This includes both the operations that perform ETL, such as SSIS packages, and the databases and files stores that support the ETL, such as Data In and Production databases.

If your environment has dozens of databases and servers that do not generate data but merely copy and transform data (and often the same data!), you are not alone. However, you likely spend a lot of time managing these duplicate efforts.

* **Normalization** involves being consistent in your ETL processing approach. You can develop an ETL package in many different ways to get to the same result. However, not all approaches are efficient, and using different approaches to accomplish similar ETL scenarios makes managing the solutions difficult.

Normalization is about being consistent in how you tackle data processing—taking a “normal” or routine implementation approach to similar tasks.

* **Standardization** requires implementing code and detailed processes in a uniform pattern. If normalization is about processes, standardization is about the environment and code practices. Standardization in ETL can involve naming conventions, file management practices, server configurations, and so on.

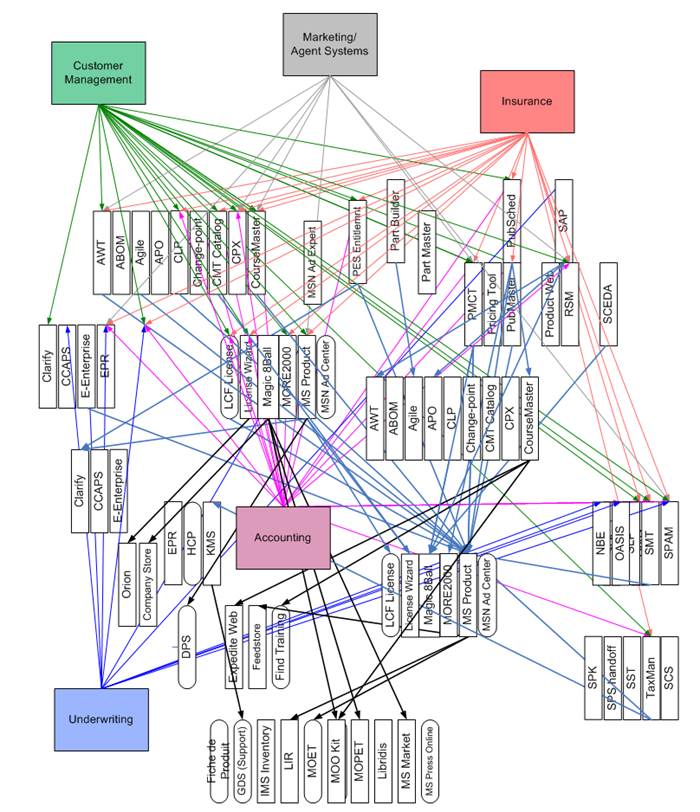
Data integration standards, like any standards, need to be defined up-front and then enforced. ETL developers and architects should implement the standards during development. You should never agree to “implement standards later.”

Let’s look at each of these practices more closely.

**Consolidation**

Suppose you work for a moderately large organization, such as an insurance company. The core systems involve policy management, claims, underwriting, CRM, accounting, and agency support. As with most insurance companies, your organization has multiple systems performing similar operations due to industry consolidation and acquisitions or to support the various insurance products offered. The supporting LOB systems or department applications far outnumber the main systems because of the data-centric nature of insurance. However, many of the systems require data inputs from the core systems, making the Web of information sharing very complicated.

Figure 3-4 shows the conceptual data layout and connection between the systems.



**Figure 3-4**: System dependency scenario for an insurance company

Each line in Figure 3-4 involves ETL of some nature. In some cases, the ETL is merely an import and export of raw data. Other cases involve more complicated transformation or cleansing logic, or even the integration of third-party data for underwriting or marketing. If each line in the diagram were a separate, uncoordinated ETL process, the management and IT support costs of this scenario would be overwhelming.

The fact is that a lot of the processes involve the same data, making consolidation of the ETL greatly beneficial. The normalization of consistent data processes (such as the summary of claims data) would help stabilize the diversity of operations that perform an aggregation. In addition, the sheer number of ETL operations involved between systems would benefit from a consolidation of servers handling the ETL, as well as from the normalization of raw file management and standardization of supporting database names and even the naming conventions of the ETL package.

**Normalization**

Because normalization applies to being consistent about the approach to processes, ETL has several layers of normalization. In fact, a large part of this chapter is dedicated to normalization, first as we look at common patterns found in ETL solutions and then later as we cover best practices.

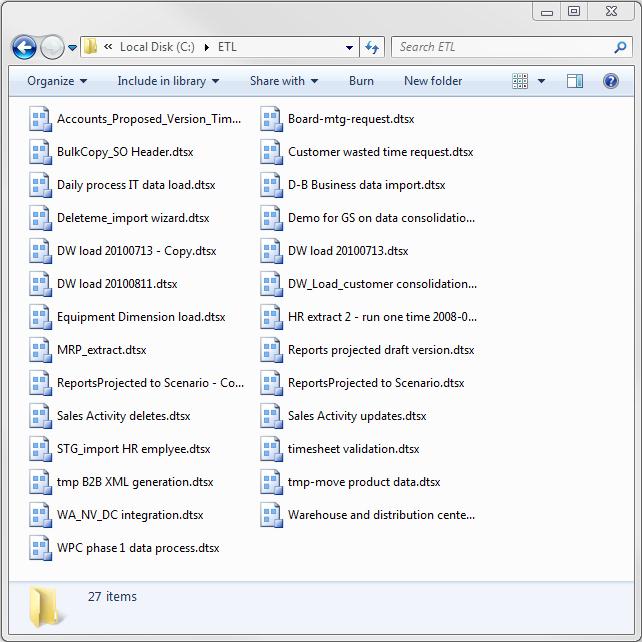
Normalization in ETL includes but is not limited to:

* Common data extraction practices across varying source systems
* Consistent approach to data lineage and metadata reporting
* Uniform practices for data cleansing routines
* Defined patterns for handling versioning and data changes
* Best practice approaches for efficient data loading

**Standardization**

Although it sounds basic, the first step toward standardization is implementing consistent naming conventions for SSIS packages. Consider the screen shot in Figure 3-5, which represents a small slice of ETL packages on a single server. For someone trying to track down an issue or identify the packages that affect a certain system, the confusion caused by the variety of naming styles creates huge inefficiencies.

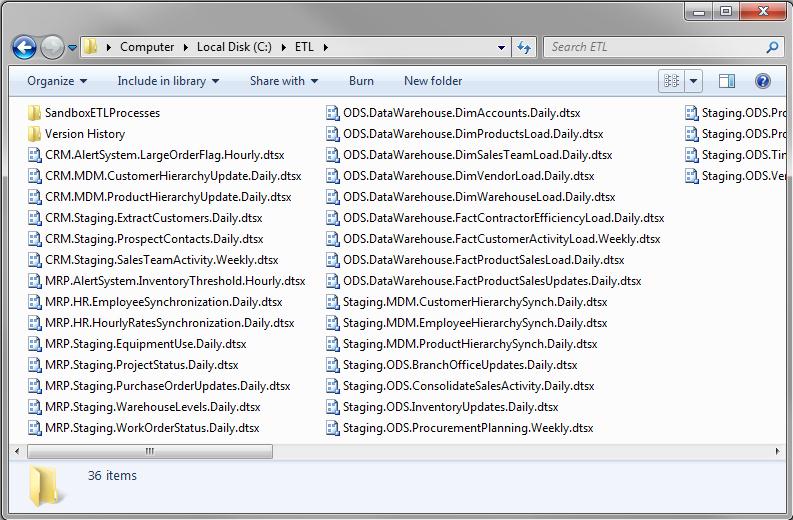
It is hard enough for an experienced developer or support engineer to remember all the names and processes, but add a new developer or IT support person to the mix, and the challenges increase.

****

**Figure 3-5**: Example of non-standard package naming

In contrast, Figure 3-6 shows another slice of packages that are named consistently. These packages follow a standard naming convention:

[Source System].[Destination System].[OperationDescription].[ExecutionFrequency].dtsx

****

**Figure 3-6**: Standard package naming simplifies management and troubleshooting

However, the point isn’t about this particular naming convention, but about the need to define and follow a consistent standard, whatever is appropriate in your environment. The ETL Framework section presents additional standards including SSIS package templates used as a foundation for all SSIS ETL development.

**Benefits of the Big 3**

In summary, the benefits of consolidation, normalization, and standardization include:

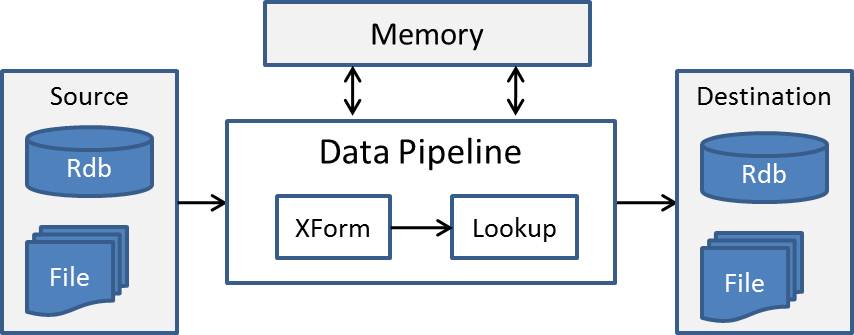
* **Improved process governance** –ETL consistency and consolidation help you achieve better overall enterprise data stewardship and effective operations.
* **Better resource transition** – As people move in and out of a support or development environment, they can focus their energies on the core business problem or technical hurdle, rather than trying to figure out where things are and what they do.
* **Enhanced support and administration** – Any ETL support team will benefit from following consistent patterns and consolidation, especially if the support organization is in a different location (such as in an off-shore operations management scenario).
* **More effective change management** – The ability to nimbly handle system changes is enhanced when you can clearly see what processes are running and those processes are consistently implemented.
* **Reduced development costs** – The implementation of development standards reduces the cost of development because in the long run, developers are able to focus more on the business requirement they are coding when they’re given clear direction and process.

Failure to address consolidation, normalization, and standardization up-front—or to stabilize an existing ETL environment that is deficient in any or all of these areas—will make your job architecting, developing, or managing data integration for your data warehouse more complicated. Each of the benefits above can be turned into a drawback without the proper standards and processes in place: difficult process management and administration, ineffective knowledge transfer, and challenges in change management of processes and systems.

The ETL Frameworks section below presents a set of package templates that provide a solid foundation for ETL developers.

### Data Integration Paradigms (ETL and ELT)

ETL products populate one or more destinations with data obtained from one or more sources. The simplest pattern is where one source loads one destination, as illustrated in Figure 3-7.



**Figure 3-7**: Simple ETL data flow

The processing steps are as follows:

1. The ETL tool retrieves data sets from the source, using SQL for relational sources or another interface for file sources.
2. The data set enters the data pipeline, which applies transformations to the data one record at a time. Intermediate data results are stored in memory.
3. The transformed data is then persisted into the destination.

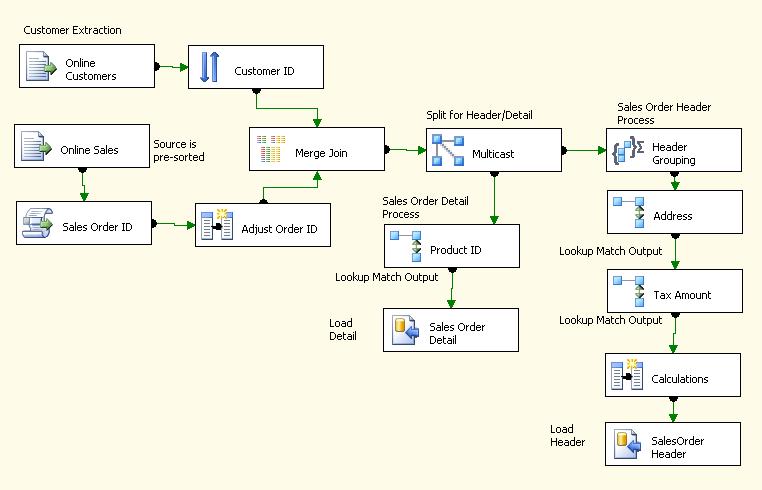
Advantages to this process are that:

* Procedural programming constructs support complex transformations.
* Storing intermediate results in memory is faster than persisting to disk.
* Inserts are efficiently processed using bulk-insert techniques.

However, the disadvantages include the following:

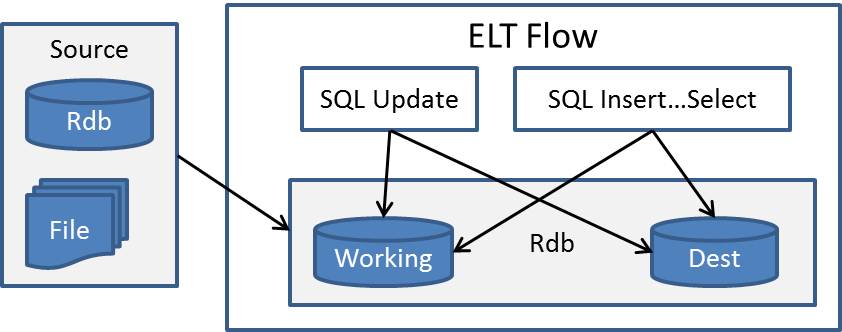
* Very large data sets could overwhelm the memory available to the data pipeline.
* Updates are more efficient using set-based processing—meaning using one SQL UPDATE statement for all records, not one UPDATE per each record.

Figure 3-8 shows an example of an SSIS data flow that performs transformation processes (joining, grouping, calculating metrics, and so on) in the pipeline. This data flow has the advantage of leveraging the memory resources of the server and can perform many of the transformation tasks in parallel. However, when memory is limited or the data set needs to entirely fit in memory, the processes will slow down.



**Figure 3-8**: SSIS data flow example

Remember that ELT—Extract, Load, and Transform—also moves data from sources to destinations. ELT relies on the relational engine for its transformations. Figure 3-9 shows a simple example of ELT processing.



**Figure 3-9**: Simple ELT data flow

The processing steps in the ELT data flow are as follows:

1. Source data is loaded either directly into the destination or into an intermediate working table when more complex processing is required. Note that transformations can be implemented within the source SQL Select statement.
2. Transformations are optionally applied using the SQL Update command. More complex transformations may require multiple Updates for one table.
3. Transformations and Lookups are implemented within the SQL Insert…Select statement that loads the destination from the working area.
4. Updates for complex transformations and consolidations are then applied to the destination.

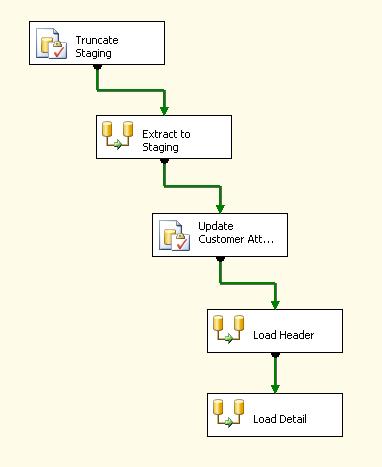
The advantages of this process include the following:

* The power of the relational database system can be utilized for very large data sets. Although, note that this processing will impact other activity within the relational database.
* SQL is a very mature language that translates into a greater pool of developers than ETL tools would.

However, you need to consider these disadvantages:

* As just noted, ELT places a greater load on the relational database system.
* You will also see more disk activity because all intermediate results are stored within a table, not memory.
* Implementing transformations and consolidations using one or more SQL Updates is more inefficient than the ETL equivalents, which make only one pass through the data and apply the changes to the destination using a single SQL statement rather than multiple ones.
* Complex transformations can exceed the capabilities of the SQL Insert and Updates statements because transformations occur at the record level not the data set level. When this occurs, SQL cursors are used to iterate over the data set, which results in decreased performance and hard-to-maintain SQL code.
* For a given transformation, the processes applied are often serialized in nature and add to the overall processing time.

Figure 3-10 shows the SSIS control flow used in more of an ELT-type operation. You can identify ELT-type operations by their multiple linear tasks, which perform either Execute SQL Tasks or straight data loads using a few working tables.



**Figure 3-10**: SSIS control flow ELT process

**Which Should You Use for Your Implementation?**

The decision about whether to use an ETL or ELT pattern for a SQL Server data integration solution is generally based on the following considerations.

**Use ETL when…**

* Working with flat files and non relational sources. ETL tools have readers which can access non-relational sources like flat files and XML files. ELT tools leverage the SQL language which requires that the data be first loaded into a relational database.
* This is a new data integration project or the current first-generation implementation is hard to manage and maintain. The visual workflows for tasks and data flows make the process easier to understand by non-developers.
* The transformations are complex. ETL’s ability to apply complex transformations and business rules far exceeds the abilities of one set-based SQL statement. Many legacy ELT solutions have become unmanageable over time because of cursor-based logic and multiple Update operations used to implement complex transformations.

**Use ELT when…**

* The data volumes being processed are very large. Huge data sets may exhaust the available memory for an ETL approach. Remember that the ETL data pipeline uses in-memory buffers to hold intermediate data results.
* The source and destination data is on the same server and the transformations are very simple. A SQL-centric ELT solution is a reasonable choice when the current database development team is not trained on SSIS. But keep in mind that complex transformations can easily translate into poorly performing, unmaintainable data integration code.

### ETL Processing Categories

One of the key benefits of a data warehouse is its ability to compare or trend an organization’s performance over time. Two important questions for every data warehouse are:

* How much time is stored in the data warehouse?
* When is the data warehouse populated?

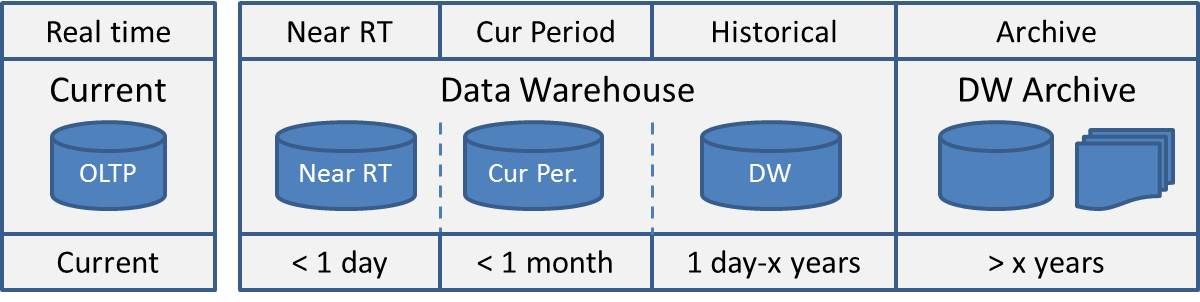
The question of how much time to store is a function of:

* **Historical reporting and the needs of business consumers.** Many industries have similar needs around the amount of historical data within their data warehouse. For example, retail organizations often report on a three-year time period, and many health care organizations keep data for seven years to conform to health care regulations.
* **Data warehouse maturity and size.** Mature data warehouses typically archive historical data older than a specified period of time because the older, less frequently referenced data degrades query and load performance.

ETL processes have traditionally loaded data warehouses on a nightly basis during non-working hours (e.g., 11pm – 7am). Note that the more global the data warehouse the less down time exists because every period of the day is working hours for some geography.

However, business consumers’ requests for real-time data are placing additional demands on the traditional ETL processing methods. In addition, many organizations are using the concept of a “current period,” where information within the current period will change frequently before being frozen at the end of the period.

Figure 3-11 organizes the data warehouse by time.



**Figure 3-11**: Data warehouse organized by time

Data warehouses organized by time have the following categories:

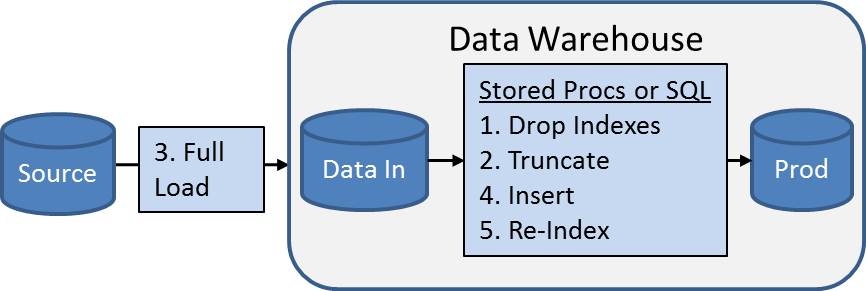
* **Archive** – This includes historical versioned data that is referenced infrequently. Archived information can either be stored in a separate database or loaded on demand into a database from flat files. Note that archived data falls within the data warehouse umbrella.
* **Historical** – The majority of data warehouse tables are loaded on a scheduled basis, usually nightly but it could be weekly or monthly for very large data warehouses.
* **Current Period** – Current period areas are frequently seen in industries where the current set of data is in progress and changes frequently before being frozen at the end of a time period, such as at the end of the month. Data is truncated and fully reloaded for the current time period.
* **Near real time** – Business consumers are frequently dissatisfied with the reporting capabilities of LOB systems. In addition, expensive queries can lock out transaction users. In these cases, your data warehouse may need to serve up real-time or near real-time data.
* **Real time** – The LOB system is the source for real-time data.

The key point is that different ETL load patterns are used for each of these time categories:

* The archive database/flat files are populated using ETL data flow and bulk inserts. The archived data is then deleted from the data warehouse. Note that this delete is an expensive operation on very large tables, so you need to allocate sufficient time for this operation.
* The historical data warehouse is populated using incremental load patterns, which are covered in the next section.
* The current period area is populated using the full load pattern, also covered in the next section.
* Real-time and near real-time data requires a more active ETL process, such as a historical process that runs every five minutes or a “push process,” which we look at in the change-detection section later in this chapter.

### Incremental Loads

Many first-generation data warehouses or data marts are implemented as “full loads,” meaning they’re rebuilt every time they’re populated. Figure 3-12 illustrates the different steps within a full load.



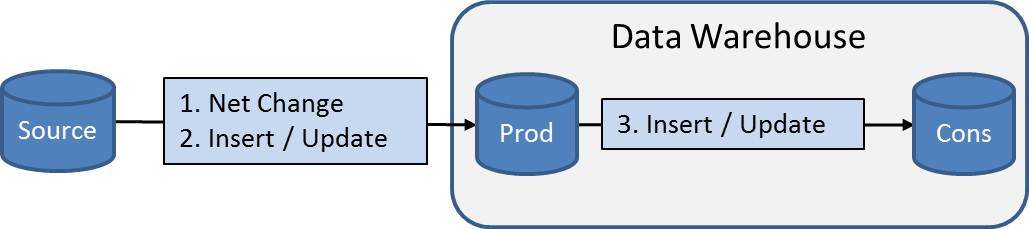
**Figure 3-12**: Full load process

The steps in a full-load process are:

1. **Drop indexes** – Indexes increase load performance times
2. **Truncate tables** – Delete all records from existing tables
3. **Bulk copy** – Load data from the source system into an “Extract In” area
4. **Load data** – Use stored procedures and SQL INSERT statements to load the data warehouse
5. **Post process** – Re-apply indexes to the newly loaded tables

However, full loads are problematic because the time to reload will eventually exceed the window of time allocated for the load process. More important, business consumers don’t have access to historical point-in-time reporting because only the most recent copy of the source system data is available in the data warehouse.

With full loads often unable to support point-in-time historical reporting, many organizations have turned to a second-generation approach that uses an “incremental load,” which Figures 3-13 and 3-14 show.



**Figure 3-13**: Incremental load with an Extract In area



**Figure 3-14**: Incremental load without an Extract In area

The steps involved in an incremental load are:

1. Load net changes from the previous load process
2. Insert/Update net changes into the Production area
3. Insert/Update the Consumption area from the Production area

The primary differences between full loads and incremental loads are that incremental loads:

* Do not require additional processing to drop, truncate, and re-index
* Do require net change logic
* Do require Updates in addition to Inserts

These factors combine to make incremental loads more efficient as well as more complex to implement and maintain. Let’s look at the patterns used to support incremental loads.

### Detecting Net Changes

The incremental ETL process must be able to detect records that have changed within the source. This can be done using either a pull technique or push technique.

* With the pull technique, the ETL process selects changed records from the source:
  + Ideally the source system has a “last changed” column that can be used to select changed records.
  + If no last changed column exists, all source records must be compared with the destination.
* With the push technique, the source detects changes and pushes them to a destination, which in turn is queried by the ETL process.

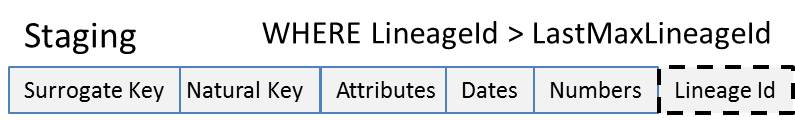
**Pulling Net Changes – Last Changed Column**

Many source system tables contain columns recording when a record was created and when it was last modified. Other sources have an integer value that is incremented every time a record is changed. Both of these techniques allow the ETL process to efficiently select the changed records by comparing the maximum value of the column encountered during the last execution instance of the ETL process. Figure 3-15 shows the example where change dates exist in the source table.



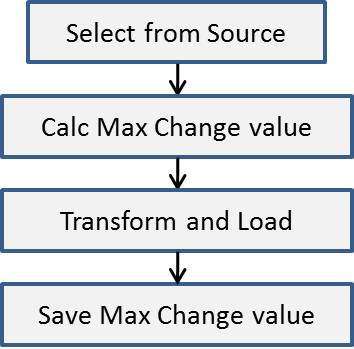
**Figure 3-15**: Detecting net changes using change date

Figure 3-16 shows how an integer value can be used to select changed records. Note that this example shows one benefit of adding an Execution lineage ID within the Production area.

****

**Figure 3-16**: Detecting net changes using an integer value

It’s the responsibility of the ETL process to calculate and store the maximum net change value for each instance for which the process is invoked, as Figure 3-17 shows.

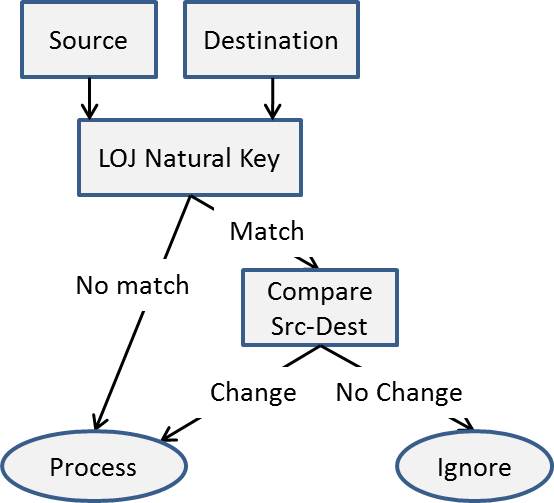


**Figure 3-17**: Calculating and saving the change value

The ETL process is then responsible for retrieving this saved max value and dynamically applying it to the source SQL statement.

**Pulling Net Changes – No Last Changed Column**

The lack of a last changed column requires the ETL process to compare all source records against the destination. Figure 3-18 shows the process flow when no last changed column exists.

****

**Figure 3-18**: Detecting net changes with no last changed column

This process flow is as follows:

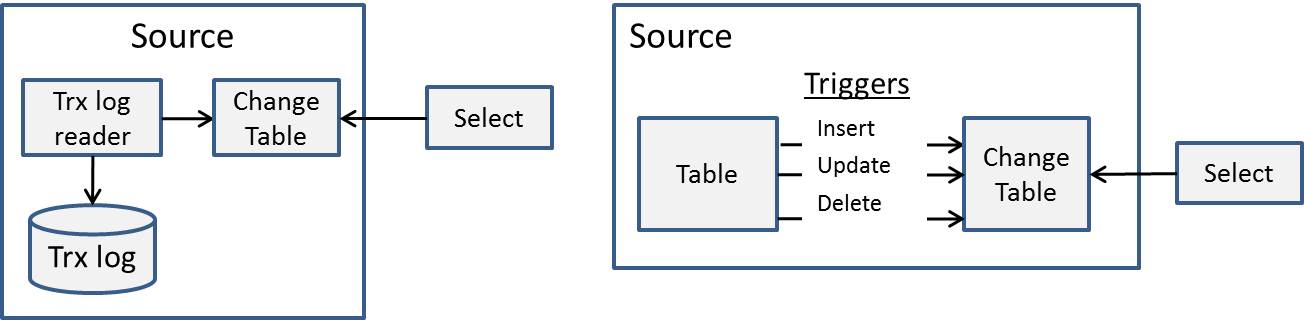
1. Join the source with the destination using a Left Outer Join. Note that this should be implemented within the ETL product when the source and destination are on different systems.
2. Process all source records that don’t exist in the destination.
3. Compare source and destination attributes when the record does exist in the destination.
4. Process all source records that have changed.

Because all records are processed, pulling net changes when there’s no last changed column

is a less efficient approach, which is especially important for very large transaction tables.

**Pushing Net Changes**

When pushing net changes, the source system is responsible for pushing the changes to a table, which then becomes the source for the ETL process. Figure 3-19 shows two common push methods.



**Figure 3-19**: Push net change options

What’s different about these two options?

* In the first scenario, the source system relational database actively monitors the transaction log to detect and then insert all changes into a destination change table.
* In the second scenario, developers create triggers that insert changes into the destination change table every time a record changes.

Note that in both cases, additional load is placed on the source system, which impacts OLTP performance. However, the transaction log reader scenario is usually much more efficient than the trigger scenario.

**Net Change Detection Guidance**

The first rule of ETL processing is that LOB source systems should not incur additional overhead during peak usage hours due to ETL processing requirements. Because ETL processing typically occurs during non-peak hours, the preferred option is a pull, not push, mechanism for net change detection.

That said, the scenarios where you might consider a push option are:

* If this process is already occurring within the source system (e.g., audit tables exist and are populated when a net change occurs in the LOB system). Note that the ETL logic will still need to filter records from the change table, so it will need to use a last changed column.
* If there is no last changed column and the logic to detect net changes is complex. One example would be a person table where a large number of columns flow to the data warehouse. The ETL logic required to compare many columns for changes would be complex, especially if the columns contain NULL values.
* If there is no last changed column and the source table is very large. The ETL logic required to detect net changes in very large transaction tables without a net change column can result in significant system resource usage, which could force the ETL processing to exceed its batch window.

### Data Integration Management Concepts

Reducing ongoing TCO for data integration operations is top priority for organizations. To reduce costs, you need to understand what contributes to the ETL TCO, as follows:

* ETL package installation and configuration from development through production
* ETL package modifications in response to hardware and software issues
* ETL package modifications for different processing options
* Tracking down system errors when they occur (e.g., when a server is down and or a disk is offline)
* Detecting programming issues

ETL developers can help ETL operations reduce ongoing TCO by building dynamic configuration and logging capabilities within their ETL packages.

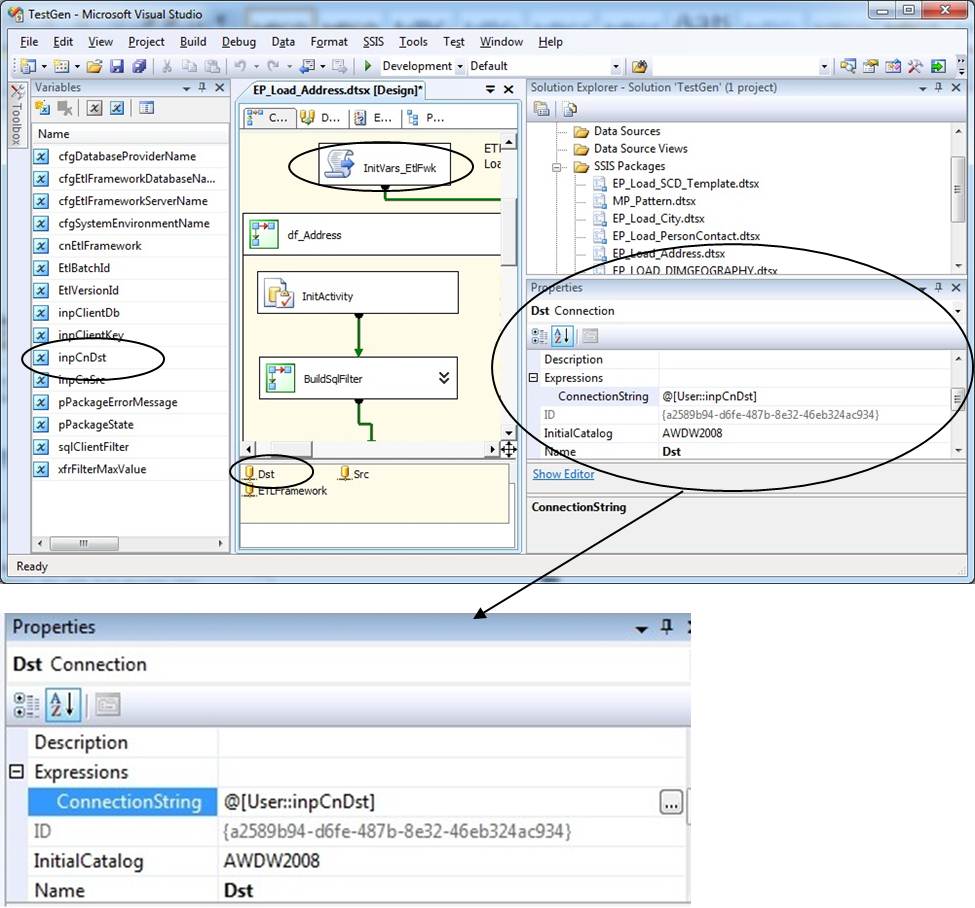
Two primary areas that developers can focus on in this area are supporting dynamic configurations and providing robust logging.

**Dynamic Configurations**

Dynamic configurations support the run-time configuration of SSIS connections and variables used within SSIS package workflows. ETL developers use the following SSIS capabilities to develop packages that support dynamic configurations:

* SSIS expressions provide a rich expression language that can be used to set almost any property or value within a package.
* Variables can either be set statically or dynamically by values or the results from an expression.
* Package configurations can be used to dynamically set variables and task properties.

Figure 3-20 shows how expressions and variables combine to dynamically configure the destination database connection.



**Figure 3-20:** Dynamic configurations: Destination connection

What’s going in Figure 3-20? Here’s the process:

* Dst is a SQL Server database connection for the destination database.
* An expression is used to set the destination ConnectionString property with the inpCnDst variable.
* The Initvars\_EtlFwk Script task then populates the inpCnDst SSIS variable with a value stored within the ETL framework’s configuration table.

Dynamic configurations are also commonly used for configuring SQL statements—for example, to add filters for incremental loads.

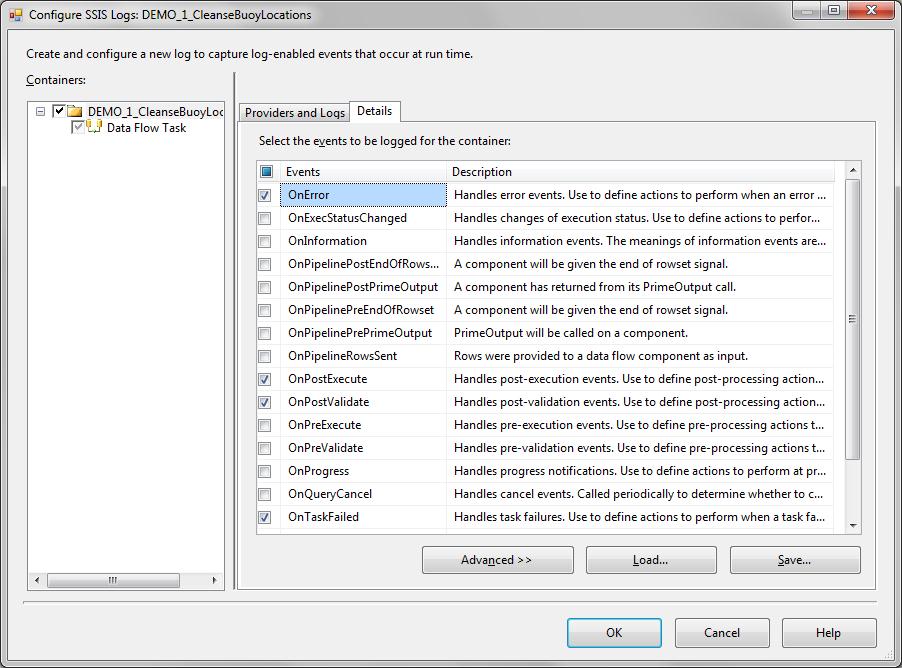
Note that this is only one example of a dynamic configuration. We’ll cover this implementation of dynamic configurations in more detail later in the ETL Framework section.

Package configurations are also a powerful tool for initializing properties and variables. The following resources provide more information about package configurations:

* [SQL Server Integration Services SSIS Package Configuration](http://www.mssqltips.com/tip.asp?tip=1405)
* [SSIS Parent package configurations. Yay or nay?](http://consultingblogs.emc.com/jamiethomson/archive/2008/08/29/ssis-parent-package-configurations-yay-or-nay.aspx)
* [SSIS Nugget: Setting expressions](http://consultingblogs.emc.com/jamiethomson/archive/2006/03/11/SSIS-Nugget_3A00_-Setting-expressions.aspx)
* [SSIS - Configurations, Expressions and Constraints](http://ewisdahl.spaces.live.com/blog/cns!23AC9944C8FA112A!419.entry?sa=708573673)
* [Creating packages in code - Package Configurations](http://www.sqlis.com/post/Creating-packages-in-code-Package-Configurations.aspx)
* [Microsoft SQL Server 2008 Integration Services Unleashed (Kirk Haselden)](http://www.amazon.com/Microsoft-Server-Integration-Services-Unleashed/dp/0672330326#_), Chapter 24 – Configuring and Deploying Solutions
* [BIDs Helper – This is a very useful add-in that includes an Expression highlighter](http://bidshelper.codeplex.com/wikipage?title=Expression%20and%20Configuration%20Highlighter&referringTitle=Home)

**Integration Services Logging**

Basic ETL execution auditing can be performed through the built-in logging feature in SSIS, which captures task events, warnings, and errors to a specified logging provider such as a text file, SQL table, or the event log. Any ETL operation needs logging to track the execution details and to do error troubleshooting; the SSIS logging provider is the first step. Figure 3-21 shows the logging event details.



**Figure 3-21:** SSIS logging events

Table 3-1 shows a few of the details captured by different logging events; the details are linearly logged and associated with the appropriate package and task within the package (package and task columns not shown).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Event | Source | starttime | endtime | Message |
| OnWarning | Customer\_Import | 2009-11-04 17:17:33 | 2009-11-04 17:17:33 | Failed to load at least one of the configuration entries for the package. Check configuration entries for "XML Config; SQL Configurations; Configuration 1" and previous warnings to see descriptions of which configuration failed. |
| OnPreExecute | Customer\_Import | 2009-11-04 17:17:34 | 2009-11-04 17:17:34 |  |
| OnPreExecute | Data Flow Task | 2009-11-04 17:17:34 | 2009-11-04 17:17:34 |  |
| OnWarning | Data Flow Task | 2009-11-04 17:17:34 | 2009-11-04 17:17:34 | Warning: Could not open global shared memory to communicate with performance DLL; data flow performance counters are not available. To resolve, run this package as an administrator, or on the system's console. |
| OnWarning | Customer\_Import | 2009-11-04 17:17:34 | 2009-11-04 17:17:34 | Warning: Could not open global shared memory to communicate with performance DLL; data flow performance counters are not available. To resolve, run this package as an administrator, or on the system's console. |

**Table 3-1:** SSIS logging output

The benefits of SSIS’s built-in logging are its simplicity and ease of configuration. However, SSIS logging falls short rather quickly when dealing with data warehouse-type ETL that has any level of complexity or volume. Here are some drawbacks to that basic logging functionality:

* The data is not normalized, and therefore it is difficult to query because it requires joining to itself several times just to get start and stop times for a single task and to identify the package.
* Scalability may be a problem because if you want to capture every package that executes to a common logging table.
* The logging shows no precedence between tasks or between parent or child packages, making it difficult to capture overall batch activity.

The following links provide more information about SSIS logging:

* [SQL Server 2005 Report Packs](http://www.microsoft.com/downloads/details.aspx?familyid=D81722CE-408C-4FB6-A429-2A7ECD62F674&displaylang=en.). This page has a link to the SQL Server 2005 Integration Services Log Reports. Note: The SSIS logging table has changed from sysdtslog90 (2005) to sysssislog (2008); you will need to change the SQL in all of the reports or create a view if running SSIS 2008 or later.
* [Integration Services Error and Message Reference](http://msdn.microsoft.com/en-us/library/ms345164.aspx). This is useful for translating numeric SSIS error codes into their associated error messages.

**Custom Logging**

An alternative to SSIS logging is to implement a custom logging solution, which is often part of an overarching ETL execution framework that manages configurations, package execution coordination, and logging.

A robust, centralized logging facility allows ETL operations to quickly see the status of all ETL activity and locate and track down issues in ETL processes. ETL developers can decide to leverage SSIS logging capabilities or access their own logging facility from within ETL packages.

We’ll expand on custom logging a bit later in the ETL Framework section.

### Batch Processing and the Enterprise ETL Schedule

As we noted earlier, most data warehouses load source data nightly during non-peak processing hours. Many of these processes are organized into a sequential series of scheduled ETL processes—what we call an “ETL batch.” ETL batches often need to be scheduled in a sequenced order due to data dependencies within the various batches. In general, there are two basic ETL batch types:

* **Source centric batches** – Data is extracted from one source
* **Destination centric batches** – One destination subject area is populated

The enterprise ETL schedule is responsible for sequencing and coordinating all of the ETL batches used to load the data warehouse. Such a schedule is necessary when there are data interdependencies within batches. A scheduling software package, such as SQL Server Agent, is typically used to sequence the ETL batches.

The following are key considerations when developing ETL batches and creating the Enterprise ETL Schedule.

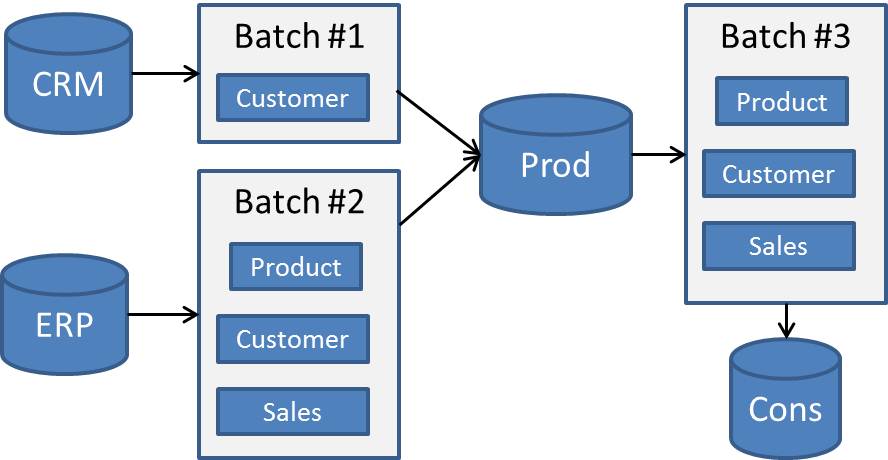
**Parallel Processing**

Running ETL processes in parallel maximizes system resources while reducing overall processing times. Because efficient ETL processes typically obtain table-level locks on destinations, the ability to run ETL processes in parallel translates into only one process loading a table at any particular point in time. This is a major consideration for ETL operations resources when creating the enterprise ETL schedule.

Note that SQL Server table partitioning can be used in support of the parallel loading of large tables. However there is still an issue with readers having access to tables or partitions that are in the process of being loaded. For more information on best practices for high volume loads into partitioned tables, see the [We Loaded 1TB in 30 Minutes with SSIS, and So Can You](http://msdn.microsoft.com/en-us/library/dd537533(SQL.100).aspx) article.

**Terminating and Restarting Batches**

One consideration when running ETL processes together in one batch is the ability to restart and resume execution at the point where the ETL process failed. On the surface, this seems like a simple concept. However, consider the fairly simple scenario presented in Figure 3-22.



**Figure 3-22:** Simple batch processing scenario

Let’s say the following batch processing occurs nightly, starting at 11pm:

1. Loading the customer record in Production area from the Customer Relationship Management (CRM) system. This is the definitive source for customer information and is where customer records are created from.
2. Loading product, customer, and sales information from the Enterprise Resource Planning (ERP) system into the Production area.
3. Loading the Consumption area product, customer, and sales tables from the Production area.

Batch #2 has a dependency on Batch #1 (Customer). Batch #3 has dependencies on batches #1 and #2. These dependencies lead to the following questions:

* What happens when Batch #1 fails with an error? Should batches #2 and #3 execute?
* What happens when Batch #2 fails? Should Batch #3 execute?
* What happens when Batch #3 fails? Should it be rerun when the problem is located and diagnosed?

The answer to all these is “it depends.” For example, batches #2 and #3 can run when Batch #1 fails if and only if:

* Incremental loads are supported in Batch #1
* Late-arriving customers are supported in the Batch #2 customer and sales ETL processing

Two primary approaches to rerunning ETL processes are:

* The ability to rerun an entire ETL batch process and still produce valid results in the destination. This supports the simplest workflow (i.e., exit batch processing when the first severe error is encountered). You rerun the entire batch once the problem is fixed.
* The ability to checkpoint an ETL batch at the point of failure. When rerun, the ETL batch then resumes processing where it left off. Note that SSIS supports the concept of a checkpoint.

In either case, one feature that does need to be supported is the ability to back out results from a particular instance or instances of a batch execution.

**Backing Out Batches**

Conceptually, an ETL batch can be viewed as one long-running transaction. ETL processes almost never encapsulate activity within a transaction due to the excessive overhead from logging potentially millions of record changes.

Because ETL processing doesn’t run within a transaction, what happens if a process within an ETL batch is faulty? What happens when source data is determined to be in error for a particular instance of an ETL load?

One answer is the ability to “back out” all the activity for one ETL batch or subsets of the ETL batch. The ETL Framework section later in this chapter presents a pattern for backing out results from a particular batch execution instance.

## ETL Patterns

Now that we have our changed source records, we need to apply these changes to the destination. This section covers the following patterns for applying such changes; we will expand on these patterns to present best practices later in the chapter:

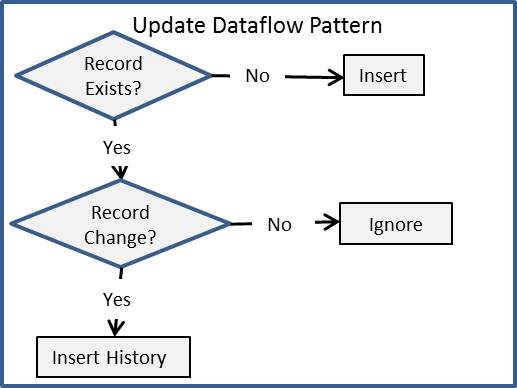
* Destination load patterns
* Versioned insert pattern
* Update pattern
* Versioned insert: net changes

### Destination Load Patterns

Determining how to add changes to the destination is a function of two factors:

* Does the record already exist in the destination?
* Is the pattern for the destination table an update or versioned insert?

Figure 3-23’s flow chart shows how the destination table type impacts how the source record is processed. Note that we will cover deletes separately in a moment.

****

**Figure 3-23**: Destination load pattern flow chart

### Versioned Insert Pattern

A versioned insert translates into multiple versions for one instance of an entity. The Kimball Type II Slowly Changing Dimension is an example of a versioned insert.

A versioned insert pattern requires additional columns that represent the state of the record instance, as shown in Figure 3-24.



**Figure 3-24**: Versioned insert supporting columns

The key versioned insert support columns are as follows:

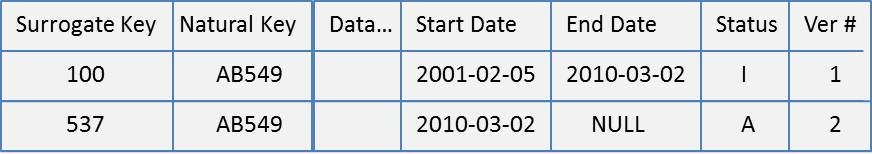
* **Start Date** – The point in time when the record instance becomes active
* **End Date** – The point in time when the record instance becomes inactive
* **Record Status** – The record status; at minimum, this is set to Active or Inactive
* **Version #** – This is an optional column that records the version of this record instance

Figure 3-25 shows an example of the first record for a versioned insert.



**Figure 3-25**: Versioned insert: First record

Let’s assume that this record changes in the source system on March 2, 2010. The ETL load process will detect this and insert a second instance of this record, as shown in Figure 3-26.



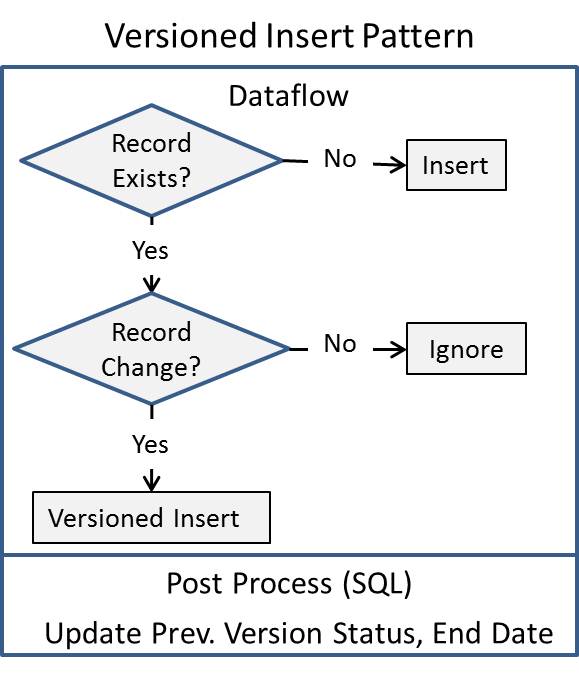
**Figure 3-26**: Versioned insert: Second record

When the second record is inserted into the table, notice that the prior record has been updated to reflect the following:

* **End date** – The record is no longer active as of this point in time
* **Record status** – Changed from Active to Inactive

The ETL process implementing this versioned insert pattern should implement the following logic for optimal performance, as Figure 3-27 shows:

* Use a bulk-insert technique to insert the new version
* Use one set-based Update to set the previous record instance’s End Date and Record Status values



**Figure 3-27**: Versioned insert pattern: Combine data flow with set-based SQL

The best practices section later in this chapter will show an SSIS example that implements this pattern.

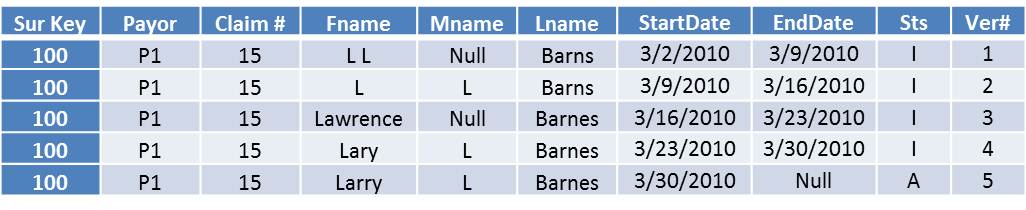
**Why Not Always Use Versioned Inserts?**

Some data warehouse implementations predominately use a versioned insert pattern and never use an update pattern. The benefit of this strategy is that all historical changes are tracked and recorded. However, one cost is that frequently changing records can result in an explosion of record versions.

Consider the following scenario: Company A’s business is analyzing patterns in health care claims and providing analytics around scoring and categorizing patient activity, which can be used by payors to set health insurance premiums. Company A:

* Receives extracts from payors in a Comma Separated Value (CSV) format
* Loads these extracts into its data warehouse

One extract is the insured party extract, which the payor provides each week. Insured party information is entered manually by the payor’s customer service representatives. Figure 3-28 shows the change activity that can occur in this scenario.



**Figure 3-28:** Heavy change activity in a versioned insert pattern

The data warehouse team should consider the update pattern for instances where changes don’t impact historical reporting. In the above example, that would reduce the number of records from five to one. In situations when human data entry can result in many small changes, the update pattern, which we cover next, could result in a table that is factors smaller than one using the versioned insert pattern.

### Update Pattern

An update pattern updates an existing record with changes from the source system. The benefit to this approach is that there’s only one record, as opposed to multiple versioned records. This makes the queries more efficient. Figure 3-29 shows the supporting columns for the update pattern.

****

**Figure 3-29:** Update pattern support columns

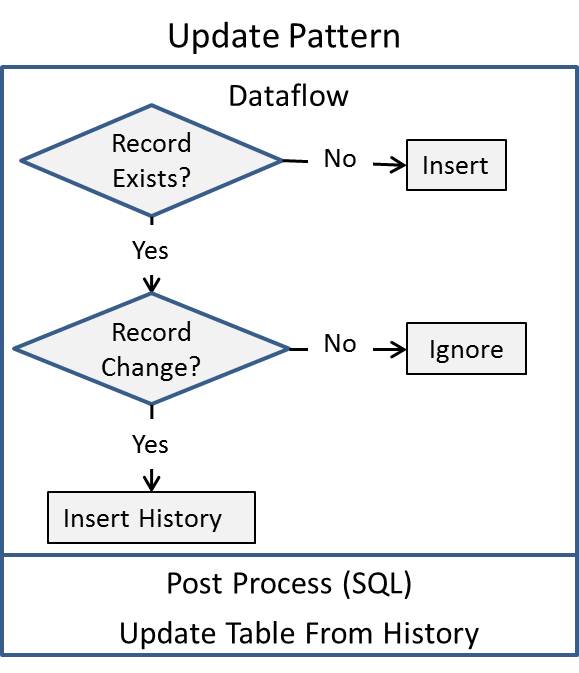
The key update support columns are:

* **Record Status** – The record status; at minimum, this is set to Active or Inactive
* **Version #** – This is an optional column that records the version of this record instance

The primary issues with the update pattern are:

* **History is not recorded.** Change histories are valuable tools for data stewards and are also useful when data audits occur.
* **Updates are a set-based pattern.** Applying updates one record at a time within the ETL tool is very inefficient.

One approach that addresses the above issues is to add a versioned insert table to the update pattern, as Figure 3-30 shows.

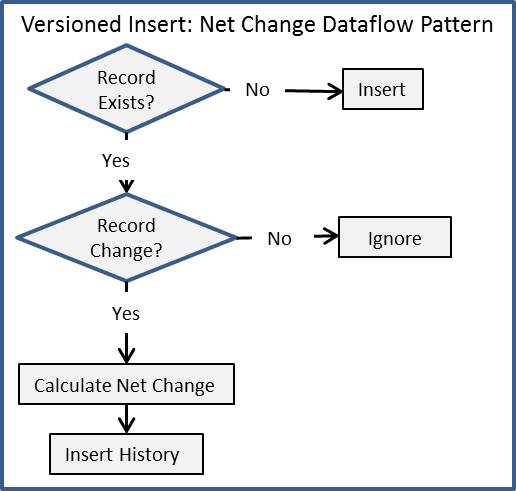


**Figure 3-30:** Update pattern: Adding a history table

Adding a history table that records all source system changes supports data stewardship and auditing and allows for an efficient set-based update of the data warehouse table.

### Versioned Insert: Net Changes

This versioned insert: net changes pattern is often used in very large fact tables where updates would be expensive. Figure 3-31 shows the logic used for this pattern.



**Figure 3-31:** Versioned insert: net change pattern

Note that with this pattern:

* Every numeric and monetary value is calculated and stored as a net change from the previous instance of the fact table record.
* There is no post-processing activity (i.e., updates to the fact table after the data flow completes). The goal is to avoid updates on a very large table.
* The lack of updates combined with the size of the underlying fact table makes the record change-detection logic more complex. The complexity comes from the need to efficiently compare the incoming fact table records with the existing fact table.

The best practices section later in this chapter will show examples of all these patterns.

## Data Quality

Data quality is of primary importance to every data warehouse. The lack of data quality directly leads to business distrust of data warehouse results, which in turn can result in extensive ongoing maintenance costs to track down each reported data issue. Worse yet, business users might stop using the data warehouse entirely.

Some common reasons why data quality issues exist include:

* Data quality issues exist in the source system—for example, LOB or core enterprise systems have some level of incomplete data, duplicates, or typographical errors.
* Connecting and correlating data across multiple systems involves data that often does not associate easily.
* External data that is part of the data warehouse rarely matches the codes or descriptions used by internal systems.
* Combining data across similar internal systems (e.g., regional, lines of business, or sub-entity systems) introduces duplicates, differing data types, and uncoordinated system keys.

The challenge for the data warehouse team is balancing the time and cost involved in trying to cleanse or connect incomplete or erroneous data. The first step is to promote data quality to a first-class citizen by doing the following:

* Implementing robust data quality checks within all ETL processes
* Correcting data quality issues where possible
* Logging data quality issues as exceptions when they can’t be corrected
* Building reporting tools for data stewards so they can assist in the detection and correction of data exceptions at the source, where they originated.

However, that’s not enough. Even correct results are often questioned by business consumers, who rarely having complete insight into how the data has been consolidated and cleansed. Providing complete transparency for all ETL processing is required so that data stewards can track a data warehouse result all the way back to the source data from which it was derived.

This section covers data quality through the following topics:

* A data quality scenario
* Data errors and exceptions
* Data stewardship and exception reporting
* Data profiling
* Data cleansing
* Data reconciliation and lineage

Let’s start by looking at a data quality scenario. We’ll then walk through the other concepts, including data profiling and data cleansing coverage, which provides guidance on how to leverage SQL Server technologies to identify and achieve a solution to data quality issues.

### Data Quality Scenario

A common data quality situation occurs when tracking customer sales across channels. Most retailers sell their product through different sales channels, such as through a Web merchant, direct marketing, or a store front. Customer identification many times isn’t recorded within the sales transaction. But without customer identification, there is no easy way to identify and track one customer’s transaction history across all sales channels.

Tracking a customer’s transaction history involves a few steps:

1. Identifying a unique customer across person-centric LOB systems (Call Center, Sales, Marketing, etc.)
2. Mapping the consolidated customers across the various sales channels
3. Mapping the final customers to transactions

The first two steps involve matching customers to identify their residence, which is often called address geocoding or postal identification. This involves matching an address received with a master list of addresses for area and being able to identify the physical geographical location. Whether the location is used or not, it produces the ability to identify matches across data sets. Of course, people move or share residences, so another step is to try and match to individuals based on name after an address is identified.

In Table 3-2, you can see that a single customer exist multiple times in different systems.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Source | Name | Address | City | Postal Code |
| Call Center | John Frame | 18 4-Oaks Dr | Monroe, LA | 71200 |
|  |  |  |  |  |
|  | J.S. Frame | 18 Four Oaks Dr | Unknown | 71200 |
| Source |  |  |  |  |
| Customer portal | Johnathan Frame | 18 Four-Oaks drive | Monroe, LA | 00000 |
| Website |  |  |  |  |
|  |  |  |  |  |

**Table 3-2:** Multiple occurrences of one customer

The master record in Table 3-3 is the cleansed record that the above source records need to match. Running the records above through an address-cleansing utility will identify that these customers are all one in the same. Several applications can handle this task and use either SQL Server as a source or integrate with SSIS.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Master Address ID | Name | Address | City | Postal Code |
| L456BDL | Jonathan S Frame | 18 Four Oaks Dr | Monroe, LA | 71200-4578 |
|  |  |  |  |  |

**Table 3-3:** One occurrence of a customer

Once the addresses and names are matched, the source transactions can be merged together with common customer association.

Examples of where bad data can enter a data warehouse system are numerous:

* Think about all those sales force automation applications where sales representatives are responsible for entering information, including prospect and customer names and contact information.
* Or consider a mortgage-origination system whose results are used to incent and compensate mortgage originators on completing a mortgage. A lack of checks and balances within the system could allow incorrect information (e.g., base salary and assets) to be entered.

Many first-generation data warehouse ETL processes in production today were developed without the checks required to detect, flag, and log data exceptions. In these cases, bad data can make its way into the data warehouse, which ultimately will result in business consumer distrust.

Once the business starts questioning the results, the burden is on the data warehouse delivery team to prove that the results are correct or find where the data is in error.

### Data Exceptions

There many types of data exceptions and this section will cover some of the more common types. As you deal with exceptions, you need to decide how to handle each situation—whether you discard the entire related record, cleanse the data, replace the bad data with an unknown, or flag the record for manual review.

**Missing or Unknown Values**

Missing values are the most common data exception because many systems do not put constraints on every data entry field. When you receive missing values in a data warehouse system, you need to decide whether to replace the value with an “Unknown” identifier or leave the value as blank.

The importance of the column also affects how you handle the data. If you are missing a required field to identify the product purchased, for example, you may need to flag the record as an exception for manual review. If the value is just an attribute where unknowns are expected, you can use a placeholder value.

Unknown values go beyond values that are missing. An unknown can be when you receive a code that does not match your master code list for the given attribute. Or let’s say you are doing a lookup from the source record to associate with other data, but there isn’t a match—this is also a type of unknown. In this case, there’s an additional option: Create a placeholder record in the lookup table or dimension. Kimball refers to this type of data exception as a “late arriving dimension.”

**Date-Type Conversion and Out-of-Range Errors**

Another common data problem, especially when dealing with text files, involves conversion problems. This problem manifests itself through truncation errors, range errors with numeric constraint issues, or general conversion issues (such as converting text to numeric).

Table 3-4 contains some examples of these kinds of data quality issues. For each situation, you need to decide whether to try to fix the issue, NULL-out the result, or flag the problem for manual review.

|  |  |  |  |
| --- | --- | --- | --- |
| Source Value | Normalized Data Type | Issue | Resolution |
| 00/00/2010 | Date | Date does not exist | NULL value or convert to 01/01/2010 |
| 1O | 2-byte integer | Typo - O used for 0 | NULL value or convert to 10 |
| A¥µBC | Non-Unicode text | Characters do not map to 1 byte | Remove value, flag for review of destination data type |
| “Marriott” | 5-character text | Truncation issue | Truncate value at 5 characters or flag for review of data type |
| 1000 | 1-byte integer | 256 is the max value for a 1-byte Integer | Discard value, convert to 100, flag for review |

**Table 3-4:** Data exception examples

Thousands of variations of data conversion and range issues can occur. The appropriate solution may involve simply ignoring the value if the attribute or measure is not core to the analysis, or going to the opposite side of the spectrum and sending the record to a temporary location until it can be reviewed.

**Names and Addresses**

The data quality scenario above illustrated the type of data exceptions that can occur with addresses and names. Company names are also a common challenge when correlating systems that contain vendors or business-to-business (B2B) customers.

Take this list of organizations:

* Home Design, INC
* Home Design, LLC
* Home design of NC
* Home Designs

They could all be the same company, or they could be separate entities, especially if they are located in different states. If you have an address, you can use it to try to match different representations of the company name to the same company, but larger organizations often have multiple locations. However, a combination of name, region, and address will often do a decent job of matching. You can also purchase corporate data that can help associate businesses and other entities.

**Uniformity and Duplication**

Data entry systems often have free-text data entry or allow a system user to add a new lookup if he or she doesn’t find the value, even if it already exists. Here are some considerations for data uniformity:

* When dealing with data in textual descriptions, it is often best to leave the raw data in an intermediate data warehouse database table for drill-down or drill-through use rather than trying to parse out the values.
* When dealing with duplication of codes and values that are more discrete, a lookup mapping table can be maintained to help in the process. Consider managing the mappings through a data validation interface to prevent operations or development from being directly involved.

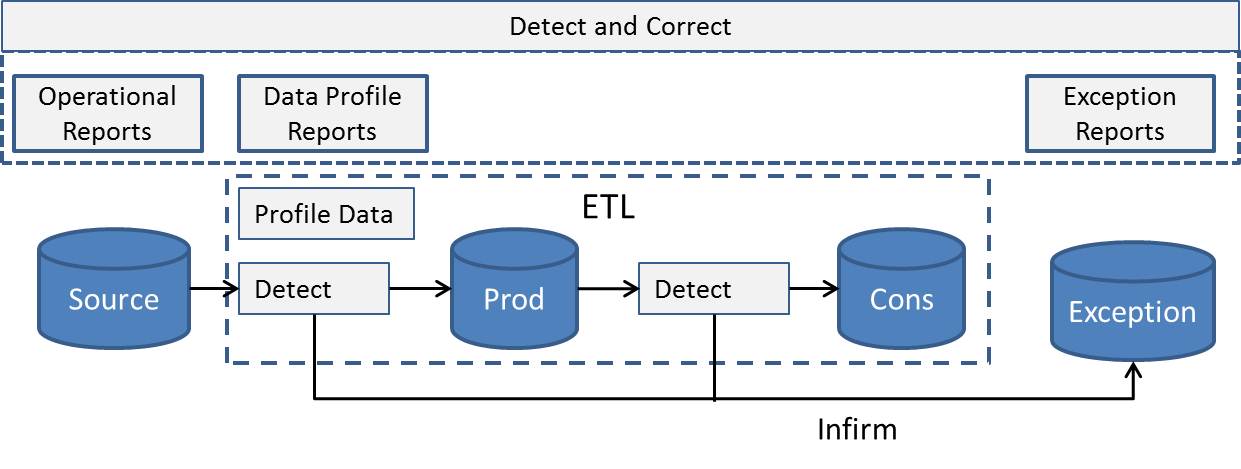
Two general types of duplication exist:

* Duplication of exact values or records that match across all columns
* Duplication when values are not exact but similar

The first step in removing duplication is identifying where duplications can exist and the type or range of variations that indicate duplication.

### Data Stewardship and Validation

Before moving on to implementation patterns, let’s step back and consider data quality from an overall perspective. Figure 3-32 illustrates the life cycle of data quality across the data warehouse data integration process. (Note that this figure doesn’t include an “Extract In” database, which is often populated from the source and in turn populates the Production data area.)



**Figure 3-32**: Data quality checkpoints in the data integration life cycle

Data warehouse efforts need a data steward who is responsible for owning everything from the initial data profiling (identifying data issues during development) to the data exception planning process and exception reporting.

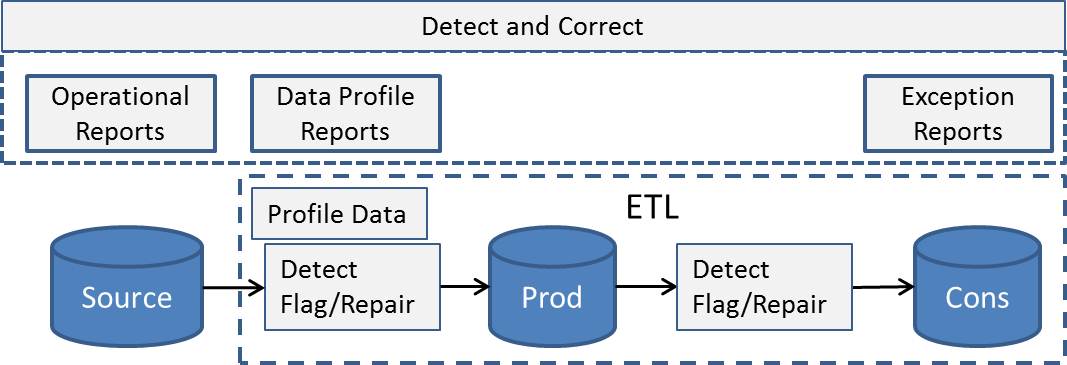
**Data Quality Checkpoints**

The earlier data exceptions are detected, the easier it is to repair the error. There are several steps in the data integration process where data exceptions can be identified:

* **Operational reports** – Detecting data quality errors from operational reports allows the data steward to repair the data error.
* **Data profile reports** – Data profiling allows the data steward to detect certain data quality errors, including out-of-range values. Identifying data exceptions at this stage enables the ETL processes to check for and log these exceptions.
* **Exception reports** – Exception reports show that the ETL process detected and logged an exception in the Source- (or Extract In-) to-Stage ETL process. The data steward can use the exception report to identify and repair the error at the source.

**Repairing/Flagging Data Exceptions**

The ETL process in Figure 3-32 shows an Exception database (Exc) where data exceptions from ETL processing are inserted. An alternative approach is to repair the data exception and/or flag the records as exceptions and allow them to flow to the data warehouse as shown in Figure 3-33.



**Figure 3-33**: Flagging exceptions

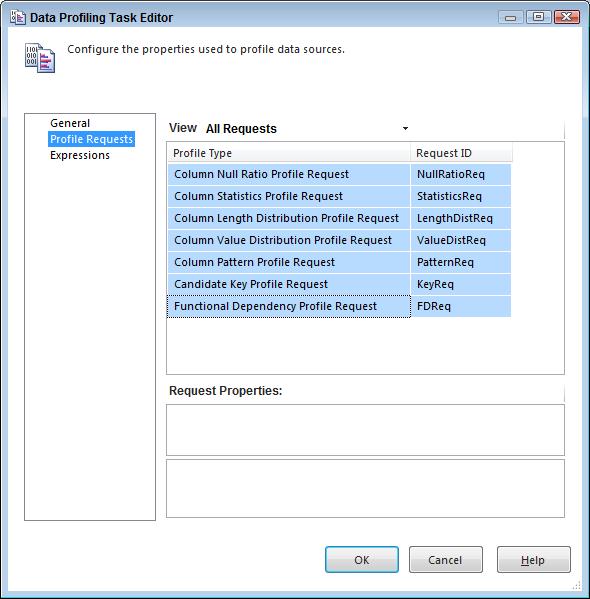
Reasons you might consider this approach include:

* The data exception is not severe and can be repaired by the ETL process. Missing values and unknown values are examples of this.
* The record has valid data along with the exceptions that need to be flagged.

### Data Profiling

One of the first tasks for the data steward is to profile the source data to document data quality issues that need to be handled by the development team. This assumes that a data map has been generated to identify which data points from the source systems and files will be used in the warehouse and how they are mapped to the destination entities.

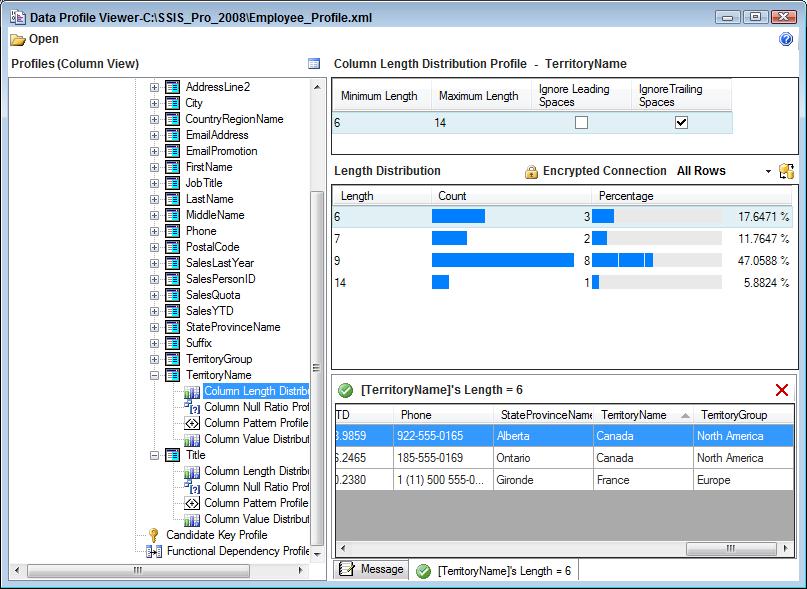
SQL Server 2008 introduced a built-in Data Profiling task in SSIS that allows an easy review of source data for this purpose. To leverage the Data Profiling task, you first have to have the data in SQL Server tables, such as in an “export in” database. Figure 3-34 shows the task editor and the types of data profiling that can be done, including NULL evaluations, value distributions and patterns, statistics on numeric columns, candidate key options, and more.



**Figure 3-34**: Data profiling in SSIS

After the task is run in SSIS, it generates an XML file that can be viewed by the Data Profile Reader, found in the SQL Server 2008 applications folder, which you can access from the Start menu.

As an example, the report in Figure 3-35 shows the column length distribution for a column. It also provides sample values and lets you include or exclude leading or trailing spaces.



**Figure 3-35**: Column length distribution

Other data profile types include pattern identification with regular expression outputs, max, min, mean value statistics, distribution of numbers, and discreteness.

You can use this collective information to help determine where and how data quality issues should be handled.

### Data Cleansing

Data cleansing is the activity of dealing with the data quality issues and data exceptions in the ETL process. Cleansing can range from simply replacing a blank or NULL value to a complicated matching process or de-duplication task.

Here are some general guidelines when implementing data cleansing:

* Multiple layers of data cleansing are often applied in a data warehouse ETL process. The first layer usually involves data parsing and handling of common data exceptions such as NULLs, unknowns, and data range problems. The second level may involve de-duplication or data correlation to a different source, data consolidation of different sources, or data aggregation and summary.
* A common mistake is to perform updates to the raw tables in the “extract in” environment. This adds a synchronous step and incurs a database performance hit because of the transaction as well as causes an I/O hit and lengthens the overall execution time. In addition, this approach invalidates the ability to go back to the extraction environment and see the raw data as it was first extracted.
* A better approach is to either apply the initial layer of data cleansing in the query that is run against the extract table or leverage the transformation capabilities in the ETL tool. Both of these strategies have much less overhead and allow more server resources and time to be applied to any complicated data cleansing.

Of course, you can also use T-SQL queries to cleanse data, employing ISNULL, NULLIF, RTRIM, LTRIM, REPLACE, etc.

When using SSIS for data cleansing, you can apply many of the transformations to different data cleansing situations. Here are some examples:

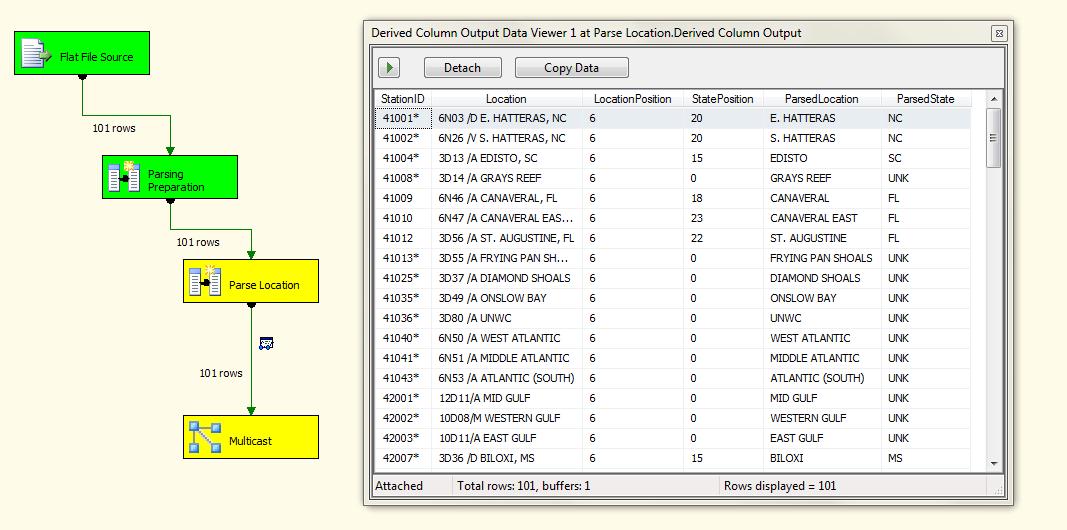
* You can use the Derived Column transformation for text parsing, NULL identification and replacement, calculations, and more.
* Lookup and Merger Join transformations can help with data correlation and association.
* Fuzzy Lookup and Fuzzy Grouping allow for complicated data associations and de-duplication.
* Pivot and Un-Pivot transformations let you change the granularity and normalization pattern of data.

As a simple example, Table 3-5 contains data from a text column that identifies rough locations of equipment. The challenge is trying to parse out the data.

|  |
| --- |
| Values |
| 6N03 /D E. HATTERAS, NC |
| 6N46 /A CANAVERAL, FL |
| 10D08/M WESTERN GULF |
| 3D35 /D LANEILLE, TX |
| 10D09/A WESTERN CARIBBEAN |

**Table 3-5:** Sample data – equipment locations

However, you can use two SSIS Derived Column transformations to parse and cleanse the data. The first transformation does some preparation textual calculations, and the second performs the primary parsing. Figure 3-36 shows the data flow and the Data Viewer with the resulting parsed data.



**Figure 3-36**: Text cleansing in SSIS

The Derived Column transformation uses the SSIS Expression Language. Tables 3-6 and 3-7 show the expressions used in each transformation. The reason two transformations are used is that the columns added in the first transformation are referenced in the second, making the expressions easier to follow and manage.

|  |  |  |
| --- | --- | --- |
| Column | Purpose | Expression |
| Location | Replace NULLs | ISNULL(Location) ? "Unknown" : TRIM(Location) |
| LocationPosition | Starting position for Location | FINDSTRING(Location,"/",1) |
| StatePosition | Starting position for State | FINDSTRING(Location,",",1) |

**Table 3-6:** Derived Column 1 expressions

|  |  |  |
| --- | --- | --- |
| Column | Purpose | Expression |
| ParsedLocation | Parse Location | SUBSTRING(Location,LocationPosition + 3,(StatePosition == 0 ? (LEN(Location) - LocationPosition + 4) :  (StatePosition - LocationPosition - 3))) |
| ParsedState | Parse State | (StatePosition == 0) ? "UNK" : SUBSTRING(Location,StatePosition + 2,LEN(Location) - StatePosition + 1) |

**Table 3-7:** Derived Column 2 expressions

The SSIS Best Practices section of this chapter provides additional approaches to and examples of de-duplication.

### Data Reconciliation

As we noted earlier, users will distrust data warehouse results when the numbers don’t add up. A common complaint is, “I get different results from different systems.” This complaint can be unfounded, but if that’s the perception, the reality often doesn’t matter. Here’s where reconciliation becomes a key component of a data steward’s responsibilities.

Reconciliation is the process of comparing results of the same metric from two different systems; the classic example is balancing a checkbook. Table 3-8 shows a sample bank statement; Table 3-9 is the checkbook register.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Account | Type | Ending Balance | |
| 31-May | 100-35 | Checking | 550.00 | |
|  |  | Count | Amount |  |
|  | Previous Balance |  | 0.00 |  |
|  | Deposits/Credits | 1 | 1000.00 |  |
|  | Checks/Debits | 4 | 450.00 |  |
| Date | Description | Number | Amount | Balance |
| 1-May | Deposit |  | 1000.00 | 1000.00 |
| 17-May | Check | 1 | 100.00 | 900.00 |
| 24-May | Check | 2 | 50.00 | 850.00 |
| 25-May | Check | 3 | 200.00 | 650.00 |
| 31-May | Check | 4 | 100.00 | 550.00 |

**Table 3-8:** Bank statement

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Description | Number | Amount | Balance |
| 1-May | Deposit |  | 1000.00 | 1000.00 |
| 15-May | Electric Utility | 1 | 100.00 | 900.00 |
| 22-May | Gas | 2 | 50.00 | 850.00 |
| 23-May | Rent | 3 | 200.00 | 650.00 |
| 30-May | Grocery Store | 4 | 100.00 | 550.00 |
| 31-May | Restaurant | 5 | 100.00 | 450.00 |

**Table 3-9:** Checkbook register

To reconcile the statement with your checkbook records, you would:

1. Compare the bank statement Ending Balance with the checkbook Balance on May 31, the date the bank statement was created.
2. If these values are the same, you’ve successfully reconciled your checking account. If these values are not the same, as in the above example, more analysis is required.
3. One next step is to compare deposit and check counts. In the above example, the bank statement shows four checks, while the checkbook ledger shows five.
4. The next step is look at the detailed check ledgers to find the discrepancy. In this example, the bank statement does not include Check #5. This is probably due to the delta between when a check is issued by the buyer to the seller and when the seller deposits the check in his or her bank account.

This simple example demonstrates the key role of reconciliation within a data warehouse and how it can grow to be an all-consuming process. It’s important to equip the data steward with tools that assist in this process. We’ll present reconciliation best practices later in this chapter.

Dates also play a significant role in any data warehouse and are a common cause for business consumers questioning data warehouse results. We’ll discuss the role of dates and time in data integration in an upcoming section.

### Lineage

Data exception and reconciliation processes both benefit when Lineage is introduced into the data integration processes. Merriam-Webster’s online dictionary defines lineage as, “A descent in a line from a common progenitor.” Lineage in data integration can be applied to:

* **Code and Data Definiton Language (DDL)**– Identifies a particular version of the ETL code or table or column definition
* **Data lineage** – Identifies the source record or records for a destination
* **Execution lineage** – Identifies the instance of the ETL process that inserted a new record or updated an existing record

**Code and DDL lineage**

Although not directly related to data quality, code and DDL change tracking does allow data integration issues and bugs to be identified.

Code lineage is used to tie an executing instance of an ETL process to a particular version of that ETL process. Capturing and logging code lineage at execution time assists error and exception analysis. The team tracking down a code error or a data exception can check to see if the code lineage had changed from the previous execution instance.

SSIS has many package-level system variables available at runtime that provide detailed information about the particular version of the package that’s executing. You can learn more about [System Variables](http://msdn.microsoft.com/en-us/library/ms141788.aspx) in SQL Server Books Online.

DDL lineage is about tracking the changes in structure to the data warehouse over time. Although often manually maintained through DDL scripts, DDL lineage assists in root cause analysis when an issue is identified. Like any system, when the physical table or key relationships need to change, the change needs to be scripted out and packaged into a deployment, including:

* Handling of data changes that are affected by the DDL change
* DDL change script encompassed within a transaction

DDL lineage is commonly handled by simply maintaining SQL scripts in a deployment folder. After the scripts are tested in a QA environment, they are usually run by the DBA at a planned downtime window and then archived for future reference. DDL changes should be documented in the project management system.

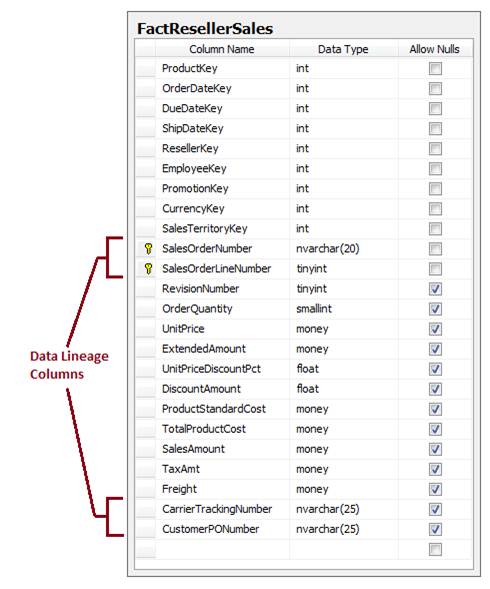
**Data Lineage**

One of the two most important types of lineage related to data warehouse systems is data lineage. Data lineage is about tracking each data row through the system from source to the data warehouse. The value of data lineage includes:

* Data validation and drill-through analysis
* Troubleshooting data questions and errors
* Auditing for compliance and business user acceptance
* Identification of records for updates or comparison

Two general approaches can be taken for data lineage:

* **Leveraging the source systems key for lineage tracking.** In some cases, using the source system keys is the easiest way to track rows throughout the integration process. Figure 3-37 highlights a couple of sets of data lineage columns in the represented fact table. The primary lineage columns are the sales order numbers, but the tracking and PO numbers also are useful for lineage.



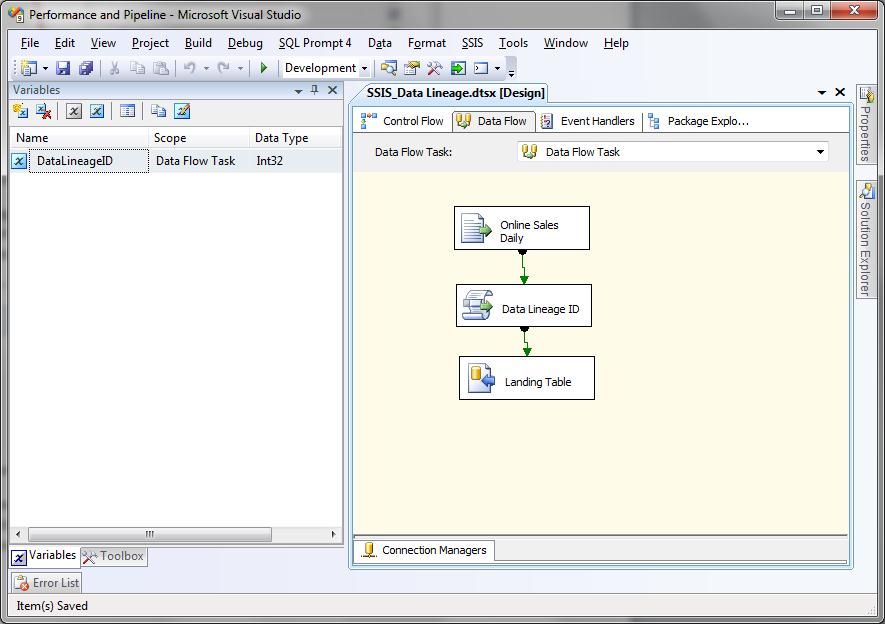
**Figure 3-37**: Data lineage example

* **Creating custom lineage identifiers in the data warehouse ETL load.** Not all sources have keys, or the keys may not identify the source record uniqueness. In this case, data lineage can be handled by generating an auto-incrementing number in the ETL, as Figure 3-38 shows.



**Figure 3-38**: Custom data lineage example

Generating a lineage is easiest to do right in the export process from the source. In SSIS, a couple lines of code will allow an auto-incrementing value to be added. Figure 3-39 shows a data flow with a Script Component that adds the lineage.



**Figure 3-39**: Data lineage in the SSIS data flow

In this example, a package variable has been created called DataLineageID of type Int32. This value is first updated in an Execute SQL task with the last lineage for the table. The data flow Script Component then simply increments an internal variable for the lineage by 1 for each row. The code in the Script Component is shown below.

Public Class ScriptMain

Inherits UserComponent

Private DataLineageID As Integer = Me.Variables.DataLineageID

Public Overrides Sub Input\_ProcessInputRow(ByVal Row As InputBuffer)

Me.DataLineageID = Me.DataLineageID + 1

Row.DataLineageID = Me.DataLineageID

End Sub

End Class

The challenges of data lineage revolve around complicated transformation logic where multiple source rows combine together into one destination row. In this case, a mapping table of surrogate keys to lineage ID can allow history and mapping to be tracked.

**Execution Lineage**

The other important lineage in data warehouses is execution lineage, which is about recording detailed information for ETL package execution. When stored as an integer value within each destination table, execution lineage allows the data steward or auditor to view the specifics of the instance of the ETL process that created or modified the record.

Execution lineage tracks:

* When a process ran
* What ETL batch it belongs to
* The precedence of steps
* The duration of steps
* Error tracking of process failures and warnings

Execution lineage is created within the context of a logging facility, which is usually a component of a larger ETL framework.

## ETL Frameworks

Successful enterprise data warehouse integration efforts typically have an ETL framework as one of their best practices. ETL frameworks at their core support dynamic configurations and centralized logging. This section will first provide an overview of ETL frameworks and then cover examples of SSIS template packages that integrate into an ETL framework.

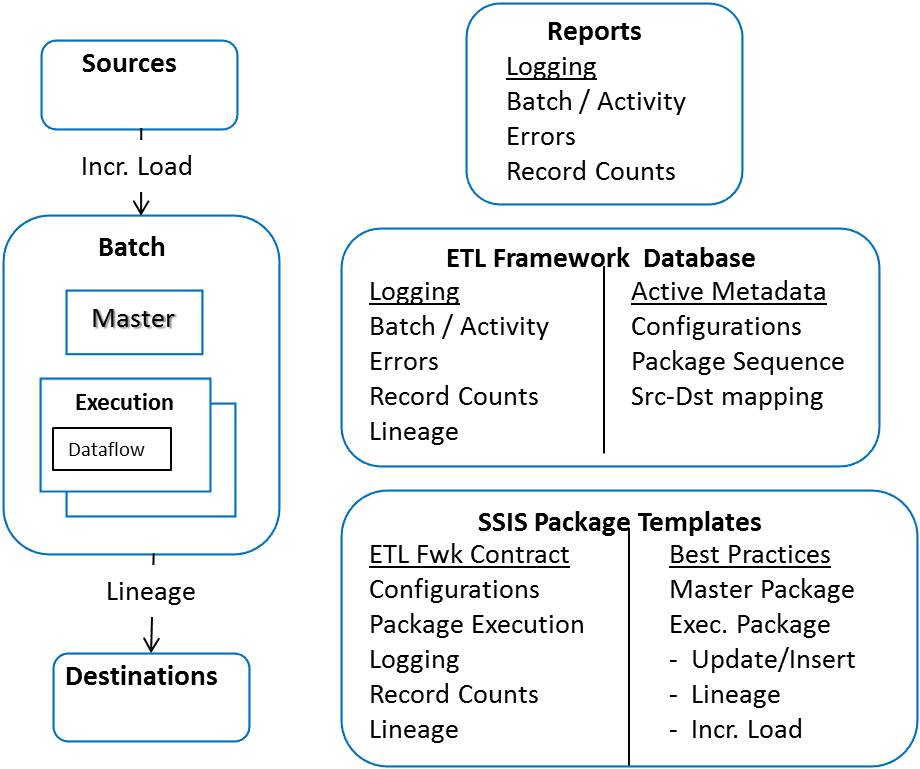
This section’s best practices, used by ETL developers doing ongoing management and monitoring of ETL processes, is targeted at architects, developers, and ETL operations staff:

* Data architects and developers are responsible for implementing the tools and frameworks used by ETL operations.
* ETL operations can be a separate team in large organizations or systems and database administrators in smaller organizations. These resources are responsible for keeping the “wheels on the bus” for all ETL processes within an organization.

Note that all the packages presented here, as well as the ETL reports shown above, can be found online at: <http://etlframework.codeplex.com/>

### ETL Framework Components

The primary components for an ETL framework are a database, template packages, and reports, as shown in Figure 3-40.



**Figure 3-40:** ETL framework components

The ETL framework database stores all metadata and logging for enterprise ETL activity:

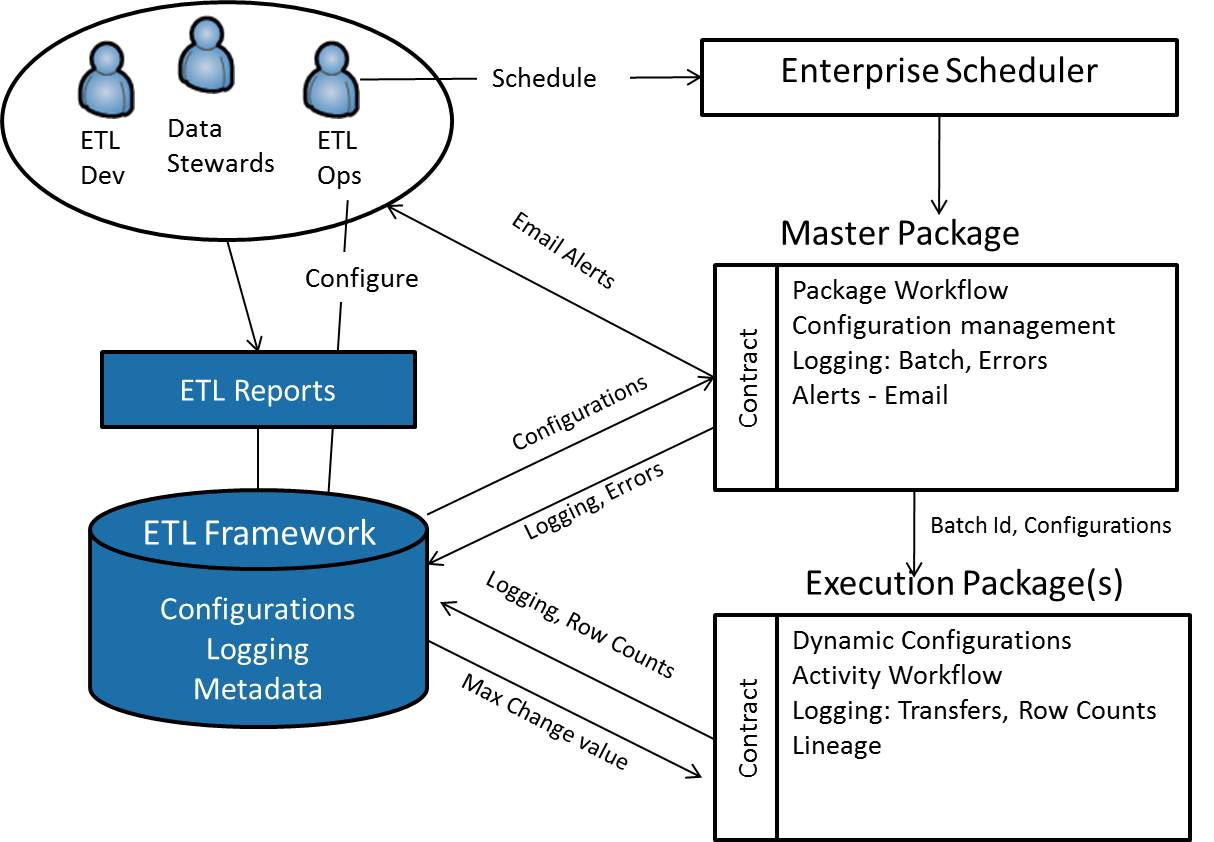
* **Active Metadata** – Consists of technical metadata tables used to drive ETL package execution. Examples include configuration tables, sequenced execution packages, and source-to-destination mappings.
* **Logging** – Contains all logging activity from master and execution packages.

Package templates are used as starting points for all ETL development. Each template interfaces with the ETL framework stored procedures; there are several package templates and components:

* **Master package** – Creates and logs batches and provides workflow for execution packages.
* **Execution package** – Creates, logs, and provides workflow for one or more data flows. In addition, the template presented in the Execution package section below provides post-processing activity for insert and update data flow patterns.
* **Execution lineage** – Unique value representing one execution instance of a data flow. As we saw earlier, this value is stored in destinations and provides a link between the destination data and the process that created or modified the data.

### Users and Interfaces

Developers, operations personnel, and data stewards are the primary users of the ETL framework. Figure 3-41 shows how they interface with the ETL framework.



**Figure 3-41:** ETL framework roles and interfaces

Let’s look a little more closely at the key roles and responsibilities involved in the ETL framework:

ETL developers…

* Develop the ETL packages and maintain and extend the ETL templates
* Maintain and extend ETL framework reports
* Assist operations with diagnosing and correcting errors

ETL operations…

* Maintain configuration parameters
* Schedule master packages
* Monitor ETL activity
* Identify and coordinate ETL error diagnostics
* Receive error alerts

Data stewards…

* Review reports for out-of-band activity
* Review, diagnose, and manage data change requests
* Receive exception alerts

Master and execution packages interface with ETL framework database objects.

Master package interfaces…

* Load configurations into SSIS variables and properties
* Log batch activity
* Log errors from master and execution packages
* Send email error and exception alerts
* Send batch and configuration information to execution packages

Execution package interfaces…

* Accept batch and configuration parameters from the master package
* Log data flow activity
* Get/set filter values for incremental loads

Most mature ETL development shops have their own version of an ETL framework. Dynamic configurations and logging are core capabilities, but they may be implemented differently across implementations.

### Configurations

Most ETL frameworks developed for SSIS use either XML configuration files or database tables for dynamic configurations.

XML configurations are typically preferred by developers who are fluent in XML. Configurations are changed by directly editing values stored within the XML file. The XML file is then loaded by either an SSIS XML configuration or a script task.

Database configuration tables are typically preferred by IT operations and DBAs who are less fluent in XML than developers. Another advantage to using configuration tables is that it supports the storing of each instance of the configuration parameters in a table. This can help later on in the diagnostics phase if an incorrect parameter value is specified.

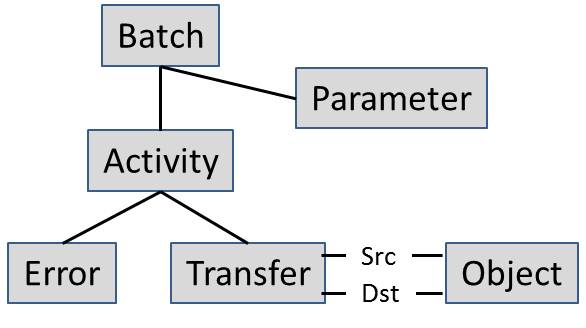
You can find more information about SSIS configurations in the following articles:

* [SQL Server Integration Services SSIS Package Configuration](http://www.mssqltips.com/tip.asp?tip=1405)
* [Simple Steps to Creating SSIS Package Configuration File](http://www.sqlservercentral.com/articles/Integration+Services+(SSIS)/66500/)
* [Reusing Connections with Data Sources and Configurations](http://msdn.microsoft.com/en-us/library/cc671619.aspx)

### Logging

Most ETL frameworks use custom logging. This is usually due to the low-level nature of SSIS logging and because SSIS lacks the concept of a Batch, which provides a logical grouping of activity. This section presents reports from an ETL framework that uses custom logging to record package activity.

The reports we’ll look at in a moment are generated from logging tables, as shown in Figure 3-42.



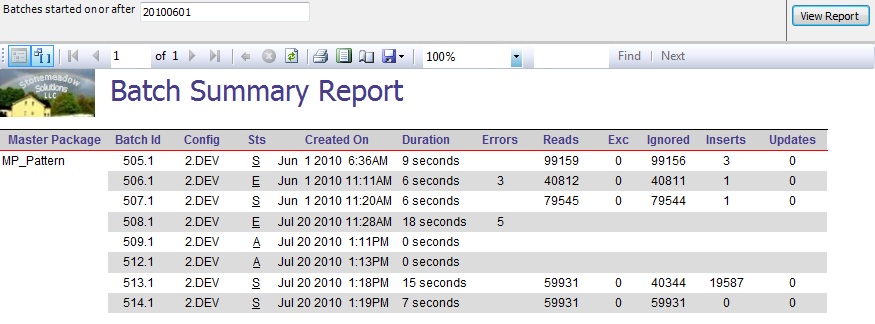
**Figure 3-42:** ETL framework logging tables

This collection of logging tables consists of:

* **Batch** – Contains one record created for each instance of a batch invocation. A batch workflow is typically implemented in a master package, which is described below.
* **Parameter** – Contains one record for each parameter used to dynamically configure a batch instance.
* **Activity** – Contains one record for an activity, a logical construct that can contain 0 or more transfers (a.k.a. data flows).
* **Error** – Contains one record for each error thrown within an ETL activity.
* **Transfer** – Contains one record for a data flow between one source and one destination.
* **Object** – Contains one record for each unique instance of an object used as a source and/or destination.

Now let’s look at some reports that leverage these tables to see the value of centralized logging for all ETL activity within an organization. ETL operations use these reports to gauge the overall health of ETL activity. Note that the examples presented below are all developed using SQL Server Reporting Services (SSRS).

Figure 3-43 shows the highest-level report, displaying ETL batch activity for a specified period of time.



**Figure 3-43:** Batch Summary Report

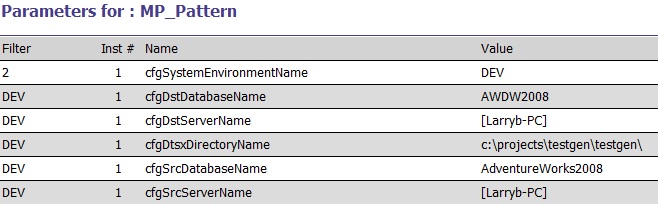
Table 3-10 summaries the report columns and provides a short description of each.

|  |  |
| --- | --- |
| Column Name | Description |
| Master Package | Name of the controller package that manages the ETL batch |
| Batch ID | Unique ID for the batch instance |
| Config | Configuration identifier used to retrieve the batch configuration parameters |
| Sts | Completion status: E = Error, S = Success, A = Active |
| Created On | Time the batch started execution |
| Duration | How long the batch took to execute |
| Errors | Total error count (if any) for this batch execution instance |
| Reads | Total number of records read from the source |
| Exc | Total number of records sent to the exception area |
| Ignored | Total number of records not processed |
| Inserts | Total number of records inserted |
| Updates | Total number of records updated |

**Table 3-10:** Batch Summary Report columns

Note that the ETL operations staff can choose to filter batches started on or after a specified date.

Clicking the Config value in the above report links to another report showing the input values used in the dynamic configuration process, as Figure 3-44 shows.

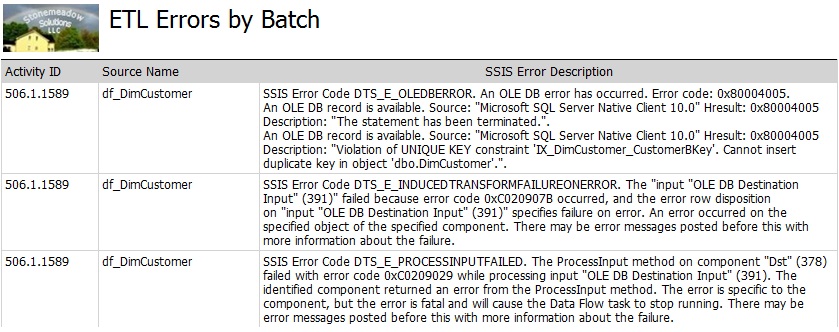


**Figure 3-44:** Batch Parameter Report

This example shows the parameters used to load dynamic configurations for the development (DEV) environment. Note that:

* Each of the parameters directly map to an SSIS package variable. A dynamic configuration task script reads the value from the configuration table. This value in turn is used to populate the SSIS variable.
* ETL operations can change a configuration by modifying values in the ETL framework configuration table.
* The directory to which the ETL packages have been deployed is determined by the cfgDtsxDirectoryName parameter value.
* The source and destination databases are initialized by the cfgxxxServerName and cfgxxxDatabaseName parameter values.

Error details are also a click away. Figure 3-45 shows the report you get when you click the Batch 506 error count field in the Batch Summary Report. This report displays details about all the errors encountered within the batch instance.



**Figure 3-45:** Batch Error Report

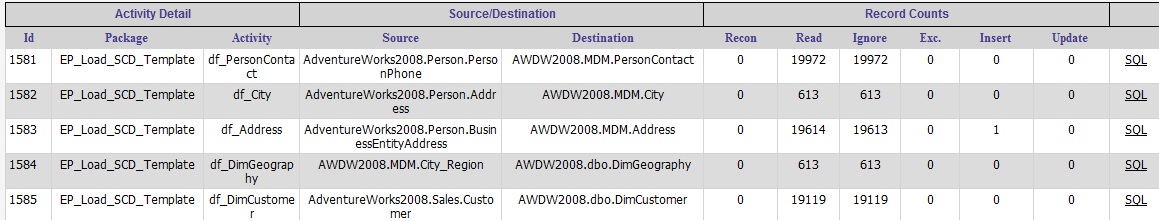
Table 3-11 summaries the Batch Error Report’s columns.

|  |  |
| --- | --- |
| Column Name | Description |
| Activity ID | Identifier for the Activity run instance. Note that 506 is the batch ID, and 1589 is the execution lineage ID. |
| Source Name | The activity that generated the error |
| Error Description | The errors thrown by the SSIS process |

**Table 3-11:** ETL Error Report columns

Notice in Figure 3-45 that there was a violation of a unique key constraint when the SSIS data flow attempted to insert a record that already existed.

Clicking the Batch Id field within the Batch Summary Report displays the Batch Detail Report, shown in Figure 3-46.



**Figure 3-46:** Batch Detail Report

Note that each report line maps to one execution lineage ID. This report provides details on the activity, the data flow source and destination, and record counts. Table 3-12 contains a brief description of each Batch Detail Report column.

|  |  |
| --- | --- |
| Column Name | Description |
| Id | Activity ID, this is the execution lineage ID |
| Package | Package name |
| Activity | Activity name |
| Source | Source name, 3-level naming (Database.Schema.Table) |
| Destination | Destination name, 3-level naming (Database.Schema.Table) |
| Rows | Reconciliation value: Read – Ignore – Exception – Insert – Update |
| Read | Records read |
| Ignored | Records ignored |
| Exc | Records sent to the exception area |
| Inserts | Total number of records inserted |
| Updates | Total number of records updated |
| SQL | Click through to the SQL used to select from the source |

**Table 3-12:** Batch Detail Report columns

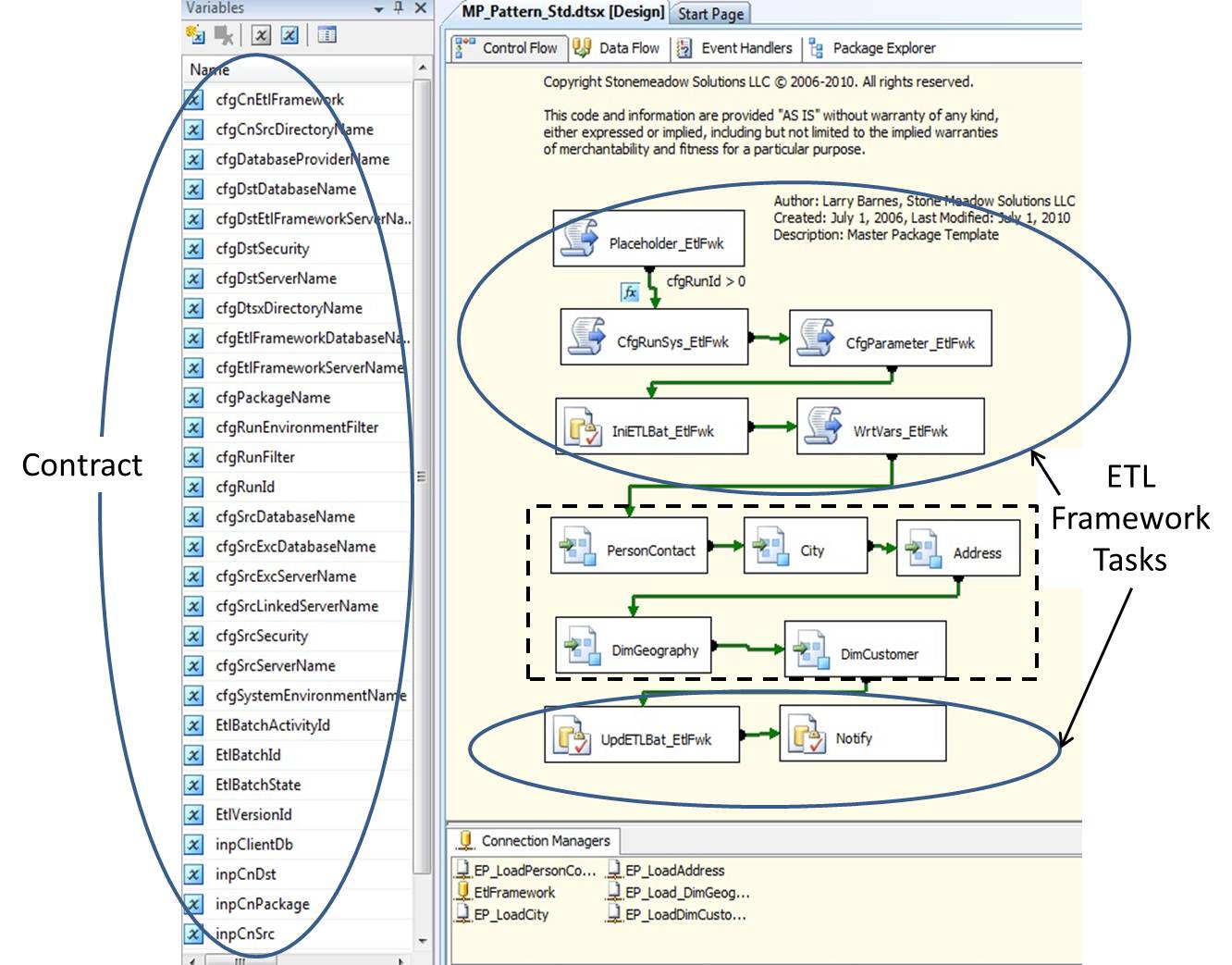
These reports show one approach for centralized custom logging for ETL activity, with all the above reports coming from the logging tables populated by an ETL framework. We will cover the ETL framework and SSIS package templates used to populate these logging tables in the next section.

But first, let’s look at examples of master and execution package templates, which interface with the ETL framework and support table-based dynamic configurations and custom logging.

### Master Package

Master packages control ETL package workflow for one ETL batch. This section provides an overview of different components within a master package.

Figure 3-47 shows a master package.



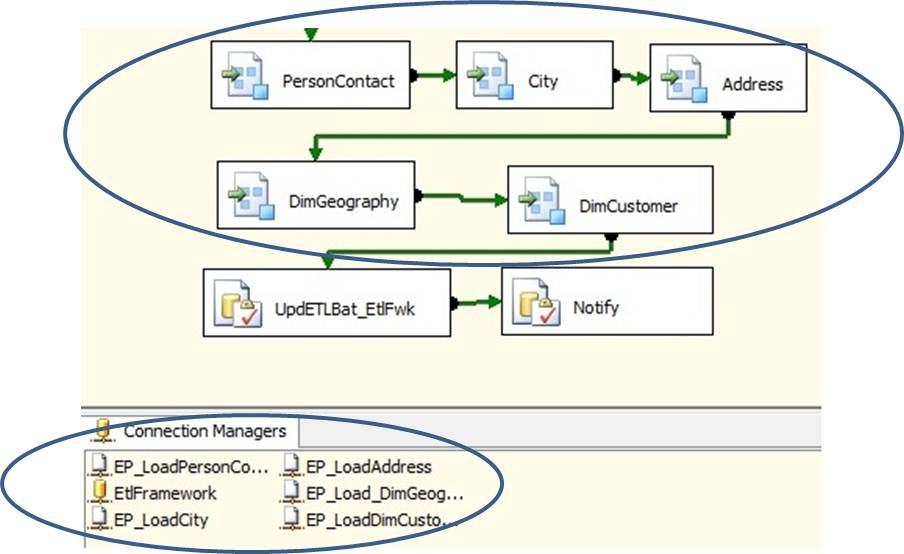
**Figure 3-47:** Master package

A master package is composed of:

* **ETL Framework tasks** – These stock tasks for every master package are responsible for loading/recording dynamic configuration instances, creating/updating the ETL batch record, and sending notifications to interested parties.
* **Execution Package tasks** – The master package developer creates the workflow for the ETL batch by connecting a sequenced set of execution package tasks.
* **Contract** – The master package’s variables can be viewed as the contract between ETL framework components and SSIS connections and tasks.

The master package developer starts with a master package template, which contains the ETL framework tasks used to interface with the ETL framework. The execution package workflow is then added to this master package template. Figure 3-48 shows an example of an execution package workflow.

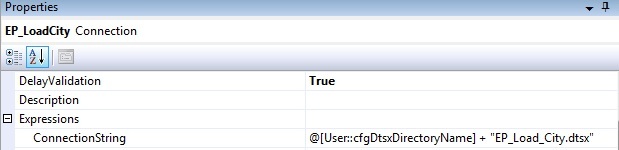
This scenario populates the AdventureWorksDW 2008 DimCustomer and DimGeography tables from the AdventureWorks2008 OLTP database. Notice how three Production data area tables (PersonContact, City, and Address) were populated prior to loading the DimGeography and DimCustomer Data warehouse tables. This illustrates how even simple scenarios require an intermediate Production data area within a data warehouse.



**Figure 3-48:** Master package execution package workflow

Here are the steps for developing this workflow:

1. Determine the workflow for the ETL batch. The above simple example shows the logical steps used to build a Geography and Customer dimension. Again notice how activity is required within the Production data area prior to loading the dimensions.
2. Create one Execution Package task for each separate ETL activity.
3. Create one file connection for each execution package.
4. Set the DelayValidation property to True for each file connection. This allows the file connection to be dynamically initialized at run time.
5. Create an expression for each ConnectionString property. Figure 3-49 shows the expression for the EP\_LoadCity file connection.



**Figure 3-49:** ConnectionString expression

The file directory where the SSIS packages reside is dynamically loaded. This simple expression now allows the ETL operations team to easily deploy packages onto a server by doing the following:

1. Creating a directory to store the ETL packages
2. Copying the ETL packages into the directory
3. Creating configuration entries (this ETL framework uses a Configuration table)
4. Using dynamic configurations to initialize the cfgDtsxDirectoryName variable at runtime

Here are a few developer notes to keep in mind:

* When DelayValidation is set to False, SSIS validates the connection metadata at package open time, not package execution time.
* Setting DelayValidation = False and having a hard-coded directory value stored in the cfgDtsxDirectoryName variable is a common developer oversight.
* The result of the above is that the wrong set of ETL packages can get invoked or, more commonly, the package will fail once it moves from the development environment to the test environment.
* OLE DB sources within data flows have a ValidateExternalMetadata property which is set to a default value of True. When set to False, the Source metadata is not checked at design time which could result in a run-time error when the source metadata changes.

**Error Handling**

The SSIS event model allows developers to build SSIS workflows that are activated when an event fires. Figure 3-50 shows the workflow for the master package On Error event.



**Figure 3-50:** On Error workflow

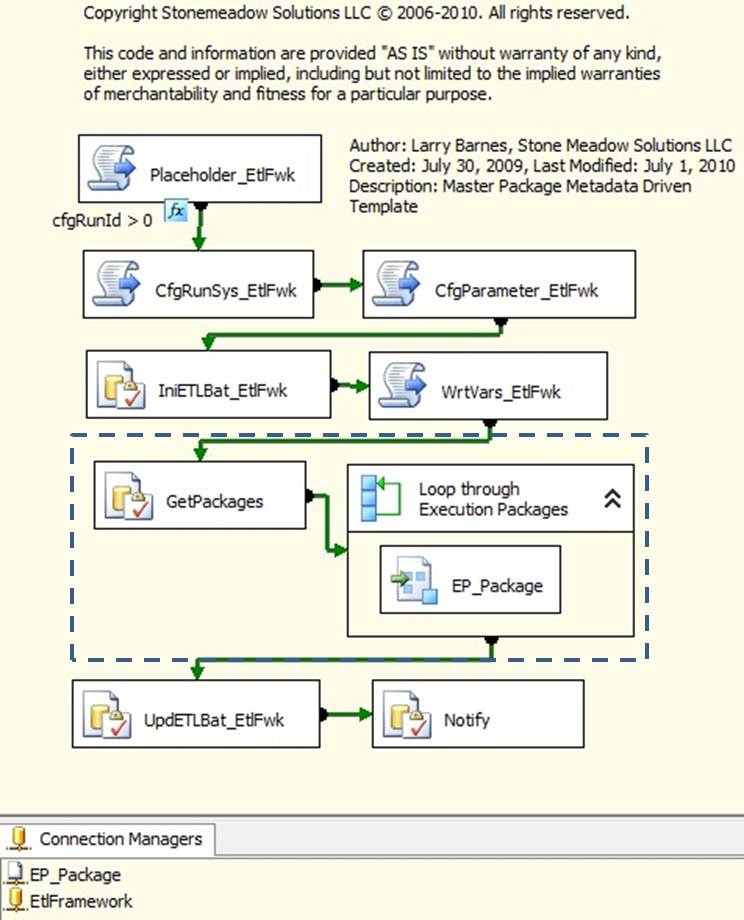
This workflow writes an error record to the ETL framework then optionally sends an alert if the error results in termination of the master package. This error record is populated with a combination of SSIS user and system variables.

SSIS system variables contain additional information that is useful when diagnosing the error. The following link has more information on the system variables available to the SSIS developer: <http://technet.microsoft.com/en-us/library/ms141788.aspx>

Note that this master package’s On Error event also captures all errors from execution packages invoked within the master package. Adding specific error handlers for execution packages is necessary only if error logic specific to the execution package is required (e.g., whether to continue processing instead of terminating processing).

**Master Package Example: Metadata-Driven Approach**

An alternative strategy for master packages is a metadata-driven approach—storing a sequenced set of execution packages within a table and using them to dynamically configure an Execution task. Figure 3-51 shows an example of a metadata-driven master package.

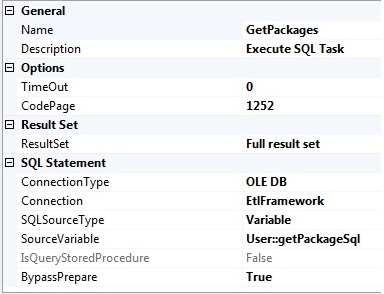


**Figure 3-51:** Metadata-driven master package

Metadata-driven master package tasks include:

* A GetPackages Execute SQL task that queries a table containing a sequenced set of execution packages and returns an ADO result set into a variable. Figure 3-52 shows this package’s properties and the SQL statement used to return the result set.
* A Foreach Loop Container to iterate through the ADO result set returned by GetPackages. This result set contains an ordered set of execution packages, and each record value is mapped to an SSIS variable.
* An EP\_Package Execute Package task that references the EP\_Package connection.
* The EP\_Package connection, which uses the following expression to dynamically set the execution package location and name:

*@[User::cfgDtsxDirectoryName] + @[User::rsPackageName] + ".dtsx"*



**Figure 3-52:** GetPackages properties with getPackageSql variable value

This sample master package allows execution packages to be added and removed from the execution flow without any changes to the code. The getPackageSql variable contains the following expression:

*"SELECT DISTINCT PackageName, SortOrder FROM MDM.EtlPackage WHERE ActiveFlag = 'A' AND MasterPackageName = '" + @[System::PackageName] + "' ORDER BY SortOrder "*

Note how the master package name, stored in the *System::PackageName* variable, is used to filter the result set.

### Execution Package

Now let’s look at an example of an execution package. Figure 3-53 shows an execution package template. Note that every task in this template is an ETL Framework task. The ETL developer is responsible for adding the data flow and initializing the Contract variables to drive task behavior.



**Figure 3-53:** Execution package template

This execution package template includes the following tasks:

* **InitVars\_EtlFwk** – Retrieves the dynamic configuration information created and persisted by the master package by using the batch ID as a filter. The batch ID is initialized using a Parent Package Variable configuration.
* **InitActivity** – Creates an ETL framework activity record. This record’s primary key is the Execution Lineage Id and is stored in the destination’s LineageId column by the data flow.
* **BuildSqlFilter** – Retrieves the last maximum value for the source system column used to log changes (e.g., ModifiedDate). This value will be inserted into the source’s Select statement to filter the number of records processed to the ones that have changed since the last invocation of the package instance.
* **dfContainer** – The container in which the ETL developer creates the data flow. Note the comments section, which lists the output variables that the data flow must populate.
* **PostProcess** – Three tasks that combine to implement set-based processing after the data flow has completed. These set-based tasks are the ones mentioned in the ETL Patterns section above. The BuildSql task creates the set-based Updates and Inserts, which are then executed by one of the UpdateTable and InsertHistory tasks. The ETL developer is responsible for setting the following variables:
  + ETL pattern – Is this an update or versioned insert pattern?
  + Natural key(s) – The natural key list is used in the update and versioned insert set-based updates to join instances of a record together.
  + Update Columns – These columns are the values updated in the update pattern.
* **UpdateActivity** – The task that updates the ETL framework activity record with execution status and completion date/time. It also inserts transfer record details (e.g., source and destination names, reconciliation row counts, and the source SQL statement).

This section’s sample execution package template isn’t meant to be the definitive example. But it does illustrate how the following key requirements can be implemented:

* **Dynamic configurations** – The dynamic initialization of connections provides for hands-off management as ETL solutions move from development to production environments.
* **Batch and execution lineage** – Creating an execution lineage ID allows data stewards to navigate through a table’s change history when business consumers question data warehouse content. Analyzing particular batch runs is possible because the lineage IDs all have a batch parent record.
* **Incremental loads** – Adding a filter to source systems containing a change column date or integers significantly reduces the records processed.
* **Post processing** – Using set-based Updates to post-process update and versioned insert patterns is more efficient than including the Updates within a procedural data flow.

This ETL framework example uses a file-based storage option for its master and execution packages. However, there are other options, and the next section provides some guidance on the key criteria for choosing a package storage approach.

### Package Storage Options

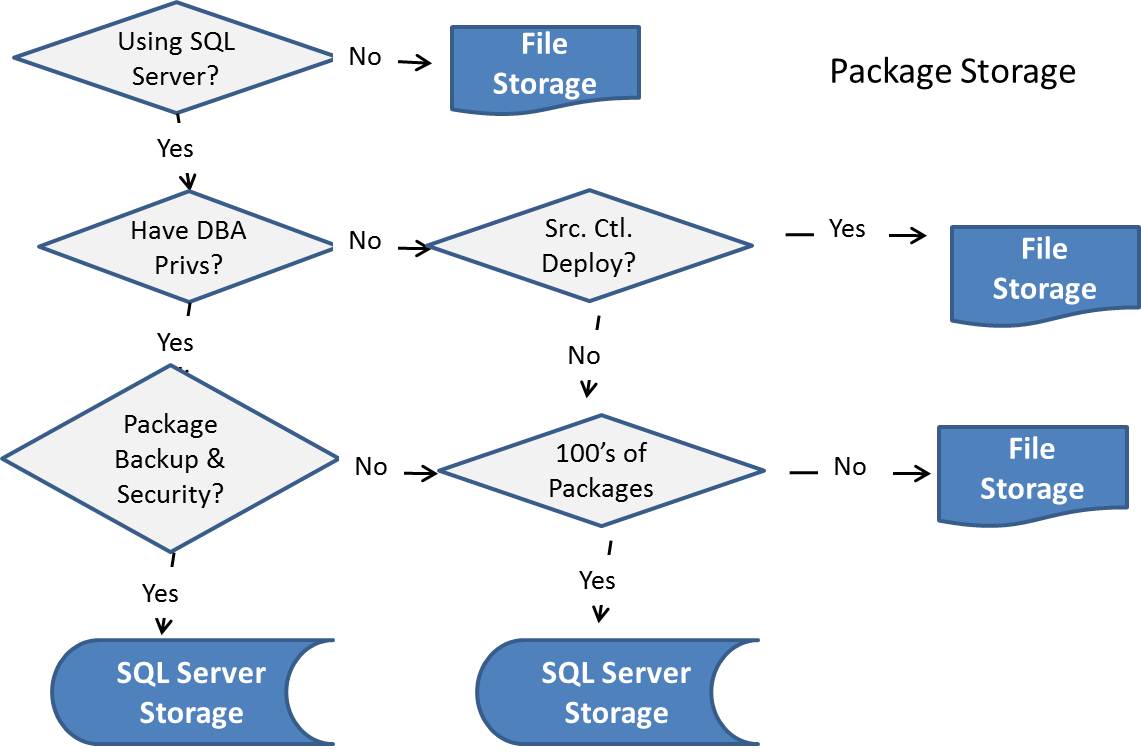
The master package example above uses a file-based option for storing SSIS packages. The benefits to the file-based storage are:

* A file copy can be used to transfer ETL packages between environments. File operations (copy and backup) are well understood by IT operations resources responsible for deploying solutions into production and QA environments.
* The SSIS service is not required for this option, which eliminates an additional moving part and simplifies the environment. For more information on the SSIS service, click on this link: <http://msdn.microsoft.com/en-us/library/ms137731.aspx>.

However, there are two other package-storage options:

* **Storing packages within the msdb database.** One benefit to this option is security: The administrator can use built-in roles (e.g., db\_ssisadmin, db\_ssisltduser and db\_ssisoperator) when granting access to a package.
* **Storing the packages within a directory monitored by the SSIS service.** A benefit to this option, which requires the SSIS service, is that the SSIS service monitors all package activity on a server.

The question then becomes, which package storage option is the most appropriate for your environment? The workflow illustrated in Figure 3-54 can be used to select the package storage option most appropriate for your environment.



**Figure 3-54:** Package storage options workflow

Essentially, SQL Server package storage is appropriate when you’re using SQL Server as a relational database and:

* Package backup and security is important
* …or you have hundreds of packages

File package storage is appropriate when:

* You don’t use SQL Server as a relational database
* …or your source control system is used to control package deployment
* …or you have a limited amount of packages

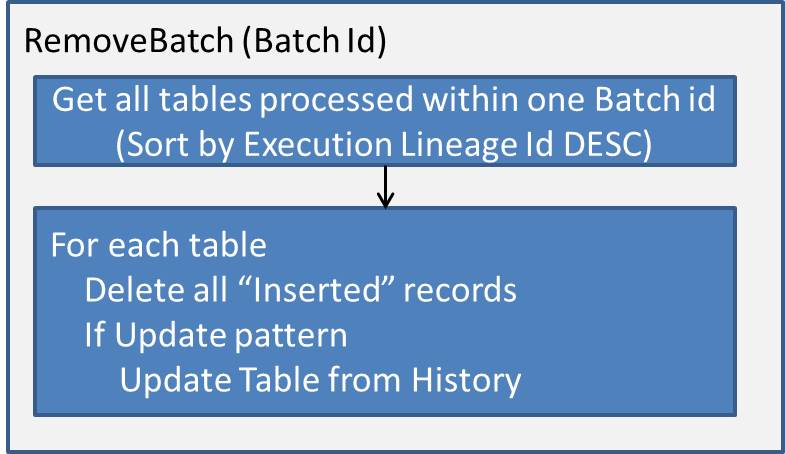
The article [Managing and Deploying SQL Server Integration Services](http://technet.microsoft.com/en-us/library/cc966389.aspx) contains good information about managing and deploying SSIS solutions.

### Backing Out Batches

As discussed earlier, the ability to back out all new and changed records for a particular batch execution instance should be a requirement for every ETL framework. This approach is necessary because ETL batches are long-running, I/O-intensive processes that don’t lend themselves to being efficiently encapsulated within a database transaction.

**Note:** It’s critical that data stewards locate processing errors as soon as possible. Backing out an ETL batch that occurred in the past (e.g., two weeks ago) would most likely require the backing out of all recent batches due the fact that incremental changes to a bad record would also result in bad data.

Execution lineage and custom logging help to simplify the task of backing out changes; Figure 3-55 shows the high-level flow of the procedure used to back out all changes from a recently executed batch.



**Figure 3-55:** Batch back-out code

The SQL used to retrieve the table set for this ETL framework is shown below:

*SELECT DISTINCT d.SchemaName + '.' + d.ObjectName as DestinationTable, x.ActivityId as LineageId, A.BatchId*

*FROM LOG.EtlXfr x INNER JOIN LOG.EtlActivity a*

*ON x.ActivityId = a.ActivityId*

*INNER JOIN LOG.EtlObject d ON x.DstId = d.ObjectId*

*WHERE A.BatchId = @BatchID*

*ORDER BY x.ActivityId DESC*

Table 13 shows the result set from this query for the scenario presented in the execution package section above.

|  |  |  |
| --- | --- | --- |
| DestinationTable | LineageId | BatchId |
| dbo.DimCustomer | 1618 | 513 |
| dbo.DimGeography | 1617 | 513 |
| MDM.Address | 1616 | 513 |
| MDM.City | 1615 | 513 |
| MDM.PersonContact | 1614 | 513 |

**Table 13:** Destination table result set for BatchId 513

Notice how the records are sorted in descending order based on the LineageId. This allows you to iterate through the record set without worrying about conflicts such as primary key–foreign key constraints. The master package executes ETL packages in an order that honors any referential integrity. The following sequence will be executed for each table. (Note that this section refers to the versioned insert and update integration patterns, which are covered in the next section.)

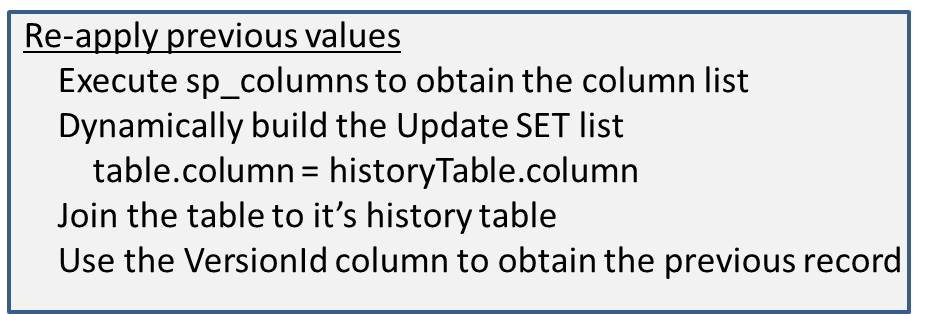
1. The first step is to delete all the newly inserted records. This is simple with the versioned insert table pattern because the only operations on the table are inserts:

*DELETE FROM @DestinationTable WHERE LineageId = @LineageId*

This is also a simple operation for the Update table pattern when there’s a version ID column or some other indicator that this was the first operation on that record, as shown below:

*DELETE FROM @DestinationTable WHERE LineageId = @LineageId AND VersionId = 1*

1. Update table patterns will require one more operation: re-applying any recently updated fields to their previous value. This is where History tables are useful. Figure 3-56 shows the pseudo-code used to dynamically build this UPDATE statement.



**Figure 3-56:** Code to re-apply the previous version

The following is pseudo-code for the UPDATE statement generated from the logic in Figure 3-56:

*UPDATE @TableName*

*SET tableName.ColumnName = h.ColumnName,...*

*FROM @TableName*

*INNER JOIN @HistoryTable h ON tableName.NaturalKey = h.NaturalKey*

*WHERE LineageId = @LineageId AND h.VersionId = tableName.VersionId - 1*

### More ETL Framework Information

The ETL framework presented in this section is available online at <http://etlframework.codeplex.com/>

It includes:

* Scripts to create the ETL framework database
* ETL framework reports
* Master and execution package templates
* Execution package examples

## Data Integration Best Practices

Now that you have a good understanding of SSIS data integration concepts and patterns, let’s look at some examples that put those ideas and patterns into action.

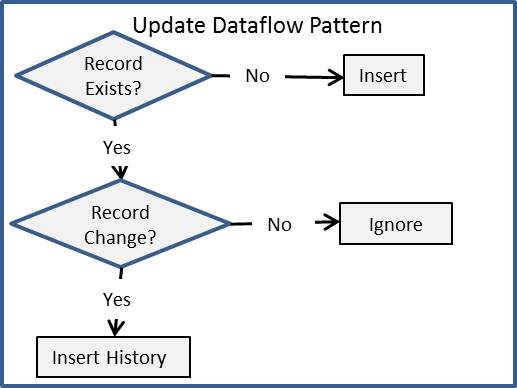
### Basic Data Flow Patterns

This section presents SSIS examples of two fundamental patterns that almost all ETL data flows are based on. These patterns are the update pattern and the versioned insert pattern. Each of these patterns can be applied to table types made popular by Kimball—Slowly Changing Dimensions Type I (SCD I) and Type 2 (SCD II) and fact tables:

* SCD I – Uses the update pattern
* SCD 2 – Uses the versioned insert pattern
* Fact table – The type of pattern you can use depends on the type of fact table being loaded. (We’ll discuss this in more detail later in this chapter).

**Update Pattern**

A table containing only the current version of a record is populated using an update data flow pattern, as Figure 3-57 shows.

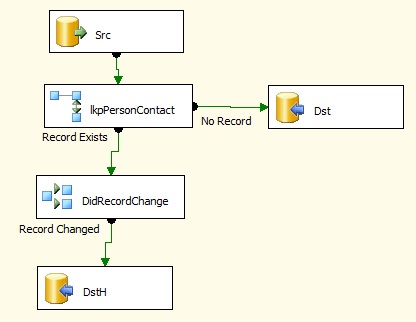


**Figure 3-57:** Update data flow pattern

Some things to keep in mind about the update data flow:

* This data flow only inserts rows into a destination. SSIS destinations have a “Table or view fast load” option, which supports very fast bulk loads.
* You can update records within a data flow by using the OLE DB Command transform. Note that using this transform results in one SQL Update per record; there is no concept of a bulk-load update within a data flow. Because of this, the pattern we present only inserts records and never updates records within the data flow.
* Every table has an associated history table that is responsible for storing all activity: One record is inserted into the history table every time the source record exists in the destination but has different values.
  + The alternative is to send the changed records into a working table that is truncated after the Update processing completes.
  + The benefit of keeping all records in a history table is that it creates an audit trail that can be used by data stewards to diagnose data issues raised by business consumers.
* There are different options for detecting record changes; these will be covered later in this chapter.

Figure 3-58 shows an SSIS update pattern data flow.

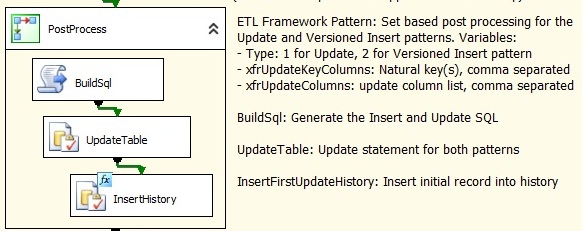


**Figure 3-58:** SSIS update data flow pattern

This SSIS update data flow contains the following elements:

* **Destination table lookup** – The lkpPersonContact transform queries the destination table to determine whether the record already exists (using the Natural key).
* **Change detection logic** – The DidRecordChange conditional split transform compares the source and destinations. The record is ignored if there’s no difference between the source and destination. The record is inserted into the history table if the source and destination differ.
* **Destination Inserts** – Records are inserted into the destination table if they don’t already exist.
* **Destination History Inserts** – Records are inserted into the destination history table if they do exist.

After the data flow completes, a post-processing routine is responsible for updating the records in the primary table with the records stored in the history table. This is implemented by the execution package template’s post-processing tasks, as Figure 3-59 shows.

****

**Figure 3-59:** SSIS update post processing tasks

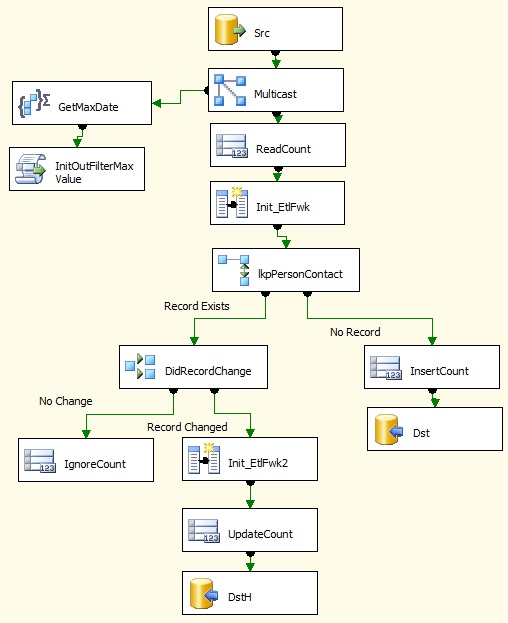
The execution package PostProcess container is responsible for all set-based activity for this pattern, including:

* Updating the table’s records using the records inserted into the history table.
* Inserting all new records (first version and into the history table). The table’s primary key is an IDENTITY column whose value isn’t known until after the insert. This means that we need to wait until after the records are inserted before moving them to the history table.

**Update Pattern – ETL Framework**

Figure 3-60 shows the updated data flow with ETL framework instrumentation transformations. These transformations:

* **Initialize supporting columns** – The Init\_EtlFwk and Init\_EtlFwk derived column transformations initialize the Record Status, Lineage Id, and Version Id supporting columns documented earlier in this chapter.
* **Save record counts** – The ReadCount, IgnoreCount, InsertCount, UpdateCount, and Row Count transformations initialize SSIS variables with record counts. These in turn are inserted into ETL framework logging tables by a post-processing task.
* **Net changes** – The GetMaxDate aggregate transformation calculates the maximum value for the Modified On date. The InitOutFilterMaxValue transformation stores this into an SSIS variable, which is then inserted into an ETL framework table. It will be used to retrieve only the changed records the next time this data flow runs.



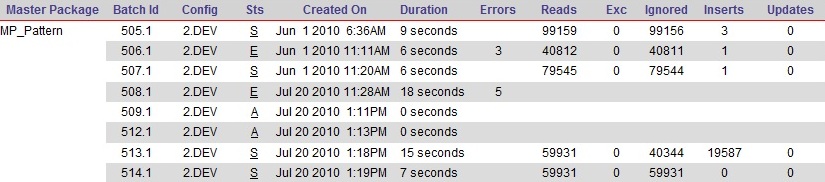
**Figure 3-60:** SSIS update data flow pattern with ETL instrumentation

**The Benefit of This Data Flow**

Your first observation might be that this data flow looks a lot more complicated that the original one. That might be the case because most frameworks require extra up-front time in the development phase. However, this is time well spent when you can avoid the daily costs of reconciling data.

The benefits to adding this additional logic are as follows:

* Supporting columns make the auditing and data stewardship tasks much easier. (See the discussion above for the benefits of adding execution lineage.)
* Row counts provide good indicators of the health of a particular data flow. A data steward can use these record counts to detect anomalies in processing and data patterns for an execution instance. Figure 3-61 shows an example of the Batch Summary Report with record counts.

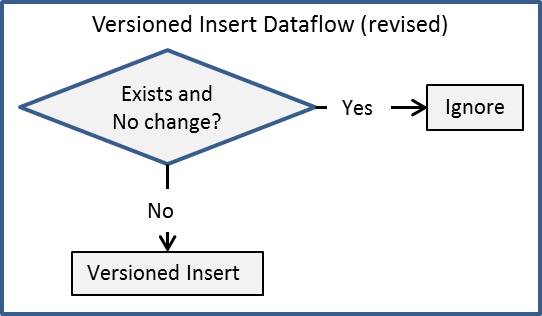
****

**Figure 3-61:** Batch Summary Report with record counts

ETL operations staff and data stewards can use the summary record counts to understand more about the activity within a particular batch instance. Further investigation, including drilling into detail reports, can then occur if any of the record counts look suspicious—either by themselves or in relation to other batch instances.

**Versioned Insert Pattern**

A table with versioned inserts is populated using a versioned insert data flow pattern, as shown in Figure 3-62. Note that the logic flow has been revised for efficiency.

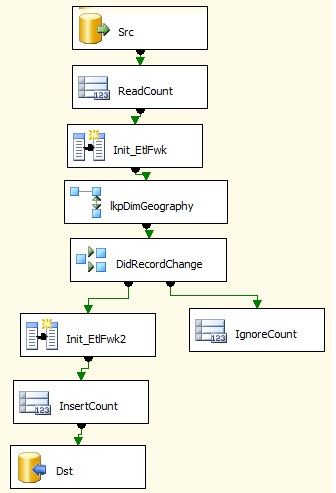


**Figure 3-62:** Versioned insert data flow pattern

Note that with the versioned insert data flow:

* All new/changed records go into the Versioned Insert table.
* This is a simpler data flow than the update data flow pattern.

Figure 3-63 shows an SSIS versioned insert data flow.



**Figure 3-63:** SSIS versioned insert data flow pattern

Some notes about this SSIS update data flow:

* The lkpDimGeography Lookup transform has the “Redirect rows to no match output” option set to “Ignore Failures.”
* The DidRecordChange conditional split task checks whether the destination’s primary key is Not Null and if the source and destination columns are the same. The record is ignored if this expression evaluates to true.
* The net change logic is not included—in this case, there’s no Modified Date column in the source.
* The ETL framework supporting column and record count transformations are similar to the previous data flow.

**Update vs. Versioned Insert**

The versioned insert pattern is simpler to implement and has less I/O activity than the update pattern. So why bother with the update pattern at all?

The answer is: There are fewer records in the update table. Fewer records translate into fewer I/Os, which mean better performance.

OK, so why do we need a history table? Couldn’t we truncate this table after it’s used to update the primary table?

The answer to these questions is: The history table allows data stewards and data auditors to track changes over time. This can be invaluable when business consumers question results from the data warehouse.

### Surrogate Keys

Surrogate keys play an integral part in every data warehouse. Determining how surrogate keys are generated is an important aspect of any data integration effort. The options for key generation when using SQL Server are well known, and each option has its pros and cons. These options are:

* **GUIDs** – These are 16-byte keys that are guaranteed to be unique across all systems.
  + GUIDs are not recommended for surrogate keys.
  + Their size (16 bytes) is at least two times larger than big integers and four times larger than integers.
* **IDENTITY** – This property generates the next unique value for a column and can only be associated with one column in a table.
  + Although Identity columns are a solid option for key generation, the key value is not determined until after an insert occurs.
  + This means the ETL developer cannot use this value within the SSIS dataflow responsible for the inserts, making implementation of patterns such as “Late arriving dimensions” difficult.
* **ETL key generation** – The ETL process itself is responsible for generating the surrogate keys. The steps within this pattern are:

1. Obtain the base value for the surrogate key (i.e., the starting point).
2. Calculate the new surrogate key.
3. Optional: Store the last surrogate key into a table for the future reference.

This following snippet shows code within a data flow script component that calculates surrogate keys:

Dim gNextKey As Integer = 0

Public Overrides Sub Input0\_ProcessInputRow(ByVal Row As

Input0Buffer)

'

' Init the next key with the Max key value the first time around

' Increment by 1 to create the next unique key

'

If gNextKey = 0 Then

gNextKey = Row.MaxId

End If

gNextKey = gNextKey + 1

End Sub

**Key Lookups**

Every ETL operation involves joining data from a source to a destination at some level. This could include:

* Dimension surrogate key lookups
* Code-to-value lookups
* Primary key lookups across systems
* Master data management key lookups

Using a relational database SQL join solution is not always an option, nor is it always efficient for the following reasons:

* When the source data and lookup table are on different environments, you would need to land the source data to the same RDBMS before performing the join.
* Flat files are a common source for data warehouse ETL operations and often require looking up codes or keys, which can’t be done at the file layer. Even if data is stored in the same type of RDBMS system, such as SQL Server, but on different servers, a cross-server join has major performance implications—especially with large data sets.
* The Data-in or Production environment is usually not on the same server as the Consumption database or databases, so you would need to add another working area to accomplish an efficient join.

Of course, using a SQL join should not be taken off the table. You should do a healthy evaluation of all options to select the appropriate solution.

SSIS has three primary mechanisms to perform key lookups, each with benefits:

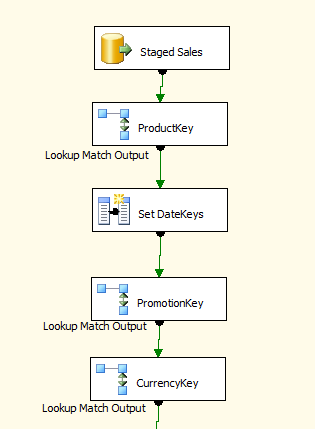
* Lookup transformation
* Merge Join transformation
* Fuzzy Lookup transformation

Let’s look briefly at each of these transformations.

**Lookup Transformation**

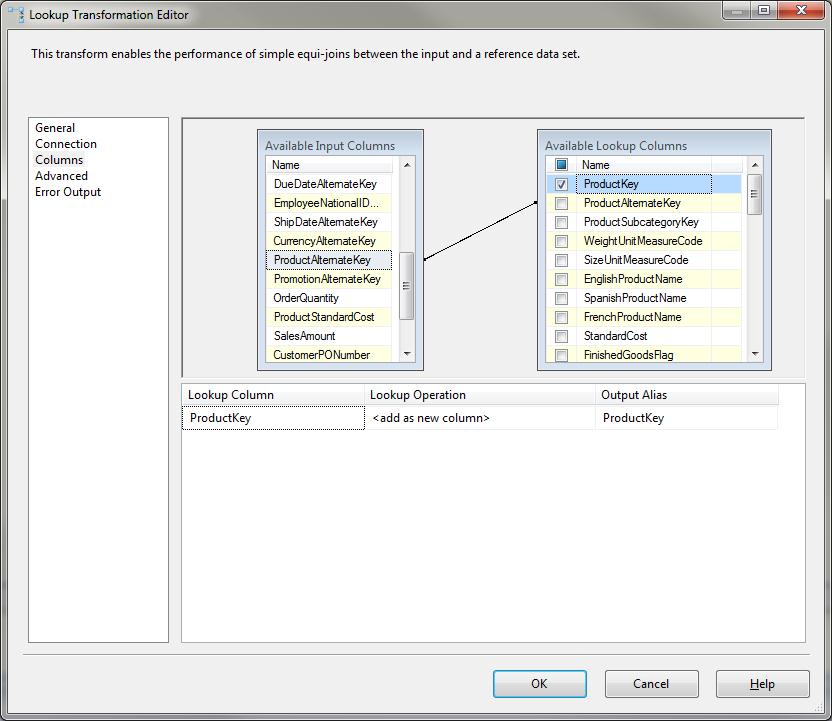
SSIS’s Lookup transformation is the most common and easiest-to-use solution. It works by matching the source rows in the data flow to a table or view that has been defined in the transformation. Usually, the Lookup is configured to load the entire table into the SSIS memory space, which alleviates the need for a join to the database and is very efficient.

Figure 3-64 shows a data flow that uses lookups to acquire keys. Note that even though the lookups are strung together in the data flow, the SSIS engine is performing the lookups simultaneously in the data flow.



**Figure 3-64:** SSIS Lookup transformation

Figure 3-65 highlights the product lookup column mapping that acquires the data warehouse product key by matching across the source key.



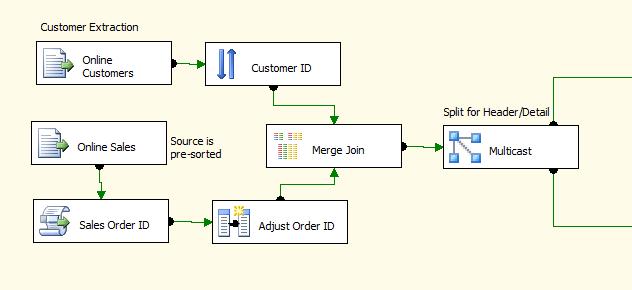
**Figure 3-65:** SSIS Lookup column matching

Here are some considerations about when to use the Lookup transformation:

* The Lookup works well when the entire lookup reference table can fit into memory. If your lookup reference table has several million rows or your ETL server is limited in memory, you will run into problems and should consider another route.
* As a rule of thumb, a 1M record table that is about 50-100 bytes wide will consume about 100MB of memory. A large 64-bit server with a lot of memory available for ETL can handle large lookup tables.
* The Lookup can be configured without cache or with partial cache. No cache means that every row will run a query against the RDBMS. Do not use this approach if you want a scalable solution. Partial cache is when the cache gets loaded as rows are queried and matched against the source system.
* You can share Lookup cache across Lookup transformations. This is a valuable capability if you need to use the same lookup table at multiple times during the same ETL.
* When the lookup does not find a match, you can either fail the process or ignore the failure and return a NULL to the output.

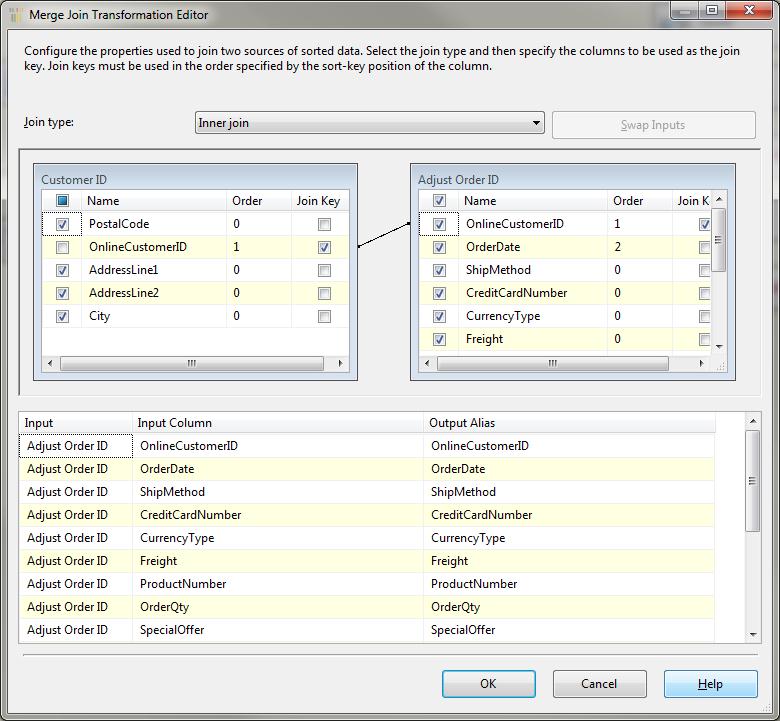
**Merge Transformation**

A second solution to looking up keys is to use the Merge transformation. Merge does a lot more than just a lookup because you can perform different join types across data coming from any source. The Merge transformation requires that the data be sorted in the order of the keys that are joined. You can use a Sort transformation for this, or if the source is already sorted (physically or through an ORDER BY), you can configure the source adapter to be pre-sorted. Figure 3-66 shows an example of a Merge Join lookup.



**Figure 3-66:** Merge Join data flow

In this example, one of the sources is already pre-sorted, and the second one uses a Sort transformation. The Merge Join brings the data together, in this case joining transactions from a flat file to a customer list also from a flat file. Figure 3-67 shows the Merge Join Transformation Editor for this example.



**Figure 3-67:** Merge Join Transformation Editor

Here, an inner join is used. However, left and full outer joins are also available (as well as a right outer join). The sources are joined across the keys, and the data sets are merged.

The following are key considerations for when to use the Merge transformation:

* The Merge Join adds a level of complexity over and above the Lookup transformation. So if your reference table will easily fit in memory, use the Lookup; otherwise, the Merge Join can be effective.
* The Merge Join allows the data to be joined even when there is more than one match per key. If you need each merge input to be matched with zero, one, or more records in the other input, the Merge Join will do that. The Lookup will always return only one row per match, even if there are duplicate records with the same key.

**Fuzzy Lookup Transformation**

* The third approach to looking up data for data warehouse ETL is the Fuzzy Lookup transformation. Very similar to the Lookup transformation, the Fuzzy Lookup joins to a reference table not on exact matches, but on possible matches where the data may not be exact but is similar.
* Note that the Fuzzy Lookup requires the lookup reference table to be in SQL Server. It also has significant processing overhead, but its uses when dealing with bad data are valuable.

### Change Detection

We briefly discussed change detection earlier, but the following examples will provide more detail. Remember that change detection involves:

* Detecting changes in existing data
* Identifying new or deleted records

Some systems track changes through audit trails, others append timestamps to records (such as CreationDate or ModifiedDate), and still other systems don’t have any identifiers.

**Working with Change Identifiers**

When working with a system that tracks changes, you can use SSIS to easily identify the changed records and process them. For example, if you have a system with CreationDatetime and LastModifiedDatetime timestamps appended to the rows and you need to process inserts and updates, the following steps are needed:

1. Merge the create and last-modified date into one value—for example:

*ISNULL(LastModifiedDatetime, CreationDateTime)*

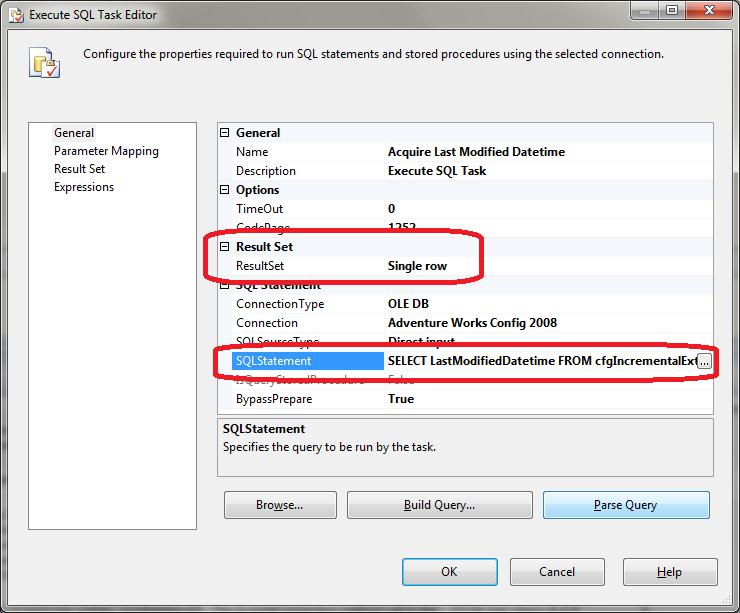
Note that this value is referred to as LastModifiedDatetime within this example.

1. Identify the maximum value of the last modified date.
2. Extract and process the changed records:

*ISNULL(LastModifiedDatetime, CreationDateTime) > LastModifiedDatetime*

1. Save the maximum value of the above ISNULL comparison.

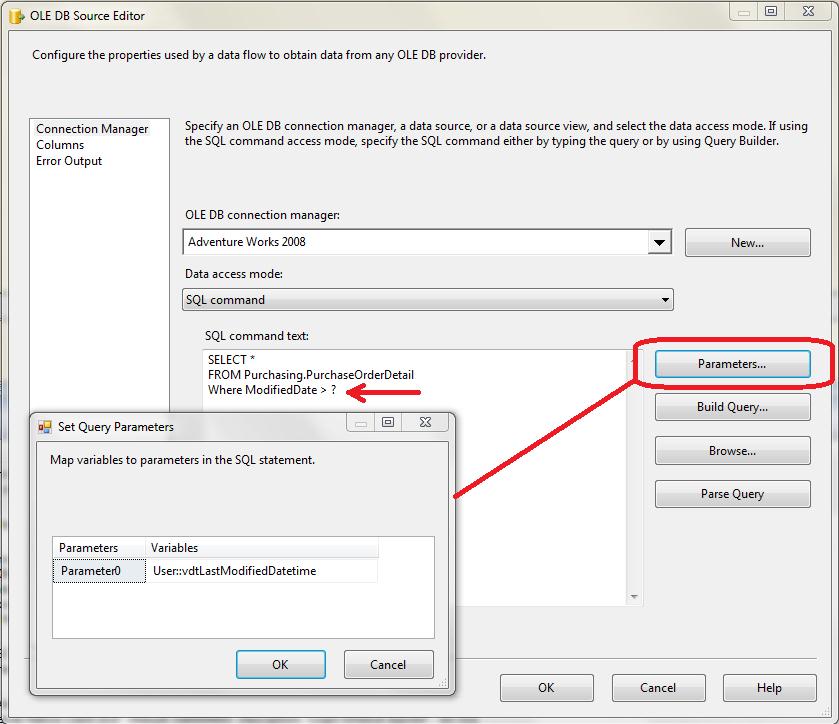
Steps number one and four imply that the LastModifiedDatetime is stored. Assuming this value is captured in a SQL Server table, SSIS can extract the table data and store the results in a package variable. Figure 3-68 shows an Execute SQL task configured to receive the results of a query into a package variable.



**Figure 3-68:** Execute SQL task and variables

Not shown in Figure 3-68 is the Result Set tab, which maps the data from the query to the variable. An alternative approach is to use a parameterized stored procedure and map input and output parameters to variables using the Parameter Mapping tab in the editor.

The second step is to extract the targeted data from the source based on the LastModifiedDatetime. You can use either a parameterized query or a variable that contains the SQL statement. Figure 3-69 shows a source adapter with a parameterized query.



**Figure 3-69:** Parameterized query

The alternative approach is to set the Data Access Mode to SQL Command from Variable and build the SQL query before the data flow task. This approach works well for non-SQL sources that do not support parameterized queries.

The final step is to save the MAX of the created or modified date to a table. You can do this with a MAX SQL query, or if you are using the data flow, you can use the Max aggregate in the Aggregate transformation.

Your situation may require slightly different steps, but this process can be adjusted to accommodate other scenarios where you have some way to identify changes.

Note that the Figure 3-69 shows a parameterized query for an OLE DB source. Working with parameters can differ across Data providers, e.g. OLE DB, ODBC, ADO and ADO.Net. The following link contains more information on this topic: <http://technet.microsoft.com/en-us/library/cc280502.aspx>.

**Change Identification through Data Comparison**

If you have a system where you are not able to identify changes through flags, dates, or audit trails, you will need to handle change detection by comparing the data. There are several approaches to doing this; the most common include:

* SQL Join Comparison
* SSIS Conditional Split
* SQL Checksum
* Slowly Changing Dimension Transformation

**SQL Join Comparison**

The first solution is to compare the data by joining the source data to the destination data across the keys and then comparing the attributes in the predicate WHERE clause. Typically, the SQL pattern to update the changes looks like this:

UPDATE [Source Table]

SET [Source Attribute Columns] = [Destination Attribute Columns]

FROM [Source Table]

INNER JOIN [Destination Table]

ON [Source Keys] = [Destination Keys]

WHERE [Source Attribute Columns] <> [Destination Attribute Columns]

If you are also identifying new records, then you would perform a LEFT OUTER JOIN where the keys are NULL, as follows:

SELECT [Source Columns]

FROM [Source Table]

LEFT OUTER JOIN [Destination Table]

ON [Source Keys] = [Destination Keys]

WHERE [Destination Key] IS NULL

The drawbacks to this approach include the need to stage the source data to a table in the same SQL instance and the process overhead to perform the joins.

**Conditional Split**

Another solution is to use an SSIS Conditional Split to compare the data from the source and destination. In the next section on dimension patterns, we look at an example where a Merge Join brings the data together and then a Conditional Split compares the results.

The Conditional Split transformation uses the SSIS expression language to evaluate a Boolean condition for every row. The following SSIS expression compares the columns from the source and the destination and for each column:

GeographyKey != DW\_GeographyKey || Title != DW\_Title || FirstName != DW\_FirstName || MiddleName != DW\_MiddleName|| LastName != DW\_LastName || BirthDate != DW\_BirthDate || MaritalStatus != DW\_MaritalStatus || Suffix != DW\_Suffix || Gender != DW\_Gender || EmailAddress != DW\_EmailAddress || TotalChildren != DW\_TotalChildren || NumberChildrenAtHome != DW\_NumberChildrenAtHome || HouseOwnerFlag != DW\_HouseOwnerFlag || NumberCarsOwned != DW\_NumberCarsOwned || AddressLine1 != DW\_AddressLine1 || AddressLine2 != DW\_AddressLine2 || Phone != DW\_Phone || DateFirstPurchase != DW\_DateFirstPurchase || CommuteDistance != DW\_CommuteDistance

Notice how the above expression assumes there are no NULL values in source or destination fields. The logic required when NULLS exist in the source and/or destination would be more complex. This is one argument for eliminating NULLs within your ETL data flow.

**SQL Checksums**

You can also use checksums (a computed hash of the binary data) across all the source and destination columns that need to be evaluated for change. In your destination table, you can generate this checksum by using the T-SQL CHECKSUM operator. To compare the destination to the source data, you need to perform the same CHECKSUM against the source from the Production area environment.

However, here are a couple of cautions when using checksums:

* Checksums or other hash algorithms are not guaranteed to be 100% accurate.
* You need to be absolutely sure that the data types and column order are the same on both the source and destination columns.

**Slowly Changing Dimension Transformation**

The final approach is to use the SSIS Slowly Changing Dimension (SCD) transformation. This built-in transformation allows comparison of source data with destination data to identify new records, changes that require an update, changes that require an insert (for preserving history), and updates to handle inferred members (when the destination is merely a placeholder key).

The SCD transformation uses the paradigm of the Ralph Kimball dimension change types, but its use goes beyond just dimension processing.

Although the SCD component is very useful for smaller tables with a few thousand rows or smaller, it does not scale well for table comparisons with hundreds of thousands or millions of rows. It incurs a lot of overhead by making a call to the database for every row to compare, every row to update, and every row for inserts.

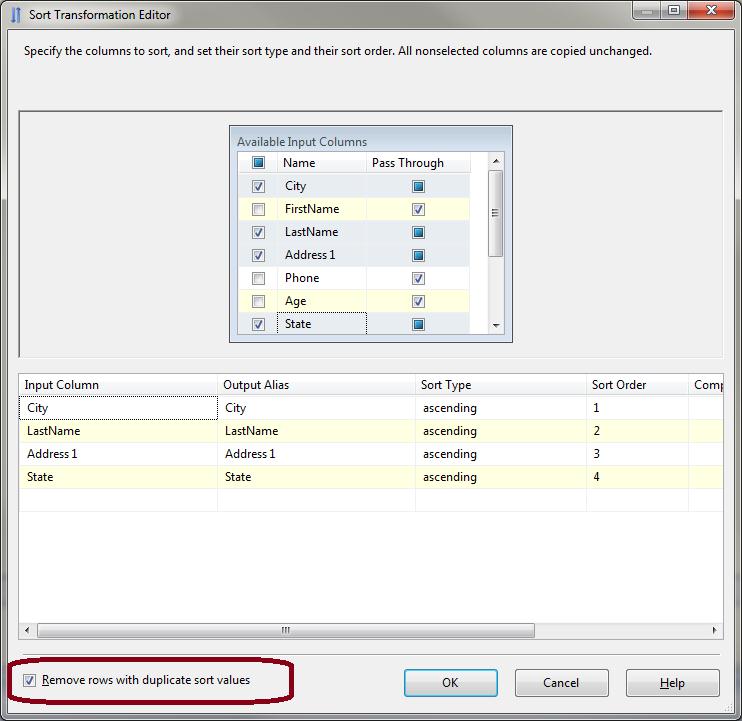
### De-duping

Especially when dealing with merged sources, duplication is often an issue. In fact, duplication can occur in any system, whether from an operator error or a customer signing up again on a site because they forgot their login.

SSIS has two built-in approaches for dealing with standard de-duplication: the Sort transformation and the Fuzzy Grouping transformation. Of course, more complicated de-duplication can be achieved through a combination of features.

**Sort Transformation with De-duplication**

The Sort transformation has the ability to remove duplicates across the selected sorted columns. Figure 3-70 shows the Sort Transformation Editor with the option to *Remove rows with duplicate sort values* selected.



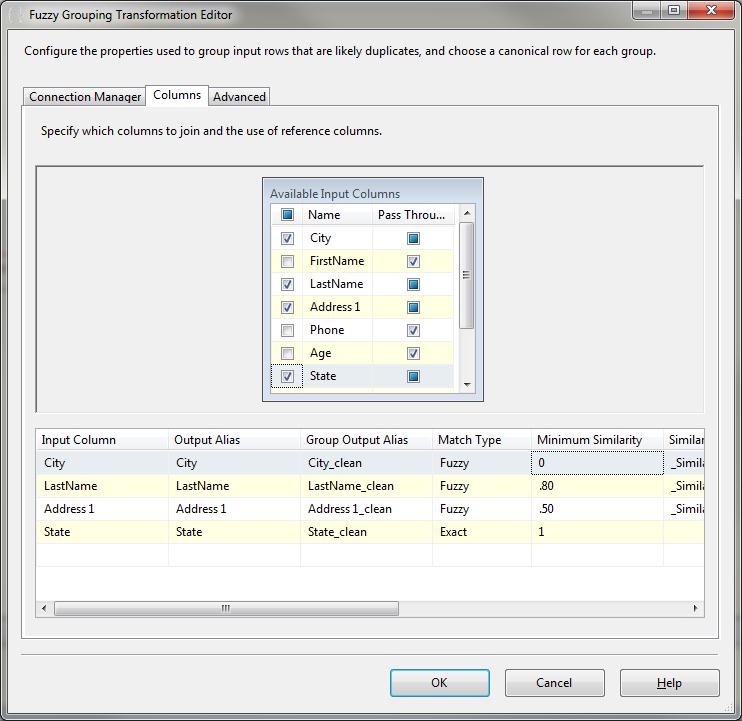
**Figure 3-70:** Sort Transformation Editor

The de-duplication feature of the Sort transformation requires that the key values match exactly. Notice that you can also pass through the other columns that aren’t involved in the de-duplication. A simple GROUP BY or DISTINCT SQL statement can do a similar operation, but if you are dealing with a file or your set of distinct columns is a subset of the column list, the Sort transformation can be a valuable option.

**Fuzzy Grouping Transformation**

The Fuzzy Grouping transformation is effective at de-duplication when the data set you are working with requires identifying duplicates across similar but not exactly matching data.

Using the Fuzzy Grouping transformation is like using any other transformation: You connect it to the data flow data set and then configure it. Figure 3-71 shows the Fuzzy Grouping Transformation Editor configured against the same data as the previous Sort transformation.



**Figure 3-71:** Fuzzy Grouping Transformation Editor

In this example, the State is set to Exact match, which means that the State has to be identical for the engine to identify more than one record as a duplicate. The other columns have similarity thresholds set as needed. Although not shown, the Advanced tab has an overall Similarity Threshold, which applies to all the columns defined in the column list.

A word of caution: Fuzzy Grouping leverages the SQL Server Database Engine for some of its work, and while it has powerful application for dealing with data quality issues, it also has a high overhead—meaning that you should perform volume testing as you are planning the ETL solution.

### Dimension Patterns

We’ve discussed many ways of loading and updating dimensions throughout this toolkit, such as how to deal with updates, record versioning, and generating keys.

Updating dimensions involves:

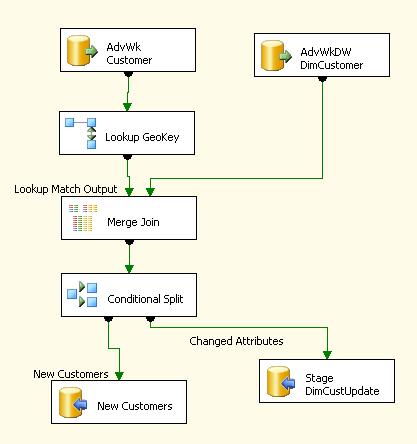
* Tracking history
* Making updates
* Identifying new records
* Managing surrogate keys

If you are dealing with a smaller dimension (in the magnitude of thousands of rows or less, as opposed to hundreds of thousands or more), you can consider using the built-in dimension processing transformation in SSIS called the Slowly Changing Dimension transformation. However, because this transformation has several performance-limiting features, it is often more efficient to build your own process.

The process of loading dimension tables is really about comparing data between a source and destination. You are typically comparing a new version of a table or a new set of rows with the equivalent set of data from an existing table. After identifying how the data has changed, you can then perform a series of inserts or updates.

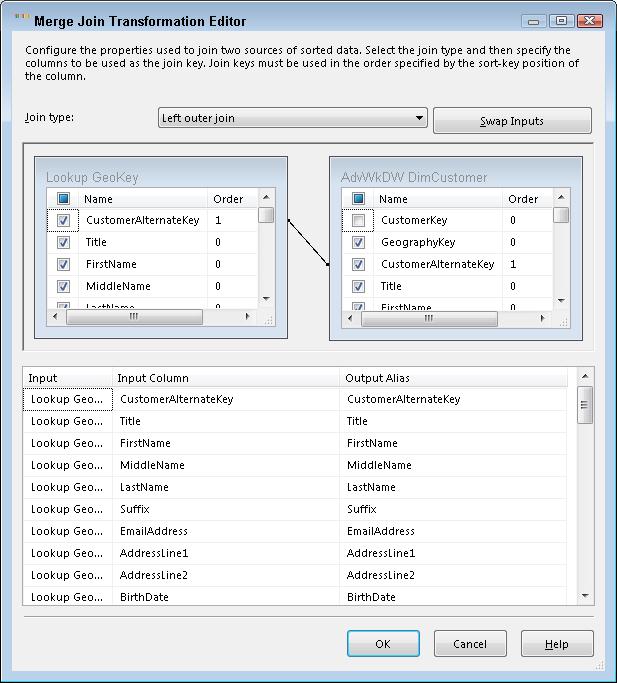
Figure 3-72 shows a quick dimension processing example, with the following general steps:

1. The top left source adapter pulls records into SSIS from a source system (or intermediate system). The top right source adapter pulls data from the dimension table itself.
2. The Merge Join compares the records based on the source key (see Figure 3-73 for the transformation details).
3. A conditional split evaluates the data, and rows are either inserted directly into the dimension table (bottom left destination adapter) or inserted into a Production data area table (bottom right destination) for an update process.
4. The final step (not shown) is a set-based update between the Production data area table and the dimension table.



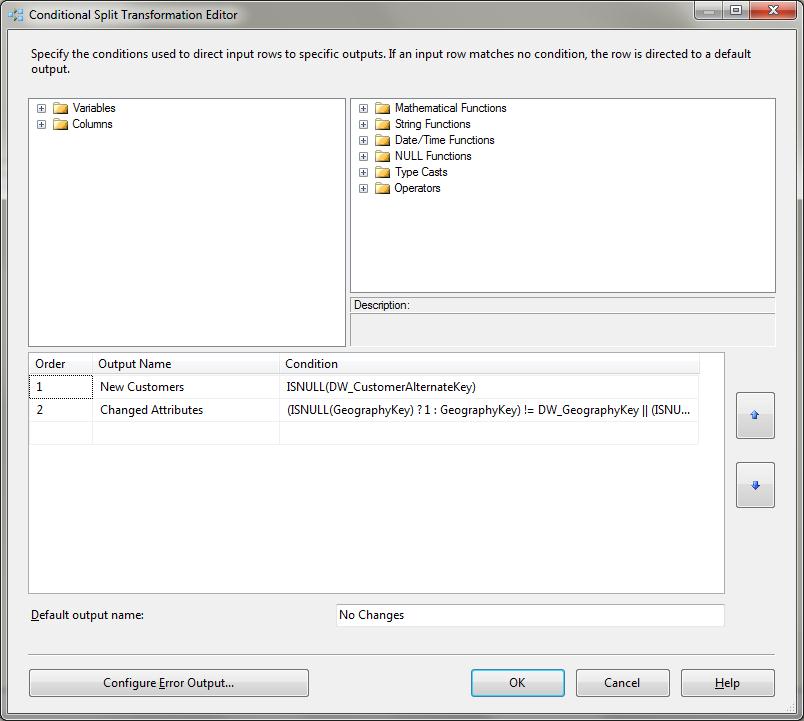
**Figure 3-72:** Dimension processing

The Merge Join performs the correlation between the source records and the dimension records by joining across the business or source key (in this case, CustomerAlternateKey). Figure 3-73 shows the setup in the Merge Join Transformation Editor. When you use this approach, be sure to set the join type to left outer join, which will let you identify new records from the source that are not yet present in the dimension table.



**Figure 3-73:** Using Merge Join for dimension processing

The last step is to compare data to determine whether a record is new or changed (or unaffected). Figure 3-74 shows the Conditional Split transformation, which does this evaluation. (The Conditional Split uses the code displayed earlier in the Change Detection section.)



**Figure 3-74:** Conditional Split transformation

The Conditional Split redirects the records directly to the dimension table through a destination adapter or to an working update table using a destination adapter followed by a set-based UPDATE statement.

The UPDATE statement for the set-based approach joins the working table directly to the dimension table and performs a bulk update, as follows:

UPDATE AdventureWorksDW2008.dbo.DimCustomer

SET AddressLine1 = stgDimCustomerUpdates.AddressLine1

, AddressLine2 = stgDimCustomerUpdates.AddressLine2

, BirthDate = stgDimCustomerUpdates.BirthDate

, CommuteDistance = stgDimCustomerUpdates.CommuteDistance

, DateFirstPurchase = stgDimCustomerUpdates.DateFirstPurchase

, EmailAddress = stgDimCustomerUpdates.EmailAddress

, EnglishEducation = stgDimCustomerUpdates.EnglishEducation

, EnglishOccupation = stgDimCustomerUpdates.EnglishOccupation

, FirstName = stgDimCustomerUpdates.FirstName

, Gender = stgDimCustomerUpdates.Gender

, GeographyKey = stgDimCustomerUpdates.GeographyKey

, HouseOwnerFlag = stgDimCustomerUpdates.HouseOwnerFlag

, LastName = stgDimCustomerUpdates.LastName

, MaritalStatus = stgDimCustomerUpdates.MaritalStatus

, MiddleName = stgDimCustomerUpdates.MiddleName

, NumberCarsOwned = stgDimCustomerUpdates.NumberCarsOwned

, NumberChildrenAtHome = stgDimCustomerUpdates.NumberChildrenAtHome

, Phone = stgDimCustomerUpdates.Phone

, Suffix = stgDimCustomerUpdates.Suffix

, Title = stgDimCustomerUpdates.Title

, TotalChildren = stgDimCustomerUpdates.TotalChildren

FROM AdventureWorksDW2008.dbo.DimCustomer DimCustomer

INNER JOIN dbo.stgDimCustomerUpdates

ON DimCustomer.CustomerAlternateKey

= stgDimCustomerUpdates.CustomerAlternateKey

### Fact Table Patterns

Fact tables have a few unique processing requirements. First, you need to acquire the surrogate dimension keys and possibly calculate measures. These tasks can be handled through Lookup, Merge Join, and Derived Column transformations.

The more difficult process is dealing with the updates, record differentials, or snapshot table requirements.

**Inserts**

Most fact tables involve inserts—it is the most common fact table pattern. Some fact tables have only inserts, which makes the ETL process perhaps the most straightforward approach. Inserts also involve bulk loading of data, index management, and partition management as necessary.

We’ll talk more about how to best handle inserts in the Destination Optimization section later in this chapter.

**Updates**

Updates to fact tables are typically handled in one of three ways:

* Through a change or Update to the record
* Via an Insert of a compensating transaction
* Using a SQL MERGE

In the case where changing fact records are less frequent or the update process is manageable, the easiest approach is to perform an UPDATE statement against the fact table. (See the earlier section on Change Detection for ways to identify changes.) The most important point to remember when dealing with Updates is to use a set-based update approach, as we showed in the Dimension Pattern section..

The second approach is to insert a compensating or net change record, rather than performing an Update. This strategy simply inserts the data that has changed between the source and the destination fact table into a new record. For example, Table 3-13 shows the current source value, Table 3-14 shows what exists in the fact table, and Table-15 shows the new record.

|  |  |
| --- | --- |
| Source ID | Measure Value |
| 12345 | 80 |

**Table 3-13:** Source data

|  |  |
| --- | --- |
| Source ID | Measure Value |
| 12345 | 100 |

**Table 3-14:** Current fact table data

|  |  |
| --- | --- |
| Source ID | Current Measure Value |
| 12345 | 100 |
| **12345** | **- 20** |

**Table 3-15:** New fact table data

The last approach is to use a SQL MERGE statement, where you land all the new or changed fact data to a working table and then use the merge to compare and either insert or update the data. The following example code shows that when the merge does not find a match, it inserts a new row; when it finds a match, it performs an update:

MERGE dbo.FactSalesQuota AS T

USING SSIS\_PDS.dbo.stgFactSalesQuota AS S

ON T.EmployeeKey = S.EmployeeKey

AND T.DateKey = S.DateKey

WHEN NOT MATCHED BY target

THEN INSERT(EmployeeKey, DateKey, CalendarYear, CalendarQuarter, SalesAmountQuota)

VALUES(S.EmployeeKey, S.DateKey, S.CalendarYear, S.CalendarQuarter, S.SalesAmountQuota)

WHEN MATCHED AND T.SalesAmountQuota != S.SalesAmountQuota

THEN UPDATE SET T.SalesAmountQuota = S.SalesAmountQuota

;

The drawback to the MERGE approach is performance. Although it simplifies the insert and update process, it also performs row-based operations (one row at a time). In situations where you are dealing with a large amount of data, you are often better off with a bulk-load insert and set-based Updates.

**Snapshot Fact Table**

A snapshot fact table is when you are capturing measure balances on a recurring basis, such as weekly or monthly inventory levels or daily account balances. Instead of capturing the transactions, the snapshot fact table is summarizing the balance. In other words, there will be one record per product or account per period.

Take the example of inventory captured once a week. Table 3-16 shows what the snapshot fact table would look like.

|  |  |  |
| --- | --- | --- |
| Product SK | Week SK | Quantity In Stock |
| 1 | 1001 (2011 Week 1) | 10 |
| 2 | 1001 (2011 Week 1) | 8 |
| 3 | 1001 (2011 Week 1) | 57 |
| 4 | 1001 (2011 Week 1) | 26 |
| 5 | 1001 (2011 Week 1) | 0 |
| 1 | 1002 (2011 Week 2) | 5 |
| 2 | 1002 (2011 Week 2) | 6 |
| 3 | 1002 (2011 Week 2) | 40 |
| 4 | 1002 (2011 Week 2) | 21 |
| 5 | 1002 (2011 Week 2) | 15 |

**Table 3-16:** Weekly Product Inventory fact table

This example shows each product’s inventory quantities for two weeks. If you were to ask what the total inventory is for ProductSK = 1, the answer would not be 15, inventory quantities are not additive. Rather it is either: the average, the first, or the last value.

The ETL for snapshot fact tables is commonly handled through one of two approaches:

* If the source has the current data but no history, you simply perform an Insert every time you reach the first day of the new time interval, using the new intervals time key (Week SK in the example above).
* On the other hand, if the source is just tracking changes to levels, you perform the updates to the latest inventory levels, and then on the first day of the new time interval, you use the prior interval’s current data as the basis for the new time period.

The difference is that one process uses the source data and creates a snapshot of the source data, and the other process uses the current fact table data as the snapshot. When creating a new snapshot from the existing fact table, you can consider using a SELECT INTO operation. If you are creating a new partition in a fact table (a SQL partition), create a table with the new data and then switch that table into the partitioned table.

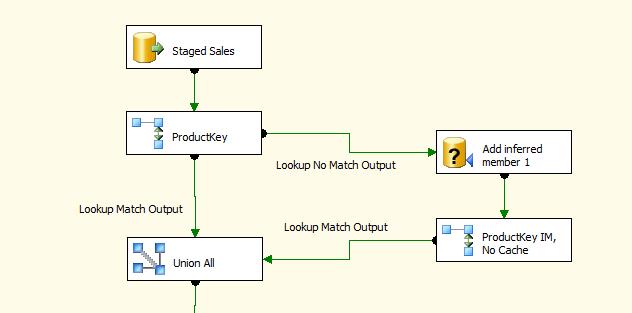
**Managing Inferred Members**

An inferred member is when you are missing the dimension member as you load the fact table. To handle this, you add a placeholder dimension record during the dimension processing.

You have three ways to manage inferred members:

* Scan the staged fact records before inserting them, create any dimension inferred members at that time, and then load the fact records.
* During the fact load, send any records that are missing to a temporary table, add the missing dimension records, and then reload those fact records to the fact table.
* In the data flow, when a missing record is identified, add the record to the dimension at that time, get the surrogate key back, and then load the dimension.

Figure 3-75 shows the third option.



**Figure 3-75:** Inferred member process

In this process, the record with the missing product key is sent to an OLE DB Command transformation, where a procedure is called to add the record to the table (after checking that it didn’t exist). The non-cached Lookup transformation then gets the surrogate key back, and the data is brought back into the data flow with a Union All transformation.

An alternative approach is to use a Script Component to handle the inferred member insert. If you can manage the surrogate key generation in the package, you can optimize the process. As an example of this kind of scripting, see the data flow scripting example in the Power of Data Flow Scripting section later in this chapter.

### Data Exception Patterns

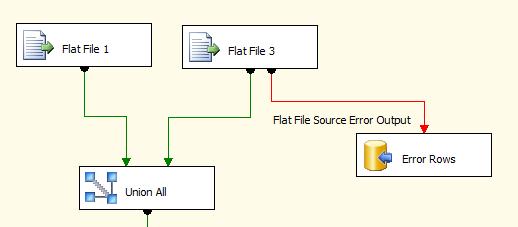
As we noted earlier in this chapter, processing data exceptions can be a difficult if not identified early in the ETL operations. Here’s a common scenario:

*You created a nightly process that loads data into a Production data area table from several sources, and you wrote a SQL INSERT statement that includes some data conversions for loading data into a reporting table. The data contains a few hundred million rows and typically takes 45 minutes to run (indexes are not dropped and re-created). Early one morning, you get a call from the off-shore ops department that the process failed. It ran for 40 minutes, hit a failure, and then took 60 minutes to roll back. Now you have only 30 minutes before users start to run reports against the table.*

Sound familiar? The challenge is that a single data exception can cause a lot of lost time. The solution is to deal with data quality, exceptions, and conversions early in the data warehouse data integration process and leverage the data flow error-row handling in SSIS.

**Data Flow-based Pattern**

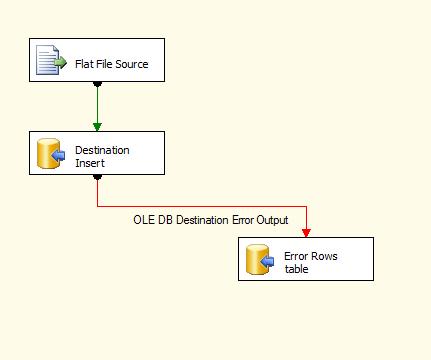
Data flow exceptions can be handled for sources, transformations, and destinations and may be caused by a data conversion error or truncation. The most common data exceptions happen when importing data from a text file, given the loose nature of text file data types and the strong nature of data types in SSIS. The best practice is to redirect failure rows to an exception table for review. Figure 3-76 shows the error row output of the Flat File Source adapter.



**Figure 3-76:** Source Adapter error row output

Because there was a conversion error in one of the input columns, all the input columns are converted to a text string and passed to the error output (which will allow the data to be reviewed) along with the error code and description. Note that the error table can either contain all of the source data or just the key values .

Now let’s look at a destination error. Figure 3-77 shows a data flow where the error rows during the insert are sent to a temporary table for a manual error review.

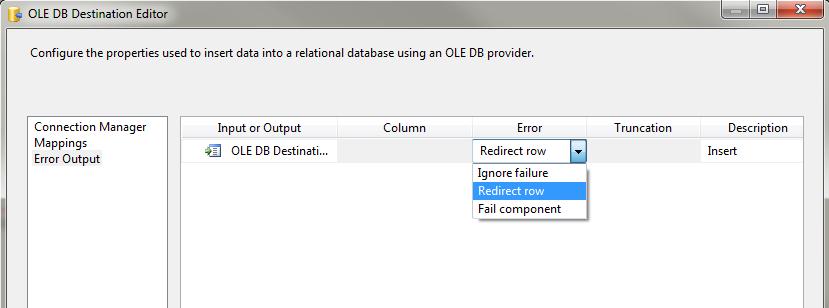


**Figure 3-77:** Data Flow error-row output

You have several options for handling errors in SSIS data flow components (sources, transformations, and destinations):

* Package failure
* Redirecting errors (as in the above example)
* Ignoring the issue. In this case, a NULL can be used instead of the value that caused the exception.

Figure 3-78 shows how you configure error row handling.



**Figure 3-78:** Error row configuration

The following are considerations when dealing with error rows:

* Use a text column when redirecting error rows to an error table. Otherwise, the conversion or exception will also occur in the temp table.
* When redirecting error rows in a destination and Fast Load is enabled, the entire batch commit size will get redirected. This is fine as long as you then try to insert one row at a time from the batch into the destination table and do a second redirect to a temporary table.

**Data Flow-based Exception Handling**

Both of the above scenarios are a reactive approach: They log all data exceptions when they are encountered. The objective for a well-run ETL development shop is to convert the reactive approach to a proactive approach.

A proactive approach adds additional transformation logic to a data flow or within the source SQL statement to correct the problem inline or flag that particular column as a data exception. This may not always be possible because some data is critical to business consumers and cannot be propagated to the destination if in error. However, other data exceptions are less critical and, in these cases, can flow to the destination as a NULL value, a code indicating that the data was in error, or a transformed version of the data exception.

Consider the AdventureWorksDW2008 DimCustomer table’s NumberChildrenAtHome column. This column is a tinyint data type. A source system value of -129 or 130 will result in a data exception due to a data overflow error. However, the following values can also be considered data exceptions: -1, 35. The value of NumberChildrenAtHome is also a data exception when its exceeds the TotalChildren value.

Adding the following logic to the source SQL statement applies these rules:

Case

WHEN NumberChildrenAtHome < 0 THEN NULL

WHEN NumberChildrenAtHome > 25 THEN NULL

WHEN NumberChildrenAtHome > TotalChildren THEN NULL

END as RepairedNumberChildrenAtHome

The data flow logic can now store the RepairedNumberChildrenAtHome value in the destination. In addition, it can also flow this record (using a Multi-cast transformation) to an exception table when the RepairedNumberChildrenAtHome value is NULL and the NumberChildrenAtHome value is NOT NULL.

Note that there are many ways to implement the above business rules within SSIS. The point is that pro-active rather than reactive data exception handling will reduce the time that data stewards will need to spend tracking down and repairing exceptions within the source data. This in turn results in lower TCO and reduces the risk of data exception analysis affecting the data warehouse availability to business consumers.

**Set-based Exception Handling**

Another approach is to pre-process the source data using set-based SQL logic to detect and log the source data to an exception table prior to it entering the data flow. For the NumberChildrenAtHome example we just looked at, the following SQL pseudo-code would detect and log the data exceptions:

INSERT ExceptionTable (Column List)

SELECT (Column List) FROM Source.Customer

WHERE (NumberChildrenAtHome < 0 OR NumberChildrenAtHome > 25 OR NumberChildrenAtHome > TotalChildren )

Note that this isn’t a scalable approach because each data exception check is implemented within separate set-based SQL statements. However, this approach does lend itself nicely to a metadata-driven solution. A metadata-driven solution would consist of a SQL code generator that would read from a metadata table containing the business rules. Table 3-17 shows an example of a metadata table used to drive a SQL data exception code generator.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Database | Table | Rule | Description |
| 1 | AdventureWorks2008 | Person.Person | NumberChildrenAtHome < 0 OR NumberChildrenAtHome > 25 | Incorrect value for the NumberChildrenAtHome column |
| 2 | AdventureWorks2008 | Person.Person | NumberChildrenAtHome > TotalChildren | NumberOfChildrenAtHome is greater than TotalChildren |

**Table 3-17:** Business rules for a set-based data exception implementation

Note that the goal of this example is to demonstrate the concept; the AdventureWorks2008 Person table stores the NumberChildrenAtHome value within an XML data type.

## SSIS Best Practices

The previous section contained best practices for common patterns. This section focuses on SSIS best practices for specific technical scenarios.

### The Power of Data Flow Scripting

With the variations of systems, customizations, and integration requirements out there, chances are you will run into a situation that is not easily solved by either a set-based SQL or SSIS out-of-the-box solution. SSIS scripting in the data flow is an alternative strategy that in many cases can be the most effective solution to a problem.

As a simple example, take the trending of products over time where the source is in constant variation and the full history of inventory needs to be processed nightly, but the system tracks inventory through Diffs. Table 3-18 shows the source.

|  |  |  |
| --- | --- | --- |
| **Week** | **Product** | **Stock Difference** |
| 48 | A | 10 |
| 50 | A | 3 |
| 51 | A | 5 |
| 52 | A | -1 |
| 50 | B | 5 |
| 52 | B | -4 |
| 49 | C | 1 |
| 50 | C | 5 |
| 51 | C | -1 |

**Table 3-18** – Inventory differences

If the goal is to fill in missing weeks where there was no stock difference and to track the total stock for each product on a weekly basis, this can be a challenging problem—especially when you are dealing with approximately 500 million records that need to get updated nightly. Table 3-19 shows the desired output.

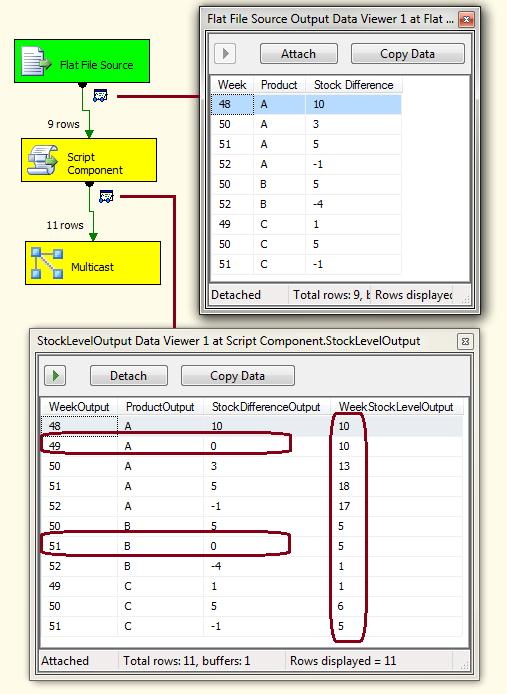
|  |  |  |  |
| --- | --- | --- | --- |
| **Week** | **Product** | **Stock Difference** | **Stock Week Level** |
| 48 | A | 10 | 10 |
| 49 | A |  | 10 |
| 50 | A | 3 | 13 |
| 51 | A | 5 | 18 |
| 52 | A | -1 | 17 |
| 50 | B | 5 | 5 |
| 51 | B |  | 5 |
| 52 | B | -4 | 1 |
| 49 | C | 1 | 1 |
| 50 | C | 5 | 6 |
| 51 | C | -6 | 0 |
| 52 | C |  | 0 |

**Table 3-19** – Inventory stock week levels

If you were thinking about how to accomplish this solution in SQL with the high volume of data, you are not going to find good options. The challenge is not a set-based challenge because you have to deal with the data sequentially in order to calculate the weekly levels. You also have to identify missing weeks where there were no changes in stock values.

Here is where an SSIS Script Component can be very effective. Because of the pipeline engine and the efficient use of memory and processor, you can achieve the above solution relatively easily.

Figure 3-79 shows a data flow with a source adapter to a flat file where the data is sourced, followed by a script transformation and a destination. It also highlights two data viewers with the source data and script output to illustrate what the script is doing.



**Figure 3-79:** Script Component results

The Script Component is configured to create a new output with the four columns shown (this is called an asynchronous script). Each row of the script compares the previous row’s product and week to determine whether the stock level needs to be reset or to add records for the missing weeks. Here’s the code used in the component:

[Microsoft.SqlServer.Dts.Pipeline.SSISScriptComponentEntryPointAttribute]

public class ScriptMain : UserComponent

{

private String myLastProduct = "";

private int myLastWeek = 0;

private int myStockLevelCalc;

public override void Input0\_ProcessInputRow(Input0Buffer Row)

{

/\*check to see if the product is the same but there has been a skipped week\*/

if (Row.Product == myLastProduct & Row.Week > myLastWeek + 1)

{

while (Row.Week > myLastWeek + 1)

{

myLastWeek = myLastWeek + 1;

/\*perform the insert for the skipped week\*/

StockLevelOutputBuffer.AddRow();

StockLevelOutputBuffer.ProductOutput = Row.Product;

StockLevelOutputBuffer.WeekOutput = myLastWeek;

StockLevelOutputBuffer.StockDifferenceOutput = 0;

StockLevelOutputBuffer.WeekStockLevelOutput = myStockLevelCalc;

}

}

/\*check for a existing product and update stock level\*/

if (Row.Product == myLastProduct)

{

myStockLevelCalc = myStockLevelCalc + Row.StockDifference;

}

/\*update the stock level for a new product\*/

else

{

myStockLevelCalc = Row.StockDifference;

}

/\*perform the insert for the existing week\*/

StockLevelOutputBuffer.AddRow();

StockLevelOutputBuffer.ProductOutput = Row.Product;

StockLevelOutputBuffer.WeekOutput = Row.Week;

StockLevelOutputBuffer.StockDifferenceOutput = Row.StockDifference;

StockLevelOutputBuffer.WeekStockLevelOutput = myStockLevelCalc;

/\*update the private variables for the next row\*/

myLastProduct = Row.Product;

myLastWeek = Row.Week;

}

}

Other examples of leveraging the SSIS Script Component include dealing with complicated pivoting scenarios or challenging data cleansing logic where you want to leverage the full function list in C# or Visual Basic.

It should be noted here that scripts transforms cannot be shared across data flows, i.e. one script transformation cannot be referenced by multiple data flows. In these cases, the SSIS team should consider keeping the master copy of the script transformation within source code version control. Any changes in this master script transformation would then need to be propogated to all instances within data flows.

### Destination Optimization (Efficient Inserts)

Much of SSIS ETL performance for data warehouse loads revolves around the interaction with the database layer, specifically around large volumes of data inserts. If you are used to dealing with ETL operations with hundreds of thousands or millions of rows (or more!), you can attest to the fact that the biggest performance gains come when you optimize table loading.

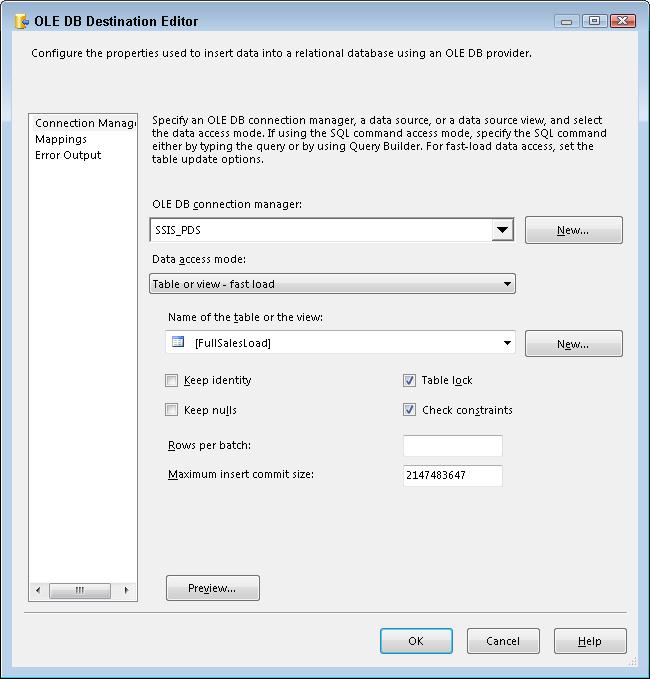
The two primary ways to optimize SSIS data loading are:

* Using bulk load settings
* Managing indexes

First, bulk table loading means inserting more than one record at a time into a table. SSIS supports both bulk load (called fast load) and the standard load. The standard data insert will write one row at a time into the table, any triggers will fire, and indexes will be updated on a row-by-row basis. When tracing the activity, you will see each row in the trace log.

You can configure SSIS to insert data in bulk though the destination adapters in the data flow. Figure 3-80 shows the OLE DB destination adapter with the data access mode set to “fast load.” This will allow thousands of records to be inserted into the table at one time. (Note that the number of records inserted is dependent on the number of columns and data types in the data flow as well as the *Rows per batch* setting; typically the OLE DB bulk load will insert about 10,000 rows at a time.)

Note that for the most efficient inserts, the recovery mode for the destination database should not be set to full mode, instead it should be set to simple or bulk logged. The implication is that a SSIS bulk load can’t be rolled back within a transaction. However, this trade off is often deemed acceptable in order to maximize performance. Refer to the Backing out batches section above for techniques that are used to roll back bulk operations.



**Figure 3-80:** OLE DB destination bulk settings

Even with fast load turned on, you may still have a bottleneck with your destination inserts. This is because any indexes on the table need to be updated with the new data that is added. When you are loading millions of rows into a table, the process of committing the rows to the table requires that the indexes are updated—a process that can take as long if not longer than it takes to insert the rows.

The most common way to deal with indexes in large tables that require large bulk inserts is to drop the indexes before loading the data and then re-create the indexes afterward. This may not sound intuitive, but it more often than not is faster than allowing the engine to reorganize the index when new data is added in bulk.

Note that re-building indexes is many times an expensive operation that results in parallel query plans. Using the MAXDOP to restrict the amount of parallel activity for index re-builds may help reduce bottlenecks on you I/O sub-system.

The following list shows the SSIS task flow for dropping and re-creating the indexes:

1. SQL Execute Task with a DROP INDEX statement.
2. Data flow task that runs the primary data loading logic with fast load.
3. SQL Execute Task with a CREATE INDEX statement.

However, if you have a large table but are inserting only a few thousand or a few hundred thousand records into the table per ETL cycle, it may be faster to leave the indexes in place. In many (but not all) cases, large tables in a data warehouse have fewer indexes, and therefore you are not rebuilding a lot of indexes on a table at once.

If you have a data warehouse table with a clustered indexed on it, you may want to keep the clustered index if the inserted data will be appended to the end of the table. For example, if your fact table is clustered on a date key and new records have the latest date key, then there is no need to drop the clustered index.

### Partition Management

Large tables (in the hundreds of millions, billions, or trillions of rows) are often partitioned to help manage physical data management and indexes. SQL Server tables support physical partitioning, where the table is made up of separate physical structures tied together into a single query-able table.

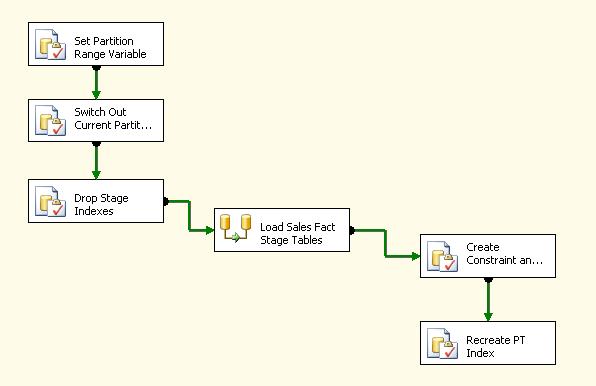
A partitioned table acts like any other table in that you can query the table and insert or update the records within the table. However, just as with index management and large data, when you are inserting data into a partitioned table, there is some overhead for the engine to determine which partition each row should be inserted into.

In addition, if the partitions have indexes on them, even more time is required for data inserting because you cannot drop an index on a partition that is part of a partitioned table.

The alternative solution for inserting into a partitioned table involves:

1. Switching out the most current partition that most of the data will be added to. This is a metadata operation with little overhead using the T-SQL SWITCH OUT command
2. Dropping some or all of the indexes on the table that has been removed from the partitioned table.
3. Inserting new warehouse data into the table using the fast load settings.
4. Recreating the indexes on the table.
5. Adding the table back to the partitioned table using the SWITCH IN command.
6. Possibly re-creating indexes that have been applied to the entire partition table (and cross partitions) if necessary.

Figure 3-81 shows an SSIS control flow of what this operation might look like.



**Figure 3-81:** ETL partition table management

The following link provides more information about table partitioning and partition management:

* [We Loaded 1TB in 30 Minutes with SSIS, and So Can You](http://msdn.microsoft.com/en-us/library/dd537533(SQL.100).aspx).
* [Designing and Tuning for Performance your SSIS packages in the Enterprise (SQL Video Series)](http://sqlcat.com/presentations/archive/2009/05/02/designing-and-tuning-for-performance-your-ssis-packages-in-the-enterprise-sql-video-series.aspx)

### SSIS Scale and Performance

The two worse offenders for ETL performance and scalability are:

* Poor ETL design patterns
* I/O bottlenecks

If you’ve worked with data warehouses for a few years, you’ve probably run into poor ETL design. Here’s a common very-bad-design scenario:

*A well-intentioned DBA or developer is tasked with transforming some data for a report or a systems integration project. A common approach is for the DBA to build a two-step process that involves three tables: two for the source data and a final one for the merged data. After review, the system or report owner reports that some information is missing and the data is not exactly right. By now, the developer is working on a new project, and the quickest way to get the user the results is to write a new step or two. Five more tables and four serialized T-SQL steps with updates later, the process is done, right? Wrong. Another business unit wants similar data. One department has found the tables and has started to use one of the intermediate tables. Before the developer knows it, the processing takes four hours and has become “mission critical.” It’s mission critical because he’s now getting calls about it at 7am when the process breaks. Even worse, it has a nickname: “The John Doe Process.”*

Recognize the scenario? The important thing to realize is that the issues didn’t develop overnight, and someone wasn’t sitting in a back room trying to cause problems. Here are the critical path issues that led to the poor design:

* **Data Stewardship: Planning** – First of all, planning didn’t happen. The business needed a solution, and the quickest one that came to mind was put in place.
* **Data Stewardship: Architecture** – Second, an architectural strategy was not thought through either because a strategy did not exist in the organization or because the developer was not stepping back and thinking about how the solution would support change or growth. The default approach was implemented because it was the simplest and fastest to develop.
* **Future Considerations** – The path of least resistance was taken at every step, which caused the solution to grow into a complicated and overburdened process. In the end, more money was spent supporting the process (with people and hardware) than it would have cost to spend the extra time up-front to ensure an effective solution.

The resulting bad design suffered from some common data challenges:

* **Serial processes** – Each part of the process was an all-or-nothing step, and each subsequent step required that the prior step complete. This approach causes delays and risk when a failure occurs.
* **High I/O** – Because the process was heavy with working tables, the I/O was a hidden bottleneck. Every time data is written to a working table, it requires persisting that data to the physical drive. And each time that happens, data is read from disk and inserted into a new table, which doubles the I/O. The differences between read and write I/O also make the I/O processes even more inefficient.
* **Dependency issues** – When an interim step was intercepted by another process, this added to the dependency chain of the system. These dependency chains can easily get out of control, which in the end will cause more complication and less agility in handling changes.

*Scaling SSIS ETL begins with the right design.* The inverse of the bullets above will give you the general guidance you need to think through SSIS design.

When you are planning your ETL process, you should be considerate of enterprise processes. Always look to get your data from the source of record or the identified enterprise standard for source data. Do not create duplication of code when you can avoid it. Leverage the SSIS data flow where you can take advantage of its features and processes. Use standards when naming and designing similar processes.

In summary, the best practices covered throughout this chapter give you a good place to start. Remember that these principles are here to guide you to the best solution, but there may be more than one right solution. One thing is for sure, there will always be more than one bad solution.

### Source Control

SSIS packages are code and should be placed under source control. Many SSIS development shops use Team Foundation Server (TFS). Note that TFS does show the differences in two SSIS packages side by side by showing the changes in the underlying XML. This sometimes is hard to follow and add ons like BIDS Helper provides a more filtered version of the differences. See the following link for more information on BIDS Helper’s features including Smart Diff: <http://www.mssqltips.com/tip.asp?tip=1892> .

## Conclusion and Resources

Data integration is critical to the success of any Data warehouse and typically represents the largest cost both for the initial development and the ongoing maintenance.

Loading large volumes of data within shrinking execution windows requires ETL developers to use industry best practices and patterns as well as best practices for SQL Server and SSIS.

More importantly, business consumers must trust the data loaded into the Data warehouse. This requires the elevation of Data quality to a first class citizen throughout the Data integration life cycle including:

* Profiling source data
* Handling and reporting data exceptions within the integration code
* Adding Data and Execution lineage throughout the integration data flow
* Creating reports that Data Stewards can use to reconcile results and identify the root cause of data exceptions

The creation of an ETL Framework and ETL template packages allow ETL developers to create consistent scalable solutions in less time. In addition, these templates reduce development maintenance costs over the lifetime of the ETL solution. ETL Framework dynamic configurations and logging make ETL operations resources more efficient and reduce the amount of resources required for ongoing ETL operations.

Finally, building your Data warehouse on the SQL Server product stack reduces overall software acquisition costs as well as the training costs for the Data warehouse team.

### Resources

This section contains links mentioned in this chapter along with other useful links on SSIS.

SSIS Sites / Blogs:

* [SQLCAT Team’s Integration Services best practices](http://sqlcat.com/top10lists/archive/2008/10/01/top-10-sql-server-integration-services-best-practices.aspx)
* [SSIS Team blog](http://blogs.msdn.com/b/mattm/)

Additional information for SSIS:

* [SSIS System Variables](http://msdn.microsoft.com/en-us/library/ms141788.aspx)
* [SSIS Service](http://msdn.microsoft.com/en-us/library/ms137731.aspx)
* [Integration Services Error and Message Reference](http://msdn.microsoft.com/en-us/library/ms345164.aspx). This is useful for translating numeric SSIS error codes into their associated error messages.
* sysssislog (2008); you will need to change the SQL in all of the reports or create a view if running SSIS 2008 or later.
* [Integration Services Error and Message Reference](http://msdn.microsoft.com/en-us/library/ms345164.aspx). This is useful for translating numeric SSIS error codes into their associated error messages.
* [Working with Parameters and Return Codes in the Execute SQL Task](http://technet.microsoft.com/en-us/library/cc280502.aspx)
* [SSIS Nugget: Setting expressions](http://consultingblogs.emc.com/jamiethomson/archive/2006/03/11/SSIS-Nugget_3A00_-Setting-expressions.aspx)

SSIS Performance

* [SSIS 2008 Data flow improvements](file:///C:\gigs\SQM\eDW%20Toolkit\Chapter3\•%09http:\blogs.msdn.com\b\michen\archive\2007\06\11\katmai-ssis-data-flow-improvements.aspx)
* [SSIS Performance Design Patterns video](http://blogs.msdn.com/b/mattm/archive/2010/06/29/ssis-performance-design-patterns-video.aspx)

Partitioned Tables:

* <http://sqlcat.com/msdnmirror/archive/2010/03/03/enabling-partition-level-locking-in-sql-server-2008.aspx>
* <http://blogs.msdn.com/b/sqlprogrammability/archive/2009/04/10/sql-server-2005-2008-table-partitioning-important-things-to-consider-when-switching-out-partitions.aspx>
* [We Loaded 1TB in 30 Minutes with SSIS, and So Can You](http://msdn.microsoft.com/en-us/library/dd537533(SQL.100).aspx).
* [Designing and Tuning for Performance your SSIS packages in the Enterprise (SQL Video Series)](http://sqlcat.com/presentations/archive/2009/05/02/designing-and-tuning-for-performance-your-ssis-packages-in-the-enterprise-sql-video-series.aspx)

Configuration and Deployment:

* [SQL Server Integration Services SSIS Package Configuration](http://www.mssqltips.com/tip.asp?tip=1405)
* [SSIS Parent package configurations. Yay or nay?](http://consultingblogs.emc.com/jamiethomson/archive/2008/08/29/ssis-parent-package-configurations-yay-or-nay.aspx)
* [SSIS - Configurations, Expressions and Constraints](http://ewisdahl.spaces.live.com/blog/cns!23AC9944C8FA112A!419.entry?sa=708573673)
* [Creating packages in code - Package Configurations](http://www.sqlis.com/post/Creating-packages-in-code-Package-Configurations.aspx)
* [Microsoft SQL Server 2008 Integration Services Unleashed (Kirk Haselden)](http://www.amazon.com/Microsoft-Server-Integration-Services-Unleashed/dp/0672330326#_), Chapter 24 – Configuring and Deploying Solutions
* [SQL Server Integration Services SSIS Package Configuration](http://www.mssqltips.com/tip.asp?tip=1405)
* [Simple Steps to Creating SSIS Package Configuration File](http://www.sqlservercentral.com/articles/Integration+Services+(SSIS)/66500/)
* [Reusing Connections with Data Sources and Configurations](http://msdn.microsoft.com/en-us/library/cc671619.aspx)
* [Managing and Deploying SQL Server Integration Services](http://technet.microsoft.com/en-us/library/cc966389.aspx)

SSIS Tools and Add ins:

* [BIDs Helper – This is a very useful add-in that includes an Expression highlighter](http://bidshelper.codeplex.com/wikipage?title=Expression%20and%20Configuration%20Highlighter&referringTitle=Home)
* [BIDs Helper Smart Diff](http://www.mssqltips.com/tip.asp?tip=1892)
* [ETL Framework](http://etlframework.codeplex.com/)
* [SQL Server 2005 Report Packs](http://www.microsoft.com/downloads/details.aspx?familyid=D81722CE-408C-4FB6-A429-2A7ECD62F674&displaylang=en.). This page has a link to the SQL Server 2005 Integration Services Log Reports. Note: The SSIS logging table has changed from sysdtslog90 (2005) to sysssislog (2008); you will need to change the SQL in all of the reports or create a view if running SSIS 2008 or later.