Allianz Global

1. Introduction

The Staging Layer covers the first major series of ETL process steps within the Data Warehouse reference architecture. The processes involved with the Staging Layer introduce data from many (often disparate) source applications into the Data Warehouse environment. In this sense the Staging Layer is for the most part literally a place where the data is collected onto the Data Warehouse environment before being integrated in the core Data Warehouse or loaded for other use-cases (i.e. analytics, ad-hoc reporting).

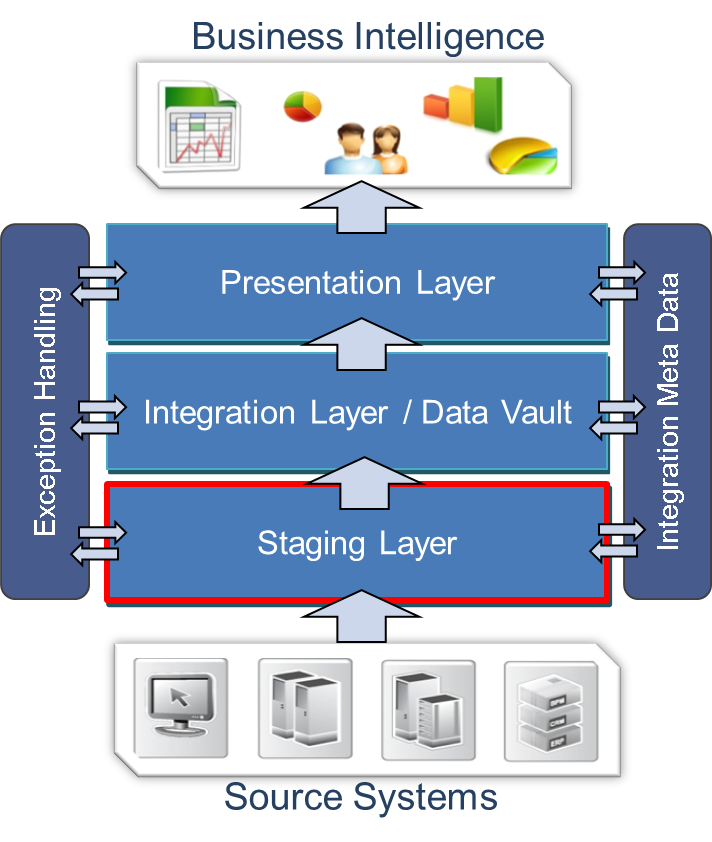
But even then many fundamental decisions are required that have repercussions throughout the rest of the design. This document defines the Staging Layer and describes the required process steps and available solutions.

Many references to other parts of the ETL Framework will be made including error handling and metadata management. Ultimately, this document provides the information to configure the Staging Layer in a flexible and modular way.

Every project or delivery has different constraints and opportunities, which is why the detailed definition of how the Staging Layer is implemented (and why) is captured into the Solution Architecture document.

1. Staging Layer overview
   1. Staging Layer overview

The position of the Staging Layer in the overall architecture is outlined in the following diagram.



The Staging Layer, or the process from source to staging, consists of two separate parts (areas):

* The Staging Area, and
* The Persistent Staging Area

(PSA, or History Area)

The Integration Metadata Model is generic, and is documented in detail in the ETL Process Control and Metadata document.

Operational metadata provides information relating to the data loading processes.

The ETL associated with the *Staging Area* copies data from different source systems to the Data Warehouse environment for further processing. In other words: the Staging Layer is responsible for the physical movement of data delta (differentials) from the source platform onto the Data Warehouse platform.

During these processes the content of the data is not changed, however changes to the format of the data types are done to conform to a standardised limited set. The Staging Area is non-persistent and therefore is emptied (truncated) every time the table is loaded with new data delta during an ETL process (truncate/insert mechanism). The important design decisions for the Staging Area are to determine the correct event date/time for each (type of) source interface and to ensure that information is interfaced correctly (i.e. delta calculation, load date/time stamp setting, order of processing etc.).

A second area can be added to the Staging Layer to archive data in the source structure: the *Persistent Staging Area* (History Area). This will preserve the (table) structure of the source applications while at the same time storing the data in a historical fashion. This allows for a number of functions including positioning as ODS and Disaster Recovery. In worst-case scenarios, where modelling decisions have been made which cannot be reverted, the History Area can be used to repopulate parts of the Data Warehouse in the updated model using the re-initialisation process.

The Staging Area and Persistent Staging Area can be used independently, or in combination with each other:

* Staging Area only – the traditional Data Warehouse configuration. The Staging Area acts as a transient environment to make sure data can be collected in batch-type environments and pushed into the Data Warehouse.
* Persistent Staging Area only – akin to a Data Lake configuration. Data delta is loaded directly in the PSA, from which it is used for further processing.
* Both Staging Area and Persistent Staging Area. A configuration intended to deprecate the PSA once the solution is mature, mainly focused on Data Warehouse solutions where the PSA is used as refactoring mechanism during development.

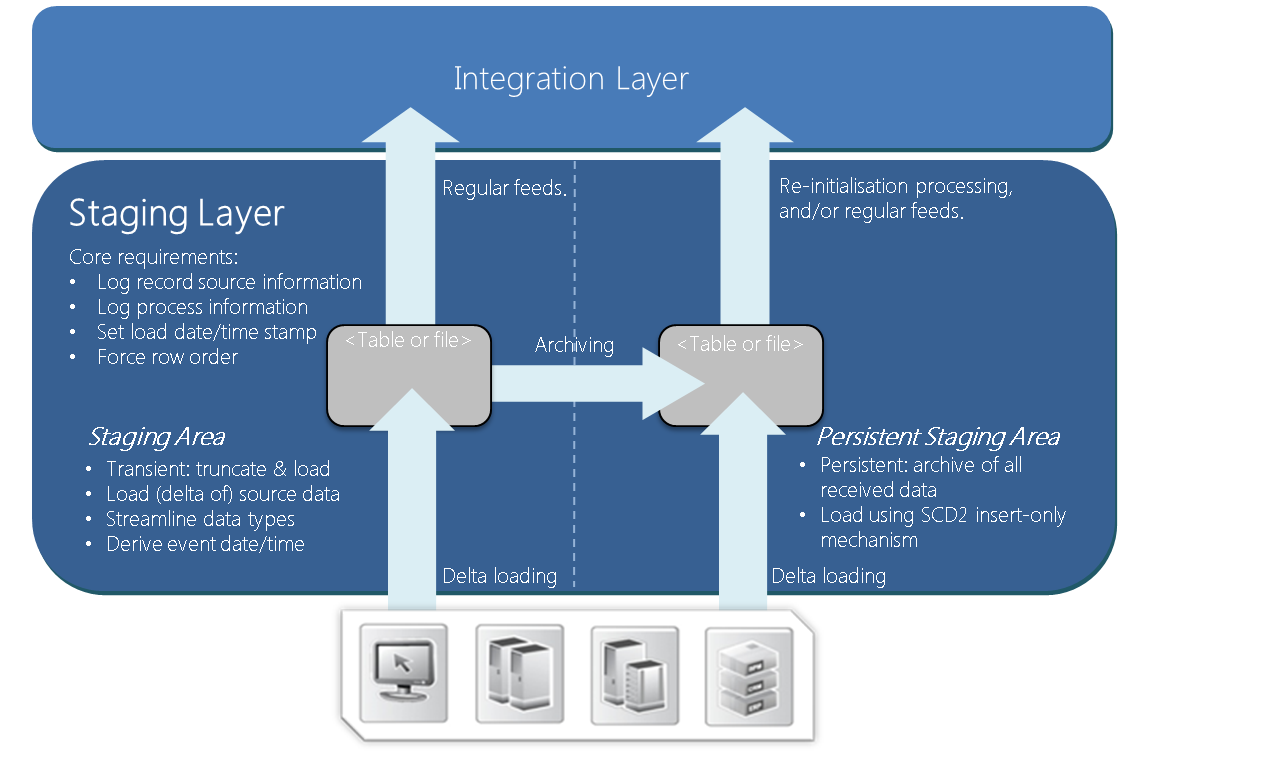
The desired specific configuration of the above is logged in the Solution Architecture documentation. It is important to state that both the Staging Area as the Persistent Staging Area do not necessarily need to be database tables, they can be any fit-for-purpose technique.

Please note: it is technically possible to directly write data delta into the core Data Warehouse layer (Integration Layer), although this is only relevant for very specific use-cases. In most situations including a PSA will deliver maximum flexibility.

* 1. Detailed Staging Layer overview
     1. Core functionality

The reference architecture specifies that data from the source systems is loaded into the Staging Layer. Both areas with the Staging Layer have a similar structure (the same structure as the source system that provides the data). The Persistent Staging Area has additional attributes to support the storage of data over time.

This approach is presented in the following diagram:



In typical solution designs, the Staging Area ETL is contains the steps where the source data is actually copied into the Data Warehouse environment. Concepts such as Change Data Capture (CDC) are implemented for this purpose.

In some cases the Staging Area process may be further broken down into two more steps, for example when a delta needs to be derived by the Data Warehouse itself (full outer joins) or when a separate landing area for flat files is required. The end result in every case is the availability of a data delta in the primary Staging Area table.

In the specific case of flat file handling the data from the flat file in loaded into a table schema where the only data type is character (for example VARCHAR, NVARCHAR2 or TEXT). In doing so, this loading process should never fail as a result of data-related issues. Data conversion to proper data types is handled in later steps, typically a part of the Integration Layer ETL.

The optional processing for the Persistent Staging Area (PSA) creates a historical view of the source data. All source data is captured as-is and the data is stored in a SCD2 insert-only fashion. While this means extra ETL development there are many benefits:

* Creating of an archive of source data in an easy to manage way
* Provide the possibility of remodelling or reloading (parts of) the Data Warehouse in case of a catastrophe or design error. It provides reloading capability for the Integration Layer
* The loading process to the Integration Layer will be easier since selections on date time values can be made
* Track and audit history in source format
* What-If analysis based on source data
* A generic foundation for the Data Warehouse can be deployed early in the project lifecycle where changes are already captured before the full Data Warehouse design has been completed

The PSA provides these benefits at the cost of extra disk space, ETL development and maintenance. However, a PSA does not necessarily need to be configured as a database; a scalable file storage environment (HDFS, Azure Data Lake or Amazon S3) can be adopted as well.

Due to the generic nature of this design depending on the ETL software used these ETL processes and table structures can be created using development patterns / automation.

* + 1. Implementing Change Data Capture

The way data is loaded into the staging tables depends very much on the source system, company guidelines and general availability. Different ways of approaching change data capture and acquiring data in general are:

* Implement a Change Data Capture (CDC) mechanism and/or replication mechanism on the source system
* Obtain a periodical delta, either through a flat file/CSV dump, by querying the entire source table through a database connection or by having source systems load data into the Staging Area directly.
* Subscribe to an Enterprise Service Bus / messaging / custom XML canonical
* Obtain a full load every run to compare against the previous full load. The delta; new or deleted records can be derived from these two data sets. If full loads are received the delta itself has to be derived before further processing can continue. After the detection of the delta the usual staging area processes can be implemented

The preference would be to use an existing interface since it bypasses a number of discussions regarding definitions of the data used in the Data Warehouse. Additionally this also provides automatic auditing capabilities. The comparison between current and previous full loads is the least attractive option since this will not be possible when dealing with large data sets. The best practice approach is to let the source systems provide the delta automatically. This provides a clear split between responsibilities and reduces a significant amount of complexity. These responsibilities are ideally documented in an interface agreement which should contain not only structure, frequency and uptime but also communication plans in case of changes and disaster recovery.

Enterprise Service Bus (ESB) solutions (including canonical definitions) are often regarded as the most advanced solution for data exchange. If possible, this is an ideal source for Data Warehouses because the ESB concept significantly reduces data issues and consistency while at the same time providing more accurate and traceable timing information about data. Depending on the type of message, ESBs can be a potential bottleneck for large volumes of data. For instance XML messages are very verbose, whereas SWIFT messages are not.

For this reason care has to be taken to estimate the ESB bandwidth for large delta volumes. In most cases ETL is regarded as the data transfer mechanism for large sets of data (typically batch oriented) while ESBs are used to exchange smaller bits of information (messages) in near real time.

* + 1. Handling Logical Deletes

When defining the connection between a source and the staging area an attempt should be made to capture logical deletes. This requires the source to have implemented a change data capture mechanism, or this can be derived from comparing full sets of data.

Either way, logical deletes should be added as a ‘CDC Operation’ attribute. If the Data Warehouse does not capture logical deletes and the source system can physically delete records inconsistencies will appear without any option to reconcile and audit these differences.

* + 1. Transforming / streamlining source data types

By their nature, source systems in an Enterprise Data Warehouse can consist of a great variety of different methods and technologies which lead to different data types. To make processing later on easier these different data types are mapped to a minimal set of values. This process is done without losing information which means no rounding of precisions / scales or limiting string lengths. Another major reason for this approach is to provide some protection from changes done in source systems. For instance, the Data Warehouse is not impacted if a text field is made bigger in the source system.

The formatting of data types is done in the following way:

* Any text attributes smaller or equal than 100 positions will be mapped to VARCHAR(100) or similar
* Any text attributes higher than 100 but smaller or equal than 1000 will be mapped to VARCHAR (1000) or similar
* The rest of the text attributes will be mapped to VARCHAR(4000)
* Dates, times, date times will be mapped to a high precision date/time (DATE, DATETIME2)
* All decimals or numeric values including Bits, Booleans and Floating Points will be mapped to a high precision numeric (NUMBER, NUMERIC(38,20))
  1. Staging Layer relationships with error handling

The default design decision is to push any business logic or exception handling to the Integration Layer.

This means that by default there is no explicit implementation of error handling in the Staging Layer since the overall design limits the occurrences of issues to a minimum. This expresses itself in various ways including the design to make source system attributes NULL-able. For this reason errors in the Staging Area are typically related to the technical environment (network outage, disk space etc.).

In this case the ETL process can simply be re-run as all ETL is designed to be run at any time, multiple times in sequence and able to process multiple delta sets in one go.

Depending on existing on-site conventions in some cases it may be required to convert data types for flat-files while loading the information to the Staging Layer. In this scenario errors related to data type conversion can occur since flat files do not provide protection against incorrect data entry.

For example, an attribute which is defined as date/time may contain a value of the 30th of February due to incorrect data entry. Processing will fail because of the domain restrictions on date/time fields. In this scenario the errors have to be explicitly captured and re-processed.

This is not the default and preferred solution. But to support this approach if required and only in this scenario for each a corresponding error table can be defined. This table will have the same schema structure and the following additional columns:

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Reasoning** |
| ERROR\_BITMAP | INTEGER | The bitmap containing multiple errors. |
| ERROR\_DESCRIPTION | VARCHAR(100) | A description of the error. |
| ERROR\_RECYCLE\_COUNT | INTEGER | The number of times the record has been processed and rejected to date. |
| ERROR\_STATUS | VARCHAR(100) | OMD error status code (for instance ‘F’ for failure’). |

As mentioned above the default error handling strategy is to continue loading and to prevent error handling to hamper the data flow. Loading data into the Integration Layer provides flexibility to choose whether or not to select data based on completeness for loading to the information marts. In this sense rejecting records because of an incorrect data type conversion does not stop the process.

Data conversion errors may also mean that the technical specification for the corresponding source system has been changed. And it is likely that this change has taken place without notice to the Data Integration team (but should be captured in an Interface Agreement). In this case processing will fail and manual intervention from support is necessary. If error tables have been implemented the special care has to be taken when errors are solved in the source system. Since the record still exists in the recycling table the new one would be removed in favour of the recycle record.

There are a number of choices that can be made on this topic, depending on the requirements:

* A partial load of source data (a result of rejected records) may have an impact downstream in the Presentation Layer (for instance inaccurate summary figures). Discretion has to be applied when deciding if the entire load process (current load and all subsequent loads) should be terminated when encountering rejected records
* In situations where accuracy in the Presentation Layer is not vital (for instance a CRM data mart), loads with a tolerable threshold on the number of rejected records may allowed to continue
* Where there is a mix of data marts where data completeness and accuracy is vital to some of the data marts and not others (Finance data mart versus CRM data mart), strategies can be put in place to permit partial load while minimizing its impact on all data marts. A possible approach is to filter out partial loads (based on an OMD ID such as BATCH\_ID) as well as all subsequent loads when populating data marts that require data to be complete and accurate

The complete error handling for each layer in the architecture is documented in detail in the error handling and recycling approach.

1. The Staging Area

This section describes both the concepts and structure of the Staging Area in detail.

* 1. Staging Area table structure

The Staging Area is modelled after the structure of the application that supplies the data. This excludes all constraints or referential integrity the source tables might have; the Data Warehouse assumes the source systems handle this, and otherwise this will be handled by the Integration / Presentation Layer processes of the Data Warehouse.

The structure of the Staging Area therefore is the same as the source table, but always containing the following process attributes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | **Required / Optional** | **Data Type / constraint** | **Reasoning** | **DIRECT / OMD equivalent** |
| Load Date/Time Stamp | Required | High precision date/time – not null | The date/time that the record has been presented to the Data Warehouse environment. | OMD Insert Datetime |
| Event Date/Time | Required | High precision date/time – not null | The date/time the change occurred in the source system. | OMD Event Datetime |
| Source System ID / Code | Required | Varchar(100) – not null | The code or ID for the source system that supplied the data. | OMD Record Source |
| Source Row ID | Required | Integer – not null | Audit attribute that captures the row order within the data delta as provided by a unique ETL execution. The combination of the unique execution instance and the row ID will always relate back to a single record. Also used to distinguish order if the effective date/time itself is not unique for a given key (due to fast-changing data). | OMD Source Row ID |
| CDC Operation | Required | Varchar(100) – not null | Information derived or received by the ETL process to derive logical deletes. | OMD CDC Operation |
| Full row hash | Optional | Character(32), when using MD5 – not null | Using a checksum for record comparison requires storing a checksum value as an attribute. Can be made optional if column-by-column comparison is implemented instead. | OMD Hash Full Record |
| ETL Process Execution ID | Required | Integer – not null | Logging which unique ETL process has inserted the record. | OMD Insert Module Instance ID |
| Upstream Hash Values | Optional | Character(32), when using MD5 – not null | Any pre-calculated hash values one may like to add to optimize upstream parallel loading (i.e. pre-hashing business keys), | OMD Hash <> |
| <source attributes> | Required | According to data type conversion table - nullable | The source attributes as available. Note that if a primary hash key is not used the natural key (source primary key) needs to be set to NOT NULL. All other attributes are nullable. | N/A |

* 1. Staging Area development guidelines

The following is a list of conventions for the Staging Area:

* When loading data from the source, always load the lowest level (grain) of data available. If a summary is required and even available as a source, load the detail data anyway.
* If the ETL platform allows it, prefix the ‘area’, ‘folder’ or ‘namespace’ in the ETL platform with ‘100\_’ because this is the first Layer in the architecture. Source definition folders, if applicable, are labelled ‘000\_’. This forces most ETL tools to sort the folders in the way the architecture handles the data.
* Source to Staging Area ETL processes use the truncate/insert load strategy. When delta detection is handled by the DWH (i.e. using a Full Outer Join) a Landing Area table can be incorporated.
* Everything is copied as-is, no transformations are done other than formatting data types. The Staging Area processing may never lead to errors!

1. The Persistent Staging

The structure of the PSA is the same as the Staging Area (including the metadata attributes). The following attributes are mandatory for the PSA tables:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | **Required / Optional** | **Data Type / constraint** | **Reasoning** | **DIRECT / OMD equivalent** |
| Primary hash key i.e. <entity>\_SK | Optional | Character(32), when using MD5 – not null | The hashed value of the source (natural) key. Part of the primary key which is issued for each record in the history table. Can be used instead of composite primary key. | N/A |
| Effective Date/Time | Required | High precision date/time – not null | The date/time that the record has been presented to the Data Warehouse environment. If a Staging Area is used these values will be inherited. | OMD Insert Datetime |
| Event Date/Time | Required | High precision date/time – not null | The date/time the change occurred in the source system. If a Staging Area is used these values will be inherited. | OMD Event Datetime |
| Source System ID / Code | Required | Varchar(100) – not null | The code or ID for the source system that supplied the data. | OMD Record Source |
| Source Row ID | Required | Integer – not null | Audit attribute that captures the row order within the data delta as provided by a unique ETL execution. The combination of the unique execution instance and the row ID will always relate back to a single record. Also used to distinguish order if the effective date/time itself is not unique for a given key (due to fast-changing data). If a Staging Area is used these values will be inherited. | OMD Source Row ID |
| CDC Operation | Required | Varchar(100) – not null | Information derived or received by the ETL process to derive logical deletes. If a Staging Area is used these values will be inherited. | OMD CDC Operation |
| Full row hash | Optional | Character(32), when using MD5 – not null | Using a checksum for record comparison requires storing a checksum value as an attribute. Can be made optional if column-by-column comparison is implemented instead. If a Staging Area is used these values will be inherited. | OMD Hash Full Record |
| ETL Process Execution ID | Required | Integer – not null | Logging which unique ETL process has inserted the record. | OMD Insert Module Instance ID |
| Current Row Indicator | Optional | Varchar(100) – not null | A flag or Boolean to indicate if the record is the most current one (in relation of the effective date). | OMD Current Record Indicator |
| Change Date/Time | Optional | High precision date/time – nullable | A derived date/time field to standardize the main business effective date/time for more harmonised upstream processing. | OMD Change Datetime |
| <source attributes> | Required | According to data type conversion table - nullable | The source attributes as available. Note that if a primary hash key is not used the natural key (source primary key) needs to be set to NOT NULL. All other attributes are nullable. | N/A |

The ETL process from the Staging Area to the PSA checks the data based on the source key and the date/time information and compares all the attribute values. This can result in the following actions:

* No history record is found: insert a new record in the history table.
* A record is found and the source attribute values are different from the history attribute values: insert a new record. Update current row indicators if adopted.
* A record is found but there are no changes found in the attribute comparison: ignore.

Note: there are other suitable approaches towards a PSA. Depending on the requirements there can also be opted for a snapshot PSA where every run inserts a new record (instance) of the source data. This creates a more redundant dataset but arguably makes reloading data to the Integration Layer easier.

When loading data delta directly into the PSA (i.e. the Staging Area is not adopted) the same rules apply as for the Staging Area.

* 1. Persistent Staging Area development guidelines

The following is a list of development conventions for the Persistent Staging Area (PSA):

* When loading the PSA from the Staging Area, always start a PSA ETL process as soon as the Staging Area is finished to ensure that there are no ‘gaps’ in the history. Since the Staging Area has the ‘truncate/insert’ load strategy, PSA data has to be processed before the next Staging Area run. During normal loads, the Integration Area has no dependency on the History Area and loading into history and integration can be done in parallel if the Data Warehouse has capacity for concurrent jobs. This is handled by the ‘Batch’ concept which guarantees the unit of work; e.g. making sure all data delta has been processed
* If the ETL platform supports it, prefix the ‘schema’, ‘area’ or ‘folder’ in the RDBMS and ETL software with ‘150\_’. The PSA is part of the first layer in the architecture, but is updated after the Staging Area (if adopted). This forces most ETL tools to sort the folders in the way the architecture handles the data
* Everything is copied as-is, no transformations are done.
* There is no requirement for error handling in PSA ETL.
* When using files, changes in file formats (schema) over time can be handled in separate file mask metadata versions.