Agspring ETL Developers Guide

**Overview**

The Agspring ETL environment exist in a well-defined framework. That framework will be spelled out in this document.

The environment is designed to accept data from many sources and combine that data into a single dataflow that terminates in the Agspring data warehouse. The data moves through in stages. Each stage will be described below.

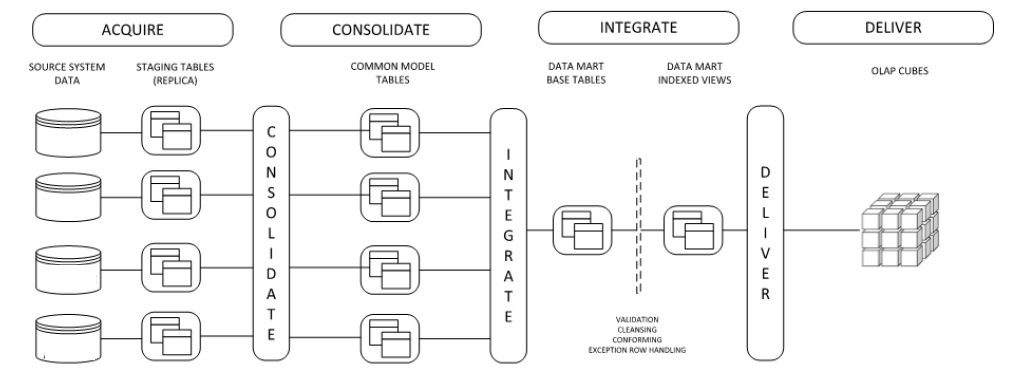


Figure 1. Warehouse load process.

Figure 1 represents a generic load process that closely resembles the current state of the ETL architecture. Each stage is only loosely coupled to the downstream stage. That is to say that if a particular piece fails, it does not fail the entire process.

Acquire

Each ETL process that pulls data into the system is entirely independent of any other process. Each dataset is pulled from its source system and placed in a staging table.

Consolidate

All data that is collected in staging tables is moved to common model tables together. The common model is a unified representation of all data across systems.

Integrate

Moving data from consolidate to integrate is accomplished in the same step as moving from acquire to consolidate. Integrate is where we move the data from the common model into the warehouse tables.

Deliver

At this time, there are no OLAP cubes in the data architecture. Data is delivered through various vectors. Most of those vectors pass through the warehouse. A few bypass the warehouse and go straight to de-normalized reporting tables.

**Relevant SQL02 Databases**

There are three databases used to deliver data to Agspring.

**BI360DW** – This is the warehouse database.

**ODS** – Operational data store. This is where all ETL functions take place. It is where staging tables and the tables that support the common model live. The stored procedures that perform ETL live here as well as any views that support monitoring of the ETL processes. This database is not accessible to Agspring business users.

**Reporting** – The database consist of de-normalized reporting tables and views that are built from tables in BI360DW. This database allows rapid access to data without having to build complex reports. Every model in production, is represented here as a de-normalized view.

**ETL Order of Processes**

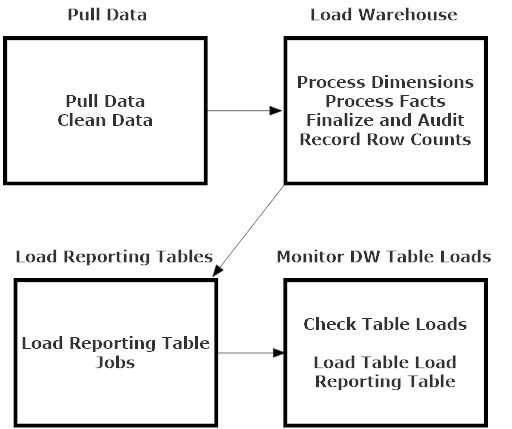


Figure 2. Actual ETL Job Sequence

Pull Data

In general, all processes that acquire data from outside systems pull and clean data in the same step. Those jobs that manage intra-day processes will contain steps to execute the entire data pipeline.

Load Warehouse

The actual warehouse load processes is a combination of loading tables and auditing processes that are used to check and make sure everything loaded ok. As you build ETL processes, you will need to make sure to add code that takes these processes into account. All dimensions are processed first; then all fact tables are processed. Fact table loads communicate back to staging to take note of what exactly was loaded to the warehouse. The Finalize and Audit processes checks staging records and reports discrepancies. Every fact table load you build will need code added to the sp\_MarkRecordsAsProcesses stored procedure. Record Row Counts is fully automated and documents how many records were loaded to each fact table and compares that to historical loads to determine if there is an anomaly in the load process.

Load Reporting Tables

This job is for those processes that load de-normalized reporting tables that reside in the Reporting database only. The reporting tables that live in ODS, and are only for data professionals, are loaded by different processes.

Monitor DW Table Loads

Check Tables Loads will report on unusual record load amounts. Either more than usual or none at all over a three day period are the trigger events. In the final step, volumetric data is dumped into a de-normalized reporting table for easy digest.

**Data Cleansing**

As with any ETL process, some of the data that comes in to the system needs to be cleaned and standardized before it is loaded. Below is a list of cleansing task that have been compiled to date.

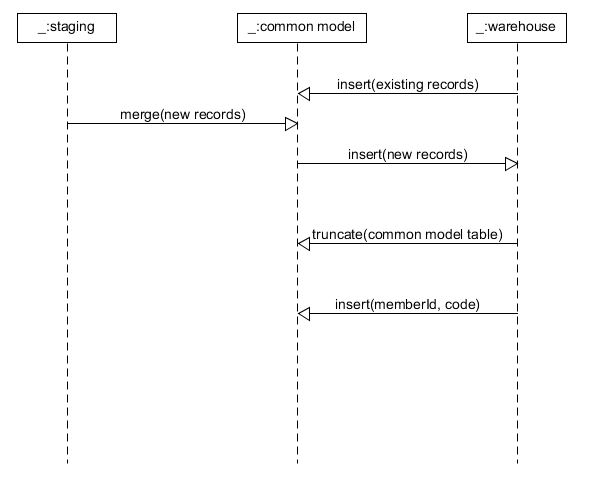
1. Translate contract types from codes to human readable terms. The function udf\_TranslateContractType helps you do this.
2. Translate unit of measure from codes to human readable terms. The function udf\_TranslateUOM helps you do this.
3. Date cleansing requires several steps. It should be turned into a function, but the function has not been written yet.
   1. Set empty dates to 20120602
   2. Set error dates to 20120601
   3. Convert remaining dates to YYYYMMDD format
   4. Set dates prior to 20120601 to 20120601
4. Numeric values coming from Smart Soft need to be cleaned up. The function dbo.udf\_CleanNumericValue lets you do this.

This represents the bare minimum that needs to be done on each dataset.

**Loading Dimensions**

Figure three shows the generalized process for loading warehouse dimensions. Currently, most of Agspring’s dimensions are Type I SCDs. Most dimension load processes follow this design pattern with a few exceptions most notably all date dimensions. Due to the unique design constraints of BI360, this load process is not as efficient as it can be. The general process is explained as follows:

1. Load the common model table from the warehouse.
2. Merge existing data with new data.
3. Determine which records are new and load those to the warehouse. This may also be a merge process if you are updating old records.
4. Truncate the common model table.
5. Load the common model table with key fields necessary to populate fact tables.



**Warehouse Load Commandments**

1. **Thou shalt not put stand-alone codes in the database.** Always import code values along with their English explanations if possible.
2. **Thou shalt not put data into generically named columns like Value1 and Value2.** Every single column should have a clear and understandable name such that the meaning of the values in that column are entirely unambiguous even without having to consult the data dictionary.
3. **Thou shalt not use 1 or 0 to represent Boolean values.** Use only Yes and No exactly as they are written here.
4. **Thou shalt not place textual filter data into fact tables.** This should be pretty self-explanatory and go without saying. However, BI360s data architecture restrictions make this short cut look attractive. **DON’T DO IT!**
5. **Thou shalt not place dates into fact tables without an appropriate connection to a date dimension.** This is a big one and you no doubt will have questions. Long story short, if you have date data, you will need to create a separate date dimension instead of just creating a view on the date dimension like you would normally. For a more complex explanation, ask a data engineer that is familiar with the restrictions of working with BI360.
6. **Thou shalt not allow flags and acronyms into the database if possible.** Values like T and B need to be translated to Truck and Barge. If you have question about a particular value, ask an Agspring SME or ask Smart Soft directly.

**Conventions**

When developing ETL processes there are a number of conventions that must be adhered to. Those conventions are largely based on common sense, but they do change depending on what you are doing and where you are doing it.

Should I Use SSIS?

Nine times out of ten, the answer to this question is going to be no. As a general rule of thumb, only use SSIS to develop ETL processes if you need to move an actual physical file or you have some reason to do something that T-SQL cannot do like interact with the file system or OS in some manner.

Most of the ETL processes move data from one database to another and linked servers have been set up in SQL02 that connect all other databases.

Conventions for SSIS

There is a standard template package. Use that as the starting point for any ETL that requires SSIS as lined out in the section above. Most of the standards are already built in to that package. As you continue to build out your process, keep the following in mind:

1. Attempt to reduce the number of hard coded values to zero in your package. Place any configurations in the configuration table.
2. Attempt to name variables following the example conventions in the package.
3. The package has two parameters currently for moving files. Be sure to use them and connect them to the Global environment.
4. Do not put non-arbitrary SQL statements in your packages. If your code is longer than three lines, turn it into a stored procedure and call that instead.
5. Script task need to be developed using C#.

Conventions for T-SQL

Every stored procedure needs to have a documentation block. The block needs to include the name of the developer, the date the script was created, a description of what the script does, and a log of the changes to the script. All T-SQL scripts need to conform to the following conventions:

1. All scripts need to be left justified.
2. All SQL reserve words need to be capitalized.

Conventions for C#

There are no published conventions specific to Agspring for the development of C# code. Attempt to conform to Microsoft standards but this isn’t a hard rule as there should not be a preponderance of C# development in the environment.

Conventions for ODS

Naming conventions in ODS appear to follow no pattern. This is because a primary rule of naming objects in this database is that they should conform to the source system. The conventions for ODS are as follows:

1. NO table in ODS should use the dbo schema. Every table needs a schema that CLEARLY identifies the function of that table. If it is a staging table, the schema needs to identify the source system of the data. Some examples:
   1. cm = common model
   2. boa = Bank of America
   3. ss = Smart Soft
2. Staging table naming conventions **FOR COLUMNS** need to conform to the naming conventions of the source system. As a matter of fact, the column names should be identical to the source system.
3. Staging table naming conventions **FOR TABLE NAMES** should follow the pattern of capitalizing every first word.
4. Common model tables should be nearly identical to the warehouse tables they mimic but stripped of all indexes, keys, and most constraints.
5. Every stage table needs to have the following columns regardless if you use them or not:
   1. Unique dims. This column is used to help determine what records actually got transferred to the warehouse.
   2. Error Record. Binary value to indicate that there is something wrong with that record.
   3. Processed. Binary record used to identify records that were actually moved to the warehouse.
   4. Run date. Timestamp of when the load was completed.

Conventions for Reporting

The reporting database does not follow any standard database naming conventions. The reporting database is mostly views exposed to users. As such, objects are named in a manner so that they are easily readable by users.

Conventions for BI360DW

Because of the restrictions on BI360, a lot of the conventions are already built into the system. The main thing is to conform to the existing pattern of names which means capitalizing the first letter of words on the column name and table label.

**Use of SQL02T**

SQL02T is the development box. It mirrors production and development should be done against this machine and tested before objects are moved to production when dealing with ODS.

BI360DW is another matter. Because of the restrictions of the system, you cannot simply script objects and deploy them to production. Objects HAVE to be created in production FIRST using data warehouse manager. The BI360DW database is replicated down to 02T every day at night. Create objects in BI360DW prod the day before you need them. Plan your work accordingly.

**Wrapping Up**

This document attempts to address the major issues involved with developing ETL against the Agspring data warehouse. Of course, we cannot address every single possible issue you might run into. Fortunately, there is a large repository of existing objects that can be used as examples and templates for future work. If you want to know how to do something, check for any existing processes that do something similar.