Blackboard
ANALYTICS™

Blackboard Intelligence

Analytics for Learn • Student Management • Finance • HR • Advancement

Technical Reference Guide

VERSION	4.2
CREATED	March 5, 2014
REVISED	July 13, 2017

Proprietary and Confidential

Table of Contents

INTRODUCTION

This document is a technical reference guide which explains the application framework, architecture, and configuration capabilities of the data warehouse tier of the Blackboard Intelligence application. The topics addressed in this document include:

1. Introduction
2. Blackboard Intelligence Application Architecture Overview
3. ETL Engine – Extract Phase
4. ETL Engine – Transform & Load Phase
5. Running the ETL
6. Data Model Metadata
7. Data Warehouse Administration – HEAdmin
8. Appendices

BACKGROUND

Blackboard Analytics recognizes that each Institution is unique and may have information needs that we have not incorporated into our standard baseline model. Blackboard Intelligence data warehouse is more than a hardwired package - it's a framework to build upon. Blackboard Intelligence data warehouse is designed to enable Institutions to extend the application in various ways, while preserving the ability to implement application upgrades that Blackboard Intelligence provides in the base product. This document provides an explanation of the major Blackboard Intelligence data warehouse architecture components, and provides an initial knowledge base for Institutions to work with the Blackboard Intelligence data warehouse.

KEY ARCHITECTURE DRIVERS

Blackboard Analytics' goal is to provide an extensible out-of-the-box application that can be easily installed and contains significant delivered content, and the framework to enable Institutions to efficiently tailor business rules and extend the application, all within a product framework with a clearly defined upgrade path. This architecture effectively supports these objectives. The sections below describe the underlying components that comprise the data warehouse tier of the application in more detail.

PREREQUISITE TECHNICAL SKILLS

The ability for Institutions to modify, customize and extend the Blackboard Intelligence data warehouse will require certain prerequisite technical skills. While a limited number of configuration settings can be defined using application web forms, the majority of tasks related to custom configuration require some depth of technical skills and knowledge.

- A comprehensive understanding of Blackboard Intelligence data warehouse architecture, including the ETL framework, ETL process, database object types and transformation components; and
- Sufficient core technical skills and expertise with the underlying SQL Server database technologies upon which the application is based.

Based on our experience with existing customers, proficiency in customizing the application will take some time and expertise; however, basic customization can be quickly achieved as long as the core technical skills are in place. We recommend that Institutions develop a specific plan to develop this proficiency that includes:

- Third-party training for core database technology skills (as required), and
- Mastering basic customization of business rules and simple data model extensions prior to attempting more complex customization objectives.

More information on the skills and knowledge base required can be found in the **Blackboard Intelligence Roles & Knowledge Requirements Guide**.

DIMENSIONAL MODELING OVERVIEW

All Blackboard Intelligence data warehouse product modules are based on the Dimensional Data Model framework which has been formalized in the data warehouse methodology developed by Ralph Kimball, founder of The Kimball Group. While there are other data warehousing methodologies, our experience has proven that Dimensional Modeling is the most effective approach for accomplishing self-service analytic reporting goals. The document assumes some basic understanding of dimensional data models. Below are some key points to highlight:

- The objective of a Dimensional Data Model is to support decision making, analytics and problem solving – it is optimized for efficiently getting information out (i.e., reporting), and not putting data in (i.e., transaction processing). Efficiency in reporting includes two aspects: i) efficiency in building reports, and ii) efficiency in report query processing.
- Implicit in dimensional modeling is the recognition that a substantial degree of management and analytic information is derived based on specific business rules.

- To accomplish the core objective of optimizing reporting efficiency, the methodology focuses on embedding substantial transformation logic in the ETL process. The benefit of this approach is i) it provides leverage since users do not have to code what may be complex logic in each report to obtain important derived information, and ii) organizations achieve better reporting integrity and consistency since the rules exist once in a formal centralized transformation rule repository.
- The key elements of a dimensional data model are:
    - Fact tables – which are single purpose, subject oriented, atomic level content. Fact tables generally consist of two types of column objects: Measures (e.g., counts and additive metrics), and Dimension keys (foreign keys which point to dimension tables), and
    - Dimension tables – which hold the list of values (i.e., look up tables) for the dimensional entities in the data model. Dimension table give context to the measures, and may include attributes, hierarchies and other dimension metadata.
- Although dimensional data models do not prescribe to third normal form (which is recommended for transactional data models), certain aspects of the model are normalized and some are not. As such, dimensional data models cannot be classified as exclusively normalized or de-normalized.
- OLAP is a database technology optimized for multidimensional analysis, not a data model. All OLAP databases are implicitly dimensional data models; however, all dimensional data models are not necessarily deployed through OLAP technology. Blackboard Intelligence incorporates both relational and OLAP technology within our data warehouse architecture.
- The notion of supporting all operational reporting objectives that may be provided within the context of a transactional database (e.g., transactional audit trails) is not a primary objective of most dimensional data model applications. We believe the ERP systems do a fine job of providing transactional audit trails so our focus is analytics and value-added reporting. Within that context however, there are many opportunities to efficiently provide certain transactional reports directly from the Blackboard Intelligence relational data models.

HEA APPLICATION ARCHITECTURE OVERVIEW

HIGH-LEVEL APPLICATION ARCHITECTURE

To the right is a graphic of the high-level Blackboard Intelligence data warehouse application architecture. There are essentially two major tiers:

- Data Warehouse Database Components: an open data warehouse platform built on SQL Server and Analysis Services,
- Front-end Reporting Layer (Dashboards, Analytics & Reporting): consisting of some delivered content based on Pyramid Analytics, Microsoft Reporting Services and other Microsoft tools, and the flexibility to incorporate other 3rd party reporting tools to leverage the DW relational and OLAP databases.

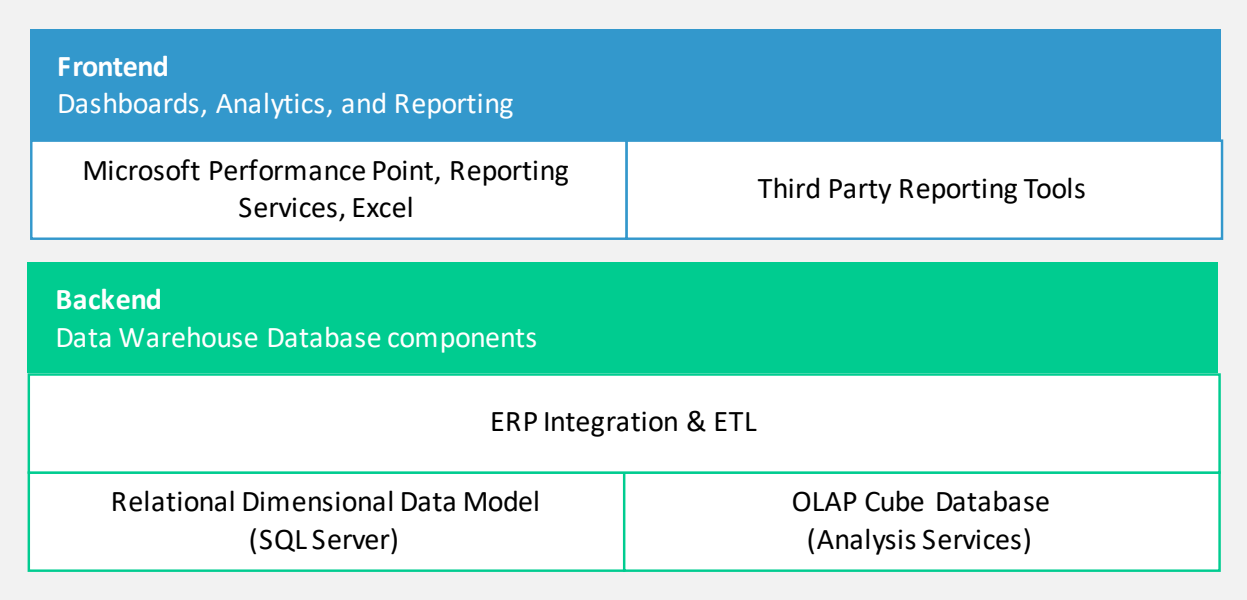


Figure 1 - Blackboard Intelligence data warehouse Application

This document provides a technical reference for the Data Warehouse tier of Blackboard Intelligence data warehouse application.

DATA WAREHOUSE APPLICATION FRAMEWORK

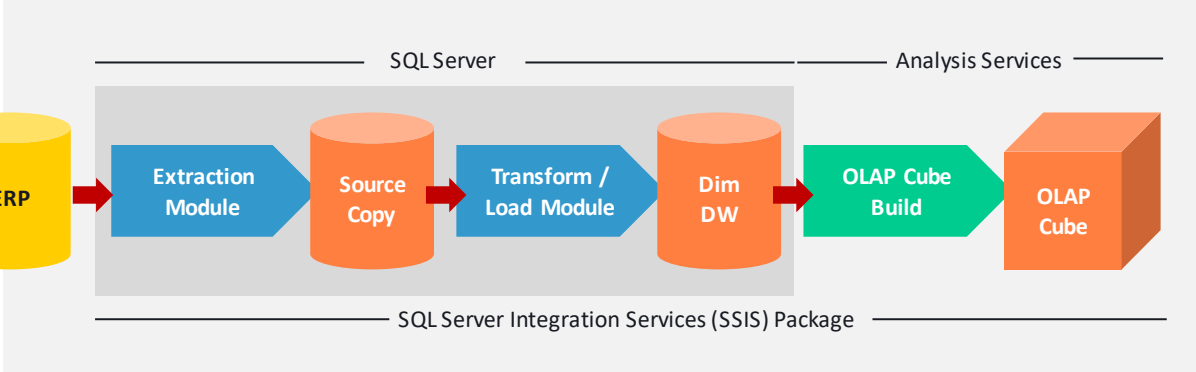


Figure 2 - Blackboard Intelligence data warehouse Application Framework

There are three major elements to the Blackboard Intelligence data warehouse:

1. **Core ETL Application Engine** – a common application framework used in all the Blackboard Intelligence data warehouse product modules from which the ETL process is built and operationally executed. There are two components to the Core HEA ETL Application:
    a. **Extraction Module** – used to define and execute the extraction and copy of source data from the ERP source database to the source tables on the DW server.
    b. **Relational Transformation & Load Module** – used to define and execute the transformation business rules that transform raw ERP data into dimensional information used for reporting and analysis.
2. **OLAP Cube Build Process –** product-specific OLAP database objects
3. **SQL Server Integration Service (SSIS) Package** – used to define and execute the end-to-end data warehouse build cycle.

Below is detailed description of each of the components for the backend data warehouse.

EXTRACT MODULE

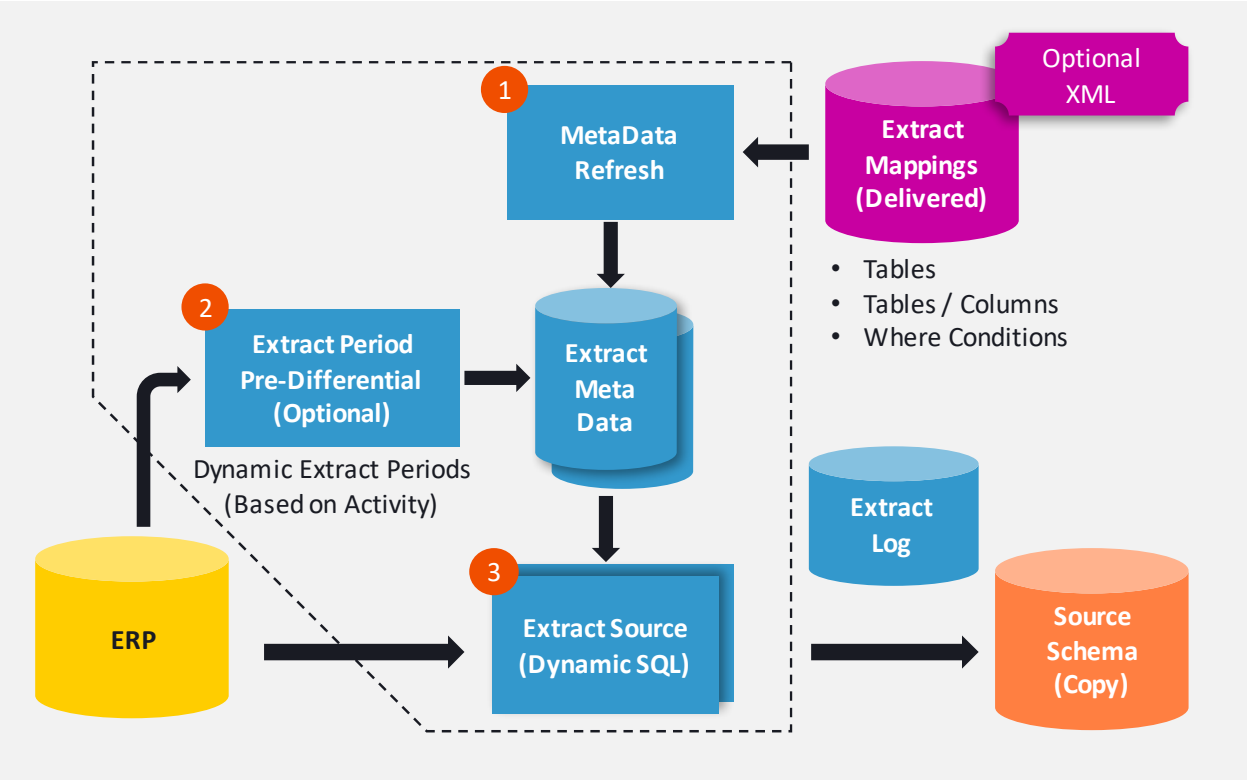


Figure 3 - HEA Core - Extract Module

Version 4.0 of the Blackboard Intelligence data warehouse introduces a new metadata-driven extract architecture enabling dynamic manipulation of extract SQL and the parallelization of extracts across all Source tables, both baseline and custom.

ERP source data is either accessed through a SQL Server Linked Server or a Learn B2 Web Service. A Linked Server is a server-level object containing a driver access method and the credentials to log into a remote database. A Learn B2 Web Service (BB Analytics for Learn only) provides a method to extract source data over the Internet without requiring direct access to the Learn database. In both cases, data is copied from the ERP to the Source schema in the HEA product module database. Three key features of the Extraction Module are shown in the picture and described here:

1. **Dynamically-generated Extract SQL** – Source tables and their columns are described in HEA Core metadata tables. The extract of a particular table, columns within a table, or WHERE clause predicates can be enabled or disabled via this metadata. For each nightly load, the current state of this metadata is used to dynamically regenerate the extract SQL passed to the ERP database.
2. **Differential Extract Possibilities** – For product modules where extracting a subset of the data from an ERP Source table is viable in the Data Warehouse

environment, an optional step in the Extract Module can be enabled to issue pre-processing queries to the ERP Source database to gather information to be used dynamically in the WHERE clause predicates of extracts run later in the process. An application of this in a Financial module extract would be to proactively identify fiscal periods that have had activity since the last load and then update the extract metadata for the General Ledger tables so that they only pull data for those periods.

3. **Extract Parallelization** – Source data extraction is initiated during the nightly process by an Integration Services package that is dynamically created based on the current extract metadata. Using the dynamic SQL created in steps 1 and 2, Integration Services will spawn as many parallel extracts as the server's resources will allow. This process is logged and written to the Data Warehouse's Load Status Report, where row counts and rows extracted per second are reported.

Whether the method is Linked Server or Web Service, there will be one primary stored procedure that performs the extract process. For the Linked Server, the procedure is **HEA.ExtractSourceData**, which extracts data for whatever Source table name is passed into it. For the Web Service, the procedure is **HEA.WebServiceExtractSource**, which processes a list of files supplied by the service and their content. The key Extract metadata is stored in tables HEA.TargetTable and HEA.TargetColumn (see Appendix I for more information on HEA metadata tables).

TRANSFORM & LOAD MODULE

Below is a diagram of the entire data warehouse application framework, highlighting the ETL Component Assembly Engine that performs the transformation of raw Source data into dimension and fact tables in the dimensional data model.

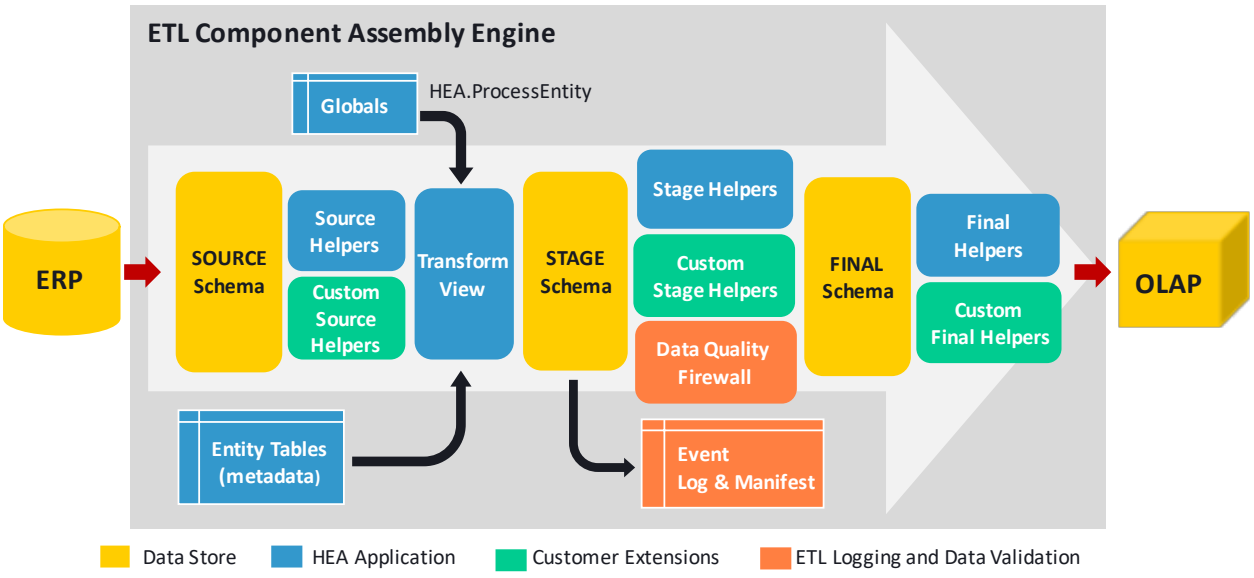


Figure 4 - Dynamic SSIS Package

DATABASES

To understand the application framework, the first concept to understand is what databases are involved in the overall flow of the ETL process. Yellow objects in the picture are "data stores" and include three general classes of databases:

- ERP Source database,
- SQL Server Relational Database Tables, and
- SQL Server Analysis Services (SSAS) database.

The SQL Server relational tables are depicted in the box titled **"ETL Component Assembly Engine"** and reside in four (4) distinct database schemas, each with a specific purpose:

- ***HEA*** – contains the Blackboard Intelligence data warehouse application (system) tables including application metadata, logs and other database components within the ETL application engine.
- ***Source*** – contains a copy of the ERP source data to be used in the transformation process. Note that in this data movement process, no joins or other transformations are performed – it is simply a straightforward movement of raw data that is accomplished very efficiently and quickly. The process to extract and transfer the source data from the ERP database (which generally resides in a SQL Server or Oracle database) is accomplished using a SQL Server **Linked Server** object that defines the interface for interacting with the external source database (or in the case of Datatel where the ERP source data has already been imported into a local database residing on the DW Server

through an outside process) or a Learn B2 **Web Service**, and the HEA Extraction Module described in the preceding section.

- ***Stage*** – contains interim dimensional data models (dimension tables and fact tables) that are used as a work-in-process environment for the ETL transformations prior to running data integrity checks and populating the final reporting schema tables. This Stage schema also creates an effective integration point which enables Institutions to modify/extend the transformation logic, while utilizing the standard data integrity checks.
- ***Final*** – contains the final dimensional data models used for both relational reporting and as a source to build OLAP cubes.

There are three additional database schemas not pictured, *CustomSource*, *CustomStage*, and *CustomFinal* which are used to house Custom dimension and fact table objects and their related ETL components.

ETL COMPONENT ASSEMBLY ENGINE

Blackboard Intelligence's ETL Component Assembly Engine is a dynamic runtime process that assembles components holding the transformation logic (i.e., business rules) to generate the dimensional data models from the underlying source data.

A dimension or fact to be loaded is called an *Entity* and there is one primary stored procedure that performs the Entity assembly process: **HEA.ProcessEntity** (which loads whatever dimension or fact table name that is passed into it). This procedure calls several code-generating sub-processes (e.g., HEA.BuildStatement) to dynamically transform the Entity by referencing stored Entity metadata that defines the *transformation objects* that contain the business rules (logic) for the dimension or fact table being loaded. The key Entity metadata is stored in the tables HEA.Entity, HEA.Dimension and HEA.Fact (see Appendix I for more information on HEA metadata tables).

The transformation objects are comprised of:

- **Transformation Views** – one or more database views that each contain a single query to perform all or most of the transformation logic; and
- **Helpers** – one or more stored procedures that are used to supplement the transformation view in cases where a single view either i) cannot accomplish the transformation logic (e.g., a transformation requiring a two-step process with a self join, such as marking students for retention, or ii) cannot efficiently perform the transformation and a specially tuned stored procedure is required. Helpers can also be used to stage data in some useful way for use later in the transformation (e.g., to perform a time-consuming data lookup just one time and store the results for use by multiple processes later). There are three types of Helpers: Source, Stage and Final

Helpers. They are named in relation to the objects they affect (See Appendix II – Naming Conventions) and reside in the Stage schema.

- Both of these types of Transformation objects indicated in **Blue** on the diagram are part of the standard application codebase and should **not be modified by customers** since future upgrades will replace these database objects.
- **Custom Helpers** – one or more stored procedures that may be used by Institutions to i) tailor transformation business rules to their own specification, or ii) extend / customize the data model by populating added dimensions and measures. Note that:
    - Every transformation process in the application includes a provision to incorporate "Custom Helpers" without any changes to the core ETL process – simply add a stored procedure and the accompanying Helper metadata to HEA.EntityHelper (see Appendices for HEA metadata tables and naming conventions) and the stored procedure will automatically be included in the ETL assembly process.
    - There are three types of "Custom Helper" stored procedures: Source Helpers that update tables in the Source schema (acting as a data pre-processor), Stage Helpers that update interim tables in the Stage schema executing before the Data Quality Firewall checks, and Final Helpers that directly update tables in the Final schema. Most Helpers would generally be Stage Helpers so that customizations are validated within the Data Quality Firewall process.
    - All "Custom Helpers" should reside in the schema called "CustomStage". The baseline Blackboard Intelligence data warehouse application has no awareness of the specific logic within the Helpers – we only provide adapters to automatically run them. As such, any Helper customizations will automatically integrate with upgrades to future versions of Blackboard Intelligence data warehouse. Institutions should always analyze version upgrade notes to determine the impact of new versions on existing customizations (e.g., a customization that has been incorporated into the base product and is no longer necessary).
    - The "Custom Helper" metadata defines an execution order within the Helper type allowing multiple Helpers to run sequentially during the same phase of the load (Source, Stage or Final) (e.g., to allow for separating customizations into their own procedures to simply the granular promotion of customizations from development to production).
- **ETL Helpers** – one or more stored procedures that are used to supplement an entire phase of the ETL process. The types of ETL Helpers are based on the phase that they execute after: Start, Extract, Load Dimensions, Load Facts, and OLAP. An ETL Helper can be used any time a process needs to execute between phases, such as populating a Staging table that is used by multiple Dimensions or executing a procedure on a remote

system before Extract begins. Like standard Helpers and Transformation Views, these should **not be modified by customers** since future upgrades will replace these database objects.

- **Data Quality Firewall (DQF)** – prior to populating the Final reporting schema from the interim Stage schema tables, a data integrity check is performed to ensure that certain integrity rules have been applied to the data. The integrity rules for each dimension and fact table are stored in tables HEA.Dimension and HEA.Fact in an XML-typed column. There are three standard functions that can be referenced within the XML code and executed based on the parameters configured for that object: The functions are:
    - HEA.ValidateMissing – checks for null or missing data.
    - HEA.ValidateFactDimMismatch – checks for invalid foreign keys in the fact table (i.e., referential integrity problems in the source data). Note that the data warehouse uses foreign key constraints to ensure referential integrity; however, the ERP databases generally do not use these constraints.
    - HEA.ValidateDuplicate – checks for duplicate source keys on both dimension and fact tables. Duplicate keys can exist in the natural source data, or may exist as a result of a transformation join based on an unexpected data condition.

Errors detected in any validation process can be classified as:

- **Warnings** – errors whereby the data is loaded into the Final reporting schema, but with a designated default value. For example, transactions with missing or invalid keys may be transformed to a valid default value within the dimension's table such as "Unknown". Warning type errors are indicated in the DW Load Status Report coded in Yellow.
- **Critical Errors** – errors whereby the data is NOT loaded into the Final reporting schema because essential elements of the data cannot be transformed with any integrity. Duplicate keys generally fall into this category, where one of the values is processed but the additional duplicate rows are not loaded into the Final reporting table. Critical errors are indicated in the DW Load Status Report are coded in Red.

Customizing DQF Rules

Institutions may apply their own custom data validation rules by i) coding the required edit or transformation function as a stored procedure, and ii) modifying the XML code in the HEA.Dimension or HEA.Fact table to reference the new function and applicable parameters. This XML can also be edited via the HEAdmin website (see Appendix III – Data Quality Firewall Routines). This metadata driven approach enables customers to integrate any custom edits, transformations or data filtering rules that can be coded in SQL. For example, a function could be developed to exclude records with certain code values, recode certain transactions to reflect ERP data conversion anomalies or create custom edit checks for certain valid values.

All errors detected are logged in the event logging engine for reporting through the DW Load Status report and subsequent research.

The Entity metadata in **HEA.Entity** and its related tables also include a number of additional transformation details referencing views containing source control totals, source keys to be used in duplicate key checks, flags to enable or disable processes and other Entity properties.

OLAP PROCESSING (ANALYSIS SERVICES)

Blackboard Intelligence utilizes SQL Server Analysis Services, an OLAP database technology, to further optimize reporting of the relational dimensional data models by re-organizing this data and storing it in a multi-relational format. The final step of the ETL is to refresh the data in the OLAP layer. During this *processing* task, data is read from the SQL Server layer, indexed, pre-aggregated and compressed to achieve optimizations in storage and query runtime. Analysis Services relies on pre-defined hierarchies and object relationships that are delivered with Blackboard Intelligence data warehouse.

The final processed reporting object is called a ***cube***, which is comprised of one or more fact tables and their related dimensions. Cubes are organized into ***measures***, the numeric information being aggregated, and ***dimensions***, the objects that segment them. Cubes can be further carved up into ***perspectives***, which are just refined views of the cube data, usually exposing just the dimensions and measures relevant to a particular subject area.

This OLAP reprocessing task rebuilds all dimensions and cube data from the data that resides in the Final schema in SQL Server at the time of processing. It does not perform an incremental update; it's a complete refresh. If there are data changes in SQL Server, it will not appear in Analysis Services until the database has been reprocessed.

EXECUTING THE ETL JOB (INTEGRATION SERVICES)

SQL Server Integration Services (SSIS) is used to coordinate the entire ETL process, which is comprised essentially of many calls to SQL Server stored procedures to perform the extracts and entity loads and a final Analysis Services processing task. Integration Services is used specifically to manage precedence constraints and to facilitate parallel processing where possible. Blackboard Intelligence's SSIS package is designed to scale and react to configuration or content changes automatically. In fact, the SSIS package itself never needs to be opened; its data source information is stored in an accompany configuration file and the package is dynamically rebuilt on each run, relying on the metadata stored in the HEA metadata tables (see Appendix I) for which tables and columns to extract, which entities to load in what order and how to load them.

In general, the MainPackage.dtsx SSIS package performs the following tasks to dynamically perform the ETL process:

1. **Extract** - Query HEA.TargetTable for the currently enabled Extracts,
    a. If the module uses the Web Service extract, connect to the Learn B2 Web Service and pull the files corresponding to the source tables first, otherwise,
    b. Construct “execute procedure” statements for HEA.BuildClauses and HEA.ExtractSourceData for each,
    c. Create tasks for each procedure call in an empty SSIS package called DetailSubPackage.dtsx so that they execute in parallel,
    d. Execute DetailSubPackage.dtsx,
    e. Clear out DetailSubPackage.dtsx for use in the next step.
2. **Transform Dimensions** - Query HEA.Entity and HEA.Dimension for the enabled Dimension loads,
    a. Construct “execute procedure” statements for HEA.ProcessEntity for each,
    b. Create tasks for each procedure call in DetailSubPackage.dtsx so that they execute in parallel,
    c. Execute DetailSubPackage.dtsx,
    d. Clear out DetailSubPackage.dtsx for use in the next step.
3. **Transform Facts** - Query HEA.Entity and HEA.Fact for the enabled Fact loads and construct calls to HEA.ProcessEntity for each
    a. Construct “execute procedure” statements for HEA.ProcessEntity for each,
    b. Create tasks for each procedure call in DetailSubPackage.dtsx with precedence constraints between the loads so that HEA.Fact.ExecutionOrder is respected and loads run in parallel where possible.
    c. Execute DetailSubPackage.dtsx,
    d. Clear out DetailSubPackage.dtsx.
4. **Additional Configuration** - Perform additional Dimension member configuration or other module-specific functions:
    a. Set Dimension Ordering based on metadata,
    b. Set Dimension Member Visibility based on metadata
    c. Execute ETL Helpers after each phase if needed.
5. **Load** - Reprocess the Analysis Services database.

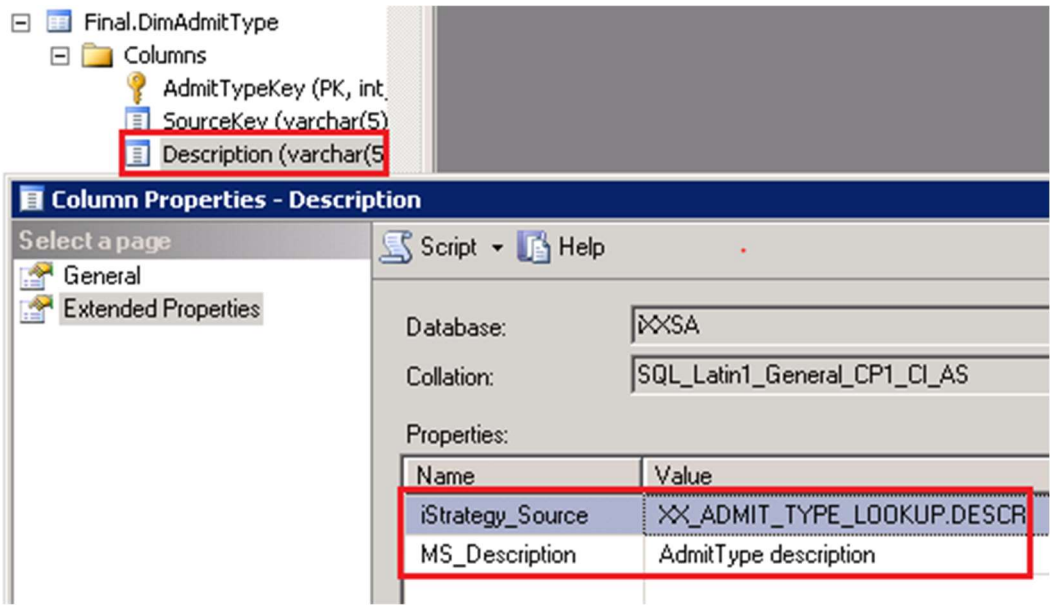
MainPackage.dtsx may call additional stored procedures throughout this process to log package events to the HEA metadata tables. See Appendix I for the key metadata tables that hold this information.

After setting the data source information for the SQL Server and Analysis Services databases in the configuration file (.DTSCONFIG file), the package can be executed manually simply by double-clicking it, or by scheduling it with SQL Server Agent.

SQL Server Agent is the SQL Server service that executes scheduled Jobs. Jobs can consist of SQL scripts, Analysis Services tasks or SSIS packages, among other things and features a robust scheduler. Blackboard Intelligence is installed with one SQL Agent Job for running the nightly load.

DATA MODEL METADATA

Blackboard Intelligence's dimensional data models are delivered with descriptive documentation and ERP mappings in detailed data model compiled help files, typically called *HigherEd Analytics.CHM* or Blackboard Intelligence data warehouse.CHM or something similar. This help file is built out documentation that is embedded in the SQL Server database using SQL Server *extended properties*. Extended properties can be viewed and edited in SQL Server Management Studio's Object Explorer by right-clicking any table, column, view or stored procedure, choosing Properties and selecting the Extended Properties page on the left.



For more information and instructions on using SQLSpec, the **Elsasoft** tool used to regenerate this documentation help file, see the **BbAnalytics Install Guide.PDF**.

EXTENDING THE DATA MODEL

The Blackboard Intelligence architecture is extensible and open allowing the Institution to extend the data model in the following ways:

- Extracting data from additional Sources,
- Creating new dimensional attributes on existing dimension tables,
- Creating new dimensions,
- Creating new measures on existing fact tables, and
- Creating new fact tables.

A set of SQL Server template scripts are provided to guide the Institution through the required SQL development to perform these tasks and adhere to the product's ETL methodology and conventions, provided that the Institution possesses the prerequisite technical skills referenced in the Introduction.

It is important to follow the conventions outlined in the templates to ensure successful subsequent Product Upgrades, as the upgrade process is specifically engineered to handle customizations that follow the Blackboard Intelligence standard.

DATA WAREHOUSE ADMINISTRATION – HEADMIN

The HEAdmin website can be used to administer and configure certain features of the Blackboard Intelligence ETL process. This site is typically accessed at http://*MyAppServer*/headmin or http://*MyAppServer*/headmin4.

Global administrative privileges to the site are granted to users in the group specified in C:\Inetpub\wwwroot\headmin\web.config. Additional granular user or group-level security can be maintained in the sites' Manage Users page.

There are several key functions common across all modules that are documented here.

LOAD STATUS REPORTS

Load Status Reports provide detailed status, audit trail, errors and control information about each daily load process. To view a specific report, select the load start time and click View Report. Each report contains the following sections:

SOURCE EXTRACT

This section contains information related to the process of extracting data from the Source ERP database. An indication of success (green icon) is noted as well as Start and End times, and the

duration of the table extract. A successful extract will also report a calculation of rows extracted per second, useful for identifying throughput issues over time.

A failure will appear in red with the underlying SQL Server or Oracle error that was encountered, if any.

DIMENSION LOADS

Information in this section includes an indication of success or failure for each of the dimension tables in the module. Dimensions are loaded in parallel so they are listed alphabetically. The result for each dimension table load is expandable to display more detailed information about the load process for that table, including load cycle control totals and audit information.

Each dimension links to a view its Manifest which is a technical log of every detail related to its processing. This information is often useful during troubleshooting issues and may be requested when contacting Blackboard Intelligence support.

FACT LOADS

Information in this section is similar to the previous section and includes an indication of success or failure for each of the fact tables in the module, which are again listed alphabetically. The result for each fact table load is expandable to display more detailed information about the load process for that table. A link to the Manifest is also present for fact tables.

HIDE UNISED MEMBERS

This section describes the actions taken based on the configuration of the column HEA.Dimension.ActiveConfig, which identifies whether a dimension should tag rows that have no corresponding fact data in any fact table such that they should not be processed in the Analysis Services layer. This has the effect of “hiding” a dimension member that is “unused” in any transaction, so that it does not appear in reports.

This is useful for hiding conversion or setup data that may appear in ERP lookup tables but are not used by the Institution, or for Date-type dimensions which are pre-populated with 150 years of dates.

OLAP PROCESSING

The final section reports the start, end and duration of the Analysis Services reprocessing task. Any errors encountered during this process are shown in red.

GLOBAL PARAMETERS CONFIGURATION

Blackboard Intelligence uses Global Parameters to customize the ETL process using Institution-specific settings. The select box lists the available global parameters. Editing each one individually will list a short description of its source and purpose, while listing them all with the Edit All button is a quick way to view or edit them all at once.

For many global parameters, there is the option to *Select From List*, which utilizes stored metadata to query data in the Source schema to identify a list of probable values.

The global parameters are configured during installation, but may require periodic maintenance over time.

OBJECT CONFIGURATION

The Object Configuration function allows for maintenance of many of the HEA metadata tables described in Appendix I. The following properties can be configured for dimensions and facts.

Property	Usage
Active	Checkbox to enable or disable the load of an Entity in the ETL process.
Logged	Checkbox to enable or disable the logging of an Entity in the ETL process. *Unlogged Entities will load but will not appear in the Load Status Report.*
Final Table	Schema.Table identifying the Final table, typically Final.Dim<Entity> or Final.Fact<Entity>.
Stage Table	Schema.Table identifying the Stage table, typically Stage.Dim<Entity> or Stage.Fact<Entity>.
Transform View	Schema.View identifying the Transform view, typically Stage.ViewDim<Entity>Transform or Stage.ViewDim<Entity>Transform.
Source Count View	Schema.View identifying the Source Count view, typically Stage.ViewDim<Entity>SourceCount or Stage.ViewDim<Entity> SourceCount. *Source Count views are used to log the expected numbers of rows to be loaded by a process for load cycle reconciliation purposes.*
Dimension Order By Config	XML document describing how to populate the dimension's OrderBy column, used by Analysis Services to order the dimension members. *Not applicable for fact tables.*

Dimension Active Config	XML document describing how to populate the dimension's Active column, optionally used by Analysis Services to exclude rows that aren't referenced by any fact tables. *Not applicable for fact tables.*
Entity Helpers	Click Edit Helpers to open a dialog for maintaining the three types of Helpers for the entity (Source, Stage, Final). If more than one Helper is defined for a particular type, the execution order can be specified as well.
Data Quality Firewall Config	XML document describing the data quality validation routines to execute during the DQF phase of the entity load. Each routine has various properties that can be edited from this dialog. (See Appendix III for DQF Validation Routine descriptions.)

APPENDICES

APPENDIX I – HEA METADATA TABLES

This section describes several of the key metadata tables used throughout the Blackboard Intellgience ETL process.

HEA.Entity	Master ETL metadata table containing one row per Entity to be processed. An Entity is a Dimension, SystemDimension or Fact. The *Code* column is the primary key which is passed into the stored procedure HEA.ProcessEntity to transform Source data into the Final schema: exec HEA.ProcessEntity @EntityCode='DimSample' Key columns: IsActive – Enables or disables processing for the Entity IsLogged – Enables or disables logging of the processing for the Entity IsCustom – Identifies the Entity as a Custom Institution-specific object Type – Identifies Entity Type and where additional metadata can be found
HEA.Fact	Child table of HEA.Entity supplying additional metadata for processing fact tables. Key columns: ExecutionOrder – Defines fact processing precedence constraints TransformView – The name of the view that transforms Source to Stage DataQualityFirewallConfig – XML defining the data validation routines to run
HEA.Dimension	Child table of HEA.Entity supplying additional metadata for processing dimension tables. Key columns: OrderByConfig – Defines the algorithm used to sort the dimension in SSAS ActiveConfig – Enables or disables the hiding of unreferenced rows in SSAS TransformView – The name of the view that transforms Source to Stage DataQualityFirewallConfig – XML defining the data validation routines to run
HEA.EntityHelper	Child table of HEA.Entity identifying the Helper stored procedures configured for each Entity. Key columns: ExecutionOrder – Defines Helper execution order within an ETL stage(Source/Stage/Final) Helper – Name of the Helper stored procedure IsCustom – Identifies a Helper as a Custom Institution-specific object
HEA.EventLoadCycle	Contains start and end times for each load cycle
HEA.EventLog	Event processing log table containing details for the whole ETL process. Key columns: Message – Text description of the Event Manifest – XML document holding technical event processing detail useful for debugging
HEA.EntityEventLog	Entity event processing log table containing details for the processing of Entities (dimensions and facts).
HEA.TargetTableEventLog	Extract processing log table containing start and end times for each Source table. Key columns: RowsProcessed – Number of Source rows extracted

	Duration – Time required to extract Source data in milliseconds RowsPerSecond – Calculation quantifying the extract throughput DetailedMessage – Text description of errors encountered, if any
HEA.TargetTable	Master metadata table for the Extract phase containing one row per table to extract. Key columns: Name – Internal reference for the Source table SourceTable – Name of the table in the ERP Source database SourceAlias – Alias to use for the Source table in generated extract SQL DoConvert – Identifies the ERP Source database as Oracle(1) or SQL Server(0) LinkedServer – Name of the SQL Server Linked Server to use for this extract SourcePrefix – Name of the table's schema in the ERP Source database IsLoadEnabled – Enables or disables the extract of this Source table
HEA.TargetColumn	Child table of HEA.TargetTable listing the columns of each Source table to extract. Key columns: Name – Name of the column in the iXXSA database TargetTableName – Name of the table in HEA.TargetTable SourceColumnName – Name of the column in the ERP Source database IsLoadEnabled - Enables or disables the extract of this column in the Source table DoTrim – Enables or disables trimming the Source column to match the defined size in the iXXSA database
HEA.GLOBAL	Institution-specific global parameters used in transform views and helpers to customize the processing for the Institution's data set.
HEA.ETLHelper	Metadata table identifying the ETL Helpers for each phase of the ETL process. Type – Start, Extract, Dimension, Fact, or OLAP. ExecutionOrder – Defines Helper execution order within an ETL stage(Type) Helper – Name of the Helper stored procedure IsCustom – Identifies a Helper as a Custom Institution-specific object IsEnabled – Enables or disables the helper
HEA.ETLHelperEventLog	ETL Helper event processing log table containing details for the processing of each helper throughout the stages of the ETL process.

APPENDIX II - NAMING CONVENTIONS

This section describes the naming conventions used throughout the SQL Server data warehouse database. In the following table, the tag "<*Entity*>" refers to a dimension or fact to be loaded into the dimensional data warehouse (i.e. a "*Student*" dimension entity or a "*Registration*" fact entity).

Type	Object	Naming Convention	Custom?
Table	**Dimension Stage Table**	Stage.Dim<Entity>	

Table	**Dimension Final Table**	Final.Dim<Entity>	
Table	**Fact Stage Table**	Stage.Fact<Entity>	
Table	**Fact Final Table**	Final.Fact<Entity>	
Table	**Source Table**	Source.*SourceTableName*	
Table	**Pre Stage Table**	Stage.Source*HelperTableName*	
Table	**Custom Dimension Stage Table**	CustomStage.Dim<Entity>	✓
Table	**Custom Dimension Final Table**	CustomFinal.Dim<Entity>	✓
Table	**Custom Fact Stage Table**	CustomStage.Fact<Entity>	✓
Table	**Custom Fact Final Table**	CustomFinal.Fact<Entity>	✓
Table	**Custom Source Table**	CustomSource.*SourceTableName*	✓
Table	**Custom Pre Stage Table**	CustomStage.Source*SourceTableName*	✓
Column	**Dimension Surrogate Key**	<Entity>Key	
Column	**Fact Surrogate Key**	<Entity>Key	
Column	**Custom Dimension Surrogate Key**	<Entity>Key	✓
Column	**Custom Fact Surrogate Key**	<Entity>Key	✓
Procedure	**Dimension Source Helper**	Stage.HelperDim<Entity>Source<*optional descr*>	
Procedure	**Dimension Stage Helper**	Stage.HelperDim<Entity>Stage<*optional descr*>	
Procedure	**Dimension Final Helper**	Stage.HelperDim<Entity>Final<*optional descr*>	
Procedure	**Fact Source Helper**	Stage.HelperFact<Entity>Source<*optional descr*>	
Procedure	**Fact Stage Helper**	Stage.HelperFact<Entity>Stage<*optional descr*>	

Procedure	**Fact Final Helper**	Stage.HelperFact<Entity>Final<*optional descr*>	
Procedure	**Custom Dimension Source Helper**	CustomStage.HelperDim<Entity>Source<*optional descr*>	✓
Procedure	**Custom Dimension Stage Helper**	CustomStage.HelperDim<Entity>Stage<*optional descr*>	✓
Procedure	**Custom Dimension Final Helper**	CustomStage.HelperDim<Entity>Final<*optional descr*>	✓
Procedure	**Custom Fact Source Helper**	CustomStage.HelperFact<Entity>Source<*optional descr*>	✓
Procedure	**Custom Fact Stage Helper**	CustomStage.HelperFact<Entity>Stage<*optional descr*>	✓
Procedure	**Custom Fact Final Helper**	CustomStage.HelperFact<Entity>Final<*optional descr*>	✓

Beyond these object naming conventions, database objects (tables, columns and procedures) are named with compound words or phrases joined without spaces using medial capital letters, or CamelCase. Additionally, words are very rarely abbreviated to limit the need for additional conventions (e.g. the column name CumulativeCreditsAttempted). In rare cases, when an abbreviated name is the generally accepted name, it is used instead of the longer less-accepted name (e.g. the column name CumulativeGPA).

APPENDIX III – DATA QUALITY FIREWALL ROUTINES

The Data Quality Firewall (DQF) supports several out-of-the-box data validation routines for identifying and logging issues found in Stage data. Each routine is a stored procedure that runs in a customized fashion for each dimension and fact table. Its configuration is stored in the DataQualityFirewallConfig XML column in tables HEA.Dimension and HEA.Fact so that each validation can be dimension and fact table-specific. These configurations can be changed by the Institution in the HEAdmin website's Object Configuration function.

For each routine, the ability to log detailed messages containing the row-level data in question is available so that actionable warning and error messages can be displayed in the Load Status Report. However, logging this row-level detail can slow down the load, so a threshold for the maximum number of detailed error messages to write to the log for a particular error is configured in HEA.EventType.Threshold. The Institution can adjust the thresholds as necessary. All of the rows containing errors and warnings will still remain in the Stage table; this feature

only limits the messages displayed in the Load Status Report. It is usually possible to diagnose major problems by inspecting the first several detailed messages. When that is not possible, rows with Errors can be found in the Stage table where the EventType column is equal to "E". Rows with Warnings will have a "W" in the EventType column.

To make the detailed messages context-specific, the DQF configurations use tags that will be replaced with information specific to the dimension, fact or specific row being validated. These tags are shown in braces: "{ }". The delivered data validations routines are described below.

VALIDATING DUPLICATES

This routine identifies rows containing the same data in one or more columns as duplicate errors or warnings.

```
<ValidationRoutines>
  <Routine ProcName="HEA.ValidateDuplicate" IsDetailLogged="N"
       DetailMessage="Duplicate found in {ColumnList} where {UniqueClause}">
    <Duplicate IsError="Y" Resolution="Resolve this issue for the SourceKeys">
      <Column Column="SourceKey" />
    </Duplicate>
    <Duplicate IsError="N" Resolution="Resolve this issue for the Columns">
      <Column Column="UniqueDescription" />
    </Duplicate>
  </Routine >
</ValidationRoutines>
```

<Routine
ProcName = *"HEA.ValidateDuplicate"*
IsDetailLogged = *"Y" or "N" for whether to write detailed messages to the log.*
Detail Message = *Detail Message to write to the log. May contain search & replace tags in braces.*
>
<Duplicate
IsError = *"Y" to tag occurrences as an Error, "N" to tag as Warning.*
Resolution = *Message to display to end user as a possible resolution*
>
<Column
Column = *Name of column in Stage table to check for duplicates*
/> *<!-- multiple Column tags can be added when more than one column defines a unique row -->*
</Duplicate>
</Routine>

VALIDATING MISSING VALUES

This routine identifies rows containing NULL or an empty string in a defined column as an error or warning.

```
<ValidationRoutines>
  <Routine ProcName="HEA.ValidateMissing" IsDetailLogged="N"
       DetailMessage="Missing value found in {ColumnList} where {UniqueClause}">
    <Missing Column="SourceKey" IsError="Y"
             Resolution="Resolve this issue for the SourceKey" />
    <Missing Column="UniqueDescription" IsError="N"
             Resolution="Resolve this issue for the SourceKey" />
</Routine >
</ValidationRoutines>
```

<Routine
ProcName = *"HEA.ValidateMissing"*
IsDetailLogged = *"Y" or "N" for whether to write detailed messages to the log.*
Detail Message = *Detail Message to write to the log. May contain search & replace tags in braces.*
>
<Missing
Column = *Name of column in Stage table to check for missing data.*
IsError = *"Y" to tag occurrences as an Error, "N" to tag as Warning.*
Resolution = *Message to display to end user as a possible resolution*
/>
<!-- multiple Missing tags can be added to check more than one column -->
</Routine>

VALIDATING FACT – DIMENSION MISMATCHES

This routine identifies rows that contain dimensional lookup data in a transaction with no matching value in the dimension table (i.e. referential integrity problems in the Source data). These can be marked as errors or warnings. For example, a transaction may identify a person's gender as "X", but the Gender lookup table only contains the values "M" and "F".

This particular data validation routine requires that the Source data is transformed in such a way that the routine can tell when this situation has occurred simply by looking at the Stage table alone. This is done by coding the Transform view to set the surrogate key values to -2 when Source data has been found for a particular dimension lookup, but it is not present as a valid SourceKey in the Dimension table. When the Fact-Dim Mismatch rule is set as a Warning, the DQF will log the message, and reset the surrogate key value back to -1, the Unknown row. If it is set as an Error, it logs the message, and leaves the surrogate key as -2. The error rows will not be inserted into the Final fact table.

This routine also allows for detail logging of the transactional lookup data that it did find and again, special staging is required for this. In the Fact-Dim Mismatch scenario described above for Gender, a SourceGender column may be added to the Stage table to temporarily house the value found in the transactional data. If a Fact-Dim Mismatch occurs, this column would be referenced by the Detail Message logging to write out that the "X" gender was found. This column would not be added to the Final table; it is intended for error message logging only.

```
<ValidationRoutines>
  <Routine ProcName="HEA.ValidateFactDimMismatch" IsDetailLogged="Y">
    <Mismatch IsError="N" Column="StudentKey" DetailMessage="Invalid Student
({SourceClause}) has been identified where {UniqueClause}. A corresponding value
was not found in Final.DimStudent."
Resolution="SourceStudent(STUDENT_TERM.PERSON_ID) contains a value that is not a
valid DimStudent.SourceKey(PERSON_LOOKUP.PERSON_ID).">
      <Source Column="SourceStudent" />
    </Mismatch>
    <Mismatch IsError="N" Column="ZipCodeKey" DetailMessage="Invalid
PersonalZipCode ({SourceClause}) has been identified where {UniqueClause}. A
corresponding value was not found in Final.DimPersonalZipCode."
Resolution="SourcePersonalZipCode(STUDENT_ADDRESSES.POSTAL) contains a value that
is not a valid DimZipCode.SourceKey(ADDRESSES.POSTAL).">
      <Source Column="SourcePersonalZipCode" />
      <Source Column="SourcePersonalCountry" />
      <Source Column="SourcePersonalCounty" />
      <Source Column="SourcePersonalState" />
    </Mismatch>
  </Routine>
</ValidationRoutines>
```

<Routine
ProcName = *"HEA.ValidateFactDimMismatch"*
IsDetailLogged = *"Y" or "N" for whether to write detailed messages to the log.*
>
<Mismatch
IsError = *"Y" to tag occurrences as an Error, "N" to tag as Warning.*
Column = *Name of column in Stage table to check for a -2 surrogate key*
Resolution = *Message to display to end user as a possible resolution*
>
<Source
Column = *Name of Column to reference in search & replace {SourceClause} tag*
/> *<!-- multiple Source tags can be added if more than one column identifies dimension source -->*
</Mismatch>
</Routine>

APENDIX IV – UNDERSTANDING THE MANIFEST

The Manifest is an XML document that logs all of the steps performed during a dimension or fact load. For each step, it logs any metadata, statistics, dynamic SQL, control totals, errors or warnings pertinent to that step in the load. At the beginning of a dimension or fact load, the Manifest is essentially empty. The Manifest is passed as a parameter to each stored procedure responsible for loading the Entity, where it is updated with any new information about that step and returned as an updated Manifest. This process continues until the Entity load is complete and the final state of the Manifest is saved in HEA.EventLog.Manifest. The Manifest for an Entity load can also be viewed from the Load Status Report, by clicking View Manifest next to each dimension and fact load.

The main purpose of the Manifest is to allow for robust and dynamic communication between the stored procedures responsible for performing the Entity transformation and load, but it is also useful for troubleshooting issues.

The following describes each major section of a sample dimension Manifest:

<Manifest>
  <ProcessMetadata> - *high-level info about what dimension is being loaded and what load cycle it's a part of*
  <Errors> - *logs SQL Server errors encountered (e.g. violation of a unique key constraint)*
  <Step Name="LogProcessStart"> - *logs how many rows are already in the Final table*
  <Step Name="BuildStatement"> - *logs all of the dynamic SQL generated for the loading of this dimension*
  <Step Name="HelperSource"> - *control totals and execution details of Custom Source Helpers, if any*
  <Step Name="TransformStage"> - *control totals and execution details of the INSERT into Stage*
  <Step Name="HelperStage"> - *control totals and execution details of Custom Stage Helpers, if any*
  <Step Name="ValidateDuplicate"> - *execution details, error and warning messages related to this DQF Validation*
  <Step Name="ValidateMissing"> - *execution details, error and warning messages related to this DQF Validation*
  <Step Name="DimensionHistory"> - *execution details for Dimension History logging*
  <Step Name="TransformFinal"> - *control totals and execution details for UPDATE and INSERT into Final*
  <Step Name="HelperFinal"> - *control totals and execution details of Custom Final Helpers, if any*
  <Step Name="SetDimensionAttributeOrderBy"> - *execution details from setting the OrderBy column values*
  <Step Name="FinalizeProcess"> - *execution details for final logging of dimension load detail*
</Manifest>

A sample fact Manifest:

<Manifest>
  <ProcessMetadata> - *high-level info about what fact is being loaded and what load cycle it's a part of*
  <Errors> - *logs SQL Server errors encountered (e.g. violation of a unique key constraint)*
  <Step Name="LogProcessStart"> - *logs how many rows are already in the Final table*
  <Step Name="BuildStatement"> - *logs all of the dynamic SQL generated for the loading of this fact*
  <Step Name="HelperSource"> - *control totals and execution details of Custom Source Helpers, if any*
  <Step Name="TransformStage"> - *control totals and execution details of the INSERT into Stage*
  <Step Name="HelperStage"> - *control totals and execution details of Custom Stage Helpers, if any*
  <Step Name="ValidateDuplicate"> - *execution details, error and warning messages related to this DQF Validation*
  <Step Name="ValidateFactDimMismatch"> - *execution details, error and warning messages related to this DQF Validation*
  <Step Name="TransformFinal"> - *control totals and execution details for DELETE and INSERT into Final*
  <Step Name="SnapshotFinal"> - *control totals and execution details for DELETE and INSERT of Snapshot Version into Final ***
  <Step Name="HelperFinal"> - *control totals and execution details of Custom Final Helpers, if any*
  <Step Name="FinalizeProcess"> - *execution details for final logging of fact load detail*
</Manifest>

***Used in Student Module only*

APPENDIX V – CODE SNIPPETS

The following code snippets show the SQL syntax for tasks commonly performed during development in the relational dimensional data warehouse.

```
/****
* Snippet: Load a Dimension
*/
exec HEA.ProcessEntity @EntityCode = 'DimSample'
go

/****
* Snippet: Load a Dimension and display the final Manifest
*/
exec HEA.ProcessEntity @EntityCode = 'DimSample', @DebugMode = 1
go

/****
* Snippet: Load a Fact Table
*/
exec HEA.ProcessEntity @EntityCode = 'FactSample'
go

/****
* Snippet: Load a Fact Table and display the final Manifest
*/
exec HEA.ProcessEntity @EntityCode = 'FactSample', @DebugMode = 1
go

/****
* Snippet: View current Extract SQL for a Source table
*/
select [Value]
from HEA.Clause
where DomainCode = 'Full Extract' and TargetTableName = 'SOURCE_TABLE'
order by Ordinal

/****
* Snippet: Rebuild the SQL extract clauses based on new metadata
*/
exec HEA.BuildClauses @DomainCode = 'Full Extract', @TargetTableName =
'SOURCE_TABLE'

/****
* Snippet: Extract a Source table
*/
exec HEA.ExtractSourceData @DomainCode = 'Full Extract', @TargetTableName =
'SOURCE_TABLE'

/****
* Snippet: View the dimension and fact manifests from the last load
*/
declare @EventLoadCycleKey int
select @EventLoadCycleKey = max(EventLoadCycleKey) from HEA.EventLoadCycle
select EventLogKey, EventTimeStamp, Message, Manifest
from HEA.EventLog where EventLoadCycleKey = @EventLoadCycleKey and TypeCode =
'Transform Completed'
go
```

APPENDIX VI – REPORT USAGE

Report Usage feature monitors and analyzes data about usage of Report Server (SSRS) and Pyramid reports and display these information in provided set of Report Server (SSRS) reports allowing users to easily browse through them. This set of reports is dependent on the installed Analytics module where the basic set of five reports is available for all modules but the three extra reports will work only in combination with A4L module. For more information about this feature please refer Report Usage SSRS Reports 4.2.docx.