

AN ETL FRAMEWORK FOR USE WITH ORACLE WAREHOUSE BUILDER

VERSION: **1.0**

PREPARED BY,
IAN ABRAMSON

Thoughtcorp Restricted

The information contained herein is proprietary to Thoughtcorp, and it shall not be used, reproduced or disclosed to others except as specifically permitted in writing by the proprietor. The recipient of this information, by its retention and use, agrees to protect the same from loss, theft or unauthorized use.

REVISION HISTORY

| Revision # | Date | Description | Revised By |
|------------|--------------|-----------------|--------------|
| 1.0 | June 1, 2009 | Initial version | Ian Abramson |
| | | | |
| | | | |

TABLE OF CONTENTS

| | |
|--|-------------------------------------|
| Revision History | 2 |
| Approvals | Error! Bookmark not defined. |
| Table of Contents | 3 |
| 1.0 Overview | 6 |
| 1.1 Audience | 6 |
| 2.0 ETL Architecture and Configuration | 7 |
| 2.1 Data Architecture | 7 |
| 2.1.1 Source System Feeds | 9 |
| 2.1.2 Data Staging | 9 |
| 2.1.3 Data Integration | 9 |
| 2.1.3.1 Primary Tier | 9 |
| 2.1.3.2 Secondary Tier | 9 |
| 2.1.4 Summarization | 10 |
| 2.1.5 Data Marts | 10 |
| Loading Methodology | 11 |
| Source System Feeds | 11 |
| File Transmission | 11 |
| File Validation | 11 |
| CNTL File Template | 12 |
| Error Handling | 12 |
| File Archiving | Error! Bookma |
| Source Data Staging | 13 |
| Source system code | 13 |
| Natural Key | 13 |
| Business Run Date | 13 |
| ETL fields | 13 |
| Process Flow – Staging Table Load | 14 |
| Template Feeds | Error! Bookmark not defined. |
| Ad-hoc Feeds | Error! Bookmark not defined. |
| Reference Data | 14 |
| Integration Layer | 15 |
| Data Transformation | 15 |

| | |
|---|----|
| Business Transformation Rules | 15 |
| Data Cleansing | 15 |
| Data Standardization | 16 |
| Dimension Tables | 16 |
| Processing of Dimension Tables | 16 |
| Slowly Changing Dimension – Type One (SCD Type 1) | 16 |
| Slowly Changing Dimension – Type Two (SCD Type 2) | 16 |
| Slowly Changing Dimension – Type Three (SCD Type 3) | 16 |
| Common Processes | 17 |
| Surrogate Key Management | 17 |
| Calculating a Hash Value | 17 |
| Hash Table | 17 |
| DB Action Column | 18 |
| Effective Dates | 19 |
| Process Flow – Dimension Table Load | 21 |
| Fact Tables | 23 |
| Late Arriving Fact Records | 23 |
| Process Flow – Fact Table Load | 23 |
| History Layer | 24 |
| Expiring Records | 25 |
| Data Mart | 26 |
| Control Table | 26 |
| Loading Data Mart Dimension Tables | 26 |
| Loading Data Mart Fact Tables | 26 |
| Mapping Design | 26 |
| Outbound Feeds | 27 |
| Metadata Management and Maintenance | 28 |
| Error Handling and Recovery | 29 |
| Data Errors | 29 |
| Environment Errors | 30 |
| Promotion Strategy | 30 |
| Batch Control Framework | 30 |
| Jobs | 31 |
| Dependencies | 32 |
| Schedules | 33 |
| Control Structures | 33 |
| Appendix A – Source System Extracts | 38 |
| Glossary | 38 |

1.0 OVERVIEW

ETL is the process of Extracting (E), Transforming (T) and Loading (L) data into a data warehouse. Extraction, transformation and loading (ETL) design specify the extraction of data from several sources, their cleansing and integration, and insertion into a data warehouse. This document presents an ETL framework for the Campaign Management Data Mart. The following areas will be addressed:

- ETL Architecture
- Loading Methodology
- Metadata Import and Maintenance
- Change Data Capture
- Error Handling and Recovery
- Promotion Strategy
- Batch Control Framework
- Development Standards

1.1 AUDIENCE

The intended audience of this document is:

- The Development Teams
- The Operations Support Teams
- Data Architecture
-

2.0 ETL ARCHITECTURE AND CONFIGURATION

This section will look at the various data tiers comprising the Data Warehouse and Data Mart architectures. This document will provide a generic example of a data warehouse/data mart solution. The selected architecture is one that will use a Hub and Spoke Approach. The information will feed into a data warehouse, where information is integrated for the Organization and stored efficiently. The information is then disseminated to data marts that are focused on delivering information that will provide value to business processes or business groups. This approach is considered a solid approach and one that can leverage the Oracle Warehouse Builder (OWB) technology to deliver on the required solution.

This document will illustrate the overall flow of data from one tier, within the data warehouse architecture, to the next by way of the ETL process.

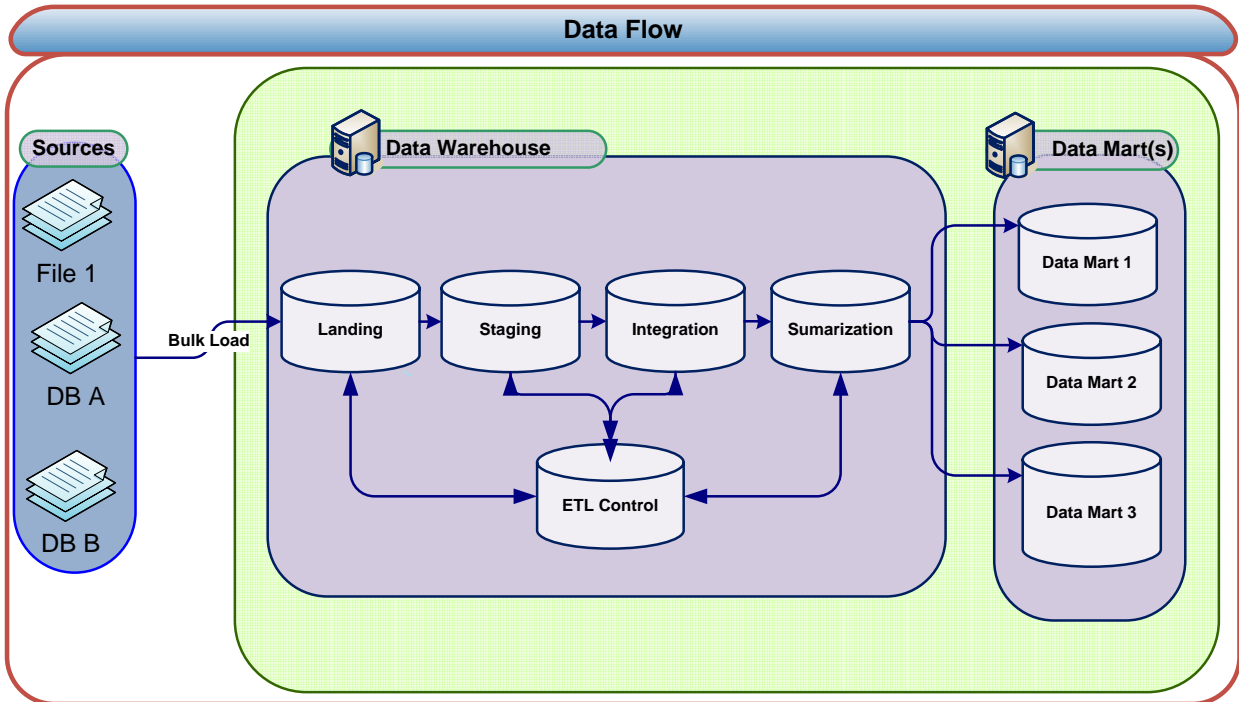
- Source System Feeds
- Data Landing
- Data Staging
- Data Integration/Historical Data Store/Dimension Conformity (Data Warehouse)
- Data Mart

Each of the information form an important part of the overall solution and but addressing each in a unique but standard manner will ensure the best possible reliability and on time delivery.

2.1 DATA ARCHITECTURE

In order to understand the ETL architecture, it is important to first review the data architecture. The data architecture as discussed is a traditional approach with information flowing from various sources to the targets in the warehouse and ultimately to the end user. This approach provides us with a view of the organization at the enterprise level and allows for powerful analysis.

Each layer of the architecture serves a purpose and is important to evolving your business reporting into a robust system with a solid and extendible foundation.



We can now review each data layer and the nature of the information that will be contained in each.

The first layer is comprised of two components. This layer is the source data into the landing tables. In this data layer information will be read from the source files provided by the various source systems into the data warehouse landing structures. This layer will perform little validation and will leverage high-speed loading facilities of Oracle. It is important to note that focusing on doing bulk-type transactions will improve performance of any type of loading requirement. It may also be within this information tier that load deltas may be calculated.

Data then moves to the Staging tier. In this tier information is standardized and validated. It is here that we ensure information from the source is correct and complete. It is here that we attempt to hold any information back from processing unless it is correct. This is a choice that will need to be reviewed for each data warehouse and whether or not information is logged as a warning versus an error must be considered. For this discussion we have selected the option of capturing and holding errors. The opportunity to recycle these records will also be addressed.

The next tier is the Integration layer. This layer may also be known as the data warehouse. The tier contains all information required by the organization at a grain that provides value to analyze the enterprise. This tier may be a traditional 3rd Normal form or dimensional data model. Either option will provide an environment that can support an enterprise-wide view. In addition this tier will contain dimensions and facts that have been conformed and standardized.

The final information layer is the Data Mart tier. This information will be provided via a dimensional data model. This information will be sourced from the Data Warehouse with the ability to track information back to the source. The data is focused on departmental and process needs and may be deployed independently.

2.1.1 SOURCE SYSTEM FEEDS

Source System Feeds supply data from operational systems in a variety of formats, ranging from file feeds to relational structures. Source system feeds may follow a regular delivery schedule or may arrive unscheduled. Some unscheduled data feeds may arrive without knowing what the structure of the file looks like.

2.1.2 DATA STAGING

The first data tier of the Data Warehouse represents a staging area for source data of different formats, reflecting the original structure and content without any transformations applied to the data. The requirements of this area are:

- Storage of file-based content in a pre-defined location. (target location for data movement processes from source)
- RDBMS capture of file-based content.
- Capture of RDBMS based source content via 1-1 mappings
- Initial validation, standardization and exception handling as per requirements specifications

Data in this area may be persistent, if required for delta calculations, or not persistent if changes are provided via a delta file.

2.1.3 DATA INTEGRATION

The Integration Area, which we refer to as well as the core of the data warehouse, will be the central target for all data transformations against source data. This area typically consists of two tiers

- Primary Tier with granular data matching source and target granularity
- Secondary Tier with derived aggregates

2.1.3.1 PRIMARY TIER

The primary staging data tier captures data at the level of granularity required by the target data model. Data object within the primary tier may be persistent or temporary, depending on their association with target entities and their change capture requirements. As such they may be subject to incremental processing or refreshed during each processing cycle.

2.1.3.2 SECONDARY TIER

The secondary tier contains data objects and content entirely derived from the primary tier by way of denormalization and/or aggregation. These structures are typically created to combine data vertically and horizontally such as:

- Temporary or persistent aggregate values, e.g. simple statistics

- Combined content for advanced processing, such as set based analytics with required result capture in the detail data store
- Set based operations for data clustering, e.g. master data integration or house holding

In almost all cases, secondary tier data objects are populated with non-persistent content and as such will be refreshed with every load cycle.

2.1.4 SUMMARIZATION

The Summarization layer provides the organization with the capabilities of pre-calculating or pre-joining information. In this layer the goal is to have summaries or aggregations that continue to serve the enterprise and are not focused on specific reports unless these reports are Enterprise-level information. The goal of this layer is to provide better performance for the reporting group.

2.1.5 DATA MARTS

The Data Marts represents a variety of summary structures that enable direct exploitation of data and information by end users using pre-built reports via portal applications or ad-hoc reporting via advanced query and reporting tools. The Data Warehouse will be used as the single source of data.

The Data Marts follow the principle of data de-normalization and aggregation and applies dimensional relational modeling techniques, also referred to as Star Schemas. In cases, these star schema models may require slight normalization of dimensions, such as creation of lookup tables for dimensional domain definitions. The resulting snowflake structures will provide certain advantages in exploiting dimensional models, such as accelerated lookup of filter criteria and are beneficial to report navigation by reporting tools such as UNICA.

3.0 LOADING METHODOLOGY

This section outlines the methodology employed to move the data from source system extracts through the various data tiers within the Data Warehouse.

- Source System Feeds
- Staging
- Reference/Dimensions
- Integration
- Summaries
- Data Marts
- Outbound Feeds

3.1 SOURCE SYSTEM FEEDS

Extract files are sent on a regular or ad-hoc schedule depending on the originating source system. Files are sent compressed and may contain a header and trailer record. The header and trailer record is used to validate the load process from the source system feeds to the Staging tables. Source systems extracts will be listed in Appendix A.

3.1.1 FILE TRANSMISSION

Data files are transferred to the Data/File Hub (IP Address) via SFTP. Files are delivered into the /incoming directory in sub-folders corresponding to the source system and extract cycle (Bill Cycle, Daily, Weekly, Monthly, and Reference data).

For example, data originating from an Operational source system will be deposited in the incoming folder under the following directory structure:

```
/incoming/project/billcycle  
/incoming/project/daily  
/incoming/project/weekly  
/incoming/project/monthly  
/incoming/project/lookup
```

3.1.2 FILE VALIDATION

Once the files are transferred to the Data/File hub, they are required to undergo a validation process before they are loaded into the staging tables. The validation process is performed at the O/S level and is accomplished through shell scripting. Statistics from the validation process is written to a flat file. This file details may then be loaded into the Load Control table before the start of the OWB load process to the staging tables. While the creation of the flat file is done by scripts, the loading of this file into the Load Control table is done using OWB.

The following steps are part of the file validation process for the a sample source:

| Steps | | Notes |
|-------|---|--|
| 1 | Poll the /land_area/incoming/XXXX incoming source system directory for the presence of a .IN file | <ul style="list-style-type: none"> Only files with an .IN file will be processed XXXX represents the 4 character source system code Report error for missing .IN file |
| 2 | For each .IN file found, make a copy of the corresponding data file in the archive directory: /land_area/archive/XXXX | |
| 3 | Move the data file from /land_area/incoming/XXXX to /land_area/processing/XXXX | |
| 4 | Uncompress extract file | <ul style="list-style-type: none"> Report error if unable to uncompress extract file. Stop processing for this file and continue to the next file Move uncompressed extract file into the rejected directory. |
| 5 | Remove header and trailer record from extract | |
| 6 | Parse header and trailer record and write statistics to CNTL.dat file | |
| 7 | For each file verify number of records in trailer record is equal to the actual number of records in the data file. | <ul style="list-style-type: none"> Report error if number of rows reported in the trailer record does not match the actual rows in the file. |

CNTL FILE HEADER TEMPLATE

FILE_SERIAL_NUMBER|FILE_CREATE_DATE|FILE_CREATE_TIME|FILE_NAME|BILL_CYCLE|CYCLE_CODE|MARKET_INDICATOR|TRAILER_RECORD_COUNT|FILE_SIZE_IN_KB|FILE_OWNER|FILE_GROUP|ARRIVAL_DATE

ERROR HANDLING

Error Codes and Description

| Error Code | Description |
|------------|--|
| 0 | FCM: Success |
| 1 | FCM: Missing OK file |
| 2 | FCM: Cannot uncompress file |
| 3 | FCM: Record count in trailer record does not match actual record count |
| 4 | FCM: Failed to move files to processing directory |
| 5 | FCM: Failed to copy file to archiving directory |
| 6 | FCM: Incoming Ah-hoc file contain more than the 55 stipulated fields |

SOURCE DATA STAGING

The data from each of the source systems will be initially collected in the staging layer (STG). The staging layer is a relational structure (Oracle) that is based on the input data file layouts. The staging layer will become the collective source of the data for the transformation and loading into the integration layer. This facilitates the process of capturing and consolidating disparate data from multiple source systems and mapping it to the requirements of the integration layer. The staging area will assist with the quality assurance as it will provide a means of querying the source level data in a DBMS environment. This will result in a better data validation mechanism.

SOURCE SYSTEM CODE

The source system code is a unique 4 character string that identifies the originating source system. During the load from flat file extracts to staging tables, the source system code is appended to each data record. The source system code is also used in the construction of the natural key as described in the following section.

NATURAL KEY

In order to identify new or changed records when we process records into the integration layer, we must first be able to uniquely identify an incoming record. The assignment of natural keys is done as the data is loaded into the staging tables. Source system analysis has to be completed for every incoming feed to identify what makes a record unique. A field has to be defined on each entity and be large enough to hold all this information. The field, NTRL_SRC_STM_KEY, has been created on all staging tables with a size of VARCHAR2(255) to accommodate this. The natural key is constructed by concatenating the source system code and one or more fields from the source record that will uniquely identify the entity in the source. Staging Table: STG_AMDD_CUST

BUSINESS RUN DATE

When loading data into the integration layer, we need to indicate the effective from date of the record. Also, when a record is expired, we need to specify what the effective end date is. The business run date BUSINESS_RUN_DATE represents the reporting business day for which the data applies and it is populated for each record as the data moves from flat file source to staging tables. This business run date is used to populate the effective from date of a newly inserted record and the effective end date of an expired record.

The business run date is populated using one of the following methods (in order of preference):

1. Use the extract date if it is available as part of the data record
2. Use the extract date if it is contained in the header or trailer record
3. Use the date attached to the name of the data file
4. Use SYSDATE

ETL FIELDS

A series of physical only standard data fields are created in each table, enabling load audit and batch control.

- LOAD_TIME – load timestamp, populated at insert time

- UPDATE_TIME – update timestamp, populated at updated time
- BATCH_ID – Batch ID, unique Identifier of the Batch Run that last touched the record (insert or update)

These fields apply to all tables of all data tiers and will have a not null constraint enabled.

PROCESS FLOW – STAGING TABLE LOAD

The steps to load the source system extracts into the staging layer are as follows:

1. The File Transfer process will place the uncompressed files in the processing directory. Sub-directories for each of the source systems are created within the processing directory.
2. The ETL process will load the file from the processing directory
3. The target table is truncated prior to loading the current day's data.
4. The data in the files will be loaded into an Oracle table in the Staging schema that has a relational equivalent definition of the source system file.
5. Minimal transformation is performed at this stage. Examples of some of the transformation that will be performed are:
 - a. Trim leading and trailing spaces for all fields
 - b. Populate the natural source system key
 - c. LOAD_TIME is assigned to SYSDATE
 - d. UPDATE_TIME is assigned to SYSDATE
 - e. Source System Code is assigned
 - f. BATCH_ID is assigned
6. If the file is successfully loaded, source extracts are moved to the archiving directory.
7. Load statistics is captured and updated in the ETL Control table.

REFERENCE/LOOKUP DATA

Reference or lookup tables are loaded from two sources:

- Source system extracts
- 3rd Party Data Providers

Reference tables that are loaded from source system feeds will be loaded after the extracts are populated in the staging area. The process to load these feeds into the staging area will follow that of the specific source system.

The steps to load reference tables from manual feeds are as follows:

1. The ETL process will load the file from the ../processing/lookup
2. The data in the files will be loaded into an Oracle table in the Reference schema
3. Minimal transformation is performed at this stage. Examples of some of the transformation that will be performed are:

- a. Trim leading and trailing spaces for all fields
 - b. LOAD_TIME is assigned
 - c. UPDATE_TIME is assigned
 - d. Source system code is set
4. Load statistics is captured in the
 5. If the file is successfully loaded, source extracts and file list are deleted from the ready directory.

INTEGRATION LAYER

After the data had been loaded into the staging tables, it is now ready for the business transformation rules (BTR) to be applied. The application of these rules is done as the data moves from the staging to the integration layer. The integration layer contains transformed data for a single processing period. At this stage, data validation, data accuracy and data type conversion rules are also performed as part of the integration layer processing.

The steps to process the data from staging to integration layer are as follows:

1. All data will be read from the staging tables for processing into the integration layer. The staging tables are refreshed every load cycle.
2. Business transformation rules are applied to the data
3. Data validation, data accuracy and data type conversion rules are applied to the data
4. ETL load statistics is captured and populated into the CNTL_INT table.
5. Transformation logic as specified by the STM is applied to the data

DATA TRANSFORMATION

Data transformation rules are applied to the data as it moved from the staging tables to the integration layer.

BUSINESS TRANSFORMATION RULES

Business transformation rules are derived from the source system analysis as well as the source to target mapping exercise. The application of the business transformation rules will be performed as the data is loaded from the staging tables into the integration layer.

DATA CLEANSING

The task of data cleansing also occurs as data is loading into the integration layer. The value that data cleansing provides at this point of the process is to ensure that all data is clean and ready for use in the rest of the warehouse. Data cleansing

DATA STANDARDIZATION

Address information will originate from different source systems. As a result, there is a need for proper address cleansing, standardization and matching rules to be applied to the data in order to have a single version of the truth for campaign management purposes. An undertaking of address cleansing, standardization and matching first require a thorough analysis of all sources from which address data will be gathered. This output of this analysis process will be a set of business rules, with the appropriate sign-off, that can be applied to the address data as it is consolidated into the integration layer.

DIMENSION TABLES

PROCESSING OF DIMENSION TABLES

The following section discusses the various types of Slowly Changing Dimension (SCD) and the approach for performing surrogate key transformation and dimension table updates. Slowly Changing Dimension (SCD) is a strategy to manage both current and historical data across the time span in data warehouses. It is considered as one of the most critical ETL tasks in tracking the history of dimension records.

SLOWLY CHANGING DIMENSION – TYPE ONE (SCD TYPE 1)

When implementing a SCD Type 1, changed information is overwritten and therefore history is lost. Only one version of the record is maintained. This might be suitable for dimension attributes where only the most recent status is relevant and keeping track of history is not required. The obvious advantage of using Type One is that it is the easiest to implement.

SLOWLY CHANGING DIMENSION – TYPE TWO (SCD TYPE 2)

SCD Type 2 retains the full history of values. When the value of an SCD Type 2 attribute changes, the current record is closed. A new record is created with the changed data values and this new record becomes the current record. Each record contains the effective from date and effective to date to identify the time period for which the record is active. For every incremental update, there will be at most one new record created in the dimension table for each active row and there will be at most one record appended to the dimension table for each new row identified in the source data.

SLOWLY CHANGING DIMENSION – TYPE THREE (SCD TYPE 3)

Occasionally, there is the need to track both, the old and the new values of a changed attribute both forward and backward across time of the change. In this case, a new field is added for the affected attribute, occasionally complemented by an effective date. Every time the attribute value changes, the expired value is stored in a PREVIOUS field and the new value is carried forward in a CURRENT field. If an effective date is maintained, it will have to be updated as well to indicate the time the change occurred. It is possible to mix Type Two and Type Three, although it results in increased application complexity and therefore the effort has to be judged.

COMMON PROCESSES

When reviewing the processing of all three types of slowly changing dimensions, the following common processes can be identified:

- Identifying new and changed Dimension Records
- Obtaining new Surrogate Keys
- Appending Dimension Records
- Updating Dimension Records

The following will demonstrate techniques to efficiently implement those processes using Oracle Warehouse Builder.

SURROGATE KEY MANAGEMENT

Since surrogate keys are usually represented by integer values that are incremented for each new dimension record. The transformation operator “Sequence Operator” will be used to generate surrogate keys. Each dimension table will have its own sequence.

CALCULATING A HASH VALUE

After the NTRL_SRC_STM_KEY has been created a mechanism is needed to identify the contents of a given record. We achieve this by populating a digest field using a hash value. The hash value is a hexadecimal string that will be used to compare whether a record has been altered since the last time we received it. One hash value is created for each type of SCD. This field is created as part of the ETL process and utilizes the MD5 function. All fields that we wish to track SCD Type 1 changes on are concatenated together and passed to the function. The resulting hash value is stored in DIGEST1. Likewise, all fields we wish to track SCD Type 2 changes are concatenated together and passed to the function. The resulting hash value is stored in DIGEST2.

MD5 Function:

```
CREATE OR REPLACE function md5hash (v_input_string in varchar2) return varchar2 is
    v_checksum varchar2(20);
begin
    v_checksum := dbms_obfuscation_toolkit.md5 (input_string => v_input_string);
    return v_checksum;
end;
/
```

HASH TABLE

Now that we have a method of identifying each record using the NTRL_SRC_STM_KEY and a way of identifying a records contents by using hash values, we need a place to store this information so that is efficiently accessible during the ETL process. To achieve this we create a corresponding XREF table for each dimension table that contains only the fields needed to perform SCD.

In addition to the NTRL_SRC_STMKEY, two additional columns, DIGEST1 and DIGEST2 are created to store the hash values.

Taking the D_PARTY dimension table as an example, let's suppose we need to track changes on the following 3 columns:

- HOME_PHONE_NUMBER
- STATUS_CODE
- MAIL_ADDR_LINE1

Furthermore, for the columns HOME_PHONE_NUMBER and STATUS_CODE we need to update the target record if there are any changes coming in from the source (SCD Type 1), and for MAIL_ADDR_LINE1, we need to create a new version of the record (SCD TYPE 2) if there is a change in address.

Integration Layer Table: D_PARTY

| D_PARTY_KEY | BAN | HOME_PHONE_NUMBER | STATUS_CODE | MAIL_ADDR_LINE1 | SOURCE_SYSTEM | NTRL_SRC_STM_KEY |
|-------------|-----------|-------------------|-------------|-----------------|---------------|------------------|
| 1 | 511218910 | 4165551212 | O | 45 CRANFIELD RD | AMD | AMD-511218910 |
| 2 | 511218920 | 9055551212 | O | P O BOX 21502 | AMD | AMD-511218920 |
| 3 | 511218930 | 5195551212 | N | 141 KING STREET | AMD | AMD-511218930 |
| 4 | 511218940 | 5185551212 | N | 10168 YONGE ST | AMD | AMD-511218940 |

DIGEST1 is calculated as follows:

```
MD5HASH (HOME_PHONE_NUMBER || STATUS_CODE) = DIGEST1
```

DIGEST2 is calculated as follows:

```
MD5HASH (MAIL_ADDR_LINE1) = DIGEST2
```

D_PARTY_XREF

| D_PARTY_KEY | DIGEST1 | DIGEST2 | SOURCE_SYSTEM | NTRL_SRC_STM_KEY |
|-------------|---------|---------|---------------|------------------|
| 1 | AAA | BBB | AMD | AMD-511218910 |
| 2 | CCC | DDD | AMD | AMD-511218920 |
| 3 | EEE | FFF | AMD | AMD-511218930 |
| 4 | GGG | HHH | AMD | AMD-511218940 |

DB ACTION COLUMN

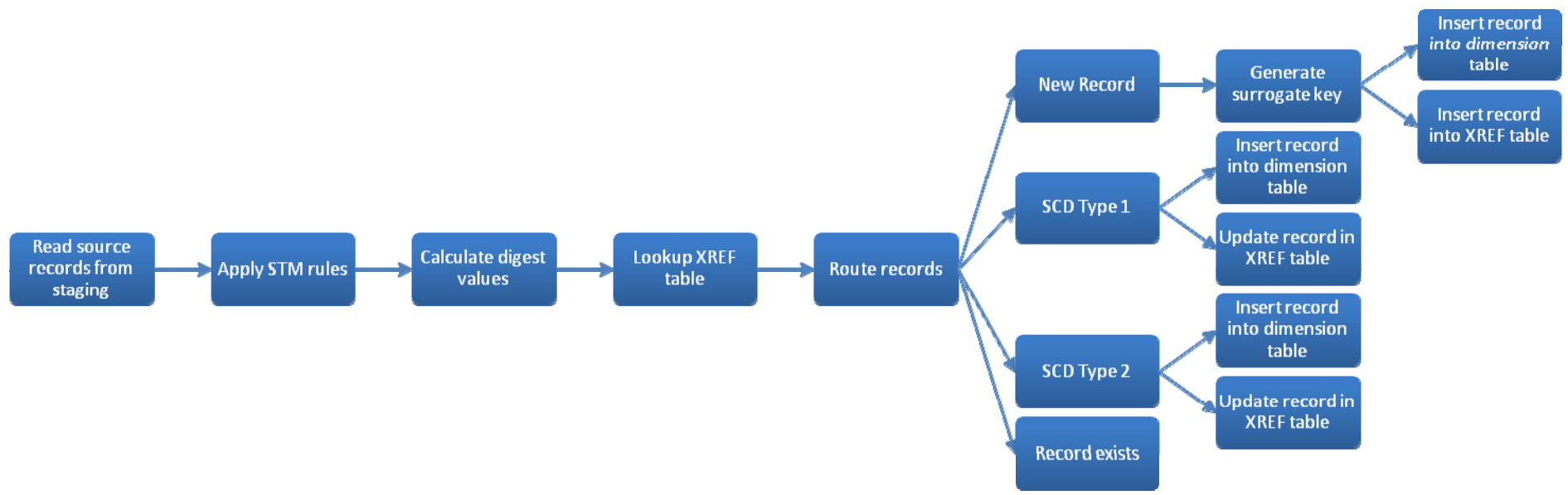
Each dimension table in the integration layer is designed to hold a single snapshot of data. In addition to the transformed data, an addition column, DB_ACTN is used to determine what action needs to be taken as the record is loaded into the history layer.

EFFECTIVE DATES

Storing multiple versions of the same record also requires that the period that the record is valid is also captured. Record effective dates are used to accomplish this. Each entity has the two fields; ROW_EFFECTIVE_DATE and ROW_END_DATE representing the effective from date and the effective to date respectively. When a record is loaded it is given a ROW_EFFECTIVE_DATE equal to the business date on the incoming record. Its ROW_END_DATE is set to the default of '31-Dec-9999'.

| D_PARTY_KEY | BAN | | ROW_EFFECTIVE_DATE | ROW_END_DATE | DB_ACTN | SOURCE_SYSTEM | NTRL_SRC_STM_KEY |
|-------------|-----------|-------|--------------------|--------------|---------|---------------|------------------|
| 1 | 511218910 | | 15-Nov-2008 | 31-Dec-9999 | I | AMD | AMD-511218910 |
| 2 | 511218910 | | 15-Nov-2008 | 31-Dec-9999 | U | AMD | AMD-511218920 |
| 3 | 511218910 | | 15-Nov-2008 | 31-Dec-9999 | I | AMD | AMD-511218930 |
| 4 | 511218910 | | 15-Nov-2008 | 31-Dec-9999 | C | AMD | AMD-511218940 |

PROCESS FLOW – DIMENSION TABLE LOAD



1. Read source records from staging table
2. Apply Business Transformation rules
3. Set:
 - a. LOAD_TIME to SYSDATE
 - b. UPDATE_TIME to SYSDATE
 - c. ROW_EFFECTIVE_DATE to BUSINESS_RUN_DATE
 - d. ROW_END_DATE to '9999-12-31'
 - e. SRC_STM_CODE to 'SSSS' where SSSS represents the source system from which the data originated.
4. Calculate two digest values
 - a. IN_DIGEST1 – for all columns that would trigger an SCD Type 1 change
 - b. IN_DIGEST2 – for all columns that would trigger an SCD Type 2 change
5. Using the IN_NTRL_SRC_STM_KEY, lookup XREF table where IN_NTRL_SRC_STM_KEY = LKP_NTRL_SRC_STM_KEY. Return LKP_SURROGATE_KEY, LKP_DIGEST1 and LKP_DIGEST2 into mapping flow.
6. If XREF.SURROGATE_KEY is null then record is new
7. If XREF.SURROGATE_KEY is not null and IN_DIGEST1 != LKP_DIGEST1 and IN_DIGEST1 = LKP_DIGEST2 then record is an SCD Type 1 change
8. If XREF.SURROGATE_KEY is not null and IN_DIGEST2 != LKP_DIGEST1 then record is and SCD Type 2 change
9. If XREF.SURROGATE_KEY is not null and IN_DIGEST1 = LKP_DIGEST1 and IN_DIGEST1 = LKP_DIGEST2 then record already exists in target. No further action necessary.
10. If record is for Insert:
 - a. Generate new SURROGATE_KEY
 - b. Insert new record into XREF
 - c. Insert record into DATA table with DB_ACTN = 'I'
11. If record is for Type 1 SCD
 - a. Update record into XREF using existing SURROGATE_KEY
 - b. Insert record into DATA table with DB_ACTN = 'U', and existing SURROGATE_KEY
12. If record is for Type 2 SCD:
 - a. Update record into XREF using existing SURROGATE_KEY
 - b. Insert record into DATA table with DB_ACTN = 'C', and existing SURROGATE_KEY

FACT TABLES

The process of loading fact tables in the integration layer mainly consists of separating fact table measures from input data and mapping business keys to surrogate keys. Fact table surrogate key transformation is performed through the associated dimension tables by using current business keys.

LATE ARRIVING FACT RECORDS

Each incoming fact record will have an associated business run date. This date will be used to populate the effective from date as it is loaded into the fact table. Since the join against the dimension table is constrained by the effective from date, the correct dimension record will always be referenced. As a result, no special processing is required for late arriving fact records.

PROCESS FLOW – FACT TABLE LOAD

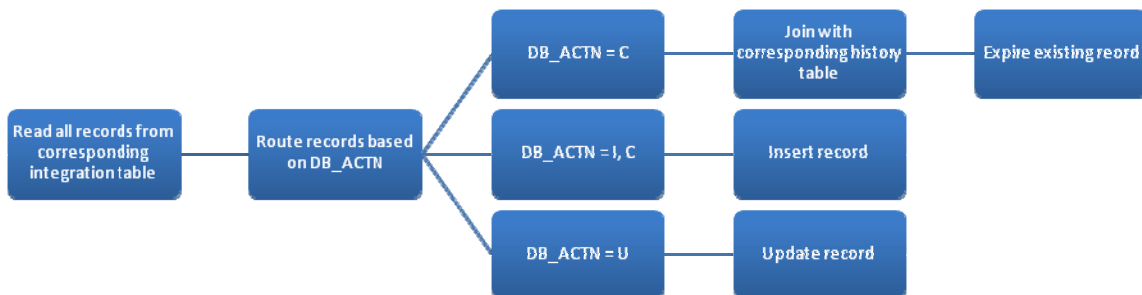


1. Read source records from staging table
2. Apply Business Transformation rules
3. Set:
 - a. LOAD_TIME to SYSDATE
 - b. UPDATE_TIME to SYSDATE
 - c. ROW_EFFECTIVE_DATE to BUSINESS_RUN_DATE
 - d. ROW_END_DATE to '9999-12-31'

- e. SRC_STM_CODE to 'SSSS' where SSSS represents the source system from which the data originated.
4. Construct NTRL_SRC_STM_KEY from input data
5. Using the IN_NTRL_SRC_STM_KEY, lookup XREF table where IN_NTRL_SRC_STM_KEY = LKP_NTRL_SRC_STM_KEY. Return LKP_SURROGATE_KEY into mapping flow.
6. Insert record into fact table

HISTORY LAYER

The integration layer consists of data from a single processing snapshot that has already been transformed and integrated. The next step of the load takes the data from the integration layer to the history layer. Records are either inserted or updated in the history layer. A record is inserted when we have received a new fact or dimension record from the source, or if we need to insert a new version of a dimension record due to a type 2 SCD change. A record is updated either due to a type 1 SCD change or if an old record is to be expired due to a type 2 SCD change. No data transformation is done at this stage. There is a one-to-one correspondence between tables in the integration and history layer. The integration layer version contains data for the single processing cycle, while the history counterpart contains all history. The DB_ACTN column drives the data load into the history layer.



EXPIRING RECORDS

For SCD Type 2 changes, when an updated dimension record is inserted in the dimension table, the existing record must be expired to ensure data consistency. This is achieved by setting the ROW_EFFECTIVE_DATE of the existing record equal to business date of the new version minus one day. For example if the first record was loaded on 15-Nov-2008 and we received an update on 19-Nov-2008 the effective dates in the history layer would look like the following:

Effective Data Initial Load

| D_PARTY_KEY | NTRL_SRC_STM_KEY | ROW_EFFECTIVE_DATE | ROW_END_DATE |
|-------------|------------------|--------------------|--------------|
| 1 | AMD-511218910 | 15-Nov-2008 | 31-Dec-9999 |

Effective Data Incremental Load

| D_PARTY_KEY | NTRL_SRC_STM_KEY | ROW_EFFECTIVE_DATE | ROW_END_DATE |
|-------------|------------------|--------------------|--------------|
| 1 | AMD-511218910 | 15-Nov-2008 | 18-Nov-2008 |
| 1 | AMD-511218910 | 19-Nov-2008 | 31-Dec-9999 |

1. Read all records from integration table
2. Route Records as follows:
 - a. If DB_ACTN = I, C(I) insert record into history table
 - i. Set LOAD_DATE = SYSDATE
 - ii. Set UPDATE_DATE = SYSDATE
 - iii. Set BATCH_ID = \$\$BATCH_ID
 - b. If DB_ACTN = U, update record in history table
 - i. Set UPDATE_TIME = SYSDATE
 - ii. Set BATCH_ID = \$\$BATCH_ID
 - c. If DB_ACTN = C(U), expire record in history table
 - i. Join records with history table
 - ii. Use ROW_EFFECTIVE_DATE = "12/31/9999 00:00:00" to filter records read from the history table. This will ensure only the current version of the record is returned from the table.
 - iii. Update record in history table
 - iv. Set ROW_END_DATE to INCOMING_ROW_EFFECTIVE_DATE
 - v. Set UPDATE_TIME = SYSDATE
 - vi. Set BATCH_ID = \$\$BATCH_ID

DATA MART

Before loading the data from history tables to the data mart, all checks and validations will be completed and passed. Once complete, an incremental data load approach will be used to load data from the history tables to the data mart tables.

CONTROL TABLE

A control table will be used to manage the tracking of the incremental data that needs to be loaded. This table will have the following format:

Table Name: CNTL_DM

Schema: DM

| Column Name | Description |
|--------------|---|
| PROCESS_NAME | Name of the OWB mapping that is loading the data exploration table |
| SOURCE_NAME | Source table for the mapping that contains incremental data |
| LAST_LD_DTM | The maximum row loaded/updated data from the source table that was previously loaded into the campaign management data mart |

The OWB processes that load the data mart tables will follow two different approaches depending on if it is a dimension or a fact table that is being loaded.

LOADING DATA MART DIMENSION TABLES

1. Select all the records in the source table that have a LOAD_TIME or UPDATE_TIME greater than the LAST_LD_DTM value stored in the CNTL_DM table for the source table – process combination and the ROW_EFFECTIVE_DATE = "12/31/9999 00:00:00".
2. For Dimension Tables:
 - a. Look up the dimension record in the target
 - b. If exists update else insert

LOADING DATA MART FACT TABLES

1. Select all the records in the source table that have a LOAD_TIME greater than the LAST_LD_DTM value stored in the CNTL_DM table for the source table – mapping combination.
2. Insert into the data mart

MAPPING DESIGN

Every data mart mapping will contain two streams:

Stream 1 – loading the data mart table:

1. Join the source table from which the records are to be read with the CNTL_DM table.
 - a. Join condition: ($$$$SOURCE_TABLE.LOAD_TIME > CNTL_DM.LAST_LD_DTM$ or $$$$SOURCE_TABLE.UPDATE_TIME > CNTL_DM.LAST_LD_DTM$)
 - b. Filter condition: $$$$SOURCE_TABLE.ROW_EFFECTIVE_DATE = "12/31/9999 00:00:00"$ and $CNTL_DM.PROCESS_NAME = $$$PROCESS_NAME$ and $CNTL_DM.SOURCE_NAME = $$$SOURCE_TABLE$

Stream 2 – Updating the CNTL_DM control table

1. Read records from control table
 - a. Filter condition: $CNTL_DM.PROCESS_NAME = $$$PROCESS_NAME$ and $CNTL_DM.SOURCE_NAME = $$$SOURCE_TABLE$ where $$$$PROCESS_NAME$ and $$$$SOURCE_TABLE$ are input parameters.
 - b. Update the record in the CNTL_DM table with SYSDATE.

OUTBOUND FEEDS

Outbound feeds.

METADATA MANAGEMENT AND MAINTENANCE

Integrated metadata architecture is an integral part of any successful Data Warehouse implementation. As such it is important to define core aspects of a metadata strategy as early as possible during the design and execute consequently. Various reasons require discipline around the management of metadata. Many, if not all of these aspects are mandatory.

- Capture of Technical Metadata, such as tables and columns. This definition is central to the layout of database structures as well as integral component when defining source to target mappings. Technical metadata has to be 100% synchronized between applications.
- Capture of Process Metadata, such as mapping steps and dependencies. In order to reliably document data lineage, it is necessary to capture the information how data flows through the Data Warehouse and what type of transformations are applied. Process metadata support impact analysis when altering Data Warehouse components such as source feeds and target data models but also help document the transformation process and support interpretation of results observed during data exploitation.
- Capture of Business Metadata, such as business definitions and domain ranges. In order for business users to better understand data fields used in reports and ad-hoc applications, it is required to capture clear and unique business definitions, ready for publishing.

In order to establish unique and coherent definitions, it is imperative to provide a single repository for capturing the various types of metadata. CA AllFusion ERWin Data Modeler will be used to centrally define most aspects of technical and business metadata. All logical and most physical data modeling as well as capture of business definitions will be performed in ERWin.

ERROR HANDLING AND RECOVERY

There are two types of errors that may occur during the ETL process:

- Data Errors – inconsistent data types, unexpected null values, orphaned fact records
- Environment Errors – database out of space

DATA ERRORS

Data errors will be captured using OWB DML error logging capabilities. Data errors are detected within the mapping logic and redirected to an error table. The structure of the error table is the same as the target table along with additional columns to capture the error details such as the error code and error message.

| Error Column | Column Description |
|------------------|---|
| ORA_ERR_NUMBER\$ | Oracle error number |
| ORA_ERR_MESG\$ | Oracle error message text |
| ORA_ERR_ROWID\$ | Rowid of the row in error (for update and delete) |
| ORA_ERR_OPTYPE\$ | Type of operation: insert (I), update (U), delete (D) |
| ORA_ERR_TAG\$ | Step or detail audit ID from the runtime audit data. This is the STEP_ID column in the runtime view ALL_RT_AUDIT_STEP_RUNS. |

In order to utilize this capability, the Shadow table name property is set for that target table to be loaded. For example, for target table D_PARTY, if the shadow table name property is set to D_PARTY_ERR, then upon deployment, the error table D_PARTY_ERR will automatically be created as well. Consequently, when the target table is dropped, the error table is also dropped.

Supposing the target table D_PARTY consist of the following columns:

| D_PARTY |
|--------------------|
| D_PARTY_KEY |
| BAN |
| HOME_PHONE_NUMBER |
| STATUS_CODE |
| MAIL_ADDR_LINE1 |
| ROW_EFFECTIVE_DATE |
| ROW_END_DATE |
| DB_ACTION |
| SOURCE_SYSTEM |
| NTRL_SRC_STM_KEY |

Then the corresponding error table D_PARTY_ERR will be as follows:

| D_PARTY_ERR |
|------------------|
| ORA_ERR_NUMBER\$ |
| ORA_ERR_MESG\$ |
| ORA_ERR_ROWID\$ |

```
ORA_ERR_OPTYPE$
ORA_ERR_TAG$
D_PARTY_KEY
BAN
HOME_PHONE_NUMBER
STATUS_CODE
MAIL_ADDR_LINE1
ROW_EFFECTIVE_DATE
ROW_END_DATE
DB_ACTION
SOURCE_SYSTEM
NTRL_SRC_STM_KEY
```

Once corrected, the error table will be loaded by the regular mapping. This is accomplished by creating a second sessions against the original mapping with the source pointing to the error table.

ENVIRONMENT ERRORS

PROMOTION STRATEGY

Promotion strategy

BATCH CONTROL FRAMEWORK

A typical warehouse has many processes, each of which must be scheduled separately. This raises issues about job dependencies, parallelism, and error detection. The ideal solution is one that automates all of these tasks with minimal user intervention. A batch control framework is developed to support best practices in various aspects of job scheduling in a production environment. This framework would help achieve the following tasks:

- Job scheduling
- Job dependencies management
- Job execution information, including:
 - Job identification
 - Job start date and time
 - Job current status
 - Job return code
 - Job history
 - Job log information

- o Job finish date and time

The major steps involved are:

1. Create each individual jobs
2. Define job dependencies
3. Schedule the jobs with Oracle Workflow

JOBS

There are four layers of ETL processing corresponding to each data tier:

- Source Data Staging
- Integration layer
- History layer
- Campaign Management Data Mart

The definition of the jobs to execute the load into each of the above target layers will be examined in this section.

One or more OWB source to target mappings is instantiated in a process flow to load the target tables in the various data tiers. A process flow describes dependencies between OWB mappings and external activities such as email, FTP, and operating system commands. These process flows will be grouped together into process flow packages.

Dependencies across mappings will be implemented within the process flow. In addition, process flows can be contained within other process flows. This capability will be utilized to implement dependencies across process flows. For example, all mappings to load a single source system will be contained in its own process flow, while process flows to load individual source system will be contained in a single parent process flow which will be scheduled as a unit.

The criteria for grouping mappings into a process flow vary on the data tier to be loaded:

- Source Data Staging – All mappings to load data into the staging tables for a given source system and a given refresh cycle will be grouped together in a single process flow. Each process flow will have as many mappings as there are extracts to be loaded into the staging tables. For example, for the source system **Do Not Call - Internal**, the three process flows to load the data into the staging tables will be:
 - o PF_STG_DNCM_DLY, PF_STG_DNCM_WKLY and PF_STG_DNCM_MTHLY, providing files are received daily, weekly and monthly. Process flows will only be built for applicable refresh cycles.
 - o Each process flow will consist of one or more staging layer mappings. For example, PF_STG_DNCM_MTHLY will consist of the following mappings:
 - MAP_STG_DNCM_DNC_REFERENCE
 - MAP_STG_DNCM_EXCLUDE_ALL
 - MAP_STG_DNCM_SMS_EXCLUDE_ALL
- Integration Layer – Process flows for the integration layer will be built based on the source system from where the data originated and subject area to be loaded. For example, data loaded into the Party subject

area from the **Do Not Call - Internal** source system will be contained in a single process flow. In this case, the process flow package will be:

- PF_INT_DNCM_PARTY
- Each process flow will consist of one or more integration layer mappings.
 - MAP_INT_DNCM_D_PARTY
 - MAP_INT_DNCM_D_PARTY_PROSPECT_LOOKUP
 - MAP_INT_DNCM_D_PROSPECT
- History Layer – Each process flow will consist of the mappings to load a particular subject area. For example, data loaded into the party subject area will be grouped together in a single process flow. In this case, the process flow package will be:
 - PF_HST_PARTY
 - Each process flow will consist of one or more history layer mappings.
 - MAP_HST_D_PARTY
 - MAP_HST_D_PARTY_PROSPECT_LOOKUP
 - MAP_HST_D_PROSPECT
- Campaign Management Data Mart – The loading of the data mart will be carried out using a single process flow.

This grouping will ensure minimal changes to the overall process if a data source is added or removed from the system.

DEPENDENCIES

There are two levels of dependencies that will be built into the scheduling system:

- **Mapping level dependencies** – Mapping level dependencies are built within a process flow. Mappings can run in parallel or in sequence. The goal is to run as many mappings in parallel within each process flow as long as referential integrity is preserved.
 - **Staging Process Flows** - Mappings within the staging process flows are executed in parallel as there is no referential integrity to preserve at this level.
 - **Integration Process Flows** - Mappings within the integration process flows are executed in a manner that preserves referential integrity
 - **History Process Flows** - Mappings within the history process flows are executed in a manner that preserves referential integrity
- **Process flow level dependencies** – A process flow in itself can be contained in another process flow. This capability allows us to implement dependencies across sources system loads, subject area loads etc.
 - **Staging Process Flows** – Each source system load into staging will have its own process flow. A single master process flow containing each source system process flow will be built to load all of staging. Dependencies between the source systems will be implemented within the master process flow if applicable.

- **Integration Process Flows** – Process flows containing mappings to load individual subject areas are contained within a master process flow to load the entire integration layer. Dependencies between subject areas are handled within the master process flow.
- **History Process Flows** – The entire history layer will be loaded using a single master process flow. This master process flow will contain individual process flows to load each of the subject areas. Any dependencies between these subject areas will be implemented within the master process flow.

SCHEDULES

Schedules are built to execute OWB process flows and are deployed to Oracle Workflow. Scheduled jobs may reference executable objects such as a process flow or a mapping.

CONTROL STRUCTURES

CNTL_SRC_FCM – This table stores details on the incoming files from the source systems. After the files are processed by the FCM scripts, the captured details are loaded to this table for verification and operational purposes. This table is loaded after the source system files have been loaded into the Staging tables. ACTUAL_RECORDS_LOADED is captured from the OWB repository after the completion of the data load.

| Column Name | Datatype | Source | Description |
|----------------------|---------------|----------------|--|
| FILE_NAME | VARCHAR2(50) | Header record | Logical file name |
| SOURCE_SYSTEM_CODE | VARCHAR2(8) | FCM Script | Source System Code of originating Source System |
| FILE_SERIAL_NUMBER | NUMBER(09) | Header record | Starts with 1 and increases by one for each file that is processed |
| PHYSICAL_FILE_NAME | VARCHAR2(100) | FCM Script | Physical file name as it is received from the source system |
| FILE_CREATE_DATE | DATE | Header record | Valid date in YYYYMMDD format |
| FILE_CREATE_TIME | DATE | Header record | Valid time in HH.MM.SS format. |
| BILL_CYCLE | VARCHAR2(6) | Header record | Bill cycle month/year. Current month/year for monthly extract. Spaces on daily, weekly extracts. |
| CYCLE_CODE | VARCHAR2(02) | Header record | Bill cycle code. Spaces on daily, weekly, monthly extracts. |
| MARKET_INDICATOR | VARCHAR2(05) | Header record | Market indicator relevant for the project. Will be set to NM3 |
| TRAILER_RECORD_COUNT | NUMBER(09) | Trailer record | Total number of detail records in the file excluding Header and Trailer. |
| ACTUAL_RECORD_COUNT | NUMBER(09) | FCM Script | Actual number of records in the file excluding Header and Trailer. |
| FCM_ERROR_CODE | INTEGER | FCM Script | FCM Error Code |

| | | | |
|-----------------------|---------------|-------------|--|
| FCM_ERROR_DESC | VARCHAR2(100) | FCM Script | FCM Error Description |
| FILE_SIZE | NUMBER(09) | FCM Script | File size in bytes |
| FILE_OWNER | VARCHAR2(20) | FCM Script | File owner |
| FILE_GROUP | VARCHAR2(20) | FCM Script | File group |
| FILE_ARRIVAL_DATE | DATE | FCM Script | Timestamp of file on UNIX file system |
| ACTUAL_RECORDS_LOADED | NUMBER(09) | ETL Process | Actual records loaded into the database table |
| TARGET_TABLE | VARCHAR2(50) | ETL Process | Staging table to which the data is being loaded |
| MAPPING_NAME | VARCHAR2(50) | ETL Process | OWB mapping name to load the data |
| LOAD_STATUS | CHAR(1) | ETL Process | Status of the load process. Was the table loaded successfully with all incoming records? |
| LOAD_STATUS_DESC | VARCHAR2(100) | ETL Process | Load status description |
| BATCH_ID | INTEGER | ETL Process | Batch ID of the run that loaded this |
| LOAD_DATE | TIMESTAMP | ETL process | Timestamp of when this record was loaded into the control table |
| UPDATE_DATE | TIMESTAMP | ETL process | Timestamp of when this record was updated in the control table |

CNTL_SRC_FILE – this table contains an inventory of all source system extracts, the target table to which they will be loaded to, and the OWB mapping name that will load the file to the staging table. This table is updated whenever there is a new source system file.

| Column Name | Datatype | Source | Description |
|---------------------------|--------------|-------------|---|
| FILE_NAME | VARCHAR2(50) | ETL Process | Logical file name |
| SOURCE_SYSTEM_CODE | VARCHAR2(8) | ETL Process | Source System Code of originating Source System |
| TARGET_TABLE | VARCHAR2(50) | ETL Process | Staging table to which the data is being loaded |
| MAPPING_NAME | VARCHAR2(50) | ETL Process | OWB mapping name to load the data |
| ACTIVE_FLAG | CHAR(1) | ETL Process | Is this file part of the load schedule? |
| LOAD_DATE | TIMESTAMP | ETL process | Timestamp of when this record was loaded into the control table |
| UPDATE_DATE | TIMESTAMP | ETL process | Timestamp of when this record was updated in the control table |

CNTL_SRC_BUSINESS_RUN_DATE - This table stores the business run date for all extracts files sent from the source systems. This table is loaded from the FCM output file CNTL_SRC.dat. It is the source of the parameter BUSINESS_RUN_DATE populated in all staging mappings.

| Column Name | Datatype | Source | Description |
|---------------------------|--------------|---------------|--|
| FILE_NAME | VARCHAR2(50) | Header record | Logical file name |
| SOURCE_SYSTEM_CODE | VARCHAR2(8) | FCM Script | Source System Code of originating Source System |
| BUSINESS_RUN_DATE | DATE | Header record | Valid date in YYYYMMDD format. File Create Date. |
| BATCH_ID | INTEGER | ETL Process | Batch ID of the run that loaded this |

| | | | |
|-------------|-----------|-------------|---|
| LOAD_DATE | TIMESTAMP | ETL process | Timestamp of when this record was loaded into the control table |
| UPDATE_DATE | TIMESTAMP | ETL process | Timestamp of when this record was updated in the control table |

CNTL_BATCH_ID - A batch id is populated in each of the target table to help identify which records were loaded by which batch run. This table stores the batch ID for a specific run date (execution of the end-to-end process). Batch IDs are never repeated. Prior to the execution of the end-to-end batch process, this table is queried for the latest batch ID.

The batch id is populated into the target table through the use of a mapping parameter. Within each mapping, a mapping parameter BATCH_ID is defined. Once the correct BATCH_ID has been determined for that batch run, the value is passed into the mapping and populated into the target table.

| Column Name | Datatype | Source | Description |
|-----------------|-----------|-------------|---|
| BATCH_ID | INTEGER | ETL Process | Batch ID of the current run |
| RUN_DATE | TIMESTAMP | ETL Process | The date the ETL Process was run for that batch ID |
| LOAD_DATE | TIMESTAMP | ETL process | Timestamp of when this record was loaded into the control table |
| UPDATE_DATE | TIMESTAMP | ETL process | Timestamp of when this record was updated in the control table |

CNTL_RUN_STATS – This table is populated from multiple OWB repository views.

| Column Name | Datatype | Source | Description |
|---------------------------|--------------|-----------------------|--|
| MAP_RUN_ID | NUMBER(22) | ALL_RT_AUDIT_MAP_RUNS | |
| EXECUTION_AUDIT_ID | NUMBER(22) | ALL_RT_AUDIT_MAP_RUNS | |
| BATCH_ID | INTEGER | ETL | Batch ID of the current run |
| DATA_TIER | VARCHAR2(3) | ETL | STG, INT, HST, DMT |
| MAP_NAME | VARCHAR2(80) | ALL_RT_AUDIT_MAP_RUNS | Name of mapping |
| START_TIME | DATE | ALL_RT_AUDIT_MAP_RUNS | Mapping start time |
| END_TIME | DATE | ALL_RT_AUDIT_MAP_RUNS | Mapping end time |
| ELAPSE_TIME | NUMBER(10) | ALL_RT_AUDIT_MAP_RUNS | Elapsed run time of mapping |
| RUN_STATUS | VARCHAR2(8) | ALL_RT_AUDIT_MAP_RUNS | Completion status of the run |
| NUMBER_RECORDS_SELECTED | NUMBER(10) | ALL_RT_AUDIT_MAP_RUNS | Number of records selected from source table |

| | | | |
|--------------------------|----------------|------------------------------|--|
| NUMBER_RECORDS_INSERTED | NUMBER(10) | ALL_RT_AUDIT_MAP_RUNS | Number of records inserted into target table |
| NUMBER_RECORDS_UPDATED | NUMBER(10) | ALL_RT_AUDIT_MAP_RUNS | Number of records updated in target table |
| NUMBER_RECORDS_DELETED | NUMBER(10) | ALL_RT_AUDIT_MAP_RUNS | Number of records deleted from target table |
| NUMBER_RECORDS_DISCARDED | NUMBER(10) | ALL_RT_AUDIT_MAP_RUNS | Number of records discarded before loading into target table |
| NUMBER_RECORDS_MERGED | NUMBER(10) | ALL_RT_AUDIT_MAP_RUNS | Number of merged records |
| EXECUTION_NAME | VARCHAR2(64) | ALL_RT_AUDIT_EXECUTIONS | Name of process flow |
| RETURN_RESULT | VARCHAR2(64) | ALL_RT_AUDIT_EXECUTIONS | Process flow return results |
| RETURN_RESULT_NUMBER | NUMBER(10) | ALL_RT_AUDIT_EXECUTIONS | Result number of process flow |
| RETURN_CODE | NUMBER(10) | ALL_RT_AUDIT_EXECUTIONS | Return code of process flow |
| EXECUTION_AUDIT_STATUS | VARCHAR2(4000) | ALL_RT_AUDIT_EXECUTIONS | Return status of process flow |
| ELAPSE_TIME_TOTAL | NUMBER(10) | ALL_RT_AUDIT_EXECUTIONS | Elapsed run time of process flow |
| RUN_ERROR_NUMBER | NUMBER2(10) | ALL_RT_AUDIT_MAP_RUN_ERRORS | Process flow run error number |
| RUN_ERROR_MESSAGE | VARCHAR2(2000) | ALL_RT_AUDIT_MAP_RUN_ERRORS | Process flow error message |
| TARGET_NAME | VARCHAR2(4000) | ALL_RT_AUDIT_MAP_RUN_ERRORS | Target table name |
| TARGET_COLUMN | VARCHAR2(80) | ALL_RT_AUDIT_MAP_RUN_ERRORS | Target table column |
| STATEMENT | VARCHAR2(2000) | ALL_RT_AUDIT_MAP_RUN_ERRORS | |
| MESSAGE_SEVERITY | VARCHAR2(4000) | ALL_RT_AUDIT_EXEC_MESSAGES | |
| MESSAGE_LINE_NUMBER | NUMBER(10) | ALL_RT_AUDIT_EXEC_MESSAGES | |
| MESSAGE_TEXT | VARCHAR2(4000) | ALL_RT_AUDIT_EXEC_MESSAGES | |
| PROC_ERR_MSG | VARCHAR2(2000) | ALL_RT_AUDIT_PROC_RUN_ERRORS | |
| PROC_STATEMENT | VARCHAR2(2000) | ALL_RT_AUDIT_PROC_RUN_ERRORS | |

Staging Load Status Codes and Description

| Error Code | Description |
|------------|---|
| 0 | STG: Success |
| 6 | STG: Number of records received does not match number of records loaded |

| | |
|-------|---|
| RPC | Remote Procedure Call |
| DB | Database |
| COE | Centre of Excellence |
| ERWin | A software application used to model and engineer databases |
| TOAD | A software application for working with Oracle Databases |
| TB | Terabyte |
| BI | Business Intelligence |
| BIS | Business Intelligence Services |
| OWB | Oracle Warehouse Builder |