Journal of Computing and Information Technology - CIT 20, 2012, 2, 95–111
doi:10.2498/cit.1002046

An ETL Metadata Model for Data Warehousing

Nayem Rahman[1], Jessica Marz[1] and Shameem Akhter[2]

[1] Intel Corporation, USA
[2] Western Oregon University, USA

Metadata is essential for understanding information stored in data warehouses. It helps increase levels of adoption and usage of data warehouse data by knowledge workers and decision makers. A metadata model is important to the implementation of a data warehouse; the lack of a metadata model can lead to quality concerns about the data warehouse. A highly successful data warehouse implementation depends on consistent metadata. This article proposes adoption of an ETL (extract-transform-load) metadata model for the data warehouse that makes subject area refreshes metadata-driven, loads observation timestamps and other useful parameters, and minimizes consumption of database systems resources. The ETL metadata model provides developers with a set of ETL development tools and delivers a user-friendly batch cycle refresh monitoring tool for the production support team.

Keywords: ETL metadata, metadata model, data warehouse, EDW, observation timestamp

1. Introduction

The data warehouse is a collection of decision support technologies, aimed at enabling the knowledge worker (executive, manager, and analyst) to make better and faster decisions [5]. A data warehouse is defined as a "subject-oriented, integrated, non-volatile and time variant collection of data in support of management's decisions" [17]. It is considered as a key platform for the integrated management of decision support data in organizations [31]. One of the primary goals in building data warehouses is to improve information quality in order to achieve certain business objectives such as competitive advantage or enhanced decision making capabilities [2, 3].

An enterprise data warehouse (EDW) gets data from different heterogeneous sources. Since operational data source and target data warehouse reside in separate places, a continuous flow of data from source to target is critical to maintain data freshness in the data warehouse. Information about the data-journey from source to target needs to be tracked in terms of load timestamps and other load parameters for the sake of data consistency and integrity. This information is captured in a metadata model. Given the increased frequency of data warehouse refresh cycles, the increased importance of data warehouse in business organization, and the increasing complexity of data warehouses, a centralized management of metadata is essential for data warehouse administration, maintenance and usage [33]. From the standpoint of a data warehouse refresh process, metadata support is crucial to data warehouse maintenance team such as ETL developers, database administrators, and the production support team.

An efficient, flexible, robust, and state of the art data warehousing architecture requires a number of technical advances [36]. A metadata model-driven cycle refresh is one such important advancement. Metadata is essential in data warehouse environments since it enables activities such as data integration, data transformation, on-line analytical processing (OLAP) and data mining [10]. Lately, in data warehouses, batch cycles run several times a day to load data from operational data source to the data warehouse. A metadata model could be used for different purposes such as extract-transform-load, cycle runs, and cycle refresh monitoring.

Metadata has been identified as one of the key success factors of data warehousing projects [34]. It captures information about data warehouse data that is useful to business users and back-end support personnel. Metadata helps data warehouse end users to understand the various types of information resources available from a data warehouse environment [11]. Metadata enables decision makers to measure data quality [30]. The empirical evidence from the study suggests that end-user metadata quality has a positive impact on end-user attitude about data warehouse data quality [11]. Metadata is important not only from end user perspective standpoint, but also from the standpoint of data acquisition, transformation, load and the analysis of warehouse data [38]. It is essential in designing, building, maintaining data warehouses. In a data warehouse there are two main kinds of metadata to be collected: business (or logical) metadata and technical (aka, ETL) metadata [38]. The ETL metadata is linked to the back-end processes that extract, transform, and load the data [30]. The ETL metadata is most often used by the technical analysts for development and maintenance of the data warehouse [18]. In this article, we will focus mainly on ETL metadata that is critical for ETL development, batch cycle refreshes, and maintenance of a data warehouse.

In data warehouses, data from external sources is first loaded into staging subject areas. Then, analytical subject area tables – built in such a way that they fulfill the needs of reports – are refreshed for use by report users. These tables are refreshed multiple times a day by pulling data from staging area tables. However, not all tables in data warehouses get changed data during every cycle refresh: the more frequently the batch cycle runs, the lower the percentage of tables that gets changed in any given cycle. Refreshing all tables without first checking for source data changes causes unnecessary loads at the expense of resource usage of database systems. The development of a metadata model that enables some utility stored procedures to identify source data changes means that load jobs can be run only when needed. By controlling batch job runs, the metadata model is also designed to minimize use of database systems resources. The model makes analytical subject area loads metadata-driven.

The model is also designed to provide the production support team with a user-friendly tool. This allows them to monitor the cycle refresh and look for issues relating to a job failure of a table and load discrepancy in the error and message log table. The metadata model provides the capability to setup subsequent cycle run behavior followed by the one-time full refresh. This works towards enabling tables to be truly metadata-driven. The model also provides developers with a set of objects to perform ETL development work. This enables them to follow standards in ETL development across the enterprise data warehouse.

In the metadata model architecture, the load jobs are skipped when source data has not changed. Metadata provides information to decide whether to run full or delta load stored procedures. It also has the capability to force a full load if needed. The model also controls collect statistics jobs running them after a certain interval or on an on-demand basis, which helps minimize resource utilization. The metadata model has several audit tables to archive critical metadata for three months to two years or more.

In Section 2 we discuss related work. In Section 3 we give a detailed description of an ETL metadata model and its essence. In Section 4, we cover metadata-driven batch processing, batch cycle flow, and an algorithm for wrapper stored procedures. The main contribution of this work is presented in Sections 3 and 4. In Section 5 we discuss use of metadata in data warehouse subject area refreshes. We conclude in Section 6 by summarizing the contribution made by this work, providing a review of the metadata model's benefits and proposing the future works.

2. Literature Research

Research in the field of data warehousing is focused on data warehouse design issues [13, 15, 7, and 25], ETL tools [20, 32, 27, and 19], data warehouse maintenance [24, 12], performance optimization or relational view materialization [37, 1, and 23] and implementation issues [8, 29]. Limited research has been done on the metadata model aspects of data warehousing. Golfarelli et al. [14] provide a model for multidimensional data which is based on business aspects of OLAP data. Huynh et al. [16] propose the use of metadata to map between object-oriented and relational environment within the metadata layer of an object-oriented data warehouse. Eder et al. [9] propose the COMET model that registers all changes to the schema and structure of data warehouses. They consider the COMET model as the basis for OLAP tools and transformation operations with the goal to reduce incorrect OLAP results. Stohr et al. [33] have introduced a model which uses a uniform representation approach based on the Uniform Modeling Language (UML) to integrate technical and semantic metadata and their interdependencies. Katic et al. [21] propose a model that covers the security-relevant aspects of existing OLAP/ data warehouse solutions. They assert that this particular aspect of metadata has seen rather little interest from product developers and is only beginning to be discussed in the research community. Shankaranarayanan & Even [30] and Foshay et al. [11] provide a good description of business metadata and associated data quality. They argue that managerial decision-making stands to benefit from business metadata. Kim et al. [22] provide a general overview of a metadata-oriented methodology for building data warehouses that includes legacy, extraction, operational data store, data warehouse, data mart, application, and metadata.

The ETL (aka, technical) metadata is not addressed in the research work noted above. In this article, we provide a comprehensive ETL metadata model from the standpoint of data warehouse refresh, metadata-driven batch cycle monitoring, and data warehouse maintenance. Our work covers a broad range of ETL metadata aspects. We provide a means to manage data warehouse refresh observation timestamps, capturing message logs to detect any load or data issues. We also discuss in detail how to control individual job run behavior of subsequent batch cycles runs. Numerous commercial ETL tools with associated metadata model are available today [35]. However, they are proprietary models only. We propose an ETL metadata model that is independent of any ETL tool and can be implemented in any database system. Our model takes care of metadata-driven refreshes in both staging and analytical [26] subject areas in a data warehouse.

Under our ETL metadata model, platform independent database specific utility tools are used to load the tables from external sources to the staging areas of the data warehouse. The proposed metadata model also enables database specific software, such as stored procedures, to perform transformation and load analytical subject areas of the data warehouse. The intent of the software is not to compete with existing ETL tools. Instead, we focus on utilizing the capabilities of current commercial database engines (given their enormous power to do complex transformation and their scalability) while using this metadata model. We first present the ETL metadata model followed by detailed descriptions of each table. We also provide experimental results (via Table: 2 to 6 in Section 5) based on our application of the metadata model in a real-world, production data warehouse.

3. The Metadata Model

Metadata is "data about data". A metadata model is critical for the successful implementation of a data warehouse [22] and integrating the data warehouse with its metadata offers a new opportunity to create a reliable management and information system. Metadata is essential for understanding information stored in data warehouses [16]. It helps increase levels of adoption and use of data warehouse data by knowledge workers and managers. The proposed ETL metadata model allows for the storage of all kinds of load parameters and metrics to make data warehouse refreshes metadata-driven. It also holds error and message logs for trouble shooting purposes. It enables tracing the historical information of batch loads. Metadata ensures that data content possesses sufficient quality for users to use it for a specific purpose [6, 11].

In data warehouse, batch job runs are automated given that they run periodically, several times a day, for efficiency reasons. In order to run thousands of jobs via batch cycles in different subject areas, the jobs are governed by an ETL metadata model to determine the load type, and to capture different load metrics, errors and messages. A wrapper stored procedure is a procedure that executes several utility procedures to make load type decisions and, based on that, it runs full or delta stored procedure or skips the load.

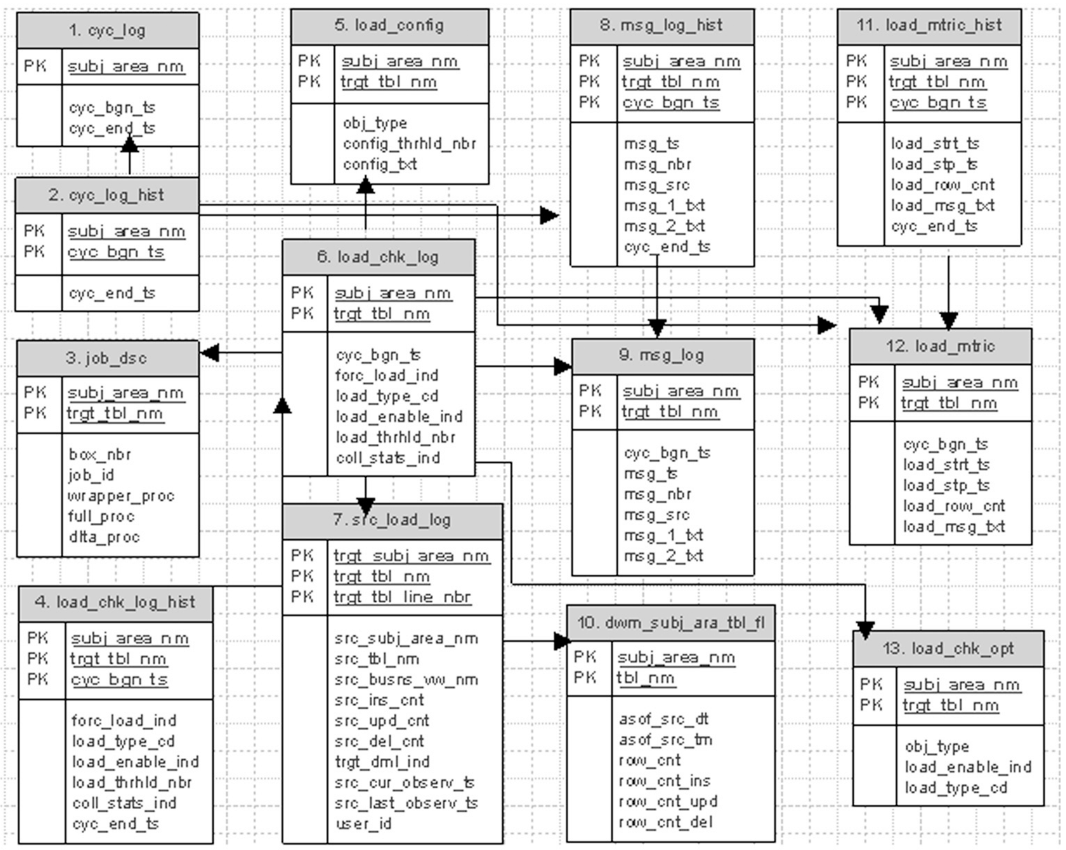


Figure 1. ETL metadata model for data warehousing (derived from [26]).

In Figure 1, we provide a metadata model that is used to capture metadata for all kinds of data warehouse refresh activities related to ETL. The model consists of metadata tables, wrapper stored procedures and utility stored procedures. The wrapper stored procedure for individual jobs first inspects the latest observation timestamp in a metadata data table for each of the source tables referenced in the load stored procedures, to detect the arrival of fresh data. The load procedures are bypassed if the source data has not changed. If source data changes are detected, the wrapper stored procedures call full or delta (aka, incremental) stored procedures based on load condition of each table load.

In the data warehouse, a staging or analytical subject area batch cycle refresh kicks off based on a notification that upstream source subject area refreshes are completed. After a subject area batch cycle begins, a pre-load job is run via utility procedure which updates the table with a cycle-begin-timestamp. A post-load procedure updates the table with a cycle-end timestamp immediately after the actual table loads are completed. The cycle-begin and cycle-end timestamps stored in this table are used by full and delta stored procedures during an actual table load and to capture load information in other metadata tables during the cycle refresh process.

In a data warehouse, each subject area normally gets refreshed several times a day. In order to keep track of refresh information for each batch cycle, refresh begin and end timestamps are important to capture. The table 'cyc_log' is used to hold subject area refresh timestamps. This lets users know the timeliness of the data a particular table holds. The cycle-begin timestamp is used while capturing load metrics for each table load.

Table 'load_chk_log' holds one row per target table in each subject area. After a job kicks off for the first time, a utility procedure checks to see whether a metadata row exists for the table. If no row is found, a default row with several parameters will be inserted on the fly to do a full refresh. After the cycle refresh is completed the column load_type_ind will be set to 'D' so that a delta load is performed in the subsequent cycles.

Sl No.	Entity Name	Entity Description
1	Cycle Log (cyc_log)	Holds subject area refresh begin and end date and time
2	Cycle Log History (cyc_log_hist)	Archives data from table 'cycle log' (#1) for last two years on a rolling basis
3	Job Description (job_desc)	Stores batch cycles job description
4	Load Check Log History (load_chk_log_hist)	Archives load parameters from table 'load check option' (#6) for last two years on a rolling basis
5	Load Configuration (load_config)	Holds threshold values to trim the archive tables (#2, 4, 8, 11) on a rolling basis.
6	Load Check Log (load_chk_log)	Holds load parameters for each table under batch cycles
7	Source Load Log (src_load_log)	Holds last source data change timestamp for each source table used to load target tables.
8	Message Log History (msg_log_hist)	Archives data from table 'message log' (#9) for last two years on a rolling basis
9	Message Log (msg_log)	Loads error messages/ information for each SQL in a procedure used to load a table. Used to troubleshoot job failures.
10	Source Table File Information (src_tbl_file_info)	Holds source data file information. Used to set up load parameters for target tables in a batch cycle in DW
11	Load Metric History (load_mtric_hist)	Archives data from table 'load metric' (#12) for last two years on a rolling basis
12	Load Metric (load_mtric)	Holds detailed load metrics for each target table load
13	Load Check Option (load_chk_opt)	Holds load parameters to determine load behavior of subsequent cycle refreshes for each table load

Table 1. Metadata model entity description.

Table ‘src_tbl_file_info’ holds a load timestamp for each staging table. Table ‘src_load_log’ stores an observation timestamp that is dependent on a source table’s last update timestamp. If a target table is loaded with data from multiple source tables, this table will store an observation timestamp for each of the source tables. The observation timestamp is the date and time at which the consistent set of triggered data was loaded into the target table. The trigger timestamp is always less-than or equal-to (<=) the observation timestamp.

Table ‘load_mtric’ stores vital load statistics (cyc_bgn_ts, load_strt_dt, load_strt_ts, load_stp _ts, load_row_cnt, and load_msg_txt, and runtime) for each target table. Statistics are also used for report-outs of runtimes. This table provides information about whether the table is loaded (with a full or delta) or the load is disabled, and the reason for the no load. The load_mtric enables monitoring progress of a batch cycle while it is running. Table ‘msg_log’ stores load errors and messages for trouble shooting and information purposes. This provides useful data to ETL programmers and production support team to trace the root cause of job failure, incorrect load or no load.

Table ‘load_chk_opt’ stores job information (subj_area_nm, trgt_tbl_nm, load_enable_ind, load_type_cd). For each job, one entry is required to exist in this table. The entries are inserted into this table via a utility stored procedure call from within the wrapper stored procedure. The load_type_cd field holds the value of ‘Full’ or ‘Delta’. Based on this information, a post-load utility stored procedure will update the ‘load_chk_log’ table to prepare it for the next cycle refresh. The column ‘load_enable_ind’ holds the value of ‘Y’ or ‘N’. Based on this information, the post-load stored procedure will update the ‘load_chk_log’ table to prepare it for the next cycle refresh.

In a data warehouse, hundreds of jobs run under different subject areas as part of multiple cycle refreshes every day. A job description for each table needs to be conveyed to production support along with job details. Each job has certain key descriptions such as job identifier, box, table name, full, delta, and wrapper stored procedures names; the ‘job_desc’ table holds this information (subj_area_nm, trgt_tbl_nm, autosys_box_nbr, job_id, wrapper_proc_nm, full_ proc_nm, dlta_proc_nm, debug_mode_ind). The entries are inserted into this table via a utility stored procedure call. This table is used by the production support team to get detailed information about each job run.

The load type of each table in a particular subject area varies; some tables are refreshed full some are refreshed incrementally; some others are loaded on a weekly or monthly basis or on the first or last day of the month. In order to control the load behavior of each target table, each source table's load observation timestamp need to be captured. The 'load_config' table holds all of this information to enable each target table's refresh according to its specific schedule.

4. Metadata-driven Batch Processing

An efficient, flexible and general data warehousing architecture requires a number of technical advances [36]. A metadata-driven batch cycle refresh of a data warehouse is one of these technical advances. Successful data warehousing is dependent on maintaining data integrity and quality in table refreshes. Metadata-driven refreshes play a prominent role in this regard. Loading inconsistent data negatively impacts data quality if an inefficient metadata model is not devised. In this article, we attempt to provide a comprehensive ETL metadata model for data warehouses.

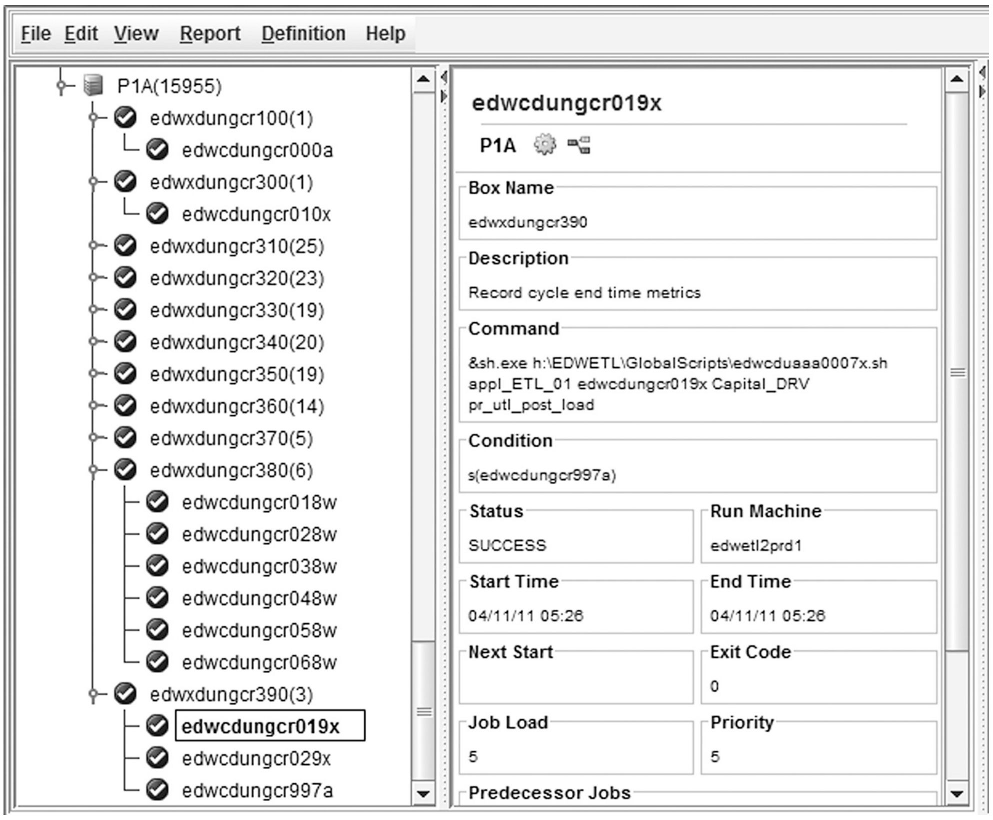
In data warehouses, each subject area is refreshed through a batch cycle. Under the batch cycle, jobs are run in different boxes to load the target tables in order of dependency on other tables in the subject area. Jobs in a subject area run under different conditions: some jobs load tables with full refreshes while other jobs refresh the tables incrementally; some other jobs in the cycle skip performing incremental loads if source data has not changed; some jobs are set to do incremental loads, but end up doing full refreshes when the source has a large number of new records or the source or target table row count does not match after an incremental refresh is performed. Our proposed ETL metadata model controls the table load to satisfy the said load conditions.

The metadata model is based on several metadata tables, utility stored procedures, wrapper stored procedure, and full and delta load stored procedures for individual table loads. The model introduces a new paradigm for batch processing by loading only those tables for which there are new records in the source tables. Inspection of the latest observation timestamp in a metadata data table for each of the source tables referenced in the incremental load stored procedures detects the arrival of fresh data. The full and incremental load procedures are bypassed if the source data has not changed. The ETL metadata model allows for storing table-level detailed load metrics (timestamp, row count, and errors and messages for trouble shooting). By providing the source and the target table last load timestamp, row count, and load threshold information, the model allows accurate incremental load processing and enables a decision about whether to perform full or incremental refresh.

4.1. Batch Cycle Process Flow

In most cases during the last decade, data warehouses were refreshed on a monthly or weekly or daily basis. Now-a-days, refreshes occur more than once per day. A batch cycle that runs based on subject areas in a data warehouse may run several times a day. A batch cycle holds several boxes which run in order of dependency. The very first box (Figure 2: edwxdungcr100) holds a gatekeeper job that looks for a notification file from one or more upstream (source) subject areas. The second box (Figure 2: edwxdungcr300) in a cycle usually holds a job that runs a pre-load job (Figure 2: edwcdungcr10x) to execute a metadata utility procedure (pr_utl_pre_load) to prepare for the current cycle refresh by doing clean-up, insert, and update of metadata tables. It deletes 'asof_dt' and 'asof_ts' from the last cycle and inserts new dt and ts for the current cycle. It writes an entry to the msg_log table to log that the new cycle has started. The pre-load procedure updates the table with the cycle-begin date and timestamp after the gatekeeper job successfully runs. A post-load procedure updates the table with cycle finish timestamp soon after the refresh is completed.

The subsequent boxes (Figure 2: boxes edwxdungcr310 to edwxdungcr380) hold actual jobs to load the target tables in the subject area. These jobs are placed in different boxes to run them in sequence and in order of dependency. A



P1A(15955)
- edwxdungcr100(1)
  - edwcdungcr000a
- edwxdungcr300(1)
  - edwcdungcr010x
- edwxdungcr310(25)
- edwxdungcr320(23)
- edwxdungcr330(19)
- edwxdungcr340(20)
- edwxdungcr350(19)
- edwxdungcr360(14)
- edwxdungcr370(5)
- edwxdungcr380(6)
  - edwcdungcr018w
  - edwcdungcr028w
  - edwcdungcr038w
  - edwcdungcr048w
  - edwcdungcr058w
  - edwcdungcr068w
- edwxdungcr390(3)
  - edwcdungcr019x
  - edwcdungcr029x
  - edwcdungcr997a

edwcdungcr019x

P1A

Box Name: edwxdungcr390

Description: Record cycle end time metrics

Command:
```
&sh.exe h:\EDWETL\GlobalScripts\edwcduaaa0007x.sh
appl_ETL_01 edwcdungcr019x Capital_DRV
pr_utl_post_load
```

Condition: s(edwcdungcr997a)

Status: SUCCESS	Run Machine: edwetl2prd1
Start Time: 04/11/11 05:26	End Time: 04/11/11 05:26
Next Start:	Exit Code: 0
Job Load: 5	Priority: 5

Predecessor Jobs

Figure 2. Batch cycle flow.

downstream-box job in the cycle might have dependencies on tables in one or more upstream boxes in the subject area. Each job calls the wrapper stored procedure for a target table to execute a full or delta stored procedure.

After all load-jobs under different boxes run to success, the last box in the cycle (Figure 2: edwxdungcr390) executes a job (Figure 2: edwcdungcr029x) to run a utility procedure (pr_utl_refresh_status) which updates a metadata table (Table 1: #5 – load_config) with source data latest timestamp and target cycle refresh finish timestamp. The analytical community has visibility into the latest data availability timestamp via a web tool. The batch cycle also executes another job (Figure 2: edwcdungcr019x) to run a metadata post-load utility procedure. The utility procedure (pr_utl_post_load) archives all load metrics for the current cycle refresh into several history or audit tables (Table 1: #2, #4, #8, #11) to retain them for a specified period of time. The post-load stored procedure is called by a batch cycle job. It archives the current cycle metadata (from load_metric, cyc_log, msg_log) to historical tables (load_metric_hist, cyc_log_hist, msg_log_hist). It updates the cyc_end_ts field of cyc_log table with current timestamp to log that the cycle load has completed. It trims the historical tables (load_metric_hist, cyc_log_hist, msg_log_hist) of their least recent cycle records on a rolling basis. One quarter's worth of records is normally preserved for investigation and research. This procedure also makes sure that the 'load_chk_log' table is reverted to the default settings. If during a particular cycle refresh we want to do a 'Full' refresh of a particular job and then if for the subsequent runs we want to do 'Delta', we use this post-load procedure to set loadtype = Delta (default).

4.2. Algorithm for Wrapper Stored Procedures

The data warehouse subject area refresh is done via batch cycle runs. Each job in a batch cycle calls a wrapper stored procedure to execute a full or delta stored procedure for a full or incremental refresh. The wrapper procedure makes the decision whether to perform a full refresh or delta refresh or to skip the refresh altogether through a few utility stored procedures that are governed by the metadata model. The load is skipped if the model finds that the source table's last observation timestamp has not changed. The full or delta refresh is done based on individual threshold values set for each target table. If the new delta counts found in the source table are within the threshold percentage the model runs a delta stored procedure; otherwise, it turns on the full load procedure.

Figure 3 shows that the wrapper procedure checks the metadata table to see if the source load current observation timestamp is greater than the observation timestamp of the last target table load. The next step of the decision flow is to check if the load type indicator is full or delta. The default is full load if no load_type value is provided.

Figure 4 provides a template consisting of the wrapper stored procedure algorithm. First, it calls a utility procedure to get the current cycle begin timestamp. The cycle begin timestamp (Table 1: #1) is used while capturing metadata for target table load. The next utility procedure (Figure 4, pr_utl_chk_tbl_changed) in the wrapper is used to check whether the source table has new records. This is done by comparing the latest observation timestamp in metadata tables, src_tbl_fl_info (Table 1: #10) and src_load_log (Table 1: #7) for the target table load. The wrapper stored procedure runs the next utility procedure (Figure 4: pr_utl_Get_LoadCheck_Info) to pull the load parameters (force_load, load_type, load_enabled, etc) from the load_chk_log table (Table 1: #6). These parameter values provide information about load type such as full or delta load or no load.

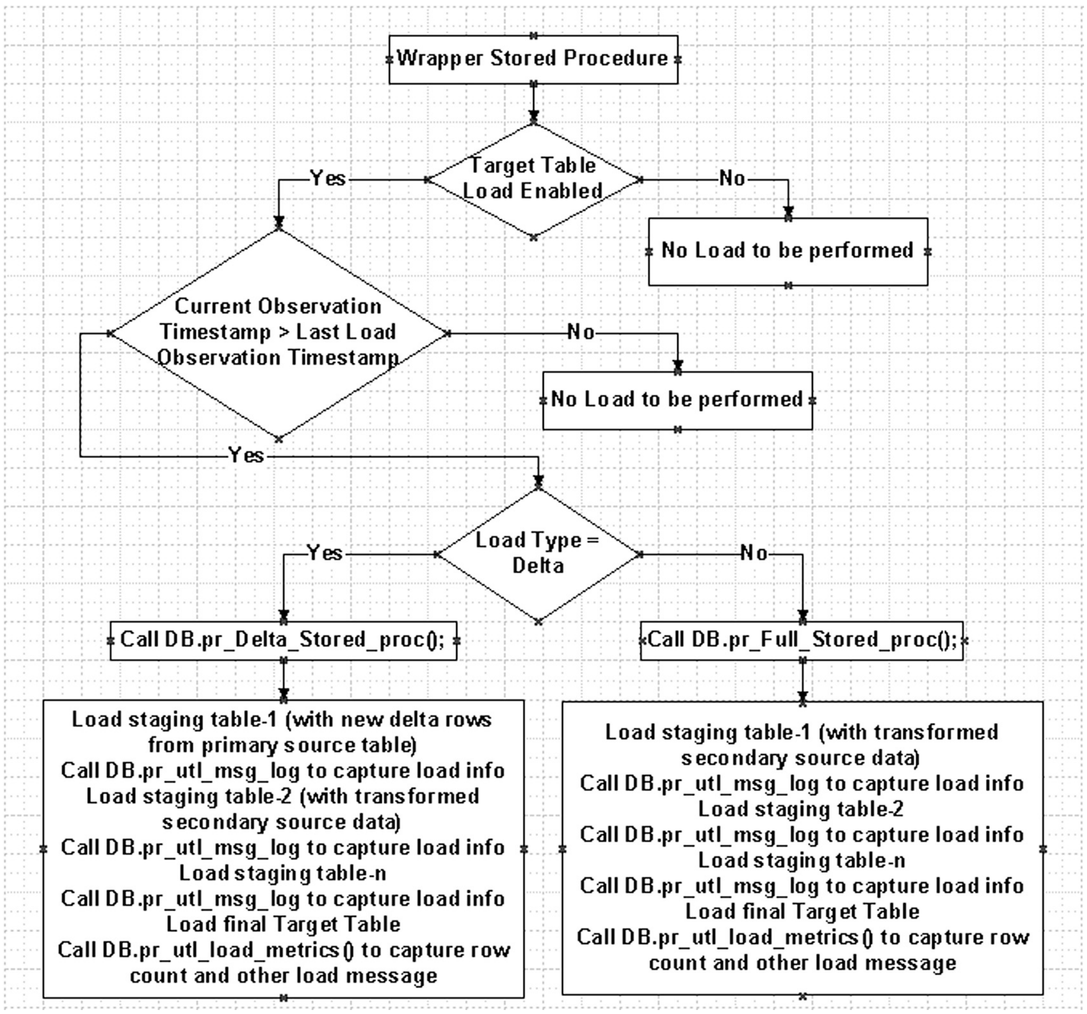

Wrapper Stored Procedure
Target Table Load Enabled
Yes
No
No Load to be performed
Current Observation Timestamp > Last Load Observation Timestamp
No
No Load to be performed
Yes
Load Type = Delta
Yes
No
Call DB.pr_Delta_Stored_proc();
Call DB.pr_Full_Stored_proc();
Load staging table-1 (with new delta rows from primary source table)
Call DB.pr_utl_msg_log to capture load info
Load staging table-2 (with transformed secondary source data)
Call DB.pr_utl_msg_log to capture load info
Load staging table-n
Call DB.pr_utl_msg_log to capture load info
Load final Target Table
Call DB.pr_utl_load_metrics() to capture row count and other load message
Load staging table-1 (with transformed secondary source data)
Call DB.pr_utl_msg_log to capture load info
Load staging table-2
Call DB.pr_utl_msg_log to capture load info
Load staging table-n
Call DB.pr_utl_msg_log to capture load info
Load final Target Table
Call DB.pr_utl_load_metrics() to capture row count and other load message

Figure 3. High level software architecture for full and delta load.

```
REPLACE PROCEDURE Procurement_DRV_MET.pr_Wpurch_ord_line()

BEGIN
DECLARE subj_ara VARCHAR(30) DEFAULT 'Procurement_DRV';
DECLARE trgt_tbl VARCHAR(30) DEFAULT 'purch_ord_line';
DECLARE load_type CHAR(5) DEFAULT 'Full';
DECLARE cycle_dt DATE;
DECLARE cycle_ts, src_cur_ts, src_lst_ts, DECLARE last_cycle_ts TIMESTAMP(0);
DECLARE trgt_row_exists,row_exists,load_thrhld,msg_nbr INTEGER DEFAULT 0;
DECLARE forc_load,load_enable,copy_back CHAR(1) DEFAULT 'Y';
DECLARE msg_txt1, msg_txt2 VARCHAR(100) DEFAULT ' ';
DECLARE dummy_msg1,dummy_msg2 VARCHAR(20) DEFAULT ' ';


_ *****************************************************************************************
** Get the time and date for current cycle refresh from the DWmgr_Xfrm_MET.v_cyc_log table.
CALL DWmgr_Xfrm_MET.pr_utl_Get_Cycle_TS(:cycle_dt,:cycle_ts,:subj_ara ) ;


_ *****************************************************************************************
** First find out how many source tables have changed for this target table since last cycle refresh
CALL DWmgr_Xfrm_MET.pr_utl_chk_tbl_changed(:subj_ara,:trgt_tbl,:src_cur_ts,:src_lst_ts,:trgt_row_exists);


_ *****************************************************************************************


** Get load parameters from the DWmgr_Xfrm_MET.v_load_chk_log table.
** The entries in this table will be used to decide the processing strategy for this object.
CALL DWmgr_Xfrm_MET.pr_utl_Get_LoadCheck_Info(:subj_ara,:trgt_tbl,:row_exists,:forc_load,:load_type,
                :load_enable,:load_thrhld,:copy_back,:last_cycle_ts,:src_cur_ts,:src_lst_ts,:trgt_row_exists);


_ *****************************************************************************************
** Next choose the load procedure execution strategy
** Strategy could be one of the following Full, Delta, or Skip the load if there are no changes in source tables.
** Note: some of the parameters are inout, and may be changed by the stored procedure during the call.
CALL DWmgr_Xfrm_MET.pr_utl_choose_load_strat(:subj_ara,:trgt_tbl,:forc_load,:load_type,:load_enable,:load_thrhld);

** Here we execute the main data stored proc: delta or full

IF (load_type = 'Full' AND load_enable = 'Y') THEN

    CALL appl_Procurement_DRV_01.pr_Fpurch_ord_line() ;

ELSEIF (load_type = 'Delta' AND load_enable = 'Y') THEN

    CALL appl_Procurement_DRV_01.pr_Dpurch_ord_line() ;

** This next line is here so that the SP will compile if there is not a call
** To the delta SP (many tables do not have deltas), and the SQL

SET dummy_msg1 = 'Running Delta';

ELSEIF (load_type = 'S' AND load_enable = 'S') THEN

SET dummy_msg2 = 'Skipping the load';

ELSE

** Here we insert an entry to the MessageLog table
CALL DWmgr_Xfrm_MET.pr_utl_write_msg_log(:subj_ara,:trgt_tbl,51,'Wrapper Stored Procedure','No load performed',
        'Check values for forc_load_ind, load_type_cd or load_enable_ind in load_chk_log table');


END IF;
_ *****************************************************************************************
** Here we Update the DWmgr_Xfrm_MET.v_load_chk_log and
** DWmgr_Xfrm_MET.v_src_load_log tables with new timestamp as the load is successful
CALL DWmgr_Xfrm_MET.pr_utl_upd_LoadCheck(:subj_ara,:trgt_tbl,:last_cycle_ts,:load_type,:load_enable,:src_cur_ts);


_ *****************************************************************************************
END;
```

Figure 4. Wrapper stored procedure template for full or incremental load.

The next utility procedure (Figure 4, pr_utl_choose_load_strat) in the wrapper is used to pass out values mainly for load_type and load_enable to the wrapper stored procedures. Based on that information, the wrapper procedure executes 'Full' or 'Delta' load stored procedures to load the target table. It might also skip the target table load if source tables have no new records. This procedure also captures errors and messages regarding load-skip, load-disabled, bad parameter values, etc (if any of these are detected) into the msg_log (Table 1: #9) and load_mtric (Table 1: #12) tables.

If the wrapper procedure decides to execute the full load stored procedure, the target table is emptied before being re-loaded. In many cases, the target table load consists of data pulls from multiple source tables and via several insert/select SQL blocks. The full stored procedure uses a utility procedure to capture load information into the metadata table. The utility procedure pr_utl_write_ent_msg_log is used to capture source table, timestamp, and row count information into the message log table (Table 1: #9) followed by each SQL block execution in the full procedure. This information is extremely helpful because it enables an ETL developer or production support person to easily and quickly determine which particular SQL block failed. This information is also useful to the investigation of any data issue in the target table.

When the wrapper procedure decides to execute the delta stored procedure for an incremental load (for instance, if the delta load threshold is exceeded or if the target table is empty, for some reason), certain steps are followed. A delta refresh is performed when small amount of new data (normally, less than 20% of the target rows) arrives in the source table during each cycle refresh [27]. A delta stored procedure performs two specific operations against the target table: one is an update against the target table to update any non-key columns with changed data from the source and the other is the insertion of any brand new records from the source.

There are several other conditions that affect load decisions [27]: (i) if the delta count exceeds a certain threshold percentage of target table row counts, then the load condition is switched to full refresh. The delta refresh in a populated table is slower because transient journaling is needed for a table that already contains some data. The delta load requires data manipulation language (DML) operations such as 'delete' and 'update' which causes more resources consumption. Hence, if a larger number of new rows arrive from source, it is more efficient to do a full refresh than an incremental refresh. Normally, a full refresh is performed when the target table needs to be loaded with more than one million rows (ii) If for some reason the target table is found to be empty, a full refresh is needed and the delta stored procedure will call the full load stored procedure (iii) If the delta count is within a predetermined threshold percentage, then the delta load will be performed. Also, delta refreshes are performed when full refreshes perform badly and can't be practically optimized any further.

Once the target table load is successful using full or delta stored procedures, the wrapper procedure calls one last utility procedure to update the metadata table to capture the observation timestamp for each source table against the target table. Thus, under an ETL metadata model, the cycle refresh is done completely based on ETL metadata information. The wrapper procedure and its associated utility procedure, the full and delta load procedures and associated utility procedures as well as metadata tables, provide ETL developers across the enterprise with a complete set of objects for extract-transform-load. It is important to mention that the stored procedures and tables are DBMS (database management system) -based as opposed to any commercial ETL tools. This ETL metadata model and associated utility procedures are useful to those data warehouses that do not use any ETL tool. This metadata model is independent of any commercial DBMS system.

5. Using Metadata in Load Process

The initial refresh of analytical tables is a full load. Subsequent refreshes are done via incremental load. During a full refresh, the existing data is truncated and a new copy of all rows of data is reloaded into the target table from the source. An incremental refresh only loads the delta changes that have occurred since last time the target table was loaded. In incremental refresh, only the delta or difference between target and source data is loaded at regular intervals. For incremental load, the observation timestamp for the previous delta load has to be maintained in an ETL metadata model. The data warehouse refresh is performed via batch cycles, so that the degree of information in a data warehouse is "predictable" [4].

In real world situations, there is a tendency by ETL programmers to design delta stored procedures to run 'stand-alone' - that is, without taking advantage of an ETL metadata model. We suggest design techniques utilizing a metadata model. The assumption is that instead of relying on conventional ETL tools, the full and incremental loads could be done via database specific stored procedures and with the help of a metadata model.

From the data warehouse side, updating large tables and related structures (such as indexes, materialized views and other integrated components) presents problems executing OLAP query workloads simultaneously with continuous data integration [28]. Our methodology minimizes the processing time and systems resources required for these update processes, as stored procedures give ample opportunity to manipulate the SQL blocks and influence the optimizer to execute queries with parallel efficiency.

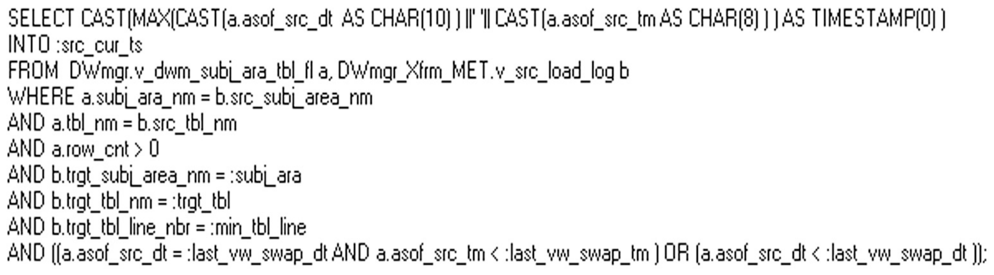
5.1. Checking Source Data Change

The wrapper procedure first calls a utility procedure to get the last source table load timestamp from the metadata table. It also checks the target table last load observation timestamp. By comparing both timestamps it becomes aware whether the source data has changed.

In Figure 5, the code block gets the last load timestamp of each of the source tables (primary, secondary, dimension, lookup, etc.) under one or several different business subject areas that are referenced in the stored procedures to load the target table. Based on that information, the source table load timestamps are pulled.

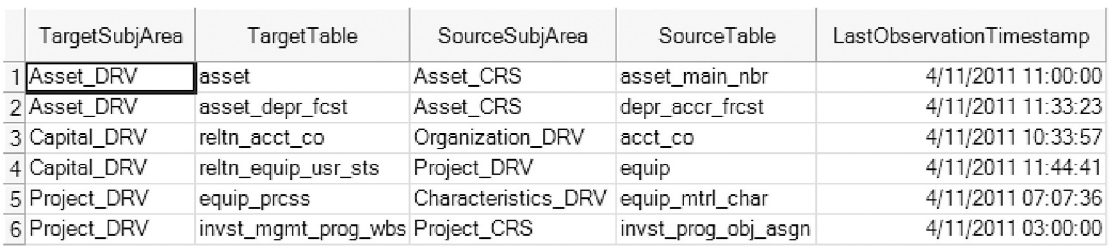
The utility stored procedure also pulls the last load timestamp for the last target table. If any of the source table load timestamps are found to be greater than the corresponding target table last load timestamps, that means new data has arrived in some source tables and a full or delta stored procedure should be executed (depending on load type code) to load the target table.

If the source load-timestamp is equal to last load timestamp of the target table, that means source data has not changed; therefore the table load is skipped and the stored procedure execution is bypassed. If the source load current timestamp is greater than the target table last load timestamp, an attempt is made to load the target table via full or delta stored procedure depending on the load indicator information in the metadata table. The default load is delta if no indicator value is provided.



```
SELECT CAST(MAX(CAST(a.asof_src_dt AS CHAR(10) ) ||' '|| CAST(a.asof_src_tm AS CHAR(8) ) ) AS TIMESTAMP(0) )
INTO :src_cur_ts
FROM  DWmgr.v_dwm_subj_ara_tbl_fl a, DWmgr_Xfrm_MET.v_src_load_log b
WHERE a.subj_ara_nm = b.src_subj_area_nm
AND a.tbl_nm = b.src_tbl_nm
AND a.row_cnt > 0
AND b.trgt_subj_area_nm = :subj_ara
AND b.trgt_tbl_nm = :trgt_tbl
AND b.trgt_tbl_line_nbr = :min_tbl_line
AND ((a.asof_src_dt = :last_vw_swap_dt AND a.asof_src_tm < :last_vw_swap_tm ) OR (a.asof_src_dt < :last_vw_swap_dt ));
```

Figure 5. Code block that detects source data change.



	TargetSubjArea	TargetTable	SourceSubjArea	SourceTable	LastObservationTimestamp
1	Asset_DRV	asset	Asset_CRS	asset_main_nbr	4/11/2011 11:00:00
2	Asset_DRV	asset_depr_fcst	Asset_CRS	depr_accr_frcst	4/11/2011 11:33:23
3	Capital_DRV	reltn_acct_co	Organization_DRV	acct_co	4/11/2011 10:33:57
4	Capital_DRV	reltn_equip_usr_sts	Project_DRV	equip	4/11/2011 11:44:41
5	Project_DRV	equip_prcss	Characteristics_DRV	equip_mtrl_char	4/11/2011 07:07:36
6	Project_DRV	invst_mgmt_prog_wbs	Project_CRS	invst_prog_obj_asgn	4/11/2011 03:00:00

Table 2. Last observation timestamp for target table refresh.

5.2. Getting the Parameters to Determine Load Strategy

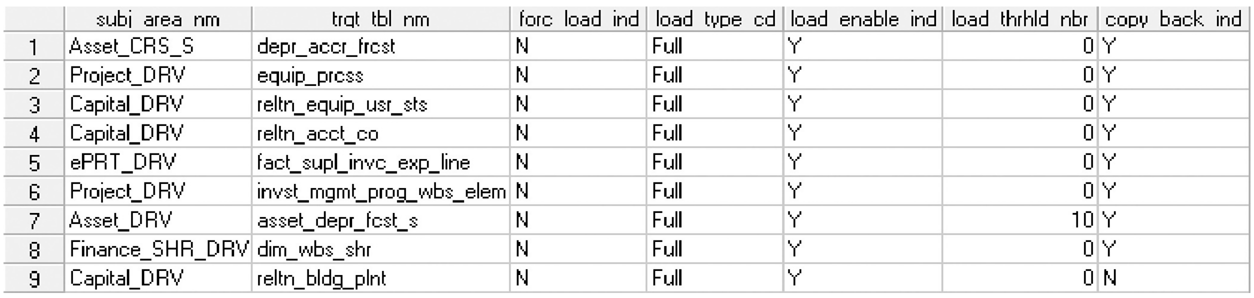
The wrapper stored procedure calls a utility stored procedure to get the target table load parameters from the 'load_chk_log' table. For each individual target table load within a subject area, a default metadata entry consisting of load conditions is inserted (Table 3) by this utility procedure during the first cycle refresh (one time entry). The default entry consists of several conditions such as 'force load indicator' ('Y' or 'N'; default 'N'), 'load type code' ('Full' or 'Delta'; default 'Full'), 'load enabled indicator' ('Y' or 'N'; default 'Y'), 'delta load threshold' (10% - 20%; default 10%), 'collect statistics' ('Y' or 'N'; default 'N'), and 'copy-back indicators ('Y' or 'N'; default 'N'). These parameters are passed to another utility procedure to prepare for target table loading.

The column forc_load_ind holds the value of 'Y' or 'N.' A value of 'Y' would allow the job to load the target table, irrespective of whether a source tables data has changed or not. A value of 'N' would let the wrapper stored procedures determine if the source data had changed and thus to decide whether or not to load the target table. The load_type_cd field holds the value of 'Full' or 'Delta.' Based on this information, the wrapper stored procedure executes the full or delta stored procedure. The column load_enable_ind holds the value of 'Y' or 'N.' A value of 'Y' allows the job to load the target table. A value of 'N' would prevent the wrapper stored procedure from loading the target table. The load_thrhld_nbr field holds the threshold value for 'Delta' load. The wrapper procedure will first determine the load type. If it is 'Full', then a full load will be performed, no matter what the threshold value is. If it is 'Delta', then the delta procedure will kick-off. The delta procedure will first determine the source row count. If the source count is within the load threshold number, the delta load will be performed. If the row count is greater than the load threshold number, then it will turn on the full load procedure.

Based on the load conditions the wrapper procedure either executes a 'Full' or 'Delta' stored procedure to load the target table, or skips the target table-load if the source table has no new records. It also generates and logs load messages into 'msg_log' and 'load_mtric' tables regarding load type, load skip, load enabled or disabled, and any unknown parameter values.

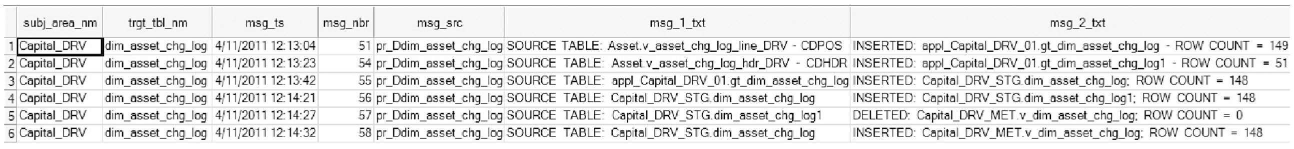
5.3. Capturing Error/ Message Logs while Loading

The table 'msg_log' stores troubleshooting information. A utility stored procedure is used to write error messages and information to the msg_log table during target table load. The utility procedure, pr_utl_write_msg_log() is called to capture error message.



	subj area nm	trgt tbl nm	forc load ind	load type cd	load enable ind	load thrhld nbr	copy back ind
1	Asset_CRS_S	depr_accr_frcst	N	Full	Y	0	Y
2	Project_DRV	equip_prcss	N	Full	Y	0	Y
3	Capital_DRV	reltn_equip_usr_sts	N	Full	Y	0	Y
4	Capital_DRV	reltn_acct_co	N	Full	Y	0	Y
5	ePRT_DRV	fact_supl_invc_exp_line	N	Full	Y	0	Y
6	Project_DRV	invst_mgmt_prog_wbs_elem	N	Full	Y	0	Y
7	Asset_DRV	asset_depr_fcst_s	N	Full	Y	10	Y
8	Finance_SHR_DRV	dim_wbs_shr	N	Full	Y	0	Y
9	Capital_DRV	reltn_bldg_plnt	N	Full	Y	0	N

Table 3. Load parameters used to determine load condition.



	subj_area_nm	trgt_tbl_nm	msg_ts	msg_nbr	msg_src	msg_1_txt	msg_2_txt
1	Capital_DRV	dim_asset_chg_log	4/11/2011 12:13:04	51	pr_Ddim_asset_chg_log	SOURCE TABLE: Asset.v_asset_chg_log_line_DRV - CDPOS	INSERTED: appl_Capital_DRV_01.gt_dim_asset_chg_log - ROW COUNT = 149
2	Capital_DRV	dim_asset_chg_log	4/11/2011 12:13:23	54	pr_Ddim_asset_chg_log	SOURCE TABLE: Asset.v_asset_chg_log_hdr_DRV - CDHDR	INSERTED: appl_Capital_DRV_01.gt_dim_asset_chg_log1 - ROW COUNT = 51
3	Capital_DRV	dim_asset_chg_log	4/11/2011 12:13:42	55	pr_Ddim_asset_chg_log	SOURCE TABLE: appl_Capital_DRV_01.gt_dim_asset_chg_log	INSERTED: Capital_DRV_STG.dim_asset_chg_log; ROW COUNT = 148
4	Capital_DRV	dim_asset_chg_log	4/11/2011 12:14:21	56	pr_Ddim_asset_chg_log	SOURCE TABLE: Capital_DRV_STG.dim_asset_chg_log	INSERTED: Capital_DRV_STG.dim_asset_chg_log1; ROW COUNT = 148
5	Capital_DRV	dim_asset_chg_log	4/11/2011 12:14:27	57	pr_Ddim_asset_chg_log	SOURCE TABLE: Capital_DRV_STG.dim_asset_chg_log1	DELETED: Capital_DRV_MET.v_dim_asset_chg_log; ROW COUNT = 0
6	Capital_DRV	dim_asset_chg_log	4/11/2011 12:14:32	58	pr_Ddim_asset_chg_log	SOURCE TABLE: Capital_DRV_STG.dim_asset_chg_log	INSERTED: Capital_DRV_MET.v_dim_asset_chg_log; ROW COUNT = 148

Table 4. Message log.

Once the cycle refresh is complete, the load metrics for the current cycle will be archived into the msg_log_hist table for future reference or investigation.

5.4. Capturing load metrics

The table 'load_mtric' (Table 1) stores vital load statistics (cyc_bgn_ts, load_strt_dt, load_strt_ts, load_stp_ts, load_row_cnt, and load_msg_txt, and runtime) for each target table. Statistics are also used for report outs of runtimes. This table provides information about whether the load is full or delta, if the load was disabled, and the reason for zero row count (in most cases). It enables the monitoring of a cycle while it is running.

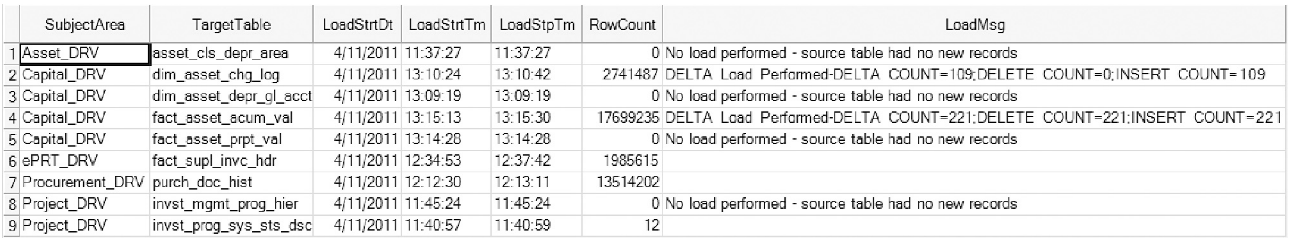
The utility stored procedure 'pr_utl_Load_Metrics()' is called from 'Full' and 'Delta' data load stored procedures to insert entries into the 'load_mtric' table during each target table load. Both full and delta stored procedures call this utility procedure to insert load metrics about each target table load into the 'load_mtric' metadata table. The row count of greater than zero and load timestamp are used as vital information for the downstream table refresh. This information is also useful to monitor cycle refreshes.

The full stored procedure contains transformation logic to first empty the target table before reloading the table with all data from the source tables. The delta refresh is done for the tables that are loaded with greater than one million rows or when full refreshes perform badly, and can't be practically optimized any further. The delta stored procedure first pulls the last observation timestamp of the target table from the metadata table and loads the target table with new delta records that have arrived after the target table was last refreshed. Once the cycle refresh is done the load metrics for the current cycle will be archived into load_mtric_hist table for future reference, investigation, and research.

5.5. Subsequent Cycle Run Behavior

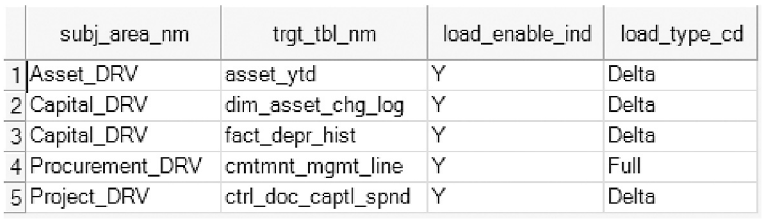
In order to automate the set-up of each target table's default load behavior, the metadata model needs to hold load parameters to be reset after each cycle refresh. The table, load_chk_opt (Figure 1: # 13) stores job information (subj_area_nm, trgt_tbl_nm, load_enable_ind, load_type_cd). For each job one entry is required to exist in this table. The entries are inserted into this table via a utility stored procedure call by wrapper stored procedures.

The load_type_cd field holds the value of 'Full' or 'Delta'. Based on this information, the post-load stored procedure will update the load_chk_log table to prepare it for the next cycle refresh. The column load_enable_ind holds the value of 'Y' or 'N.' Based on this info, the post-load stored procedure will update the load_chk_log



	SubjectArea	TargetTable	LoadStrtDt	LoadStrtTm	LoadStpTm	RowCount	LoadMsg
1	Asset_DRV	asset_cls_depr_area	4/11/2011	11:37:27	11:37:27	0	No load performed - source table had no new records
2	Capital_DRV	dim_asset_chg_log	4/11/2011	13:10:24	13:10:42	2741487	DELTA Load Performed-DELTA COUNT=109;DELETE COUNT=0;INSERT COUNT=109
3	Capital_DRV	dim_asset_depr_gl_acct	4/11/2011	13:09:19	13:09:19	0	No load performed - source table had no new records
4	Capital_DRV	fact_asset_acum_val	4/11/2011	13:15:13	13:15:30	17699235	DELTA Load Performed-DELTA COUNT=221;DELETE COUNT=221;INSERT COUNT=221
5	Capital_DRV	fact_asset_prpt_val	4/11/2011	13:14:28	13:14:28	0	No load performed - source table had no new records
6	ePRT_DRV	fact_supl_invc_hdr	4/11/2011	12:34:53	12:37:42	1985615	
7	Procurement_DRV	purch_doc_hist	4/11/2011	12:12:30	12:13:11	13514202	
8	Project_DRV	invst_mgmt_prog_hier	4/11/2011	11:45:24	11:45:24	0	No load performed - source table had no new records
9	Project_DRV	invst_prog_sys_sts_dsc	4/11/2011	11:40:57	11:40:59	12	

Table 5. Metrics showing load skipped since source data has not changed (sample rows).



	subj_area_nm	trgt_tbl_nm	load_enable_ind	load_type_cd
1	Asset_DRV	asset_ytd	Y	Delta
2	Capital_DRV	dim_asset_chg_log	Y	Delta
3	Capital_DRV	fact_depr_hist	Y	Delta
4	Procurement_DRV	cmtmnt_mgmt_line	Y	Full
5	Project_DRV	ctrl_doc_captl_spnd	Y	Delta

Table 6. Table with load type code to control load behavior.

table to prepare it for the next cycle refresh. Once the cycle refresh is done, the load_chk_log table will be updated based on information (load _type_cd, load_enable_ind) in the 'load_chk_opt' table via the post_load utility stored procedure. The next cycle refresh will occur based on updated information in the load_chk_table.

If a particular target table under a subject area needs to be loaded with 'Delta' refresh (by default 'Full' refresh occurs), one entry for that target table needs to be inserted into this table.

5.6. Load Configurations

To retain the cycle refresh information for a particular subject area in the metadata archive tables (cyc_log_hist, load_chk_log_hist, load_ mtric_hist and msg_log_hist), one entry against each _hist table must be inserted into the metadata table (Table 7) – 'load_config' (Figure 1: #5). This table is used for multiple purposes. For example, if we decide to retain history worth 90 days, entries need to be inserted into this table accordingly in order to enable the post-load procedure to retain records for the last 90 days and delete on a rolling basis any records that are older than 90 days.

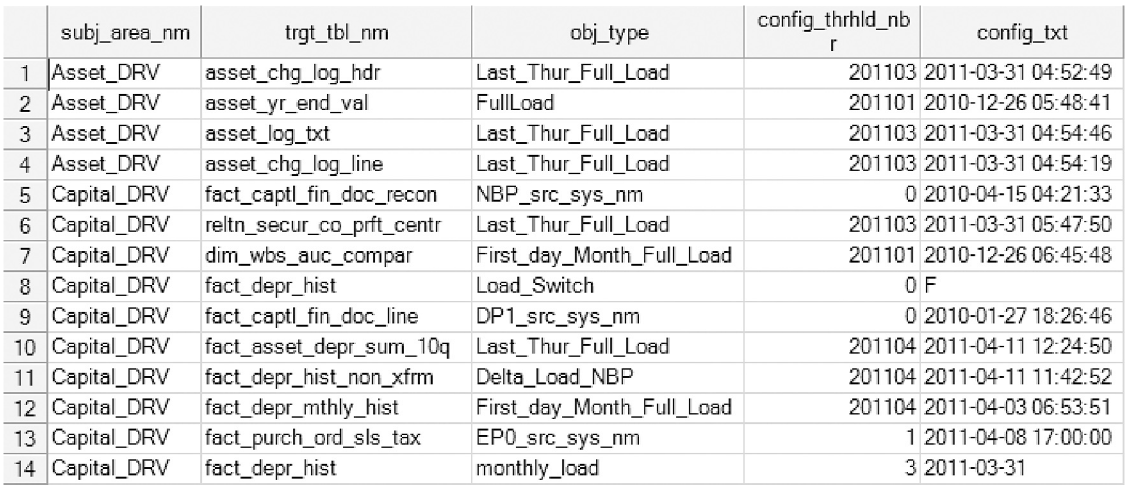
The table 'load_config' (subj_area_nm, trgt_tbl_ nm, obj_type, config_thrhld_nbr, config_txt) provides space to hold any staging data or input in order to prepare for a full or delta load. This is also used to trim the history tables on a rolling basis. The config_thrhld_nbr field holds the threshold value for history tables. The post-load procedure will delete least recent data from the history tables based threshold number.

The config_txt column will give ETL developers the opportunity to stage any value wanted in order to prepare for and load a target table. This field can also be used to prepare report outs of a particular table refresh, which can then be viewed by global report users via any web tool.

Table 7 shows information kept to track target table load conditions. For example, in a given subject area, certain tables could be loaded on the first day of month, while other tables could be loaded on the last day of month. Another set of tables could be loaded on a particular day of the last week of month. Another set of tables could be loaded once per day instead of several times per day. To enable this kind of load behavior, the load timestamp each of the target tables need to be tracked. This information could be used to control and determine the next load time.

6. Conclusions and Future Work

In this article, we showed that by using an ETL metadata model, data warehouse refreshes can be made standardized, while load behavior can be made predictable and efficient. The model can help identify source tables that do not have new data. Performing refreshes through a metadata-driven approach enables resource savings in the data warehouse. It helps avoid unnecessary loads when source data has not



	subj_area_nm	trgt_tbl_nm	obj_type	config_thrhld_nbr	config_txt
1	Asset_DRV	asset_chg_log_hdr	Last_Thur_Full_Load	201103	2011-03-31 04:52:49
2	Asset_DRV	asset_yr_end_val	FullLoad	201101	2010-12-26 05:48:41
3	Asset_DRV	asset_log_txt	Last_Thur_Full_Load	201103	2011-03-31 04:54:46
4	Asset_DRV	asset_chg_log_line	Last_Thur_Full_Load	201103	2011-03-31 04:54:19
5	Capital_DRV	fact_captl_fin_doc_recon	NBP_src_sys_nm	0	2010-04-15 04:21:33
6	Capital_DRV	reltn_secur_co_prft_centr	Last_Thur_Full_Load	201103	2011-03-31 05:47:50
7	Capital_DRV	dim_wbs_auc_compar	First_day_Month_Full_Load	201101	2010-12-26 06:45:48
8	Capital_DRV	fact_depr_hist	Load_Switch	0	F
9	Capital_DRV	fact_captl_fin_doc_line	DP1_src_sys_nm	0	2010-01-27 18:26:46
10	Capital_DRV	fact_asset_depr_sum_10q	Last_Thur_Full_Load	201104	2011-04-11 12:24:50
11	Capital_DRV	fact_depr_hist_non_xfrm	Delta_Load_NBP	201104	2011-04-11 11:42:52
12	Capital_DRV	fact_depr_mthly_hist	First_day_Month_Full_Load	201104	2011-04-03 06:53:51
13	Capital_DRV	fact_purch_ord_sls_tax	EP0_src_sys_nm	1	2011-04-08 17:00:00
14	Capital_DRV	fact_depr_hist	monthly_load	3	2011-03-31

Table 7. Table that keeps track of load conditions for different load types.

changed. The model captures all source observations timestamps as well as target table load timestamps. The model helps make it possible to run batch cycles unattended because it enables load behavior for each table for the next cycle refresh.

We presented an approach for refreshing data warehouses based on an ETL metadata model. Our approach has clear benefits as it allows for skipping table refreshes when source data has not changed. It helps to improve load performance in terms of response time and database system's CPU and IO resource consumption. The ETL metadata model, along with utility procedure and wrapper procedures, provides ETL programmers with a powerful tool for building a data warehouse by following certain rules. We further provided a template of the wrapper stored procedure.

Our ETL metadata model achieves several key objectives. It enables the capture of semantic metadata. It makes batch cycle refreshes metadata-driven. It helps to reset the load behavior of each table for the next cycle run. It enables load consistency across the data warehouse. It provides ETL programmers with a set of standard objects that makes their development work easier. The model provides the production support team with resources to monitor the data warehouse subject area refreshes. By introducing an ETL metadata model, we are able to automate data warehouse batch cycle refresh processes. Each of these achievements helps improve day-to-day maintenance of data warehouses. The use of this ETL metadata model in a production data warehouse environment reveals that the model runs efficiently in batch processing, cycle monitoring, and data warehouse maintenance. In a future endeavor we intend to develop macros to generate automated views as well as full and delta stored procedure templates with the help of this metadata model.

Acknowledgments

We would like to thank Julian Emmerson, formerly Senior Application Developer at Intel, for assisting and guiding us in designing the initial version of this metadata model. We are also grateful to Wally Haven, a Senior Application Developer at Intel, and the anonymous reviewers whose comments have improved the quality of the paper substantially.

References

[1] D. Agrawal, A. Abbadi, A. Singh, & T. Yurek, Efficient View Maintenance at Data Warehouse, *ACM SIGMOD Record*, 26(2), pp. 417–427, 1997.

[2] F. Bensberg, Controlling the Data Warehouse – a Balanced Scorecard Approach, in *Proceedings of the 25th International Conference on Information Technology Interfaces ITI 2003*, June 16–19, 2003, Cavtat, Croatia.

[3] S. Brobst, M. McIntire and E. Rado, (2008), Agile Data Warehousing with Integrated Sandboxing, *Business Intelligence Journal*, Vol. 13, No. 1, 2008.

[4] M. Chan, H. V. Leong and A. Si, Incremental Update to Aggregated Information for Data Warehouses over Internet, in *Proceedings of the 3rd ACM International Workshop on Data Warehousing and OLAP (DOLAP'00)*, November 2000, McLean, VA, USA.

[5] S. Chaudhuri and U. Dayal, An Overview of Data Warehousing and OLAP Technology, *SIGMOD Record*, 26(1), March 1997.

[6] I. Chengaiur-Smith, D. Baliou and H. Pazer, The Impact of Data Quality Information on Decision Making: An Exploratory Analysis, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, No. 6, November–December, 1999.

[7] T. Chenoweth, K. Corral and H. Demirkan, Seven Key Interventions for Data Warehouses Success, *Communications of the ACM*, January 2006, Vol. 49, No. 1.

[8] B. Czejdo, J. Eder, T. Morzy and R. Wrembel, Design of a Data Warehouse over Object-Oriented and Dynamically Evolving Data Sources, in *Proceedings of the 12th International Workshop on Database and Expert Systems Applications*, 2001.

[9] J. Eder, C. Koncilia and T. Morzy, The COMET Metamodel for Temporal Data Warehouses, in *Proceedings of the 14th International Conference on Advanced Information System Engineering (Caise 2002)*, Toronto, Canada, LNCS 2348, pp. 83–99.

[10] H. Fan and A. Poulovassilis, Using AutoMed Metadata in Data Warehousing Environments, in *Proceedings of the 6th ACM international workshop on Data warehousing and OLAP*, New Orleans, Louisiana, USA, pp. 86–93.

[11] N. Foshay, A. Mukherjee and A. Taylor, Does Data Warehouse End-User Metadata Add Value?, *Communications of the ACM*, November 2007, Vol. 50, No. 2.

[12] C. GARCÍA, Real Time Self-Maintainable Data Warehouse, in *Proceedings of the 44th Annual Southeast Regional Conference (ACM SE'06)*, March, 10–12, 2006, Melbourne, Florida, USA.

[13] S. R. GARDNER, Building the Data Warehouse, *Communications of the ACM*, 41(9).

[14] M. GOLFARELLI, D. MAIO AND S. RIZZI, The Dimensional Fact Model: a Conceptual Model for Data Warehouses, *International Journal of Cooperative Information Systems*, Vol. 7(2&3), pp. 215–247.

[15] J. H. HANSON AND M. J. WILLSHIRE, Modeling a faster data warehouse, *International Database Engineering and Applications Symposium (IDEAS 1997)*.

[16] T. N. HUYNH, O. MANGISENGI AND A. M. TJOA, Metadata for Object-Relational Data Warehouse, in *Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'2000)*, Stockholm, Sweden, June 5–6, 2000.

[17] W. H. INMON, *Building the Data Warehouse*, 3rd Edition, John Wiley, 2002.

[18] M. JENNINGS, *Use of Operational Meta Data in the Data Warehouse*, (2003), Last Accessed on 12/12/2011: `http://businessintelligence.com/article/13`

[19] T. JÖRG AND S. DESSLOCH, Towards Generating ETL Processes for Incremental Loading, in *Proceedings of the 2008 International Symposium on Database Engineering & Applications (IDEAS'08)*, September 10–12, 2008, Coimbra, Portugal.

[20] A. KARAKASIDIS, P. VASSILIADIS AND E. PITOURA, ETL Queues for Active Data Warehousing, in *Proceedings of the 2nd International Workshop on Information Quality in Information Systems, IQIS 2005*, Baltimore, MD, USA.

[21] N. KATIC, J. QUIRCHMAYR, S. M. STOLBA AND A. M. TJOA, A Prototype Model for Data Warehouse Security Based on Metadata, *IEEE Xplore*, 1998.

[22] T. KIM, J. KIM AND H. LEE, A Metadata-Oriented Methodology for Building Data Warehouse: A Medical Center Case, *Informs & Korms – 928*, Seoul 2000 (Korea).

[23] W. J. LABIO, R. YERNENI AND H. GARCIA-MOLINA, Shrinking the Warehouse Update Window, in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of data (SIGMOD'99)*, Philadelphia, PA.

[24] B. LIU, S. CHEN AND E. A. RUNDENSTEINER,Batch Data Warehouse Maintenance in Dynamic Environments, in *Proceedings of the CIKM'02*, pp. 68–75, 2002.

[25] S. LUJAN-MORA AND M. PALOMAR, Reducing Inconsistency in Integrating Data from Different Sources, *International Database Engineering & Applications Symposium (IDEAS '01)*, 2001.

[26] N. RAHMAN, Refreshing Data Warehouses with Near Real-Time Updates, *Journal of Computer Information Systems*, Vol. 47, Part 3, 2007, pp. 71–80.

[27] N. RAHMAN, P. W. BURKHARDT AND K. W. HIBRAY, Object Migration Tool for Data Warehouses, *International Journal of Strategic Information Technology and Applications (IJSITA)*, 2010, Vol. 1, Issue 4, 2010, pp. 55–73.

[28] R. J. SANTOS AND J. BERNARDINO, Real-Time Data Warehouse Loading Methodology, in *Proceedings of the 2008 International Symposium on Database Engineering & Applications (IDEAS'08)*, September 10–12, 2008, Coimbra, Portugal.

[29] A. SEN AND A. P. SINHA, A Comparison of Data Warehousing Methodologies, *Communications of the ACM*, Vol. 48, Issue 3, March 2005.

[30] G. SHANKARANARAYANAN AND A. EVEN, The Metadata Enigma, *Communications of the ACM*, Vol. 49, Issue 2, 2006, pp. 88–94.

[31] B. SHIN, An Exploratory Investigation of System Success Factors in Data Warehousing, *Journal of the Association for Information Systems*, Vol. 4, 2003.

[32] A. SIMITSIS, P. VASSILIADIS AND T. SELLIS, Optimizing ETL Processes in Data Warehouses, in *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, 5–8 April 2005, Tokyo, Japan.

[33] T. STOHR, R. MULLER AND E. RAHM, An Integrative and Uniform Model for Metadata Management in Data Warehousing Environments, in *Proceedings for the International Workshop on Design and Management of Data Warehouses (DMDW'99)*, Heidelberg, Germany, June 14–16, 1999.

[34] T. VETTERLI, A. VADUVA AND M. STAUDT, Metadata Standards for Data Warehousing: Open Information Model vs. Common Warehouse Metadata, *ACM SIGMOD Record*, Vol. 29, Issue 3, September 2000.

[35] C. WHITE, *Data Integration: Using ETL, EAI, and EII Tools to Create an Integrated Enterprise*, The Data Warehousing Institute, October, 2005.

[36] J. WIDOM, Research Problems in Data Warehousing, in *Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM'95)*, November 1995, Baltimore, MD, USA.

[37] Y. ZHUGE, J. L. WIENER AND H. GARCIA-MOLINA, Multiple View Consistency for Data Warehousing, in *Proceedings of the Thirteenth International Conference on Data Engineering*, April 7–11, 1997, Birmingham, U.K.

[38] J. V. ZYL, M. VINCENT AND M. MOHANIA, Representation of Metadata in a Data Warehouses, in *Proceedings of IEEE'98*, IEEE Xplore, 1998.

Received: March, 2012
Revised: May, 2012
Accepted: May, 2012

Contact addresses:
Nayem Rahman
IT Business Intelligence (BI)
Intel Corporation
Mail Stop: AL3-85
5200 NE Elam Young Pkwy
Hillsboro, OR 97124-6497, USA
e-mail: nayem.rahman@intel.com

Jessica Marz
Intel Software Quality
Intel Corporation
RNB – Robert Noyce Building
2200 Mission College Blvd,
Santa Clara, CA 95054, USA
e-mail: jessica.marz@intel.com

Shameem Akhter
Department of Computer Science
Western Oregon University
345 N. Monmouth Ave.
Monmouth, OR 97361, USA
e-mail: sakhter09@mail.wou.edu

NAYEM RAHMAN is a Senior Application Developer in IT Business Intelligence (BI), Intel Corporation. He has implemented several large projects using data warehousing technology for Intel's mission critical enterprise DSS platforms and solutions. He holds an MBA in Management Information Systems (MIS), Project Management, and Marketing from Wright State University, Ohio, USA. He is a Teradata Certified Master. He is also an Oracle Certified Developer and DBA. His most recent publications appeared in the International Journal of Strategic Information Technology and Applications (IJSITA) and the International Journal of Technology Management & Sustainable Development (IJTMSD). His principal research areas are active data warehousing, changed data capture and management in temporal data warehouses, change management and process improvement for data warehousing projects, decision support system, data mining for business analysts, and sustainability of information technology.

JESSICA MARZ has worked in Intel's IT division Enterprise Data Warehouse Engineering Group as both a Business Intelligence application developer and program manager. She holds a BA in English from UCLA, an MBA from San Jose State University, and a JD from Santa Clara University.

SHAMEEM AKHTER has a Master of Social Science (MSS) degree in Economics from the University of Dhaka, Bangladesh. Currently, she is pursuing her MS. in Management and Information Systems at Western Oregon University, USA. Her most recent publications on IT sustainability and data warehousing appeared in the International Journal of Technology Management & Sustainable Development (IJTMSD) and International Journal of Business Intelligence Research (IJBIR) respectively. Her research interest includes database systems, data warehousing, decision support system, and information systems implementation.