The Journal of the Southern Association for Information Systems

Volume 2 | Issue 1 Article 2

6-30-2014

Temporal Data Update Methodologies for Data Warehousing

Nayem Rahman
Intel Corporation, nayem.rahman@intel.com

TEMPORAL DATA UPDATE METHODOLOGIES FOR DATA WAREHOUSING

Nayem Rahman
Intel Corporation
nayem.rahman@intel.com

ABSTRACT

Data warehouse applications and business intelligence (BI) communities have been increasingly feeling the need to collect temporal data for accountability and traceability reasons. This necessitates building capabilities to capture and view transaction lineage in data warehouses based on timestamps. Data as of a point in time can be obtained across multiple tables or multiple subject areas, resolving consistency and synchronization issues. In this article we discuss implementations such as temporal data update methodologies, coexistence of load and query against the same table, performance of load and report queries, and maintenance of views on top of the tables with temporal data. We show how to pull data from heterogeneous sources and how to perform join operations with inequality predicates with more than one source table and then load them in analytical subject area tables maintaining transaction lineage. We propose several business views based on timestamp filters for use by different applications.

Keywords

Temporal Data, Data Warehouse, Row-Effective Timestamp, Row-Expired Timestamp, Data Update Methodology, Transaction Lineage.

INTRODUCTION

In a competitive business environment, successful businesses are data driven. A data warehouse architecture selection is founded on business needs (Ariyachandrs and Watson 2010). The business executives would want to make strategic as well as tactical business decisions (Brobst et al. 2008) with accurate information at the right time. The accuracy of information is dependent on detailed data as well as time-varying data. The data warehouse with time-varying data is instrumental in strategic decision making. The business requirements for temporal data go beyond what is typical of conventional database implementation.

Customer transactions keep changing over time with changing customer behavior patterns (Apeh and Gabrys 2013). Temporal data is concerned with time-varying data. Time-varying data states that each version of a record is relevant to some moment in time (Inmon et al. 2001; Martin and Abello 2003; and Snodgrass 2010). The temporal aspects normally consist of valid-time and transaction-time. Valid time defines the time period when a particular tuple is true in modeled reality, while the transaction time defines the time period when that particular tuple is captured in the database (Martin and Abello 2003; and Torp et al. 2000).

A temporal data warehouse is significantly different from an operational database in many respects (Shin 2003). Operational source systems are usually non-temporal and maintain only current state of data as opposed to complete history of data (Bruckner and Tjoa 2002; and Rahman 2008a) with transaction lineage. Data warehouses are always maintained to hold large volumes of historical data.

Data management and warehousing is considered the foundation of business intelligence (BI) and analytics (Chen et al. 2012). During the last decade data warehousing has achieved prominence. Scattered databases and data-marts are being consolidated into more useful data warehouses. The advent of new information technologies and techniques such as temporal data warehousing presents unique opportunities for firms to enhance their customer agility (Roberts and Grover 2012). This also speaks for maturity of data warehousing technologies (Sen et al. 2006). Temporal data warehousing has gained prominence among different stakeholders including suppliers, business users, and researchers because of user popularity and management patronage (Jensen 2000).

"A temporal data warehouse is a repository of historical information, originating from multiple, autonomous, (sometimes) heterogeneous and non-temporal sources. It is available for queries and analysis (such as data mining) not only to users interested in current information but also to those interested in researching past information to identify relevant trends (Amo and Alves 2000)."

W.H. Inmon defines temporal data warehouse as "a collection of integrated, subject-oriented databases designed to support the DSS function, where each unit of data is relevant to some moment in time. The data warehouse contains atomic data and lightly summarized data (Inmon 2002)." In this definition time-varying means the possibility to keep different values of the same record according to its changes over time (Malinowski and Zimányi 2006).

Temporal data warehouses provide a history of serialized changes to data identified by times when changes occurred (Golfarelli and Rizzi 2009). This allows for querying the current state as well as past states of a record (Fegaras and Elmasri 1998). Conventional databases provide users only current state of data which is true as of a single point in time (Ozsoyoglu and Snodgrass 1995). Users of a data warehouse are not only interested in the current state of data, but also in the transaction lineage as to how a particular record has evolved over time (Bruckner and Tjoa 2002). A record inserted in a database is never physically deleted (Chountas et al. 2004). A new record or a new version of an existing record is always added to reflect a transaction lineage for that data. Thus an evolving history of data is maintained in the temporal data warehouse.

Temporal data has important applications in many domains (Jensen 1999; Jestes 2012). Most of those domains applications can benefit from a temporal data warehouse (Thomas and Datta 2001; and Yang and Widom 1998) such as banking, retail sales, financial services, medical records, inventory management, telecommunications, and reservation systems. In the case of a bank account, an account holder's balance will change after each transaction. The amount or descriptions of a financial document will change for business purposes. Such data is often valuable to different stakeholders and should be stored in both current state and all previously current states.

Although there are clear benefits and demand for temporal database management systems (DBMS), there are only a few commercially available (Snodgrass 2010; and Torp 1998). Most of the current commercial databases are non-temporal and hence, they do not provide a special temporal query language, a temporal data definition language, or a temporal manipulation language (Bellatreche and Wrembel 2013; Kaufmann 2013; Mkaouar et al. 2011; and TimeConsult 2013).

In the absence of a temporal DBMS, we argue that an effort should be made to take advantage of current commercial databases and allow for handling multiple versions of data including past, current, and future states of data. This can be done with application coding for handling multiple versions of data. The current commercial relational databases with a high-level language such as SQL are mature enough to manage complex data transformations (Stonebraker et al. 2005) and also have performance improvement measures, such as various efficient algorithms for indexing. The improvements in the area of disk storage technology and declining cost of data storage (Chaudhuri et al. 2011) have also made it possible to efficiently store and manage temporal data with all transaction lineages (Ahn and Snodgrass 1986; and Torp 1998).

The temporal database implementations could be done by extending a non-temporal data model into a temporal data model and building temporal support into applications. Two timestamp fields need to be added to each table of the conventional data model. The new columns consist of 'row effective timestamp' and 'row expired timestamp' which hold date and time values to identify each individual row in terms of their present status such as past or current, or future.

The data warehouses are refreshed at a certain time intervals with data from different operational databases. In order to keep data warehouses run efficient and to maintain consistent data in the warehouse it is important that data arrive in the warehouse in a timely fashion and be loaded via batch cycle runs. Since data warehouse consists of thousands of tables in multiple different subject areas the table refreshes must be done in order of dependencies via batch cycles. Batch refreshes have proven to be an efficient method of loading from the standpoint of performance (Brobst et al. 2008) and data consistency. Another aspect of storing data in data warehouses is that

https://aisel.aisnet.org/jsais/vol2/iss1/2
DOI: 10.17705/3JSIS.00005

initially data is captured in staging subject areas (Ejaz and Kenneth 2004; and Hanson and Willshire 1997) with one to one relation between operational source and data warehouse staging area tables. Analytical subject areas are refreshed from the staging area tables. The analytical subject area refresh requires collecting data from more than one subject area or more than one table from a particular staging subject area.

The purpose of this article is to discuss implementations such as temporal data update methodologies, viewing of data consistently, coexistence of load and query against the same table, performance improvement of load and report queries, and maintenance of views. The intended result is a temporal data warehouse that can be used concurrently to load new data and allow various reporting applications to return results consistent with their selected time slice.

LITERATURE REVIEW

The data warehouse refreshes have been a research topic for more than a decade. The research is mostly related to storing and maintaining the current state of data. Current state of data fails to provide data lineage information. Discarding updates between two refresh points of time with periodic complete reloads leads to a loss of transaction lineage (Vavouras et al. 1999). Most previous work on data warehousing focused on design issues, data maintenance strategies in connection with relational view materialization (Huq et al. 2010) and implementation aspects (Chen et al. 2010; Kim et al. 2004; Samtani et al. 1998; and Widom 1995). There has been little research work done to date on capturing transaction lineage and the temporal view maintenance problem (Yang and Widom 1998) and most of the previous research ignores the temporal aspects of data warehousing (Bruckner and Tjoa 2002). There is a consensus that Information Systems research must respond to theoretical contributions and make attempt to solving the current and anticipated problems of practitioners (Sein et al. 2011). There is a need for coming up with mechanisms to store transaction lineage in conventional databases. Current commercial database systems provide little built-in capabilities to capture transaction lineage or to support query language for temporal data management (Mahmood et al. 2010). As of today, a few companies started providing time-referenced data storing functionality and SQL facilities in their DBMS system (Chau and Chittayasothorn 2008; and Snodgrass 2010). In data warehouses, data comes from many sources and data warehouse refreshes happen several times a day. Data warehouse is a shared environment and the data in it is typically used by so many applications. These applications may need a different time-slice of data. The data warehouses must cope with the temporal granularities of data (Terenziani 2012).

Temporal data warehouses raise many issues including consistent aggregation in presence of time-varying data, temporal queries of multidimensional data, storage method, and temporal view materialization (Bellatreche and Wrembel 2013; and Malinowski and Zimányi 2006). The temporal aggregation problem was studied in (Yang and Widom 2001; Yufei et al. 2004; and Zhang et al. 2001) to address the challenges of temporal data. Much research is now being done to improve the efficiency of range aggregate queries in a temporal data warehouse (Feng et al 2005). Kaufmann (2013) presents a native support of temporal features for in-memory analytics databases. Stonebraker (2011) suggests new SQL for business intelligence (BI) queries as they are so resource-heavy that they get in the way of timely responses to transactions. The new SQL for temporal data with timestamp-based versioning is also very much needed.

Wang et al. (2012) study the “problem of how to maintain temporal consistency of real-time data in distributed real-time systems.” Malinowski and Zimanyi (2008) provide a conceptual model for temporal data warehouses that “support for levels, attributes, hierarchies, and measures.” Chau and Chittayasothorn (2008) proposed a temporal object relational SQL language with attribute time-stamping – a superset of OR SQL language. Viqueira and Lorentzos (2007) propose an SQL extension for the management of spatio-temporal data. Mkaouar et al. (2011) study how to simplify querying and manipulating temporal facts in SQL by integrating time in a native manner. Li et al. (2010), Kvet and Matiasko (2013), and Jestes et al. (2012) provide insights in ranking large temporal data. Gupta et al. (2013) provide an overview of outlier detections for various forms of temporal data.

In this article, we focus on an innovative (Downes and Nunes 2013; and Ramiller and Swanson 2009) approach for dealing with transaction lineage and storing them with time-stamp granularities. We present methodologies for refreshing data warehouses with time-varying data via batch cycles. This is suitable for large data warehouses with hundreds of subject areas and thousands of tables where refreshes occur in a span of one to four-hour windows. We propose the use of conventional extract-transform-load (ETL) tools to extract data from source systems and load the staging subject areas in the data warehouse without performing any kind of transformation tasks. As soon as staging tables are refreshed, the data warehouse software performs transformations to insert new rows in the actual data warehouse (analytical subject areas) tables and also update the tables by applying row expired timestamps to the preexisting rows that correspond to the newly arrived rows. ETL represents the most important stage of the (temporal) data warehouse design as 70% of the risk and effort attributed to this stage (Berkani et al. 2013). We also examine the possibility of using metadata tables (Rahman et al. 2012) to recompile views based on subject area refresh timestamps. We show that there are opportunities to use different performance improvement features, such as indexing (Li et al. 2010) conventional commercial databases to load and query temporal data in the commercial non-temporal databases (Kaur et al. 2013) as these features are very important to handle large volume of transaction lineage data.

TEMPORAL DATA UPDATE METHODOLOGIES

During the past two decades, several dozen temporal data models have been generated, all with timestamps being included as integral components (Torp et al. 2000). There are very few (Chau and Chittayasothorn 2008; and Snodgrass 2010) commercial databases yet on the market, perhaps due to the complex nature of temporal data. This article presents a technical outline of how to use the conventional commercial databases to update with temporal data. The goal is to make sure data consistency is maintained, and load and query performance is not compromised. Updating data warehouses with temporal data is a mechanism for storing the lineage of data in the tables. It captures all changes made to a data row over time (transaction lineage). Deleted rows are not physically deleted; they are labeled to exhibit expiration instead. Updated rows are handled by expiring the existing rows and inserting the new version of the rows. Both current and historical time slices are available to any user by manipulating view filter criteria with less-than-equal-to (<=) predicates, because each version of a row share the same key (Ahn and Snodgrass 1986). Temporal data must be evaluated by joins with inequality predicates; rather than equality predicates used in conventional data evaluation techniques (Gao et al. 2005).

A temporal data warehouse provides a consistent view of data for customer queries while data is being loaded into the same table being queried. It provides transaction lineage of data and mechanisms to harmonize (aggregate and synchronize) data based on time slicing. The business needs for several different time-varying data can be met using a temporal data warehouse.

Figure 1 presents a temporal data warehousing environment. The data comes from operational databases. The source systems provide change timestamp for temporal data. In a data warehouse source data is initially landed in staging subject areas. The data is then moved to downstream layers of data warehouse such as layers 2 (L2) and 3 (L3). L2 subject areas are shared subject areas used by any application(s) that needs data. An L3 subject area is dedicated to a specific application. Each L3 subject area provides an analytical environment used for reporting and business intelligence (BI) purposes.

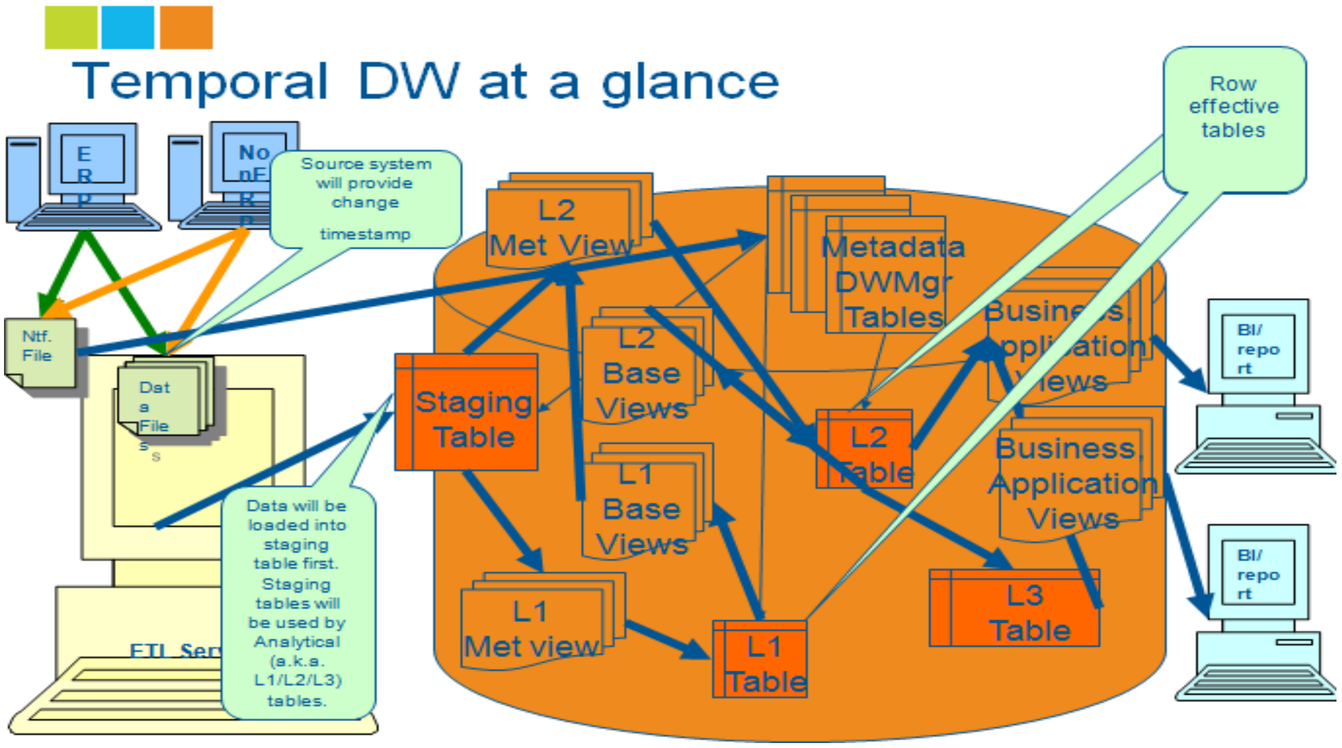
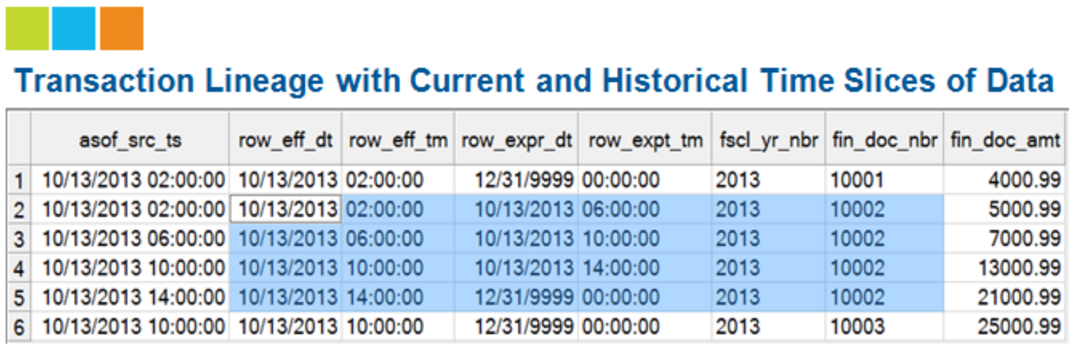

Temporal DW at a glance
ERP
NonERP
Source system will provide change timestamp
Ntf. File
Data Files
ETL Server
Data will be loaded into staging table first. Staging tables will be used by Analytical (a.k.a. L1/L2/L3) tables.
Staging Table
L2 Met View
L2 Base Views
L1 Base Views
L1 Met view
L1 Table
Metadata DWMgr Tables
L2 Table
Business Application Views
Business Application Views
L3 Table
Row effective tables
BI/ report
BI/ report

Figure 1: A Typical Temporal Data Warehousing Environment.

In order to implement temporal data update methodology the data model will have four additional columns, such as 'row effective date', 'row effective time', 'row expired date' and 'row expired time', to mark row effective date/time and row expired date/time against each row in the table. To make each row unique, the row effective date and time columns need to be part of primary key. The data from the operational databases will arrive in a timely fashion via flat files. The cycle refresh time intervals can be 30 minutes, one, two, three, or four hours, etc based on the needs of the business organization. The data manipulation (DML) code (I/U/D) and data row change timestamp are provided by the source system (Malinowski and Zimányi 2006) in the data files. The data row change timestamp will be used as row effective date and time. '9999-12-31 12:00:00' will be used as row expired timestamp however the presence of this high value indicates an 'active' or current row. Time stamping is used to tag each record with some moment in time when a record is created or passed from one environment to another (Inmon et al. 2001).

Immediately after staging tables are loaded from source system the data warehouse SQL will be used to process and expired the rows if multiple versions have arrived in a file. It is likely that during a cycle refresh a data file will contain multiple versions of rows for a particular record, with the same primary key. In that case, each previous version will be expired with the row effective timestamp of the immediate next version of that row. Only the current version will have the expired timestamp value '9999-12-31 12:00:00'. For example, if the same key has a DML code 'I' (insert) followed by 'U' (update) in that case only the row with 'U' will be treated as the active row. This will insure rows are expired in the staging table in case there are multiple versions of rows arriving via a source data file in a given cycle. Next the rows with both 'U' and 'I' will be inserted in the final target table. Following insert into the target table all rows with 'D' (delete) in the staging table will be used to expire the corresponding row in the target table. 'D' rows are not deleted physically in the target table. All the existing rows with DML code 'U' will be expired in the target table and new rows inserted. All these steps are used to perform incremental refreshes (Rahman 2010) with temporal data. In case of full refresh as initial load, the current version of rows will be used to perform the load. The change date of current row will be used as row effective timestamp and '9999-12-31 12:00:00' as expired timestamp.

Table-1 shows how data look in the final target table in the data warehouse:



Transaction Lineage with Current and Historical Time Slices of Data

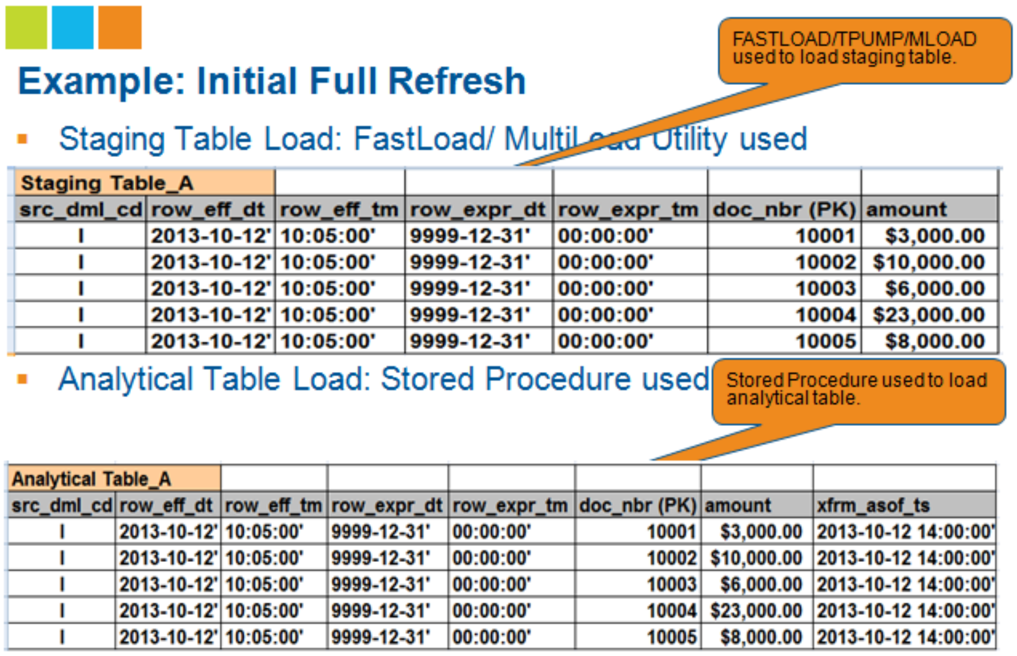
	asof_src_ts	row_eff_dt	row_eff_tm	row_expr_dt	row_expt_tm	fscl_yr_nbr	fin_doc_nbr	fin_doc_amt
1	10/13/2013 02:00:00	10/13/2013	02:00:00	12/31/9999	00:00:00	2013	10001	4000.99
2	10/13/2013 02:00:00	10/13/2013	02:00:00	10/13/2013	06:00:00	2013	10002	5000.99
3	10/13/2013 06:00:00	10/13/2013	06:00:00	10/13/2013	10:00:00	2013	10002	7000.99
4	10/13/2013 10:00:00	10/13/2013	10:00:00	10/13/2013	14:00:00	2013	10002	13000.99
5	10/13/2013 14:00:00	10/13/2013	14:00:00	12/31/9999	00:00:00	2013	10002	21000.99
6	10/13/2013 10:00:00	10/13/2013	10:00:00	12/31/9999	00:00:00	2013	10003	25000.99

Table 1: Current and historical time slices of data for the same record.

Among the highlighted rows (with the same key: fin_doc_nbr & fscl_yr_nbr) in Table 1, the last row is the current row which is active with row expired date and time as '9999-12-31 12:00:00'.

Process of Loading Derived Tables

In data warehouses, quite often derived tables are created for analytical purposes. These tables are loaded by pulling data from multiple tables and doing various aggregations, summations, and other computations. The response time of standard repetitive queries can be improved significantly if the answers of complex reporting query are stored in a simple table with keyed access (Gardner 1998). These table structures are created in such a way that they fulfill the reporting needs of different business applications. The report tools will point to these tables via simple (SELECT *) views. When loading the derived tables, by pulling data from primary source tables and dimension or header tables, row effective timestamp ranges need to be generated for dimension/ header/ secondary source table to make a relationship between the row effective date and the time on primary/ fact table rows.



Example: Initial Full Refresh

FASTLOAD/TPUMP/MLOAD used to load staging table.

- Staging Table Load: FastLoad/ MultiLoad Utility used

Staging Table_A						
src_dml_cd	row_eff_dt	row_eff_tm	row_expr_dt	row_expr_tm	doc_nbr (PK)	amount
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10001	$3,000.00
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10002	$10,000.00
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10003	$6,000.00
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10004	$23,000.00
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10005	$8,000.00

- Analytical Table Load: Stored Procedure used

Stored Procedure used to load analytical table.

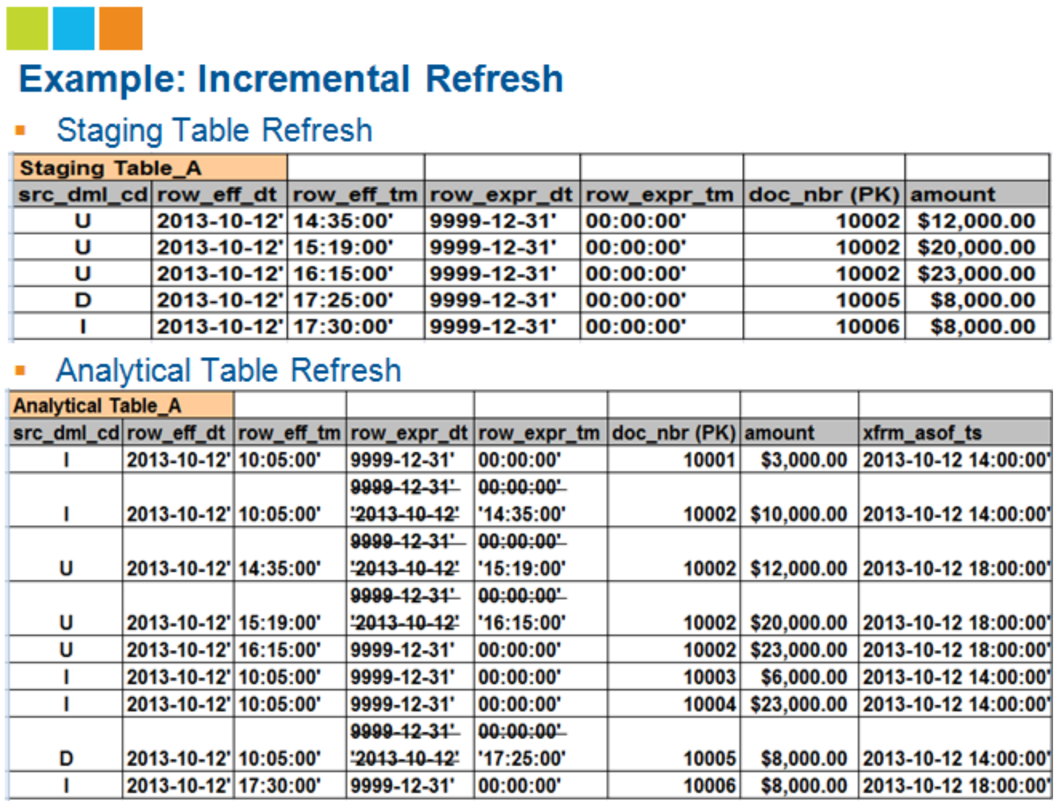
Analytical Table_A							
src_dml_cd	row_eff_dt	row_eff_tm	row_expr_dt	row_expr_tm	doc_nbr (PK)	amount	xfrm_asof_ts
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10001	$3,000.00	2013-10-12 14:00:00'
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10002	$10,000.00	2013-10-12 14:00:00'
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10003	$6,000.00	2013-10-12 14:00:00'
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10004	$23,000.00	2013-10-12 14:00:00'
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10005	$8,000.00	2013-10-12 14:00:00'

Figure 2: Table showing data after Initial Full Refresh.

https://aisel.aisnet.org/jsais/vol2/iss1/2
DOI: 10.17705/3JSIS.00005

Figure 2 is an example of loading staging and analytical subject area tables. Initial full refreshes in staging subject areas are performed using database utilities such as Fast-Load and Multi-Load. The downstream analytical subject areas are refreshed with data from staging tables. Since both source (staging) and target (analytical) tables reside in the same data warehouse database-specific software such as stored procedures and macros could be used to update the analytical subject areas.

The SQL in the form of stored procedures can be used to load data into derived tables. To do a full or incremental refresh, when joining between source primary and secondary or dimension tables or joining between line and header tables, the table join has to be based on primary key + row_eff_ts columns. As both primary and secondary/ dimension tables will hold transaction lineage it is important to make a one to one relationship for transactions lineage data using row_eff_ts columns. In this case the secondary/dimension table row_eff_ts must be less than or equal to the row_eff_ts column of primary source table. Note that the secondary/ dimension table data must come from the source at the same time or during a previous refresh in order to have the primary source table row effective timestamp match else data will be filtered out in loading the target table. This might happen when the primary source table and the secondary/ dimension tables is joined with an 'inner join' (instead of left outer) conditions.



Example: Incremental Refresh

- Staging Table Refresh

Staging Table_A						
src_dml_cd	row_eff_dt	row_eff_tm	row_expr_dt	row_expr_tm	doc_nbr (PK)	amount
U	2013-10-12'	14:35:00'	9999-12-31'	00:00:00'	10002	$12,000.00
U	2013-10-12'	15:19:00'	9999-12-31'	00:00:00'	10002	$20,000.00
U	2013-10-12'	16:15:00'	9999-12-31'	00:00:00'	10002	$23,000.00
D	2013-10-12'	17:25:00'	9999-12-31'	00:00:00'	10005	$8,000.00
I	2013-10-12'	17:30:00'	9999-12-31'	00:00:00'	10006	$8,000.00

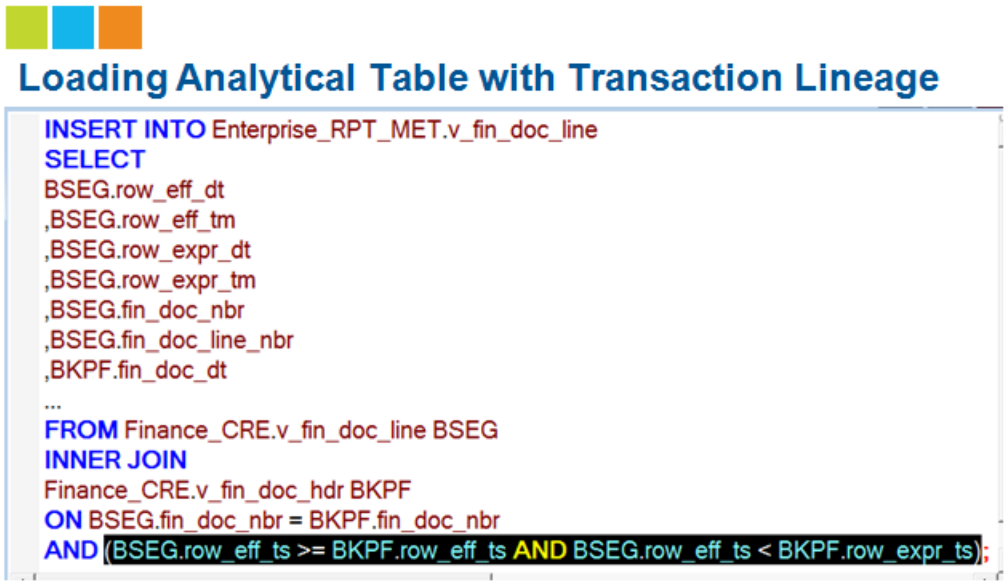
- Analytical Table Refresh

Analytical Table_A							
src_dml_cd	row_eff_dt	row_eff_tm	row_expr_dt	row_expr_tm	doc_nbr (PK)	amount	xfrm_asof_ts
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10001	$3,000.00	2013-10-12 14:00:00'
I	2013-10-12'	10:05:00'	~~9999-12-31'~~ ~~'2013-10-12'~~	~~00:00:00'~~ '14:35:00'	10002	$10,000.00	2013-10-12 14:00:00'
U	2013-10-12'	14:35:00'	~~9999-12-31'~~ ~~'2013-10-12'~~	~~00:00:00'~~ '15:19:00'	10002	$12,000.00	2013-10-12 18:00:00'
U	2013-10-12'	15:19:00'	~~9999-12-31'~~ ~~'2013-10-12'~~	~~00:00:00'~~ '16:15:00'	10002	$20,000.00	2013-10-12 18:00:00'
U	2013-10-12'	16:15:00'	9999-12-31'	00:00:00'	10002	$23,000.00	2013-10-12 18:00:00'
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10003	$6,000.00	2013-10-12 14:00:00'
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10004	$23,000.00	2013-10-12 14:00:00'
D	2013-10-12'	10:05:00'	~~9999-12-31'~~ ~~'2013-10-12'~~	~~00:00:00'~~ '17:25:00'	10005	$8,000.00	2013-10-12 14:00:00'
I	2013-10-12'	17:30:00'	9999-12-31'	00:00:00'	10006	$8,000.00	2013-10-12 18:00:00'

Figure 3: Table showing data after Incremental Refresh followed by Full Refresh.

Figure 3 is an example of a table update with incremental data (Rahman 2010). In Analytical Table-A, for doc_nbr 10002, we can see that every time changed data comes from source the most recent row for a key gets updated with a new date (row_expr_dt) and time (row_expr_tm). The most current row for a particular key (e.g., fin_doc = 10002) displays the row_expr_dt as '9999-12-31'.

The SQL in Figure 4 shows how the join relating to row_eff_ts should appear:



Loading Analytical Table with Transaction Lineage

```
INSERT INTO Enterprise_RPT_MET.v_fin_doc_line
SELECT
BSEG.row_eff_dt
,BSEG.row_eff_tm
,BSEG.row_expr_dt
,BSEG.row_expr_tm
,BSEG.fin_doc_nbr
,BSEG.fin_doc_line_nbr
,BKPF.fin_doc_dt
...
FROM Finance_CRE.v_fin_doc_line BSEG
INNER JOIN
Finance_CRE.v_fin_doc_hdr BKPF
ON BSEG.fin_doc_nbr = BKPF.fin_doc_nbr
AND (BSEG.row_eff_ts >= BKPF.row_eff_ts AND BSEG.row_eff_ts < BKPF.row_expr_ts);
```

Figure 4: Temporal Relation between primary source and dimension tables.

The 'and' clause highlighted in Figure 4 is a less-than-equal-to predicate which generates the one to one relationship for transaction lineage data of two tables. The join facilitates the one to one relationship should there be lower and upper bounds of the timestamp intervals that are not the same in the joining tables. The dimension table may have different lower and upper bounds of timestamps for current record, compared to primary source or fact table, as dimension data changes slowly.

VIEWING CONSISTENT DATA IN REPORTING ENVIRONMENT

The biggest challenge for creating an integrated data model in a data warehouse is to provide a consistent view (Radeschutz et al. 2013) of the data across subject areas. A data warehouse with temporal data must be able to manage data in a shared, reliable, time-sliced and efficient way (Hartmann et al. 2012). Aggregation and synchronization of data across subject areas provide unique challenges. View maintenance in a temporal data warehouse is complicated (Amo and Alves 2000) because they have to deal with multiple versions of data in a table. We propose separate application specific views to allow applications to have consistent view of data with separate time filters as needed by the users.

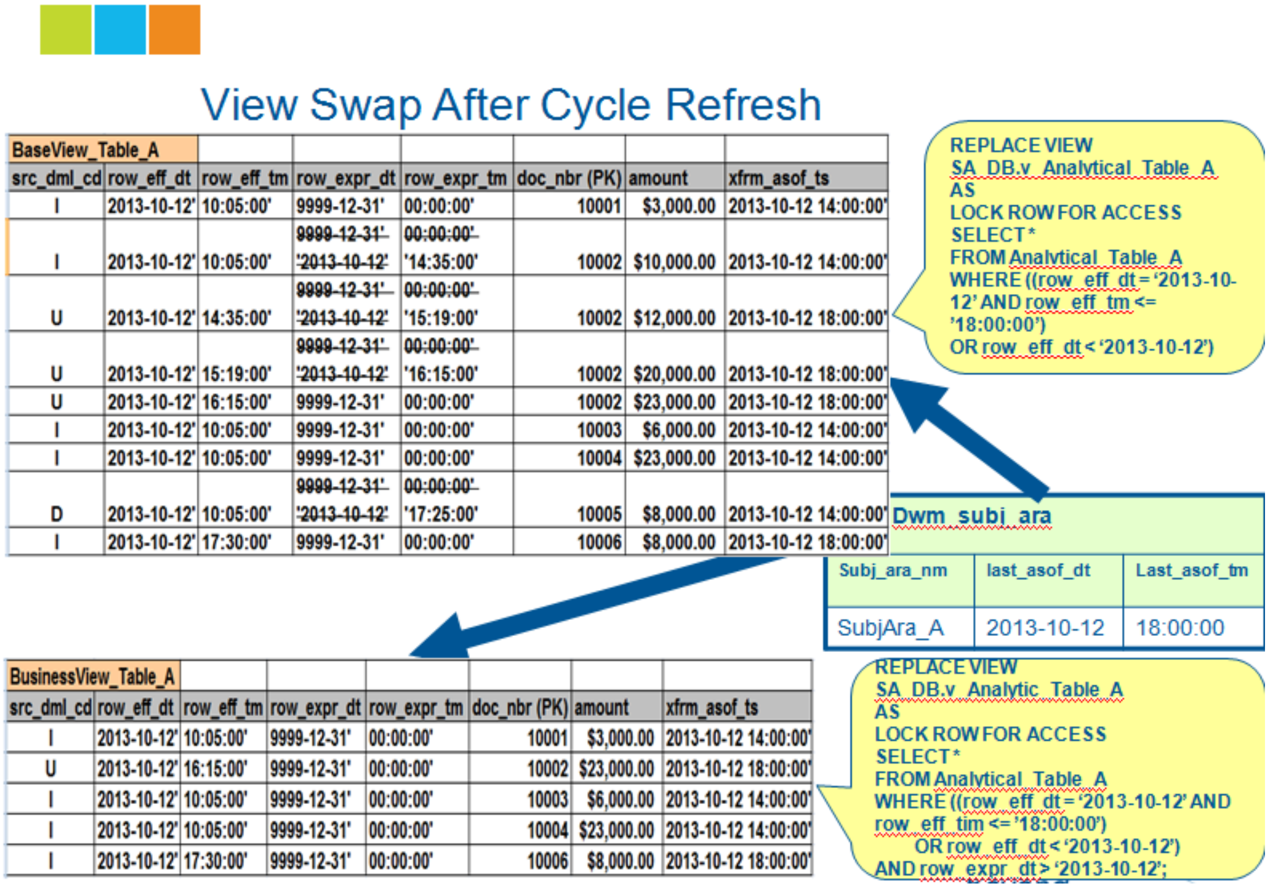
There are several different business application needs which can be filled by providing a separate set of views with different timestamp filters as required. Business users do not want to see data showing up in their reports as it is being appended or changed in a given table. Also, users would like to have all the updates to related tables in a subject area completed before a report shows any of that new cycle's data. They want to maintain "data latency" in this particular case. The report users want to see data based on the most current cycle refresh that occurred across all tables in the subject area(s).

The application specific views with timestamp filters are defined and dynamically recompiled right after each individual application's upstream subject area refreshes are completed per service level agreement (SLA). The view filters are based on 'row effective date', 'row effective time', and 'row expired date' and 'row expired time'. The row effective and expired timestamps associated with each subject area refresh begin and end timestamps are captured in the data warehouse metadata model (Kim et al. 2000) during the cycle refresh and later used for view recompilation.

Base Views

https://aisel.aisnet.org/jsais/vol2/iss1/2
DOI: 10.17705/3JSIS.00005

The base views that point to the target table, will have the 'lock for access' locking mechanism defined in the view. This will allow row level access to the table no matter if the table is being updated. The views will be defined with timestamp filter row_eff_ts <= last_refresh_ts. These views will be recompiled in the end of each cycle refresh. Figure 5 shows a complete set of data after cycle refresh followed by views swap. The Base-View for Table A shows all change history for a particular key.



BaseView_Table_A							
src_dml_cd	row_eff_dt	row_eff_tm	row_expr_dt	row_expr_tm	doc_nbr (PK)	amount	xfrm_asof_ts
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10001	$3,000.00	2013-10-12 14:00:00'
I	2013-10-12'	10:05:00'	~~9999-12-31'~~ ~~'2013-10-12'~~	~~00:00:00'~~ '14:35:00'	10002	$10,000.00	2013-10-12 14:00:00'
U	2013-10-12'	14:35:00'	~~9999-12-31'~~ ~~'2013-10-12'~~	~~00:00:00'~~ '15:19:00'	10002	$12,000.00	2013-10-12 18:00:00'
U	2013-10-12'	15:19:00'	~~9999-12-31'~~ ~~'2013-10-12'~~	~~00:00:00'~~ '16:15:00'	10002	$20,000.00	2013-10-12 18:00:00'
U	2013-10-12'	16:15:00'	9999-12-31'	00:00:00'	10002	$23,000.00	2013-10-12 18:00:00'
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10003	$6,000.00	2013-10-12 14:00:00'
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10004	$23,000.00	2013-10-12 14:00:00'
D	2013-10-12'	10:05:00'	~~9999-12-31'~~ ~~'2013-10-12'~~	~~00:00:00'~~ '17:25:00'	10005	$8,000.00	2013-10-12 14:00:00'
I	2013-10-12'	17:30:00'	9999-12-31'	00:00:00'	10006	$8,000.00	2013-10-12 18:00:00'

```
REPLACE VIEW
SA_DB.v_Analytical_Table_A
AS
LOCK ROW FOR ACCESS
SELECT *
FROM Analytical_Table_A
WHERE ((row_eff_dt = '2013-10-
12' AND row_eff_tm <=
'18:00:00')
OR row_eff_dt < '2013-10-12')
```

Dwm_subj_ara		
Subj_ara_nm	last_asof_dt	Last_asof_tm
SubjAra_A	2013-10-12	18:00:00

BusinessView_Table_A							
src_dml_cd	row_eff_dt	row_eff_tm	row_expr_dt	row_expr_tm	doc_nbr (PK)	amount	xfrm_asof_ts
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10001	$3,000.00	2013-10-12 14:00:00'
U	2013-10-12'	16:15:00'	9999-12-31'	00:00:00'	10002	$23,000.00	2013-10-12 18:00:00'
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10003	$6,000.00	2013-10-12 14:00:00'
I	2013-10-12'	10:05:00'	9999-12-31'	00:00:00'	10004	$23,000.00	2013-10-12 14:00:00'
I	2013-10-12'	17:30:00'	9999-12-31'	00:00:00'	10006	$8,000.00	2013-10-12 18:00:00'

```
REPLACE VIEW
SA_DB.v_Analytic_Table_A
AS
LOCK ROW FOR ACCESS
SELECT *
FROM Analytical_Table_A
WHERE ((row_eff_dt = '2013-10-12' AND
row_eff_tim <= '18:00:00')
      OR row_eff_dt < '2013-10-12')
AND row_expr_dt > '2013-10-12';
```

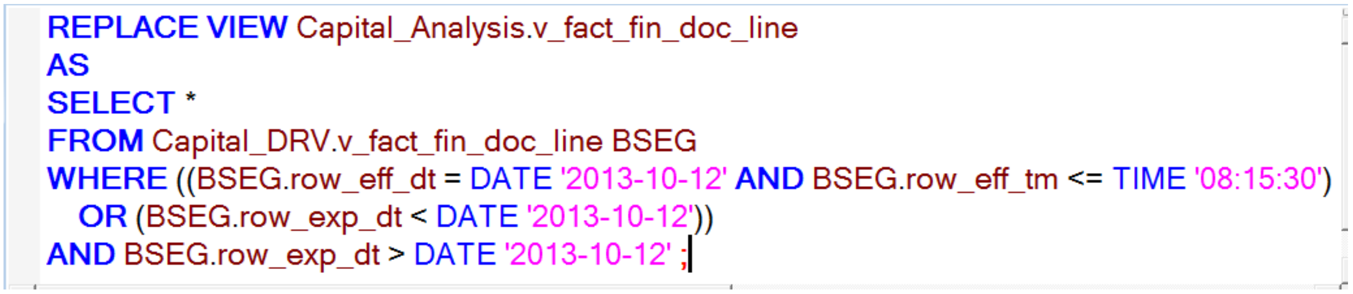
Figure 5: Table showing data after view re-point to active set of data.

Source Application Views

Source application business views will be defined on top of base views (for dirty reads). These views will provide data with filter based on application needs. For example, row_eff_ts <= application_reqd_asof_ts and row exp_ts > application_reqd_asof_ts. These views will be recompiled at end of last cycle refresh of an application. In these views the row uniqueness is maintained via business views. Figure 5 shows BusinessView for Table A. The view is showing the most current version of rows for each key (e.g., doc_nbr) value.

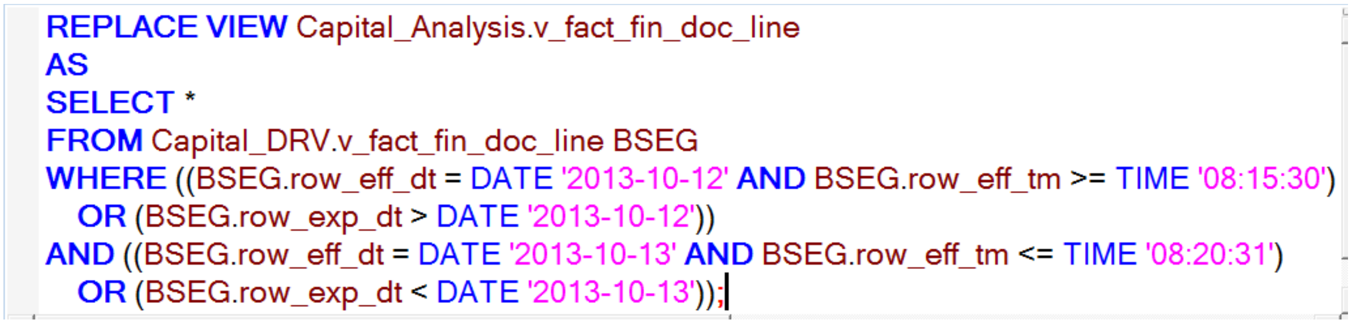
The data warehouse is a shared environment. Some subject areas are application specific while some others are shared by more than one subject area. Each application has its own SLA for data freshness. Also there are dependencies between subject area refreshes. All these factors make applications use different time slices of data. For example, finance related subject areas may run six times a day such as 2:00am 6:00am, 10:00am, 2:00pm, 6:00pm, and 10:00pm. On the other hand, capital related subject areas may run three times a day such as 3:00am, 11:00am and 7:00pm. Both these applications share some common subject areas. They use some application specific subject areas, too. This requires data to be "frozen" via a different set of views on the proper time slice to make data consistent and available per each applications specific business needs and SLA. For example a finance application might want to display finance data right after the finance subject areas load (e.g., 10:00am) while the capital application would not want to display just refreshed data right at that moment because capital analysis is based on the previous load or the analysis cannot begin until the other related subject areas have completed loading. In that case, a capital analyst will use data based on a separate set of views with previous refresh timestamp filter specified in the view definition. The finance analysis will see finance data up to the latest refresh via a different set of views. This way, data as of a point in time can be obtained across multiple tables or multiple subject areas, resolving consistency and synchronization issues. In this case two applications will be provided data freshness based on their individual SLA.

The report users normally want to see data based on the most recent cycle refresh that occurred across all tables in the subject area(s). For that particular time slice they like data demographics to remain as-is for analysis purposes. So, they may be provided with business views for each table that will show any data up to a point in time as needed. The reports will not see any new data that is being loaded as part of current cycle refresh until the cycle is finished. The report users will run queries in a business view with below timestamp filters in Figure 6 and 7:



```
REPLACE VIEW Capital_Analysis.v_fact_fin_doc_line
AS
SELECT *
FROM Capital_DRV.v_fact_fin_doc_line BSEG
WHERE ((BSEG.row_eff_dt = DATE '2013-10-12' AND BSEG.row_eff_tm <= TIME '08:15:30')
   OR (BSEG.row_exp_dt < DATE '2013-10-12'))
AND BSEG.row_exp_dt > DATE '2013-10-12' ;
```

Figure 6: Filters to pull rows up to a particular cycle refresh.



```
REPLACE VIEW Capital_Analysis.v_fact_fin_doc_line
AS
SELECT *
FROM Capital_DRV.v_fact_fin_doc_line BSEG
WHERE ((BSEG.row_eff_dt = DATE '2013-10-12' AND BSEG.row_eff_tm >= TIME '08:15:30')
   OR (BSEG.row_exp_dt > DATE '2013-10-12'))
AND ((BSEG.row_eff_dt = DATE '2013-10-13' AND BSEG.row_eff_tm <= TIME '08:20:31')
   OR (BSEG.row_exp_dt < DATE '2013-10-13'));
```

Figure 7: Filters to pull rows based on a particular time slice.

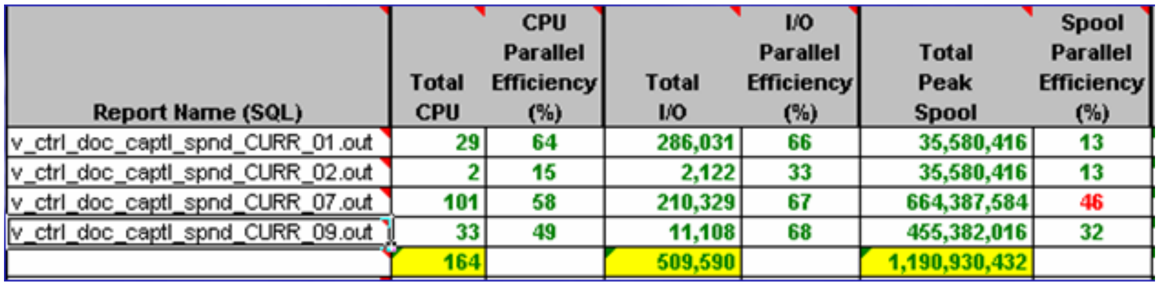
Query Performance

In acquiring data warehouse database systems customers take cost control and performance optimization as the critical evaluation criteria (Feinberg and Beyer 2010). Performance is referred to a product's primary operating characteristics (Gavin 1987). To improve database query performance, commercial databases have come up with several efficient indexes. The row effective date columns may have partitioned primary index (PPI) defined on them. That will make queries faster as the partition primary index pulls rows based on partition number instead of a full table scan. When a query is run with filters on PPI columns the DBMS will directly pull data based on particular bucket(s) instead of scanning the whole table.

Based on a SQL score-card on both PPI and non-PPI tables it was found that the SQL uses only 33% of the resources to pull rows from a PPI table in relation to a non-PPI table. The run time is also less in the same proportion. The potential gain derived from partitioning a table is the ability to read a small subset of the table

https://aisel.aisnet.org/jsais/vol2/iss1/2
DOI: 10.17705/3JSIS.00005

instead of the entire table. Queries which specify a restrictive condition on the partitioning column will avoid full table scans. By defining a PPI on 'row effective date' the report query performance was found to be four times faster and CPU savings about 33%.



Report Name (SQL)	Total CPU	CPU Parallel Efficiency (%)	Total I/O	I/O Parallel Efficiency (%)	Total Peak Spool	Spool Parallel Efficiency (%)
v_ctrl_doc_captl_spnd_CURR_01.out	29	64	286,031	66	35,580,416	13
v_ctrl_doc_captl_spnd_CURR_02.out	2	15	2,122	33	35,580,416	13
v_ctrl_doc_captl_spnd_CURR_07.out	101	58	210,329	67	664,387,584	46
v_ctrl_doc_captl_spnd_CURR_09.out	33	49	11,108	68	455,382,016	32
	164		509,590		1,190,930,432	

Figure 8: Resource Usage: PPI vs. No PPI tables.

Figure 8 shows a comparison of query response time and computational resource savings between PPI and No-PPI queries. The first query was run to pull 53K rows, with no PPI defined. The response time was eight seconds and CPU consumption was 29 seconds in row one. The same query was run against the same table with PPI defined on row effective date. For the second run the response time was one second and resource consumption was two seconds per row two. The first two rows show the resource usage statistics. A second query was run to pull 424K rows, with no PPI defined. The response time was 25 seconds and resource consumption was 101 CPU seconds in row three. The same query was run against the same table with PPI defined on row effective date. This second run response time was four seconds and resource consumption was 33 seconds in row four.

There are many techniques to improve performance (Rahman 2013) of data warehouse queries, ranging from commercial database indexes and query optimization. A number of indexing strategies have been proposed for data warehouses in literature and are heavily used in practice.

ENSURING DATA QUALITY IN LOADING AND VIEWING

Once Joshua Boger, CEO and founder of Vertex Pharmaceuticals said that, "I've never made a bad decision. I've just had bad data (Pisano et al. 2006)". This speaks for the importance of providing quality data to users for internal business decision making and external regulatory compliance (Bai et al. 2012; and Baskarada and Koronios 2013). Data quality is essential for business intelligence success (Isik et al. 2013). Better data quality has a positive influence on sales, profit making, and value added (Xiang et al. 2013). System quality has positive influence on data quality and information quality (Hwang and Xu 2008). During the load process, from operational source to the temporal data warehouse, performing transformation and loading from staging area to analytical area of data warehouse, utmost care must be taken to avoid data corruption. This data quality might be compromised for many reasons. While joining multiple source tables if join operators do not consider inequality predicates the integrity of timestamp-based data might be compromised. Data quality might be compromised due to lack of maintaining referential integrity (Ordonez and Garcia-Garcia 2008) as well. Data quality also might be compromised while retrieving data from analytical tables viewing information via reporting, business intelligence, and data mining tools. This could happen due to missing join criteria or bad transformation logic. All care must be taken to ensure the most accurate data retrieval. This is very true in the case of temporal data, as it is stored with different versions of time-referenced data.

COEXISTENCE OF LOAD AND QUERY-RUN

During the load process by the stored procedures, the DBMS will use 'write lock', by default, to perform DML during the new cycle refresh. The report SQL can be defined by 'locking for access' lock to retrieve the rows for a specific time slice. Both read and write locks are compatible. The author of this article conducted a test by running a stored procedure which performed update and insert operations on the active table via one database session. The stored procedure updated 19 million rows to expire them and inserted another 19 million rows with new row effective timestamp. At the same time a report query was run repeatedly via another session. The DBA monitored the activities of these two sessions to see if there was any blocking or waiting since both write and read access were occurring on the same active table at the same time. There was no blocking or waiting found as the DBMS 'write lock' and 'lock for access' locks were compatible.

In order to make sure that report users see consistent data they will be provided business views with last data refresh. Every time all related subject area refresh for a particular application is completed, the views will be re-compiled with new timestamps or the views will be pointed to a metadata (Rahman et al. 2012) table, via a join condition, to get the most recent refresh timestamp and use it as filters in report queries.

CONCLUDING REMARKS

In this article, we have proposed temporal data update methodologies for data warehousing. The goal here is to come up with mechanisms for capturing transaction lineage for each record in data warehouse tables. We identified the key areas of temporal data warehouse refreshes based on practical experience in data warehouse implementation. Many database applications need to retain all previous and current states of data from accountability and traceability perspectives (Jensen and Lomet 2001). We provided methodologies to extract operational data from heterogeneous operational sources and to capture them in staging areas of the data warehouse with transaction lineage. We showed how to pull source data from more than one source staging tables in DW, how to perform join operations with inequality predicates for more than one joins and then load them in analytical subject area tables. We have provided SQL syntax to pull and load data from staging area tables to analytical tables maintaining transaction lineage.

We proposed load methodologies by way of database specific load utilities and software components such as stored procedures and macros. We showed onetime full refresh and subsequent incremental refreshes of temporal data in conventional data warehouse. In order to implement temporal data update methodologies we proposed a data model consisting of four additional columns, such as 'row effective date', 'row effective time', 'row expired date' and 'row expired time', to mark row effective date/time and row expired date/time against each row in the table. In order to make each row unique we proposed the row effective date and time columns to be part of primary key.

The transaction lineage that we capture in data warehouse provides different applications and analytical community read data whatever the subset of data they need is based on different time slices. Temporal joins have been presented for temporal relations. Also suggestions have been made to take advantage of several indices offered by current commercial databases.

We proposed several business views based on timestamp filters for use by different applications. Application specific business views have been defined with timestamps as needed by individual applications. There are several different business application needs which can be filled by providing a separate set of views with different timestamp filters as required. The application specific views with timestamp filters are defined and dynamically recompiled right after each individual application's upstream subject area refreshes are completed per service level agreement (SLA). The temporal data update methodologies presented in this article should sufficiently meet the needs of application owners and customers as it takes into consideration several factors such as providing common data source with different time-varying data.

ACKNOWLEDGMENTS

Earlier version of this article was presented at the 14th Americas Conference on Information Systems (AMCIS 2008) (Rahman 2008a) and the 2008 Teradata Partners User Group Conference & Expo (Rahman 2008b). This article has been substantially revised, based on comments received during conference presentation and comments received from anonymous reviewers of this journal. The author would like to thank the anonymous reviewers for their constructive comments provided on earlier drafts of this article. Many thanks to Yatish K Goel, at Intel, for contributing to the initial version of figure 1. The author also thanks Joan Schnitzer, Senior Systems Analyst at Intel, for an excellent editing job.

REFERENCES

1. Ahn, I. and Snodgrass, R. 1986. "Performance Evaluation of a Temporal Database Management System," *ACM SIGMOD Record*, (15:2), pp. 96-107.
2. Amo, S.D. and Alves, M.H.F. 2000. "Efficient Maintenance of Temporal Data Warehouses," in *Proceedings of the 2000 International Symposium on Database Engineering & Applications*, pp. 188-96.
3. Apeh, E. and Gabrys, B. 2013. "Detecting and Visualizing the Change in Classification of Customer Profiles Based on Transactional Data," *Evolving Systems*, (4), pp. 27-42, DOI: 10.1007/s12530-012-9065
4. Ariyachandrs, T. and Watson, H. 2010. "Key Organizational Factors in Data Warehouse Architecture Selection," *Decision Support Systems*, (49), pp. 200-212.

   http://dx.doi.org/10.1016/j.dss.2010.02.006
5. Bai, X., Nunez, M. and Kalagnanam, J.R. 2012. "Managing Data Quality Risk in Accounting Information Systems," *Information Systems Research*, (23:2), pp. 453-473.
6. Baskarada, S. and Koronios, A. 2013. "Data, Information, Knowledge, Wisdom (DIKW): A Semiotic Theoritical and Empirical Exploration of the Hierarchy and its Quality Dimension," *Australasian Journal of Information Systems*, (18:1), pp. 5-24.
7. Bellatreche, L. and Wrembel, R. 2013. "Special Issue on: Evolution and Versioning in Semantic Data Integration Systems," *J Data Semant*, (2), pp. 57-59, DOI 10.1007/s13740-013-0020-6.
8. Berkani, N., Bellatreche, L. and Khouri, S. 2013. "Towards a Conceptualization of ETL and Physical Storage of Semantic Data Warehouses as a Service," *Cluster Comput*, pp. 1-17, DOI: 10.1007/s10586-013-0266-7.

   http://dx.doi.org/10.1007/s10586-013-0266-7
9. Brobst, S., McIntire, M. and Rado, E. 2008. "Agile Data Warehousing with Integrated Sandboxing," *Business Intelligence Journal*, (13:1), pp. 1-10.
10. Bruckner, R. and Tjoa, A. 2002. "Capturing Delays and Valid Times in Data Warehouses—Towards Timely Consistent Analyses," *Journal of Intelligent Information Systems*, (19:2), pp. 169 – 190.
11. Chau, V.T.N. and Chittayasothorn, S. 2008. "A Temporal Object Relational SQL Language with Attribute Timestamping in a Temporal Transparency Environment," *Data & Knowledge Engineering*, (67), pp. 331-361.

    http://dx.doi.org/10.1016/j.datak.2008.06.008
12. Chaudhuri, S., Dayal, U. and Narasayya, V. 2011. "An Overview of Business Intelligence Technology," *Communications of the ACM*, (54:8), pp. 88-98.
13. Chen, H., Chiang, R.H.L. and Storey, V.C. 2012. "Business Intelligence and Analytics: From Big Data to Big Impact," *MIS Quarterly*, (36:4), pp. 1165-1188.
14. Chen, L., Rahayu, W. and Taniar, D. 2010. "Towards Near Real-Time Data Warehousing," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications*, pp. 1150-1157.

    http://dx.doi.org/10.1109/AINA.2010.54
15. Chountas, P., Petrounias, I., Vasilakis, C.,Tseng, A., El-Darzi, E.,Atanassov, K. and Kodogiannis, V. 2004. "On Uncertainty and Data-Warehouse Design," in *Proceedings of the Third International Conference on Advances in Information Systems*, ADVIS 2004, Izmir, Turkey, October 20-22, 2004, pp. 4-13.

16. Downes, L. and Nunes, P.F. 2013. "The Big Idea: Big-Bang Disruption," *Harvard Business Review*, March 2013, pp. 1-12.
17. Ejaz, A. and Kenneth, R. 2004. "Utilizing Staging Tables in Data Integration to Load Data into Materialized Views," in *Proceedings of the First International Symposium on Computational and Information Science (CIS'04)*, Shanghai, China, December 16-18, 2004, pp. 685-691.
18. Fegaras, L. and Elmasri, R. 1998. "A Temporal Object Query Language," in *Proceedings of the IEEE Fifth International Workshop on Temporal Representation and Reasoning*, IEEE Computer Society Press, pp. 51-59.
19. Feinberg, D. and Beyer, M.A. 2010. "Magic Quadrant for Data Warehouse Database Management Systems," *Gartner Research*, ID No. G00173535, pp. 2-38.
20. Feng, Y, Li, H. Agrawal, D. and Abbadi, A. 2005. "Exploiting Temporal Correlation in Temporal Data Warehouses," in *Proceedings of the 10th International Conference on Database Systems for Advanced Applications*, DASFAA 2005, pp. 662-675.
21. Gao, D., Jensen, C.S., Snodgrass, R.T. and Soo, M.D. 2005. "Join Operations in Temporal Databases," *The VLDB Journal*, (14), pp. 2-29.

    http://dx.doi.org/10.1007/s00778-003-0111-3
22. Gardner, S.R. 1998. "Building the Data Warehouse," *Communications of the ACM*, (41:9), pp. 52-60.
23. Gavin, D.A. 1987. "Competing on the Eight Dimensions of Quality," *Harvard Business Review*, November-December, 1987, pp. 101-109.
24. Golfarelli, M. and Rizzi, S. 2009. "A Survey on Temporal Data Warehousing," *International Journal of Data Warehousing & Mining*, (5:1), pp. 1-17.
25. Gupta, M., Gao, J., Aggarwal, C.C. and Han, J. 2013. "Outlier Detection for Temporal Data: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, (25:1), pp. 1-20.
26. Hanson, J.H. and Willshire, M.J. 1997. "Modeling a faster data warehouse," in *International Database Engineering and Applications Symposium (IDEAS '97)*, pp. 260-265.
27. Hartmann, S., Kirchberg, M. and Link, S. 2012. "Design by Example for SQL Table Definitions with Functional Dependencies," *The VLDB Journal*, (21), pp. 121-144.

    http://dx.doi.org/10.1007/s00778-011-0239-5
28. Huq, M.R., Wombacher, A. and Apers, P.M.G. 2010. "Facilitating Fine Grained Data Provenance Using Temporal Data Model," in *Proceedings of the VLDB '10 Conference*, September 13-17, 2010, Singapore., pp. 1-6.
29. Hwang, M.I. and Xu, H. 2008. "A Structural Model of Data Warehousing Success," *The Journal of Computer Information Systems*, (49:1), pp. 48-56.
30. Inmon, W.H. 2002. "Building the Data Warehouse," 3rd Edition, John Wiley.
31. Inmon, W.H., Terdeman, R.H., Norris-Montanari, J., and Meers, D. 2001. "Data Warehousing for E-Business," 1st Edition, John Wiley.
32. Isik, O., Jones, M.C. and Sidorova, A. 2013. "Business Intelligence Success: The Roles of BI Capabilities and Decision Environments," *Information & Management*, (50), pp. 13-23.

    http://dx.doi.org/10.1016/j.im.2012.12.001
33. Jensen, C.S. 1999. "Temporal Data Management," *IEEE Transactions on Knowledge and Data Engineering*, (11:1), pp. 36-44.
34. Jensen, C.S. 2000. "Introduction to Temporal Database Research," Retrieved on October 12, 2013 from: http://www.cs.aau.dk/~csj/Thesis/pdf/chapter1.pdf , 1-27.

https://aisel.aisnet.org/jsais/vol2/iss1/2
DOI: 10.17705/3JSIS.00005

35. Jensen, C.S. and Lomet, D.B. 2001. "Transaction Time-stamping in (Temporal) Databases," in *Proceedings of the 27th VLDB Conference*, Roma, Italy, 2001, pp. 1-10.

36. Jestes, J., Phillips, J.M., Li, F. and Tang, M. 2012. "Ranking Large Temporal Data," in *Proceedings of the VLDB Endowment*, (5:10), pp. 1-12.

37. Kaufmann, M. 2013. "Storing and Processing Temporal Data in a Main Memory Column Store," in *Proceedings of the VLDB Endowment*, (6:12), pp. 1-6.

38. Kaur, N.K., Chaudhary, N. and Singh M. 2013. "Implementation of a Temporal Database on a Conventional Database," *International Journal of Computer Science and Communication Engineering (IJCSCE)*, Special Issue, pp. 228-231.

39. Kim, N., Moon, S. and Lee, S. 2004. "Conflict Order-Based View Refreshment Scheme for Transaction Management in Data Warehouse Environment," *Journal of Computer Information Systems*, Winter 2003-2004, pp. 105-111.

40. Kim, T., Kim, J., and Lee, H. 2000. "A Metadata-Oriented Methodology for Building Data Warehouse: A Medical Center Case," *Informs & Korms* - 928, Seoul 2000 (Korea).

41. Kvet, M. and Matiasko, K. 2013. "Temporal Data Modeling: Future Valid Data Processing," in *Proceedings of the 2013 Society of Digital Information and Wireless Communications (SDIWC2013)*, pp. 268-282.

42. Li, F., Yi, K. and Le, W. 2010. "Top-k Queries on Temporal Data," *The VLDB Journal*, (19), pp. 715-733.

http://dx.doi.org/10.1007/s00778-010-0186-6

43. Mahmood, N., Burney, A. and Ahsan, K. 2010. "A Logical Temporal Relational Data Model," *International Journal of Computer Sciences Issues (IJCSI)*, (7:1), pp. 1-9.

44. Malinowski, E. and Zimányi, E. 2006. "A conceptual solution for representing time in data warehouse dimensions," in *Proceedings of the 3rd Asia-Pacific conference on Conceptual Modeling (APCCM 2006)*, (53), pp. 45-54.

45. Malinowski, E. and Zimanyi, E. 2008. "A Conceptual Model for Temporal Data Warehouses and its Transformation to the ER and the Object-Relational Models," *Data & Knowledge Engineering*, (64), pp. 101-133.

http://dx.doi.org/10.1016/j.datak.2007.06.020

46. Martin, C. and Abello, A. 2003. "A Temporal Study of Data Sources to Load a Corporate Data Warehouse," in *Proceedings of the 5th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2003)*, Prague, Czech Republic, September 3-5, 2003, pp. 119-118.

47. Mkaouar, M., Bouaziz, R. and Moalla, M. 2011. "Querying and Manipulating Temporal Databases," *International Journal of Database Management Systems (IJDMS)*, (3:1), pp. 1-17.

48. Ordonez, C. and Garcia-Garcia, J. 2008. "Referential Integrity Quality Metrics," *Decision Support Systems*, (44), pp. 495-508.

http://dx.doi.org/10.1016/j.dss.2007.06.004

49. Ozsoyoglu, G. and Snodgrass, R. 1995. "Temporal and Real-Time Databases: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, (7:4), pp. 513-532.

50. Pisano, G., Fleming, L. and Strick, E.P. 2006. Vertex Pharmaceuticals: R&D Portfolio Management (A), HBS Case No. 9-604-101, Harvard Business School Publishing, Boston, 2006.

51. Radeschutz, S., Schwarz, H. and Niedermann, F. 2013. "Business Impact Analysis - A Framework for a Comprehensive Analysis and Optimization of Business Processes," *Comput Sci Res Dev*, pp. 1-18.

http://dx.doi.org/10.1007/s00450-013-0247-3

52. Rahman, N. 2008a. "Updating Data Warehouses with Temporal Data," in *Proceedings of the 14th Americas Conference on Information Systems (AMCIS 2008)*, Toronto, ON, Canada, August 14-17, 2008, 323.

53. Rahman, N. 2008b. “Refreshing Teradata Warehouse with Temporal Data,” *Presented at the 2008 Teradata Partners User Group Conference & Expo*, Mandalay Bay Resort, Las Vegas, Nevada, USA, October 12-16, 2008.
54. Rahman, N. 2010. “Incremental Load in a Data Warehousing Environment,” *International Journal of Intelligent Information Technologies (IJIIT)*, 6, 3, 1-16.

    http://dx.doi.org/10.4018/jiit.2010070101
55. Rahman, N. 2013. “Measuring Performance for Data Warehouses - A Balanced Scorecard Approach,” *International Journal of Computer and Information Technology (IJCIT)*, (4:1), pp. 1-7.
56. Rahman, N., Marz, J. and Akhter, S. 2012. “An ETL Metadata Model for Data Warehousing,” *Journal of Computing and Information Technology (CIT)*, (20:2), pp. 95-111.
57. Ramiller, N.C. and Swanson, B. 2009. “Mindfulness Routines for Innovating with Information Technology,” *Journal of Decision Systems*, (18:1), pp. 13-26.
58. Roberts, N. and Grover, V. 2012. “Leveraging Information Technology Infrastructure to Facilitate a Firm's Customer Agility and Competitive Activity: An Empirical Investigation,” *Journal of Management Information Systems*, (28:4), pp. 231-269.
59. Samtani, S., Mohania, M., Kumar, V and Kambayashi, Y. 1998. “Recent Advances and Research Problems in Data Warehousing,” in *Proceedings of Advances in Database Technologies: ER '98 Workshops on Data Warehousing and Data Mining, Mobile Data Access, and Collaborative Work Support and Spatio-Temporal Data Management*, Singapore, November 19-20, 1998, pp. 81-92.
60. Sein, M., Henfridsson, O., Purao, S., Rossi, M. and Lindgren, R. 2011. “Action Design Research,” *MIS Quarterly*, (35:1), pp. 37-56.
61. Sen, A., Sinha, A.P. and Ramamurthy, K. 2006. “Data Warehousing Process Maturity: An Exploratory Study of Factors Influencing User Perceptions,” *IEEE Transactions on Engineering Management*, (53:3), pp. 440-455.
62. Shin, B. 2003. “An Exploratory Investigation of System Success Factors in Data Warehousing,” *Journal of the Association for Information Systems*, (4), pp. 141-170.
63. Snodgrass, R.T. 2010. “A Case Study of Temporal Data,” Teradata Corporation White Paper, Retrieved on October 1, 2013 from: http://www.teradata.com/Search.aspx?id=8550&q=white papers, pp. 1-21.
64. Stonebraker, M. 2011. “New Opportunities for New SQL,” *Communications of the ACM*, (55:11), pp. 10-11.
65. Stonebraker, M., Cetintemel, U. and Zdonik, S. 2005. “The 8 Requirements of Real-Time Stream Processing,” *SIGMOD Record*, (34:4).

    http://dx.doi.org/10.1145/1107499.1107504
66. Terenziani, P. 2012. “Temporal Aggregation on User-Defined Granularities,” *J Intell Inf Syst*, (38), pp. 785-813.

    http://dx.doi.org/10.1007/s10844-011-0179-y
67. Thomas, H. and Datta, A. 2001. “A Conceptual Model and Algebra for On-Line Analytical Processing in Decision Support Databases,” *Information Systems Research*, (12:1), pp. 83 - 102.
68. TimeConsult. 2013. “What is Temporal Data?”, Retrieved on October 12, 2013 from: http://www.timeconsult.com/TemporalData/TemporalData.html
69. Torp, K. 1998. “Implementation Aspects of Temporal Databases,” Retrieved on October 12, 2013 from: http://www.cs.aau.dk/NDB/phd_projects/torp.html
70. Torp, K., Jensen, C., and Snodgrass, R. 2000. “Effective timestamping in databases,” *The VLDB Journal*, (8:3-4), pp. 267-288.

71. Vavouras, A., Gatziu, S. and Dittrich, K. 1999. "Modeling and Executing the Data Warehouse Refreshment Process," in *Proceedings of the International Symposium on Database Applications in Non-Traditional Environments (DANTE '99)*, pp. 66-73.
72. Viqueira, J.R.R and Lorentzos, N.A. 2007. "SQL Extension for Spatio-Temporal Date," *The VLDB Journal*, (16:2), pp. 179-200.
73. Wang, J., Han, S., Lam, K.Y. and Mok, A.K. 2012. "Maintaining Data Temporal Consistency in Distributed Real-Time Systems," *Real-Time Syst*, (48), pp. 387-429.

    http://dx.doi.org/10.1007/s11241-012-9150-4
74. Widom, J. 1995. "Research Problems in Data Warehousing," in *Proceedings of the 4th Int'l Conference on Information and Knowledge Management*, CIKM '95, Baltimore, MD, USA, November 1995, pp. 25-30.
75. Xiang, J.Y., Lee, S. and Kim, J.K. 2013. "Data Quality and Firm Performance: Empirical Evidence from the Korean Financial Industry," *Inf Technol Manag*, (14), pp. 59-65.

    http://dx.doi.org/10.1007/s10799-012-0145-6
76. Yang, J. and Widom, J. 1998. "Maintaining Temporal Views Over Non-temporal Information Sources for Data Warehousing," in *Proceedings of the 6th International Conference on Extending Database Technology, Advances in Database Technology – EDBT '98*, pp. 389-404.
77. Yang, J. and Widom, J. 2001. "Incremental computation and maintenance of temporal aggregates," in *Proceedings of the International Conference on Data Engineering (ICDE '01)*.

    http://dx.doi.org/10.1109/ICDE.2001.914813
78. Yufei Tao, Y., Papadias, D., and Faloutsos, C. 2004. "Approximate temporal aggregation," in *Proceedings of the International Conference on Data Engineering (ICDE '04)*.
79. Zhang, D., Markowetz, A., Tsotras, V., Gunopulos, D., and Seeger, B. 2001. "Efficient computation of temporal aggregates with range predicates," in *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 237-245.

    http://dx.doi.org/10.1145/375551.375600

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, and an Oracle Certified Developer and DBA. His most recent publications on data warehousing appeared in Proceedings of the IEEE 26th Canadian Conference of Electrical and Computer Engineering (CCECE 2013) and the International Journal of Computer and Information Technology (IJCIT). His principal research interests include Active Data Warehousing, Changed Data Capture and Management in Temporal Data Warehouses, Data Mining for Business Analysts, Big Data Applications, and Sustainability of Information Technology.