Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2008 Proceedings

Americas Conference on Information Systems (AMCIS)

2008

Updating Data Warehouses with Temporal Data

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

Updating Data Warehouses with Temporal Data

Nayem Rahman
Enterprise Data Warehouse Engineering – ETL, Intel Corporation
nayem.rahman@intel.com

ABSTRACT

There has been a growing trend to use temporal data in a data warehouse for making strategic and tactical decisions. The key idea of temporal data management is to make data available at the right time with different time intervals. The temporal data storing enables this by making all the different time slices of data available to whoever needs it. Users with different data latency needs can all be accommodated. Data can be "frozen" via a view on the proper time slice. Data as of a point in time can be obtained across multiple tables or multiple subject areas, resolving consistency and synchronization issues. This paper will discuss implementations such as temporal data updates, coexistence of load and query against the same table, performance of load and report queries, and maintenance of views against the tables with temporal data.

Keywords

Temporal Data, Temporal Data Warehouse, Row-Effective Timestamp, Row-Expired Timestamp.

INTRODUCTION

In today's competitive business environment, successful businesses are data driven. The business executives would want to make strategic as well as tactical business decisions [3] with accurate information at the right time. The accuracy of information is dependent on detailed data as well as time-varying data. The data warehousing 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.

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 [11, 15]. 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 [15, 25].

A temporal data warehouse is significantly different from an operational database in many respects [20]. Operational source systems are usually non-temporal and maintain only current state of data as opposed to complete history of data [4] with transaction lineage. Data warehouses are always maintained to hold large volumes of historical data.

During the last decade data warehousing has achieved prominence. Scattered databases and data-marts are being consolidated into more useful data warehouses. Temporal data warehousing has gained prominence among different stakeholders including suppliers, business users, and researchers because of user popularity and management patronage [12].

"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 [2]."

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 [10]." In this definition time-varying means the possibility to keep different values of the same record according to its changes over time [14].

Temporal data warehouses provide a history of serialized changes to data identified by times when changes occurred. This allows for querying the current state as well as past states of a record [6]. Conventional databases provide users only current state of data which is true as of a single point in time [16]. 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 [4]. A record inserted in a database is never physically deleted [5]. A new record or a new version of an existing record is always added to reflect a transaction on that data. Thus an evolving history of data is maintained in the temporal data warehouse.

Many applications can benefit from a temporal data warehouse [22, 28] such as retail sales, financial services, medical records, inventory control, 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 are often valuable to different stakeholders. They should be stored in both current state and all previously current states.

Although there are clear benefits of and demand for temporal database management systems (DBMS) they are not commercially available [24]. The current commercial databases are non-temporal and hence, they do not provide a special temporal query language, a temporal data definition language, and a temporal manipulation language [23].

In the absence of a temporal DBMS I 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 matured enough to express complex data transformations [21] and also have performance improvement measures, such as different kinds of efficient algorithms for indexing. The improvements in the area of disk storage technology have also made it possible to efficiently store and manage temporal data with all transaction lineages [1, 24].

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 proved to be an efficient method of loading from the standpoint of performance [3] and data consistency. Another aspect of storing data in data warehouses is that initially data are captured in staging subject areas [9, 17] 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 paper 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.

RELATED WORKS

Data updates are important issues for temporal data warehousing. 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. Discarding updates between two refresh points of time when performing periodic complete reloads leads to a loss of transaction lineage [26]. Most previous work on data warehousing focused on design issues, data maintenance strategies in connection with relational view materialization and implementation issues [13, 19, 27]. There has been little research work done to date on the temporal view maintenance problem [28] and most of the previous research ignores the temporal aspects of data warehousing [4].

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 [14]. The temporal aggregation problem was studied in [29, 30, 31] 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 [7].

In this paper, I 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 hours window. I 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 inert 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. I also examine the possibility of using metadata tables to recompile views based on subject area refresh timestamps. I show that there are opportunities to use different performance improvement

features, such as indexing, of existing commercial databases to load and query temporal data in the commercial non-temporal databases.

DATA UPDATE METHODOLOGY

During the past two decades, several dozen temporal data model have appeared, all with timestamps being integral components [25]. However, there is no commercial database on the market yet due to complex nature of temporal data. This paper presents a technical outline of how to use the existing 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 row and inserting the new version of the row. 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 [1].

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. It provides mechanism 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.

In order to implement temporal data update methodology the existing data model will have four additional columns, such as 'row effective date', 'row effective time', 'row expired date' and 'row expired time', in order to mark row effective date/time and row expired date/time against each row in the table. In order 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 [14] 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 [11].

Immediately after staging tables are loaded from source system the data warehouse SQL will be used to process and expire 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. For '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 with temporal data. In case of full refresh as initial load, the current version of rows will be used to perform load. The change date of current row will be used as row effective timestamp and '9999-12-31 12:00:00' as expired timestamp.

The following table shows how data look in the final target table in the data warehouse:

	asof_src_ts	row_eff_dt	row_eff_tm	row_expr_dt	row_expr_tm	fscl_yr_nbr	fin_doc_nbr	fin_doc_amt
1	2005-08-19 02:00:00	2005-08-19	02:00:00	9999-12-31	00:00:00	2005	10001	3000.50
2	2005-08-19 02:00:00	2005-08-19	02:00:00	2005-08-19	06:00:00	2005	10002	5000.50
3	2005-08-19 06:00:00	2005-08-19	06:00:00	2005-08-19	10:00:00	2005	10002	7000.50
4	2005-08-19 10:00:00	2005-08-19	10:00:00	2005-08-19	14:00:00	2005	10002	14000.50
5	2005-08-19 14:00:00	2005-08-19	14:00:00	9999-12-31	00:00:00	2005	10002	20000.50
6	2005-08-19 10:00:00	2005-08-19	10:00:00	9999-12-31	00:00:00	2005	10003	25000.59

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

Among the highlighted rows (with the same key) 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'.

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 [8]. 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. In order to load such 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 relation with row effective date and time of primary/ fact table rows.

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 one to one relation for transactions lineage data by 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 will be joined with an 'inner join' (instead of left outer) conditions. The SQL below shows how the join relating to row_eff_ts should appear:

```
-- Insert derived table: fin_doc_line
INSERT INTO Enterprise_DRV_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
...
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_hdr
AND BSEG.fin_doc_line_nbr = BKPF.fin_doc_line_nbr
AND (BSEG.row_eff_ts >= BKPF.row_eff_ts AND BSEG.row_eff_ts < BKPF.row_eff_ts);
```

Figure 1. Temporal Relation between primary source and dimension tables.

The 'and' clause highlighted in black in Figure 1 is a less-than-equal-to predicate to make one to one relation for transaction lineage data of two tables. The join allows perform one to one relation in case lower and upper bounds of the timestamp intervals 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 BY REPORTING TOOLS

The biggest challenge for creating an integrated data model in a data warehouse is to provide a consistent view of the data across subject areas. Aggregation and synchronization of data across subject areas provide unique challenges. View maintenance in a temporal data warehouse is complicated [2] because they have to deal with multiple versions of data in a table. I propose separate application specific views to allow applications to have consistent view of data with separate time filters as need 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. The report users want to see data based on the most 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 [18] during the cycle refresh and later used for view recompilation.

Base Views: 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.

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.

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 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. If finance application wants to see finance data right after the finance subject areas load (e.g., 10:00am) but, the capital analysis does not want to see just refreshed data right at that moment because it (capital analysis) was doing analysis based on previous load. Or it might be waiting since other related subject areas are not finished yet. 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 2 and 3:

```
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 '2008-04-28' AND BSEG.row_eff_tm <= TIME '05:30:24')
    OR (BSEG.row_exp_dt < DATE '2008-04-28'))
AND BSEG.row_exp_dt > DATE '2008-04-28' ;
```

Figure 2. 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 '2008-04-28' AND BSEG.row_eff_tm >= TIME '05:30:24')
    OR (BSEG.row_exp_dt > DATE '2008-04-28'))
AND ((BSEG.row_eff_dt = DATE '2008-04-29' AND BSEG.row_eff_tm <= TIME '05:31:24')
    OR (BSEG.row_exp_dt < DATE '2008-04-29')) ;
```

Figure 3. Filters to pull rows based on a particular time slice.

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. Current commercial databases have come up with several other efficient indexes to improve query performance.

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 instead of the entire table. Queries which specify a restrictive condition on the partitioning column 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 4. Resource Usage: PPI vs. No PPI tables.

Figure 4 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 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.

SIMULTANEOUS 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 paper 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 table, via a join condition, to get the most recent refresh timestamp and use it as filters in report queries.

CONCLUSION

The data acquisition methodology presented in this paper 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. Application specific business views have been defined with timestamps as needed by individual applications. Temporal joins have been presented for temporal relations. Also suggestions have been made to take advantage of several indices offered by current commercial databases.

ACKNOWLEDGEMENTS

The author wishes to thank the anonymous referees for the time they spent on the detailed comments that were helpful in improving this paper. The author also thanks Peter W Burkhardt 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, 96-107.
2. Amo, S.D. and Alves, M.H.F. (2000) Efficient Maintenance of Temporal Data Warehouses, Proceedings of the 2000 International Symposium on Database Engineering & Applications, 188-96.
3. Brobst, S. and Ballinger, C. (2003) Active Data Warehousing: Why Teradata Warehouse is the Only Proven Platform, October 2003, Available online at http://www.teradata.com/typdf. aspx?a=83673&b=86860 (retrieved on 05/21/2006), pp. 1-12.
4. 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, 169 – 190.
5. 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.
6. Fegaras, L. and Elmasri, R. (1998) A Temporal Object Query Language, In IEEE Proc. Fifth International Workshop on Temporal Representation and Reasoning, IEEE Computer Society Press, 51-59.
7. 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, 662-675.
8. Gardner, S.R. (1998) Building the Data Warehouse, Communications of the ACM, 41, 9, 52-60.
9. Hanson, J.H. and Willshire, M.J. (1997) Modeling a faster data warehouse, International Database Engineering and Applications Symposium (IDEAS '97), 260-265.
10. Inmon, W.H. (2002) Building the Data Warehouse, 3rd Edition, John Wiley.
11. Inmon, W.H., Terdeman, R.H., Norris-Montanari, J., and Meers, D. (2001), Data Warehousing for E-Business, 1st Edition, John Wiley.
12. Jensen, C.S. Introduction to Temporal Database Research, http://www.cs.aau.dk/~csj/Thesis/pdf/chapter1.pdf
13. 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.
14. 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, 45-54.
15. 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.
16. Ozsoyoglu, G. and Snodgrass, R. (1995) Temporal and Real-Time Databases: A Survey, IEEE Transactions on Knowledge and Data Engineering, 7, 4, 513-532.
17. Rahman, N. (2007) Refreshing Data Warehouses With Near Real Time Updates, Journal of Computer Information Systems, spring 2007, 71-80.
18. Rahman, N. (2005) Intelligent Metadata Model in a Teradata Warehousing Environment, Annual Teradata Partners User Group Conference and Expo, Walt Disney World Swan/Dolphin Resort, Orlando, FL, USA, September 18-22, 2005.
19. 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.
20. Shin, B. (2003) An Exploratory Investigation of System Success Factors in Data Warehousing, Journal of the Association for Information Systems, Vol. 4.
21. Stonebraker, M., Cetintemel, U. and Zdonik, S. (2005) The 8 Requirements of Real-Time Stream Processing, SIGMOD Record, 34, 4.
22. 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, 83 - 102.

23. TimeConsult (2008) What is Temporal Data?, http://www.timeconsult.com/TemporalData/TemporalData.html, Retrieved on April 19, 2008
24. Torp, K. (1998) Implementation Aspects of Temporal Databases, http://www.cs.aau.dk/NDB/phd_projects/torp.html, Retrieved on April 20, 2008.
25. Torp, K., Jensen, C., and Snodgrass, R. (2000) Effective timestamping in databases, The VLDB Journal, 8, 3-4, 267-288.
26. Vavouras, A., Gatziu, S. and Dittrich, K. (1999) Modeling and Executing the Data Warehouse Refreshment Process, International Symposium on Database Applications in Non-Traditional Environments (DANTE '99), 66-73.
27. 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.
28. 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, 389-404.
29. Yang, J. and Widom, J. (2001) Incremental computation and maintenance of temporal aggregates, In Proc. Int. Conf. on Data Engineering (ICDE '01).
30. Yufei Tao, Y., Papadias, D., and Faloutsos, C. (2004) Approximate temporal aggregation, In Proc. Int. Conf. on Data Engineering (ICDE '04).
31. 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, 237 - 245.