Real Time Data Warehousing using Dynamic SQL

Sweety V. Batavia

Abstract- Data warehouse synchronization is a complex process in heterogeneous database environment. Maintaining the uniformity of data in real-time is a fundamental problem of data synchronization. In this paper we present a methodology to synchronize data warehouse in heterogeneous database environment in almost real-time using dynamic SQL approach. We capture all DML SQL query from source database and pass this to the processing module. Structure and DBMS systems at the source and target could be different; hence it is not possible to execute the source SQL directly into the target. A processing module runs continuously in the back end to read changes and prepare the dynamic SQL by referring the Reference Data. Reference Data acts as heart of the processing module that prepares the dynamic SQL. It specifies the mapping between source and target with all details. Another process executes the SQL query in the target system. If any error is encountered while executing the query in the target, it will be moved to the error processing module where it will be retried after some delay. Our preliminary experimental results evidence the effectiveness of the proposed method.

Keywords – real-time data warehouse synchronization, dynamic SQL, heterogeneous database, SQL query capture

I. INTRODUCTION

One of the most intricate parts of building any data warehouse is the process of extracting, transforming, cleansing, and loading (ETL) the data from the source system. This process is usually done in batch mode and typically involves downtime of the data warehouse, so no users are able to access it while the load takes place. As today's decisions in the business world become more real-time, the systems that support those decisions need to keep up. Hence, it is very important that Data Warehouse provide real-time view of data. Performing ETL of data in real-time introduces additional challenges. When loading data continuously in real-time, there can't be any system downtime, it need to cater to Heterogeneous database environment between source and target, a single change in source can impact multiple tables in the target. Need of maintaining the summary (pre-aggregated) data in data warehouse. It also needs to provide effective error and exception handling for no data loss in the process. The method should be scalable to handle large volume that may come. In this paper, we will explore the problems of maintaining the real-time data warehouse. To solve these problems, we are using dynamic SQL approach. Here, based on the changes made in the source database, we will analyze the impact of it using the reference functional map and then prepare the SQL on the fly that will be fired to the target database to sync it with the source database. The reference functional map stores the mapping between source and target. These details are stored at the table and column level.

Manuscript published on 30 June 2013.

* Correspondence Author (s)
Sweety Batavia, Department of Computer Engineering, Mumbai University, India.

It is very powerful concept and can provide the mapping of source vs. target in case of various actions like insert/update/delete in the source and any transformation needed on the source data. In dynamic SQL approach, the queries to be fired at the target are not known in advance. The queries are dynamically built at the run time based on the event at the source. The SQL built will be fired directly at the target hence there is no dependency on the source database. This will help a lot in case when there are multiple sources of the warehouse. The other big advantage of dynamic SQL technique is that it is very light weight and consumes very less resources. Hence multiple threads of the app can be run. The reference functional data will be cached at the start of the application hence the process will take almost no time to generate the SQL to be fired at the target. This can help us reduce the data warehouse synchronization time lag to less than one or two second of the actual event. In this paper, we focus only on the incremental refresh of the warehouse. This paper does not deal with the full refresh of the warehouse at the beginning of the time.

II. PRESENT STATE OF THE PROBLEM

Below diagram shows the typical synchronization method of warehouse database.

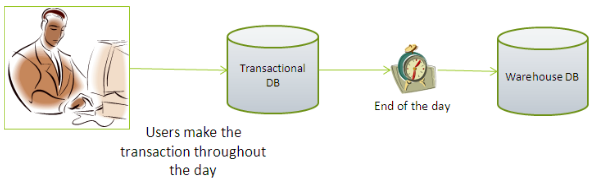

Figure 1 : Typical synchronization method of warehouse database

User uses the application and makes the transaction throughout the day across the globe. Traditionally transactions and changes are applied to the source database at the end of the day or in regular intervals.

One of the techniques is to replicate the data [1][8] across servers. This is often done in homogeneous environment. Database replication is traditionally handled in two ways, either with eager replication or with lazy replication [2]. Eager replication implies an atomic commitment protocol and is slow and deadlock prone. It also needs the structure of two database exactly same. Lazy replication is efficient, but does not enforce consistency between the replicas. This can lead to data quality issues in the target server. View Synchronization [3][4][5] is used to refresh the materialized view. This technique uses materialized view to keep the data up-to-date. Materialized view keeps pre-computed data based on the view definition. It updates itself as soon as changes are done at the base tables. This is very useful when the materialized view resides on the same server as of the source server. In most cases, warehouse servers are geographically different from the transactional one.

Analysis has been on the existing methods used by most of the organization for the data warehouse synchronization. I have analyzed the model in few firms like Amazon and Banking industry firms. This approach mainly involves syncing the data warehouse from the operational database in periodic basis [7] which is end of the day in most cases. Recently, due to major changes in the information systems and business environments, senior management and business users need the data warehouse to be as fresh as possible with highest quality. Techniques mentioned above fail to meet these criteria. Recently, there have been many researches to device the technique that can help us keep data warehouse up-to-date. Some of these recent studies suggest SQL based data synchronization [9], ETL tools available in the market [10][11][12]. These studies suggest using SQLs or ETL tools like Informatica Power Center. SQL based synchronization [9] suggests use pre defined SQLs to update the warehouse based on the source data changes. Here predefined SQLs are stored that will be run periodically to keep the data between source and target in sync. This can be better than many other methods we have discussed so far. However, it also fails to solve few issues like data quality and stale data. There are various tools available in the market can serve partial purpose of data warehouse synchronization in real-time. These tools are having few basic flows as they either do not support heterogeneous environment or they are not well configurable or there can be potential data quality issues due to lack of proper error handling.

We face following problems or issues with this synchronization approaches explained above.

Stale data – as we can see that warehouse is updated at time interval. This means that any changes happening at the transactional database will not have immediate effect in the warehouse data. This leads to out dated data in warehouse. Any user running the report will not be able to get the latest picture of the data.

Consumes huge resources – Current synchronization process like end of the day process often becomes quite heavy because it has to process all the data changes throughout the day together. This can consume huge resources like CPU, Servers, network bandwidth and memory. Due to this, the routine reporting and data mining/reporting work get impacted severely.

Long system downtime - As it updates huge volume of the data together. Reporting and data mining processes can not run in parallel to the synchronization process. If they run in parallel, it will result into even slower response time. If reports try to consume data that is in process of updating, reports may either encounter deadlock or the data fetched by the reports could be partially updated. Hence the data quality of the reports can not be relied. It is undesirable situation and hence all the reporting work is suspended till the data load is running and warehouse becomes inaccessible hence downtime is needed.

Not able to cater to heterogeneous database – various tools and methods explained above can not work on the heterogeneous environment. Techniques like replication or tools provided by Oracle work only on the homogeneous systems. They do not work when source and target are of different types. In most data warehouse cases, source and targets are of different types. It is also possible that there is more than one source of the data. These limitations make the specialized tools less useful in the real life scenarios.

Requires extensive coding – as the table structure and requirement of the source transactional and target warehouse database would be very different, they is a need to translate and enrich the source data before it is loaded into the target. All the business logic to translate the source data into the warehouse data has to be coded in the ETL tool. This needs experts and expensive developers to code all the business logic needs to be part of the ETL process.

Difficult to change – In most of the systems, changes are continuously made due the dynamic business scenarios. When this is done, we need to make sure that all the dependencies are also taken care well. If all business logic is maintained in the code, it means that any small change in the source or target databases needs lot of changes in the code. This can lead to instability if the code change impacts other modules too.

III. METHODOLOGY

After studying and considering the shortcomings of the existing methods used in the industry and some of the research work on the topic, I have tried to address the issues using much robust and scalable design. Here is the proposed architecture to sync the warehouse database in real-time and to address the problem discussed above.

The main improvement in the architecture is in the processing module (#7). The proposed design is divided into multiple modules. Each module is designed for some specialized need. When these modules are put together, it gives one complete solution with many advantages. It also allows us to reuse the modules or add multiple instances of these modules in parallel to make it more efficient.

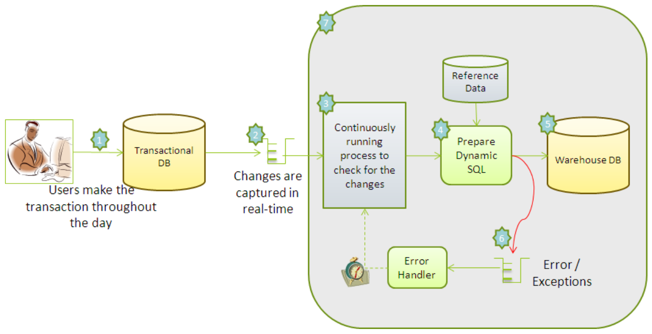
7
1
Transactional DB
Users make the transaction throughout the day
2
Changes are captured in real-time
3
Continuously running process to check for the changes
Reference Data
4
Prepare Dynamic SQL
5
Warehouse DB
6
Error / Exceptions
Error Handler

Figure 2: Proposed Architecture to synchronize warehouse database

#1, 2 – Change Data Capture (CDC) – This module mainly resides next the transactional database. Its main purpose is to listen to the changes being done in the source system. Any changes made in the transactional database are captured as soon as they are made. This is mainly provided by the database vendor as form of replication or RTDS (Real time data service). The changes are written to Message Queue (MQ) or file system with the complete details.

#3, 4, 5 - Transform, Enrich and Load using dynamic SQL – Before we load the data into the warehouse, we need to cleanse the data and then enrich it. We also need to transform the source data into the format that can be easily applied to the target. We are going to discuss this module in detail in the “PROCESSING MODULE” section.

#6 - Error and Exception handling – when the changes are being processed, there are chances that we may encounter some error or issues. This can be due to any locking or temporary issue with the database. If we encounter any error or exception while updating the target, message will be moved to error handler. This error handler will also have retry logic. The problematic messages will be retried after certain delay. If it fails again, it will be marked as failure so that administrator of the system can take necessary actions.
#7 - Scalability – The system is highly scalable. If required, multiple instances of the entire module can be run in parallel to take care of the large volume of changes.
By using the dynamic SQL approach and the Reference Data explained above, we can address the issues with the data warehouse refresh very well. The delay between source and target is not more than 2-3 seconds which makes the data warehouse almost in sync with the source. The processing module is very light in resource utilization. It needs to cache the Reference data and then have to keep building the dynamic SQL as and when changes are done at the source. Hence the scope of processing at one time is just one message at a time. This makes it very light. Because of this, we do not need any downtime as our process can run in parallel with the regular users of the warehouse. In the proposed process, we are not dependent on the source as we only rely on the message published by CDC module. Hence we are able to cater to any number and nature of source and target. This addresses the issues of existing methods not able to cater to the heterogeneous database. The proposed technique is driven more on configuration (reference data) than coding. Once the module preparing the dynamic SQL is built, we just need to maintain the reference data well. Hence, it makes it very easy to maintain and any future changes.

IV. PROCESSING MODULE

Reference Data acts as a core of the processing module while generating the dynamic SQL. One update in operational database may affect multiple tables in warehouse database. It helps identify what all tables need to be updated at the target for any changes made in the source. The reference data also maps the source column with target column and transformation needed in the source data. It stores these detail in such manner so that it can help derive the dynamic SQL for any DML statements at the source. The reference data acts as a heart of the next module i.e. preparing the dynamic SQL.
Dynamic SQL module will refer to the reference data and along with the message received from the “Change data capture” module. It will first identify what exactly has changed from the message. Based on that, it will iterate through the reference data to derive the impact.
As a single change in operational database results in multiple changes in warehouse database, multiple SQL queries are required to be generated referring the referenced data. For different RDMS’s there will be some mismatches in the syntax of SQL query for data types and functions in target database. Using the Reference data, it will map the source with target. This mapping is defined at table and column level. It also maps if any transformation of source data is needed or not.
After preparing the dynamic SQL query, it executes changed query in the target database. All the queries prepared for a single change at the source are put in a single database transaction. Hence if we encounter any error while processing any SQL, entire transaction will be rolled back. This will help us keep the data consistency at the target. If execution succeeds, it will be moved to “Archive” directory. On the other hand, if it fails, it will be moved to error handling module where it will be retried after certain delay.
Example –
A new row is inserted in the source as below. The new row and reference data for the source table sales_master is given below.

Table 1 – sample row inserted in sales_master table.

Table	**sales_master**		
Column	**Salesman_e_id**	**Product_id**	**Sales_amount**
	E123	P234	500

Table 2 - Sample XML message that will be delivered to us by “Change data capture” module-

```
<dbEvent eventId="uniqueeventID">
            <insert tableName="sales_master">
                        <values>
                                    <column
columnName="sales_e_id">E123</column>
                                    <column
columnName="sales_amount">500</column>
                                    <column
columnName="product_id">234</column>
                        </values>
            </insert>
</dbEvent>
```

Table 3 – example of reference_data_master

Table	**reference_data_master**		
Column	**mapping_id**	**source table**	**target table**
	1	sales_master	total_sales_by_salesman
	2	sales_master	total_sales_by_product

Table 4 – example of reference_data_detail

Table	**reference_data_detail**						
Column	**mapping _id**	**source column**	**target column**	**transformation**	**source action**	**target action**	**target_primary_ column_flag**
	1	sales_amount	total_amount	%sales_amount%	INSERT	INSERT	No
	1	salesman_e_id	employee_id	%salesman_e_id%	INSERT	INSERT	Yes
	1	sales_amount	total_amount	total_amount = total_amount + %sales_amount%	INSERT	UPDATE	No
	1	salesman_e_id	employee_id	%salesman_e_id%	INSERT	UPDATE	Yes
	2	sales_amount	total_amount	%sales_amount%	INSERT	INSERT	No
	2	product_id	product_id	%product_id%	INSERT	INSERT	Yes
	2	sales_amount	total_amount	total_amount = total_amount + %sales_amount%	INSERT	UPDATE	No
	2	product_id	product_id	product_id	INSERT	UPDATE	Yes

Based on this, the dynamic SQL will be prepared as below and the same will be fired at the target.

Dynamic SQL built for the target table total_sales_by_salesman –
If not exists (select * from total_sales_by_salesman where employee_id = “E123”) then INSERT INTO total_sales_by_salesman

(employee_id, total_amount) VALUES (“E123”, 500) ELSE UPDATE total_sales_by_salesman set total_amount = total_amount + 500 WHERE employee_id = “E123”
Dynamic SQL built for the target table total_sales_by_product –
If not exists (select * from total_sales_by_product where product_id = “P234”) then INSERT INTO total_sales_by_product (product_id, total_amount) VALUES (“P234”, 500) ELSE UPDATE total_sales_by_product set total_amount = total_amount + 500 WHERE employee_id = “P234”

V. SCOPE OF IMPROVEMENT

The proposed design is highly scalable. As the volume of the data increases everyday, we can scale up this design to meet the growing needs. Multiple instances of the entire module can be run in parallel to take care of the large volume of changes. This can be easily achieved using multi-process server component. It is also possible to introduce a load balancer in between the process. This load balancer will divide the incoming flow of the changes bases on a pre-defined logic. Here, there will be only one module that will keep of collecting the changed data. This module will then pass the changes to load balancer. Based on the load balancing algorithm put in the balancer, it will group the changes logically. This grouping can be based on anything. E.g. it can be based on the type of change or set of tables containing same kind of data (client data, employee data, revenue data etc.). Then the changes will be routed to the respective change processing units. In this, multiple instances of the change processing units will be listening to the load balancer. As soon as load balancer assigns the task to the processing unit, changes will be processed.

VI. CONCLUSION

I have demonstrated methodology for maintaining uniformity in operational database and warehouse database in heterogeneous database environment. I have researched on uniformity maintenance of data and its structure before starting the real time data synchronization among the heterogeneous database systems. So far as my knowledge, this is the first implementation of real time data warehousing using Dynamic SQL query. In this research, I have designed a new methodology. The main advantages of this method are that it does not depend on vendor of database and keeps the warehouse database up-to-date, supports heterogeneous databases.

REFERENCES

[1] Matthias Wiesmann and Andre´ Schiper, “Comparison of Database Replication Techniques Based on Total Order Broadcast”, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 17, NO. 4, APRIL 2005

[2] J.N. Gray, P. Helland, P. O’Neil, and D. Shasha, “The Dangers of Replication and a Solution,” Proc. 1996 Int’l Conf. Management of Data, pp. 173-182, 1996.

[3] Amy J. Lee, Anisoara Nica, and Elke A. Rundensteiner, “The EVE Approach: View Synchronization in Dynamic Distributed Environments”, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 14, NO. 5, SEPTEMBER/OCTOBER 2002

[4] E.A. Rundensteiner, A.J. Lee, and A. Nica, “On Preserving Views in Evolving Environments,” Proc. Fourth Int’l Workshop Knowledge Representation Meets Databases (KRDB’97): Intelligent Access to Heterogeneous Information, pp. 13.1-13.11, Aug. 1997.

[5] J. Widom, “Research Problems in Data Warehousing,” Proc. Int’l Conf. Information and Knowledge Management, pp. 25-30, Nov. 1995.

[6] Y. Zhuge, H. Garcı´a-Molina, J. Hammer, and J. Widom, “View Maintenance in a Warehousing Environment,” Proc. SIGMOD, pp. 316-327, May 1995.

[7] Isabel Cristina Italiano and João Eduardo Ferreira, “Synchronization Options for Data Warehouse Designs”, 0018-9162/06 © 2006 IEEE Publ i s h ed by t h e IEEE Computer Society

[8] Claudio Basile, Zbigniew Kalbarczyk, and Ravishankar K. Iyer “Active Replication of Multithreaded Applications”, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 17, NO. 5, MAY 2006

[9] Md. Iqbal Hossain and Muhammad Masroor Ali, “SQL Query Based Data Synchronization in Heterogeneous Database Environment” in conference on IEEE Computer Communication and Informatics (ICCCI), 2012

[10] Informatica. PowerCenter. In the Web, available at: http://www.informatica.com/products/powercenter/

[11] “Oracle GoldenGate,” http://www.oracle.com/.

[12] “WisdomForce WisdomForce DatabaseSync Real Time Change Data Capture and Replication,” http://www.wisdomforce.com

[13] X. Yu, “The application study of heterogeneous database synchronization,” in International Conference on Electronic Computer Technology (ICECT), Kuala Lumpur, 2010

[14] W. Xiaoli and Y. Yuan, “Xml-based heterogeneous database integration system design and implementation,” in 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), 2010, Chengdu, 2010

[15] “Dbmoto: Real-time data replication and data integration using snapshot, incremental and synchronization replications,” http://www.hitsw. com/products services/dbmoto/dbmoto dsheet.html.

Prof. Sweety Batavia, Faculty, Department of Computer Engineering, Mumbai University, has completed her degree of Bachelor of Engineering in Computer Science from Saurashtra University. She has been contributing in academics from six years. Areas of research interest are database management, data mining, networking and operating systems.