Towards handling incremental load for anomalies in near real time data warehouse

MOHAMMED MUDDASIR N
Dept. of IS&E,
VVCE, Mysuru, Karnataka,
India
mdmsir@gmail.com

RAGHUVEER K
Dept. of IS&E,
NIE, Mysuru, Karnataka,
India
raghunie@yahoo.com

R DAYANAND
Technical Director,
India
Dayanand_7@yahoo.com

Abstract:-Refreshment anomalies occur in a data warehousing environment while performing Extract Transform and Load (ETL) to get the data for analysis from sources. There could be several reasons for the anomalies like not able to capture the delta on time, system time out, duplicate entries due to outer join operations and many more. Once anomalies are detected the compensation operation is executed to get the data that was missing into the data warehouse. In this work we would like to analyze scenario where it is necessary to perform incremental loads based on priority in an ongoing data warehouse maintenance work. The work proposes a novel approach to decide on when to perform ETL so that refreshment anomalies do not occur and to maintain integrity of data such that analytics queries always provide right information to the analyst. Two novelties have been discussed in this work one is to have a threshold before compensation of updates and two is while performing compensation updates prioritize the query with less freshness interval to have more time limits for the updates to be completed.

Key-words: -Data warehouse, Extract Transform Load, Incremental Load, Refreshment anomalies, query priority.
Received: June 16, 2020. Revised: October 20, 2020. Accepted: November 5, 2020. Published: December 7, 2020.

1 Introduction

Real time analysis is a new normal in various domains that include e-commerce, banking, retails, finance etc. Every business is trying to analyze the data available with them as part of getting to know the good bad about their products or services in the sight of customers or end users. Data collected through transactions or web site visits or feedbacks etc. is not suitable for analysis. This data has to be cleaned, transformed and later used for analysis. The process of getting transactional data into analytical environment (data warehouse) is known as ETL. Traditionally ETL was done during off peak hours but the demand for real time analysis has made ETL also in real time. But the very instant of capture data could not be present for analysis and there is a slight delay to get the data for analysis in real time, hence the process is known as near real time ETL. Initially while building the data warehouse the ETL is done on the entire data available at the transaction site, but after the initial load new data added at the transaction site are added using incremental loads. The reason is the incremental loads take less time because of fewer volumes to capture only the deltas to be loaded that were added to transaction site after initial loads. After the first incremental load, second, third and all subsequent loads are incremental in nature. The incremental load in case of near real time ETL could lead to several anomalies. There could be missing or incorrect values; delays in capturing the deltas could leave the two databases in an inconsistent state. This work focuses on resolving the inconsistent database states and to make sure anomalies due to delays in capture of deltas is resolved. The idea to capture deltas before answering the queries was proposed in [1][2]. Also the idea to have threshold of number of rows or any other similar value before performing ETL was proposed in [3]. In this work we combine these two ideas, i.e. we put the query on hold put capture deltas only if it reaches a threshold value. Also a novel approach to assign priority to a particular query is proposed that helps to execute

queries with very less freshness interval. This work has an algorithm to make sure not all queries are stalled and only those that have a source data reaching a threshold value along with a freshness criterion are only put on hold. This makes sure that in case of multiple queries being executed in parallel, queries have got priority and in prioritized queries also not every query is put on hold. The overall execution time of all the queries is reduced but not stalling every query. The experiment were done on TPC-DS data results show less time for query execution with minimal difference in source and destination databases. The motivation for this work is to reduce query waiting time and also to give updated data by making only critical queries to stall for getting updated data.

2 Related Works

In this section we review the existing work on anomaly detection and resolution in a data warehouse environment. Earlier the data warehouses were updated based on information received from source about the deltas being added [4]. In this scenario data warehouse would query the source requesting it to send the updates. The anomaly here is in case of data warehouse query being answered there could be concurrent deltas added at the source leading to insertion anomalies or deletion anomalies. The solution proposed by the authors of [4] is to have eager compensation algorithm (ECA). In ECA using the concept of database joins and set differences between the source and destination tables is used to avoid anomalies due to inter leaving of queries and updates. Similar anomalies could be possible in case of multiple sources, this was handled in [5] using strobe family of algorithm. The strobe algorithm prepares a list of actions to be executed to bring the data warehouse in a consistent state. View maintenance using minimal join of delta from one table with available data warehouse table is proposed in [6]. The algorithms are named SWEEP and Nested SWEEP dealing with multiple sources that are updated during different times and each is autonomous. The idea is to join delta of one table that is updated with stored value in data warehouse of another table. Materialized view maintenance using version numbers was proposed in [7]. The idea is only highest version number in the processing queue gets to be updated at the data warehouse. Hence all the intermediate updates leading to anomalies are avoided. Sirius approach [8] deals with various refreshment issues by mapping source and destination databases using object oriented concepts. The issue of heterogeneity is handled in their work by developing a global schema that consists of meta-data information. Anomalies due to outer joins are discussed in [9]. The problem of redundancy could occur if the data warehouse are were build using outer joins to keep track of records that do not match the join conditions. If at some point in time the match occurs there could be redundant tuple one with null values and one without null value. The one with null values have to be identified and deleted. Correction strategies for distributed updates is discussed in [10]. The problem identified is current schema changes with record updates and several updates taking different times to finish. The solution is an agent to keep track of all the changes and later once every change is tracked update the data warehouse. Incremental updates that are safe are only propagated through a algorithm to identify and remove rows that are created as side effects is proposed in magic set safe updates[11]. Problem of delay is capturing deltas and it solution is proposed in [12]. ETL solution such as snap shot source, time stamp source are proposed to overcome refreshment anomalies. Change data propagation to compensate for the limitation of change data capture techniques is proposed in [13][14]. Change propagation logs with source tables queried for changes using equivalence preserving transformation rules as a solution is their main contribution. Other techniques such as incremental view maintenance [15] and 24/7 high availability[16] are also dealing with incremental data warehouse maintenance.

Various strategies problems and solution for incremental view maintenance were proposed previously. Compared to previous work the approach in this work proposes a hybrid approach to stall query as proposed in [1][2] and update deltas only if certain threshold is reached as proposed in [3]. The novelty in this work is in calculating freshness interval and to decide query priority. The assumption is query details must be available as part of meta-data.

In our previous work on refreshment anomalies [17] We had implemented the idea of stalled query for getting updates. As compared to this work in our previous work the updates were stored in temporary tables, here we have a novel approach to calculate the threshold before stalling, later to have freshness interval to determine whether it's required to stall all

queries on only some finally to limit the time given for performing the updates.

3 Decide on incremental loading cycle.

Refreshment anomalies in ETL occur if there are changes in the source that are not available to the data warehouse during query execution. Availability of real time data for analysis at the data warehouse depends on two main factors, freshness criteria, and data integrity. Freshness criteria is called as when part of incremental loading [8] helps to decide the right interval or time for performing the refreshment. This could be 1.regular interval 2.based on threshold values calculated using availability of data, information gain of data and 3.number of queries that require data from this source. The regular interval set for performing ETL, is normal time frame set to perform incremental load that could refreshes the data warehouse every day, hour and any time set by the administrator. The threshold based refreshment could be triggered when the source data itself reaches a threshold value in terms of count or information it provided. If the number of queries are limited and hit only at certain period then refreshment could be done at regular interval. If the queries hit are varied across time and may need real time data then refreshment cannot be at regular interval it has to be done using delta updates mechanism proposed in [1][2]. Before answering the query execution starts put it on hold and update the relevant tables using the deltas available in the source. Data integrity is the availability of right information without any missing values or failures in the process of ETL [18]. Integrity is achieved using ETL testing to make sure there are no logical errors from joins or job failures due to slow query execution and duplicates in case of slowly changing dimensions.

4 ETL based on threshold value

The idea of deciding to perform ETL based on non-static condition like number of records updated etc. was proposed in [3]. If data reaches a threshold value and queries are more for this data then instead of waiting for refreshment interval perform ETL immediately. Even if threshold value is reached and no queries then do not perform ETL wait for the interval to perform the next incremental load. If queries are more and threshold value is less do not perform ETL and again wait for the next interval to perform incremental load. Threshold value could be the number of new rows available or it could be domain specific value such as max sales, average sales etc. Number of queries is just the count of how many analytics task are performed on the data warehouse. The delta updates proposed in [1][2] is enhanced with threshold criteria in deciding to perform refreshment or not. Also the threshold value helps in maintaining the data integrity so that we could cross check on the count of number of new rows refreshed.

5 Prioritize query to reduce execution time

Once it is decided to perform delta updates based on threshold value next consideration is how much time is given to perform the ETL so the query that is waiting for the compensated records is not over loaded. Normal ETL is done by round robin or round robin by blocks in case of multiple sources. If there is a query marked as critical then based on freshness cycle the following is done. Critical query are checked for refreshment of data. If data is having freshness interval of 1 day or more means avoid ETL and execute the query. If freshness interval of data source for the query is less then put query on hold and perform ETL and then later after this answer the query that shall have latest information.

To illustrate the above taking e-commerce as example, there could be three possible sources, structured, semi-structured and un-structured. The structured source could be the sales transaction by a customer, semi-structured could be the click stream analysis and un-structured could be review comments etc. While a ETL system takes three sources like the above and refreshment is done in a round robin fashion to get updates on each, the refreshment time given to each source depend on the freshness criteria of the data desired. In case of sales the freshness is high and hence the refreshment interval i.e. the time given for the structured source to finish is high, then the time given for click stream analysis source is moderate and the time given for refreshment of unstructured source is less. This helps to bring in most recent data for analysis in case of critical requirement.

The query is put on hold and ETL is performed to make sure the query does not miss any new updates, while doing so if the ETL takes a lot of time then the query execution that is put on hold shall become overwhelmed. To avoid this certain limit has to be set

based on the type of query. High priority query is given more waiting time to finish the delta updates, although this leads to more latency but it gives updated records. Low priority query may not need updated records but requires to finish with less latency hence waiting times is low.

Priority is decided based on type of analysis. Example three types of sources structured, semi-structures, and unstructured are used for analysis. Each contains different information like structured could be sales data how many unit of item sold, semi-structured would be click stream analysis, demo graphics of the user, unstructured could be feedback. If the analysis query is for sales then structured source is given more time to complete delta updates, if click stream analysis then semi-structured source is given more time, and if feedback analysis is to be done then unstructured source is given more time to complete.

In case of multiple queries for the same type of analysis are being executed. To decide upon the priority of queries the freshness interval is taken as a parameter. If the freshness interval of the sources for the query is very low then priority of the query is high. Low freshness interval is set for the source that is being constantly updated and those updates are required for critical analysis. If there are n numbers of queries the most critical query is the one with less freshness interval. The Algorithm makes sure high priority query gets the most recent for critical analytical requirement.

The above is the idea of prioritize the query based on type of analytics and freshness criteria. Table1 illustrates the meta-data for the idea presented. In this meta-data hypothetical values are used for structured data as having less freshness interval with high priority and hence given more time for refreshment.

Table1: Meta data for refreshment

Query	Source	Freshness interval	Priority	Refreshment Time
1	Structured	Less	High	High
2	Semi –structured	Moderate	Moderate	Moderate
3	Un-Structured	High	Low	Low

If at a given point in time query analyzing the un-structured data source is being hit, also no other query is being hit at this time, then to put the query in hold and perform ETL is decided based on the content of the Meta data. Check for freshness interval and compare with the last refreshment for example if the freshness interval is 1 hour and last refreshment was done within 1 hour then the query need not be put on hold and directly executed on the available data. Similar rules are applied for other types of sources. This mechanism does not put all the queries on hold and starts ETL. The queries with less freshness interval are only put on hold.

6 Algorithm stalled query update

Step1: Begin

Step2: Query execution on data warehouse received

Step3: Stall the query

Step4: Check threshold value at source about the number of rows count changed

Step5: Count>Threshold? Go-to Step6 else; Go-to Step11

Step6: Check Priority and Freshness

Step7: Freshness is high? Go-to Step8 else; Go-to Step11

Step8: Update the source

Step9: Watch update time> set limit? Go-to Step10 else; Go-to Step11

Step10: Stop Update

Step11: Execute Query

Step12: End

7 Implementation and Results

The simulation of different types for sources was done using a standard bench mark tool. The implementation was done using TPC-DS bench

mark[19] data with a scale factor of 1 producing data of size 1GB on a machine with core i5 having 4GB RAM as configuration. TPC-DS contains various types of sources and related ad-hoc analytic queries. As shown in the fig1 [19]below we have taken three sources such as catalog sales, web sales and store sales and executed relevant queries that access these sources. The bench mark for having structured, semi-structured and unstructured data at once was not found hence TPC-DS having all structured sources is taken for demo. The experiments show priority of the query based on threshold and freshness criteria as explain in the previous sections.

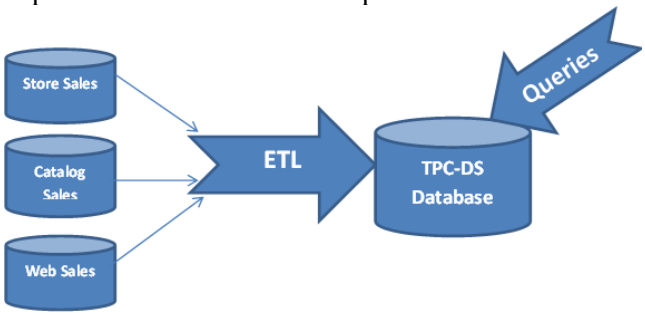

Figure1: Partial TPC-DS sources and queries

The simulation requires meta-data for knowing the priority based on freshness. The threshold value of available deltas at source in terms of count of rows is calculated dynamically upon query execution. The TPC-DS bench mark give 99 queries and we have taken randomly 3 queries each related to store sales, catalog sales and web sales as shown Table2; the table shows the hypothetical values of freshness for the query number 6 through 82.

Table2: Implementation meta-data

Sales Type	Query Number	Freshness
Store Sales	6	1
Store Sales	27	1
Store Sales	82	1
Catalog Sales	15	2
Catalog Sales	18	2
Catalog Sales	20	2
Web Sales	12	3
Web Sales	45	3
Web Sales	62	3

Freshness is calculated valued based on difference between Previous Query Execution Completion Time (PQECT) and Current Query Execution Start Time (CQEST). If the difference is above a set threshold value then the query is stalled delta updates take place before the start of execution. In Table2 the freshness for web sales is shown as high because threshold value of difference between PQECT and CQEST is set has low. The reason for a setting a low threshold value is that the sources for web sales are updated at a rate higher that the update rate of catalog sales and store sales.

Initially every source had data to the tunes of few lakh rows on which related queries were executed. Later incremental load of 2lakh rows each for catalog sales and store sales was done. Web sales had an incremental load of 3lakh rows. A delay of 6 seconds was uniformly assumed for incremental loads. Web sales also had a high freshness value as shown in table2. This made the source web sales as candidate for stalling the execution as compared to catalog sales and store sales. The results show the time in milliseconds for execution of queries in the meta-data table2. First are the catalog sales we could see there is delay when the queries are stalled for delta updates, similar results are seen for store sales and web sales. Fig2 show catalog sales query execution similarly Fig3 and Fig4 show store sales and web sales query execution. The overall idea to stall only those queries with a less freshness interval which is web sales in this case is done by only stalling web sales queries and naturally the execution time of overall query suit has improved as show in Fig5.

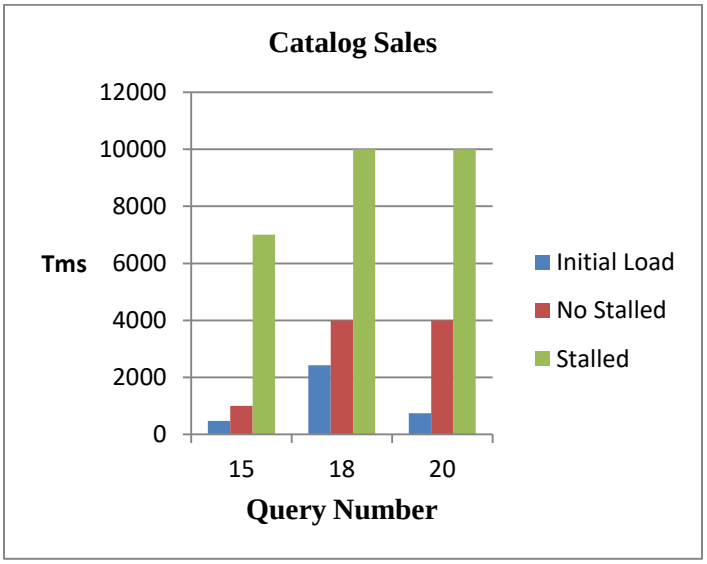

Figure2: Catalog Sales Query Execution

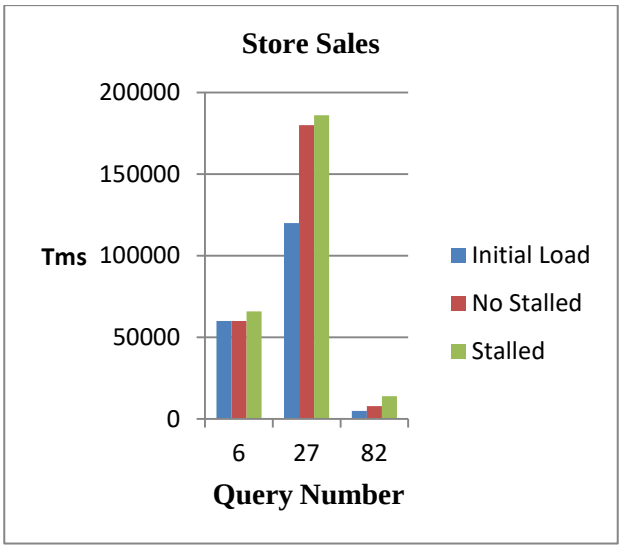

Figure3: Store Sales Query Execution

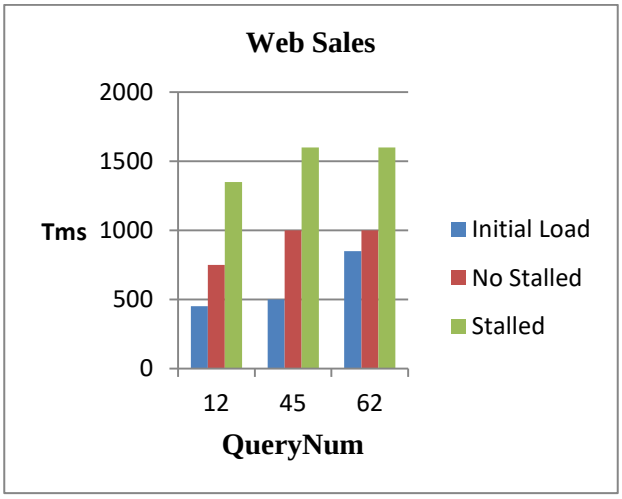

Figure4: Web Sales Query Execution

8 Discussions

The proposed approach has reduced the overall query execution time from a given suite of queries. As seen in the result every query is not stalled and only web sales queries are stalled that makes no query waiting time for stores sales and catalog sales. Advantage is not all the queries are stalled; disadvantage is that we need build a meta-data for queries by recording previous execution history to use the algorithm. Building meta-data dynamically is proposed as a future enhancement.

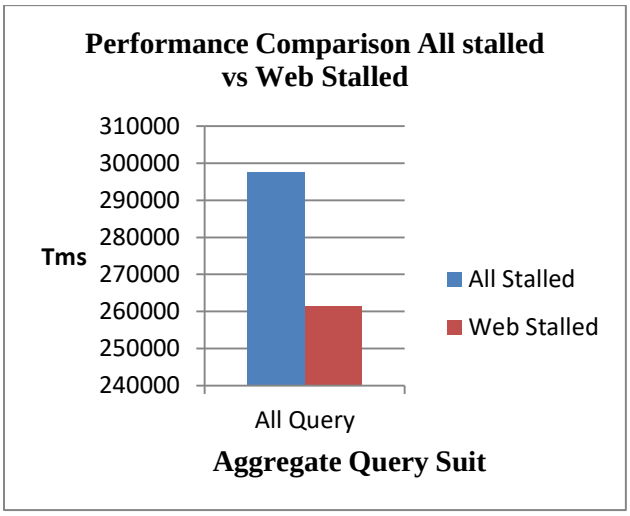

Figure5: Overall Query Suit Execution

9 Conclusion and future enhancement

The assumption in this work is queries are known in advance and their detail such as query number is known. If a new query apart from the once stored in meta-data arrives then the algorithm does not work until the query information is inserted in to the meta-data table. Also every query execution has to be tracked from the meta-data table to know the priority. Stalling a query requires it to be present in the meta-data after which the threshold is calculated and later deltas are updated based on priority given by freshness attribute. As the result show there is an improvement in overall query execution time for the entire suit of queries present in the meta-data. To overcome the limitation of having to store query information in meta-data. In future the enhancement is of adding the new queries to meta-data dynamically are so that even new queries could be handled effectively for refreshment anomalies shall be taken up.

References:

[1] and S. D. Weiping Qu, Vinanthi Basavaraj, Sahana Shankar, "Real-Time Snapshot Maintenance with Incremental ETL Pipelines in Data Warehouses," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9263, pp. 28–39, 2015, doi: 10.1007/978-3-319-22729-0.

[2] W. Qu, "Incremental ETL Pipeline Scheduling for Near Real-Time Data Warehouses 1," no. Btw, pp. 299–308, 2017.

[3] L. Chen, W. Rahayu, and D. Taniar, "Towards near real-time data warehousing," *Proc. - Int. Conf. Adv. Inf. Netw. Appl. AINA*, pp. 1150–1157, 2010, doi: 10.1109/AINA.2010.54.

[4] Y. Zhuge, H. García-Molina, J. Hammer, and J. Widom, "View maintenance in a warehousing environment," in *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, 1995, pp. 316–327, doi: 10.1023/A:1008698814840.

[5] Y. Zhuge, H. Garcia-Molina, and J. L. Wiener, "Consistency Algorithms for Multi-Source Warehouse View Maintenance," *Distrib. Parallel Databases*, vol. 6, no. 1, pp. 7–40, 1998, doi: 10.1023/A:1008698814840.

[6] D. Agrawal, A. El Abbadi, A. Singh, and T. Yurek, "Efficient view maintenance at data warehouses," pp. 417–427, 1997, doi: 10.1145/253260.253355.

[7] T. W. Ling and E. K. Sze, "Materialized view maintenance using version numbers," *Proc. - 6th Int. Conf. Database Syst. Adv. Appl. DASFAA 1999*, pp. 263–270, 1999, doi: 10.1109/DASFAA.1999.765760.

[8] A. Vavouras, S. Gatziu, and K. R. Dittrich, "Modeling and executing the data warehouse refreshment process," vol. 2000, no. January, pp. 66–73, 2003, doi: 10.1109/dante.1999.844943.

[9] A. G. ; I. S. H. V. J. Mumick, "Data Integration Using Self-Maintainable Views," in *Advances in Database Technology — EDBT '96*, 1997, pp. 9–19, doi: 10.14220/9783666565441.9.

[10] Z. Shu, S. Li, Y. Zuo, X. Zhou, and Y. Tang, "Correction strategy for view maintenance anomaly after schema and data updating concurrently," *Proc. 9th Int. Conf. Comput. Support. Coop. Work Des.*, vol. 2, no. 031542, pp. 1046–1051, 2005, doi: 10.1109/cscwd.2005.194333.

[11] A. Behrend and T. Jörg, "Optimized incremental ETL jobs for maintaining data warehouses," *ACM Int. Conf. Proceeding Ser.*, pp. 216–224, 2010, doi: 10.1145/1866480.1866511.

[12] T. Jörg and S. Dessloch, "Near real-time data warehousing using state-of-the-art ETL tools," *Lect. Notes Bus. Inf. Process.*, vol. 41 LNBI, pp. 100–117, 2010, doi: 10.1007/978-3-642-14559-9_7.

[13] T. Jörg and S. Deßloch, "Towards generating ETL processes for incremental loading," *ACM Int. Conf. Proceeding Ser.*, vol. 299, pp. 101–110, 2008, doi: 10.1145/1451940.1451956.

[14] T. Jorg and S. Dessloch, "Formalizing ETL Jobs for Incremental Loading of Data Warehouses," *Datenbanksysteme Business, Technol. Web*, pp. 327–346, 2009.

[15] X. Zhang, L. Yang, and D. Wang, "Incremental view maintenance based on data source compensation in data warehouses," *ICCASM 2010 - 2010 Int. Conf. Comput. Appl. Syst. Model. Proc.*, vol. 2, no. Iccasm, pp. V2-287-V2-291, 2010, doi: 10.1109/ICCASM.2010.5620479.

[16] R. J. Santos, J. Bernardino, and M. Vieira, "24/7 real-time data warehousing: A tool for continuous actionable knowledge," *Proc. - Int. Comput. Softw. Appl. Conf.*, pp. 279–288, 2011, doi: 10.1109/COMPSAC.2011.44.

[17] N. Mohammed Muddasir and K. Raghuveer, "A Novel Approach to Handle Huge Data for Refreshment Anomalies in Near Real-Time ETL Applications," *Advances in Intelligent Systems and Computing*, vol. 1154. pp. 545–554, 2020, doi: 10.1007/978-981-15-4032-5_50.

[18] H. Homayouni, S. Ghosh, and I. Ray, *Data Warehouse Testing*, 1st ed., vol. 112. Elsevier Inc., 2019.

[19] M. M. Susanne Englert, "Transaction Processing Performance Council (TPC) www.tpc.org info@tpc.org Legal Notice," 2018.