Proposed Techniques to Optimize the DW and ETL Query for Enhancing data warehouse efficiency

Abhishek Gupta[1], Arun Sahayadhas[2]
PhD Scholar[1], Application Development Senior Analyst[1], Professor[2]
Dept of Computer Science and Engineering[1,2]
Vels Institute of Science, Technology and Advanced Studies, [1,2], Accenture Services Pvt Ltd[1]
Chennai, India[1,2]
abhishekresearchsecrets@gmail.com[1] , arun.se@velsuniv.ac.in[2]

***Abstract*— To improve time-bound decision capability of any business corporates, they have to perform data-analysis based on historical and current data. For handling this data, data warehouse (DWH) plays a crucial role. Further-more as time passes, the data also grows exponentially, because in data warehouse historical data also needs to be maintained. If we further drill now-a-days Extract Transform and Load (ETL) tools are being used to perform extract, transform and loading of data. These ETL tools are based on any specific language like in case of Informatica ETL tool, which is based on Structured Query Language (SQL) language. So, it's imperative to enhance the efficiency of these languages to enhance the capacity of ETL tools and this will significantly impact the performance of data warehouse ETL work, and by this data can also be provided in faster manner for analysis and reporting purposes. Additionally, data quality is also an important factor, as data-analysis will be performed on the same data supplied by data warehouse. So, it's highly concerned to improve the data quality by applying data-quality checks before supplying for data-analysis purpose. So, we have worked in this direction and gone through various research papers which are mainly focused on query optimization and data quality improvements. After going through all research papers, we have not only summarized the performance tuning tips given in these papers but also worked and experimented various things, and then we have proposed some unique techniques for query optimization and improved data quality techniques. After which we have incorporated it in data warehouse to enhance overall performance of data warehouse. These proposed techniques can be applied in any place wherever queries are being used to extract/transform and/or load the data to enhance the performance.**

Keywords—Data Warehouse, Incremental Load, Indexing Techniques, Query Optimization, Slowly Changing Dimensions, Statistics generation.

I. INTRODUCTION

Today in the world of data-science, whole business data either its current or historical data is handled by data warehouse. So, data warehouse plays a very crucial role in data-handling. Data warehouse is a central repository which contains business current as well as historical data, which it sourced from heterogeneous data-sources and after cleansing/transforming it is moves for the reporting and analytics purpose. To maintain this whole process data warehouse architectures are designed. In [1] a comprehensive survey has been performed to enhance efficiency of data warehouse. This paper has formalized steps to design data warehouse and has performed survey on the techniques that can be used to optimize efficiency of data warehouse. In the figure 1, the basic enterprise data warehouse architecture has been showcased:

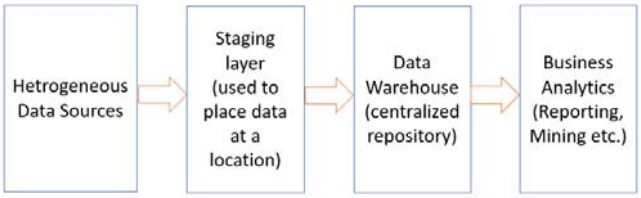

Figure 1. Enterprise Data Warehouse

In data warehouse whole data needs to move from heterogeneous data sources to heterogeneous targets after proper scrubbing or transformation of data. This whole process of Extraction, Scrubbing and Loading is called as ETL process. Data gets extracted from the targets using various business analytics tools, which are used for various purposes like to generate reports, for data mining and other works.

There are so many ETL tools available now-a-days in the market for performing ETL operations efficiently. Out of these tools the most used ETL tool is Informatica. It has been observed that all ETL tools are built on some specific language which can interact with the database and would be able to extract the data like Informatica ETL tool is designed based on SQL. By optimizing these languages on which these ETL tools are dependent, the operational speed of the tools can be greatly improved. As this is a very vast area and we can't showcase optimization tips for all languages, we have limited this paper to showcase SQL optimization tips.

The optimization of queries, can be performed using below steps:
1. Check for the execution plan cost of the query, for which the optimization needs to be performed.
2. Work on the optimization of execution plan cost, as much as the execution plan cost is optimized that much the query will also be optimized.

There are so many researches have been performed by various researchers to optimize of execution plan cost. We have performed a deep study and observed that Indexing can be used to enhance the query performance, but there are so many indexing algorithms are available. It's required to choose a proper indexing algorithm, which will be helpful to increase the performance. In [2,11] comparative study on these indexing algorithms have been performed. But these papers are biased on the indexing algorithms only and by only optimizing indexing it can not be assured that query is fully optimized, sometimes it happens that even after applying indexing queries does not become optimized a little. So, it is required to work on other performance tuning tips, which are

978-1-7281-9180-5/20/$31.00 ©2020 IEEE

defined in these papers. Apart from indexing algorithms there are some other query performance tuning tips that have been explained in [3,5,6,7,8,9,12]. These papers have summarized various tips through which the query optimization can be performed. But excessively/ aggressively usage of performance tuning tips can also generate overhead on the query and the query will become more complex to handle and difficult to understand sometimes. Which makes maintenance of query a difficult task. Rather than that it is better to follow the steps of optimization which have been explained above (optimize the execution plan cost). As after experimenting we have observed that without even using any performance tuning tips and by just changing the sequence of tables in join (table having less data can be placed first followed by table having more data), performance can be improved at greater extent. Like this example other things can be performed which should aim to optimize the execution plan cost, and the tuning tips that have been explained in these papers can be used as an add-on to further optimize the execution plan cost.

Query optimization is one of aspects for which we should work, but still there are other aspects also which should not be ignored. Out of those aspects, one aspect is writing accurate query. A tiny mistake in query can generate wrong output. These mistakes generally occur when NULL or Blank data is present in the storage tables. It's highly imperative to use NULL and Blanks properly, else this can generate erroneous data. By proper handling of this data accuracy can be enhanced greatly. Second aspect is data security to make sure that data warehouse/target which contains business sensitive data, should not be accessible by all. Third aspect is maintaining the historical data using data warehouse, where it is maintained with the help of slowly changing dimensions (SCD) concepts. The implementation concept for slowly changing dimensions has been explained in [4,10,13]. These papers can be referred while implementing SCD types in data warehouse design. These papers are biased on SCD types but have not accompanied the concept that how to use these SCD types wisely. There is still requirement to work on formalization of a generalized concept that can be used on universal basis.

II. Proposed Techniques

1. For maintaining historical data in data warehouse, slowly changing dimensions are used. Out of these slowly changing dimensions most frequently used dimensions are SCD Type 1/2 and 3. Which have been elaborated in figure 2,3:

Supplier Details in DWH

Supplier_key	Supplier_code	Supplier_State	Effective_start_date	Effective_end_date
500	XYZ	CA	20-Jul-2020	31-Dec-9999

Supplier Details Modified coming from Source

Supplier_code	Supplier_State	Effective_start_date
ABC	IL	30-Aug-2020

Figure 2.Data in DWH and Modified data from Source

SCD Type I

Supplier_code	Supplier_State
ABC	IL

SCD Type II

Supplier_key	Supplier_code	Supplier_State	Effective_start_date	Effective_end_date
500	XYZ	CA	20-Jul-2020	30-Aug-2020
501	ABC	IL	30-Aug-2020	31-Dec-9999

SCD Type III

Supplier_key	Supplier_code	Supplier_Code_ Previous	Supplier_Code_ current	Supplier_State
500	ABC	XYZ	ABC	IL

Figure 3. Slowly Changing Dimensions Type 1/2/3

Each SCD types have their own pros and cons. So, we have proposed to use 3 data warehouses that will be segregated based on SCD types and as per the data requirements, queries can be applied to the specific data warehouse for analysis and debugging purpose and significant results can be obtained with less time. As query needs to handle less or significant amount of data.

2. We have proposed to use the Incremental load in the data warehouse. So that only limited amount of data needs to be taken from the source itself, as it will be filtered based on specific parameters/columns like date at the source level itself.

3. We Should always use the Indexed data warehouse as well as indexed data marts. Along with this we also need to decide that which kind of and volume of data we will be going to use, so accordingly we should choose the appropriate indexing algorithm. A sample example of Indexing is shown in figure 4:

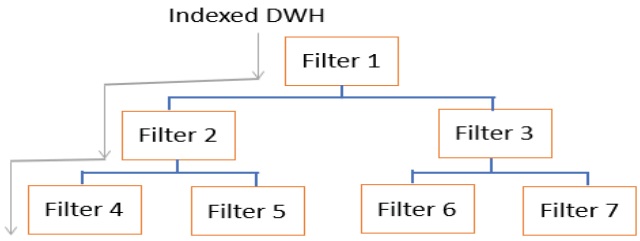

Figure 4. Indexed Data Warehouse

Data warehouse architecture for the point 4,5 and 6 have been illustrated in figure 5:

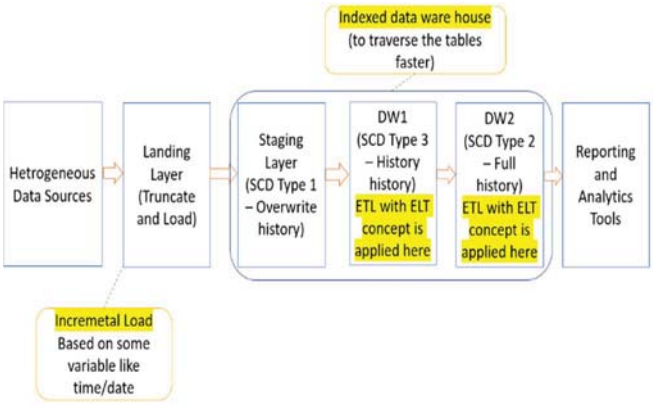

Figure 5. Proposed Data Warehouse Design

4. We should always use the parameterized queries wherever necessary. Which not only gives freedom of user-interaction ease in the complex queries handling but also helps in data-security concerns. For example, the parameterized values can be passed to triggers and can be checked that what necessary actions needs to be taken. It has been showcased using an example in figure 6:

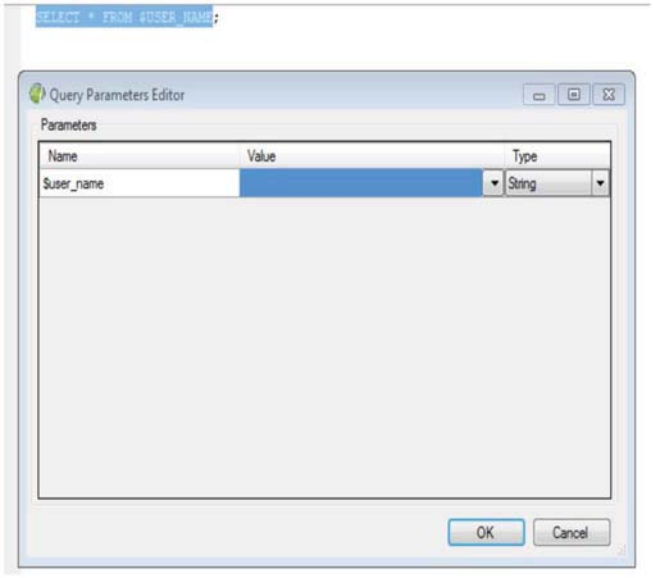

Figure 6. Parameterized Query

5. In some-places it's imperative to have the columns as unique values to treat it as the primary key but these columns are not the primary keys at physical level, so, this rules get broken when we have to use the that table as driving table and another table as secondary table being used as left join and having more than one values for those columns for which we were looking it as unique. On that time, we can handle it by concatenating the columns given in the secondary table and can make the driving table columns unique. This concept has been illustrated with an example in figure 7:

Driving Table

Col1	Col2
100	A
101	B

Secondary Table

Col1	Col3	Col4
100	C	X
100	D	Y

left joined based on col1

Output, we have concatenate COL2 and 4

Col1	Col2	Col3	Col4
100	A	C,D	X,Y
101	B	NULL	NULL

Figure 7. Example of Concatenation of data

But after that some-times we have the requirements like to take the specific value from column 4 based on column 3. Which makes the task difficult as first we need to check for the exact place where the data is situated in column 3 and then we need to check respective value in the column 4. To handle this issue, we have implemented the universal rule for the same, at this time we are handling this at informatica expression level, that can be easily converted to SQL or any other language. The expression has been showcased in figure 8:

```
1  IIF(COL3 = 'C',COL4,
2  IIF(REG_MATCH(UPPER(COL3),'C,.*'),
3  SUBSTR(COL4,1,INSTR(COL4,',')-1),
4  IIF(REG_MATCH(UPPER(COL3),'.*C'),
5  SUBSTR(COL4,INSTR(COL4,',',-1)+1,LENGTH(COL4)),
6  IIF(
7  REG_MATCH(UPPER(COL3),'.*,C,.*')
8  AND LENGTH (SUBSTR(COL3,1,INSTR(COL3,'C')))
9  > LENGTH(REG_REPLACE(SUBSTR(COL3,1,INSTR(COL3,'C')),',',',')),
10 SUBSTR(COL4,INSTR(COL4,',',1,LENGTH (SUBSTR(COL3,1,INSTR(COL3,'C')))
11 - LENGTH(REG_REPLACE(SUBSTR(COL3,1,INSTR(COL3,'C')),',',',')))+1,
12 (INSTR(COL4,',',1,(LENGTH (SUBSTR(COL3,1,INSTR(COL3,'C')))
13 -
14 LENGTH(REG_REPLACE(SUBSTR(COL3,1,INSTR(COL3,'C')),',',',')))+1)-
15 INSTR(COL4,',',1,LENGTH (SUBSTR(COL3,1,INSTR(COL3,'C')))
16 -
17 LENGTH(REG_REPLACE(SUBSTR(COL3,1,INSTR(COL3,'C')),',',',')))-1
18 ),
19 ''
20 ))))
```

Figure 8. De-concatenation algorithm

6. Always try to retrieve data from the views wherever there is possibility that data can be changed based on time or rules. So, for example if a specific view say view1 is being used at more than 10 places. So, it's not practical approach to reload or modify data in all places, rather than that it would be preferable to use the views that will be refreshed automatically, when the main table gets modified. This concept works more effectively where the view is being generated based on more than 1 table, as it places the data at one place like a table, and we don't have to work on join queries to extract data from more than 1 table as the view will automatically handle it.

7. Whenever the query is written, we must ensure that NULLs and Blanks are being handled properly. For example if we are doing join on two tables and we want to make sure that NULLs and Blanks should be treated as the same then we should not write data directly it can be re-written as "NVL(column,'')", so NULL will also be converted to blanks and can be treated as same. Apart from that SQL does not allow make condition as "NULL = NULL", so practically when we put join on tables based on columns having NULL values SQL filter out the data and doesn't show those values as output of query. So, this concept of NVL can also handle this issue by putting columns having NULLs with NVL, which will replace NULLs with specified values and match the records and show those records also as a output. So, this concept also supports data loss prevention techniques.

8. We have proposed a technique which can reduce the snow-flake schema to star schema, using a coded value, in which the binary-coded values will have a fixed length as per the requirements, and connected to the dimension table in the form of star schema. We have showcased it using an example, in figure 9 and 10:

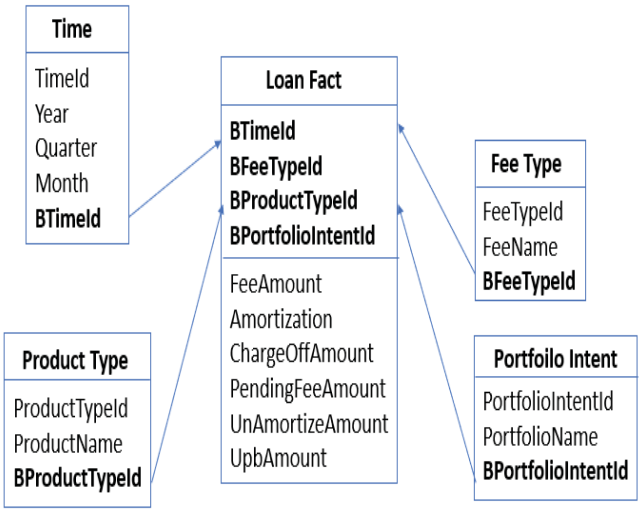

Figure 9. Proposed Snowflake to Star schema conversion

TIMEID	EFFECTIVE YEAR	EFFECTIVE QUARTER	EFFECTIVE MONTH	BTIMEID
1	2000	1	1	00010010001
2	2000	2	2	00010100010
3	2000	3	3	00010110011
...	...	...	...	...
365	2008	4	12	10011001100

Figure 10. Example of Binary-coded data

Here the BTIMEID column has been coded as below for 2008.4.12, which is the combination of Byear = 2008, Bquarter=4, Bmonth=12 columns: -

Binary code for year: 2008 – 2000 + 1 = 9; Binary code for 9 = 1001
Binary code for quarter: Binary code for 4 = 100;
Binary code for month: Binary code for 12 = 1100
The combined binary-coded value of 2008.4.12 is 10011001100

9. After analysis we have found that if statistics/stats are applied on the tables being used in the data warehouse then the efficiency of query can be enhance drastically as stats directly impact the execution plans, which are used to optimize the queries effectively. So, it's imperative to always have the stats updated. After that analysis we have found that there are no specific algorithms are available to or derived for updating the Stats on the table. Furthermore, we also have analyzed that only writing stats update on table is not a good practice, while working on stats update on any table it is required to consider other required factors too. So, after taking those factors into consideration, we have proposed an algorithm to update stats on specific tables whenever the queries are fired on that using algorithmic steps given in figure 11:

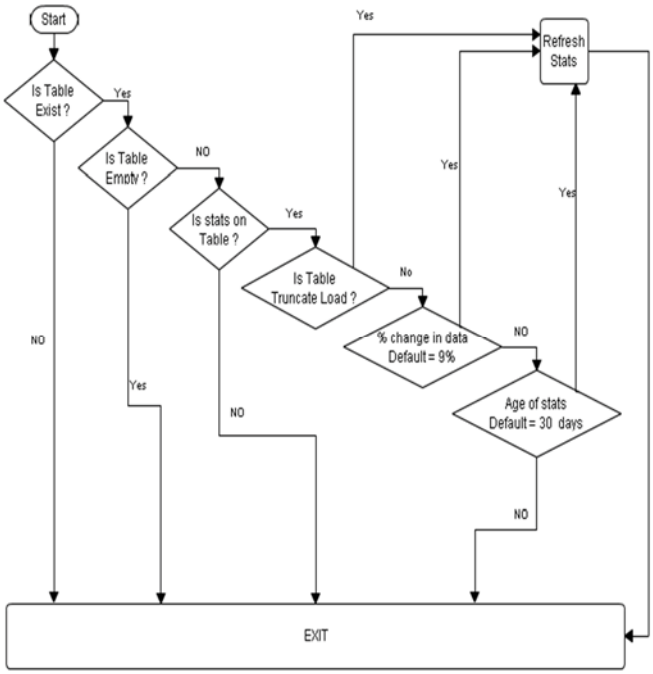

Figure 11. Statistics update algorithm

10. We should always try to execute the queries parallelly, whenever and wherever it's possible. We have proposed an algorithmic way to do this, illustrated below as an example:

Say the query, which we want to execute is as below:
SELECT empid, deptid, AVG (Salary) FROM Employee GROUP BY empid, deptid HAVING AVG (Salary) < (SELECT MAX (AVG (salary)) FROM dept WHERE empid IN (SELECT empid FROM emp-history WHERE Dept-id BETWEEN 100 AND 200) GROUP BY empid, deptid);
So, the tables which are being used in this are: Employee, dept, emp-history. Then We have written an algorithm which will segregate the queries during the pre-processing stage. So, this query will also be segregated into three subqueries based on the presence of commas, brackets, key words, relational expressions mentioned and functions, which are as follows: -

Sub query 1: SELECT empid, deptid, AVG (Salary) FROM Employee GROUP BY empid, deptid HAVING AVG (Salary) <
Sub query 2: SELECT MAX (AVG (salary)) FROM dept WHERE empid IN
Sub query 3: SELECT empid FROM emp-history WHERE Dept-id BETWEEN 100 AND 200 GROUP BY empid, deptid
After pre-processed sub-query set generation, we need to identify the objects and attributes of each sub query which are as follows:
Sub query1: Base table -1; Base table name - Employee; Attributes used-2; Attribute names- empid, Salary; Functions used- 1
Sub query2: Base table -1; Base table name - dept; Attributes used-1; Attribute names-salary; Functions used-1
Sub query3: Base table -1; Base table name - emp-history; Attributes used-1; Attribute name-dept-id;

After this we need to make the query-tree, and remove the un-necessary clauses, by which all three queries can run

independently and can generate the results independently. After generating the results, we again have to re-check and re-arrange the queries based the dependency-rules, as below:

SELECT empid FROM emp-history WHERE Dept-id BETWEEN 100 AND 200;
SELECT MAX (AVG (salary)) FROM dept WHERE empid IN (sub query3) GROUP BY empid, deptid;
SELECT empid, deptid, AVG (Salary) FROM Employee GROUP BY empid, deptid HAVING AVG (Salary) < (sub query1);

III. Analytical Results

This paper has been designed after thorough study and analysis of various research papers. In this paper we have not only discussed about the performance tuning tips proposed by these papers can be used to tune the query but also, we have proposed some new and unique techniques through which queries be tuned more efficiently. In this paper we have proposed a data warehouse architecture which uses the incremental load as well this architecture has incorporated SCD type 1/2 and 3. We have used SCD type 1/2 and 3, to make data entries possible in three ways and as per the requirements queries can be fired on specific DW to grab the required data from specific DWH. So, this operation will limit the handling of datasets and performance can be increased. Further-more we have proposed to use indexing algorithms and stats on tables efficiently as per the requirements in data warehouse. Further-more we have proposed to use the parameterized queries, which will enhance data-security as well as makes the human interaction possible and ease the process to handle big and complex queries. Additionally, we have proposed to use the views on the top of main tables, so if data varies with time then views will be refreshed automatically and will require less operations need to be performed. We also have introduced that how to convert snow-flake schema to star schema, which will reduce the number of joins and so, the query will become less complex. Along with this we have also introduced that NULLs and Blanks should be handled effectively along with an example else the query can draw the wrong results. We also have proposed concatenation and de-concatenation operations to maintain uniqueness of any specific column. We have also proposed an algorithm to run the query parallelly, which significantly boost the query performance due to parallelism.

IV. Conclusion

In this paper we have showcased the significance of Data warehouse for business corporates. After which we have introduced about various terminologies used in data science, which is imperative to understand the paper in more meaningful way. After which we have showcased importance of languages like SQL, which are used as back-bone to perform various operations in data warehouse. By optimizing these languages ex. SQL, the efficiency of data warehouse operations can be greatly enhanced. So, for this purpose we have studied various papers and literatures and discussed about the performance tuning tips presented/ Proposed by various researchers. Apart from that we also have proposed some techniques through which we can get favorable results. The techniques which are being demonstrated here can also be less or more can be applied to other languages also to achieve performance tuning.

V. Future Scope

In this paper we proposed some new techniques to enhance query optimization. Using these tips and techniques the future data warehouse designs can become more efficient. Further-more there is still future-scope to work on real-time data aspects, to make real-time data-based designs more efficient.

References

[1] Gupta, Abhishek. (2020). A Comprehensive Survey to Design Efficient Data Warehouse for Betterment of Decision Support Systems for Management and Business Corporates. International Journal of Management (IJM), IAEME Publication, Volume 11, Issue 7, July 2020, pp. 463-471.

[2] Hashim, R.T. (2019). A Comparative Study of Indexing using Oracle and MS-SQL Server for Relational Database Management Systems.

[3] Pooja Wankhade , Dr. Vaishali Deshmukh. (2019). Comparative Analysis of Query Optimization Techniques in Database Systems. International Journal of Science & Engineering Development Research (www.ijsdr.org), ISSN:2455-2631, Vol.4, Issue 3, page no.515 - 519, March-2019

[4] Bhide et al. (2016). SLOWLY CHANGING DMENSION ATTRIBUTES IN EXTRACT, TRANSFORM, LOAD PROCESSES. United States Patent, (10) Patent No.: US 9,311,368 B2, Date of Patent: Apr. 12, 2016

[5] Thangam, A.R., & Peter, D.S. (2016). An Extensive Survey on Various Query Optimization Techniques.

[6] Habimana, J.D. (2015). Query Optimization Techniques - Tips For Writing Efficient And Faster SQL Queries. International Journal of Scientific & Technology Research, 4, 22-26.

[7] Gupta, S., Tandel, G.S., & Pandey, U. (2015). A Survey on Query Processing and Optimization in Relational Database Management System.

[8] Johnson, T. (2013). A Study on the Role of Equivalence Rules in the Enhancement of Query Performance.

[9] Navita Kumari, 2012, Sql Server Query Optimization Techniques – Tips For Writing Efficient and Faster Queries, International Journal Of Scientific and Research Publications, Volume 2, Issue 6.

[10] Braden et al. (2012). SYSTEMS AND METHODS FOR STORING AND QUERYING SLOWLY CHANGING DIMIENSIONS. United States Patent, Patent No.: US 8,260,822 B1, Date of Patent: Sep. 4, 2012

[11] Gupta, Manoj & Badal, Dharmendra. (2012). COMPARATIVE STUDY OF INDEXING TECHNIQUES IN DBMS.

[12] Ziani, B., & Ouinten, Y. (2011). Improving Star Join Queries Performance: A Maximal Frequent Pattern Based Approach for Automatic Selection of Indexes in Relational Data Warehouses. 2011 International Conference on Internet Computing and Information Services, 119-122.

[13] Griffin et al. (2005). METHOD OF MANAGING SLOWLY CHANGING DIMENSIONS. United States Patent, Patent No.: US 6,847,973 B2, Date of Patent: Jan. 25, 2005.