© 2015 IJSRSET | Volume 1 | Issue 3 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Query Optimizer for the ETL Process in Data Warehouses

Bhadresh Pandya[1], Dr. Sanjay Shah[2]

[1]Professor, Department of Computer Science, Kadi Sarva Vishwavidyalaya, Gandhinagar , Gujarat, India

[2]Director, S. V Institute of Computer Studies, Kadi, Gujarat, India

ABSTRACT

ETL (Extraction-Transformation-Loading) process is responsible for extracting data from several sources, cleansing, transforming, integrating and loading into a data warehouse. Extraction process accesses large amount of data by executing several complex queries in source databases. These queries are repetitive and executed at regular interval to refresh the data warehouse. Extraction of data from source must be completed in a certain time window; hence it is necessary to optimize its execution time. In this paper, we delve into the optimization of queries by recommending indices which reduces cost of the queries and improves performance of the queries.

Keywords: Extraction Transformation Loading, Data Warehouses, Query Optimizer, Business Intelligence, Execution Plan of Query, Database Tuning, Query Tuning, Performance Tuning

I. INTRODUCTION

In the area of business intelligence, data warehouse plays an important role. Designing and implementing a data warehouse requires using different tools and techniques to be used and applied to effective implementation and maintenance of data warehouse. The category of tools which are responsible for extraction of data from several source systems, their cleansing, transformation and inserting them into a data warehouse are called ETL tools. Execution time of these ETL process need to be optimized so that it can be completed in specified time window [1].

The functionality of ETL tools involve prominent tasks which include a) the identification and extraction of relevant information from source systems b) transformation and integration of information extracted from several source systems into a common format c) cleaning these data as per the database and business rules d) loading quality assured data to the data warehouse. The design and implementation of these tasks in the ETL process is labor-intensive activity which consumes large portion of the data warehouse projects [6].

Data extraction process involves execution of complex queries on the relational databases of operational source systems having large amount of data. In this paper, we propose the heuristic algorithm which works on certain key parameters of the query such as tables accesses, columns accesses, conditions applied on the columns, and using statistics of tables and columns it determines optimal index set to be built which generated better execution plan and reduces the cost of query.

II. METHODS AND MATERIAL

Data warehouses have become integral part of decision making process for the businesses. To provide enterprise level view of data for analysis of different functions, data from all the operational systems need to be cleaned, transformed and integrated at enterprise level. There is no de facto standard for architecture and construction of data warehouses. However quality and completeness of data coming from different heterogeneous sources play a major role in success of data warehouse along with flexible and scalable architecture.

Enterprise database applications are often characterized by a large volume of data and high demand with regard to query response time and transaction throughput. Beside investing in new powerful hardware, database tuning plays an important role for fulfilling the requirements. However, database tuning requires a thorough knowledge about system internals, data characteristics, applications and the query workload. Among others index selection is a main tuning task. The problem is to decide how queries can be supported by creating indexes on certain columns [13].

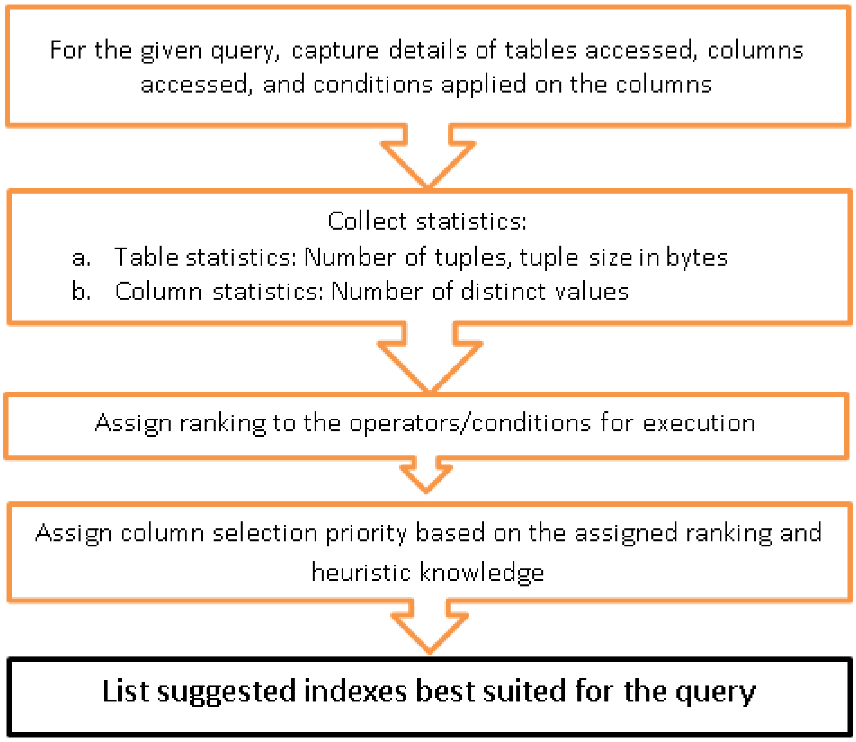
The problem of low performance in data extraction of ETL process in the data warehouse can be critical because of the major impact in using the data from data warehouse, the project can be compromised. In this case there are several techniques that can be applied to reduce queries' execution time and to improve the performance [12].

Reducing I/O

Locating some selected records from the large tables based on the selection criteria is a common task of extraction process. Deriving the result set efficiently from the operational databases is often difficult due the complex nature of the data and query. The simplest way of evaluating a query is to do a full scan of the tables and apply specified conditions which induces higher disk I/O. The most commonly used indexing method is the B-Tree which is very effective for on-line transaction processing (OLTP). Almost every database product has a version of this indexing method. As disk I/O operation is time consuming database operation, focusing on the efficiency of disk I/O is an effective means for improving performance and scalability [10].

The task of improving performance of query and building suitable indexes is done by database administrator, according to his knowledge and expertise. This is both subjective and quiet hard to achieve when number of queries is very large [11]. Proposed algorithm simplifies this task by recommending indexes which result in better execution plan reducing I/O significantly.

The Index Recommending Heuristic Algorithm

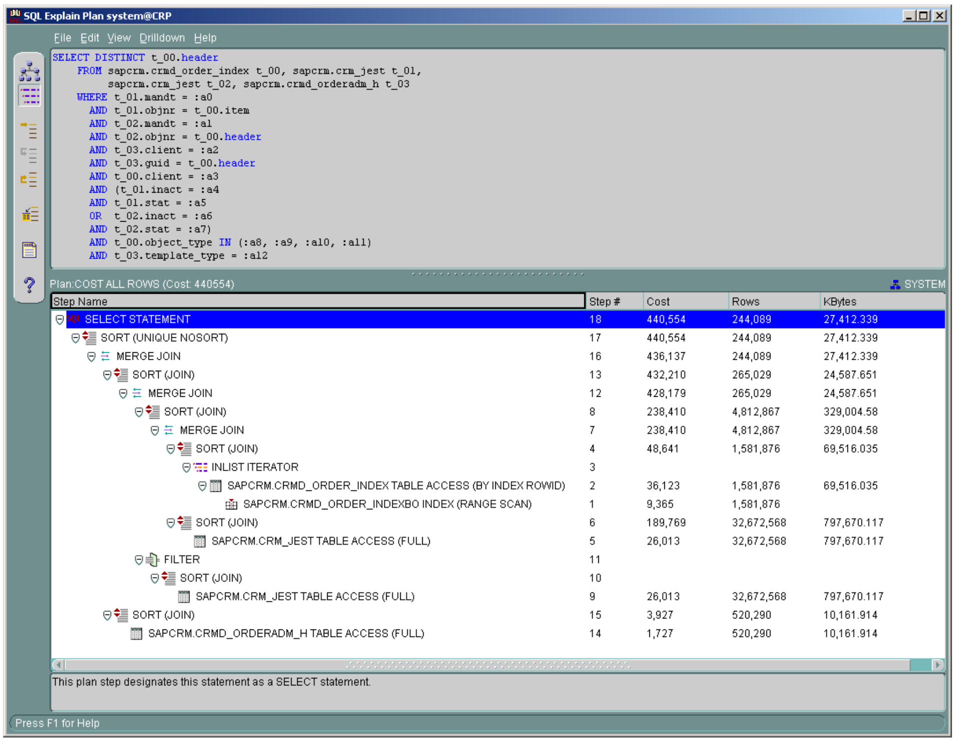


For the given query, capture details of tables accessed, columns accessed, and conditions applied on the columns
Collect statistics:
a. Table statistics: Number of tuples, tuple size in bytes
b. Column statistics: Number of distinct values
Assign ranking to the operators/conditions for execution
Assign column selection priority based on the assigned ranking and heuristic knowledge
List suggested indexes best suited for the query

Figure 1: Heuristic algorithm for index recommendation

III. RESULTS AND DISCUSSION

The algorithm was tested on real-time large databases on different modules in real-application environment. Examples of execution plan before putting recommended indexes in place and after creating are as below:



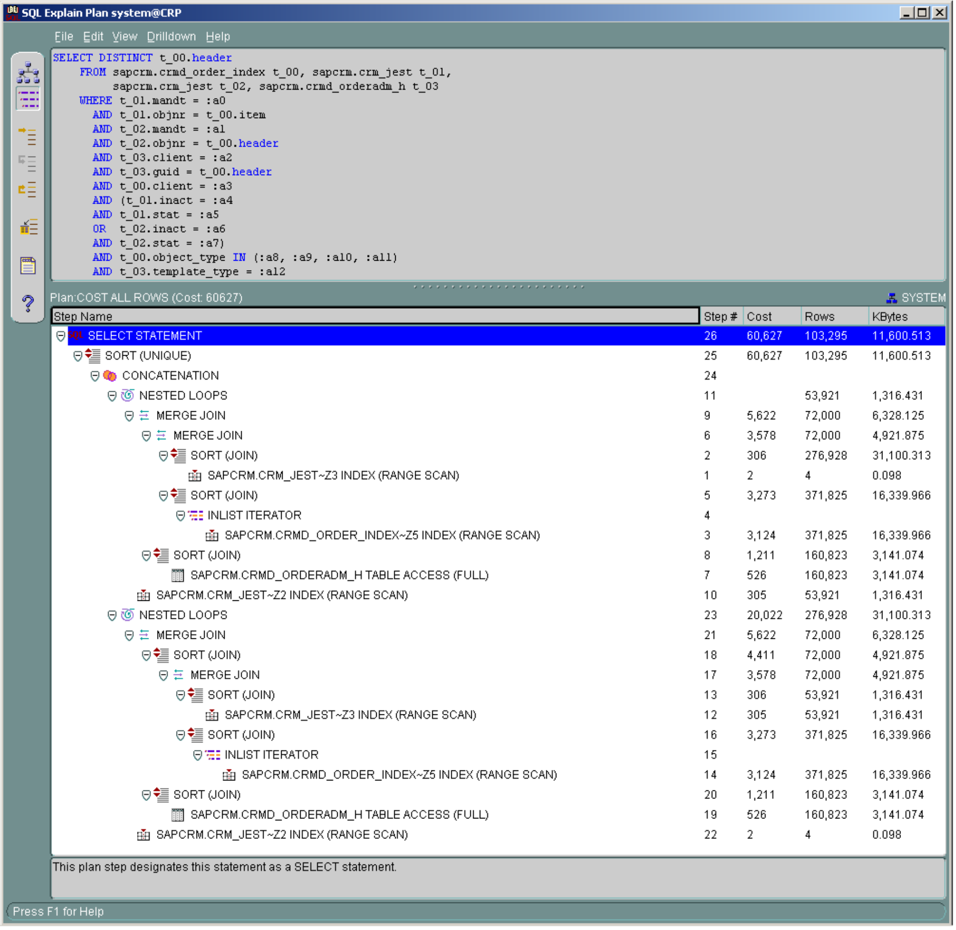
```
SELECT DISTINCT t_00.header
    FROM sapcrm.crmd_order_index t_00, sapcrm.crm_jest t_01,
         sapcrm.crm_jest t_02, sapcrm.crmd_orderadm_h t_03
    WHERE t_01.mandt = :a0
      AND t_01.objnr = t_00.item
      AND t_02.mandt = :a1
      AND t_02.objnr = t_00.header
      AND t_03.client = :a2
      AND t_03.guid = t_00.header
      AND t_00.client = :a3
      AND (t_01.inact = :a4
      AND t_01.stat = :a5
      OR  t_02.inact = :a6
      AND t_02.stat = :a7)
      AND t_00.object_type IN (:a8, :a9, :a10, :a11)
      AND t_03.template_type = :a12
```

Plan:COST ALL ROWS (Cost: 440554)

Step Name	Step #	Cost	Rows	KBytes
SELECT STATEMENT	18	440,554	244,089	27,412.339
SORT (UNIQUE NOSORT)	17	440,554	244,089	27,412.339
MERGE JOIN	16	436,137	244,089	27,412.339
SORT (JOIN)	13	432,210	265,029	24,587.651
MERGE JOIN	12	428,179	265,029	24,587.651
SORT (JOIN)	8	238,410	4,812,867	329,004.58
MERGE JOIN	7	238,410	4,812,867	329,004.58
SORT (JOIN)	4	48,641	1,581,876	69,516.035
INLIST ITERATOR	3			
SAPCRM.CRMD_ORDER_INDEX TABLE ACCESS (BY INDEX ROWID)	2	36,123	1,581,876	69,516.035
SAPCRM.CRMD_ORDER_INDEXBO INDEX (RANGE SCAN)	1	9,365	1,581,876	
SORT (JOIN)	6	189,769	32,672,568	797,670.117
SAPCRM.CRM_JEST TABLE ACCESS (FULL)	5	26,013	32,672,568	797,670.117
FILTER	11			
SORT (JOIN)	10			
SAPCRM.CRM_JEST TABLE ACCESS (FULL)	9	26,013	32,672,568	797,670.117
SORT (JOIN)	15	3,927	520,290	10,161.914
SAPCRM.CRMD_ORDERADM_H TABLE ACCESS (FULL)	14	1,727	520,290	10,161.914

This plan step designates this statement as a SELECT statement.

Figure: 2 : Original Execution Plan of Query – A



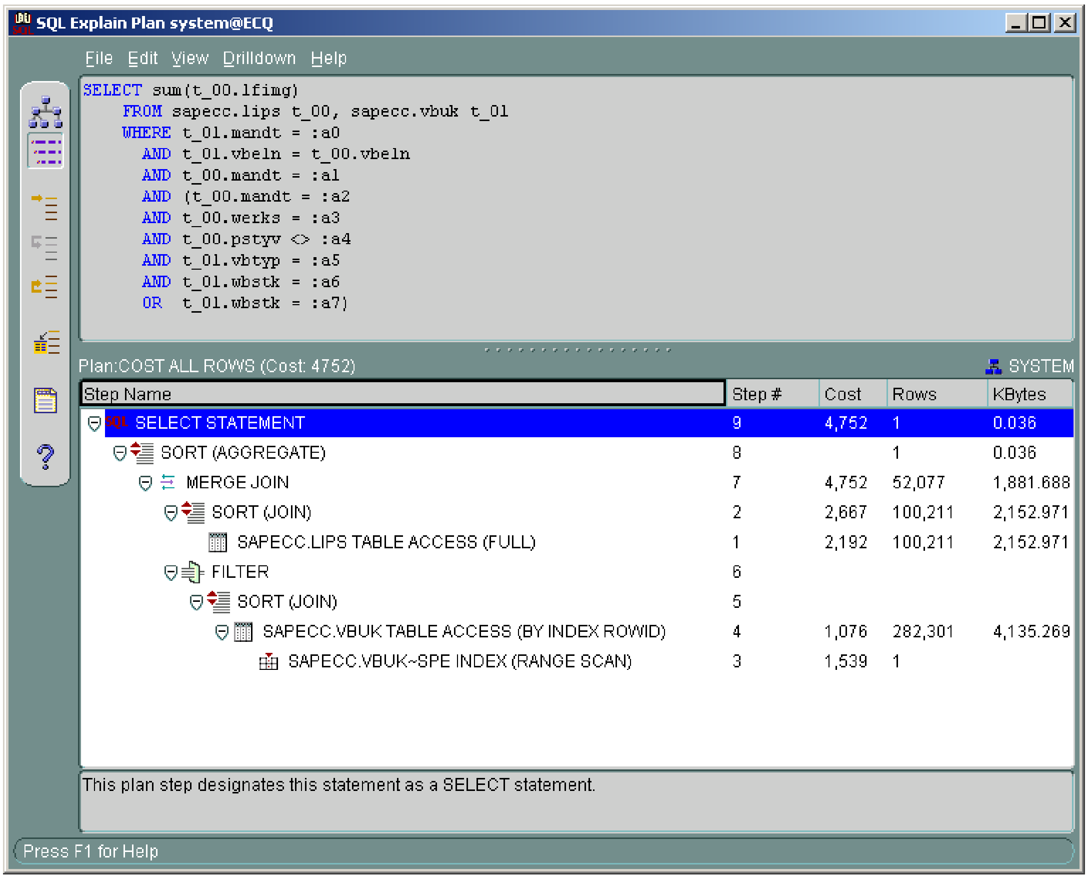
```
SELECT DISTINCT t_00.header
    FROM sapcrm.crmd_order_index t_00, sapcrm.crm_jest t_01,
         sapcrm.crm_jest t_02, sapcrm.crmd_orderadm_h t_03
    WHERE t_01.mandt = :a0
      AND t_01.objnr = t_00.item
      AND t_02.mandt = :a1
      AND t_02.objnr = t_00.header
      AND t_03.client = :a2
      AND t_03.guid = t_00.header
      AND t_00.client = :a3
      AND (t_01.inact = :a4
      AND t_01.stat = :a5
      OR  t_02.inact = :a6
      AND t_02.stat = :a7)
      AND t_00.object_type IN (:a8, :a9, :a10, :a11)
      AND t_03.template_type = :a12
```

Plan:COST ALL ROWS (Cost: 60627)

Step Name	Step #	Cost	Rows	KBytes
SELECT STATEMENT	26	60,627	103,295	11,600.513
SORT (UNIQUE)	25	60,627	103,295	11,600.513
CONCATENATION	24			
NESTED LOOPS	11		53,921	1,316.431
MERGE JOIN	9	5,622	72,000	6,328.125
MERGE JOIN	6	3,578	72,000	4,921.875
SORT (JOIN)	2	306	276,928	31,100.313
SAPCRM.CRM_JEST~Z3 INDEX (RANGE SCAN)	1	2	4	0.098
SORT (JOIN)	5	3,273	371,825	16,339.966
INLIST ITERATOR	4			
SAPCRM.CRMD_ORDER_INDEX~Z5 INDEX (RANGE SCAN)	3	3,124	371,825	16,339.966
SORT (JOIN)	8	1,211	160,823	3,141.074
SAPCRM.CRMD_ORDERADM_H TABLE ACCESS (FULL)	7	526	160,823	3,141.074
SAPCRM.CRM_JEST~Z2 INDEX (RANGE SCAN)	10	305	53,921	1,316.431
NESTED LOOPS	23	20,022	276,928	31,100.313
MERGE JOIN	21	5,622	72,000	6,328.125
SORT (JOIN)	18	4,411	72,000	4,921.875
MERGE JOIN	17	3,578	72,000	4,921.875
SORT (JOIN)	13	306	53,921	1,316.431
SAPCRM.CRM_JEST~Z3 INDEX (RANGE SCAN)	12	305	53,921	1,316.431
SORT (JOIN)	16	3,273	371,825	16,339.966
INLIST ITERATOR	15			
SAPCRM.CRMD_ORDER_INDEX~Z5 INDEX (RANGE SCAN)	14	3,124	371,825	16,339.966
SORT (JOIN)	20	1,211	160,823	3,141.074
SAPCRM.CRMD_ORDERADM_H TABLE ACCESS (FULL)	19	526	160,823	3,141.074
SAPCRM.CRM_JEST~Z2 INDEX (RANGE SCAN)	22	2	4	0.098

This plan step designates this statement as a SELECT statement.

Figure-3: Execution plan of Query – A after creating recommended indexes



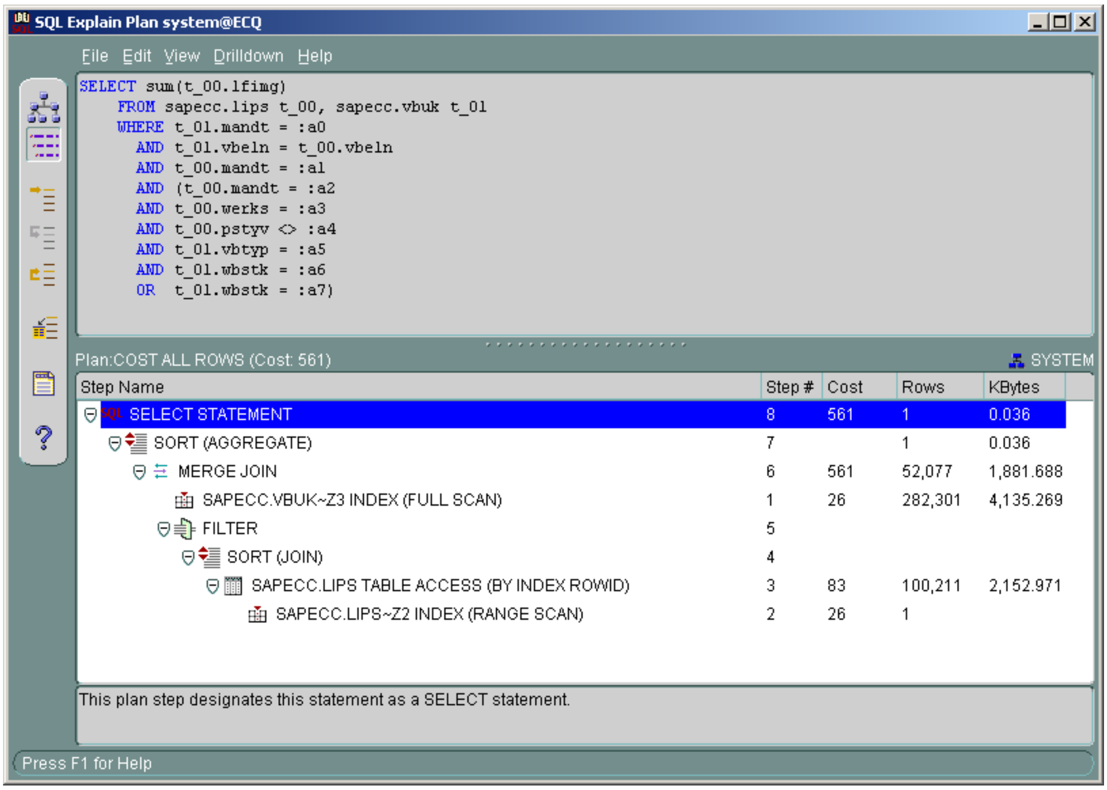
```
SELECT sum(t_00.lfimg)
    FROM sapecc.lips t_00, sapecc.vbuk t_01
    WHERE t_01.mandt = :a0
      AND t_01.vbeln = t_00.vbeln
      AND t_00.mandt = :a1
      AND (t_00.mandt = :a2
      AND t_00.werks = :a3
      AND t_00.pstyv <> :a4
      AND t_01.vbtyp = :a5
      AND t_01.wbstk = :a6
      OR  t_01.wbstk = :a7)
```

Plan:COST ALL ROWS (Cost: 4752)

Step Name	Step #	Cost	Rows	KBytes
SELECT STATEMENT	9	4,752	1	0.036
SORT (AGGREGATE)	8		1	0.036
MERGE JOIN	7	4,752	52,077	1,881.688
SORT (JOIN)	2	2,667	100,211	2,152.971
SAPECC.LIPS TABLE ACCESS (FULL)	1	2,192	100,211	2,152.971
FILTER	6			
SORT (JOIN)	5			
SAPECC.VBUK TABLE ACCESS (BY INDEX ROWID)	4	1,076	282,301	4,135.269
SAPECC.VBUK~SPE INDEX (RANGE SCAN)	3	1,539	1	

This plan step designates this statement as a SELECT statement.

Figure - 4 : Original Execution Plan of Query – B



```
SELECT sum(t_00.lfimg)
    FROM sapecc.lips t_00, sapecc.vbuk t_01
    WHERE t_01.mandt = :a0
      AND t_01.vbeln = t_00.vbeln
      AND t_00.mandt = :a1
      AND (t_00.mandt = :a2
      AND t_00.werks = :a3
      AND t_00.pstyv <> :a4
      AND t_01.vbtyp = :a5
      AND t_01.wbstk = :a6
      OR  t_01.wbstk = :a7)
```

Plan:COST ALL ROWS (Cost: 561)

Step Name	Step #	Cost	Rows	KBytes
SELECT STATEMENT	8	561	1	0.036
SORT (AGGREGATE)	7		1	0.036
MERGE JOIN	6	561	52,077	1,881.688
SAPECC.VBUK~Z3 INDEX (FULL SCAN)	1	26	282,301	4,135.269
FILTER	5			
SORT (JOIN)	4			
SAPECC.LIPS TABLE ACCESS (BY INDEX ROWID)	3	83	100,211	2,152.971
SAPECC.LIPS~Z2 INDEX (RANGE SCAN)	2	26	1	

This plan step designates this statement as a SELECT statement.

Figure - 5 : Execution plan of Query – B after creating recommended indexes

Comparative Cost

Query Label	System	Original Cost	Cost After Indexing
Q1	ECQ	319	1
Q2	ECQ	922	513
Q3	ECQ	938	21
Q4	ECQ	1458	1
Q5	ECP	2009	1
Q6	ECQ	4752	561
Q7	CRQ	6077	3
Q8	CRP	8450	2466
Q9	CRQ	14673	15
Q10	CRP	440554	60627

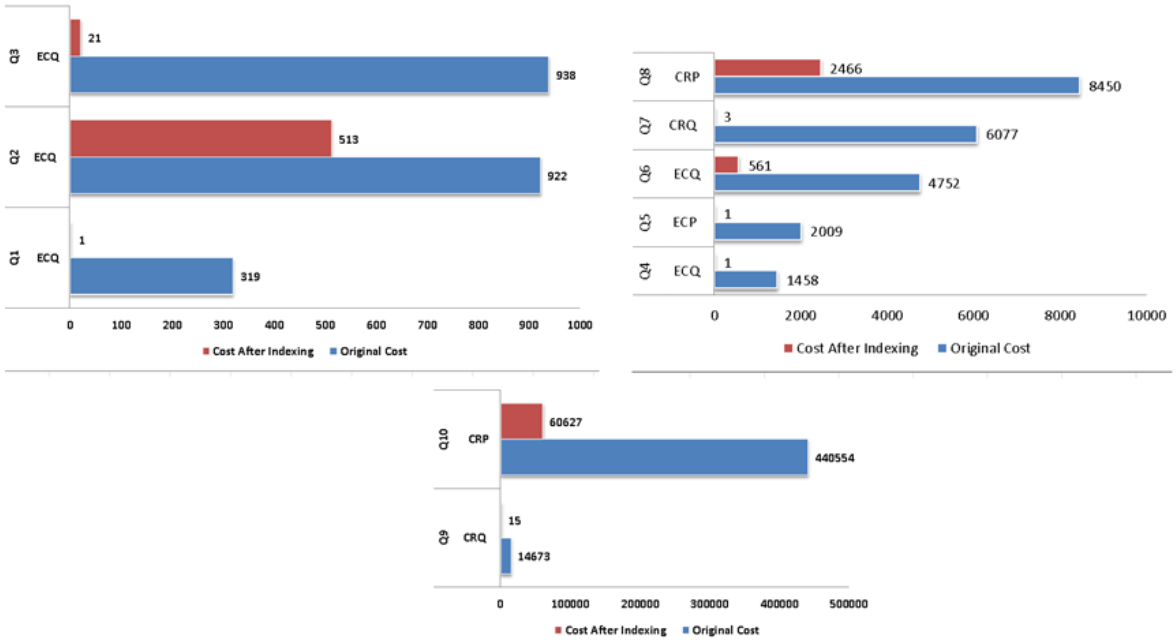


Figure - 6 : Comparative of Original Cost and Cost After Indexing

IV. CONCLUSION

ETL process is a vital component of Data Warehouse responsible for its successful implementation. Extraction process extracts data from various operational source systems having large amount of data by executing several complex queries which need to completed in a specified time window. Database tuning and query tuning plays a major role in performance tuning, apart from adding new hardware. Results of proposed algorithm of recommending creation of indexes to tune performance queries are encouraging, which reduces cost of execution significantly and thus reducing execution time. Future work in this direction is planned to develop an automated tool to recommend set of indexes for ETL process tuning by including required details in the design of metadata repository as a part of total data warehouse architecture.

V. REFERENCES

[1] AlkisSimitsis, PanosVassiliadis, TimosSellis, Optimizing ETL Processes in Data Warehouses, In Proc. ICDE, pages 564–575, 2005.

[2] E. Malinowski, E. Zima´nyi, Hierarchies in a multidimensional model: From conceptual modeling to logical representation, Data & Knowledge Engineering, 2005 Elsevier

[3] Josep Aguilar-Saborit, Victor Munte´s-Mulero, Calisto ZuzarteJosep-L. Larriba-Pey, Star join revisited: Performance internals for cluster architectures, Data & Knowledge Engineering, 2007 Elsevier

[4] Michel Schneider, Integrated vision of federated data warehouses, Data Integration and the Semantic Web, 2006

[5] Songting Chen, Cheetah: A High Performance, Custom Data Warehouse on Top of MapReduce, 36th International Conference on Very Large Data Bases, September 13-17, 2010, Singapore.

[6] Umeshwar Dayal, Malu Castellanos, Alkis Simitsis, Kevin Wilkinson, Data Integration Flows for Business Intelligence, EDBT, 2009

[7] XuanThi Dung •WennyRahayu • David Taniar, A high performance integrated web data warehousing, Cluster Computing, 2007 - Springer

[8] Benoit Dageville, Dinesh Das, Karl Dias, Khaled Yagoub, Mohamed Zait, Mohamed Ziauddin, Automatic SQL Tuning in Oralce 10g, VLDB Conference, Canada 2004

[9] M. GOLFARELLI, S. RIZZI, E. SALTARELLI, Index selection techniques in data warehouse systems, In Proc. DMDW, 2002

[10] Kurt Stockinger, Kesheng Wu, Bitmap Indices for Data Warehouses, In Data Warehouses and OLAP. 2007. IRM Press. London

[11] Stéphane Azefack, Kamel Aouiche, Jérôme Darmont, Dynamic index selection in data warehouses, In 4th International Conference on Innovations in Information Technology, 2007, Dubai

[12] Adela Bâra, Ion Lungu, Manole Velicanu, Vlad Diaconiţa, Iuliana Botha, IMPROVING QUERY PERFORMANCE IN VIRTUAL DATA WAREHOUSES, WSEAS TRANSACTIONS on INFORMATION SCIENCE & APPLICATIONS, 2008

[13] Kai-Uwe Sattler, Eike Schallehn, Ingolf Geist, Autonomous Query-driven Index Tuning, in International Database Engineering & Applications Symposium, Portugal, 2004