*2015 International Conference on Information Processing (ICIP)*
*Vishwakarma Institute of Technology. Dec 16-19, 2015*

# Optimizing SQL Queries in OLAP Database Systems

Vamsi Krishna Myalapalli, Karthik Dussa
Open Text Corporation,
Mind Space IT Park, Hitec City,
Hyderabad, India
{vamsikrishna.vasu, karthikdussa29}@gmail.com

***Abstract:*** **The augmenting data in the contemporary world triggers Data Warehouses (OLAP-Online Analytical Processing Systems) to maintain tremendous amount of data. Apart from being a data repository, data warehouses should also supply analytical and statistical data efficiently on demand. Data warehouses support unique features such as data mining, ad-hoc querying on data collected and integrated from many of the computerized systems used in organization. The issue of data extraction/updating is the most important factor for organizations to deploy real-time data warehouse solutions. This paper proposes several optimizing techniques for ensuring high performance data warehousing. The execution of this paper could serve as a tuning/bench-marking/management tool for overhauling data warehouse querying practices and processing. Experimental fall-outs of our implementation advocate that query performance and operational costs were significantly optimized.**

**Index Terms—*Data Warehouse Tuning; OLAP Tuning; OLAP Query Tuning, SQL Tuning, SQL Optimization; Query Tuning; Query Optimization.***

## I. INTRODUCTION

About 80 percent of the problems pertaining to database performance can be diminished by pre tuning the SQL query even before its actual deployment. Typically, data warehouses (OLAP Databases) encompass a massive amount of data, but it is defined by more than just size. With tremendous amount of data to summarize and aggregate, lot of data warehouse(s) are I/O bound and the database administrator must consider a server, which would optimize disk I/O throughput.

Commercial Data Ware house (DW) systems are growing from tiny-scale application(s) to corporate wide system(s) and we must be postured to help the system grow without service interruption.

A DW is typically summarized active data, which spans across the complete enterprise. Data is inserted via cleaning-up and aggregate procedure over a pre-determined duration e.g. day-to-day, once a month or three-monthly. The crucial notion in DW is the conception that the information is loaded along a time-line. DW should bolster the prerequisites of a huge range of end users and might encompass atomic data apart from summarized data. A DW might mix the theories of OLAP, OLTP (Online Transaction Processing), and Decision Support Systems (DSS) into single physical data-structure.

Typically, DWs are challenged by disk I/O, ensued due to the higher proportion of FTSs and IFSs (Full Table Scan and Index Full Scan). Big data buffer cache memories in utmost OLTP (Online Transaction Processing) systems make them CPU bound.

Further paper is organized in the following manner. Section 2 explains the related work and background. Section 3 deals with the proposed model i.e. methodologies. Section 4 demonstrates pragmatic approach. Section 5 explains comparative analysis and finally section 6 concludes the paper.

## II. BACKGROUND AND RELATED WORK

A normalized database encompasses great quantity of data in small volume; where as in DW we spend storage for faster access via de-normalization. DWs are typically De-Normalized structure and experience 90 % reporting queries and only 10% ad-hoc querying.

Query Optimization can take form of either Rule Based or Cost Based referred as Rule Based Optimization (RBO) or Cost Based Optimization (CBO) respectively. High Performance SQL [1] explained RBO and CBO level tuning models. It made part of the queries faster via CBO approach and rest others via RBO approach.

An Appraisal to Optimize SQL Queries [2] explained basic model that rewrites sundry queries to minimize CPU Cost and raise the index utilization. It also reduced rate of parsing (hard parse) and increased query reuse (soft parse).

Augmenting Database Performance via SQL Tuning [3] explained Query tuning through Index Tuning, Access types and Hints and Database tuning through session handling.

High Performance PL/SQL [4] explained the tuning model to reduce the rate of Context Switching (an overhead) among SQL and PL/SQL engines.

The cache hit-ratio is not appropriate for those data warehouse systems that usually perform FTSs, or which implement ALL_ROWS [5] SQL tuning, Data warehouses need user specific memory instead. Typically, Data warehouses are so data demanding, so always there will be a problem towards completely consuming the CPU power.

## III. PROPOSED BENCHMARK

All DWs typically perform aggregation via SQL. We have tried rewriting our raw queries, which run on production database and witnessed optimized versions for the earlier ones. The following are several scenarios that we have come across. All the methodologies that were implemented in this paper justified query optimization in statistics and resource utilization.

*1) Deter FTS:* FTS retrieves each and every bit of data from the table i.e. triggers huge amount of disk I/O. This type of access will always become a night mare for the database as

it entails retrieving billions of records. FTS would arise in the following cases

a) WHERE clause absence.
b) Index and Statistics of table are not updated.
c) Absence of filtering of rows at WHERE clause.

*2) Enforcing Multi-leveled Partition of Tables:* This ensures that data is stored in a precise scheme. On governing where data is stored on disk, DBMS engine could minimize the rate of disk I/O mandatory to service every query. This also increases the rate of parallelism in query execution.

Partitioning a table will permit Parallel

a) FTSs which run upto 15 times faster.
b) DML execution(s).
c) Table Re-organization (i.e. de-fragmentation etc).
d) ETL (Extract, Transform, Load) operations.

Big amount of shared memory does not benefit data DSS and DW in which lot of data access is accomplished by a parallelized FTS. When DBMS carries out a parallel FTS, the database blocks are directly read into user specific memory, bypassing RAM data buffer. Partition Pruning offers consistent query performance.

*3) Enforcing Multi-leveled Partition of Indexes:* Index partitioning will offer the following performance benefits

a) Loading data via DML operation(s).
b) Loading data via DDL operation(s).
c) Querying data via SELECT queries.

Following are the maintenance benefits

a) Rebuilding indexes.
b) Setting indexes unusable/invisible at partition-level.

| Feature | Performance | Manageability | Scalability | Availability |
|---|---|---|---|---|
| Read Only Partitions | ✓ | ✓ | ✓ | |
| Pruning (Partition Elimination) | ✓ | | ✓ | ✓ |
| Partition Wise Joins | ✓ | | ✓ | |
| Parallel DML | ✓ | | | |
| Archiving | ✓ | ✓ | ✓ | |
| Exchange Partition | ✓ | ✓ | ✓ | ✓ |
| Partition Truncation | ✓ | ✓ | ✓ | |
| Local Indexes | ✓ | ✓ | ✓ | ✓ |

**Figure 3.1: Partition Feature Matrix**

The matrix (Figure 3.1) exhibits the Partitioning Feature Matrix and corresponding merits and demerits.

*4) Read-Only Tablespace(s):* If the DBA has time sequence data warehouse where data would ultimately turn static, enforcing tablespace partitioning and converting the old tablespace(s) as read only would greatly enhance performance. If a tablespace is made as read only, DBMS engine will bypass the read consistency protocol, minimizing overhead and implying faster through-put.

Read only Tablespaces will also offer the benefit of faster check pointing, back-up and recovery.

*5) Precocious Data Buffer Administration:* By enforcing various block sizes, we could pre allocate warehouse object(s) to individual data buffer(s) and confirm that the working set of recurrently referred data always gets cached. Also we have to re-allocate memory frames among db_cache_size (cache memory) and the pga_aggregate_target (user specific memory) region towards enhancing through-put of DW. Hence, we have to keep track of cache memory and user specific memory.

*6) Materialized View (MV):* MVs views enforce cloning and permits us to pre summarize and pre join table(s). On top of everything MVs permit query rewrite capability, thus any query can take advantage of the pre-summarization and they will be auto rewritten to refer the aggregate view. This obviates the necessity to perform futile and expensive FTS (minimizing disk I/O) apart from minimizing developers effort to rewrite a query to optimize it.

In fact, the DBAs must

a) Track time series report(s) of query access path(s) by observing work load and then wisely construct the best effective MVs.
b) Persistently investigate if any MVs could be created further, so as to reduce disk I/O.

*7) Sustaining MVs:* Since MVs are snapshots, it would be very often to refresh them. Also when refreshing I/O bottleneck(s) can be seen, so refreshing should be done in an efficient way i.e. not refresh during peak times.

*8) Tracking for FTS and Hash Join:* Tracking for FTSs and Hash Joins plays a crucial role. So as they could be substituted with Star Joins which follows star schema data model. The objective of star transformation is to process the minimized set of information from the fact table. The star conversion evades the FTS of the fact table.

*9) Enforcing Multiple Data Buffers:* This will cache and segregate indexes and dimension tables, while allocating adequate RAM for the FTSs and IFSs. Even during mode change processing ETL and Roll-ups, DBMS engine will auto recognize variation in data access and re-allot the RAM region(s) for accommoding the present execution.

*10) Deem Star Query Optimization:* Star query tuning ensures that complex DSS queries run faster. A bitmap index must be created upon every foreign key column of the fact table(s). Also we must enable start transformation rewrite.

*11) Caching the Data:* Smaller and recurrently referred dimension tables must be cached in order to minimize I/O and turn around time.

*12) Multiple Block-Sizes:* All indexes in database which are processed through range scans and those objects which are to be accessed through FTS(s) or IFS(s) must be stored in a blocksize of 32KB. Enforcing multiple block sizes can greatly enhance I/O but it requires the knowledge of I/O landscape.

Smaller block size increases random access and reduces block contention whereas larger block size has less overhead and favors FTSs and sequential access.

*13) FTS Friendly Block Size:* Object(s) that perform FTSs must be positioned in a large block size, with the db_file_multiblock_read_count parameter fixed to the block size of that tablespace.

*14) Data Mart (DM) Tuning:* DMs are typically equivalent to OLAP databases. DMs are particular use databases. DM is typically produced from a DW for a particular department or division for critical report needs. Information in DM is typically summarized on particular duaration like day-to-day, once-a-month or three-monthly.

DM tuning is typically for reporting. We enhance a DM for huge aggregations and sorts. In addition, we might consider partitioning a DM to enhance physical accessing to huge data set(s).

*15) Prevent Un-necessary Sorting:* An abundant deal of time could be lost by needless filtering and sorting. E.g. using "union" instead-of "union all". Since "union all" will obviate the need to sort and remove duplicated rows [2].

*16) Data Deleting and Purging:* Organized data purging from DW is one of the most resouce demanding and problematic operations. This has to be done intermittently in a minute time. Typically this time is also fine for de-fragmenting and reloading data for optimal query execution.

*17) Minimizing Row Chaining and Fragmentation:* Row chaining and fragmentation would lead to severe operational and performance effect on DW. The degree of affect that DW would experience is mostly determined by the landscape of individual work-load.

*18) Dimensionalizing the Tables:* Dimensionalize the table as much as possible. A dimension describes hierarchy of (parent/child) relationship(s) among column(s), where all the columns may not have to come from same table.

Dimensionalizing increases the likelihood for query re-write as they assist in launching functional dependencies among columns. Apart from this, dimension(s) could explain intratable relationship(s) which cannot be explained by constraints. Defining dimension does not acquire extra storage. Instead, defining dimension(s) introduces meta-data which describe inter and intra dimensional relationship(s) within schema. Prior to creating a MV, the primary step is to review schema and define dimensions because this could considerably enhance the likelihood of re-writing query.

*19) Compressing Data Segments:* This results in minimizing time to perform bulk inserts, deletes and eliminates intra block repeated column values apart from minimizing the space required for a segment. More the data is repeated higher the compression. Table(s) with higher repeats comprises those with lot of recurred Foreign Keys, e.g. Summary and Fact Tables in a DW system.

*20) Optimizing Third Normal Form Queries:* Some DWs implement 3NFs. Compared to Star schema, 3NF schema usually have higher number of tables because of normalization procedure. 3NF schema is usually selected for huge DWs, specifically environments with considerable data load necessity, which are used to feed DMs and perform long running queries. 3NF schemas offers neutral schema design, independent of data-usage or application considerations and require fewer data-transformations than more normalized schema like star schema.

The biggest table(s) in 3NF must be partitioned to permit partition wise join(s). The best partition method is composite range hashed partition, with most common join key selected as the hash-partition key. Parallelism is frequent greatly used in 3NF situations and parallelism must usually be facilitated.

*21) Leveraging Stream(s):* Stream dependent feeding mechanism(s) could take a snapshot of the necessary data changes on the active database and transfer the same to the target data ware-house. Using the redo information obtained from stream capture process evades needless over-head on the underlying real-time database.

*22) Capturing Asynchronous Data Changes:* Capturing the data that is changed determines incremental extraction, this permits modified data extraction only. E.g. Extracting data on a weekly plan, then DW processes only the data which was modified form the previous extraction.

*23) When Not to Enforce Parallelism:* This technique is aimed at reducing cost to organization. Although parallelism profits DSS and DW environments. The following are some scenarios where parallelism is unpropitious.

a) Environment where the usual query/transaction is very tiny (< few seconds). e.g. OLTP systems. (But parallel DML can fasten batch jobs on OLTPs). Parallelism is not beneficial as cost incurred in co-ordinating parallel execution servers. The cost for the co-ordination might out-weigh the advantages of parallel execution.
b) Environment(s) where processor, I/O or memory resources were currently deeply operated. Parallelism is intended to exploit extra obtainable hardware

resources. If no such resource(s) were obtainable, then parallelism will not lead to benefits and certainly might hurt performance.

*24) Leveraging Parallel DML:* Parallel DML is mostly implemented for speeding large DML operations on huge database objects. It is beneficial in DSS where scalability and performance of accessing big objects is vital. The following are scenarios where parallel DML must be enforced.

a) DW system table refreshing.
b) Building intermediate summary tables.
c) Using Scoring tables.
d) Historic table update.
e) Batch jobs running.

*25) Operations to be Parallelized:* Parallelism requires precise statistics to do optimally. Parallel execution must be leveraged over following scenarios.

a) Accessing methodolgy: E.g. Table scans, IFSs and partitioned Index Range Scans (IRS).
b) Join Methodology: E.g. Star Tranformation, Nested loop, hash and Sort merge.
c) DDL Queries: E.g. CREATING INDEX, CREATE TABLE AS SELECT, rebuilding index, rebuilding index partition(s) and split/move/coalesce partitions.
d) DML Queries: E.g. Insert As Select, Merge, Update and Deletes.
e) Parallel Query: We must parallelize queries and sub-queries in SELECTs and query parts of DDL and DML queries.
f) Miscellaneous: E.g. SELECT DISTINCT, NOT IN, UNION, GROUP BY, UNION ALL, ROLL UP and CUBE and table functions and aggregates.

## IV. EXPERIMENTAL SETUP

In this section, the methodologies specified in earlier section are paraphrased via pragmatic queries, which could serve as exemplars.
The enhanced queries are tuned as per CBO statistics.

1. SQL> select name from employee where id = 532;

*Instead of*

SQL> select * from employee; // Results in FTS
c) Updating statistics of table and concerned Indexes.
Oracle
SQL> analyze table <table_name> compute statistics;
Updating statistics of specific Index
SQL> analyze index <index_name> compute statistics;
SQL Sever
SQL> update statistics <table_name>;

2. Partitioning table during creation. The following query creates ‘List partitioning’ with 2 partitions.

```
SQL> create table pt (deptno number(10), state varchar2(2))
     partition by list (state) (partition nw values ('or', 'wa'),
     partition sw values ('az', 'ca', 'nm'));
```

3. SQL> create unique index ix_pt on pt (deptno, state) global partition by range (deptno)
(partition dept_100 values less than (100),
partition dept_max values greater than (100));
Constructing non-partitioned index over partitioned table is identical to constructing non-partitioned indexes over non- partitioned table.

4. The following query would make a tablespace read only
SQL> alter tablespace <tablespace> read only;

5. SQL> sho paramter db_cache_size
SQL> sho parameter pga_aggregate_target
SQL> alter system set “db_cache_size” = 449M;
// Allocates 499MB to buffer.
SQL> alter system set “pga_aggregate_target” = 116M;
// Allocates 116MB to User specific memory.

6. SQL> create materialized view <mv_name>
build immediate
refresh complete enable query rewrite as
<actual_query_goes_here>;
SQL> alter session set query_rewrite_enabled = true;
The above parameter must be true, if DBMS engine has to rewrite a query.

7. Refreshing a materialized view.
SQL> exec dbms_mview.refresh(‘<mat_view_name>’,’c’);
Parameter ‘c’ indicates complete refresh.

8. Executing 3 sub queries to obtain list of foreign key(s) for matching with fact table, retrieving bitmaps for every foreign key, mix them, and then retrieving result records.

```
SQL> select stock.sale_area, time.fiscal_duration,
     sum(sale.dollar_sale) revenue, sum(dollar_sale)-
     sum(dollar_cost) earning from sale where
     stock_key in ( select stock_key from stock where
     sale_area in ('west', 'southwest')) and time_key in
     ( select time_key from time where quarter in ('2008',
     '2009', '2011')) and product_key in (select
     product_key from product where department
     = 'test');
```

*Instead of*

```
SQL> select stock.sale_area, time.fiscal_duration,
     sum(sale.dollar_sale) revenue, sum(dollar_sale) -
     sum(dollar_cost) earning from sale, stock, time,
     product where sale.stock_key = stock.stock_key
     and sale.time_key = time.time_key and
     sale.product_key = product.product_key and
     time.fiscal_duration in ('2008','2009','2011') and
     product.department = 'test' and stock.sale_area
     in ('texas','sanfran') group by stock.sale_area,
     time.fiscal_duration;
```

9. SQL> alter system set “DB_BLOCK_BUFFERS” = <num_of_buffers>;

10. This will enable Start Transformation Rewrite
SQL> alter session set
“STAR_TRANSFORMATION_ENABLED” = TRUE;

11. SQL> alter table <table_name> cache;

12. SQL> sho parameter “DB_BLOCK_SIZE”;
SQL>alter system set “DB_BLOCK_SIZE” = 32K;

13. SQL> sho parameter Db_File_Multiblock_Read_Count;
We should apply the following value for ensuring performance
db_file_multiblock_read_count =
Maximum I/O chunk size / Db_block_size

15. SQL> select id, name from student union all select id, name from emps;
*Instead of*
SQL> select id, name from student union select id, name from emps;

16. Typically, if a table is dropped it enters recycle bin.
Purging recycle bin permanently removes all the data from recycle bin.
SQL> purge recyclebin;
If individual table has to be deleted permanently then following query has to be used
SQL> drop table <tabll_name> purge;

17. SQL> alter tablespace <tablespace_name> coalesce;
This would recover space from honey-comb fragmentation.
We can also de-allocate un-used space from objects
SQL> alter table <table-name> deallocate unused space;
SQL> alter index <table-name> deallocate unused space;

19. Segments are analogous to tables in database.
SQL> create table <tabl_name> (col_list) compress;
Compressing an existing table (segment)
SQL> alter table <tabl_name> compress;

20. SQL> create table emp_hash (id number(5), name varchar2(30), sal number(8)) partition by hash(id) partitions 4 store in (tbs1, tbs2, tbs3, tbs4);
Here tbs1, tbs2,tbs3, tbs4 are tablespaces.
This ensures that we build partitions in a round-robin fashion over tablespaces.

23. SQL> alter session disable parallel dml;

25. SQL> alter session enable parallel dml;

## V. OUTPUT SCREENS AND COMPARISION

Some of the results are represented beneath in the form of Output screens to describe comparative results.

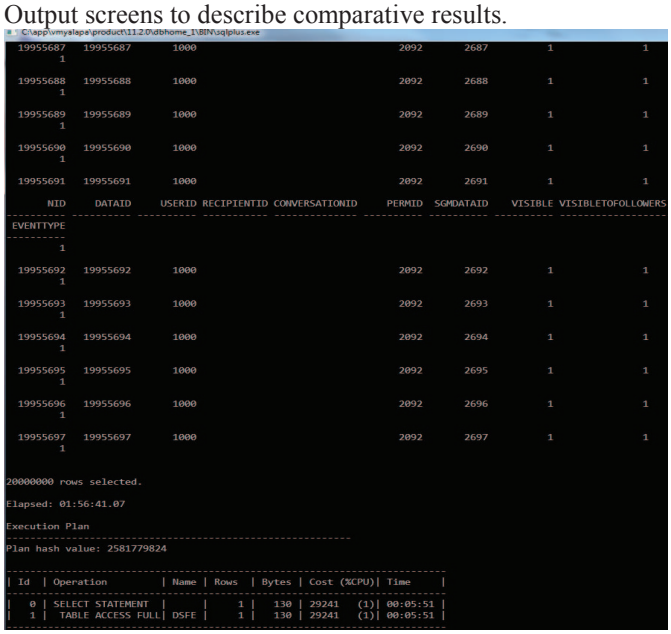



**Fig 5.1: Demonstrating FTS on 20 Million Rows (#2)**

The figure 5.1 demonstrates FTS on table holding 20 Million rows, which took (approx.) 2 hours for query execution.

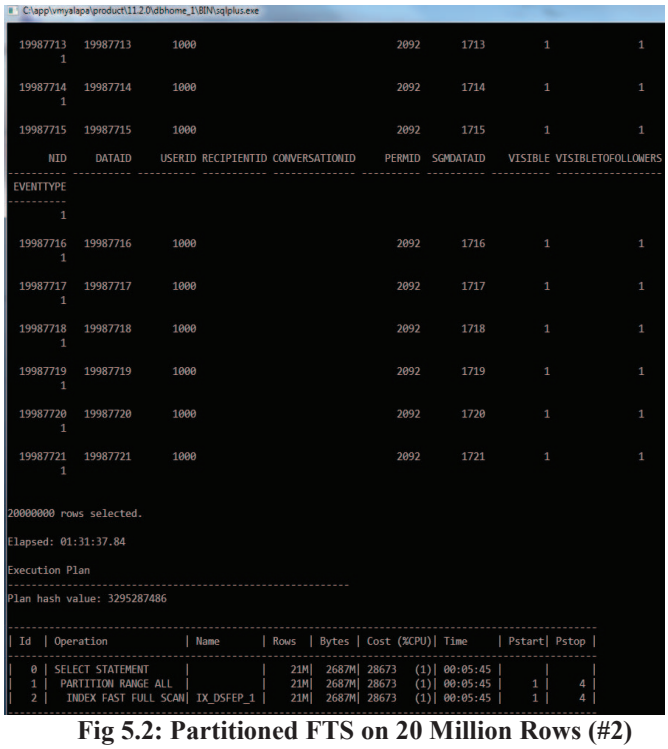
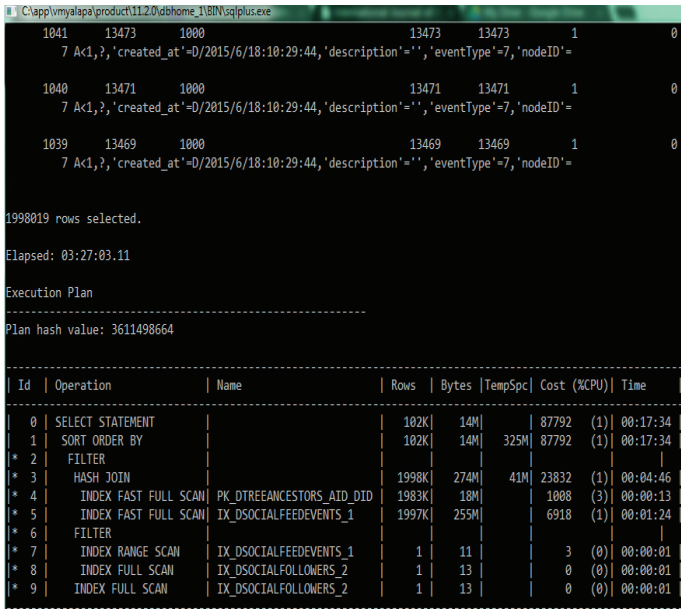



**Fig 5.2: Partitioned FTS on 20 Million Rows (#2)**

The figure 5.2 demonstrates FTS on partitioned table containing 20 Million rows, which took approximately 90 minutes for query execution. Thus partitioning a table, lead to reduction in access time by 25%.



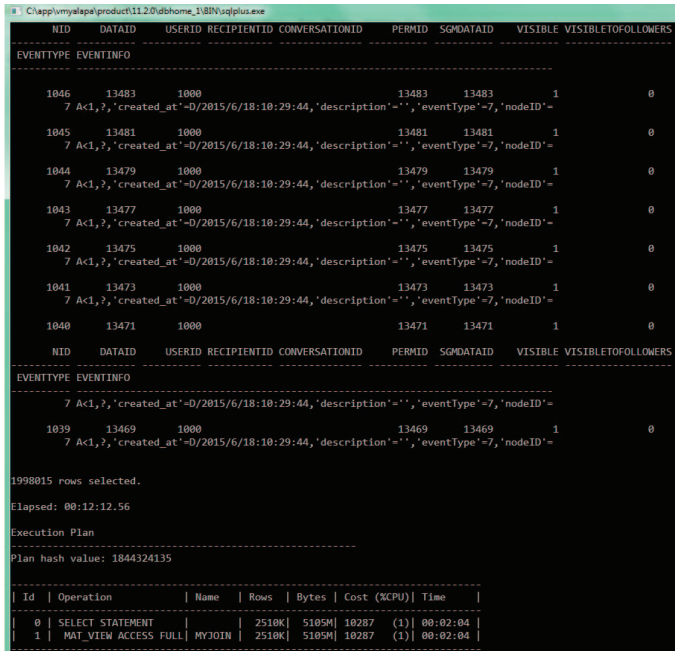
```
C:\app\vmyalapa\product\11.2.0\dbhome_1\BIN\sqlplus.exe

      1041      13473       1000                            13473      13473          1             0
        7 A<1,?,'created_at'=D/2015/6/18:10:29:44,'description'='','eventType'=7,'nodeID'=

      1040      13471       1000                            13471      13471          1             0
        7 A<1,?,'created_at'=D/2015/6/18:10:29:44,'description'='','eventType'=7,'nodeID'=

      1039      13469       1000                            13469      13469          1             0
        7 A<1,?,'created_at'=D/2015/6/18:10:29:44,'description'='','eventType'=7,'nodeID'=


1998019 rows selected.

Elapsed: 03:27:03.11

Execution Plan
----------------------------------------------------------
Plan hash value: 3611498664

-------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name                     | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                          |  102K|    14M|       | 87792   (1)| 00:17:34 |
|   1 |  SORT ORDER BY         |                          |  102K|    14M|   325M| 87792   (1)| 00:17:34 |
|*  2 |   FILTER               |                          |      |       |       |            |          |
|*  3 |    HASH JOIN           |                          | 1998K|   274M|    41M| 23832   (1)| 00:04:46 |
|*  4 |     INDEX FAST FULL SCAN| PK_DTREEANCESTORS_AID_DID | 1983K|    18M|       |  1008   (3)| 00:00:13 |
|*  5 |     INDEX FAST FULL SCAN| IX_DSOCIALFEEDEVENTS_1   | 1997K|   255M|       |  6918   (1)| 00:01:24 |
|*  6 |    FILTER              |                          |      |       |       |            |          |
|*  7 |     INDEX RANGE SCAN   | IX_DSOCIALFEEDEVENTS_1   |    1 |    11 |       |     3   (0)| 00:00:01 |
|*  8 |     INDEX FULL SCAN    | IX_DSOCIALFOLLOWERS_2    |    1 |    13 |       |     0   (0)| 00:00:01 |
|*  9 |    INDEX FULL SCAN     | IX_DSOCIALFOLLOWERS_2    |    1 |    13 |       |     0   (0)| 00:00:01 |
```

**Fig 5.3: Representing IFS on 20 Million Rows (#6)**

The figure 5.3 performs IFS on 2 Crore records and retrieves 20 lakh rows (approx.) that satisfy the condition.

Statistics: Elapsed Time – 207 Min 3 Sec, Rows Processed – 61,82,003, Bytes Accessed – 57,50,00,037, CPU Cost – 2,07,342, CPU Time – 39 sec, 154 ms.

Then we have created a MV (capable of incremental/fast refresh) and executed the same query.



```
C:\app\vmyalapa\product\11.2.0\dbhome_1\BIN\sqlplus.exe

       NID     DATAID     USERID RECIPIENTID CONVERSATIONID     PERMID  SGMDATAID    VISIBLE VISIBLETOFOLLOWERS
---------- ---------- ---------- ----------- ---------- ---------- ---------- ---------- ------------------
 EVENTTYPE EVENTINFO
---------- --------------------------------------------------------------------------------

      1046      13483       1000                            13483      13483          1             0
        7 A<1,?,'created_at'=D/2015/6/18:10:29:44,'description'='','eventType'=7,'nodeID'=

      1045      13481       1000                            13481      13481          1             0
        7 A<1,?,'created_at'=D/2015/6/18:10:29:44,'description'='','eventType'=7,'nodeID'=

      1044      13479       1000                            13479      13479          1             0
        7 A<1,?,'created_at'=D/2015/6/18:10:29:44,'description'='','eventType'=7,'nodeID'=

      1043      13477       1000                            13477      13477          1             0
        7 A<1,?,'created_at'=D/2015/6/18:10:29:44,'description'='','eventType'=7,'nodeID'=

      1042      13475       1000                            13475      13475          1             0
        7 A<1,?,'created_at'=D/2015/6/18:10:29:44,'description'='','eventType'=7,'nodeID'=

      1041      13473       1000                            13473      13473          1             0
        7 A<1,?,'created_at'=D/2015/6/18:10:29:44,'description'='','eventType'=7,'nodeID'=

      1040      13471       1000                            13471      13471          1             0

       NID     DATAID     USERID RECIPIENTID CONVERSATIONID     PERMID  SGMDATAID    VISIBLE VISIBLETOFOLLOWERS
---------- ---------- ---------- ----------- ---------- ---------- ---------- ---------- ------------------
 EVENTTYPE EVENTINFO
---------- --------------------------------------------------------------------------------
        7 A<1,?,'created_at'=D/2015/6/18:10:29:44,'description'='','eventType'=7,'nodeID'=

      1039      13469       1000                            13469      13469          1             0
        7 A<1,?,'created_at'=D/2015/6/18:10:29:44,'description'='','eventType'=7,'nodeID'=


1998015 rows selected.

Elapsed: 00:12:12.56

Execution Plan
----------------------------------------------------------
Plan hash value: 1844324135

-------------------------------------------------------------------------------------
| Id  | Operation            | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |         |  2510K|  5105M| 10287   (1)| 00:02:04 |
|   1 |  MAT_VIEW ACCESS FULL| MYJOIN  |  2510K|  5105M| 10287   (1)| 00:02:04 |
```

**Fig 5.4: IFS on 20 Million Rows Using MV (#6)**

The following are statistics for IFS on 20 Million Rows using MV.

Elapsed Time – 12 Min 12 Sec, Rows Processed – 50,20,000, Bytes Accessed – 10210000000, CPU Cost – 20,574, CPU Time – 4 Sec, 8 ms.

Elapsed time and CPU time were reduced by more than 94% and 99% respectively apart from reduction in other factors.

## VI. CONCLUSION

DW query tuning necessitates each bit of our ingenuity to evade harmful way. FTSs and scattered reads constitutes the major part of the whole database time. Implementing perpetual SQL tuning, irrespective of modifications to environment ensures performance tuning i.e. minimizing time & space complexities.

DWs tending to by-pass the data buffer(s) to make parallel FTSs, and enhancing throughput of disk I/O is the utmost critical bottleneck. The key process in a DW is typically recording a small to intermediate leveled analytical processing.

Evaluating real-time physical I/O delays is a significant move in enhancing performance. Tuning DW is an iterative process (it's never 'done') and it may also necessitate tuning OLTP queries as well.

In a nutshell, DW optimization is all about minimizing I/O of disk. DWs are disk intensive and necessitate an architecture that looks after that the processors are running at saturated level.

This paper has introduced some novel techniques in achieving high performance DW administration. Comprehensively with the methodologies that we have implemented and proposed the factors i.e. I/O, CPU Cost and amount of data processed were drastically minimized as evinced in the earlier section.

## REFERENCES


[1] Vamsi Krishna Myalapalli and Pradeep Raj Savarapu, "High Performance SQL", 11th IEEE India International Conference on Emerging Trends in Innovation and Technology, Dec 2014, Pune, India.

[2] Vamsi Krishna Myalapalli and Muddu Butchi Shiva, "An Appraisal to Optimize SQL Queries", IEEE International Conference on Pervasive Computing, January 2015, Pune, India.

[3] Vamsi Krishna Myalapalli, Thirumala Padmakumar Totakura and Sunitha Geloth, "Augmenting Database Performance via SQL Tuning", IEEE International Conference on Energy Systems and Application, October 2015, Pune, India.

[4] Vamsi Krishna Myalapalli and Bhupati Lohit Ravi Teja, "High Performance PL/SQL Programming", IEEE International Conference on Pervasive Computing, January 2015, Pune, India.

[5] Vamsi Krishna Myalapalli, "Wait Event Tuning in Database Engine", Advances in Computing and Management, Springer Publications, November 2015.

[6] Vamsi Krishna Myalapalli, Keesari Prathap Reddy and ASN Chakravarthy, "Accelerating SQL Queries by Unravelling Performance Bottlenecks in DBMS Engine", IEEE International Conference on Energy Systems and Application, October 2015, Pune, India.

[7] Vamsi Krishna Myalapalli and Karthik Dussa "Overhauling PL/SQL Applications for Optimized Performance", IJIRSET International Journal, Volume 4, Issue 9, September 2015.

[8] Paul Lane, "Oracle Database Data Warehousing Guide 11g Release 1", B28313-02, September 2007.

[9] Vamsi Krishna Myalapalli, "Revamping SQL Queries for Cost Based Optimization", (Unpublished).

[10] Vamsi Krishna Myalapalli, Karthik Dussa and Thirumala Padmakumar Totakura, "An Appraisal to Overhaul Big Data Processing in Cloud Computing Environments", Volume 4, Issue 10, IJIRSET Journal.