Tuning the SQL Query in order to Reduce Time Consumption

Mr.P.Karthik[1], Prof.G.Thippa Reddy[2], Mr.E.Kaari Vanan[3]

**[1]M.Tech Software technology
School of Information Technology and Engineering
VIT University
Vellore, Tamil Nadu, India-632014**

**[2]Assistant Professor
School of Information Technology and Engineering
VIT University
Vellore, Tamil Nadu, India-632014**

**[3]M.Tech Software technology
School of Information Technology and Engineering
VIT University
Vellore, Tamil Nadu, India-632014**

Abstract

The optimizer examines parsed and normalized queries, and information about database objects. The input to the optimizer is a parsed SQL query and statistics about the tables, indexes, and columns named in the query. The output from the optimizer is a query plan. The query plan is compiled code that contains the ordered steps to carry out the query, including the access methods (table scan or index scan, type of join to use, join order, and so on) to access each table. Using statistics on tables and indexes, the optimizer predicts the cost of using alternative access methods to resolve a particular query. It finds the best query plan – the plan that is least the costly in terms of I/O [1].For many queries, there are many possible query plans. Adaptive Server selects the least costly plan, and compiles and executes it. The query process very faster and less cost per query. This query optimization gives the High performance of the system and less stress on the database when data transmission occurs and the efficient usage of database engine and lesser memory consumed.

Keywords: *Optimization, Indexing, Normalization, Query Processing, Control methods and search.*

1. Introduction

Query tuning is a method of tuning the query and re-writing a query such that it runs faster. It includes adding an index changing the schema or modifying transaction links etc. There are two ways to see that query is running slowly.

You look at its query plan and see that relevant indexes are not used. (The query plan is the method chosen by the optimizer to execute the query).

1.1 Importance of Tuning

- Reduce response time for SQL processing
- To find a more efficient way to process workload
- Improve search time by using indexes
- Join data efficiently between 2 or more tables

1.2 Role of Hardware & Design [2]

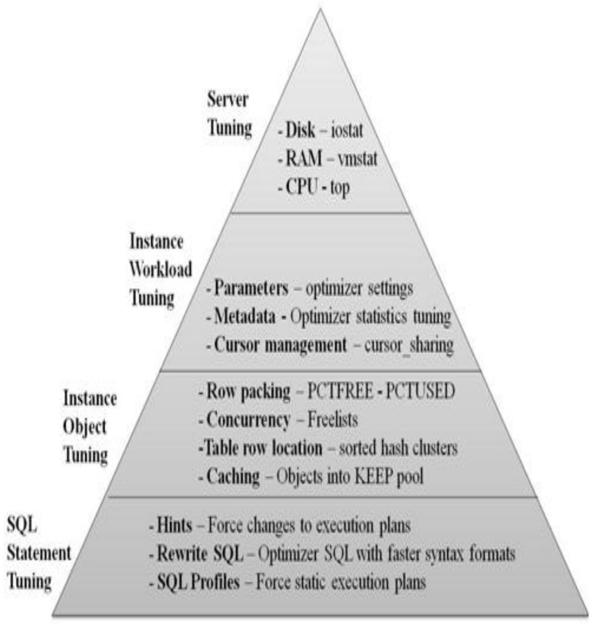
- Memory, Disk & CPU speed can improve performance
- Increased hardware does not always result into better performance
- Poor application design accounts for over 70% of performance issues
- Do Performance design review early in development

2. Tuning Individual Oracle Statements

The acronym SQL stands for Structured Query Language. SQL is an industry standard database query language that was adopted in the mid-1980s.[4] It should not be confused with commercial products such as Microsoft SQL Server or open source products such as MySQL, both of which use the acronym as part of the title of their products.

2.1 Do this before you start individual SQL statement Tuning

This broad-brush approach can save thousands of hours of tedious SQL tuning because you can hundreds of queries at once. Remember, you MUST do this first, else later change to the optimizer parameters or statistics may un-tune your SQL.


Server Tuning
- Disk – iostat
- RAM – vmstat
- CPU - top
Instance Workload Tuning
- Parameters – optimizer settings
- Metadata - Optimizer statistics tuning
- Cursor management – cursor_sharing
Instance Object Tuning
- Row packing – PCTFREE - PCTUSED
- Concurrency – Freelists
-Table row location – sorted hash clusters
- Caching – Objects into KEEP pool
SQL Statement Tuning
- Hints – Force changes to execution plans
- Rewrite SQL – Optimizer SQL with faster syntax formats
- SQL Profiles – Force static execution plans

Remember, you must ALWAYS start with system-level SQL Tuning, else later changes might undo your tuned execution plans:

- **Optimize the server kernel** - You must always tune your disk and network I/O subsystem (RAID, DASD bandwidth, network) to optimize the I/O time, network packet size and dispatching frequency.
- **Adjusting your optimizer statistics** - You must always collect and store optimizer statistics to allow the optimizer to learn more about the distribution of your data to take more intelligent execution plans. Also, histograms can hypercharge SQL in cases of determining optimal table join order, and when making access decisions on skewed WHERE clause predicates.
- **Adjust optimizer parameters** - Optimizer optimizer_mode, *optimizer_index_caching*, o*ptimizer_index_cost_adj*.
- **Optimize your instance** - Your choice of *db_block_size*, *db_cache_size*, and OS parameters (db_file_multiblock_read_count, cpu_count, &c), can influence SQL performance.
- **Tune your SQL Access workload with physical indexes and materialized views** - Just as the 10g SQL Access advisor recommends missing indexes and missing materialized views, you should always optimize your SQL workload with indexes, especially function-based indexes, a Godsend for SQL tuning.

Oracle11g Note: The Oracle 11g SQL Performance Analyzer (SPA), is primarily designed to speed up the holistic SQL[8] tuning process.

Once you create a workload (called a SQL Tuning Set, or STS), Oracle will repeatedly execute the workload, using sophisticated predictive models (using a regression testing approach) to accurately identify the salient changes to SQL execution plans, based on your environmental changes. Using SPA, we can predict the impact of system changes on a workload, and we can forecast changes in response times for SQL after making any change, like parameter changes, schema changes, hardware changes, OS changes, or Oracle upgrades. For details, see the book Oracle11g new Features.

Once the environment, instance, and objects have been tuned, the Oracle administrator can focus on what is probably the single most important aspect of tuning an Oracle database: tuning the individual SQL statements. In this final article in my series on Oracle tuning, I will share some general guidelines for tuning individual SQL statements to improve Oracle performance.

3. Oracle SQL Tuning goals

Oracle SQL tuning is a phenomenally complex subject. Entire books have been written about the nuances of Oracle SQL tuning; however, there are some general guidelines that every Oracle DBA follows in order to improve the performance of their systems. Again, see the book "Oracle_Tuning:[3] The Definitive Reference", for complete details. The goals of SQL tuning focus on improving the execution plan to fetch the rows with the smallest number of database "touches" (LIO buffer gets and PIO physical reads).

- **Remove unnecessary large-table full-table scans-** Unnecessary full-table scans cause a huge amount of unnecessary I/O and can drag-down an entire database. The tuning expert first evaluates the SQL based on the number of rows returned by the query. The most common tuning remedy for unnecessary full-table scans is adding indexes. Standard b-tree indexes can be added to tables, and bitmapped and function-based indexes can also eliminate full-table scans. In some cases, an unnecessary full-table scan can be forced to use an index by adding an index hint to the SQL statement.

- **Cache small-table full-table scans-** In cases where a full-table scan is the fastest access method, the administrator should ensure that a dedicated data buffer is available for the rows. In Oracle8 and beyond, a small table can be cache by forcing it into the KEEP pool.

- **Verify optimal index usage-** Oracle sometimes has a choice of indexes, and the tuning professional must examine each index and ensure that Oracle is using the proper index.

- **Materialize your aggregations and summaries for static tables** - One features of the Oracle 10g SQL Access advisor is recommendations for new indexes and suggestions [6] for materialized views. Materialized views pre-join tables and pre-summarize data, a real silver bullet for data mart reporting databases where the data is only updated daily. Again, see the book "Oracle Tuning: The Definitive Reference", for complete details on SQL tuning with materialized views.

These are the goals of SQL [9] tuning in a nutshell. However, they are deceptively simple, and to effectively meet them, we need to have a thorough understanding of the internals of Oracle SQL. Let's begin with an overview of the Oracle SQL optimizers.

4. Rules Followed for Query Tuning in this Project

- Rewrite the query such that indexes are used properly

1. 'or' should be written with a union
   for e.g. instead of writing where a=1 or b=3 write where a=1 union b=3
2. relational operator replace with between
   for e.g. instead of writing where a>2 and b<5 write where a between 2 and 5
3. There shouldn't be any formulae along with the attributes in where clause for e.g. where 2*a=40 this should be removed wherever found.

- Avoid using Join if it is not necessary[5] for e.g. suppose SID attribute in table Sailors is a primary key that refers to another attribute SID in table Reserved so the query is – Select S.SID from Sailors S, Reserved R where S.SID=R.SID instead of writing this the query should be changed to Select R.SID from Reserve R;

- Avoid the keyword DISTINCT if it is not necessary for e.g. Select Distinct ssnum from Employee where dept = 'Information Technology' here there is no need of writing DISTINCT keyword because ssnum is itself a primary key.

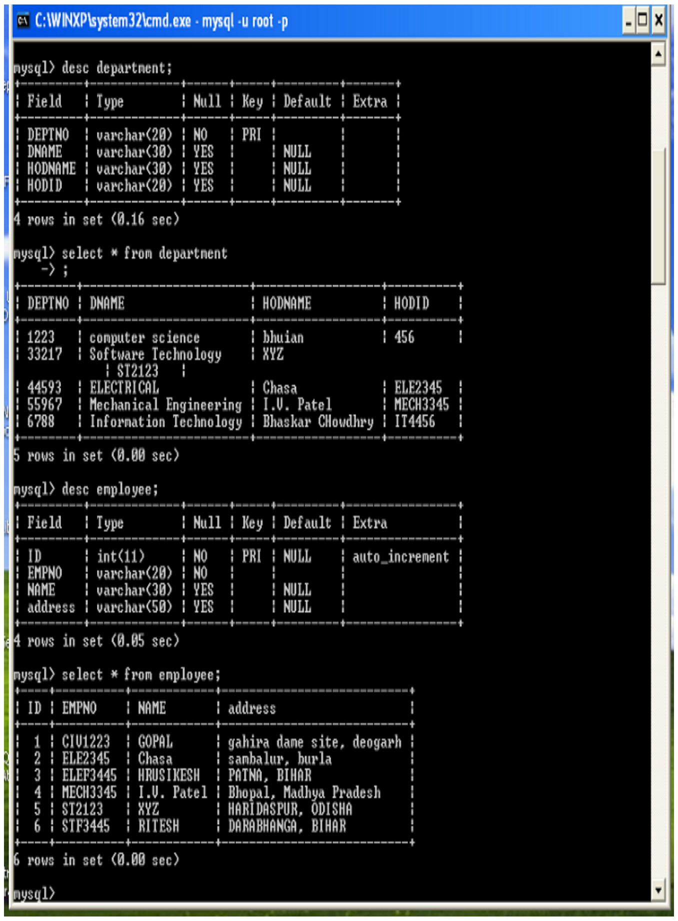
- Avoid using same nested query for e.g. there are 2 tables Employee and Techdept and the query is Select ssnum from Employee where dept in (Select dept from Techdept); instead of writing this we can write select ssnum from Employee, Techdept where Employee.dept = Techdept.dept;

- Avoid using temporary relations if not necessary but where there is necessity there it should be used like- a query for finding all information department employees who earn more than $40000 is written as Select * into Temp from Employee where Salary > 40000; then the next query is Select ssnum from Temp where Temp.dept = 'Information Systems'; so there is no need for using aTemp here and it can be changed to Select ssnum from Employee where Employee.dept = 'Information System' and Salary>40000;[10].

- If a query is long then break it into parts for e.g. select * from T1 T2 where <condition1> group by <attribute> having <condition2>....this should be written as where select * from T1 T2 where <condition2> [11]it will automatically group and show no need of writing groupby.

5. **Proposed Work:**

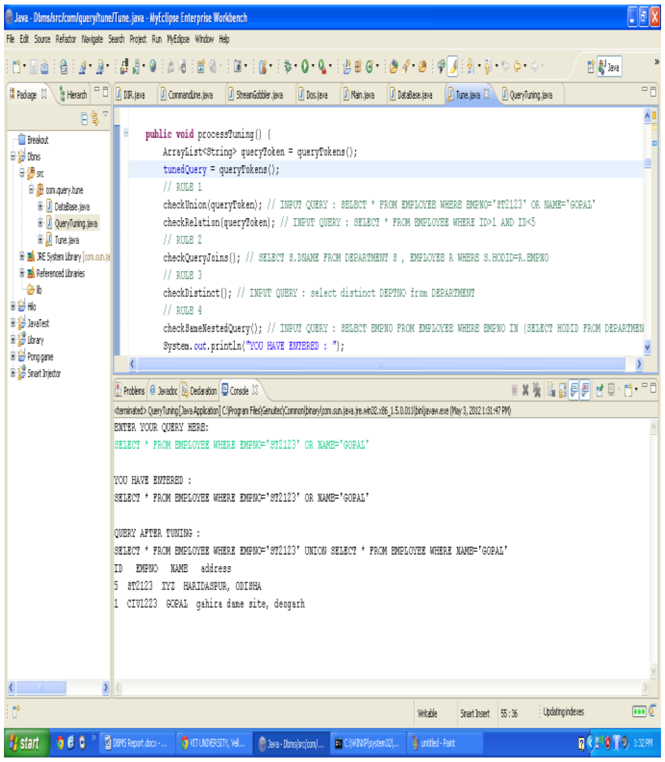
There are three programs in our project. QueryTuning.java contains the main function which calls another two java programs Tune.java and Database.java. Tune.java implements all the rules described above and the DataBase.java does the required database connection and fetches data from the SQL tables.[12] We have created two tables named employee and department which are shown below in the snapshot.



```
C:\WINXP\system32\cmd.exe - mysql -u root -p

mysql> desc department;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| DEPTNO  | varchar(20) | NO   | PRI |         |       |
| DNAME   | varchar(30) | YES  |     | NULL    |       |
| HODNAME | varchar(30) | YES  |     | NULL    |       |
| HODID   | varchar(20) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
4 rows in set (0.16 sec)

mysql> select * from department
    -> ;
+--------+------------------------+----------------+---------+
| DEPTNO | DNAME                  | HODNAME        | HODID   |
+--------+------------------------+----------------+---------+
| 1223   | computer science       | bhuian         | 456     |
| 33217  | Software Technology    | XYZ            |
  | ST2123 |
| 44593  | ELECTRICAL             | Chasa          | ELE2345 |
| 55967  | Mechanical Engineering | I.V. Patel     | MECH3345 |
| 6788   | Information Technology | Bhaskar CHowdhry | IT4456 |
+--------+------------------------+----------------+---------+
5 rows in set (0.00 sec)

mysql> desc employe;
+---------+-------------+------+-----+---------+----------------+
| Field   | Type        | Null | Key | Default | Extra          |
+---------+-------------+------+-----+---------+----------------+
| ID      | int(11)     | NO   | PRI | NULL    | auto_increment |
| EMPNO   | varchar(20) | NO   |     |         |                |
| NAME    | varchar(30) | YES  |     | NULL    |                |
| address | varchar(50) | YES  |     | NULL    |                |
+---------+-------------+------+-----+---------+----------------+
4 rows in set (0.05 sec)

mysql> select * from employe;
+----+----------+----------+--------------------------+
| ID | EMPNO    | NAME     | address                  |
+----+----------+----------+--------------------------+
|  1 | CIV1223  | GOPAL    | gahira dane site, deogarh |
|  2 | ELE2345  | Chasa    | sambalur, burla          |
|  3 | ELEF3445 | HRUSIKESH | PATNA, BIHAR            |
|  4 | MECH3345 | I.V. Patel | Bhopal, Madhya Pradesh |
|  5 | ST2123   | XYZ      | HARIDASPUR, ODISHA       |
|  6 | STF3445  | RITESH   | DARABHANGA, BIHAR        |
+----+----------+----------+--------------------------+
6 rows in set (0.00 sec)

mysql>
```

Fig. 1 Tables used in the project

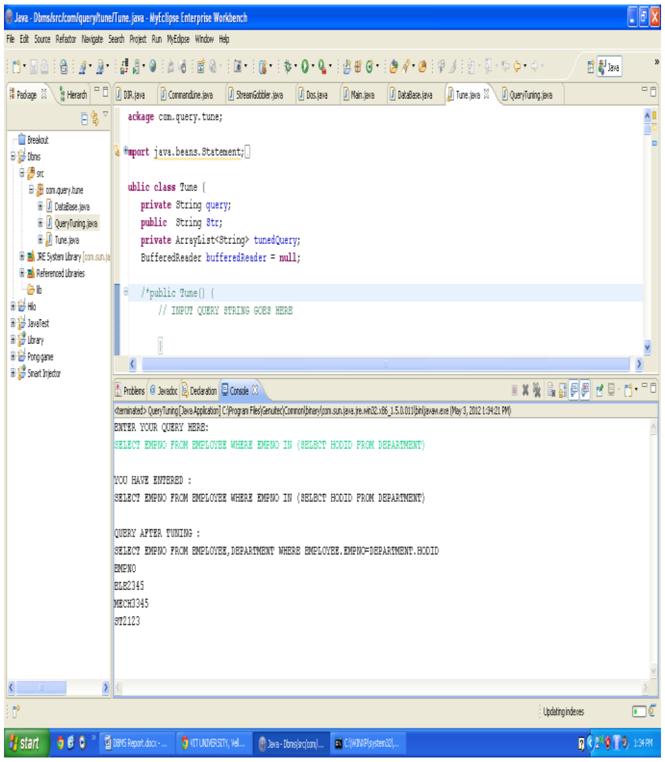
6. Results and Discussion:



```java
public void processTuning() {
    ArrayList<String> queryToken = queryTokens();
    tunedQuery = queryTokens();
    // RULE 1
    checkUnion(queryToken); // INPUT QUERY : SELECT * FROM EMPLOYEE WHERE EMPNO='ST2123' OR NAME='GOPAL'
    checkRelation(queryToken); // INPUT QUERY : SELECT * FROM EMPLOYEE WHERE ID>1 AND ID<5
    // RULE 2
    checkQueryJoins(); // SELECT S.DNAME FROM DEPARTMENT S , EMPLOYEE R WHERE S.HODID=R.EMPNO
    // RULE 3
    checkDistinct(); // INPUT QUERY : select distinct DEPTNO from DEPARTMENT
    // RULE 4
    checkSameNestedQuery(); // INPUT QUERY : SELECT EMPNO FROM EMPLOYEE WHERE EMPNO IN (SELECT HODID FROM DEPARTMEN
    System.out.println("YOU HAVE ENTERED : ");
```

```
<terminated> QueryTuning [Java Application] C:\Program Files\Genuitec\Common\binary\com.sun.java.jre.win32.x86_1.5.0.011\bin\javaw.exe (May 3, 2012 1:31:47 PM)
ENTER YOUR QUERY HERE:
SELECT * FROM EMPLOYEE WHERE EMPNO='ST2123' OR NAME='GOPAL'

YOU HAVE ENTERED :
SELECT * FROM EMPLOYEE WHERE EMPNO='ST2123' OR NAME='GOPAL'

QUERY AFTER TUNING :
SELECT * FROM EMPLOYEE WHERE EMPNO='ST2123' UNION SELECT * FROM EMPLOYEE WHERE NAME='GOPAL'
ID   EMPNO   NAME   address
5  ST2123  XYZ  HARIDASPUR, ODISHA
1  CIV1223  GOPAL  gahira dane site, deogarh
```

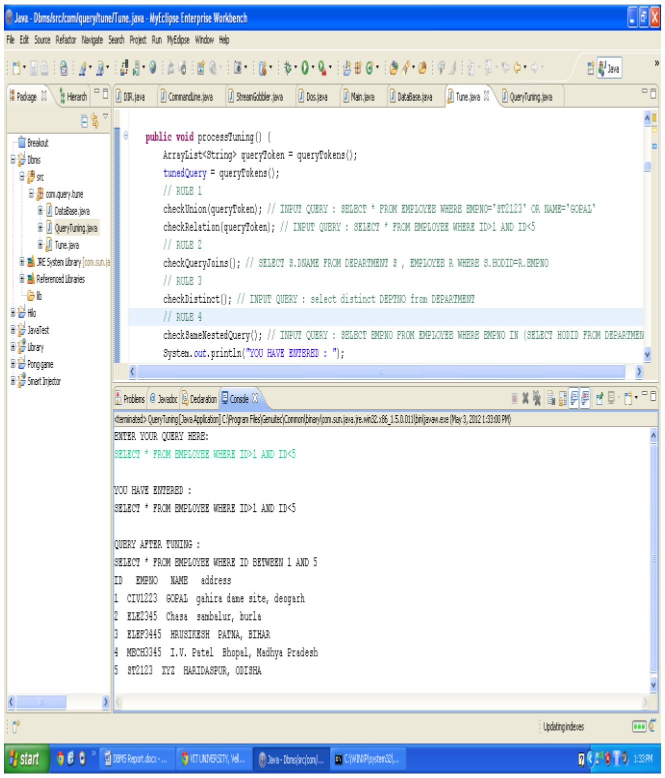
Fig. 2 Change OR to UNION



```java
ackage com.query.tune;

import java.beans.Statement;

ublic class Tune {
  private String query;
  public  String Str;
  private ArrayList<String> tunedQuery;
  BufferedReader bufferedReader = null;

  /*public Tune() {
      // INPUT QUERY STRING GOES HERE

      }
```

```
<terminated> QueryTuning [Java Application] C:\Program Files\Genuitec\Common\binary\com.sun.java.jre.win32.x86_1.5.0.011\bin\javaw.exe (May 3, 2012 1:34:21 PM)
ENTER YOUR QUERY HERE:
SELECT EMPNO FROM EMPLOYEE WHERE EMPNO IN (SELECT HODID FROM DEPARTMENT)

YOU HAVE ENTERED :
SELECT EMPNO FROM EMPLOYEE WHERE EMPNO IN (SELECT HODID FROM DEPARTMENT)

QUERY AFTER TUNING :
SELECT EMPNO FROM EMPLOYEE,DEPARTMENT WHERE EMPLOYEE.EMPNO=DEPARTMENT.HODID
EMPNO
ELE2345
MECH3345
ST2123
```

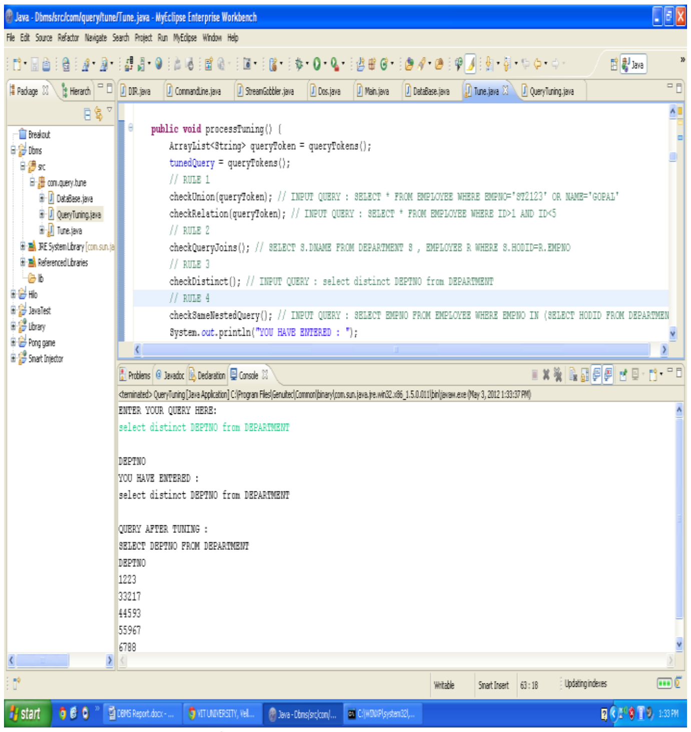
Fig. 3 Remove NESTED SELECT statement



```
public void processTuning() {
    ArrayList<String> queryToken = queryTokens();
    tunedQuery = queryTokens();
    // RULE 1
    checkUnion(queryToken); // INPUT QUERY : SELECT * FROM EMPLOYEE WHERE EMPNO='ST2123' OR NAME='GOPAL'
    checkRelation(queryToken); // INPUT QUERY : SELECT * FROM EMPLOYEE WHERE ID>1 AND ID<5
    // RULE 2
    checkQueryJoins(); // SELECT S.DNAME FROM DEPARTMENT S , EMPLOYEE R WHERE S.HODID=R.EMPNO
    // RULE 3
    checkDistinct(); // INPUT QUERY : select distinct DEPTNO from DEPARTMENT
    // RULE 4
    checkSameNestedQuery(); // INPUT QUERY : SELECT EMPNO FROM EMPLOYEE WHERE EMPNO IN (SELECT HODID FROM DEPARTMEN
    System.out.println("YOU HAVE ENTERED : ");
```

```
<terminated> QueryTuning [Java Application] C:\Program Files\Genuitec\Common\binary\com.sun.java.jre.win32.x86_1.5.0.011\bin\javaw.exe (May 3, 2012 1:33:00 PM)
ENTER YOUR QUERY HERE:
SELECT * FROM EMPLOYEE WHERE ID>1 AND ID<5

YOU HAVE ENTERED :
SELECT * FROM EMPLOYEE WHERE ID>1 AND ID<5

QUERY AFTER TUNING :
SELECT * FROM EMPLOYEE WHERE ID BETWEEN 1 AND 5
ID   EMPNO   NAME   address
1  CIV1223  GOPAL  gahira dame site, deogarh
2  ELE2345  Chasa  sambalur, burla
3  ELEF3445  HRUSIKESH  PATNA, BIHAR
4  MECH3345  I.V. Patel  Bhopal, Madhya Pradesh
5  ST2123  XYZ  HARIDASPUR, ODISHA
```

Fig. 4 Change AND to BETWEEN



```
public void processTuning() {
    ArrayList<String> queryToken = queryTokens();
    tunedQuery = queryTokens();
    // RULE 1
    checkUnion(queryToken); // INPUT QUERY : SELECT * FROM EMPLOYEE WHERE EMPNO='ST2123' OR NAME='GOPAL'
    checkRelation(queryToken); // INPUT QUERY : SELECT * FROM EMPLOYEE WHERE ID>1 AND ID<5
    // RULE 2
    checkQueryJoins(); // SELECT S.DNAME FROM DEPARTMENT S , EMPLOYEE R WHERE S.HODID=R.EMPNO
    // RULE 3
    checkDistinct(); // INPUT QUERY : select distinct DEPTNO from DEPARTMENT
    // RULE 4
    checkSameNestedQuery(); // INPUT QUERY : SELECT EMPNO FROM EMPLOYEE WHERE EMPNO IN (SELECT HODID FROM DEPARTMEN
    System.out.println("YOU HAVE ENTERED : ");
```

```
<terminated> QueryTuning [Java Application] C:\Program Files\Genuitec\Common\binary\com.sun.java.jre.win32.x86_1.5.0.011\bin\javaw.exe (May 3, 2012 1:33:37 PM)
ENTER YOUR QUERY HERE:
select distinct DEPTNO from DEPARTMENT

DEPTNO
YOU HAVE ENTERED :
select distinct DEPTNO from DEPARTMENT

QUERY AFTER TUNING :
SELECT DEPTNO FROM DEPARTMENT
DEPTNO
1223
33217
44593
55967
6788
```

Fig. 5 Remove use of DISTINCT keyword where not necessary

7. Conclusion:

In this project we discussed about the ways of tuning an SQL query in such a manner that it decreases the time consumed by the query during its runtime and also filter some keywords that are not needed to use in the query like for example, the use of DISTINCT keyword has no meaning when we have a search based on a primary key attribute, etc. Also tuning the SQL query increases the performance of the Select, Update and other Data Definition, Data Manipulation and Data Control operations. The main area where further modifications can be done is doing an SQL query optimization in the parallel or distributed databases where the databases are stores in different locations geographically separated. Here there will be a requirement of more complex algorithms for tuning the SQL query.

References

[1] Apers, P.M.G., Hevner, A.R., Yao, S.B. Optimization Algorithms for Distributed Queries. IEEE Transactions on Software Engineering, Vol 9:1, 1983.

[2] Bancilhon, F., Maier, D., Sagiv, Y., Ullman, J.D. Magic sets and other strange ways to execute logic programs. In Proc. of ACM PODS, 1986.

[3] Bernstein, P.A., Goodman, N., Wong, E., Reeve, C.L, Rothnie, J. Query Processing in a System for Distributed Databases (SDD-1), ACM TODS 6:4 (Dec 1981).

[4] Chaudhuri, S., Shim K. An Overview of Cost-based Optimization of Queries with Aggregates. IEEE DE Bulletin, Sep. 1995. (Special Issue on Query Processing).

[5] Chaudhuri, S., Shim K. Including Group-By in Query Optimization. In Proc. of VLDB, Santiago, 1994.

[6] Chaudhuri, S., Shim K. Query Optimization with aggregate views: In Proc. of EDBT, Avignon, 1996.

[7] Chaudhuri, S., Dayal, U. An Overview of Data Warehousing and OLAP Technology. In ACM SIGMOD Record, March 1997.

[8] Chaudhuri, S., Shim K. Optimization of Queries with Userdefined Predicates. In Proc. of VLDB, Mumbai, 1996.

[9] Chaudhuri, S., Krishnamurthy, R., Potamianos, S., Shim K. Optimizing Queries with Materialized Views. In Proc. of IEEE Data Engineering Conference,Taipei, 1995.

[10] Chaudhuri, S., Gravano, L. Optimizing Queries over Multimedia Repositories. In Proc. of ACM SIGMOD, Montreal, 1996.

[11] Chaudhuri, S., Motwani, R., Narasayya, V. Random Sampling for Histogram Construction: How much is enough In Proc. Of ACM SIGMOD, Seattle, 1998.

Mr.P.Karthik B.E., (M.Tech)., He is pursuing M.Tech in the stream of Software Technology at VIT University, Vellore, Tamil Nadu, India and currently doing internship with Alcatel-Lucent, Chennai. He received Bachelor degree in Computer Science and Engineering from Anna University of Technology, Coimbatore, Tamil Nadu, India. He presented research papers in International and National Conferences and his interested areas are Cloud Computing and Databases.

Mr.E.Kaari vanan B.E., (M.Tech)., graduated from Veltech Multitech Engineering College (Affiliated to Anna University Chennai)in 2011.presently pursuing M.Tech Software Technology at VIT UNIVERSITY, Vellore, Tamil Nadu, India and presented research papers in International and National Conferences, his interested areas are Object Oriented Systems and Databases.

Prof. Thippa Reddy.G, M.E.,(CSE) He is currently working as an Assistant Professor at VIT University. He has an experience of above 6 years in the teaching field. He received his Masters Degree in Computer Science and Engineering from Sasurie College of Engineering, affiliated to Anna University, Coimbatore in the year 2010. He received his Bachelors Degree in Computer Science and Engineering in the year 2003 from SVH College of Engineering, affiliated to Nagarjuna University, Guntur. He presented research papers in several International and National Conferences. His interested areas are Cloud Computing, Databases, and Theory of Computation.