Implementing Mobile Agent to Improve Structured Query Language (SQL) Performance

Prof. Dr. Alaa H Al-Hamami & **Imad Hasan Saleh**
Dean of Graduate College for Computing Studies
Alaa_hamami@yahoo.com
Jordan, 2009.

M.SC
imadsaleh71@yahoo.com
Jordan, 2009.

Abstract

Putting any application in production phase can reveal a lot of factors and problems of how transactions and queries use the database. Because any database application performance totally depends on impeded SQL commands written inside it. To enhance the performance, the application source should be modified and the SQL command should be rewritten again.

This paper presents a mobile agent as a middle layer between application interface front end and database back end. The responsibilities of the mobile agent are catching the SQL commands sent by application before reaching the database then examining these commands, correct them if they have errors, and rewrite them in a tuned format if the SQL commands are not tuned. The mobile agent will focus on rewriting the SQL commands without application modification.

Keywords: SQL Tuning, Sequential Search, Binary Search, Transparent layer, Index usage, Mathematical operations, Join Statement.

1. Introduction

Performance Tuning can be achieved in several stages and implemented in all system development life cycle faces. Tuning spans through system development life cycle, starts from analysis stage and extends to deployment stage, but focus in tuning increased in design and development stages. Application tuning is team dependent, and can be achieved by joining the force of database administrators, system administrators, analysts and programmers. So these parties can define their objectives very clearly and can put measurable tasks, because tuning tasks can be achieved in a short time in contrast of general problems.

Mobile Agent (MA) systems have been seen as a promising paradigm for the design and implementation of distributed applications. A mobile agent is a program that can autonomously migrate between various nodes of a network and perform computations on behalf of a user. Some of the benefits provided by MAs for creating distributed applications include reduction in network load, overcoming network latency, faster interaction and disconnected operations [1].

Mobile agent is defined as computer software and its data which has the ability to move from one machine to another transparently to continue its work in the second machine. Also mobile agent has the ability to learn from previous experience as shown in Figure (1).

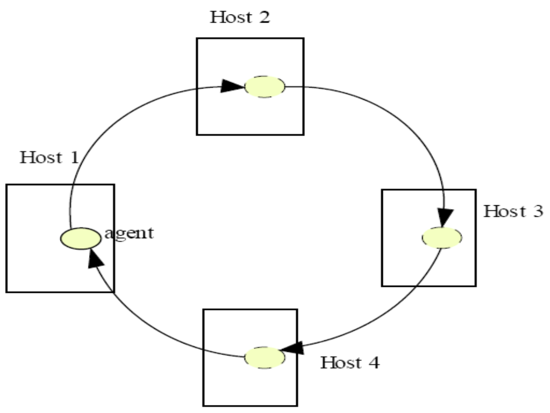


Figure (1) Mobile Agent Movement

Client server model verses mobile agent model. Most of new technologies prefer to use mobile agent instead of traditional client server model to overcome the client server limitations such as flexibility, scalability, cost, fault tolerance, and of course portability. All of these features are exists in client server architecture but can be enhanced and improved if we switched to mobile agent model. In client server model data processed in pipelined fashion at server side and all client information has to be transferred to server throw network. In mobile agent model the agent will move from client to client and process the information locally at client side and the architecture will be as multi server model. So instead of moving data between machines to process it, the program (agent) moves to data location to process it locally as shown in Figure (2). This approach leads us to Grid technology.

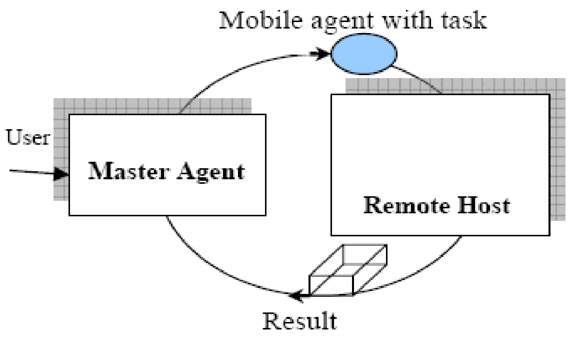


Figure (2) Mobile Agent

By using mobile agent model, processing task can be partitioned into multiple lightweight agents. This family of agents distributed among the cluster and competes for computing resources. This approach of computation is advantageous in that the system operates as an autonomous entity. Agents execute as a collaborative team, working around node failures and system bottlenecks. Additional computing resources can be added and exploited dynamically, enhancing both the system flexibility and scalability [7].

One of the best justifications for using mobile agent is that the paradigm is especially appropriate for computing on network devices. This is so because with proper implementation, mobile agents:

1- Allow efficient and economical use of communication channels which may have low bandwidth, high latency.
2- Enable the use of portable, low-cost, personal communications devices to perform complex tasks even when the device is disconnected from the network.
3- Another attractive property of the mobile agent paradigm is that it allows an application to be truly distributed, as much as the tasks involved in an application,

2. The Problem Statement

The aim of this paper is to execute SQL commands faster than original commands; this can be achieved by implementing many things like: rewriting the SQL command again with new structure to use existing database object efficiently, using mobile agent that navigates through existing servers to minimize network overheads and roundtrips, minimize the incorrect SQL commands written by naïve users by automatically correct them before reaching the RDBMS.

As we know any database application has two parts as shown in Figure (3):

1- The first part (Application) which contains code and application logic, in these part programmers write the SQL commands.
2- The second part (RDBMS) is the physical structure of the application which contains tables, relations, indexes, views and other objects. In fact this part will receive the SQL commands from the first part and then executes these commands to return results back.

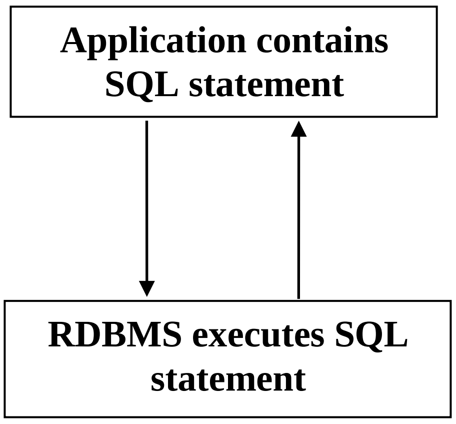

Application contains SQL statement
RDBMS executes SQL statement

Figure (3) Parts of Database Applications

The application may work slow and may have to be revised for the following reasons:

1- Some programmers are novice so they write SQL commands inefficiently and incorrectly, the commands may contain mistakes, syntax errors, also SQL commands may need more time than what it should take, and need a lot of I/O roundtrip.
2- Expert programmers may write a very complex SQL commands but their concern to get the correct results and they do not

concern or give any attention to performance and efficiency.
3- Some SQL commands may work efficiently in development phase, but when we move to production they work very inefficient.
4- Number of users working on the application and the amount of data entered and retrieved from the application may make some SQL commands work inefficiently.
5- Queries may be generated programmatically.

These facts show us that most of programmers and database users can write SQL command in a correct syntax but they do not have the required experience and knowledge to know how the RDBMS optimizer will choose the execution plan to execute the command. So if SQL command written in a professional way, this may help the optimizer to choose to best execution path for this command.

3. The Proposed Solution

The proposed solution will presents the mobile agent as a middle layer between application interface and database system, as in the Figure (4) taking into consideration that the data dictionary is already build and will configured:

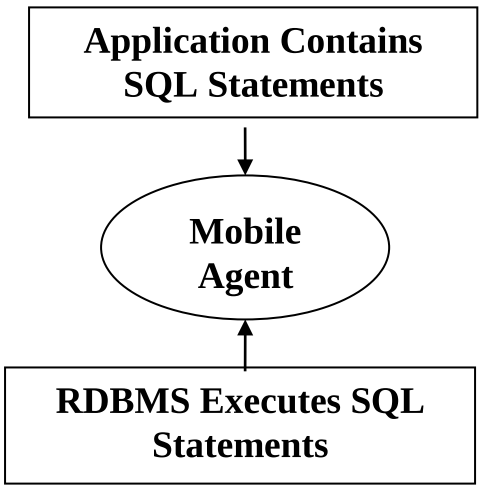

Application Contains SQL Statements
Mobile Agent
RDBMS Executes SQL Statements

Figure (4) Parts of Database Applications with Mobile Agent

The system (agent) use-cases and its interaction with actors are shown in Figure (5).

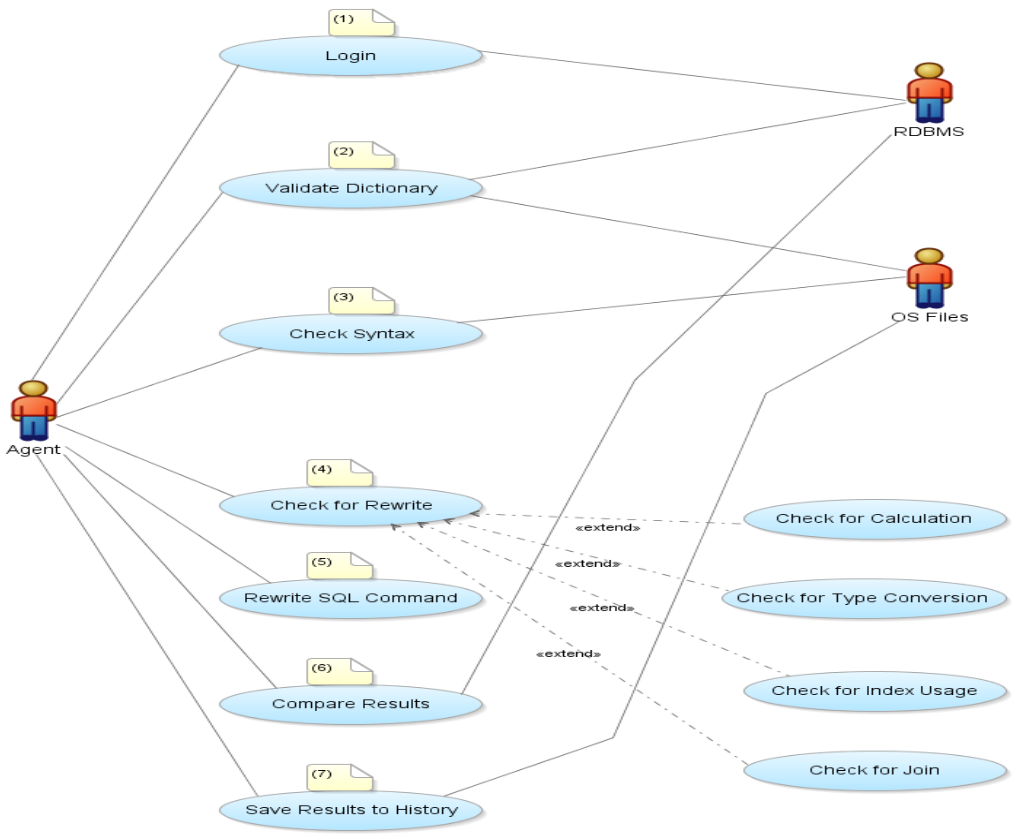


Figure (5) The Agent Use-Case Diagram

Figure (5) shows us three actors:

1- Agent actor as a primary player in the system which will instantiate most of the system use-cases, all of system operations going to start from this actor.
2- RDBMS actor to represent the database that the agent is going to handle, then the agent will execute SQL commands in this database before and after rewriting.

3- OS Files actor to represent the agent dictionary that will be used as a Meta data for the agent where ever the agent travels between hosts.

Actors communicate with use-cases; use-cases are described as bellow:

1- Login use-case to establish connection between our system (agent) and database using authorized username, password and correct connection string.
2- Validate dictionary use-case is going to check if the agent has the correct and matched dictionary as the database, if the agent has dictionary (Meta data) different than database, this use-case will remove and create the dictionary again then save this dictionary in the operating system files.
3- Check syntax use-case will check if the SQL command received has any syntax error before proceeding for tuning this command, and it will report an error if the command is not valid.
4- Check for rewrite use-case will examine the SQL command to see if there is any condition that exists for rewriting the command. These conditions are described in the use-cases that extend from this use-case like checking if the SQL command has mathematical operations, the SQL command has implicit type conversion, the SQL command dose not use indexes, or the join condition is not written in a correct format. All of these conditions will make the SQL command run slowly, it needs to be rewritten.
5- Rewrite use-case will rewrite the SQL command if any command has any condition described previous use-case.
6- Compare results use-case will get all of metrics and statistics for old SQL command (before rewriting) and new SQL command (after rewriting) to show the deference between them and highlight the gain that the agent made by rewriting the SQL command.
7- Save results use-case will store the metrics resulted from pervious use-case to file system for future analysis.

4. Experimental Work

This section presents the tests done by the agent and the performance measurements of old SQL statement before rewriting compared by new SQL statement after rewriting. This section will discuss four types of rewriting. Rewriting SQL command with new structure taking into consideration to make the four recommendations:

- Remove the mathematical operations from the left side of the statement WHERE clause and rewrite it in the right side of the statement.

- Remove and implicit type conversion so the RDBMS will not do any hidden type conversion. And the RDBMS will execute the statement as it is.

- Try any possibility to make the SQL statement to use index instead using full table scan.

- Use none join predicate as a driving table if the SQL statements joins more than two tables.

The test database used in this paper inspired from conference system that servers million of users, Figure (6) shows the database design of the test system. As shown in Figure (3), the COUNTRIES relation used to store the ids, names and codes of client country with client id as a primary key. The CLIENTS relation to store information about clients like name, gender, phone, and more, the CLIENTS relation has client id as primary key and country id as foreign key to COUNTRIES relation. The CONFERENCES relation stores information about each conference done by any clients like conference start date and time, conference end date and time, caller id, and called id. The CONFERENCES relation has conference id as primary key and it has two foreign keys to CLIENTS relation, first one for caller id and second one for called id. The CONFERENCE_DETAILS relation stores information about conference lines as the text sent between each clients, status of each message sent, and if the message received or not. This relation has a sequence as primary key; it has foreign key to CONFERENCES relation and two foreign keys to CLIENTS relation one as sender and the other as receiver.

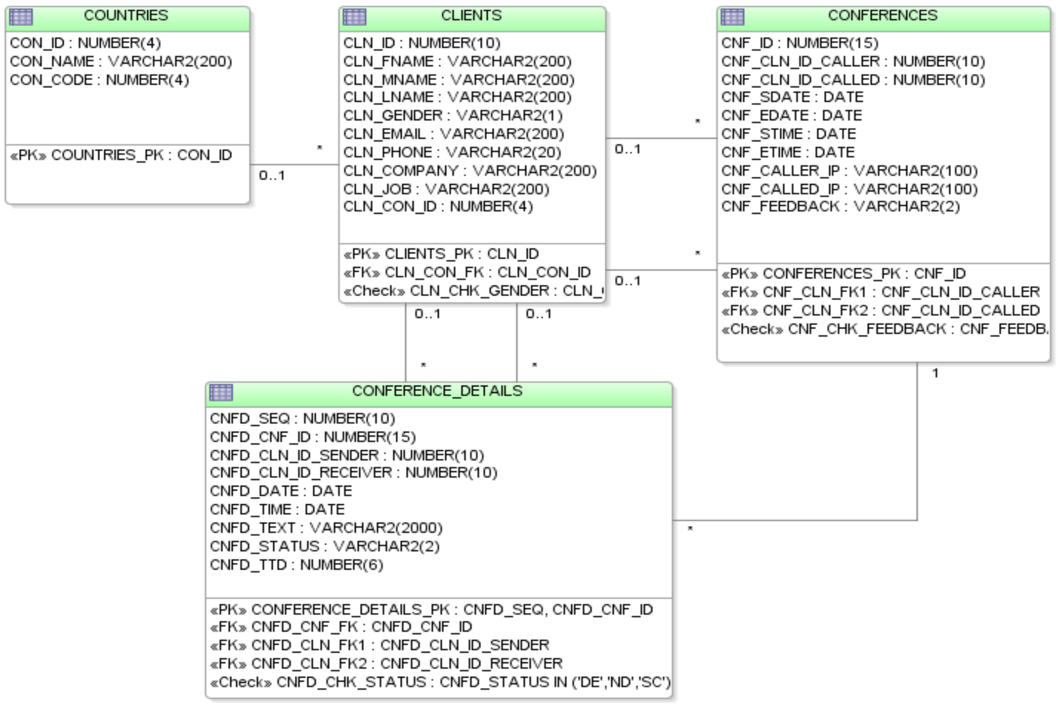

COUNTRIES
CON_ID : NUMBER(4)
CON_NAME : VARCHAR2(200)
CON_CODE : NUMBER(4)
«PK» COUNTRIES_PK : CON_ID
CLIENTS
CLN_ID : NUMBER(10)
CLN_FNAME : VARCHAR2(200)
CLN_MNAME : VARCHAR2(200)
CLN_LNAME : VARCHAR2(200)
CLN_GENDER : VARCHAR2(1)
CLN_EMAIL : VARCHAR2(200)
CLN_PHONE : VARCHAR2(20)
CLN_COMPANY : VARCHAR2(200)
CLN_JOB : VARCHAR2(200)
CLN_CON_ID : NUMBER(4)
«PK» CLIENTS_PK : CLN_ID
«FK» CLN_CON_FK : CLN_CON_ID
«Check» CLN_CHK_GENDER : CLN_
CONFERENCES
CNF_ID : NUMBER(15)
CNF_CLN_ID_CALLER : NUMBER(10)
CNF_CLN_ID_CALLED : NUMBER(10)
CNF_SDATE : DATE
CNF_EDATE : DATE
CNF_STIME : DATE
CNF_ETIME : DATE
CNF_CALLER_IP : VARCHAR2(100)
CNF_CALLED_IP : VARCHAR2(100)
CNF_FEEDBACK : VARCHAR2(2)
«PK» CONFERENCES_PK : CNF_ID
«FK» CNF_CLN_FK1 : CNF_CLN_ID_CALLER
«FK» CNF_CLN_FK2 : CNF_CLN_ID_CALLED
«Check» CNF_CHK_FEEDBACK : CNF_FEEDB
CONFERENCE_DETAILS
CNFD_SEQ : NUMBER(10)
CNFD_CNF_ID : NUMBER(15)
CNFD_CLN_ID_SENDER : NUMBER(10)
CNFD_CLN_ID_RECEIVER : NUMBER(10)
CNFD_DATE : DATE
CNFD_TIME : DATE
CNFD_TEXT : VARCHAR2(2000)
CNFD_STATUS : VARCHAR2(2)
CNFD_TTD : NUMBER(6)
«PK» CONFERENCE_DETAILS_PK : CNFD_SEQ, CNFD_CNF_ID
«FK» CNFD_CNF_FK : CNFD_CNF_ID
«FK» CNFD_CLN_FK1 : CNFD_CLN_ID_SENDER
«FK» CNFD_CLN_FK2 : CNFD_CLN_ID_RECEIVER
«Check» CNFD_CHK_STATUS : CNFD_STATUS IN ('DE','ND','SC')
0..1
*
0..1
*
0..1
*
0..1
0..1
*
*
1
*

Figure (6) Database Design for Conference System

First Scenario: Full Table Scan verses Binary Search

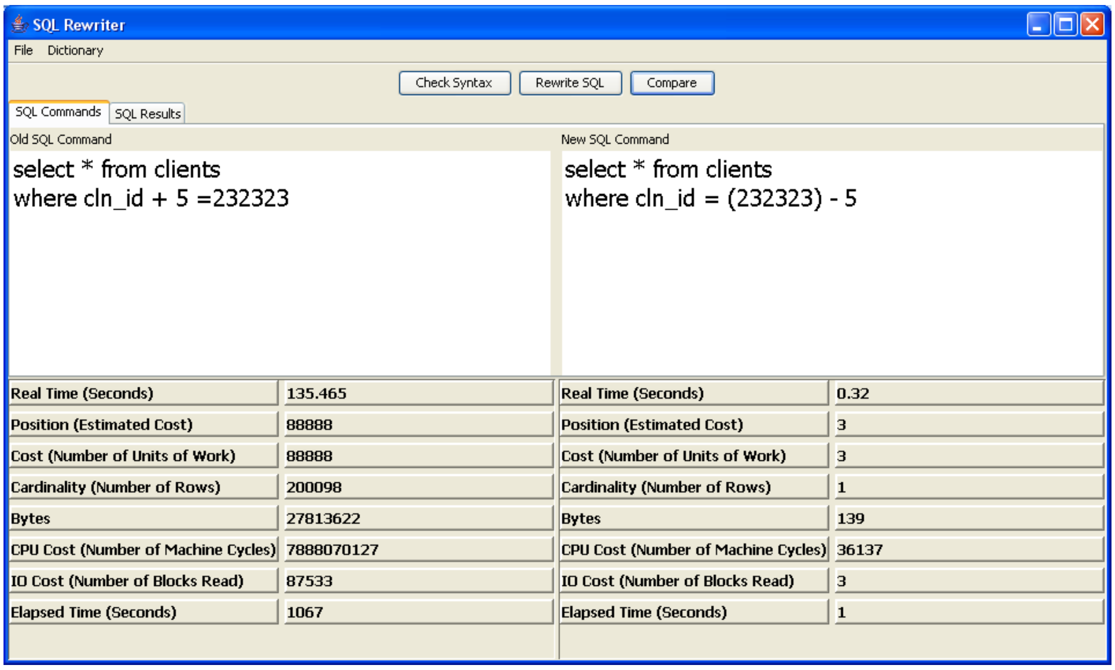
In this scenario we sent SQL statements to the agent with WHERE clause that make the database optimizer to make full table scan decision, then collect the output generated from the agent, the following SQL statements are just an example of statements sent to agent:

```
1- SELECT * FROM CLIENTS
   WHERE CLN_ID + 5 = 232323;

2- SELECT * FROM CLIENTS   WHERE
   ABS(CLN_ID) = 232323;

3- SELECT * FROM CLIENTS   WHERE
   CLN_PHONE = 9615811581;

4- SELECT * FROM CLIENTS
   WHERE
   TO_NUMBER(CLN_PHONE) =
   9615811581;

5- SELECT * FROM CONFERENCES
   WHERE
   TO_CHAR(CNF_SDATE,'MM/YYYY')
   = '10/2008';

6- SELECT * FROM CONFERENCES
   WHERE
   TRUNC(CNF_SDATE) = '22/10/2008';
```

Then we watched the transformed SQL commands generated by the agent who makes the database optimizer to use index search instead of full table scan, then collect the output generated from the agent, the following SQL statements are example of statements generated from the agent:

```
1-  SELECT * FROM CLIENTS
    WHERE   CLN_ID  = 232323 – 5;

2-  SELECT * FROM CLIENTS
    WHERE
    CLN_ID = 232323 OR
    CLN_ID = -232323

3-  SELECT * FROM CLIENTS
    WHERE CLN_PHONE = '9615811581';

4-  SELECT * FROM CLIENTS
    WHERE CLN_PHONE = '9615811581';

5-  ALTER SESSION SET
    NLS_DATE_FORMAT = 'MM/YYYY';

SELECT * FROM CONFERENCES WHERE
CNF_SDATE = '10/2008';

6-  ALTER SESSION SET
    NLS_DATE_FORMAT=
    'DD/MM/YYYY';

SELECT * FROM CONFERENCES WHERE
CNF_SDATE = '22/10/2008';
```

The previous SQL statements are generated from the agent as a result of the SQL statement sent to agent in the first group in the order. So we can match the first statement in first group to first statement in second group and so on to see how the WHERE clause ware rewritten to use the index search feature as shown in Figure(7).



Old SQL Command:

```
select * from clients
where cln_id + 5 =232323
```

New SQL Command:

```
select * from clients
where cln_id = (232323) - 5
```

Real Time (Seconds)	135.465	Real Time (Seconds)	0.32
Position (Estimated Cost)	88888	Position (Estimated Cost)	3
Cost (Number of Units of Work)	88888	Cost (Number of Units of Work)	3
Cardinality (Number of Rows)	200098	Cardinality (Number of Rows)	1
Bytes	27813622	Bytes	139
CPU Cost (Number of Machine Cycles)	7888070127	CPU Cost (Number of Machine Cycles)	36137
IO Cost (Number of Blocks Read)	87533	IO Cost (Number of Blocks Read)	3
Elapsed Time (Seconds)	1067	Elapsed Time (Seconds)	1

Figure (7) Mathematical Operation Rewriting

Second Scenario: Join Statements

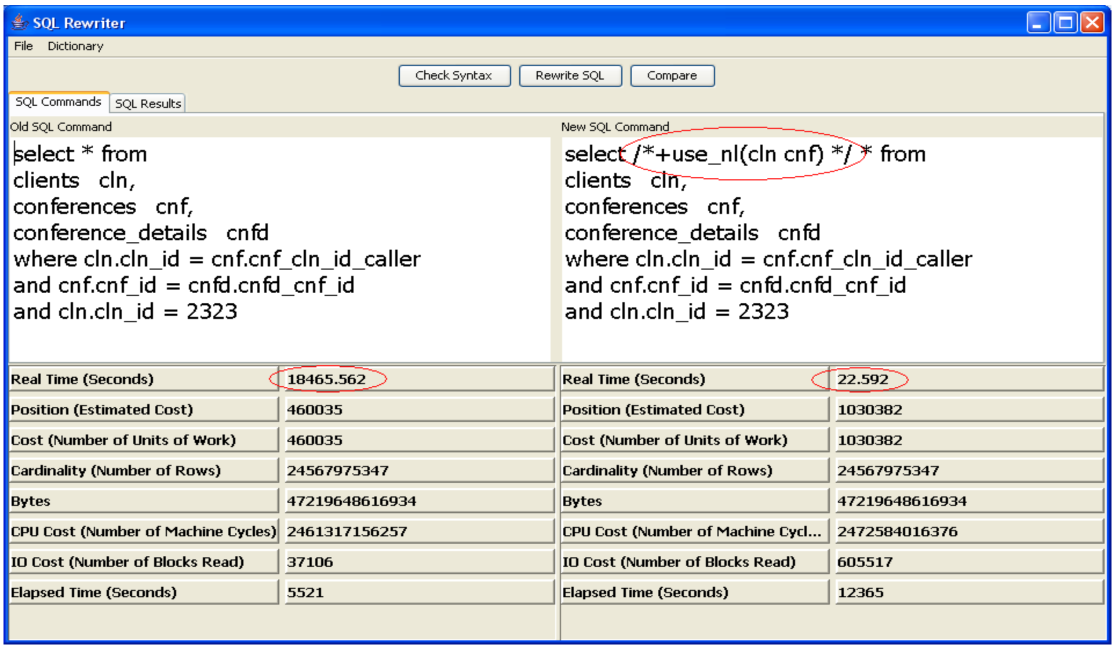
In this scenario we sent join SQL statements to the agent without any attention of none join predicate, the following SQL statements are just an example of this type of statements:

```
1-  SELECT * FROM
    COUNTRIES CON, CLIENTS CLN,
    CONFERENCES CNF
    WHERE CON.CON_ID =
    CLN.CLN_CON_ID
  AND  CLN.CLN_ID=
  CNF.CNF_CLN_ID_CALLER
  AND  CON.CON_NAME = 'JORDAN';


2-SELECT * FROM
  CLIENTS CLN, CONFERENCES CNF,
  CONFERENCE_DETAILS CNFD
  WHERE CLN.CLN_ID =
  CNF.CNF_CLN_ID_CALLER
  AND  CNF.CNF_ID =
  CNFD.CNFD_CNF_ID
  AND  CLN.CLN_ID = 2323;
```

Then we watched the transformed SQL commands generated by the agent who makes none predicate join as a driving table:

```
1-  SELECT /*+USE_NL(CON CLN) */ *
    FROM
    COUNTRIES CON, CLIENTS CLN,
    CONFERENCES CNF
    WHERE CON.CON_ID =
    CLN.CLN_CON_ID
    AND         CLN.CLN_ID =
    CNF.CNF_CLN_ID_CALLER
    AND         CON.CON_NAME  =
    'JORDAN';


2-  SELECT /*+USE_NL(CLN CNF) */ *
    FROM
    CLIENTS CLN, CONFERENCES CNF,
    CONFERENCE_DETAILS CNFD
    WHERE CLN.CLN_ID =
    CNF.CNF_CLN_ID_CALLER
    AND         CNF.CNF_ID =
    CNFD.CNFD_CNF_ID
    AND         CLN.CLN_ID = 2323;
```

The previous SQL statements generated from the agent as a result of the SQL statement sent to agent in the order, as show in Figure(8).



Old SQL Command:

```
select * from
clients  cln,
conferences  cnf,
conference_details  cnfd
where cln.cln_id = cnf.cnf_cln_id_caller
and cnf.cnf_id = cnfd.cnfd_cnf_id
and cln.cln_id = 2323
```

New SQL Command:

```
select /*+use_nl(cln cnf) */ * from
clients  cln,
conferences  cnf,
conference_details  cnfd
where cln.cln_id = cnf.cnf_cln_id_caller
and cnf.cnf_id = cnfd.cnfd_cnf_id
and cln.cln_id = 2323
```

Real Time (Seconds)	18465.562	Real Time (Seconds)	22.592
Position (Estimated Cost)	460035	Position (Estimated Cost)	1030382
Cost (Number of Units of Work)	460035	Cost (Number of Units of Work)	1030382
Cardinality (Number of Rows)	24567975347	Cardinality (Number of Rows)	24567975347
Bytes	47219648616934	Bytes	47219648616934
CPU Cost (Number of Machine Cycles)	2461317156257	CPU Cost (Number of Machine Cycl...	2472584016376
IO Cost (Number of Blocks Read)	37106	IO Cost (Number of Blocks Read)	605517
Elapsed Time (Seconds)	5521	Elapsed Time (Seconds)	12365

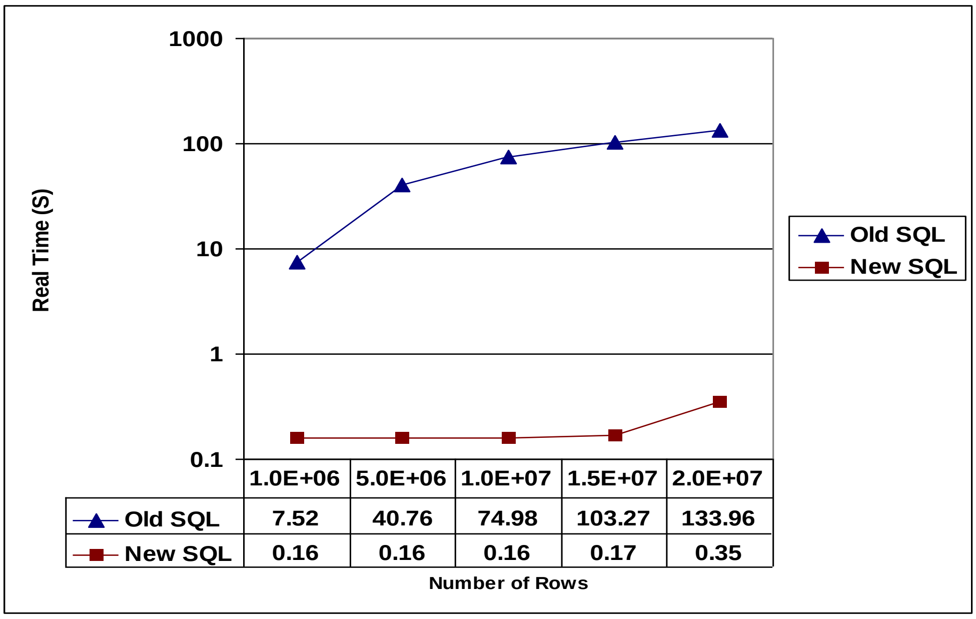
Figure (8) Join Rewriting

SQL command real time execution is the most important measurement used to evaluate the performance between old command and new command. Figure (9) illustrates a logarithmic chart comparison for real time.

The real time results are plotted as logarithmic chart has been analyzed the comparisons detailed as follows:

1- The real time coefficient for old SQL commands increased dramatically according to number of rows in the table, because the old SQL commands use sequential search which is depend on the size of the table and the RDBMS has to go through each record in the table.

2- The real time coefficient for new SQL commands increased slightly with increasing number of rows, because the new SQL command use binary search.



	1.0E+06	5.0E+06	1.0E+07	1.5E+07	2.0E+07
▲ Old SQL	7.52	40.76	74.98	103.27	133.96
■ New SQL	0.16	0.16	0.16	0.17	0.35

Figure (9) Logarithmic Chart for Real Time Results Comparison

5. Conclusion

The mobile agent that receives a SQL command and rewrites it in intelligent way without affecting the resulting data and acquiring better performance has to be intelligent. An agent came into the scene holding the important features to do the job like portability, reliability, self maintained.

Most of the researches focus in deep in the area of SQL command performance enhancements, and most of them donated very good conclusions. Some of them focused on enhancing database optimizer capabilities, others focused on rewriting SQL command by using materialized views. But all of them focused in the area after the database captures the SQL command. This paper did the opposite side; it focused in the area before the SQL command reaches the database. So this paper tried to enhance the performance of the database by sending a well done, error free, and a professional SQL commands. Of course when the database receives a good SQL command it will behave in a good performance.

The agent acts as connection layer between the clients and database server, in the clients side the agent receives the SQL commands and start to study these commands to see if it needs to be rewritten or not. In the server side the agent has to keep an online dictionary of the database to enhance the agent to have a good decision about the SQL commands received by clients. Linking clients with server has to be secured, so the agent has to maintain security issues and authorizations.

References

[1] Brian Brewington, Robert Gray, Katsuhiro Moizumi, David Kotz, George Cybenko and Daniela Rus, "Mobile agents in distributed information retrieval", 1999, Thayer School of Engineering, Department of Computer Science, Dartmouth College, Hanover, New Hampshire.

[2] Bryan Genet, Annika Hinze, "Open Issues in Semantic Query Optimization in Relational DBMS", 2003, Department of Computer Science, University of Waikato, New Zealand.

[3] Chang-Sup Park, Myoung Ho Kim, Yoon-joon lee, "Rewriting OLAP Queries Using Materialized Views and Dimension Hierarchies in Data Warehouses" 2000, supervised by IITA.

[4] Danny B. Lange and Mitsuru Oshima, "Seven Good Reasons for Mobile Agents ", Communications of ACM , vol. 42, no. 3, March 1999.

[5] J.O. Kephart, D.M. Chess. "The Vision of Autonomic Computing", IEEE Computers,36(1), 2003, pp. 41 – 52.

[6] John P. Mckenna, "Aggregate Navigation using Materialized Views and Query Rewrite". 2002, Counterpoint Technologies, Inc.

[7] K. B. Manwade, and G. A. Patil, "Performance Analysis of Parallel Client-Server Model Versus Parallel Mobile Agent Model" 2008, ISSN 2070-3740.

[8] Priya Vennapusa, "Oracle Database SQL Tuning Workshop", 2003, Oracle Corporation.

[9] Yu Jiao and Ali R. Hurson, "Mobile Agents in Mobile Data Access Systems", 2002, Computer Science and Engineering Department Pennsylvania State University