Query Optimization Techniques for OLAP Applications: An ORACLE versus MS-SQL Server Comparative Study

Ahmed El-Ragal
Arab Academy For Science and Technology
College of Management and Technology
Management Information Systems department
P.O. Box 1029, AASTMT, Miami
Alexandria, EGYPT
aelragal@mis.aast.edu

Yehia Thabet
Arab Academy For Science and Technology
College of Management and Technology
Management Information Systems department
P.O. Box 1029, AASTMT, Miami
Alexandria, EGYPT
y_thabet@mis.aast.edu

Abstract

Query optimization in OLAP applications is a novel problem. A lot of research was introduced in the area of optimizing query performance, however great deal of research focused on OLTP applications rather than OLAP. In order to reach the output results OLAP queries extensively asks the database, inefficient processing of those queries will have its negative impact on the performance and may make the results useless. Techniques for optimizing queries include memory caching, indexing, hardware solutions, and physical database storage. Oracle and MS SQL Server both offer OLAP optimization techniques, the paper will review both packages' approaches and then proposes a query optimization strategy for OLAP applications. The proposed strategy is based on use of the following four ingredients: 1- intermediate queries; 2- indexes both B-Trees and Bitmaps; 3- memory cache (for the syntax of the query) and secondary storage cache (for the result data set); and 4- the physical database storage (i.e. binary storage model) accompanied by its hardware solution.

Keywords: query optimization, OLAP, caching, indexing, MS SQL Server, Oracle.

1. Introduction to query optimization

Query Optimization (QO) is a process the success of which affects the entire database (DB) performance [15]. There are basically two types of DB:

1. Archival DB e.g. data warehouse (DW) that is primarily used for query retrievals;
2. Transactional DB e.g. Online Transaction Processing (OLTP) that is primarily used for data maintenance i.e. insert, update, and delete operations.

Efficient QO strategy is a major task in the archival DB types. When a DBMS parses a query it decides the best plan (i.e. strategy) to execute it based on statistics it retains about DB structure, indexes, and number of distinct values. Query optimizer is that part of DBMS that decides which query plan is the best [15]. Relational systems offer the users access to data via high-level language and it's the responsibility of the system to select efficient plans to execute queries called query evaluation plans (qeps) [6].

2. Query optimization mechanisms

There are many query optimization mechanisms. The mechanisms fall into two main categories: hardware (H/W) and software (S/W). *Following is a description of both*:

1. *The H/W mechanisms*:
Currently DB servers make extensive use of multiple processors. DB severs use symmetric multi-processor (SMP) or massive parallel processor (MPP) technology [20]. Some database Management Systems (DBMS) make use of these technologies. DBMS break down a query into parts and process them in parallel by different processors. The most common approach for that is by replicating the query so each copy works against portion of the DB which is usually horizontally partitioned [15] [20]. The same query is to run on each portion in parallel in a separate processor then intermediate results are combined to create the final query set as if the query was running once on the entire DB. A study by [20] reported that the query time was cut by 50% by using parallel processing compared to a table scan (T = TW/P; where T is time, TW is table scan time, and P number of processors). The same study [20] indicated that creating an index using parallel processing was reduced to 5 seconds from nearly 7 minutes.
2. *The S/W mechanisms*:
Indexing the DB is one of the best and cheapest methods for improving performance. An index is a data structure that represents a column stored in a certain order [7]. DB optimizers scan the appropriate index to identify any target rows faster than scanning the entire table. If the table is indexed, a binary search for files is carried out on the blocks rather than on the records. A binary search usually accesses $\log_2$ (b) blocks which is considered an improvement over linear search that is on average accesses (b/2) blocks when found or b blocks if not found [9]. Caching and physical storage are other two

mechanisms through which queries could be optimized [5] [11] [23].

3. Foundations and assumptions

A DB consists of a number of relations. A relation R contains attributes att_1, att_2, …, att_n. R is a subset of the Cartesian product $dom(att_1) \times dom(att_2) \times dom(att_n)$, where $dom(att_j)$ is a set of values for att_j. A tuple t_i is an ordered list of attribute values which has an associated unique identifier (t_id). An expression *e* is used to derive relations and is defined as a group of predicates on some attributes [5]. The length of an expression is the number of attributes involved in the expression. Expressions of length equals 1 are called elementary expressions. For example, an expression of length 2 is like the following:

< price $\in$ [100,200] $\wedge$ type = 'local' >

An expression e_{sub} is a sub expression of *e* if each elementary expression of e_{sub} is included in *e* and *length* (*e*) > *length* (e_{sub}). An expression e' is an extension of *e* if *e* is a sub expression of e' and *length* (e') – *length* (*e*) = 1. An expression e' is a reduction of *e* if e' is a sub expression of *e* and *length* (*e*) – *length* (e') = 1. An expression e' is a neighbor of *e* if e' is an extension of *e* or e' is a reduction of *e*. A generalization enlarges the range of an elementary expression, whilst a specialization reduces the range of an elementary expression [5].

4. Previous work and the research gap

[1] Introduced their query optimization mediator system on which they made no difference between OLTP and OLAP applications. Their contribution resulted in a caching mechanism that allows use of query results of previous queries in case the source is not readily available.

[5] Implemented a framework for query optimization to support data mining applications. The framework proposed concentrated on two main factors; search strategies (hill climber, simulated annealing, and genetic algorithms), and physical DB design.

[6] Proposed a framework for processing a sequence of interdependent queries for which a multi-query optimization is required. Their optimization plan includes: determining on the basis of the dependencies between queries which order they should be specified and which results should we store, then each query is passed separately to the DB optimizer.

[11] Suggested a generalized projections (GP) query optimization technique. GP captures aggregations, group by, projections with duplicate elimination, and duplicate preserving projections. The GP pushes down query trees for select-project-join queries which use any aggregate function. The pushing down technique will result in the removal of tuples and attributes early and consequently leading to smaller intermediate relations and reducing the query cost.

[23] Focused on optimizing selection queries using Bitmaps. For static query optimization, divide the selection into continuous and discrete ones and suggested algorithms for evaluating discrete selections using bit-sliced indexes including time and space constraints.

Reviewing the previous research work-up to our knowledge- indicates that no efforts have been exerted to introduce QO strategy for OLAP applications. The main contribution of this paper is in proposing a novel QO strategy which will focus mainly on supporting OLAP applications.

5. On Line Analytical Processing (OLAP)

Although relational database management systems (RDBMS) are powerful solutions for a wide range of commercial and scientific applications, they are not designed to address the multidimensional information requirements of the modern business analyst, for example forecasting, and classification [3].

The key driver for the development of OLAP is to enable the multi-dimensional analysis [19]. Although all the required information can be formulated using relational database and accessed via SQL, the two dimensional relational model of data and SQL have some serious limitations for investigating complex real world problems. Also slow response time and SQL functionality are a source of problems [3]. OLAP is a continuous and iterative process; an analyst can drill down to see much more details and then he can obtain answers to complex questions.

OLAP represents the use of a set of graphical tools that provides users with multidimensional views of their data and permits them to analyze the data by utilizing simple windowing techniques [15]. OLAP refers to DSS and EIS computing applications [22].

Whilst multidimensionality is the core of a number of OLAP systems available [19], there is a list of elements that determine which OLAP product to purchase:

1. *Multidimensional conceptual view*. The tool must support users with the level of dimensionality needed to enable the required analysis to be carried out;
2. *Transparency*. The heterogeneity of input data sources should be transparent to the users to prevent their productivity decreasing;
3. *Accessibility*. The OLAP system should only access the data required for analysis;
4. *Consistent reporting performance*. As the number of dimensions increases and the database size grows, users will expect the same level of performance;
5. *Client/Server architecture*. The OLAP system has to be compatible with the client/server architectural principles;

6. *Generic dimensionality*. Every data dimension should be in both its structure and operational capabilities;
7. *Multi-user support*. The OLAP system must be able to support a multi-user environment;
8. *Flexible reporting*. The ability to arrange rows, columns, and cells in a way that facilitates visual analysis.

Decision makers should prioritize the previous list elements to reflect their business needs.

Several researchers have stated that OLAP is an independent technique and is as powerful as the data mining process and techniques [2] [10] [13]. [19] stated that in every data mining application the analyst should expect to find some relationships between the variables that describe the data set. These expected relationships need confirmation and any OLAP tool can work well in either confirming or denying these relationships. Because of this, OLAP is one of the data mining techniques applied in the early stages of the data mining process. However, unlike other data mining techniques, OLAP does not learn and hence can not search for new solutions [17] [2].

OLAP involves several basic analytical operations including consolidation, drill-down and statistical techniques [17]:

1. *Consolidation*. Consolidation involves the aggregation of data, e.g. the total number of students at the university, total courses, and average GPA;
2. *Drill-down*. This is the opposite of consolidation and involves more detailed inspection of the underlying data, e.g. the break down of the total number of students into different nationalities that belong to the different majors with different GPA. Drill-down is like adding another attribute to the original report/query [15];
3. *Slicing and dicing*. Slicing and dicing refers to the ability to look at the database from different viewpoints.

6. Query optimization techniques in OLAP applications

QO for OLAP applications differ from optimizing queries for OLTP applications in the following areas:

1. OLAP applications are based on data warehouse which is used for READ type transactions;
2. OLAP applications are characterized by complex queries;
3. In OLTP voluminous data are processed as soon as entered;
4. OLAP users are managers and analysts whereas OLTP users are clerks, professionals (non-managers);
5. OLAP activities are generating queries, ad hoc reports, and statistical reports;
6. OLTP are always equipped with SQL, whilst OLAP front-ends include: DSS, visualization techniques, and/or data mining techniques [22]. Data mining refers to the process of nontrivial extraction of knowledge and discovery of previously unknown information patters.

Because of the prestated reasons this paper is focused on designing strategy for optimizing queries running in OLAP environments. Oracle approach is introduced followed by MS SQL Server approach, and then the details of our suggested approach.

6.1 ORACLE approach

Oracle DBMS make use of MPP. Assume that a SALES relation has many records; therefore query processing on that relation will be so slow. To make sure that table scans are executed in parallel using 5 processors:

```
ALTER TABLE SALES PARALLEL 5;
```

Another parallel scanning option happens during query definition.

```
SELECT /*+ FULL (SALES) PARALLEL (SALES, 5)
*/ COUNT (*)
FROM SALES
WHERE SALESPERSON = 'ALY';
```

The /* */ indicate hint to Oracle which overrides any query plan suggested by the optimizer.

In Oracle adjacent secondary memory space may contain records from various tables with different record structures [15]. Records from two or more joined relations may be stored on the same area on disk by declaring a cluster which is identified by the attribute by which the relations are already joined. Clustering reduces the time required to access related records compared to the normal allocation on various scattered areas on disk. Example is the following:

```
CREATE CLUSTER SALE (CLUSTERKEY
(CHAR(10));

CREATE TABLE PRODUCT_TYPE (ID NUMBER
UNIQUE NOT NULL, DESC CHAR(20), NAME
VARCHAR(25), CONSTRAINT PK_PRODT
PRIMARY KEY(ID)) CLUSTER SALE (ID);

CREATE TABLE PRODUCT (CODE NUMBER
UNIQUE NOT NULL, LOCATION CHAR(20), ID
NUMBER, CONSTRAINT PK_PROD PRIMARY
KEY(ID), CONSTRAINT FK_PRODT_P FOREIGN
KEY (ID) REFERENCES PRODUCT_TYPE (ID))
CLUSTER SALE (ID);
```

Accessing records from the cluster is done via an index on the cluster key. Clustering records is utilized if records are static; if frequent data maintenance occurs clusters create waste space.

An index is a like the structure of a tree that permits direct access to data records in tables [18]. In Oracle indexes are classified by their logical and physical

implementation. From the logical implementation point of view there are:

1. *Single column index* that has only one attribute in the index;
2. *Concatenated columns index* that has composite columns in the index up to 32 or combined size does not exceed 1/3 of Oracle data block size;
3. *Unique index* that guarantees no two rows will have duplicate values;
4. *Nonunique index* on which multiple rows exist for the same value.

From the physical implementation point of view there are:

1. *Partitioned index* is utilized with large table to keep the index entries in several segments which improves scalability and manageability. Partitioned index is used with partitioned tables one for each partition [18];
2. *Nonpartitioned index;*
3. *B-Tree* or balanced tree, on top of which there is the root of the index that has entries which point to the next level of an index. Next level is a branch blocks which also point to the blocks at the next level. Finally, leaf level includes index entries that refer to rows in tables. Oracle also allows the use of reverse key index with B-Tree index structure [18]. A reverse index reverses the bytes of each indexed column whilst retaining the column order. Reverse index is useful for queries which have equality predicates but they never work with range queries.
4. *Bitmap index* is a table of bits such that a row represents the distinct values of a key and each column is a bit which if on implies that the record for that bit column has the associated value [15]. The bitmap index is also organized as a tree but the leaf node stores bitmaps for each value instead of a list of ROWID's in B-Tree. Oracle recommends the utilization of a bitmap index over B-tree in the following cases:
    a. For attributes with low cardinality;
    b. The WHERE clause include OR operators;
    c. Read-Only of low activity attributes.

Table (1): Bitmap index

Row ID	Value	B_7	B_6	B_5	B_4	B_3	B_2
1	3	0	0	0	0	1	0
2	4	0	0	0	1	0	0
3	5	0	0	1	0	0	0
4	4	0	0	0	1	0	0
5	6	0	1	0	0	0	0
6	7	1	0	0	0	0	0
7	3	0	0	0	0	1	0
8	2	0	0	0	0	0	1
9	6	0	1	0	0	0	0
10	5	0	0	1	0	0	0

In a very simple form, bitmap index of an attribute is one vector of bits per attribute value, where the size of each bitmap is equal to the cardinality of the indexed relation. Bitmaps are encoded such that the i^{th} record of v value in an indexed attribute iff the i^{th} bit in the bitmap associated with the attribute value v is set to 1 and the i^{th} bit for the other bitmaps are set to 0 [4]. Table 1 provides an example.

Bitmap indexes have been implemented in many DBMS including Oracle, Sybase, and Informix. A major advantage of bitmap index is that bitmap manipulations uses bit-wise operators (AND, OR, XOR, NOT) are efficiently supported by hardware.

Oracle depends on two QO approaches: ***rule-based***, and ***cost-based*** [12]. The rule-based optimizer ranks the qeps by ranking their possible different paths according to the speed of executing each path. It ignores the table size and the distribution of data. This means that the rule based optimizer will always choose to use an index on a small table even though table scan could be more efficient.

The cost-based optimizer depends on the available access paths and statistics stored in the data dictionary e.g. number of rows and cardinality of a table to decide which access path is of least cost. The following statement analyzes the relation student, views could also be analyzed.

```
ANALYZE TABLE STUDENT COMPUTE
STATISTICS;
```

Use of the analyze command is done whenever changes made to the DB to ensure that the DB optimizer depends on the most updated contents. Oracle recommends running the ANALYZE command after each data load, and before creating summary table in data warehouses [12].

Join is also optimized in Oracle by choosing the best method for doing the join. A join operation combines tuples from two or more relations based on common attributes (i.e. join condition). The Oracle QO chooses the best join method of the following (sort-merge join, nested loop join, hash join, partition wise join) for more details on join methods refer to [9] [12]. For star queries, each dimension is joined to the fact using the primary key-foreign key relationships. Oracle QO uses bitmap index to retrieve rows from the fact then the result is then join to the dimension tables, instead of computing the Cartesian product of the dimensions. Oracle recommends creating a bitmap index on every attribute on the fact table [12].

6.2 MS-SQL Server approach

SQL server 2000 uses memory for its stored procedures, ad hoc and prepared Transact-SQL statements, and triggers. The most significant limitation to SQL server is disk I/O; however, utilizing memory will maximize the advantages of caching and minimize memory swapping.

Memory and lock management are both managed by SQL server not the DB administrator (DBA), the server allocates buffers from the available system memory and releases them when not required [7]. This is also true for the locks, SQL server allocates and releases locks based on the activity level and the expected usage queries will make of the data.

MS SQL supports two index types; clustered and nonclustered to enhance the fast return of result sets [16]. Without indexes a query forces SQL Server to scan the entire table to find matches. A DB index contains one or more column values from tables and pointers to the corresponding record. When performing a query QO uses an index to locate matching records. MS SQL server uses B-Tree structures to store both index types. Microsoft recommends creating index on columns that are frequently used in queries. For instance, query optimizer will use reg_no index to process the following query:
SELECT * FROM STUDENT WHERE REG_NO = 1000;

Creating index on every column in the DB tables will negatively affect performance [16]. Insert, Update, or Delete transactions will trigger the index manager to update the table indexes.

Clustered index contains table records in the leaf node of the B-Tree structure. There is only one clustered index per table. When creating PK constraint in a table without clustered index, SQL Server creates clustered index on the PK column, if a clustered index already exist a nonclustered index is created. An attribute defined as unique automatically creates nonclustered index. On most situations, clustered indexes are created before nonclustered. Index information is obtained by the following statement:

SP_HELPINDEX STUDENT

Following is a create index example in MS SQL Server:

CREATE UNIQUE CLUSTERED INDEX INDEX_1
ON STUDENT(NAME, GPA DESC)
WITH FILLFACTOR = 60

The fill factor sets up the index so that the leaf node index pages are 40% full, leaving 60% space to include additional key entries.

If a clustered index exists (on table or view) any nonclustered index on the same object uses it as their index key [16]. Dropping a clustered index by the DROP INDEX causes all nonclustered to be rebuilt so they use RID as a bookmark. In case the clustered index is re-created using CREATE INDEX all nonclustered indexes are rebuilt to refer to the clustered index rather than RID.

DBCC DBREINDEX (STUDENT, REG_NO, 60)

6.3 MS® SQL Server™ data retrieval policy

SQL Server language is able to filter data at the server so that only the minimum data required is returned to clients which will minimize expensive Client/Server network traffic.

This means that WHERE clauses must be restrictive enough to retrieve only the data that is required by the application. It is always more efficient to filter data at the server than to send it to the client and filter it in the application (this applies to columns). An application issues SELECT * FROM... statement which requires the server to return all column data to the client, whether or not the client application has bound these columns for use in program variables. Selecting only the necessary columns by name avoids high network traffic.

This also makes your application more robust in the event of table definition changes, because newly added columns are not returned to the client application. Moreover, performance depends on how the application requests a result set from the server. An application based on Open Database Connectivity (ODBC), statement options set prior to executing a query to determine how the application requests a data set from the server. By default, MS® SQL Server™ 2000 sends the data set the most efficient way.

MS SQL Server assumes that an application will fetch all rows from a default result set immediately. Therefore, an application must buffer any rows that are not used immediately but could be needed later. This buffering requirement makes it necessary for you to specify (using Transact-SQL) only the data needed.

Query analyzer is a graphical tool in SQL Server that provides information about queries in nodes; each node represents a step in the query to execute [8]. For instance, a SELECT statement shows the following information:

a. Estimated number of rows returned by the statement;
b. Estimated size of rows;
c. Estimated I/O cost;
d. Estimated CPU cost;
e. The number of times the statement was executed during running the query.

6.4 Comparative techniques: ORACLE versus MS-SQL

The following table (20)[1] summarizes the differences between Oracle and MS SQL with regard to QO.

[1] Adapted from [21].

Table (2): Oracle vs. MS SQL Server

Feature	Oracle	MS SQL
Indexing options	B-Tree and Bitmap index structures Reverse key index	B-Tree only
Partitioning	Range, Hash, List and composite partitioning	Cube partitioning DB portioning and file groups
Self-tuning	Self-tuning memory, free space, I/O management	Memory management, lock management
Smart advisors	Memory wizard MTTR advisor Summary advisor Virtual index advisor	Cube wizard Query analyzer
Others	Star Schema	Star Schema and Snowflake designs

It is not the intension of this paper to say which of the Servers (i.e. Oracle and MS SQL) outperforms the other, instead to review the QO approaches in both Servers and introduce an integrated suggested approach QO approach for OLAP applications. However, reviewing both servers' mechanisms is useful for people who intent to develop OLAP applications and are not aware of the details of QO supported by each server.

7. Suggested QO approach for OLAP applications

Our QO suggested approach for OLAP applications includes:

1. The use of intermediate queries. The following procedure is proposed:
   - As a new query (q_n) is processed, it is compared to the previous queries in cache;
   - If there is no attribute intersection between q_n and the previous queries then q_n is processed from scratch;
   - If there is intersection between q_n and a previous query q_m then:
     - If same operator exist:
       - E.g. q_n : AGE > 10, q_m : AGE > 20. Then finding tuples that satisfy AGE between 11 and 20 then the result Union AGE > 20 from cache.
       - E.g. q_n : AGE > 20, q_m : AGE > 10. Then finding tuples that satisfy AGE greater than 20 as a subset of cache.
     - If different operators:
       - E.g. q_n : AGE > 10, q_m : AGE < 15. Then find the intersection between the two queries (i.e. 11, 12, 13, 14) from cache, then find tuples that satisfy AGE >= 15 and Union them.
   - If there is intersection between q_n and more than one previous query q_m , q_j , q_l then process the new query q_n with the one with which it has the highest selectivity (n/N; where n is the number of records selected and N is the total number of records in a relation).

2. The use of indexes both B-Trees and Bitmap where relevant. Following is a B tree versus Bitmap comparison which helps determines when to use each of which:

Table (3): B-tree vs Bitmap [Adapted from [18]]

B-tree	Bitmap
High cardinality attributes	Low cardinality attributes
Inexpensive updates	Expensive
Inefficient for OR queries	Efficient for OR queries
OLTP	OLAP

3. The use of memory cache (for the syntax of the query), and secondary storage cache (for the result data set).
4. Physical DB storage and its hardware solution.

Points 3, and 4 of the suggested QO approach are explained in the following sections.

7.1 Pointer-based Binary Storage Model "BSM"

The idea of the binary storage model is to store each attribute with a unique location identifier in a separate table. In [5] if a non-elementary expression is processed each attribute is accessed with its physical address. We propose a modified Binary storage model called Pointer-Based binary storage model. In this model we have to specify the origin table "Table that contains the primary key" This table contains the primary key and a number of pointers equal to n-1 where n is the number of columns in the virtual table including the primary key. The following example shows the proposed modeling technique.

Example. Consider the following student relation (i.e. table):

Table (4): Student table

Student ID	Student Name	Student GPA	Student AGE
91131911	Ahmed	2.5	25
95176112	Yehia	2.97	24
98117611	Mohamed	3.68	26
99876511	Ali	3.92	21

We name the last shown table structure *'**virtual table structure**'* as it is stored in a different way than it appears to the user.

7.2 Physical Structure :

Each column will be stored in a table with a pointer that points to the original location.

Student Column

Table (5): Physical structure

Physical Location	Student ID Physical Location	Student Name
3000	1101	Ahmed
3001	1305	Ali
3002	2107	Mohamed
3003	908	Yehia

.

GPA column

Table (6): Physical structure-1

Physical Location	Student ID Physical Location	GPA
4000	1305	2.5
4001	2107	2.97
4002	1101	3.68
4003	908	3.92

AGE column

Table (7): Physical structure-2

Physical Location	Student ID Physical Location	AGE
5000	1305	21
5001	2107	26
5002	1101	25
5003	908	24

Student Original Table

Table (8): Student Original table

Physical Location	Student ID	Name Physical Location	Age Physical Location	GPA Physical Location
908	91131911	3003	5003	4003
1101	99876511	3000	5002	4002
1305	98117611	3001	5000	4000
2107	95176112	3002	5001	4001

Suppose that we want to execute the following query:
SELECT * FROM STUDENT WHERE GPA > 2.5

In this case we will search in the GPA table which is sorted, so binary search will be suitable to be applied. After determining the set of tuples that match the selection criterion data tuples are retrieved from the origin table that are stored in physical locations similar to those stored in the pointer attribute.

After determining those tuples data is retrieved from the rest of tables as shown:

Table (9): Results

Student ID Physical Location	GPA
2107	2.97
1101	3.68
908	3.92

Then data exist in physical locations are retrieved from Original table.

Table (10): Source table

Physical Location	Student ID	Name Physical Location	Age Physical Location	GPA Physical Location
908	91131911	3003	5003	4003
1101	99876511	3000	5002	4002
2107	95176112	3002	5001	4001

And then data is retrieved from the locations appear in table.

The above algorithm seems to have considerable cost. In fact this is not true because this pointer point to the physical location so there is no search after retrieving the original table selected rows.

7.3 Proposed H/W architecture to speed up search

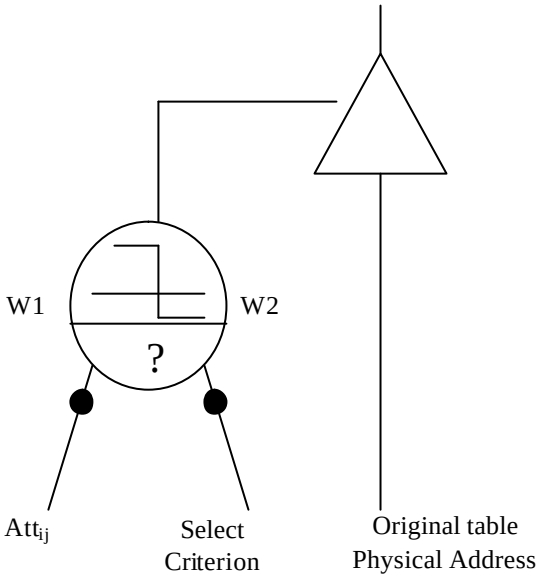

Figure (1): Suggested H/W architecture

The previous Architecture works as follows:

The Att_{ij} is feed to the network with the selected criteria, if the where operator is > then W1, W2 equal -1,1 respectively. if the where operator is < then W1,W2 equal 1,-1 respectively. If it is equal then both assumptions are valid.

The activation function is the inverse of the bipolar function. It is defined by:

$$f(x) = \begin{cases} 1 if x < 0 \\ 0 if x >= 0 \end{cases} \quad (1)$$

The output of the neuron is feed to a tri-State device in order to either output zero and that means there is no match or outputs the physical address in case of match.

Note that VLSI (very large scale integrated circuits) technology allows the integration of huge number of replicates of this architecture in one chip. And so parallel processing for the query is achievable.

Assumptions and constraints:

1. For the technique to be efficient, each table has to be sorted in order to achieve efficient data retrieval "binary search". Note that we don't need to use B-tree algorithm as it is too costly for a sorted table.
2. Each group of tables constituting a virtual table must be stored in the same cluster in order to speed up the operation.

7.4 Query Result Cashing

Query cashing process is divided into two sub tasks "memory cash and secondary storage cash". We cash the query semantics and its native language transformation into memory whilst the results of the query are cashed in secondary storage i.e. hard-disk.

Since the storage media is limited in space, it is required to develop a replacement technique in order to replace existing queries by newer ones. In this paper we offer a priority-based technique that is based on the following mathematical equation:

$Q_{IP} = B(Qi) + H(Qi)$

Where Q_P is the priority of the query number I. Whenever a new query is executed, it is cashed on disk and it is given a starting value B(Qi) where B(Qi) is a function that retrieves the base value of the priority and it is calculated as follows:

$$B(Qi) = \lceil Avg(Query\ \Pr iorities) \rceil$$

and H(Qi) is a function that represents the Query Hit Ratio which initially equals 0. Query Hit Ratio is the number of times this query has been used to answer another query.

Note that we formulated Q_{IP} to include B(Qi) in order to give a fair chance for new queries to remain cashed. That is why we do not consider B(Qi) a part of the Exact Q_{IP} Value so each time a new query is cashed the function B(Qi) is decremented by 1 for all queries until it reaches 0. Note That If Q2 is a specification to Q1 and Q1 is cashed then Q2 is not cashed.

Cash Modification:
Any cashed Query that can be considered as an extension of any other cashed query must be removed.

8. Conclusion

- Indexes, memory caches, portioning, clustering, hardware solutions (e.g. parallel processing) are all mechanisms to optimize query performance;
- Query optimization for OLTP applications is different from QO for OLAP applications;
- Bitmaps are common indexes for OLAP applications;
- Both MS SQL and Oracle Servers support optimize queries differently;
- A suggested QO strategy for OLAP applications should include the following components:
  - Intermediate queries;
  - Indexes both B-Trees and Bitmaps;
  - Memory cache (for the syntax of the query) and secondary storage cache (for the result data set);
  - The physical database storage (i.e. binary storage model) accompanied by its hardware solution.

9. Future work

- Implementing the suggested QO approach for OLAP applications in ORACLE. *The reasons for which we did not implement our suggested approach in a case study are the following:*
  - It is very hard to communicate with the DBMS' query optimizers;
  - Some DBMS (e.g. MS SQL) do not support some of the suggested techniques like bitmap indexes;
  - It is costly to implement the suggested hardware (H/W) solution using VLSI (very large scale integrated circuits). But this does not mean that the H/W solution is infeasible, it is feasible for large organizations who store millions of records;
  - Dumping the cached queries from memory and secondary storage requires adding new module to the existing DBMS.
- Handling sub queries by using heuristics to determine which intermediate queries to retain.

References

[1] Adali, S., Candan, K., Papakonstantion, Y., and Subrahmanian, V. (1996), ***Query caching and optimization in distributed mediator systems***, proceedings of SIGMOD 1996, pp.137-148.

[2] Adriaans, P., and Zantinge, D. (1996), ***Data Mining***, Addison Wesley Longman Limited, Harlow.

[3] Berson, A. (1996), ***Client/Server Architecture***, McGraw-Hill, New York.

[4] Chan, C., and Ioannidis, Y. (1999), ***Bitmap index design and evaluation***, technical report, department of Computer Science, Wisconsin-Madison University.

[5] Choenni, S., and Siebes, A. (1996). '***A framework for query optimization to support data mining***', CWI, Amsterdam.

[6] Choenni, S., Kersten, M., Van den Akker, J., and Saad, A. (1996). '***On multi-query optimization***', CWI, Amsterdam.

[7] Craig, R. (1999). '***Performance Management***', ENT, July vol. 4: 13, pp.41.

[8] Craig, R., Vivona, J., and Bercovitch, D. (1999), ***Microsoft Data Warehousing: Building Distributed Decision Support Systems***, John Wiley & Sons Inc., New York.

[9] Elmasri, R., and Navathe, S. B. (2000), ***Fundamentals of Database Systems***, 3rd ed., Addison Wesley, Massachusetts.

[10] Fayyad, U., Piatetsky, G., and Smyth, P. (1996), 'From Data Mining To Knowledge Discovery: An Overview', In Fayyad, U., Piatetsky, G., and Smyth, P. (Eds), ***Advances In Knowledge Discovery And Data Mining***, AAAI Press/ The MIT Press, California, pp. 1-34.

[11] Harinarayan, V., and Gupta, A. (1997), ***Generalized Projections: a powerful query-optimization technique***, technical report, department of Computer Science, Stanford University.

[12] Hobbs, L., and Hillson, S., (2000), ***Oracle 8i Data Warehousing***, Digital Press, Boston.

[13] Laudon, K., and Laudon, J., P. (2001), ***Essentials of Management Information Systems: Organization and Technology in the Networked Enterprise***, 4th ed., Prentice Hall International, Inc., New Jersey.

[14] ***MCDBA SQL Server 7.0 Administration Study Guide,*** (1999). Osborne McGraw-Hill, California.

[15] McFadden, F., Hoffer, J., and Prescott, M. (1999). ***Modern Database Management***, 5th ed., Addison Weslley, Massachusetts.

[16] ***Microsoft SQL Server 2000 Database Design and Implementation training kit*** (2001). Microsoft press, Washington.

[17] O'brien, J. (1996), '***Introduction to Information Systems'***, Irwin, Chicago.

[18]***ORACLE 8: Database Administration volume 2*** (1998). Oracle University, California.

[19] Pyle, D. (1999), ***Data Preparation For Data Mining***, Morgan Kaufmann Publishers, Inc., California.

[20] Schumacher, R. (1997). '***Oracle Performance Strategies'***, DBMS 10, (May): pp.89-93.

[21] ***Technical Comparison of Oracle Database vs. IBM DB2 UDB: Focus on Performance*** (2002). An Oracle white paper.

[22] Turban, E., and Aronson, J. (1998), ***Decision Support Systems and Intelligent Systems***, 5th ed., Prentice Hall, New Jersey.

[23] Wu, M. (1998), ***Query Optimization for Selections using Bitmaps***, Germany.