An Optimization Algorithm for Physical Database Design

ADI-CRISTINA MITEA
Computer Science Department
"Lucian Blaga" University of Sibiu
Zaharia Boiu street No. 2-4, 2400 Sibiu
ROMANIA

Abstract: - Databases are almost everywhere. Today, software applications used in different domains contains and manage databases. The performance of these applications is determined in a great deal by the database structure and its capability to respond in a short and consistent way to the users needs. So, database design is an important step in the information system life cycle and it should be done in a proper manner. The paper presents an optimization algorithm for physical database design based on "Cleanroom" model. The algorithm helps to obtain an optimal structure for the database and to eliminate errors from the database designing process.

Key-Words: - physical database design, transactions analysis, "Cleanroom" model

1 Introduction

We live in an information technology era. Computers are used in industry, in medicine, in education, in science, in entertainment. They are practically everywhere. The software produced for these computers uses in a great proportion databases. The database can be considered the foundation of these information systems. If we want to have a solid and robust information system it is important that the foundation is well designed. So, database design is a part of a larger process called information system design. In this system, the data are not only collected, stored and retrieved, they are also transformed into valuable information which helps decision makers to take the decisions that allows their organizations to grow and to become more powerful.

Each information system has a life cycle and database design is a part of it [1]. A good database design is essential for the performance and success of the future information system. The process of designing the database is divided in three phases: conceptual design, logical design and physical design. Conceptual database design is the process of building a conceptual data model for the data used by an organization or a company, which is totally independent from the physical implementation details, the DBMS used to implement the database, the application programs, the programming languages and the hardware platform. Logical database design is the process of transforming the conceptual data model into a logical data model, based on a chosen data model, but still independent from physical implementation details. Physical database design is the process of deciding physical storage structures for the data files and access methods used to make an efficient data access [2].

2 Physical database designing process

Physical database design is the third and last phase of database designing process, but it plays a very important role in the performance and success of the final database. The phase deals with decisions about physical storage structures for data files and decisions about the best access methods used to make an efficient data access. If the physical structure of the database is not well suited, it will be very poor performance when the database will be in its operational phase.

The designing process of a database starts with the conceptual design phase. In this phase, the designer has to understand and specify all database requirements and to elaborate the conceptual model for it. Conceptual database model is, afterwards translated by the designer into a logical schema based on a chosen data model. Starting from the logical schema of the database, the physical database designer has to elaborate the physical schema of the database, according with the characteristics of the chosen DBMS and the previous database designing process.

An important part of the physical designing process of a database is represented by transactions analysis. The paper presents a different method for analyzing the transactions performed on a database and an optimization algorithm for physical database design. To facilitate the designer decisions I propose a

physical designing method based on "Cleanroom" model. This method uses formal specification and mathematical validation techniques to support designing process and to eliminate designing errors.

"Cleanroom" system development model [3] was inspired from the semiconductors manufacturing process. It is an incremental model and each step (increment) is formal specified and validated through static verification techniques. The model philosophy is based on errors avoidance rather then on errors detection and repair. Static verification techniques are used through the system development phases to assure that designing errors are eliminated from the designing process. The proposed physical designing method is based on this philosophy, too. The transactions analysis phase is formal specified and mathematical validation rules are applied in the designing process to eliminate designer errors. This helps designers to elaborate the database structure on a good and accurate basis and to choose the best access methods for it.

3 The algorithm

The optimization algorithm proposed in this paper, touches especially the transactions analysis phase from the physical database designing process. It is a novel method, based on formal specification and mathematical validation rules, which assist the designer in choosing the best structure for the database.

Transactions are logical units of work [4], which support concurrent access on a database. A transaction is a sequence of several operations performed on a database that transforms a consistent database state into another consistent database state. Each transaction has to conform to the following properties: *atomicity*, *consistency*, *isolation* and *durability*.

The major goal of transactions analysis phase is to completely understand transactions functionality and to determine which transactions have a greater influence on database performance and dictate database structure and access methods.

To make an efficient physical database design it is necessary to bring together all the knowledge about transactions performed on the database. These information are about *quantity* and *quality* of a transaction. For each transaction is important to know:

- an estimated frequency;
- response time constraint;
- peak load hours;
- all relations accessed by a transaction;
- types of access for each relation (read/write);
- attributes relevant for the transaction performance.

One way transactions analysis is made is through transactions usage maps. This is a formal method for transactions analysis and uses graphical representation [2]. If there is a large database, with a lot of relations, and a great number of transactions performed on it, is rather difficult to build the global transactions usage map and to make the analysis on it.

The paper proposes another method to make this analysis, which is easier to be done and introduces some metrics and rules to help the designer choose the best solution.

All the transactions, identified at the requirements specification level of the conceptual design phase, have to be analyzed to see which are more often used, which relations they involve and which are the operations performed in each transaction. This information helps to identify the best structures for the data files and the best auxiliary access structures for them.

For each transaction the *quantity* information is collected and summarized in a table having the following structure (Table 1):

Table 1. Quantity information about transactions.

Transaction ID	Transaction Name	Transaction Frequencies	Peak Load Hours	Maximum Response Time	Calculated Grade G_T

Where

- *Transaction ID* – is a unique identifier chosen to identify a transaction.
- *Transaction Name* – is the name given to the transaction.
- *Transaction Frequencies* – is the estimated frequency for the transaction.
- *Peak Load Hours* – are the hours when the transaction is more often processed.
- *Maximum Response Time* – is the constraint about the response time of the transaction.
- *Calculated Grade* – is a grade calculated for the transaction, which indicate if the transaction is a critical one.

It is stipulated in literature [5] that almost every application has a 20% of transactions, which make 80% of transformations in the database. The database designer needs to determine which are the transactions from the 20% area. For each transaction is calculated a grade which indicate if the transaction is a critical one and it belongs to the 20% area. That helps to

eliminate the transactions, which are performed only a
few times and at large intervals of time.

The transaction grade is calculated after the formula (1):

$$G_{Ti} = pf_{Ti}Gf_{Ti} + pc_{Ti}Gc_{Ti} + pt_{Ti}Gt_{Ti} \quad (1)$$

Where
pf_{Ti} – is the weight of transaction frequency in the final grade of transaction Ti
Gf_{Ti} – is the grade of transaction frequency for transaction Ti
pc_{Ti} - is the weight of concurrency execution in the final grade of transaction Ti
Gc_{Ti} - is the grade of concurrency execution for transaction Ti
pt_{Ti} - is the weight of response time constraint in the final grade of transaction Ti
Gt_{Ti} - is the grade of response time constraint for transaction Ti

The transaction frequency grade Gf_{Ti} is obtained with (2).

$$Gf_{Ti} = F_{Ti} \frac{\Delta G}{F_{MAX} - F_{MIN}}. \quad (2)$$

Where
F_{Ti} – is the estimates frequency for transaction Ti
ΔG – is the grades range
F_{MAX} – is the maximum value for transactions frequencies
F_{MIN} – is the minimum value for transactions frequencies ($F_{MIN} \neq 0$)

The transaction concurrency execution grade Gc_{Ti} is calculated as indicated below.

A Δt reference time interval is chosen. A day is divided in *i* time intervals Δt_i according with Δt ($0<i<n$, $n = \frac{24\ h}{\Delta t}$) .

A $V_1(i)$ array is build.

$$V_1(i) = (N_1, N_2, \ldots, N_i, \ldots, N_n)$$

Where $N_i = \sum_{j=1}^{M} (F_{Ti} * Ph)$

M - is the total number of transactions identified in the requirements specification phase
F_{Ti} – is the estimates frequency for transaction Ti
Ph = 0 , if the transaction Ti is not executed in Δt_i interval
Ph = 1 , if the transaction Ti is executed in Δt_i interval

The array $V_1(i)$ is transformed in array $V_2(i)$ according with (3):

$$V_2(i) = V_1(i) \frac{\Delta G}{V_{1MAX} - v_{1MIN}}. \quad (3)$$

Where
$V_1(i)$ – is the current value from array V_1
ΔG – is the grades range
V_{1MAX} – is the maximum value from array V_1
V_{1MIN} – is the minimum value from array V_1 ($V_{1MIN} \neq 0$)

The transaction concurrency execution grade Gc_{Ti} can now be obtained with (4):

$$Gc_{Ti} = MAX(V_2(j)) \quad (4)$$

Where
j corresponds to the interval Δt in which the transaction Ti is located.

The transaction response time constraint grade is obtained with (5).

$$Gt_{Ti} = \frac{\Delta G(R_{MAx} - R_{MIN} - R_{Ti})}{R_{MAX} - R_{MIN}} \quad (5)$$

if $0<R_{Ti} < R_{MAX}$

or

$Gt_{Ti} = 0$ if $R_{Ti} = 0$

or

$Gt_{Ti} = \Delta P$ if $R_{Ti} = R_{MAX}$

Where
ΔG – is the grades range
R_{Ti} – is the value of the response time constraint for transaction Ti
R_{MAX} – is the maximum value for response time constraints
R_{MIN} – is the minimum value for response time constraints ($R_{MIN} \neq 0$)

A threshold grade G_T is chosen.

The list $L_T = \{T_a, T_b, \ldots, T_m\}$ is build up with all transactions which has $G_{Ti} \geq G_T$.

For the transactions belonging to list L_T the analysis is continued. Information about all the operations performed by each transaction from list L_T are identified and collected in table 2. The type of operation is indicated and also the relation on which that operation is performed.

For each transaction, another grade Gf'_{Ti} is calculated with (6).

$$Gf'_{Ti} = F_{Ti}\, Nop_{Ti} \frac{\Delta G}{F_T Nop_T \big|_{MAX} - F_T Nop_T \big|_{MIN}}. \quad (6)$$

Where
F_{Ti} – is the estimates frequency for transaction Ti
Nop_{Ti} - is the total number of operations performed by transaction Ti
ΔG – is the grades range
$F_T Nop_T \big|_{MAX}$ – is the maximum value for total number of operations performed by a transaction taking in consideration also the transaction frequency
$F_T Nop_T \big|_{MIN}$ – is the minimum value for total number of operations performed by a transaction taking in consideration also the transaction frequency ($F_T Nop_T \big|_{MIN} \neq 0$)

A threshold grade Gf' is chosen.

The list $L'_T = \{T_a, T_b, \ldots, T_m\}$ is build up with all transactions which has $Gf'_{Ti} \geq Gf'$. At this point, the quantity transactions analysis is finished. The transactions from list L'_T have great influence on

database activity and have to be analysed more closely. For them the quality analysis is continued.

Table 2. Transactions quality analysis.

Relation / Transaction	Relation $_1$				Relation $_2$				Relation $_3$				…Relation $_n$				Nop	Trans. Grade Gf'
	S	I	U	D	S	I	U	D	S	I	U	D	S	I	U	D		
Trans_ID$_1$																		
Trans_ID$_2$																		
Trans_ID$_3$																		
…																		
Trans_ID$_m$																		
For L'$_T$ transactions																		
Op./Rel.																		
Total S/(I+U+D)																		
Total I/(U+D)																		
Total (S+I+U+D)																		
Relation Grade G_R																		
Concurrency Grade Gc_R																		
For L_R relations																		
$I_{activity}$																		
$I_{type\_activity}$																		

Where

N_{SRi} – is the total number of SELECT operations performed on relation Ri

N_{IUDRi} – is the total number of INSERT+UPDATE+DELETE operations performed on relation Ri

N_{SIUDRi} – is the total number of SELECT+INSERT+UPDATE+DELETE operations performed on relation Ri

A relation grade is also calculated with the following formula:

$$G_{Ri} = N_{SIUDRi} \frac{\Delta G}{N_{SIUDMAX} - N_{SIUDMIN}}. \quad (7)$$

Where

G_{Ri} – is the grade calculated for relation Ri

N_{SIUDRi} – is the total number of operations performed on relation Ri

ΔG – is the grades range

$N_{SIUDMAX}$ – is the maximum value for N_{SIUD}

$N_{SIUDMIN}$ – is the minimum value for N_{SIUD} ($N_{SIUDMIN} \neq 0$)

A threshold grade G_R is chosen for relation grades.

The list $L_R = \{R_a, R_b, \ldots, R_m\}$ is build up with all relations

which has $G_{Ri} \geq G_R$.

These relations are the critical one. There is a lot of activity performed on those relations. The analysis continues to determine if those relations are suitable for auxiliary access structures.

The estimated size for each base relation is calculated taking in consideration the estimated

For each database relation Ri, N_{SRi}, N_{IUDRi} and N_{SIUDRi} are calculated.

number of rows for that relation and the medium length of a row. All the relations, who have a dimension less than a threshold dimension *Dim*, are eliminated from list L_R. *Dim* is the table dimension from which an auxiliary access structure is efficient. (*Dim* ≈ 150 Ko).

I defined two metrics to measure the activity performed on a database relation and to be able to select the relations suitable for auxiliary access structures:

- $I_{activity}$ – is the value of the activity index for each relation.
- $I_{type\_activity}$ – is the value of the activity type index for each relation.

For the relations and transactions selected in the previous steps the indices will be calculated and analyzed.

The activity index is defined as follow in (8):

$$I_{activity} = \frac{N_S}{N_I + N_U + N_D} \quad (8)$$

Where

N_S – is the number of search operations

N_I – is the number of insert operations

N_U – is the number of update operations

N_D – is the number of delete operations

If $I_{activity} > 1$ the relation is more often searched then updated, deleted or inserted, so it is a good candidate for an auxiliary access structure.

If $I_{activity} < 1$ the relation is more often updated, deleted or inserted then searched, so it is not a good candidate for an auxiliary access structure

If the update and/or delete operations do

not imply more than 10-15% of the database, another index should be analyzed. This index is called activity type index:

$$I_{type\_activity} = \frac{N_I}{N_U + N_D} \quad (9)$$

Where

N_I – is the number of insert operations

N_U – is the number of update operations

N_D – is the number of delete operations

If $I_{type\_activity} > 1$ the relation is not suitable for an auxiliary access structure.

If $I_{type\_activity} < 1$ the relation is suitable for an auxiliary access structure.

After the suitable relations for auxiliary access structures are identified, the attributes for the index keys had to be identified. For this task the following analysis is made.

Table 3. Green and red attributes.

Relation Name	Primary Key	Transaction ID	Attributes		Index Keys	Selectivity	Index Type
			Green	Red			

Where

- *Relation Name* – is the name of a base relation obtained from table 2
- *Primary Key* – is the primary key of the relation
- *Transaction ID* – is the transaction identifier for the transaction who generates the green or red attributes
- *Green Attributes* – are attributes suitable to be used in the index key
- *Red Attributes* – are attributes not suitable for the index key
- *Index Keys* – are the possible index keys for that relation
- *Selectivity* – is the number of distinct values for the index key. If there is a small number, of distinct values for the index key, it will be more suitable to choose a bitmap structure for the index. If there is a big number it will be more suitable to use a B tree index structure.
- *Index Type* – is the type of the index structure. It could be a normal B tree index, a bitmap index, a reverse index, if the concurrent usage degree of that relation is high, a partitioned local or global index, if that relation is partitioned, a join index, if there is a join operation between that relation and another relation from the database [6,7,8,9]

Green attributes correspond to the attributes used in search criteria, join criteria, selection criteria for updates and deletes which do not modify more that 10-15 % of the database.

Red attributes correspond to the attributes used in update operations and also are involved by insert and delete operations.

The rules for red and green attributes selection are:

INSERT
Red attributes = All attributes
(especially for massive inserts)
UPDATE
Red attributes = Updated attributes
Green attributes = WHERE conditions attributes
DELETE
Red attributes = All attributes (for massive deletes)
Green attributes = WHERE conditions attributes
SELECT
Green attributes = WHERE conditions attributes
Green attributes = JOIN conditions attributes
Green attributes = GROUP BY conditions attributes
Green attributes = ORDER BY conditions attributes
Green attributes = attributes with DISTINCT option

For each relation it will be obtained a list of green attributes and a list of red attributes. Each green attribute and each red attribute has a counter associated with it, which indicates how often this attribute is used. Those lists are used to generate the possible index keys. From these possible index keys, those, which have the counter greater than a threshold value, are selected to represent the index keys for the auxiliary access structures build for database relations.

For large databases, partitioning is used to improve concurrency. To determine which are the relations with a higher degree of concurrency, the following analysis is made:

A threshold grade Gc is chosen for transaction concurrency execution.

The list $Lc_T = \{T_a, T_b, \ldots, T_m\}$ is build up with all transaction from table 1 which has $Gc_{Ti} \geq Gc$.

For each base relation R_i, the number N_{Ri} of transactions from list Lc_T, which execute operations on transaction R_i, is calculated with (10).

$$Gc_{Ri} = N_{Ri} \_\_\frac{\Delta G}{N_{MAX} - N_{MIN}}. \quad (10)$$

Where

N_{Ri} – is the number of transactions which can be executed concurrently on relation Ri

ΔG – is the grades range

N_{MAX} – is the maximum value for N_{Ri}

N_{MIN} – is the minimum value for N_{Ri} ($N_{MIN} \neq 0$)

A threshold grade Gc_R is chosen.

The list $Lc_R = \{R_a, R_b, \ldots, R_m\}$ is build up with all relations from table 2 which has $Gc_{Ri} \geq Gc_R$.

For all relations from list Lc_R the opportunity of a partition is analysed.

All the needed information is collected in a table like table 4.

Table 4. Partition analysis

Relation Name	Transaction ID	Predicate	Attribute	Value	Partition Key	Partition Type

Where

- *Relation Name* – is the name of the base relation which need to be partitioned
- *Transaction ID* – is the identifier of the transaction which performs operations on that relation
- *Predicate* – is the predicate for a possible partition on that relation
- *Attribute* – are the attributes implied in the predicate
- *Value* - are the values for the attributes
- *Partition Key* – is a possible partition key for that relation
- *Partition Type* – is the type of the partition (HASH, RANGE, LIST, etc [6, 7, 8, 9])

The partitioning keys are made of the most used attributes in the predicates of the relation. For each relation from list Lc_R, the possible partitioning keys are identified, taking in consideration the attributes used in WHERE conditions for select, update and delete operations and in JOIN and GROUP BY conditions of selects. Each of these possible partitioning keys has a counter associated with it, which determines the most suitable partitioning key. The attributes values for the partition key determine the partition type: hash, range list.

At this point, the best data structures for all database relations can be selected and, also, the best access methods for them.

4 Conclusion

The paper presents a different approach to physical database design. It is based on "Cleanroom" model philosophy and it introduces a novel method for transactions analysis. A designing algorithm is proposed and the steps of this algorithm try to optimize the database structure. Different grades are calculated for each transaction and each relation defined in the conceptual database design phase, and according with threshold values and some metrics and rules introduced by the method, the optimal structure of the database is obtained. This analysis helps designers to elaborate the database structure on a good and accurate basis and to choose the best access methods for it

Proposed transactions analysis method is a useful tool for relational or object-relational database designers. It is a designing formal method and it contains static verification techniques to validate the designing process. The idea is based on "Cleanroom" model and try to eliminate the designing errors from the designing process with the aid of mathematical static verification and validation techniques. At the end, a database storage structure is obtained and, also, additional access structures that will improve the database performance. The errors are eliminated from the design and the physical schema of the database is obtained on a consistent and accurate basis. The costs of the testing phase and the final total costs of the system development are reduced in this manner.

It is important for the success of the future application that this designing phase is well done. A bad data files storage structure and auxiliary access structures for that data files can harm very much the database performance and can compromise the hole information system success. This method for transactions analysis helps designers to take the best decisions and to obtain the optimal structure for the database.

Another great advantage of this method is that it can be automated because is based on mathematically validation techniques. In fact, now, I am testing the software application that follows this designing algorithm. With the help of this software the physical designing process will be much easier to be done and the costs of the designing process will be reduced.

References:

[1] P. Rob and C. Coronel, *"Database systems – design, implementation and management"*, Boyd & Fraser Publishing Company, 1995.

[2] T. Connolly, C. Begg and A. Strachan, *"Database systems – A practical approach to design, implementation and management"*, Addison-Wesley Longman Limited, 1998.

[3] H.D. Mills, M. Dyer and R.C. Linger, *"Cleanroom software engineering"*, IEEE Software 4, 5, 1987, pp. 19-25.

[4] C.J. Date, *"An introduction to database systems"*, Addison-Wesley Longman Limited, 1994.

[5] G. Wiederhold, *"Database design"*, New York McGraw-Hill, 1983.

[6] *"Oracle 8i Concepts"*, Oracle Corporation, Release 8.1.5, 1999.

[7] *"Oracle 9i Concepts"*, Oracle Corporation, Release 9.1.2, 2002.

[8] *"SQL Server 2000"*, Microsoft Corporation, 2000.

[9] *"DB2 Fundamentals"*, IBM Corporation, 2000.