Local and Global Query Optimization Mechanisms for Relational Databases

Kazuhiro Satoh, Masashi Tsuchida, Fumio Nakamura, Kazuhiko Oomachi

Systems Development Laboratory, Hitachi, Ltd.

1099 Ohzenji, Asao-Ku, Kawasaki 215, JAPAN

ABSTRACT

In a relational database management system (DBMS), query optimization is essential since the DBMS must produce candidate internal access strategies from a high level query and select the optimal one. The goals of implementing a good optimization mechanism are: (a) to maximize the performance of selected access strategies and (b) to minimize the optimization overhead. We have developed an optimization mechanism, built it in a relational DBMS product at Hitachi. The mechanism can estimate precise tuple selectivity of predicates in queries for optimal access strategy selection, and employs a new method, called the cascade method, for optimization overhead reduction. This paper also proposes a global optimization mechanism which executes optimization over mutiple database queries.

1. INTRODUCTION

One of the most important points of relational database management systems (DBMSs) is the prevention of performance degradation due to supporting high-level user interfaces. In order to use a DBMS based on the network or hierarchical model, the user must specify how to access to a database (here we call it "how-representation") in detail using DML (Data Manipulation Language) commands which essentially provide navigational database access to the user. In a relational DBMS, on the other hand, the user only specifies what to access to the database (here we call it "what-representation"), set-level access of database records, in other words, using the query language without bothering about the structural details of the database. Therefore, the relational DBMS must translate "what-representation" to "how-representation" to execute database access requests. In the translation process, query optimization is needed since there can be many "how-representation"s corresponding to one "what-representation". During query optimization, the DBMS selects the most efficient "how-representation", as shown in Figure 1.

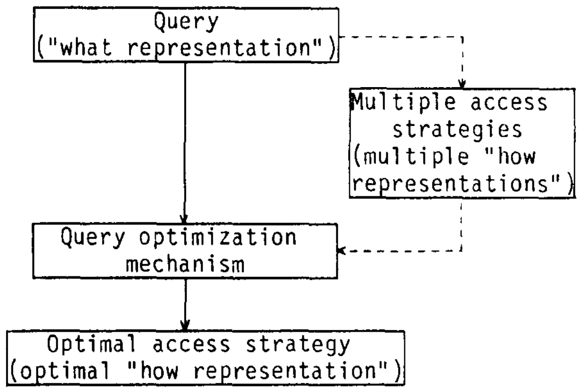

Query
("what representation")
Multiple access strategies
(multiple "how representations")
Query optimization mechanism
Optimal access strategy
(optimal "how representation")

Fig.1 Role of query optimization

As described above, query optimization is essential in a relational DBMS since the DBMS must produce candidate internal access strategies (i.e., "how-representation"s) from a high-level query (i.e., "what-representation") and select the optimal one. Therefore, the DBMS must provide a good query optimization mechanism. The goals of implementing the good optimization

mechanism are:
(a) to maximize the performance of selected access strategies, and
(b) to minimize the optimization overhead.

There have been many papers related to query optimization(6). We classify optimization into two categories: local and global optimization. Local optimization is for a single query (i.e., intra-query), while global optimization is for multiple queries (i.e., inter-query). Most research has been done on local optimization (1,2,5,7,8,9,10,12,13,14,16).

(1,2,8,9,10,13,16) are based on the cost evaluation method. (2) and (16) proposed cost models for join operation in a two-variable query. On the other hand, (5,7,12,14) are based on the heuristic rule method. (7) proposed a semantic query processing method using artificial intelligence type rules to generate efficient "how-representation".

By the cost evaluation method, it is possible to select the optimal "how-representation" corresponding to a specified "what representation", but it requires a high optimization overhead. With the heuristic rule method, however, the optimization overhead is low, but the problem is the optimality of "how representation" selected.

We have developed a hybrid mechanism which combines the above two methods, have built it in a relational DBMS product in Hitachi, and have made an enhancement of the mechanism. The mechanism can estimate precise tuple selectivity of predicates in queries for optimal access strategy selection, and employs a new method, called the cascade method, for optimization overhead reduction.

There have not been many studies(3,6) in global optimization . But global optimization offers the potentiality of improvement their performance of DBMSs. From this point, we propose a mechanism for global optimization.

The paper is divided into six sections. Section 2 describes the classification of optimization for query processing and presents outlines. Section 3 presents the details of the local optimization method we propose. In section 4, we describe a global optimization method. Section 5 is a discussion of the propositions and section 6 summarizes the paper.

2. OPTIMIZATION OF QUERY PROCESSING IN RELATIONAL DATABASES

In this section, we classify query optimization into two categories from the view point of the number of database commands involved in the process of optimization (see Figure 2).

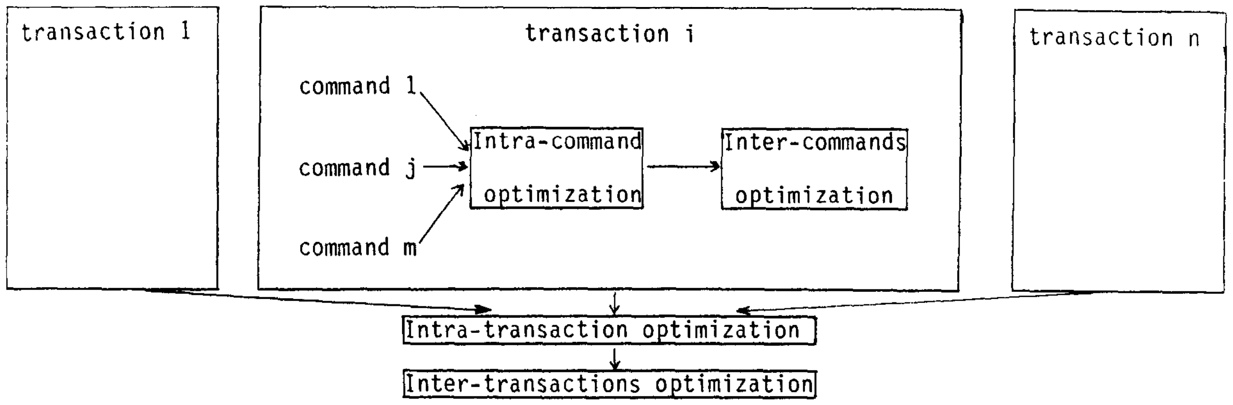
(1) Local optimization of query processing

Local optimization is for single database commands. The chief problem of this type of optimization is how to translate a "what representation" into an efficient "how representation". There are several methods for this optimization:
(a) the cost evaluation method .
(b) the heuristic rule method .
(c) the hybrid method .

(a) is a method that selects the optimal "how representation", which requires the minimum cost in respect with some performance factors (e.g., cpu time, input/output time for secondary storage, storage spaces) among "how representation"s for the given "what representation" through the cost evaluation mechanism(1,8,9,10,13). In this cost evaluation, the following characteristics are considered: data characteristics (e.g., distribution of attributes' values, selectivity of predicates, access paths available, relationships among attributes), database characteristics (e.g., internal scheme information, database size), system characteristics (e.g., data processing mechanisms that the DBMS can use, cpu power, storage device power, buffer pool size), and query characteristics (e.g., query pattern, query complexity). System R's optimization mechanism(1,10) is an example of the method (a). However, some of the characteristics described above are not considered in the system. One of the disadvantages of this method is optimization overhead, especially the generation of all candidate "how representation"s and their cost evaluation.

(b) is a method that generates optimal "how representation"s through the rules (algebraic transformations, common sub-expression recognition, etc.) which are set up beforehand, based on mathematical theorems, data characteristics and query characteristics described above(5,7,12,14). An example that employs the method (b) is PRTV(5,14) which is an experimental system. One of the disadvantages of this method is in the degree of accuracy for optimal "how representation"s established. Compared with the method (a), however, the optimization overhead is small.

(c) is a method that combines (a) and (b). Therefore, this method has both the cost evaluation mechanism and the rule application mechanism. The method reduces the number of "how representation"s to be generated by applying the rule application mechanism, evaluates costs of remaining "how representation"s using the cost evaluation mechanism and decides the


transaction 1
transaction i
transaction n
command 1
command j
command m
Intra-command optimization
Inter-commands optimization
Intra-transaction optimization
Inter-transactions optimization

(a) Classification of optimization

Classification	Object of each optimization			Processing outlines
Global optimization(1)	Transactions (Inter-transaction optimization)			* Clustering of transactions * Scheduling of transaction-clusters
Global optimization(2)	A transaction (Intra-transaction optimization)	Commands (Inter-commands optimization)		* Clustering of commands * Scheduling of commands-clusters
Local optimization(1)		A command (Intra-command optimization)	(Boolean expression optimization)	* Query modification * Clustering of predicates * Pruning of processing strategy with rules * Cost evaluation of processing strategy
Local optimization(2)			A predicate (Predicate optimization)	* Selection of a suitable evaluation method for each predicate

(b) Processing outlines of each optimization

Fig. 2 Classification of query optimization

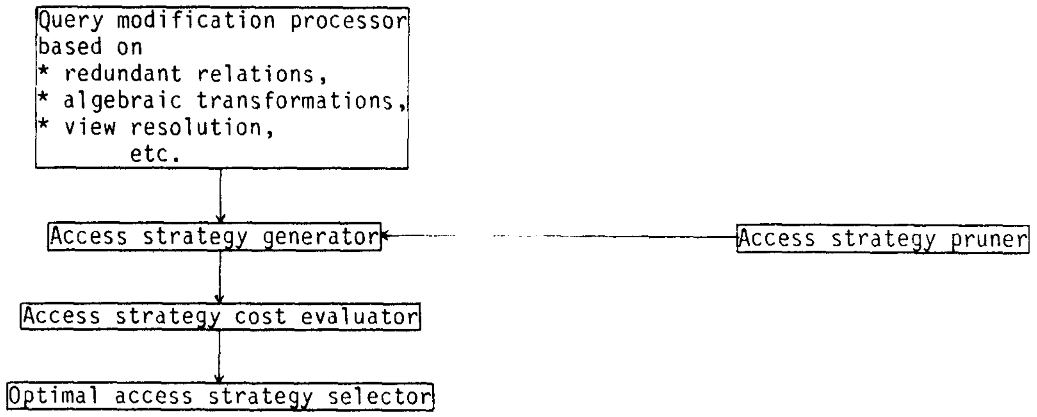

Query modification processor based on
* redundant relations,
* algebraic transformations,
* view resolution,
etc.
Access strategy generator
Access strategy pruner
Access strategy cost evaluator
Optimal access strategy selector

Fig 3 Outline of local optimization

optimal "how representation". Therefore, (c) is a complementary method that would remedy the disadvantages of the methods (a) and (b). An outline of this optimization mechanism is in Figure 3.

(2) Global optimization of query processing

Global optimization is for multiple database commands in a transaction or multiple transactions. In this type of optimization, database commands are clustered and scheduled for execution by estimating the overlapping degrees between query results. In some cases, new database commands are generated which provide query results covering necessary information for executing more than two database commands. The aim of this optimization is the improvement of DBMS performance by reducing overhead of loading database data from secondary storages. The mechanism for this optimization is described in Section 4.

3. PROPOSAL FOR A LOCAL OPTIMIZATION METHOD

This section describes the details of a new optimization mechanism. It includes a description of a new method, called the cascade method, for an efficient combination of the cost evaluation and heuristic rule methods, accurate tuple selectivity estimation, and statistical information management.

3.1 Evaluation orders of predicates in a retrieval condition

(1) General representations of a retrieval condition

A retrieval condition is a boolean expression, BE , composed of a simple predicate P_i (i=1, 2,..., n), i.e., database operation (selection, restriction, join, set inclusion operation, set membership operation, etc.), using logical operator AND($\wedge$), OR($\vee$), NOT($\neg$) and parentheses, () . Thus, the retrieval condition can be represented with either form as described below:

(a) Disjunctive Normal Form : BED

$$BED = \bigvee_{i=1}^{n} (\bigwedge_{j=1}^{m_i} P_j^i) \quad .$$

(b) Conjunctive Normal Form : BEC

$$BEC = \bigwedge_{i=1}^{n} (\bigvee_{j=1}^{m_i} P_j^i) \quad .$$

where, logical NOTs are eliminated from a BED or a BEC according to the De Morgan's theorem.

(2) Processing cost and evaluation orders of predicates

We discuss evaluation orders of predicates in a retrieval condition and the processing cost of the condition formed with a BED or BEC .

(A) Definitions

s_j^i and $t_j^i(\underline{PM})$ relating to a P_j^i are defined as follows:

(a) s_j^i : ratio of the data that a predicate P_j^i is true, i.e., tuple selectivity of a predicate P_j^i.

(b) $t_j^i(\underline{PM})$: processing cost for a unit datum to get data satisfying a predicate P_j^i, with a data processing mechanism $\underline{PM}$ which the DBMS can use. This cost includes the preprocessing cost needed for application of the $\underline{PM}$.

(B) A processing cost

Using s_j^i and $t_j^i(\underline{PM})$, processing costs of both conjunction and disjunction can be represented as follows:

(a) $\bigwedge_{j=1}^{m_i} P_j^i$ (conjunction)' cost:

$$T_c^i((\underline{PM}_1)^{n_{i1}} \,\&\, (\underline{PM}_2)^{n_{i2}} \,\&\, \dots \,\&\, (\underline{PM}_{m_i})^{n_{im_i}})$$
$$= \sum_{j=1}^{m_i} (t_j^i(\underline{PM}_j)^{n_{ij}} * \prod_{l=1}^{j-1} s_l^i) \quad .$$

(b) $\bigvee_{j=1}^{m_i} P_j^i$ (disjunction)' cost:

$$T_d^i((\underline{PM}_1)^{n_{i1}} \,\&\, (\underline{PM}_2)^{n_{i2}} \,\&\, \dots \,\&\, (\underline{PM}_{m_i})^{n_{im_i}})$$
$$= \sum_{j=1}^{m_i} (t_j^i(\underline{PM}_j)^{n_{ij}} * \prod_{l=1}^{j-1} \bar{s}_l^i) \quad .$$

where, $\bar{s}_l^i = 1 - s_l^i$.

$(\underline{PM}_j)^{n_{ij}}$ means to select one $\underline{PM}$ from among n_{ij} $\underline{PM}$s for the P_j^i. $((\underline{PM}_1)^{n_{i1}} \,\&\, (\underline{PM}_2)^{n_{i2}} \,\&\, \dots \,\&\, (\underline{PM}_{m_i})^{n_{im_i}})$ means one of the combinations of $\underline{PM}$s for the condition formed by (a) or (b).

Now, if the condition formed by (a) or (b) is evaluated with the order P_1^i, P_2^i, ..., $P_{m_i}^i$, then there are more than $\prod_{j=1}^{m_i} n_{ij}$ "how representation"s for each condition described above, considering $\underline{PM}$s for each predicate P_j^i. Considering the ordering of each predicate P_j^i, there are more than $m_i! * \prod_{j=1}^{m_i} n_{ij}$ "how representation"s. Therefore, in order to determine the optimal "how representation", a large number of "how representation"s have to be generated for the specified condition and computed for their costs and then selected, which would give the minimal processing cost. Similarly, the processing cost of a BED or a BEC is described as follows:

(c) BED's cost:

$$T(BED) = \sum_{i=1}^{N} (T_c^i((\underline{PM}_1)^{n_{i1}} \,\&\, (\underline{PM}_2)^{n_{i2}} \,\&$$

$$\ldots \ \& \ (\underline{PM}_{m_i})^{n_{i m_i}}) * \prod_{q=1}^{i-1} \bar{s}_q{}^c)$$

$$= \sum_{i=1}^{N}((\sum_{j=1}^{m_i} (t_j{}^i(\underline{PM}_j)^{n_{i j}} * \prod_{l=1}^{j-1} s_l{}^i)) * \prod_{q=1}^{i-1} \bar{s}_q{}^c).$$

(d) BEC's cost:

$$T(BEC) = \sum_{i=1}^{N}(T_d{}^i((\underline{PM}_1)^{n_{i1}} \ \& \ (\underline{PM}_2)^{n_{i2}} \ \& \ \ldots \ \& \ (\underline{PM}_{m_i})^{n_{i m_i}})\prod_{q=1}^{i-1} s_q{}^d)$$

$$= \sum_{i=1}^{N}((\sum_{j=1}^{m_i} (t_j{}^i(\underline{PM}_j)^{n_{i j}} * \prod_{l=1}^{j-1} \bar{s}_l{}^i)) * \prod_{q=1}^{i-1} s_q{}^d).$$

In these cases, considering the order of each conjunction (or disjunction), there are more than $N! * \prod_{i=1}^{N} (m_i!) * \prod_{j=1}^{m_i} n_{i_j} * \ldots * \prod_{j_n=1}^{m_N} n_{N_{j_N}}$ "how representation"s.

(C) Evaluation orders of predicates

Here, relational expressions are shown that give the evaluation order for each predicate, conjunction or disjunction in the retrieval condition such that the processing cost becomes minimum:

(a) <u>a relational expression for a BED</u>

When a BED is to be processed with such an order for each conjunction that the relational expression described below is satisfied, the BED's cost becomes minimum:

$$\sum_{j=1}^{m_1}(t_j{}^1(\underline{PM}_j)^{m_{1j}} * \prod_{l=1}^{j-1} s_l{}^1)/s_1{}^c \leq \cdots \leq \sum_{j=1}^{m_N}(t_j{}^N(\underline{PM}_j)^{m_{Nj}} * \prod_{l=1}^{j-1} s_l{}^N)/s_N{}^c.$$

(b) <u>a relational expression for a BEC</u>

When a BEC is to be processed with such an order for each disjunction that the relational expression described below is satisfied, the BEC's cost becomes minimum:

$$\sum_{j=1}^{m_1}(t_j{}^1(\underline{PM}_j)^{m_{1j}} * \prod_{l=1}^{j-1} \bar{s}_l{}^1)/\bar{s}_1{}^d \leq \cdots \leq \sum_{j=1}^{m_N}(t_j{}^N(\underline{PM}_j)^{m_{Nj}} * \prod_{l=1}^{j-1} \bar{s}_l{}^N)/\bar{s}_N{}^d.$$

Since these relational expressions are obtained simply, we will omit proof. Relational expressions for both conjunctions and disjunctions which construct BED's or BEC's are similarly obtained. They are described as follows:

(c) <u>a relational expression for a conjunction</u>

When a conjunction is to be processed with such an order for each predicate that a relational expression described below is satisfied, the conjunction's cost becomes minimum:

$$t_1{}^i(\underline{PM}_1)^{n_{i1}}/\bar{s}_1{}^i \leq t_2{}^i(\underline{PM}_2)^{n_{i2}}/\bar{s}_2{}^i \leq \cdots \leq t_{m_i}{}^i(\underline{PM}_{m_i})^{n_{i m_i}}/\bar{s}_{m_i}{}^i .$$

(d) <u>a relational expression for a disjunction</u>

When a disjunction is to be processed with such an order for each predicate that a relational expression described below is satisfied, disjunction's cost becomes minimum:

$$t_1{}^i(\underline{PM}_1)^{n_{i1}}/s_1{}^i \leq t_2{}^i(\underline{PM}_2)^{n_{i2}}/s_2{}^i \leq \cdots \leq t_{m_i}{}^i(\underline{PM}_{m_i})^{n_{i m_i}}/s_{m_i}{}^i .$$

Proceeding on the basis of the above relational expressions and setting up an appropriate processing mechanism considering the results (i.e., size, content's format) of the previous predicate's processing, an efficient "how representation" for a retrieval condition can be determined. Next, an algorithm to determine such an efficient "how representation" is shown.

(3) Algorithm for determining an efficient "how representation" (cascade method)

step 1(generation of parsing tree)

This step generates a parsing tree using syntax and semantics analyzers. At the same time, query transformation is executed through query modification, using view resolution, knowledge-base information, and detection or removal of inconsistency or redundancy in a retrieval condition, and removal of logical NOT operators by means of De Morgan's theorem.

step 2(generation of retrieval-expression tree)

Based on the above parsing tree, this step generates a retrieval-expression tree, RET as shown Figure 4. This time, detection of common predicates or common predicate-expressions and bookkeeping of the related information for such predicates are also executed. A RET is composed of elements described below:

* <u>a root node</u>:

This is a starting point of a RET and has a pointer to a relation node.

* <u>a relation node</u>:

This describes a relation-id related to a RET and has pointers to other different relation and to a conjunction (or disjunction) node related to the relation.

* <u>a conjunction (or disjunction) node</u>:

This describes a conjunction (or a disjunction) related to the relation. This node has three pointers, i.e., a pointer to a next different conjunction (or disjunction) applied to the same relation, a pointer to other relation's sub-conjunction (or sub-disjunction) included within a same conjunction (or disjunction) (this pointer is called a conjunction (or disjunction) group chain), and a pointer to a simple condition node.

* a simple condition node:

This describes a simple predicate(e.g., selection, restriction, join, and others concerning set operations) applied to a relation. This node has three kinds of pointers, i.e., a pointer to a different simple predicate next in order, a pointer to the same cross operation predicate in another relation (this pointer is called a cross operation condition chain), and pointers to the same kind of predicates , which may concern other relations, (these pointers are for random evaluation of the same kind of predicates and so are called a random evaluation condition chain).

step 3(computation of simple predicate's tuple selectivity)

This step computes tuple selectivity of each simple predicate $P_j{}^i$ using statistical information, relationships among predicates (this concept is explained in Section 3.2), and expressions for selectivity estimation. At this time, the RET is updated based on detection and reduction of 'always-true' or 'always-false' predicates or predicate-expressions. This step is repeatedly executed for each predicate in each conjunction (or disjunction).

step 4(computation of conjunction (or disjunction)'s selectivity)

Using results of step 3, this step computes tuple selectivity for each conjunction C_i (or disjunction D_i). Similar to step 3, detection and reduction of 'always-true' or 'always-false' predicate-expressions, i.e.,C_i (or D_i) is executed and the RET may be updated if necessary. This step is repeatedly executed for every conjunction (or disjunction).

step 5(setting of minimum cost processing mechanism for each predicate)

This step computes the processing cost for each predicate $P_j{}^i$ corresponding to each processing mechanism $\underline{PM}_j$, which can be used to execute the predicate; it then determines an optimal processing mechanism $\underline{PM}_j$ which will require minimum processing cost among processing machanisms for the predicate. At this time, $t_j{}^i(\underline{PM}_j)$, which is the processing cost for an unit datum, is derived from this minimum processing cost. On setting up this processing mechanism, the system uses rules based on characteristic information, e.g., processing data size, applicable access path, and tuple selectivity. With these rules, the system can prune processing mechanisms which need not be considered for the predicate. This step is repeatedly executed for each predicate in each conjunction (or disjunction).

step 6(determination of evaluation order for each predicate in C_i (or D))

Using results of step 5, this step computes $t_j{}^i(\underline{PM}_j)/s_j{}^i$(or $t_j{}^i(\underline{PM}_j)/\bar{s}_j{}^i$), and then determines an evaluation order for each predicate in each conjunction (or disjunction), based on the relational expression described above (C). That is to say, at first, the system determines a predicate for evaluation first using results of step 5 and the same relational expression; then the predicate, which should be evaluated next, is determined, considering the result size of the previous predicate, applicable processing mechanisms, applicable access pathes, and relationships of this predicate with other predicates, for the remaining predicates. This step, combined with step 5, is repeatedly executed using the same relational expression like the greedy method until the evaluation order is determined. . In this way, the evaluation order for each predicate in each conjunction (or disjunction) is determined. This step is repeatedly executed for every conjunction (or disjunction). At this time, the RET is reconstructed according to the evaluation order of each predicate, i.e., the pointers described above will be changed.

step 7(computation of conjunction (or disjunction)'s processing cost)

Using the results of both step 5 and 6, this step computes the processing cost for each conjunction (or disjunction), tracing conjunction (or disjunction) group chains if they exist.

step 8(setting of processing order for each conjunction (or disjunction))

Using results of steps 4 and 7, this step computes the following value,

$$\left(\sum_{j=1}^{m_i}\left(t_j{}^i(\underline{PM}_j)*\prod_{l=1}^{j-1} s_l{}^i\right)\right)/s_j{}^c$$

$$\left(\text{or } \left(\sum_{j=1}^{m_i}\left(t_j{}^i(\underline{PM}_j)*\prod_{l=1}^{j-1} \bar{s}_l{}^i\right)\right)/\bar{s}_j{}^d\right)$$

for each conjunction (or disjunction), and then determines an evaluation order for each conjunction (or disjunction), based on the relational expression described above (C), as is similar to steps 5 and 6. At this point, the RET is reconstructed according to the evaluation order for each conjunction (or disjunction). In this way, an efficient "how representation", i.e., access strategy for the retrieval condition expression, is determined. As a result, the RET structure obtained denotes the efficient "how representation". Then the system constructs intermediate language codes and object codes for the retrieval condition expression using this RET structure.

3.2 Estimation of tuple selectivity

In local optimization, the evaluation cost of each predicate and its selectivity are very important factors, as described above in 3.1. Here, at first, problems about the conventional estimation method for tuple selectivity are discussed, and then the proposed method for solving them is described.

(1)Problems of the conventional selectivity estimation method of a predicate

(A) Uniform distribution assumption

One of the problems in conventional estimation methods for tuple selectivity is the assumption that distribution of the attribute's values is uniform.

In general, this distribution is not uniform. In conventional methods, as tuple selectivity is estimated only according to the following values, e.g., the number of unique values, minimum and maximum value of the attribute, the accuracy degree of tuple selectivity estimation is not very high . Therefore, in order to estimate tuple selectivity more correctly, the system needs to use more detailed statistical information.

(B) No consideration of relationship among predicates

And as there are no considerations about relationships among predicates, for example, inclusion(or dependancy) or exclusion relationship, the estimation of tuple selectivity is not correct. Here, insufficiency of the conventional estimation method and its revised estimation method are described (for simplicity, a single relation and two predicates, P_1, P_2, are used and the notation $s(P_i)$ is used for tuple selectivity of the predicate P_i (i=1 or 2)).

(a) tuple selectivity for logical AND/OR among predicates related to the same attribute :

(*1) case of logical AND, P_1 AND P_2 :
- conventional estimation: $s(P_1)*s(P_2)$.
- revised estimation:
  - 0 (exclusive case).
  - $MIN(s(P_1), s(P_2))$ (inclusive case).
  - $s(P_1)*s(P_2)$ (obscure case).

(*2) case of logical OR, P_1 OR P_2 :
- conventional estimation
  - $s(P_1)+s(P_2)-s(P_1)*s(P_2)$.
- revised estimation:
  - $s(P_1)+s(P_2)$ (exclusive case).
  - $MAX(s(P_1), s(P_2))$ (inclusive case).
  - 1 (other case).

(b) tuple selectivity for logical AND/OR among predicates related to different attributes :

(*1) case of logical AND, P_1 AND P_2 :
- conventional estimation: $s(P_1)*s(P_2)$.
- revised estimation:
  - 0 (exclusive case).
  - $s(P_1)*s(P_2)$ (obscure case).
  - $MIN(s(P_1), s(P_2))$ (inclusive case).

(*2) case of logical OR, P_1 OR P_2 :
- conventional estimation:
  - $s(P_1)+s(P_2)-s(P_1)*s(P_2)$.
- revised estimation:
  - $s(P_1)+s(P_2)$ (exclusive case).
  - $MAX(s(P_1), s(P_2))$ (inclusive case).
  - $s(P_1)+s(P_2)-s(P_1)*s(P_2)$(other case).

In this way, by considering relationships among the predicates, it is possible to estimate tuple selectivity more correctly.

(2) Estimation methods of tuple selectivity

Based on the above discussion, new estimation methods for tuple selectivity are proposed here. And new notions of tuple selectivity, i.e., logical selectivity and physical selectivity are introduced as follows:

(a) logical selectivity:

This selectivity is derived from data based on logical frequency of the attribute's values, without considering their physical placement. We call this selectivity **L-selectivity.** Conventional selectivity is categorized into this L-selectivity.

(b) physical selectivity:

On the contrary, this selectivity is derived from a consideration of physical placement of the attribute's values: for example, the number of physical pages which contain the attribute's values. We call this selectivity **P-selectivity.** Using this P-selectivity, processing costs of predicates can be estimated more accurately.

(A) A method using frequency distribution of attribute's values

This estimation method is based on frequency distribution management of the attribute's values; it also considers physical placement of the attribute's values. This management will be described briefly later.

(B) A method using relationships among predicates

As described above, using relationships among predicates, it is possible to estimate tuple selectivity of predicates selectivity and cost evaluation.

4. PROPOSAL FOR A GLOBAL OPTIMIZATION METHOD

In this section, a global optimization method is proposed. This optimization is for multiple database commands in a transaction or for multiple transactions. In this optimization, database commands or transactions are clustered and scheduled for execution according to their meanings or query patterns.

4.1 Mechanism of the global optimization method

The proposed global optimization mechanism comprises the following two managers:
(a) transactions clustering manager.
(b) commands clustering manager.
Here, we describe only the construction of the manager (a). The manager (b) is represented by replaceing the word transaction in (a) to command, so we will omit the descriptions.

(1) Transactions clustering manager
This manager comprises the following components:
(A) Transaction analyzer
This analyzer analyzes each transaction entered into the system and manages the following tables:
(a) transaction management table.
This table manages the contents (i.e., source programs) of each transaction.
(b) transaction-relation relationship management table.
This table manages relationships between each transaction and relations which need to be processed. In this table, relationships between transaction and relation's attributes are also maintained.
(B) Transaction clustering processor
This processor executes clustering of transactions using above tables and manages the following table:
(a) transaction-cluster management table.
(C) Transaction-cluster execution scheduler
This scheduler determines an execution-order of each transaction-cluster using above tables and manages the following table:
(a) transaction-cluster execution-order management table.
(D) Transaction-cluster analyzer
This analyzer analyzes each transaction-cluster and obtains relationships among transactions within each transaction-cluster, and also executes estimation of both result sizes and filtering effectiveness. After this processing, the analyzer produces and manages the following table:
(a) transaction effectiveness estimation table.
(E) Transaction execution scheduler
This scheduler determines the execution order for each transaction within a transaction-cluster according to the above estimation table and manages the following table:
(a) transaction execution-order management table.
(F) Transaction modification processor
This processor modifies an original transaction into an efficient form according to the specified execution order and manages the following table:.
(a) modified transaction management table.

The configuration of the global optimization mechanism is illustrated in Figure 5.

4.2 The processing outline of the global optimization method

The processing outline of the global optimization method is as follows:
First of all, transactions entered into the system are maintained in a queue for processing by the system controller. The transaction clustering manager, then if possible, fetches a transaction one by one from the queue and then the transaction is analyzed by the transaction analyzing processor. An this time, this processor produces several tables as described above.
Then, using these tables, clustering processing of transactions is executed by the transaction clustering processor. In this process, transactions which acquire common relations to be processed are grouped in clusters.
In turn, these clusters are given individual execution orders determined by the transaction-cluster execution scheduler according to the entered order of each transaction within a transaction-cluster into the system, the access type (e.g., read, write) of each transaction, and the number of distinct relations to be processed.
Then transaction-clusters are fetched by the transaction-cluster analyzer one by one, based on an execution-order determined in advance and are analyzed in order to obtain relationships among transactions and to estimate result sizes or filtering effectiveness of transactions within each transaction-cluster.
Next, using above results, an execution-order for each transaction within each transaction-cluster is determined by the transaction execution scheduler according to the entered order of each transaction

view. The following are introduced for this purpose:

(a) setting a high threshold for the distance value in order to lighten partition management of attribute's values without exercising an adverse influence on a degree of accuracy for estimation of tuple selectivity or processing cost.

(b) limiting the number of attributes that the system can manage for the sake of reducing the total size of statistical information.

(c) managing statistical information for the sampled data extracted from databases.

(d) not supporting dynamic maintenance of statistical information in order to reduce the performance degradation of the system.

(e) collecting monitored information in other temporary files and then updating statistical information using the above information at the initiation/termination or at the checkpoint/recovery of the system.

5.2 The global optimization mechanism

The following topics are discussed in regard to the proposed mechanism:
- clustering of transactions (or commands).
- query modification of original queries.

(1) Clustering of transactions(or commands)

Clustering of transactions (or commands) based on their meanings or query patterns is effective in query processing. Therefore, it is a very important subject how to determine the number of the transactions (or commands) in a transaction (or command)-cluster .

In order to cluster the transactions (or commands), the system provides the timer to calculate the queueing time of transactions (or commands) and a counter to count up the number of transactions (or commands) entered into the system. Transactions to be clustered are determined according to comparisons with the threshold value of the queueing time and the threshold number of transactions which have been set in advance.

However, it is difficult to determine these thresholds. These values must be determined by considering the running mode, the running load, the database size, and so on, of the system. For example, as for the TSS type jobs, it may prefer to consider the queueing time, and, as for the batch type jobs, it may prefer to consider the number of queueing transactions.

(2) Query modification of original queries

In query modification for global optimization, the system must have a dynamic query (and also temporary relation) definition and a management facility which are likely to consume much computing and disk access time. Query modification is worth the effort if the amount of common data between two queries is large. Therefore, whether to modify clustered commands (or transactions) should be determined by considering filtering factors and query modification overhead.

6. SUMMARY

We have classified query optimization into two categories: local and global optimization, and proposed a new local optimization method which improves both efficiency of selected access strategies and optimization overhead. We have also proposed a global optimization method which executes optimization over multiple queries.

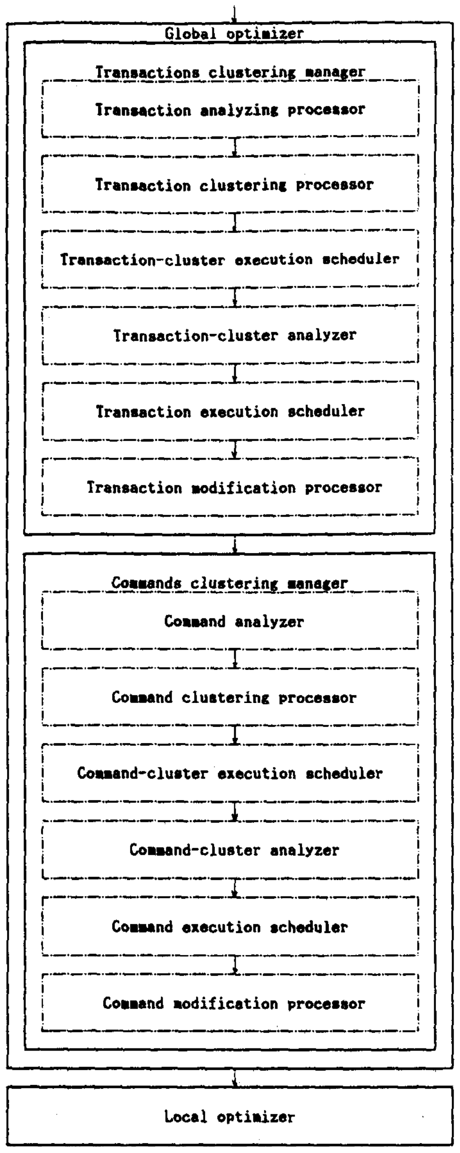
ACKNOWLEDGEMENTS

We would like to thank Dr. Jun Kawasaki and Mr. Kohichiroh Ishihara of Hitachi Systems Development Laboratory for giving us an opportunity to do the research. We also thank our colleagues for their fruitful suggestions and discussions for the content of this paper.

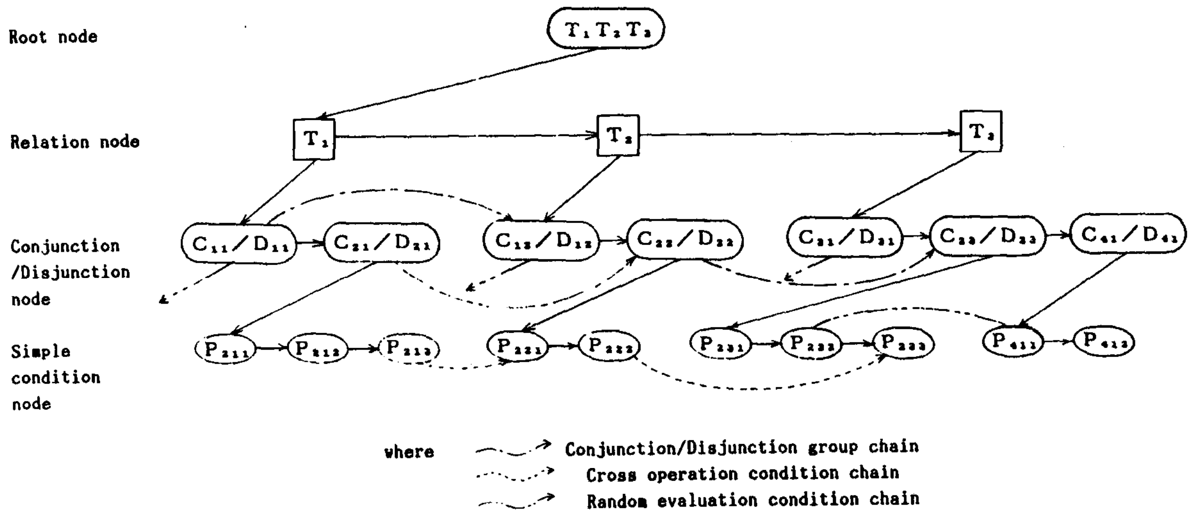
REFERENCES

(1)Astrahan,M.M.,Blasgen,M.W.,Chamberlin,D.D., Eswaran,K.P.,Gray,J.N.,Griffith,P.P.,King,W.F. Lorie,R.A.,Mcjones,P.R.,Mehl,J.W.,Putzolu,G.R. Traiger,I.L.,Wade, and Watson,V., "System R: A Relational Approach to Data Base Management", ACM Trans. Database syst. 1, 2 (June 1976).
(2)Blagen,M.W., and Eswaran,K.P., "Storage and Access in relational databases", IBM Syst. J. 16(1976), 363-377.
(3)Finkelstein,S., "Common expression analysis in database applications", Proc. ACM-SIGMOD Inter. Conf. (Orlando,JUNE. 1982), 235-245.
(4)Hanani,M.Z., "An Optimal Evaluation of Boolean Expressions in an Online Query System", Comm. ACM 20, 5 (May 1977), 344-347.
(5)Hall,P.A.V., "Optimization of single expressions in a relational data base system", IBM J. R. & D. 20, 3(March 1976), 244-257.
(6)Jarke,M. and Koch,J., "Query Optimization in Database Systems", ACM Computing Surveys 16, 2(Jun. 1984), 111-152.
(7)King,J.J., "QUIST: A system for semantic query optimization in relational databases", Proc. 7th Inter. Conf. VLDB (Cannes,Sept. 1981), 510-517.
(8)Makinouchi,A.,Tezuka,M.,Kitakami,H., and

Adachi,S., "The Optimization Strategy for Query Evaluation in RDB/V1", Proc. 7th Inter. Conf. VLDB (Cannes,Sept. 1981), 518-529.
(9)Rosenthal,A.,and Reiner,D., "An architecture for query optimization", Proc. ACM-SIGMOD Conf. (Orland,June 1982), 246-255.
(10)Selinger,P.G.,Astrahan,M.M.,Chamberlin,D.D. Lorie,R.A.,and Price,T.G., "Access path selection in a relational database management system", Proc. ACM-SIGMOD Conf. (Boston,May 1979), 23-34.
(11)Shapiro,G.P., and Connel,C., "Accurate Estimation of the Number of Tuples Satisfying a condition", Proc. ACM-SIGMOD Conf. (Boston,June 1984), 256-296.
(12)Smith,J.M., and Chang,P.Y.T., "Optimizing the performance of a relational algebra database interface", Comm. ACM 18, 10(Oct. 1975), 568-579.
(13)Stonebraker,M., Wong,E., Kreps,P., and Held,G., "The design and implementation of INGRES", ACM Trans. Database Syst. 1, 3(Sept. 1976), 189-222.
(14)Todd,S.J.P., "The Peterlee Relational Test Vehicle - A system overview", IBM Sys. J. 15, 4(Apri. 1976).
(15)Ullman,J.D., "Principles of Database Systems", Computer Science Press, Rockville,Md. (1982).
(16)Yao,S.B., "Optimization of query evaluation algorithms", ACM Trans. Database Syst. 4, 2(June 1979), 133-155.


Global optimizer
Transactions clustering manager
Transaction analyzing processor
Transaction clustering processor
Transaction-cluster execution scheduler
Transaction-cluster analyzer
Transaction execution scheduler
Transaction modification processor
Commands clustering manager
Command analyzer
Command clustering processor
Command-cluster execution scheduler
Command-cluster analyzer
Command execution scheduler
Command modification processor
Local optimizer

F I G. 5 Configuration of the global optimization method



F I G. 4 S t r u c t u r e o f t h e R E T (Retrieval Expression Tree)

TABLE 1 L-(P-)selectivity estimation expressions

predicate type	L-(P-)selectivity estimation expression
attr="val"	L-: $(Cj_1/Dj_1)/\Sigma Cj$ P-: $(Pj_2/Dj_2)/\Sigma Pj$
$attr_1=attr_2$	L-: $\Sigma(MIN(Cj_{11}/Dj_{11},Cj_{12}/Dj_{12}))/\Sigma Cj$ P-: $\Sigma(MIN(Pj_{21}/Dj_{21},Pj_{22}/Dj_{22}))/\Sigma Pj$
attr θ "val" $\theta=\{<,\leqq,\geqq,>\}$	L-: $(((max-"val")/(max-min))*Cj_1/Dj_1+\Sigma Cj_0)/\Sigma Cj$ (for $\geqq,>$) L-: $((("val"-min)/(max-min))*Cj_1/Dj_1+\Sigma Cj_0)/\Sigma Cj$ (for $\leqq,<$) P-: $(((max-"val")/(max-min))*Pj_2/Dj_2+\Sigma Pj_0)/\Sigma Pj$ (for $\geqq,>$) P-: $((("val"-min)/(max-min))*Pj_2/Dj_2+\Sigma Pj_0)/\Sigma Pj$ (for $\leqq,<$)
attr BETWEEN val_1 AND val_2	L-: $(((max_1-"val_1")/(max_1-min_1))*Cj_{21}/Dj_{21}+\Sigma Cj_0$ $+(("val_2"-min_2)/(max_2-min_2))*Cj_{22}/Dj_{22})/\Sigma Cj$ P-: $(((max_1-"val_1")/(max_1-min_1))*Pj_{21}/Dj_{21}+\Sigma Pj_0$ $+(("val_2"-min_2)/(max_2-min_2))*Pj_{22}/Dj_{22})/\Sigma Pj$
attr IN (val-list)	L-: $\Sigma(Cj_{10}/Dj_{10})/\Sigma Cj$ P-: $\Sigma(Pj_{20}/Dj_{20})/\Sigma Pj$
attr IN subquery	L-: $(\overline{Cj_1/Dj_1})*E/\Sigma Cj$ P-: $(\overline{Pj_2/Dj_2})*E/\Sigma Pj$
$pred_1$ OR $pred_2$	$s(pred_1)+s(pred_2)$ (for exclusion) $MAX(s(pred_1),s(pred_2))$ (for inclusion) $s(pred_1)+s(pred_2)-s(pred_1)*s(pred_2)$ (for other case)
$pred_1$ AND $pred_2$	0 (for exclusion) $MIN(s(pred_1),s(pred_2))$ (for inclusion) $s(pred_1)*s(pred_2)$ (for other case)
NOT pred	1-s(pred)

Cj_1	: frequency of a specified attribute's value within a specified partition
Pj_2	: number of pages including a specified attribute's value within a specified partition
Dj_1	: distinct attribute's values within a specified partition
Dj_2	: distinct page number within a specified partition
ΣCj	: total frequency of values in a specified attribute(i.e., total number of tuples)
ΣPj	: total number of pages including values of a specified attribute
ΣCj_0	: total frequency of values within partitions satisfy a specified condition
ΣPj_0	: total number of pages within partitions satisfy a specified condition
E	: expected number of attribute's values satisfy a specified subquery

$(\overline{Cj_1/Dj_1})$,$(\overline{Pj_2/Dj_2})$: average of (Cj_1/Dj_1),(Pj_2/Dj_2) within appropriate partitions