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 multiple 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 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.

![Fig.1 Role of query optimization](image)

As described above, query optimization is essential in a relational DBMS since the DBMS must produce candidate internal access strategies (i.e., "how-representation") 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...
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 of 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 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 PRV(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
(a) Classification of optimization

<table>
<thead>
<tr>
<th>Classification</th>
<th>Object of each optimization</th>
<th>Processing outlines</th>
</tr>
</thead>
</table>
| Global optimization(1) | Transactions (Inter-transaction optimization)                                               | * Clustering of transactions  
* Scheduling of transaction-clusters |
| Global optimization(2) | Commands (Inter-commands optimization)                                                      | * Clustering of commands  
* Scheduling of commands-clusters |
| Local optimization(1)  | A transaction (Intra-transaction optimization)                                              | * Query modification  
* Clustering of predicates  
* Pruning of processing strategy with rules  
* Cost evaluation of processing strategy |
| Local optimization(2)  | A command (Intra-command 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. 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 (\(\lor\)), NOT (\(\neg\)) and parentheses, ( ). Thus, the retrieval condition can be represented with either form as described below:

(a) Disjunctive Normal Form: BED

\[ \text{BED} = \bigvee_{i=1}^{n} \left( \bigwedge_{j=1}^{m} P_{ij} \right) \]

(b) Conjunctive Normal Form: BEC

\[ \text{BEC} = \bigwedge_{i=1}^{n} \left( \bigvee_{j=1}^{m} P_{ij} \right) \]

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

\( s_{ij} \) and \( t_{ij}(PM) \) relating to a \( P_{ij} \) are defined as follows:

(a) \( s_{ij} \) : ratio of the data that a predicate \( P_{ij} \) is true, i.e., tuple selectivity of a predicate \( P_{ij} \).

(b) \( t_{ij}(PM) \) : processing cost for a unit datum to get data satisfying a predicate \( P_{ij} \), with a data processing mechanism PM which the DBMS can use. This cost includes the preprocessing cost needed for application of the PM.

(b) A processing cost

Using \( s_{ij} \) and \( t_{ij}(PM) \), processing costs of both conjunction and disjunction can be represented as follows:

(a) \( \bigwedge_{i=1}^{n} P_{ij} \) (conjunction) cost:

\[ T_{C_{ij}}(\{P_{1}\}_{i=1}^{n}, \{P_{2}\}_{i=1}^{n} \ldots \{P_{m}\}_{i=1}^{n}) = \sum_{j=1}^{m} \left( t_{ij}(PM_{j}) \cdot s_{ij} \right) \]

(b) \( \bigvee_{i=1}^{n} P_{ij} \) (disjunction) cost:

\[ T_{D_{ij}}(\{P_{1}\}_{i=1}^{n}, \{P_{2}\}_{i=1}^{n} \ldots \{P_{m}\}_{i=1}^{n}) = \sum_{j=1}^{m} \left( t_{ij}(PM_{j}) \cdot s_{ij} \right) \]

where, \( s_{1} = 1 - s_{ij} \).

(\( P_{ij} \)) \( \{PM\} \) means to select one PM from among \( \{PM\} \) for the \( P_{ij} \), \( \{P_{1}\}_{i=1}^{n}, \{P_{2}\}_{i=1}^{n} \ldots \{P_{m}\}_{i=1}^{n} \)

where, \( \{PM\} \) means one of the combinations of PMs for the condition formed by (a) or (b).

Now, if the condition formed by (a) or (b) is evaluated with the order \( P_{1}, P_{2}, \ldots, P_{m} \), then there are more than \( m_{1} \cdot m_{2} \cdot \ldots \cdot m_{i} \) "how representation"s for each \( j \) condition described above, considering PMs for each predicate \( P_{ij} \). Considering the ordering of each predicate \( P_{ij} \), there are more than \( m_{1} \cdot m_{2} \cdot \ldots \cdot m_{i} \) "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:

\[ T_{BED} = \sum_{i=1}^{N} \left( T_{C_{ij}}(\{P_{1}\}_{i=1}^{n}, \{P_{2}\}_{i=1}^{n} \ldots \{P_{m}\}_{i=1}^{n}) \right) \]
... & \left(\sum_{i=1}^{N} m \right)_{i=1}^{j=1} \left(\sum_{j=1}^{q} c_{j} \right)

(d) BEC's cost:
\[
T(BEC) = \sum_{i=1}^{N} \left( t_{1}^{i}(PM_{1}) \right)^{n_{1}^{i}} \leq \sum_{i=1}^{N} \left( t_{2}^{i}(PM_{2}) \right)^{n_{2}^{i}} \leq \cdots \leq \sum_{i=1}^{N} \left( t_{m}^{i}(PM_{m}) \right)^{n_{m}^{i}}.
\]

(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) a relational expression for a BED
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:
\[
\sum_{j=1}^{N} \left( t_{j}^{i}(PM_{j}) \right)^{m_{j}^{i}} s_{j}^{i} \leq \cdots \leq \sum_{j=1}^{N} \left( t_{j}^{i}(PM_{j}) \right)^{m_{j}^{i}} s_{j}^{i}.
\]

(b) a relational expression for a BEC
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:
\[
\sum_{j=1}^{N} \left( t_{j}^{i}(PM_{j}) \right)^{m_{j}^{i}} s_{j}^{i} \leq \cdots \leq \sum_{j=1}^{N} \left( t_{j}^{i}(PM_{j}) \right)^{m_{j}^{i}} s_{j}^{i}.
\]

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) a relational expression for a conjunction
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:
\[
t_{1}^{i}(PM_{1})^{n_{1}^{i}} s_{1}^{i} \leq t_{2}^{i}(PM_{2})^{n_{2}^{i}} s_{2}^{i} \leq \cdots \leq t_{m}^{i}(PM_{m})^{n_{m}^{i}} s_{m}^{i}.
\]

(d) a relational expression for a disjunction
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:
\[
t_{1}^{i}(PM_{1})^{n_{1}^{i}} s_{1}^{i} \leq t_{2}^{i}(PM_{2})^{n_{2}^{i}} s_{2}^{i} \leq \cdots \leq t_{m}^{i}(PM_{m})^{n_{m}^{i}} s_{m}^{i}.
\]

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 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:

* a root node: This is a starting point of a RET and has a pointer to a relation node.
* a relation node: 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.
* a conjunction (or disjunction) node: 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 $P_j$ 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 $P_j$ pertaining to each processing mechanism $PM_j$, which can be used to execute the predicate; it then determines an optimal processing mechanism $PM_j$ which will require minimum processing cost among processing mechanisms for the predicate. At this time, $t_j(PM_j)$, 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 $D_i$))
Using results of step 5, this step computes $t_j(PM_j)/s_j$ (or $t_j(PM_j)/s_j^D$), 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 paths, 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, 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.

(a) 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: $s(P_1) \times s(P_2)$.
- revised estimation:
  - 0 (exclusive case).
  - $s(P_1) \times s(P_2)$ (obscure case).
  - MIN($s(P_1)$, $s(P_2)$) (inclusive case).

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

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

(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: $s(P_1) \times s(P_2)$.
- revised estimation:
  - 0 (exclusive case).
  - $s(P_1) \times s(P_2)$ (obscure case).
  - MIN($s(P_1)$, $s(P_2)$) (inclusive case).

(*2) case of logical OR, $P_1$ OR $P_2$:
- conventional estimation:
  - $s(P_1) + s(P_2) - s(P_1) \times s(P_2)$.
- revised estimation:
  - $s(P_1) + s(P_2)$ (exclusive case).
  - $s(P_1) + s(P_2)$ (inclusive case).
  - $s(P_1) + s(P_2) - s(P_1) \times s(P_2)$ (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 replacing 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 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. At 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
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. Kohihiroh 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


(8) Makinouchi, A., Tezuka, M., Kitakami, H., and


---

**FIG. 4 Structure of the RET (Retrieval Expression Tree)**

- **Root node**: $T_1 \cdot T_2 \cdot T_3$
- **Relation node**: $T_1$
- **Conjunction/Disjunction node**: $C_{11}/D_{11} \cdot C_{21}/D_{21} \cdot C_{31}/D_{31} \cdot C_{41}/D_{41}$
- **Simple condition node**: $P_{11} \cdot P_{12} \cdot P_{13}$

---

**FIG. 5 Configuration of the global optimization method**

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

---

**FIG. 5 Configuration of the global optimization method**

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

---

**FIG. 5 Configuration of the global optimization method**

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

---

**FIG. 5 Configuration of the global optimization method**

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

---

**FIG. 5 Configuration of the global optimization method**

- **Local optimizer**
- **Command clustering manager**
  - Command analyzer
  - Command clustering processor
  - Command-cluster execution scheduler
  - Command-cluster analyzer
  - Command execution scheduler
  - Command modification processor
- **Global optimizer**
  - Transaction clustering manager
    - Transaction analyzing processor
    - Transaction clustering processor
    - Transaction-cluster execution scheduler
    - Transaction-cluster analyzer
    - Transaction execution scheduler
    - Transaction modification processor
TABLE 1  L-(P-)selectivity estimation expressions

<table>
<thead>
<tr>
<th>Predicate type</th>
<th>L-(P-)selectivity estimation expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr=&quot;val&quot;</td>
<td>L-: ( \frac{C_j}{D_j} ) / \Sigma C_j</td>
</tr>
<tr>
<td></td>
<td>P-: ( \frac{P_j}{D_j} ) / \Sigma P_j</td>
</tr>
<tr>
<td>attr = attr_a</td>
<td>L-: ( \Sigma (\frac{\text{MIN}(C_{j1},D_{j1},C_{j2},D_{j2})}{C_j}) / \Sigma C_j )</td>
</tr>
<tr>
<td></td>
<td>P-: ( \Sigma (\frac{\text{MIN}(P_{j1},D_{j1},P_{j2},D_{j2})}{P_j}) / \Sigma P_j )</td>
</tr>
</tbody>
</table>
| attr \( \theta \) "val" <,\leq,\geq,\>
|               | \( \Sigma (\frac{(\text{MAX} - \text{MIN})}{(\text{MAX} - \text{MIN}) \times D_j + \Sigma C_j}) / \Sigma C_j \) (for \( \geq, \geq \)) |
|               | \( \Sigma (\frac{(\text{MAX} - \text{MIN})}{(\text{MAX} - \text{MIN}) \times D_j + \Sigma P_j}) / \Sigma P_j \) (for \( \leq, \leq \)) |
| attr BETWEEN val_1 AND val_2 | \( \Sigma (\frac{(\text{MAX} - \text{MIN})}{(\text{MAX} - \text{MIN}) \times D_j + \Sigma C_j}) + (\frac{(\text{MAX} - \text{MIN})}{(\text{MAX} - \text{MIN}) \times D_j + \Sigma P_j}) / \Sigma P_j \) |
| attr IN (val-list) | L-: \( \Sigma (\frac{C_{j1}}{D_{j1}}) / \Sigma C_j \) |
|               | P-: \( \Sigma (\frac{P_{j1}}{D_{j1}}) / \Sigma P_j \) |
| attr IN subquery | L-: \( \frac{C_{j1}}{D_{j1}} \) \* E / \Sigma C_j |
|               | P-: \( \frac{P_{j1}}{D_{j1}} \) \* E / \Sigma P_j |
| 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) |

Ci_j : frequency of a specified attribute's value within a specified partition  
Pj_j : number of pages including a specified attribute's value within a specified partition  
Dj_j : distinct attribute's values within a specified partition  
Dj_j : distinct page number within a specified partition  
\( \Sigma C_j \) : total frequency of values in a specified attribute (i.e., total number of tuples)  
\( \Sigma P_j \) : total number of pages including values of a specified attribute  
\( \Sigma C_j \) : total frequency of values within partitions satisfy a specified condition  
\( \Sigma P_j \) : total number of pages within partitions satisfy a specified condition  
E : expected number of attribute's values satisfy a specified subquery  
\( (C_{j1}/D_{j1}),(P_{j1}/D_{j1}) \) : average of \( (C_{j1}/D_{j1}),(P_{j1}/D_{j1}) \) within appropriate partitions