Index Selection in Relational Databases

Sunil Choenni* Henk Blanken* Thiel Chang**

*Department of Computer Science, University of Twente
P.O. Box 217, NL–7500 AE Enschede, The Netherlands

**Department of Research and Development, G.A.K.
P.O. Box 8300, NL–1005 CA Amsterdam, The Netherlands

Abstract

Intending to develop a tool which aims to support the physical design of relational databases can not be done without considering the problem of index selection. Generally the problem is split into a primary and secondary index selection problem and the selection is done per table. Whereas much attention has been paid on the selection of secondary indices relatively less is known about the selection of a primary index and the relation between them. These are exactly the topics of this paper.

1 Introduction

At the University of Twente in cooperation with the G.A.K. a tool is being developed which aims to support the physical design of relational databases [2].

A problem of considerable interest in the physical design of databases is the selection of a good set of indices. Indices can be considered as auxiliary files that allow to retrieve tuples satisfying certain selection predicates without having to examine the whole relation. On the other hand, updating the database causes an index to be updated to remain consistent with the new database state. So, an index speeds up retrieval and slows down maintenance.

In general two types of indices can be distinguished: primary and secondary indices. In the case of a primary index, the tuples in the relation are ordered on the indexed attribute. This is not the case for a secondary index.

The aim of index selection is to find an optimal or near-optimal set of indices consisting of at most one primary index and zero or more secondary indices. Because the choice of a primary index influences the choice of the secondary index set and conversely, solving the problem in a naive way requires for r relations an exploration of $\prod_{i=1}^{r} n_i 2^{n_i-1} + 2^{n_i}$ sets, in which n_i is the number of attributes in the i-th relation.

As far as we know two strategies or combinations of these strategies are frequently followed to reduce the complexity of the index selection problem. First, the selection of indices are determined per relation. The eventually solution is achieved by the union of the solutions of each relation [3, 6].

Second, selection of indices is split into a primary and a secondary index selection problem. In some papers it is assumed that the primary index has been chosen already and they pay only attention to the problem of secondary index selection [1, 3, 6]. These approaches ignore the cost due to the existence of a primary index. In other papers [5, 11] candidate primary indices are determined on basis of heuristics. Then, for each candidate primary index which is considered as given the determination of the set of secondary indices takes place. Finally the best index set is chosen.

While the first strategy is analysed extensively and has been theoretical founded in [10] such an analysis is missing for the remaining strategy. The second strategy assumes a certain order in designing an index set namely first the primary index is selected and then the secondary index set is selected. On the first sight an alternative for the second strategy is to select first the secondary index set and then the primary index. In which order primary and secondary index selection should be performed is one of the topic of this paper. As far as we know such a theoretical study is still missing.

Through the years several more or less advanced cost models have been developed on basis of which the selection of secondary indices took place, see among others [1, 6, 11]. For the selection of a primary index generally some heuristics are applied [5, 9]. The evaluation of some of these heuristics is another topic of this paper.

2 Primary and secondary indices.

This section is devoted to the relation between a primary index and secondary indices. Indices are supposed to be organized often as B^+-trees. Each node in the tree coincides with a page. The leaf level consists of {key,TID-list} pairs for every unique value of the indexed attribute(s). Figure 1(a) represents a primary index on the column *name* of a relation R(*name*, *age*, *residence*, *bloodgroup*) and figure 1(b) a secondary index on the column *bloodgroup* of R. In general the processing of a query roughly consists of two steps; first the number of tuples which satisfies possibly the WHERE clause of a query is determined; then these tuples are retrieved [3]. Since a primary index may be considered as a special kind of a secondary index the optimizer may treat a primary index and secondary indices as same in processing the first step. In the second step it may use the ordening property

0-8186-4212-2/93 $03.00 © 1993 IEEE

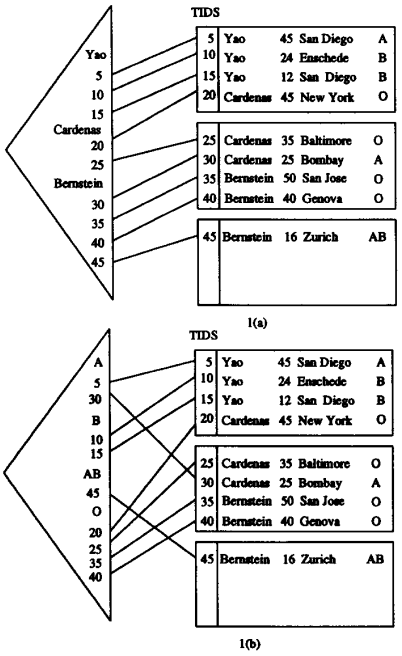

Figure 1: (a) Represents a primary index on attribute *name* of relation R and (b) represents a secondary index on attribute *bloodgroup* of R

of the primary index if at least both types of indices may be used. The following example illustrates this.

Example 1 Let $P(name, address, education)$ be a relation consisting of 4000 tuples. The storage configuration of P consists of a secondary index set A = {education} and a primary index on *name*. The selectivity factors[1] (sf) of the indexed attributes are $sf_{name} = \frac{1}{30}$, $sf_{education} = \frac{1}{30}$ and a page contains 20 tuples (pl). Let us consider the following query w:
SELECT *address* FROM P
WHERE *name* = 'xxxx' AND *education* ='zzzz'
Applying below mentioned formula[2] to estimate the number of tuples (nt) which satisfies the WHERE clause of query w yields $nt = 5$.

$$nt_w(\{A \cup name\}) = \lceil n_P * \prod_{\alpha_j \in \{A \cup name\}} sf_{\alpha_j} * d_w(\alpha_j) \rceil \quad (1)$$

in which n_P is the cardinality of P and $d_w(\alpha_j) = 1$ if α_j appears in the WHERE clause otherwise $d_w(\alpha_j) = \frac{1}{sf_{\alpha_j}}$.

The number of pages accesses (NPA) required to fetch nt tuples from p pages is computed by:

$$NPA(nt, p) = \begin{cases} nt & \text{if } nt \leq \frac{p}{2} \\ \lceil \frac{nt+p}{3} \rceil & \text{if } \frac{p}{2} < nt \leq 2p \\ p & \text{if } 2p < nt \end{cases}$$

The order property of the primary index implies that the tuples satisfying the predicate *name* = 'xxxx' are distributed over $\frac{n_P * sf_{name}}{pl} = 7$ pages. Then $NPA(5, 7) = 4$. □

This example illustrates how primary and secondary indices can be used to decrease the processing costs of a query. These indices have to be kept up to date; the costs involved in updating the indices are called the maintenance costs. The update of a primary index is more expensive than the update of a secondary index. The update costs of a secondary index are independent of other existing indices. The update cost of a primary index is dependent of all existing indices since since all secondary indices should be adapted [4].

3 General cost models

We will develop a cost model for index selection to prove the dependency between primary and secondary index selection. For this purpose we make some reasonable assumptions.

We deal with relational databases which are stored on an external paged memory. The frequency of tuple insertions and tuple deletions is such that the total number of tuples of each relation remains constant in two consecutive choices of index set. Furthermore, the attributes are mutually independent and the values are uniformly distributed.

For a given value a of an attribute α_j, use of an index produces a list of tuple identifiers (TIDs). These TIDs allow direct retrieval of the stored tuples possessing the value a for attribute α_j.

The number of page accesses will be taken as cost factor and pages contain only tuples of one relation.

Before going into details we introduce a list of symbols in Table 1.

Load on a relation
The following 4 operations (each divided in several steps) are distinguished on a database.

- Query:
  1. Select the relevant tuples.
  2. Output the relevant tuples.
- Update:
  1. Select the relevant tuples.
  2. Update the value of the specified attributes.
  3. Place the tuples on the proper location.
  4. Update the relevant indices.
- Deletion:
  1. Select the relevant tuples.
  2. Remove the tuples.
  3. Update the relevant indices.
- Insertion of a tuple:
  1. Select the location to store the tuple.

[1] The selectivity factor is the reciprocal of the number of different values of an attribute.

[2] We assume that the attributes are mutually independent and that the values are uniformly distributed.

$\alpha_1, ..., \alpha_n$ = attributes of a relation
n_R = cardinality of relation R
pl = page length in number of tuples
$np = \frac{n_R}{pl}$ = # pages required to store relation R
W = workload
$w_1 ..., w_n$ = operations of a workload
f_{w_i} = frequency of operation w_i
sf_{α_i} = selectivity factor of attribute α_i
A = unspecified secondary index set
$C_{acc}(A, w)$ = access cost to A for processing w
$C_{ms}(\alpha_j)$ = maintenance cost of a secondary index α_j
$C_{mp}(\alpha_j, A)$ = maintenance cost of a primary index on α_j taking into account the secondary index set A
$C_{mpp}(\alpha_j) = C_{mp}(\alpha_j, \{\})$
$nt_w(A)$ = # tuples to be retrieved in processing w with A
$C_{rets}(nt(A), p)$ = retrieving costs of $nt(A)$ tuples from p pages
$C_{retp}(\alpha_p, w)$ = retrieving costs in processing w using primary index α_p

Table 1: List of used symbols

2. Insert the tuple.
3. Update the relevant indices.

We concentrate on cost factors that influence index selection. Since the output of a query is independent of the index set, this step can be ignored.

The first step of an update, a deletion or an insertion is analogous to the first step of a query. The updating of specified attributes in an update as well as the insertion and deletion of tuples (step 2) take an amount of time which is independent of the existing set of indices.

The third step of an update becomes significant if the primary index is updated. In this case the tuples which are hit by the update have to be moved to another location. This means that the location has to be selected which is comparable with the first step. In the case secondary indices are updated this step can be neglected because the tuples will not be moved.

The last step of the operations (updating indices) depends of course on the selected indices.

Cost function for secondary index selection.

Let us elaborate the first step of a load operation. To determine the selection cost of relevant tuples the following actions have to be performed:

1. Access all indices corresponding to attributes specified in the query; this gives a list TIDs per index.
2. Intersect the lists in order to determine the TIDs of the tuples that satisfy the conjunction of the conditions on the attributes an index exists for.
3. Retrieve the tuples according to the result of the previous action.
4. Discard the tuples not satisfying the condition on the attributes without an index (false drops).

The total selection cost of the relevant tuples is the sum of the costs resulting from actions 1 and 3 since the actions 2 and 4 take place in main memory. The cost of step 1 for an operation w and an index set A is given by $C_{acc}(A, w)$.

The cost of action 3 depends on the # tuples (nt) which has to be retrieved and the # pages from which these tuples have to be retrieved (see example 1). The tuples have to be retrieved from np pages. The # tuples which has to be retrieved depends of course on the operation w and the index set A; this will be denoted as $nt_w(A)$. The cost of retrieving $nt_w(A)$ tuples from np pages will be denoted as $C_{rets}(nt_w(A), np)$.

To determine the cost of step 4 of an operation we assume that I is the insertion frequency, D the deletion frequency on a relation and U_j the α_j value update frequency. Let ic_j be the insertion cost for placing the pair {key,TID} after the last pair that has the same key value and dc_j be the deletion cost of a pair {key,TID} or insertion cost of the same pair with any TID. The maintenance cost of a secondary index α_j ($C_{ms}(\alpha_j)$) becomes:

$$C_{ms}(\alpha_j) = I * ic_j + D * dc_j + U_j * 2 * dc_j$$

For more details w.r.t C_{ms} we refer to [8].

Taking into account the frequencies of each operation w (f_w) of a workload W the cost of processing W with a secondary index set A is given by:

$$CFS(A, W) = \sum_{\alpha_j \in A} C_{ms}(\alpha_j) + \sum_{w \in W} f_w * (C_{acc}(A, w) + C_{rets}(nt_w(A), np))$$

Note, that $C_{rets}(_w(A), np) \leq np$ and $C_{rets}(n_R, np) = np$. For more details w.r.t CFS we refer to [3].

Cost function for primary index selection.

We derive a cost function assuming that each operation of a workload is resolved by the primary index or by a sequential scan. The secondary index set is assumed to be empty. Performing step 1 of a load operation with a primary index α_p entails the actions:

1. Access the primary index.
2. Retrieve tuples satisfying the selection criteria.
3. Discard the tuples not satisfying the condition on the remaining attributes (false drops).

The total cost of selecting the relevant tuples is the sum of the first two actions since action 3 takes place in main memory. The cost of accessing the primary index α_p for resolving an operation w is given by $C_{acc}(\alpha_p, w)$. Assuming that w can be resolved by α_p the # tuples which has to be retrieved in action 2 is $n_R * sf_{\alpha_p}$. The cost of retrieving the tuples is $C_{retp}(\alpha_p, w) = \frac{n_R * sf_{\alpha_p}}{pl}$, in which pl is the page length.

The third step of an update operation becomes relevant if an update is performed in which the primary index (α_p) is involved. This entails cost due to the movement of the tuples to another location. Before

moving the tuples the proper location has to be selected which requires $\frac{n_R * sf_{\alpha_p}}{pl}$ page accesses.

The last step of an update, deletion and insertion requires an adjusting of the primary index. Updating the primary index requires $U_p * 2 * dc_p$ page accesses. The cost involved with insertions and deletions are $I * ic_p$ respectively $D * dc_p$. So, the maintenance cost of a primary index α_p ($C_{mpp}(\alpha_p)$) is:

$C_{mpp}(\alpha_p) = I * ic_p + D * dc_p + U_p(\frac{n_R * sf_{\alpha_p}}{pl} + 2 * dc_p)$

The cost function for primary index selection becomes:

$$CFP(\alpha_p, W) = C_{mpp}(\alpha_p) + \sum_{w \in W} f_w * (C_{acc}(\alpha_p, w) + (\frac{n_R * sf_{\alpha_p}}{pl} * d_w(\alpha_p)))$$

in which $d_w(\alpha_p) = 1$ if α_p is used in processing w otherwise $d_w(\alpha_p) = \frac{1}{sf_{\alpha_p}}$.

Cost function for index selection

Assuming that the processing of the selection part of an operation will be done with minimal cost we derive a cost function for index selection. As a consequence a primary index and secondary indices will be used in combination if both are available (see example 1). Further once a secondary index set and a primary index are available they have to be maintained. An update of a primary index entails an adjusting of all secondary indices. So, the maintenance cost of a primary index with a non-empty index set A is: $C_{mp}(\alpha_p, A) = C_{mpp}(\alpha_p) + I * \sum_{\alpha_j \in A} ic_j + D * \sum_{\alpha_j \in A} dc_j + U_p * \sum_{\alpha_j \in A} 2 * dc_j$

The general cost function for index selection becomes:

$$CF(A, \alpha_p, W) = \sum_{\alpha_j \in A} C_{ms}(\alpha_j) + C_{mp}(\alpha_p, A) + \sum_{w \in W} f_w * (C_{acc}(A \cup \alpha_p, w) + C_{rets}(nt_w(A \cup \alpha_p), \frac{n_R * sf_{\alpha_p}}{pl} * d_w(\alpha_p))) \quad (2)$$

Proposition 1 The problem of primary and secondary index selection are dependent.

Proof Note, $CF(\{\}, \alpha_p, W) = CFP(\alpha_p, W)$ and $CF(A, \{\}, W) = CFS(A, W)$. It can be easily verified that $CF(A, \alpha_p, W) \neq CF(\{\}, \alpha_p, W) + CF(A, \{\}, W)$ □.

The problem of index selection may be formulated now as: *determine α_p and A such that equation (2) is minimal* which is a tough task. Heuristic methods to optimize equation (2) is the topic of the next section.

4 Methods for index selection

We compare two heuristic methods both aiming to optimize the cost function CF. In these methods the order of primary index selection and secondary index selection plays a role. We start with a definition of sub-optimal index configurations. Then the heuristic method will be presented and compared.

Definition 1 Let (A_p, α_p) be an index configuration consisting of a primary index α_p and a secondary index set A_p. The index configuration (α_p, A_p) is sub-optimal w.r.t. α_p iff $\forall i\ CF(A_p, \alpha_p, W) \leq CF(A_i, \alpha_p, W)$

Heuristic method 1

This method assumes that first the primary index is selected and then the secondary indices are selected. For all attributes α_p we evaluate $CF(\{\}, \alpha_p, W)$ and the α_p with the lowest cost is chosen as primary index. Then the primary index is filled in equation (2) and optimized. This boils down on the selection of an optimal set of secondary indices given the primary index. Proposition 2 describes the output of the algorithm. The proof of the proposition is trivial

Proposition 2 Heuristic method 1 will lead to a sub-optimal index configuration.

To find the optimal solution we have to explore a number of sub-optimal solutions which is equal to the number of attributes of the relation. Because the selection of secondary indices is a laborious task this will be generally infeasible. So, it is desired that the sub-optimal solution has to be close to the optimal solution.

Intuitively heuristic method 1 divides the workload in two parts. The choice of the primary index means that a part of the workload is processed efficiently with this index. The selection of secondary indices will be mainly based on the decrease of the processing cost of the remaining part of the workload taking into account the previous part of the workload. In general there will be relatively not much to gain by the addition of a secondary index set to the part of the workload which is solved already by the primary index α_p. The maximal gain which can be achieved per operation is $np * sf_{\alpha_p} - 1$. In practice $np * sf_{\alpha_p}$ will be small. So, operations which are resolved already by a primary index will weakly or will not benefit at all of the addition of a secondary index set. This means that the gain achieved by a primary index is *hardly dependent* of a secondary index set.

The success of this method depends on the choice of the primary index. If the choice will be good the eventually index configuration will be good. Example 2 shows the drawback of choosing the attribute which yields the lowest cost in processing the workload as primary index.

Example 2 Suppose a workload consisting of 5 operations on a relation R. The cost required to process each operation if a primary index is allocated on α_1 respectively α_2 as well as the maintenance cost of each primary index C_{mpp} is given in the following table.

	w_1	w_2	w_3	w_4	w_5	C_{mpp}	total
α_1	5	15	15	15	15	10	75
α_2	5	5	5	15	15	20	65

It is clear that according the table α_2 will be chosen as primary index. But the prospects that the costs will be decrease further by the addition of a secondary index set is worse starting from α_2 than starting from α_1. Starting from α_2 a decrease of the cost can be achieved probably from at most two workload parts while starting from α_1 a cost reduction can be achieved from at most four workload parts. As a consequence it may occur that the sub-optimal pair (A_1, α_1) will be better than (A_2, α_2), in other words the processing cost of the workload with the pair (A_1, α_1) may be lower than with the pair (A_2, α_2). □

Heuristic method 2

We first determine an optimal secondary index set A_{opt} assuming no primary index. For this purpose several algorithms are known [1, 3, 6]. Then equation (2) is optimized further by choosing the attribute as the primary index which causes the greatest decrease of the function. Proposition 3 describes the output of the algorithm. For the proof we refer to [4].

Proposition 3 Heuristic method 2 will not necessarily lead to a sub-optimal solution.

To find the sub-optimal solution w.r.t. α_{opt} we have to optimize the function CF again considering the primary index α_{opt}.

For the optimal solution we have to explore a number of sub-optimal solutions which is equal to the number of possible secondary index sets. Because the number of secondary index sets is exponential with the number of attributes of a relation this strategy is infeasible. So, it is desired that the proposed solution has to be close to the optimal solution.

Some objections can be made to this method. In determining the secondary index set some operations may play a bigger role than they would have deserved if there would be a primary index. This is due to the fact that the gain achieved by a secondary index is *strongly dependent* of the choice of the primary index. This will be shown in the following example.

Example 3 $P(name, address, birthdate, education)$ is a relation consisting of 4000 tuples. The selectivity factors of the attributes are: $sf_{name} = \frac{1}{30}$, $sf_{address} = \frac{1}{10}$, $sf_{birth-date} = \frac{1}{40}$, $sf_{education} = \frac{1}{30}$ and $pl = 20$.

Suppose that secondary index selection for a workload W results into $A = \{\text{birth-date}\}$ and consider the following query $z \in W$:
SELECT address FROM P
WHERE $name$ = 'xxxx' AND $birth - date$ = 'yyyy'
Solving the query with A requires at most 100 page accesses (neglecting the access cost to the index) which is better than a sequential scan. So, this query has supported the index set A. Suppose that optimizing the function CF (given A) results into a primary index on *name*. Solving the query with the primary index combined with the secondary index set requires at most 4 page accesses[3]. So, z has supported the choice of the primary index on *name*. Solving the query which only the primary index requires at most 7 page accesses. So, the secondary index set reduces the cost actually with 3 or 4 page accesses if the primary index is available while it provides a cost reduction of 100 page accesses if the primary index lacks. Without the primary index z will support the secondary index set stronger than in the case the primary index exists. □

[3]To estimate the # tuples satisfying z we use (1).

Comparing the heuristic methods

Heuristic method 1 leads to a sub-optimal solution which is not the case for heuristic method 2. In principle we may obtain a sub-optimal solution with heuristic method 2 but this entails a considerable additional complexity. Moreover, there is no guarantee that the achieved sub-optimal solution will be better than the sub-optimal solution achieved by heuristic method 1.

Another observation is that a secondary index set hardly influences the gain achieved by a primary index while the primary index strongly influences the gain which may be achieved by a secondary index set. This is a justification to select first the primary index and then the secondary index set because once a primary index is chosen the selection of secondary indices can be done independently for the part of the workload which is not resolved by the primary index.

On basis of these observations we can conclude that the first heuristic method is better. However, a good choice of a primary index is crucial as has been shown in example 2.

5 Primary index selection heuristics

Good heuristics for determining candidate indices should take the following into account:

- Retrieval costs of the total workload using the candidate index.
- Maintenance costs of the candidate index.
- Prospects for possibly further decrease of the costs in processing the workload.

Finding heuristics which take the last item into account is hard. We try to incorporate this item in existing heuristics knowing this is a hard task. First, we evaluate some well-known heuristicis in the view of performance.

One well-known heuristic is to choose the logical key of a relation as primary index. Because this attribute is unique it has a great selectivity. Furthermore, logical keys are more or less static. This may be profitable if the operations in the workload make use of the attribute. However, the heuristic does not take the workload into account which may lead to unpredictable results. Another heuristic is used in the first heuristic method of the previous section: choose the attribute which causes the lowest cost in processing the workload as primary index. This heuristic takes the maintenance costs as well as the gain achieved due to the decrease of the retrieval costs into account. We attempt to extend this heuristic such that it is able

to take rather the prospects for possibly further reduction of the processing cost of the workload into account.

Let α_p be a candidate primary index and $W_{\alpha_p} = \{w_1, w_2, ..., w_n\}$ the operations of the workload which can not be resolved by α_p. For each $w_i \in W_{\alpha_p}$ we determine the secondary index set (A_i) which yields the maximal gain g_i. This information will be stored as the triple (w_i, A_i, g_i). In practice database administrators can produce these secondary index sets quickly for single operations on basis of heuristics [7, 9]. Note, $g_i = np - (C_{acc}(A_i, w_i) + C_{rets}(nt_{w_i}(A_i, np)) + C_{ms}(A_i))$ and $\sum_{i=1}^{n} g_i$ is an pessimistic estimation for the further decrease of the costs. We apply the following technique to make a choice for a primary index.

We take the union of all the A_i's for which holds that $g_i > 0$ resulting in a set A_u. Then we process the original workload with the index configuration (A_u, α_p). This has the following advantages. First the maintenance cost of indices which are counted more than once can be filtered out. Suppose we have for example the triples $(w_i, \{a, b\}, g_i)$ and $(w_2, \{b, c\}, g_j)$ in which $g_i,\ g_j > 0$. In determining g_i as well as g_j the maintenance cost of allocating an index on attribute b is taken into account while it is sufficient to take this cost only once into account.

The second advantage is that triples for which hold $g_i = 0$ may contribute still to the decrease of the processing cost. Suppose we have a triple $(w_k, \{\}, 0)$ and that processing w_k with the best index set would result into $(w_k, \{b, c\}, -4)$. But suppose that $A_u \supseteq \{b, c\}$ then g_k will become positive and will contribute to the decrease of the processing cost.

For the choice of a primary index we determine for each attribute α_i the index configuration pair (A_u, α_i). Then we process the original workload with the pair (A_u, α_i) resulting into an estimation for the processing cost EPC. The attribute which yields the lowest cost for EPC will be chosen as primary index.

This may be a way to take possibly prospects for further decrease of the processing cost w.r.t. a workload roughly into account.

6 Conclusions

In developing a tool which aims to support the design of physical databases the problem of index selection is of considerable interest. Solving the problem in an analytical way is a tough task. Therefore, we adopt the idea to split the problem into a primary and secondary index selection problem. Because these problems are mutually dependent the question raises in which order primary and secondary index selection should be performed. We explored this question and conclude that primary index selection should precede secondary index selection. The motivation is that secondary index selection hardly influence the gain achieved by a primary index. So, we can be sure that the contribution to the decrease of the processing cost due to a primary index will be about the same independent of what the secondary index set will be. This is not the case starting from a secondary index set. Another advantage to precede primary index selection by secondary index selection is that it always will lead to a sub-optimal solution. This is not the case starting with secondary index selection.

However, starting from a primary index restricts the number of possible secondary index sets which seems to be pleasant. But the drawback is that starting from an arbitrary primary index may lead to a sub-optimal index configuration which may be unsatisfactory. Therefore, the selection of a primary index has to be done carefully. We evaluated several heuristics for primary index selection. All the heuristics select a primary index without taking into account the consequence of the choice for possibly further reduction of the processing cost. We extend one of the heuristic such that it takes rather the prospects for further reduction of the processing cost into account in making a choice for a primary index.

References

[1] Barcucci, E., A., Pinzani, R., Sprugnoli, R., Optimal Selection of Secondary Indexes, IEEE TSE, Vol. 16, No. 1, pp. 32-38 (1990).

[2] Choenni, R., Blanken, H.M., Chang, S.C., A Framework for a Tool for Physical Database Design, in Proc. Computing Science in the Netherlands '92, pp. 96-107.

[3] Choenni, R., Blanken, H.M., Chang, S.C., Reducing the Complexity in the Selection of an Optimal Set of Secondary Indices, accepted for publication.

[4] Choenni, R., Blanken, H.M., Chang, S.C., On the Selection of Indices in Relational Databases, Memoranda 92-72, University of Twente, 15p (1992).

[5] Finkelstein, S., Schkolnick, M., Tiberio, P., Physical Database Design for Relational Databases, in ACM TODS, Vol. 13, No. 1, pp. 91-128 (1988).

[6] Ip, M.Y.L., Saxton, L.V., Raghavan, V.V., On the Selection of an Optimal Set of Indexes, in IEEE TSE, Vol. SE-9, No. 2, pp. 135-143 (1983).

[7] Rozen, S., Shasha, D., A Framework for Automating Database Design, in Proc. Int. Conf. VLDB, pp. 401-411, (1991).

[8] Schkolnick, M., Tiberio, P., Estimating the cost of updates in a Relational database, in ACM TODS, Vol.10, No.2, pp. 163-179 (1985).

[9] Walraven, H.G., KOFDO: Een kennissysteem voor ondersteuning van het fysiek database ontwerp, Master thesis, University of Twente, 88p (1990).

[10] Whang, K., Wiederhold, G., Sagalowicz, D., Separability - An approach to physical database design, in Proc. Int. Conf. VLDB , pp. 487-500 (1981).

[11] Whang, K., Index selection in relational databases, in Foundations of Data Organization, Ghosh, S., Kambayashi, Y., Tanaka, K., (eds), pp. 487-500, (1987).