TOPYDE: A Tool for Physical Database Design

Sunil Choenni[1], Henk Wagterveld[1], Henk M. Blanken[1], and Thiel Chang[2]

[1] University of Twente, Dept. of Computer Science, P.O. Box 217,
7500 AE Enschede, The Netherlands
[2] GAK, Dept. of R & D, P.O. Box 8300, 1005 CA Amsterdam, The Netherlands

Abstract. We describe a tool for physical database design based on a combination of theoretical and pragmatic approaches. The tool takes as input a relational schema, the workload defined on the schema, and some additional database characteristics and produces as output a physical schema. For the time being, the tool is tuned towards Ingres.

1 Introduction

The design of databases takes place on several levels. One of these levels is the physical level. Typical subproblems on this level are, among others, selection of storage structures, secondary index selection, vertical fragmentation, materialization, etc. Solving these subproblems requires a sophisticated understanding of physical design options and query optimization strategies of the optimizer, and involve estimating query costs, which is a tedious and error-prone process when done manually. Moreover, several subproblems are NP-complete, such as the selection of an optimal set of secondary indices. Research in this area has been shifted to the problem of determining a good physical design instead of an optimal design [2, 5]. A physical design is considered as good if a competent human database designer would produce the same or a worse design with the same available information.

We present a tool, called TOPYDE, that takes as input, among others, a relational schema, the workload defined on the schema, and other database characteristics, such as page size, cardinality of a relation, etc., and produces for each relation a storage structure (including an ordering attribute or clustering index) and a set of secondary indices. This is called a physical schema. An *overall* physical schema is obtained by the union of the physical schema of each relation involved in the relational schema. For the time being, ordering attributes and indices concern *single* attributes. and a secondary index is stored as a *Btree*.

Although TOPYDE does not cover the overall problem of physical design, it covers the most crucial parts. Moreover, TOPYDE can be easily extended with vertical fragmentation and materialization. We agree with Navathe et al. [7] that vertical partitioning precedes the selection of a physical schema. In [7], vertical fragmentation algorithms are presented that partition a relation into a set of fragments. Such algorithms can serve as a preprocessor for TOPYDE. In practice, materialization is often done as last; this means after the selection of a physical schema. So, TOPYDE can be extended by a postprocessor that aims to improve the physical schema selected by it.

The body of TOPYDE is built on the efforts reported in [3, 4, 5, 11] and is tuned towards *Ingres* [6]. Roughly, TOPYDE is based on the following two principles, which contribute to the control of the complexity involved in the selection of physical schemas.

- For each relation involved in a relational schema, a physical schema is *separately* generated. In [2], we justify the use of this principle for Ingres.
- For each relation a storage structure is selected followed by the selection of a set of secondary indices. This is justified in [2]. The selection of storage structures is performed by applying knowledge rules. The selection of secondary indices is performed by applying an algorithm, which is also tuned towards Ingres [2], and if necessary knowledge rules are used as well.

TOPYDE is based on the so-called *integrated* approach, which can be considered as a combination of a *knowledge-based* and an *optimizer-based* approach.

In a knowledge-based approach, physical schemas are selected on the basis of knowledge rules. These rules are mainly based on heuristics used by human experts. The motivation for a knowledges-based approach is, among others, that experts can formulate rich sets of heuristics to reduce substantially the number of physical schemas to be considered [4]. In an optimizer-based approach, physical schemas are selected on the basis of information extracted from the optimizer, such as estimated processing cost for a database operation [5, 8].

As thoroughly discussed in [2], both approaches have their strong points and flaws. The integrated approach combines the strong points of knowledge-based and optimizer-based approaches [2, 3].

The remainder of this paper is organized as follows. Section 2 is devoted to the architecture of TOPYDE. A knowledge system and Ingres are two main components of TOPYDE. In two consecutive sections, we discuss these components. Implementation decisions with regard to TOPYDE and experiments with TOPYDE are discussed in Section 5. Finally, Section 6 concludes the paper.

2 Architecture of TOPYDE

TOPYDE is based on the integrated approach [2, 3]. The integrated approach produces physical schemas by combining knowledge of (human) experts, information extracted from optimizers, and mathematical properties. Due to space restrictions we will not discuss the integrated approach but the results of applying the approach in designing TOPYDE.

TOPYDE selects a good physical schema, given a relational schema, the workload defined on the schema, and other database characteristics available in a data dictionary. The user has the possibility to specify a set of storage structures from which a storage structure for a relation should be selected and other requirements. The architecture of TOPYDE is given in Figure 1.

TOPYDE generates on the basis of a set of knowledge rules and an algorithm a number of physical schemas for each relation. The knowledge rules are derived from heuristics used by human database designers. The algorithm is based on *mathematical properties*, which are also stored as *knowledge rules*, and is used

for the selection of secondary indices. The mathematical properties are derived by investigating a number of cost functions for secondary index selection.

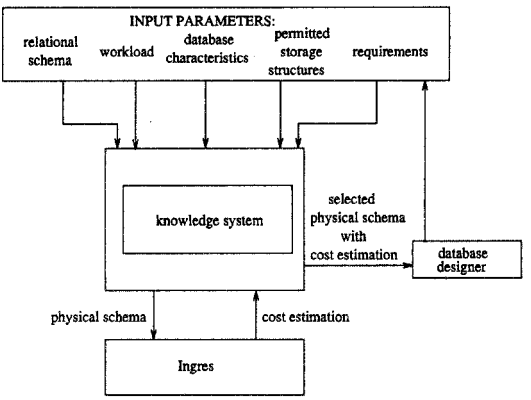

Fig. 1. Architecture of TOPYDE

With each physical schema generated for a relation by TOPYDE, a belief value is associated, expressing the confidence in the schema as being a good one. TOPYDE passes a number of physical schemas with 'high' belief values to the Ingres optimizer. The Ingres optimizer estimates for each schema the cost in processing the workload with this schema. Finally, the schema with the lowest cost is selected for a relation. The *overall* physical schema is obtained by the union of the physical schema of each relation involved in a relational schema.

Let us discuss the reason for passing a number of physical schemas to Ingres and not only the physical schema with the highest belief value. Since belief values are estimations, it may occur that the highest belief value is not the actually highest belief value, e.g., because the estimations are too rough. By passing a number of physical schemas (with high beliefs) to the Ingres optimizer, the chances are better to find a good physical schema [2].

3 Knowledge system

In this section, we discuss how knowledge rules are obtained from heuristics and mathematical properties. First, we give a brief background in the principles used for modelling heuristics and mathematical properties into knowledge rules.

In [3, 4], it is observed that heuristics used by (human) database designers have an uncertain character and contain some degree of ignorance. For example, if a database designer uses a heuristic that says that a subset of the available storage structures will be good in 80% of all cases, this does not automatically implies that in the remaining 20% the subset will be bad. It rather implies that the heuristic can not predict what the scenario will be in 20% of the cases, implying ignorance. These heuristics demand an adequate way of modelling. In [3, 2], it is motivated that the Dempster-Shafer theory [9] is most suitable, since

it offers the possibility to express uncertainty and ignorance by means of belief functions, and it has a rule to *combine* different belief functions.

In the following, knowledge rules consist of an antecedent and a consequent part. To the consequent part a so-called *basic probability assignment* (bpa) is associated, i.e., a belief value between 0 and 1 expressing the confidence in the consequent. Full confidence to a consequent is expressed by belief value 1, while no confidence is expressed by 0. For more details, we refer to [2, 3].

Knowledge rules based on heuristics A number of knowledge rules (with bpa) obtained from heuristics are reported in [10]. These knowledge rules are the result of interviewing 6 experts in the field of physical design of Ingres databases and studying literature focussed towards the design of Ingres databases (e.g. [6]).

We performed the interviews in two phases. In the first phase, we gathered about 48 heuristics used by experts for physical database design. After analysing these heuristics, it appeared that experts select physical schemas on the basis of a limited number of criteria, which is summarized below. Criteria 1–3 are based on database characteristics and criteria 4–6 on workload characteristics.

1. Number of pages required to store a relation. Depending on the required number of pages to store a relation, several storage structures are supported for a relation. For example, if the number of pages required to store a relation is small (≤ 5 pages), then a Heap is often supported as storage structure.
2. The selectivity factor of an attribute. This criterion appears in heuristics to select ordering attributes and indices. In general, an attribute α_j of a relation R_i that has a very small selectivity factor —i.e., $s_{\alpha_j} n_{\text{tup}}^{R_i} < 2$, in which s_{α_j} is the selectivity factor of α_j and $n_{\text{tup}}^{R_i}$ is the cardinality of relation R_i— belongs to the serious candidates that are eligible as hashing attribute. In heuristics for secondary index selection, the selectivity factor of an attribute α_j plays a role in combination with the selectivity factors of other attributes that are candidates for secondary indices.
3. The ratio of the length of an attribute to the length of a tuple. This criterion is used to reduce the number of candidates for ordering attributes and indices. For example, if the ratio of the length of an attribute of relation R_i to the length of a tuple of R_i is greater than 10%, then the attribute is neither a serious candidate for an ordering attribute nor for an index.
4. The ratio of maintenance to retrieval operations on an attribute. This criterion is used in heuristics to select storage structures, ordering attributes, and indices. For example, if the ratio of maintenance to retrieval operations on an attribute α_j is more than 10%, then α_j is neither a candidate for an ordering attribute nor for a clustering index. On the other hand, if the ratio is less than 10%, then depending on the ratio different storage structures (with α_j as ordering attribute) are supported.
5. The ratio of equality to range predicates on an attribute α_j. This criterion appears in heuristics that choose between Hash and Btree or Isam storage structure. For example, if there are 'few' range predicates defined on an attribute α_j, i.e., the ratio of range to equality predicates is less than 1%, then Hash as storage structure is supported with α_j as ordering attribute.

6. The result of an operation. This criterion is used to select storage structures. For example, if the ratio of queries having as constraint that the result should be ordered on an attribute α_j to the total number of queries in a workload is greater than 20%, then the storage structures Btree and Isam with ordering attribute α_j or a clustering index on α_j, are supported.

In the second phase, we transformed the heuristics into production rules (i.e., in the form of IF *condition(s)* THEN *conclusion(s)*). Then, we asked two experts to associate a basic probability assignment (bpa) with each production rule, i.e., to assign a value between 0 and 1 to each consequent, if they agreed with a production rule. We observed that the experts had no difficulties in associating a bpa to a production rule. Finally, we asked another expert to judge the associated bpa to each production rule assigned by the two experts. This expert globally agreed with the associated bpa to each production rule.

In order to combine the obtained knowledge rules, it is demanded that these rules are Dempster-Shafer independent. In [2], it is argued that the obtained knowledge rules are independent for two reasons. First, the conditions in the antecedent part of a knowledge rule do not concern physical schemas. Consequently, a knowledge rule is independent of the results of other knowledge rules. Second, the criteria on which the conditions of a knowledge rule are based are totally different, for example, the number of pages required to store a relation does not influence the selectivity factor at all. □

Knowledge rules based on mathematical properties The knowledge rules that are based on mathematical properties are derived by investigating 3 generally accepted cost functions for secondary index selection. Each cost function estimates the cost of processing a workload defined on a single relation with a given secondary index set, taking into account the benefits and drawbacks of secondary indices. The investigation of these functions resulted into a number of mathematical properties. On the basis of these properties, we have devised an algorithm for secondary index selection. The algorithm is tuned towards the cost function as probably used by Ingres [2]. We briefly sketch the algorithm and the knowledge rules corresponding to the mathematical properties.

The algorithm divides a workload defined on a relation into a number of disjunctive groups of operations, and selects for each group an advantageous and a disadvantageous set of secondary indices. The advantageous set of secondary indices of a group speeds up retrievals, while the disadvantageous set slows down maintenance with regard to the operations of that group. The division of the workload into groups is based on the required number of tuple retrievals in processing an operation with a given secondary index set and the used cost function.

On the basis of the number of operations with their weighted frequencies in a group G, a belief value is assigned to the advantageous and disadvantageous set of G. The belief value for an advantageous and disadvantageous set of G is the quotient of the weighted frequencies of the operations in G and the weighted frequencies of all operations in the workload.

The knowledge base contains 3 knowledge rules that are based on mathemat-

ical properties. The first rule says that if a secondary index i_h is dropped from a secondary index set I and the processing cost of the operations in a group G increases, then i_h belongs to the advantageous set of G. The second rule says that if an index i_h is added to a set I and the processing cost of the operations in G does not decrease, then i_h belongs to the disadvantageous set of G. The last rule is used to divide a workload into groups. It determines the operations that may benefit from a set of candidate secondary index sets. For an exact description of these rules and the algorithm, we refer to [2]. □

4 Ingres

In this section, we describe the storage structures and join techniques offered by Ingres and how to obtain the processing cost of an operation from Ingres.

Storage structures and join techniques To store a relation in Ingres we can choose among Heap, Hash, Isam, and Btree [6]. Ingres provides the possibility to have ordering attributes or indices consisting of more than one attribute. Since we have assumed in Section 1 that ordering attributes and indices consist of single attributes, we will not make use of this possibility. For a detailed discussion with regard to the storage structures the reader is referred to [6].

To process joins in which two relations are involved, the Ingres optimizer uses several techniques: 2 variants of sort-merge, nested-loop, and Cartesian product [6]. To process joins in which more than two relations are involved, the Ingres optimizer determines the join order at a logical optimization step. Once the order has been determined, the join is processed in pairs [2]. □

Processing cost The optimizer is responsible for the generation of an evaluation plan for a given operation. An evaluation plan specifies the actual (basic) operations (joins, projection, etc.) that should be performed in order to process a database operation. In Ingres, such an evaluation plan is called a *Query Execution Plan (QEP)*. QEPs are produced for queries and *not* for other database operations.

From a QEP we may read the basic operations that should be performed in order to process a database operation and the cost entailed by performing each (basic) operation. A QEP is represented as a binary tree and at the top node we can read the *total processing cost* of a query. Since Ingres does not produce QEPs for insertions, deletions, or updates, we apply the following technique.

To perform deletions and updates, relevant tuples to be deleted or updated should be selected first and in case of insertions the proper location should be selected first. The selection cost of relevant tuples or location can be forced from Ingres by formulating a proper query. Suppose we have the following deletion: DELETE FROM *relation* WHERE *name* = 'Tutiram'. We force the selection cost from Ingres by offering the query SELECT * FROM *relation* WHERE *name* ='Tutiram'. After selecting the relevant tuples or proper location, the following steps are performed in order to complete an insertion, a deletion, or an update.

In an insertion, the selected tuples are inserted, in a deletion these tuples are deleted, and in an update the values of specified attributes in these tuples are updated. Since these steps are independent of chosen physical schemas and QEPs, and the cost of these steps is a constant, they do not play a role in determining whether a physical schema is better than another one or not. Therefore, we neglect the cost of these steps in the estimation of the processing cost of a deletion, an insertion, or an update.

Insertions, deletions, and updates entails *maintenance* cost of indices. We have derived cost formulas to estimate the maintenance cost in [2]. □

5 Implementation and experiments

Implementation Currently, TOPYDE is running in a VAX/VMS environment at GAK. The knowledge system of TOPYDE is implemented in an expert system shell, called Aion Development System (ADS) [1]. We choose for ADS because ADS is available at GAK, it supports knowledge rules as knowledge representation technique, and it provides the possibility to reason with uncertainty. Despite the fact that ADS does not support the Dempster-Shafer theory as formalism for reasoning with uncertainty, the combination rule of Dempster and basic probability assignments have been implemented without much effort [10].

The implementation of the algorithm to select secondary indices has been realized in ADS.

In Section 4, we have noted that additional cost formulas are required in order to estimate the processing cost of insertions, deletions, and updates. These formulas, presented in [2], are implemented in C. Finally, the interface between TOPYDE and Ingres has been realized in C and Embedded SQL [2]. □

Experiments Two test cases have been passed to TOPYDE and the results are good. The first case, called Case I, consists of 3 relations on which 10 database operations are defined, and is adopted from [8]. The relations, database characteristics and workload are given in Figure 2. We note that s stands for selectivity factor and f for frequency. The second case, called Case II, consists of 14 relations and 47 operations. Case II is derived from a "large" problem at GAK consisting of about 60 relations and 2000 operations. A closer look at this "large" problem learns us that it can be split into a number of smaller subproblems, since some groups of relations are hardly (or not) related with other groups of relations. Furthermore, a number of operations could be taken together into one operation [2]. We discuss the results produced by TOPYDE for the two cases.

Case I TOPYDE selects the following overall physical schema for the input given in Figure 2. The cost to handle the workload with this overall physical schema is 22624 page accesses.

- *Part* is hashed on attribute *partno* and the secondary index set is empty.
- *Order* is stored as Heap with secondary indices on *orderno, date,* and *suppno.*
- *Quote* is hashed on attribute *partno* with a secondary index on *suppno.*

Relations & Database characteristics:
Part(partno, qonhand, descrip), Order(orderno, partno, suppno, date, qty, oinfo)
Quote(suppno, partno, minqty, maxqty, price, remarks)

Part: # tuples is 8000

	partno	*qonhand*	*descrip*
type	integer	integer	char(184)
1/*s*	8000	4000	8000
range	(0, 7999)	(0, 7999)	("0", "Z...Z")

Order: # tuples is 24000

	orderno	*partno*	*suppno*	*date*	*qty*	*oinfo*
type	char(6)	integer	char(3)	integer	integer	char(71)
1/*s*	12000	8000	100	400	12000	24000
range	("0","Z...Z")	(0,7999)	("A...A","Z...Z")	(19850101, 19930101)	(0,999999)	("0","Z...Z")

Quote: # tuples is 72000

	suppno	*partno*	*minqty*	*maxqty*	*price*	*remarks*
type	char(3)	integer	integer	integer	money	char(15)
1/*s*	100	8000	4000	4000	32000	72000
range	("0","Z...Z")	(0,7999)	(0,999999)	(0,999999)	(0.10,1000)	("0","Z...Z")

Workload:

(f_1 : 5,w_1: SELECT *Quote.suppno, Quote.price* FROM *Quote*
WHERE *Quote.partno* = :partno AND *Quote.minqty* < 1000
AND *Quote.maxqty* > 2000)

(f_2 : 5,w_2: SELECT *Order.orderno, Order.partno, Order.date, Order.qty, Part.descrip*
FROM *Order, Part*
WHERE *Order.date* ≤ 19871216 AND *Order.date* ≥ 19870000
AND *Order.suppno* = :suppno AND *Order.partno* = *Part.partno*)

(f_3 : 5,w_3: SELECT MIN(*Quote.price*), MAX(*Quote.price*) FROM *Quote*
WHERE *Quote.partno* = :partno AND *Quote.suppno* = :suppno)

(f_4 : 20,w_4: INSERT INTO Order
VALUES (:orderno, :partno, :suppno, :date, :qty, :oinfo))

(f_5 : 10,w_5: SELECT DISTINCT *Order.partno, Order.qty,* FROM *Order*
WHERE *Order.orderno* = :orderno ORDER BY qty)

(f_6 : 20,w_6: UPDATE *Quote* SET *Quote.price* = :price
WHERE *Quote.partno* = :partno AND *Quote.suppno* = :suppno
AND *Quote.minqty* = :minqty)

(f_7 : 20,w_7: DELETE FROM *Order*
WHERE *Order.orderno* = :orderno AND *Order.suppno* = :suppno
AND *Order.partno* = :partno)

(f_8 : 10,w_8: SELECT *Part.partno, Part.descrip, Quote.price* FROM *Part, Quote*
WHERE *Quote.suppno* = :suppno AND *Part.partno* = *Quote.partno*)

(f_9 : 2,w_9: SELECT *Order.suppno, Order.orderno, Part.partno, Part.descrip*
FROM *Order, Part, Quote*
WHERE *Order.date* = :date AND *Quote.price* < 1000.00
AND *Order.suppno* = *Quote.suppno* AND *Order.partno* = *Part.partno*)

(f_{10} : 5,w_{10}: DELETE FROM *Order*
WHERE *Order.date* > 19921201 AND *Order.orderno* = :orderno
AND *Order.suppno* = :suppno AND *Order.partno* = :partno)

Fig. 2. Input for Case I

TOPYDE assigned high belief factors (about 0.9) to the selected storage structure of each relation. Therefore, we do not investigate other storage structures for each relation. The algorithm described in Section 3 could decide for each attribute in a relation whether the corresponding index was advantageous or disadvantageous with regard to the whole workload. For example, for relation *Order* the algorithm resulted into the advantageous set {*orderno, date, suppno*} and disadvantageous set {*partno, qty, oinfo*} for the whole workload, which means that {*orderno, date, suppno*} is an optimal secondary index set.

A qualitative analysis of the above-mentioned physical schema shows that it is a good schema. For example, the fact that *Order* is stored as a Heap can be understood, because there are 2 delete operations and 1 insert operation on the relation, each with a relatively high frequency. Since there are relatively many selections on *orderno, date*, and *suppno*, the secondary indices on these attribute can be understood.

In [8], the input given in Figure 2 has been passed to a tool for physical database design, called DAD-I. A fair comparison between the results produced by TOPYDE and DAD-I is not possible, since DAD-I offers multi-attribute indices, while TOPYDE does not offer multi-attribute indices yet. Furthermore, how to compute maintenance cost for multi-attribute indices is not reported in [8]. The overall physical schema produced by DAD-I for Case I is:

- *Part* is hashed on attribute *partno* and the secondary index set is empty.
- *Order* is stored as Btree and clustered on *orderno*. A multi-attribute index to *date, orderno, partno, qty, suppno* is allocated and stored as Btree. Note, *orderno* is indexed twice.
- *Quote* is hashed on *partno*. A multi-attribute index to *suppno, partno, price* and a secondary index to *maxqty* is allocated, both stored as Isam.

Ingres estimates for this physical schema 14726 page accesses as processing cost. However, the maintenance cost is *not* included in these 14726 page accesses. □

Case II As noted before, Case II is derived from a real-life problem at GAK, which could be divided into a number of smaller subproblems. The biggest subproblem consists of 14 relations and 47 operations. On 12 of these relations 5 or less operations are defined. On one relation 8 and on another 12 operations are defined. We pass the two last mentioned relations to TOPYDE with their database characteristics and the operations defined on them. The proposed solutions for these relations by TOPYDE can be considered as good. This statement is based on a qualitative analysis of the solutions and a comparison with the solutions produced by database designers at GAK. □

It is not sensible to draw conclusions about TOPYDE on the basis of the two test cases, because both test cases are relatively small and TOPYDE does not offer multi-attribute indices. Since database administrators often use multi-attribute indices in a physical schema, a fair and precise comparison is not possible. However, it is shown that a tool based on the integrated approach, as described in [2, 3], can be implemented and that it may be viable.

With regard to the two test cases, we conclude that they are not adequate to evaluate TOPYDE, since the solutions for the test cases have been easily

found. The algorithm for selection of secondary indices was sufficient to select a secondary index set for almost each relation (except one), since there was no need to split the workload into groups. The assigned storage structure to each relation was generated with a high belief factor. As a consequence, there was no need to pass other storage structure for a relation to Ingres.

From the two test cases, we have also learnt that a reliable evaluation of TOPYDE will be a laborious task for two reasons. First, we have to find a representative number of non-trivial problems from real-life to pass to TOPYDE and database administrators who are competent to solve these problems too. Second, the input of relations, database characteristics, workload, and physical schemas in Ingres is a time consuming process. □

6 Conclusions

The design and implementation of TOPYDE can be marked as successful. We have passed two test cases to TOPYDE, one adopted from the literature and the other derived from a real-life problem at GAK. The results produced by TOPYDE for these test cases can be considered as good. However, the test cases are not adequate to draw conclusions about TOPYDE self for two reasons. First, the results have been easily found by TOPYDE for these cases. Second, even though the test cases would be adequate, it is irresponsible to draw conclusions on the basis of just two test cases. In Section 5, we have argued that a reliable evaluation of TOPYDE, and, in general, for each tool for physical database design, will be a laborious task. However, we feel that a reliable evaluation of tools for physical database design is useful, since there is a need for such tools.

References

1. ADS-manuals version 6.0, Aion Corporation, 1991.
2. Choenni, R., On the Automation of Physical Database Design, Ph.D thesis, University of Twente, The Netherlands, 1995.
3. Choenni, R., Blanken, H.M., Chang, S.C., On the Automation of Physical Database Design, In: *Proc. ACM/SIGAPP Symp. on Applied Computing 93*, 1993, 358-368.
4. Dabrowski, C.E., Jefferson, K.J., A Knowledge-Based System for Physical Database Design, NBS Special Publication 500-151, 1988, 51p.
5. Finkelstein, S., Schkolnick, M., Tiberio, P., Physical Database Design for Relational Databases. *ACM Trans. on Database Systems 13(1)*, 1988, 91-128.
6. Ingres DBA-guide, Alameda, Relational Technology Inc., 1990.
7. Navathe, S., Ceri, S., Wiederhold, G., Dou, J., Vertical Partitioning Algorithms for Database Design. *ACM Trans. on Database Systems 9(4)*, 1989, 680-710.
8. Rozen, S., Automating Physical Database Design, Ph.D thesis, University of New-York, USA, 1993.
9. Shafer, G., A Mathematical Theory of Evidence, Princeton University Press, 1976.
10. Wagterveld, H., TOPYDE: Een Kennissysteem voor een tool ter ondersteuning van fysiek database ontwerp, M.Sc. thesis, University of Twente, The Netherlands, 1994.
11. Whang, K., Wiederhold, G., Sagalowicz, D., Separability - An approach to physical database design, In: *Proc. 7th Int. Conf on Very Large Data Bases*, 487-500, 1981.