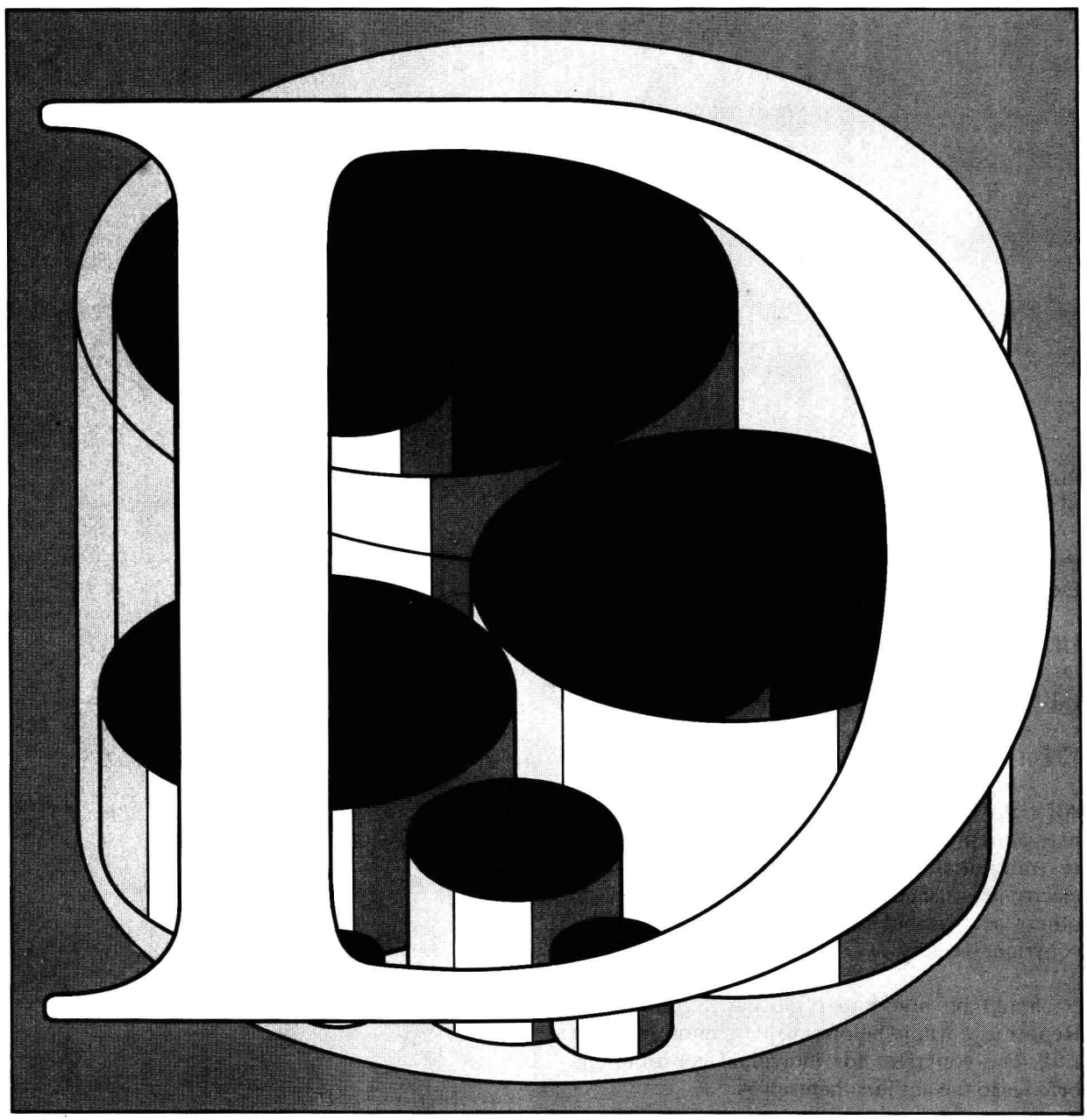
Database performance

State of the Art Report 12:4

Database performance

State of the Art Report 12:4

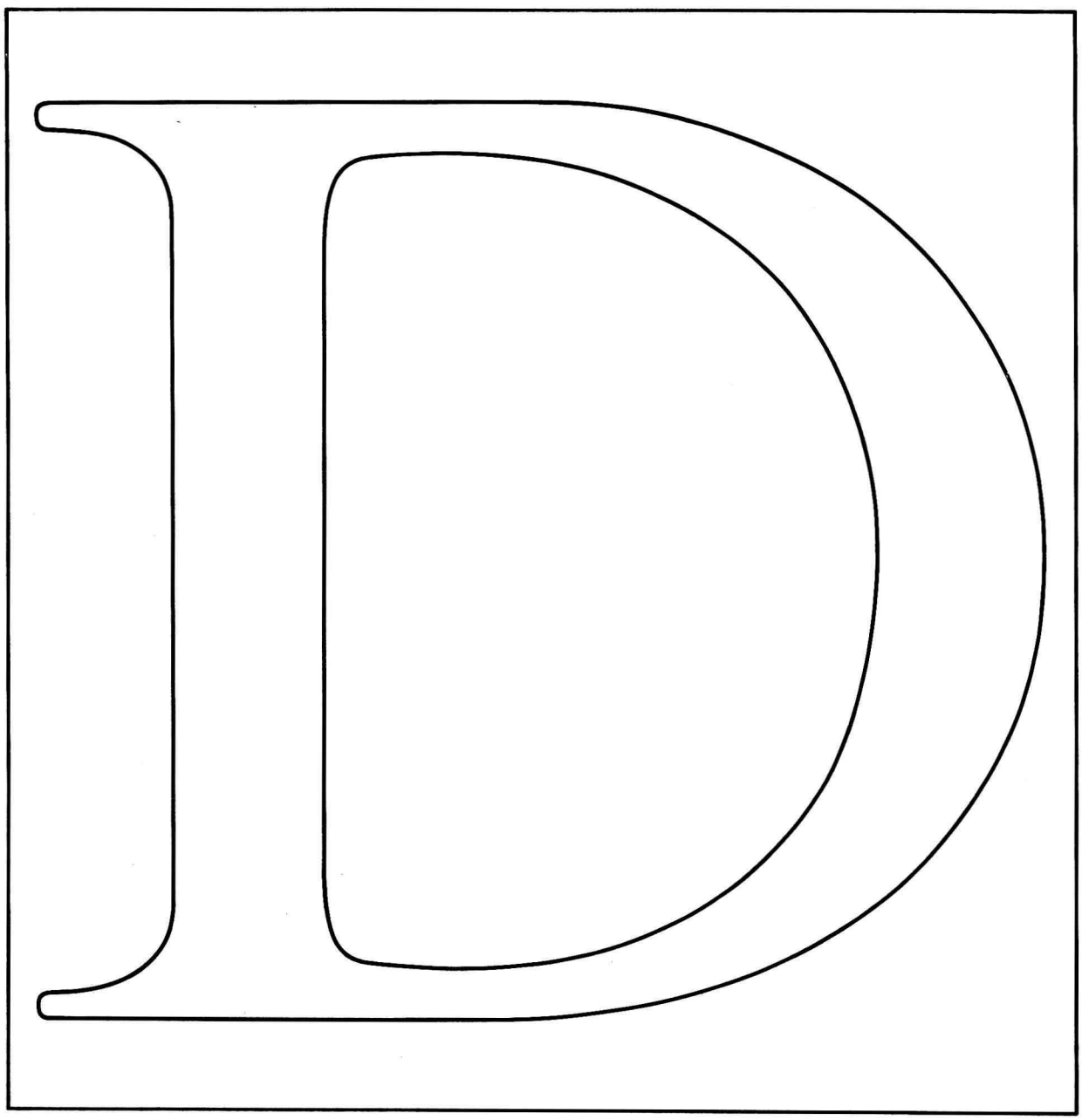


Published by Pergamon Infotech Limited,
Maidenhead, Berkshire, England.

Printed by A Wheaton & Company Limited,
Exeter, Devonshire, England.

UDC 681.3
Dewey 658.505
ISBN 0 08 028 5899

Database performance



Foreword

The widespread acceptance of the database approach to meeting the data needs of large and small information systems has in many cases been in spite of poor responsiveness or throughput or in spite of the considerable cost of acceptable performance in terms of primary and secondary storage, processing power and cycles, and other system resources. It is of course quite acceptable, for pragmatic reasons, to require that the approach should work for particular applications, and that it should be powerful, convenient and flexible, before examining associated performance issues in detail. Furthermore, in a sharing system, it is often possible to tolerate the poor performance of some secondary or non-urgent tasks provided that a few primary or critical tasks are efficiently served. However, the natural companion to such a qualified acceptance is a general expectation on the part of the users that performance problems will become less acute and at some time may even disappear altogether. The reasons given for this hope include the following:

- Improvements in the high-level, application-oriented design of databases
- Better software packages for databases
- Improvements in physical storage techniques and structures
- Better mappings between various components of the database system
- Innovations in hardware
- The development of a number of techniques specifically aimed at improvement of the performance/cost ratios, ie query optimisation and concurrency control.

Significant progress has been made in these areas in recent years and it is important for users, manufacturers, researchers and system designers to be aware of these achievements if they are to exploit the full potential of the database approach. Yet database performance as a subject has rarely had a comprehensive, unified and detailed treatment in the literature. This is probably due to the fact that the body of the knowledge which can be associated with the subject is immense in scope and often the detail in this treasury is of such interest *per se*, or has such interesting primary applications, that the presenters omit to put their concepts into the perspective of the performance of the system as a whole. What is required in such a treatment is an examination of a database system, at its highest level, as a unified system for providing a buffer or store between the supplier of data and its users, and of those components of the system which affect overall performance most. Only in this way will a true perspective be kept as advances are assessed.

During the lifetime of a database system, several distinct phases can be identified, each having its own causes of poor performance and each requiring appropriate techniques for enhancing performance. This, therefore, provides a second requirement in a comprehensive study of database performance — the need to distinguish between the design, development and operational stages of the system.

The phase which will have the highest impact on the system's performance is the design phase. The design process can be broken down into several subprocesses or layers, each being associated with different questions, decisions and parameter evaluations affecting performance. Such a layered model enhances understanding and control of the design process. Even in the early design layers, which include such

activities as data modelling and the mapping of a conceptual model to a chosen DBMS, decisions which will affect the ultimate performance of the system are unavoidable. A very critical aspect of high-level design is the accurate capture of the users' requirements in terms of *how well* a function is to be carried out, stated in terms of the responsiveness, utilisation or throughput performance indices, as well as the data attributes involved. At a lower design level, the physical data structures are chosen, data grouping or clustering methods are selected and the data attributes to be used for indexing are selected. The supply of alternatives for these and other choices have attracted much attention from researchers and implementors; corresponding progress can be reported for these and other system features affecting performance (ie data compression facilities, concurrency control algorithms and buffer management methods).

During the operational phase of a database system's lifetime, it may be necessary to reorganise or restructure the database due to degraded or otherwise inadequate performance. The system components providing efficient query evaluation, especially for *ad hoc* queries, also really come into play during this phase, although there is clearly a good case for dealing with these components along with others such as the compression, scheduling and buffering methods.

In the practical running of performance evaluation studies, opportunities are presented for exploiting some of the experimental and proprietary aids available for the modelling and measurement of performance. There is a considerable challenge in the management of such projects, both in the choice of appropriate tools and methods and in general project planning and control.

With the designers of database machines claiming to have developed attractive alternatives to software implementations of some system components, and the challenges of developing distributed databases (some on very small computers) and large knowledge bases with acceptable performance, it is clear that database performance will remain an exceedingly interesting subject.

D A Bell: Editor

Publisher's note

This Report is divided into three parts:

1 Invited Papers.
2 Analysis.
3 Bibliography.

The Invited Papers in this State of the Art Report examine various aspects of database performance. If a paper cites references they are given at the end of the Invited Papers section, numbered in the range 1-99 but prefixed with the first three letters of the Invited Paper author's name.

The Analysis has the following functions:

1 Assesses the major advances in database performance.
2 Provides a balanced analysis of the state of the art of database performance.

The Analysis is constructed by the editor of the Report to provide a balanced and comprehensive view of the latest developments in database performance. The editor's personal analysis of the subject is supplemented by quotations from the Invited Papers and current literature, written by leading authorities on the subject.

The following editorial conventions are used throughout the Analysis:

1 Material in Times Roman (this typeface) is written by the editor.

2 Material in Times Italic (*this typeface*) is contributed by the person or publication whose name precedes it. The contributor's name is set in Times Italic. Numbers in parentheses in the ranges 100-232 or 001-099 following the name refer to the original source as specified in the Analysis references or the Bibliography, respectively, which both follow the Analysis. References within the text are numbered in the same way. A contributor's name without a reference refers to an Invited Paper published in this Report.

3 The quotations in the Analysis are arranged at the discretion of the editor to bring out key issues. Three or four dots within a single quotation indicate that a portion of the original text has been removed by the editor to improve clarity.

The Bibliography is a specially selected compilation of the most important published material on the subject of database performance. Each key item in the literature is reviewed and annotated to assist in selecting the required information.

Contents

Editor's foreword			vii
Publisher's note			ix
Invited Papers			
1 Query optimisation in relational database systems		*S Ceri*	3
2 Query optimisation in relational databases using improved approximations of selectivities		*S Christodoulakis*	21
3 The PRECI placement and indexing technique		*S M Deen*	39
4 Performance issues in concurrency control		*J Grimson*	55
5 The use of cluster analysis for the enhancement of database performance		*J A Hoffer*	65
6 Data dictionary features for reorganisation, restructuring and query optimisation		*C I Johnston and A S Stone*	83
7 Data access methods and structuring to enhance performance		*W Litwin*	93
8 Performance analysis for DBMS evaluation in information systems planning		*K Murphy*	109
9 Modelling distributions of secondary keys		*W B Samson and A Bendell*	121
10 Performance-oriented database design laboratory		*W Staniszkis, S Orlando and P Rullo*	131
Invited Paper references			161
Analysis			
1 Performance evaluation in database systems			173

2 Enhancing performance in conceptual and logical databases 191

3 Physical database design 203

Physical access methods 205

Selecting indexing attributes 215

Data compression and redundancy 221

Placement techniques 225

4 Other systems features critical to performance 231

Buffer management 233

Concurrency control 237

Contributions from other prominent system components 245

5 Performance improvement in an operational database system 249

Query optimisation 251

Reorganisation 259

Restructuring 265

6 Conducting performance evaluation studies 271

7 Performance issues in conspicuous innovations 287

Analysis references 301

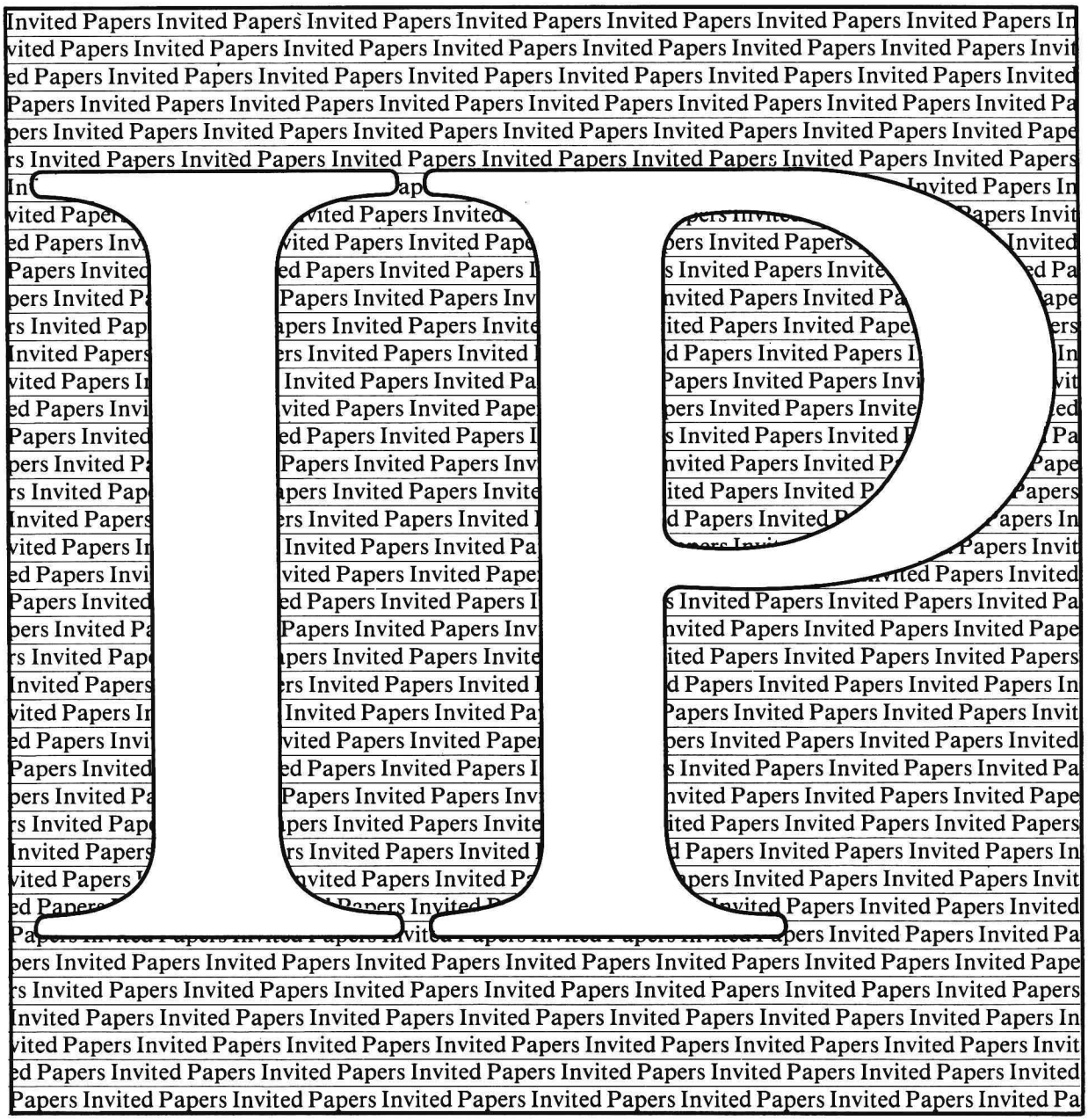
Bibliography

An annotated bibliography on database performance *D A Bell* 311

Index

Subject and contributor index 331

Invited Papers

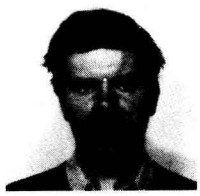


1: Query optimisation in relational database systems

S Ceri

Milan Polytechnic
Milan
Italy

This paper reviews some of the query optimisation methods in relational database systems: the goals and importance of query optimisation are discussed, and the basic models and techniques for query optimisation which apply both to centralised and distributed systems are reviewed. An important but very simple optimisation consists in applying selections and projections as early as possible; thus, the most difficult problems of query optimisation are due to the execution of joins. Query processing methods for centralised databases are exemplified by two very different approaches; the query decomposition method of Ingres, and the query optimisation approach of System R; and for distributed databases: the use of semi-join reduction, and the query optimisation approach of System R.



S Ceri

Professor Ceri received his doctorate degree in electrical engineering from the Politecnico di Milano, Italy, in 1978. At present he is Research Associate at the Electrical Engineering Department of the Politecnico di Milano. He performs research in database management, primarily in distributed databases, database design and the use of relational databases for the development of software systems. His principal activity for the past five years has been in distributed databases, where he has studied the correctness, optimisation and synchronisation of distributed transactions, and the design of the distribution of database schemata. He is co-author of the book: 'Distributed databases: principles and systems', published by McGraw-Hill in 1984, and he is author of over 30 papers published in journals and conference proceedings. During 1983 and 1984, he has been visiting professor at Stanford University.

Query optimisation in relational database systems

Introduction

The relational model and languages have been developed with the goal of allowing an easy interaction with the users. The relational model of data incorporates all its semantics directly in relations, which are viewed by the user simply as tables with rows and columns. Relational languages are non-procedural, ie they do not require the specification of a procedure for collecting the result of a query, but they simply require definition of the logical properties of the result. The *optimisers* are software modules which assume as input an internal representation of the query, produced by the parsers, and perform the complex task of selecting an access strategy to the data. Thus, optimisers substitute for some of the functions of application programmers; the performance of systems is heavily influenced by the performance of optimisers.

In general, the same query can be executed in many different ways; this is due to the following two facts:

1 Equivalence transformations can transform a given user query into several other queries which produce the same result.

2 The operations of a query can be mapped to the internal structures of a database in many different ways; for instance, a join operation between two relations can be executed using several different methods.

An optimiser must perform both query transformation and operations mapping in order to produce an access strategy. However, it is relevant to distinguish the above two steps in query optimisation.

The first step corresponds to a 'logical' transformation of the query; thus, it can be studied independently on the different internal structures which constitute the storage system of a relational database. Moreover, many transformations are always beneficial, and thus they need not be evaluated on a cost basis.

The second step requires instead the quantitative evaluation of alternatives, which is done on the basis of their estimated performance. This step requires the use of an 'operations research' approach. Since the query optimisation problem has a complexity which is combinatorial with the number of its decision variables, algorithms used in the second step are typically heuristic and do not make an exhaustive search of the possible solutions: therefore, the result of an optimiser is not the optimal solution, but is nevertheless a good solution.

In the evaluation of the efficiency of an optimiser it is necessary to compare the features of the solution produced by it with the amount of time required for computing the solution. In certain cases, an optimiser which is capable of producing the optimal solution via a long computation does not perform so well as an optimiser which finds only a good solution quickly. This consideration applies to the case of on-line

queries, while it is not necessarily true for precompiled queries, ie queries embedded into application programs. In the latter case, a long computation performed by the optimiser could be compensated by the increase in the performance of queries, since they will be executed several times. Thus, the most sophisticated optimisers should behave differently with interactive queries and with precompiled queries.

Query optimisation can be aimed at two different goals: either the minimisation of the delay between the issuing of a query and the production of an answer, or the minimisation of the cost associated with a query execution strategy. The two approaches lead to the production of access strategies with different features.

The first approach leads to the use of parallelism: if possible, queries are decomposed into independent subqueries which can be processed in parallel. Subqueries are divided into two classes: critical and non-critical. The subqueries which are processed in parallel with other subqueries, but require a shorter time than the others, are not considered 'critical'. The value of the goal function for a given execution strategy is just the summation of times of the critical subqueries.

In an on-line query, it might be relevant to consider the delay between the issuing of the query and the production of the first row (or screen) of the result. In fact, this is the time in which the user remains 'idle'; but once the first row is produced at the terminal the user will probably read the result more slowly than the system computes the remaining rows or screens.

The second approach does not emphasise parallelism, and aims at the minimisation of the use of resources. In this case, the value of the goal function associated with a given execution strategy is the weighted sum of resource utilisation.

The use of a mixed approach is possible. In fact, when one of the two approaches is used, say delay minimisation, it is convenient to select from the strategies with the same delay the one with the minimum cost or *vice versa*.

In a centralised system, the critical resources for a query execution strategy are the CPU utilisation and the number of I/O operations. Though CPU utilisation can be important for certain operations (eg sorts), the critical factor for most systems is the number of I/O operations. Thus, access methods (such as hash tables, links and indices) are developed for reducing the number of I/O operations required for accessing a relational database. Moreover, special database machines are developed for the same purpose.

In a distributed database, a critical resource is data and message transmission on the computer network. Data transmission is required to move portions of the database in order to collect the result at the user's site. Message transmission is required in order to control the execution of queries. With large databases, message transmission is much less expensive and time consuming than data transmission, and thus it is not considered by the optimisers. In geographically distributed networks, data transmission is typically the most critical factor with respect to CPU and I/O utilisation since transmission is slower than that of an I/O channel by three orders of magnitude. In a local network the I/O and transmission speeds are comparable.

In order to understand the importance of query optimisation, let us consider a simple query which requires the join of two relations, R and S, and the selection of one of them, say R. Let us assume that initially R and S have 1000 rows each, that the join has a factor 10 with respect to R (ie 10 rows are produced in the result of the join for each row of R), and that the selection has a selectivity of 0.01 (ie one row is selected out of 100 rows of R).

The 'logical' optimisation consists in the evaluation of the selection *before* the join. In this way, R is initially reduced to 10 rows, which are later joined with S, producing 100 rows of the result. If, instead, the operations were performed in the opposite order, a temporary result of 10 000 tuples would be generated by the join operation; then, the selection would be applied to this temporary result, producing the same 100 rows of the first solution. In terms of resource utilisation, the first solution uses less CPU time because smaller relations are manipulated and the I/O operations are saved for storing and retrieving the temporary result. In a distributed system, assuming that R and S are stored at different sites and that R is sent to S in the final execution strategy, the first solution requires the transmission of 10 rows instead of 1000 rows.

Next, the optimiser must select a method for performing the selection and join. In the evaluation of the selection, the optimiser must determine which access methods can take advantage of the selection condition, and use the best method. In the example, let us assume that there exists an index which allows the direct accessing of the 10 rows selected by the condition; then the optimiser should select this index. Finally, a join method should be selected by the optimiser in order to produce the final result.

Basic techniques in query optimisation

In this section, the basis for query optimisation is developed. It is assumed that the reader is familiar with the relational model and algebra presented in, for instance *(CER1, CER2, CER3)*. Here, the notation of *(CER3)* is used; therefore:

1 REL(ATTR1,ATTR2,..ATTR*n*) indicates the relation schema of the relation REL, having ATTR1, ATTR2,... ATTR*n* as attributes. Upper case letters are used for relations and attribute names; lower case letters indicate tuples (eg *t* is a tuple of REL). The value assumed by the attribute ATTR of the tuple *t* is denoted by ATTR[*t*].

2 The following operations of relational algebra are used in this paper:
- $SL_{Condition}R$ indicates a selection of the relation R
- $PJ_{Attribute\ list}$ R indicates a projection of R
- R **CP** S indicates the Cartesian product of R and S
- R **UN** S indicates the union of R and S
- R $JN_{Join\ condition}$ S indicates the join of R and S.

Query languages are more powerful than relational algebra; they express some features which have no counterpart in algebra, such as aggregate functions, ordering of tuples within relations, grouping of tuples having common values, etc. However, the class of queries considered in this paper, called select-project-join queries, can be formulated as relational algebra expressions involving just selections, projections and joins. These queries cover a large and relevant subset of all possible queries and are mostly studied in the literature of query optimisation.

Consider the following relation schemata:

EMP(NAME, DNO, SAL, SOCSECNO, SKILL)
DEPT(DNO, LOCATION, MANAGER, ADDRESS)

The following example is a select-project-join query:

$$PJ_{NAME,MGR}\ SL_{LOC='LONDON'\ AND\ SAL\ <\ 35.000}\ (EMP\ JN_{DNO=DNO}\ DEPT)$$

The meaning of this example query in English is: 'Select the employees and the managers of their departments such that departments are located in London and employees earn less than £35 000'. The example query can be formulated in several relational languages; its formulations in SQL *(CER4)*, Query-By-Example *(CER5)* and QUEL *(CER6)* are shown in Figure 1. Assuming relational algebra for writing queries instead of any one of the relational languages above has the advantage of allowing a general treatment of query optimisation; in fact, each system is capable of parsing the query and producing an internal form of it which is roughly equivalent to its expression in algebra.

A structural representation of an expression of relational algebra is given by operator trees. Operator trees are totally equivalent to relational algebra expressions; however, their structure indicates more clearly the order of applications of operations. Figure 2a shows the operator tree of the example query. The 'leaves' represent the relations appearing in the expression; the 'root' represents the result of the expression. The operator tree in Figure 2a indicates that the join operation is applied first, followed by the selection, followed by the projection.

Expressions of relational algebra can be transformed using equivalence transformations, ie transformations which produce other expressions with the same meaning. Equivalence transformations are

```
SELECT NAME, MGR
FROM EMP, DEPT
WHERE EMP. DNO = DEPT. DNO
AND LOC = 'LONDON'
AND SAL<35.000
```

a)

EMP	NAME	DNO	SAL	SOCSECNO	SKILL
	P.JONES	D1	<35.000		

DEPT	DNO	LOCATION	MANAGER	ADDRESS
	D1	LONDON	P.SMITH	

b)

```
Range of E is EMP
Range of D is DEPT
Retrieve into RESULT (NAME = E.NAME, MGR = D.MGR)
Where E.DNO = D.DNO
and E.SAL<35.000
and D.LOC = 'LONDON'
```

c)

Figure 1a: Example query in SQL
Figure 1b: Example query in QBE
Figure 1c: Example query in QUEL
Figure 1: Example query