Databases-Role and Structure

An advanced course

P.Bailey
D.M.R.Bell
E.Bertino
P.Buneman
W.P.Cockshott
K.J.Chisholm
D.Daniels
P.A.Dearnley
S.M.Deen
W.A.Gray
L.M.Haas
D.C.Hendry
R.G.Johnson
Y.G.Kollias
J.S.Knowles
K.G.Kulkarni
B.Lindsay
W.Litwin
G.Lohman
Y.Masunaga
C.Mohan
R.Morrison
P.Ng
E.A.Oxborrow
U.Schiel
P.Selinger
M.R.Shave
P.Wilms
R.Yost

Edited by P.M.Stocker, P.M.D.Gray and M.P.Atkinson

Databases – Role and Structure

AN ADVANCED COURSE

Edited by

P. M. STOCKER

School of Computing Studies and Accountancy, University of East Anglia

P. M. D. GRAY

Dept of Computing Science, Aberdeen University

M. P. ATKINSON

Dept of Computer Science, University of Edinburgh

CAMBRIDGE UNIVERSITY PRESS

Cambridge

London New York New Rochelle

Melbourne Sydney

Published by the Press Syndicate of the University of Cambridge
The Pitt Building, Trumpington Street, Cambridge CB2 1RP
32 East 57th Street, New York, NY 10022, USA
296 Beaconsfield Parade, Middle Park, Melbourne 3206, Australia

First published 1984

Printed in Great Britain at the University Press, Cambridge

Library of Congress catalogue card no.: 84-5909

British Library Cataloguing in Publication Data

Databases–Role and Structure.

1. Data base management 2. File organization
(Computer science)
I. Stocker, P. M. II. Gray, P. M. D.
III. Atkinson, M. P.
001.64'42 QA76.9.D3

ISBN 0-521-25430-2

CONTENTS

INTRODUCTION

OVERALL SUMMARY OF THE MATERIAL

Two database systems have firmly established themselves from the early days of the subject: IMS and CODASYL. Both are systems which connect data together into a single integrated structure. Both have changed appreciably with the years; both have latterly been influenced by more recent developments in relational systems and both have been constrained by the form of their beginning and the requirement for backward compatibility. This requirement to follow a smooth historical path rather than to make a discontinuous change arises out of sheer economic necessity. The immediate past contains a large investment, the income from which cannot just be given up. Moreover, fresh capital and labour with appropriate skills are scarce. Thus any account of database systems concerned with interfacing and standardisation must start from an acceptance that changes must be compatible with, and provide a smooth development path from, the present situation concerning database implementations in thousands of organisations.

CODASYL has many years of controlled development through formalised standards. It is in fact a standard; particular implementations by manufacturers have distinct names, IDS (Honeywell), IDMS (Cullinane), DMS1100 (Univac) and form well-developed operational systems. The development of CODASYL is briefly outlined in the article by Knowles and Bell, 'The CODASYL Model'. The article is included because it presents CODASYL up to the 1981 standard for the DDL (Data Description Language) and DSDL (Data Storage Description Language). In introducing the DSDL a significant step is taken towards the definition of separated areas of the overall database structure, although the DSDL is not yet part of the CODASYL standard. It is, perhaps, worth elaborating this point further.

At the time of the early CODASYL standards the description of a database was seen as consisting of two parts. The first was the data definition language, which defined the structure which had been provided to accommodate the data; the second was the data manipulation language defining the way in which data could be inserted into the structure, deleted from it, changed or retrieved.

In 1972 the ANSI/SPARC Committee produced a Report directed towards partitioning the database specification. This is discussed in the paper: 'ANSI-SPARC Architecture and its Implementation in the PRECI model' by S.M. Deen. The architecture proposes three levels of Data Definition:

i) the Conceptual Level capturing the semantics of the real-world model,

ii) the External Level capturing the views seen by the user,

iii) the Internal Level capturing the essential nature of the implementation at physical or near physical level.

The early CODASYL DDL was a combination of (ii) and (iii), in that the user (or at least the database programmer) saw a logical structure which was in 1-1 correspondence with the physical structure. The Conceptual Level is not present in CODASYL at the present time, though CODASYL systems are usually supported by a Data Dictionary which contains some aspects of a conceptual schema. The progression from DDL into DDL plus DSDL achieves the separation of logical description and physical description.

The Course contained a paper on Relational Systems corresponding to Knowles' paper on CODASYL. This is omitted here because the material is adequately covered, at much greater length, in a number of easily available books. General observation from the lecture is included here, however, because it is relevant to what follows. Readers should also see the report of the tutorial lecture by Litwin, 'Successes and Failings of Modern Database Systems', for a different look at the history of ANSI/SPARC and Relational Systems. Whilst agreeing about the key contributions of Relational Systems it takes a more pessimistic and radical view about the usefulness of integrated conceptual schemas.

Viewed in retrospect it can be seen that the primary impact of the Relational Model was the use of relations to define the logical content of a database and the use of relational algebra precisely to specify queries, and it is probably these aspects which were the inspirational force behind Codd's seminal paper. Important as these are there are two other aspects which have had at least as important a consequence on database development.

The first is that, unlike the CODASYL DML, relational specification of the logical data says nothing concerning the proposed form of implementation. Equally, the non-procedural form of relational queries says nothing concerning the form and order of processing. Early relational systems simply implemented physical structures which were direct images of the logical ones. These implementations were naturally inefficient and so supporting index structures were added. These led to consideration of the choice of ways in which a relational query might be processed, and hence the study of query optimisation and automatic path selection. This in turn led to an interest in automatic path selection in non-relational CODASYL systems. For that optimisation to be possible the query must be specified in a non-procedural manner, in order that the whole process required of the CODASYL system is specified at the outset. Thus one reaches the position where a query is expressed in relational terms but applied to a CODASYL system, resulting in the possibility of automatic, optimised, CODASYL DML generation. This is covered in Gray ('Implementing the Join Operation on a CODASYL DBMS'). The second feature of relational systems is that it is much easier to modify the data description, to include new material in the database, than it is with CODASYL or IMS. This is due to the fact that the CODASYL and IMS structures are linked at DDL definition time, whereas in the relational system it is at query process time.

The preceding two paragraphs stress that the important factors are the complete non-procedural specification of the queries and the run time structural binding. Because these two important advances first developed in relational systems there is a tendency to attribute them to the relational model only. Gray's work shows that given a non-procedural query description, optimized, automatic query processing is possible on a non-relational implemented structure linked at definition time. Equally, one could define a database structure which consisted of, say, hierarchic units rather than relations, with the hierarachic units linked at query run time rather than at data definition time. This would show many of the advantages associated with relational systems.

It is clear from the foregoing that there should be a complete distinction between the logical structure of the data and the physically implemented structure. This has been appreciated for a number of years, but there is no doubt that it is difficult fully to establish this distinction. Undoubtedly one reason for this is that the operational systems which are currently available do not permit any more than nominal flexibility in achieving the separation.
At the present one can say that the statement 'the DDL is a logical description' is approximately true for most systems. The virtual record mechanism described in the DSDL (Knowles, 'The CODASYL Model') provides facilities for one form of physical separation, but the DDL, DSDL distinction is not yet implemented on available CODASYL systems.

The article by Stocker, ('Relational Tutorial') examines the representation and definition of the mapping between a logical schema and a physical schema for relational systems and with some reference to the case where the physical schema is of a CODASYL or IMS nature. This article is concerned with representation rather than optimisation. A CODASYL schema may always be viewed as a complex stored join of a set of relations. The article by Gray ('Implementing the Join Operation on CODASYL DBMS'), is a detailed analysis of that particular issue. It goes considerably further in that it is concerned with optimisation, to find the best mechanism to actually realise the join which is latent in the CODASYL data structure.

The most simple divergence between logical schema and physical schema is the case where basically there is a one-to-one mapping between them but the physical schema contains additional indexes to the date. This affects automatic path selection in that the selector has to decide which indexes to use and at what point in the sequence. Earlier than this, however, at the last system definition point, the database administrator (whether human or automated) must have taken the decision concerning which indexes to maintain. The article by Kollias, ('A Generalised Model for the Selection of Secondary Indexes'), describes a formal tool for the index selection problem.

So far, in this article, the database schemas have been discussed in the rather old-fashioned form of logical and physical schema. The *logical schema discussed so far has been concerned with data rather* than with the semantics of data. This concept of a logical schema is not so strong as that of the conceptual schema contained in the ANSI/SPARC architecture. The article by Deen, ('ANSI/SPARC Architecture and its Implementation in PRECI') describes this architecture and there is no need further to discuss it here. It leads the reader into three articles: Stocker et al ('(PROTEUS: A Search for Standard Components in a Heterogeneous Distributed Database System'), Schiel ('The Semantic Data Model and its Mapping to an Internal Relational Model'), and Gray ('The Functional Data Model related to the CODASYL Model').

All four articles are concerned with the problem of the overall database *system. Three of them, those by* Deen, Gray and Schiel, contain instances of a conceptual schema in the ANSI/SPARC sense. That of Deen is the closest to the preceding discussion and is concerned with incorporating semantics into a schema which at the same time relates to the near implementation schemas of existing database systems. Gray's paper concerns the functional data model, discussed later in this Introduction in more detail, particularly in connection with the relationship between database languages and programming languages.

The functional data model comprises both data description language and data manipulation language in a compact and integrated manner. Just as in earlier articles, where the mappings from *relational data description and query languages to the corresponding* CODASYL were considered, so it is possible to consider mapping from a functional language into both, though semantic meaning may be lost. The paper by Gray is concerned with such mappings.

Conceptual schemas (like programming languages) must meet a variety *of needs for an even greater variety of individuals.* As with programming languages much depends upon personal task and environment. The functional language is sparse with few constructs; the conceptual schema of Schiel represents an alternative view, being rich in alternative constructs. Schiel's article contains details of a complete mapping system, from conceptual schema to logical specification of the data storage implementation.

The fourth article by Stocker et al is discussed later in connection with the implementation of a distributed database system. It suffices here to note that the earlier portion is also concerned with the conceptual schema, but, in that case, a target schema is provided into which a functional or 'rich' schema can be mapped. Thus the PROTEUS system described there is uncommitted to a particular conceptual schema, freedom of choice remaining with the distributed nodes. Deen's paper is also concerned with a complete implementation, PRECI, and, as will be discussed later, a distributed version of it, PRECI*.

An important theme in the conference was the influence of the new experimental distributed architectures on the old debate about centralisation and decentralisation. We have seen how the ANSI/SPARC model calls for a unifying conceptual schema whilst the relational model is more neutral on this matter.

The tutorial paper by Deen ('Issues in Distributed Databases'), discusses the centralisation issue under various headings: control, heterogeneity or homogeneity, integrity and user views. It shows the different options that can be taken and is a useful introduction to the four other papers.

The paper by Gray and Litwin, ('The Successes and Failings of Modern Database Systems') is most radical, calling for the establishment of a multi-database where each node controls its own data, using its own DBMS and its own DML. However, each database must provide a view to others via a common, non-procedural language, of those items which it is prepared to share. It also advocates the handling of bibliographic text and graphic images across the net, which current data models inhibit.

The paper by Stocker et al, describing the Proteus system, is also concerned with linking heterogeneous systems. It is concerned to establish standard internal languages for the transmission both of schemas and queries. The schema of each database can be expressed in an ACS (Abstract Conceptual Schema) which is represented by a number of meta-relations. These include information on the compatibility (or comparability) of data items values and also on integrity constraints. The standard internal languages are extensible to allow the representation of integrity constraints and of queries whose external source forms are complex expressions (e.g. as used in DAPLEX (The Functional Data Model and the Data Language DAPLEX, D.W. Shipman, ACM Transactions of Database Systems, Vol. 6, No. 1, March, 1981). Each site has to provide a mapping from its own schema and query language, and implementations are in hand for Functional, Relational and CODASYL systems. Given this facility it is possible to set up a global schema at a central site, which can be used in the process of expanding a network query into queries against local schemas and in combining the results.

However, this is not essential and any site can request a schema from another site for two-way communication. The current system is only for global queries and avoids the problems of concurrency and internodal consistency.

The paper on R* by Selinger et al ('The Impact of Site Autonomy on R*, A Distributed Relational DBMS') addresses the problem of internodal consistency in a way which preserves site autonomy - the essence of a decentralised system. The crucial principle is that no site should have to consult another site about what it does with its own data. Any change that is made is marked by a change of version number and thus any site which requests an action based on an out-of-date version may be asked to form a new plan and try again. The only problem that arises is on commitment of a transaction involving several sites. Site autonomy allows a local site to back out unilaterally and so jeopardise inter-site consistency. The system is homogeneous. All sites use the R* DBMS software and so there is automatically a standard schema language and a standard query language (based on SQL). The paper discusses the interesting question of the use of 'system-wide names' so that queries can explicitly refer to data at other sites (there is no global schema). Names used in the SQL language may be bound to different system-wide names on recompilation by using different catalogues.

The paper on PRECI* by Deen ('PRECI* a Project for Distributed Databases') describes a system which has the greatest potential for centralisation of the four systems discussed. It is based on the ANSI/SPARC concept at two levels (local and global). The external schemas of the individual nodes appear as 'partition schemas' which play the role of nodal storage schemas to an overall 'Federated Canonical Schema' (FCS) at the global level. At the local level each node has its own three-level schema. The FCS uses the same notation as PRECI to describe the database as a number of relations. A 'Mapping Division' states which attributes in relations at different nodes are compatible and what data conversion should be performed to make them comparable. Allowance is made for replicating data at several nodes and the problems of maintaining internodal consistency are discussed.

These four papers show a range of approaches. In each one the pressure for decentralisation and site autonomy is argued, but is conceded to a different extent. In all cases the principle of logical/physical data independence is built in and all systems use a relational (non-procedural) data manipulation language. The principles of commitment, locking and recovery at individual nodes, seem to be widely agreed but there are problems with maintaining both internodal consistency and full site autonomy.

The theme of data models and database construction running through this earlier part of the course addresses the organisation and storage of data. There is little point in storing data without arrangements to use it. These arrangements typically depend on a language to manipulate and extract data pertinent to some query or investigation. In many cases the language will also contain facilities for amending the data.

Strictly speaking, all the languages which enable such operations are programming languages, but it has become conventional to use the term query languages for those that are limited to extraction and simple derivation data. In fact, many available query languages also have update facilities and so should be called data manipulation languages. Two approaches to presenting these facilities exist, in one a language, usually interactive, stands alone, and is referred to as an interactive query language. The other approach is to embed the database language in some other general purpose programming language. In this case it is usually referred to as a data manipulation language or sub-language. Knowles gives examples of such a language in his tutorial chapter on 'The CODASYL Model' standard, when he presents its DML.

Both these approaches to providing languages for utilising the data in databases have been demonstrated to be workable, both are often provided in commercially marketed DBMS and are widely used as mechanisms for opereating on the data in databases. The query languages are perhaps less well established in practical use, but there are dramatic examples of their effectiveness. In spite of this neither method can be considered ideal. With query languages the compromise between ease of learning the language and what can be achieved using the language has not been fully explored. Developments in this area are a mixture of detailed improvements to the consistency and style of the language, to its support environment and to its *functionality*.

The presentation by Knowles and Hendry is a good example of such development. They describe a language, RASQL, which provides a query language for RAPPORT databases. The language they have implemented is very similar to that provided in many other databases, particularly the SQL of SYSTEM-R. However, they show how parameterisation, naming of sub-queries and a good query language support environment, can enhance the capabilities of such languages.

Such work does not address the function served by the embedded languages. Embedding is valuable when a complex computation is to be performed on the data. In many applications, such as computer-aided design, the entire use of the data falls into this category. In many applications data is always used via large and often complex programs, either to provide tailored interfaces, specific checking or sophisticated derivation of data. Almost all collections of data require some programs to be written to use the data. One can envisage extending the query languages to take over more of this role or of improving the programming language interface to stored data.

The final days of the Course considered this issue in some detail, as it is an area of practical importance which is often neglected. Two chapters deal with an approach based on the Functional Data Model. Gray gives an introduction to the notational form of an elaboration of this model developed by Shipman called DAPLEX, and shows how it relates to the more familiar CODASYL model. DAPLEX organises its data in terms of entities or objects, and functions over these objects. A function yielding an object performs the same role as a data item in CODASYL or an attribute in the relational model. But in the functional model it is not necessary for the data designer to provide functions (attributes) which act as unique identifiers, since the objects themselves are values. In DAPLEX sets of such values are allowed, the results of multivalued functions which can be used to model one-to-many and many-to-many relationships, and which are analogous to the owner-member sets of CODASYL. A useful feature of DAPLEX is property inheritance. Objects may be arranged in hierarchies of types, so that an instance of a sub-type inherits all the properties of the corresponding instances of its super-types. This, and the direct use of objects as values, captures common semantics of data which have to be explicitly added to the relational and CODASYL models.

The DAPLEX language was designed as an extension to a general purpose programming language such as COBOL or ADA. The article by Atkinson and Kulkarni reports a different utilisation of the DAPLEX idea. They describe an implementation of a language like DAPLEX and discuss how it might be consistently extended to provide a complete general purpose language with all the functions required for managing long-term data. They suggest that the view mechanism can be used in the traditional way for information hiding and protection, with greater flexibility than Shipman identified. But they also note that if the total language is designed with consistent rules, then the normal declaration mechanisms used in conjunction with views provide additional functionality. Such declarations allow the evolution of data types, equivalent to editing schemas or metadata; they allow the use of the view as the unit of transaction; provide a way of defining the context of an experiment on the data and provide a notation for describing federated databases. These extensions to the functionality of the language are still the topic of research and experiment.

The chapter by Buneman also considers the functional model of computation. This model is attractive as a basis for integrating databases and programming languages cleanly, something which has proved difficult with the relational and CODASYL models. Work on applicative languages demonstrates that a wide variety of programs can be concisely and elegantly represented as functions. Collections of data can also be viewed as stored functions. The same notation can be used to describe both. All queries and data derivations can then be represented as the evaluation of a function. Buneman describes FQL, a functional query language with simple semantics. It is a good candidate as an intermediate query language, decoupling external languages designed to meet particular consumer needs from the idiosyncrasies of particular data models and database management systems. Buneman has shown that it can be implemented against relational and CODASYL datasbases, and that, by utilising the technique known as lazy evaluation, the implementations can be efficient.

The following chapter by Atkinson and Kulkarni has already been mentioned. But it is placed here as it is considered another experiment in the search for a consistent and simple way of merging programming languages and databases. There is still much research to be done; for example, no language yet gives satisfactory concurrent access to data of any type and any persistence.

Thus the book takes the reader from an up-to-date report on current practice in using and building databases, through the topics which are well established but not yet widely applied, to an identification of research problems it is hoped will challenge some of its readers.