E. J. Yannakoudakis and C. P. Cheng

Standard Relational and Network Database Languages

关系和网状数据库的标准语言 [英]

Springer-Verlag
World Publishing Corp

E. J. Yannakoudakis and C. P. Cheng

Standard Relational and Network Database Languages

With 20 Figures

Springer-Verlag
World Publishing Corp

E. J. Yannakoudakis, BSc, PhD, CEng, FBCS
Postgraduate School of Computer Sciences, University of Bradford,
Bradford, West Yorkshire BD7 1DP, UK

C. P. Cheng, BSc, MSc, PhD, MBCS
Department of Mathematical Studies, Hong Kong Polytechnic,
Kowloon, Hong Kong

ISBN 3-540-19537-8 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-19537-8 Springer-Verlag New York Berlin Heidelberg

British Library Cataloguing in Publication Data
Yannakoudakis, E.J., *1950* -
Standard relational and network database
languages.
1. Machine - readable files. Software
I. Title II. Cheng, C.P., *1947*
005.74
ISBN 3-540-19537-8

Library of Congress Cataloging-in-Publication Data
Yannakoudakis, E.J., 1950-
Standard relational and network database languages / E.J.
Yannakoudakis and C.P. Cheng.
p. cm.
Bibliography: p.
Includes index.
ISBN 0-387-19537-8 (U.S.)
1. Data base management. 2. Programming languages (Electronic
computers) I. Cheng, C.P. II. Title.
QA76.9.D3Y365 1988
005.74--dc 19 88-31118

Reprinted by World Publishing Corporation, Beijing, 1990
for distribution and sale in The People's Republic of China only
ISBN 7 - 5062 - 0750 - 8

Preface

For any type of software to become standard, whether a third generation language or an integrated project support environment (IPSE), it must undergo a series of modifications and updates which are a direct result of theoretical and empirical knowledge gained in the process. The database approach to the design of general purpose information systems has undergone a series of revisions during the last twenty years which have established it as a winner in many different spheres of information processing, including expert systems and real-time control.

It is now widely recognised by academics and practitioners alike, that the use of a database management system (DBMS) as the underlying software tool for the development of information/knowledge based systems can lead to environments which are: (a) flexible, (b) efficient, (c) user-friendly, (d) free from duplication, and (e) fully controllable.

The concept of a DBMS is now mature and has produced the software necessary to design the actual database holding the data. The database languages proposed recently by the International Organisation for Standardisation (ISO) are thorough enough for the design of the necessary software compilers (i.e programs which translate the high level commands into machine language for fast execution by the computer hardware).

The ISO languages adopt two basic models of data and therefore two different sets of commands: (a) the relational, implemented via the relational database language (RDL), and (b) the network, implemented via the network database language (NDL).

RDL is based on an IBM product called structured query language (SQL), whereas NDL is an extension of the previous proposal for a simple network language by the CODASYL (Conference On DAta SYstems and Languages) committee. So, the maturity of CODASYL (originally proposed in 1971) coupled with the theoretical foundations of the relational model make RDL and NDL very good candidates for the design of database compilers.

This book describes both RDL and NDL, details the syntax of their respective commands, and gives realistic examples to illustrate their

use. In this sense, it is a textbook which will be of use to both students and practitioners of the database technology. In summary, the book is intended for the student taking courses (either at undergraduate or postgraduate level) on computer science, the database administrator, the database programmer who will use the commands described to develop application programs, and finally the database analyst who collects the enterprise data and proceeds to outline the logic of each application for implementation with a database language.

The book is in three parts. Part I describes the database management system as it should be, that is, the various facilities it should offer the database administrator, the programmer and the general user. Part II describes the ISO RDL including the schema and view definition language, the module language, and the data manipulation language (otherwise known as structured query language or SQL). Part III describes the ISO NDL including the commands to create the schema and the subschema, the module language and the data manipulation language.

Acknowledgements

The specification of the two ISO database languages has been a long procedure involving hundreds of specialists from all over the world. We extend our thanks to all the people involved in this important task, particularly the members of the Database Committee of the British Standards Institute (BSI) and the members of the X3H2 Technical Committee on Databases of the American National Standards Institute (ANSI).

May 1988

E J Yannakoudakis
C P Cheng

Contents

Part I
THE DATABASE ENVIRONMENT

1 Database Management Systems

1.1 Introduction

Computer-based information systems which make use of a database management system (DBMS) evolve around the concepts of field, aggregate fields, record type, and file.

(a) *Field*: The smallest unit of data which is meaningful and can represent a real-world object (e.g SALARY, NAME, JOB-TITLE). A field can be atomic, in which case it cannot be decomposed into other subfields without losing semantic information; an example of a decomposition which may be meaningless under certain applications is the integer and the decimal parts of field SALARY. Or it can be decomposable into discrete components as is the case with field NAME, which can be split into the subfields INITIALS and SURNAME. Alternative terms for field are data item and attribute.

(b) *Aggregate field*: This is a named group of fields forming a discrete structure which represents a real-world object (e.g ADDRESS, comprising the fields STREET-NO, STREET-NAME, CITY and POST-CODE).

(c) *Record type*: A collection of fields forming an inter-record structure which constitutes a logical *entity*. An example record type called STAFF is presented in Figure 1.1, comprising the fields NAME, ADDRESS, JOB-TITLE, and the inter-link component which is used to associate each member of staff

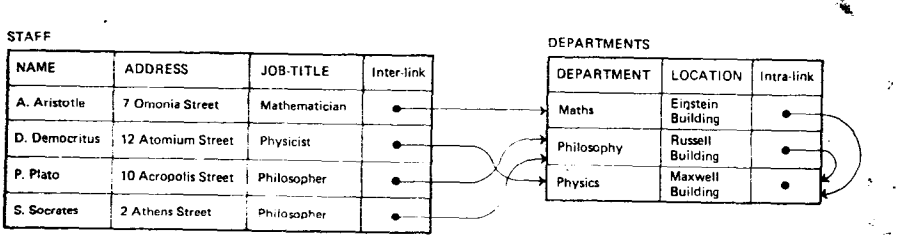

Figure 1.1 Intra- and inter-record structures.

with a department. The intra-link component of record type DEPARTMENT is used to link departments which run joint courses (a maximum of one link per record occurrence in both files STAFF and DEPARTMENT).

(d) *File*: A collection of occurrences under the same record type. The occurrence of a value under a field conforms with the specification of a single *domain* (a pre-specified set of values which can be integer, real, character, etc). Example files are presented in Figure 1.1, where a null link value signifies the end of a chain (relationship).

Clearly, the use of linked structures which are implemented on some language or other can involve one, two, or more links per record occurrence, offering alternative access paths to cluster logically related records.

Ideally, a set of files should be available for a variety of users and applications within an organisation, in such a way as to minimise redundancy (i.e duplicate record/field occurrences), while maintaining: (a) access flexibility, (b) data shareability, (c) data integrity, (d) security, and (e) performance and efficiency.

We are now in a position to define the terms *DBMS* and *database* [Yannakoudakis, 1988] as follows:

> A database is a collection of well-organised records within a commonly available mass storage medium. It serves one or more applications in an optimal fashion by allowing a common and controlled approach to adding, modifying and retrieving sets of data. The DBMS is a suite of computer programs which perform these operations in a standardised and fully controllable manner.

Moreover, a DBMS offers the facility to define file control data (e.g number of fields, type of each field, number of records, etc) separate from the logic of applications, ensuring in effect *data independence*. The latter concept is of the utmost importance in a database environment and can imply:

(a) Data on the devices (e.g disks) can be manipulated independent of the logic of the applications which access it. This is referred to as *storage independence.*

(b) The view an application has of its data can be altered without affecting the stored values. This is referred to as *logical independence.*

The set of very high level commands available to both users and programmers alike, make some of the more complex conceptual operations very easy to implement under the umbrella of a DBMS. (High level commands, beyond those available in current third generation programming languages (3GL) such as Pascal, Ada, C, and COBOL, are collectively referred to as fourth generation languages (4GL).)

The comprehensive environment offered by the DBMS for the speedy development of applications regardless of the type of data they process (e.g text, numeric, graphical, image), makes it a very attractive proposition to data processing managers of today.

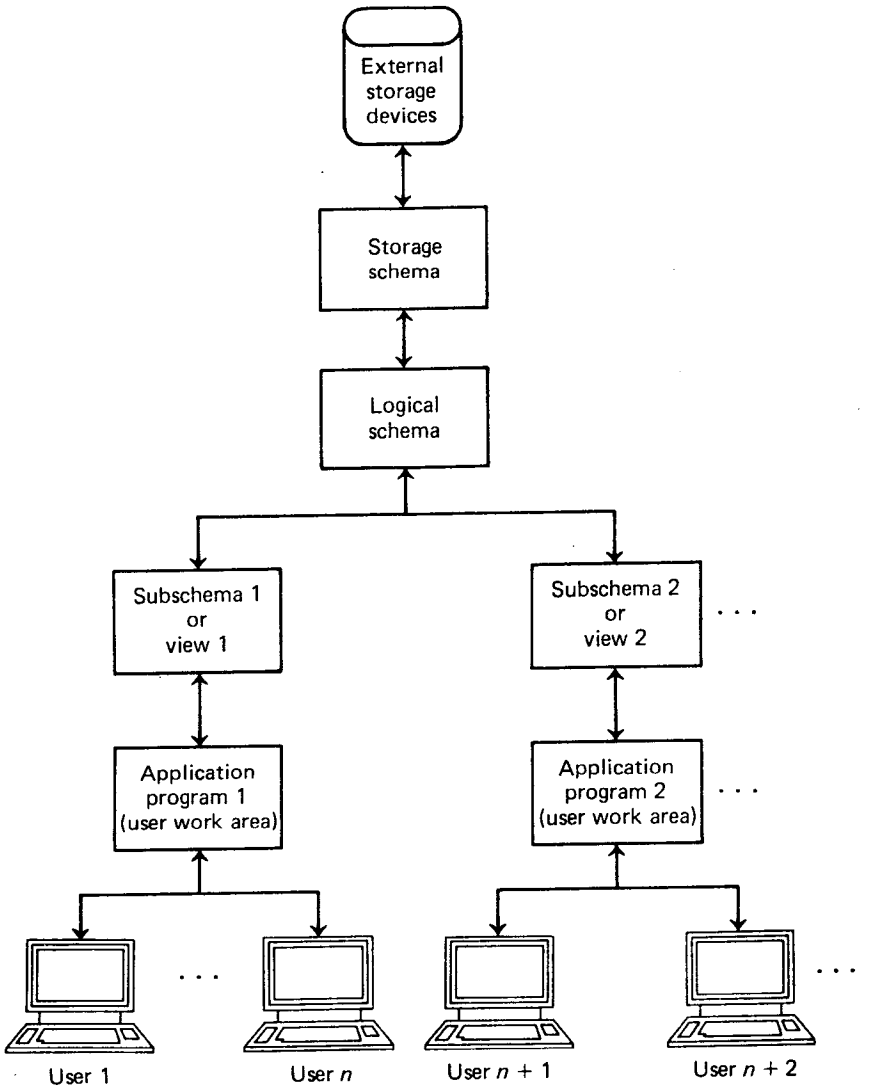

Figure 1.2 Major components of a database.

1.2 The Three Architectural Levels

With traditional file-based information systems data is stored under predefined record types which can be linked at the intra- or inter-level, that is, within and between files. The example file structure presented in Figure 1.1 illustrates both intra- and inter-fields which have to be defined explicitly within 3GLs, but not necessarily with a 4GL.

The definition of intra- and inter-links within 3GL programs is not particularly difficult, provided the complete file(s) can be stored on primary memory. If the files are too large to be held in the primary memory, then secondary memory (e.g a disk) is also used. One way of utilising both primary and secondary storage is to apply virtual storage techniques, joining in effect the available storage slots and creating a 'contiguous' block.

If virtual memory cannot be utilised, either because the operating system cannot handle it, or because the files are too large for the available virtual storage, then the definition of intra- and inter-links within 3GL programs can be problematic. This is due to the fact that a generalised record storage and access mechanism must be able to cope with two different types of pointers (addresses): (a) memory, and (b) secondary storage addresses.

If both storage independence and logical independence are to be maintained by the DBMS, then it must be able to cope with all possible manipulative operations upon the stored data as well as upon the views users have of it. It should also be able to cope with manipulative operations that ultimately result in the creation, deletion and updating of intra- and inter-links.

To this end, it becomes necessary to introduce a three-level architecture involving: (a) the schema, (b) the subschema, and (c) the storage schema, the totality of which can maintain data independence while offering all the advantages we discussed in Section 1.1. Figure 1.2 presents an outline of this architecture which can be studied in connection with the following.

1.2.1 Logical Schema

This refers to the logical structure of the database and aims to define all the entities (record types), and their relationships.

To understand what the source logical schema can possibly contain, compare it with that portion of a 3GL program which defines the record types in terms of their name, fields, type of each field, length of each field (in bits, bytes, etc), and the way they are linked together (either explicitly or implicitly). For example, compare the logical schema with the DATA DIVISION of a COBOL program.

So, the logical schema allows the definition of data independent of the logic which manipulates it. Data formats, in general, are defined once and the fact that the schema contains the entire set of fields makes it easier to detect synonyms, homonyms and apparent definitional duplication. (Application programs which

do not operate on a database must, by definition, contain a complete set of commands - possibly duplicated in other non-database programs - which describe fully each and every field they use.)

Unless otherwise stated, when we refer to a 'schema' we will imply a logical schema.

1.2.2 Logical Subschema

Given that the logical schema contains the definition of the entire database while individual applications usually require only a subset of this, it becomes necessary to specify the exact portion(s) of the logical schema which can be accessed by users and programs. The logical structure and corresponding data (including definitions and actual occurrences) of this subset is known as a *logical subschema.* (We frequently refer to this as subschema.)

So, a subschema corresponds to the view of an application and its users. If the DBMS forces each and every application program to *invoke* (reference) a subschema before it can compile successfully, then the subschema will also be introducing, to a certain extent, some form of data security since only the data (record types, field, etc) included in the subschema can actually be processed by the application program.

Clearly, in an organisation, there are as many subschemata (plural for subschema) as there are applications, although a subschema may be common to two or more different applications.

1.2.3 Storage Schema

After the logical schema has been defined the database designer considers the manipulative operations implied by the applications and proceeds to map and represent the logical schema on physical data structures which are referred to, collectively, as the *storage schema.* (Alternative equivalent terms are internal schema and physical schema.)

The storage schema comprises the definition of a set of integrated files, including flexible access paths, indexes, buffer sizes, blocking, device areas, etc which become ready (are initialised) to accept, retrieve, and generally maintain the actual values of logical schema fields.

When the database is first set up usage frequency statistics may not always be available, unless the enterprise is converting from a file-based system to a DBMS and past statistical data is available. So, the only sure way to proceed with the design of the storage schema is in an incremental and iterative manner, by collecting statistics on the usage of fields, access paths (links), etc, and subsequently using this information in order to tune the storage structures which ultimately affect the performance of the database.

1.3 Database Models

It is the type of data model and underlying data structures, which are supported by a DBMS, that will ultimately determine the viability and flexibility of accessing the various attributes of a given organisation. The types of data structures supported will also dictate the features of the data storage description language itself.

Before the database designer adopts a model or a data structure, it becomes necessary to understand data at a very much higher level - the conceptual - which is independent of logical and storage schemata. The conceptual model of an organisation is established following detailed data analyses and functional analyses, application by application. After the conceptual model has been established, the logical model can be designed by utilising only the information contained within the conceptual model.

In this sense, a *data model* represents and reflects accurately the relationships that exist among a set of record types, data items or fields. There are three major types of models where record types and their relationships may be defined as far as the logical schema is concerned [Yannakoudakis, 1988]:

(a) Hierarchic (or tree)
(b) Network
(c) Relational

Each of these corresponds to a different approach to viewing an integrated set of record types at the logical schema level, irrespective of any underlying data structures at the storage schema level.

Under the hierarchic model each record occurrence has no more than one 'parent' record occurrence. Under the network model a record occurrence may have more than one 'parent' record occurrence associated with it. Under the relational model the presence of common attributes (keys) among record types forms the basis of binding record types together; here, the relationships are implicit rather than explicit as is the case with both hierarchic and network models. (The models and their corresponding languages we discuss in this book are the relational and the network.)

1.3.1 Hierarchic Model

This model can only handle one-to-one and one-to-many relationships (e.g one department many employees, many courses many departments: see Figure 1.1). The relationship many-to-one is not allowed in a hierarchy.

Generally, an element in a hierarchy can be thought of as a distinct record type which may 'own' one or more other distinct record types (the *members*). The *owner* and the *members* then form what is frequently referred to as a *set type*.

The hierarchic model binds record types together so that occurrences under a

given level can be used to retrieve others directly below it. If a record occurrence is not directly below another, then it cannot be accessed directly; if direct retrieval is necessary then the occurrence is frequently duplicated under the 'parent' occurrence.

1.3.2 Network Model

This type of model allows more than one *owner* (parent) per record type. In other words, a record occurrence can be owned by two or more other record occurrences. This helps to reduce the redundancy introduced by the hierarchic model, by defining multiple incoming pointers to each record occurrence.

Although the network model allows a variable number of incoming pointers to a record occurrence, it is customary, in practice, to use the same number of 'links' per record occurrence, since variable length records are difficult to represent at the storage schema (i.e physically). While the structure is utilised and where extra links become necessary, then a tag within each occurrence may be used to indicate the presence of an overflow area which holds the rest of the links. The overflow area itself can have a fixed number of links per occurrence.

Networks sometimes give rise to *loops* or cycles where an occurrence appears to be linked to itself. This category of link occurs when a record type is defined as member and owner of the same single record type set.

If the DBMS cannot support a network model of data directly (i.e many-to-many relationships), then the network can be split into a number of one-to-many relationships which form an equivalent logical model. This becomes in effect a hierarchic model with owners and members which may be duplicated.

1.3.3 Relational Model

The relational model uses keys (primary and secondary) to form relationships among record types which are referred to as *relations*. In other words, the relationships are established through keys which are common between relations; the relationships are implicit and independent of physical implementation at the internal schema.

Each relation has associated with it a table which can be permanent and is otherwise referred to as a *base table*. The occurrences (rows) in a table are known as *tuples*. A *virtual table* is derived (synthesised) upon invocation of the corresponding virtual relation commonly known as a *view*.

The synthesis of views from one or more base tables can lead to certain difficulties when it comes to updating, inserting, or deleting occurrences in the virtual table. For example, given the following relations, based on the information presented in Figure 1.1

```
STAFF(KEY, NAME, ADDRESS, JOB_TITLE, DEPT)
DEPARTMENT(DEPT_NAME, LOCATION)
```