Theory and Practice of Relational Databases

second edition

Stefan Stanczyk

Bob Champion

Richard Leyton

Theory and Practice of Relational Databases

Second edition

Stefan Stanczyk
School of Computing and Mathematical Sciences
Oxford Brookes University, Oxford, UK

Bob Champion

Richard Leyton

London and New York

First published 1990 by Pitman Publishing
Second impression published 1993 by UCL Press
This edition published 2001 by Taylor & Francis
11 New Fetter Lane, London EC4P 4EE

Simultaneously published in the USA and Canada
by Taylor & Francis Inc.,
29 West 35th Street, New York, NY 10001

Taylor & Francis is an imprint of the Taylor & Francis Group

Printed and bound in Great Britain by
Biddles Ltd, Guildford and King's Lynn

Publisher's Note

This book has been produced from camera-ready copy supplied by the editors.

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Cataloging in Publication Data
Stanczyk, Stefan, 1945-
Theory and practice of relational databases/ Stefan Stanczyk. -- 2nd ed.
p.cm
Includes biographical references and index.
ISBN 0-415-24701-2 (cloth : alk. paper) -- ISBN 0-415-24702-0 (pbk. : alk. paper)
1. Relational databases. 2. Database management. I. Title

QA 76.9.D3 S698 2001
005.75'6--dc21 2001027206

ISBN 0-415-24702-0
ISBN 0-415-24701-2

Theory and Practice of Relational Databases

Preface

First published a decade ago, the ***Theory and Practice of Relational Databases*** gained fairly noticeable popularity, particularly amongst those readers to whom it was primarily addressed - the students. After a decade, however, any book needs reviewing, for the field will have developed, presentation could be improved, choice of topics might be reflected upon and, importantly, the comments from the readers addressed.

Databases evolved into a classic component of computing degrees. The subject became well supported by a wealth of research, exceptional industrial experience and numerous books covering a wide range of topics. However, books on databases run into voluminous proportions and tend to cover the whole spectrum of the subject thus constituting a monographic source of reference rather than being a learning aide.

The book we are presenting now is meant to be just that - a tutorial text that assists the process of learning. It is supposed to have a technological bias, to present the chosen topics in a concise manner, and to incite better understanding through explanations and illustrative examples. In short, the book is meant to retain those features that made the previous edition successful.

Naturally, the book does not aspire to cover all aspects of databases nor does it pretend to present the relational theory in its entirety. The focus is on a coherent, systematic coverage of database design. The primary objective of this book is to present a reasonably comprehensive explanation of the process of the development of database application systems within the framework of the set processing paradigm.

Amongst the rich variety of data models advocated by their authors from time to time, the relational approach has prevailed. However, for applications that require

processing of complex data structures the relational approach may not necessarily be advantageous. Application software built around the relational DBMS may require user-defined, complex data structures, appropriate to the domain of that application. Furthermore, certain types of applications do not naturally lend themselves to the relational paradigm. Thus we chose, and not without discussion and controversies, to include a separate chapter covering the object-oriented paradigm as applicable to databases.

Other than that, how different is this edition from the previous one? Well, we have noticed that learning Relational Calculus distracted many students from appreciating the principles of the relational model rather than contributing to its deeper understanding. Relational calculus, being isomorphic to algebra, does not greatly enhance the model *per se*; rather it constitutes an alternative view on its operational part. Accordingly, the chapter on calculus was removed and the exposition of relational algebra strengthened to include the aspects of query optimization and algorithms for algebraic operations.

Furthermore, we have taken the view that relational algebra , being a programming language, deserves a proper software support. Thus LEAP, a straightforward relational DBMS, written by Richard Leyton as an open-source project, has been presented in a separate chapter. The system, equipped with an algebraic interface, can be downloaded from the book's website and used freely by the readers. This brought about an additional benefit - the readers are given an opportunity to study some internal mechanisms of DBMS by inspecting, and possibly experimenting with, the source code - an experience that cannot easily be attempted with any commercially available DBMS.

SQL - by now firmly established as **the** database language - received in this present edition a far more prominent exposition. A reasonably comprehensive description and explanation of the language (largely based on *Programming in SQL*, Pitman 1993) with particular attention to its set-theoretical pedigree is presented; developing more complex SQL programs out of primitive expressions follows. Secondly, an overview of a programming extension is given using as an example PL/SQL - the system developed by ORACLE to support, amongst others, user-defined data types, conditional and looping structures, exception handling, functions, procedures, triggers and packages. Importantly, we have constrained the presentation of SQL to its existing software support; hence no details of constructs specific to say SQL 3 are considered.

The case study gathers all relevant solutions analyzed throughout the book and

culminates in a fairly extensive model for a University Information System, partly implemented in **mySQL** and using **Apache** to illustrate the problems and solutions of developing an interface between a database system and the Internet. The choice of software tools was dictated by their free availability, their technical merits and a reasonably high popularity amongst the database developers.

Finally, the book was given a semi-interactive companion, that is its own Internet site **www.theorypractice.org** with references to books, software, discussion groups, research centres and software producers.

In the context of the above amendments, the structure of the book is now as follows:

Chapter 1	Database approach to information systems, the generic 3-level database architecture, characteristics of various database models.
Chapter 2	Entity-Attribute-Relationship as a technology-independent notation for logical data modelling.
Chapter 3	The relational data model introduced as a formal system composed of the structural, behavioural and operational parts. Properties of the model. Relational representation of EAR.
Chapter 4	Relational Algebra and its use for data manipulation. Implementation and optimization of algebraic expressions.
Chapter 5	Description of a didactic DBMS based on relational algebra. Structure, operations and an example of running the system.
Chapter 6	Functional dependencies, Boyce Codd Normal Form. Normalization as mechanism for optimizing the relational structures.
Chapter 7	Multivalued and join dependencies and the relevant normal forms. Selected theoretical aspects of normalization.
Chapter 8	Structured Query Language. Data definition and manipulation. Specifying and programming update transactions.
Chapter 9	Object-orientation as a means for developing databases with inherently complex structures. Object relational database model.
Chapter 10	Procedural extensions to SQL with particular attention to non-relational retrievals and update transactions.
Chapter 11	Case study

We gratefully acknowledge the comments from our colleagues and students, past and present, in the School of Computing and Mathematical Sciences of Oxford Brookes University.

We feel greatly indebted to Dilys Alam and Grant Soanes, our editors at Taylor & Francis. That this book has appeared is a credit to their patience, support and encouragement. We are also very grateful for Alison Nick's sterling work as our Project Manager.

Finally, our thanks to Urszula, Lucille and Frances for them being more than forgiving of our preoccupation with writing this book.

Oxford, June 2001

Contents

Preface		**vii**
1.	**Introduction**	**1**
	1.1 The concept of database	1
	1.2 Database architecture	5
	1.3 Logical database models	8
2.	**Data modelling**	**11**
	2.1 Modelling the real world	11
	2.2 Entity - Attribute – Relationship modelling	13
	2.3 Exercises	26
3.	**The relational model**	**30**
	3.1 Fundamental concepts	30
	3.2 Normalized relations	36
	3.3 Integrity constraints	40
	3.4 Representation of EAR models by relations	44
	3.5 Exercises	52
4.	**Relational algebra**	**53**
	4.1 Processes and their abstractions	53
	4.2 Primitive retrieval operations	56
	4.3 Queries as compound algebraic expressions	71
	4.4 Optimization of algebraic expressions	75
	4.5 Exercises	85
5.	**LEAP - the algebraic DBMS**	**87**
	5.1 Introduction	87
	5.2 Leap architecture	88
	5.3 A sample run of LEAP	97
	5.4 Exercises	100
6.	**Normalization**	**101**
	6.1 Designing relations	101
	6.2 Functional dependency. BCNF normalization	103
	6.3 Exercises	120

7. Further normalization **121**
7.1 Multivalued dependency. Fourth normal form 121
7.2 Join dependency. Fifth normal form 125
7.3 Axioms of dependency theory 131
7.4 Transitive dependencies. Third normal form 136
7.5 Exercises 143

8. Structured Query Language **144**
8.1 Introduction 144
8.2 Defining database objects 145
8.3 Querying the database 155
8.4 Modifying the data 168
8.5 Exercises 175

9. Object databases **176**
9.1 Rationale 176
9.2 The object-oriented paradigm 179
9.3 Modelling complex objects for databases 180
9.4 Data definition and manipulation 185
9.5 Object-relational databases 189

10. SQL extensions **194**
10.1 Introduction 194
10.2 Basic programming structures 195
10.3 Procedures, functions and triggers 203

11. Case study - implementation **205**
11.1 Introduction 205
11.2 Software installation 207
11.3 Implementation 208
11.4 Exercises 231

Appendix A: Solutions to exercises **233**

Appendix B: Denotations, logic, sets **243**
Denotations 243
Algebra of propositions 244
Set operations 245

Bibliography **247**

Index **251**

CHAPTER 1

Introduction

1.1 THE CONCEPT OF A DATABASE

Proper information support is of paramount importance for the management of an enterprise. The successful operation of a road network, a railway system, a bank, a production company or service providers depends on relevant, precise and up-to-date information. The relevant decisions, whether instantaneous (e.g. those taken in real-time production control) or long term (defining strategies or policies, for example), should be made on the basis of multiple facts and these must be properly aggregated, evaluated and analysed in some acceptable time.

Unless the enterprise is small, the task of management is usually divided into a number of coherent functions, such as research and development, planning, production, sales, etc. Each of these functions takes a specific view on the operation of the enterprise as a whole; all of them taken together aim to achieve the ultimate goal - prosperity of the company, successful running of a project, smooth operation of services, or whatever the objectives might be.

Although separately carried out, the management functions are not necessarily disconnected. On the contrary, they affect and influence one another. For instance, financial circumstances determine in some sense planning and production, and limit allocation of resources for research. Production, in turn, determines sales and provides some feedback for research programmes, and so on. Consequently, some decisions made within the scope of one function may overlap with other decisions in some other areas. Also, several managers may use the same data, perhaps differently perceived, aggregated or formatted.

In conventional data processing, each of the management functions is supported by a separate information system. These systems, which operate within some environment (computer hardware, specialized equipment for data collections,

specially trained operators), have their own 'private' files and their own 'private' processes developed in a programming language that is most suitable for a particular application.

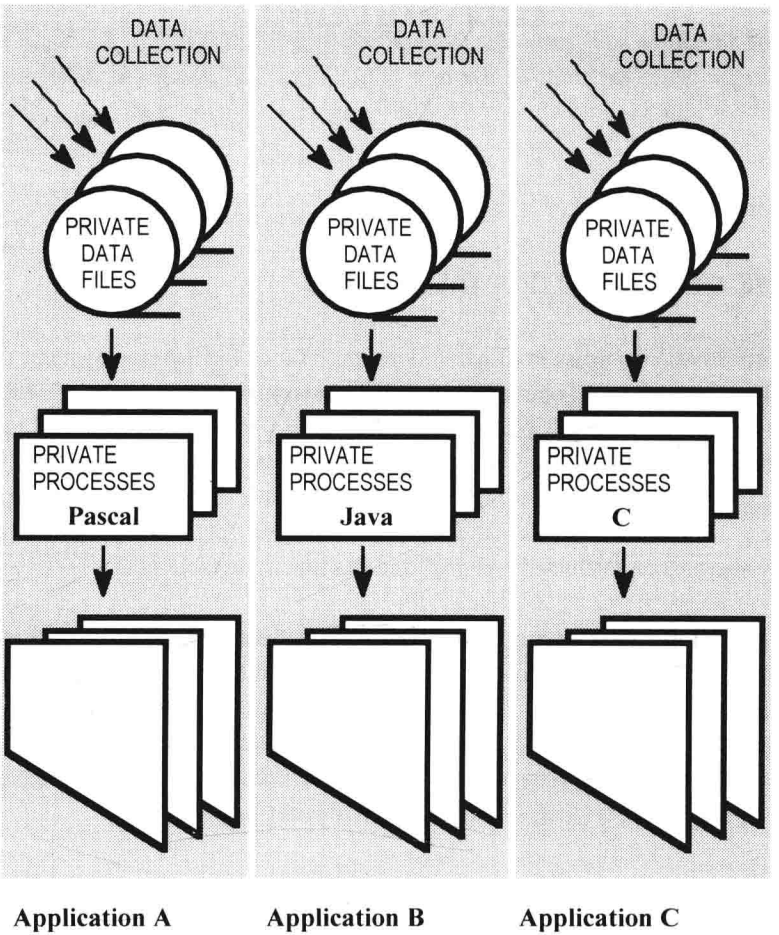

Fig. 1.1 Disjoint information systems

This situation is far from satisfactory. The most commonly appreciated reasons for this dissatisfaction are:

Redundancy of data

Several files contain the same data. The data is likely to be separately collected according to some specific procedure devised for each of the subsystems; a possible use of a sophisticated equipment for data collection makes the whole

process rather expensive. Moreover, the data duplicates are most certainly to be separately updated, thereby involving the risk of inconsistency.

Non-interchangeability of data
Suppose one of the applications is to be extended to incorporate some new functions requested by the users. To produce the required results, this application may need some new data that is not available in its own files but happens to be present in some other system's files. However, due to several reasons - different file organization, different formatting of data, idiosyncrasies of programming languages - the other system's files may not be directly accessible. Hence some additional (and in fact unnecessary) programming effort is needed to convert the relevant files into the form acceptable by the application in question.

Non-interchangeability of processes
Numerous routines are common for all of the applications (sorting, searching, organizing and processing data structures are the prime examples), yet they must be coded separately, according to the specific programming languages' requirements. Again, some waste of programming effort occurs.

Non-transparency of the application software
A considerable part of the application software handles purely data processing matters and this conceals the application logic rather than bringing it out. It is, then, rather difficult to reconstruct the application logic by reading the relevant code - these two types of information are expressed at completely different levels of abstraction.

Inflexibility of the application software
The application software (which essentially represents processes, not the data) contains some built-in knowledge about the data (such as data types and range of variables, for example). This knowledge is duplicated in every program that uses the relevant data and makes the global data consistency control difficult. Moreover, should these types, ranges etc. change (for whatever reason - extending a field size and incorporating a new field into a record may serve as a typical example) considerable reprogramming must necessarily be done throughout the whole application software.

Uncontrolled expansion
There is no mechanism to control in any systematic way a possible (and

likely) growth of both the data and the processes, neither is there any form to balance the conflicting requirements. Inevitably new data, new collection and updating procedures, and new processes will be added to the systems, thus making the system programming support and resource allocation increasingly difficult.

To summarize, the management of the enterprise is not supported by any coherent method for corporate control of the data. Yet the data is one of the enterprise's assets, just as valuable as human resources, buildings, machines and finances are. *The database approach to information systems provides the management of the enterprise with means to impose centralized control over its operational data.* This is the main advantage (and indeed, the objective) of having a database system implemented.

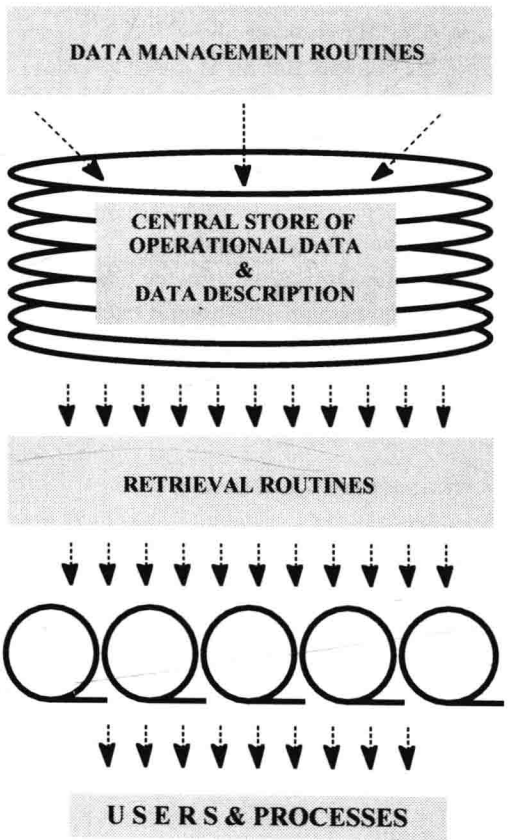

Fig. 1.2 A simplified database structure

A general concept of a database is depicted in Fig. 1.2. We shall give a detailed account of its structure shortly, but at this point we can view a database as a structured collection of operational data together with a description of that data.

The heart of the database system is then a central store of data - an integrated collection of records with any excessive redundancy eliminated (some duplication may occur for e.g. validation purposes). The data is shared among all the users of the system be they casual interrogators, application programmers (or programs themselves) or the *Database Administrator* (DBA).

The DBA (a team rather than a single person) can be thought of as a supreme controller who supervises every aspect of the database existence. In particular, the DBA is responsible for the database information content, the security and integrity of data, the storage structure and access strategy and for monitoring the performance of the database - making necessary adjustments whenever necessary.

All communication between the physical representation of the data and any user is done through the *Database Management System* (DBMS). This means that virtually every activity in the system (including defining and modifying database structures, inserting, deleting and updating values, and all kinds of retrievals) is controlled by the DBMS.

The DBMS contains a variety of facilities including a *data definition language* (to create and modify the database structures - files, users and their privileges), a *query language* (which supports all forms of retrieval and updating) and numerous interfaces to liaise with the operating system, telecommunication system, programming languages and other utility software. It also contains data validation routines and maintains a *Data Dictionary* - a complete description of the database structure and content.

1.2 DATABASE ARCHITECTURE

The database architecture whose brief account is the subject of this section was proposed by the ANSI/X3/SPARC group (Tsichritzis, 1978) in an attempt to provide a general framework for database systems, quite irrespective of their underlying data models (hierarchical, network or relational).

The database architecture (see Fig. 1.3) essentially comprises 3 levels - *conceptual*, *external* and *internal* - in an attempt to separate the logical and the

physical aspects of the system. The main idea is to provide a framework that makes it possible to consider the data separately from processing and to insulate the data from all implementational aspects, be they hardware constraints, or software facilities, or whatever.

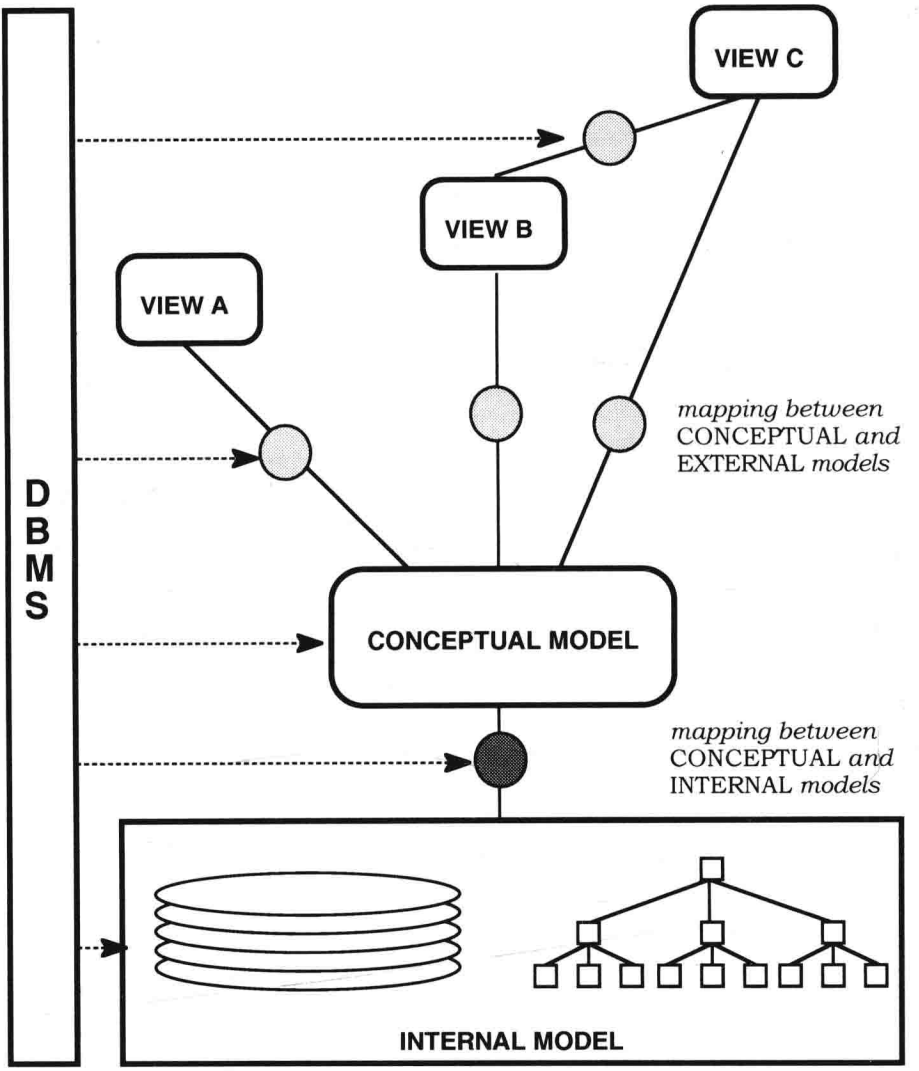

Fig. 1.3 The ANSI/X3/SPARC architecture for a database system

The **conceptual model** is a common, unconstrained view of the data. It is a

model that contains all the relevant (to the information system being developed) facts recorded in some suitable notation. At this point it is immaterial how this data is going to be processed or stored; all that counts is its relevance and truthfulness. The conceptual model is supposed to be a true image of the Real World as perceived by all parties concerned - the users and the developers alike.

Since all the data in the database is integrated, only a relatively small portion of it is of interest to a particular user . We call this portion of the data a **view**. There can be many separate or overlapping views according to the specific user's requirements. The views can be created or destroyed as circumstances dictate, hence the whole structure of views is dynamic.

The **internal model** represents the actual storage representation of all the data in the database.There is obviously just one internal model and it is closely connected to the actual software facilities provided by the computer system on which the database is implemented.

All the models are recorded (stored and kept up-to-date by the DBMS in both the source and the object form) in terms of a Data Sub-Language (DSL) as **schemas**. The *conceptual schema* comprises definitions of all the logical units of data together with their types, the logical relationships among them and the appropriate validation procedures. The conceptual schema does not address the questions of storage structure and access strategy in any way; although written in DSL it does not depend on any particular programming language.

Every view is described by means of an *external schema* (also stored by the DBMS). It contains descriptions of each of the various types of external records which are defined on conceptual records but not necessarily in a one-to-one correspondence. The *internal schema* (again stored by the DBMS) defines the structure of the internal records and contains information on possible indices, applicability of field values for hashing or indexing and similar properties or physical relationships.

The mappings CONCEPTUAL ↔ EXTERNAL and CONCEPTUAL ↔ INTERNAL (both of them stored by the DBMS, of course) ensure the database model coherence and facilitate data independence.

The notion of **data independence** is fundamental to the database theory. It gives the DBA the freedom of changing both the physical and the logical aspects of the database system without disturbing the applications built on the database.