

EXTRACT

(Chapter 1)

E. F. CODD

The
RELATIONAL
MODEL
for
DATABASE
MANAGEMENT

VERSION 2

The RELATIONAL MODEL for DATABASE MANAGEMENT: VERSION 2

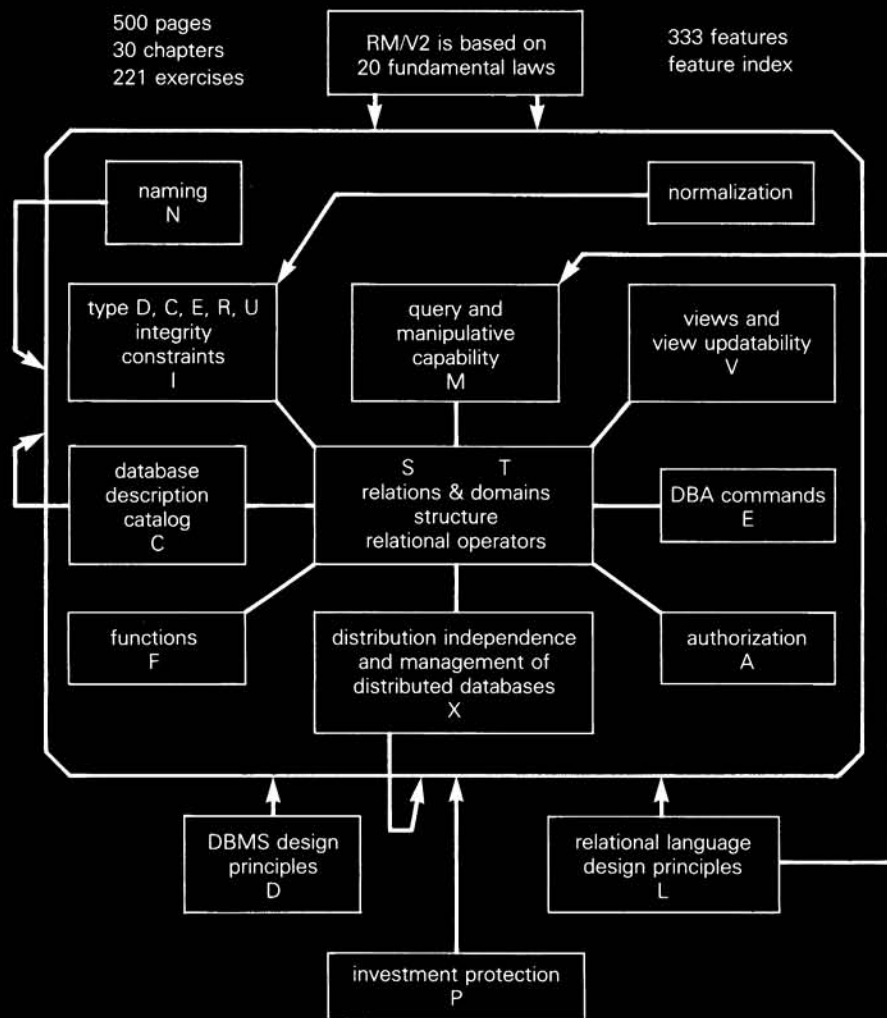


About the Author

Dr. Edgar F. Codd joined IBM in 1949 as a programming mathematician for the Selective Sequence Electronic Calculator. During the 1950s he participated in developing several important IBM products. Beginning in 1968, Dr. Codd turned his attention to the management of large commercial databases and developed the relational model as a foundation. Since the mid-1970s, Dr. Codd has been working persistently to encourage vendors to develop relational DBMSs and to educate users, DBMS vendors, and standards committees regarding the services such a DBMS should supply and why users need all these services.

In 1985, Dr. Codd established two lecturing and consulting companies in San Jose. These companies specialize in all aspects of relational database management, relational database design, and evaluation of products that are claimed to be relational.

Continued on back flap



333 features / 18 classes // average # of features per class = 18


classes	A	B	C	D	E	F	6	} 18 classes of features
	I	J		L	M	N	5	
	P	Q			S	T	4	
	V		X			Z	3	

E. F. CODD

The Relational Model

for Database Management

▪ ***Version 2*** ▪

 **ADDISON-WESLEY PUBLISHING COMPANY**
Reading, Massachusetts • Menlo Park, California • New York
Don Mills, Ontario • Wokingham, England • Amsterdam
Bonn • Sydney • Singapore • Tokyo • Madrid • San Juan

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial caps or all caps.

The programs and applications presented in this book have been included for their instructional value. They have been tested with care, but are not guaranteed for any particular purpose. The publisher does not offer any warranties or representations, nor does it accept any liabilities with respect to the programs or applications.

Library of Congress Cataloging-in-Publication Data

Codd, E. F.

The relational model for database management : version 2 / E.F. Codd.
p. cm.

Includes index.

ISBN 0-201-14192-2

1. Data base management. 2. Relational data bases. I. Title.

QA76.9.D3C626 1990

005.75'6—dc20

89-6793

CIP

Copyright © 1990 by Addison-Wesley Publishing Company, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America.

ABCDEFGHIJ-HA-943210

To fellow pilots and aircrew
in the Royal Air Force
during World War II

and the dons at Oxford.

These people were the source of my determination to fight for what I believed was right during the ten or more years in which government, industry, and commerce were strongly opposed to the relational approach to database management.

■ **PREFACE** ■

Today, if you have a well-designed database management system, you have the keys to the kingdom of data processing and decision support. That is why there now exists a prototype machine whose complete design is based on the relational model. Its arithmetic hardware is a quite minor part of the architecture. In fact, the old term “computer system” now seems like a misnomer.

My first paper dealing with the application of relations (in the mathematical sense) to database management was a non-confidential IBM research report made available to the general public that was entitled *Derivability, Redundancy, and Consistency of Relations stored in Large Data Banks* [Codd 1969]. I placed a great deal of emphasis then on the preservation of integrity in a commercial database, and I do so now. In this book, I devote Chapters 13 and 14 exclusively to that subject.

Another concern of mine has been, and continues to be, precision. A database management system (DBMS) is a reasonably complex system, even if unnecessary complexity is completely avoided. The relational model intentionally does not specify how a DBMS should be built, but it does specify what should be built, and for that it provides a precise specification.

An important adjunct to precision is a sound theoretical foundation. The relational model is solidly based on two parts of mathematics: first-order predicate logic and the theory of relations. This book, however, does not dwell on the theoretical foundations, but rather on all the features of the relational model that I now perceive as important for database users, and therefore for DBMS vendors. My perceptions result from 20 years of practical experience in computing and data processing (chiefly, but not exclusively, with large-scale customers of IBM), followed by another 20 years of research.

I believe that this is the first book to deal exclusively with the relational approach. It does, however, include design principles in Chapters 21 and 22. It is also the first book on the relational model by the originator of that model. All the ideas in the relational model described in this book are mine, except in cases where I explicitly credit someone else.

In developing the relational model, I have tried to follow Einstein's advice, "Make it as simple as possible, but no simpler." I believe that in the last clause he was discouraging the pursuit of simplicity to the extent of distorting reality. So why does the book contain 30 chapters and two appendixes? To answer this question, it is necessary to look at the history of research and development of the relational model.

From 1968 through 1988, I published more than 30 technical papers on the relational model [Codd 1968–Codd 1988d]. I refer to the total content of the pre-1979 papers as Version 1 of the relational model (RM/V1 for brevity).

Early in 1979, I presented a paper to the Australian Computer Society at Hobart, Tasmania, entitled "Extending the Database Relational Model to Capture More Meaning," naming the extended version RM/T (T for Tasmania). My paper on RM/T later appeared in *ACM Transactions on Database Systems* [Codd 1979]. My aim was for the extensions to be tried out first in the logical design of databases and subsequently to be incorporated in the design of DBMS products, but only if they proved effective in database design.

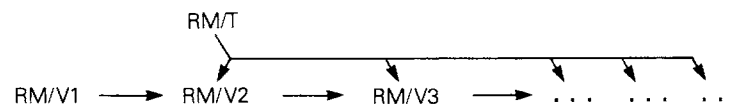
Progress in this direction has been much slower than I expected. Vendors of DBMS products have in many cases failed to understand the first version RM/V1, let alone RM/T. One of the reasons they offer is that they cannot collect all the technical papers because they are dispersed in so many different journals and other publications.

This book collects in one document much of what has appeared in my technical papers, but with numerous new features, plus more detailed explanation (and some emphasis) on those features of RM/V1 and RM/V2 that capture some aspects of the meaning of the data. This emphasis is intended to counter the numerous allegations that the relational model is devoid of semantics. I also hope that this document will challenge vendors to get the job done.

Figure P.1 is intended to show how features of RM/T are expected to be gradually dropped into the sequence of versions RM/V2, RM/V3, The dropping will be gradual to allow DBMS vendors and consumers time to understand them.

RM/V2 consists of 333 features. A few of these features are of a proscriptive nature, which may sound surprising or inappropriate. However,

Figure P.1 Relationship between the Various Versions of the Relational Model



they are intended to improve the understanding of the relational model and help DBMS vendors avoid extensions that at first glance seem quite harmless, but later turn out to block extensions needed to advance the DBMS from a primitive to a basic status. The most famous example of a proscriptive feature in the computing field was Dijkstra's assertion that new programming languages should exclude the GO TO command.

The features of RM/V2 include all of the features of RM/V1, roughly 50 of them. Thus, this book covers both versions of the relational model. However, except for some of the advanced operators in Chapter 5, there is no sharp boundary between RM/V1 and RM/V2. This is partly due to changes in some of the definitions to make them more general—for example, entity integrity and referential integrity. Incidentally, the new definitions [Codd 1988a] were available to DBMS vendors well before their first attempts to implement referential integrity.

Domains, primary keys, and foreign keys are based on the meaning of the data. These features are quite inexpensive to implement properly, do not adversely affect performance, and are extremely important for users. However, most DBMS vendors have failed to support them, and many lecturers and consultants in relational database management have failed to see their importance.

Most of the new ideas in RM/V2 have been published in scattered technical journals during the 1980s. What is different about this version of the relational model? Is all of RM/V1 retained?

Versions 1 and 2 are at the same high level of abstraction, a level that yields several advantages:

- independence of hardware support;
- independence of software support;
- occasionally, vendors can improve their implementations “under the covers” without damaging their customers’ investment in application programs, training of programmers, and training of end users.

A strong attempt has been made to incorporate all of RM/V1 into RM/V2, allowing programs developed to run on RM/V1 to continue to operate correctly on RM/V2. The most important additional features in RM/V2 are as follows:

- a new treatment of items of data missing because they represent properties that happen to be inapplicable to certain object instances—for example, the name of the spouse of an employee when that employee happens to be unmarried (Chapters 8 and 9);
- new features supporting all kinds of integrity constraints, especially the user-defined type (Chapter 14);
- a more detailed account of view updatability, which is very important for users but has been sadly neglected by DBMS vendors (Chapter 17);

- some relatively new principles of design applied to DBMS products and relational languages (Chapters 21 and 22);
- a more detailed account of what should be in the catalog (Chapter 15);
- new features pertaining to the management of distributed databases (Chapters 24 and 25);
- some of the fundamental laws on which the relational model is based (Chapter 29).

A few of the ideas in RM/T have been incorporated into RM/V2. Many, however, are being postponed to RM/V3 or later versions, because the industry has not been able to maintain an adequate pace of product development and improvement. Additionally, errors made in the design of DBMS products along the way are also hindering progress—often it is necessary to continue to support those errors in order to protect a customer's heavy investment in application programs.

In this book, I attempt to emphasize the numerous semantic features in the relational model. Many of these features were conceived when the model was first created. The semantic features include the following:

- domains, primary keys, and foreign keys;
- duplicate values are permitted within columns of a relation, *but duplicate rows are prohibited*;
- systematic handling of missing information independent of the type of datum that is missing.

These features and others go far beyond the capabilities of pre-relational DBMS products.

Except in Chapter 30, very little is said about models for database management other than the relational model. The relational model, invented in 1969, was the first model for database management. Since then, it has become popular to talk of many other kinds of data models, including a network data model, a hierarchical data model, a tabular data model, an entity-relationship model, a binary relationship model, and various semantic data models.

Historically, it has often been assumed that the hierarchic and network data models pre-dated not only the relational model, but also the availability of hierarchical and network DBMS products. Actually, judging by what has been published, no such models existed before the relational data model was invented or before non-relational DBMS products became available. With the sole exception of relational systems, database management system products existed before any data model was created for them.

The motivations for Version 2 of the relational model included the following five:

1. all of the motivations for Version 1;
2. the errors in implementing RM/V1, such as the following:
 - a. duplicate rows permitted by the language SQL;
 - b. primary keys have either been omitted altogether, or they have been made optional on base relations;
 - c. major omissions, especially of all features supporting the meaning of the data (including domains);
 - d. indexes misused to support semantic aspects;
 - e. omission of almost all the features concerned with preserving the integrity of the database.
3. the need to assemble all of the relational model in one document for DBMS vendors, users, and inventors of new data models who seem to be unaware of the scope of the relational model and the scope of database management;
4. the need for extensions to Version 1, such as the new kinds of joins, user-defined integrity, view updatability, and features that support the management of distributed databases;
5. the need for users to realize what they are missing in present relational DBMS products because only partial support of the relational model is built into these products.

In Appendix A, the features index, there is a specialized and comprehensive index to all of the RM/V2 features. This index should facilitate the cross-referencing that occurs in the description of several features. In addition to the exercises at the end of each chapter, simple exercises in predicate logic and the theory of relations appear in Appendix B. The reference section, in addition to full citations to the many papers and books cited in the text, includes a short bibliographical essay.

I have tried to keep the examples small in scale to facilitate understanding. However, small-scale examples often do not show many of the effects of the large scale of databases normally encountered.

Finally, I would like to acknowledge the encouragement and strong support provided by friends and colleagues, especially Sharon Weinberg, to whom I am deeply indebted. I also wish to thank the reviewers of my manuscript for their many helpful comments: Nagraj Alur, Nathan Goodman, Michel Melkanoff, Roberta Rousseau, Sharon Weinberg, and Gabrielle Wiorkowski.

I hope that all readers of this book—whether they are students, vendors, consultants, or users—find something of value herein.

E. F. Codd

Menlo Park, California

■ CHAPTER 1 ■

Introduction to Version 2 of the Relational Model

1.1 ■ What Is a Relation?

The word “relation” is used in English and other natural languages without concern for precise communication. Even in dictionaries that attempt to be precise, the definitions are quite loose, uneconomical, and ambiguous. *The Oxford English Dictionary* devotes a whole page of small print to the word “relation.” A small part of the description is as follows:

That feature or attribute of things which is involved in considering them in comparison or contrast with each other; the particular way in which one thing is thought of in connexion with another; any connexion, correspondence, or association, which can be conceived as naturally existing between things.

On the other hand, mathematicians are concerned with precise communication, a very high level of abstraction, and the economy of effort that stems from making definitions and theorems as general as possible. A special concern is that of avoiding the need for special treatment of special cases except when absolutely necessary. The generally accepted definition of a relation in mathematics is as follows:

Given sets S_1, S_2, \dots, S_n (not necessarily distinct), R is a relation on these n sets if it is a set of n -tuples, the first component of which is drawn from S_1 , the second component from S_2 , and so on.

2 ■ Introduction to Version 2 of the Relational Model

More concisely, R is a subset of the Cartesian product $S_1 \times S_2 \times \dots \times S_n$. (For more information, see Chapter 4.) Relation R is said to be of *degree* n . Each of the sets S_1, S_2, \dots, S_n on which one or more relations are defined is called a *domain*.

It is important to note that a mathematical relation is a set with special properties. First, all of its elements are tuples, all of the same type. Second, it is an unordered set. This is just what is needed for commercial databases, since many of the relations in such databases are each likely to have thousands of tuples, sometimes millions. In several recently developed databases, there are two thousand millions of tuples. In such circumstances, users should not be burdened with either the numbering or the ordering of tuples.

The relational model deals with tuples by their information content, not by means extraneous to the tuples such as tuple numbers, tuple identifiers, or storage addresses. The model also avoids burdening users with having to remember which tuples are next to which, in any sense of “nextness.”

As one consequence of adopting relations as the user’s perception of the way the data is organized, application programs become independent of any ordering of tuples in storage that might be in effect at some time. This enables the stored ordering of tuples to be changed whenever necessary without adversely affecting the correctness of application programs.

Changes in the stored ordering may have to be made for a variety of reasons. For example, the pattern of traffic on the database may change, and consequently the ordering previously adopted may no longer be the most suitable for obtaining good performance.

A mathematical relation has one property that some people consider counter-intuitive, and that does not appear to be consistent with the definition in *The Oxford English Dictionary*. This property is that a unary relation (degree one) can conform to this definition. Thus, a mathematical relation of degree greater than one does inter-relate two or more objects, while a mathematical relation of degree one does not. In some cases, intuition can be a poor guide. In any event, whether the concept of a unary relation is counter-intuitive or not, mathematicians and computer-oriented people do not like to treat it any differently from relations of higher degree.

In applying computers effectively (whether in science, engineering, education, or commerce) there is, or should be, a similar concern for precise communication, a high level of abstraction, and generality. If one is not careful, however, the degree of generality can sometimes be pursued beyond what is needed in practice, and this can have costly consequences.

A relation R in the relational model is very similar to its counterpart in mathematics. When conceived as a table, R has the following properties:

- each row represents a tuple of R ;
- the ordering of rows is immaterial;
- all rows are distinct from one another in content.

From time to time, objects are discussed that violate the last item listed, but that are mistakenly called relations by vendors of database management systems (abbreviated DBMS). In this book, such objects are called *improper relations* or *corrupted relations*. Reasons why improper relations should *not* be supported in any database management system are discussed in Chapter 23.

The fact that relations can be perceived as tables, and that tables are similar to flat files, breeds the false assumption that the freedom of action permitted [with] tables or flat files must also be permitted when manipulating relations. The manipulation differences are quite strong. For example, rows that entirely duplicate one another are not permitted in relations. The more disciplined approach of the relational model is largely justified because the database is shared by many people; in spite of the heavy traffic, all of the information in that database must be maintained in an accurate state.

The concept of a relation in the relational model is slightly more abstract than its counterpart in mathematics. Not only does the relation have a name, but each column has a distinct name, which often is not the same as the name of the pertinent set (the domain) from which that column draws its values. There are three main reasons for using a distinct column name:

1. such a name is intended to convey to users some aspect of the intended meaning of the column;
2. it enables users to avoid remembering positions of columns, as well as which component of a tuple is next to which in any sense of “nextness;”
3. it provides a simple means of distinguishing each column from its underlying domain. A column is, in fact, a particular use of a domain.

One reason for abandoning positional concepts altogether in the relations of the relational model is that it is not at all unusual to find database relations, each of which has as many as 50, 100, or even 150 columns. Users therefore are given an unnecessary burden if they must remember the ordering of columns and which column is next to which. Users are far more concerned with identifying columns by their names than by their positions, whether the positions be those in storage or those in some declaration. It makes much more sense for a user to request an employee’s date of birth by name than by what its position happens to be (for example, column # 37).

One reason for discussing relations in such detail is that there appears to be a serious misunderstanding in the computer field concerning relations. There is a widely held misconception that, for one collection *S* of data to be related to another collection *T*, there must exist a pointer or some kind of link from *S* to *T* that is exposed to users. A pointer to *T*, incidentally, has as its value the storage address of some key component of *T*. A recent article [Sweet 1988] shows that this false notion still exists.

Table 1.1 Relations in Mathematics Versus Relations in the Relational Model

Mathematics	Relational Model
Unconstrained values	Atomic values
Columns not named	Each column named
Columns distinguished from each other by position	Columns distinguished from each other and from domains by name
Normally constant	Normally varies with time

For many reasons, pointers are extremely weak in supporting relations. In fact, an individual pointer is capable of supporting no more than a relation of degree 2, and even then supports it in only one direction. Moreover, pointers tend to foster needlessly complex structures that frustrate interaction with the database by casual users, especially if they are not programmers. The slow acceptance of artificial intelligence (AI) programs has been largely due to the use of incredibly complex data structures in that field. This supports the contention that AI researchers write their programs solely for other AI researchers to comprehend.

It is therefore a basic rule in relational databases that there should be no pointers at all in the user's or programmer's perception. For implementation purposes, however, pointers can be used in a relational database management system "under the covers," which may in some cases allow the DBMS vendor to offer improved performance.¹

The term "relation" in mathematics means a fixed relation or constant, unless it is explicitly stated to be a variable. In the relational model exactly the reverse is true: every relation in the relational model is taken to be a variable unless otherwise stated. Normally it is the extension of relations (i.e., the tuples or rows) that is subject to change. Occasionally, however, new columns may be added and old columns dropped without changing the name of the relation.

The distinctions between the relation of mathematics and that of the relational model are summarized in Table 1.1.

A final note about relations: every tuple or row coupled with the name of the relation represents an assertion. For example, every row in the EMPLOYEE relation is an assertion that a specific person is an employee of the company and has the immediate single-valued properties cited in the row. Every row in the CAN_SUPPLY relation is an assertion that the cited supplier can supply the cited kind of part with the cited speed in the cited minimum package and at the cited cost. It is this general fact that makes relational databases highly compatible with knowledge bases.

1. Occasionally it is necessary in this book to discuss some features of database management that are at too low a level of abstraction to be included in the relational model. When this occurs, such features are said to be *under the covers* or *hidden from the user's view*.

In a reasonably complete approach to database management, it is not enough to describe the types of structure applied to data. In the recent past, numerous inventors have stopped at that point, omitting the operators that can be used in query and manipulative activities, data-description techniques, authorization techniques, and prevention of loss of integrity (see [Chen 1976] for an example). All of these capabilities should be designed into the DBMS from the start, not added afterwards. If a similar approach were adopted in medical science, all that would be taught is anatomy. Other important subjects such as physiology, neurology, and cardiology would be omitted. Database management has many facets in addition to the types of structure applied to data. Of key importance is the collection of operators that can be applied to the proposed types of data structure.

The relational model provides numerous operators that convert one or more relations into other relations; these are discussed in Chapters 4 and 5. Very few of these operators were conceived by mathematicians before the relational model was invented. One probable reason for this was the widely held belief that any problem expressed in terms of relations of arbitrary degree can be reduced to an equivalent problem expressed in terms of relations of degree one and two. My work on normalizing relations of assorted degrees shows this belief to be false.

1.2 ■ The Relational Model

A database can be of two major types: production-oriented or exploratory. In commerce, industry, government, or educational institutions, a production-oriented database is intended to convey at all times the state of part or all of the activity in the enterprise.

An exploratory database, on the other hand, is intended to explore possibilities (usually in the future) and to plan possible future activities. Thus, a production-oriented database is intended to reflect reality, while an exploratory database is intended to represent what might be or what might happen. In both cases the accuracy, consistency, and integrity of the data are extremely important.

Database management involves the sharing of large quantities of data by many users—who, for the most part, conceive their actions on the data independently from one another. The opportunities for users to damage data shared in this way are enormous, unless all users, whether knowledgeable about programming or not, abide by a discipline.

The very idea of a discipline, however, is abhorrent to many people, and I understand why. For example, I have encountered those who oppose a special feature of the relational model, namely, the prohibition of duplicate rows within all relations. They declare, “Why shouldn’t I have duplicate rows if I want them? I am simply not prepared to give up my freedom in this regard.” My response is as follows. If the data were a purely private concern (to just this single user), it would not matter. If, on the other hand, the data is shared or is likely to be shared sometime in the future, then *all*

of the users of this data would have to agree on what it means for a row to be duplicated (perhaps many times over). In other words, the sharing of data requires the sharing of its meaning. In turn, the sharing of meaning requires that there exist a single, simple, and explicit description of the meaning of every row in every relation. This is necessary even though one user may attach more importance to some facet of the meaning than some other user does.

Returning to the questionable support of duplicate rows, if the DBMS supports duplicate rows or records, it must be designed to handle these duplicates in a uniform way. Thus, there must be a general consensus among *all of the users of a DBMS product* regarding the meaning of duplicate rows, and this meaning should *not* be context-sensitive (i.e., it should not vary from relation to relation). My observation is that no such consensus exists, and is not likely to exist in the future.

For this reason and others, the discipline needed for the successful sharing of important data should be embodied within the database management system. The relational model can be construed as a highly disciplined approach to database management. Adherence to this discipline by users is enforced by the DBMS provided that this system is based whole-heartedly on the relational model.

As a normal mode of operation, if a user wishes to interpret the data in a database differently from the shared meaning, the DBMS should permit that user to extract a copy of the data from the database for this purpose (provided that the user is suitably authorized), and should disallow re-entry of that data into the database.

The management of shared data presents significantly tougher problems than the management of private data. Also, the role of shared data in efficiently carrying out business and government work is rapidly becoming a central concern. These two facts strongly suggest that no compromises be made on the quality of systems that manage the sharing of data simply to support a small minority of users of private data.

1.2.1 Relation as the Only Compound Data Type

From a database perspective, data can be classified into two types: atomic and compound. Atomic data *cannot* be decomposed into smaller pieces by the DBMS (excluding certain special functions). Compound data, consisting of structured combinations of atomic data, *can* be decomposed by the DBMS.

In the relational model there is only one type of compound data: the relation. The values in the domains on which each relation is defined are required to be atomic with respect to the DBMS. A relational database is a collection of relations of assorted degrees. All of the query and manipulative operators are upon relations, and all of them generate relations as results. Why focus on just one type of compound data? The main reason is that any additional types of compound data add complexity without adding power.

This is particularly true of the query and manipulative language. In such a language it is essential to have at least four commands: **retrieve**, **insert**, **update**, and **delete**. If there are N distinct types of compound data, then for these four operations $4N$ commands will be necessary. By choosing a single compound data type that, by itself, is adequate for database management, the smallest value (one) is being selected for N .

In non-relational approaches to database management, there was a growing tendency to expose more and more distinct types of compound data. Consequently, the query and manipulative languages were becoming more and more complicated, and at the same time significantly less comprehensible to users, even those who were knowledgeable about programming.

Relational databases that have many relations, each with few rows (tuples), are often called *rich*, while those that have few relations, each with many rows, are called *large*. Commercial databases tend to be large, but not particularly rich. Knowledge databases tend to be rich, but not particularly large.

About six years after my first two papers on the relational model [Codd 1969 and 1970], Chen [1976] published a technical paper describing the entity-relationship approach to database management. This approach is discussed in more detail in Chapter 30, which deals with proposed alternatives to the relational model. Although some favor the entity-relationship approach, it suffers from three fundamental problems:

1. Only the structural aspects were described; neither the operators upon these structures nor the integrity constraints were discussed. Therefore, it was not a data model.
2. The distinction between entities and relationships was not, and is still not, precisely defined. Consequently, one person's entity is another person's relationship.
3. Even if this distinction had been precisely defined, it would have added complexity without adding power.

Whatever is conceived as entities, and whatever is conceived as relationships, are perceived and operated upon in the relational model in just one common way: as relations. An entity may be regarded as inter-relating an object or identifier of an object with its immediate properties. A relationship may be regarded as a relation between objects together with the immediate properties of that relationship.

1.2.2 Inter-relating the Information Contained in Distinct Relations

Some people who are used to past approaches find it extremely difficult to understand how information in distinct relations can possibly be inter-related by the relational model without the explicit appearance in the user's perception of pointers or links.

8 ■ Introduction to Version 2 of the Relational Model

The fundamental principle in the relational model is that all inter-relating is achieved by means of comparisons of values, whether these values identify objects in the real world or whether they indicate properties of those objects. A pair of values may be *meaningfully compared*, however, if and only if these values are drawn from a common domain.

Some readers may consider the “common domain” constraint to be an unnecessary restriction. The opportunities for comparing values even with this constraint, however, are vastly superior in numbers and quality over the old approach of requiring pointers, links, or storage contiguity. Regarding the numbers, it should be remembered that not only object-identifiers can be compared with each other, but also simple properties of objects. Regarding the quality, those relational operators that involve the comparing of values require the values that are compared to be drawn from a common domain. In this way, these operators protect users from making very costly kinds of errors.

An example may help. Suppose the database contains serial numbers of suppliers and serial numbers of parts. Then, the immediate properties of a supplier contained in the SUPPLIER relation can be inter-related to the immediate properties of a supplier’s capability contained in the CAPABILITIES relation by means of a single relational operator. This operator is the **equi-join**, and its application in this case involves comparing for equality the serial numbers of suppliers in the SUPPLIERS relation with those serial numbers in the CAPABILITIES relation.

Suppose that values for the serial numbers of suppliers and parts happen to have the same basic data type (i.e., character strings of the same length). Naturally, it is not meaningful to compare the supplier serial number in the SUPPLIER relation with the part serial number in the CAPABILITIES relation, even though they happen to have the same basic data type. Thus, the domain concept plays a crucial role in the inter-relating game. In fact, in Chapter 3 I discuss the general problem of determining of a given collection of relations whether they are all inter-relatable; domains are an essential and central concept in that discussion.

1.2.3 Examples of Relations

Two examples of relations in the relational model, described next, are intended to convey the structural uniformity of the approach to representing the information in relational databases for users (including application programmers). The first of these examples is the parts relation P, which identifies and describes each kind of part carried in inventory by a manufacturer.

P#	Part serial number
PNAME	Part name
SIZE	Part size
QP	Quantity of parts

OH_QP Quantity of parts on hand
 OO_QP Quantity of parts on order
 MOH_QP Minimum quantity of parts to be in inventory

There are only four domains: P#, PNAME, SIZE, and QP.

P	P#	PNAME	SIZE	QP		
				OH_QP	OO_QP	MOH_QP
	p1	nut	10	500	300	400
	p2	nut	20	800	0	300
	p3	bolt	5	400	200	300
	p4	screw	12	1200	0	800
	p5	cam	6	150	150	100
	p6	cog	15	120	200	100
	p7	cog	25	200	50	100

This relation has six columns and therefore is of degree six. All of the rows (seven in this example) constitute the *extension* of the parts relation P. Sometimes the extension of a relation is called its *snapshot*. The remaining descriptive information constitutes the *intension* of the parts relation P.

The second example, the capabilities relation C, is intended to provide information concerning which suppliers can supply which kinds of parts.

In many approaches to database management, such a concept is treated entirely differently from the information concerning parts, differently from both the structural and the manipulative points of view. (Most of these approaches are pre-relational.) Parts are called entities, while each capability is called a relationship between suppliers and parts. The problem is that capabilities have immediate properties just as parts do. Examples of properties that are applicable to capabilities are the speed of delivery of parts ordered, the minimum package size adopted by the supplier as the unit of delivery, and the price of this unit delivered.

This example may help the reader understand why, in the relational model, precisely the same structure is adopted for capabilities as for parts—and, more generally, precisely the same structure is adopted for entities as for relationships between entities.

S# supplier serial number
 P# part serial number
 SPEED number of business days to deliver
 QP quantity of parts
 UNIT_QP minimum package
 MONEY U.S. currency
 PRICE price in U.S. dollars of minimum package

10 ■ Introduction to Version 2 of the Relational Model

There are five domains: S#, P#, TIME, QP, and MONEY.

C	S#	P#	SPEED	UNIT_QP	PRICE
	s1	p1	5	100	10
	s1	p2	5	100	20
	s1	p6	12	10	600
	s2	p3	5	50	37
	s2	p4	5	100	15
	s3	p6	5	10	700
	s4	p2	5	100	15
	s4	p5	15	5	300
	s5	p6	10	5	350

This relation has five columns and is therefore of degree five. All of the rows (nine in this example) constitute the *extension* of the capabilities relation C. The remaining descriptive information constitutes the *intension* of relation C.

1.2.4 Omission of Features

When implementing a relational database management system, many questions arise regarding the relational model. Occasionally, support for some basic feature has been omitted due to it being assessed as useless. Unfortunately, the relational model has always had features that are inextricably intertwined. This means that omission of one feature of the model in a DBMS can inhibit implementation of numerous others. For example, omission of support for primary keys and foreign keys (defined in Section 1.8) jeopardizes the implementation of

- view updatability (see Chapter 17),
- the principal integrity constraints (see Chapter 13), and
- logical data independence (see Chapter 20).

1.2.5 The Goals of RM/V2

Version 2 of the relational model (abbreviated RM/V2) now has 333 features, which are even more inextricably intertwined than the approximately 50 features of Version 1 (abbreviated RM/V1). Most of the original definitions and features of RM/V1 have been preserved unchanged in Version 2. A very few of the original definitions and features have been extended to become broader in scope!

In late 1978 I developed an extended version of the relational model called RM/T [Codd 1979]. A principal aim was to capture more of the meaning of the data. Acceptance of the ideas in this version has been exceptionally slow. Consequently, it seems prudent to develop a sequence

of versions V1, V2, V3, . . . that are more gradual in growth. As the development of this sequence proceeds, certain features of RM/T will be selected and incorporated in appropriate versions.

My goals in developing Version 2 of the relational model included all those for the original relational model, RM/V1. Three of the most important of these goals were, and remain,

1. simplifying interaction with the data by users
 - a. who have large databases,
 - b. who need not be familiar with programming, and
 - c. who normally conceive their interactions independently from all other users;
2. substantially increasing the productivity of those users who are professional programmers;
3. supporting a much more powerful tool for the database administrator to use in controlling who has access to what information and for what purpose, as well as in controlling the integrity of the database.

If these goals were attained, and I believe they have been, the market for DBMS products would be expanded enormously. This suggests one more goal, namely, that a very strong emphasis be placed on the preservation by the DBMS of database integrity. Chapters 13 and 14 are devoted to the treatment of integrity by the relational model. The DBMS products available so far have supported very few of the integrity features of the model.

It is the database administrator (abbreviated DBA) who is responsible for imposing controls on the database: controls that are adequate for the DBMS to maintain the database in a state of full integrity, as well as controls that permit access for specified purposes to only those users with authorized access for those purposes. The DBMS, however, must provide the DBA with the tools to carry out his or her job. Pre-relational DBMS products failed to provide adequate tools for this purpose.

Implementation of a high-performance DBMS that supports every feature of RM/V2 is not claimed to be easy. In fact, it is quite a challenging task. There are already clear indications that the DBMS products leading in performance and in fault tolerance will be those based on new hardware and software architectures, both of which exploit the many opportunities for concurrency provided by the relational model.

1.2.6 The Relational Model as an Abstract Machine

The term “abstract” scares many people who work in computing or data processing, even though they deal with abstractions every day. For example, speed and distance are abstractions. An airline reservation is an abstraction. Bits and bytes are abstractions. So are computer commands.

In my book *Cellular Automata* [Codd 1968], I make use of at least four levels of abstraction to explain concisely how a self-reproducing computer that is capable of computing all computable functions might be designed from a large number (in fact, millions) of simple identical cells, each of which interacts with only its immediate neighbors.

It is useful to think of RM/V2 as an abstract machine. Its level of abstraction is sufficiently high that it can be implemented in many distinctly different ways in hardware, in software, or in both. This machine can be advantageously treated by all DBMS vendors, standards committees, and DBMS users as an abstract machine standard.

For example, consider the structural features introduced in Chapter 2. Their level of abstraction is necessary for enabling different types of hardware and software (possibly from different vendors) to communicate with one another about their databases. The abstract machine must be complemented with standards that deal with the following:

- the physical representation of data for inter-computer communication;
- transaction-control signals to facilitate adequate control of each transaction that straddles two or more computer systems (e.g., the signal from one system to the other “Are you ready to commit your data?”);
- specific relational languages that have specific syntax.

These topics are discussed in detail in Chapters 24 and 25.

1.2.7 The Structured Query Language (SQL)

Many people may contend that a specific relational language, namely SQL, already exists as a standard. SQL, standing for structured query language, is a data sublanguage invented by a group in IBM Research, Yorktown Heights, N.Y. [IBM 1972].

SQL was invented in late 1972. Although it was claimed that the language was based on several of my early papers on the relational model, it is quite weak in its fidelity to the model. Past and current versions of this relational language are in many ways inconsistent with both of the abstract machines RM/V1 and RM/V2. Numerous features are not supported at all, and others are supported incorrectly. Each of these inconsistencies can be shown to reduce the usefulness and practicality of SQL.

The most noteworthy error in several current implementations is that SQL permits the use and generation of improper or corrupted relations, that is, “relations” that contain duplicate tuples or rows. In Chapter 23 this problem is examined in sufficient detail to demonstrate the seriously adverse consequences of this very simple infidelity to the model. As each feature of RM/V2 is introduced in this book, the attempt is made to comment on SQL’s support, non-support, or violation of the feature.

Several relational languages other than SQL have been developed. An

example that I consider superior to SQL is Query Language (abbreviated QUEL). This language was invented by Held and Stonebraker at the University of California, Berkeley, and was based on the language ALPHA [Codd 1972]. More attention is devoted to SQL, however, because it has been adopted as an ANSI standard, and is supported in numerous DBMS products.

1.2.8 An Abstract Machine Standard

The computing field clearly needs an abstract machine standard for database management for at least the following reasons:

- The intrinsic importance of computer-based support for the sharing of business information interactively and by program.
- The users involved in this sharing normally conceive their modifications of the information independently of one another.
- Clearly the field of database management is moving toward the management of distributed databases, and at each of the sites involved there may be hardware and software from a variety of vendors. Intercommunication among these systems will be a vital requirement.
- The boundary between hardware and software is moving out from the von Neumann position. Hardware is taking on more of the tasks previously handled by basic software, and already there are products in which numerous components of operating systems and DBMS are supported by hardware. An abstract machine standard for database management should enable this boundary to move without the necessity of continual reformulation of a new standard.

The relational model deals with database management from an abstract, logical point of view only, never at the detailed level of bits and bytes. Does this make the relational model incomplete? If incomplete in this sense, the model is intended to be this way. It is important to stop short of prescribing how data should be represented and positioned in storage, and also how it should be accessed. This not only makes users and programmers more productive, but also permits both hardware and software vendors to compete in lower-level techniques for obtaining good performance.

This is an area of considerable technical significance in which DBMS vendors can productively compete with one another. In the case of DBMS products that are carefully based on the relational model, such competition need not adversely affect the users' investment in training and application development, precisely because their perception is at a high level of abstraction.

One final reason for a high level of abstraction is that the choice of representation for data in storage and the choice of access methods depend heavily on the nature and volume of the traffic on the database.

14 ■ Introduction to Version 2 of the Relational Model

Publication of the relational model in the June 1970 issue of *Communications of the Association for Computing Machinery* [Codd 1970] preceded the completion of development of relational DBMS products by at least a decade. The model is more abstract than these systems and has an existence that is completely independent of them.

This is an advantage in many ways. First, it provides a useful goal, target, and tool for the designers and developers of new DBMS products. Second, it provides a special kind of standard against which dissimilar DBMS products can be measured and compared. No DBMS product or data sub-language marketed in the western world today fully supports each and every feature of the relational model, even Version 1 [Codd 1969, 1970, 1971a–d, 1974a]. Third, it provides a foundation upon which theoretical work in database management has been and will continue to be based.

1.3 ■ The Transaction Concept

Brief reference was made to the concept of a transaction in the preceding discussion of the additional kinds of standards that are now needed. In the relational model, this concept has a precise definition.

A *transaction* is a collection of activities involving changes to the database, all of which must be executed successfully if the changes are to be committed to the database, and none of which may be committed if any one or more of the activities fail. Normally, such a collection of activities is represented by a sequence of relational commands. The beginning of the sequence is signaled by a command such as BEGIN or BEGIN TRANSACTION. Its termination is signaled by a command such as END or COMMIT—or, if it is necessary to abort the transaction, ABORT.

A simple example of a transaction is that in which a bank customer requests the bank to transfer \$1,000 from his checking account into his savings account. In the bank's computer program the first action is to check that there is at least \$1,000 in the customer's checking account. If so, this amount is deducted from the balance in that account. The next action is to credit the customer's savings account with the \$1,000.

If the first action were successful and the second action failed (due to hardware malfunction, for example), the customer would lose the \$1,000; this would be unacceptable to most customers. Therefore, this is a case in which both actions must succeed or neither must cause any change in the database.

In DBMS products two methods of handling a transaction are as follows:

1. to delay storing in the database any data generated during execution of a transaction until the DBMS encounters a COMMIT or END TRANSACTION command, and then store all of this data;
2. to write details of each change in the recovery log as each change is generated, and immediately record the change in the database. This log

is then used for recovery purposes if an ABORT TRANSACTION command is to be executed.

1.4 ■ Classification of RM/V2 Features

Each feature of the relational model RM/V2 is assigned to one of the 18 classes listed in Table 1.2. The table includes the number of the chapter in which each class of features is described. Each letter identifies the class. Each feature has a unique label. Thus, in the feature RS-9, R stands for relational, S for the structure class, and 9 for the ninth feature in that class. The numbering of features within a class should be interpreted as a distinctive label only, not as an ordering of importance.

There is no claim that the features of RM/V2 are all independent of one another. In fact, as discussed earlier, there are numerous inter-dependencies among the features. A minimal, totally non-redundant set would be more difficult to understand, would probably reduce user productivity significantly, and would probably lead to even more errors by vendors in designing their relational DBMS products. Of course, I am not advocating

Table 1.2 The 18 Classes of Features and the Pertinent Chapters

Chapter	Label	Class
2	S	Structural
3	T	Extended data types
4	B	Basic operators
5	Z	Advanced operators
6	N	Naming
7	E	Elementary commands
10	Q	Qualifiers
11	J	Indicators
12	M	Manipulative
13,14	I	Integrity
15	C	Catalog
16,17	V	View
18	A	Authorization
19	F	Scalar and aggregate functions
20	P	Protection of user investment in the DBMS, the database, application programming, and user training
21	D	DBMS design principles
22	L	Language-design principles
24,25	X	Distributed database management

16 ■ Introduction to Version 2 of the Relational Model

the other extreme, namely complexity, since this runs counter to comprehensibility.

Two very important concepts of relational DBMS products are the *catalog* and *views*. Some think that the basic relational model does not mention the catalog or views, but these concepts were discussed in my early papers on the relational model, although not by these names. I referred to the catalog as the *relational schema* and to views as *derived relations* whose definitions were stored in the schema. In this book I have adopted the System R terms “catalog” and “view” [Chamberlin et al. 1981] because they are concise and very usable, and are now quite widely used. System R was one of three DBMS prototypes developed in distinct divisions of IBM and based on the relational model.

When DBMS products are evaluated today, the evaluation should include fidelity of the product to the relational model, and specifically RM/V2. In part, this is required because almost all vendors claim that their DBMS products are relational. Therefore, one important concern for potential users of these products is that they reap all the benefits of fidelity to the relational model.

As with RM/V1, the features that are included in RM/V2 are intended to be helpful for all users of relational DBMS, both application programmers and end users. Also, as with RM/V1, they are intended to help the designers, implementors, and evaluators of relational DBMS products. RM/V2 features include all the features of RM/V1, together with the following:

- new features that cover important aspects of relational DBMS not previously included in RM/V1, either because they were overlooked or because I considered them too obvious to mention, until I discovered that many people had not realized their obvious importance;
- new features that are in line with the original RM/V1, but at a slightly lower level of abstraction.

A DBMS product is *fully relational* in the 1990s if it fully supports each and every one of the features of RM/V2 defined in this book. A DBMS product that is not fully relational can nevertheless qualify to be called *relational* in the early 1990s by fully supporting each one of the roughly 40 features listed in Appendix A.

Like the basic relational model RM/V1, all the features of RM/V2 are based on the practical requirements of users, database administrators, application programmers, security staff, and their managers. Along with the description of each feature, I attempt to explain the practical reasons for that feature.

The relational model RM/V2 is based on the original model RM/V1 and on a single fundamental rule, which I call Rule Zero:

For any system that is advertised as, or claimed to be, a relational database management system, *that system must be able to manage*

databases entirely through its relational capabilities, no matter what additional capabilities the system may support.

This must hold *whether or not* the system supports any non-relational capabilities of managing data. Any DBMS that does *not* satisfy this Rule Zero is *not* a *relational* DBMS.

The danger to buyers and users of a system that is claimed to be a *relational* DBMS and fails on Rule Zero is that these buyers and users will expect all the advantages of a truly relational DBMS, but will fail to receive them.

One consequence of Rule Zero is that any system claimed to be a relational DBMS must support database insert, update, and delete at the *relational* level (multiple-record-at-a-time). (See Feature RM-6 in Chapter 12.) Another consequence is the necessity of supporting the information feature and the guaranteed-access feature. (See Feature RS-1 in Chapter 2 and Features RM-1 and RM-2 in Chapter 12.)

Incidentally, “multiple-record-at-a-time” includes the ability to handle those situations in which zero or one record happens to be retrieved, inserted, updated, or deleted. In other words, a relation (often carelessly called a table) may have either zero tuples (rows) or one tuple (row) and still be a valid relation. Note that although it may be unusual for a relation to have either zero rows or one row, it does not receive special treatment in the relational model, and therefore users do not have to treat such a relation in any special way either.

1.5 ■ Tables versus Relations

Actually, the terms “relation” and “table” are not synonymous. As discussed earlier, the concept of a relation found in mathematics and in the relational model is that of a *special kind of set*. The relations of the relational model, although they may be *conceived* as tables, are then special kinds of tables. In this book they are called *R-tables*, although the term “relation” is still used from time to time to emphasize the underlying concept of mathematical sets, to refer to the model, or to refer to languages developed as part of implementations of the model.

R-tables have no positional concepts. One may shuffle the rows without affecting information content. Thus, there is no “nextness” of rows. Similarly, one may shuffle the columns without affecting information content, providing the column heading is taken with each column. Thus, there is no “nextness” of columns.

Normally, neither of these shuffling activities can be applied with such immunity to arrays. That is why I consider it extremely misleading to use the term “array” to describe the structuring of data in the relational model.

Those relations, or *R-tables*, that are internally represented by stored data in some implementation-defined way are called the *base relations* or

base R-tables. All R-tables other than base R-tables are called *derived relations* or, synonymously, *derived R-tables*. An example of a derived relation is a *view*. A view is a virtual R-table defined in terms of other R-tables, and is represented by its defining expression only.

In both RM/V1 and RM/V2, *duplicate rows are not permitted in any relations*, whether base relations, views, or any other type of relations. For details, see Features RS-3 and RI-3 in Chapters 2 and 13, respectively. This rule has been applied in all of my technical papers on the relational model, even the first one [Codd 1969].

In RM/V2, duplicate rows are still excluded from all relations. They are excluded from *base R-tables*, primarily as a step to retain integrity of the database: each row in such a table represents an object whose distinctiveness is lost if duplicate rows are allowed in these R-tables. A very fundamental property of the relational model is that each object about which information is stored in the database must be uniquely and explicitly identified, and thereby distinguished from every other object. As we shall see, the unique identifier is the name of the R-table, together with the primary key value. This fundamental property, an integrity-preservation feature, is not enforced by any other approach to database management.

Duplicate rows are still excluded from all *derived R-tables* for semantic reasons (see Fundamental Law 20 in Chapter 29). They are also excluded because such duplicates severely reduce the interchangeability of sequencing of operators within a relational command or in a sequence of commands. This reduction in interchangeability has two serious consequences (see Chapter 23 for more detail):

1. it reduces the optimizability of relational commands;
2. it imposes severe conceptual problems and severe constraints on users.

Two of the early prototypes of relational DBMS products were developed in the mid-1970s by the System R team at IBM Research in San Jose, Calif. [Chamberlin et al. 1981] and the INGRES team at the University of California Berkeley [Stonebraker 1986]. Curiously, both of these teams made the same two criticisms of the relational model:

1. the expected loss of performance if duplicate rows had to be eliminated in several types of relational operations without the user explicitly requesting that elimination;
2. the alleged impossibility of applying statistical functions correctly to columns that happen to have duplicate values legally.

Based on the first point, I conditionally agreed to the idea that duplicate rows should be permitted in derived R-tables *only*, not in base R-tables. The condition for this concession was that all of the effects upon the relational operators be carefully examined for possibly damaging consequences.

On the second point, I found myself in strong disagreement, because the mistake made in these two prototypes was to apply as a first step the projection operator (see Chapter 4) on the column or columns for which statistics were needed. Instead, I advised the researchers to apply the statistical function first in the context of whatever relation was given, and then apply the projection operator, only if such action were necessary for other reasons.

It now appears that neither project adequately examined the severely damaging effects of duplicate rows (1) on the operators and (2) on common interpretability by users (see Chapter 23).

This latter concern is related to the fact that, when hundreds, possibly thousands, of users share a common database, *it is essential that they also share a common meaning for all of the data therein that they are authorized to access*. There does not exist a precise, accepted, context-independent interpretation of duplicate rows in a relation. These adverse consequences are the reason that I still find that duplicate rows in any relation are unacceptable.

Let us turn our attention to a table that is extracted from a non-relational source for storage in a relational database. If it happens to contain duplicate rows, these duplicate rows can easily be removed by means of a special operator (see Feature RE-17 in Chapter 7). This operator removes all except one occurrence of each row that has multiple occurrences. It leaves the table unchanged if it happens to contain no duplicate rows.

From an evaluation standpoint, the RM/V2 features defined in this book have been created with primary concern for those DBMS products that are designed to support multiple users concurrently accessing shared data and engaged in tasks that can be (and often are) conceived independently of one another. Therefore, some of the features are not applicable to a DBMS intended for very small computer systems, particularly single-user systems such as personal computers.

An example of such a feature is concurrency control. Although locking as a form of concurrency control is mentioned in very few features of RM/V2, it is accompanied by the phrase “or some alternative technique for concurrency control that is at least as powerful as locking (and provably so).” I plan to say more about the subject of locking in a forthcoming book on computer-aided development (CAD) and engineering extensions to the relational model.

The logic that is usually encountered in data processing is propositional logic, often called Boolean logic, which deals with only two truth-values: TRUE and FALSE. In the field of database management, one reason that propositional logic was considered adequate in the past is that, before the relational model, logic was considered relevant to query products only, and such products normally supported the use of logic in the querying of single files only. Two truth values were considered adequate because no attempt was made to handle missing values in a uniform and systematic manner across the entire database.

Mathematical logic plays a central role in the relational model. In RM/V1 the logic is *three-valued*, first-order predicate logic, where the three truth-values are TRUE, FALSE, and MAYBE. This logic is substantially more powerful than propositional logic. The MAYBE truth-value means that the DBMS cannot decide whether a truth-valued expression is TRUE or FALSE due to values missing from the database.

In RM/V2 this logic is extended to *four-valued*, first-order predicate logic, where the four truth-values are TRUE, FALSE, MAYBE BUT APPLICABLE, and MAYBE BUT INAPPLICABLE. This is especially relevant when it becomes necessary to handle information that may contain some database values that are applicable but missing because they have not been entered yet, and some values that are missing because the property in question is inapplicable to the pertinent object (see Chapter 8 on missing information).

1.6 ■ Terminology

In this account of RM/V2, several terms that are now popular in database management are used, instead of the longer established and more carefully defined mathematical terms. Any ambiguity that is perceived in the use of the database-oriented terms can be resolved by referring to Table 1.3.

The degree n of a relation is the number of columns, which can be any positive integer, including the special case of a unary relation for which $n = 1$. A relational database is perceived by all users, whether application programmers or end users, as a collection of relations of assorted degrees. Each relation can be thought of as inter-relating the identifying properties of a type of object with the other immediate properties of that type of object. Every value appearing in a relation is treated by the DBMS as atomic, except for certain special functions that are able to decompose certain kinds of values (see Chapter 19).

The phrases “delete duplicates” and “delete redundant duplicates” mean *delete all occurrences except one* of an object (the object is determined by the context in which this phrase is used, and it is usually a complete row of an R-table).

Table 1.3 Mathematical and Database Terms

Mathematical Term	Database Term
Relation of degree n	R-table with n columns
Attribute	Column of R-table
Domain	Extended data type
Tuple	Row of R-table
Cardinality of relation	Number of rows in R-table

In this book the terms “interrogation,” “query,” and “retrieval” are used synonymously. Each of these terms denotes a read-only operation. No data modification is involved. Notwithstanding their names, identifying the database languages SQL and QUEL as just query languages is quite incorrect, since both support much more than interrogation.

The terms “modification” and “manipulation” are used whenever data modification is involved, whether it be data entry, deletion, or updating. Except where otherwise indicated, the term “updating” denotes a particular kind of modification, namely, modification applied to values already within the database. Therefore, updating is normally an operation that is distinct from both data entry and deletion.

When applied to any database activity, the term “dynamically” means *without* bringing any database traffic to a halt.

1.7 ■ Role of Language in the Relational Model

Early in the development of the relational model (1969-1972), I invented two languages for dealing with relations: one algebraic in nature, and one based on first-order predicate logic [Codd 1971a]. I then proved that the two languages had equal expressive power [Codd 1971d], but indicated that the logic-based language would be more optimizable (assuming that flow tracing was not attempted) and easier to use as an interface to inferential software on top of the DBMS.

During subsequent development of the relational model, I have avoided the development of a specific language with specific syntax. Instead, it seemed appropriate that my work remain at a very high level of abstraction, leaving it to others to deal with the specific details of usable languages. Thus, the relational model specifies the semantics of these languages, and does not specify the syntax at all.

The abbreviation RL denotes the principal relational language supported by the DBMS—a language intended specifically for database management, and one that is not guaranteed to be usable for the computation of all computable functions. RM/V2 specifies the features that RL should have, and the specification is (as we just saw) semantic, not syntactic. Examples of existing relational languages are SQL and QUEL, although neither of these supports more than half the relational model.

The power of RL includes that of four-valued, first-order predicate logic [Church 1958, Suppes 1967, Stoll 1961, Pospesil 1976]. The complete power of RL should be fully exploitable in at least the following contexts:

- retrieval (database description, contents, and audit log);
- view definition;
- insertion, update, and deletion;
- handling missing information (independent of data type);

- integrity constraints;
- authorization constraints;
- if the DBMS is claimed to be able to handle distributed data, distributed database management with distribution independence, including automatic decomposition of commands by the DBMS and automatic recomposition of results by the DBMS (see Feature RP-4 in Chapter 20).

One of the main reasons that “object-oriented” DBMS prototypes and products are not going to replace the relational model and associated DBMS products is these systems appear to omit support for predicate logic. It will take brilliant logicians to invent a tool as powerful as predicate logic. Even then, such an invention is not an overnight task—once invented, it might well take more than a decade to become accepted by logicians. Thus, features that capture more of the meaning of the data should be added to the relational model [Codd 1979], instead of being proposed as replacements.

In the development of application programs, a relational language normally needs as a partner a host language such as COBOL, PL/1, FORTRAN, or some more recently developed programming language. Some relational DBMS support several such host languages to be used as partners, although the user is normally required to select just one for developing an application program. In this book I occasionally use the term “HL” (for “host language”) to denote such a language.

Languages are being developed that are significantly higher in level than COBOL, PL/1, and FORTRAN; such languages frequently include statements that must be translated into RL. Thus, an important requirement for RL is that it be both convenient and powerful in two roles: as a source language and as a target language.

Sometimes I am asked why I do not extend relational languages to include the features of PROLOG or of someone’s favorite “fourth-generation” language. My usual reply is that I do not wish to tie the destiny of the relational model to any tool that has not been overwhelmingly accepted or does not appear to be defined at the same level of abstraction as the relational model. Moreover, I believe that the days of monstrous programming languages are numbered, and that the future lies with specialized sublanguages that can inter-communicate with one another.

1.8 ■ Keys and Referential Integrity

The term “key” has been used in the computing field for a long time, and with a great variety of meanings. In the relational model the term is normally qualified by the adjectives “candidate,” “primary,” and “foreign,” and each of these phrases has a precisely defined meaning.

Each base R-table has exactly one primary key. This key is a combination of columns (possibly just one column) such that

- the value of the primary key in each row of the pertinent R-table identifies that row uniquely (i.e., it distinguishes that row from every other row in that R-table);
- if the primary key is composite and if one of the columns is dropped from the primary key, the first property is no longer guaranteed.

Sometimes these two properties are called the *uniqueness property* and the *minimality property*, respectively. Note, however, that “minimality” in this context does not mean the shortest in terms of bits or bytes or the one having the fewest components.

It is equally valid to interpret the uniqueness property in terms of object identification: the value of the primary key in each row of the pertinent R-table identifies the particular object represented by that row uniquely within the type of objects that are represented by that relation. Everywhere else in the database that there is a need to refer to that particular object, the *same* identifying value drawn from the *same* domain is used. Any column containing those values is called a *foreign key*, and each value in that column is called a *foreign key value*.

Referential integrity is defined as follows:

Let D be a domain from which one or more primary keys draw their values. Let K be a foreign key, which draws its values from domain D . Every unmarked value which occurs in K must also exist in the database as the value of the primary key on domain D of some base relation.

Incidentally, a value in the database is *marked* if and only if it is missing. The subject of missing information is discussed in some detail in Chapters 8 and 9.

The case in which K is a combination of columns, and some (perhaps all) of the component values of a foreign key value are allowed to be marked as missing, needs special attention. *Those components of such a foreign key value that are unmarked should adhere to the referential-integrity constraint.* This detail is not supported in many current DBMS products, even when the vendors claim that their products support referential integrity.

To make use of this definition, it is necessary to understand primary keys (PK) and foreign keys (FK). The example in the following subsection is intended to give the reader some understanding of the semantic nature of these keys.

1.8.1 Semantic Aspects of Primary and Foreign Keys

Notice that referential integrity applies to pairs of keys only, one a primary key PK and the other a foreign key FK. The keys may be simple (single-column) or composite (two or more columns). The DBMS should not require

that each and every combination of simple keys within a single relation be treated as a foreign key, even if that combination appears as a composite primary key in the database. This is clearly an issue that is related to the meaning of the data.

Suppose, for example, that a database contains the relations listed in Table 1.4.

Table 1.4 Example Relations in a Database

Relation	Meaning	Primary Key
R1	Suppliers	S#
R2	Parts	P#
R3	The <i>capabilities</i> of suppliers to supply parts, including price and speed of delivery	(S#,P#)
R4	<i>Orders</i> for parts placed with specified suppliers, including date the order was placed	(S#,P#, DATE)

To avoid an extra relation and keep the example simple, assume that every order is a one-line order (that is, only one kind of part appears on any order) and that it is impossible for two orders with the same order date to refer to identical kinds of parts.

Suppose that each of two companies has a database of this kind. In company A, however, the relation R3 is used as advisory information, and there is no requirement that every combination of (S#,P#) that appears in R4 must appear in R3. In company B, on the other hand, R3 is used as controlling information: that is, if an order is placed for part p from supplier s, it is company policy that there must be at least one row in relation R3 stating that p is obtainable from s, and incidentally indicating the price and the speed of delivery. Of course, there may be other rows in R3 stating that p is obtainable from other suppliers. Thus, if referential integrity were applied to the combination (S#,P#) as primary key in R3 and foreign key in R4, it would be correct in company B, but incorrect in company A.

There are two ways in which this example (and similar ones) could be handled:

1. Make the referential integrity constraint applicable to all PK-FK pairs of keys (whether simple or composite) in which one key PK is declared to be primary, and the other key FK is declared to be foreign. In company B, declare the (S#,P#) combination in R4 as a foreign key that has as its target the (S#,P#) primary key of R3. In company A, avoid altogether the declaration that (S#,P#) in R4 is a foreign key.
2. Make the referential integrity constraint applicable to simple PK-FK pairs of keys only, and require the DBA to impose a referential con-

straint on just those compound PK-FK pairs of keys for which the constraint happens to be applicable in his or her company—by specifying a user-defined integrity constraint, expressed in the relational language.

Method 1 is the approach now adopted in the relational model. It makes the foreign-key concept a more semantic feature than does Method 2. After all, the concepts of keys in the relational model were always intended to identify objects in the micro-world that the database is supposed to represent. In other words, keys in the relational model act as surrogates for the objects being modeled. Once again, Method 1 is adopted.

1.8.2 Primary Keys on a Common Domain

Let us consider an example of the fact that primary keys on a given domain can occur in more than one base relation. This is the database in which there are two base R-tables that provide the immediate properties of suppliers: one for the domestic suppliers, one for the foreign suppliers. There would normally be some properties in the foreign suppliers table that do not occur in the domestic suppliers table. Each R-table has as its primary key the supplier serial number. Nevertheless, the database may contain several R-tables that include the supplier serial number as a foreign key without making any distinction regarding the R-tables in which the corresponding primary key value resides. In general, that value may reside as a primary key value in one, two, or even more R-tables.

No assumption is made in either RM/V1 or RM/V2 concerning the adoption of the tighter discipline of the extended relational model RM/T [Codd 1979]. For example, there is no requirement that type hierarchies be incorporated in the database design, wherever they are appropriate. Moreover, there is no requirement that, for each primary key, a unary relation (called the *E-relation* in RM/T) exists to list all of the distinct values in use for that primary key.

A second example in the non-distributed case is that of a base relation R that happens to have many columns, but a large amount of traffic on only 20% of these columns (call this A) and a very modest amount on the remaining 80% (call this B). In such a case the DBA may decide to improve performance by storing the data in the form of two base relations instead of one:

1. a projection of R onto its primary key together with A;
2. a projection of R onto its primary key together with B.

A specific feature of the relational model that requires support in the DBMS for multiple primary keys from a common domain is RS-10, described in the next chapter.

In the case of distributed database management, it is not at all uncommon

to have the information distributed in such a way that several relations at several sites all have a primary key based on a common domain. See Section 24.4 for a detailed discussion of the relational approach to distributing data.

1.8.3 Comments on Implementation

Referential integrity is discussed further in Chapter 13. It should be implemented as far as possible as a special case of user-defined integrity (see Chapter 14) because of their similarities. One such common need, for example, is to give the DBA or other authorized user the freedom to specify linguistically how the system is to react to any attempt to violate these integrity constraints, whether the constraints are referential or user-defined.

Further, it should be remembered that referential integrity is a particular application of an *inclusion constraint* (sometimes called an inclusion dependency). Such a constraint requires that the set of distinct values occurring in some specified column, simple or composite, must be a subset of the values occurring in some other specified column (simple or composite, respectively). In the case of referential integrity, the set of distinct simple FK values should be a subset of the set of distinct simple PK values drawn from the same domain.

Inclusion constraints, however, may apply between other pairs of attributes also (e.g., non-keys). When declared and enforced, such additional constraints reflect either business policies or government regulations. One would then like the DBMS to be designed in such a way as to provide reasonably uniform support for referential integrity and these additional (user-defined) inclusion constraints.

1.9 ■ More on Terminology

The following terms are used in connection with relational languages and user-defined functions.

- *retrieval targeting*: specifying the kinds of database values to be extracted from the database, and then possibly specifying transformations to be applied to occurrences of these values;
- *retrieval conditioning*: specifying a logical condition in a retrieval or manipulative statement of a particular relational language for the purpose of conditioning access;
- *PK-targeting*: finding the primary key(s) corresponding to any given foreign key;
- *FK-targeting*: finding the foreign key(s) corresponding to any given primary key;

- *PK-based projection*: a projection that includes the *primary key* of the operand R-table;
- *non-PK projection*: a projection that does not include the *primary key* of the operand R-table.

1.10 ■ Points to Remember

Four important points concerning relations follow:

1. every relation is a set;
2. *not* every set is a relation;
3. every relation can be perceived as a table;
4. *not* every table is a correct perception of a relation.

Designers of the relational DBMS products of many vendors appear to be ignorant of these facts or to have ignored them.

Exercises

Note that, for some exercises, additional chapters are identified as sources of more information.

- 1.1 Identify the 18 classes of features in RM/V2. Supply a brief description of each class.
- 1.2 When a relation is perceived as a table, what are the special properties of that table? Is the ordering of columns crucial? Is the ordering of rows crucial? Can the table contain duplicate rows?
- 1.3 The terms “table” and “relation” are not synonymous. Supply a simple example of a table that is neither a relation of the relational model nor a relation of mathematics.
- 1.4 What is your position on the entity-relationship approach? (See also Chapter 30.) Will it replace the relational model? Give five technical reasons for your answer.
- 1.5 What is a transaction in the relational model? Describe an application that illustrates that there is a practical need for this concept.
- 1.6 Are either of the following statements true about the structures of the relational model?

They are merely flat files.

They are merely tables.

In each case, if your answer is no, give an example of a flat file that is not a relation or an example of a table that is not a relation.

28 ■ Introduction to Version 2 of the Relational Model

- 1.7 What is your position on the object-oriented approach? (See also Chapter 30.) Will it replace the relational model? Give five reasons for your answer. You may wish to postpone this exercise, as well as Exercise 1.8, until you have absorbed Chapter 28.
- 1.8 Can any object-oriented concepts be added to the relational model without violating any of the principles on which the model is based? Which concepts? (See also Chapter 30.)
- 1.9 When designing a database, is it possible to anticipate all of the uses to which the data will be put? Is it possible to anticipate the batch load, on-line teleprocessing load, and interactive query load for the next three, five, or seven years? Conclude from your answer what properties the DBMS should have if it is to protect the user's investment in application programs. (See also Chapter 26.)
- 1.10 Are duplicate rows needed in a relation? If so, what for? Supply an example. Should duplicate rows be allowed in any relation? State reasons why or why not, whichever is applicable, and supply examples. (See also Chapters 2 and 23.)
- 1.11 In RM/V2 does the prohibition of duplicate rows within every relation imply that no duplicate values (e.g., currency values) can occur in any column? Explain.