

FUNDAMENTALS OF DATABASE SYSTEMS

Third Edition



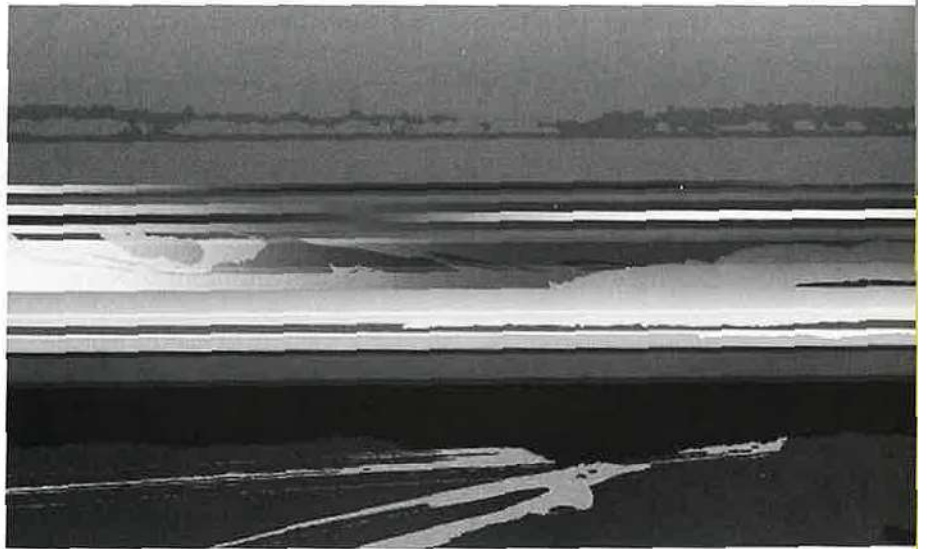
ELMASRI | NAVATHE

Sterling EX2004

IPR2025-00270

Page 1 of 33

FUNDAMENTALS OF *Third Edition* DATABASE SYSTEMS



Ramez Elmasri

*Department of Computer Science Engineering
University of Texas at Arlington*

Shamkant B. Navathe

*College of Computing
Georgia Institute of Technology*



ADDISON-WESLEY

An imprint of Addison Wesley Longman, Inc.

Reading, Massachusetts • Menlo Park, California • New York • Harlow, England
Don Mills, Ontario • Sydney • Mexico City • Madrid • Amsterdam

Sterling EX2004

IPR2025-00270

Page 2 of 33

Acquisitions Editor: Maite Suarez-Rivas
Associate Editor: Katherine Harutunian
Cover Design: Lynne Reed
Composition: Northeast Compositors
Proofreader: Diane Freed

Cover Art: "Crystal Zone"—Original hand-pulled limited edition serigraph by Tetsuro Sawada. Exclusive Sawada publisher and distributor: Buschlen/Mowatt Fine Arts Ltd., Main Floor, 1445 West Georgia Street, Vancouver, Canada V6G 2T3 (604) 682-1234.

Certain diagrams have been reproduced herein with the permission of Oracle Corporation. Copyright © Oracle Corporation, 1997. All rights reserved.

This book was typeset in FrameMaker 5.5 on a Power Macintosh G3. The fonts used were Bodega, Goudy, and Lucida Sans.

Copyright ©2000 by Ramez Elmasri and Shamkant B. Navathe

All rights reserved. No part of this publication may be reproduced, stored in a database or retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or any other media embodiments now known or hereafter to become known, without the prior written permission of the publisher. Printed in the United States of America.

Library of Congress Cataloging-in-Publication Data

Elmasri, Ramez.

Fundamentals of database systems / Ramez Elmasri, Shamkant B. Navathe. — 3rd ed.

p. cm.

Includes bibliographical references.

ISBN 0-8053-1755-4

1. Database management. I. Navathe, Sham. II. Title.

QA76.9.D3E57 2000

005.74—dc21

99-30620

CIP

ISBN 0-8053-1755-4

1 2 3 4 5 6 7 8 9 10-DOH-0302010099

Sterling EX2004
IPR2025-00270
Page 3 of 33

CONTENTS

PART 1: BASIC CONCEPTS

Chapter 1 Databases and Database Users 2

- 1.1 Introduction 4
- 1.2 An Example 5
- 1.3 Characteristics of the Database Approach 8
- 1.4 Actors on the Scene 12
- 1.5 Workers behind the Scene 13
- 1.6 Advantages of Using a DBMS 14
- 1.7 Implications of the Database Approach 18
- 1.8 When Not to Use a DBMS 19
- 1.9 Summary 19
- Review Questions 20
- Exercises 21
- Selected Bibliography 21

Chapter 2 Database System Concepts and Architecture 23

- 2.1 Data Models, Schemas, and Instances 24
- 2.2 DBMS Architecture and Data Independence 27
- 2.3 Database Languages and Interfaces 30
- 2.4 The Database System Environment 32

2.5	Classification of Database Management Systems	35
2.6	Summary	37
	Review Questions	38
	Exercises	39
	Selected Bibliography	39

Chapter 3 Data Modeling Using the Entity-Relationship Model 41

3.1	Using High-Level Conceptual Data Models for Database Design	42
3.2	An Example Database Application	44
3.3	Entity Types, Entity Sets, Attributes, and Keys	45
3.4	Relationships, Relationship Types, Roles, and Structural Constraints	52
3.5	Weak Entity Types	59
3.6	Refining the ER Design for the COMPANY Database	60
3.7	ER Diagrams, Naming Conventions, and Design Issues	61
3.8	Summary	66
	Review Questions	67
	Exercises	67
	Selected Bibliography	72

Chapter 4 Enhanced Entity-Relationship and Object Modeling 73

4.1	Subclasses, Superclasses, and Inheritance	74
4.2	Specialization and Generalization	76
4.3	Constraints and Characteristics of Specialization and Generalization	80
4.4	Modeling of UNION Types Using Categories	86
4.5	An Example UNIVERSITY EER Schema and Formal Definitions for the EER Model	90
4.6	Conceptual Object Modeling Using UML Class Diagrams	93

4.7	Relationship Types of Degree Higher Than Two	95
4.8	Data Abstraction and Knowledge Representation Concepts	100
4.9	Summary	105
	Review Questions	105
	Exercises	106
	Selected Bibliography	110
Chapter 5 Record Storage and Primary File Organizations		113
5.1	Introduction	114
5.2	Secondary Storage Devices	117
5.3	Parallelizing Disk Access Using RAID Technology	122
5.4	Buffering of Blocks	127
5.5	Placing File Records on Disk	128
5.6	Operations on Files	133
5.7	Files of Unordered Records (Heap Files)	135
5.8	Files of Ordered Records (Sorted Files)	136
5.9	Hashing Techniques	139
5.10	Other Primary File Organizations	148
5.11	Summary	149
	Review Questions	149
	Exercises	150
	Selected Bibliography	153
Chapter 6 Index Structures for Files		155
6.1	Types of Single-Level Ordered Indexes	156
6.2	Multilevel Indexes	166
6.3	Dynamic Multilevel Indexes Using B-Trees and B ⁺ -Trees	169
6.4	Indexes on Multiple Keys	183
6.5	Other Types of Indexes	186
6.6	Summary	187
	Review Questions	188

Exercises	188
Selected Bibliography	190

PART 2: RELATIONAL MODEL, LANGUAGES, AND SYSTEMS 193

Chapter 7 The Relational Data Model, Relational Constraints, and the Relational Algebra 195

7.1	Relational Model Concepts	196
7.2	Relational Constraints and Relational Database Schemas	202
7.3	Update Operations and Dealing with Constraint Violations	209
7.4	Basic Relational Algebra Operations	211
7.5	Additional Relational Operations	226
7.6	Examples of Queries in Relational Algebra	230
7.7	Summary	232
	Review Questions	233
	Exercises	235
	Selected Bibliography	240

Chapter 8 SQL—The Relational Database Standard 243

8.1	Data Definition, Constraints, and Schema Changes in SQL2	245
8.2	Basic Queries in SQL	251
8.3	More Complex SQL Queries	261
8.4	Insert, Delete, and Update Statements in SQL	275
8.5	Views (Virtual Tables) in SQL	278
8.6	Specifying General Constraints as Assertion	282
8.7	Additional Features of SQL	283
8.8	Summary	284

Review Questions	284
Exercises	286
Selected Bibliography	288

Chapter 9 ER- and EER-to-Relational Mapping, and Other Relational Languages 289

9.1	Relational Database Design Using ER-to-Relational Mapping	290
9.2	Mapping EER Model Concepts to Relations	295
9.3	The Tuple Relational Calculus	299
9.4	The Domain Relational Calculus	308
9.5	Overview of the QBE Language	310
9.6	Summary	317
	Review Questions	318
	Exercises	319
	Selected Bibliography	320

Chapter 10 Examples of Relational Database Management Systems: Oracle and Microsoft Access 323

10.1	Relational Database Management Systems: A Historical Perspective	324
10.2	The Basic Structure of the Oracle System	325
10.3	Database Structure and Its Manipulation in Oracle	329
10.4	Storage Organization in Oracle	333
10.5	Programming Oracle Applications	337
10.6	Oracle Tools	343
10.7	An Overview of Microsoft Access	344
10.8	Features and Functionality of Access	351
10.9	Summary	355
	Selected Bibliography	355

**PART 3: OBJECT-ORIENTED AND EXTENDED RELATIONAL
DATABASE TECHNOLOGY 357****Chapter 11 Concepts for Object-Oriented Databases 359**

- 11.1 Overview of Object-Oriented Concepts 361
- 11.2 Object Identity, Object Structure, and Type Constructors 363
- 11.3 Encapsulation of Operations, Methods, and Persistence 369
- 11.4 Type Hierarchies and Inheritance 374
- 11.5 Complex Objects 377
- 11.6 Other Object-Oriented Concepts 379
- 11.7 Summary 382
- Review Questions 383
- Exercises 384
- Selected Bibliography 384

**Chapter 12 Object Database Standards, Languages, and
Design 385**

- 12.1 Overview of the Object Model of ODMG 386
- 12.2 The Object Definition Language 399
- 12.3 The Object Query Language 404
- 12.4 Overview of the C++ Language Binding 413
- 12.5 Object Database Conceptual Design 414
- 12.6 Examples of ODBMSs 417
- 12.7 Overview of the CORBA Standard for Distributed Objects 428
- 12.8 Summary 431
- Review Questions 432
- Exercises 432
- Selected Bibliography 433

**Chapter 13 Object Relational and Extended Relational
Database Systems 435**

- 13.1 Evolution and Current Trends of Database Technology 436
- 13.2 The Informix Universal Server 438
- 13.3 Object-Relational Features of Oracle 8 448
- 13.4 An Overview of SQL3 451
- 13.5 Implementation and Related Issues for Extended Type Systems 458
- 13.6 The Nested Relational Data Model 459
- 13.7 Summary 461
- Selected Bibliography 462

PART 4: DATABASE DESIGN THEORY AND METHODOLOGY 463**Chapter 14 Functional Dependencies and Normalization for
Relational Databases 465**

- 14.1 Informal Design Guidelines for Relation Schemas 467
- 14.2 Functional Dependencies 476
- 14.3 Normal Forms Based on Primary Keys 483
- 14.4 General Definitions of Second and Third Normal Forms 490
- 14.5 Boyce-Codd Normal Form 493
- 14.6 Summary 496
- Review Questions 496
- Exercises 497
- Selected Bibliography 500

**Chapter 15 Relational Database Design Algorithms and
Further Dependencies 501**

- 15.1 Algorithms for Relational Database Schema Design 502
- Sterling EX2004
IPR2025-00270
Page 10 of 33

15.2	Multivalued Dependencies and Fourth Normal Form	514
15.3	Join Dependencies and Fifth Normal Form	519
15.4	Inclusion Dependencies	520
15.5	Other Dependencies and Normal Forms	521
15.6	Summary	523
	Review Questions	524
	Exercises	524
	Selected Bibliography	525
Chapter 16 Practical Database Design and Tuning		527
16.1	The Role of Information Systems in Organizations	528
16.2	The Database Design Process	532
16.3	Physical Database Design in Relational Databases	551
16.4	An Overview of Database Tuning in Relational Systems	554
16.5	Automated Design Tools	560
16.6	Summary	563
	Review Questions	564
	Selected Bibliography	565

PART 5: SYSTEM IMPLEMENTATION TECHNIQUES 568

Chapter 17 Database System Architectures and the System Catalog 569

17.1	System Architectures for DBMSs	570
17.2	Catalogs for Relational DBMSs	573
17.3	System Catalog Information in ORACLE	577
17.4	Other Catalog Information Accessed by DBMS Software Modules	581
17.5	Data Dictionary and Data Repository Systems	582
17.6	Summary	583

Review Questions	584
Exercises	584
Selected Bibliography	585

Chapter 18 Query Processing and Optimization 585

18.1	Translating SQL Queries into Relational Algebra	587
18.2	Basic Algorithms for Executing Query Operations	588
18.3	Using Heuristics in Query Optimization	604
18.4	Using Selectivity and Cost Estimates in Query Optimization	615
18.5	Overview of Query Optimization in ORACLE	624
18.6	Semantic Query Optimization	625
18.7	Summary	626
	Review Questions	626
	Exercises	627
	Selected Bibliography	628

Chapter 19 Transaction Processing Concepts 629

19.1	Introduction to Transaction Processing	630
19.2	Transaction and System Concepts	637
19.3	Desirable Properties of Transactions	640
19.4	Schedules and Recoverability	641
19.5	Serializability of Schedules	644
19.6	Transaction Support in SQL	654
19.7	Summary	656
	Review Questions	657
	Exercises	657
	Selected Bibliography	659

Chapter 20 Concurrency Control Techniques 661

20.1	Locking Techniques for Concurrency Control	662
20.2	Concurrency Control Based on Timestamp Ordering	677

20.3	Multiversion Concurrency Control Techniques	674
20.4	Validation (Optimistic) Concurrency Control Techniques	677
20.5	Granularity of Data Items and Multiple Granularity Locking	678
20.6	Using Locks for Concurrency Control in Indexes	683
20.7	Some Other Concurrency Control Issues	684
20.8	Summary	685
	Review Questions	686
	Exercises	687
	Selected Bibliography	687
Chapter 21	Database Recovery Techniques	689
21.1	Recovery Concepts	690
21.2	Recovery Techniques Based on Deferred Update	696
21.3	Recovery Techniques Based on Immediate Update	701
21.4	Shadow Paging	702
21.5	The ARIES Recovery Algorithm	704
21.6	Recovery in Multidatabase Systems	708
21.7	Database Backup and Recovery from Catastrophic Failures	709
21.8	Summary	709
	Review Questions	710
	Exercises	711
	Selected Bibliography	713
Chapter 22	Database Security and Authorization	715
22.1	Introduction to Database Security Issues	715
22.2	Discretionary Access Control Based on Granting/Revoking of Privileges	718
22.3	Mandatory Access Control for Multilevel Security	723
22.4	Introduction to Statistical Database Security	726

22.5	Summary	728
	Review Questions	728
	Exercises	729
	Selected Bibliography	729

PART 6: ADVANCED DATABASE CONCEPTS & EMERGING APPLICATION 731

Chapter 23 Enhanced Data Models for Advanced Applications 733

23.1	Active Database Concepts	734
23.2	Temporal Database Concepts	744
23.3	Spatial and Multimedia Databases	757
23.4	Summary	760
	Review Questions	761
	Exercises	762
	Selected Bibliography	763

Chapter 24 Distributed Databases and Client-Server Architecture 765

24.1	Distributed Database Concepts	766
24.2	Data Fragmentation, Replication, and Allocation Techniques for Distributed Database Design	771
24.3	Types of Distributed Database Systems	778
24.4	Query Processing in Distributed Databases	781
24.5	Overview of Concurrency Control and Recovery in Distributed Databases	786
24.6	An Overview of Client-Server Architecture and Its Relationship to Distributed Databases	789
24.7	Distributed Databases in Oracle	791
24.8	Future Prospects of Client-Server Technology	793
24.9	Summary	795

	Review Questions	795
	Exercises	796
	Selected Bibliography	798
Chapter 25	Deductive Databases	801
25.1	Introduction to Deductive Databases	802
25.2	Prolog/Datalog Notation	803
25.3	Interpretation of Rules	808
25.4	Basic Inference Mechanisms for Logic Programs	810
25.5	Datalog Programs and Their Evaluation	813
25.6	Deductive Database Systems	827
25.7	Deductive Object-Oriented Databases	832
25.8	Applications of Commercial Deductive Database Systems	834
25.9	Summary	835
	Exercises	836
	Selected Bibliography	839
Chapter 26	Data Warehousing and Data Mining	841
26.1	Data Warehousing	842
26.2	Data Mining	855
26.3	Summary	870
	Review Exercises	871
	Selected Bibliography	872
Chapter 27	Emerging Database Technologies and Applications	73
27.1	Databases on the World Wide Web	874
27.2	Multimedia Databases	880
27.3	Mobile Databases	886
27.4	Geographic Information Systems	891
27.5	Genome Data Management	898
27.6	Digital Libraries	905

Appendix A	Alternative Diagrammatic Notations	909
Appendix B	Parameters of Disks	913
Appendix C	An Overview of the Network Data Model	917
Appendix D	An Overview of the Hierarchical Data Model	941
Selected Bibliography		B-1
Index		I-1

CHAPTER 7

The Relational Data Model, Relational Constraints, and the Relational Algebra

This chapter opens Part II of the book on relational databases. The relational model was first introduced by Ted Codd of IBM Research in 1970 in a classic paper [Codd 1970], and attracted immediate attention due to its simplicity and mathematical foundations. The model uses the concept of a *mathematical relation*—which looks somewhat like a table of values—as its basic building block, and has its theoretical basis in set theory and first order predicate logic. In this chapter we discuss the basic characteristics of the model, its constraints, and the relational algebra, which is a set of operations for the relational model. The model has been implemented in a large number of commercial systems over the last twenty or so years.

Because of the amount of material related to the relational model, we have devoted the whole of Part II of this textbook to it. In Chapter 8, we will describe the SQL query language, which is the *standard* for commercial relational DBMSs. Chapter 9 presents additional topics concerning relational databases. Sections 9.1 and 9.2 present algorithms for designing a relational database schema by mapping a conceptual schema in the ER or EER model (see Chapters 3 and 4) into a relational representation. These mappings are incorporated into many database design and CASE¹ tools. The remainder of Chapter 9 presents some other relational languages. Chapter 10 presents an overview of two commercial relational DBMSs—ORACLE and Microsoft ACCESS. Chapters 14 and 15 in Part IV of the

1. CASE stands for Computer Aided Software Engineering.

book present another aspect of the relational model, namely the formal constraints of functional and multivalued dependencies; these dependencies are used to develop a relational database design theory based on the concept known as *normalization*.

Data models that preceded the relational model include the hierarchical and network models. They were proposed in the sixties and were implemented in early DBMSs during the seventies and eighties. Because of their historical importance and the large existing user base for these DBMSs, we have included a summary of the highlights of these models in Appendices C and D. These models and systems will be with us for many years and are today being called *legacy systems*.

In this chapter, we will concentrate on describing the basic principles of the relational model of data. We begin by defining the modeling concepts and notation of the relational model in Section 7.1. Section 7.2 is devoted to a discussion of relational constraints that are now considered an important part of the relational model and are automatically enforced in most relational DBMSs. Section 7.3 defines the update operations of the relational model and discusses how violations of integrity constraints are handled.

In Section 7.4 we present a detailed discussion of the relational algebra, which is a collection of operations for manipulating relations and specifying queries. The relational algebra is an integral part of the relational data model. Section 7.5 defines additional relational operations that were added to the basic relational algebra because of their importance to many database applications. We give examples of specifying queries that use relational operations in Section 7.6. The same queries are used in subsequent chapters to illustrate various languages. Section 7.7 summarizes the chapter.

For the reader who is interested in a less detailed introduction to relational concepts, Sections 7.1.2, 7.4.7, and 7.5 may be skipped.

7.1 Relational Model Concepts

The relational model represents the database as a collection of *relations*. Informally, each relation resembles a table of values or, to some extent, a “flat” file of records. For example, the database of files that was shown in Figure 1.2 is considered to be in the relational model. However, there are important differences between relations and files, as we shall soon see.

When a relation is thought of as a table of values, each row in the table represents a collection of related data values. We introduced entity types and relationship types as concepts for modeling real-world data in Chapter 3. In the relational model, each row in the table represents a fact that typically corresponds to a real-world entity or relationship. The table name and column names are used to help in interpreting the meaning of the values in each row. For example, the first table of Figure 1.2 is called *STUDENT* because each row represents facts about a particular student entity. The column names—Name, StudentNumber, Class, Major—specify how to interpret the data values in each row, based on the column each value is in. All values in a column are of the same data type.

In the formal relational model terminology, a row is called a *tuple*, a column header is called an *attribute*, and the table is called a *relation*. The data type describing the types of

values that can appear in each column is called a *domain*. We now define these terms—*domain*, *tuple*, *attribute*, and *relation*—more precisely.

7.1.1 Domains, Attributes, Tuples, and Relations

A **domain** D is a set of **atomic values**. By **atomic** we mean that each value in the domain is **indivisible** as far as the relational model is concerned. A common method of specifying a domain is to **specify a data type from which the data values forming the domain are drawn**. It is also useful to specify a name for the domain, to help in interpreting its values. Some examples of domains follow:

- **USA_phone_numbers**: The set of 10-digit phone numbers valid in the United States.
- **Local_phone_numbers**: The set of 7-digit phone numbers valid within a particular area code in the United States.
- **Social_security_numbers**: The set of valid 9-digit social security numbers.
- **Names**: The set of names of persons.
- **Grade_point_averages**: Possible values of computed grade point averages; each must be a real (floating point) number between 0 and 4.
- **Employee_ages**: Possible ages of employees of a company; each must be a value between 15 and 80 years old.
- **Academic_department_names**: The set of academic department names, such as Computer Science, Economics, and Physics, in a university.
- **Academic_department_codes**: The set of academic department codes, such as CS, ECON, and PHYS, in a university.

The preceding are called **logical definitions of domains**. A **data type or format** is also specified for each domain. For example, the data type for the domain **USA_phone_numbers** can be declared as a character string of the form **(ddd)ddd-dddd**, where each d is a **numeric (decimal) digit** and the first three digits form a valid telephone area code. The data type for **Employee_ages** is an integer number between 15 and 80. For **Academic_department_names**, the data type is the set of all character strings that represent valid department names. A domain is thus given a name, data type, and format. Additional information for interpreting the values of a domain can also be given; for example, a numeric domain such as **Person_weights** should have the units of measurement—pounds or kilograms.

A **relation schema** R , denoted by $R(A_1, A_2, \dots, A_n)$, is made up of a **relation name** R and a list of attributes $A_1, A_2, \dots, A_n$. Each **attribute** A_i is the name of a role played by some domain D in the relation schema R . D is called the **domain** of A_i and is denoted by $\text{dom}(A_i)$. A relation schema is used to **describe a relation**; R is called the **name of this relation**. The **degree of a relation** is the number of attributes n of its relation schema.

An example of a relation schema for a relation of degree 7, which describes university students, is the following:

STUDENT(Name, SSN, HomePhone, Address, OfficePhone, Age, GPA)

STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

Figure 7.1 The attributes and tuples of a relation STUDENT.

For this relation schema, **STUDENT** is the name of the relation, which has seven attributes. We can specify the following previously defined domains for some of the attributes of the STUDENT relation: $\text{dom}(\text{Name}) = \text{Names}$; $\text{dom}(\text{SSN}) = \text{Social_security_numbers}$; $\text{dom}(\text{HomePhone}) = \text{Local_phone_numbers}$; $\text{dom}(\text{OfficePhone}) = \text{Local_phone_numbers}$, and $\text{dom}(\text{GPA}) = \text{Grade_point_averages}$.

A relation (or relation state)² r of the relation schema $R(A_1, A_2, \dots, A_n)$, also denoted by $r(R)$, is a set of n -tuples $r = \{t_1, t_2, \dots, t_m\}$. Each n -tuple t is an ordered list of n -values $t = \langle v_1, v_2, \dots, v_n \rangle$, where each value v_i , $1 \leq i \leq n$, is an element of $\text{dom}(A_i)$ or is a special null value. The i^{th} value in tuple t , which corresponds to the attribute A_i , is referred to as $t[A_i]$. The terms relation **intension** for the schema R and relation **extension** for a relation state $r(R)$ are also commonly used.

Figure 7.1 shows an example of a STUDENT relation, which corresponds to the STUDENT schema specified above. Each tuple in the relation represents a particular student entity. We display the relation as a table, where each tuple is shown as a row and each attribute corresponds to a column header indicating a role or interpretation of the values in that column. Null values represent attributes whose values are unknown or do not exist for some individual STUDENT tuples.

The above definition of a relation can be restated as follows. A relation $r(R)$ is a mathematical relation of degree n on the domains $\text{dom}(A_1), \text{dom}(A_2), \dots, \text{dom}(A_n)$, which is a subset of the Cartesian product of the domains that define R :

$$r(R) \subseteq (\text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n))$$

The Cartesian product specifies all possible combinations of values from the underlying domains. Hence, if we denote the number of values or cardinality of a domain D by $|D|$, and assume that all domains are finite, the total number of tuples in the Cartesian product is:

$$|\text{dom}(A_1)| * |\text{dom}(A_2)| * \dots * |\text{dom}(A_n)|$$

2. This has also been called a **relation instance**. We will not use this term because *instance* is also used to refer to a single tuple or row.

Out of all these possible combinations, a relation state at a given time—the **current relation state**—reflects only the valid tuples that represent a particular state of the real world. In general, as the state of the real world changes, so does the relation, by being transformed into another relation state. However, the schema *R* is relatively static and does not change except very infrequently—for example, as a result of adding an attribute to represent new information that was not originally stored in the relation.

It is possible for several attributes to *have the same domain*. The attributes indicate different roles, or interpretations, for the domain. For example, in the *STUDENT* relation, the same domain *Local_phone_numbers* plays the role of *HomePhone*, referring to the “home phone of a student,” and the role of *OfficePhone*, referring to the “office phone of the student.”

7.1.2 Characteristics of Relations

The earlier definition of relations implies certain characteristics that make a relation different from a file or a table. We now discuss some of these characteristics.

Ordering of Tuples in a Relation. A relation is defined as a set of tuples. Mathematically, elements of a set have *no order* among them; hence tuples in a relation do not have any particular order. However, in a file, records are physically stored on disk so there always is an order among the records. This ordering indicates first, second, *i*th, and last records in the file. Similarly, when we display a relation as a table, the rows are displayed in a certain order.

Tuple ordering is not part of a relation definition, because a relation attempts to represent facts at a logical or abstract level. Many logical orders can be specified on a relation; for example, tuples in the *STUDENT* relation in Figure 7.1 could be logically ordered by values of Name, SSN, Age, or some other attribute. The definition of a relation does not specify any order: there is *no preference* for one logical ordering over another. Hence, the relation displayed in Figure 7.2 is considered *identical* to the one shown in Figure 7.1. When a relation is implemented as a file, a physical ordering may be specified on the records of the file.

Ordering of Values within a Tuple, and an Alternative Definition of a Relation. According to the preceding definition of a relation, an *n*-tuple is an *ordered list* of *n* values, so the ordering of values in a tuple—and hence of attributes in a relation schema

STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21

Figure 7.2 The relation *STUDENT* from Figure 7.1, with a different order of tuples.

$t = \langle (\text{Name, Dick Davidson}), (\text{ssn, 422-11-2320}), (\text{HomePhone, null}), (\text{Address, 3452 Elgin Road}), (\text{OfficePhone, 749-1253}), (\text{Age, 25}), (\text{GPA, 3.53}) \rangle$

$t = \langle (\text{Address, 3452 Elgin Road}), (\text{Name, Dick Davidson}), (\text{ssn, 422-11-2320}), (\text{Age, 25}), (\text{OfficePhone, 749-1253}), (\text{GPA, 3.53}), (\text{HomePhone, null}) \rangle$

Figure 7.3 Two identical tuples when order of attributes and values is not part of the definition of a relation.

definition—is important. However, at a logical level, the order of attributes and their values are not really important as long as the correspondence between attributes and values is maintained.

An **alternative definition** of a relation can be given, making the ordering of values in a tuple unnecessary. In this definition, a relation schema $R = \{A_1, A_2, \dots, A_n\}$ is a set of attributes, and a relation $r(R)$ is a finite set of mappings $r = \{t_1, t_2, \dots, t_m\}$, where each tuple t_i is a mapping from R to D , and D is the union of the attribute domains; that is, $D = \text{dom}(A_1) \cup \text{dom}(A_2) \cup \dots \cup \text{dom}(A_n)$. In this definition, $t[A_i]$ must be in $\text{dom}(A_i)$ for $1 \leq i \leq n$ for each mapping t in r . Each mapping t_i is called a tuple.

According to this definition, a tuple can be considered as a set of $\langle \text{attribute}, \text{value} \rangle$ pairs, where each pair gives the value of the mapping from an attribute A_i to a value v_i from $\text{dom}(A_i)$. The ordering of attributes is not important, because the attribute name appears with its value. By this definition, the two tuples shown in Figure 7.3 are identical. This makes sense at an abstract or logical level, since there really is no reason to prefer having one attribute value appear before another in a tuple.

When a relation is implemented as a file, the attributes are physically ordered as fields within a record. We will use the first definition of relation, where the attributes and the values within tuples are ordered, because it simplifies much of the notation. However, the alternative definition given here is more general.

Values in the Tuples. Each value in a tuple is an atomic value; that is, it is not divisible into components within the framework of the basic relational model. Hence, composite and multivalued attributes (see Chapter 3) are not allowed. Much of the theory behind the relational model was developed with this assumption in mind, which is called the first normal form assumption.³ Multivalued attributes must be represented by separate relations, and composite attributes are represented only by their simple component attributes. Recent research in the relational model attempts to remove these restrictions by using the concept of nonfirst normal form or nested relations (see Chapter 13).

The values of some attributes within a particular tuple may be unknown or may not apply to that tuple. A special value, called null, is used for these cases. For example, in Figure 7.1, some student tuples have null for their office phones because they do not have an office (that is, office phone does not apply to these students). Another student has a

3. We discuss this assumption in more detail in Chapter 14.

null for home phone, presumably because either he does not have a home phone or he has one but we do not know it (value is *unknown*). In general, we can have *several types* of null values, such as “value unknown,” “value exists but not available,” or “attribute does not apply to this tuple.” It is possible to devise different codes for different types of null values. Incorporating different types of null values into relational model operations has proved difficult, and a full discussion is outside the scope of this book.

Interpretation of a Relation. The relation schema can be interpreted as a declaration or a type of **assertion**. For example, the schema of the STUDENT relation of Figure 7.1 asserts that, in general, a student entity has a Name, SSN, HomePhone, Address, OfficePhone, Age, and GPA. Each tuple in the relation can then be interpreted as a fact or a particular instance of the assertion. For example, the first tuple in Figure 7.1 asserts the fact that there is a STUDENT whose name is Benjamin Bayer, SSN is 305-61-2435, Age is 19, and so on.

Notice that some relations may represent facts about entities, whereas other relations may represent facts about relationships. For example, a relation schema MAJORS (StudentSSN, DepartmentCode) asserts that students major in academic departments; a tuple in this relation relates a student to his or her major department. Hence, the relational model represents facts about both entities and relationships *uniformly* as relations.

An alternative interpretation of a relation schema is as a **predicate**; in this case, the values in each tuple are interpreted as values that satisfy the predicate. This interpretation is quite useful in the context of logic programming languages, such as PROLOG, because it allows the relational model to be used within these languages. This is further discussed in Chapter 25 when we discuss deductive databases.

7.1.3 Relational Model Notation

We will use the following notation in our presentation:

- A relation schema R of degree n is denoted by $R(A_1, A_2, \dots, A_n)$.
- An n -tuple t in a relation $r(R)$ is denoted by $t = \langle v_1, v_2, \dots, v_n \rangle$, where v_i is the value corresponding to attribute A_i . The following notation refers to **component values** of tuples:
 - Both $t[A_i]$ and $t.A_i$ refer to the value v_i in t for attribute A_i .
 - Both $t[A_u, A_w, \dots, A_z]$ and $t.(A_u, A_w, \dots, A_z)$, where $A_u, A_w, \dots, A_z$ is a list of attributes from R , refer to the subtuple of values $\langle v_u, v_w, \dots, v_z \rangle$ from t corresponding to the attributes specified in the list.
- The letters Q, R, S denote relation names.
- The letters q, r, s denote relation states.
- The letters t, u, v denote tuples.
- In general, the name of a relation schema such as STUDENT *also indicates* the current set of tuples in that relation—the *current relation state*—whereas STUDENT(Name, SSN, ...) refers *only* to the relation schema.

APPENDIX

An Overview of the Hierarchical Data Model

This appendix provides an overview of the hierarchical data model.¹ There are no original documents that describe the hierarchical model, as there are for the relational and network models. The principles behind the hierarchical model are derived from Information Management System (IMS), which is the dominant hierarchical system in use today by a large number of banks, insurance companies, and hospitals as well as several government agencies. Another popular hierarchical DBMS is MRI's System-2000 (which was later sold by SAS Institute).

In this appendix we present the concepts for modeling hierarchical schemas and instances, the concept of a virtual parent-child relationship, which is used to overcome the limitations of pure hierarchies, and the constraints on the hierarchical model. A few examples of data manipulation are included.

1. The complete chapter on the hierarchical data model and the IMS system from the second edition of this book is available at the Web site for the book. This appendix is an edited excerpt of that chapter.

D.1 Hierarchical Database Structures

D.1.1 Parent-Child Relationships and Hierarchical Schemas

The hierarchical model employs two main data structuring concepts: records and parent-child relationships. A **record** is a collection of **field values** that provide information on an entity or a relationship instance. Records of the same type are grouped into **record types**. A record type is given a name, and its structure is defined by a collection of named **fields** or **data items**. Each field has a certain data type, such as integer, real, or string.

A **parent-child relationship type (PCR type)** is a 1:N relationship between two record types. The record type on the 1-side is called the **parent record type**, and the one on the N-side is called the **child record type** of the PCR type. An **occurrence** (or **instance**) of the PCR type consists of *one record* of the parent record type and a number of records (zero or more) of the child record type.

A **hierarchical database schema** consists of a number of hierarchical schemas. Each **hierarchical schema** (or **hierarchy**) consists of a number of record types and PCR types.

A hierarchical schema is displayed as a **hierarchical diagram**, in which record type names are displayed in rectangular boxes and PCR types are displayed as lines connecting the parent record type to the child record type. Figure D.1 shows a simple hierarchical diagram for a hierarchical schema with three record types and two PCR types. The record types are DEPARTMENT, EMPLOYEE, and PROJECT. Field names can be displayed under each record type name, as shown in Figure D.1. In some diagrams, for brevity, we display only the record type names.

We refer to a PCR type in a hierarchical schema by listing the pair (parent record type, child record type) between parentheses. The two PCR types in Figure D.1 are (DEPARTMENT, EMPLOYEE) and (DEPARTMENT, PROJECT). Notice that PCR types *do not* have a name in the hierarchical model. In Figure D.1 each *occurrence* of the (DEPARTMENT, EMPLOYEE) PCR type relates one department record to the records of the *many* (zero or more) employees who work in that department. An *occurrence* of the (DEPARTMENT, PROJECT) PCR type relates a department record to the records of projects controlled by that department. Figure D.2 shows two PCR occurrences (or instances) for each of these two PCR types.

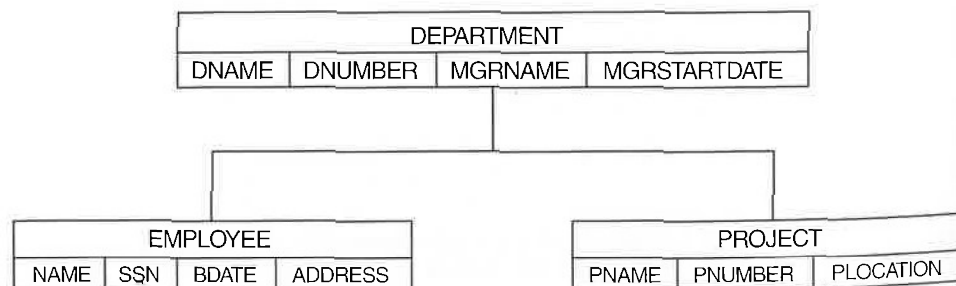


Figure D.1 A hierarchical schema.

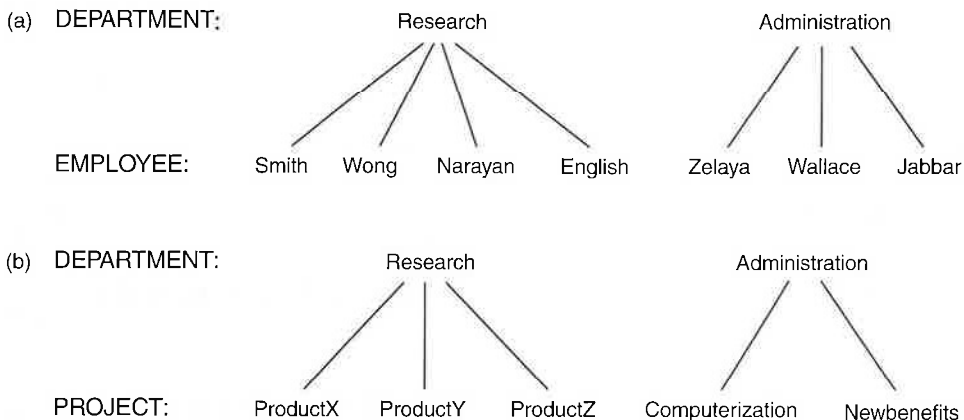


Figure D.2 Occurrences of Parent-Child Relationships. (a) Two occurrences of the PCR type (DEPARTMENT, EMPLOYEE). (b) Two occurrences of the PCR type (DEPARTMENT, PROJECT).

D.1.2 Properties of a Hierarchical Schema

A hierarchical schema of record types and PCR types must have the following properties:

1. One record type, called the **root** of the hierarchical schema, does not participate as a child record type in any PCR type.
2. Every record type except the root participates as a child record type in *exactly one* PCR type.
3. A record type can participate as parent record type in any number (zero or more) of PCR types.
4. A record type that does not participate as parent record type in any PCR type is called a **leaf** of the hierarchical schema.
5. If a record type participates as parent in more than one PCR type, then *its child record types are ordered*. The order is displayed, by convention, from left to right in a hierarchical diagram.

The definition of a hierarchical schema defines a **tree data structure**. In the terminology of tree data structures, a record type corresponds to a **node** of the tree, and a PCR type corresponds to an **edge** (or **arc**) of the tree. We use the terms *node* and *record type*, and *edge* and *PCR type*, interchangeably. The usual convention of displaying a tree is slightly different from that used in hierarchical diagrams, in that each tree edge is shown separately from other edges (Figure D.3). In hierarchical diagrams the convention is that all edges emanating from the same parent node are joined together (as in Figure D.1). We use this latter hierarchical diagram convention.

The preceding properties of a hierarchical schema mean that every node except the root has exactly one parent node. However, a node can have several child nodes, and in

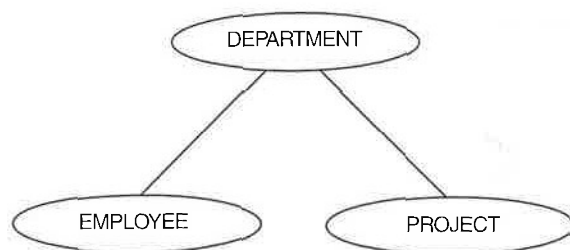


Figure D.3 A tree representation of the hierarchical schema in Figure D.1.

this case they are ordered from left to right. In Figure D.1 EMPLOYEE is the first child of DEPARTMENT, and PROJECT is the second child. The previously identified properties also limit the types of relationships that can be represented in a hierarchical schema. In particular, M:N relationships between record types *cannot* be directly represented, because parent-child relationships are 1:N relationships, and a record type *cannot participate as child* in two or more distinct parent-child relationships.

An M:N relationship may be handled in the hierarchical model by allowing duplication of child record instances. For example, consider an M:N relationship between EMPLOYEE and PROJECT, where a project can have several employees working on it, and an employee can work on several projects. We can represent the relationship as a (PROJECT, EMPLOYEE) PCR type. In this case a record describing the same employee can be duplicated by appearing once under *each* project that the employee works for. Alternatively, we can represent the relationship as an (EMPLOYEE, PROJECT) PCR type, in which case project records may be duplicated.

EXAMPLE 1: Consider the following instances of the EMPLOYEE:PROJECT relationship:

Project	Employees Working on the Project
A	E1, E3, E5
B	E2, E4, E6
C	E1, E4
D	E2, E3, E4, E5

If these instances are stored using the hierarchical schema (PROJECT, EMPLOYEE) (with PROJECT as the parent), there will be four occurrences of the (PROJECT, EMPLOYEE) PCR type—one for each project. The employee records for E1, E2, E3, and E5 will appear *twice each as child records, however, because each of these employees works on two projects*. The employee record for E4 will appear *three times—once under each of projects B, C,*

and D and may have number of hours that E4 works on each project in the corresponding instance.

To avoid such duplication, a technique is used whereby several hierarchical schemas can be specified in the same hierarchical database schema. Relationships like the preceding PCR type can now be defined across different hierarchical schemas. This technique, called **virtual relationships**, causes a departure from the "strict" hierarchical model. We discuss this technique in Section D.2.

D.1.3 Hierarchical Occurrence Trees

Corresponding to a hierarchical schema, many hierarchical occurrences, also called **occurrence trees**, exist in the database. Each one is a **tree structure** whose root is a single record from the root record type. The occurrence tree also contains all the children record occurrences of the root record and continues all the way to records of the leaf record types.

For example, consider the hierarchical diagram shown in Figure D.4, which represents part of the COMPANY database introduced in Chapter 3 and also used in Chapters 7, 8, and 9. Figure D.5 shows one hierarchical occurrence tree of this hierarchical schema. In the occurrence tree, each node is a **record occurrence**, and each arc represents a parent-child relationship between two records. In both Figures D.4 and D.5, we use the characters D, E, P, T, S, and W to represent **type indicators** for the record types DEPARTMENT, EMPLOYEE, PROJECT, DEPENDENT, SUPERVISEE, and WORKER, respectively. A node N and all its descendent nodes form a **subtree** of node N. An **occurrence tree** can be defined as the subtree of a record whose type is of the root record type.

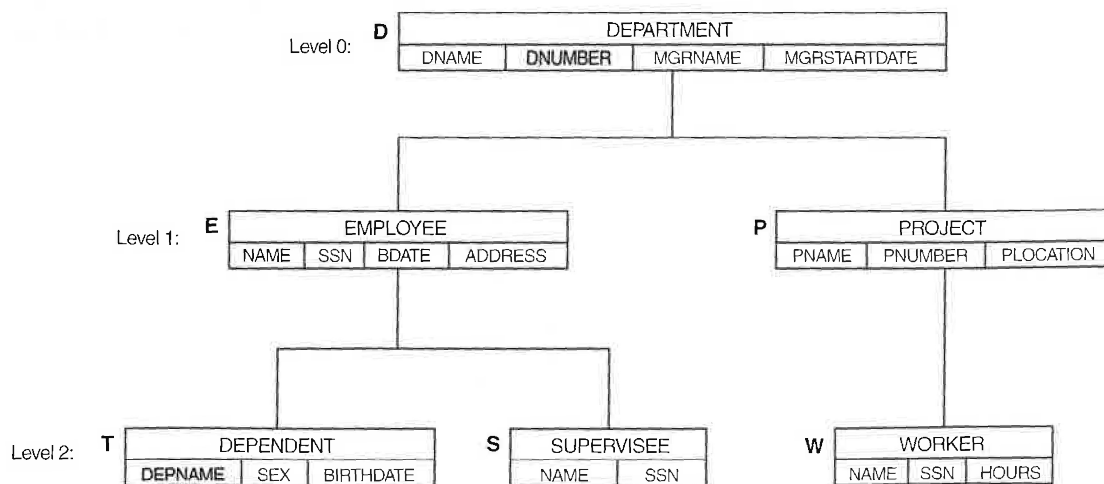


Figure D.4 A hierarchical schema for part of the COMPANY database.

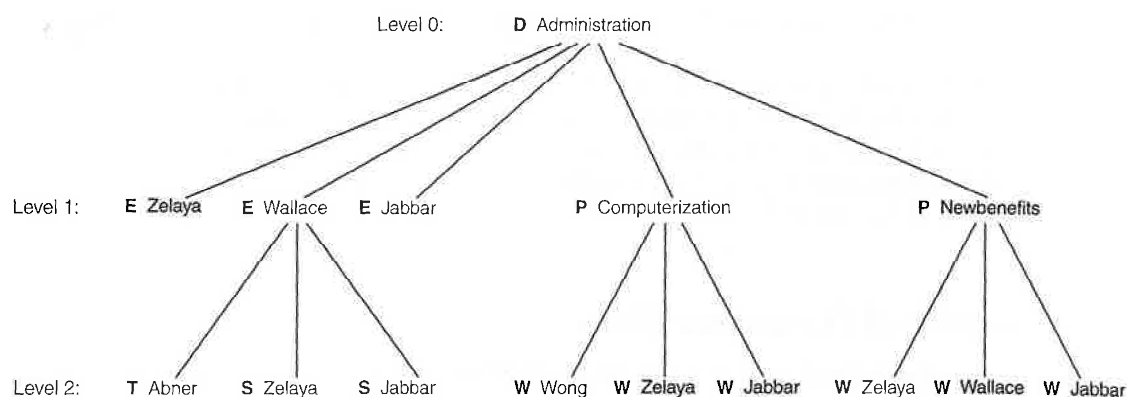


Figure D.5 An occurrence tree of the schema in Figure D.4.

D.1.4 Linearized Form of a Hierarchical Occurrence Tree

A hierarchical occurrence tree can be represented in storage by using any of a variety of data structures. However, a particularly simple storage structure that can be used is the **hierarchical record**, which is a linear ordering of the records in an occurrence tree in the **preorder traversal** of the tree. This order produces a sequence of record occurrences known as the **hierarchical sequence** (or **hierarchical record sequence**) of the occurrence tree; it can be obtained by applying a recursive procedure called the **pre-order traversal**, which visits nodes depth first and in a left-to-right fashion.

The occurrence tree in Figure D.5 gives the hierarchical sequence shown in Figure D.6. The system stores the type indicator with each record so that the record can be distinguished within the hierarchical sequence. The hierarchical sequence is also important because hierarchical data-manipulation languages, such as that used in IMS, use it as a basis for defining hierarchical database operations. The HDML language we discuss in

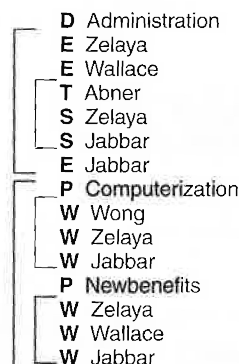


Figure D.6 Hierarchical sequence for the occurrence tree in Figure D.5.

Section D.3 (which is a simplified version of DL/1 of IMS) is based on the hierarchical sequence. A **hierarchical path** is a sequence of nodes $N_1, N_2, \dots, N_i$, where N_1 is the root of a tree and N_j is a child of N_{j-1} for $j = 2, 3, \dots, i$. A hierarchical path can be defined either on a hierarchical schema or on an occurrence tree. We can now define a **hierarchical database occurrence** as a sequence of all the occurrence trees that are occurrences of a hierarchical schema. For example, a hierarchical database occurrence of the hierarchical schema shown in Figure D.4 would consist of a number of occurrence trees similar to the one shown in Figure D.5, one for each distinct department.

D.1.5 Virtual Parent-Child Relationships

The hierarchical model has problems when modeling certain types of relationships. These include the following relationships and situations:

1. M:N relationships.
2. The case where a record type participates as child in more than one PCR type.
3. N-ary relationships with more than two participating record types.

Notice that the relationship between EMPLOYEE and EPOINTER in Figure D.7(a) is a 1:N relationship and hence qualifies as a PCR type. Such a relationship is called a **virtual parent-child relationship (VPCR) type**.² EMPLOYEE is called the **virtual parent** of EPOINTER; and conversely, EPOINTER is called a **virtual child** of EMPLOYEE. Conceptually, PCR types and VPCR types are similar. The main difference between the two lies in the way they are implemented. A PCR type is usually implemented by using the hierarchical

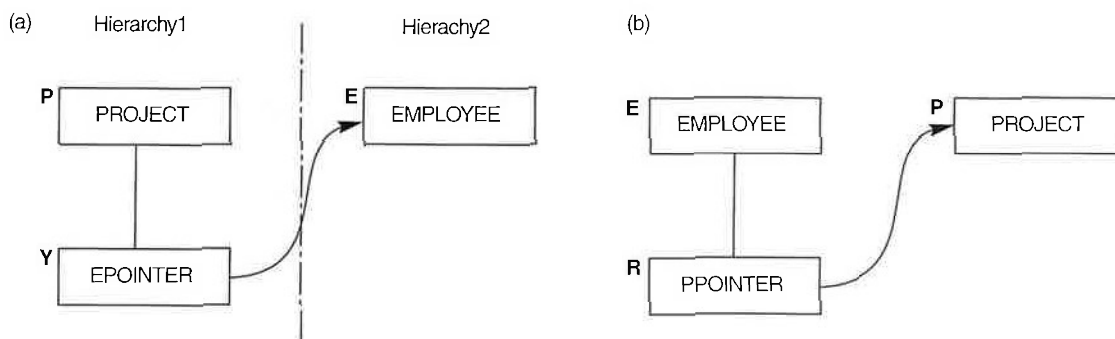


Figure D.7 Representing M:N relationships using Virtual Parent Child Relationships. (a) EMPLOYEE as virtual parent. (b) PROJECT as virtual parent.

2. The term "virtual" is not used in the IMS system, but it is used here to simplify the distinction between hierarchical relationships within one hierarchy (called Physical in IMS) and across hierarchies (called Logical in IMS).

sequence, whereas a VPCR type is usually implemented by establishing a pointer (a physical one containing an address, or a logical one containing a key) from a virtual child record to its virtual parent record. This mainly affects the efficiency of certain queries.

Figure D.8 shows a hierarchical database schema of the COMPANY database that uses some VPCRs and has no redundancy in its record occurrences. The hierarchical database schema is made up of two hierarchical schemas—one with root DEPARTMENT, and the other with root EMPLOYEE. Four VPCRs, all with virtual parent EMPLOYEE, are included to represent the relationships without redundancy. Notice that IMS may not allow this because an implementation constraint in IMS limits a record to being virtual parent of at most one VPCR; to get around this constraint, one can create dummy children record types of EMPLOYEE in Hierarchy 2 so that each VPCR points to a distinct virtual parent record type.

In general, there are many feasible methods of designing a database using the hierarchical model. In many cases, performance considerations are the most important factor in choosing one hierarchical database schema over another. Performance depends

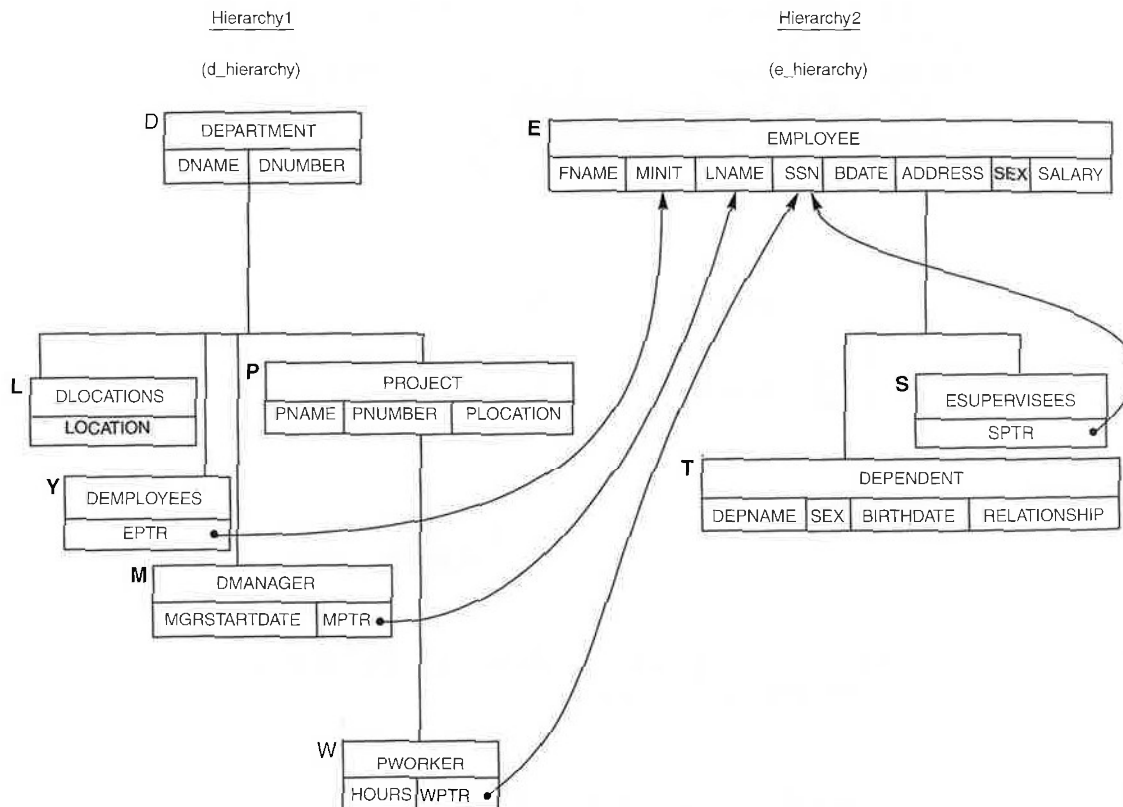


Figure D.8 Using VPCR types to eliminate redundancy in the COMPANY database.

on the implementation options available—for example, whether certain types of pointers are provided by the system and whether certain limits on number of levels are imposed by the DBA.

D.2 Integrity Constraints and Data Definition in the Hierarchical Model

D.2.1 Integrity Constraints in the Hierarchical Model

A number of built-in **inherent constraints** exist in the hierarchical model whenever we specify a hierarchical schema. These include the following constraints:

1. No record occurrences except root records can exist without being related to a parent record occurrence. This has the following implications:
 - a. A child record cannot be inserted unless it is linked to a parent record.
 - b. A child record may be deleted independently of its parent; however, deletion of a parent record automatically results in deletion of all its child and descendent records.
 - c. The above rules do not apply to virtual child records and virtual parent records.
2. If a child record has two or more parent records from the *same* record type, the child record must be duplicated once under each parent record.
3. A child record having two or more parent records of *different* record types can do so only by having at most one real parent, with all the others represented as virtual parents. IMS limits the number of virtual parents to one.
4. In IMS, a record type can be the virtual parent in *only one* VPCR type. That is, the number of virtual children can be only one per record type in IMS.

D.2.2 Data Definition in the Hierarchical Model

In this section we give an example of a hierarchical data definition language (HDDL), which is *not* the language of any specific hierarchical DBMS but is used to illustrate the language concepts for a hierarchical database. The HDDL demonstrates how a hierarchical database schema can be defined. To define a hierarchical database schema, we must define the fields of each record type, the data type of each field, and any key constraints on fields. In addition, we must specify a root record type as such; and for every nonroot record type, we must specify its (real) parent in a PCR type. Any VPCR types must also be specified.

In Figure D.9, either each record type is declared to be of type root or a single (real) parent record type is declared for the record type. The data items of the record are then listed along with their data types. We must specify a virtual parent for data items that are of type *pointer*. Data items declared under the KEY clause are constrained to have unique values for each record. Each KEY clause specifies a separate key; in addition, if a single KEY clause lists more than one field, the combination of these field values must be unique in

“An excellent update of a book long noted for its outstanding depth and breadth.”

—Emilia E. Villarreal

California Polytechnic State University, San Luis Obispo

“This is the only database book that has full coverage of what I consider to be the required introductory topics in the design, implementation, and management of databases and database systems.”

—Joan Peckham

University of Rhode Island

“... the most up-to-date database technologies and leading topics in the frontier of database research at the turn of a new millennium.”

—Junping Sun

Nova Southeastern University

Fundamentals of Database Systems combines clear explanations of theory and design, broad coverage of models and real systems, and excellent examples with up-to-date introductions to modern database technologies. This edition is completely revised and updated, and reflects the latest trends in technological and application development. Professors Elmasri and Navathe focus on the relational model and include coverage of recent object-oriented developments. They also address advanced modeling and system enhancements in the areas of active databases, temporal and spatial databases, and multimedia information systems. This edition also surveys the latest application areas of data warehousing, data mining, web databases, digital libraries, GIS, and genome databases.

New to the Third Edition

- Includes reorganized material on data modeling to clearly separate entity relationship modeling, extended entity relationship modeling, and object-oriented modeling
- Presents expanded coverage of the object-oriented and object/relational approach to data management, including ODMG and SQL3
- Uses examples from real database systems including Oracle™ and Microsoft Access®
- Includes discussion of decision support applications of data warehousing and data mining, as well as emerging technologies of web databases, multimedia, and mobile databases
- Covers advanced modeling in the areas of active, temporal, and spatial databases
- Provides coverage of issues of physical database tuning
- Discusses current database application areas of GIS, genome, and digital libraries

Access the latest information about Addison-Wesley books from our World Wide Web site at www.awlonline.com



Addison-Wesley is an imprint
of Addison Wesley Longman, Inc.

9 0000
Sterling EN2004
9 780805 317558 PR2025-00270
ISBN 0-8053-1755-4 Page 33 of 33