FUNDAMENTALS OF DATABASE SYSTEMS

Third Edition



ELMASRI | NAVATHE

Sterling EX2011
IPR2025-00270
Page 1 of 69

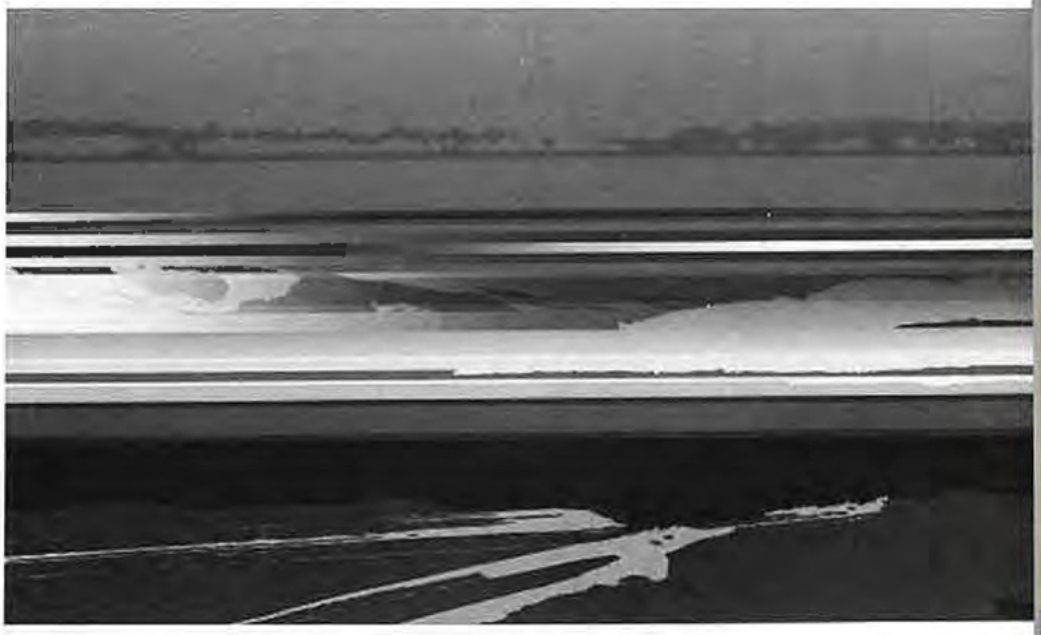
FUNDAMENTALS OF DATABASE SYSTEMS

Third Edition

Sterling EX2011
IPR2025-00270
Page 2 of 69

FUNDAMENTALS OF DATABASE SYSTEMS

Third Edition



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 EX2011
IPR2025-00270
Page 3 of 69

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 99-30620
005.74—dc21 CIP

ISBN 0-8053-1755-4

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

Sterling EX2011
IPR2025-00270
Page 4 of 69

To my wife, Amalia
and to my family.
R. E.

Dedicated to my three greatest teachers,
Father Shri. B. S. Navathe
and grandfathers
Shri. S. N. Navathe and Shri. G. R. Sardesai
and
to my wife Aruna for her infinite patience
S. B. N.

Sterling EX2011
IPR2025-00270
Page 5 of 69

PREFACE

This book introduces the fundamental concepts necessary for designing, using, and implementing database systems and applications. Our presentation stresses the fundamentals of database modeling and design, the languages and facilities provided by database management systems, and system implementation techniques. The book is meant to be used as a textbook for a one- or two-semester course in database systems at the junior, senior, or graduate level, and as a reference book. We assume that readers are familiar with elementary programming and data-structuring concepts and that they have had some exposure to basic computer organization.

We start in Part 1 with an introduction and a presentation of the basic concepts from both ends of the database spectrum—conceptual modeling principles and physical file storage techniques. We conclude the book in Part 6 with an introduction to influential new database models, such as active, temporal, and deductive models, along with an overview of emerging technologies and applications, such as data mining, data warehousing, and Web databases. Along the way—in Parts 2 through 5—we provide an in-depth treatment of the most important aspects of database fundamentals.

The following key features are included in the third edition:

- The entire book has a self-contained, flexible organization that can be tailored to individual needs.
- Complete and updated coverage is provided on the relational model—including new material on Oracle and Microsoft Access as examples of relational systems—in Part 2.

Sterling EX2011
IPR2025-00270
Page 6 of 69

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

Sterling EX2011
IPR2025-00270
Page 7 of 69

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

Sterling EX2011
IPR2025-00270
Page 8 of 69

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 Stotage 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 (Sorred Files) 136
5.9 Hashing Techniques 139
5.10 Other Primary File Organizarions 148
5.11 Summary 149
Review Questions 149
Exercises 150
Selecred Bibliography 153

Chapter 6 Index Structures for Files 155

6.1 Types of Single-Level Ordered Indexe 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

Sterling EX2011
IPR2025-00270
Page 9 of 69

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

Sterling EX2011
IPR2025-00270
Page 10 of 69

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 Funcrionality of Access 351
10.9 Summary 355
Selected Bibliography 355

Sterling EX2011
IPR2025-00270
Page 11 of 69

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

Sterling EX2011
IPR2025-00270
Page 12 of 69

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 EX2011
IPR2025-00270
Page 13 of 69

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

Sterling EX2011
IPR2025-00270
Page 14 of 69

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 672

Sterling EX2011
IPR2025-00270
Page 15 of 69

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 Authorizatiou 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

Sterling EX2011
IPR2025-00270
Page 16 of 69

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

Sterling EX2011
IPR2025-00270
Page 17 of 69

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

Sterling EX2011
IPR2025-00270
Page 18 of 69

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

Sterling EX2011
IPR2025-00270
Page 19 of 69

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 CASE[1] 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.

Sterling EX2011
IPR2025-00270
Page 20 of 69

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

Sterling EX2011
IPR2025-00270
Page 21 of 69

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.

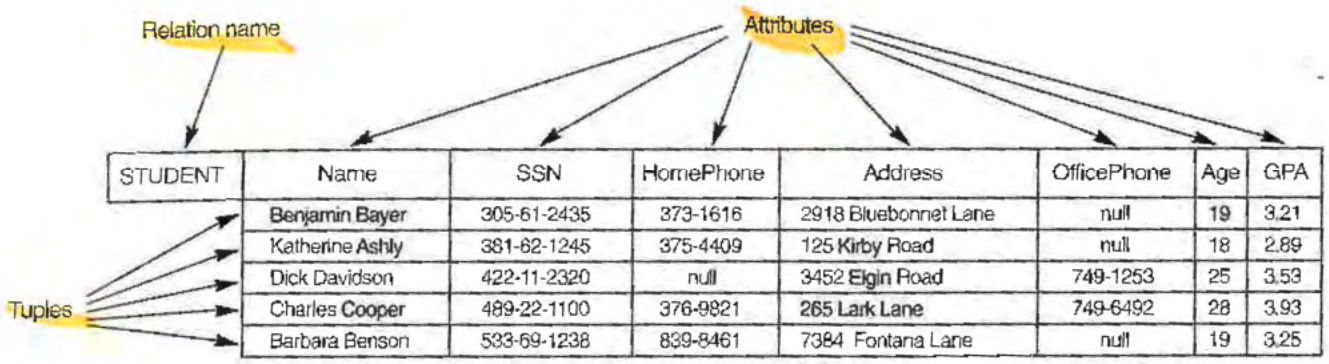
The preceding are called *logical* definitions of domains. A **data type or format** is also specified for each domain. For example, the data type for the domain USA_phone_numbers can be declared as a character string of the form (ddd)ddd-dddd, where each d is a numeric (decimal) digit and the first 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 $R(A_1, A_2, \ldots, A_n)$, is made up of a relation name R and a list of attributes $A_1, A_2, \ldots, A_n$. 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 $dom(A_i)$. 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)

Sterling EX2011
IPR2025-00270
Page 22 of 69



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: dom(Name) = Names; dom(SSN) = Social_security_numbers; dom(HomePhone) = Local_phone_numbers, dom(OfficePhone) = Local_phone_numbers, and dom(GPA) = Grade_point_averages.

A **relation (or relation state)**[2] r of the relation schema $R(A_1, A_2, \ldots, A_n)$, also denoted by r(R), is a set of n-tuples $r = \{t_1, t_2, \ldots, t_m\}$. Each **n-tuple** t is an ordered list of n values $t = \langle v_1, v_2, \ldots, v_n \rangle$, where each value v_i, $1 \leq i \leq n$, is an element of $dom(A_i)$ 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 $dom(A_1)$, $dom(A_2)$, . . ., $dom(A_n)$, which is a **subset** of the **Cartesian product** of the domains that define R:

$$r(R) \subseteq (dom(A_1) \times dom(A_2) \times \ldots \times dom(A_n))$$

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:

$$|dom(A_1)| * |dom(A_2)| * \ldots * |dom(A_n)|$$

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.

Sterling EX2011
IPR2025-00270
Page 23 of 69

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.

Sterling EX2011
IPR2025-00270
Page 24 of 69

t = < (Name, Dick Davidson),(SSN, 422-11-2320),(HomePhone, null),(Address, 3452 Elgin Road),
(OfficePhone, 749-1253),(Age, 25),(GPA, 3.53)>

t = < (Address, 3452 Elgin Road),(Name, Dick Davidson),(SSN, 422-11-2320),(Age, 25),
(OfficePhone,749-1253),(GPA, 3.53),(HomePhone, null)>

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 $R = \{A_1, A_2, \ldots, A_n\}$ is a set of attributes, and a relation $r(R)$ is a finite set of **mappings** $r = \{t_1, t_2, \ldots, t_m\}$, where each tuple t_i is a mapping from R to D, and D is the union of the attribute domains; that is, $D = dom(A_1) \cup dom(A_2) \cup \ldots \cup dom(A_n)$. In this definition, $t[A_i]$ must be in $dom(A_i)$ 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 (<attribute>, <value>) pairs, where each pair gives the value of the mapping from an attribute A_i to a value v_i from $dom(A_i)$. 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.

Sterling EX2011
IPR2025-00270
Page 25 of 69

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 $R(A_1, A_2, \ldots, A_n)$.
- An n-tuple t in a relation r(R) is denoted by $t = <v_1, v_2, \ldots, v_n>$, 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 $t[A_u, A_w, \ldots, A_z]$ and $t.(A_u, A_w, \ldots, A_z)$, where $A_u, A_w, \ldots, A_z$ is a list of attributes from R, refer to the subtuple of values $<v_u, v_w, \ldots, v_z>$ 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.

Sterling EX2011
IPR2025-00270
Page 26 of 69

- An attribute A can be qualified with the relation name R to which it belongs by using the *dot notation* R.A—for example, STUDENT.Name or STUDENT.Age. This is because the same name may be used for two attributes in different relations. However, all attribute names *in a particular relation* must be distinct.

As an example, consider the tuple t = <'Barbara Benson', '533-69-1238', '839-8461', '7384 Fontana Lane', null, 19, 3.25> from the STUDENT relation in Figure 7.1; we have t[Name] = <'Barbara Benson'>, and t[SSN, GPA, Age] = <'533-69-1238', 3.25, 19>.

7.2 Relational Constraints and Relational Database Schemas

In this section, we discuss the various restrictions on data that can be specified on a relational database schema in the form of constraints. These include domain constraints, key constraints, entity integrity, and referential integrity constraints. Other types of constraints, called *data dependencies* (which include *functional dependencies* and *multivalued dependencies*), are used mainly for database design by normalization and will be discussed in Chapters 14 and 15.

7.2.1 Domain Constraints

Domain constraints specify that the value of each attribute A must be an atomic value from the domain dom(A). We have already discussed the ways in which domains can be specified in Section 7.1.1. The data types associated with domains typically include standard numeric data types for integers (such as short-integer, integer, long-integer) and real numbers (float and double-precision float). Characters, fixed-length strings, and variable-length strings are also available, as are date, time, timestamp, and money data types. Other possible domains may be described by a subrange of values from a data type or as an enumerated data type where all possible values are explicitly listed. Rather than describe these in detail here, we discuss the data types offered by the SQL2 relational standard in Section 8.1.2.

7.2.2 Key Constraints and Constraints on Null

A *relation* is defined as a *set of tuples*. By definition, all elements of a set are distinct; hence, all tuples in a relation must also be distinct. This means that no two tuples can have the same combination of values for *all* their attributes. Usually, there are other subsets of attributes of a relation schema R with the property that no two tuples in any relation state r of R should have the same combination of values for these attributes. Suppose that we denote one such subset of attributes by SK; then for any two *distinct* tuples t_1 and t_2 in a relation state r of R, we have the constraint that

$$t_1[SK] \neq t_2[SK]$$

Any such set of attributes SK is called a superkey of the relation schema R. A superkey SK specifies a *uniqueness constraint* that no two distinct tuples in a state r of R can have

Sterling EX2011
IPR2025-00270
Page 27 of 69

the same value for SK. Every relation has at least one default superkey—the set of all its attributes. A superkey can have redundant attributes, however, so a more useful concept is that of a *key*, which has no redundancy. A key K of a relation schema R is a superkey of R with the additional property that removing any attribute A from K leaves a set of attributes K' that is not a superkey of R. Hence, a key is a *minimal superkey*—that is, a superkey from which we cannot remove any attributes and still have the uniqueness constraint hold.

For example, consider the STUDENT relation of Figure 7.1. The attribute set {SSN} is a key of STUDENT because no two student tuples can have the same value for SSN.[4] Any set of attributes that includes SSN—for example, {SSN, Name, Age}—is a superkey. However, the superkey {SSN, Name, Age} is not a key of STUDENT, because removing Name or Age or both from the set still leaves us with a superkey.

The value of a key attribute can be used to identify uniquely each tuple in the relation. For example, the SSN value 305-61-2435 identifies uniquely the tuple corresponding to Benjamin Bayer in the STUDENT relation. Notice that a set of attributes constituting a key is a property of the relation schema; it is a constraint that should hold on *every* relation state of the schema. A key is determined from the meaning of the attributes, and the property is *time-invariant*; it must continue to hold when we insert new tuples in the relation. For example, we cannot and should not designate the Name attribute of the STUDENT relation in Figure 7.1 as a key, because there is no guarantee that two students with identical names will never exist.[5]

In general, a relation schema may have more than one key. In this case, each of the keys is called a **candidate key**. For example, the CAR relation in Figure 7.4 has two candidate keys: LicenseNumber and EngineSerialNumber. It is common to designate one of the candidate keys as the **primary key** of the relation. This is the candidate key whose values are used to *identify* tuples in the relation. We use the convention that the attributes that form the primary key of a relation schema are underlined, as shown in Figure 7.4. Notice that, when a relation schema has several candidate keys, the choice of one to

CAR

LicenseNumber	EngineSerialNumber	Make	Model	Year
Texas ABC-739	A69352	Ford	Mustang	96
Florida TVP-347	B43696	Oldsmobile	Cutlass	99
New York MPO-22	X83554	Oldsmobile	Delta	95
California 432-TFY	C43742	Mercedes	190-D	93
California RSK-629	Y82935	Toyota	Camry	98
Texas RSK-629	U028365	Jaguar	XJS	98

Figure 7.4 The CAR relation with two candidate keys: LicenseNumber and EngineSerialNumber.

4. Note that SSN is also a superkey.

5. Names are sometimes used as keys, but then some artifact—such as appending an ordinal number—must be used to distinguish between identical names.

Sterling EX2011
IPR2025-00270
Page 28 of 69

become primary key is arbitrary; however, it is usually better to choose a primary key with a single attribute or a small number of attributes.

Another constraint on attributes specifies whether null values are or are not permitted. For example, if every STUDENT tuple must have a valid, non-null value for the Name attribute, then Name of STUDENT is constrained to be NOT NULL.

7.2.3 Relational Databases and Relational Database Schemas

So far, we have discussed single relations and single relation schemas. A relational database usually contains many relations, with tuples in relations that are related in various ways. In this section we define a relational database and a relational database schema. A **relational database schema** S is a set of relation schemas $S = \{R_1, R_2, \ldots, R_m\}$ and a set of **integrity constraints** IC. A **relational database state**[6] DB of S is a set of relation states $DB = \{r_1, r_2, \ldots, r_m\}$ such that each r_i is a state of R_i and such that the r_i relation states satisfy the integrity constraints specified in IC. Figure 7.5 shows a relational database schema

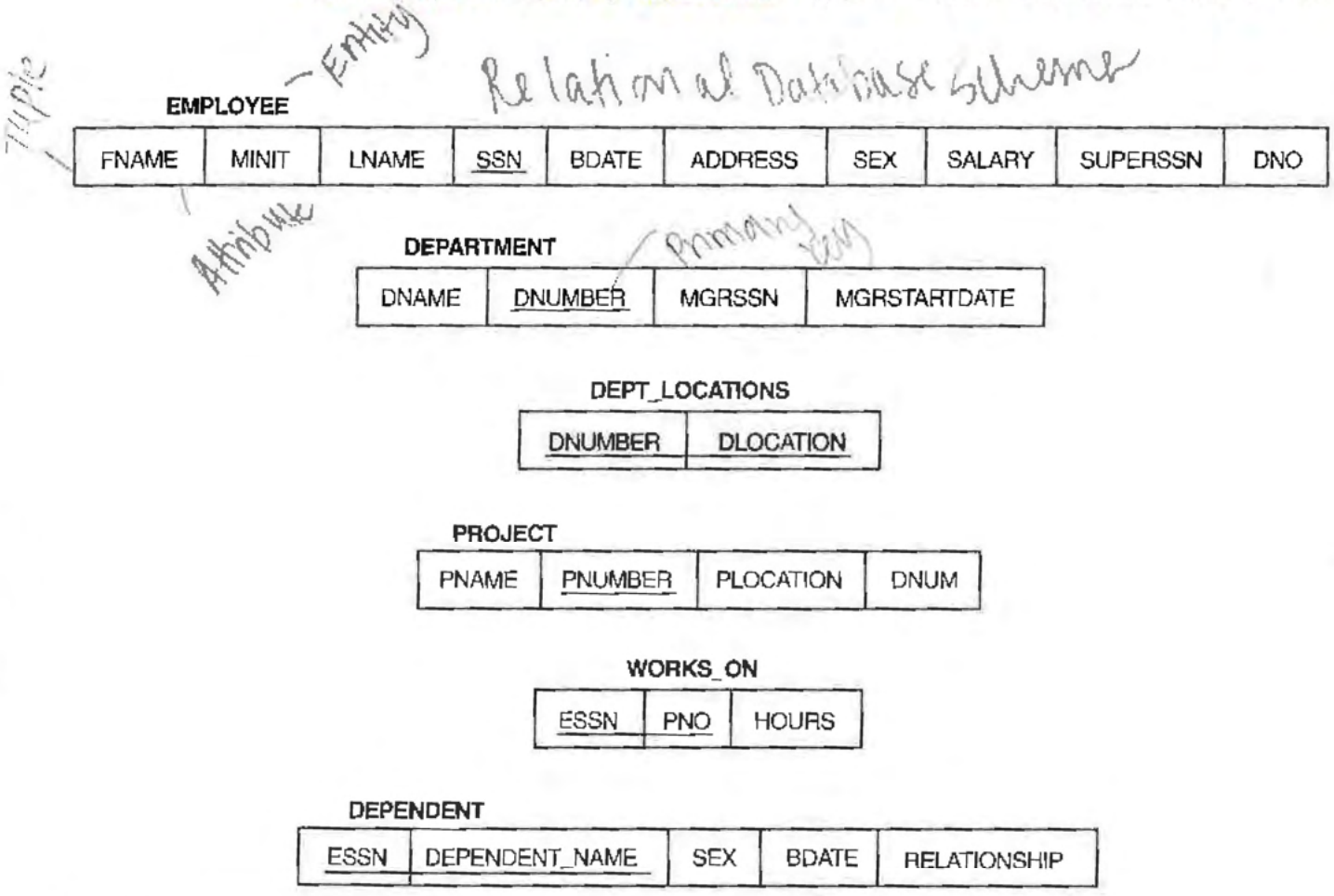


Figure 7.5 Schema diagram for the COMPANY relational database schema; the primary keys are underlined.

6. A relational database *state* is also called a relational database *instance*.

Sterling EX2011
IPR2025-00270
Page 29 of 69

Relational Db State

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE

Figure 7.6 One possible relational database state corresponding to the COMPANY schema.

that we call COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}. Figure 7.6 shows a relational database state corresponding to the COMPANY schema. We will use this schema and database state in this chapter and in Chapters 8 through 10 for defining example queries in different relational languages. When we refer to a relational database, we implicitly include both its schema and its current state.

Sterling EX2011
IPR2025-00270
Page 30 of 69

In Figure 7.5, the DNUMBER attribute in both DEPARTMENT and DEPT_LOCATIONS stands for the same real-world concept—the number given to a department. That same concept is called DNO in EMPLOYEE and DNUM in PROJECT. Attributes that represent the same real-world concept may or may not have identical names in different relations. Alternatively, attributes that represent different concepts may have the same name in different relations. For example, we could have used the attribute name NAME for both PNAME of PROJECT and DNAME of DEPARTMENT; in this case, we would have two attributes that share the same name but represent different real-world concepts—project names and department names.

In some early versions of the relational model, an assumption was made that the same real-world concept, when represented by an attribute, would have *identical* attribute names in all relations. This creates problems when the same real-world concept is used in different roles (meanings) in the same relation. For example, the concept of social security number appears twice in the EMPLOYEE relation of Figure 7.5: once in the role of the employee's social security number, and once in the role of the supervisor's social security number. We gave them distinct attribute names—SSN and SUPERSSN, respectively—in order to distinguish their meaning.

Each relational DBMS must have a Data Definition Language (DDL) for defining a relational database schema. Current relational DBMSs are mostly using SQL for this purpose. We present the SQL DDL in Section 8.1.

Integrity constraints are specified on a database schema and are expected to hold on every database state of that schema. In addition to domain and key constraints, two other types of constraints are considered part of the relational model: entity integrity and referential integrity.

7.2.4 Entity Integrity, Referential Integrity, and Foreign Keys

The **entity integrity constraint** states that no primary key value can be null. This is because the primary key value is used to identify individual tuples in a relation; having null values for the primary key implies that we cannot identify some tuples. For example, if two or more tuples had null for their primary keys, we might not be able to distinguish them.

Key constraints and entity integrity constraints are specified on individual relations. The **referential integrity constraint** is specified between two relations and is used to maintain the consistency among tuples of the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an *existing tuple* in that relation. For example, in Figure 7.6, the attribute DNO of EMPLOYEE gives the department number for which each employee works; hence, its value in every EMPLOYEE tuple must match the DNUMBER value of some tuple in the DEPARTMENT relation.

To define referential integrity more formally, we first define the concept of a *foreign key*. The conditions for a foreign key, given below, specify a referential integrity constraint between the two relation schemas R_1 and R_2. A set of attributes FK in relation

Sterling EX2011
IPR2025-00270
Page 31 of 69

schema R_1 is a **foreign key** of R_1 that references relation R_2 if it satisfies the following two rules:

1. The attributes in FK have the same domain(s) as the primary key attributes PK of R_2; the attributes FK are said to **reference** or refer to the relation R_2.
2. A value of FK in a tuple t_1 of the current state $r_1(R_1)$ either occurs as a value of PK for some tuple t_2 in the current state $r_2(R_2)$ or is null. In the former case, we have $t_1[FK] = t_2[PK]$, and we say that the tuple t_1 **references** or **refers to** the tuple t_2. R_1 is called the **referencing relation** and R_2 is the **referenced relation.**

In a database of many relations, there are usually many referential integrity constraints. To specify these constraints, we must first have a clear understanding of the meaning or role that each set of attributes plays in the various relation schemas of the database. Referential integrity constraints typically arise from the *relationships among the entities* represented by the relation schemas. For example, consider the database shown in Figure 7.6. In the EMPLOYEE relation, the attribute DNO refers to the department for which an employee works; hence, we designate DNO to be a foreign key of EMPLOYEE, referring to the DEPARTMENT relation. This means that a value of DNO in any tuple t_1 of the EMPLOYEE relation must match a value of the primary key of DEPARTMENT—the DNUMBER attribute—in some tuple t_2 of the DEPARTMENT relation, or the value of DNO *can be null* if the employee does not belong to a department. In Figure 7.6 the tuple for employee 'John Smith' references the tuple for the 'Research' department, indicating that 'John Smith' works for this department.

Notice that a foreign key can *refer to its own relation*. For example, the attribute SUPERSSN in EMPLOYEE refers to the supervisor of an employee; this is another employee, represented by a tuple in the EMPLOYEE relation. Hence, SUPERSSN is a foreign key that references the EMPLOYEE relation itself. In Figure 7.6 the tuple for employee 'John Smith' references the tuple for employee 'Franklin Wong,' indicating that 'Franklin Wong' is the supervisor of 'John Smith.'

We can *diagrammatically display referential integrity constraints* by drawing a directed arc from each foreign key to the relation it references. For clarity, the arrowhead may point to the primary key of the referenced relation. Figure 7.7 shows the schema in Figure 7.5 with the referential integrity constraints displayed in this manner.

All integrity constraints should be specified on the relational database schema if we want to enforce these constraints on the database states. Hence, the DDL includes provisions for specifying the various types of constraints so that the DBMS can automatically enforce them. Most relational DBMSs support key and entity integrity constraints, and make provisions to support referential integrity. These constraints are specified as a part of data definition.

The preceding integrity constraints do not include a large class of general constraints, sometimes called *semantic integrity constraints*, that may have to be specified and enforced on a relational database. Examples of such constraints are "the salary of an employee should not exceed the salary of the employee's supervisor" and "the maximum number of hours an employee can work on all projects per week is 56." Such constraints can be specified and enforced by using a general purpose **constraint specification language.** Mechanisms called

Sterling EX2011
IPR2025-00270
Page 32 of 69

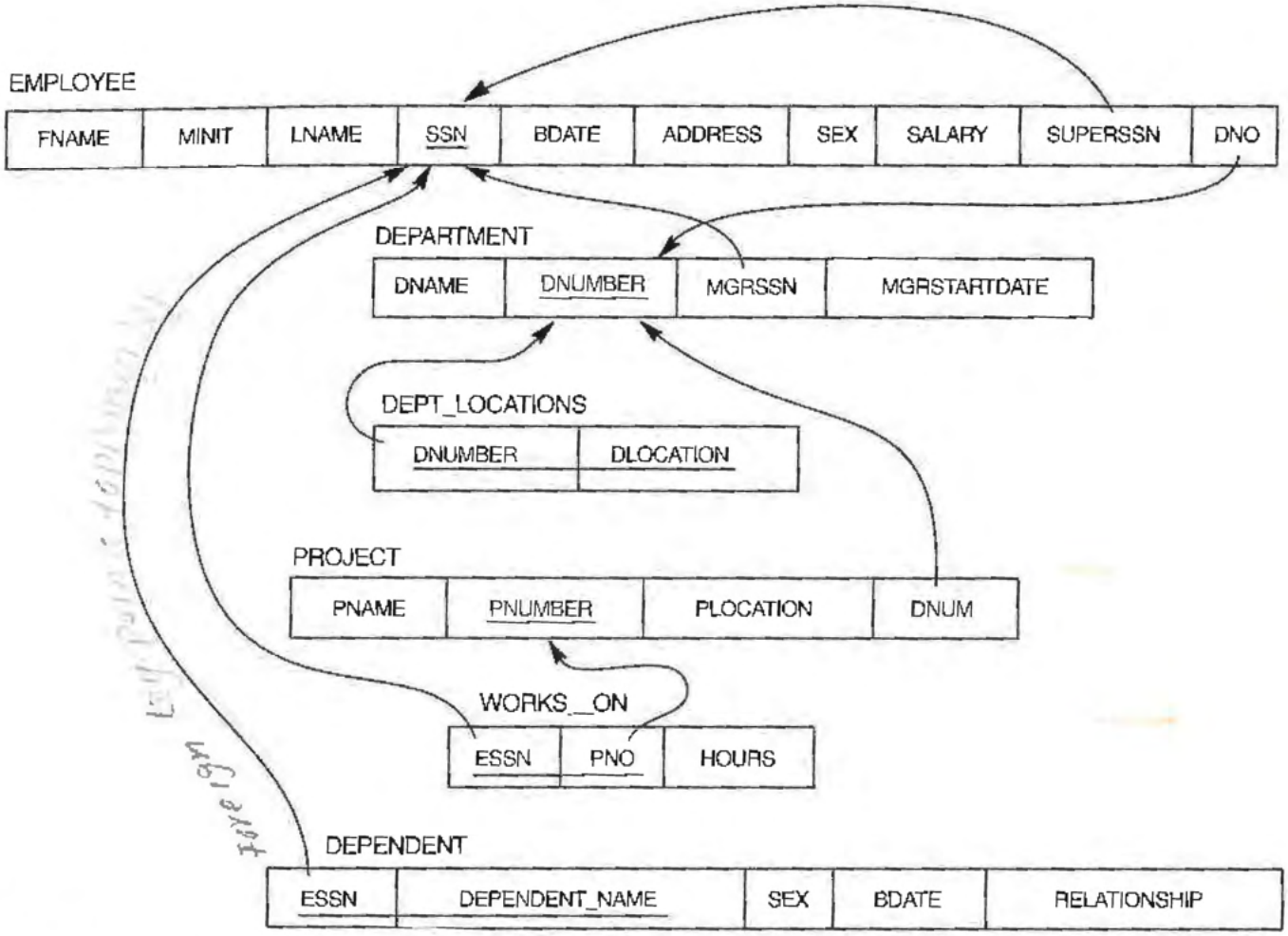


Figure 7.7 Referential integrity constraints displayed on the COMPANY relational database schema diagram.

triggers and assertions can be used. In SQL2, a CREATE ASSERTION statement is used for this purpose (see Chapters 8 and 23).

The types of constraints we discussed above may be termed as *state constraints*, because they define the constraints that a *valid state* of the database must satisfy. Another type of constraints, called *transition constraints*, can be defined to deal with state changes in the database.[7] An example of a transition constraint is: "the salary of an employee can only increase." Such constraints are typically specified using active rules and triggers, as we shall discuss in Chapter 23.

7. State constraints are also called *static constraints*, and transition constraints are called *dynamic constraints*.

Sterling EX2011
IPR2025-00270
Page 33 of 69

7.3 Update Operations and Dealing with Constraint Violations

The operations of the relational model can be categorized into *retrievals* and *updates*. The relational algebra operations, which can be used to specify retrievals, are discussed in detail in Section 7.4. In this section, we concentrate on the update operations. There are three basic update operations on relations: (1) insert, (2) delete, and (3) modify. **Insert** is used to insert a new tuple or tuples in a relation; **Delete** is used to delete tuples; and **Update** (or **Modify**) is used to change the values of some attributes in existing tuples. Whenever update operations are applied, the integrity constraints specified on the relational database schema should not be violated. In this section we discuss the types of constraints that may be violated by each update operation and the types of actions that may be taken if an update does cause a violation. We use the database shown in Figure 7.6 for examples and discuss only key constraints, entity integrity constraints, and the referential integrity constraints shown in Figure 7.7. For each type of update, we give some example operations and discuss any constraints that each operation may violate.

7.3.1 The Insert Operation

The **Insert** operation provides a list of attribute values for a new tuple t that is to be inserted into a relation R. Insert can violate any of the four types of constraints discussed in the previous section. Domain constraints can be violated if an attribute value is given that does not appear in the corresponding domain. Key constraints can be violated if a key value in the new tuple t already exists in another tuple in the relation r(R). Entity integrity can be violated if the primary key of the new tuple t is null. Referential integrity can be violated if the value of any foreign key in t refers to a tuple that does not exist in the referenced relation. Here are some examples to illustrate this discussion.

1. Insert <'Cecilia', 'F', 'Kolonsky', null, '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, null, 4> into EMPLOYEE.
   - This insertion violates the entity integrity constraint (null for the primary key SSN), so it is rejected.
2. Insert <'Alicia', 'J', 'Zelaya', '999887777', '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, '987654321', 4> into EMPLOYEE.
   - This insertion violates the key constraint because another tuple with the same SSN value already exists in the EMPLOYEE relation, and so it is rejected.
3. Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1960-04-05', '6357 Windswept, Katy, TX', F, 28000, '987654321', 7> into EMPLOYEE.
   - This insertion violates the referential integrity constraint specified on DNO because no DEPARTMENT tuple exists with DNUMBER = 7.

Sterling EX2011
IPR2025-00270
Page 34 of 69

4. Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, null, 4> into EMPLOYEE.

   • This insertion satisfies all constraints, so it is acceptable.

If an insertion violates one or more constraints, the default option is to *reject the insertion*. In this case, it would be useful if the DBMS could explain to the user why the insertion was rejected. Another option is to attempt to *correct the reason for rejecting the insertion*, but this is typically not used for violations caused by Insert; rather, it is used more often in correcting violations for Delete and Update. The following examples illustrate how this option may be used for Insert violations. In operation 1 above, the DBMS could ask the user to provide a value for SSN and could accept the insertion if a valid SSN value were provided. In operation 3, the DBMS could either ask the user to change the value of DNO to some valid value (or set it to null), or it could ask the user to insert a DEPARTMENT tuple with DNUMBER = 7 and could accept the insertion only after such an operation was accepted. Notice that in the latter case the insertion can **cascade** back to the EMPLOYEE relation if the user attempts to insert a tuple for department 7 with a value for MGRSSN that does not exist in the EMPLOYEE relation.

7.3.2 The Delete Operation

The **Delete** operation can violate only referential integrity, if the tuple being deleted is referenced by the foreign keys from other tuples in the database. To specify deletion, a condition on the attributes of the relation selects the tuple (or tuples) to be deleted. Here are some examples.

1. Delete the WORKS_ON tuple with ESSN = '999887777' and PNO = 10.

   • This deletion is acceptable.

2. Delete the EMPLOYEE tuple with SSN = '999887777'.

   • This deletion is not acceptable, because tuples in WORKS_ON refer to this tuple. Hence, if the tuple is deleted, referential integrity violations will result.

3. Delete the EMPLOYEE tuple with SSN = '333445555'.

   • This deletion will result in even worse referential integrity violations, because the tuple involved is referenced by tuples from the EMPLOYEE, DEPARTMENT, WORKS_ON, and DEPENDENT relations.

Three options are available if a deletion operation causes a violation. The first option is to *reject the deletion*. The second option is to *attempt to cascade (or propagate) the deletion* by deleting tuples that reference the tuple that is being deleted. For example, in operation 2, the DBMS could automatically delete the offending tuples from WORKS_ON with ESSN = '999887777'. A third option is to *modify the referencing attribute values* that cause the violation; each such value is either set to null or changed to reference another valid tuple. Notice that, if a referencing attribute that causes a violation is *part of the primary key*, it *cannot be* set to null; otherwise, it would violate entity integrity.

Sterling EX2011
IPR2025-00270
Page 35 of 69

Combinations of these three options are also possible. For example, to avoid having operation 3 cause a violation, the DBMS may automatically delete all tuples from WORKS_ON and DEPENDENT with ESSN = '333445555'. Tuples in EMPLOYEE with SUPERSSN = '333445555' and the tuple in DEPARTMENT with MGRSSN = '333445555' can have their SUPERSSN and MGRSSN values changed to other valid values or to null. Although it may make sense to delete automatically the WORKS_ON and DEPENDENT tuples that refer to an EMPLOYEE tuple, it may not make sense to delete other EMPLOYEE tuples or a DEPARTMENT tuple. In general, when a referential integrity constraint is specified, the DBMS should allow the user to *specify which of the three options* applies in case of a violation of the constraint. We discuss how to specify these options in SQL2 DDL in Chapter 8.

7.3.3 The Update Operation

The Update operation is used to change the values of one or more attributes in a tuple (or tuples) of some relation R. It is necessary to specify a condition on the attributes of the relation to select the tuple (or tuples) to be modified. Here are some examples.

1. Update the SALARY of the EMPLOYEE tuple with SSN = '999887777' to 28000.
   - Acceptable.
2. Update the DNO of the EMPLOYEE tuple with SSN = '999887777' to 1.
   - Acceptable.
3. Update the DNO of the EMPLOYEE tuple with SSN = '999887777' to 7.
   - Unacceptable, because it violates referential integrity.
4. Update the SSN of the EMPLOYEE tuple with SSN = '999887777' to '987654321'.
   - Unacceptable, because it violates primary key and referential integrity constraints.

Updating an attribute that is neither a primary key nor a foreign key usually causes no problems; the DBMS need only check to confirm that the new value is of the correct data type and domain. Modifying a primary key value is similar to deleting one tuple and inserting another in its place, because we use the primary key to identify tuples. Hence, the issues discussed earlier under both Insert and Delete come into play. If a foreign key attribute is modified, the DBMS must make sure that the new value refers to an existing tuple in the referenced relation (or is null).

7.4 Basic Relational Algebra Operations

In addition to defining the database structure and constraints, a data model must include a set of operations to manipulate the data. A basic set of relational model operations constitute the relational algebra. These operations enable the user to specify basic retrieval requests. The result of a retrieval is a new relation, which may have been formed from one or more relations. The algebra operations thus produce new relations, which can be

Sterling EX2011
IPR2025-00270
Page 36 of 69

4. Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, null, 4> into EMPLOYEE.

   • This insertion satisfies all constraints, so it is acceptable.

If an insertion violates one or more constraints, the default option is to *reject the insertion*. In this case, it would be useful if the DBMS could explain to the user why the insertion was rejected. Another option is to attempt to *correct the reason for rejecting the insertion*, but this is typically not used for violations caused by Insert; rather, it is used more often in correcting violations for Delete and Update. The following examples illustrate how this option may be used for Insert violations. In operation 1 above, the DBMS could ask the user to provide a value for SSN and could accept the insertion if a valid SSN value were provided. In operation 3, the DBMS could either ask the user to change the value of DNO to some valid value (or set it to null), or it could ask the user to insert a DEPARTMENT tuple with DNUMBER = 7 and could accept the insertion only after such an operation was accepted. Notice that in the latter case the insertion can **cascade** back to the EMPLOYEE relation if the user attempts to insert a tuple for department 7 with a value for MGRSSN that does not exist in the EMPLOYEE relation.

7.3.2 The Delete Operation

The **Delete** operation can violate only referential integrity, if the tuple being deleted is referenced by the foreign keys from other tuples in the database. To specify deletion, a condition on the attributes of the relation selects the tuple (or tuples) to be deleted. Here are some examples.

1. Delete the WORKS_ON tuple with ESSN = '999887777' and PNO = 10.

   • This deletion is acceptable.

2. Delete the EMPLOYEE tuple with SSN = '999887777'.

   • This deletion is not acceptable, because tuples in WORKS_ON refer to this tuple. Hence, if the tuple is deleted, referential integrity violations will result.

3. Delete the EMPLOYEE tuple with SSN = '333445555'.

   • This deletion will result in even worse referential integrity violations, because the tuple involved is referenced by tuples from the EMPLOYEE, DEPARTMENT, WORKS_ON, and DEPENDENT relations.

Three options are available if a deletion operation causes a violation. The first option is to *reject the deletion*. The second option is to *attempt to cascade (or propagate) the deletion* by deleting tuples that reference the tuple that is being deleted. For example, in operation 2, the DBMS could automatically delete the offending tuples from WORKS_ON with ESSN = '999887777'. A third option is to *modify the referencing attribute values* that cause the violation; each such value is either set to null or changed to reference another valid tuple. Notice that, if a referencing attribute that causes a violation is *part of the primary key*, it *cannot be* set to null; otherwise, it would violate entity integrity.

Sterling EX2011
IPR2025-00270
Page 37 of 69

Combinations of these three options are also possible. For example, to avoid having operation 3 cause a violation, the DBMS may automatically delete all tuples from WORKS_ON and DEPENDENT with ESSN = '333445555'. Tuples in EMPLOYEE with SUPERSSN = '333445555' and the tuple in DEPARTMENT with MGRSSN = '333445555' can have their SUPERSSN and MGRSSN values changed to other valid values or to null. Although it may make sense to delete automatically the WORKS_ON and DEPENDENT tuples that refer to an EMPLOYEE tuple, it may not make sense to delete other EMPLOYEE tuples or a DEPARTMENT tuple. In general, when a referential integrity constraint is specified, the DBMS should allow the user to *specify which of the three options* applies in case of a violation of the constraint. We discuss how to specify these options in SQL2 DDL in Chapter 8.

7.3.3 The Update Operation

The **Update** operation is used to change the values of one or more attributes in a tuple (or tuples) of some relation R. It is necessary to specify a condition on the attributes of the relation to select the tuple (or tuples) to be modified. Here are some examples.

1. Update the SALARY of the EMPLOYEE tuple with SSN = '999887777' to 28000.
   - Acceptable.
2. Update the DNO of the EMPLOYEE tuple with SSN = '999887777' to 1.
   - Acceptable.
3. Update the DNO of the EMPLOYEE tuple with SSN = '999887777' to 7.
   - Unacceptable, because it violates referential integrity.
4. Update the SSN of the EMPLOYEE tuple with SSN = '999887777' to '987654321'.
   - Unacceptable, because it violates primary key and referential integrity constraints.

Updating an attribute that is neither a primary key nor a foreign key usually causes no problems; the DBMS need only check to confirm that the new value is of the correct data type and domain. Modifying a primary key value is similar to deleting one tuple and inserting another in its place, because we use the primary key to identify tuples. Hence, the issues discussed earlier under both Insert and Delete come into play. If a foreign key attribute is modified, the DBMS must make sure that the new value refers to an existing tuple in the referenced relation (or is null).

7.4 Basic Relational Algebra Operations

In addition to defining the database structure and constraints, a data model must include a set of operations to manipulate the data. A basic set of relational model operations constitute the **relational algebra.** These operations enable the user to specify basic retrieval requests. The result of a retrieval is a new relation, which may have been formed from one or more relations. The algebra operations thus produce new relations, which can be

Sterling EX2011
IPR2025-00270
Page 38 of 69

further manipulated using operations of the same algebra. A sequence of relational algebra operations forms a **relational algebra expression,** whose result will also be a relation.

The relational algebra operations are usually divided into two groups. One group includes set operations from mathematical set theory; these are applicable because each relation is defined to be a set of tuples. Set operations include UNION, INTERSECTION, SET DIFFERENCE, and CARTESIAN PRODUCT. The other group consists of operations developed specifically for relational databases; these include SELECT, PROJECT, and JOIN, among others. The SELECT and PROJECT operations are discussed first, because they are the simplest. Then we discuss set operations. Finally, we discuss JOIN and other complex operations. The relational database shown in Figure 7.6 is used for our examples.

Some common database requests cannot be performed with the basic relational algebra operations, so additional operations are needed to express these requests. Some of these additional operations are described in Section 7.5.

7.4.1 The SELECT Operation

The SELECT operation is used to select a *subset* of the tuples from a relation that satisfy a **selection condition.** One can consider the SELECT operation to be a *filter* that keeps only those tuples that satisfy a qualifying condition. For example, to select the EMPLOYEE tuples whose department is 4, or those whose salary is greater than $30,000, we can individually specify each of these two conditions with a SELECT operation as follows:

$$\sigma_{DNO=4}(EMPLOYEE)$$

$$\sigma_{SALARY>30000}(EMPLOYEE)$$

In general, the SELECT operation is denoted by

$$\sigma_{<selection\ condition>}(R)$$

where the symbol σ (sigma) is used to denote the SELECT operator, and the selection condition is a Boolean expression specified on the attributes of relation R. Notice that R is generally a *relational algebra expression* whose result is a relation; the simplest expression is just the name of a database relation. The relation resulting from the SELECT operation has the *same attributes* as R. The Boolean expression specified in <selection condition> is made up of a number of clauses of the form

<attribute name> <comparison op> <constant value>, or

<attribute name> <comparison op> <attribute name>

where <attribute name> is the name of an attribute of R, <comparison op> is normally one of the operators {=, <, ≤, >, ≥, ≠}, and <constant value> is a constant value from the attribute domain. Clauses can be arbitrarily connected by the Boolean operators AND, OR, and NOT to form a general selection condition. For example, to select the tuples for all employees who either work in department 4 and make over $25,000 per year, or work in department 5 and make over $30,000, we can specify the following SELECT operation:

$$\sigma_{(DNO=4\ AND\ SALARY>25000)\ OR\ (DNO=5\ AND\ SALARY>30000)}(EMPLOYEE)$$

Sterling EX2011
IPR2025-00270
Page 39 of 69

(a)

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
Franklin	T	Wong	333445555	1955-12-08	638 Voss,Houston,TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry,Bellaire,TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 FireOak,Humble,TX	M	38000	333445555	5

(b)

LNAME	FNAME	SALARY
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

(c)

SEX	SALARY
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Figure 7.8 Results of SELECT and PROJECT operations. (a) $\sigma_{(DNO=4 \text{ AND } SALARY>25000) \text{ OR } (DNO=5 \text{ AND } SALARY>30000)}(EMPLOYEE)$. (b) $\pi_{LNAME, FNAME, SALARY}(EMPLOYEE)$. (c) $\pi_{SEX, SALARY}(EMPLOYEE)$

The result is shown in Figure 7.8(a). Notice that the comparison operators in the set $\{=, <, \leq, >, \geq, \neq\}$ apply to attributes whose domains are *ordered values*, such as numeric or date domains. Domains of strings of characters are considered ordered based on the collating sequence of the characters. If the domain of an attribute is a set of *unordered values*, then only the comparison operators in the set $\{=, \neq\}$ can be used. An example of an unordered domain is the domain Color = {red, blue, green, white, yellow, . . .} where no order is specified among the various colors. Some domains allow additional types of comparison operators; for example, a domain of character strings may allow the comparison operator SUBSTRING_OF.

In general, the result of a SELECT operation can be determined as follows. The <selection condition> is applied independently to each tuple t in R. This is done by substituting each occurrence of an attribute A_i in the selection condition with its value in the tuple $t[A_i]$. If the condition evaluates to true, then tuple t is **selected.** All the selected tuples appear in the result of the SELECT operation. The Boolean conditions AND, OR, and NOT have their normal interpretation as follows:

- (cond1 AND cond2) is true if both (cond1) and (cond2) are true; otherwise, it is false.
- (cond1 OR cond2) is true if either (cond1) or (cond2) or both are true; otherwise, it is false.
- (NOT cond) is true if cond is false; otherwise, it is false.

The SELECT operator is unary; that is, it is applied to a single relation. Moreover, the selection operation is applied to *each tuple individually*; hence, selection conditions cannot involve more than one tuple. The **degree** of the relation resulting from a SELECT operation is the same as that of R. The number of tuples in the resulting relation is always *less than or equal to* the number of tuples in R. That is, $|\sigma_c(R)| \leq |R|$ for any condition C.

Sterling EX2011
IPR2025-00270
Page 40 of 69

The fraction of tuples selected by a selection condition is referred to as the **selectivity** of the condition.

Notice that the SELECT operation is **commutative**; that is,

$$\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$$

Hence, a sequence of SELECTs can be applied in any order. In addition, we can always combine a **cascade** of SELECT operations into a single SELECT operation with a conjunctive (AND) condition; that is:

$$\sigma_{<cond1>}(\sigma_{<cond2>}(\ldots(\sigma_{<condn>}(R))\ldots)) = \sigma_{<cond1> \text{ AND } <cond2> \text{ AND} \ldots \text{ AND } <condn>}(R)$$

7.4.2 The PROJECT Operation

If we think of a relation as a table, the SELECT operation selects some of the *rows* from the table while discarding other rows. The PROJECT operation, on the other hand, selects certain *columns* from the table and discards the other columns. If we are interested in only certain attributes of a relation, we use the PROJECT operation to *project* the relation over these attributes only. For example, to list each employee's first and last name and salary, we can use the PROJECT operation as follows:

$$\pi_{LNAME, FNAME, SALARY}(EMPLOYEE)$$

The resulting relation is shown in Figure 7.8(b). The general form of the PROJECT operation is

$$\pi_{<attribute\ list>}(R)$$

where π (pi) is the symbol used to represent the PROJECT operation and <attribute list> is a list of attributes from the attributes of relation R. Again, notice that R is, in general, a *relational algebra expression* whose result is a relation, which in the simplest case is just the name of a database relation. The result of the PROJECT operation has only the attributes specified in <attribute list> and *in the same order as they appear in the list*. Hence, its **degree** is equal to the number of attributes in <attribute list>.

If the attribute list includes only nonkey attributes of R, duplicate tuples are likely to occur; the PROJECT operation *removes any duplicate tuples*, so the result of the PROJECT operation is a set of tuples and hence a valid relation.[8] This is known as **duplicate elimination.** For example, consider the following PROJECT operation:

$$\pi_{SEX, SALARY}(EMPLOYEE)$$

The result is shown in Figure 7.8(c). Notice that the tuple <F, 25000> appears only once in Figure 7.8(c), even though this combination of values appears twice in the EMPLOYEE relation.

8. If duplicates are not eliminated, the result would be a **multiset** or **bag** of tuples rather than a set. As we shall see in Chapter 8, the SQL language allows the user to specify whether duplicates should be eliminated or not.

Sterling EX2011
IPR2025-00270
Page 41 of 69

The number of tuples in a relation resulting from a PROJECT operation is always less than or equal to the number of tuples in R. If the projection list is a superkey of R—that is, it includes some key of R—the resulting relation has the *same number* of tuples as R. Moreover,

$$\pi_{<list1>}(\pi_{<list2>}(R)) = \pi_{<list1>}(R)$$

as long as <list2> contains the attributes in <list1>; otherwise, the left-hand side is an incorrect expression. It is also noteworthy that commutativity *does not* hold on PROJECT.

7.4.3 Sequences of Operations and the RENAME Operation

The relations shown in Figure 7.8 do not have any names. In general, we may want to apply several relational algebra operations one after the other. Either we can write the operations as a single **relational algebra expression** by nesting the operations, or we can apply one operation at a time and create intermediate result relations. In the latter case, we must name the relations that hold the intermediate results. For example, to retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a SELECT and a PROJECT operation. We can write a single relational algebra expression as follows:

$$\pi_{FNAME, LNAME, SALARY}(\sigma_{DNO=5}(EMPLOYEE))$$

Figure 7.9(a) shows the result of this relational algebra expression. Alternatively, we can explicitly show the sequence of operations, giving a name to each intermediate relation:

$$DEP5\_EMPS \leftarrow \sigma_{DNO=5}(EMPLOYEE)$$

$$RESULT \leftarrow \pi_{FNAME, LNAME, SALARY}(DEP5\_EMPS)$$

It is often simpler to break down a complex sequence of operations by specifying intermediate result relations than to write a single relational algebra expression. We can also use this technique to rename the attributes in the intermediate and result relations. This can be useful in connection with more complex operations such as UNION and JOIN, as we shall see. To rename the attributes in a relation, we simply list the new attribute names in parentheses, as in the following example:

$$TEMP \leftarrow \sigma_{DNO=5}(EMPLOYEE)$$

$$R(FIRSTNAME, LASTNAME, SALARY) \leftarrow \pi_{FNAME, LNAME, SALARY}(TEMP)$$

The above two operations are illustrated in Figure 7.9(b). If no renaming is applied, the names of the attributes in the resulting relation of a SELECT operation are the same as those in the original relation and in the same order. For a PROJECT operation with no renaming, the resulting relation has the same attribute names as those in the projection list and in the same order in which they appear in the list.

We can also define a **RENAME** operation—which can rename either the relation name, or the attribute names, or both—in a manner similar to the way we defined SELECT

Sterling EX2011
IPR2025-00270
Page 42 of 69

(a)

FNAME	LNAME	SALARY
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

(b)

TEMP	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren,Houston,TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss,Houston,TX	M	40000	888665555	5
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak,Humble,TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice,Houston,TX	F	25000	333445555	5

R	FIRSTNAME	LASTNAME	SALARY
	John	Smith	30000
	Franklin	Wong	40000
	Ramesh	Narayan	38000
	Joyce	English	25000

Figure 7.9 Results of relational algebra expressions.
(a) $\pi_{LNAME,\ FNAME,\ SALARY}\ (\sigma_{DNO=5}(EMPLOYEE))$.
(b) The same expression using intermediate relations and renaming of attributes.

and PROJECT. The general RENAME operation when applied to a relation R of degree n is denoted by

$$\rho_{S(B1,\ B2,\ \ldots,\ Bn)}(R) \text{ or } \rho_S(R) \text{ or } \rho_{(B1,\ B2,\ \ldots,\ Bn)}(R)$$

where the symbol ρ (rho) is used to denote the RENAME operator, S is the new relation name, and $B_1, B_2, \ldots, B_n$ are the new attribute names. The first expression renames both the relation and its attributes; the second renames the relation only; and the third renames the attributes only. If the attributes of R are $(A_1, A_2, \ldots, A_n)$ in that order, then each A_i is renamed as B_i.

7.4.4 Set Theoretic Operations

The next group of relational algebra operations are the standard mathematical operations on sets. For example, to retrieve the social security numbers of all employees who either work in department 5 or directly supervise an employee who works in department 5, we can use the UNION operation as follows:

$$DEP5\_EMPS \leftarrow \sigma_{DNO=5}(EMPLOYEE)$$

$$RESULT1 \leftarrow \pi_{SSN}(DEP5\_EMPS)$$

$$RESULT2(SSN) \leftarrow \pi_{SUPERSSN}(DEP5\_EMPS)$$

$$RESULT \leftarrow RESULT1 \cup RESULT2$$

The relation RESULT1 has the social security numbers of all employees who work in department 5, whereas RESULT2 has the social security numbers of all employees who

Sterling EX2011
IPR2025-00270
Page 43 of 69

RESULT1	SSN
	123456789
	333445555
	666884444
	453453453

RESULT2	SSN
	333445555
	888665555

RESULT	SSN
	123456789
	333445555
	666884444
	453453453
	888665555

Figure 7.10 Query result after the UNION operation: RESULT ← RESULT1 ∪ RESULT2.

directly supervise an employee who works in department 5. The UNION operation produces the tuples that are in either RESULT1 or RESULT2 or both (see Figure 7.10).

Several set theoretic operations are used to merge the elements of two sets in various ways, including **UNION, INTERSECTION,** and **SET DIFFERENCE.** These are binary operations; that is, each is applied to two sets. When these operations are adapted to relational databases, the two relations on which any of the above three operations are applied must have the same **type of tuples;** this condition is called *union compatibility*. Two relations $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots, B_n)$ are said to be **union compatible** if they have the same degree n, and if $dom(A_i) = dom(B_i)$ for $1 \leq i \leq n$. This means that the two relations have the same number of attributes and that each pair of corresponding attributes have the same domain.

We can define the three operations UNION, INTERSECTION, and SET DIFFERENCE on two union-compatible relations R and S as follows:

- UNION: The result of this operation, denoted by $R \cup S$, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.
- INTERSECTION: The result of this operation, denoted by $R \cap S$, is a relation that includes all tuples that are in both R and S.
- SET DIFFERENCE: The result of this operation, denoted by $R - S$, is a relation that includes all tuples that are in R but not in S.

We will adopt the convention that the resulting relation has the same attribute names as the *first* relation R. Figure 7.11 illustrates the three operations. The relations STUDENT and INSTRUCTOR in Figure 7.11(a) are union compatible, and their tuples represent the names of students and instructors, respectively. The result of the UNION operation in Figure 7.11(b) shows the names of all students and instructors. Note that duplicate tuples appear only once in the result. The result of the INTERSECTION operation (Figure 7.11c) includes only those who are both students and instructors. Notice that both UNION and INTERSECTION are *commutative operations*; that is

$$R \cup S = S \cup R, \text{ and } R \cap S = S \cap R$$

Both union and intersection can be treated as n-ary operations applicable to any number of relations as both are *associative operations*; that is

$$R \cup (S \cup T) = (R \cup S) \cup T, \text{ and } (R \cap S) \cap T = R \cap (S \cap T)$$

Sterling EX2011
IPR2025-00270
Page 44 of 69

(a)

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

(b)

FN	LN
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

FN	LN
Susan	Yao
Ramesh	Shah

(d)

FN	LN
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

FNAME	LNAME
John	Smith
Ricardo	Browne
Francis	Johnson

Figure 7.11 Illustrating the set operations UNION, INTERSECTION, and DIFFERENCE. (a) Two union compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT − INSTRUCTOR. (e) INSTRUCTOR − STUDENT.

The DIFFERENCE operation is *not commutative*; that is, in general

$$R - S \neq S - R$$

Figure 7.11(d) shows the names of students who are not instructors, and Figure 7.11(e) shows the names of instructors who are not students.

Next we discuss the **CARTESIAN PRODUCT** operation—also known as **CROSS PRODUCT** or **CROSS JOIN**—denoted by **x**, which is also a binary set operation, but the relations on which it is applied do not have to be union compatible. This operation is used to combine tuples from two relations in a combinatorial fashion. In general, the result of R(A1,

Sterling EX2011
IPR2025-00270
Page 45 of 69

$A_2, \ldots, A_n$) x $S(B_1, B_2, \ldots, B_m)$ is a relation Q with n + m attributes $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$, in that order. The resulting relation Q has one tuple for each combination of tuples—one from R and one from S. Hence, if R has n_R tuples and S has n_S tuples, then R x S will have $n_R * n_S$ tuples. The operation applied by itself is generally meaningless. It is useful when followed by a selection that matches values of attributes coming from the component relations. For example, suppose that we want to retrieve for each female employee a list of the names of her dependents; we can do this as follows:

FEMALE_EMPS $\leftarrow \sigma_{SEX='F'}$(EMPLOYEE)

EMPNAMES $\leftarrow \pi_{FNAME, LNAME, SSN}$(FEMALE_EMPS)

EMP_DEPENDENTS $\leftarrow$ EMPNAMES x DEPENDENT

ACTUAL_DEPENDENTS $\leftarrow \sigma_{SSN=ESSN}$(EMP_DEPENDENTS)

RESULT $\leftarrow \pi_{FNAME, LNAME, DEPENDENT\_NAME}$(ACTUAL_DEPENDENTS)

The resulting relations from the above sequence of operations are shown in Figure 7.12. The EMP_DEPENDENTS relation is the result of applying the CARTESIAN PRODUCT operation to EMPNAMES from Figure 7.12 with DEPENDENT from Figure 7.6. In EMP_DEPENDENTS, every tuple from EMPNAMES is combined with every tuple from DEPENDENT, giving a result that is not very meaningful. We only want to combine a female employee tuple with her dependents—namely, the DEPENDENT tuples whose ESSN values match the SSN value of the EMPLOYEE tuple. The ACTUAL_DEPENDENTS relation accomplishes this.

The CARTESIAN PRODUCT creates tuples with the combined attributes of two relations. We can then SELECT only related tuples from the two relations by specifying an appropriate selection condition, as we did in the preceding example. Because this sequence of CARTESIAN PRODUCT followed by SELECT is used quite commonly to identify and select related tuples from two relations, a special operation, called JOIN, was created to specify this sequence as a single operation. We discuss the JOIN operation next.

7.4.5 The JOIN Operation

The JOIN operation, denoted by $\bowtie$, is used to combine *related tuples* from two relations into single tuples. This operation is very important for any relational database with more than a single relation, because it allows us to process relationships among relations. To illustrate join, suppose that we want to retrieve the name of the manager of each department. To get the manager's name, we need to combine each department tuple with the employee tuple whose SSN value matches the MGRSSN value in the department tuple. We do this by using the JOIN operation, and then projecting the result over the necessary attributes, as follows:

DEPT_MGR $\leftarrow$ DEPARTMENT $\bowtie_{MGRSSN=SSN}$ EMPLOYEE

RESULT $\leftarrow \pi_{DNAME, LNAME, FNAME}$(DEPT_MGR)

The first operation is illustrated in Figure 7.13. Note that MGRSSN is a foreign key and that the referential integrity constraint plays a role in having matching tuples in the

Sterling EX2011
IPR2025-00270
Page 46 of 69

FEMALE_EMPS	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle,Spring,TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry,Bellaire,TX	F	43000	888665555	4
	Joyce	A	English	453453453	1972-07-31	5631 Rice,Houston,TX	F	25000	333445555	5

EMPNAMES	FNAME	LNAME	SSN
	Alicia	Zelaya	999887777
	Jennifer	Wallace	987654321
	Joyce	English	453453453

EMP_DEPENDENTS	FNAME	LNAME	SSN	ESSN	DEPENDENT_NAME	SEX	BDATE	• • •
	Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	• • •
	Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	• • •
	Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	• • •
	Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	• • •
	Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	• • •
	Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	• • •
	Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	• • •
	Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	• • •
	Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	• • •
	Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	• • •
	Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	• • •
	Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	• • •
	Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	• • •
	Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	• • •
	Joyce	English	453453453	333445555	Alice	F	1986-04-05	• • •
	Joyce	English	453453453	333445555	Theodore	M	1983-10-25	• • •
	Joyce	English	453453453	333445555	Joy	F	1958-05-03	• • •
	Joyce	English	453453453	987654321	Abner	M	1942-02-28	• • •
	Joyce	English	453453453	123456789	Michael	M	1988-01-04	• • •
	Joyce	English	453453453	123456789	Alice	F	1988-12-30	• • •
	Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	• • •

ACTUAL_DEPENDENTS	FNAME	LNAME	SSN	ESSN	DEPENDENT_NAME	SEX	BDATE	• • •
	Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	• • •

RESULT	FNAME	LNAME	DEPENDENT_NAME
	Jennifer	Wallace	Abner

Figure 7.12 An illustration of the CARTESIAN PRODUCT operation.

DEPT_MGR	DNAME	DNUMBER	MGRSSN	• • •	FNAME	MINIT	LNAME	SSN	• • •
	Research	5	333445555	• • •	Franklin	T	Wong	333445555	• • •
	Administration	4	987654321	• • •	Jennifer	S	Wallace	987654321	• • •
	Headquarters	1	888665555	• • •	James	E	Borg	888665555	• • •

Figure 7.13 Illustrating the JOIN operation.

Sterling EX2011
IPR2025-00270
Page 47 of 69

referenced relation EMPLOYEE. The example we gave earlier to illustrate the CARTESIAN PRODUCT operation can be specified, using the JOIN operation, by replacing the two operations:

$$\text{EMP\_DEPENDENTS} \leftarrow \text{EMPNAMES} \times \text{DEPENDENT}$$

$$\text{ACTUAL\_DEPENDENTS} \leftarrow \sigma_{\text{SSN=ESSN}}(\text{EMP\_DEPENDENTS})$$

with a single JOIN operation:

$$\text{ACTUAL\_DEPENDENTS} \leftarrow \text{EMPNAMES} \bowtie_{\text{SSN=ESSN}} \text{DEPENDENT}$$

The general form of a JOIN operation on two relations[9] $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots, B_m)$ is:

$$R \bowtie_{\text{<join condition>}} S$$

The result of the JOIN is a relation Q with n + m attributes $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$ in that order; Q has one tuple for each combination of tuples—one from R and one from S—*whenever the combination satisfies the join condition*. This is the main difference between CARTESIAN PRODUCT and JOIN: in JOIN, only combinations of tuples *satisfying the join condition* appear in the result, whereas in the CARTESIAN PRODUCT *all* combinations of tuples are included in the result. The join condition is specified on attributes from the two relations R and S and is evaluated for each combination of tuples. Each tuple combination for which the join condition evaluates to true is included in the resulting relation Q *as a single combined tuple*.

A general join condition is of the form:

<condition> AND <condition> AND . . . AND <condition>

where each condition is of the form $A_i \theta B_j$, A_i is an attribute of R, B_j is an attribute of S, A_i and B_j have the same domain, and θ (theta) is one of the comparison operators $\{=, <, \leq, >, \geq, \neq\}$. A JOIN operation with such a general join condition is called a **THETA JOIN.** Tuples whose join attributes are null *do not* appear in the result. In that sense, the join operation does *not* necessarily preserve all of the information in the participating relations.

The most common JOIN involves join conditions with equality comparisons only. Such a JOIN, where the only comparison operator used is =, is called an **EQUIJOIN.** Both examples we have considered were EQUIJOINs. Notice that in the result of an EQUIJOIN we always have one or more pairs of attributes that have *identical values* in every tuple. For example, in Figure 7.13, the values of the attributes MGRSSN and SSN are identical in every tuple of DEPT_MGR because of the equality join condition specified on these two attributes. Because one of each pair of attributes with identical values is superfluous, a new operation called **NATURAL JOIN**—denoted by * —was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.[10] The standard definition of NATURAL JOIN requires that the two join attributes (or each pair of join attributes) have

9. Again, notice that R and S can be the relations that result from general *relational algebra expressions*.

10. NATURAL JOIN is basically an EQUIJOIN followed by removal of the superfluous attributes.

Sterling EX2011
IPR2025-00270
Page 48 of 69

(a)

PROJ_DEPT	PNAME	PNUMBER	PLOCATION	DNUM	DNAME	MGRSSN	MGRSTARTDATE
	ProductX	1	Bellaire	5	Research	333445555	1988-05-22
	ProductY	2	Sugarland	5	Research	333445555	1988-05-22
	ProductZ	3	Houston	5	Research	333445555	1988-05-22
	Computerization	10	Stafford	4	Administration	987654321	1995-01-01
	Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
	Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

(b)

DEPT_LOCS	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE	LOCATION
	Headquarters	1	888665555	1981-06-19	Houston
	Administration	4	987654321	1995-01-01	Stafford
	Research	5	333445555	1988-05-22	Bellaire
	Research	5	333445555	1988-05-22	Sugarland
	Research	5	333445555	1988-05-22	Houston

Figure 7.14 An illustration of the NATURAL JOIN operation. (a) PROJ_DEPT ← PROJECT * DEPT. (b) DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS.

the same name in both relations. If this is not the case, a renaming operation is applied first. In the following example, we first rename the DNUMBER attribute of DEPARTMENT to DNUM—so that it has the same name as the DNUM attribute in PROJECT—then apply NATURAL JOIN:

$$\text{PROJ\_DEPT} \leftarrow \text{PROJECT} * \rho_{(\text{DNAME,DNUM,MGRSSN,MGRSTARTDATE})}(\text{DEPARTMENT})$$

The attribute DNUM is called the **join attribute**. The resulting relation is illustrated in Figure 7.14(a). In the PROJ_DEPT relation, each tuple combines a PROJECT tuple with the DEPARTMENT tuple for the department that controls the project, but *only one join attribute* is kept.

If the attributes on which the natural join is specified *have the same names in both relations*, renaming is unnecessary. For example, to apply a natural join on the DNUMBER attributes of DEPARTMENT and DEPT_LOCATIONS, it is sufficient to write:

$$\text{DEPT\_LOCS} \leftarrow \text{DEPARTMENT} * \text{DEPT\_LOCATIONS}$$

The resulting relation is shown in Figure 7.14(b), which combines each department with its locations and has one tuple for each location. In general, NATURAL JOIN is performed by equating *all* attribute pairs that have the same name in the two relations. There can be a list of join attributes from each relation, and each corresponding pair must have the same name.

A more general but non-standard definition for NATURAL JOIN is

$$Q \leftarrow R *_{(\langle list1 \rangle),(\langle list2 \rangle)} S$$

Sterling EX2011
IPR2025-00270
Page 49 of 69

In this case, <list1> specifies a list of i attributes from R, and <list2> specifies a list of i attributes from S. The lists are used to form equality comparison conditions between pairs of corresponding attributes; the conditions are then ANDed together. Only the list corresponding to attributes of the first relation R—<list 1>—is kept in the result Q.

Notice that if no combination of tuples satisfies the join condition, the result of a JOIN is an empty relation with zero tuples. In general, if R has n_R tuples and S has n_S tuples, the result of a JOIN operation $R \bowtie_{<\text{join condition}>} S$ will have between zero and $n_R * n_S$ tuples. The expected size of the join result divided by the maximum size $n_R * n_S$ leads to a ratio called **join selectivity**, which is a property of each join condition. If there is no join condition, all combinations of tuples qualify and the JOIN becomes a CARTESIAN PRODUCT, also called CROSS PRODUCT or CROSS JOIN.

The join operation is used to combine data from multiple relations so that related information can be presented in a single table. Note that sometimes a join may be specified between a relation and itself, as we shall illustrate in Section 7.5.2. The natural join or equijoin operation can also be specified among multiple tables, leading to an *n-way join*. For example, consider the following three-way join:

$$((\text{PROJECT} \bowtie_{\text{DNUM=DNUMBER}} \text{DEPARTMENT}) \bowtie_{\text{MGRSSN=SSN}} \text{EMPLOYEE})$$

This links each project to its controlling department, and then relates the department to its manager employee. The net result is a consolidated relation where each tuple contains this project-department-manager information.

7.4.6 A Complete Set of Relational Algebra Operations

It has been shown that the set of relational algebra operations $\{\sigma, \pi, \cup, -, \times\}$ is a **complete** set; that is, any of the other relational algebra operations can be expressed as a *sequence of operations from this set*. For example, the INTERSECTION operation can be expressed by using UNION and DIFFERENCE as follows:

$$R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R))$$

Although, strictly speaking, INTERSECTION is not required, it is inconvenient to specify this complex expression every time we wish to specify an intersection. As another example, a JOIN operation can be specified as a CARTESIAN PRODUCT followed by a SELECT operation, as we discussed:

$$R \bowtie_{<\text{condition}>} S \equiv \sigma_{<\text{condition}>} (R \times S)$$

Similarly, a NATURAL JOIN can be specified as a CARTESIAN PRODUCT preceded by RENAME and followed by SELECT and PROJECT operations. Hence, the various JOIN operations are also *not strictly necessary* for the expressive power of the relational algebra; however, they are very important because they are convenient to use and are very commonly applied in database applications. Other operations have been included in the relational

Sterling EX2011
IPR2025-00270
Page 50 of 69

algebra for convenience rather than necessity. We discuss one of these—the DIVISION operation—in the next section.

7.4.7 The DIVISION Operation

The DIVISION operation is useful for a special kind of query that sometimes occurs in database applications. An example is "Retrieve the names of employees who work on *all* the projects that 'John Smith' works on." To express this query using the DIVISION operation, proceed as follows. First, retrieve the list of project numbers that 'John Smith' works on in the intermediate relation SMITH_PNOS:

$$\text{SMITH} \leftarrow \sigma_{\text{FNAME='John' AND LNAME='Smith'}}(\text{EMPLOYEE})$$

$$\text{SMITH\_PNOS} \leftarrow \pi_{\text{PNO}}(\text{WORKS\_ON} \bowtie_{\text{ESSN=SSN}} \text{SMITH})$$

Next, create a relation that includes a tuple <PNO, ESSN> whenever the employee whose social security number is ESSN works on the project whose number is PNO in the intermediate relation SSN_PNOS:

$$\text{SSN\_PNOS} \leftarrow \pi_{\text{ESSN,PNO}}(\text{WORKS\_ON})$$

Finally, apply the DIVISION operation to the two relations, which gives the desired employees' social security numbers:

$$\text{SSNS(SSN)} \leftarrow \text{SSN\_PNOS} \div \text{SMITH\_PNOS}$$

$$\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME}}(\text{SSNS} * \text{EMPLOYEE})$$

The previous operations are shown in Figure 7.15(a). In general, the DIVISION operation is applied to two relations $R(Z) \div S(X)$, where $X \subseteq Z$. Let $Y = Z - X$ (and hence $Z = X \cup Y$); that is, let Y be the set of attributes of R that are not attributes of S. The result of DIVISION is a relation T(Y) that includes a tuple t if tuples t_R appear in R with $t_R[Y] = t$, and with $t_R[X] = t_S$ *for every tuple* t_S in S. This means that, for a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with *every* tuple in S.

Figure 7.15(b) illustrates a DIVISION operator where X = {A}, Y = {B}, and Z = {A, B}. Notice that the tuples (values) b_1 and b_4 appear in R in combination with all three tuples in S; that is why they appear in the resulting relation T. All other values of B in R do not appear with all the tuples in S and are not selected: b_2 does not appear with a_2 and b_3 does not appear with a_1.

The DIVISION operator can be expressed as a sequence of π, $\times$, and $-$ operations as follows:

$$T_1 \leftarrow \pi_Y(R)$$

$$T_2 \leftarrow \pi_Y((S \times T_1) - R)$$

$$T \leftarrow T_1 - T_2$$

Sterling EX2011
IPR2025-00270
Page 51 of 69

(a)

SSN_PNOS	
ESSN	PNO
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS	
PNO	
1	
2	

SSNS	
SSN	
123456789	
453453453	

(b)

R		
A	B	
a1	b1	
a2	b1	
a3	b1	
a4	b1	
a1	b2	
a3	b2	
a2	b3	
a3	b3	
a4	b3	
a1	b4	
a2	b4	
a3	b4	

S	
A	
a1	
a2	
a3	

T	
B	
b1	
b4	

Figure 7.15 Illustrating the DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) $T \leftarrow R \div S$.

Sterling EX2011
IPR2025-00270
Page 52 of 69

7.5 Additional Relational Operations

Some common database requests—which are needed in commercial query languages for relational DBMSs—cannot be performed with the basic relational algebra operations described in Section 7.4. In this section we define additional operations to express these requests. These operations enhance the expressive power of the relational algebra.

7.5.1 Aggregate Functions and Grouping

The first type of request that cannot be expressed in the basic relational algebra is to specify mathematical **aggregate functions** on collections of values from the database. Examples of such functions include retrieving the average or total salary of all employees or the number of employee tuples. Common functions applied to collections of numeric values include SUM, AVERAGE, MAXIMUM, and MINIMUM. The COUNT function is used for counting tuples or values.

Another common type of request involves grouping the tuples in a relation by the value of some of their attributes and then applying an aggregate function independently to each group. An example would be to group employee tuples by DNO, so that each group includes the tuples for employees working in the same department. We can then list each DNO value along with, say, the average salary of employees within the department.

We can define an AGGREGATE FUNCTION operation, using the symbol $\mathfrak{F}$ (pronounced "script F"),[11] to specify these types of requests as follows:

<grouping attributes> $\mathfrak{F}$ <function list> (R)

where <grouping attributes> is a list of attributes of the relation specified in R, and <function list> is a list of (<function> <attribute>) pairs. In each such pair, <function> is one of the allowed functions—such as SUM, AVERAGE, MAXIMUM, MINIMUM, COUNT—and <attribute> is an attribute of the relation specified by R. The resulting relation has the grouping attributes plus one attribute for each element in the function list. For example, to retrieve each department number, the number of employees in the department, and their average salary, while renaming the resulting attributes as indicated below, we write:

$\rho_{R(DNO,\ NO\_OF\_EMPLOYEES,\ AVERAGE\_SAL)}$ (DNO $\mathfrak{F}$ COUNT SSN, AVERAGE SALARY (EMPLOYEE))

The result of this operation is shown in Figure 7.16(a).

In the above example, we specified a list of attribute names—between parentheses in the rename operation—for the resulting relation R. If no renaming is applied, then the attributes of the resulting relation that correspond to the function list will each be the concatenation of the function name with the attribute name in the form <function>_<attribute>. For example, Figure 7.16(b) shows the result of the following operation:

DNO $\mathfrak{F}$ COUNT SSN, AVERAGE SALARY(EMPLOYEE)

11. There is no single agreed-upon notation for specifying aggregate functions. In some books "script A" is used.

Sterling EX2011
IPR2025-00270
Page 53 of 69

(a)

R	DNO	NO_OF_EMPLOYEES	AVERAGE_SAL
	5	4	33250
	4	3	31000
	1	1	55000

(b)

DNO	COUNT_SSN	AVERAGE_SALARY
5	4	33250
4	3	31000
1	1	55000

(c)

COUNT_SSN	AVERAGE_SALARY
8	35125

Figure 7.16 An illustration of the AGGREGATE FUNCTION operation. (a) $R(\text{DNO, NO\_OF\_EMPLOYEES, AVERAGE\_SAL}) \leftarrow {}_{\text{DNO}}\Im_{\text{COUNT SSN, AVERAGE SALARY}}(\text{EMPLOYEE})$. (b) ${}_{\text{DNO}}\Im_{\text{COUNT SSN, AVERAGE SALARY}}(\text{EMPLOYEE})$. (c) $\Im_{\text{COUNT SSN, AVERAGE SALARY}}(\text{EMPLOYEE})$.

If no grouping attributes are specified, the functions are applied to the attribute values of *all the tuples* in the relation, so the resulting relation has a *single tuple only*. For example, Figure 7.16(c) shows the result of the following operation:

$$\Im_{\text{COUNT SSN, AVERAGE SALARY}}(\text{EMPLOYEE})$$

It is important to note that, in general, duplicates are *not eliminated* when an aggregate function is applied; this way, the normal interpretation of functions such as SUM and AVERAGE is computed.[12] It is worth emphasizing that the result of applying an aggregate function is a relation, not a scalar number—even if it has a single value.

7.5.2 Recursive Closure Operations

Another type of operation that, in general, cannot be specified in the basic relational algebra is **recursive closure.** This operation is applied to a **recursive relationship** between tuples of the same type, such as the relationship between an employee and a supervisor. This relationship is described by the foreign key SUPERSSN of the EMPLOYEE relation in Figures 7.6 and 7.7, which relates each employee tuple (in the role of supervisee) to another employee tuple (in the role of supervisor). An example of a recursive operation is to retrieve all supervisees of an employee e at all levels—that is, all employees e′ directly supervised by e; all employees e″ directly supervised by each employee e′; all employees e‴ directly supervised by each employee e″; and so on. Although it is

12. In SQL, the option of eliminating duplicates before applying the aggregate function is available by including the keyword DISTINCT (see Chapter 8).

Sterling EX2011
IPR2025-00270
Page 54 of 69

straightforward in the relational algebra to specify all employees supervised by e *at a specific level*, it is difficult to specify all supervisees at *all* levels. For example, to specify the SSNs of all employees e′ directly supervised—*at level one*—by the employee e whose name is 'James Borg' (see Figure 7.6), we can apply the following operation:

$$\text{BORG\_SSN} \leftarrow \pi_{\text{SSN}}(\sigma_{\text{FNAME='James' AND LNAME='Borg'}}(\text{EMPLOYEE}))$$

$$\text{SUPERVISION}(\text{SSN1, SSN2}) \leftarrow \pi_{\text{SSN, SUPERSSN}}(\text{EMPLOYEE})$$

$$\text{RESULT1}(\text{SSN}) \leftarrow \pi_{\text{SSN1}}(\text{SUPERVISION} \bowtie_{\text{SSN2=SSN}} \text{BORG\_SSN})$$

To retrieve all employees supervised by Borg at level 2—that is, all employees e″ supervised by some employee e′ who is directly supervised by Borg—we can apply another JOIN to the result of the first query, as follows:

$$\text{RESULT2}(\text{SSN}) \leftarrow \pi_{\text{SSN1}}(\text{SUPERVISION} \bowtie_{\text{SSN2=SSN}} \text{RESULT1})$$

To get both sets of employees supervised at levels 1 and 2 by 'James Borg,' we can apply the UNION operation to the two results, as follows:

$$\text{RESULT} \leftarrow \text{RESULT2} \cup \text{RESULT1}$$

The results of these queries are illustrated in Figure 7.17. Although it is possible to retrieve employees at each level and then take their UNION, we cannot, in general, specify a query such as "retrieve the supervisees of 'James Borg' at all levels" without utilizing a looping mechanism.[13] An operation called the *transitive closure* of relations has been proposed to compute the recursive relationship as far as the recursion proceeds.

7.5.3 OUTER JOIN and OUTER UNION Operations

Finally, we discuss some extensions of the JOIN and UNION operations. The JOIN operations described earlier match tuples that satisfy the join condition. For example, for a NATURAL JOIN operation R * S, only tuples from R that have matching tuples in S—and vice versa—appear in the result. Hence, tuples without a *matching* (or *related*) tuple are eliminated from the JOIN result. Tuples with null in the join attributes are also eliminated. A set of operations, called OUTER JOINs, can be used when we want to keep all the tuples in R, or those in S, or those in both relations in the result of the JOIN, whether or not they have matching tuples in the other relation. This satisfies the need of queries where tuples from two tables are to be combined by matching corresponding rows, but some tuples are liable to be lost for lack of matching values. In such cases an operation is desirable that would preserve all the tuples whether or not they produce a match.

For example, suppose that we want a list of all employee names and also the name of the departments they manage *if they happen to manage a department;* we can apply an operation LEFT OUTER JOIN, denoted by ⟕, to retrieve the result as follows:

$$\text{TEMP} \leftarrow (\text{EMPLOYEE} \text{ ⟕}_{\text{SSN=MGRSSN}} \text{DEPARTMENT})$$

$$\text{RESULT} \leftarrow \pi_{\text{FNAME, MINIT, LNAME, DNAME}}(\text{TEMP})$$

13. We will discuss recursive queries further in Chapter 25 when we give an overview of deductive databases. Also, the SQL3 standard includes syntax for recursive closure.

Sterling EX2011
IPR2025-00270
Page 55 of 69

(Borg's SSN is 888665555)

	(SSN)	(SUPERSSN)
SUPERVISION	SSN1	SSN2
	123456789	333445555
	333445555	888665555
	999887777	987654321
	987654321	888665555
	666884444	333445555
	453453453	333445555
	987987987	987654321
	888665555	null

RESULT 1	SSN
	333445555
	987654321

(Supervised by Borg)

RESULT 2	SSN
	123456789
	999887777
	666884444
	453453453
	987987987

(Supervised by Borg's subordinates)

RESULT	SSN
	123456789
	999887777
	666884444
	453453453
	987987987
	333445555
	987654321

(RESULT1 ∪ RESULT2)

Figure 7.17 A two-level recursive query.

The LEFT OUTER JOIN operation keeps every tuple in the *first* or *left* relation R in R ⟕ S; if no matching tuple is found in S, then the attributes of S in the join result are filled or "padded" with null values. The result of these operations is shown in Figure 7.18.

A similar operation, RIGHT OUTER JOIN, denoted by ⟖, keeps every tuple in the *second* or right relation S in the result of R ⟖ S. A third operation, FULL OUTER JOIN, denoted by ⟗, keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed. The three outer join operations are part of the SQL2 standard (see Chapter 8).

The OUTER UNION operation was developed to take the union of tuples from two relations if the relations are *not union compatible*. This operation will take the UNION of tuples in two relations that are **partially compatible,** meaning that only some of their attributes are union compatible. It is expected that the list of compatible attributes includes a key for both relations. Tuples from the component relations with the same key are represented only once in the result and have values for all attributes in the result. The attributes that are not union compatible from either relation are kept in the result, and tuples that have no values for these attributes are padded with null values. For example, an OUTER UNION can be applied to two relations whose schemas are STUDENT(Name, SSN, Department, Advisor) and FACULTY(Name, SSN, Department, Rank). The resulting relation schema is R(Name, SSN, Department, Advisor, Rank), and all the tuples from both

Sterling EX2011
IPR2025-00270
Page 56 of 69

RESULT	FNAME	MINIT	LNAME	DNAME
	John	B	Smith	null
	Franklin	T	Wong	Research
	Alicia	J	Zelaya	null
	Jennifer	S	Wallace	Administration
	Ramesh	K	Narayan	null
	Joyce	A	English	null
	Ahmad	V	Jabbar	null
	James	E	Borg	Headquarters

Figure 7.18 The LEFT OUTER JOIN operation.

relations are included in the result. Student tuples will have a null for the Rank attribute, whereas faculty tuples will have a null for the Advisor attribute. A tuple that exists in both will have values for all its attributes.[14]

Another capability that exists in most commercial languages (but not in the basic relational algebra) is that of specifying operations on values after they are extracted from the database. For example, arithmetic operations such as +, −, and * can be applied to numeric values.

7.6 Examples of Queries in Relational Algebra

We now give additional examples to illustrate the use of the relational algebra operations. All examples refer to the database of Figure 7.6. In general, the same query can be stated in numerous ways using the various operations. We will state each query in one way and leave it to the reader to come up with equivalent formulations.

QUERY 1

Retrieve the name and address of all employees who work for the 'Research' department.

$$\text{RESEARCH\_DEPT} \leftarrow \sigma_{\text{DNAME='Research'}}(\text{DEPARTMENT})$$

$$\text{RESEARCH\_EMPS} \leftarrow (\text{RESEARCH\_DEPT} \bowtie_{\text{DNUMBER=DNO}} \text{EMPLOYEE})$$

$$\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, ADDRESS}}(\text{RESEARCH\_EMPS})$$

This query could be specified in other ways; for example, the order of the JOIN and SELECT operations could be reversed, or the JOIN could be replaced by a NATURAL JOIN (after renaming).

14. Notice that OUTER UNION is equivalent to a FULL OUTER JOIN if the join attributes are all the common attributes of the two relations.

Sterling EX2011
IPR2025-00270
Page 57 of 69

QUERY 2

For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

$$\text{STAFFORD\_PROJS} \leftarrow \sigma_{\text{PLOCATION='Stafford'}}(\text{PROJECT})$$

$$\text{CONTR\_DEPT} \leftarrow (\text{STAFFORD\_PROJS} \bowtie_{\text{DNUM=DNUMBER}} \text{DEPARTMENT})$$

$$\text{PROJ\_DEPT\_MGR} \leftarrow (\text{CONTR\_DEPT} \bowtie_{\text{MGRSSN=SSN}} \text{EMPLOYEE})$$

$$\text{RESULT} \leftarrow \pi_{\text{PNUMBER, DNUM, LNAME, ADDRESS, BDATE}}(\text{PROJ\_DEPT\_MGR})$$

QUERY 3

Find the names of employees who work on *all* the projects controlled by department number 5.

$$\text{DEPT5\_PROJS}(\text{PNO}) \leftarrow \pi_{\text{PNUMBER}}(\sigma_{\text{DNUM=5}}(\text{PROJECT}))$$

$$\text{EMP\_PRJO}(\text{SSN, PNO}) \leftarrow \pi_{\text{ESSN, PNO}}(\text{WORKS\_ON})$$

$$\text{RESULT\_EMP\_SSNS} \leftarrow \text{EMP\_PRJO} \div \text{DEPT5\_PROJS}$$

$$\text{RESULT} \leftarrow \pi_{\text{LNAME, FNAME}}(\text{RESULT\_EMP\_SSNS} * \text{EMPLOYEE})$$

QUERY 4

Make a list of project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

$$\text{SMITHS}(\text{ESSN}) \leftarrow \pi_{\text{SSN}}(\sigma_{\text{LNAME='Smith'}}(\text{EMPLOYEE}))$$

$$\text{SMITH\_WORKER\_PROJ} \leftarrow \pi_{\text{PNO}}(\text{WORKS\_ON} * \text{SMITHS})$$

$$\text{MGRS} \leftarrow \pi_{\text{LNAME, DNUMBER}}(\text{EMPLOYEE} \bowtie_{\text{SSN=MGRSSN}} \text{DEPARTMENT})$$

$$\text{SMITH\_MANAGED\_DEPTS}(\text{DNUM}) \leftarrow \pi_{\text{DNUMBER}}(\sigma_{\text{LNAME='Smith'}}(\text{MGRS}))$$

$$\text{SMITH\_MGR\_PROJS}(\text{PNO}) \leftarrow \pi_{\text{PNUMBER}}(\text{SMITH\_MANAGED\_DEPTS} * \text{PROJECT})$$

$$\text{RESULT} \leftarrow (\text{SMITH\_WORKER\_PROJS} \cup \text{SMITH\_MGR\_PROJS})$$

QUERY 5

List the names of all employees with two or more dependents.

Strictly speaking, this query cannot be done in the *basic relational algebra*. We have to use the AGGREGATE FUNCTION operation with the COUNT aggregate function. We assume that dependents of the *same* employee have *distinct* DEPENDENT_NAME values.

$$T_1(\text{SSN, NO\_OF\_DEPTS}) \leftarrow {}_{\text{ESSN}}\mathfrak{F}_{\text{COUNT DEPENDENT\_NAME}}(\text{DEPENDENT})$$

$$T_2 \leftarrow \sigma_{\text{NO\_OF\_DEPS} \geq 2}(T_1)$$

$$\text{RESULT} \leftarrow \pi_{\text{LNAME, FNAME}}(T_2 * \text{EMPLOYEE})$$

Sterling EX2011
IPR2025-00270
Page 58 of 69

QUERY 6

Retrieve the names of employees who have no dependents.

$$ALL\_EMPS \leftarrow \pi_{SSN}(EMPLOYEE)$$

$$EMPS\_WITH\_DEPS(SSN) \leftarrow \pi_{ESSN}(DEPENDENT)$$

$$EMPS\_WITHOUT\_DEPS \leftarrow (ALL\_EMPS - EMPS\_WITH\_DEPS)$$

$$RESULT \leftarrow \pi_{LNAME,\ FNAME}(EMPS\_WITHOUT\_DEPS * EMPLOYEE)$$

QUERY 7

List the names of managers who have at least one dependent.

$$MGRS(SSN) \leftarrow \pi_{MGRSSN}(DEPARTMENT)$$

$$EMPS\_WITH\_DEPS(SSN) \leftarrow \pi_{ESSN}(DEPENDENT)$$

$$MGRS\_WITH\_DEPS \leftarrow (MGRS \cap EMPS\_WITH\_DEPS)$$

$$RESULT \leftarrow \pi_{LNAME,\ FNAME}(MGRS\_WITH\_DEPS * EMPLOYEE)$$

As we mentioned earlier, the same query can in general be specified in many different ways. For example, the operations can often be applied in various sequences. In addition, some operations can be used to replace others; for example, the INTERSECTION operation in Query 7 can be replaced by a NATURAL JOIN. As an exercise, try to do each of the above example queries using different operations.[15] In Chapters 8 and 9 we will show how these queries are written in other relational languages.

7.7 Summary

In this chapter we presented the modeling concepts provided by the relational model of data. We also discussed the relational algebra and additional operations that can be used to manipulate relations. We started by introducing the concepts of domains, attributes, and tuples. We then defined a relation schema as a list of attributes that describe the structure of a relation. A relation, or relation state, is a set of tuples that conform to the schema.

Several characteristics differentiate relations from ordinary tables or files. The first is that tuples in a relation are not ordered. The second involves the ordering of attributes in a relation schema and the corresponding ordering of values within a tuple. We gave an alternative definition of relation that does not require these two orderings, but we continued to use the first definition, which requires attributes and tuple values to be ordered, for convenience. We then discussed values in tuples and introduced null values to represent missing or unknown information.

15. When queries are optimized (see Chapter 18), the system will choose a particular sequence of operations that corresponds to an execution strategy that can be executed efficiently.

Sterling EX2011
IPR2025-00270
Page 59 of 69

We then discussed the relational model constraints, starting with domain constraints, then key constraints, including the concepts of superkey, candidate key, and primary key, and the NOT NULL constraint on attributes. We then defined relational databases and relational database schemas. Additional relational constraints include the entity integrity constraint, which prohibits primary key attributes from being null. The interrelation constraint of referential integrity was then described, which is used to maintain consistency of references among tuples from different relations.

The modification operations on the relational model are Insert, Delete, and Update. Each operation may violate certain types of constraints. Whenever an operation is applied, the database state after the operation is executed must be checked to ensure that no constraints are violated.

We then described the basic relational algebra, which is a set of operations for manipulating relations that can be used to specify queries. We presented the various operations and illustrated the types of queries for which each is used. Table 7.1 lists the various relational algebra operations we discussed. The unary relational operators SELECT and PROJECT, as well as the RENAME operation, were discussed first. Then we discussed binary set theoretic operations requiring that relations on which they are applied be union compatible; these include UNION, INTERSECTION, and SET DIFFERENCE. The CARTESIAN PRODUCT operation is another set operation that can be used to combine tuples from two relations, producing all possible combinations. We showed how CARTESIAN PRODUCT followed by SELECT can identify related tuples from two relations. The JOIN operations can directly identify and combine related tuples. Join operations include THETA JOIN, EQUIJOIN, and NATURAL JOIN.

We then discussed some important types of queries that *cannot* be stated with the basic relational algebra operations. We introduced the AGGREGATE FUNCTION operation to deal with aggregate types of requests. We discussed recursive queries and showed how some types of recursive queries can be specified. We then presented the OUTER JOIN and OUTER UNION operations, which extend JOIN and UNION.

Review Questions

7.1. Define the following terms: *domain*, *attribute*, *n-tuple*, *relation schema*, *relation state*, *degree of a relation*, *relational database schema*, *relational database state*.

7.2. Why are tuples in a relation not ordered?

7.3. Why are duplicate tuples not allowed in a relation?

7.4. What is the difference between a key and a superkey?

7.5. Why do we designate one of the candidate keys of a relation to be the primary key?

7.6. Discuss the characteristics of relations that make them different from ordinary tables and files.

7.7. Discuss the various reasons that lead to the occurrence of null values in relations.

7.8. Discuss the entity integrity and referential integrity constraints. Why is each considered important?

7.9. Define *foreign key*. What is this concept used for? How does it play a role in the join operation?

Sterling EX2011
IPR2025-00270
Page 60 of 69

Table 7.1 Operations of Relational Algebra

Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, or $R_1 \bowtie_{\text{(<join attributes 1>), (<join attributes 2>)}} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{<join condition>}} R_2$, or $R_1 *_{\text{(<join attributes 1>), (<join attributes 2>)}} R_2$ or $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation R(X) that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

7.10. Discuss the various update operations on relations and the types of integrity constraints that must be checked for each update operation.
7.11. List the operations of relational algebra and the purpose of each.
7.12. What is union compatibility? Why do the UNION, INTERSECTION, and DIFFERENCE operations require that the relations on which they are applied be union compatible?
7.13. Discuss some types of queries for which renaming of attributes is necessary in order to specify the query unambiguously.
7.14. Discuss the various types of JOIN operations. Why is theta join required?
7.15. What is the FUNCTION operation? What is it used for?

Sterling EX2011
IPR2025-00270
Page 61 of 69

7.16. How are the OUTER JOIN operations different from the (inner) JOIN operations? How is the OUTER UNION operation different from UNION?

Exercises

7.17. Show the result of each of the example queries in Section 7.6 as it would apply to the database of Figure 7.6.

7.18. Specify the following queries on the database schema shown in Figure 7.5, using the relational operators discussed in this chapter. Also show the result of each query as it would apply to the database of Figure 7.6.

a. Retrieve the names of all employees in department 5 who work more than 10 hours per week on the 'ProductX' project.
b. List the names of all employees who have a dependent with the same first name as themselves.
c. Find the names of all employees who are directly supervised by 'Franklin Wong'.
d. For each project, list the project name and the total hours per week (by all employees) spent on that project.
e. Retrieve the names of all employees who work on every project.
f. Retrieve the names of all employees who do not work on any project.
g. For each department, retrieve the department name and the average salary of all employees working in that department.
h. Retrieve the average salary of all female employees.
i. Find the names and addresses of all employees who work on at least one project located in Houston but whose department has no location in Houston.
j. List the last names of all department managers who have no dependents.

7.19. Suppose that each of the following update operations is applied directly to the database of Figure 7.7. Discuss *all* integrity constraints violated by each operation, if any, and the different ways of enforcing these constraints.

a. Insert <'Robert', 'F', 'Scott', '943775543', '1952-06-21', '2365 Newcastle Rd, Bellaire, TX', M, 58000, '888665555', 1> into EMPLOYEE.
b. Insert <'ProductA', 4, 'Bellaire', 2> into PROJECT.
c. Insert <'Production', 4, '943775543', '1998-10-01'> into DEPARTMENT.
d. Insert <'677678989', null, '40.0'> into WORKS_ON.
e. Insert <'453453453', 'John', M, '1970-12-12', 'SPOUSE'> into DEPENDENT.
f. Delete the WORKS_ON tuples with ESSN = '333445555'.
g. Delete the EMPLOYEE tuple with SSN = '987654321'.
h. Delete the PROJECT tuple with PNAME = 'ProductX'.
i. Modify the MGRSSN and MGRSTARTDATE of the DEPARTMENT tuple with DNUMBER = 5 to '123456789' and '1999-10-01', respectively.
j. Modify the SUPERSSN attribute of the EMPLOYEE tuple with SSN = '999887777' to '943775543'.
k. Modify the HOURS attribute of the WORKS_ON tuple with ESSN = '999887777' and PNO = 10 to '5.0'.

Sterling EX2011
IPR2025-00270
Page 62 of 69

AIRPORT

AIRPORT_CODE	NAME	CITY	STATE

FLIGHT

NUMBER	AIRLINE	WEEKDAYS

FLIGHT_LEG

FLIGHT_NUMBER	LEG_NUMBER	DEPARTURE_AIRPORT_CODE	SCHEDULED_DEPARTURE_TIME	ARRIVAL_AIRPORT_CODE	SCHEDULED_ARRIVAL_TIME

LEG_INSTANCE

FLIGHT_NUMBER	LEG_NUMBER	DATE	NUMBER_OF_AVAILABLE_SEATS	AIRPLANE_ID	DEPARTURE_AIRPORT_CODE	DEPARTURE_TIME	ARRIVAL_AIRPORT_CODE	ARRIVAL_TIME

FARES

FLIGHT_NUMBER	FARE_CODE	AMOUNT	RESTRICTIONS

AIRPLANE_TYPE

TYPE_NAME	MAX_SEATS	COMPANY

CAN_LAND

AIRPLANE_TYPE_NAME	AIRPORT_CODE

AIRPLANE

AIRPLANE_ID	TOTAL_NUMBER_OF_SEATS	AIRPLANE_TYPE

SEAT_RESERVATION

FLIGHT_NUMBER	LEG_NUMBER	DATE	SEAT_NUMBER	CUSTOMER_NAME	CUSTOMER_PHONE

Figure 7.19 The AIRLINE relational database schema.

7.20. Consider the AIRLINE relational database schema shown in Figure 7.19, which describes a database for airline flight information. Each FLIGHT is identified by a flight NUMBER, and consists of one or more FLIGHT_LEGS with LEG_NUMBERS 1, 2, 3, etc. Each leg has scheduled arrival and departure times and airports and has many LEG_INSTANCES—one for each DATE on which the flight travels. FARES are kept for each flight. For each leg instance, SEAT_RESERVATIONS are kept, as are the AIRPLANE used on the leg and the actual arrival and departure times and airports. An AIRPLANE is identified by an AIRPLANE_ID and is of a particular AIRPLANE_TYPE. CAN_LAND relates

Sterling EX2011
IPR2025-00270
Page 63 of 69

AIRPLANE_TYPES to the AIRPORTS in which they can land. An AIRPORT is identified by an AIRPORT_CODE. Specify the following queries in relational algebra:

a. For each flight, list the flight number, the departure airport for the first leg of the flight, and the arrival airport for the last leg of the flight.
b. List the flight numbers and weekdays of all flights or flight legs that depart from Houston Intercontinental Airport (airport code 'IAH') and arrive in Los Angeles International Airport (airport code 'LAX').
c. List the flight number, departure airport code, scheduled departure time, arrival airport code, scheduled arrival time, and weekdays of all flights or flight legs that depart from some airport in the city of Houston and arrive at some airport in the city of Los Angeles.
d. List all fare information for flight number 'CO197'.
e. Retrieve the number of available seats for flight number 'CO197' on '1999-10-09'.

7.21. Consider an update for the AIRLINE database to enter a reservation on a particular flight or flight leg on a given date.

a. Give the operations for this update.
b. What types of constraints would you expect to check?
c. Which of these constraints are key, entity integrity, and referential integrity constraints, and which are not?
d. Specify all the referential integrity constraints on Figure 7.19.

7.22. Consider the relation

CLASS(Course#, Univ_Section#, InstructorName, Semester, BuildingCode, Room#, TimePeriod, Weekdays, CreditHours).

This represents classes taught in a university, with unique Univ_Section#. Identify what you think should be various candidate keys, and write in your own words the constraints under which each candidate key would be valid.

7.23. Consider the LIBRARY relational schema shown in Figure 7.20, which is used to keep track of books, borrowers, and book loans. Referential integrity constraints are shown as directed arcs in Figure 7.20, as in the notation of Figure 7.7. Write down relational expressions for the following queries on the LIBRARY database:

a. How many copies of the book titled *The Lost Tribe* are owned by the library branch whose name is 'Sharpstown'?
b. How many copies of the book titled *The Lost Tribe* are owned by each library branch?
c. Retrieve the names of all borrowers who do not have any books checked out.
d. For each book that is loaned out from the 'Sharpstown' branch and whose DueDate is today, retrieve the book title, the borrower's name, and the borrower's address.
e. For each library branch, retrieve the branch name and the total number of books loaned out from that branch.
f. Retrieve the names, addresses, and number of books checked out for all borrowers who have more than five books checked out.

Sterling EX2011
IPR2025-00270
Page 64 of 69

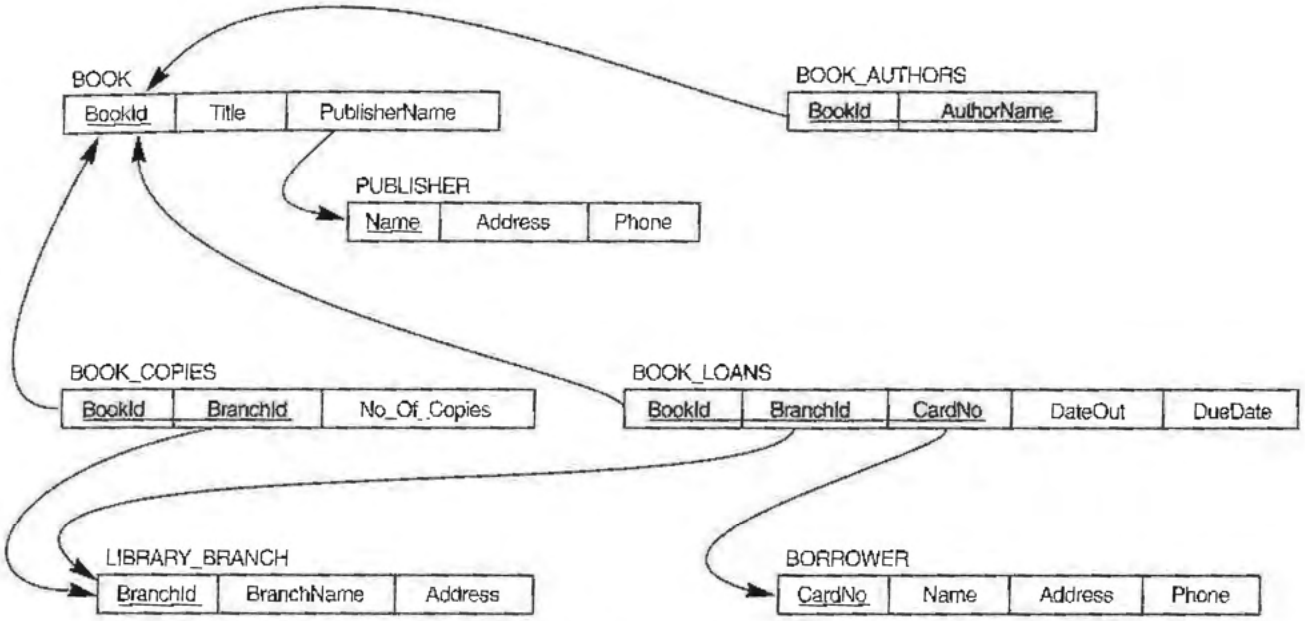


Figure 7.20 A relational database schema for a LIBRARY database.

g. For each book authored (or coauthored) by 'Stephen King,' retrieve the title and the number of copies owned by the library branch whose name is 'Central.'

7.24. Consider the following six relations for an order processing database application in a company:

CUSTOMER(Cust#, Cname, City)

ORDER(Order#, Odate, Cust#, Ord_Amt)

ORDER_ITEM(Order#, Item#, Qty)

ITEM(Item#, Unit_price)

SHIPMENT(Order#, Warehouse#, Ship_date)

WAREHOUSE(Warehouse#, City)

Here, Ord_Amt refers to total dollar amount of an order; Odate is the date the order was placed; Ship_date is the date an order is shipped from the warehouse. Assume that an order can be shipped from several warehouses. Specify the foreign keys for the above schema, stating any assumptions you make. Then specify the following queries in relational algebra:

a. List the Order# and Ship_date for all orders shipped from Warehouse number 'W2'.

b. List the Warehouse information from which the Customer named 'Jose Lopez' was supplied his orders. Produce a listing: Order#, Warehouse#.

c. Produce a listing: CUSTNAME, #OFORDERS, AVG_ORDER_AMT, where the middle column is the total number of orders by the customer and the last column is the average order amount for that customer.

Sterling EX2011
IPR2025-00270
Page 65 of 69

d. List the orders that were not shipped within 30 days of ordering.
e. List the Order# for orders that were shipped from *all* warehouses that the company has in New York.

7.25. Consider the following relations for a database that keeps track of business trips of salespersons in a sales office:

SALESPERSON(SSN, Name, Start_Year, Dept_No)

TRIP(SSN, From_City, To_City, Departure_Date, Return_Date, Trip_ID)

EXPENSE(Trip_ID, Account#, Amount)

Specify the foreign keys for the above schema, stating any assumptions you make. Then specify the following queries in relational algebra:

a. Give the details (all attributes of TRIP relation) for trips that exceeded $2000 in expenses.
b. Print the SSN of salesman who took trips to 'Honolulu'.
c. Print the total trip expenses incurred by the salesman with SSN = '234-56-7890'.

7.26. Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:

STUDENT(SSN, Name, Major, Bdate)

COURSE(Course#, Cname, Dept)

ENROLL(SSN, Course#, Quarter, Grade)

BOOK_ADOPTION(Course#, Quarter, Book_ISBN)

TEXT(Book_ISBN, Book_Title, Publisher, Author)

Specify the foreign keys for the above schema, stating any assumptions you make. Then specify the following queries in relational algebra:

a. List the number of courses taken by all students named 'John Smith' in Winter 1999 (i.e., Quarter = 'W99').
b. Produce a list of textbooks (include Course#, Book_ISBN, Book_Title) for courses offered by the 'CS' department that have used more than two books.
c. List any department that has *all* its adopted books published by 'BC Publishing'.

7.27. Consider the two tables T1 and T2 shown in Figure 7.21. Show the results of the following operations:

a. $T1 \bowtie_{T1.P = T2.A} T2$
b. $T1 \bowtie_{T1.Q = T2.B} T2$
c. $T1 \overline{\bowtie}_{T1.P = T2.A} T2$
d. $T1 \overline{\bowtie}_{T1.Q = T2.B} T2$
e. $T1 \cup T2$
f. $T1 \bowtie_{(T1.P = T2.A \text{ AND } T1.R = T2.C)} T2$

7.28. Consider the following relations for a database that keeps track of auto sales in a car dealership (Option refers to some optional equipment installed on an auto):

CAR(Serial-No, Model, Manufacturer, Price)

Sterling EX2011
IPR2025-00270
Page 66 of 69

Table T1

P	Q	R
10	a	5
15	b	8
25	a	6

Table T2

A	B	C
10	b	6
25	c	3
10	b	5

Figure 7.21 A database state for the relations T1 and T2.

OPTIONS(Serial-No, Option-Name, Price)

SALES(Salesperson-id, Serial-No, Date, Sale-price)

SALESPERSON(Salesperson-id, Name, Phone)

First, specify the foreign keys for the above schema, stating any assumptions you make. Next, populate the relations with a few example tuples, and then show an example of an insertion in the SALES and SALESPERSON relations that *violates* the referential integrity constraints and another insertion that does not. Then specify the following queries in relational algebra:

a. For the salesperson named 'Jane Doe', list the following information for all the cars she sold: Serial#, Manufacturer, Sale-price.
b. List the Serial# and Model of cars that have no options.
c. Consider the natural join operation between SALESPERSON and SALES. What is the meaning of a left outer join for these tables (do not change the order of relations). Explain with an example.
d. Write a query in relational algebra involving selection and one set operation and say in words what the query does.

Selected Bibliography

The relational model was introduced by Codd (1970) in a classic paper. Codd also introduced relational algebra and laid the theoretical foundations for the relational model in a series of papers (Codd 1971, 1972, 1972a, 1974); he was later given the Turing award, the highest honor of the ACM, for his work on the relational model. In a later paper, Codd (1979) discussed extending the relational model to incorporate more meta-data and semantics about the relations; he also proposed a three-valued logic to deal with uncertainty in relations and incorporating NULLs in the relational algebra. The resulting model is known as RM/T. Childs (1968) had earlier used set theory to model databases. More recently, Codd (1990) published a book examining over 300 features of the relational data model and database systems.

Since Codd's pioneering work, much research has been conducted on various aspects of the relational model. Todd (1976) describes an experimental DBMS called PRTV that directly implements the relational algebra operations. Date (1983a) discusses outer joins.

Sterling EX2011
IPR2025-00270
Page 67 of 69

Schmidt and Swenson (1975) introduces additional semantics into the relational model by classifying different types of relations. Chen's (1976) Entity Relationship model, which was discussed in Chapter 3, was a means to communicate the real-world semantics of a relational database at the conceptual level. Wiederhold and Elmasri (1979) introduces various types of connections between relations to enhance its constraints. Work on extending relational operations is discussed by Carlis (1986) and Ozsoyoglu et al. (1985). Cammarata et al. (1989) extends the relational model integrity constraints and joins. Extensions of the relational model are discussed in Chapter 13. Additional bibliographic notes for other aspects of the relational model and its languages, systems, extensions, and theory are given in Chapters 8, 9, 10, 13, 14, 15, 18, 22, 23, and 24.

Sterling EX2011
IPR2025-00270
Page 68 of 69

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



Sterling EX2011
IPR2021-00270
Page 69 of 69

ISBN 0-8053-1755-4