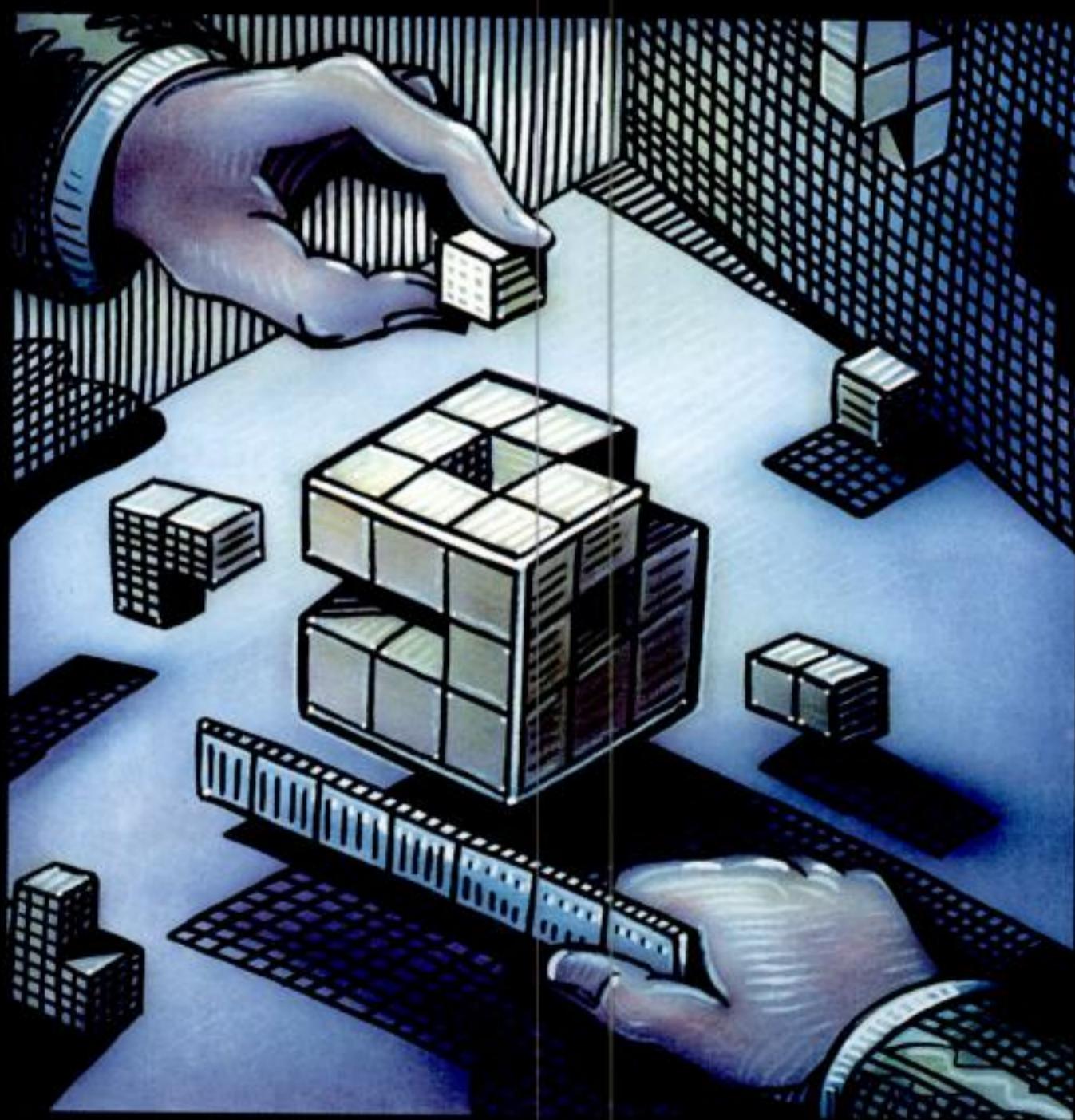


DATABASE MANAGEMENT SYSTEMS



G K GUPTA

DATABASE MANAGEMENT SYSTEMS

G K Gupta

*Adjunct Professor of Computer Science
Monash University, Clayton
Australia*



Tata McGraw Hill Education Private Limited
NEW DELHI

McGraw-Hill Offices

**New Delhi New York St Louis San Francisco Auckland Bogotá Caracas
Kuala Lumpur Lisbon London Madrid Mexico City Milan Montreal
San Juan Santiago Singapore Sydney Tokyo Toronto**



Tata McGraw-Hill

Published by the Tata McGraw Hill Education Private Limited,
7 West Patel Nagar, New Delhi 110 008.

Database Management Systems

Copyright © 2011, by Tata McGraw Hill Education Private Limited.

No part of this publication may be reproduced or distributed in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise or stored in a database or retrieval system without the prior written permission of the publishers. The program listings (if any) may be entered, stored and executed in a computer system, but they may not be reproduced for publication.

This edition can be exported from India only by the publishers,
Tata McGraw Hill Education Private Limited.

ISBN (13): 978-0-07-107273-1

ISBN (10): 0-07-107273-X

Vice President and Managing Director —McGraw-Hill Education, Asia Pacific Region: *Ajay Shukla*

Head—Higher Education Publishing and Marketing: *Vibha Mahajan*

Manager: Sponsoring—SEM & Tech Ed.: *Shalini Jha*

Assoc. Sponsoring Editor: *Suman Sen*

Development Editor: *Surbhi Suman*

Sr Copy Editor: *Nimisha Kapoor*

Sr Production Manager: *Satinder S Baveja*

Sr Production Manager: *P L Pandita*

Dy Marketing Manager—SEM & Tech Ed.: *Biju Ganesan*

Sr Product Specialist—SEM & Tech Ed.: *John Mathews*

General Manager—Production: *Rajender P Ghansela*

Asst General Manager—Production: *B L Dogra*

Information contained in this work has been obtained by Tata McGraw-Hill, from sources believed to be reliable. However, neither Tata McGraw-Hill nor its authors guarantee the accuracy or completeness of any information published herein, and neither Tata McGraw-Hill nor its authors shall be responsible for any errors, omissions, or damages arising out of use of this information. This work is published with the understanding that Tata McGraw-Hill and its authors are supplying information but are not attempting to render engineering or other professional services. If such services are required, the assistance of an appropriate professional should be sought.

Typeset at Tej Composers, WZ-391, Madipur, New Delhi 110063, and printed at Sheel Printers Pvt. Ltd., D-132, Hoisery Complex, Phase-II, Noida.

Cover Printer: Sheel Printers

RAZYCRZCDXCLA

Contents

<i>Preface</i>	xvi
<i>Visual Guide</i>	xxi

1. INTRODUCTION TO DATABASE MANAGEMENT 1

<u>1.1 Introduction</u>	2
<u>1.2 An Example of a Database</u>	3
<u>1.3 What is a Database?</u>	5
<u>1.4 Characteristics of Databases</u>	10
<u>1.5 Data Governance and Importance of Databases</u>	11
<u>1.6 History of Database Software</u>	12
<u>1.7 File Systems vs Database Systems</u>	13
<u>1.8 What is a DBMS?</u>	14
<u>1.9 Users of a Database System</u>	16
<u>1.10 Advantages of using an Enterprise Database</u>	18
<u>1.11 Concerns when using an Enterprise Database</u>	20
<u>1.12 Three-Level DBMS Architecture and Data Abstraction</u>	21
<u>1.13 Data Independence</u>	24
1.14 DBMS System Architecture	25
1.15 Database Administrator	28
1.16 Designing an Enterprise Database System	29
1.17 Future of Databases—Web Databases	31
1.18 Choosing a DBMS	32
<i>Summary</i>	33
<i>Review Questions</i>	34
<i>Short Answer Questions</i>	35
<i>Multiple Choice Questions</i>	36
<i>Exercises</i>	38
<i>Projects</i>	39
<i>Lab Exercises</i>	39
<i>Bibliography</i>	40

2. ENTITY-RELATIONSHIP DATA MODEL 43

2.1 Introduction	44
2.2 Benefits of Data Modeling	45
<u>2.3 Types of Models</u>	46

<u>2.4 Phases of Database Modeling</u>	<u>46</u>
2.5 The Entity-Relationship Model	48
2.6 Generalization, Specialization and Aggregation	68
2.7 Extended Entity-Relationship Model	71
<u>2.8 The Database Design Process</u>	<u>74</u>
2.9 Strengths of the E-R Model	76
2.10 Weaknesses of the E-R Model	76
<u>2.11 A Case Study of Building an E-R Model</u>	<u>78</u>
<u>2.12 Evaluating Data Model Quality</u>	<u>89</u>
<i>Summary</i>	<u>91</u>
<i>Review Questions</i>	<u>92</u>
<i>Short Answer Questions</i>	<u>92</u>
<i>Multiple Choice Questions</i>	<u>93</u>
<i>Exercises</i>	<u>96</u>
<i>Projects</i>	<u>98</u>
<i>Lab Exercises</i>	<u>99</u>
<i>Bibliography</i>	<u>101</u>

3. RELATIONAL MODEL

104

<u>3.1 Introduction</u>	<u>105</u>
<u>3.2 Data Structure</u>	<u>106</u>
<u>3.3 Mapping the E-R Model to the Relational Model</u>	<u>109</u>
<u>3.4 Data Manipulation</u>	<u>118</u>
<u>3.5 Data Integrity</u>	<u>119</u>
<u>3.6 Advantages of the Relational Model</u>	<u>124</u>
<u>3.7 Rules for Fully Relational Systems</u>	<u>125</u>
3.8 Earlier Database Models —Hierarchical and Network Models	127
<i>Summary</i>	<u>132</u>
<i>Review Questions</i>	<u>132</u>
<i>Short Answer Questions</i>	<u>133</u>
<i>Multiple Choice Questions</i>	<u>133</u>
<i>Exercises</i>	<u>136</u>
<i>Projects</i>	<u>139</u>
<i>Lab Exercises</i>	<u>141</u>
<i>Bibliography</i>	<u>141</u>

4. RELATIONAL ALGEBRA AND RELATIONAL CALCULUS

145

<u>4.1 Introduction</u>	<u>146</u>
<u>4.2 Relational Algebra</u>	<u>146</u>
<u>4.3 Relational Algebra Queries</u>	<u>163</u>
<u>4.4 Relational Calculus</u>	<u>167</u>
<u>4.5 Relational Algebra vs Relational Calculus</u>	<u>180</u>
<i>Summary</i>	<u>181</u>
<i>Review Questions</i>	<u>182</u>

<i>Short Answer Questions</i>	182
<i>Multiple Choice Questions</i>	183
<i>Exercises</i>	185
<i>Projects</i>	186
<i>Lab Exercises</i>	186
<i>Bibliography</i>	187
5. STRUCTURED QUERY LANGUAGE (SQL)	188
<i>5.1 Introduction</i>	189
<i>5.2 SQL</i>	190
<i>5.3 Data Definition—CREATE, ALTER and DROP Commands</i>	194
<i>5.4 Data Manipulation—SQL Data Retrieval</i>	197
<i>5.5 Views—Using Virtual Tables in SQL</i>	231
<i>5.6 Using SQL in Procedural Programming Languages—Embedded SQL and Dynamic SQL</i>	234
<i>5.7 Data Integrity and Constraints</i>	239
<i>5.8 Triggers and Active Databases</i>	241
<i>5.9 Data Control—Database Security</i>	241
<i>5.10 Further Data Definition Commands</i>	242
<i>5.11 Summary of Some SQL Commands</i>	244
<i>5.12 SQL Standards</i>	246
<i>Summary</i>	246
<i>Review Questions</i>	247
<i>Short Answer Questions</i>	248
<i>Multiple Choice Questions</i>	249
<i>Exercises</i>	258
<i>Projects</i>	260
<i>Lab Exercise</i>	260
<i>Bibliography</i>	261
6. NORMALIZATION	267
<i>6.1 Introduction</i>	268
<i>6.2 Possible Undesirable Properties of Relations and Schema Refinement</i>	269
<i>6.3 Functional Dependency—Single-Valued Dependencies</i>	274
<i>6.4 Single-Valued Normalization</i>	279
<i>6.5 Desirable Properties of Decompositions</i>	285
<i>6.6 Multivalued Dependencies</i>	289
<i>6.7 Denormalization</i>	296
<i>6.8 Inclusion Dependency</i>	298
<i>Summary</i>	299
<i>Review Questions</i>	299
<i>Short Answer Questions</i>	300
<i>Multiple Choice Questions</i>	301
<i>Exercises</i>	306

<i>Project</i>	308
<i>Lab Exercises</i>	309
<i>Bibliography</i>	309

7. PHYSICAL STORAGE AND INDEXING

312

7.1 Introduction	313
7.2 Different Types of Computer Memories	314
7.3 Secondary Storage—Hard Disks	317
7.4 Buffer Management	321
7.5 Introduction to File Structures	322
7.6 Unstructured and Unordered Files (Heap Files)	323
7.7 Sequential Files (Sorted Files)	324
7.8 Index and Index Types	326
7.9 The Indexed-Sequential File	330
7.10 The B-tree	332
7.11 The B ⁺ -tree	341
7.12 Advantages and Disadvantages of B-trees and B+-trees	342
7.13 Static Hashing	343
7.14 External Hashing—Files on Disk	352
7.15 Dynamic Hashing for External Files	352
7.16 Inverted Files	360
<i>Summary</i>	361
<i>Review Questions</i>	362
<i>Short Answer Questions</i>	363
<i>Multiple Choice Questions</i>	364
<i>Exercises</i>	369
<i>Projects</i>	370
<i>Lab Exercises</i>	370
<i>Bibliography</i>	371

8. QUERY PROCESSING

373

8.1 Introduction	374
8.2 Issues in Query Optimization	375
8.3 Steps in Query Processing	377
8.4 System Catalog or Metadata	377
8.5 Query Parsing	379
8.6 Query Optimization	380
8.7 Access Paths	387
8.8 Query Code Generation	388
8.9 Query Execution	389
8.10 Algorithms for Computing Selection and Projection	389
8.11 Algorithms for Computing a Join	391
8.12 Computing Aggregation Functions	395
8.13 Cost Based Query Optimization in System R	396
8.14 Correlated Subqueries	401

8.15 Semantic Query Optimization	402
<i>Summary</i>	403
<i>Review Questions</i>	404
<i>Short Answer Questions</i>	405
<i>Multiple Choice Questions</i>	405
<i>Exercises</i>	409
<i>Projects</i>	410
<i>Lab Exercise</i>	411
<i>Bibliography</i>	411
9. TRANSACTION MANAGEMENT AND CONCURRENCY	414
9.1 Introduction	415
9.2 The Concept of a Transaction	416
9.3 Examples of Concurrency Anomalies	419
9.4 Schedules	422
9.5 Schedules and Recoverability	423
9.6 Serializability	424
9.7 Hierarchy of Serializable Schedules	431
9.8 Concurrency Control and Enforcing Serializability	432
9.9 Deadlocks	435
9.10 Lock Granularity	438
9.11 Multiple Granularity and Intention Locking	439
9.12 Nonlocking Technique—Timestamping Control	441
9.13 Nonlocking Technique—Optimistic Control	443
9.14 Transaction Support in SQL	444
9.15 Evaluation of Concurrency Control Mechanisms	445
<i>Summary</i>	446
<i>Review Questions</i>	446
<i>Short Answer Questions</i>	447
<i>Multiple Choice Questions</i>	448
<i>Exercises</i>	451
<i>Projects</i>	453
<i>Lab Exercises</i>	453
<i>Bibliography</i>	454
10. DATABASE BACKUP AND RECOVERY	457
10.1 Introduction	458
10.2 Database System Failure	459
10.3 Database Backup	461
10.4 Recovery from a Failure	465
10.5 The Concept of a Log	466
10.6 Recovery and Buffer Management	469
10.7 Log-based Transaction Recovery Techniques	471
10.8 Types of Recovery Techniques	473
10.9 Log-based Immediate Update Recovery Technique	475

10.10 Log-based Deferred Update Recovery Technique	477
10.11 System Checkpoints	479
10.12 Implementing a Log-based Recovery System—ARIES	479
10.13 Summary of Log-based Methods	480
10.14 Shadow Page Schemes	481
10.15 Evaluation of Recovery Techniques	482
10.16 Recovering from a Media Failure	483
<i>Summary</i>	483
<i>Review Questions</i>	484
<i>Short Answer Questions</i>	485
<i>Multiple Choice Questions</i>	485
<i>Exercises</i>	488
<i>Projects</i>	488
<i>Lab Exercises</i>	489
<i>Bibliography</i>	489

11. DATABASE SECURITY

492

11.1 Introduction	493
11.2 Security Violations	494
11.3 Identification and Authentication	497
11.4 Authorization or Access Control	499
11.5 Security of Statistical Databases	508
11.6 Audit Policy	512
11.7 Internet Applications and Encryption	513
11.8 Security of Outsourced Databases	515
<i>Summary</i>	515
<i>Review Questions</i>	516
<i>Short Answer Questions</i>	517
<i>Multiple Choice Questions</i>	517
<i>Exercises</i>	520
<i>Projects</i>	521
<i>Lab Exercises</i>	521
<i>Bibliography</i>	522

12. INTEGRITY CONSTRAINTS AND ACTIVE DATABASES

524

12.1 Introduction	525
12.2 What is Database Integrity?	526
12.3 Enforcement Principles	529
12.4 Integrity Constraints in SQL:1999	531
12.5 SQL and Syntactic Integrity	532
12.6 SQL and Semantic Integrity	537
12.7 Active Database Systems	543
<i>Summary</i>	545
<i>Review Questions</i>	546

<i>Short Answer Questions</i>	546
<i>Multiple Choice Questions</i>	547
<i>Exercises</i>	549
<i>Project</i>	550
<i>Lab Exercises</i>	550
<i>Bibliography</i>	550

13. DISTRIBUTED DATABASES 553

13.1 Introduction	554
13.2 Features of a Distributed Database System	557
13.3 Types and Examples of Distributed Database Systems	558
13.4 Advantages and Disadvantages of Distributed Systems	560
13.5 Objectives in Designing a Distributed Database System	562
13.6 Distributed System Design—Fragmentation	566
13.7 Distributed Design Issue—Data Replication and Data Allocation	576
13.8 Distributed Query Processing	578
13.9 An Example of Distributed Query Processing	579
13.10 Distributed Transaction Processing—Concurrency and Recovery	582
<i>Summary</i>	583
<i>Review Questions</i>	584
<i>Short Answer Questions</i>	585
<i>Multiple Choice Questions</i>	585
<i>Exercises</i>	588
<i>Projects</i>	589
<i>Lab Exercises</i>	590
<i>Bibliography</i>	590

14. OBJECT-ORIENTED DATABASES 592

14.1 Introduction	593
14.2 The Object Model	595
14.3 Object Specification Languages	605
14.4 Object Query Language (OQL)	610
14.5 Two 1990 Manifestos	616
14.6 ODMS vs RDBMS	619
14.7 Object-Relational Model and OO Features of SQL	619
14.8 Why ODMS are Not Used More Widely?	620
<i>Summary</i>	620
<i>Review Questions</i>	621
<i>Short Answer Questions</i>	622
<i>Multiple Choice Questions</i>	623
<i>Exercises</i>	625
<i>Projects</i>	626
<i>Lab Exercises</i>	627
<i>Bibliography</i>	627

15. DATA WAREHOUSES AND OLAP	630
15.1 Introduction	631
15.2 Example Queries Made by Managers	632
15.3 A Separate Enterprise Database	635
15.4 Operational Data Store (ODS)	635
15.5 Data Warehouse	636
15.6 Reasons for Implementing a Data Warehouse	638
15.7 When Implementing a Data Warehouse May Not be Appropriate	639
15.8 Guidelines for Implementing an Enterprise Data Warehouse	639
15.9 Architecture of a Data Warehouse System	642
15.10 Data Preparation—Extract Transform Load (ETL)	643
15.11 Data Modeling for Data Warehousing	646
15.12 OLAP	650
15.13 Characteristics of OLAP Systems	652
15.14 The Multidimensional View and Data Cube	653
15.15 Data Cube Implementations	657
15.16 Data Cube Operations	659
15.17 Guidelines for OLAP Implementation	661
15.18 Data Warehousing Case Study	662
<i>Summary</i>	663
<i>Review Questions</i>	663
<i>Short Answer Questions</i>	664
<i>Multiple Choice Questions</i>	665
<i>Exercises</i>	667
<i>Project</i>	668
<i>Lab Exercises</i>	668
<i>Bibliography</i>	669
16. DATA MINING	672
16.1 What is Data Mining?	673
16.2 The Data Mining Process	675
16.3 Association Rules Mining	676
16.4 Classification and Prediction	681
16.5 Clustering	682
16.6 Text Data Mining	683
16.7 Web Data Mining	685
16.8 Guidelines for Successful Data Mining	687
16.9 Data Mining Case Studies	688
<i>Summary</i>	691
<i>Review Questions</i>	692
<i>Short Answer Questions</i>	693
<i>Multiple Choice Questions</i>	693
<i>Exercises</i>	696
<i>Projects</i>	697

<i>Lab Exercises</i>	699
<i>Bibliography</i>	700
17. WEB DATABASES AND XML	703
17.1 Introduction	704
17.2 Web Documents vs Classical Text Documents	705
17.3 Web Terminology	705
17.4 Web Structure	706
17.5 Information Retrieval Terminology	707
17.6 Classical Information Retrieval	708
17.7 Storage and Retrieval on the Web—The Search Engine	710
17.8 Building the Search Engine Database—The Crawler	711
17.9 Querying the Web	713
17.10 Introduction—Markup Languages	715
17.11 Semistructured Data	716
17.12 XML	717
17.13 XML Databases	724
17.14 Relational XML Database	725
17.15 Native XML Database	726
<i>Summary</i>	730
<i>Review Questions</i>	730
<i>Short Answer Questions</i>	731
<i>Multiple Choice Questions</i>	732
<i>Exercises</i>	734
<i>Projects</i>	734
<i>Lab Exercises</i>	735
<i>Bibliography</i>	736
18. EMERGING DATABASE TECHNOLOGIES	739
18.1 Introduction	740
18.2 Cloud Computing and Data Management	743
18.3 Mobile Databases	749
18.4 Dealing with Massive Datasets—MapReduce and Hadoop	753
<i>Summary</i>	757
<i>Review Questions</i>	758
<i>Short Answer Questions</i>	759
<i>Multiple Choice Questions</i>	759
<i>Exercises</i>	761
<i>Projects</i>	762
<i>Lab Exercises</i>	762
<i>Bibliography</i>	763
Appendix A—Answers to Multiple Choice Questions	765
Index	768

Preface

Database technology occupies a very important position in our increasingly computerized society. Almost all computer systems whether they are supermarket checkout systems, air traffic control systems, railways seat reservation systems, planning systems in governments or automatic teller machines (ATMs) with which most of us interact in our daily lives in urban India, have a database system at their core. Therefore, all computer science, information technology, business administration and information science students need to learn about the basic theory and practice of database systems since it is fundamental to much of computer technology.

This book is intended to be a comprehensive text for undergraduate and graduate level database management courses offered in computer science, information systems and business management disciplines. It can be used as a primary read through one-semester by students pursuing BCA, MCA, BSc, MSc and related computer science courses. It can be used as a reference reading by BTech, BE and MTech students and industry professionals.

This book is based on many years of experience of teaching database courses at a number of universities including James Cook University, Monash University and Bond University in Australia, Asian Institute of Technology (AIT) in Thailand, International University in Germany, and VIT University in India. Writing this book has been a wonderful experience for me.

What makes this book unique?

The book differs from other books on database management since it is written especially for students in India and that is why the examples used in the book are primarily those that I hope will interest Indian students. One major example used to illustrate most database concepts that is carried throughout the book is based on cricket One Day International (ODI) matches. Some other real-life examples have also been included. These examples have been used to illustrate several concepts in database design, query processing, distributed databases, object-oriented databases and a number of other topics.

Instead of focusing on how to build the database management system software, this textbook focuses on how to build database applications since most students once they graduate will be developing database applications rather than building database software. Therefore in this book, we concentrate on the concepts underlying the relational data model rather than on any particular commercial DBMS, as these concepts are the foundation of database processing. In a similar way, in the portion of the book which studies transaction processing, we concentrate on the concepts underlying the properties of transactions and the technical issues involved in their implementation, rather than how they are implemented in any particular commercial DBMS.

This text has a logical flow of topics and is supplemented with numerous chapter-end pedagogical features, including multiple-choice questions with answers (409), review questions with section references for answers (342), exercises (246), and short-answer questions (348). Projects (43) and lab exercises (56) have been provided to enhance practical learning along with several examples and real-life case studies for holistic understanding of the subject. Important definitions and key terms have been emphasized in every chapter for a good grasp of concepts.

Book Overview

The contents of the book are described ahead briefly.

Chapter 1 introduces the subject with a simple example of a cricket database and its features followed by a definition of the term ‘database’. A number of popular websites are listed to illustrate the amount of information that many websites now have and is required to be stored in databases. **Chapter 2** discusses the Entity-Relationship model in detail. Representing an Entity-Relationship model by an Entity-Relationship diagram is explained along with generalization, specialization and aggregation concepts. **Chapter 3** describes the relational model and its three major components, viz., data structure, data manipulation and data integrity. Mapping of the E-R model to the relational model is explained. This is supported by a brief description of older database models: hierarchical and network models. **Chapter 4** elucidates data manipulation in relational model including relational algebra and relational calculus. A number of examples of their use are also presented. **Chapter 5** on SQL discusses data retrieval features of SQL using simple examples that involve only one table, followed by examples that involve more than one table using joins and subqueries, as well as built-in functions and GROUPING and HAVING clauses. Use of SQL in enforcing data integrity is also mentioned. **Chapter 6** explains theoretical basis of normalization using the concept of functional dependency followed by a study of single-valued normalization and multivalued normalization.

The structure of information storage on a magnetic disk is described along with the most commonly used data structures, B-tree and hashing in **Chapter 7**. The query optimization process is covered in detail in **Chapter 8**. The steps involved in the process, and rules of thumb that are often used in query optimization, estimating the size of intermediate results in processing a query are presented herein. Algorithms for relational algebra operations are then discussed. In **Chapter 9**, the concept and properties of a transaction are described along with concepts of a schedule and serializability. Locking and two-phase locking (2PL) are introduced followed by definition of deadlocks, their detection, prevention and resolution. Database recovery from a failure, the concept of log, and recovery techniques are described in **Chapter 10**. Types of recovery techniques, immediate update technique, deferred update technique, system checkpoints, shadow page technique, and evaluation of recovery techniques are also presented in this chapter. **Chapter 11** elucidates security violations and sources of violations in a database system highlighting various components of database security, discretionary access control and non-discretionary control or mandatory control techniques along with security of statistical databases. **Chapter 12** gives the definition of database integrity, lists two types of integrity: syntactic integrity and semantic integrity, and explains how integrity constraints are implemented in SQL3.

Features of a distributed database are described in **Chapter 13**. Objectives in designing distributed databases and fragmentation are discussed using a number of examples. Issues in distributed query processing and transaction processing are briefly mentioned here. **Chapter 14** defines an object-oriented database as based

primarily on the data structure called the *object*; the concept originated from the concept of class which was proposed in the 1960s. How object-oriented data model is an attempt to build a richer database model is explained in this chapter. **Chapter 15** gives details of type of database for management. It explains data warehouse, including design of a data warehouse and guidelines for implementation. Online analytical processing (or OLAP) is introduced and characteristics of OLAP systems are listed. The data mining process and several data mining techniques are described in **Chapter 16**. Markup languages and semistructured data followed by a description of XML including an example and introduction to XML elements, attributes, namespace, document type definition (DTD), document object model (DOM), XML Schema and XQuery are discussed in **Chapter 17**. XML databases are also discussed in this chapter. **Chapter 18** introduces emerging database technologies and discusses in detail three of these, viz., cloud computing, mobile databases and MapReduce.

To the Instructor

This book deals with the most important topics of database management. The book includes many examples which I believe are essential in getting students motivated and in helping them understand the course content. To further this aim, I have also presented varied revision exercises. At the end of each chapter, I have included a list of references in the book that encompasses some of the most highly cited papers in database management.

To the Student

Database management is an important and interesting area and I hope you will enjoy learning about it. The book is written primarily for you. I have provided many examples to help you learn the concepts and the techniques. A large number of references have been provided, highlighting a small number, specially selected for you since they are easy to read. I understand that most students do not have the time to read references other than the prescribed textbooks and lecture notes but these references should be useful if you decide to learn more about database techniques during your studies or afterwards.

Review, short-answer and multiple-choice questions, and exercises have been provided at the end of every chapter. Please try to solve all of these to better understand the material in the book.

Online Learning Centre

This book is supported by an online learning centre which can be accessed at <http://www.mhhe.com/gupta/dbms>. The centre comprises the following additional study resources for both instructors and students.

For Instructors

- Chapter-wise Power Point Slides
- Solution Manual
- Tutorial Questions with Answers

For Students

- 5 sets of Self-tests with Answers
- Solution for 5 Projects given in the book
- Links for further reading

Acknowledgements

I would like to thank a number of people who have helped in reading, correcting and giving ideas during the writing of this book. In particular Joe Incigneri read an earlier manuscript in 2009 and helped in improving the readability of the book. Jeanette Niehus has helped in drawing all the diagrams using Microsoft Visio.

I also gratefully acknowledge the suggestions given by the reviewers during the initial process. Their comments have added value to parts of this book.

S K Gupta	<i>Indian Institute of Technology Delhi, New Delhi</i>
R C Joshi	<i>Indian Institute of Technology Roorkee, Uttarakhand</i>
S K Jain	<i>National Institute of Technology, Kurukshetra, Haryana</i>
B Biswas	<i>Institute of Technology, Banaras Hindu University, Varanasi, Uttar Pradesh</i>
Amit Kanskar	<i>NRI Institute of Science and Technology, Bhopal, Madhya Pradesh</i>
Naresh K Nagwani	<i>National Institute of Technology, Raipur, Chattisgarh</i>
Ritika Wason	<i>Institute of Information Technology and Management, New Delhi</i>
Manish Mahajan	<i>ABES Institute of Technology, Ghaziabad, Uttar Pradesh</i>
Parteek Bhatia	<i>Thapar University, Patiala, Punjab</i>
Harmeet Malhotra	<i>Institute of Information Technology and Management, New Delhi</i>
Itu Snigdha	<i>Birla Institute of Technology, Mesra, Jharkhand</i>
Ranjan Mohanty	<i>Kalinga Institute of Industry and Technology, Bhubaneswar, Orissa</i>
P Paul	<i>Birla Institute of Technology, Mesra, Jharkhand</i>
Sarda Nandlal	<i>Indian Institute of Technology Bombay, Maharashtra</i>
V B Gadekar	<i>Vishwakarma Institute of Technology, Pune, Maharashtra</i>
K Sudha Sadhasivam	<i>PSG College of Technology, Coimbatore, Tamil Nadu</i>
N Jaisankar	<i>Vellore Institute of Technology, Vellore, Tamil Nadu</i>
S Chandaradekaran	<i>Madurai Kamaraj University, Madurai, Tamil Nadu</i>
Abdul Nazeer K A	<i>National Institute of Technology, Calicut, Kerala</i>
Valli Kumari	<i>University College of Engineering, Andhra University, Visakhapatnam, Andhra Pradesh</i>
Ravi Thorat	<i>Wesley PG College, Secunderabad, Andhra Pradesh</i>
M Palanivel	<i>Sreenivasa Institute of Technology, Andhra Pradesh</i>
Harish Yagneshwar	<i>Accenture India Ltd, Hyderabad, Andhra Pradesh</i>

It is always a pleasure to receive feedback from students and I am thankful to Shikha Kansal, Hemant Sharma, Abhinav Kansal, Toby L Thomas, Vivek Mittal, Varun Sharma, Abhimanyu, Divya Gupta, Richa Goyal, Deepti Goyal, Ritika Mittal, Amit Singh, Kaushal Kumar, Tarun Kumar Singh, Bijendra Tyagi, Rahul Rathore, Prerna Sharma, Nidhi Sehgal, and Anubhav Jain from Institute of Information Technology and Management, New Delhi and ABES Institute of Technology, Ghaziabad for their valuable feedback on the script.

Finally I would like to thank the editorial staff at Tata McGraw-Hill. I am grateful to Ms Surabhi Shukla for her assistance and support to develop this book during her time of service at Tata McGraw-Hill. I know it has not been easy for her to deal over a year with an author who is thousands of miles away.

I should be grateful if I could be informed of any errors discovered and any suggestions you might have about how the book could be improved for the next edition. You can write to me at gkgupta@acm.org.

G K Gupta

Publisher's Note

Have something to tell us? Please feel free to write to us with your comments at tmh.csefeedback@gmail.com. Please mention the title and author's name as the subject. We look forward to hearing from you.

visual guide

Example 1.1

Consider the table presented in Table 1.1 that presents a list of the top ten batsmen in one day international (ODI) cricket in terms of the total number of runs scored. The data was current on August 2010.

Table 1.1 Example of a simple database¹

Player	Span	Matches	Innings	Runs	Avg.	Strike Rate	100s
SR Tendulkar	1989–2010	442	471	17596	45.42	89.26	46
ST Jayasuriya	1989–2010	444	432	13428	32.43	91.22	28
Ricky Ponting	1995–2010	351	342	13672	32.85	80.84	29
Izhamainul-Haq	1991–2007	378	350	11739	39.52	74.24	19
SC Ganguly	1992–2007	311	300	11363	41.02	73.70	22
JH Kallis	1996–2010	303	289	10838	43.72	72.72	17
R. Dravid	1996–2009	339	313	10785	39.43	73.17	32
BC Lara	1990–2007	299	289	10405	40.48	79.51	19
Mohammed Yousuf	1996–2010	282	267	9624	42.39	75.15	15
AC Gilchrist	1996–2008	287	279	9619	35.89	96.94	16

It should be noted that in the above database the batting average for a player is not obtained by dividing the total number of runs scored by the total number of innings played. It is obtained by dividing the total runs scored by the total number of innings in which the player got out. The number of times the player was not out is not counted and the number of times a batsman was not out is not included in the table above because of space considerations. For example, Sachin Tendulkar was not out 38 times while Adam Gilchrist was not out only 11 times.

The information in Table 1.1 is quite interesting. There are three players at the top ten ODI batsmen order from India and none from England. Also the number of runs scored and the number of centuries made by Sachin Tendulkar are so high as compared to other players that it is unlikely he would be overtaken by another batsman, for example, Sanath Jayasuriya or Ricky Ponting, in the near future. Also, let us not forget that Tendulkar is an active cricketer who continues to score runs.

2.5 THE ENTITY-RELATIONSHIP MODEL

The Entity-Relationship model (E-R model) is a widely used database modeling technique because it is simple and relatively easy to understand. The method facilitates communication between the database designer and the end user. It enables an abstract global view (that is, the *enterprise conceptual schema*) of the enterprise application to be developed without any consideration of efficiency of the ultimate database. The Entity-Relationship model was originally proposed in 1976 by Peter Chen although many variants of the technique have subsequently been devised.

E-R modeling is based on the concept that organizations consist of people, facilities and objects. These components of an enterprise interact with each other to achieve certain goals of the enterprise. As an example, when dealing with an enterprise that deals with cricket, the batsmen interact with bowlers, fielders and umpires to achieve the aim of playing a match. Another example is to look at a book publishing business which involves books, authors, printers, distributors, editors, sales data and so on.

All such things that are of interest to the enterprise are called entities and their interactions are called relationships. To develop a database model based on the E-R model involves identifying the entities and relationships of interest. These are often displayed in an E-R diagram. Before discussing these, we need to describe an entity and a relationship.

2.5.1 Entities and Entity Sets

An entity is a real or abstract object. It may be a person, a place, a building, or an event like a contract or a concert or an order that can be distinctly identified and is of interest. An entity exists independent of what does or does not exist in an enterprise. A specific cricket player, for example, Sachin Tendulkar with player ID 89061 or a venue Eden Gardens with a venue ID 65478 or a country India are examples of entities assuming that the database application requires information about them. Although entities are often concrete like a company, an entity may be abstract like an idea, concept or an event, for example, an ODI match 2688 between India and Australia or a Delhi to Mumbai flight number 1A123. Whether an entity is real or abstract, the model makes no distinction between them but they must be something about which information is important.

In-depth coverage of E-R Model, Relational Model, Transaction Management and Concurrency, Query Processing, Backup and Recovery

An example based on Cricket ODI matches to illustrate database concepts, is used throughout the book, to especially connect with Indian students

The chapter is organized as follows. We start by describing cloud computing in Section 18.2. This section includes discussion of characteristics of cloud computing, benefits of cloud computing, cloud computing architecture, some cloud computing applications, cloud computing economics, and security concerns in cloud computing. Section 18.3 is devoted to mobile databases. This section deals with characteristics of mobile devices, challenges of mobile databases and locality-based services. Section 18.4 deals with MapReduce that is designed to handle huge amount of data that is usually distributed over 100s or 1000s of commodity computers.

18.2 CLOUD COMPUTING AND DATA MANAGEMENT

Cloud computing was identified as one of the emerging technologies in the Clarendon Meeting.

What is cloud computing and why is there so much talk about it? In a recent interview, the new Infosys CEO Mr Gopal Krishnan identified cloud computing as one of the technologies on which Infosys is focusing.

Cloud computing is a concept we all are already familiar with since we all use emails from Hotmail, Gmail, Rediffmail, Yahoo! and so on. By using these email systems we access, store and send emails using someone else's computer and memory installed somewhere else. Gmail and Hotmail now claim to provide several gigabytes of memory to each email user for free. Therefore, we are using cloud computing! Somewhere out there someone is providing computing and storage to use email. The comparing is in "The clouds" or we may say it is providing cloud computing for free.

Although email was first used in the 1970s as computer network became more widely used. Few people knew about email then but then networks became ubiquitous and the Web was invented. This led to the invention of Hotmail by Sabe Bhuria in California and that meant that we did not have to worry about our own computers managing the email traffic. Someone else's machine could do it all. We only had to connect to the Hotmail website and create a Hotmail account and use it from anywhere in the world. Currently we have Hotmail, Yahoo!, Gmail and Rediffmail which altogether have almost one billion email accounts. From 2010, Monash University is moving all student emails to the clouds using Gmail and the university will stop worrying about some 50,000 student email accounts. The staff will also move to Gmail. It makes a lot of economic sense to do that.

In addition to email, the most popular cloud services include social networking sites (about 500 million people that use Facebook are being social in the cloud), microblogging and blogging services such as Twitter and WordPress, video-sharing sites like YouTube, photo-sharing sites such as Flickr, document and applications sharing sites like Google Docs, social-bookmarking sites like Delicious, business sites like eBay, and ranking, rating and commenting sites such as Yelp and TripAdvisor.

There are many definitions of cloud computing. We present an informal definition.

Definition—Cloud Computing

Cloud computing is computing services offered on the Internet that may provide computing resources such as processing, storage and software that are normally not physically present at the user's location and the user pays only for the computing resources and services used.

A discussion of new technologies like Cloud Computing and Mobile Databases

- Discuss the responsibilities of a DBA. Explain what would happen if a DBA was not responsible for an enterprise database system? Why does the DBA need to understand query optimization? Explain the role of a DBA in a simple database like the ODI database if it is being used by many people.
- Could a modern university be supported if a file processing system was used to manage enterprise information? List three major difficulties that are likely to arise if no database system was available to the university. Discuss the difficulties.
- Study the role of data governance by looking at some recent literature on data governance. What benefits can be obtained by an enterprise using data governance which they cannot get with current database systems? Discuss.
- Select an enterprise that you are familiar with (for example, a school, college, company, small business, club or association). List all the information that this enterprise uses.
- Give examples to illustrate three major advantages and three areas of concern when using an enterprise database system. Discuss their importance.
- List five queries you might wish to pose to the ODI database given in Tables 1.1, 1.2 and 1.3.
- Why is data independence important? Discuss its importance by giving an example.
- Present a conceptual view of a student database. Present some external views given by different types of end users and show the mappings of these views to the conceptual view.

PROJECTS

- If you have access to the Indian Railways Timetable, describe how you would computerize it and how much memory space the computerized timetable would take? What other information would be useful in the timetable?
- Select a local company large enough to have a lot of information about its operations. Arrange to meet someone in that company who can tell you about the information they have and the ways they manage and use it. Write a brief report on the same.

LAB EXERCISES

- Northern India University (NIU) is a large institution with several campuses scattered across north India. Academically, the university is divided into a number of faculties, such as Faculty of Arts and Faculty of Science. Some faculties operate on a number of campuses. Faculties, in turn, are divided into schools. For example, the School of Physics, the School of Information Technology, etc. The schools are not divided further into departments.

Each student in the University is enrolled in a single course of study which involves a fixed core of subjects specific to that course as well as a number of electives taken from other courses. Each course is offered by one particular school. Each school has a number of lecturing staff. Students are awarded a grade for each subject they chose.

List all the information that you need to store in a database for NIU.

Focuses on building database applications to aid professional learning

```
SELECT [DISTINCT] ALL <List of Columns, Functions, Constants, etc>
FROM <List of Tables or Views>
[WHERE <Conditions>]
[GROUP BY <Grouping Column(s)>]
[HAVING <Condition>]
[ORDER BY <Ordering Column(s)> [ASC|DESC]]
```

```
UPDATE <Table Name>
SET <Column Name> = <Value>
WHERE <Condition>;
INSERT INTO <Table Name> [<Column List>]
VALUES (<Value List>);
DELETE FROM <Table Name> WHERE <Condition>;
CREATE VIEW <View Name> AS <Query>;
CREATE TABLE <Table Name>
<Column Name> <Data Type> [<Size>] <Column Constraint>, ... Other Columns;
```

Tables may have a variety of constraints as listed below. These are discussed in detail in Chapter 11 on Integrity.

NULL or NOT NULL
UNIQUE
PRIMARY KEY
CHECK
DEFAULT
FOREIGN KEY
UNIQUE
TRIGGER
ASSERTION

Pedagogy featuring Projects and Lab Exercises besides Multiple Choice Questions for holistic study

Peter P. S. Chen



Peter Chen is the creator of the Entity-Relationship model (E-R model), which serves as the foundation of many systems analysis, design methodologies, and computer-aided software engineering (CASE) tools. His first paper on the E-R model was published in 1976 and was included in the '38 most influential papers in Computer Science' by a survey of over 1,000 computer scientists.

Currently, Professor Chen is a distinguished Chair Professor at NTNU, Taiwan. He held the position of M. J. Foster distinguished Chair Professor of Computer Science at Louisiana State University in the United States from 1983–2008.

Dr. Chen holds a PhD in Computer Science/Applied Mathematics from Harvard University.

Profiles of eminent scholars and professionals in the field of database management



Introduction to Database Management

1

OBJECTIVES

- Introduce the importance of information for every enterprise.
- Discuss and define the concept of a database.
- Describe data governance and why databases are becoming so important.
- Briefly describe the evolution of database software.
- Describe differences between file systems and database systems.
- Explain database management systems (DBMS). The advantages and disadvantages of using an enterprise database.
- Discuss the three-level database architecture and data abstraction.
- Describe the different types of database users.
- Explain the concept of data independence.
- Present a DBMS system architecture.

KEYWORDS

Database, database management, database management systems, DBMS, data independence, data abstraction, metadata, file systems, advantages and disadvantages of using a database system, three-tier architecture, DBMS architecture and system architecture, end users, catalog, concurrency, transaction management, recovery, applications programmer, web databases, database history, database administrator, database design, requirements, testing, maintaining, importance of databases, characteristics of databases, selecting DBMS software, application programs, systems analysts, application programmers, cricket database, consistency.

The more this power of concentration, the more knowledge is acquired, because this is the one and only method of acquiring knowledge... in doing anything, the stronger the power of concentration, the better will that thing be done. This is the one call, the one knock, which opens the gate of nature, and lets out the flood of light.

Swami Vivekananda (1863–1902)

1.1 INTRODUCTION

Most modern enterprises except some small ones have data that they need to store in ways which will be easy to retrieve later. One of the most common types of database is a list of names and addresses of people who regularly deal with the enterprise. Originally when a business is small such lists may have been written on paper, then in a word processor or spreadsheet, but as the business and lists grow it becomes difficult to deal with the information stored in a word processor or a spreadsheet. Database technology may then be required.

Database management is about managing information resources of an enterprise. The enterprise may be a big company, a small business, a government department or a club; each of them generate a variety of data. The data can be, for example, about customers, credit cards, telephone calls, lottery tickets, orders, library books, students, employees, airline flights, etc. All such data (or information, we won't discuss the difference between the two) for every type of enterprise is a valuable resource and therefore, just like any other valuable resource, it needs to be managed effectively. With the advent of the World Wide Web and its widespread use in business, the amount of data that is being generated is growing exponentially, making data management even more important. Imagine the data being generated and stored by companies like amazon.com, eBay, Google, YouTube and Wikipedia that needs to be managed. On the other hand, consider the amount of data being generated by mobile phones in India. A lot of data must be stored for accounting purposes (information about every call and each sms must be stored for accounting) while other data, like the content of various sms messages, must also be stored for retrieval incase of security checks or more serious criminal proceedings. Thus, the amount of data being generated in the modern world using new technologies is mind boggling.

This chapter is organized as follows. A simple example of a cricket database is presented in the next section and a number of features of this database are discussed. In Section 1.3, the term database is defined and examples related to cricket are presented. An example of a university database is also presented. A number of popular websites are listed to show the amount of information that many websites now carry and which must be stored in databases. Section 1.4 describes a number of characteristics of databases and Section 1.5 discusses data governance and importance of databases for enterprises. Section 1.6 presents a brief history of database software.

Section 1.7 discusses the differences between file systems and database systems. A definition of database management system (DBMS) is presented along with some features of a DBMS in Section 1.8. Variety of users of database systems is described in Section 1.9. The next two sections present a number of advantages and concerns of using a centralized enterprise database system in a large enterprise as compared to the branches maintaining local database systems using conventional file systems. Sections 1.12 and 1.13 are devoted to explaining the three-level DBMS architecture, data abstraction and data independence. A typical DBMS architecture diagram is presented in Section 1.14 along with a description of its different components.

The role of the database administrator is discussed in Section 1.15. Section 1.16 explains the steps involved in designing an enterprise database system. The chapter concludes with a brief introduction to web databases followed by a number of guidelines for selecting a DBMS before making a purchase.

1.2 AN EXAMPLE OF A DATABASE

A simple example of a database is presented below.

Example 1.1

Consider the table presented in Table 1.1 that presents a list of the top ten batsmen in one day international (ODI) cricket in terms of the total number of runs scored. The data was current on 21 August 2010.

Table 1.1 Example of a simple database¹

Player	Span	Matches	Innings	Runs	Avg.	Strike Rate	100s
SR Tendulkar	1989–2010	442	431	17598	45.12	86.26	46
ST Jayasuriya	1989–2010	444	432	13428	32.43	91.22	28
RT Ponting	1995–2010	351	342	13072	42.85	80.84	29
Inzamam-ul-Haq	1991–2007	378	350	11739	39.52	74.24	10
SC Ganguly	1992–2007	311	300	11363	41.02	73.70	22
JH Kallis	1996–2010	303	289	10838	45.72	72.72	17
R Dravid	1996–2009	339	313	10765	39.43	71.17	12
BC Lara	1990–2007	299	289	10405	40.48	79.51	19
Mohammed Yousuf	1998–2010	282	267	9624	42.39	75.15	15
AC Gilchrist	1996–2008	287	279	9619	35.89	96.94	16

It should be noted that in the above database the batting average for a player is not obtained by dividing the total number of runs scored by the total number of innings played. It is obtained by dividing the total runs scored by the total number of innings in which the player got out. The number of times the player was not out is not counted and the number of times a batsman was not out is not included in the table above because of space considerations. For example, Sachin Tendulkar was not out 38 times while Adam Gilchrist was not out only 11 times.

The information in Table 1.1 is quite interesting. There are three players in the top ten ODI batsmen order from India and none from England. Also the number of runs scored and the number of centuries made by Sachin Tendulkar are so high as compared to other players that it is unlikely he would be overtaken by another batsman, for example, Sanath Jayasuriya or Ricky Ponting, in the near future. Also, let us not forget that Tendulkar is an active cricketer who continues to score runs.

1. Full names of all Pakistani players have been used, since they are widely known by their full names.

There are some other things we should note about the database:

1. Each table in the database requires a name so that it can be easily identified by that name. We have named our table the *bestODIBatsmen*².
 2. The names given to columns in the top row are of special importance in a database. They are the titles of the columns and each column is referred to by that name. If we wish to state the number of ODI matches played by a player, it will be referred to by the name *Matches*. The name *matches* with a lower case *m* will not be acceptable and other names like *Games* will also not work.
 3. If some data has been typed incorrectly (and errors in databases are not uncommon), for example, if the name Tendulkar in the table above is mistyped as Tendulker, then there is no simple way to retrieve information about Tendulkar. Although any person interested in cricket would know that Tendulker is really Tendulkar, the computerized database has no intelligence.
 4. The second column with the name *Span* gives entries like 1975–85 as the year the player started playing ODI cricket and the last year he played. This data is treated as one value and usually there is no simple way to separate the two dates and query the database on them separately.
 5. The database above is ordered according to the total runs scored. Although such ordering is helpful when humans view the database, the ordering of data when stored in a computer is often not known and may change over time. For example, over time Ricky Ponting and Jacques Kallis are likely to make more runs and would be expected to climb up the ladder of best batsmen. Although unlikely, given the lead Sachin Tendulkar has, but Ricky Ponting in theory could take over as the highest scorer of ODI runs. In a computerized database, it is inefficient to be swapping rows of the database to maintain the desired order, when the computer can be used to sort the table, whenever the user needs it sorted on any column. For example, some users may want the table sorted on the batting average while others on the strike rate. As a corollary, there is no concept of the first row or the last row in a computerized database.
 6. In the example database, a number of columns are related. For example, batting average is based on the total runs scored and the number of innings played (and the number of innings in which the player was not out), which is itself related to the number of matches played. Such relationships in a database are not desirable, since a user may by mistake increase the number of matches when a player starts a new match, and then forget to increase the number of innings, or the total number of runs scored, or the batting average. The data therefore from time to time could be inaccurate. This does not matter so much for our simple cricket database but it would not be acceptable, say, in a bank accounting database. There are ways of overcoming such problems which are discussed later.
 7. A related problem with Table 1.1 is that all the information in it is an aggregation derived from information about each ODI. The example database in Table 1.1 has been used because the information is interesting for people who love cricket but it is not a very good idea to maintain tables of aggregates, since it is difficult to check the accuracy of the information, and it is easy to make a mistake in updating such summary information. If we store information about each ODI in the database then the aggregation can be carried out by the system ensuring there are no mistakes.
-
2. Once a table has a name, it may be written in a compact form by its schema. For example, Table 1.1 may be written as *bestODIBatsmen(Player, Span, Matches, Innings, Runs, Avg, Strike Rate, 100s)*. This notation has also been used later.

8. Furthermore, there are some, perhaps not so obvious, problems with our example database. Firstly, there is no convenient way to show that a player is still playing. For example, we have put 1989–2010 in *Span* for Sachin Tendulkar. This does not tell us that he is still playing. Although Adam Gilchrist played during 1996–2008, he has retired and therefore is not playing any more. Furthermore, the database assumes that a player always plays for only one country. We leave it to the reader to discover if there have been any players for which this is not true.

These issues and a variety of other issues related to computerized databases are discussed further in the following chapters.

1.3 WHAT IS A DATABASE?

Whether it is a cricket database or an airline booking database, databases have become essential in all aspects of modern life. It is often said that we live in an information society and that information is a very valuable resource (or, in other words, information is power). In this information society, the term *database* has become a rather common term although its meaning appears to have become somewhat vague, as the importance of database systems has grown. Some people use the term database of an organization to mean all the data in the organization (whether computerized or not). Other people use the term to mean the software that manages the data. We will use it to mean a collection of computerized information such that it is available to many people for various uses. A definition of a database may be given as.

A database is a well organized collection of data that are related in a meaningful way which can be accessed in different logical orders but are stored only once. The data in the database is therefore integrated, structured and shared.

According to this definition, the simple database given in the cricket example in Table 1.1 is not really a database. We may consider it a *toy database* and will continue to keep calling it a database and use it when appropriate. Similarly a *database* that someone might have for personal use on his/her home PC would not be considered a database. Also, a collection of tables that are not related to each other, for example, a table of cricket players and a table of swimmers, unless they can be related through the country they come from or in some other way, cannot be considered a database.

The main features of data in a database are as follows:

- It is well organized.
- It is related.
- It is accessible in different orders without great difficulty.
- It is stored only once.

It is assumed that operations (for example, update, insert and retrieve) on the database can be carried out in a simple and flexible way. Also, since a database tends to be a long term resource of an organization, it is expected that planned as well as unplanned applications can (in general) be carried out without great difficulty.

Example 1.2

We have noted above that a single table like Table 1.1 is not really a database. Now let us add another table. This table is also a summary of performance of a number of players in ODI matches. Consider Table 1.2 given below which gives information about the best bowlers in ODI matches based on the total number of wickets taken before 21 August 2010. Information in this table is also quite interesting. There is still no player from the UK and only one player appears in both Tables 1.1 and 1.2. Only two bowlers from India and three from Sri Lanka appear in Table 1.2.

Table 1.2 The best ODI bowlers (Table Name: *bestbowlers*)

Player	Span	Matches	Balls	Runs	Wickets	Ave	Econ	ST
M Muralitharan	1993–2010	337	18169	11885	515	23.07	3.92	35.2
Wasim Akram	1984–2003	356	18186	11812	502	23.52	3.89	36.2
Waqar Younis	1989–2003	262	12698	9919	416	23.84	4.68	30.5
WPUJC Vaas	1994–2008	322	15775	11014	400	27.53	39.4	39.2
SM Pollock	1996–2008	303	15712	9631	393	24.50	3.67	39.9
GD McGrath	1993–2007	250	12970	8391	381	22.02	3.88	34.0
A Kumble	1990–2007	271	14496	10412	337	30.89	4.30	43.0
J Srinath	1991–2003	229	11935	8847	315	28.08	4.44	37.8
B Lee	2000–2009	185	9442	7428	323	22.99	4.72	29.2
ST Jayasuriya	1989–2009	444	14838	11825	322	36.72	4.78	46.0

Now let us ask the question: Do the two tables, Table 1.1 and Table 1.2, together form a database? Although the information is well organized, the information in the two tables is difficult to relate. We do have some common data, for example, the names of those players that appear in both the tables (ST Jayasuriya is the only one) but the relationship is not strong. So it may be concluded that these two tables also do not form a database.

Example 1.3

Let us now consider a third table. Table 1.3, called *Players*, includes information about all the players given in Table 1.1 and 1.2. As we have noted earlier, there is no concept of sorting of rows in a real database. Although the information in Table 1.3 has been sorted on the column Country but it could easily be sorted according to any other column. Also, there are no duplicates in the table.

Now let us again ask the same question as we did earlier. Do the three tables, Tables 1.1, 1.2 and 1.3 together form a database? By including this table, the relationship between the three tables becomes stronger. We can now find out which players are best ODI batsmen and come from India, which could not be done before. It is also now possible, for example, to prepare a list of all Indian players who are either best ODI batsmen or best ODI bowlers. We can therefore conclude that the three tables together could be called a database since they have the properties of a database and assuming the tables are being shared.

Table 1.3 List of some players that have played in ODIs (Table Name: *Players*)

Name	Country	Place of birth	Year of birth
RT Ponting	Australia	Launceston	1974
AC Gilchrist	Australia	Bellingen	1971
GD McGrath	Australia	Dubbo	1970
B Lee	Australia	Wollongong	1976
SR Tendulkar	India	Mumbai	1973
SC Ganguly	India	Calcutta	1972
R Dravid	India	Indore	1973
A Kumble	India	Bangalore	1970
J Srinath	India	Mysore	1969
Inzamam-ul-Haq	Pakistan	Multan	1970
Wasim Akram	Pakistan	Lahore	1966
Waqar Younis	Pakistan	Vehari	1969
Mohammed Yousuf	Pakistan	Lahore	1974
JH Kallis	South Africa	Cape Town	1975
SM Pollock	South Africa	Port Elizabeth	1973
ST Jayasuriya	Sri Lanka	Matara	1969
WPUJC Vaas	Sri Lanka	Muttumagala	1974
M Muralitharan	Sri Lanka	Kandy	1972
BC Lara	West Indies	Santa Cruz	1969

We have included limited information in the example tables because of space considerations. For example, we could include the number of times a batsman was not out, each batsman's highest score, each bowler's best bowling performance, each player's date of first ODI played and so on.

Just like in cricket, in every modern enterprise, a large amount of data is generated about its operations. This data is sometimes called *operational data*. The operational data includes the data an organization must necessarily maintain about its operation but does not include temporary results or any transient information.

Since data is a valuable resource for every enterprise, often a great deal of money is spent collecting, storing, maintaining and using it. The running of a modern enterprise depends on proper maintenance of its operational data.

A Case Study

The case study deals with the operational data of an enterprise; the enterprise being a university. A university's operational data includes information about courses, students, academic staff, other staff, and enrolments. There is much more information of course, for example, departments, library data, financial data as well as data about research and about university facilities.

8 | Database Management Systems

The information may include the following:

1. Student personal data (studentID, student's name, gender, current address, home address, date of birth, nationality)
2. Student academic data (studentID, student's school results, courses completed and grades, current enrolment)
3. Academic staff data (staff ID, staff member's name, gender, current address, date of birth, nationality, academic qualifications, appointment history, current appointment, salary history, current salary, sabbatical leave information, recreational leave, sick leave)
4. Research data (projectID, researcherID, funding, publications, research students)
5. Non-academic staff data (staffID, staff member's name, gender, current address, date of birth, nationality, qualifications, experience, appointment history, current appointment, salary history, current salary, recreational leave, sick leave)
6. Courses (or units) offered data (courseID, course name, department, syllabus, lecturer, quota if any)
7. Financial data (budget information, receipts, expenditure)

The above data is only a very small sample of data that a university generates (for example, data generated by the library is not included and data about facilities, for example, buildings and equipment is not included) and we leave it to the reader to think of other information that should be included for each of the items listed above. In fact, in the early 1990s, the Australian universities initiated a project to develop a unified database model for all student information that could be used by all Australian universities. After months of effort by a team of experts and spending millions of dollars, the project was cancelled in spite of the team having developed the specifications of the information which was more than 1000-pages long. The difficulty was that not all the universities agreed with the collection of information that was developed.

The above data would have a number of classes of users, for example:

1. Instructors who need information about enrolments in the courses that they teach and heads of departments about finances of their departments. These are users that use the information in the database but do not develop their own software. They are often called *end users*.
2. Programmers in the IT services unit who develop programs (called *application programs*) to produce reports that are needed regularly by the government, university administration and the departments. These programmers are called *applications programmers*. They are also responsible for developing programs that assist end users in retrieving the information that they need in their work.
3. The Registrar or some other person is in-charge of the database and makes decisions about the kind of information that is to be stored in the database, and also decides who can modify and access which parts of the database.

For a large modern enterprise, it is difficult to overestimate the value of its databases. From time to time, we read stories in the press describing enterprises that were not able to handle their data effectively. In fact, in 2002, there was a case of an Australian university using the wrong database software to process its data which resulted in a loss of millions of dollars. Some interesting stories of database and software development failures are given in Glass (1997, 2001).

To conclude, we note that data

- is a valuable resource and an investment and
- should be carefully managed like other resources (for example, manpower).

Example 1.4

Very Large Databases

The example given below illustrates the amount of information currently being circulated on the Web and its fast paced growth.

Computing has changed enormously since the early days of very large and very expensive computers. This is partly because of technological developments in computer hardware and software and partly because of the development of the World Wide Web. Some of these trends can be illustrated by the popularity of the following websites:

1. Google, Yahoo and MSN—Every day, millions of people use these three websites that enable people to search the Web, use chatrooms, free email and a number of other services.
2. Ebay—A site that enables people to sell and buy new and used items by way of a person-to-person auction.
3. YouTube—Every day, millions of people watch this website that enables them to put and share videos on the Web.
4. MySpace—Every day a large number of people visit this website that facilitates social networking.
5. Facebook—It is another social networking website that connects people through social networks and encourages online interaction through groups and communities.
6. Wikipedia—A free online collaborative encyclopaedia that includes a huge amount of information on almost every topic.
7. Skype—This is a nearly free person-to-person telecommunications application that uses the Internet for making phone calls.
8. Flickr—A large number of people use this website to share photos.
9. Blogger—Free personal journal publishing tool that allows people to publish their own blogs and share them with others.
10. Fotolog—A website that enables people to share photos and blogs.

The reader is encouraged to consider the size of these sites and find out how much information each of these sites carry. Most of these sites are growing rapidly and have millions of users. For example, Facebook is reported to have more than 500 million users in September 2010³ (less than six years after it was started) and there are 30 billion pieces of content shared each month. MySpace, also of a similar size, had 100 billion rows of data, 14 billion comments on the site, 20 billion mails, 50 million mails per day (more than Yahoo, Hotmail or Google), 10 billion friend relationships, 1.5 billion images, and 8 million images being uploaded per day as per data published in 2008.

3. Refer to the statistics posted at

<http://www.facebook.com/press/info.php?statistic>

Also look at <http://blog.compete.com/2009/02/09/facebook-myspace-twitter-social-network>

How would one manage such a huge amount of information?

In addition to these trends, there are a number of other new trends in computing. For example, the popularity of devices like iPod and iPhone, the digitization of millions of old books, the development of Internet TV and the availability of wireless broadband. All these developments deal with processing, storing and communicating information. Although the computer was developed to solve numerical problems, it has increasingly become an information-handling device. However, the conceptual basis of the computer architecture has not changed so significantly.

1.4 CHARACTERISTICS OF DATABASES

Various characteristics distinguish the database approach from conventional file processing systems. Some of these differences are discussed in Section 1.7. In this section a list of various characteristics of database systems is given.

1.4.1 Persistent Data

A database only stores persistent data and once data has been stored in a database system it always stays there unless deleted by an explicit request.

1.4.2 Metadata and Self-describing Nature

A database system contains not only the database itself but also the descriptions of the data structures and its constraints. This is called *metadata* which may be defined as *data about data*. In the context of a database, metadata includes information about how the data is stored in the computer storage. Metadata, discussed in more detail later, is an integral part of the database and stored in the database itself. It is used not only by database managers but also by database end-users. This separation makes database system very different from traditional file-based systems in which data definition was a part of the application programs.

1.4.3 Data Independence

Perhaps the most important characteristic of a database system is data independence which may be defined as the ability to separate the database definition from the physical storage organization of the database. Therefore, information about the physical structure of the database is stored in the system catalog (the database metadata) and not in the application programs. Therefore any changes in the physical structure of the database do not impact the application programs. Data independence is discussed in more detail in Section 1.13.

1.4.4 Access Flexibility and Security

In modern database systems no restrictions are placed on the content the users can access unless the user is not authorized to access some data. Security is enhanced by using the concept of database views which

are subsets of the database. Different views may be defined for different classes of users thereby controlling access to the database system. Each view might contain only the data relevant for a user or a group of users.

Most modern database systems, other than those used for personal use, are multiuser systems that allow multiple users to access the system at the same time. Providing this facility to the users requires concurrency control to ensure that several users accessing the same data item at the same time do so without violating the database consistency.

1.5 DATA GOVERNANCE AND IMPORTANCE OF DATABASES

An enterprise has many assets (for example, people, property, intellectual property) but an important asset whose value is often not recognized is the enterprise data. This is however changing in today's competitive world. Every enterprise needs a 'single version of the truth' but it is not always possible to find that version. Many years ago, the company, Coca Cola, could not even be sure of the number of bottles of soft drinks they were producing each day, in all their plants. Therefore, many enterprises are now recognizing the importance of *data governance*. Data governance essentially refers to decision-making regarding effective management of an enterprise's data including data quality, metadata, and data access.

Since the 1960s there has been a continual and rapid growth in the use of database technology which can deal with some aspects of data governance. Databases have become an important part of every company with data that needs to be managed. Not only has there been a dramatic growth in the number of such systems but also enormous growth in the amount of information stored in them. The complexity of applications of database systems has also been growing simultaneously. A report produced by Lyman and Varian at the University of California at Berkeley is a fascinating study on the amount of information in the world and its pace of growth.

Databases contain an enterprise's data or what we call information. As noted earlier, we do not wish to discuss the meanings of data, information and knowledge since there is no need to establish either a hierarchy or a temporal sequence in discussing data and information. It appears information is a term that is more in vogue these days than data. No one talks about the discipline of data processing any more. Instead we are more likely to use the term 'information technology'.

It is said that information is power and we live in an information society. In some sense, every society, whether developing or developed, is an information society although the more complex a society, the more central information is to its activities. Therefore as a country develops, it uses more information. We can easily understand that information has great value. Let us give some examples to illustrate that value:

- The website Google has nothing but information, and its market capitalization is well over US\$100 billion.
- All the websites listed above in Example 1.4 are totally driven by information and they have significant impact on human societies all over the world.
- Warren Buffett, an American investor, has become one of the richest men in the world by just trading information.

There are many other signs of importance and growth of information. The sale of hard disks has been growing exponentially and the price per MByte has been coming down exponentially so much so that a TByte disk in India was selling for below 4,500 INR in January 2011.

Information has therefore become important and an element of commerce. Earlier, success was based on criteria as finance, manufacturing, land, food and so on, but today successful businesses are those that can manage their information. Managing information involves effective collection, organisation, storage, processing and presentation of information. For this reason effective database management and data governance in an enterprise are of great importance.

1.6 HISTORY OF DATABASE SOFTWARE

In the early days of computing, computers were mainly used for solving mathematical problems. Even in solving mathematical problems, it was observed that some tasks were common to many problems that were being solved. It therefore was considered desirable to build special software modules that performed frequently occurring computing tasks such as computing $\text{Sin}(x)$. This led to the development of mathematical software libraries that are now an integral part of any computer system and most users do not even know that such libraries exist.

By the late 1950s, storage, maintenance and retrieval of non-numeric data had become important. Again, it was observed that some data processing tasks (for example, sorting) occurred quite frequently. This led to the development of *generalized* software modules for some of the most common and frequently carried out tasks.

A general module by its nature tends to be somewhat less efficient than a module designed for a specific problem. In the late fifties and early sixties, when hardware costs were high, use of general modules was not popular since it became a matter of trade-off between hardware and software costs. In the last 30–40 years, hardware costs have been going down dramatically while the costs of building software have not. It is therefore no more a trade-off between hardware and software costs since hardware is relatively cheap and building reliable software is expensive. It therefore makes sense to build generalized software that many enterprises can use.

The earliest known use of the term *database* was in 1963 and the first database management systems were developed in the 1960s. Database management systems evolved from generalized modules for file processing (some of them were called SORT/MERGE packages) as the users demanded more extensive and more flexible facilities for managing increasing amounts of data. This development of database management systems allowed effective storage and retrieval from large databases which was not possible before. Data storage media (disks) were still quite expensive as the amount of data which businesses and governments wanted to process continued to grow. During the next 10–20 years a number of DBMS software packages became available. Most of the early DBMS software were vendor specific, for example, IBM developed a system called IMS (Information Management System) in the mid 1960s which is still being used to process data in some legacy systems. Another early DBMS was called IDMS (Integrated Data Management Store). The data models used by these were the network model based on Bachman's ideas, and the hierarchical model used in the IMS database system. Both IMS and IDMS were non-relational systems. More details about the database models used in these systems is given at the end of Chapter 3. There were other database systems that were

released at the end of 1960s. For example, a DBMS called ADABAS was released by a German company, Software AG in 1970. It was initially released for IBM mainframes. It used an inverted list database.

Database technology has undergone major changes since the early database systems of the 1960s and 1970s which were based on the hierarchical and network models that were developed in the 1960s. With the proposal of the relational database model around 1970, research was started for building relational DBMS at IBM and University of California, Berkeley. Most modern systems are now based on the relational database model which is discussed in Chapter 3. An interesting early history of database systems is presented by Fry and Sibley (1976).

1.7 FILE SYSTEMS VS DATABASE SYSTEMS

There are a number of major differences between a file processing system and a database system. A file processing system often uses a relatively simple data structure to store information about one particular thing. For example, a file may be about players consisting of player records including names, date of birth and nationality while another file may be about matches consisting of match records including date, ground and the teams competing. A file may be using file management software, for example, like the widely used Indexed Sequential Access Method (ISAM) but most file processing systems are implemented for a single application. The same data may be repeated in more than one file and the physical storage structure of the same data may be different for different files if the application programs were written by different programmers.

Therefore in most file management systems, every application carries out its own file processing even if two application programs are carrying out very similar tasks. The programs may involve each user defining and using one or more files that he/she needs. For example, in a university environment, the admissions department may have defined its own files and its application programs may be making use of those files while academic departments may have their own file systems with neither of the departments knowing anything about the other departments' files although the files of both the departments are dealing with students. Such independent maintenance and processing of data not only leads to wastage of computer storage, it also leads to redundancy or replication of data which can further lead to data inconsistencies. Furthermore, the files of the two departments cannot have any inter-relationship among the data stored in their files even if the data is related.

In a database system, no application program accesses the files directly since there is a single repository of data that is defined and maintained by the database system and then accessed by all users. The access to the database is handled by the database management system. When an application program needs to access some data then it must call upon the database system to provide it the data that is required. The database system not only manages all the data, but as already noted, it also contains complete definition or description of the database structure and constraints called metadata.

A database system also allows multiple users to access the same database at the same time. Such use requires concurrency control so that several users trying to update the same data at the same time are controlled. Concurrency control is discussed in Chapter 9. This is just not possible in a conventional file system. Moreover to enforce security, virtual tables called views may be created in database systems and users are restricted to only one or more views of the data that they are allowed to see. This is discussed in Chapter 5.

In a traditional file system, if some changes are made in the structure of one or more files, the changes will affect all the application programs that use them. On the other hand, in case of a database system, the structure of the database is stored separately and therefore changes to data structure do not normally lead to changes in the application programs. This property is known as *data independence* and is discussed later in this chapter in Section 1.13.

Another major difference between file systems and database systems is that a database can represent complex relationships among the information that is stored in the database and can therefore retrieve and update related data easily which is not possible in file processing systems. Finally, there are a number of additional characteristics of a database that are not possible in a file system. These include an ability to define and enforce integrity constraints for the data that is stored in the database. A database also usually provides facilities for backup and recovery. These are discussed in Chapter 11.

In a summary, file processing systems may not be very stable over time because file processing tasks may change over time and these changes may in turn require expensive changes to be made to the files. Furthermore, in file systems, there is a possibility of information being duplicated and this can lead to redundancy and wastage of computer storage which in turn may also result in data inconsistencies. A database system on the other hand usually does not suffer from such problems since the database system does not need to be modified if new data processing is needed. A database may be shared by many users and many applications leading to higher productivity. In addition, it provides facilities for concurrency control, security, integrity and recovery.

1.8 WHAT IS A DBMS?

As discussed earlier, a database is a well organized collection of data. Before the database can be used, the database needs to be stored in the computer in some way so that data from the database may be retrieved when required and modified when necessary. Simple tables like the cricket database given above may even be stored in an Excel spreadsheet file and retrieved, viewed and printed when the need arises. However, when a database is large, for example, an airline's booking database; storing the database in a spreadsheet is not practical since the database is too large and needs to allow many users from different parts of the world to access it at the same time. The users should be able to carry out operations like insertion, deletion and retrieval (and others, for example, sorting). To provide such facilities, any large database needs to be managed by a substantial package of software. This software is commonly called a *Database Management System (DBMS)*. The primary purpose of a DBMS is to allow a user to store, update and retrieve data in abstract terms and thus make it easy to maintain and retrieve information from a database. A DBMS relieves the user from having to know about exact physical representations of data and having to specify detailed algorithms for storing, updating and retrieving data. Without a DBMS, a database would be of little use. With a DBMS, a database becomes a powerful tool to support the operations of an enterprise and to understand and analyse the enterprise.

A database management system may be defined as follows:

A Database Management System (DBMS) is a large software package that controls the specification, organization, storage, retrieval and update of data in a database.

A DBMS carries out many different tasks including the provision of various facilities to enable a user to specify and build a database, and to access and modify data once the database has been built. The DBMS is an intermediate link between the physical database, the computer and the operating system, and on the other hand, the users. To provide different facilities to different types of users, a DBMS normally provides one or more specialized programming languages often called *database languages*. Different DBMS may provide different database languages, but in recent times all DBMS are based on the same abstract model of a database, the relational data model. Given that the underlying model of almost all modern database systems is the same, a language called Structured Query Language or SQL has been declared as a de facto standard. The relational model and SQL is discussed in Chapter 3 and 5 respectively.

Let us consider a simple example of SQL, given in Fig. 1.1, for retrieving data from the database of *bestODIBatsmen*.

This query would retrieve the names of the batsmen that are in the table *bestODIBatsmen* (Table 1.1) and have played more than 400 ODI matches (that is, SR Tendulkar and ST Jayasuriya).

```
SELECT Player
FROM bestODIBatsmen
WHERE matches > 400
```

Figure 1.1 Structure of a simple SQL query

SELECT, FROM and WHERE are reserve words in SQL. Some people do not like the keyword SELECT because what is really meant is either find the names or print the names or retrieve the names but it is the keyword used in SQL. The word FROM must always be followed by the name of the table where the information is to be found (there can be more than one table containing the same information, as we will see later) and WHERE is followed by one or more conditions that the information must satisfy.

One simple way of looking at the way the DBMS processes this query is to think of the software looking at every row in the table in turn and selecting only those rows for which the WHERE condition is true. The DBMS then provides only the values from the columns whose names follow the keyword SELECT for the rows that have been selected.

It should be noted that retrieving data from a structured database like that given in Tables 1.1, 1.2 and 1.3 is quite different than retrieving data from the Web using a search engine, since the Web stores data which is not as well structured as a table.

Since a DBMS is a generalized (large) piece of software, every time a new database is set up using the software, the steps given below must be followed:

1. The database must be defined using the DBMS. Although the simple cricket database in Table 1.1 has only one table, any significant database would have much more information and many tables (more than 100 tables are not uncommon). The structure of the information (for example, number of tables and their columns and each column's type) needs to be specified to the DBMS. This logical structure of the database is often called a *schema* of the database that is to be set up. Obviously some language is required to be able to communicate the structure to the DBMS. A language that allows the database to be described to the DBMS as well as provide facilities for changing the database (for example, adding a column to the table *bestbasmen*; Table 1.1 above may need a column specifying the number of times the player was not out) and for defining and changing physical data structures is called the *data definition language (or DDL)*.
2. The database must be loaded. Put the initial data that is available into the computer using the DDL. Obviously, a database is dynamic and information is added, modified and deleted from it all the time.

3. The database may now be queried. New data can be inserted, deleted and updated, once the data has been stored in the computer. To manipulate, modify and retrieve information, a language for manipulating and retrieving data stored in the DBMS is needed. Such a language is called a *data manipulation language (DML)*.

Each DBMS needs a DDL to define the database and a DML to retrieve and modify information in the database. The two languages, DDL and DML, may be parts of a unified database language as, for example, in SQL which is discussed in Chapter 5. We also note that SQL provides facilities for use in two different ways:

1. *Query Language*—SQL provides powerful facilities to interact with the database. A query language is designed to be simple so that it can be used by nonprogrammers.
2. *Extended Host Language*—Although the SQL query language provides powerful interactive facilities to database users, it is often necessary to retrieve information from a database and process it in a way that cannot be done by the query language. It then becomes necessary for a database system to provide facilities in a standard programming language like Java or C++ to enable the user to interact with the database through software that has been developed in one of these standard programming languages. The programming language that is extended is usually called a *host language*.

To summarize, a database system consists of the items listed in Fig. 1.2.

Some years ago, as noted earlier, DBMS packages marketed by computer manufacturers (for example, IBM) were designed to run only on that manufacturer's machines. The market has changed dramatically since the 1980s and now independent software houses are designing and selling DBMS software (for example, ORACLE or Microsoft SQL Server) that can run on many different types of machines.

- The database (data)
- A DBMS (software)
- A DDL and a DML (part of the DBMS)
- Application programs

Figure 1.2 Components of a database system

1.9 USERS OF A DATABASE SYSTEM

There are several ways of classifying database users. They can be classified by the way the users interact with the system or by their roles in a database system. We will present a top-down approach, starting from the database administrators.

1.9.1 Database Administrators

We do not discuss the role of database administrators here since their role is discussed later in the chapter in detail in Section 1.15.

1.9.2 Database Designers

In large database design projects there are usually two types of designers, viz., logical database designers and physical database designers. The logical designers help in identifying data and relationships between them (as we will see in the next chapter) without taking into account the software that will be used. Once the data and relationships have been identified and modeled, the physical designers look at the ways the database

model can be best mapped to physical storage (for example, hard disks) to deliver the best performance. Issues of physical storage are discussed in Chapter 7 and a number of possible data structures and their characteristics are described. The database designers may need to discuss with the database users which aspects of performance are important to them.

1.9.3 System Analysts and Application Programmers

These highly technical users determine the requirements of end users. Systems analysts develop specifications for application programs that are needed to meet these requirements. Application programmers then implement these specifications as application programs that are appropriately tested, documented and maintained. Such programs need to provide the functionality that is required by end users for whom the programs are being designed. The programmers may also be involved in designing and implementing some DBMS modules, for example, for implementing catalog, interface processors, data access, and interfaces to the software packages.

1.9.4 End Users

End users are a great variety of users that use a database in their work. They use the database for querying, updating, and generating reports. Not all end users are using a database every day, for example, I don't use the library database every day but I do use it regularly. Users like me may be called *casual end users* of the library database. In some cases, a user may be using a database even less frequently than I use the library catalogue. For example, I may check my airline bookings when I have booked a flight but this might happen only 2–3 times a year. Such users may be called *occasional end users*.

The end users normally learn only a few facilities that they use repeatedly. These facilities may include ways to use a database query language to specify their query requests. For example, while accessing the university library catalogue I need to learn the method to search it at an advanced level, reserve a book, renew a loan and browse through some electronic books and journals available in the Monash library.

1.9.5 Naive and Sophisticated Users

Some end users have jobs that require constant querying and updating a database, using standard types of queries and updates (sometime called *canned* transactions) that have been carefully programmed and tested. They do not need to understand the DBMS or the underlying database. Such users may be called *naïve users* and include, for example, a bank teller, a cashier, a policeman, or a nurse. On the other hand, some users are quite sophisticated and are familiar with the DBMS that they are using, including the underlying structure of the database. These users may be called *sophisticated users* and they are likely to be using SQL interactively or in an embedded form to perform their queries. These users include engineers, computer scientists and business analysts who are thoroughly familiar with the facilities of the DBMS and the underlying database and are able to implement its applications to meet their advanced requirements.

1.9.6 Workers Behind the Scene

They are not classified as database users since they typically do not use the database as a part of their job. They include tool developers that develop tools for database design, performance monitoring, graphical interfaces, and test data generation. Another group of users behind the scenes is operators and maintenance personnel as

well as system administration personnel who are responsible for the actual running and maintenance of the hardware and software environment for the database system.

1.10 ADVANTAGES OF USING AN ENTERPRISE DATABASE

A question that often arises in large organizations is why local branches can't keep local information on a local computer using their own favourite software? Some small branches may feel that they do not need to use a DBMS since they have always managed their data using their favourite file processing system software (for example, Filemaker Pro⁴). There are many reasons why using such file processing software and maintaining information locally is often not a good idea.

In a central enterprise database system, the data is managed by the enterprise DBMS and all access to the data is through the DBMS providing a key to effective data processing. This contrasts with local data being stored and processed locally perhaps using different DBMS and without any central control. The local database may be based on considerable local knowledge of data structures being used. Although distributed database technology (discussed in Chapter 13) allows an enterprise to develop a conceptual view of the database even if the database is physically distributed, such distributed databases require considerable central monitoring and control.

In a database system, whether it is distributed or not, the enterprise DBMS provides the interface between the application programs and the data. When changes are made to data representation, the metadata⁵ maintained by the DBMS is changed but the DBMS continues to provide data to application programs in the same way as before. The DBMS handles the task of transformation of data whenever necessary.

This independence between the programs and the data, as noted earlier, is data independence. Data independence is important because every time some change needs to be made to the database structure, the application programs that were being used before the change should normally continue to work. To provide

4. FileMaker Pro is a cross-platform file processing software that now allows some database functionality including SQL and web publishing support as well as support for forms and reports.
5. In the context of database, metadata is data about the database, for example, names of tables, number of columns in each of them, their types, number of rows in each table, etc. The term 'meta' comes from a Greek word that means 'alongside, with, after, next'. In addition to database systems, metadata applies to many situations. For example, when an MS Word document is created, it is automatically saved into a file with additional document metadata information that is hidden from the user. This metadata information can hint at authorship, times and dates of revisions and other tidbits. Some of this metadata can be viewed by looking at the properties of the document. Third-party programs are available that will crack open even more metadata hidden in MS Word documents. Another example of metadata is the catalogue in a library. The catalogue is metadata since it is about resources available in the library. In the context of a database, the metadata is stored in the *data dictionary*, which is a term for data describing the database. The database software uses this metadata to interpret and execute SQL queries (as discussed in Chapter 7). Metadata can also be used as a window into the database and to learn about objects and data within the database.

Web pages also have metadata about them. Web document metadata may include the title, author, and modification date of a web document, copyright and licensing information about the document, its content rating, or the availability schedule for some shared resource. Given the importance of web metadata, a Resource Description Framework (RDF) has been designed to represent it. This metadata can itself be presented in a database.

a high degree of data independence, a DBMS must include a well-designed metadata management system.

When an enterprise uses a DBMS, all enterprise data may be integrated into one system, thus reducing redundancies (redundancy occurs when exactly the same data is stored in more than one place) and making data management more efficient.

In addition, the integrated DBMS provides centralized control of the operational data. Some advantages of data independence, integration and centralized control are listed in Fig. 1.3.

These advantages can be elaborated as given below.

- Redundancies and inconsistencies can be reduced
- Better service can be provided to users
- The cost of developing and maintaining systems is low
- Standards can be enforced
- Security can be improved
- Integrity can be improved
- A data model must be developed

Figure 1.3 Advantages of using a database system

- *Redundancies and inconsistencies can be reduced*—Data inconsistencies are often encountered in everyday life. For example, I have come across situations when a new address is communicated to an organization that we deal with (for example, a bank, telephone company, or an electricity company) which results in some communications from that organization being sent to the new address while others continue to be mailed to the old address. Such situations are now less common since combining all the data in a single database results in each piece of information being stored only once. This reduces redundancy. It also reduces inconsistencies, since the enterprise does not have two different addresses anymore. And it reduces the cost of collection, storage and updating of data.
- *Better service can be provided to the users*—A DBMS is often used to support enterprise operations and to provide better service to the enterprise users of the system and to its customers. Once an enterprise establishes a centralized database, the availability of information and the possibility of its revision is likely to improve and the users can obtain new and combined information as soon as it is available. Also, use of a DBMS allows users that do not know programming to interact with the data more easily. The ability to quickly obtain new and combined information is becoming increasingly important for improving services to demanding customers in a competitive environment. Furthermore, various levels of government bodies are now requiring organizations to provide more information about their activities. A centralized DBMS makes it easy to respond to such unforeseen information requests.
- *The cost of developing and maintaining systems is lower*—Since the data and application programs are independent of each other, each of them can usually be changed without affecting the other. This can be particularly important for a website that is supported by a web database. In spite of the large initial cost of setting up a database, normally the overall cost of setting up a database and developing and maintaining application programs, for example, for increasingly important web applications is lower than using earlier technologies.
- *Standards can be enforced*—Since all access to the centralized database has to go through the DBMS, enterprise standards are easier to enforce. Standards may include the naming of the data, the format of the data, and the structure of the data. Furthermore, since all the enterprise data is stored together, data backup may be carried out more regularly and more efficiently.
- *Security can be improved*—Setting up a centralized database makes it easier to enforce security restrictions. Since the data is centralized, it is easier to control who has access to what parts of the database. However, setting up a database can also make it easier for a determined person to breach security. This is discussed in the next section.

- *Integrity can be improved*—Since the data of the organization using a database is often centralized and would be used by many users at the same time, it is essential to enforce appropriate integrity controls. Integrity may be compromised in many ways. For example, someone may make a mistake in data input and the salary of a full-time employee may be entered as Rs. 4,000 rather than Rs. 40,000. A college student may be shown to have borrowed books from the college library but he/she has no enrolment in the college. Salary of a staff member working in one department may be coming out of the budget of another department.

If a number of users are allowed to update the same data item at the same time, there is a possibility that the result of a number of updates will not be the same as intended. For example, in an airline DBMS we could have a situation where two travel agents sell the same airline seat at the same time and the total number of bookings made is larger than the capacity of the aircraft. In Chapter 9, various ways of avoiding such situations are discussed. However, since all data is stored only once, it is often easier to maintain integrity in modern database systems than in older systems.

- *A data model must be developed*—Perhaps the most important advantage of setting up an enterprise database system is the requirement that an overall enterprise data model be built. If an enterprise were to set up several database systems, it is likely that the overall enterprise view would not be considered. Building an overall view of the enterprise data, although often an expensive and onerous exercise, is usually cost-effective in the long term. It also usually provides flexibility as changes are often necessary as the enterprise changes in response to changing demands of the clients and/or customers.

1.11 CONCERNS WHEN USING AN ENTERPRISE DATABASE

As noted earlier, a centralized enterprise DBMS provides online access to the database for many users concurrently. Because of the large number of users, (assuming that the enterprise is large), who will be accessing the enterprise database, the enterprise may incur additional risks as compared to a more restrictive older style file processing system in the areas listed in Fig. 1.4.

These issues can be elaborated as given below.

- Enterprise vulnerability may be higher
- Confidentiality, privacy and security may be compromised
- Data quality and integrity may be lower
- Data integrity may be compromised
- The cost of building and using a DBMS can be high
- It may difficult to change the database when necessary

Figure 1.4 Disadvantages of using a DBMS

- *Enterprise vulnerability*—Centralizing all the data of an enterprise in one database often means that the database becomes an indispensable resource. The survival of a large enterprise may then depend on reliable information being made available from its database on a 24/7 basis. If the database system goes down for an hour or two, the enterprise may be paralysed while the system is down. At the time of writing this book, the university computer system has been going down frequently due to installation of some new network hardware. When the network has been down, most staff members have not been able to do much work and many have just gone home. In fact an enterprise is not only dependent on its databases, it can even be dependent on information in a search engine like Google. For example,

when there were problems with Google in January 2009, many staff members found it difficult to keep working effectively. The enterprise can also become vulnerable to destruction of the enterprise database or to unauthorized modification of the database.

- *Confidentiality, privacy and security*—When information is centralized and is made available to users from remote locations, it is possible to apply stringent controls on who is able to see what information and who is able to modify what information. However the possibilities of abuse can also increase because some hackers may be able to penetrate the security controls. To reduce the chances of unauthorized users accessing enterprise information, it is necessary to take technical, administrative and possibly legal measures, but an intruder with sufficient resources may still be able to breach database security. A recent University of Cambridge report⁶, issued in March 2009, describes how agents of the Chinese government were able to penetrate computers in the office of the Dalai Lama as well as many other computer systems in 103 countries.⁷
- *Data quality*—Since a database provides its users the convenience to access information remotely, adequate controls are needed to control users updating data and thus control data quality and maintain integrity. With an increased number of users accessing data directly, there is an increased risk that data quality and/or integrity will be compromised partly because of input errors. For example, at the time of writing this book, it has been reported that there was an input error in the university salary database which resulted in a person who retired continuing to be paid after the retirement date.
- *The cost of building and using a DBMS*—The database approach provides a flexible approach to data management where new applications can be developed relatively inexpensively. The flexibility is not without its costs, the major cost being that of modeling and building the database, and training or employing personnel. As noted earlier, building a database for a large and complex organization can be hugely expensive but such an organization cannot operate without an effective database system. For example, building a database for an international airline like Air India or a large bank like ICICI that will continue to meet the enterprise needs in the future is a very challenging task.
- *Inability to change the database when necessary*—In a competitive global environment every business has to be able to change quickly to meet new threats from competitors. An airline may merge with another or develop a partnership with a regional airline. A bank may start an insurance business or develop a stock broking business. The database should be such that it can be modified to deal with such major changes to the business. Not all database systems are easily adaptable to new business demands, in particular when a business undergoes a merger or acquisition. If the database system is inflexible, the enterprise may lose its competitive advantage after a dramatic change like a merger or acquisition.

1.12

THREE-LEVEL DBMS ARCHITECTURE AND DATA ABSTRACTION

We now discuss a conceptual framework for a database management system. Several different frameworks have been suggested since the 1970s. For example, a framework may be based on the functions that the various

-
6. S. Nagaraja and R. Anderson, *The Snooping Dragon: Social-Malware Surveillance of the Tibetan Movement*, Technical Report Number 746, Computer Laboratory, University of Cambridge, UK, March 2009.
 7. J. Markoff, *Vast Spy System Loots Computers in 103 Countries*, New York Times, March 28, 2009.

components of a DBMS provides to its users. It may also be based on different users' views of data that are possible within a DBMS. We consider the latter approach.

A commonly used view of data is the three-level architecture suggested by ANSI/SPARC (American National Standards Institute/Standards Planning and Requirements Committee). ANSI/SPARC produced an interim report in 1972 followed by a final report in 1977. These reports proposed an architectural framework for databases. Under this approach, a database is considered as containing data about an *enterprise*. The three-level architecture presented in Fig. 1.5 provides three different views of the data. These views are discussed in some detail below.

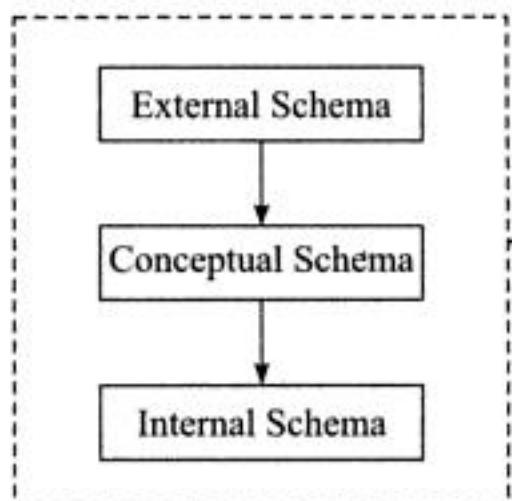


Figure 1.5 ANSI/SPARC DBMS architecture

1.12.1 External View

Each individual user has his/her own view of his/her enterprise database. This view may be a restricted view of the database and the same database is likely to provide a number of different views for different classes of users. In general, the end users and even the application programmers are only interested in a subset of the database if the database is large and complex. For example, a department head in a university may only be interested in the departmental finances and student enrolments in his/her department but not in the buildings and grounds information. The librarian would not be expected to have any interest in the information about departmental finances. The staff payroll office may have no particular interest in student enrolments in computer science unless those numbers have an impact on staff salaries.

1.12.2 Conceptual View

The conceptual view is the information model of the enterprise and contains the view of the whole enterprise without any concern for its physical implementation. This view is usually more stable than the other two views. In a database, it may be desirable to change the way some of the data is stored (internal view) to improve performance while there is no change in the conceptual view of the database. Also, from time to time, a user's view (external view) of the database will change if the user's responsibilities or needs change. The conceptual view is the overall enterprise view of the database and it includes all the information that is of interest and is to be represented in the database. The conceptual view is represented by the conceptual schema which includes details of each of the various types of data.

1.12.3 Internal View

This is the physical or storage view of the database which mostly does not need to be understood by the users. The person managing the database needs to understand this view since the efficiency of the database depends on the way it is stored. The following aspects are considered at this level:

1. What data structure will be used?
2. What access paths do users require to retrieve data, for example, specification of primary and secondary keys, indexes, and sequencing?

3. Miscellaneous, for example, data compression and encryption techniques, and optimization of the internal structures.

Efficiency considerations are the most important at the internal level and the data structures used to represent the database are chosen to provide an efficient database. The internal view does not deal with the physical devices directly. It is only necessary for the internal view to look at a physical device as a collection of physical pages and allocate storage in terms of logical pages.

The separation of the conceptual view from the internal view enables a logical description of the database without the need to specify physical structures. This is usually called *physical data independence*. When we separate the external views from the conceptual view we are able to change the conceptual view without changing the external views. This separation is sometimes called *logical data independence*. Data independence is discussed in more detail in Section 1.13.

Example 1.5

Assuming the three-level view of the database, a number of mappings are needed to enable the users working with one of the external views. For example, the staff payroll office in a university may have an external view of the database that consists of the following information only:

1. Each staff member's employee number, full name and home address.
2. Each staff member's tax information, for example, number of dependants and other relevant information.
3. Each staff member's bank information including the account number where salary is deposited.
4. Each staff member's employment status, salary level and leave information.

We have only listed a sample of information that the payroll office requires. Normally there would be other information including information that deals with payments, provident fund, retirement or pension scheme that an employee participates in.

The conceptual view of the company's database may contain detailed information about the managerial staff, full-time staff, and casual staff as well as the company's financial information and so on. A mapping will have to be created where all the staff members in the different categories are combined into one category for the payroll office. This will need to be mapped to the external view for the payroll office. Also, if there are some changes in the conceptual view, the external view will remain the same even if the mapping is changed.

1.12.4 Data Abstraction

The three-level database architecture allows a clear separation of the overall database view (conceptual view) from the external users' understanding of the database and from the physical data structure layout. A database system that is able to separate the three different views of data is likely to be flexible and adaptable. This flexibility and adaptability is called *data abstraction*. It is a powerful concept because it provides a number of advantages as mentioned below:

1. *Portability*—Portability is important since it is often desirable to be independent of a specific database vendor. The application programs need to be DBMS vendor specific, thus making it easier to replace DBMS software from one vendor to another.

2. *Flexibility and Adaptability*—Flexibility is important since enterprise requirements change according to business conditions and it is important that a database system be able to meet the requirements of new services.
3. *Application Decoupling and Data Independence*—These are important, as discussed earlier, as it is desirable to separate the underlying physical data and the applications. The applications can therefore make use of the data without specific knowledge of the database. This is called *data independence* and is discussed in more detail in the next section.

1.13 DATA INDEPENDENCE

As discussed earlier, data independence deals with independence between the way the data is structured and the programs that manipulate it. Application programs in a database system can continue to work in a certain sense independent of the storage structure of the database. Given data independence, a DBMS may change the structure of the data without having to change the application programs. This is not possible in conventional file processing systems which have a very low level of data independence since the file structure and the application programs are tightly coupled together. Also, no changes can be made to the file structure without making a corresponding change in the application programs. The programs use detailed knowledge of the file structure in manipulating data.

It is possible to have a high level of data independence in database systems only because the application programs do not deal with data in the database directly. Instead the database management software serve data to the application programs in a pre-defined and agreed manner that is independent of the way the data is stored by the system.

The three-level DBMS architecture discussed in the last section illustrates the two types of data independence possible. The first is called *logical data independence*. The second is called *physical data independence*. Logical data independence is the idea of changing the conceptual view, for example, adding a new column in a table, without affecting the external views. Thus it allows logical changes to be made to a database without worrying about the application programs. Physical data independence, on the other hand, is the idea of changing the internal view; for example, changing the data structure used for storing the database, without affecting the conceptual or the external views.

Data independence implies that the application programs should not need to know any of the following:

- The ordering of the data fields in a record
- The ordering of the records in a file
- The size of each record
- The size of each field
- The format and type of each data item
- Whether a file sorted or not
- The type of data structure being used to store some of the data

Data independence does not mean that the user does not need to know:

1. The names of files in which the relevant data is to be found
2. The names of the data fields that the user wishes to access

*image
not
available*

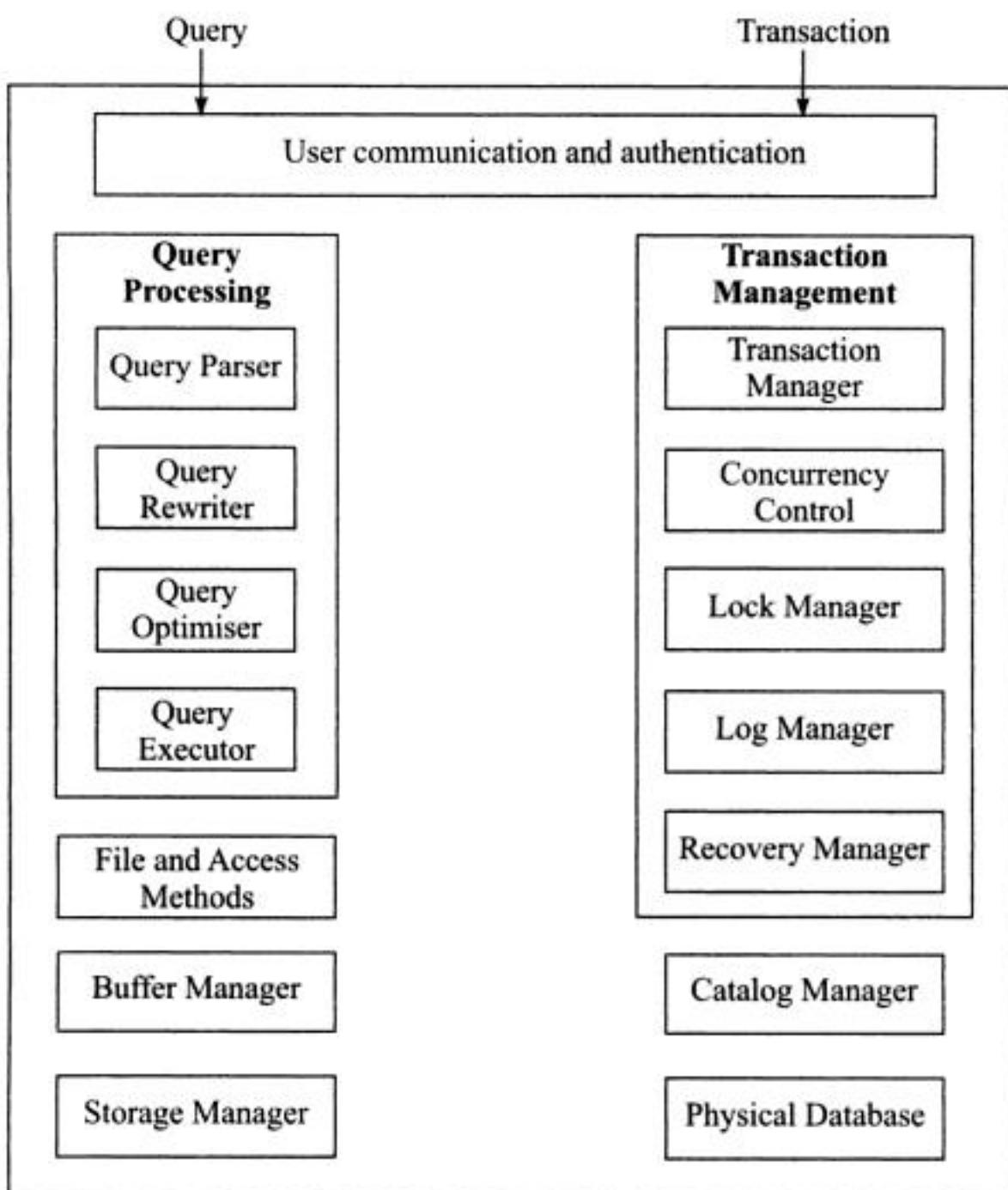


Figure 1.6 System architecture of a DBMS

procedural language programs. Query processing including details about the various components of a query processor are discussed in Chapter 7.

In a nonprocedural or a declarative language like SQL, no sequence of operations is explicitly given and therefore a query may be processed in a number of different ways, often called *plans*, where each plan might have a different sequence of operations. Optimization in declarative languages therefore is a more complex task, since it not only involves code optimization (ways to carry out operations that need to be carried out) but also selecting the best plan including selecting the best access paths. In many situations, especially if the database is large and the query is complex, a very large number of alternative plans are often possible and it is then not practical to enumerate them all and select the best. Then, it is necessary to consider a small number of possible plans and select the best option from those. Since the savings from query optimization can be substantial, it is acceptable that a DBMS spend some time in finding the best plan to process the query. Again, of course, we assume that we are dealing with queries that are expected to consume a significant amount of computing resources. There is no point in spending time optimizing queries that are so simple that almost any approach could be used to execute them in a small amount of time.

Further details of query processing are provided in Chapter 8.

*image
not
available*

*image
not
available*

*image
not
available*

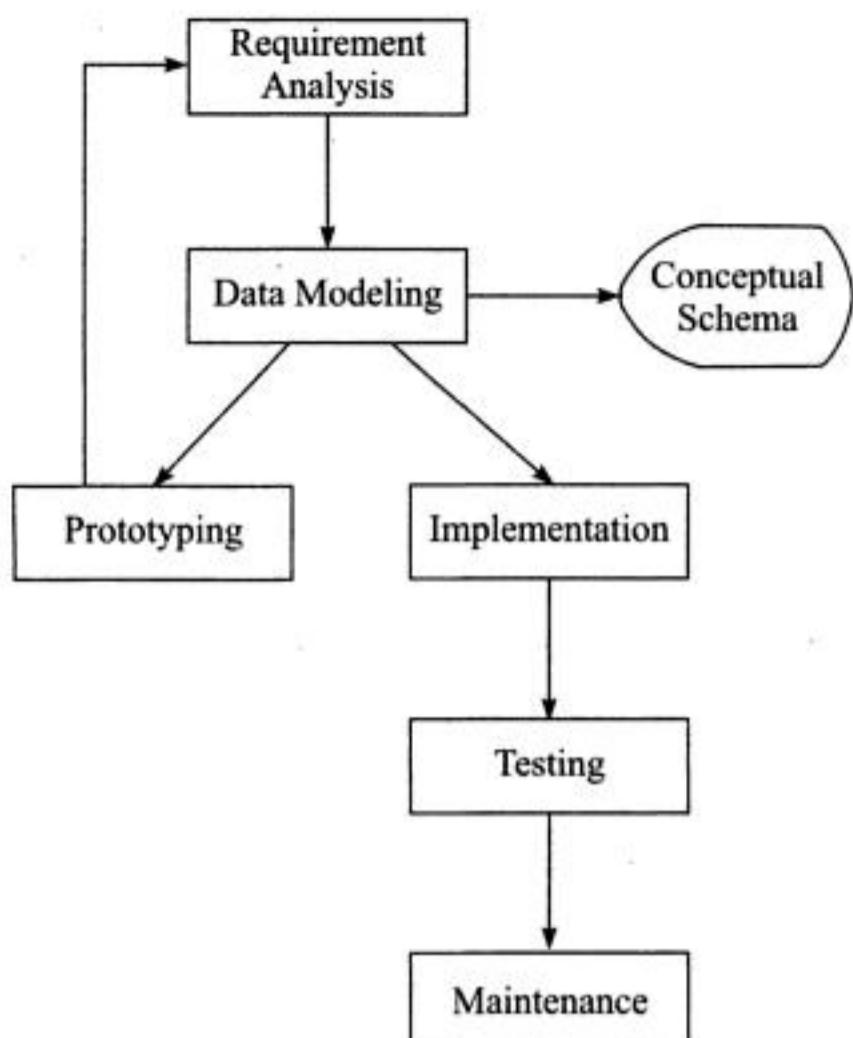


Figure 1.8 Database Design Lifecycle

1.16.1 Requirements Analysis

Each large software project involves requirement analysis to ensure that the software to be developed meets the requirements of the enterprise (or the client). For a database system, requirement analysis involves fully understanding the requirements of the proposed system and describing them precisely, including the queries that are likely to be posed. These requirements must be explored in-depth ensuring that a clear and concise description of the requirements is prepared. The description should include all the properties of the objects of interest, ways they are related to other objects and frequency of access. Nature and volume of the transactions should also be explored and documented. Furthermore, the possibility of growth of the data in future should also be considered. This can be a very challenging task.

A requirements specification document should be prepared based on all this information. This document must be accepted and signed by both parties, viz., the database designer and the client. Once signed, the document is then used in all design phases that follow.

1.16.2 Data Modeling

Once the first phase has been completed, the information content of the proposed database system is known and the requirements specification document is available, the information must be modeled using a data modeling technique. This is the most important phase of the database design because the quality of the database system and its adaptability is often significantly influenced by the quality of the data model. This

*image
not
available*

*image
not
available*

*image
not
available*

- A DBMS is a large software package designed to manage information.
- A DBMS has many modules including query processor, concurrency control and file manager.
- A database system has many different types of users, for example, end users, naive users and sophisticated users.
- There are a number of advantages and concerns when an enterprise database system is used, for example, reduction of redundancy and database security.
- A database system may be looked as three-level architecture; the three levels are external, conceptual and internal.
- Importance of concepts of data abstraction and data independence has been explained.
- A simple architecture of DBMS has been presented. It has many modules including query processor, concurrency control and recovery manager.
- The database administrator is responsible for managing the database system.
- The responsibilities of a DBA are described including database design, data loading and database monitoring.
- Developing a database system is similar to developing any major software since it includes requirements analysis, design and testing steps.
- Web databases are becoming increasingly important because of widespread use of the World Wide Web.
- It is important to remember a number of factors when buying a DBMS including the importance of DBMS effectiveness and database tools.

REVIEW QUESTIONS

1. Give an example of a simple database related to information that you use regularly. (Section 1.2)
2. List names of some large Indian websites that collect huge amount of information. (Section 1.3)
3. What is a database? Discuss its main features and explain the importance of each feature. (Section 1.3)
4. List the characteristics of databases. (Section 1.4)
5. What is data governance and why do enterprises require it? (Section 1.5)
6. When did database management become important in computing? When was the relational database system invented? (Section 1.6)
7. Describe the disadvantages of using a file processing system compared to a database system? (Section 1.7)
8. What is a DBMS? What tasks does a DBMS carry out? (Section 1.8)
9. What classes of users use database systems? (Section 1.9)
10. Discuss the advantages of using the DBMS approach as compared to using a conventional file system. (Section 1.10)
11. Discuss the concerns in using the DBMS approach as compared to using a conventional file system. (Section 1.11)
12. List the three views for a database system in the three-level architecture. Describe the role of these views. (Section 1.12)

*image
not
available*

*image
not
available*

*image
not
available*

13. As a result of data independence, which one of the following is **not** true?
 - (a) The user does not need to know the size of each file record.
 - (b) The user does not need to know if the file is sorted.
 - (c) The user does not need to know the name of the file that has the relevant information.
 - (d) The user does not need to know the ordering of the fields in the records.
14. Which one of the following is a major difference between a file processing system and a database system?
 - (a) File processing system files may be main memory resident while database is disk resident.
 - (b) There is no data independence in file-processing systems.
 - (c) File systems are able to represent relationships between various data while databases cannot.
 - (d) File processing system can be used by many users.
15. Which of the following are important components of the DBMS architecture?

(a) Query optimizer	(b) Database manager
(c) File manager	(d) Physical database
(e) All of the above are important	
16. When choosing a DBMS which of the following are important?

(a) Effectiveness	(b) Efficiency
(c) The name of the company	(d) Good end user tools
(e) All of the above are important	

EXERCISES

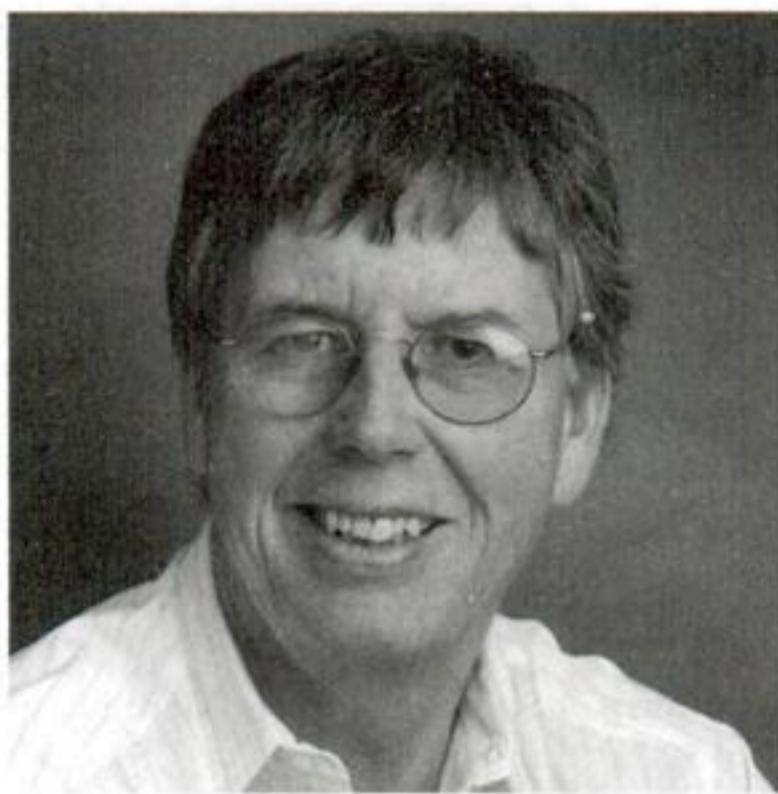
1. Discuss the importance of information for modern enterprises. Give some examples to illustrate your answer.
2. Explain the relationship between an enterprise, its data processing needs and its database.
3. Why should you choose a database system instead of simply storing data in a spreadsheet? When would it make sense not to use a database system?
4. What is logical data independence and why is it important? Give a practical example when logical data independence is important.
5. What is meant by logical and physical structure of data? Explain the difference between logical and physical data independence.
6. Explain the difference between external view, internal view, and conceptual view in three-tier database architecture. How are these different schema layers related to concepts of logical and physical data independence?
7. Which of the following plays an important role in representing information about the real world in a database? Explain briefly.
 - Data definition language
 - Data manipulation language
8. Would a DBMS designed for use on a PC by a single user also have the same components as shown in Fig. 1.6? Explain your answer.

*image
not
available*

*image
not
available*

*image
not
available*

Michael Stonebraker—Pioneer in Database Systems



Michael Stonebraker has been a pioneer in the field of database research and technology for more than thirty years. He was the main architect of the first relational DBMS INGRES, and the object-relational DBMS, POSTGRES. These database systems were developed at the University of California at Berkeley where Stonebraker was Professor of Computer Science for twenty five years. More recently he has moved to MIT where he has been involved in working primarily on novel DBMS architectures.

Professor Stonebraker is the author of a large number of research papers and books on database technology, operating systems and the architecture of system software services. He was awarded the ACM System Software Award in 1992 for his work on INGRES. He was also awarded the first annual Innovation award by the ACM SIGMOD special interest group in 1994 and was elected to the National Academy of Engineering in 1997. He is also an ACM Fellow.

Stonebraker obtained his Bachelor's degree from Princeton University and a PhD from the University of Michigan in 1971.

*image
not
available*

*image
not
available*

*image
not
available*

2.3 TYPES OF MODELS

There are many different types of models. Models may be classified in the following classes according to the aim of modeling:

1. *Descriptive*—The primary purpose of such models is to describe or understand phenomena or complex machinery. For example, meteorologists build models to understand and predict weather; investment companies build models of share markets to understand the share market and to predict share prices or share index movements; and a high school teacher may build simple models to describe or explain some scientific concept to students. The aim for such models is to describe or understand what is being modeled.
2. *Prescriptive*—The primary purpose of prescriptive models is to clearly specify what a piece of machinery or software is supposed to do. In some cases prototype models may be built to specify how something is supposed to behave or look. The aim for such models is to specify what is being modeled. As an example, it has been shown that prescriptive models of human problem solving are able to explain how humans solve problems.
3. *Representative*—The primary purpose of such models is to simulate the behaviour of some phenomena or machinery. For example, computer games that simulate racing cars may be considered representative models. Representative models have been built in many other fields, for example, models of face recognition techniques. The aim for such models is to simulate what is being modeled.

A model does not have to belong to one class; it can belong to more than one class. Database models are normally prescriptive since they prescribe or specify what the database system is supposed to do although they also serve a descriptive role in helping communication between the designer and the customer by describing what the database would include. Therefore before the data available in an enterprise can be put in a DBMS, a database model or an overall abstract view of the enterprise data must be developed. The view can then be translated into a form that is acceptable by the DBMS. Although at first sight a modeling task may appear trivial, it is a very complex process for large database systems since the process often involves making compromises between what the users want and what can be realistically implemented. For example, if a user wants one single model to model all the cricket statistics that are available at the *cricinfo* website (<http://stats.cricinfo.com/ci/engine/current/records/index.html>), the data model would be very complex.

2.4 PHASES OF DATABASE MODELING

In the last chapter we presented the different phases of database design life cycle. One of the phases of the design life cycle was database modeling. This phase is discussed in more detail in this section.

The complexity of mapping the enterprise data to a database management system can be reduced by dividing the process into two phases. The first phase as noted above involves building an overall view (or model) of the *real world* which is the enterprise. This process is often called *conceptual modeling*. The objective of the model is to represent, as accurately as possible, the information structures of the enterprise that are of interest and independent of all physical considerations. This model is sometimes called an *enterprise conceptual*

*image
not
available*

- A DBMS is a large software package designed to manage information.
- A DBMS has many modules including query processor, concurrency control and file manager.
- A database system has many different types of users, for example, end users, naive users and sophisticated users.
- There are a number of advantages and concerns when an enterprise database system is used, for example, reduction of redundancy and database security.
- A database system may be looked as three-level architecture; the three levels are external, conceptual and internal.
- Importance of concepts of data abstraction and data independence has been explained.
- A simple architecture of DBMS has been presented. It has many modules including query processor, concurrency control and recovery manager.
- The database administrator is responsible for managing the database system.
- The responsibilities of a DBA are described including database design, data loading and database monitoring.
- Developing a database system is similar to developing any major software since it includes requirements analysis, design and testing steps.
- Web databases are becoming increasingly important because of widespread use of the World Wide Web.
- It is important to remember a number of factors when buying a DBMS including the importance of DBMS effectiveness and database tools.

REVIEW QUESTIONS

1. Give an example of a simple database related to information that you use regularly. (Section 1.2)
2. List names of some large Indian websites that collect huge amount of information. (Section 1.3)
3. What is a database? Discuss its main features and explain the importance of each feature. (Section 1.3)
4. List the characteristics of databases. (Section 1.4)
5. What is data governance and why do enterprises require it? (Section 1.5)
6. When did database management become important in computing? When was the relational database system invented? (Section 1.6)
7. Describe the disadvantages of using a file processing system compared to a database system? (Section 1.7)
8. What is a DBMS? What tasks does a DBMS carry out? (Section 1.8)
9. What classes of users use database systems? (Section 1.9)
10. Discuss the advantages of using the DBMS approach as compared to using a conventional file system. (Section 1.10)
11. Discuss the concerns in using the DBMS approach as compared to using a conventional file system. (Section 1.11)
12. List the three views for a database system in the three-level architecture. Describe the role of these views. (Section 1.12)

*image
not
available*

*image
not
available*

*image
not
available*

relationship with the employee. Similarly, history of employment of an employee if included in the database would not have a primary key without the support of the employee's primary key. Such entities that require a relationship to be used in identifying them are called *weak entities*. Entities that have primary keys are called *strong* or *regular entities*. Similarly, a relationship may be weak or strong (or regular). A *strong relationship* is between entities each of which is strong; otherwise the relationship is a *weak relationship*. For example, any relationship between the children of the employees and the schools they attend would be a weak relationship. A relationship between the *Employee* entity set and the *Project* entity set is a strong relationship.

A weak entity is also called a *subordinate entity* since its existence depends on another entity (called the *dominant entity*). This is called *existence dependence*. The weak entity therefore does not possess sufficient attributes to form a primary key. This is in contrast to a regular or strong entity that has key attributes and therefore does not exist because of another entity. The relationship of a weak entity to its owner entity type is called the *identifying relationship*.

It is interesting to look at the sources of weakness in a weak entity. Often it is because of the hierarchy of entity sets. For example, species name and genus name may form a key of an entity called plant.

Another example of weak entity is that a customer has several accounts with a bank. Every transaction on each account has transaction number but different transactions in different accounts could have the same number so the transaction number by itself is not sufficient to form a primary key. Therefore a transaction cannot be uniquely identified without the account number and is therefore a weak entity set. For a weak entity set to be meaningful, it must be part of a one-to-many relationship set. This relationship set should not have any descriptive attributes.

A weak entity set does not have a primary key but we need a means of distinguishing among the weak entities. The discriminator of a weak entity set is a set of attributes that allows distinction between instances of a weak entity to be made. Therefore, the primary key of a weak entity is found by taking the primary key of the strong entity on which it is existence-dependent combined with the discriminator of the weak entity set.

Although existence dependency does not imply ID dependency, a weak entity is often *ID dependent* on the dominant entity. This method of identifying entities by relationships with other entities can be applied recursively until a strong entity is reached. Although the relationship between the dominant entity and a weak entity is usually one-to-many, in some situations the relationship may be many-to-many. For example, a company may employ both parents of some children. The dependence is then may be many-to-many.

Terms that are useful:

- (a) *Weak entity*—an entity that depends on another entity for identification.
- (b) *Weak entity relation*—a relation that is used for identifying entities, for example, the relationship between employees and their dependants.
- (c) *Regular entity relation*—a relation not used for identifying entities.

2.5.5 Naming Conventions

It is best to be consistent while giving names to entities, relationships and attributes. One commonly used practice is to use a singular name for all entities and use verbs for all relationships. In many respects it doesn't

*image
not
available*

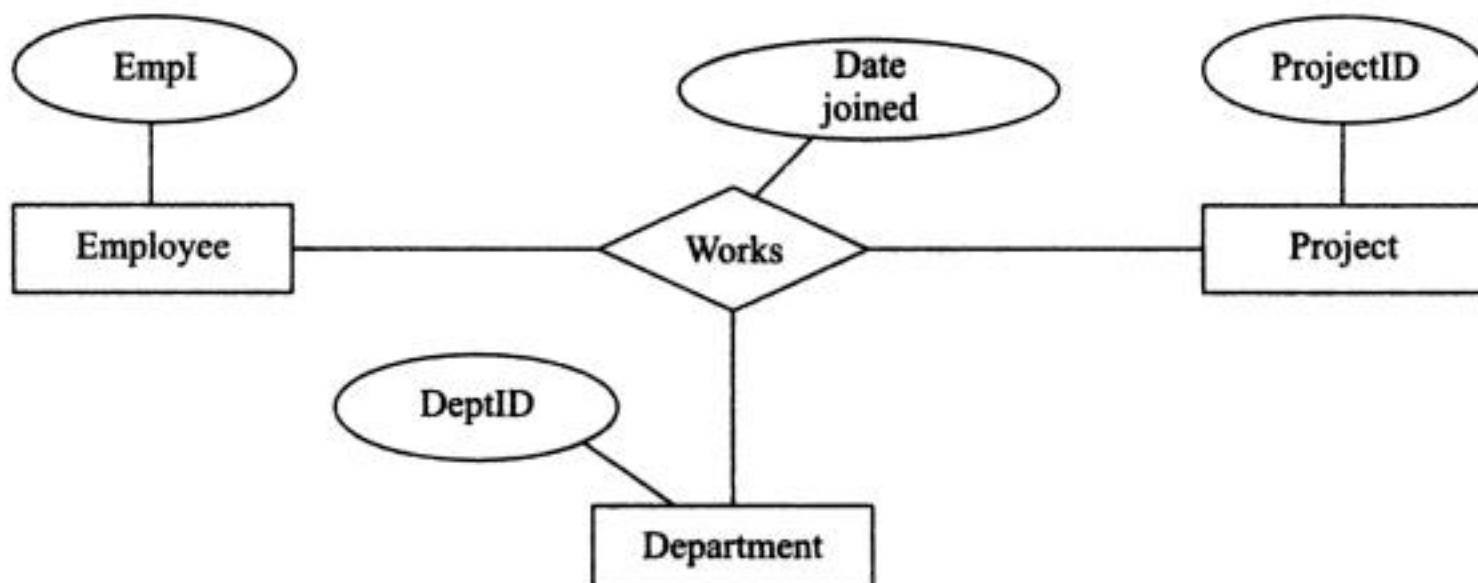


Figure 2.7(a) A ternary relation between *Employee*, *Project* and *Department*

therefore have three one-to-many binary relationships in the ternary relationship. Can they be represented by three binary relationships as shown in Fig. 2.7(b)?

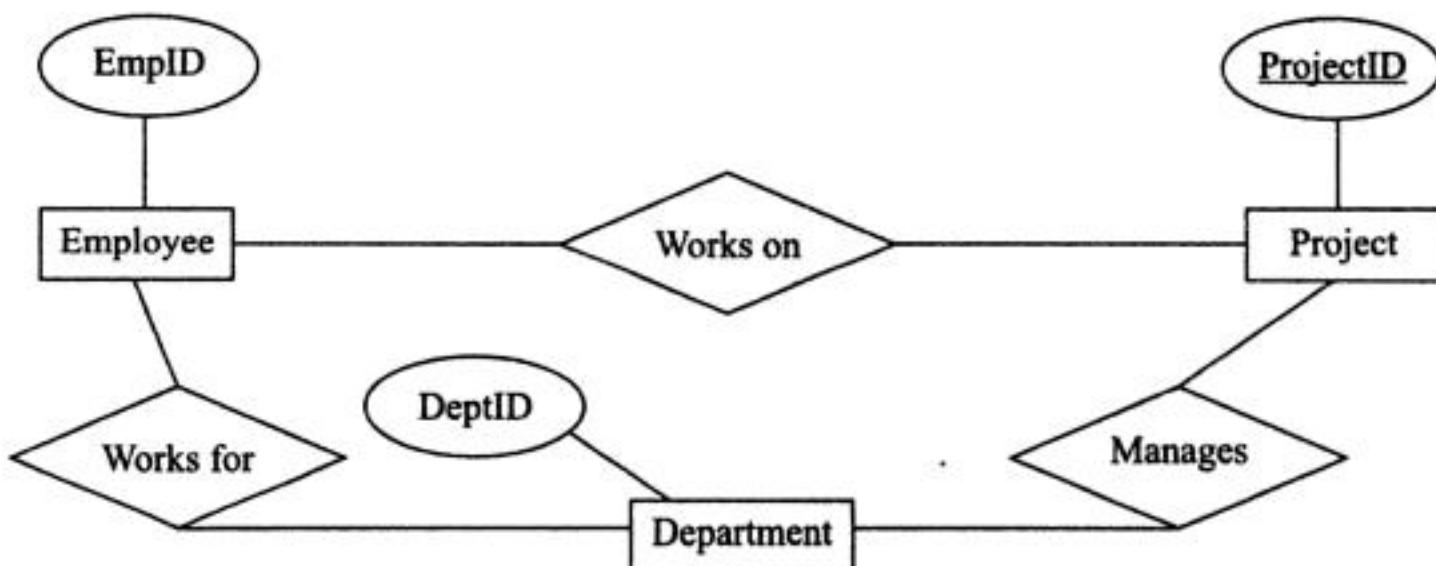


Figure 2.7(b) Three binary relations between the relations *Employee*, *Project* and *Department*

The three binary relationships shown in Fig. 2.7(b) are in fact not able to represent the same information as the ternary relationship in Fig. 2.7(a). What information can be represented in the E-R diagram shown in Fig. 2.7(a) that cannot be represented in the diagram shown in Fig. 2.7(b)? Is there information that the model in Fig. 2.7(b) can represent but the model in Fig. 2.7(a) cannot? We leave it to the reader to answer these questions.

Another E-R model that attempts to model the same information as the ternary relationship replaces the ternary relationship in Fig. 2.7(a) with a weak entity *Works* and three binary relationships between this entity and the other three entities.

2.5.9 E-R Diagram Conventions

A number of issues regarding conventions may arise while developing a database model. The entity-relationship model provides no guidance on how to deal with them. For example,

- Should one cross lines in the Entity-Relationship diagram when the model is complex?

*image
not
available*

*image
not
available*

*image
not
available*

Aggregation in E-R modeling may be implemented by including a relationship between the supertype entity and the component entities. It should be noted that when aggregation is used, deleting an instance of the supertype entity would normally involve deletion of all its component entity instances.

Example Generalization

The example in Tables 2.4, 2.5 and 2.6 illustrates the concept of generalization. Table 2.4 is an entity *Person* in a university. It includes all persons at the university including students and staff since quite a lot of information about them is common. Not all such information which may include address, telephone number, date of birth, place of birth, date of joining the university, citizenship and so on is shown in the table because of space constraints.

Table 2.4 The supertype entity *Person*

ID	First Name	Last Name	Gender	Status	Category
67543	Arvind	Kumar	Male	FT	Student
76849	Bala	Narasimhan	Male	FT	Academic
86348	Salman	Khan	Male	PT	Both
72398	Deepaka	Dixit	Female	FT	Academic

Let us assume that the entity *Person* is a generalization of entities *Academic* and *Student*. All entity instances in *Person* also belong to one of the subtype entities, *Student* and *Academic*. The subtypes need not include information that is common to them and which is included in the supertype entity *Person*. The subtype entities only need to include information that is relevant to them because these entities will inherit the information from the supertype entity. In fact, the information in the subtype entities also need not be duplicated in the supertype entity. The supertype and subtype entities are related by their primary key.

Table 2.5 The subtype entity *Student*

ID	Degree Enrolled	Date Enrolled
67543	BSc	1 March 2007
86348	PhD	22 July 2006

Table 2.6 The subtype entity *Academic*

ID	Highest Qualification	Discipline	Position
76849	PhD	Maths	Professor
72398	PhD	Computer Science	Professor
86348	BCom(Hons)	Business	Tutor

Note that all entity instances in *Person* participate in one of the lower level entities and one entity instance participates in both *Student* and *Academic*. Therefore, this model allows overlaps in the subtype entities.

*image
not
available*

*image
not
available*

*image
not
available*

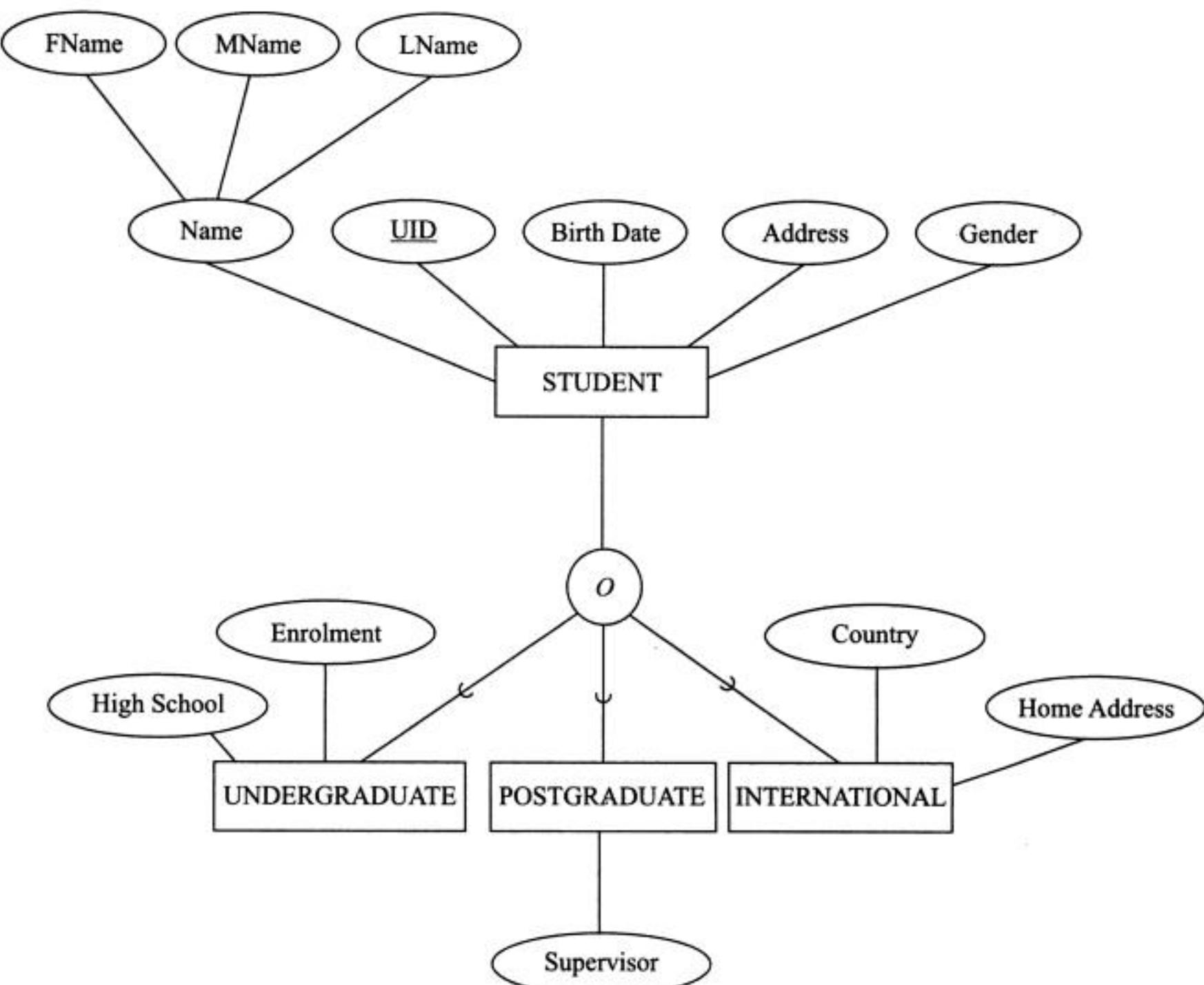


Figure 2.10 EER Diagram showing supertype and subtypes in specialization

level diagrams do. Not all such subtypes and supertypes need to be included in the database implementation; some might be just useful in helping communication.

2.8 THE DATABASE DESIGN PROCESS

As noted in Chapter 1, database design is a complex and challenging process that requires commitment of all the stakeholders including the senior management of the enterprise. In Chapter 1, we noted the following major steps which have now been modified to apply the E-R model approach.

1. *Requirements Analysis*—This step was described in Chapter 1. It involves studying the proposed database application and discussing with the client and other users what is required. A formal set of requirements specifications is prepared and documented.
2. *Identify Entity Sets*—The next step is to use the requirements specifications to identify entity sets that are of interest. This step is independent of the hardware and database software that is to be used.

*image
not
available*

*image
not
available*

*image
not
available*

B. R. Krishna or B. R. M. Krishna. A database may need to include all these versions of the name. How should one store them as attributes?

The guideline for multivalue attributes in entity-relationship is that such an attribute should be considered an entity. Although conceptually multivalue attributes create no difficulties, problems arise when the E-R model is translated for implementation using a relational database system discussed in Chapter 3. As an example, suppose we have an entity called project in a company database and one of the attributes is funding code. If the database model allows a number of funding codes for one project then we have a difficulty and it might be best to have another entity called funding code and then have a relation between the entity type project and the entity type funding code.

Although we have indicated above that an entity set should normally contain information about its properties, a multivalue attribute even if it has no properties other than its value has to be considered an entity.

5. Complexity. E-R models for nontrivial database applications can become rather complex since entity sets tend to have many attributes which contribute to the complexity of the model. The difficulty can be overcome by using a top-down approach so that the model is displayed at several different levels of abstraction with the top level just showing entities or even groups of entities and with the details provided by models at lower levels.

Finally, the E-R model offers limited constraint representation, limited relationship representation, no representation of data manipulation and in some cases results in loss of information. A detailed discussion of weaknesses of the E-R model is presented in the book by Thalheim.

2.11 A CASE STUDY OF BUILDING AN E-R MODEL

We now consider a nontrivial example of building an E-R model. In this example we will attempt to model the information that is generated in a cricket ODI match. We cannot fully model the information generated in an ODI match so our model will be quite limited in scope.

The primary aim in presenting this long example is to provide the reader some understanding about the complex art and science of model building. For this reason the example will include early models with a variety of errors in them. The models will then be improved to hopefully overcome all the problems identified. We emphasise that no solution for a significantly complex problem is unique and there is never a perfect solution.

Normally, an ODI match information is presented as a scorecard as described in the Tables 2.7 to 2.13 below. The information in this example comes from an ODI match played between Australia and India in Sydney in March 2008. India won this match. We present a part of the scorecard which shows the batting and bowling performances of both innings. Surprisingly, there is no standard definition of a scorecard. For example, some may include the number of minutes a batsman was at the crease while others do not, but we assume that the scorecard given below is acceptable. R in the scorecard stands for number of runs scored by the batsman, M for the number of minutes the player was batting, B is the number of balls the batsman faced and 4s and 6s are the number of fours and sixes scored. SR is the strike rate obtained by dividing R by B (converted to a percentage).

*image
not
available*

Looking at the scorecard we find that in the bowling figures, O stands for number of overs, M stands for number of overs in which no runs were scored (called maidens), R stands for the number of runs scored by the batsmen off the bowler's bowling. Finally, W is the number of players (wickets) that the bowler dismissed during the innings and $Econ$ is the economy rate which is equal to R/O .

Table 2.10 gives details of the Australian inning's scorecard.

Table 2.10 Australian innings (Target: 259 runs from 50 overs)

Batsman	How out?	R	M	B	4s	6s	SR
AC Gilchrist	c Dhoni b Kumar	2	3	3	0	0	66.7
ML Hayden	run out (Yuvraj Singh/Harbhajan Singh)	55	122	68	7	0	80.9
RT Ponting	c Yuvraj Singh b Kumar	1	9	7	0	0	14.3
MJ Clarke	b Kumar	17	29	22	1	0	77.3
A Symonds	lbw b Harbhajan Singh	42	81	56	2	1	75
MEK Hussey	c Dhoni b Sreesanth	44	66	42	3	0	105
JR Hopes	c Chawla b Pathan	63	105	80	4	1	78.8
B Lee	b Kumar	7	20	12	0	0	58.3
MG Johnson	c Dhoni b Sreesanth	8	9	6	1	0	133
NW Bracken	c Chawla b Pathan	1	7	2	0	0	50
SR Clark	not out	0	2	0	0	0	-
Extras	(lb 2, w 7)	9					
Total	(all out; 49.4 overs; 230 mins)	249			(5.01 runs per over)		

Table 2.11 gives details of the fall of wickets in the Australian inning.

Table 2.11 The Fall of wickets in the Australian inning

Fall of wickets: 1-2 (Gilchrist, 0.3 ov), 2-8 (Ponting, 2.3 ov), 3-32 (Clarke, 8.6 ov), 4-121 (Hayden, 25.4 ov), 5-123 (Symonds, 25.6 ov), 6-199 (Hussey, 41.6 ov), 7-228 (Lee, 46.4 ov), 8-238 (Johnson, 48.2 ov), 9-247 (Bracken, 49.2 ov), 10-249 (Hopes, 49.4 ov)

Table 2.12 gives details of the Indian bowling in the Australian inning.

Table 2.12 Indian bowling in the Australian inning

Bowling	O	M	R	W	Econ
P Kumar	10	2	46	4	4.6
S Sreesanth	9	0	43	2	4.77

Contd.

Table 2.12 Contd.

Bowling	O	M	R	W	Econ
IK Pathan	8.4	0	54	2	6.23
Harbhajan Singh	10	0	44	1	4.4
PP Chawla	9	0	45	0	5
Yuvraj Singh	3	0	15	0	5

There is more information about a match that one may want to include in the database. Where was the match played? Was it a day match or a day-night match? On which date was the match played? Who were the umpires? Who was the match referee? Who was the TV umpire? Who won the toss? What was the result? Who won the Player of the Match award? Some may even want to know how many people attended the match. Also, one might want to include who the 12th man was and who the substitute fielder was, if substitutes were needed. There is also ball-to-ball information that could be included in an ODI database. There is thus an enormous amount of information that is generated in each ODI match. Some of this information is listed in Table 2.13.

We will now try to simplify the above information for our case study. We don't want it to be so complex that the model becomes so large that it cannot be used in this book. The simplifications listed in Table 2.14 are made with a view that they do not reduce the case study to a toy example.

Table 2.13 Other information about the ODI

- Toss: India won and chose to bat first
- Series: India won the best-of-3-finals 2-0
- Player of the match: P Kumar (India)
- Player of the series: NW Bracken (Australia)
- Umpires: AL Hill (New Zealand) and SJA Taufel
- TV umpire: BNJ Oxenford
- Match referee: JJ Crowe (New Zealand)
- Reserve umpire: TP Laycock

Table 2.14 List of assumptions made in designing the ODI database model

1. The number of extras made is not considered.
2. The date the match was played is not considered.
3. Questions like who were the umpires, who was the match referee and who was the TV umpire are ignored.
4. The toss result is not considered.
5. The Player of the Match award is not considered.
6. No information is stored about the number of people who attended the match.
7. No information about the 12 th man and substitute fielders is considered in our model.

This list of assumptions is perhaps not complete. We should add any other assumptions we discover to this list.

The situation is still quite complex. First, all the information about fall of wickets (FOW) given above appears unstructured. We will now structure it so that we can understand it better. To do so we transform the given information to a tabular form as given in the two Tables 2.15 and 2.16, for the two innings. This information is now easier to comprehend and model.

Table 2.15 Fall of wickets in the Indian inning

<i>FOW</i>	<i>Player out</i>	<i>Score when out</i>	<i>Over when out</i>
1	Uthappa	1/94	20.5
2	Gambhir	2/121	25.2
3	Yuvraj Singh	3/175	34.6
4	Tendulkar	4/205	39.2
5	Sharma	5/209	41.1
6	Pathan	6/237	47.2
7	Dhoni	7/240	47.6
8	Harbhajan Singh	8/249	48.5
9	Kumar	9/255	49.3

The column *Score when out* is given as scores like 1/94, which means the score was 94 with one wicket down. It should be noted that the information on how many wickets were down is redundant but we will not deal with it at this time. The column *Over when out* has information like 20.5, which means that the player was out on the fifth ball of the 21st over.

Table 2.16 Fall of wickets in the Australian Innings

<i>FOW</i>	<i>Player out</i>	<i>Score when out</i>	<i>Over when out</i>
1	Gilchrist	1/2	20.5
2	Ponting	2/8	25.2
3	Clarke	3/32	34.6
4	Hayden	4/121	39.2
5	Symonds	5/123	41.1
6	Hussey	6/199	47.2
7	Lee	7/228	47.6
8	Johnson	8/238	48.5
9	Bracken	9/247	49.3
10	Hopes	10/249	49.4

Example Requirement

We want to look at possible ways of modeling the above ODI match information. We want all the information in a scorecard to be part of our model including fall of wickets and bowling performance. How do we model it?

Option 1

What is the simplest model that we can design for the problem? What are the entities? What are the relationships? One possibility is to try to model the match information in a similar way to what is presented in the scorecard. That is, we have an entity *Match* that provides some basic information about the match. The

batting performance can be represented by an entity *Batsman* and the bowling performance by *Bowler* and the fall of wickets by an entity *FOW*. The entities and relationships are then as follows:

Entities

The entities in the current E-R model are given in Fig. 2.12

Match
Batsman
Bowler
FOW

Figure 2.12 List of entities

Relationships

Figure 2.13 gives the list of relationships between the entities listed in Fig. 2.12.

M_Batting – between Match and Batsman
M_Bowling – between Match and Bowler
M_FOW – between Match and FOW

Figure 2.13 List of relationships

The E-R model then looks like that shown in the E-R diagram in Fig. 2.14.

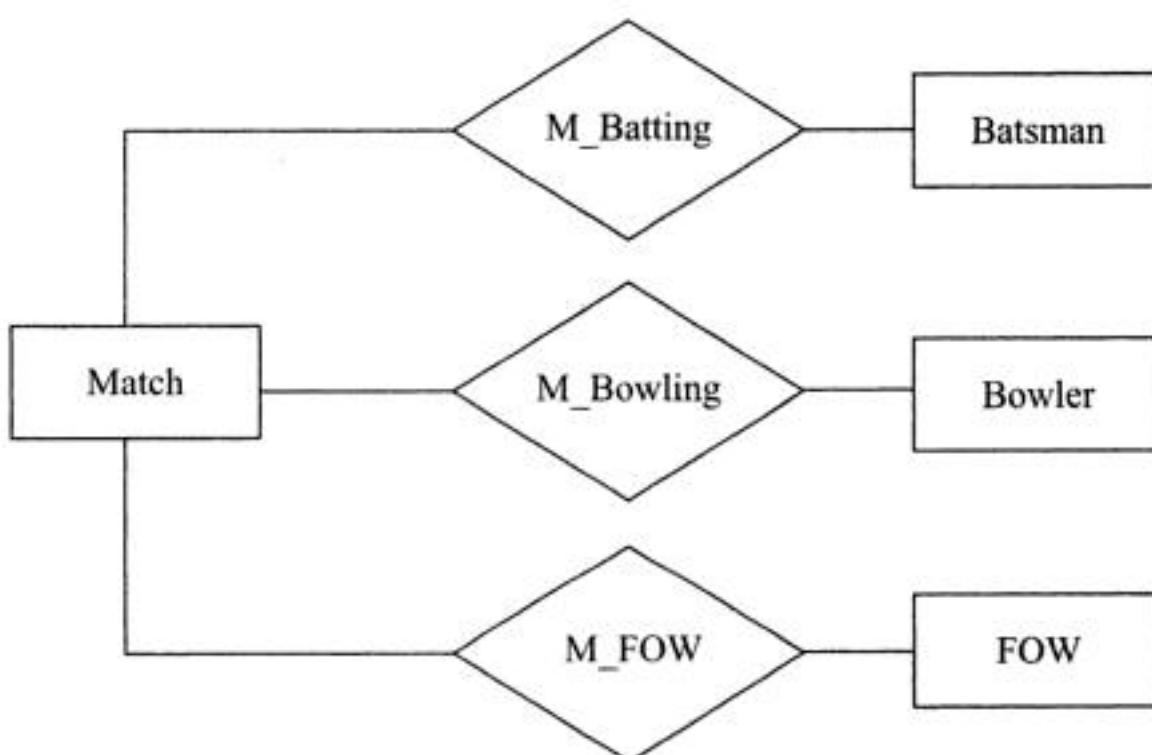


Figure 2.14 Option 1: E-R model for the ODI match database

We now list attributes for each entity.

Match

Match needs to have the attributes listed in Fig. 2.15.

Home team
 Visiting team
 Ground
 Date
 Result
 (Other attributes like Umpires, Referee
 Player of the match, etc., may be included)

Figure 2.15 List of attributes of *Match***Batsman**

Batsman needs to have the attributes listed in Fig. 2.16.

Batsman name and How out
 Number of runs
 Number of minutes
 Number of balls
 Number of fours
 Number of sixes

Figure 2.16 List of attributes of *Batsman***Bowler**

Bowler needs to have the attributes listed in Fig. 2.17.

Bowler name
 Number of overs
 Number of maidens
 Number of runs
 Number of wickets

Figure 2.17 List of attributes of *Bowler***FOW**

FOW needs to have the attributes that are listed in Fig. 2.18.

Wicket number
 Score
 Over
 Batsman out

Figure 2.18 List of attributes of *FOW*

Note that we have not included information about the strike rate for batsmen or economy rate for bowlers since these can be computed given the other information.

All the relationships are one-to-many and we assume that there are no attributes for each of the three relationships. These relationships can therefore be represented by migrating the *Match* primary key to all the other relations. This concept will be discussed in detail in the next chapter.

Let us now discuss what is wrong with this design. Readers are encouraged to list all the mistakes in this design before reading Option 2 below.

Option 2

The above design has many problems. Here is a list of some of them:

1. Although *Match* has a primary key (consisting of the attributes Home Team, Visiting Team and Date), the other entities do not.
2. None of the entities includes the country that the player is representing.

To fix the first problem regarding primary keys we introduce an artificial identifier in the *Match* entity. We will call it the Match ID. Artificial identifiers may be included in other entities as well but we will fix the primary key problem in the remaining entities in another way. The second problem is solved by including the country attribute where needed. We now obtain the following model:

Match

Match now has the attributes listed in Fig. 2.19 (Primary key is Match ID).

Batsman

Batsman now has the attributes listed in Fig. 2.20 (Primary key is Match ID, Player order. Why?).

Match ID
Home Team
Visiting Team
Ground
Date
Result
(Other attributes, like Umpires, Referee, Player of the match, etc. may be included)

Figure 2.19 List of attributes of *Match*

Match ID
Player order
Batsman name and How out
Country
Number of runs
Number of minutes
Number of balls
Number of fours
Number of sixes

Figure 2.20 List of attributes of *Batsman*

Bowler

Bowler now has the attributes listed in Fig. 2.21 (Primary key is Match ID and Bowler name, assuming the name is unique).

Match ID
Bowler name
Country
Number of overs
Number of maidens
Number of runs
Number of wickets

Figure 2.21 List of attributes of *Bowler*

*image
not
available*

Option 4

To be able to uniquely identify players, we need to have a unique identifier for each player. We can introduce an attribute called Player ID for each player and if we wanted to be able to search on a player's last name then we could establish an entity called *Player*.

Let us assume the entity *Player* includes the attributes listed in Fig. 2.24.

Player ID
Player last name
Player first name
Country
When born
Place of birth
Year of first test

Figure 2.24 List of attributes of *Player*

The *Batsman* entity's attributes will change to those listed in Fig. 2.25.

Match ID
Player order
Player ID
How out
Score when out
Number of runs
Number of minutes
Number of balls
Number of fours
Number of sixes

Figure 2.25 List of attributes of *Batsman*

The *Bowler* entity will also change to as given in Fig. 2.26

Match ID
Player ID
Number of overs
Number of maidens
Number of runs
Number of wickets

Figure 2.26 List of attributes of *Bowler*

The *Match* entity will not change.

Option 5

Let us now critically review the solution. Clearly *Match* and *Player* need to be entities. Do *Batsman* and *Bowler* need to be entities? Could they be relationships between the entity *Player* and the entity *Match* since

each batting and bowling performance is related to a player and an ODI match? Making them relationships changes the E-R model to the simple model displayed in Fig. 2.27.

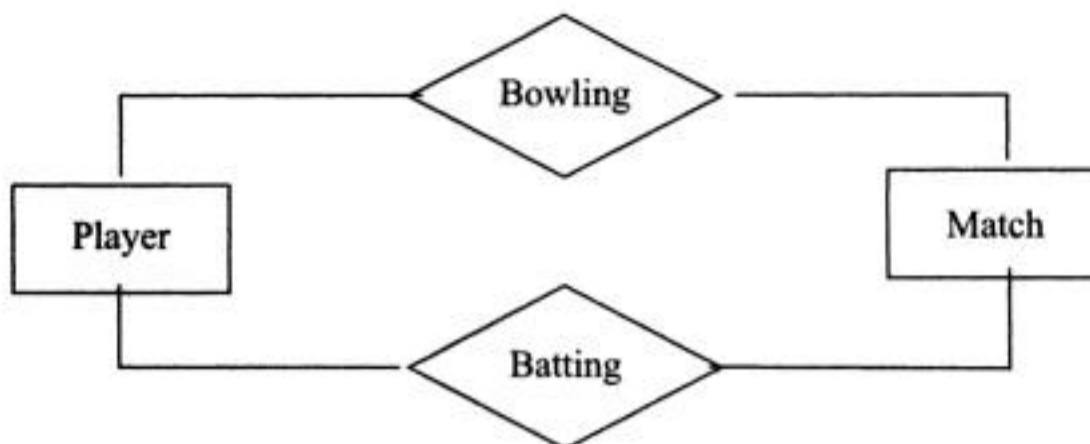


Figure 2.27 E-R diagram for Option 5

The two relationships are many-to-many since each player bats and bowls in many matches and each match has more than one batsman and bowler.

There is no change in the list of attributes for the entities *Player* and *Match*. The attributes for the relationships are given below:

The *Batting* relationship's attributes will change to those listed in Fig. 2.28.

Player ID
Match ID
Player order
How out
Score when out
Number of runs
Number of minutes
Number of balls
Number of fours
Number of sixes

Figure 2.28 List of attributes of *Batting*

The *Bowling* relationship's attributes will also change to those listed in Fig. 2.29.

Player ID
Match ID
Number of overs
Number of maidens
Number of runs
Number of wickets

Figure 2.29 List of attributes of *Bowling*

How do we find out *FOW*?

We leave this as the final solution. Although some problems remain and further modifications will be required.

Final Critique

As noted there are many problems with the final solution. It only shows that modeling is quite a challenging task. Let us list some of the problems that we are able to identify:

1. No information on extras, byes, leg byes, wides and no balls is included.
2. No information on the 12th man and substitutes is included.
3. No information is included about batsmen who did not bat in an innings.
4. In fact it is not possible to list the teams that played in a given match.
5. It is not possible to represent who was the captain of the team in a match and who was the wicketkeeper.
6. The total runs scored in an innings may be derived but is not included as an attribute.
7. The ‘How out’ attribute does not provide sufficient information to print the scorecard.
8. The Duckworth–Lewis (or D/L) system of deciding a match winner when the game is interrupted by rain and the second team is not able to bat for the same number of overs as the first team is not considered.
9. More ODI matches are now being played at neutral venues (for example, Kuala Lumpur or Dubai). There are also triangular ODI series where two visiting teams play each other. These issues are not considered in our model.

We leave it to the reader to list other weaknesses of the final model and to attempt to improve the model.

2.12 EVALUATING DATA MODEL QUALITY

As we have noted, building a conceptual model of an enterprise can be very complex. Building a data model can be more of an art than a science. Once a model has been built, one may want to evaluate the quality of the model and validate how faithfully the model represents the enterprise that is being modeled. If more than one model is available then how does one select the best model?

Evaluating model quality is a difficult issue. Researchers who have investigated the issue of model quality have often proposed that the quality be judged based on a list of desirable properties. Unfortunately these lists almost always lack any fundamental basis and are often based on a person’s modeling experience. Another difficulty with such lists of desirable properties is that the properties are often quite subjective and difficult to measure. For example if we consider readability to be an important quality feature then how can we measure readability so as to conclude that one model is more readable than another?

Nevertheless, a list is useful in providing some guidance on what a modeler should keep in mind while developing a model. We present a consolidated list based on some of the references cited at the end of this chapter. The properties include many commonsense syntactic and semantic properties that one would expect of high quality models. Firstly, one expects that the model be simple and easily understandable by the users of the model. For example, it is expected that the names of the entities and relationships reflect enterprise knowledge and therefore be self-documenting. The model has to be syntactically correct, that is, the model constructs have not been used incorrectly; for example, a construct using a relationship between an existing relationship and an entity has not been used. The model must be complete in that every piece of information

that is required in the database has been included in the model and no unnecessary information has been included. Minimality and nonredundancy require that information normally be included only once. If subclass and superclass structures are used in the model then each subclass must have some property that is not in the superclass for a subclass to be justified. Given that almost every computer system is modified during its lifetime as enterprise requirements change, some suggest that it is desirable that the model be flexible and extendible so that the model is adaptable to changing enterprise requirements, although it is not quite clear how one could judge if a given model is indeed flexible.

Table 2.17 provides a list of quality criteria.

These criteria can be described as given below:

1. *Completeness*—A good model should be complete as far as the client's requirements are concerned. Assessing completeness should be possible with the support of the client as well as assistance from a number of users who understand the proposed model well. An external expert may also be required. The temptation to incorporate requirements beyond what has been agreed to should be curbed.
2. *Correctness*—A good model should model all aspects of the proposed database system as well as business rules of the enterprise. Assessing correctness is not subjective but would require assistance of the client and may require assistance from an expert with knowledge of the database application as well as of database systems.
3. *Consistency*—A good model must not have any contradictions. Consistency, correctness and completeness are the three basic requirements of any good model. Consistency is perhaps more technical than correctness and completeness, although correctness and completeness can also be quite technical. The client may not be able to assist much in checking consistency. Assistance of a number of users who understand the proposed model well may be required. An external expert may also be required.
4. *Minimality*—A good model ought to be compact and should not include redundancy. To assess minimality it is necessary to have assistance from a number of users who understand the proposed model well. An external expert who has good knowledge of data modeling may also be required.
5. *Readability*—A good model should not only be understandable to users who have been involved in the database design but also by users who have not been involved (for example, by a user who takes over the responsibility of the database system after the data modeler is gone and the system has been implemented). Readability can be a somewhat subjective quality. The client's views about readability may also be useful.
6. *Self-explanation*—Related to readability, this criterion proposes for entity, relationship and attributes' names to be such that users can easily understand them. The client's views about self-explanation may be useful. Users may also be consulted since some of them are going to be unsophisticated users and their ability to understand the model is important.

Table 2.17 List of data model quality criteria

- Completeness
- Correctness
- Consistency
- Minimality
- Readability
- Self-explanation
- Extensibility
- Performance

7. *Extensibility*—A good model should be able to accommodate business changes because enterprise requirements change as the enterprise changes but it is almost impossible to judge extensibility without knowing what kind of extensions might be required.
8. *Performance*—Although a data model is developed without considering its performance, it should be possible to make some changes to a model without impacting the model's logical structure.

There are a number of other properties that some experts suggest should be added to this list. These include simplicity, validity, flexibility, understandability, implementability, expressiveness, normality (or normalization), scope, reusability, transformability and levels of detail. We do not discuss these since some of them overlap with the properties we have already described above. For example, are simplicity and understandability similar to readability and self-explanation? Is flexibility similar to extensibility?

For a good discussion of quality in database modeling, refer to Chapter 6 of the book by Batini, Ceri and Navathe.

SUMMARY

In this chapter we have discussed the following topics:

- Models have been used by people for a long time, for example, by architects, scientists and engineers.
- There are many benefits of modelling before building including focussing on the essentials, product or process improvement and exploring alternatives.
- Models can be of three different types, viz., descriptive, prescriptive or representative.
- Database modelling has at least two phases. The first phase involves building an enterprise conceptual model. In the second phase, the conceptual model is mapped to the database model to be used.
- A simple approach of database modeling is called the Entity-Relationship modeling (ERM).
- ERM involves identifying entities (like a person, a player or a book), relationships (like batting, marriage) and attributes (like age, address).
- Relationships may be unary (e.g., employee and supervisor), binary (like employee and the company) or ternary (like employee, project, and department).
- Binary relationships may be 1-1, 1-m or m-n.
- Attributes may be of several types including simple or composite, single-valued or multivalued, stored or derived.
- An E-R model may be represented as Chen's E-R diagram or as Crow's Foot diagram.
- Different notations are used for both types of diagrams. In the E-R diagram a rectangle represents an entity and a diamond a relationship. In the Crow's Foot notation, lines joining entities represent relationships.
- A number of examples have been used to illustrate the diagrams. One of the examples is about employee-department-project.
- All entities and relationships must have a primary key that identifies each instance of the entity set and each entity of a relationship set.
- The concepts of generalization, specialization and aggregation have been explained.

- Extended Entity Relationship modeling and diagrams have been presented.
- Strengths and weaknesses of the ERM have been described. The strengths include the simplicity of the approach while one of the weaknesses is the difficulty of precisely defining an entity.
- A case study to model ODI scorecard is described to illustrate the ERM is presented.
- Finally data quality of a model was discussed and a number of criteria, for example, completeness, correctness and consistency were discussed.

REVIEW QUESTIONS

1. Describe some fields in which modelling is used successfully. (Section 2.2)
2. What are different types of models? (Section 2.3)
3. Describe the phases of database modeling. (Section 2.4)
4. What is an Entity-Relationship model? (Section 2.5)
5. What is an entity and an entity set? (Section 2.5.1)
6. List some naming conventions for an entity and a relationship. (Section 2.5.1)
7. What is a relationship between entities? What are their roles? (Section 2.5.2)
8. Explain, 1-1 relationship, 1- m relationship and m - n relationship? (Section 2.5.2)
9. What is cardinality ratio? (Section 2.5.2)
10. What is an attribute and value sets? (Section 2.5.3)
11. Explain Chen's an Entity-Relationship diagram? (Section 2.5.5)
12. What are ternary relationships? Case a ternary relationship be converted into a number of binary relationships. (Section 2.5.6)
13. What is the purpose of generalization, specialization and aggregation? (Section 2.6)
14. Give two examples of additional features of the Extended Entity-Relationship model? (Section 2.7)
15. List the steps that should be taken in the database design process. (Section 2.8)
16. What are the strengths of the Entity-Relationship model? (Section 2.9)
17. What are the weaknesses in database modeling? (Section 2.10)
18. How can one evaluate an Entity-Relationship model's quality? (Section 2.12)

SHORT ANSWER QUESTIONS

1. What is an entity?
2. What is a relationship?
3. Name three entities that you know should be part of any university database.
4. Name the primary key for each of the entities.
5. Name a superkey of each of the three entities.
6. Name three attributes of each entity you have identified.
7. Identify any relationships that exist between the three entities you have identified.
8. Name the primary key of each of the relationships.

9. List the phases of database modeling.
10. Give examples of 1-1 relationships, 1-*m* relationships and *m-n* relationships?
11. What symbol represents a *m:n* relationship in a E-R diagram?
12. What is a superkey? How does it differ from a primary key?
13. What is cardinality ratio?
14. Give an example of an Entity-Relationship diagram?
15. What are ternary relationships?
16. Can every ternary relationship be represented as a set of binary relationships without adding any more entities?
17. Give an example of a binary relationship by showing an E-R diagram.
18. Give an example of three entities and a ternary relationship by drawing an E-R diagram.
19. Give an example of a unary relationship. Draw its diagram.
20. List some naming conventions for an entity.
21. Give an example of an extended Entity-Relationship model?
22. Explain the steps that should be taken in the database design process.
23. List two major weaknesses of the Entity-Relationship model.

MULTIPLE CHOICE QUESTIONS

1. Which of the following professionals build models in their profession?
 - (a) Teachers
 - (b) Architects
 - (c) Engineers
 - (d) Computer architects
 - (e) All of the above
2. Which of the following are benefits of modeling?
 - (a) It helps focus on the essentials.
 - (b) It helps communication and understanding.
 - (c) It helps explore alternatives.
 - (d) It can help product or process improvement.
 - (e) All of them
3. Models may be classified into a number of classes. Which one of the following is **not** one of these classes?
 - (a) Descriptive
 - (b) Prescriptive
 - (c) Representative
 - (d) Prototype
4. Which one of the following is **not** correct?
 - (a) The E-R model was first proposed by E. F. Codd.
 - (b) The E-R model is a popular high-level conceptual data model.
 - (c) The E-R model is used for conceptual design of database applications.
 - (d) The major strength of the E-R model is its simplicity and readability.
5. E-R modeling is part of the database design process. Which one of the following the E-R modeling process does **not** include?
 - (a) Requirements analysis
 - (b) Conceptual design
 - (c) Logical design
 - (d) Security and authorization design
6. Which one of the following is most **unlikely** to be an entity?
 - (a) A person
 - (b) A job or position
 - (c) A travel plan
 - (d) An address

7. Which two of the following are most **unlikely** to be relationships?
 - (a) A lecture
 - (b) An enrolment
 - (c) A city
 - (d) A shipment
 - (e) A date
8. Which one of the following is **unlikely** to be an attribute?
 - (a) Location
 - (b) Enrolment
 - (c) Address
 - (d) Date of birth
9. Which one of the following is **not** included in the classical E-R model?
 - (a) Entities
 - (b) Relationships
 - (c) Integrity constraints
 - (d) Attributes
10. Which one of the following is **not** correct?
 - (a) Each entity must have attributes.
 - (b) Each relationship must have attributes.
 - (c) Some attribute or attributes must be able to uniquely identify each entity instance.
 - (d) Identifiers of the entities participating in the relationship are sufficient to uniquely identify each relationship instance.
11. Which of the following are correct?
 - (a) Relationships represent real-world associations among one or more entities.
 - (b) Relationships have a physical or conceptual existence other than that inherited from their entity associations.
 - (c) Relationships are described in terms of their degree, connectivity and cardinality.
 - (d) Each entity participating in a relationship plays a role.
 - (e) All of the above are correct.
12. Depending on the nature of an attribute, it may belong to one of the pair of types below. Which of the following pair could **not** be considered to be an attribute type?
 - (a) Simple or composite
 - (b) Single-valued or multivalued
 - (c) Stored or derived
 - (d) Null or non-null
 - (e) All of them
13. Which one of the following is a correct notation in E-R diagrams?
 - (a) Entities are ovals
 - (b) Relationships are rectangles
 - (c) Attributes are diamonds
 - (d) Weak entities are double rectangles
14. Which of the following is **not** correct?
 - (a) The E-R model helps to avoid looking at the details and concentrate on the nature of the problem.
 - (b) The E-R model avoids looking at database efficiency.
 - (c) The E-R model helps carry out communication with the client.
 - (d) The E-R model is the only technique available for database design.
15. Which two of the following are **not** correct?
 - (a) A ternary relationship can sometimes be represented by two binary relationships.
 - (b) Any two entities may have any number of binary relationships between them.
 - (c) Generally attributes of relationships are assigned only to many-to-many relationships and not to one-to-one or one-to-many relationships.
 - (d) Each entity participating in a relationship must have a relationship instance for each entity instance.

16. Which one of the following is correct?
- All relationships may be converted to binary relationships.
 - All relationships may be converted to 1:1 relationships.
 - All relationship attributes may be attached to one of the participating entities.
 - All relationships may be represented by a table in a database.
17. For each of the following relationships, indicate the type of relationship (i.e., 1:1, 1:m, m:n).
- Enrolment (a relationship between entities student and subject)
 - Works in (a relationship between entities department and staff)
 - Marriage (a relationship between entities person and person)
 - Manager (between employee and department)
 - Dependent (a relationship between entities person and person)
18. Which one of the following comments about the Entity-Relationship model is correct?
- Differences between entity and attribute are always clear.
 - The procedure of identifying entities and attaching attributes to them is an iterative process.
 - Differences between an entity and a relationship are always clear.
 - Differences between an attribute and a relationship are always clear.
 - None of the above
19. Consider two E-R models about courses, lecturers and time of lecture. Each course has only one lecturer while each lecturer teaches several courses. Model A consists of two entities and one relationship joining them. The entities are lecturer and course and the relationship is lectures. Model B consists of three entities; the first and the third are the same as above but the second entity is called lecture. The first and second entity are joined by a relationship called 'gives' while the second and the third entities are joined by a relationship called 'of'.
- Which of the following are correct?
- Both models allow a course to have more than one lecture from the same lecturer.
 - Model B is more appropriate if information about all lectures, past and present, is to be stored.
 - Model A does not allow lecture date and time to be stored.
 - Both models provide exactly the same information.
20. Which one of the following is correct?
- A ternary relationship must either be 1:1:1, 1:1:m or p:m:n.
 - 1:1:1 ternary relationships exist commonly.
 - Crow's foot notation makes the E-R diagram more compact.
 - It is not possible to represent ternary relationships using Crow's foot notation.
21. Which one of the following is correct?
- If a model includes a supertype vehicle and subtypes car, truck and bus then that must be generalization.
 - If a model includes a supertype vehicle and subtypes car, truck and bus then that must be specialization.
 - If a model includes a supertype computer and subtypes monitor, keyboard, motherboard then that must be specialization.
 - If a model includes a supertype vehicle and subtypes car, truck and bus then that may be generalization or specialization.

22. EER model extends the E-R model. Which one of the following is **not** in the EER model?
- (a) Supertypes and subtypes
 - (b) Inheritance
 - (c) Aggregation
 - (d) Objects
23. Which of the following comments about the E-R model are correct?
- (a) Attributes of 1:1 relationships can be represented as attributes of either of the participating entity types.
 - (b) Attributes of 1: n relationships can be represented as attributes of either of the participating entity types.
 - (c) Attributes of a relationship involving more than two entity types can be represented as attributes of either of the participating entity types.
 - (d) Ternary relationships normally do not have attributes.
 - (e) None of the above.

EXERCISES

1. Identify which of the following are entities, relationships and attributes. List any assumptions that you have made in making your choices and list any attributes that are appropriate for each. What might be the primary key for each entity and relationship that has been identified?
 - (a) Employees
 - (b) Customers
 - (c) Patients
 - (d) Shares
 - (e) Houses
 - (f) Offices
 - (g) Employment position
 - (h) Lecture
 - (i) Examination
 - (j) Grade
 - (k) Address
 - (l) Telephone number
 - (m) Dependents
2. Show how an entity, a relationship and an attribute are represented in E-R diagrams. Draw a diagram showing the entities university department, degree, course, lecture theatre, and instructor and a number of relationships between them.
3. Explain by drawing a diagram of how you would model a course that has many prerequisites and which is a prerequisite for many courses?
4. Explain by drawing a diagram of how you would model a student that is enrolled in many courses and each course has many enrolments.
5. Explain by drawing a diagram of how you would model a driving license which may be for driving a car, bus, truck, taxi, scooter or a tourist bus. Use a superclass/subclass model. List all the information that each entity should possess.
6. Develop an Entity-Relationship model for Indian movies. It should include where it was shot, details of the actors, directors, producers, the movie language and so on.
7. Can the E-R model express a relationship between two or more relationships? Would such a relationship ever be required in real life?
8. Construct an E-R diagram for student information in a Department of Computer Science in a university. This should include personal information about the student, enrolment information, and assessment information. Do not include information about library or finances.

9. Briefly describe the database design process described in this chapter. Explain why the design process is often iterative.
10. Build an E-R model for major banks of the world. Choose may be the top 500 banks or a 1000 or more. Study what are the major pieces of information about these banks available on the Web. Use the information to build a model. Are there any relationships that you can find?
11. Can a weak entity itself have a weak entity dependent on it? Give an example.
12. The two E-R diagrams (Figs 2.30 and 2.31), of the employees, departments and projects are given below. Consider the following queries:
 - (a) Which employees work on project J1?
 - (b) Which departments are involved in project J1?
 - (c) Which parts are being supplied to project J1?
 - (d) How many employees of department D1 working on project J1?
 - (e) Are there parts that S1 can supply but is not supplying currently?
 - (f) Are there parts that J1 uses but are not being supplied currently?
 - Which of the above queries can be made if the first model is used? _____ (Write appropriate query labels A, B, ..., etc., in the space above)
 - Which of the above queries can be made if the second model is used? _____
 - Which of the above queries cannot be answered by any of the models? _____

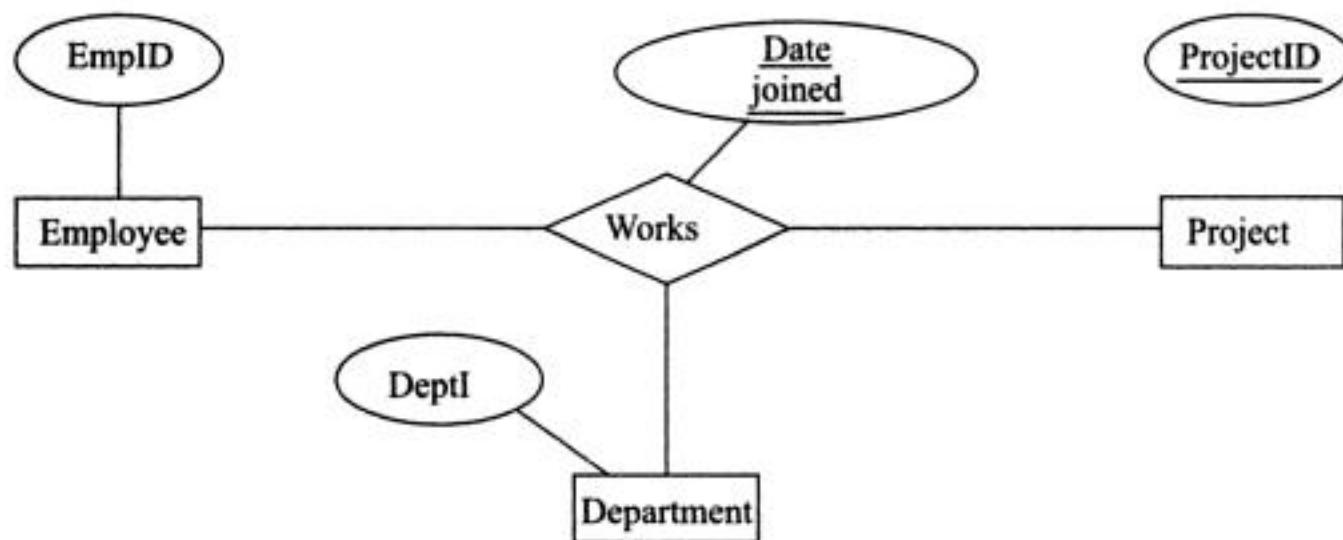


Figure 2.30 A ternary relation between Employee, Project and Department

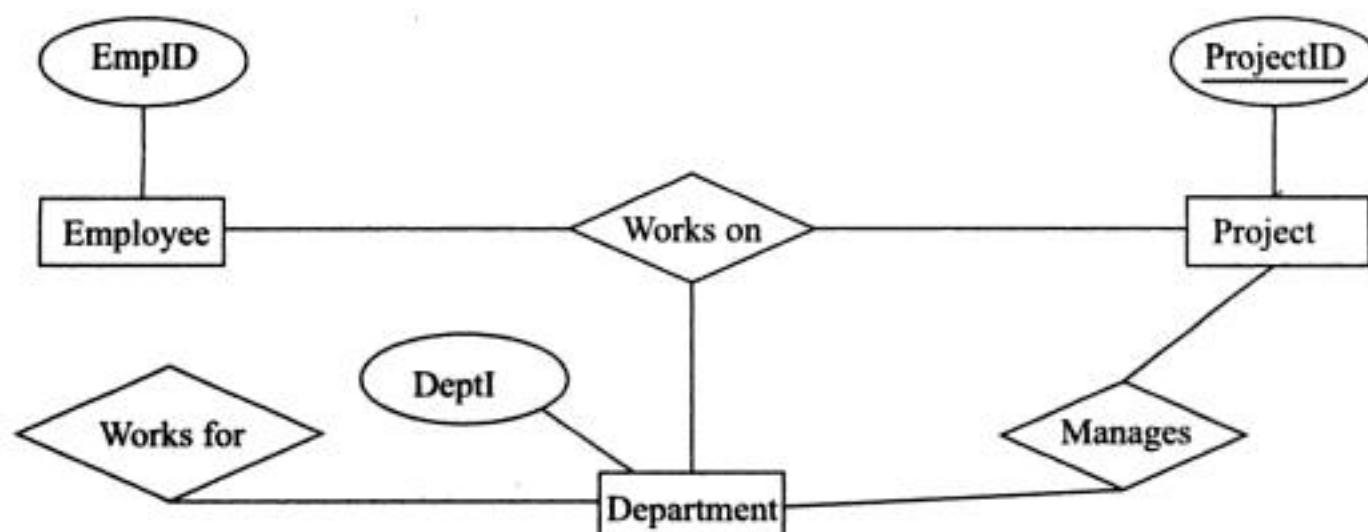


Figure 2.31 Three binary relations between Employee, Project and Department

13. Describe how the following two sets of entities may be included in a generalization hierarchy.
 - (a) Book, journal, magazine, newspaper
 - (b) Truck, car, bicycle, scooter, rickshaw, bus, airplane
14. Describe how the entity animal (in a zoo) can be developed into a specialization hierarchy.

PROJECTS

For each E-R modeling project, present assumptions, develop a model and discuss strengths and weaknesses of the model that you have designed.

1. Western India Medical Clinic (WIMC)

Design a database for WIMC located in Mangalore. The clinic has a number of regular patients and new patients come to the clinic regularly. Patients make appointments to see one of the doctors. Several doctors attend the clinic and they each have their own hours when they attend the clinic. Some doctors are general practitioners (GPs) while others are specialists (for example, cardiologists, dermatologists, endocrinologists). Doctors' hours are defined and fixed by each doctor but can be changed if necessary. Different doctors may charge different fees. Doctors send patients to other clinics for X-Rays and blood tests. These reports are sent from the clinics to doctors electronically. Doctors write medical certificates. Patients have families and the family relationships are important. Medical records of each patient need to be maintained. Information on prescriptions, insurance, allergies, etc., needs to be maintained.

Design an E-R model for the medical clinic.

2. Country/Rivers/Deserts/Mountains/Animals

A database of countries, their rivers, deserts, mountains and animals is to be developed. We want to store a variety of information about the countries depending on availability of information, e.g., name, population, leader's name, number of newspapers per 1000 people, amount of alcohol consumed per person, energy consumed per person, pollutants generated per person, etc. For rivers, we would want to store information like name, country or countries it passes through, source of the river, destination, how long, etc. For mountains, we would want to store information like name, height, country, part of which mountain range, etc. For animals, we need to store information like name, type (mammal, bird, etc.), found in which countries, and some of their characteristics.

Design an E-R model for this problem.

3. All India Mega Constructors (AIMC)

AIMC is an international company that specializes in large-scale construction. They build such things as bridges, dams, office blocks, hotels and factories. The company is divided into a number of departments. Each department specializes in one form of construction. For example, department 654 deals exclusively with the building of bridges, regardless of their geographical location. Head office, however, is interested in the activities of the various geographical areas in which the company operates. This is for political and financial reasons. Each individual construction activity is termed a project. Each project is given a code, which is used to uniquely identify the project in any reports.

Prior to the commencement of construction, the project is analysed into a number of tasks. This is done by the project leader and his or her assistants. Each task is given a code, unique to that project. For each task, a time estimate is made. A monthly budget is developed for the expected lifetime of the project. As each project progresses, actual expenditure is monitored.

There are several types of reports to be produced. For example, a departmental summary is often needed, a project status report may be required and a summary of projects in a particular area is often asked for.

Design an E-R model for the company.

4. The New Programming Company (NPC)

NPC started a venture capital company that decided to employ a number of unemployed computing graduates from the Northern India University (NIU). These graduates were unfortunate enough to graduate in the middle of the global financial crisis (GFC) in 2008. Having become part of the new venture, the graduates started advertising as the best programming company in India even before they had completed any programming job. Although their company name had the word 'new' in it, they claimed that had expertise in old programming languages like FORTRAN, COBOL, Algol, Pascal and Lisp.

The company has three branches called P1, P2 and P3 located in Patna, Kochi and Bhubaneswar. The company's Kochi branch has recently applied to develop software for Qatar Telecom.

The company operates in the following manner. Each time an employee starts on a task, he/she is given a job card on which the employee records the number of hours worked each day. The job card is returned to the supervisor once the assignment is completed. The job card has a variety of information on it including the job card ID, employee ID, job ID, and for each day, the date, the number of hours involved and the nature of work done.

Once the job is completed, NPC sends a bill for services to the customer company. The total charge is computed by adding the number of hours devoted to the job based on the job card multiplied by Rs. 750 which is the company's hourly rate. Once a payment is received the company issues a receipt.

At the end of each month, each employee's performance is evaluated based on the job cards. Total number of hours worked is computed and the income generated by each employee for the company is computed. Bonuses are awarded based on a complex formula for each employee who earns more than one lakh for that month.

Develop an Entity-Relationship model for the company.

LAB EXERCISES

1. Study the E-R diagram (Fig. 2.32) on the next page and do the following:
 - (a) Describe the database that is modeled by the diagram.
 - (b) Are the naming conventions being followed in this diagram? If not, suggest change of names wherever appropriate.
 - (c) Find any mistakes that you can identify in the diagram. Explain and correct them.
 - (d) List all the constraints that the model imposes on the database.

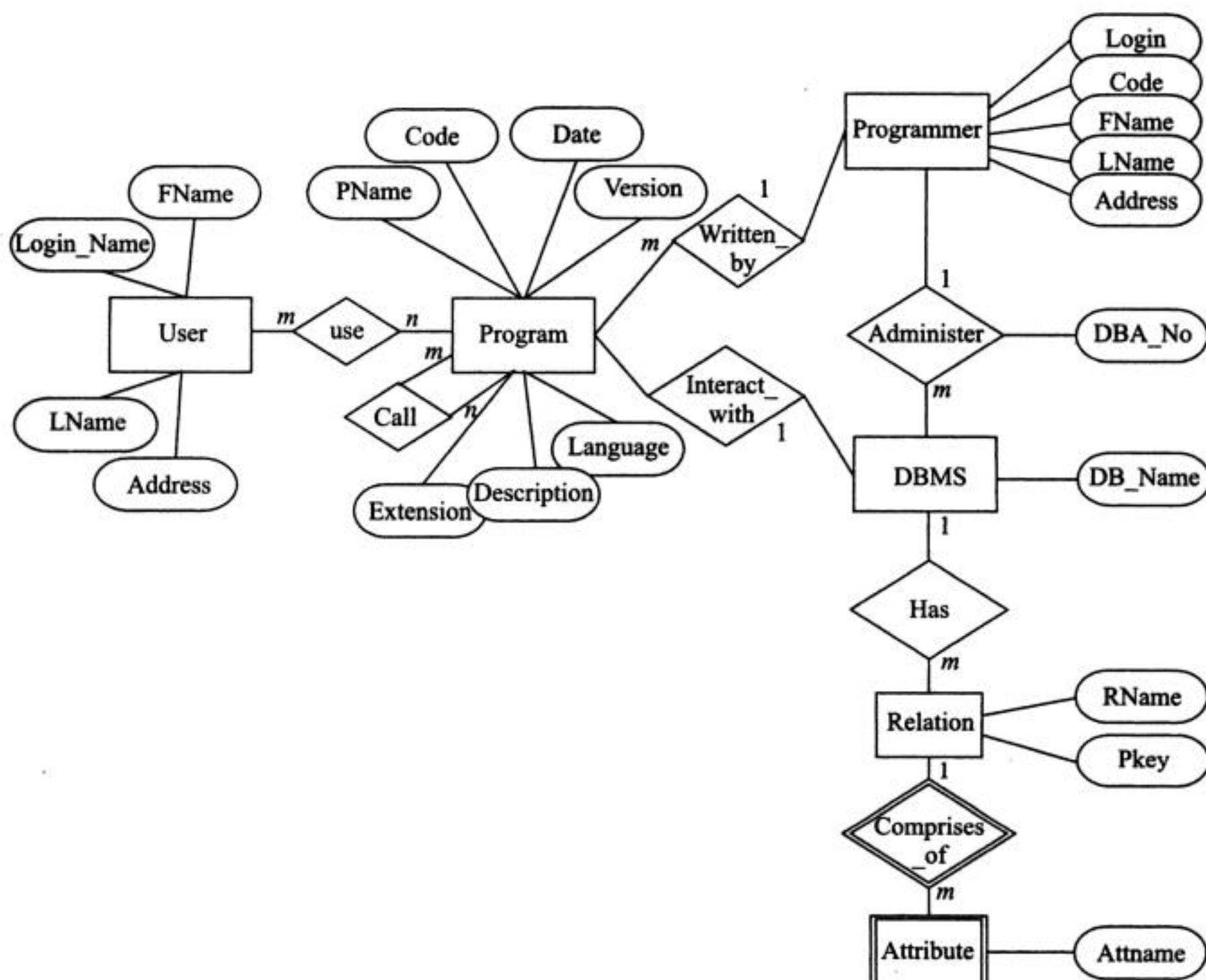


Figure 2.32

- (e) Should the entity relations be a weak entity? If it is and therefore the relationship 'Maintains data in' is a weak relationship, what will be the primary keys of the weak relationships and weak entities.
 - (f) Modify the E-R model by using generalization
2. Consider the case study described in Section 2.11.
- (a) What were the requirements of the case study?
 - (b) Were the requirements met by the model developed?
 - (c) If not, modify the model to ensure that it does meet the requirements.
3. Consider the two E-R diagrams (Figs 2.33 and 2.34) below that model a simple horse-racing database.
- (a) Describe the database by studying these two diagrams.
 - (b) Find any mistakes that you can identify in each of the diagrams. Explain and correct the mistakes in them.
 - (c) Which one of the diagrams models the database better? Are there constraints that apply to one diagram but not the other? Are there queries that could be answered by one of the diagrams but not the other?

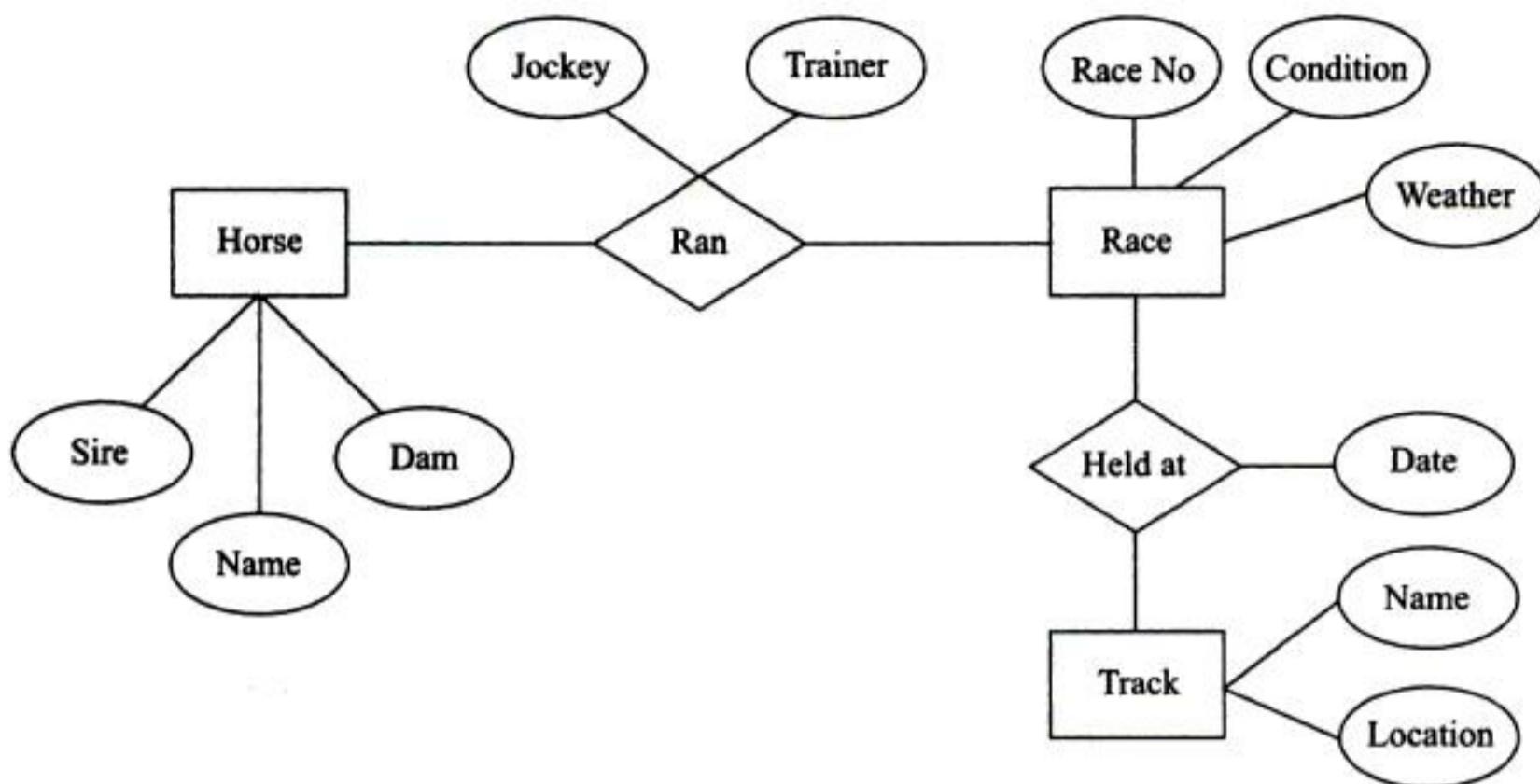


Figure 2.33

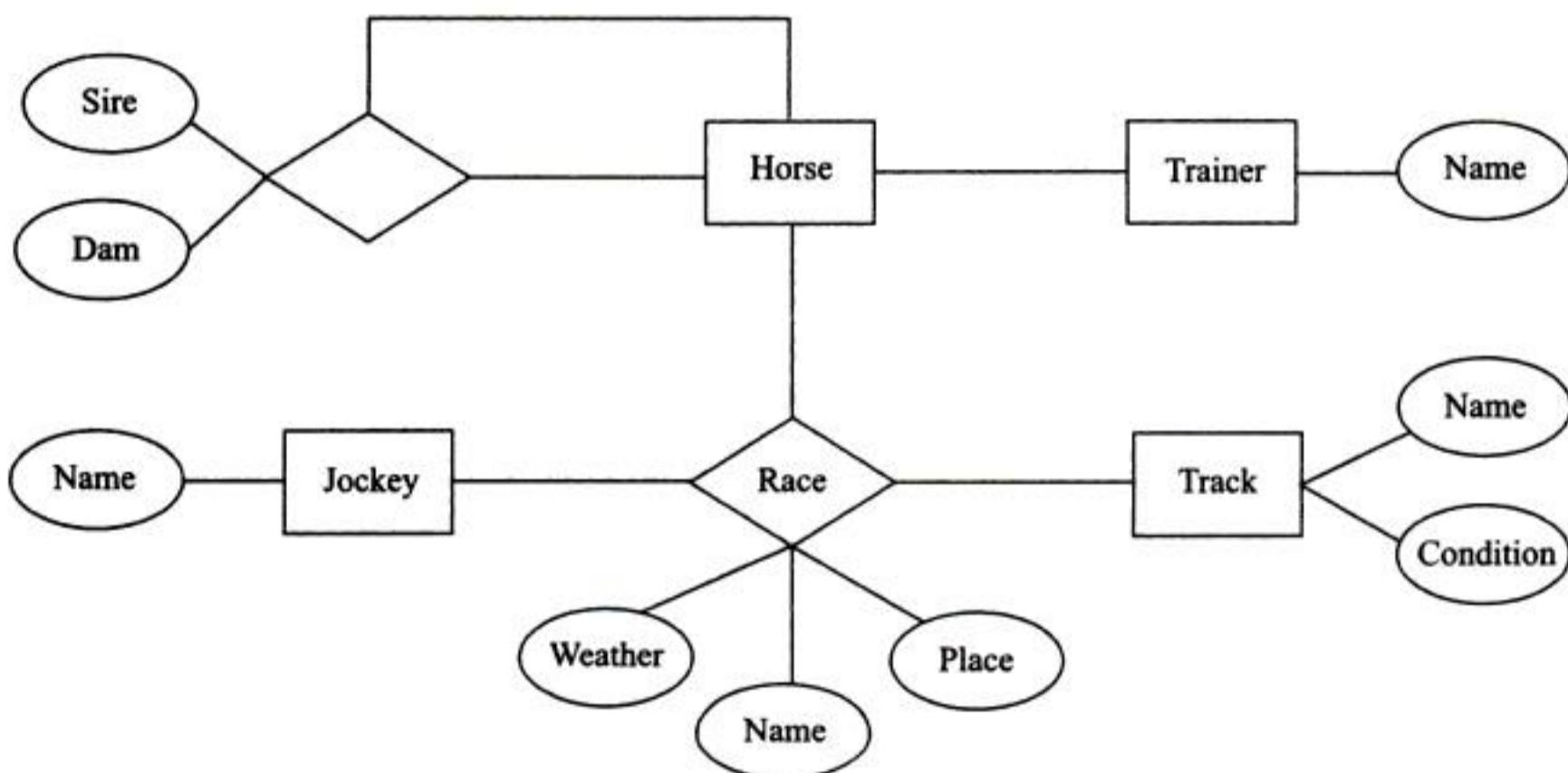


Figure 2.34

BIBLIOGRAPHY

For Students

The book by Batini, Ceri and Navathe and the book by Theorey are good introductions to E-R database modeling. The book by Baqui and Earp is also good. Most database textbooks have chapters on E-R modeling. The notes by Jung and Reddy are also worth consulting.

For Instructors

There are a number of good textbooks on database modeling. On Entity-Relationship modeling, the original 1976 paper by Chen is listed below. The books by Batini, Ceri and Navathe, by Thalheim and by Simsion are excellent. Halpin's book is different since it deals with the Object Relational Model (ORM) rather than with the E-R model and is worth reading if you want to look at another approach to database modeling. The paper by Herden looks at the quality of database models and presents an approach to measuring quality of a model. The papers by Burkett and Herden are good surveys of a number of approaches to database model quality.

- Baqui, S., and R. Earp, *Database Design Using Entity-Relationship Diagrams*, Auerbach, 2003, 264 pp.
- Batini, C., S. Ceri, and S.B. Navathe, *Conceptual Database Design: An Entity-Relationship Approach*, Benjamin Cummings, 1992.
- Burkett, W. C., "Database Schema Design Quality Principles". <http://wwwscf.usc.edu/wcb/dmq/dmqmain.html>, 1998.
- Chen, P. P., "The Entity-Relationship Model: Towards a unified view of data", *ACM Trans. Database Systems*, Vol. 1, pp 9-36, 1976.
- Chen, P. P., *Entity Relationship Approach to Logical Database Design*, Monograph Series, QED Information Sciences Inc., Wellesley, Ma, 1977.
- Chen, P. P. (Ed.), *Entity-Relationship to Systems Analysis and Design*, North-Holland, New York/Amsterdam, 1980.
- Elmasri, R., and S. B. Navathe, *Fundamentals of Database Systems*, Fifth Edition, Addison-Wesley, 2006.
- Halpin, T., *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design, using ORM with ER and UML*, Morgan Kaufmann, 2001.
- Herden, O., "Measuring quality of database schemas by reviewing – concept, criteria and tool", *Proceedings of the 5th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, Budapest (Ungarn), pp 59-70, 2001. <http://www.iro.umontreal.ca/~sahraouh/qaoose01/Herden.PDF>
- Hernandez, M. J., *Relational Design for Mere Mortals: A Hands-on Guide to Relational Database Design*, Second Edition, Addison-Wesley, 2003.
- Jung, A., and U. Reddy, "Entity-Relationship Modeling", School of Computer Science, University of Birmingham, 2009. <http://www.cs.bham.ac.uk/~udr/db2/handout2.pdf>
- Ng, P., "Further analysis of the entity-relationship approach to database design", *IEEE Transactions on Software Engineering*, Vol. DE-7, No. 1, pp 85-98, 1981.
- Simsion, G.C., and G. C. Witt, *Data Modeling Essential*, Third Edition, Morgan Kaufmann, 2005.
- Spaccapietra, S. (Ed.), *Entity-Relationship Approach: Ten Years of Experience in Information Modelling*, Proceedings of the Entity-Relationship Conference, North-Holland, 1987.
- Song, I. Y., M. Evans, and E. K. Park, "A comparative analysis of entity-relationship diagrams", *Journal of Computer and Software Engineering*, Vol. 3, No. 4, 1995, pp 427-459.
- Theorey, T. J., *Database Modelling and Design: Logical Design*, Fourth Edition, Morgan Kaufmann, 1999.
- Theorey, T. J., D. Yang, and J. P. Fry, "Logical Design Methodology for Relational Databases", *ACM Computing Surveys*, pp 197-222, June 1986.
- Thalheim, B., *Entity-Relationship Modelling*, Springer, 2000.
- The University of Texas at Austin, "Introduction to Database Modelling", *Information Technology Services*. <http://www.utexas.edu/its/database/datamodeling>.

Peter P. S. Chen



Peter Chen is the creator of the Entity-Relationship model (E-R model), which serves as the foundation of many systems analysis, design methodologies, and computer-aided software engineering (CASE) tools. His first paper on the E-R model was published in 1976 and was included in the '38 most influential papers in Computer Science' by a survey of over 1,000 computer scientists.

Currently, Professor Chen is a distinguished Chair Professor at NTHU, Taiwan. He held the position of M. J. Foster distinguished Chair Professor of Computer Science at Louisiana State University in the United States from 1983–2008.

Dr. Chen holds a PhD in Computer Science/Applied Mathematics from Harvard University.



Relational Model

3

OBJECTIVES

- Introduce the relational model and its three components.
- Discuss the data structure used in the relational model.
- Define the properties of a relation and explain relational terminology.
- Describe how an Entity-Relationship model may be mapped to the relational model.
- Describe how information is retrieved and modified in the relational model.
- Explain how data integrity is maintained in the relational model.
- Discuss the advantages and disadvantages of using the relational model.
- Discuss the older database models, the network and hierarchical models.

KEYWORDS

Relational model, mapping E-R model, relations, tables, atomic domain, domains, tuples, cardinality, degree, attributes, rows, tables, columns, schema, constraints, candidate key, primary key, foreign key, existential integrity, referential integrity, domains, nulls, integrity, hierarchical model, network model.

Earth provides enough to satisfy every man's need, but not every man's greed.

Mahatma Gandhi

3.1 INTRODUCTION

In the last chapter we discussed the Entity-Relationship (E-R) model, a technique for building a logical model of an enterprise. Logical models are high level abstract views of an enterprise data. In building these models little thought is given to representing the database in the computer or retrieving data from it. At the next lower level, a number of data models are available that provide a mapping for a logical data model like the E-R model. These models specify a conceptual view of the data as well as a high level view of the implementation of the database on the computer. They do not concern themselves with the detailed bits and bytes view of the data.

In the 1960s, there were two widely used data models at this intermediate level. These were the *hierarchical model* and the *network model*. These are briefly discussed in Section 3.7. Readers interested in a more detailed discussion on the hierarchical and network models should refer to some of the older database books. As noted in the first chapter, a new model called the relational model was proposed by E. F. Codd in 1970. The model is based on the concept of *mathematical relation* which looks like a table of values. In the beginning the computing industry was sceptical of the relational approach; many computing professionals thought the approach could not be efficient for managing large databases. This however changed as prototypes of relational database systems were built and shown to be efficient. The first two prototypes were the Model R at IBM San Jose Research Lab in California and INGRES at the University of California at Berkeley. With the success of the prototypes, a number of commercial relational database management systems became available and the hierarchical and network models were superseded by the relational model. Some computing installations however continue to use the old DBMS to process their legacy data. During the last 30 years or so, a number of other new models have also been proposed. Some are based on an object oriented approach to viewing the data, others consider an object relational approach while the development of XML itself is leading to another way of looking at databases. These topics will be discussed in Chapters 14 and 17 respectively. The main focus of this chapter is the relational model.

All the data models provide facilities for representing entities and their attributes as well as relationships. The data about entities and their attributes is handled in a similar way in all the models. The representation is a record which is a collection of attribute values (including an identifier of the entity) although the structure of the record may be somewhat different. The relationships are however represented very differently in the models. We start by looking at the simplest model, i.e., the relational model.

One might ask the question why we should bother about the relational model when we already have the E-R model. In some ways, the two models are quite different. Using the E-R model allows us to define the structure of the enterprise data more easily than using the relational model but the simplicity of the relational model allows powerful query languages to be designed that could not be employed with the E-R model. Essentially the E-R model is used to build a conceptual view of the database that is implemented using the relational model.

This chapter is organized as follows. Section 3.2 deals with the data structure used in the relational model. Mapping of the E-R model to the relational model is described here. Data manipulation is briefly discussed since it is discussed in detail in Chapter 4. The final component of the relational model, data integrity is discussed in Section 3.4 including existential integrity, referential integrity, domains and nulls. Advantages of the relational model are presented in Section 3.5. Codd's rules for fully relational systems are then listed in Section 3.6 followed by a brief description of older database models; hierarchical and network models in Section 3.7.

3.1.1 The Relational Model

As noted earlier, the relational model was proposed by Codd in 1970. The model deals with database management from an abstract point of view. It provides specifications of an abstract database management system. The model is very easy to understand although it is based on a solid theoretical foundation that includes predicate calculus and the theory of relations. However, to use a database management system based on the relational model, users do not need to master these theoretical foundations.

Codd defined the relational model as consisting of the following three components:

1. *Data Structure*—A collection of data structure types for building a database.
2. *Data Manipulation*—A collection of operators that may be used to retrieve, derive, build a database or modify data stored in the data structures.
3. *Data Integrity*—A collection of rules that implicitly or explicitly define a consistent database state or change of state.

Each of the three components are discussed in Sections 3.2, 3.3 and 3.4 respectively.

3.2 DATA STRUCTURE

The beauty of the relational model is the simplicity of its data structure. The fundamental property of the relational structure is that all information about the entities and their attributes as well as the relationships is presented to the user as tables (called *relations*¹) and nothing but tables. A database therefore is a collection of relations. Each row of each table consists of an entity occurrence or a relationship occurrence. Each column refers to an attribute. The model is called relational and not tabular because tables are a lower level of abstraction than the mathematical concept of relation. Tables give the impression that positions of rows and columns are important. In the relational model, it is assumed that no ordering of rows and columns is defined.

Figure 3.1 gives number of properties of a relation:

The properties listed in Fig. 3.1 are simple and based on practical considerations. The first property ensures that only one type of information is stored in each relation. The second property ensures that each column is named uniquely. This has several benefits. The names can be chosen to convey what each column is and the

1. We will use the two terms *table* and *relation* interchangeably.

- Each relation contains only one record type.
- Each relation has a fixed number of columns (attributes) that are explicitly named. Each attribute name within a relation is unique.
- No two rows (tuples) in a relation are the same.
- Each item or element in the relation is atomic, that is, in each row, every attribute has only one value that cannot be decomposed into smaller components.
- Rows have no ordering associated with them since a relation is a set of rows and sets have no ordering associated with them.
- Columns have no ordering associated with them (although most commercially available systems do) since each column has a unique name and that column's value in each row can be easily identified by the column name.

Figure 3.1 Properties of a relation in the relational model

names enable one to distinguish between the column and its domain. Furthermore, the names are much easier to remember than the position of each column if the number of columns is large.

The third property of not having duplicate rows appears obvious but is not always accepted by all DBMS users and designers. The property is essential since no sensible context free meaning can be assigned to a number of rows that are exactly the same.

The fourth property requires that each element in each relation be atomic and that it cannot be decomposed into smaller elements. For example, if an attribute value is 1976–90 then it is not possible to look at just the first part of the value (i.e., 1976). In the relational model, the only composite or compound type (data that can be decomposed into smaller pieces) is a table. This simplicity of structure leads to relatively simple database languages.

The fifth property deals with ordering of the rows in a relation. A relation is a set of rows. As in other types of sets, there is no ordering associated with the rows in a table. Just like the second property, the ordering of the rows should not be significant since in a large database no user should be expected to have any understanding of the positions of the rows of a table. Also, in some situations, the DBMS should be permitted to reorganize the data and change row orderings if that is necessary for efficiency reasons. The columns are identified by their names. The rows on the other hand are identified by their contents, possibly the attributes that are unique for each row.

A table is a set of rows and is closely related to the concept of relation in mathematics. Each row in a relation may be viewed as an assertion. For example, a row in a relation *Player* may assert that a player by the name of *Sachin Tendulkar* has *player_id 89001* and represents *India*. Similarly, a row in a table *Match* may assert that one of the ODI matches was held on *2 March 2008* in *Sydney*. The tables in our ODI database will be presented a little later in this section.

As noted earlier, in the relational model, a relation is the only data structure. The relational terminology is defined in Fig. 3.2.

- *Relation* — It is essentially a simple table. It may be defined as set of rows (or tuples), each row therefore has the same columns (or attributes). Since a relation is a sort of tuple, there is no ordering associated with the tuples. (As noted earlier, we will continue to use both terms relation and table to mean the same).
- *Tuple* — It is a row in the relation (we will use both the terms tuple and row to mean the same).
- *Attribute* — It is a column in the relation.
- *Degree of a relation* — It is the number of columns in the relation.
- *Cardinality of a relation* — It is the number of rows in the relation.
- *N-ary relation* — It is a relation with degree N .
- *Domain* — It is a set of atomic (that is, indivisible) values that each element in a column is permitted to take. Domains may be given unique names. The same domain may be used by a number of different columns.
- *Candidate key and Primary key* — It has been discussed in the last chapter and defined below.

Figure 3.2 Relational Terminology

A formal definition of candidate key is given below:

Definition—Candidate Key

An attribute (or a set of attributes) is called a candidate key of the relation if it satisfies the following properties:

- (a) the attribute or the set of attributes uniquely identifies each tuple in the relation (called the uniqueness property), and
- (b) if the key is a set of attributes then no subset of these attributes has the property (a) (called the minimality property).

There may be several distinct sets of attributes that may serve as candidate keys but every table must always have at least one.

Primary key may be defined as follows:

Definition—Primary Key

One (and only one) of the candidate keys is arbitrarily chosen as the primary key of the table. Primary key therefore has the properties of uniqueness and minimality.

We will often use symbols like R or S to denote a table and r and s to denote a row. When we write $r \subset R$, we are stating that row r is a row in table R .

3.3 MAPPING THE E-R MODEL TO THE RELATIONAL MODEL

In the relational model, information about entities and relationships is represented in the same way, that is, by tables. Since the structure of all information is the same, the same operators may be applied to them.

We consider the following ODI cricket database, developed in the last chapter, to illustrate the mapping of an E-R model to the relational model as well as basic concepts of the relational data model. We first reproduce the E-R diagram for the database as shown in Fig. 3.3.

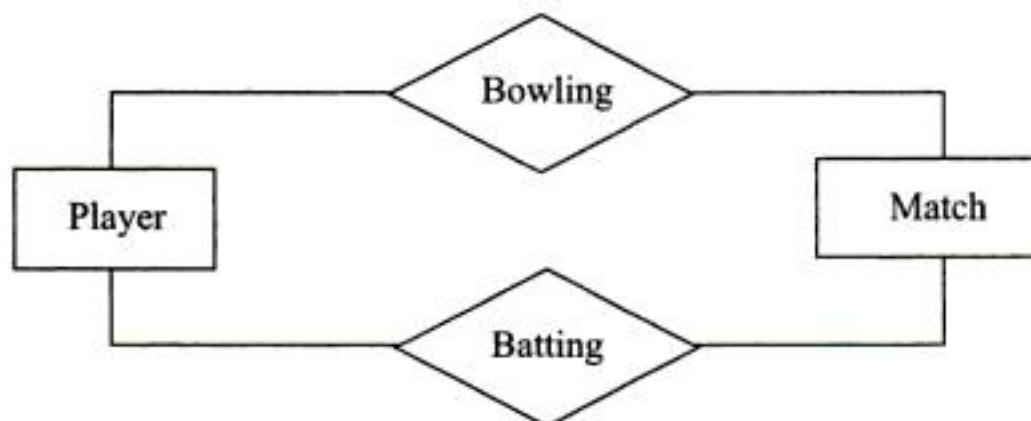


Figure 3.3 E-R diagram for ODI matches

We have not included the attributes in the E-R diagram in Fig. 3.3. The attributes are given below. Some of the attribute names have been changed so that there are no spaces in the names and the names are shorter for ease of display.

The entity *Match* has the attributes given in Fig. 3.4 (its primary key is *Match ID*). We have made some minor changes to the attribute names used in the last chapter to make the names a bit more meaningful.

We have changed the attribute names *Home Team* and *Visiting Team* to *Team1* and *Team2* respectively because many ODI matches are being held in third countries, for example, India and South Africa played on 1st July 2007 in Belfast. We will not discuss such matches in the examples that follow. We will also change the names of attributes to remove spaces within names that are not acceptable in SQL and will shorten several names and replace some by names used by the cricinfo (www.cricinfo.com) website, for example, <http://contentaus.cricinfo.com/engvind/engine/series/258449.html> to make it easier to use them in the examples that follow.

The entity *Player* has its attributes listed in Fig. 3.5 (The primary key is *Player ID*):

MatchID
Team1
Team2
Ground
Date
Winner

Figure 3.4 The attributes for the entity *Match*

PlayerID
LName
FName
Country
YBorn
BPlace
FTest

Figure 3.5 The attributes for the entity *Player*

The relationship *Batting* has its attributes listed in Fig. 3.6 (The primary key is *Match ID* and *PID* together).

MatchID
PID
Order
HOut
FOW
NRuns
Mts
NBalls
Fours
Sixes

Figure 3.6 The attributes for the relationship *Batting*

The relationship *Bowling* entity has its attributes listed in Fig. 3.7 (The primary key is *MatchID* and *PID* together).

MatchID
PID
NOvers
Maidens
NRuns
NWickets

Figure 3.7 The attributes for the relationship *Bowling*

Now that we have all the details of the E-R model, we consider mapping it to the relational model. The mapping is quite straightforward since the relational model closely approximates the E-R Model. We will show this by transforming the E-R model in Fig. 3.3 (including the attribute lists in Figs 3.4 to 3.7) into a set of relations.

We will follow the steps given below for mapping an E-R diagram into a relational model.

3.3.1 Step 1—Mapping Strong Entities

Each strong entity in an E-R model is represented by a table in the relational model. Each attribute of the entity becomes a column of the table. For example, the entities *Player* and *Match* in Fig. 3.3 may be directly transformed into two tables of the same names given in Tables 3.1 and 3.2 respectively. We have populated these tables using real ODI data. We have only selected matches in which India has played. Note that in almost all entries one of the teams, either *Team 1* or *Team 2*, is the winner but other results are possible including abandoned, no result, draw or cancelled.

Table 3.1 below is the representation of the entity *Match* in the E-R model of Fig. 3.3.

Table 3.1 The relation Match

<i>MatchID</i>	<i>Team 1</i>	<i>Team 2</i>	<i>Ground</i>	<i>Date</i>	<i>Winner</i>
2324	Pakistan	India	Peshawar	6/2/2006	Team 1
2327	Pakistan	India	Rawalpindi	11/2/2006	Team 2
2357	India	England	Delhi	28/3/2006	Team 1
2377	West Indies	India	Kingston	18/5/2006	Team 2
2404a	Sri Lanka	India	Colombo	16/8/2006	Abandoned
2440	India	Australia	Mohali	29/10/2006	Team 2
2449	South Africa	India	Cape Town	26/11/2006	Team 1
2480	India	West Indies	Nagpur	21/1/2007	Team 1
2493	India	West Indies	Vadodara	31/1/2007	Team 1
2520	India	Sri Lanka	Rajkot	11/2/2007	Team 2
2611	England	India	Southampton	21/8/2007	Team 1
2632	India	Australia	Mumbai	17/10/2007	Team 1
2643	India	Pakistan	Guwahati	5/11/2007	Team 1
2675	Australia	India	Melbourne	10/2/2008	Team 2
2681	India	Sri Lanka	Adelaide	19/2/2008	Team 1
2688	Australia	India	Sydney	2/3/2008	Team 2
2689	Australia	India	Brisbane	4/3/2008	Team 2
2717	Pakistan	India	Karachi	26/6/2008	Team 2
2750	Sri Lanka	India	Colombo	24/8/2008	Team 2
2755	Sri Lanka	India	Colombo	27/8/2008	Team 2

Table 3.2 below is the representation of the entity *Player* in E-R model of Fig. 3.3.

Table 3.2 The relation Player

<i>PlayerID</i>	<i>LName</i>	<i>FName</i>	<i>Country</i>	<i>YBorn</i>	<i>BPlace</i>	<i>FTest</i>
89001	Tendulkar	Sachin	India	1973	Mumbai	1989
90001	Lara	Brian	West Indies	1969	Santa Cruz	1990
95001	Ponting	Ricky	Australia	1974	Launceston	1995
96001	Dravid	Rahul	India	1973	Indore	1996
96002	Gibbs	Herschelle	South Africa	1974	Cape Town	1996
92001	Warne	Shane	Australia	1969	Melbourne	1992
95002	Pollock	Shaun	South Africa	1973	Port Elizabeth	1995
99003	Vaughan	Michael	England	1974	Manchester	1999

Contd.

Table 3.2 Contd.

PlayerID	LName	FName	Country	YBorn	BPlace	FTest
92003	Ul-Huq	Inzamam	Pakistan	1970	Multan	1992
94004	Fleming	Stephen	New Zealand	1973	Christchurch	1994
93002	Streak	Heath	Zimbabwe	1974	Bulawayo	1993
90002	Kumble	Anil	India	1970	Bangalore	1990
93003	Kirsten	Gary	South Africa	1967	Cape Town	1993
95003	Kallis	Jacques	South Africa	1975	Cape Town	1995
94002	Vaas	Chaminda	Sri Lanka	1974	Mattumagala	1994
92002	Muralitharan	Muthiah	Sri Lanka	1972	Kandy	1992
97004	Vettori	Daniel	New Zealand	1979	Auckland	1997
25001	Dhoni	M S	India	1981	Ranchi	2005
23001	Singh	Yuvraj	India	1981	Chandigarh	2003
96003	Ganguly	Saurav	India	1972	Calcutta	1996
99002	Gilchrist	Adam	Australia	1971	Bellingen	1999
24001	Symonds	Andrew	Australia	1975	Birmingham	2004
99001	Lee	Brett	Australia	1976	Wollongong	1999
91001	Jayasuriya	Sanath	Sri Lanka	1969	Matara	1991
21001	Sehwag	Virender	India	1978	Delhi	2001
98001	Afridi	Shahid	Pakistan	1980	Khyber Agency	1998
98002	Singh	Harbhajan	India	1980	Jalandhar	1998
27001	Kumar	Praveen	India	1986	Meerut	NA
27002	Sharma	Ishant	India	1988	Delhi	2007

3.3.2 Step 2-Mapping Weak Entities

Although Fig. 3.3 E-R diagram does not include a weak entity, we must deal with weak entities if they are in the model. Each weak entity may be mapped to a relation that includes all simple attributes of the entity as well as the primary key of the dominant entity. The primary key of the relation representing a weak entity will be the combination of the primary key of the dominant entity and the partial key of the weak entity.

3.3.3 Step 3-Mapping Relationships

As noted in Chapter 2, there can be three types of relationships, viz., 1:1, 1:m and m:n. We deal with each of them.

Step 3(a)-Mapping-One-to-one Binary Relationships

Figure 3.3 does not include any 1:1 relationships but some situations are likely to give rise to such relationships.

We should note that although each relationship may be mapped to a table in the relational model, not all relationship types need to be represented by separate tables. For 1:1 key propagation may be used. In a 1:1 relationship, each instance of the first entity in a relationship is only related to one instance of the other entity. Therefore it is straightforward to represent the relationship by either putting the primary key of the first entity in the second entity or the primary key of the second entity in the first. This is called *key propagation*.

Such situations of 1:1 relationship may occur in *Department* and *Manager* entities since each department must have a manager and each manager must manage one department. We may choose one of the entities say entity *Department* which should already been mapped into a relation and include the primary key of the other entity, *Manager* (which would also have been mapped to a relation) in the *Department* relation. This is then a foreign key in the relation (that represents *Department*). If the relationship has simple attributes, those may also be included in this relation that represents *Department*.

In some situations when mapping 1:1 relationship between entities *Department* and *Manager*, both entities and the relationship may be merged into one and represented by one relation. As noted earlier, the other extreme solution of course is to represent the relationship as a table that includes primary keys of both entities as well as all the attributes of the relationship.

Step 3(b)–Mapping One-to-many Relationships

Figure 3.3 does not include any 1:*m* relationships but many situations give rise to such relationships.

In a 1:*m* relationship, one entity instance of the first entity may be related to a number of instances of the second entity but each instance of the second entity is related to only one instance of the first entity. This may be the case if we had entities *Employee* and *Manager*. Key propagation is therefore also possible in 1:*m* relationships. We cannot put the primary key of the related instances of *Employee* in the entity *Manager* because for this instance of the entity *Manager* there are likely to be a number of related instances of the *Employee*. We can however propagate the primary key of *Manager* to the related instance because each instance of *Employee* is related to only one instance of *Manager*. The primary key of *Manager* in *Employee* then becomes a foreign key in the relation *Employee*. If the relationship has simple attributes, those may also be included in the relation *Employee*.

Step 3(c)–Mapping Many-to-many Relationships

Key propagation is not possible for *m:n* relationships because it is not possible to propagate primary keys of instances of a related second entity to an instance of the related first entity, since there may be more than one such instance of the second entity. Similarly, it is not possible to propagate primary keys of a related first entity to the related second entity since there may be more than one such instance of the first entity. Many-to-many relationships therefore must be represented by a separate table.

Therefore each binary *m:n* relationship must be represented by a new relation. The primary keys of each of the entities that participates in the relationship are included in this relation. The combination of these keys will make up the primary key for the relation. Also include any attributes of the relationship in this relation.

For example, when the relationship *Batting* in Fig. 3.3 is transformed into Table 3.3 *Batting*, the primary key of the table is the combination of the primary keys of the two entities *Match* and *Player*. Similarly when *Bowling* in Fig. 3.3 is transformed into Table 3.4 *Bowling*, the primary key of the table is the combination of

the primary keys of the two entities *Match* and *Player*. The primary keys of both tables *Batting* and *Bowling* therefore are the same which does not create any problems.

The relationships *Batting* and *Bowling* are both many-to-many relationships since each batsman and each bowler plays in more than one match and each match has more than one batsman and more than one bowler. We therefore transform these two relationships into Table 3.3 and Table 3.4 respectively.

We have again populated the two tables, Table 3.3 and Table 3.4, with real ODI data. The data in both these tables is very limited given the space limitations. We have included part of the batting and bowling information from only two ODI matches, one between Australia and India in Brisbane in March 2008 (Match ID 2689) and the other between Sri Lanka and India played in Colombo on 27th August 2008 (Match ID 2755). India won both these matches. More data for these tables is presented in the appendix at the end of Chapter 5. This data may be useful for some examples and exercises in Chapters 4 and 5.

The relationship *Batting* is mapped to the relational model as shown in Table 3.3.

Table 3.3 The relation Batting

MatchID	PID	Order	HOut	FOW	NRuns	Mts	NBalls	Fours	Sixes
2755	23001	3	C	51	0	12	6	0	0
2755	25001	5	C	232	71	104	80	4	0
2755	91001	1	C	74	60	85	52	8	2
2755	94002	7	LBW	157	17	23	29	1	0
2755	92002	11	NO	NO	1	11	7	0	0
2689	89001	2	C	205	91	176	121	7	0
2689	23001	4	C	175	38	27	38	2	2
2689	25001	5	C	240	36	52	37	2	1
2689	99002	1	C	2	2	3	3	0	0
2689	95001	3	C	8	1	9	7	0	0
2689	24001	5	C	123	42	81	56	2	1
2689	99001	8	B	228	7	20	12	0	0
2689	27001	9	C	255	7	7	7	1	0
2755	27001	9	B	257	2	7	6	0	0
2689	98002	8	LBW	249	3	7	3	0	0
2755	98002	8	RO	253	2	7	4	0	0

The relationship *Bowling* in the E-R model given in Fig. 3.3 is mapped to the relational model in Table 3.4.

The above database schema may also be written as in Fig. 3.8.

The E-R diagram in Fig. 3.3 has now been mapped into the relational schema in Fig. 3.8 consisting of four tables. Each table schema presents the structure of the table by specifying its name and the names of its attributes enclosed in parenthesis. The primary key of a relation may be marked by underlining although we have not done so in Fig. 3.8.

Table 3.4 The relation *Bowling*

<i>MatchID</i>	<i>PID</i>	<i>NOvers</i>	<i>Maidens</i>	<i>NRuns</i>	<i>NWickets</i>
2689	99001	10	0	58	1
2689	24001	3	0	27	1
2689	23001	3	0	15	0
2755	94002	9	1	40	1
2755	92002	10	0	56	1
2755	91001	4	0	29	0
2755	23001	10	0	53	2
2755	98002	10	0	40	3
2689	98002	10	0	44	1

Match(*MatchID*, Team1, Team2, Ground, Date, Winner)
 Player(*PlayerID*, LName, Country, YBorn BPlace FTest)
 Batting(*MatchID*, *PID*, Order, HOut, FOW, NRuns, Mts, NBalls, Fours, Sixes)
 Bowling(*MatchID*, *PID*, *NOvers*, Maidens, NRuns, NWickets)

Figure 3.8 Database schema for the ODI cricket database

Now to summarize the mapped relational model, we note that the four tables, Table 3.1 to Table 3.4, *Match*, *Player*, *Batting* and *Bowling*, have degree 6, 7, 10 and 6 respectively and cardinality 20, 29, 14 and 7 respectively. The primary key of the table *Match* is *MatchID*, of table *Player* is *PlayerID*, of table *Batting* is (*MatchID*, *PID*) and finally the primary key of the table *Bowling* is (*MatchID*, *PID*). The table *Match* has another candidate key (Ground, Date) as well one or two others. If we can assume the names to be unique then *Player* has (LName, FName) as a candidate key. If the names are not unique but the names LName, FName and FTest together are unique, then the three attributes together form a candidate key. Note that both *PlayerID* and (*PlayerID*, Country) cannot be candidate keys, only *PlayerID* can.

3.3.4 Key Propagation

We further illustrate the use of key propagation by considering the simple E-R model in Fig. 3.9.

**Figure 3.9** A simple database

Let the relationship *Occupies* be many-to-one, that is, one employee may occupy only one office but a room may be occupied by more than one employee. Let the attributes of the entity *Employee* be employee number (*e_num*), name (*name*) and *status*. Attributes of the entity *Office* are office number (*o_num*), *capacity* and

building. *Occupies* has attributes *date*, *e_num* and *o_num* that include the primary keys of the two entities *Employee* and *Office*. One way to represent the above database is to have the three tables whose schema is given in Fig. 3.10.

<i>Employee(e_num, name, status)</i>
<i>Occupies(e_num, o_num, date)</i>
<i>Office(o_num, capacity, building)</i>

Figure 3.10 Three relations representing the E-R diagram in Fig. 3.9

There is of course another possibility. Since the relationship *Occupies* is many-to-one, that is, each employee occupies only one office, we can represent the relationship by including the primary key of the table *Office* in the entity *Employee*. This is propagation of the key of the entity *Office* to entity *Employee*. We then obtain the database schema given in Fig. 3.11.

<i>Employee(e_num, name, status, o_num, date)</i>
<i>Office(o_num, capacity, building)</i>

Figure 3.11 Two relations representing the E-R diagram in Fig. 3.9

Note that key propagation would not work if the relationship *Occupies* is many-to-many.

As noted earlier, if the relationship *Occupies* is 1:1, key propagation is possible in both directions.

3.3.5 A More Complex Example of Mapping an E-R Diagram

We now look at a larger example. An E-R diagram for a programming database project was presented in the last chapter. We reproduce the E-R diagram in Fig. 3.12. Although it looks quite complex, it is not really as complex as one would find for a relatively small real-life database. For a database of a large company, the E-R diagram might even cover the whole wall of a large room!

We now try to map the diagram in Fig. 3.12 into the relational model using the method we have already described. We first look at all the entities and then at all the relationships.

Step 1—Strong Entities

There are five strong entities according to the Entity-Relationship diagram in the figure. These five entities are directly converted into relations as below. The relations are shown with their attributes. Note that different entities may have the same attribute names since this does not create any confusion.

1. User (FName, LName, Login_Name, Address)
2. Program (PName, Code, Date, Version, Extension, Description, Language)
3. Programmer (Login, Code, FName, LName, Address)
4. DBMS (DB_name)
5. Relation (RName, PKey)

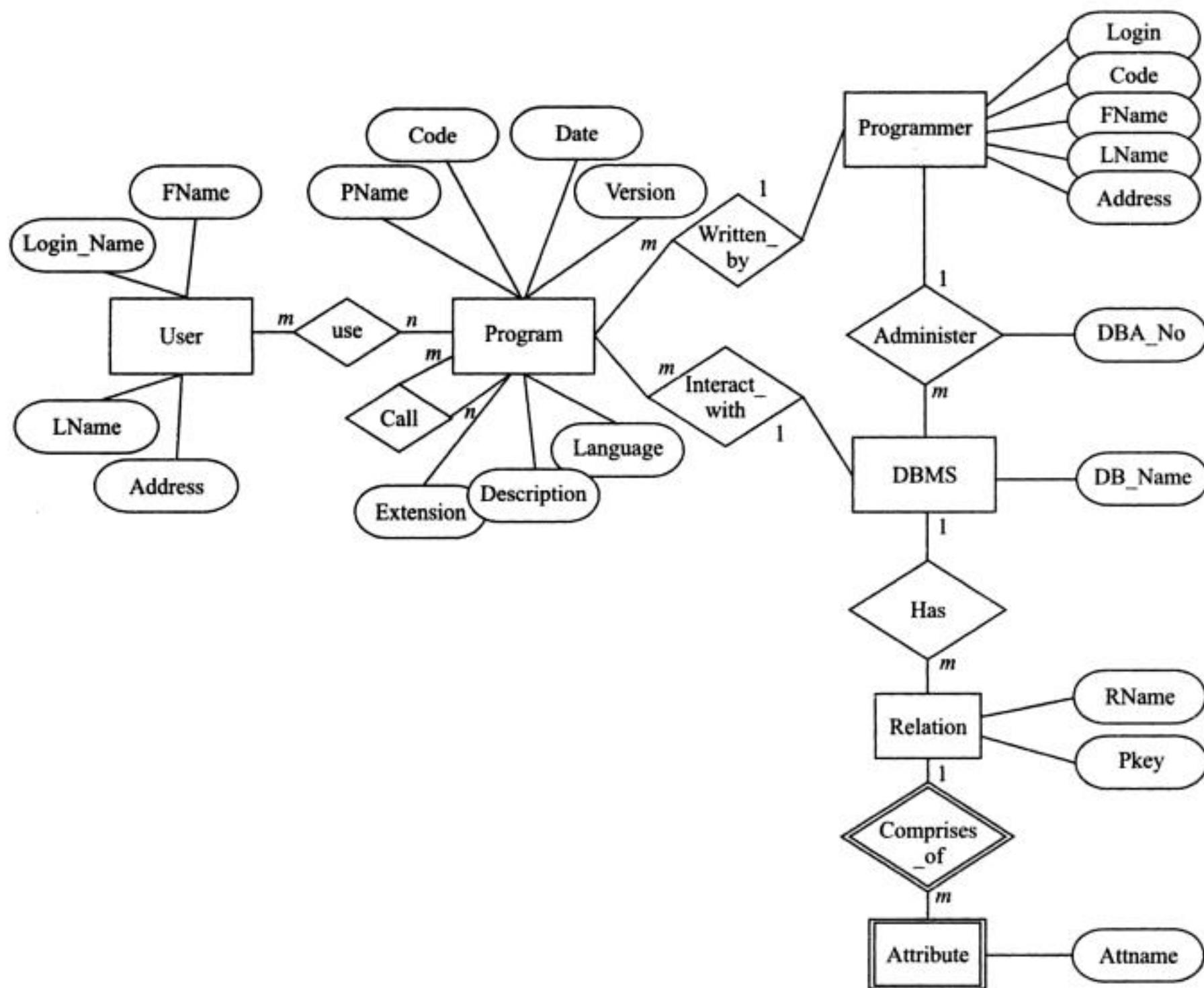


Figure 3.12 An E-R Diagram for a programming database

Step 2–Weak Entities

We do have a weak entity called attribute since it depends on there being a relation whose attributes this entity represents. Note that the primary key of the dominant entity is in this relation.

6. Attribute (RName, Attname)

Step 3(a)–One-to-one Relationships

There are no 1:1 relationships in this model

Step 3(b)–One-to-many Relationships

Two solutions are possible for each of these relationships. One possibility is key propagation. The other possibility is to map each relationship into a new relation.

- (i) Either represent the relationship Written_by using key propagation which involves moving primary key of programme (*Login*) to *Program* or the relationship may be mapped to a new relation that has

primary key of both entities that it joins; Written_by (PName, Login). There is a clear assumption here that a program has only one programmer.

- (ii) Either represent the relationship Interact_with by key propagation which involves moving DB_name to Program or the relationship may be mapped to a new relation that has primary key of both entities that it joins Interact_with (PName, DB_name)
- (iii) Either represent the relation Administer by key propagation which involves moving Login to DBMS or the relationship may be mapped to a new relation that has primary key of both entities that it joins Administer (Login, DB_name). There is a clear assumption here that each DBMS has only one administrator.

Therefore we have the following three relations if we do not use key propagation:

7. Written_by (PName, Login)
8. Interact_with (PName, DB_name)
9. Administer (Login, DB_name)

Step 3(c)-Many-to-many Relationships

Now we deal with the four $m:n$ relationships. All such relationships must be mapped to new tables. These tables include primary keys of both joining relations as well as any attributes that the relationship might have.

10. Use (Login_name, PName)
11. Call (PName1, PName2)
12. Has (DB_name, Relation, RName, PKey)
13. Comprises_of (RName, Atname)

This E-R model therefore translates to 13 tables in the relational model.

Once the E-R diagram has been mapped into relational tables, the tables should be created using the CREATE TABLE command discussed in Chapter 5 but an important task remains before the tables may be created. For each attribute we need to determine the attribute types.

3.4 DATA MANIPULATION

The data manipulation part of the relational model makes set processing facilities available to the user. Since relational operators are able to manipulate tables, the application programs do not need to use loops. For example, to read one row and then the next and then the one after until there are no more rows left. Avoiding loops can result in a significant increase in the productivity of application programmers.

The primary purpose of a database in an enterprise is to be able to provide information to the users within the enterprise. The process of querying a relational database is in essence a way of manipulating the tables that are the database. For example, in the cricket database presented in Tables 3.1 to 3.4, one may wish to find the names of all players from India, or a list of all the matches played by Brian Lara.

We will show how such queries may be answered by using data manipulation languages that are available for relational databases. Two formal languages were presented by Codd when he proposed the relational

model. These are called *relational algebra* and *relational calculus*. They are discussed in the next chapter, Chapter 4. These formal languages are followed by a detailed discussion of the commercial query language SQL in Chapter 5.

3.5 DATA INTEGRITY

We noted at the beginning of the chapter that the relational model has three main components: data structure, data manipulation and data integrity. The aim of data integrity is to specify rules that implicitly or explicitly define a consistent database state or change of state. These rules may include facilities like those provided by most programming languages for declaring data types which constrain the user from operations like comparing data of different data types and assigning a variable of one type to another of a different type. This is done to stop the user from doing things that generally do not make sense and to help in ensuring that the data in a database is consistent. In a DBMS, integrity constraints play a similar role.

The integrity constraints are necessary to avoid situations like the following:

1. Some data has been inserted in the database but it cannot be identified (that is, it is not clear which object or entity the data is about).
2. A player is shown to have played in a match but no data about him is available in the table *Player* that has information about players.
3. During processing a query, a player ID is compared with a match ID (this should never be required).
4. A player has retired from cricket but is still shown to be playing in a match.
5. In the table *Batting*, a player is shown to be batting at order greater than 11.
6. In the table *Batting* a player's FOW is lower than the number of runs he scored.
7. In the table *Bowling* a bowler has bowled more than 10 overs in an ODI.

We may define an integrity constraint as a condition specified on a database that restricts the data that can be stored in the database. A database system then enforces these constraints. Many different kinds of constraints may be defined on a database to overcome problems like those listed above. We discuss some constraints in this section and others in Chapter 11.

First we note that integrity constraints on a database may be divided into the following two types:

1. *Static Integrity Constraints*—These are constraints that define valid states of the data. These constraints include designations of primary keys.
2. *Dynamic Integrity Constraints*—These are constraints that define side-effects of various kinds of transactions (for example, insertions and deletions) carried out in the database. Full discussion of such constraints is outside the scope of this book but we will discuss the SQL *trigger* operation and others in Chapter 11.

We now discuss certain static integrity features of the relational model. We discuss the following features:

1. Primary Keys
2. Foreign Keys and Referential Integrity
3. Domains
4. Nulls

3.5.1 Entity Integrity Constraint (or Primary Key Constraint)

We have earlier defined the concepts of candidate key and primary key. From the definition of a candidate key, it should be clear that each table must have at least one candidate key even if it is the combination of all the columns in the table, since all rows in a table are distinct. Often a table may have more than one candidate key.

As discussed earlier, the primary key of a table is an arbitrarily but permanently selected candidate key. The primary key is important since it is the sole identifier for the rows in a table. Any row in a database may be identified by specifying the table name, the primary key and its value. Also for a row to exist in a table it must be *identifiable* and therefore it must have a primary key value. After all, each relation has information about real world things and therefore each row must be identifiable. Also no two rows in the table *Player* can have the same value of the primary key of the table. Furthermore, primary key values serve as references to the rows that they identify. Given the fundamental importance of the primary keys, the database design should always identify the primary key of a table.

Not having a primary key value specified is equivalent to saying that some entity has no identity. This is not acceptable.

We have earlier defined a candidate key of a relation as an attribute (or a set of attributes) if it satisfies the properties in Fig. 3.13.

- The attribute or the set of attributes uniquely identifies each tuple in the relation (called the *uniqueness* property).
- If the key is a set of attributes then no subset of these attributes has the uniqueness property (called the *minimality* property).

Figure 3.13 Essential properties of a candidate key

The first of these two properties of a candidate key (and therefore of the primary key) of a table requires that each key value uniquely identify only one row in the table. For example, more than one player may have the same name but they must have a unique player ID.

The second property requires that no collection of attributes that include the primary key may be called a key in spite of that collection being able to uniquely identify each tuple since they include the primary key which by itself can uniquely identify each row.

As a result of the candidate key properties listed in Fig. 3.13, the relational data model imposes the integrity constraints given in Fig. 3.14 on primary keys.

- No component of a primary key value can be null.
- No two rows can have the same primary key.
- Attempts to change the value of a primary key must be carefully controlled.

Figure 3.14 Integrity constraints for primary keys

The first constraint is necessary because when we want to store information about some entity, then we must be able to identify it to avoid difficulties. For example, if a table *Batting* has (MatchID, PID) as the primary key then allowing rows like those shown in Table 3.5 is going to lead to ambiguity since two rows

in the table may or may not be for the same player or for the same match and the integrity of the database is therefore compromised. Having NULLS for attributes other than the primary key attributes does not create any difficulties although one might find the missing information odd in this case.

Table 3.5 Problems with NULLs in primary key

MatchID	PID	NOvers	Maidens	NRuns	NWickets
2689	NULL	10	NULL	58	1
NULL	24001	NULL	0	27	1
2689	23001	3	0	15	0

The problem identified above may be overcome by SQL commands. For example, we consider the following CREATE TABLE command in SQL.

```
CREATE TABLE Match
  (MatchID INTEGER,
   Team1 CHAR(15),
   Team2 CHAR(15),
   Ground CHAR(20),
   Date CHAR(10),
   Result CHAR(10)
  UNIQUE (MatchID)
  CONSTRAINT MatchPK PRIMARY KEY (MatchID))
```

Figure 3.15 Applying a primary key constraint in SQL

Such commands are discussed in more detail in Chapter 5 and Chapter 12.

The second constraint deals with changing of primary key values. Since the primary key is the identifier of an entity instance, changing it needs very careful controls. For example, consider the information in the fragment of table *Bowling* in Table 3.6. Suppose we decide to change the primary key of the first row from (2689, 24001) to (2690, 24005) for some reason (it is difficult to think of a reason in this instance). Clearly this should not be allowed without strict controls.

Table 3.6 Problems with changing the primary key

MatchID	PID	NOvers	Maidens	NRuns	NWickets
2689	24001	3	0	27	1
2689	23001	3	0	15	0
2755	91001	4	0	29	0

Codd has suggested the following three possible approaches:

Method 1

Only a select group of users should be authorized to change primary key values.

Method 2

Updates on primary key values are banned. If it was necessary to change a primary key, the row should first be deleted and a new row with a new primary key value but the same other values should be inserted. Of course, this does require that the old values of attributes be remembered and be reinserted in the database.

Method 3

A different command for updating primary keys is made available in the DBMS. Making a distinction between altering the primary key and another attribute of a table would remind users that care needs to be taken in updating primary keys.

3.5.2 Foreign Keys and Referential Integrity

We have discussed some problems related to modifying primary key values. Additional problems can arise in modifying primary key values because one table may include references to another. These references include the primary key value of a row which may be referred to in one or more other tables of the database. The integrity constraints generally include that every value of the primary key of one table (sometimes called the *referenced* table) that appears in another table (i.e., the *referencing* table) must occur as a value of the primary key in the referenced table. Furthermore, when a primary key is modified (for example, a row from the table might be deleted), each of these references to a primary key must either be modified accordingly or be replaced by NULL values. Only then can we maintain referential integrity.

Before we discuss referential integrity further, we define the concept of a *foreign key*. The concept is important since a relational database consists of tables only (no pointers) and relationships between the tables are implicit, based on references to primary keys of other tables. These references are called *foreign keys*.

Definition—Foreign Key

The foreign key in table *R* is a set of columns (possibly one) whose values are required to match those of the primary key of some table *S*. *R* and *S* are not necessarily distinct.

It should be noted that the foreign key and primary key must be defined on the same domains. As an example, let us consider the database schema in Fig. 3.16.

Match(MatchID, Team1, Team2, Ground, Date, Winner)
Player(PlayerID, LName, FName, Country, YBom, BPlace, FTest)
Batting(MatchID, PID, Order, HOut, FOW, NRuns, Mts, NBalls, Fours, Sixes)
Bowling(MatchID, PID, NOvers, Maidens, NRuns, NWickets)

Figure 3.16 Database schema for the ODI cricket database

We now note the following:

1. The table *Match* has the primary key MatchID but no foreign key.
2. The table *Player* has the primary key PlayerID but no foreign key.
3. The table *Batting* has the primary key (Match ID, PID) and foreign keys MatchID and PID.
4. The table *Bowling* has the primary key (Match ID, PID) and foreign keys MatchID and PID.

Several questions now arise. What do the foreign keys refer to? Could (MatchID, PID) be foreign keys as well as primary keys in tables *Batting* and *Bowling*?

Well, the answers are relatively straightforward. MatchID is referring to the primary key of the table *Match* while PID is referring to the primary key of the relation *Player*. (MatchID, PID) together are not foreign keys because *Batting* and *Bowling* do not refer to each other.

In some cases like the following, there may be a foreign key that refers to its own table, for example, if the *supervisor-no* is an *empno*.

Employee (*empno*, *empname*, *supervisor-no*, *dept*)

Therefore, foreign keys are the implicit references in a relational database. The following constraint is called the referential integrity constraint:

Definition—Referential Integrity

If a foreign key F in table R refers to and matches the primary key P of table S then every value of F must either be equal to a value of P or be wholly NULL.

The justification for the referential integrity constraint is simple. If there is a foreign key in a relation (that is, the table is referring to another table) then the foreign key value must match with one of the primary key values in the table to which it refers. That is, if an object or entity is being referred to, the constraint ensures the referred object or entity exists in the database. It is not required that the foreign key values be always non-null. Sometimes a foreign key may be null, for example, a supervisor may not yet be assigned for an employee. Often a foreign key may be a component of a primary key and then the existential integrity constraint prohibits the appearance of nulls.

A foreign key in some DBMS may refer the column(s) of another table that have a combined UNIQUE constraint as well as the NOT NULL constraint.

Just as primary key updates need to be treated in a special way, one must deal with foreign key updates carefully. Foreign key updates do not need to be restricted in quite the same way as primary key updates. The system does need to ensure that the update values appear in the referenced table. Otherwise the update should be rejected.

The foreign key constraint may be specified in SQL as given in Fig. 3.17.

```
CREATE TABLE Match
  (MatchID INTEGER,
   Team1 CHAR(15),
   Team2 CHAR(15),
   Ground CHAR(20),
   Date CHAR(10),
   Winner CHAR(10)
  UNIQUE (MatchID)
  PRIMARY KEY (MatchID)
  FOREIGN KEY (MatchID) REFERENCES Match
  FOREIGN KEY (PID) REFERENCES Player)
```

Figure 3.17 Applying the foreign key constraint in SQL

3.5.3 Domains

We have noted that not all commercial database systems provide facilities for specifying domains. Domains could be specified as in Table 3.7.

Note that *NAME1* and *NAME2* are both character strings of length 10 but they belong to different (semantic) domains. It is important to denote different domains to

- (a) constrain unions, intersections, differences and equi-joins of tables, and
- (b) let the system check if two occurrences of the same database value denote the same real world object.

The constraint on union-compatibility and join-compatibility is important so that only those operations that make sense are permitted. For example, a join on class number and student number would make no sense even if both attributes are integers. The user should not be permitted to carry out such operations (or at least be warned when it is attempted).

3.5.4 Nulls

The issue of null values in a database is extremely important. A value in a database may be null for several reasons. The most obvious reason is that the value is not known at the present time. For example, a new employee's home telephone number will not be known until a phone is connected to his/her residence. Another reason why some values in a database may be null is that the attribute may not be applicable. For example, an employee database may have an attribute spouse which is not applicable for a single employee. For female employees the maiden name column might be inapplicable if the employee is single or if the employee did not change her name after marriage. In yet another situation, some information may not be known and will never be known. It is possible that an older migrant or a refugee may not know their date of birth. The person may not even know where he/she was born. We know of a situation when a student did not even know what his nationality was. His mother was from Bangladesh and his father from Burma and neither of the two countries was willing to provide him with a passport!

Having null values in a database creates a variety of issues beginning from simple ones like how to define the average of a collection of values if some are null to comparing null values with non-null values.

Further discussion of this interesting and complex issue is beyond the scope of this book.

Table 3.7 Specifying domains

CREATE DOMAIN NAME1	CHAR(10)
CREATE DOMAIN Player1	INTEGER
CREATE DOMAIN NAME2	CHAR(10)
Etc.	

3.6 ADVANTAGES OF THE RELATIONAL MODEL

Codd in 1970 indicated that there were four major objectives in designing the relational data model. These are

1. *Data Independence*—To provide a sharp and clear boundary between the logical and physical aspects of database management.

2. *Simplicity*—To provide a simpler structure than was being used at that time. A simple structure is easy to communicate to users and programmers and a wide variety of users in an enterprise can interact with a simple model.
3. *Set-Processing*—To provide facilities for manipulating a set of records at a time so that programmers are not operating on the database record by record.
4. *Sound Theoretical Background*—To provide a theoretical background for the database management field.

3.7 RULES FOR FULLY RELATIONAL SYSTEMS

If a system supports all relational algebra operators and the two integrity constraints (entity integrity and referential integrity) then the system may be called *fully relational*. Codd provided 12 rules (plus rule 0) to evaluate the properties of a relational model. A fully relational DBMS needs to satisfy these rules. The rules are

1. *Rule 0*—For any system that is advertised as, or claims to be, a relational database management system, that system must be able to manage databases entirely through its relational capabilities.

This basic rule states that all database management should be based on the relational model. The rule is included only because when relational database systems were becoming popular, some of the nonrelational database systems started advertising their systems as relational systems.

Data Structure Rules

2. *Rule 1*—All information in a relational database is represented explicitly at the logical level and in exactly one way, by values in tables.

There are many advantages in having this simple data structure. The data manipulation language can be simple. Additional software that interfaces with a relational DBMS can be based on the assumption that all data is represented in tables. The early nonrelational systems had much more complex data structures and the application programs were then also complex.

3. *Rule 2*—Each and every datum (atomic value) in a relational database is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.

This rule requires that the only way to access data in a relational model is by specifying the name of the table, the column(s) and its primary key value(s). No other traversing of the database, for example row-by-row, should be required.

4. *Rule 3*—Null values (distinct from the empty character string or a string of blank characters are distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.

Most database systems have some data that is unavailable or inapplicable. The relational model requires that this data be treated more systematically than by defining a special value, for example -9999, for each attribute.

5. *Rule 4*—The database description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

This rule deals with the database metadata which should be able to be manipulated in the same way as the regular data making it easier for database users to interrogate it.

Data Manipulation Language Rules

6. *Rule 5*—A relational system may support several languages and various modes of terminal use (for example, the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible using some well-defined syntax, as character strings and that is comprehensive in supporting all of the following items:

- Data definition
- View definition
- Data manipulation (interactive and by program)
- Integrity constraints
- Transaction boundaries (such as begin, commit and roll-back)

This rule requires that the user has access to at least one language that carries out all relational database operations

7. *Rule 6*—All views that are theoretically updatable are also updatable by the system.

This is a difficult requirement since it is not easy for a database system to determine if the update request by the user to update a view is reasonable.

8. *Rule 7*—The capability of handling a base table or a derived table as a single operand applies not only to the retrieval of data but also to the insertion, update and deletion of data.

As we will show in the next two chapters, insertion, update and deletion are able to use a table as a single operand enabling a set of tuples to be inserted, updated or deleted in one command.

Data Independence Rules

9. *Rule 8*—Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods.

This rule deals with data independence, one of the most important characteristic of a relational database system. It may be defined as the ability to separate the database definition from the physical storage structure of the database. Therefore any changes in the physical structure of the database should not impact the application programs.

10. *Rule 9*—Application programs and terminal activities remain logically unimpaired when information-preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

This again relates to data independence.

11. *Rule 10*—Integrity constraints specific to a particular relational database must be definable in the relational data sub-language and storable in the catalog, not in the application programs.

This rule requires that integrity constraints should not be included in the application programs and therefore application programs are easier to write. Also, it ensures that all application programs are subjected to exactly the same integrity constraints.

12. *Rule 11*—A relational DBMS has distribution independence.

This rule relates to distributed database systems which are discussed in Chapter 13.

13. *Rule 12*—If a relational system has a low level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational languages (multiple-records-at-a-time).

This rule essentially requires that no low level language be used since that may bypass the rules of a relational system.

3.8 EARLIER DATABASE MODELS – HIERARCHICAL AND NETWORK MODELS

As noted earlier, the relational database model replaced the earlier models in the 1980s. Two database models that were popular in the late 1960s and 1970s and now superseded by the relational DBMS is also now discussed. Some database systems even today use these old database systems. IMS, IDMS and TOTAL are examples of DBMS that were based on these models and were used widely. Some DBMS that were not based on either of these two models were also in use in the 1970s. These included ADABAS from a German company called Software AG. ADABAS was an inverted list database.

3.8.1 Network Data Model

The network model was developed by Charles Bachman and his colleagues at the General Electric Company in 1964. The model was standardized in 1971 by the CODASYL group (Conference on Data Systems Languages). The model used directed acyclic graphs (called Bachman diagrams) with nodes and edges where nodes represent the entity sets and edges (or links) represent the relationships between the entity sets. The links could be either 1:1 or 1:*m* so many-to-many relationships had to be decomposed into simpler relationships. The model allowed each child record to have multiple parents allowing 1:*n* relationships. The major problem with the network model was the complexity of designing a database system and its maintenance.

In the network model, we can represent each player as a node and their batting and bowling records also as nodes. They are then connected by edges or links. Finding batting and bowling information for a player was relatively easy since it only required searching for the relevant player node record and then finding his bowling and batting records by following the edges. Finding the names of players who scored a century involved searching the nodes for batting and finding nodes that had a score of more than 99 and then following the edges to find player nodes.

As noted above, a number of widely used database systems were based on the network model. These included IDMS (Computer Associates), IDS II (Honeywell) and DMS-1100 (Unisys).

Some typical commands used in a network database system are presented in Fig. 3.18.

A simple example of a network model database is presented in Fig. 3.19.

Command Name	Description
FIND	Locate record
GET	Transfer record to memory
OBTAIN	Combine FIND and GET
STORE	Insert a new record
ERASE	Delete record

Figure 3.18 Examples of commands used in a network database model

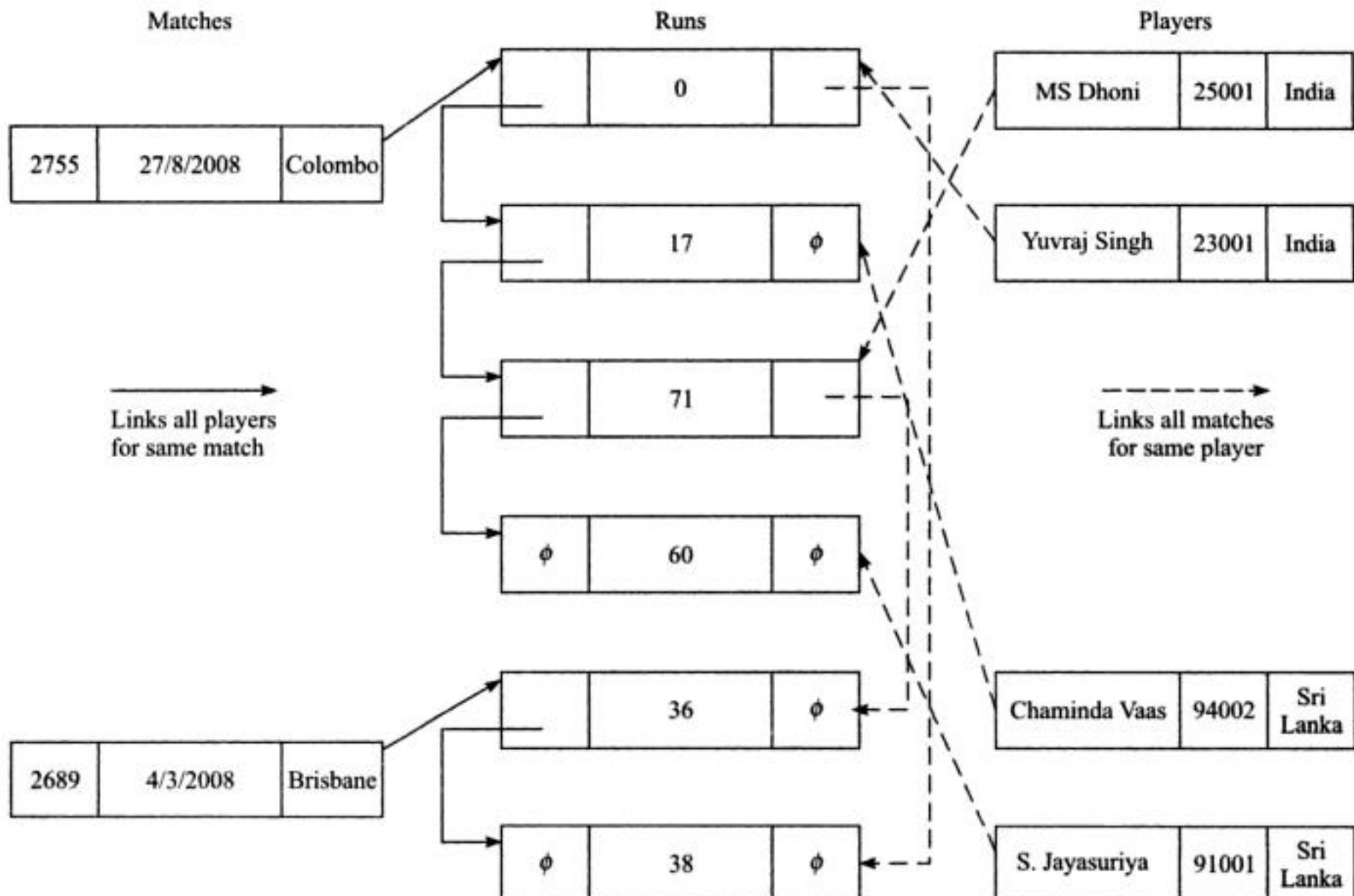


Figure 3.19 A typical network database model

Figure 3.19 shows a few records in a network model for the cricket database. There are two matches at the left-hand side and four players on the right-hand side. The players and matches are connected by the records in the middle which gives details of batting. We have only shown runs in these records but other information could be included. All the batting records of a match are linked in a chain with solid lines while all the batting records of a player are linked by the dotted lines on the right-hand side. Similar records could include all the bowling information. These chains allow one to traverse through all batting records of a match as well as all batting records of a player and these chains must be followed if one wanted to know how many runs Jayasuriya scored in match 2755. This is clearly very different than what happens in the relational model.

We leave it as an exercise for the reader to work out how insertions, deletions and updates take place in the two models and also compare these models with the relational model.

In summary, the network model was conceptually simple, efficient for some database problems and provided sufficient flexibility in that all relationships could be represented without much redundancy. Given that the network model was navigational, the performance of network DBMS was generally not high and some changes to the underlying database were difficult to implement. For example, changing relationships required physical reorganization of the data as well as changes to the application programs and therefore provided only limited data independence. Also these early DBMS were not robust. For example, a failure in the system occasionally left dangling references to data which had to be recovered.

3.8.2 Hierarchical Data Model

The hierarchical model was another model that formed the basis of the earliest database systems. It was a somewhat more restrictive model and could be considered a subset of the network model.

The hierarchical model was developed at IBM and was the basis of the IMS database system. IBM never supported the network model. IMS was based on the thinking that all views in life are hierarchical and therefore any database application could also be modelled as a hierarchy. The basic building block of the hierarchical model was called a *Physical Database Record* (PDBR) which consisted of a root segment and a hierarchy of segments (called *child segments*). Each segment was like a row in a table consisting of a set of related fields. Each PDBR was defined by a database description (DBD) that was coded by a programmer.

Since IBM already had some file processing packages, the IMS hierarchical model used them for storage and retrieval of data. These included the Sequential Access Method (SAM), Indexed Sequential Access Method (ISAM), Virtual Sequential Access Method (VSAM) and Overflow Sequential Access Method (OSAM). The IMS data manipulation used Data Language I (DL/I) that had a set of commands used with a host language like COBOL. DL/I commands essentially involved traversing the hierarchical database structure.

The hierarchical model was based on a tree structure which may be defined as follows.

Definition—Tree

A tree is a hierarchical structure that is a finite set of nodes connected by edges such that there is one specially designated node called the root of the tree. The remaining nodes are partitioned into n ($n \geq 0$) disjoint sets, each of which is also a tree.

It represented data as relationships between tree nodes called parent records (also called *segments* in the hierarchical model) and child records (or segments) as a tree. These relationships between the two record types may only be one-to-many. For example, we could represent each player as a parent segment and their batting and bowling records as child segments. Finding batting and bowling information for a player is relatively easy since it only requires searching for the relevant parent record and then finding his bowling and batting child records, but many other types of queries are more difficult. For example, finding names of players who scored a century in Melbourne is more difficult because it involves going through each player's parent record then searching the child records to check if there is a record that has more than 99 runs and is about a match in Melbourne. In summary, the following should be noted about the hierarchical model:

1. The model was efficient for searching the parent records and their child records since there was less redundant data.
2. Searching through the child records was more complicated and very different than in the relational model.
3. There were difficulties in representing many-to-many relationships.

Some commands used in IMS, a well-known hierarchical database system, are presented in Fig. 3.20.

In using the hierarchical model to represent the cricket database including the tables *Player*, *Match*, *Batting* and *Bowling* difficulties arise because the hierarchical model assumes all information to be hierarchical in nature. In the cricket database it is not clear which entity may be considered the parent entity and which as the child entity.

Command Name	Acronym	Description
GET UNIQUE	GU	Retrieve record
GET NEXT	GN	Sequential retrieval
GET NEXT WITHIN PARENT	GNP	Sequential retrieval of same parent
DELETE	DLET	Delete existing segment
INSERT	ISRT	Add new segment

Figure 3.20 Examples of commands used in a hierarchical database model

We will attempt to model the information assuming that the *Match* entity is parent, *Player* is the child of the entity *Match* and *Batting* and *Bowling* are child entities of *Player*.

The cricket database in the hierarchical model is presented in Fig. 3.21.

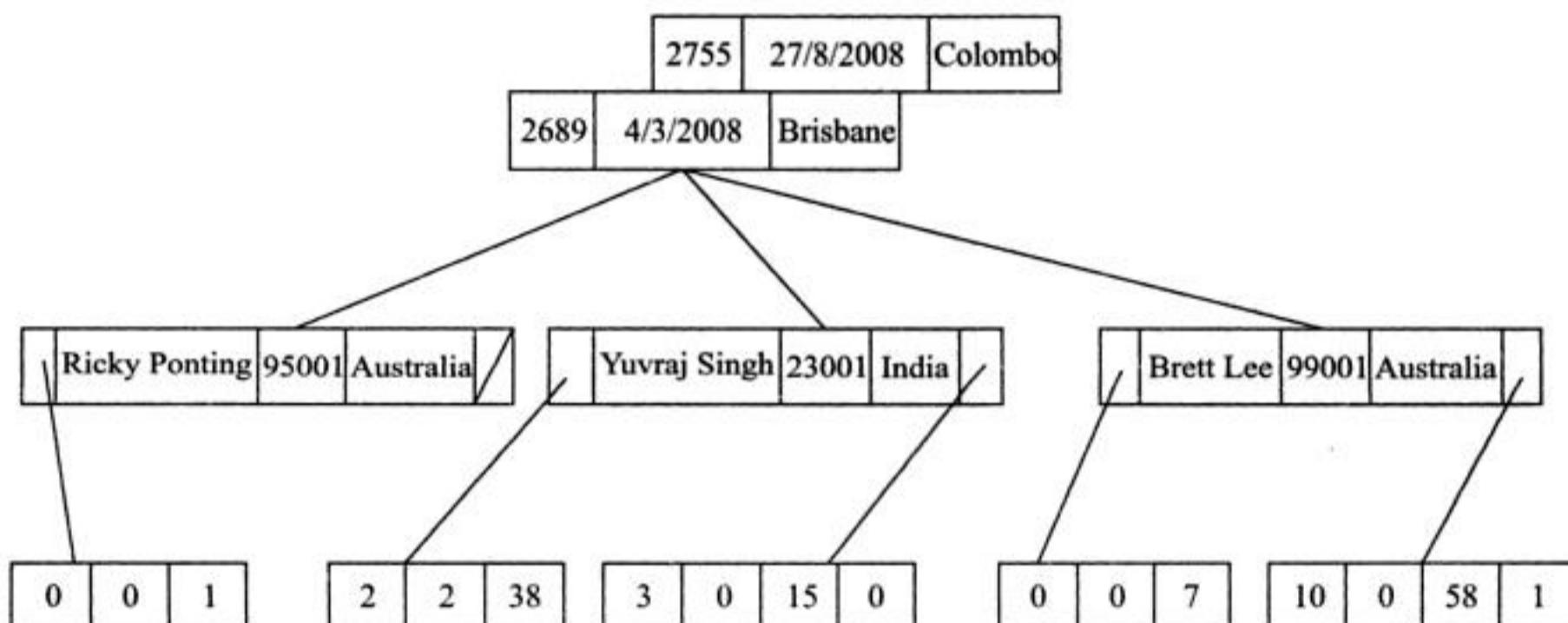


Figure 3.21 A typical hierarchical database model

Figure 3.21 shows only a few records. There are two matches at the root. One of them is shown to have three children players that the parent entity points to and each player has one or two children which are the batting or bowling records of the player. The first field is a pointer to the batting child record and the last field is a pointer to the bowling child record.

Given the above structure, the hierarchical model requires the following:

1. A parent record type may own an arbitrary number of child record types.
2. No single occurrence of a record type may have more than one parent record type.

In summary, software based on the hierarchical model provided an efficient representation of hierarchical structures, efficient single key search (if the hierarchical structure was the application view of the data), fast update performance for some updates. It was a conceptually simple model. On the other hand, hierarchical database systems had difficulty with many-to-many relationships, it lacked flexibility (nonhierarchical relationships were difficult to represent), performance for nonhierarchical accesses was poor, lack of maintainability, for example, changing relationships often required physical reorganization of data as well as changes to the application programs.

3.8.3 Comparison of Hierarchical, Network and Relational Models

Both the network and hierarchical models were developed in the 1960s and were conceptually simple when they were proposed but suffered from a number of serious disadvantages partly because both were navigational systems. The disadvantages included low level of data independence, difficulty in representing many-to-many relationships. For a large database, the logical data model became very complex and since the data manipulation languages were procedural the application programs became complex as well and difficult to maintain. In both hierarchical and network models, physical and logical data independence was difficult to achieve, since the data manipulation languages and therefore the application programs often contained a wide variety of information about the physical organisation of the database.

Hierarchical model's single parent rule forces redundancy. It can be eliminated in the network model since the network model allows more flexibility and a record type may have an arbitrary number of parents.

The three models are compared in Fig. 3.22 below.

Item	Hierarchical	Network	Relational
Data structure	Hierarchy	Tree of records	Tables
Data Access	Navigational	Navigational	Non-navigational
Method of access	Procedural First access the root and then navigate to the record of interest.	Procedural Data access may start from any entity.	Nonprocedural SELECT, FROM, WHERE
User must know	Structure of the database	Structure of the database	Names of the tables
Performance	Was fast for some simple queries	Was fast for some simple queries	Good for many types of queries
Mathematical basis	None	None	Has mathematical foundation
Query language	Use DI/I through COBOL	COBOL based commands	SQL
Data integrity	None	None	Basic integrity is part of the model
Modifying the database	Difficult	Difficult	Relatively easy
Data independence	None	None	Part of the model

Figure 3.22 A comparison of the three database models

SUMMARY

This chapter has covered the following points:

- The relational data model was invented by E. F. Codd in 1970. The three major components of the model are data structure, data manipulation and data integrity.
- Data structure component of the relational model required that all information presented in a database be as relations (essentially tables).
- Each relation must meet a number of properties, for example, atomic values, no duplicate tuples, no ordering of tuples, etc.
- Relational terminology includes terms like a relation (or a table), tuple (or a row), degree (number of columns), cardinality (number of rows), attribute (column), domain, candidate key and primary key.
- The primary key of a relation has the properties of uniqueness and minimality.
- An Entity-Relationship model discussed in Chapter 2 may be mapped to relational model. Mapping includes mapping each entity and each many-to-many relationship to a table.
- Data integrity in the relational model includes constraints relating to primary keys (called existential integrity) and foreign keys (called referential integrity) as well as domains and NULLs.
- Relational data model has the advantages of data independence, simplicity, set processing and sound theoretical background.
- Codd defined 13 rules that a fully relational model must satisfy including data structure rules, data manipulation rules and data independence rules.
- The basic concepts of two older database models, the network model and the hierarchical model, have been presented. The hierarchical model is based on the concept that all data has hierarchical nature.
- Disadvantages of the hierarchical and network database models have been described and the models are compared to the relational model. These include different data structures, different database languages, lack of integrity in the older models, and low level of data independence in the older models.

REVIEW QUESTIONS

1. What are the major components of the relational model? (Section 3.1)
2. What is the meaning of the terms relation, tuple, degree, cardinality, attribute, domain, candidate key and primary key? (Section 3.2)
3. Give two examples to illustrate the properties of a primary key? (Section 3.2)
4. Present an E-R model and show how it should be mapped to the relational model. How is an entity mapped? How is a relationship mapped? (Section 3.2.1)
5. Describe constraints related to the primary key, foreign key, domains and NULLs. (Section 3.4)
6. List some of the advantages of the relational model. (Section 3.5).
7. List five rules for fully relational systems. (Section 3.6)
8. Describe the main features of the hierarchical data model. (Section 3.7)
9. Describe the main features of the network data model. (Section 3.7)

10. Using an example, illustrate the features of the hierarchical and the network model. List some of the disadvantages of these models. Why do you think the products based on these models could not compete with the products based on the relational model? (Section 3.7)

SHORT ANSWER QUESTIONS

1. What are the three components of the relational model?
2. List three properties of a relation.
3. What is the data structure used in the relational model?
4. List three major properties that a relation must have?
5. What is the cardinality of a relation? What is the cardinality of the relation in Fig. 3.3?
6. What is the degree of a relation? What is the degree of the relation in Fig. 3.2?
7. Define the candidate key of a relation.
8. How is an entity in an E-R model mapped to the relational model?
9. How is a many-to-many relationship in an E-R model mapped to the relational model?
10. How is a one-to-one relationship in an E-R model mapped to the relational model?
11. How is a weak entity in an E-R model mapped to the relational model?
12. What is a primary key and existential integrity?
13. What is a foreign key and referential integrity?
14. Why DBMS products based on the network model and the hierarchical model were not able to compete with relational DBMS?
15. What is the major difference between the network model and hierarchical model?

MULTIPLE CHOICE QUESTIONS

1. The main components (more than one) of the relational model are
 - (a) data structure
 - (b) data manipulation
 - (c) data recovery
 - (d) data integrity
2. Which one of the following is correct?
 - (a) A relation may have duplicate rows.
 - (b) A relation may have duplicate columns.
 - (c) A relation may have ordering associated with the rows.
 - (d) A relation may have non-atomic attribute values.
3. Which one of the following is allowed in the relational model?
 - (a) Repeating groups
 - (b) Pointers
 - (c) Record identifiers other than the primary key
 - (d) Specification of domains

4. Which two of the following are correct? Rows in the relational data model are required to be unordered because
 - (a) it makes the database more efficient.
 - (b) the user should not be burdened with having to remember which row is next to which row.
 - (c) numbering rows is expensive.
 - (d) changes in ordering may have to be made for a variety of reasons.
5. Which one of the following is correct? In using a relational database consisting of a number of tables, a user only needs to know
 - (a) the number of rows in each relation.
 - (b) the number of columns in each relation.
 - (c) the access paths to the data in each relation.
 - (d) the names of the tables and the names of their relevant columns.
6. The concept of domain is fundamental to the relational data model. Which of the following statements regarding domains is **not** correct?
 - (a) Conceptually, domains play a role similar to data types in programming languages.
 - (b) Domains are user defined while the data types are built into the system.
 - (c) A domain is a set of scalar or atomic values, all of the same type.
 - (d) There can be no more than one attribute in each relation defined on each domain.
7. Which two of the following properties must belong to the primary key of a relation?
 - (a) It identifies each row uniquely.
 - (b) It is an attribute or a set of attributes on which an index is built.
 - (c) It is an attribute or a set of attributes for which missing values are not allowed.
 - (d) A proper subset of it cannot be a key.
8. Which one of the following is correct?
 - (a) Each candidate key is a primary key.
 - (b) Each primary key is a foreign key.
 - (c) Each foreign key is a primary key in some relation.
 - (d) Each foreign key is a candidate key.
9. Which one of the following is correct?
 - (a) Referential integrity requires that a foreign key may not be NULL.
 - (b) Referential integrity requires that a foreign key may not be partially NULL.
 - (c) Existential integrity requires that a primary key may not be NULL although it may be partially NULL.
 - (d) A high level of integrity requires that there be no NULL values in the database.
10. Consider the following relation that stores information about offices, telephones and occupants of offices:

office(OfficeNum, TelephoneNum, Occupant)

Each office has a unique office number.

There is only one telephone in each office and the telephone number is unique. Each office is occupied by only one person although a person may have more than one office. Occupant is a unique identification for persons occupying rooms.

Which one of the following is correct?

- (a) OfficeNum and TelephoneNum are both candidate keys.
- (b) OfficeNum, TelephoneNum and (OfficeNum, TelephoneNum) are all candidate keys.
- (c) (OfficeNum, occupant) and (TelephoneNum, occupant) are the candidate keys.
- (d) OfficeNum must be the primary key.

11. Consider again the relation *office(OfficeNum, TelephoneNum, Occupant)*. Now assume that each office may have more than one occupant and more than one telephone and a telephone may be shared between two or more occupants. Furthermore, each occupant may have more than one office. OfficeNum, TelephoneNum and occupant names are unique.

Which one of the following is correct?

- (a) OfficeNum and TelephoneNum are both candidate keys.
- (b) OfficeNum, TelephoneNum and OfficeNum, TelephoneNum are all candidate keys.
- (c) (OfficeNum, occupant) and (TelephoneNum, occupant) are the candidate keys.
- (d) (OfficeNum, TelephoneNum, occupant) is the candidate key.

12. Which of the following are true? The value of an attribute in a table may be NULL because

- (a) the value is not known.
- (b) the value is not applicable.
- (c) the value will never be available.
- (d) All of the above

13. Which one of the following is correct?

- (a) Every subset of a relation is a relation.
- (b) Every subset of a tuple is a tuple.
- (c) Every subset of an attribute is an attribute.
- (d) Every subset of a foreign key is a foreign key.

14. Which of the following are included in Codd's rules for fully relational systems?

- (a) NULL values are independent of data type.
- (b) All views that are theoretically updatable are also updatable by the system.
- (c) A relational database has distribution independence.
- (d) All of the above

15. Which of the following are correct?

- (a) Hierarchical model was developed before the relational model.
- (b) Object-oriented database model was developed before the relational model.
- (c) The network database model was developed before the relational model.
- (d) The Entity-Relationship model was developed before the relational model.

16. Which one of the following is **not** correct?

- (a) The network model used acyclic graphs with nodes and edges.
- (b) The hierarchical model was based on a tree structure in which there was one specially designated node called the root of the tree.
- (c) Hierarchical model had difficulty representing many-to-many relationships.
- (d) The network model is navigational while the hierarchical model is not.

17. When a row is deleted which one of the following techniques should be used to maintain integrity?
- The row is deleted and nothing else is done.
 - The row is deleted and the references to the deleted primary key, if any, are replaced by NULL.
 - The delete operation is not allowed if the row's primary key is a target of a foreign key.
 - The row is deleted as well as the rows from other tables that have foreign keys that have the deleted primary key as their target.

EXERCISES

- What are the properties of a relation? Discuss each property and explain why it is important.
- Define a candidate key, a primary key and a foreign key. Can the primary key have null values? Can the foreign key have null values? Can one relation have more than one candidate key? Can one relation have more than one primary key? Can one relation have more than one foreign key? Explain and give examples to illustrate your answers.
- Discuss the following concepts and explain their importance.
 - Integrity constraint
 - Referential integrity
 - Existential integrity
- Relate each of the relational terms on the left-hand side to their informal equivalents on the right-hand side. Note that not all terms on the right-hand side have a matching term on the left-hand side.

Table 3.8 Terminology

<i>Relation</i>	<i>Pointer</i>
Tuple	Table
Primary key	Record or row
Candidate key	Link
Attribute	Field or column
Domain	File
Cardinality	A unique identifier
Degree	Pool of legal values Current values Number of attributes in a table Number of tuples Number of fields in the primary key

5. Consider the following student database:

Student(ID, Name, Address, City, Tutor)
 Enrolment(ID, Code, Marks)
 Course(Code, Title, Department)

*image
not
available*

For each relation, write the degree of the relation, its cardinality, all the candidate keys and the primary key.

7. Consider the following two relations in an example database with EmpID and DeptID as primary keys of the two tables respectively:

Table 3.12 The table Emp

EmpID	Ename	ManagerID	DeptID
126	Krishna Kumar	129	22
NULL	Prakash Patel	NULL	22
129	Milkha Singh	NULL	26
135	Rabindra Singh	173	99
149	Nizam Khan	129	NULL
173	David Abraham	NULL	NULL

Table 3.13 The table Dept

DeptID	Dname	MgrID
22	Sales	128
26	Production	126
75	Finance	129
81	Deliveries	149
89	NULL	173

- (a) Identify all foreign keys.
 (b) Do the above relations satisfy the existential and referential integrity constraints? Discuss.
 (c) If the tuple with EmpID = 129 in the *Emp* table is deleted, what should be done to maintain integrity of the database?
8. Consider the cricket database in Tables 3.1 to 3.4. List some integrity constraints that should apply to these tables.
9. A simple database is to be designed to store information about rooms and telephone numbers in an enterprise. The only information to be stored is room numbers and telephone numbers in each of them. Room numbers and telephone numbers are assumed to be unique.

One way to represent the information is to have a relational schema *room(RoomNum, PhoneNum)*. Is this relation adequate to represent the information in each of the following cases? If not, present an alternative database design consisting of one or more relational schemas for each of the cases below.

Consider the following four situations:

- (a) Each phone number is assigned to only one room and each room has a unique phone number.
 (b) A phone number may (with extensions) be assigned to more than one room and a room may have more than one phone each with a different number.

- (c) Each phone number is assigned to only one room but some rooms do not have any phones while others may have one or more phones.
- (d) Each phone number is assigned to only one room and each room has at least one phone. Some rooms have more than one phone.

PROJECTS

We now present a number of projects that may be used by instructors as a basis for designing projects for their course. We have used similar projects successfully for many years in our classes. A methodology for projects which may suit some instructors is proposed below.

Project Implementation Suggestion

It is proposed that each project should consist of the following three stages:

Stage 1

In the first stage, students are required to design an Entity-Relationship model for one (possibly allocated by the lecturer) of the database applications described below as well as others that are suggested by the lecturer.

Students should make suitable assumptions where necessary when implementing their project.

The following should be included with the Entity-Relationship diagram:

1. A discussion of the needs of the enterprise. A collection of typical queries that the system is expected to answer.
2. A list of all assumptions.
3. A list of all entities and relationships and their attributes and primary keys (most of this information can be included in the Entity-Relationship diagram).
4. A discussion of the strengths and weaknesses of the proposed solution.
5. And, of course, the complete Entity-Relationship diagram.

The design and the accompanying documentation should be submitted by the given date.

Stage 2

Revise the E-R diagram based on comments received from the lecturer. Build a set of ten sensible queries, some easy, some more complex that use subqueries, GROUP BY and aggregation functions. Submit these by the due date for Stage 2.

Stage 3

Implement the database using Oracle (or another system that is available). Populate the database. Formulate the queries and run them using the database. Submit a report including the results of the queries by the given date. The report should clearly identify objectives, discuss methodology, present assumptions and present results and discuss conclusions.

Bonus marks may be given to students who build a suitable user interface for their system.

A significant component of the assignment marks should be awarded on the basis of the quality of documentation.

1. A Small Airport

Design a database for a small airport, for example, in a place like Haridwar in Uttarakhand or Puri in Orissa. The database is designed to keep track of private airplanes, their owners, airport employees and pilots. Each airplane has an identification number, is of a particular type and has a parking space in a particular hangar. Each plane has a purchase date, its type has a model number, and a capacity in terms of number of passengers. Each hangar has a number and the number of planes that may park in it. The database also needs to keep track of owners of the planes and the employees that have worked on each plane and on what date. The maintenance record of each plane that includes the number of hours spent servicing the plane and a code for the type of service done. Some planes provide charter services to large companies like Infosys or Reliance. The database stores details of each such flight.

2. College of Pharmacy

Design a database system for a College of Pharmacy, Faculty of Medicine of the North Indian University (NIU) located in Mussorie². The faculty has research projects in various stages of development, i.e., current, pending and complete. Each project is funded by a single grant by the Indian Government. Usually, the major portion of this research grant is used to pay the salaries of research assistants and study's subjects. The various drugs and equipment used in the experiments are often provided by one or more drug companies, for example, Dr. Reddy's Pharmaceutical company or Ranbaxy Laboratories Ltd. A project studies the effects of the drug on many subjects, some of whom have unequal therapeutic regimens. Several employees of the College of Pharmacy work on each project, which is led by a principal investigator (PI). Each principal investigator may control several projects. When a study is completed, a research report is written describing the results of the study.

3. Bus Company Routes

Design a database system for managing information about routes supported by a bus company called The All India Bus Company with head office in Nagpur. Each route served by the company has a starting place and an ending place, but it can go through several intermediate stops. The company is distributed over several branches. Not all the cities where the buses stop have a branch. However, each branch must be at a city located along the bus routes. There can be multiple branches in the same city and also multiple stops in the same city. One bus is assigned by the company to one route; some routes can have multiple buses. Each bus has a driver and a conductor, who are assigned to the bus for the day.

4. Bus Company Office

Design the database for the administration and reservation office of a bus company called The All India Bus Company. Each passenger can book a seat on a given portion of the routes serviced by each bus; routes have a starting place, an ending place and several intermediate places. Passengers can specify whether they want to be in the smoking or non-smoking section. Some passengers can get on the bus even if they do not have a reservation, when some seats are left vacant. With each reservation, the last name, initials and telephone number of the passenger are stored. Sometimes, trips are not made because of bad weather conditions; in this case, passengers holding reservations are notified. At the end of the trip, the driver's assistant reports to the company the total number of tickets purchased on the bus by passengers and also report these figures to the administrative office of the branch at the route's destination.

2. All names used in these project descriptions are fictitious

LAB EXERCISES

1. List the properties of the relational model. For each property describe the consequences if a database system did not follow that property.
2. Map the E-R diagram (Fig. 3.23) given on the next page to relational model tables. List any problems you discover with the model and explain how to fix them.
3. Would some additional syntactical or semantic integrity help in maintaining database integrity? Try to think of situations when a database's integrity is violated and suggest integrity constraints that will prevent that violation.
4. Discuss why the relational databases became so popular in the 1980s and DBMS based on network and hierarchical models disappeared.

BIBLIOGRAPHY

For Students

A number of books listed below are easy to read for students. A fascinating account of the history of IBM's System R is available in the paper by Chamberlain et al.

For Instructors

Codd's papers and his out-of-print book (which may be downloaded from the Web) are the authoritative documents on the relational model. C. J. Date's books are also a good source of information. The 1982 book by Ullman is a somewhat advanced treatment of database systems.

Chamberlain, D., et al., "A History and Evaluation of System R", *Communications of the ACM*, Vol. 24, No. 10, 1981, pp 632-646.

Codd, E. F., "A relational model of data for large shared data banks", *Communications of the ACM*, Vol. 13, No. 6, 1970, pp 377-387.

Codd, E. F., "Relational Database: A practical foundation for productivity", *Communications of the ACM*, Vol. 25, No. 2, 1982, pp 109-117.

Codd, E. F., "Extending the relational model to capture more meaning", *ACM Transactions on Database Systems*, Vol. 4, No. 4, 1979, pp 397-434.

Codd, E. F., *The Relational Model for Database Management: Version 2*, Addison-Wesley Publishing Company, Inc, 1990. (This book is out of print now but may be downloaded from <http://portal.acm.org/citation.cfm?id=77708>.)

Date, C. J., *An Introduction to Database Systems*, Eighth Edition, Addison-Wesley, 2003.

Date, C. J., *Database in Depth: Relational Theory for Practitioners*, O'Reilly Media, 2005.

Maier, D., *The Theory of Relational Databases*, Computer Science Press, 1983. <http://web.cecs.pdx.edu/~maier/TheoryBook/TRD.html>

Ramakrishnan, R., and J. Gehrke, *Database Management Systems*, Third Edition, McGraw-Hill, 2002.

Silberchatz, A., H. Korth, and S. Sudarshan, *Database Systems Concepts*, Fifth Edition, McGraw-Hill, 2005.

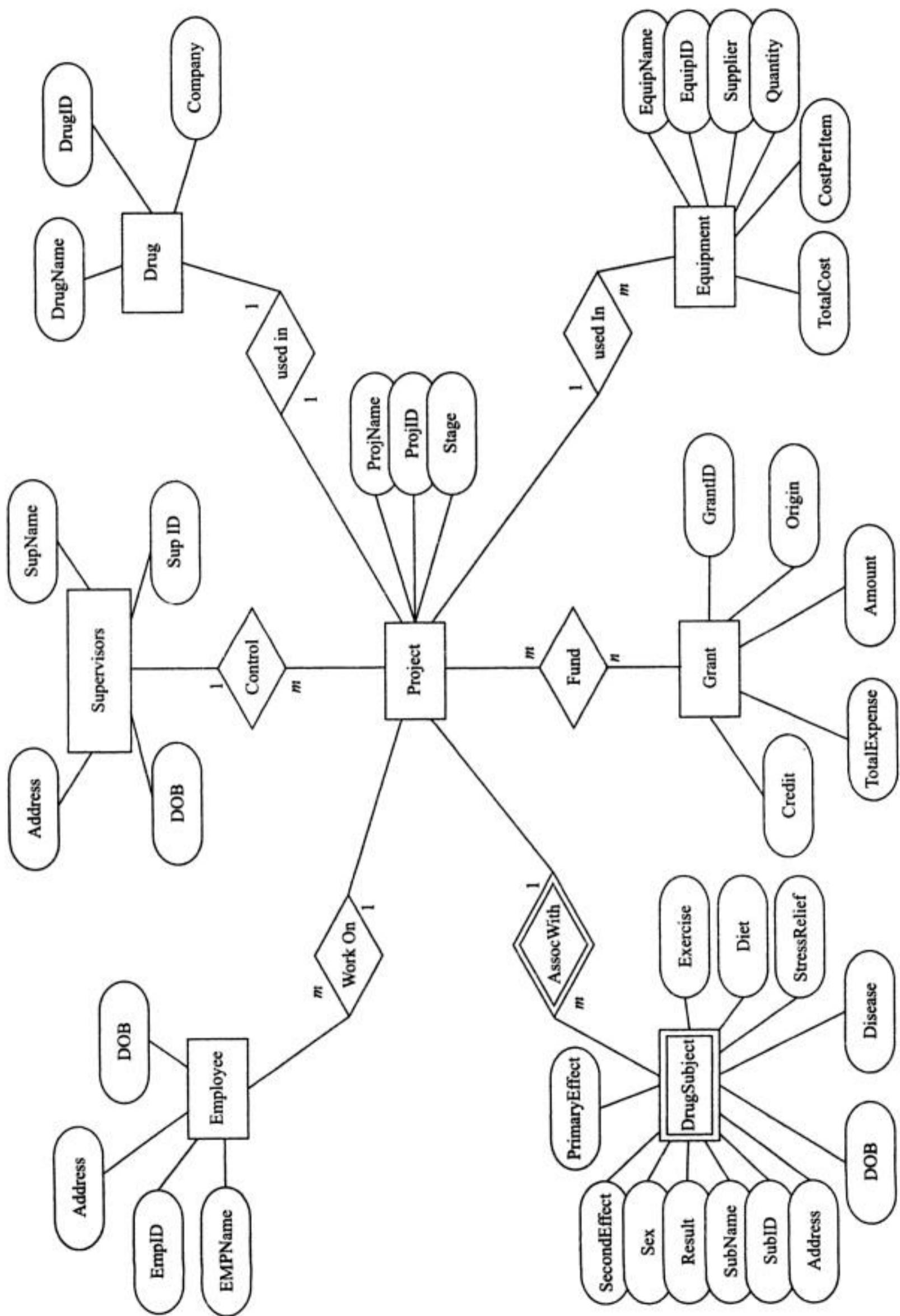
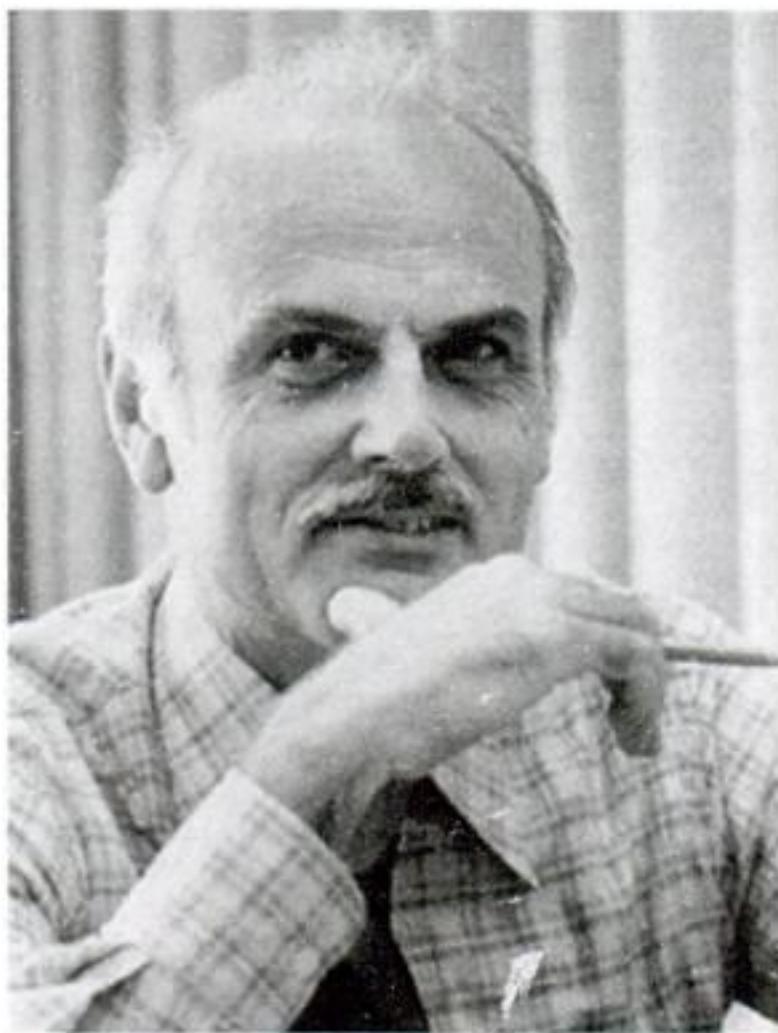


Figure 3.23

- Garcia-Molina, H., J. D. Ullman, and J. Widom, *Database Systems: The Complete Book*, Prentice Hall, 2002.
- Kifer, M., Bernstein A., and P. M. Lewis, *Database Systems: An Application-Oriented Approach*, Second Edition, Addison-Wesley, 2006.
- Ullman, J. D., *Principles of Database Systems*, Computer Science Press, 1982. (May be downloaded from <http://www.filestube.com/d/database+systems+ullman>)

Edgar (Ted) Codd (1923–2003)—Inventor of the Relational Model



In the 1960s and 1970s, Ted Codd worked on his theories of database management publishing his paper *A Relational Model of Data for Large Shared Data Banks* in 1970. In that paper he proposed replacing the hierarchical or navigational structure with simple tables containing rows and columns. In order for the relational model to be accepted, it had to be proved by an implementation. This was the goal of the System R project at IBM's San Jose Research Laboratory in 1973. Other pioneers who rushed to exploit Codd's relational concepts in the 1970s included Mike Stonebraker at the University of California Berkeley who developed the database system Ingres.

Ted Codd was born in England. He studied mathematics and chemistry at the Oxford University and served as a pilot in the Royal Air Force during World War II. After the war, in 1948, he moved to New York and joined IBM as a programmer. He received his doctorate in computer science from the University of Michigan. He worked for many years at the IBM's Almaden Research Center in San Jose.



Relational Algebra and Relational Calculus

4

OBJECTIVES

- Describe how information is retrieved and modified in the relational model.
- Describe relational algebra.
- Present examples of using operators used in relational algebra.
- Describe relational calculus.
- Explain tuple relational calculus (TRC) and present examples.
- Explain domain relational calculus (DRC) and present examples.
- Present foundations for SQL.
- Discuss relative advantages and disadvantages of relational algebra and relational calculus.

KEYWORDS

Relation, relational algebra, relational calculus, selection, restriction, projection, cross product, join, natural join, equi-join, division, union, intersection, difference, procedural language, declarative language, well-formed formulas, queries, tuple relational calculus, TRC, domain relational calculus, DRC, universal quantifier, existential quantifier.

I measure the progress of a community by the degree of progress which women have achieved.

B. R. Ambedkar (Author of the Indian Constitution, 1891–1956)

4.1 INTRODUCTION

Codd in 1970 presented the relational model and with it two formal languages, called relational algebra and relational calculus. Both data manipulation languages provide facility to retrieve information from a relational database. Relational algebra is a procedural language that uses certain primitive operators that take tables as inputs and produce tables as outputs. The language may be considered operational, that is, it provides a step-by-step procedure for computing the result. Each relational algebra operator is either unary or binary, that is, it performs data manipulation on one or two tables. Relational algebra is not similar to programming languages like C or C++, that is, it does not include a looping construct.

The basic relational algebra commands were introduced in the 1970 paper by Codd. These are as follows:

- SELECT
- PROJECT
- UNION
- PRODUCT
- JOIN
- DIVIDE

The relational calculus as opposed to the relational algebra is a declarative language. A statement in relational calculus essentially specifies which tuples (or their columns) are to be retrieved. Therefore relational calculus does not provide a step-by-step procedure. We consider two slightly different forms of relational calculus. These are called *Domain Relational Calculus* (DRC) and *Tuple Relational Calculus* (TRC).

This chapter is organized as follows. The next section, Section 4.2 deals with relational algebra. All relational algebra operators are explained using a number of examples. Section 4.3 deals with relational calculus. The domain relational calculus and the tuple relational calculus are explained along with a number of example queries.

4.2 RELATIONAL ALGEBRA

Database queries in relational algebra are based on specifying operations on tables. The relational algebra expressions are similar to algebraic expressions (for example, $2*a + b*c$) that we are familiar with except that we are now dealing with tables and not numbers or simple variables.

We will define a number of operations that can be applied to tables of all sizes for selecting data from a relational database. The operations include the usual set operations like union and intersection as well as operations for selecting a number of rows or one or more columns from a table.

The result of applying one or more relational operations to a set of tables (possibly one) is always a new table.

We first discuss three relational algebra operations that are sufficient for formulating a large number of useful queries. These operations are *selection*, *projection* and *join*. We will also discuss *Cartesian product* since it is needed in defining the join. Further relational operators are discussed later in the section.

4.2.1 Unary Relational Operator—Selection (also called Restriction)

Unary relational algebra operators operate on a single specified table and display results in table form all or some rows or column of the table.

The operation of selecting certain rows from a table is called *selection* (or *restriction*). Usually one wants to select rows that meet a specified condition, for example, players from India. Selection may then be used. It is a unary operator since it operates only on a single table.

We illustrate this simple concept by showing a table in Fig. 4.1 which has five columns labelled *A*, *B*, *C*, *D* and *E* and five rows. A selection condition (for example, attribute *C* not being equal to *c*₂) results in selecting three rows of the table.

More formally, selection of a table *R* is a subset table *S* that includes all those rows in *R* that meet a condition specified by the user. The number of columns (degree) in *S* is the same as that of *R*. The number of rows (cardinality) of *S* is equal to the number of rows in *R* that meet the specified condition.

The result of applying selection to a table *R*, such that the column *A* (which may be one or more columns) has value *a* will be denoted by *R[A=a]*. Similarly, *R[A>a]* is a selection on *R* such that column *A* has values greater than *a*. A more mathematical way of writing selection is to use the notation $\sigma_{A=a}[R]$ to denote *R[A=a]*¹. More generally $\sigma_p[R]$ is used to denote a restriction on table *R* which selects rows that satisfy the condition *p*.

Let us now consider two examples of the use of the operator selection. Suppose we wish to find all the Indian players that are in the database, we do so by selecting rows in table *Player* where the value of the attribute *Country* is India. We write the selection as in Fig. 4.2.

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>a</i> ₁	<i>b</i> ₁	<i>c</i> ₂	<i>d</i> ₁	<i>e</i> ₁
<i>a</i> ₂	<i>b</i> ₂	<i>c</i> ₁	<i>d</i> ₁	<i>e</i> ₁
<i>a</i> ₃	<i>b</i> ₁	<i>c</i> ₂	<i>d</i> ₂	<i>e</i> ₂
<i>a</i> ₄	<i>b</i> ₁	<i>c</i> ₁	<i>d</i> ₃	<i>e</i> ₃
<i>a</i> ₅	<i>b</i> ₁	<i>c</i> ₃	<i>d</i> ₁	<i>e</i> ₄

Figure 4.1 Selection—Selecting three rows in a table

Player [country = 'India'] or $\sigma_{\text{country} = \text{'India'}}[\text{Player}]$

Figure 4.2 Using SELECTION in relational algebra

1. The Greek symbol is easy to remember since the operator selection and the Greek letter sigma (σ) both start with the letter *s*. We will from time to time use Greek symbols since they are often used in advanced database textbooks but this notation may be skipped if the reader feels uncomfortable with it.

The result of applying the above selection operation to Table 3.2 *Player* is presented in Table 4.1.

Table 4.1 Result of a selection operator

PlayerID	LName	FName	Country	YBorn	BPlace	FTest
89001	Tendulkar	Sachin	India	1973	Mumbai	1989
96001	Dravid	Rahul	India	1973	Indore	1996
90002	Kumble	Anil	India	1970	Bangalore	1990
25001	Dhoni	M. S.	India	1981	Ranchi	2005
23001	Singh	Yuvraj	India	1981	Chandigarh	2003
96003	Ganguly	Saurav	India	1972	Calcutta	1996
21001	Sehwag	Virender	India	1978	Delhi	2001
98002	Singh	Harbhajan	India	1980	Jalandhar	1998
27001	Kumar	Praveen	India	1986	Meerut	NA
27002	Sharma	Ishant	India	1988	Delhi	2007

Similarly, if we wish to find out what matches have been played in Australia, we select those rows from table *Match* that have column *Team1* value equal to ‘Australia’ assuming that *Team1* is the home team and these matches are not being played at a neutral country. We could write it in relational algebra as in Fig. 4.3.

$\text{Match} [\text{Team1} = \text{'Australia'}] \text{ or } \sigma_{\text{Team1} = \text{'Australia'}}[\text{Match}]$

Figure 4.3 Using the selection operation in relational algebra

The result of applying the selection operation in Fig. 4.3 to Table 3.1 *Match* is presented in Table 4.2 which has only two rows in it.

Table 4.2 Result of applying a selection operator to Match

MatchID	Team1	Team2	Ground	Date	Winner
2688	Australia	India	Sydney	2/3/2008	Team2
2689	Australia	India	Brisbane	4/3/2008	Team2

4.2.2 Another Unary Relational Operator—Projection

Projection is the relational algebra operation of selecting certain columns from a table *R* to form a new table *S*. For example, one may only be interested in the list of names from a table that has a number of other attributes. Projection may then be used. Like selection, projection is a unary operator that is applied to only one table.

We illustrate the concept of projection by showing the table given in Fig. 4.4 which has four columns labeled *B*, *C*, *D* and *E* and five rows. A projection of the table on attributes *C* and *E* results in selecting the two

highlighted columns of the table. Note that the first and second rows in the table with two columns are duplicates.

$R[A]$ is a projection of R on A , A being a list of columns, say three columns a , b and c (the columns do not need to be distinct). Although the table R has no duplicate rows (since no table is permitted to have duplicate rows), a projection of R might have duplicate rows as illustrated by the projection in Fig. 4.4. The duplicate rows, if any, are removed from the result of a projection. The degree of $S = R[A]$ is equal to the number of columns in A . The number of rows in S is equal to the number of rows in R if no duplicates are present in $R[A]$. If duplicates are present then the number of rows in S is equal to the number of rows in R minus the number of duplicates removed from the result. Removal of duplicates from large tables can be computationally expensive.

A projection of R on columns a , b , c is denoted by $R[a,b,c]$. Some authors use $\pi_{a,b,c}[R]$ to denote the projection². More generally, $\pi_s(R)$ is a projection of table R where s is a list of columns, possibly only one column.

Let us now consider a simple example. If we wish to obtain a list of names of players, we may apply the projection operator to the table *Player* on attributes *FName* and *LName*. We write this projection as in Fig. 4.5.

$\text{Player}[\text{Fname}, \text{LName}] \text{ or } \pi_{\text{Fname}, \text{LName}}[\text{Player}]$

Figure 4.5 Using the projection operator in relational algebra

The result of applying the projection operator as given in Fig. 4.5 to Table 3.2 *Player* would lead to the result presented in Table 4.3.

Table 4.3 The result of applying the projection in Fig. 4.5 to Table 3.2

<i>FName</i>	<i>LName</i>
Sachin	Tendulkar
Brian	Lara
Ricky	Ponting
Rahul	Dravid
Herschelle	Gibbs
Shane	Warne
Shaun	Pollock
Michael	Vaughan
Inzamam	Ul-Huq

Contd.

- Once again, the Greek symbol for projection is easy to remember since the operator projection and the Greek letter pi (π) both start with the letter *p*.

<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
b_1	c_1	d_1	e_1
b_2	c_1	d_1	e_1
b_1	c_2	d_2	e_2
b_1	c_1	d_3	e_3
b_1	c_3	d_1	e_4

Figure 4.4 Projection—Projecting two columns in a table

Table 4.3 Contd.

<i>FName</i>	<i>LName</i>
Stephen	Fleming
Heath	Streak
Anil	Kumble
Gary	Kirsten
Jacques	Kallis
Chaminda	Vaas
Muthiah	Muralitharan
Daniel	Vettori
MS	Dhoni
Yuvraj	Singh
Saurav	Ganguly
Adam	Gilchrist
Andrew	Symonds
Brett	Lee
Sanath	Jayasuriya
Virender	Sehwag
Shahid	Afridi
Harbhajan	Singh
Praveen	Kumar
Ishant	Sharma

Note that in the above query we have changed the ordering of the attributes to print the names of the players as first name followed by the last name. Similarly, a list of player, last names and their player ID could be obtained by applying the projection given in Fig. 4.6 to the table *Player*, Table 3.2.

 $\text{Player}[\text{PlayerID}, \text{LName}] \text{ or } \pi_{\text{PlayerID}, \text{LName}}[\text{Player}]$
Figure 4.6 Another example using the projection operator

The result of applying the projection operation given in Fig. 4.6 to Table 3.2 is presented in Table 4.4. To save space we only show the first 18 players, not all the 27 players.

To find the player IDs of all the players that have batted in either match in the database, we apply the projection given in Fig. 4.7 to the Table *Batting*, Table 3.3.

 $\text{Batting}[\text{PID}] \text{ or } \pi_{\text{PID}}[\text{Batting}]$
Figure 4.7 Projection applied to the *Batting* table

Table 4.4 The result of applying the projection in Fig. 4.6 to Table 3.2

<i>PlayerID</i>	<i>LName</i>
89001	Tendulkar
90001	Lara
95001	Ponting
96001	Dravid
96002	Gibbs
92001	Warne
95002	Pollock
99003	Vaughan
92003	Ul-Huq
94004	Fleming
93002	Streak
90002	Kumble
93003	Kirsten
91003	Kallis
94002	Vaas
92002	Muralitharan
97004	Vettori
More...	More...

The result of applying the projection given in Fig. 4.7 to Table 3.3 *Batting* is presented in Table 4.5.

We obtain Table 4.9 once the duplicate rows from the resulting table are removed. Also, it should be noted that appropriate names of the columns and tables must be used in the relational algebra expressions. The player ID is given the name *PID* in the table *Batting* to save space to fit the table on a page. Also, note that this projection cannot give us a list of player names that have batted since the table *Batting* does not include player names. We will later discuss how to obtain player names of those who batted using information from two tables, Tables 3.2 and 3.3.

It should be noted that the ordering of rows in Tables 4.7, 4.8 and 4.9 is not defined and should not be expected to be the same as displayed in Tables 3.1, 3.2 and 3.3.

Table 4.5 The result of applying the projection given in Fig. 4.7 to Table 3.3

<i>PID</i>
91001
94002
92002
89001
23001
25001
99002
95001
24001
99001
27001

The two unary operators, selection and projection, may be used together to first select some rows of a table and then select only some of the columns of the selected rows. We will consider such examples later in the chapter.

4.2.3 Binary Relational Operation—Cartesian Product

The *Cartesian product* (also known as the *cross product*) of any two tables R and S (of degree m and n respectively) is a table $R \times S$ of degree $m + n$. This product table has all the columns that are present in both the tables R and S and the rows in $R \times S$ are all possible combinations of each row from R combined with each row from S . The number of rows in $R \times S$ therefore is pq if p is the number of rows in S and q in R . As noted earlier, the number of columns of $R \times S$ is $m + n$ if m and n are the number of columns in R and S respectively.

Consider the Cartesian product of the tables *Player* and *Batting*. We first reproduce the schema for the two tables below:

$$\begin{aligned} \textit{Player}(\textit{PlayerID}, \textit{LName}, \textit{FName}, \textit{Country}, \textit{YBorn}, \textit{BPlace}, \textit{FTest}) \\ \textit{Batting}(\textit{MatchID}, \textit{PID}, \textit{Order}, \textit{HOut}, \textit{FOW}, \textit{NRuns}, \textit{Mts}, \textit{NBalls}, \textit{Fours}, \textit{Sixes}). \end{aligned}$$

The product will have the following attributes:

$$\textit{Product}(\textit{PlayerID}, \textit{LName}, \textit{FName}, \textit{Country}, \textit{YBorn}, \textit{BPlace}, \textit{FTest}, \textit{MatchID}, \textit{PID}, \textit{Order}, \textit{HOut}, \textit{FOW}, \textit{NRuns}, \textit{Mts}, \textit{NBalls}, \textit{Fours}, \textit{Sixes})$$

We cannot show what this table looks like since the number of columns is too large to fit on a page. The number of rows is also quite large. How many rows does the product have? Therefore, we will use simpler relations that have only a few columns and only a few rows as shown in Tables 4.6, 4.7 and 4.8. These tables have been given the names *Player2*, *Batting2* and *Bowling2* to differentiate them from *Player*, *Batting* and *Bowling* in Tables 3.3, 3.4 and 3.5.

Table 4.6 The relation *Player2*

<i>PlayerID</i>	<i>LName</i>	<i>FName</i>
89001	Tendulkar	Sachin
24001	Symonds	Andrew
99001	Lee	Brett
25001	Dhoni	MS

Table 4.7 The relation *Batting2*

<i>MID</i>	<i>PID</i>	<i>Order</i>	<i>NRuns</i>
2689	89001	2	91
2689	99001	8	7
2689	24001	5	42
2755	25001	5	71

Table 4.8 The relation *Bowling2*

<i>MatchID</i>	<i>PID</i>	<i>NWickets</i>
2689	99001	1
2755	91001	0
2689	24001	1

Table 4.9 shows the Cartesian product of Tables 4.6 and 4.7.

Table 4.9 Cartesian product of tables *Player2* and *Batting2*

PlayerID	LName	FName	MID	PID	Order	NRuns
89001	Tendulkar	Sachin	2689	89001	2	91
24001	Symonds	Andrew	2689	89001	2	91
99001	Lee	Brett	2689	89001	2	91
25001	Dhoni	MS	2689	89001	2	91
89001	Tendulkar	Sachin	2689	99001	8	7
24001	Symonds	Andrew	2689	99001	8	7
99001	Lee	Brett	2689	99001	8	7
25001	Dhoni	MS	2689	99001	8	7
89001	Tendulkar	Sachin	2689	24001	5	42
24001	Symonds	Andrew	2689	24001	5	42
99001	Lee	Brett	2689	24001	5	42
25001	Dhoni	MS	2689	24001	5	42
89001	Tendulkar	Sachin	2755	25001	5	71
24001	Symonds	Andrew	2755	25001	5	71
99001	Lee	Brett	2755	25001	5	71
25001	Dhoni	MS	2755	25001	5	71

The number of columns of the Cartesian product in Table 4.13 is 7 ($=3 + 4$) and the number of rows is 16 (4×4). The product may not appear to be a terribly useful operator to the reader since it combines many rows that have no association with each other, except the four rows highlighted. We will now show that a Cartesian product followed by a selection operation that selects the highlighted rows is a very important operator called the *join*. This is discussed below.

4.2.4 Binary Relational Operator—Join

The join operator may be considered as the most powerful of the relational algebra operations. The join of two related tables R and S ($R \bowtie S$) is a restriction of their Cartesian product $R \times S$ such that a specified condition is met. The result is a single table. The join is normally defined on a column a of table R and a column b of table S , such that the attributes are from the same domain and are therefore related. The specified condition (called the *join condition* or *join predicate*) is a condition on columns $R.a$ and $S.b$ ³.

The most commonly needed join is the join in which the specified condition is equality of the two columns ($R.a = S.b$) in $R \times S$. This join is called an *equi-join*. The rows highlighted in Table 4.9 are the rows that

3. The terminology $R.x$ is used to denote column x of table R .

form the equi-join. Since the equi-join by definition has two equal columns one of these may be removed by applying the projection operator. The resulting table is called the *natural join*.

Equi-Join

As noted earlier, the number of columns in the equi-join is $m+n$ if the degrees of R and S are m and n respectively. The degree of the natural join is $m + n - 1$ since one column has been removed from the equi-join. The number of rows in the result table of the join depends on the number of rows that satisfy the join condition.

To illustrate equi-join and natural join, we consider the tables *Player2* and *Batting2* (Tables 4.10 and 4.11). There is an implicit association between the two tables since the table *Player2* provides personal information about players while *Batting2* provides information about their batting. One may be wondering why the two tables are separate. The reason becomes clear if we consider what would happen when the two tables are combined. The personal information of a player is now repeated for each *Batting* (and *Bowling*) row for the player. This is not desirable not only because it wastes storage. This is discussed in more detail in Chapter 6.

The Cartesian product of *Player2* and *Batting2* has two columns that are from the domain *PlayerID*. If the column names in the two tables were the same (they are not in this case), we would need to give one of them a different name in the Cartesian product since each column name in a table must be unique.

If we look at Table 4.9, we find that the Cartesian product table has many rows that are unlikely to be of any use. The relationship between the two Tables 4.6 and 4.7 is based on the two columns *PlayerID* and *PID* and therefore all the rows in the Cartesian product table that do not have the same values of these two columns are of little use.

If we apply a selection to the Table 4.9 such that the values of the two attributes *PlayerID* and *PID* (the highlighted columns) are the same, we obtain Table 4.10.

Table 4.10 Equi-join of tables *Player2* and *Batting2*

<i>PlayerID</i>	<i>LName</i>	<i>FName</i>	<i>MID</i>	<i>PID</i>	<i>Order</i>	<i>NRuns</i>
89001	Tendulkar	Sachin	2689	89001	2	91
99001	Lee	Brett	2689	99001	8	7
24001	Symonds	Andrew	2689	24001	5	42
25001	Dhoni	MS	2755	25001	5	71

Note that players will not appear in the above table if they have not batted in a match in our database. Also a player appearing in the *Batting2* table and not in the *Player2* table (which should not normally happen) will also not appear in the equi-join table.

Table 4.10 is an *equi-join* since it has been obtained from the Cartesian product by applying the restriction that the values of the two columns that are from the same domain be equal. These are the columns on which the tables are said to be joined. Note that a player may appear in Table 4.6 *Player2* but may not appear in the join Table 4.10 if he did not have an entry in Table 4.7 *Batting2*.

Natural Join

If we now apply a projection to Table 4.10 so that one of the two identical columns is removed, the resulting table is the *natural join* of the Table 4.6 and Table 4.7. The natural join is shown in Table 4.11.

Table 4.11 Natural join of tables *Player2* and *Batting2*

PlayerID	LName	FName	MID	Order	NRuns
89001	Tendulkar	Sachin	2689	2	91
99001	Lee	Brett	2689	8	7
24001	Symonds	Andrew	2689	5	42
25001	Dhoni	MS	2755	5	71

Table 4.11 provides the names and the runs scored in each match by the players in Table 4.6 and 4.7.

One might be wondering if a natural join is possible between two tables when the tables have more than one column in common. The formal definition of natural join does in fact include that possibility. The natural join may be defined as follows.

Definition—Natural Join

The natural join of two tables R and S (written as $R \bowtie S$) is obtained by applying a selection and a projection to the Cartesian product $R \times S$ as follows:

1. For each column 'a' that is common to both tables R and S , we select rows that satisfy the condition $R.a = S.a$.
2. For each column 'a' that is common to both tables R and S , we project out the column $S.a$. Therefore if there are 'm' columns common to both tables, 'm' duplicate columns are removed from the Cartesian product.

4.2.5 Join on More Than One Column

To illustrate the natural join of tables that have more than one common column, we consider the tables *Batting2* and *Bowling2*. These two tables have two columns in common, namely the match ID and the player ID. What would be the meaning of joining *Batting* and *Bowling* requiring that both the columns be equal?

We first present the Cartesian product of the two relations (Tables 4.7 and 4.8) in Table 4.12. Note that we have to use the column name *BPID* since duplicate column names are not allowed.

To obtain the natural join we first find the equi-join by deleting all the rows in which the match IDs and player IDs do not match. The highlighted rows are the only rows where both columns match and we obtain Table 4.13 by applying a selection to obtain the highlighted two rows. All other rows in the table are deleted as a result of the equi-join condition.

Table 4.12 Cartesian product of *Batting2* and *Bowling2*

<i>MID</i>	<i>PID</i>	<i>Order</i>	<i>NRuns</i>	<i>MatchID</i>	<i>BPID</i>	<i>NWickets</i>
2689	89001	2	91	2689	99001	1
2689	99001	8	7	2689	99001	1
2689	24001	5	42	2689	99001	1
2755	25001	5	71	2689	99001	1
2689	89001	2	91	2755	91001	0
2689	99001	8	7	2755	91001	0
2689	24001	5	42	2755	91001	0
2755	25001	5	71	2755	91001	0
2689	89001	2	91	2689	24001	1
2689	99001	8	7	2689	24001	1
2689	24001	5	42	2689	24001	1
2755	25001	5	71	2689	24001	1

Table 4.13 Equi-join of the tables *Bowling2* and *Batting2* on two common attributes

<i>MID</i>	<i>PID</i>	<i>Order</i>	<i>NRuns</i>	<i>MatchID</i>	<i>BPID</i>	<i>NWickets</i>
2689	99001	8	7	2689	99001	1
2689	24001	5	42	2689	24001	1

To obtain the natural join, the two duplicate columns that are highlighted in Table 4.13 are removed from the table and we obtain Table 4.14 as the result:

Table 4.14 Natural join of the tables *Bowling2* and *Batting2* on two common attributes

<i>MID</i>	<i>PID</i>	<i>Order</i>	<i>NRuns</i>	<i>NWickets</i>
2689	99001	8	7	1
2689	24001	5	42	1

So what does Table 4.14 tell us?

Essentially this table provides us with information for those players that have both batted and bowled in a particular ODI match. The table provides us with their batting performance as well as their bowling performance. If a player only bowled in a match and did not bat in that match or only batted but did not bowl then that player would not appear in this table.

Let us summarize. We obtained the natural join by taking the Cartesian product and then removing those rows that did not have matching values of the common columns resulting in an equi-join followed by removal of the duplicate attributes. The result is called the natural join.

We repeat that an equi-join or a natural join of two matching tables is possible only if both tables have one or more columns that are from the same domains. It would make no sense to have an equi-join on number of wickets in *Bowling2* with batting order in *Batting2*, even if both columns are integers.

Before concluding this discussion of the operation join, we note that when a join is needed to process a query, the procedure used is much more efficient than that which we have used to illustrate the join above in tables 4.10 to 4.14. A simple and more efficient join algorithm involves the following steps:

1. Take the first row from the first table.
2. Search the second table for a row that matches the value of the column on which the tables are being joined in the first table. When a match is found, the two rows are put together to form a row of the join table.
3. The search of the second table is continued for further matches till the second table is exhausted.
4. Now, the next row of the first table is selected and steps 2 and 3 are carried out. This process is continued until the first table is exhausted.

This algorithm is called the *nested scan* method.

In the above algorithm, we have assumed that we are forming an equi-join. A number of other join algorithms are available to process equi-joins more efficiently. These algorithms will be discussed in Chapter 7.

As noted earlier, a major criticism of the relational model for several years was that systems based on the model would always be inefficient because of the high cost of performing joins. In recent years it has been shown that joins can be computed efficiently, particularly so when indexes are available. This has resulted in a number of commercial relational DBMS products that have been shown to have very good performance.

In addition to the operators that we have discussed, there are a number of other relational operators. These are discussed now.

4.2.6 Theta Join

A join of R and S with join condition $a = b$ or $a < b$ is usually written as in Fig. 4.8.

Let *theta* be one of the operators \diamond , $<$, $>$, \geq , \leq . We may now define a more general concept of *theta-join* of table R (on attribute a) and table S (on attribute b). The theta-join is a restriction on the Cartesian product of R and S such that the condition $a \theta b$ is satisfied where θ is one of the above operations. We do not consider theta join any further.

 $R[a = b]S$ or $R[a < b]S$

Figure 4.8 Specifying the join condition

4.2.7 Outer Join

The join operations presented so far only deal with tuples from the two tables that match the join condition. The remaining tuples from both tables are eliminated in computing the join although these tuples can often have useful information. For example, if we are joining the *Player* table with the *Batting* table using *PID* equality as the joining condition then a player that has never batted will not be included in the join. This might be useful information in some situations. For instance, if a student table is joined with an enrolment table,

we may be interested in finding out the names of students who have not enrolled. In joining enrolment with courses, we may be interested in finding names of courses that have no enrolments.

There are a number of variants of the outer join. As noted above, the join of the *Player* table with the *Batting* table using *PlayerID* equality as the joining condition does not include the players that have never batted. A join including this information is called the *left outer join*. If the join also included batting information on players that batted but were not in the *Player* table (normally there should be no such rows in *Batting*) then the join is called a full outer join. If the join did not include information about players that did not bat but included information about players that batted but were not in the *Player* table then the join is called a *right outer join*.

We leave it as an exercise to find the left outer join of *Player2* and *Batting2*. Outer join is discussed again in Chapter 5 with its implementation in SQL.

4.2.8 Set Operations

Set Union

The union operator is the usual set union of two tables *R* and *S* which are union-compatible. Two tables are called union-compatible if they are of the same degree *m* and are written as

$$\begin{aligned} R(a_1, a_2, \dots, a_m) \\ S(b_1, b_2, \dots, b_m) \end{aligned}$$

For each *i* (*i* = 1, 2, ..., *m*), *a_i* and *b_i* must have compatible domains (e.g., either both are integers or both are strings of the same length). The number of columns in the union of *R* and *S*, *R* ∪ *S*, is the same as that of *R* and *S* and the number of rows is *a* + *b* if *a* and *b* are the number of rows in *R* and *S* respectively and there are no duplicate rows in *R* and *S*. *R* ∪ *S* therefore consists of all rows that are either in *R* or *S* or common in both *R* and *S*. As an example, consider two tables, Table 4.15 and Table 4.16, given below obtained by applying projection to the tables *Batting* (Table 3.3) and *Bowling* (Table 3.4). Table 4.15 *Batting3* gives the columns *MatchID* and *PID* from the table *Batting* while the projection on *Bowling* results in *Bowling3* with columns *MatchID* and *PID* in Table 4.16.

If we were interested in finding all the players who have batted or bowled or both, we can take a union of the two tables *Batting3* and *Bowling3* (Tables 4.15 and 4.16) and obtain the result in Table 4.17.

There are many duplicates in Table 4.17 and the duplicate pairs have been highlighted in the table and therefore the table does not satisfy the definition of a relation. We have not removed duplicates from the above union to illustrate that duplicates may occur after union. The rows have been sorted to make it easier

Table 4.15 *Batting3* obtained by projection of the Table 3.3 *Batting*

<i>MatchID</i>	<i>PID</i>
2755	23001
2755	25001
2755	91001
2755	94002
2755	92002
2689	89001
2689	23001
2689	25001
2689	99002
2689	95001
2689	24001
2689	99001
2689	27001
2755	27001

Table 4.16 *Bowling3* obtained by projection of the Table 3.4 *Bowling*

<i>MatchID</i>	<i>PID</i>
2689	99001
2689	24001
2689	23001
2755	94002
2755	92002
2755	91001
2755	23001

to identify duplicates which must be removed if the result is to satisfy the properties of a relation. There are 14 duplicate rows and therefore seven of them need to be removed.

A projection of the above table, Table 4.17, on attribute *PID* now provides the list of players that have either batted or bowled or both in the matches in the database. The duplicates have been removed and the list in Table 4.18 has been sorted. Table 4.18 therefore has only 11 rows after removing duplicate rows for players 23001, 24001, 99001, 91001, 92002 and 94002.

Table 4.17 Union of Tables 4.15 and 4.16

<i>MatchID</i>	<i>PID</i>
2689	23001
2689	23001
2689	24001
2689	24001
2689	25001
2689	27001
2689	89001
2689	95001
2689	99001
2689	99001
2689	99002
2755	23001
2755	23001
2755	25001
2755	27001
2755	91001
2755	91001
2755	92002
2755	92002
2755	94002
2755	94002

4.2.9 Set Intersection

The intersection relational operator is the usual set intersection of two tables *R* and *S* which are union-compatible. The result of the intersection is a table consisting of all the rows that are common to both tables *R* and *S*. The number of columns in the intersection $R \cap S$ is the same as that of *R* and *S* and the number of rows in the intersection is equal to the number of rows common to both *R* and *S*.

As an example, let us again consider the earlier example of Tables 4.15 and 4.16, *Batting3* and *Bowling3*. Suppose now that we wish to find those players that bowled as well as batted in the same matches. One way is to formulate the query would be to find the intersection of the two tables. The intersection is formulated in Fig. 4.9.

Table 4.18 Projection of Table 4.17 which was a union of two tables

<i>PID</i>
23001
24001
25001
27001
89001
91001
92002
94002
95001
99001
99002

$Bowling3 \cap Batting3$

Figure 4.9 Intersection of the tables *Batting3* and *Bowling3*

The result of the intersection is a table with two columns (*MatchID*, *PlayerID*) for the players that are common to both tables *Bowling3* and *Batting3*, and hence have batted and bowled in the same match. The result of the intersection is given in Table 4.19.

Note that Table 4.19 is the same as Table 4.16 since every player that bowled in that table also batted in Table 4.15.

Furthermore, we should note that we could not take an intersection of the tables *Batting2* and *Bowling2* (Tables 4.7 and 4.8) since the number of columns in those two tables are not the same.

4.2.10 Set Difference

The difference operator when applied to two tables *R* and *S* (written as $R - S$) results in a table that consists of all the rows in the first table that are not also in the second table. If the intersection of *R* and *S* is null then $R - S = R$, otherwise $R - S = R - R \cap S$, that is, $R - S$ is obtained by removing from *R* the rows that are also in *S*. The number of columns in $R - S$ is the same as that of *R* and *S* and the number of rows in the intersection is equal to the number of rows of *R* minus the number of rows in the intersection $R \cap S$.

We again consider the example of the two Tables 4.15 and 4.16. If we want to find the players that batted in a match but did not bowl in it, we could formulate the query by the difference as presented in Fig. 4.10.

The result given in Table 4.20 is obtained when we take the difference of Tables 4.15 and 4.16.

Let us now look at the meaning of the difference *Bowling3*—*Batting3*. This difference would be a table that consists of all the rows in the first table that are not also in the second table. Therefore it is the list of players that have bowled but not batted in a match. For Tables 4.15 and 4.16, the difference *Bowling3*—*Batting3* is null since there are no bowlers in Table 4.16 that have not also batted.

4.2.11 Division⁴

The concept of division is related to the concept of Cartesian product in that $R \times S$ divided by *S* yields *R*. Division therefore is essentially an inverse of Cartesian product in the same way as arithmetic division is the inverse of multiplication.

Table 4.19 Result of the intersection operation in Fig. 4.9

MatchID	PID
2689	99001
2689	24001
2689	23001
2755	94002
2755	92002
2755	91001
2755	23001

$Batting3 - Bowling3$

Figure 4.10 Difference of two tables

Table 4.20 Difference of the Tables 4.15 and 4.16, *Batting3* and *Bowling3*

MatchID	PID
2689	25001
2689	27001
2689	89001
2689	95001
2689	99002
2755	25001
2755	27001

4. Part of this section may be omitted without loss of continuity.

Example

Before discussing the division operator further, we present an example.

Consider a table R and a table S as given in Tables 4.21 and 4.22.

To simplify our example, we have chosen a very simple table S that consists of only two rows as given in Table 4.22.

Table 4.22 The table S

PID
99001
27001

The result of dividing relation R by relation S is given below in Table 4.23.

Table 4.23 gives the match IDs (in this case, only one) in which both players listed in Table 4.22 have played. That is the function of the division operator.

We first define division for a simple case when a table R has only two columns and the table S has only one column. Let the domain of the second column of R be the same as of the column of S . The division of R by S , or R/S is then a table with only one column that has the same domain as the first column of R , such that each row in R/S occurs in R concatenated with every row of S .

When the tables R and S are of degree greater than 2, division can be defined similarly. For example, if R was $R(a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n)$ and S was $S(b_1, b_2, \dots, b_n)$ then R/S would be a table $T(a_1, a_2, \dots, a_n)$ such that R had at least all rows that $T \times S$ would have. We may write R as having columns $R[A, B]$ and S having column B only where A and B each could consist of more than one column.

Finding those values of attribute A that appear with each value of B in S is not directly possible in relational algebra because relational algebra does not have a direct divide operator. Therefore the division algorithm involves finding values of attribute A in R that do not appear with all values of B in S . This is complex but is possible as shown below. R divided by S is then given by a relation T given the relational algebra expression in Fig. 4.11.

Let us now explain what the above difficult to understand expression means. Before the explanation, we repeat that $R[A, B]/S[B]$ is a table $T[A]$ such that the relation R includes each value of A in T with every value of B in the table S . How does the expression in Fig. 4.11 find those values of A ? We will use a step-by-step description of the expression in Fig. 4.11.

Table 4.21 The table R

MatchID	PID
2755	23001
2755	25001
2755	91001
2755	94002
2755	92002
2689	89001
2689	23001
2689	25001
2689	99002
2689	95001
2689	24001
2689	99001
2689	27001
2755	27001

Table 4.23 The result of dividing table R by table S

MatchID
2689

$$T = R[A] - ((R[A] \times S) - R)[A]$$

Figure 4.11 Relational algebra expression for division

1. We first find a table that has each value of attribute A that exist in R appearing with every value of attribute B that exists in S . This can be done by first finding all values of A in R which is $R[A]$ and then computing $R[A] \times S$, the Cartesian product of $R[A]$ and S . The product has two columns, A and B , same two columns as R does, with each value of column A in R combined with every value of column B in S .
2. We now find the values of attribute A that do not appear with each value of attribute B in S . This can be done by using the product relation found above (the product has each value of column A in R combined with every value of column B in S) and subtract the relation R from it. That means we find $R[A] \times S - R$ which is a table that has only those rows that are in the Cartesian product but not in R . In other words, these are the rows that are not in R because the A value in each of these rows does not appear with each value of B that appears in S .
3. Carry out a projection to find the values of attributes A that do not appear with every value of attributes B in S . This is simply done by taking the projection $(R[A] \times S - R)[A]$ which is a table with only one column, A . It only has those values of column A that are present in R but which do not appear with every value of column B in S .
4. Find the result of the division. To obtain the result we subtract the values of A found above from values of A in R since that gives us the values of A that do appear with every value of B in S . Therefore $R[A] - (R[A] \times S - R)[A]$ are the values of column A that are present in R and each of which appears with every value of B in S . Hence this is the result of dividing R by S .

Let us now again consider the tables R and S given in Tables 4.21 and 4.22 above.

We now illustrate how the formula $T = R[A] - ((R[A] \times S) - R)[A]$ is used in computing the division. $R[A]$ is given in Table 4.24. All the duplicates have been removed. There are only two ODI matches in R .

Now the Cartesian product $R[A] \times S$ is given in Table 4.25. This table shows what R would look like if all the players in S had played in all the matches in R .

Now $(R[A] \times S - R)$ is given by Table 4.26. These are rows that are in the product $R[A] \times S$ but not in R . That is, these rows show the matches in which players in S did not play. There is only one such row in this table.

Now we take the projection of Table 4.26 on $MatchID$ and subtract it from column A of R . This gives us the result of the division.

The division R/S is given by $R[A] - (R[A] \times S - R)[A]$ which is now given by Table 4.27.

Table 4.24 The table $R[A]$

MatchID
2755
2689

Table 4.25 The table $R[A] \times S$

MatchID	PID
2755	99001
2689	99001
2755	27001
2689	27001

Table 4.26 The table $R[A] \times S - R$

MatchID	PID
2755	99001

Table 4.27 The table $R/S = T = R[A] - (R[A] \times S - R)[A]$

MatchID
2689

This is the result of the division. It is a list of all matches that the players in S given in Table 4.22 have played together. In this case, for simplicity, Table 4.22 had only two players.

We note that $R/S = T$ does not imply that $S \times T = R$, since R could well include other rows. These rows are discarded in the same way as the remainder is discarded in the integer division.

The division of R by S where the attributes x in R and y in S are union-compatible is written as in Fig. 4.12.

If the division also involves a projection operator as is often necessary, then it may be written as given in Fig. 4.13.

This concludes our discussions of relational algebra operators.

The relational algebra described above does not include capabilities like counting and summing. Such capabilities are obviously needed in a practical system. They are included in the commercial language SQL which is discussed in the next chapter.

$R[x/y]S$

Figure 4.12 Division of R by S on attributes x and y

$R[Ax][x/y]S[y]$

Figure 4.13 Division involving projections

4.3 RELATIONAL ALGEBRA QUERIES

We now present a number of queries on the cricket database presented in Tables 3.1 to 3.4. The queries range from very simple to quite complex and have been formulated using relational algebra that we have discussed.

4.3.1 Only Projection

- (A1) Find the match IDs of matches that do have some bowling.

This is a simple query that only involves a projection on table *Bowling*. It is formulated in Fig. 4.14.

$Bowling[MatchID]$

Figure 4.14 Relational algebra formulation for query A6

The result is given in Table 4.28. Note that all duplicates have been removed.

Table 4.28 Result of query A1 - Match IDs of matches that do have some bowling

MatchID
2689
2755

4.3.2 Selection combined with Projection

- (A2) Find the names of all players who played their first test after 2000.

In relational algebra notation, the query may be written as in Fig. 4.15⁵

$Player[Fname, LName][FTest > 2000] \text{ or } \pi_{Fname, LName}(\sigma_{FTest > 2000}[Player])$
--

Figure 4.15 Relational Algebra for query A2

Note that the query in Fig. 4.15 involves a projection which projects the result to display the first and last name of the selected players. Before the projection a selection is carried out which selects the players who played their first test after 2000. Clearly the projection cannot be done before the selection because the projection would then have discarded the FTest column.

- The result of query in Fig. 4.15 is given in Table 4.29.
- (A3) Find the player IDs of all players who batted in the match 2755.

This query involves a selection and a projection to be applied to the table *Batting*. We carry out the selection first to find all players that have batted in match 2755 and then carry out the projection to get their IDs. The projection could not be carried out before the selection since the *MID* column will then be lost in projection and we could not apply selection.

The query may be written as given in Fig. 4.16.

The result of relational algebra operation in Fig. 4.16 is given in Table 4.30.

4.3.3 Using the Join of Two Tables

- (A4) Find the names of all players who batted in match 2755.

This is a more complex query since it involves information that cannot be accessed without looking at two tables since the table *Batting* has the batting information and the table *Player* has information about players' names. To answer the query, we carry out a natural join of the tables *Player* and *Batting* on the attribute *PlayerID* and then apply selection *MatchID* = 2755. This is followed by a projection on attributes *FName* and *LName*.

In relational algebra notation, we may write the above query as in Fig. 4.17.

We are using the symbol \bowtie to indicate the natural join operator. The result of the query posed in Fig. 4.17 is given in Table 4.31.

Table 4.29 Result of the projection in query A2

FName	LName
MS	Dhoni
Yuvraj	Singh
Andrew	Symonds
Virender	Sehwag
Ishant	Sharma

$Batting[MatchID = '2755'][PID]$

Figure 4.16 Relational algebra formula for query A3

Table 4.30 Result of the selection and projection in query A3

PID
23001
25001
91001
94002
92002
27001

5. The Greek symbol notation for relational algebra queries are only used for some selected queries.

$$(Player [PlayerID = PID] \text{ Batting}) [\text{MatchID} = '2755'] [Fname, LName]$$

or $\pi_{Fname, Lname} [\sigma_{\text{MatchID} = '2755'} (Player \bowtie \text{Batting})]$

Figure 4.17 Relational algebra expression for query A4**Table 4.31** Result of query A4—Names of all players who batted in match 2755

FName	LName
Chaminda	Vaas
Muthiah	Muralitharan
MS	Dhoni
Yuvraj	Singh
Sanath	Jayasuriya
Praveen	Kumar

These are the names of players whose player IDs were in Table 4.30.

4.3.4 Using Joins of Three Tables

- (A5) Find the ground names and dates of all matches in which any player with last name Dhoni has batted.

This query is even more complex since it must involve the three tables *Match*, *Player* and *Batting*. The player name information is in table *Player*, the batting information is in table *Batting* and the ground names are available in table *Match*. One approach to formulating this query is to join *Player* and *Batting* on attribute *PlayerID* and then join the resulting table with table *Match* on the attribute *MatchID*. We then apply a selection that the player name is Dhoni and then take the projection to find names of matches. We assume that the attributes *Ground* and *Date* provide the information required. In relational algebra, the query may be written as in Fig. 4.18.

$$(Player \bowtie \text{Batting}) \bowtie \text{Match} [\text{LName} = 'Dhoni'] [\text{Ground}, \text{Date}]$$
Figure 4.18 Relational algebra formula for query A5

The result of query formulated in Fig. 4.18 is given in Table 4.32.

Table 4.32 Result of query A5—Ground names and dates of all matches in which Dhoni has batted

Ground	Date
Brisbane	4/3/2008
Colombo	27/8/2008

4.3.5 Using Difference

- (A6) Find the player IDs of all Indian players who have not batted in any match.

Since *Batting[PID]* is the list of players that have batted in one or more of the matches, the difference between *Player[PlayerID]* of Indian players and *Batting[PID]* is the list of Indian players that have not batted in any match in the database because the difference consists of all the rows in the first table that are not also in the second table.

The above query may therefore be formulated as follows in Fig. 4.19.

$$(Player[Country = 'India']) [PlayerID] - Batting [PID] \text{ or} \\ \pi_{PlayerID}[\sigma_{Country=india} Player] - \sigma_{PID}[Batting]$$

Figure 4.19 Relational algebra expression for query A6

Note that the selection is done first in the query in Fig. 4.19.

We are only considering information in Tables 3.2 and 3.3 which include only part of the information of the two ODI matches. For example, Harbhajan Singh's ID is not in the result table below although he did bat in match 2755.

The result is given in Table 4.33.

- (A7) Find the player IDs of players who have bowled only in match 2755.

To find players who have bowled only in match 2755 we must ensure they have not bowled in any other match in our very small database. Therefore, the query involves finding the difference between the set of players who have bowled in match 2755 and those who have bowled in matches other than 2755. Any player who has bowled in 2755 as well as other matches will appear in both sets and will therefore not appear in the difference.

$$Bowling[MatchID = '2755'] [PID] - Bowling[MatchID <> '2755'] [PID]$$

Figure 4.20 Relational algebra expression for query A7

The result of query posed in Fig. 4.20 is given in Table 4.34.

- (A8) Find the match IDs of matches that include all Indian players in our player table that were born after 1985.

We first find the list of all Indian players born after 1985 by applying a restriction and a projection to the table *Player*. Then we find a list of all player IDs and match IDs for all players that have batted and divide

Table 4.33 Result of query A6—Player IDs of Indian players who have not batted in any match

PlayerID
96001
90001
96003
21001
98002
27002

Table 4.34 Result of query A7—Player IDs of players who have bowled only in match 2755

PID
94002
92002
91001
23001

that list by the list of Indian players born after 1985 in the database. The result of the division will be those matches that had all Indian players batting in them.

$Batting[PID, MatchID] / (Player[Country = 'India'] \wedge Yborn > 1985) [PlayerID]$
or $\pi_{PID, MatchID}[Batting] / \pi_{PlayerID}(\sigma_{Country = 'India'}[Player])$

Figure 4.21 Relational algebra expression for query A8

The result is given in Table 4.35.

The above queries would have given the reader a flavour of how relational algebra can be used to answer user queries. We would like to note that the formulations presented above are not the only possible formulations for the queries that we had. A query may often be formulated in more than one way.

A question that often arises at this point is, are there better (that is, more efficient) sequences of operations than those listed above to answer the queries? For example, in query A3 above, could we carry out the restriction on the table *Player* before the join? This is clearly an important question but we will not deal with it here. There are usually a number of different sequences that may be used to process a query. Some of these sequences would obviously be more efficient than others. This topic is presented in detail in Chapter 7 on query optimization.

Relational algebra is like the assembler programming language. Just as there are many higher level programming languages like C++, there are other relational query languages as well. We discuss relational calculus next since this was another language proposed by E. F. Codd.

Table 4.35 Result of query A8—Match IDs of matches that include Indian players born after 1985

MatchID
2689
2755

4.4 RELATIONAL CALCULUS⁶

Relational algebra provides a collection of operators and a relational algebra expression explicitly states what operations are to be done first and followed by other operations, and so on. Relational algebra, therefore provides a procedural language that can be used in specifying a sequence of operations that ought to be used in obtaining the result that the user wants. Algebra is thus like a procedural programming language where a program written in the language clearly provides a sequence of instructions that the machine follows to obtain the result. In contrast, relational calculus involves specifying conditions (predicates) that the result must satisfy. Relational calculus, therefore is a nonprocedural language for defining the table of information that the user wants to retrieve. The commercial query language SQL is also a nonprocedural language based on relational calculus.

We will briefly discuss how to define conditions using predicate calculus since it provides the basis for relational calculus. Predicate calculus is a powerful technique for logical reasoning. To introduce it, we need to discuss propositional logic first. The reader need not worry about terms like ‘predicate calculus’ or ‘propositional logic’ since they are simple concepts that are explained ahead. The background given in the next few pages can be skipped by readers not interested in the background.

6. The first part of this section may be skipped without any loss of continuity.

4.4.1 Propositions

A proposition is just a statement. It could be ‘Tendulkar has batted in match 2755’ or ‘Shane Warne was born in Melbourne’. These are atomic propositions and may be denoted by symbols like A and B and their value is either true or false. Compound propositions contain two or more atomic propositions.

A compound proposition with two components may be of the type (A AND B) or it may be of the type (A OR B). Such propositions are called *formulas*. Their truth depends on the truth values of their components. When a sequence of propositions is joined by ANDs (e.g., A AND B AND C) we call it a *conjunction* of propositions and the conjunction is true only if all atomic propositions are true. When a sequence of propositions is joined by ORs (e.g., A OR B OR C), we call it a *disjunction* of propositions. The disjunction is true when at least one of the atomic propositions is true and false only when all the propositions are false.

Since we are dealing with propositions that have values true or false, the theorems of Boolean algebra apply. We assume the reader is familiar with, for example, De Morgan’s Laws which are given in Fig. 4.22.

$$\begin{aligned} \text{NOT } (A \text{ AND } B) &= (\text{NOT } A) \text{ OR } (\text{NOT } B) \\ \text{NOT } (A \text{ OR } B) &= (\text{NOT } A) \text{ AND } (\text{NOT } B) \end{aligned}$$

Figure 4.22 De Morgan’s Laws

Consider the first expression. (A AND B) is true only if A and B both are true. It is false if either A or B is false or both A and B are false. The opposite of (A AND B), therefore is false only if (A AND B) is true which implies that A and B both are true or both ($\text{NOT } A$) as well as ($\text{NOT } B$) are false. We can similarly explain the second equation.

4.4.2 The Implies Operator

We now consider formulas that have the form ‘if A then B ’. The formula specifies that B must be true if A is. It does not say anything about the value of B if A is false. The truth table for ‘if A then B ’ is

It can be shown that *if A then B* is equivalent to *(not A) or B*. *If A then B* may be conveniently written as $A \rightarrow B$ (that is, A implies B). It may be shown that $A \rightarrow B$ is equivalent to $(\text{not } B) \rightarrow (\text{not } A)$.

The above notation can be quite useful in logical reasoning but is limited in its power of expression as we shall soon see. We first briefly note how the process of reasoning occurs. Reasoning is based primarily on two rules:

1. If A is true and we are given $A \rightarrow B$, then B must be true.
2. If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$.

Table 4.36 Truth table for ‘if A then B ’

A	B	$\text{If } A \text{ then } B$
T	T	T
T	F	F
F	T	T
F	F	T

The first rule is called *modus ponens* and the second rule is called the *chain rule*. Logical reasoning involves proving the truth of a proposition, given a set of propositions that are true. For example, we may be given propositions as follows:

1. If it rains (A), it must be cloudy (B).
2. If it is cloudy (B), I don’t play golf (C).

3. It is raining (A)
4. Derive – I don't play golf (C)

The above propositions may be written as follows:

- $A \rightarrow B$
- $B \rightarrow C$
- A is true
- B is therefore true given $A \rightarrow B$
- C is therefore true given $B \rightarrow C$

Similarly, we may be given a set of propositions about players and their batting and bowling and we may, for example, wish to prove the truth in ‘Dravid has played with Tendulkar’.

Sometimes we may face a situation, such that the proposition that we wish to prove can be shown to be true for all values of atomic propositions in it. Such a proposition is called *tautology*.

As we noted earlier, propositional calculus is useful but limited in power. In the next two sections we present facilities that improve the power of propositional logic. The more powerful logic is called *predicate calculus* or *first order logic*.

4.4.3 Parameterized Propositions

Propositions may be generalized by parameterization. For example, ‘Tendulkar played in match 2755’ may be written as $M2755(\text{Tendulkar})$ and its value is true if Tendulkar did play in match 2755. We assume $M2755$ is a predicate meaning ‘playing in 2755’. Similarly, $M2755(\text{Dravid})$ would be false if Dravid did not play in match 2755.

By introducing parameterization we have introduced a more compact notation. Rather than assigning new variables to all propositions regarding players who played in match 2755 we may now use $M2755(\cdot)$. Parameterization gives us a power similar to that provided by a looping facility in a programming language. If a programming language does not have any looping facility, we could still do what we wish but it would require writing similar statements over and over again, which would be very tedious. Parameterization helps us to overcome such tedious repetition.

A parameterized proposition is called a *predicate*.

Once we introduce parameters in propositions, we need a mechanism to specify what values these variables might take. This is done by the *universal* and *existential quantifiers*.

4.4.4 Universal and Existential Quantifiers

The universal quantifier (*for all* or \forall) and the existential quantifier (*there exists* or \exists) allow use of variables in the predicates enabling very compact representation of propositions. Suppose we want to express the following statement ‘If a player is a captain in match 2755 then he is a player in that match’, we may write it as in Fig. 4.23.

$$\forall x(C2755(x) \rightarrow M2755(x))$$

Figure 4.23 Using the universal quantifier

We assume $C2755(x)$ in Fig. 4.32 is a proposition that x is a captain in match 2755 and $M2755(x)$ states that x has played in match 2755.

Similarly, to express the statement that ‘some players in match 2755 are captains’, we may write as in Fig. 4.24

$$\exists x(M2755(x) \text{ and } C2755(x))$$

The statement in Fig. 4.33 means ‘there is at least one person who is a player in match 2755 and is a captain in that match’.

Figure 4.24 Using the existential quantifier

It should be noted that the statement in Fig. 4.25 following is not equivalent to the above statement

The reason is as follows: the statement in Fig. 4.25 means ‘*there does exist a player for whom $M2755(x) \rightarrow C2755(x)$ is true*’. This will always be true since one of the players will always be a captain.

$$\exists x(M2755(x) \rightarrow C2755(x))$$

Figure 4.25 Using the existential quantifier

This leads us to a rule of thumb in writing predicate calculus statements; a universal quantifier usually goes with an implication while the existential quantifier goes with a conjunction.

We note that expressions containing the universal quantifier may be replaced by equivalent expressions containing existential quantifiers. For example, consider the formula in Fig. 4.26.

$$\forall x(P(x)) \text{ is equivalent to } \text{Not}(\exists x(\text{Not } P(x)))$$

Figure 4.26 Relation between existential and universal quantifiers

Each variable appearing in a formula is either *free* or *bound*. The bound variables are those that have been defined in one of the quantifiers. Others are called free. For example, consider the formula in Fig. 4.27.

$$\forall x(P(x) \rightarrow R(x, y))$$

Figure 4.27 Using the universal quantifier

x is a bound variable while y is free. Bound variables are like formal parameters, the variable name being legal only within the scope of the variable and the name of the variable being immaterial. That is, the above formula in Fig. 4.36 may be written as in Fig. 4.28.

$$\forall a(P(a) \rightarrow R(a, y))$$

Figure 4.28 Using the universal quantifier

The formula in Fig. 4.28 has the same meaning as in Fig. 4.27. A formula that has no free variables is called a *sentence*.

4.4.5 Relational Calculus

Relational calculus is closely related to predicate calculus since tables in a relational database may be thought of as predicates. For example, given a table

$$\text{Player}(\text{PlayerID}, \text{LName}, \text{FName}, \text{Country}, \text{YBorn}, \text{BPlace}, \text{FTest})$$

the expression $\text{Player}(a, b, c, d, e, f, g)$ may be thought of as a predicate which is true if a row with values a, b, c, d, e, f, g exists in the table *Player*.

*image
not
available*

We now present a number of queries and formulate them in relational calculus.

- (C1) Find the names and IDs of all players.

The relational calculus formulation of query C1 is given in Fig. 4.31.

The above calculus expression specifies that we are interested in the values of the columns *FName*, *LName* and *PlayerID* of all rows *s*, such that *s* is in table *Player* (that is, all players). The result is presented in Table 4.37.

$\{s.Fname, s.Lname, s.PlayerID \mid s \in Player\}$

Figure 4.31 Tuple relational calculus expression for query C1

Table 4.37 Names and IDs of all players

<i>FName</i>	<i>LName</i>	<i>PlayerID</i>
Sachin	Tendulkar	89001
Brian	Lara	90001
Ricky	Ponting	95001
Rahul	Dravid	96001
Herschelle	Gibbs	96002
Shane	Warne	92001
Shaun	Pollock	95002
Michael	Vaughan	99003
Inzamam	Ul-Huq	92003
Stephen	Fleming	94004
Heath	Streak	93002
Anil	Kumble	90002
Gary	Kirsten	93003
Jacques	Kallis	95003
Chaminda	Vaas	94002
Muthiah	Muralitharan	92002
Daniel	Vettori	97004
MS	Dhoni	25001
Yuvraj	Singh	23001
Saurav	Ganguly	96003
Adam	Gilchrist	99002
Andrew	Symonds	24001
Brett	Lee	99001
Sanath	Jayasuriya	91001
Virender	Sehwag	21001
Shahid	Afridi	98001
Harbhajan	Singh	98002
Praveen	Kumar	27001
Ishant	Sharma	27002

- (C2) Find IDs of all players from India.

The tuple relational calculus formulation of query C2 is given in Fig. 4.32.

$\{e.PlayerID \mid e \in Player \wedge (e.Country = 'India')\}$

Figure 4.32 Tuple relational calculus expression for query C2

The above tuple relational calculus expression specifies that we wish to retrieve values of the column *PlayerID* of all rows *e*, such that *e* is a member of the table *Player* and the value of the column *Country* of *e* is India.

The result of the query in Fig. 4.32 is presented in Table 4.38.

Table 4.38 Result of query C2 posed in Fig. 4.32

<i>PlayerID</i>
89001
96001
90002
25001
23001
96003
21001
98002
27001
27002

- (C3) Find the names of all players from India who have batted in a match.

The tuple relational calculus formulation of query C3 is given in Fig. 4.33.

$\{s.FName, s.LName \mid s \in Player \wedge \exists y(y \in Batting \wedge s.PlayerID = y.PID \wedge s.Country = 'India')\}$

Figure 4.33 Tuple relational calculus formulation for query C3

The above expression specifies the columns that we wish to retrieve. The columns that we are interested in are *FName* and *LName* and we want all rows *s*, such that *s* is a member of table *Player* and there exists a row *y* that is in table *Batting* such that the value of the column named *PlayerID* in *s* which is in *Player* and *PID* in *y* which is in *Batting* is the same and the value of the column *Country* of row *s* is India. In other words, *s* is retrieved only if it names a player from India and there is an entry for this player in *Batting*.

The result is presented in Table 4.39.

Table 4.39 Result of query C3 posed in Fig. 4.33

FName	LName
Sachin	Tendulkar
Rahul	Dravid
Anil	Kumble
MS	Dhoni
Yuvraj	Singh
Saurav	Ganguly
Virender	Sehwag
Harbhajan	Singh
Praveen	Kumar
Ishant	Sharma

- (C4) Find player IDs of players who have batted in either match 2755 or 2689.
- The tuple relational calculus formulation of this query is given in Fig. 4.34.

$\{e.PlayerID \mid e \in Player \wedge \exists j(j \in Batting \wedge e.PlayerID = j.PID \wedge j.MatchID = '2755' \vee j.MatchID = '2689')\}$

Figure 4.34 Tuple relational calculus expression for query C4

We are now looking for values of the column *PlayerID* from rows *e*, such that *e* is a row in the table *Player* and there exists a row in table *Batting* for that player, such that the Match ID is either 2755 or 2689. Note that the above formulation will not retrieve any duplicates of *PlayerID* even if the player has batted in both matches 2755 and 2689 (why not?). This query could also have been formulated by specifying only the table *Batting*, such that PIDs could be retrieved from that table for rows that met the given condition.

The result of query C4 as formulated in tuple relational calculus in Fig. 4.34 is presented in Table 4.40. It should be noted that the result has no duplicates. We would obtain duplicates if we directly retrieve all the PIDs from the table *Batting* since some players have played in both matches 2755 and 2689.

- (C5) Find player IDs of players batting in both matches 2755 and 2689.

This query is little bit different than the queries we have seen so far and may be formulated as in Fig. 4.35.

Table 4.40 Result of query C4 as posed in Fig. 4.34

PID
23001
25001
91001
94002
92002
89001
99002
95001
24001
99001
27001

$$\{e. PlayerID \mid e \in Player \wedge \exists j, k (j \in Batting \wedge k \in Batting \wedge e. PlayerID = j. PID \wedge e. PlayerID = k. PID \wedge j. MatchID = '2755' \wedge k. MatchID = '2689')\}$$

Figure 4.35 Tuple relational calculus expression for query C5

In this query formulation, we are looking for values of the column *PlayerID* from rows *e*, such that *e* is a row in the table *Player* and there exist two rows *j* and *k* in table *Batting* for that player (the formulation ensures that player ID in all three rows *e*, *j* and *k* from the two tables are identical), such that one of them corresponds to match 2755 and the other to match 2689.

The result of query C5 is presented in Table 4.41.

- (C6) Find player IDs of players bowling in 2755 but not in 2689.

This query is also different than the two earlier queries and may be formulated as in Fig. 4.36.

$$\{e. PlayerID \mid e \in Player \wedge \exists j \wedge \text{not} (\exists k) (j, k \in Bowling \wedge e. PlayerID = j. PID \wedge e. PlayerID = k. PID \wedge j. MatchID = '2755' \wedge k. MatchID = '2689')\}$$

Figure 4.36 Tuple relational calculus expression for query C6

In this query formulation, we are looking for values of the column *PlayerID* from rows *e*, such that it is a row in table *Player* and there exists a row *j* in table *Bowling* for that player, such that the Match ID is 2755 and, in addition, there does not exist a row *k* in *Bowling* for him, such that the Match ID is 2689. Note the construct *not* ($\exists k$) which means that a row *k* meeting the condition specified does not exist.

The result of query C6 is presented in Table 4.42.

- (C7) Find the names of the cricket grounds that Dhoni has batted on.

This query is somewhat more complex because player names are in the table *Player*, batting information is in the table *Batting* and ground information is in the table *Match*. The query may be formulated in tuple relational calculus as presented in Fig. 4.37.

Table 4.41 Result of query C5 as posed in Fig. 4.35

PID
23001
25001
27001

Table 4.42 Result of query C6 as formulated in Fig. 4.36

PlayerID
91001
92002
94002

$$\{s. Ground \mid s \in Match \wedge \exists y, z (y \in Player \wedge z \in Batting \wedge z. PID = y. PlayerID \wedge z. MatchID = s. MatchID \wedge y. LName = 'Dhoni')\}$$

Figure 4.37 Tuple relational calculus formula for query C7

The above tuple relational calculus formula specifies that we wish to retrieve values of the attribute *Ground* from rows *s*, such that *s* is in table *Match* and there exist rows *y* and *z*, such that *y* is in table *Player* and *z* is in table *Batting* meeting the conditions specified. The conditions are that the row *y* is the row in table *Player* for a player whose last name is Dhoni and *z* is a row in *Batting* for the

player y (that is, it is for the player with player ID of Dhoni) in match s . Note that we have not checked the first name of Dhoni assuming that there is only one player by that last name. The result of query $C7$ is presented in Table 4.43.

- (C8) Find the names of players who have not batted in any match in the ODI cricket database given in Tables 3.1 to 3.4.

This query can be formulated in a simple way because it only requires checking that a player has not batted.

$\{s. FName, LName \mid s \in Player \wedge \text{Not } \exists e (e \in Batting \wedge e. PID = s. PlayerID)\}$
--

Figure 4.38 Tuple relational calculus formula for query C8

The above tuple relational calculus expression specifies that we wish to retrieve values of the columns $FName$ and $LName$ from rows s , such that row s is in the table $Player$ and a row e does not exist in the table $Batting$, such that this row e is for the player s .

The result of query $C8$ is given in Table 4.44.

Table 4.44 Result of query C8 as formulated in Fig. 4.38

<i>FName</i>	<i>LName</i>
Brian	Lara
Rahul	Dravid
Herschelle	Gibbs
Shane	Warne
Shaun	Pollock
Michael	Vaughan
Inzamam	Ul-Huq
Stephen	Fleming
Heath	Streak
Anil	Kumble
Gary	Kirsten
Jacques	Kallis
Daniel	Vettori
Saurav	Ganguly
Virender	Sehwag

Table 4.43 Result of Query C7

<i>Ground</i>
Brisbane
Colombo

- (C9) Find the match IDs of matches that have batting records.

This is a simple query using only one table. It can be formulated as given in Fig. 4.39.

The formula in Fig. 4.39 simply finds the values of the column $MatchID$ of each row s such that the row s is in table $Batting$.

There is however a weakness in the above formulation. It will lead to many duplicate tuples since a match has many batting records. If we wish to eliminate the duplicates, a better formulation of the query C9 would be as in Fig. 4.40.

$$\{s.\text{MatchID} \mid s \in \text{Batting}\}$$

Figure 4.39 Tuple relational calculus expression for query C9

$$\{s.\text{MatchID} \mid s \in \text{Match} \wedge \exists e (e \in \text{Batting} \wedge e.\text{MatchID} = s.\text{MatchID})\}$$

Figure 4.40 A better tuple relational calculus expression for query C9

This formulation retrieves rows from table *Match* (not from *Batting* which is likely to have duplicate values of *MatchID*) and this ensures that there will be only one row for each match in the table *Match*.

The result of the tuple relational calculus formulation in Fig. 4.40 is given in Table 4.45.

- (C10) Find the player IDs of players who have only bowled in match 2755.

In this query, we need to check that a player has bowled in match 2755 but has not bowled in any other match in the database. The tuple relational calculus formulation for this query is given in Fig. 4.41.

Table 4.45 Result of query C9 as posed in Fig. 4.40

MatchID
2689
2755

$$\{e.\text{PlayerID} \mid e \in \text{Player} \mid \exists j \wedge \text{not}(\exists k) (j, k \in \text{Batting} \wedge e.\text{PlayerID} = j.\text{PID} \wedge e.\text{PlayerID} = k.\text{PID} \wedge k.\text{MatchID} = '2755' \wedge k.\text{MatchID} \neq '2755')\}$$

Figure 4.41 Tuple relational calculus formulation for query C10

In the formulation in Fig. 4.50, we look for values of the attribute *PlayerID* of each row *e*, such that it is in the table *Player* and there exists a row *j* in table *Batting* for the same player as in *e* that is for batting in match 2755 and, in addition, there does not exist a row *k* in *Batting*, such that the batting is for the same player as in *e* in a match other than 2755.

The result is given in Table 4.46.

- (C11) Find the player IDs of players who have bowled in all the matches played in Australia.

We found in the discussion on relational algebra that queries involving ‘all’ usually involve the relational algebra operator division which was somewhat difficult to understand. It is easier to formulate such queries in tuple relational calculus.

The query may be formulated as in Fig. 4.42.

Table 4.46 Result of query C10 as posed in Fig. 4.41

PID
99001
24001
23001
94002
92002
91001
23001

$$\{k.\text{PlayerID} \mid k \in \text{Player} \wedge \forall e (e \in \text{Match} \wedge e.\text{Team} = 'Australia' \wedge \exists s (s \in \text{Bowling} \wedge s.\text{PID} = k.\text{PlayerID} \wedge s.\text{MatchID} = e.\text{MatchID}))\}$$

Figure 4.42 Tuple relational calculus expression for query C11

In this query, the tuple relational calculus expression specifies that the values of the attribute *PlayerID* are to be retrieved from each row *k*, such that *k* is in the table *Player* and for each match *e* that is played in Australia there exists a row *s* in the table *Bowling*, such that it is a bowling performance for the player *k* in the match *e*. The $\forall e$ (for all *e*) expression makes solving the problems which were more difficult to solve in relational algebra since there was no operation similar to the operation for all in tuple relational calculus. Unfortunately SQL also does not have an operator similar to for all and therefore problems that involve such operations in SQL are rather painful.

The result is given in Table 4.47.

Table 4.47 Result of query C11 posed in Fig. 4.42

MatchID	PID
2689	99001
2689	24001
2689	23001

4.4.7 Domain Relational Calculus

As noted earlier, domain relational calculus (DRC) is very similar to the tuple relational calculus that we have presented in the last section except that rather than using variables ranging over tuples in TRC, the variables range over single domains of attributes in DRC. Therefore to specify a relation with *n* attributes it is necessary to specify *n* domain variables one for each attribute.

The basis of DRC essentially is that a relational database consists of the domains that represent the sets of objects in the database. A query language based on DRC therefore consists of variables that range on a single domain of the relational database. It is claimed that DRC therefore manipulates the basic database objects since domain values interact more directly with the semantics expressed by the relations and produces simpler and more English-like languages.

The following query shows the general form of a DRC query

$$\{<PlayerID, LName, FName, Country, YBorn, BPlace, FTest, F(PlayerID, LName, FName, Country, YBorn, BPlace, FTest)>\}$$

In this query *PlayerID*, *LName*, *FName*, *Country*, *YBorn*, *BPlace*, *FTest* represent domain variables that the user wishes to retrieve and *F(.)* is a formula that specifies the condition. The formula *F* may be very simple, for example, stating that all these domains together form the relation *Player*. The formula *F(.)* is shown below.

$$<PlayerID, LName, FName, Country, YBorn, BPlace, FTest> \in Player$$

We now present a number of queries and formulate them in domain relational calculus.

- (D1) Find the names and IDs of all players.

The domain relational calculus formulation of query D1 is given in Fig. 4.43.

$$\{<Fname, Lname, PlayerID> \mid \exists Country, YBorn, Bplace, FTest (<PlayerID, LName, FName, Country, YBorn, BPlace, FTest>) \in Player\}$$

Figure 4.43 Domain relational calculus expression for query D1

*image
not
available*

We are now looking for values of the domain $PlayerID$ from the relation $Player$ such that there exists a row in table $Batting$ for that player where the $Match ID$ is either 2755 or 2689. Note that the above formulation will not retrieve any duplicates of $PlayerID$ even if the player has batted in both matches 2755 and 2689 (why not?). This query could also have been formulated by specifying only the table $Batting$ such that $PIDs$ could be retrieved from that table for rows that met the given condition.

- (D5) Find player IDs of players batting in both matches 2755 and 2689.

This query is little bit different than the queries we have seen so far and may be formulated as in Fig. 4.47.

In this query formulation, we are looking for values of the domain $PlayerID$ from rows the table $Player$ for which there exist two rows in the table $Batting$ for that player, such that one of them corresponds to match 2755 and the other to match 2689.

4.4.8 Well-Formed Formula (WFF)

The queries formulated in tuple relational calculus and domain relational calculus above should have given the reader some idea about the features of the two variants of relational calculus. We have however not specified what expressions in calculus are acceptable. This is elaborated ahead.

We have already discussed the concept of free and bound variables. Also we have indicated earlier, that the general form of relational calculus expressions is $\{(x,y,z) \mid P(x, y, z)\}$ where P is a formula in which the variables x, y, z are free. An acceptable formula is called a *well-formed formula* or WFF. WFFs may be defined recursively as in Fig. 4.48.

1. Any term is a WFF (a term is either an indexed tuple or a comparison of two indexed tuples).
2. If F and a WFF, so is $(\text{not } F)$.
3. If F and G are WFFs, so are $(F \wedge G)$ and $(F \vee G)$.
4. If F is a WFF in which x appears as a free variable, then $\exists x(F)$ and $\forall x(F)$ are WFFs.
5. No other formulas are WFFs.

Figure 4.48 Definition of well-formed formulas in tuple relational calculus

4.5

RELATIONAL ALGEBRA VS RELATIONAL CALCULUS

It is useful to discuss the selective power of relational algebra and relational calculus. Any language that has the power of relational calculus is called *relationally complete*. Codd in his paper in 1970 proposed that any language that was not relationally complete should be considered inadequate. He showed that relational algebra is equivalent to relational calculus and is therefore relationally complete. It is in fact possible to design algorithms that will translate a relational algebra expression into an equivalent relational calculus expression and vice versa. These algorithms are called *reduction algorithms*. We will not discuss them here, but the reader can refer to Codd (1970) or Ullman (1982).

Given that relational algebra and relational calculus have equivalent selective power, one may now ask which one of these two languages should be preferred. Codd, in his 1971 paper, claimed that relational calculus was superior to algebra although the algebra did not require use of quantifiers for all (\forall) and there exists (\exists). He gave four reasons for preferring relational calculus:

1. *Ease of Augmentation*—Since the algebra and the calculus provided only basic selective power, it is necessary to augment that power by providing a facility to invoke a number of library functions (for example, average, sum, maximum). This augmentation is more easily done in relational calculus than in algebra.
2. *Scope for Search Optimization*—Since the calculus only specifies what is to be retrieved and not how it is to be retrieved, it is easier for a database system to optimize database search to answer queries that are posed in calculus. Queries posed in relational algebra can also be optimized but such optimization is likely to be only local since it would be difficult for a system to derive the properties of the data that needs to be retrieved given the operators that the user has specified.
3. *Authorization Capability*—It is likely to be easier to specify authorization by using relational calculus since authorization is likely to be based on the properties of the data that a user may be authorized to access or update.
4. *Closeness to Natural Language*—It is much more natural for users in relational calculus to specify what data they wish to retrieve than how the data is to be retrieved. Thus a language based on relational calculus is likely to be more user-friendly and easier to use for users with limited computing background.

As noted above, the relational model requires that the language available to the user should at least provide the power of relational algebra without making use of iterations or recursion statements. Relational calculus does that. So does the commercial language, SQL, which we will discuss in detail in Chapter 5.

SUMMARY

This chapter has covered the following:

- Codd defined two formal languages, viz., relational algebra and relational calculus.
- Relational algebra is a procedural language that includes the operators selection, projection, Cartesian product, equi-join, natural join, outer join, union, intersection, difference and division.
- Relational algebra notation has been presented. For example, a selection may be written as *Players* [*Country* = ‘India’] and the notation has been used to formulate a variety of queries.
- Selection and projection are unary operators, operating on a single table.
- Cartesian product and join are binary operators, operating on two tables. Join is used to combine two related tables on a join condition, for example, the table *Player* and *Batting*.
- Outer join is a join operator in which matching rows from two tables as well as rows that do not match are shown. An outer join may be left outer join or a right outer join or a full outer join.
- It is possible to use set operations in relational algebra including union, intersection and difference.
- Division operator may be applied to two tables and is related to the Cartesian product operator, it may be considered the opposite of the product.

- Background of predicate calculus and relational calculus is presented including meaning of propositions, the implied operator and parameterized propositions.
- Meaning of the existential and universal quantifiers is explained and use of calculus to formulate a variety of queries is illustrated using a number of examples.
- Two notations of relational calculus, viz., the tuple relational calculus (TRC) and domain relational calculus (DRC) are presented.
- TRC uses tuple variables (rows of a table) while DRC uses variables that range over domains to formulate queries.
- Well-formed formulas (WFF) are the acceptable expressions in calculus; WFF is defined.
- Relative merits of algebra and calculus and their relative power to express queries are discussed. The calculus may be preferred for ease of augmentation and for better scope for optimization.

REVIEW QUESTIONS

1. What is the function of operations restriction, projection, Cartesian product, equi-join, natural join, outer join, union, intersection, difference, and division? (Section 4.2)
2. Describe the relational algebra notation for restriction, projection and natural join. (Section 4.2)
3. Give examples to explain a database query? Show how to use relational algebra to formulate a variety of queries. (Section 4.2)
4. What is the function of division operation? Can it be expressed in terms of other relational algebra operations?
5. Explain the relational calculus notation and well-formed formulas. (Section 4.3)
6. Explain the notation used in expressing queries in tuple relational calculus. (Section 4.3)
7. Give examples to explain the meaning of the existential quantifier and universal quantifier. (Section 4.4)
8. Describe how to use tuple relational calculus to formulate a variety of queries. (Section 4.4)
9. Give a number of examples of using domain relational calculus to formulate a variety of queries. (Section 4.4)
10. Select two queries and formulae them using tuple relational calculus and the domain relational calculus. (Section 4.4)
11. Explain the relative merits of algebra and calculus and their relative power to express queries. (Section 4.5)

SHORT ANSWER QUESTIONS

1. What is relational algebra?
2. What is relational calculus?
3. What is relational algebra operator selection?
4. Give an example of using selection in Table 4.5.
5. What is relational algebra operator projection?

6. Give an example of applying projection to Table 4.13.
7. What is relational algebra operator Cartesian product?
8. What is the Cartesian product of Tables 4.11 and 4.12?
9. How is the equi-join different from natural join?
10. Define the properties of two relations to be union-compatible.
11. Are Figs 4.11 and 4.12 union-compatible?
12. Find the natural join from Cartesian product of Tables 4.11 and 4.12.
13. What is the left outer join? What does the operation do?
14. What is the right outer join?
15. Find the full outer-join of Tables 4.10 and 4.11. What does the result tell us?
16. What is the relational operator division used for?
17. Give an example of two small relations and divide them to illustrate division.
18. Give examples of universal and existential quantifiers.
19. Express the query ‘find all the information about players from India’ using TRC and DRC.

MULTIPLE CHOICE QUESTIONS

1. Which one of the following is **not** correct?
 - A restriction selects one or more rows of a relation.
 - A projection selects one or more columns of a relation.
 - A join glues each row of one relation with all the rows of the other.
 - A difference gives all the rows in the first relation that are not in the second.
2. Which one of the following is correct?
 - A Cartesian product of two relations is the same as their union.
 - A join of two relations is a selection on their Cartesian product.
 - A join is always an equi-join or a natural join.
 - The degree of the equi-join of two relations is the same as the degree of the natural join of the same relations.
3. Consider the following query:

$$(((P \text{ WHERE COLOUR} = 'RED') [P\#] \text{ JOIN } SP) [S\#] \text{ JOIN } S) [\text{SNAME}]$$
 Which one of the following set of operations does the above query involve?

(a) 1 selection, 2 joins, 3 projections	(b) 1 projection, 2 joins, 3 selections
(c) 2 selections, 2 joins, 3 projections	(d) 1 selection, 2 joins, 3 intersections
4. Which one of the following does not always have the same list of attributes that the operands have?

(a) Project	(b) Select	(c) Union	(d) Difference
-------------	------------	-----------	----------------
5. Which one of the following always results in fewer rows than the relation R ? Assume neither of the two relations is empty.

(a) $R \bowtie S$	(b) $R - S$	(c) R/S	(d) R intersection S
(e) None of the above			

6. Which one of the following always results in at least as many rows as in the relation R ? Assume neither of the two relations is empty.
- $R \bowtie S$
 - R union S
 - $R - S$
 - R/S
 - R intersection S
7. Which two of the following operators require elimination of duplicates?
- Projection
 - Intersection
 - Difference
 - Union
 - Join
8. Which one of the following is a complete set of algebra operations?
- selection, projection, union, difference and Cartesian product
 - selection, projection and join
 - selection, projection and Cartesian product
 - selection, projection, division and Cartesian product
9. Consider the following binary operators. Which one of the operator does not require the two operands to be union-compatible (that is, having exactly the same number and types of attributes)?
- Intersection
 - Difference
 - Union
 - Division
10. Relational division has operands and results named as given below. Which of the following is not a relation?
- Dividend
 - Divisor
 - Quotient
 - Remainder
 - None of the above
11. $R[A \text{ op } B]S$ is a join of the relations R and S . Which one of the following is **not** correct?
- $R[A = B]S$ may be empty even when R and S are not
 - $R[A = B]S = S[B = A]R$
 - $R[A < B]S = S[B > A]R$
 - $R[A < B]S = R \times S - R[A > B]S - R[A = B]S$
 - $R[A < B]S = R \times S - S[B < A]R$
12. Which of the following are correct?
- $R \bowtie S = S \bowtie R$
 - $R - S = S - R$
 - $R/S = S/R$
 - $S \cup T = (S - T) \cup (S \text{ intersect } R) \cup (T - S)$
13. Consider the join of two relations R and S that are of degree x and y respectively. Which one of the following is correct?
- The degree of an equi-join of relations R and S is $x + y$.
 - Degree of a natural-join of relations R and S is always $x + y - 1$.
 - If the cardinality of R is a and of S is b , then the cardinality of the division R/S is a/b .
 - A projection of relation R has the same cardinality as R but the degree of the projected relation is equal to the number of attributes that the projection specifies.
14. Which one of the following is correct? Note that \cup represents union while \cap represents intersection.
- $(R \cup S) \cap T = (R \cap T) \cup S$
 - $R - S = S - R$
 - $(R/S) \times S = R$
 - $S \cup T = (S - T) \cup (S \cap R) \cup (T - S)$
15. Let $T = R/S$ where R is an enrolment relation and S is a subject relation with degrees 2 and 1 and cardinalities m and n respectively. The attributes of R are student_id and subject_id and the attribute of S is subject_id. Which of the following are correct? (Note: $T \times S$ is the Cartesian product of T and S).

- (a) T is of degree 1.
 (b) T has cardinality no larger than m/n .
 (c) $(R - T \times S)[\text{student_id}]$ does not include any student who is in T .
 (d) T are all the students doing all the subjects in S .
 (e) All of the above
16. Let R be an enrolment relation and S be a subject relation. The relation R has two attributes `student_id` (call it A) and `subject_id` (call it B) while the relation S consists of only one attribute `subject_id` (call it also B).
 Which one of the following is correct?
 (a) $((R[A] \times S) - R)[A]$ is the list of `student_id` values (attribute A) in R .
 (b) $R[A] - ((R[A] \times S) - R)[A]$ is a null relation.
 (c) $(R - R[A] \times S)[A]$ is the list of all the student IDs for those doing all the subjects in S .
 (d) The expression $R[A] - ((R[A] \times S) - R)[A]$ is the list of all the student IDs for students not doing all the subjects in S .
17. Which one of the following is **not** correct?
 (a) Relational algebra provides a collection of operations that can actually be used to build some desired relation; the calculus provides a notation for formulating the definition of that desired relation.
 (b) In relational algebra, one specifies what operations must be executed while in relational calculus, the specification of operations is left to the system.
 (c) Relational algebra is procedural while relational calculus is nonprocedural.
 (d) Using relational calculus one may specify more complex queries than using relational algebra.

EXERCISES

1. Consider the following student database:

`Student(ID, Name, Address, City, Tutor)`

`Enrolment(ID, Code, Marks)`

`Course(Code, Title, Department)`

Formulate the following queries in relational algebra using the student database above. Assume that the attribute `Tutor` is the ID of the tutor.

- (a) List all the student names.
- (b) List all school and course names.
- (c) Find student names of students that have no tutor.
- (d) Find student names of students that are doing computer science courses.
- (e) Find student names of students that have Suresh Chandra as tutor.

2. The following questions deal with Cartesian product and join:

- (a) Consider the relations in the student database exercise above. How many rows will there be in the Cartesian product of the three tables?
- (b) How many attributes does the natural join of the two tables `Student` and `Enrolment` have? Explain.

3. Express
 - (a) Division R/S in terms of projection, Cartesian product and difference.
 - (b) Natural join in terms of projection, Cartesian product and selection.
 - (c) Intersection in terms of projection, union and difference.
4. Consider the following two queries for the cricket database:
 - (a) Find the youngest player in Table 3.1
 - (b) Find the oldest player in Table 3.1

Can the above two queries be formulated using relational calculus? If yes, formulate the two queries.
5. Explain the motivation for TRC and DRC and discuss the differences between the two approaches.
6. Show how a TRC query may be written as SQL and SQL query may be written as a TRC query.
7. Study the basics of QBE and use QBE in Microsoft Access to understand it and relate it to DRC. Can a DRC be translated to QBE? Show by using an example.
8. Write a query that involves a division and show how it may be expressed in TRC.

PROJECTS

1. Write algorithms and then implement them in the programming language of your choice for all the relational algebra operations. This could be a group project.
2. Using the Web, search for open source software for relational calculus (TRC or DRC). Install the software and test it by posing some queries to it.

LAB EXERCISES

1. Formulate the following queries for the ODI database using relational algebra:
 - (a) Find the player's last name that was the youngest amongst the players in the database when he played in his first test match.
 - (b) Find the ID of the player that hit the largest number of sixes in an ODI match in the example database.
 - (c) Find the list of names of players that have scored a century in the example ODI database.
 - (d) Find the average strike rate of players that have scored a century in the example ODI database.
 - (e) Find the names of players that have taken four wickets or more in the example ODI database.
 - (f) What is the maximum number of wickets that a bowler has taken in an ODI match in the example database? Find the ID of that bowler.
 - (g) Find the name of the youngest player in the example database.
 - (h) Find the IDs of players that have never scored a four or a six in all matches they have batted in.
2. Formulate the above queries in Tuple Relational Calculus.

BIBLIOGRAPHY

For Students

A number of books listed below are easy to read for students.

For Instructors

Codd's papers and his out-of-print book (which may be downloaded from the Web) are the authoritative documents on the relational model. C. J. Date's books are also a good source of information.

Codd, E. F., "A relational model of data for large shared data banks", *Communications of the ACM*, Vol. 13, No. 6, 1970, pp 377-387.

Codd, E. F., "Relational database: a practical foundation for productivity", *Communications of the ACM*, Vol. 25, No. 2, 1982, pp. 109-117.

Codd, E. F., *The Relational Model for Database Management: Version 2*, Addison-Wesley Publishing Company, Inc, 1990. (This book is out of print now but may be downloaded from <http://portal.acm.org/citation.cfm?id=77708>.)

Date, C. J., "The Outer Join", In *Relational Database: Selected Writings*, Addison-Wesley, 1986.

Date, C. J., *Database in Depth: Relational Theory for Practitioners*, O'Reilly Media, 2005.

Garcia-Molina, H., J. D. Ullman, and J. Widom, *Database Systems: The Complete Book*, Prentice Hall, 2002.

Garcia-Molina, H., J. D. Ullman, and J. Widom, *Database System Implementation*, Prentice Hall, 2000.

Kifer, M., Bernstein, A., and P. M. Lewis, *Database Systems: An Application-Oriented Approach*, Second Edition, Addison-Wesley, 2006.

Ullman, J. D., *Principles of Database Systems*, Computer Science Press, 1982. (May be downloaded from <http://www.filestube.com/d/database+systems+ullman>)



Structured Query Language (SQL)

5

OBJECTIVES

- To explain how SQL is used for data definition.
- To describe how SQL is used for retrieving information from a database.
- To explain how SQL is used for inserting, deleting and modifying information in a database.
- To explain the use of SQL in implementing correlated and complex queries including outer joins.
- To discuss embedded SQL and dynamic SQL.
- To discuss how data integrity is maintained in a relational database using SQL.

KEYWORDS

Queries, SQL, CREATE TABLE, CREATE INDEX, CREATE VIEW, CREATE DOMAIN, DDL, DROP TABLE, DROP INDEX, DROP VIEW, DROP DOMAIN, SELECT, INSERT, WHERE, GROUP BY, DELETE, UPDATE, VIEW, grouping, aggregation, functions, AVG, COUNT, MIN, MAX, SUM, nested queries, subqueries, outer join, integrity constraints, primary key, secondary key, domain, NULL, embedded SQL, dynamic SQL, complex queries, correlated queries, existential integrity, referential integrity, UNIQUE, triggers, active databases.

An eye for an eye makes the whole world blind.

Mahatma Gandhi

5.1 INTRODUCTION

We have discussed the relational model in Chapter 3, which is a model for implementing a database that has been modeled using Entity-Relationship modeling or a similar modeling methodology. The relational model is widely used because of its simplicity. It is a conceptual view of the data as well as a high level view of the implementation of the database on the computer. The model does not concern itself with the detailed bits and bytes view of the data.

In the last chapter, we studied two formal data manipulation languages in detail, viz., relational algebra and relational calculus. It is not appropriate to use either of them in a commercial database system that would be used by users with a computing background as well as by those that have little computing knowledge because calculus and algebra are not user-friendly languages and they do not provide facilities for counting or for computing averages, or maximum and minimum values. In this chapter we discuss another query language called the Structured Query Language (SQL) that is much easier for users. It is a commercial query language based on relational calculus.

The query language SQL is used in almost all commercial relational DBMS. Although other query languages exist, SQL has become a de facto standard for relational database systems.

SQL is a nonprocedural language that originated at IBM during the building of the well-known experimental relational DBMS called System R. Originally the user interface to System R was called SEQUEL designed by Donald Chamberlain and Raymond Bryce in 1974. The language was later modified in 1976–77 and its name changed to SEQUEL2. These languages have undergone significant changes over time and the current language was named the Structured Query Language (SQL) in the early 1980s.

Just as for other programming languages, the American National Standards Institute (ANSI) published a standard definition of SQL in 1986 which was subsequently adopted by the International Organization for Standardization (ISO) in 1987. Since then there have been a number of SQL standards, the most recent being SQL : 2003. We will give more details about the SQL standards at the end of this chapter. The focus of this chapter is on the basic SQL constructs. We will not discuss some of the newer features included in the more recent standards.

This chapter is organized as follows. After introducing SQL next the data definition features of SQL are described in Section 5.3. Section 5.4 covers data retrieval features of SQL. This section includes some simple examples of SQL that involve only one table followed by examples that involve more than one table using joins and subqueries, examples using built-in functions followed by explanation of correlated and complex SQL queries. GROUPING and HAVING are presented next. SQL update commands and additional data definition commands are then discussed. Views are described in Section 5.5 and Section 5.6 discusses embedded SQL and dynamic SQL. Section 5.7 explains how SQL may be used for enforcing the final component of the relational model and data integrity. Existential integrity, referential integrity, domains, and nulls have also been discussed here. Triggers are briefly described in Section 5.8. Authentication and authorization

is discussed briefly in Section 5.9. Additional data definition commands are described in Section 5.10. A summary of SQL commands is presented in Section 5.11 and SQL standards are discussed in Section 5.12.

5.2 SQL

As noted above, SQL is designed to support data definition, data manipulation, and data control in relational database systems.

A user of a DBMS using SQL may use the query language interactively or through a host language like C++. An interactive web page could also be the front end of the SQL query in which the user supplies some parameters before the query is executed by the server. We will mainly discuss the interactive use of SQL although SQL commands embedded in a procedural language will also be discussed briefly.

SQL consists of the following facilities:

- *Data Definition Language (DDL)*—Before a database can be used, it must be created. Data definition language is used for defining the database to the computer system. The DDL includes facilities to create and destroy databases and database objects, for example, for creating a new table or a view, deleting a table or a view, altering or expanding the definition of a table (by entering a new column) and creating an index on a table, as well as commands for dropping an index. DDL also allows a number of integrity constraints to be defined either at the time of creating a table or later.
- *Data Manipulation Language (DML)*—These DML facilities include commands for retrieving information from one or more tables and updating information in tables including inserting and deleting rows. The commands SELECT, UPDATE, INSERT and DELETE are available for these operations. The SELECT command forms the basis for retrieving data from the database. The command UPDATE forms the basis for modifying information that is already in the database. INSERT is used for inserting new information in the database while DELETE is used for removing information from the database.
- *Data Control Language (DCL)*—The DCL facilities include database security control including privileges and revoke privileges. Therefore DCL is responsible for managing database security including management of which users have access to which tables and what data.
- *New Features*—Each new standard has introduced a variety of new features in the language. For example, SQL:2003 includes XML. Unfortunately we will not be able to cover these features in this book.

It should be noted that SQL does not require any particular appearance of SQL queries and is not case sensitive except for object names. Although it is much easier to read queries that are well structured and indented, SQL does not require it. In fact, the whole query could be typed in one long line if that is what the user wishes to do. Sometimes though different DBMS may use somewhat different conventions. We will use the convention of writing SQL commands as upper case letters like we did in the last paragraph and names of tables and their columns in lower case. In our view this makes it easier to read the queries.

We first study some of the data definition features of the language in the next section.

We will continue to use the ODI cricket database to illustrate various features of the query language in this section. As a reminder, the schemas of the four tables *Match*, *Player*, *Batting* and *Bowling* are reproduced

in Fig. 5.1 and the tables are reproduced in Tables 5.1, 5.2, 5.3 and 5.4. Additional data for these tables is presented at the end of this chapter. For example, Appendix 5.1 includes information on another 50 players. Appendix 5.2 includes another 35 rows for the table *Batting* and Appendix 5.3 includes another 35 rows for the table *Bowling*. These may be used for exercises that require more data.

Match (MatchID, Team1, Team2, Ground, Date, Winner)
Player (PlayerID, LName, FName, Country, YBorn, BPlace, FTest)
Batting (MatchID, PID, Order, HOut, FOW, NRuns, Mts, Nballs, Fours, Sixes)
Bowling (MatchID, PID, NOvers, Maidens, NRuns, NWickets)

Figure 5.1 Database schema for the ODI cricket database

We now reproduce the sample tables for the database that were presented in Chapter 3. The first table, Table 5.1, is *Match*.

Table 5.1 The relation *Match*

MatchId	Team1	Team2	Ground	Date	Winner
2324	Pakistan	India	Peshawar	6/2/2006	Team1
2327	Pakistan	India	Rawalpindi	11/2/2006	Team2
2357	India	England	Delhi	28/3/2006	Team1
2377	West Indies	India	Kingston	18/5/2006	Team2
2404a	Sri Lanka	India	Colombo	16/8/2006	Abandoned
2440	India	Australia	Mohali	29/10/2006	Team2
2449	South Africa	India	Cape Town	26/11/2006	Team1
2480	India	West Indies	Nagpur	21/1/2007	Team1
2493	India	West Indies	Vadodara	31/1/2007	Team1
2520	India	Sri Lanka	Rajkot	11/2/2007	Team2
2611	England	India	Southampton	21/8/2007	Team1
2632	India	Australia	Mumbai	17/10/2007	Team1
2643	India	Pakistan	Guwahati	5/11/2007	Team1
2675	Australia	India	Melbourne	10/2/2008	Team2
2681	India	Sri Lanka	Adelaide	19/2/2008	Team1
2688	Australia	India	Sydney	2/3/2008	Team2
2689	Australia	India	Brisbane	4/3/2008	Team2
2717	Pakistan	India	Karachi	26/6/2008	Team2
2750	Sri Lanka	India	Colombo	24/8/2008	Team2
2755	Sri Lanka	India	Colombo	27/8/2008	Team2

The second table, Table 5.2, is *Player*.

Table 5.2 The relation Player

PlayerID	LName	FName	Country	YBorn	BPlace	FTest
89001	Tendulkar	Sachin	India	1973	Mumbai	1989
90001	Lara	Brian	West Indies	1969	Santa Cruz	1990
95001	Ponting	Ricky	Australia	1974	Launceston	1995
96001	Dravid	Rahul	India	1973	Indore	1996
96002	Gibbs	Herschelle	South Africa	1974	Cape Town	1996
92001	Warne	Shane	Australia	1969	Melbourne	1992
95002	Pollock	Shaun	South Africa	1973	Port Elizabeth	1995
99003	Vaughan	Michael	England	1974	Manchester	1999
92003	Ul-Huq	Inzamam	Pakistan	1970	Multan	1992
94004	Fleming	Stephen	New Zealand	1973	Christchurch	1994
93002	Streak	Heath	Zimbabwe	1974	Bulawayo	1993
90002	Kumble	Anil	India	1970	Bangalore	1990
93003	Kirsten	Gary	South Africa	1967	Cape Town	1993
95003	Kallis	Jacques	South Africa	1975	Cape Town	1995
94002	Vaas	Chaminda	Sri Lanka	1974	Mattumagala	1994
92002	Muralitharan	Muthiah	Sri Lanka	1972	Kandy	1992
97004	Vettori	Daniel	New Zealand	1979	Auckland	1997
25001	Dhoni	MS	India	1981	Ranchi	2005
23001	Singh	Yuvraj	India	1981	Chandigarh	2003
96003	Ganguly	Saurav	India	1972	Calcutta	1996
99002	Gilchrist	Adam	Australia	1971	Bellingen	1999
24001	Symonds	Andrew	Australia	1975	Birmingham	2004
99001	Lee	Brett	Australia	1976	Wollongong	1999
91001	Jayasuriya	Sanath	Sri Lanka	1969	Matara	1991
21001	Schwag	Virender	India	1978	Delhi	2001
98001	Afridi	Shahid	Pakistan	1980	Khyber Agency	1998
98002	Singh	Harbhajan	India	1980	Jalandhar	1998
27001	Kumar	Praveen	India	1986	Meerut	NULL
27002	Sharma	Ishant	India	1988	Delhi	2007

The third table is Table 5.3 *Batting*.

Table 5.3 The relation *Batting*

MatchID	PID	Order	HOut	FOW	NRuns	Mts	Nballs	Fours	Six
2755	23001	3	C	51	0	12	6	0	0
2755	25001	5	C	232	71	104	80	4	0
2755	91001	1	C	74	60	85	52	8	2
2755	94002	7	LBW	157	17	23	29	1	0
2755	92002	11	NO	NO	1	11	7	0	0
2689	89001	2	C	205	91	176	121	7	0
2689	23001	4	C	175	38	27	38	2	2
2689	25001	5	C	240	36	52	37	2	1
2689	99002	1	C	2	2	3	3	0	0
2689	95001	3	C	8	1	9	7	0	0
2689	24001	5	C	123	42	81	56	2	1
2689	99001	8	B	228	7	20	12	0	0
2689	27001	9	C	255	7	7	7	1	0
2755	27001	9	B	257	2	7	6	0	0
2689	98002	8	LBW	249	3	7	3	0	0
2755	98002	8	RO	253	2	7	4	0	0

The fourth and the last table, Table 5.4, is *Bowling*.

Table 5.4 The relation *Bowling*

MatchID	PID	NOvers	Maidens	NRuns	NWickets
2689	99001	10	0	58	1
2689	24001	3	0	27	1
2689	23001	3	0	15	0
2755	94002	9	1	40	1
2755	92002	10	0	56	1
2755	91001	4	0	29	0
2755	23001	10	0	53	2
2755	98002	10	0	40	3
2689	98002	10	0	44	1

We will now discuss the various features of SQL.

5.3

DATA DEFINITION—CREATE, ALTER AND DROP COMMANDS

As noted in the last section, DDL includes commands to create and destroy databases and database objects. The basic DDL commands available in SQL are discussed below. Additional DDL commands are presented later in this chapter in Section 5.4.12.

5.3.1 CREATE Command

CREATE is the most commonly used DDL command. It may be used to create a database, a schema, a new table, a new index, a new view, a new assertion or a new trigger.

We only look at creating an empty table. To create the tables *Match*, *Player*, *Batting* and *Bowling*, we use the CREATE TABLE command. Before formulating a CREATE TABLE command, it is necessary that each column type be looked at and likely values considered so that its data type be defined. In Fig. 5.2, we have arbitrarily assumed that *Team1* and *Team2* are 15 characters long but that will not be sufficient if we wanted to write one of the teams as Republic of South Africa. Therefore, care must be taken in defining data types for each column. Similarly, the data type of *Ground*, CHAR(20), is not able to represent names like Melbourne Cricket Ground or M.A. Chidambaram Stadium in Chennai or Vidarbha Cricket Association Stadium in Nagpur. CHAR(60) might be a better data type if full ground names are to be used for the column *Ground*. In addition to data type for each column, it is necessary to define the primary key and any foreign keys as well as any columns that are NOT NULL or UNIQUE.

The basic form of the CREATE TABLE command is given in Fig. 5.2.

The CREATE TABLE command as shown in Fig. 5.2 is used to create a new table *Match*. It includes a specification of the names of the table as well as name of all its columns, separated by comma, and their data types. For each column, a name and data type must be specified. SQL data types are discussed in Section 5.3.4. Integrity constraints may also be specified in this command. In the above data definition, we have specified that the column *MatchID* is the primary key of the table and as noted earlier it will not allow a row with a NULL primary key.

Each column name is unique starting with a letter and made up of letters, numbers or an underscore (_) but no SQL reserve words.

The CREATE command also allows default values to be assigned to the columns. If no default value is specified, the system will usually provide a NULL as the default. It is also possible to specify constraints in the CREATE TABLE command. We will discuss constraints later in this chapter.

```
CREATE TABLE Match
(MatchID INTEGER,
Team1 CHAR(15),
Team2 CHAR(15),
Ground CHAR(20),
Date CHAR(10),
Result CHSR(10),
PRIMARY KEY (MatchID))
```

Figure 5.2 SQL for Data Definition including specification of primary key

There are several other CREATE commands, for example, CREATE DOMAIN, CREATE INDEX, CREATE VIEW. We will discuss them later in this chapter in Section 5.6.

5.3.2 ALTER Command

The ALTER command may be used to modify an existing table. ALTER may be used for adding or dropping a column, changing the column definition as well as adding or dropping a table constraint. For example, Fig. 5.3 shows how to add a column to the table *Batting* to include information on how many times a batsman was not out.

The new column appears as the rightmost column in the table. Once the table definition has been changed, the values of the column would need to be inserted. The default value is NULL.

A column in a table may be dropped if required as illustrated in Fig. 5.4.

If several columns need to be dropped then several ALTER commands are required. More examples of ALTER are given in Section 5.7.

```
ALTER TABLE Batting
ADD COLUMN Not_Out INTEGER
```

Figure 5.3 Adding a Column to a Table

```
ALTER TABLE Batting
DROP COLUMN Not_Out
```

Figure 5.4 Dropping a Column in a Table

5.3.3 Destroying a Table using the DROP Command

A table may be destroyed by using the DROP TABLE command as given in Fig. 5.5.

Dropping a table may have side effects. An index or a view may have been defined on the table and one or more columns in the table may be foreign keys in other tables. We may specify CASCADE like in Fig. 5.6 if we want all references to the table to be dropped as well. It is also possible to use RESTRICT which will not allow the DROP command to be executed if there are dependencies on the table.

There are several other DROP commands, for example, DROP DOMAIN, DROP INDEX, DROP VIEW. We will discuss them later in this chapter in Section 5.7.1.

```
DROP TABLE Batting
```

Figure 5.5 Dropping a Table

```
DROP TABLE Batting CASCADE
```

Figure 5.6 Dropping a Table using CASCADE

5.3.4 SQL Data Types

As shown in Fig. 5.2, it is required that the data type of each column be specified when a table is defined. Data types deal with numbers, text, date, time and binary objects. There are a large number of data types defined in SQL3. We list only the most commonly used data types. We have divided them in three categories.

Numerical Types

There are a variety of numerical data types available in SQL. We list some of them in Fig. 5.7.

Data Type	Description
SMALLINT	Integer, precision 5
INTEGER	Integer precision 10
REAL	Mantissa about 7
FLOAT(p)	Mantissa about 16
DOUBLE PRECISION	Same as FLOAT

Figure 5.7 SQL numerical data types

Several other numerical types are available including INTEGER(p), DECIMAL(p, s), NUMERIC(p, s) and FLOAT (p) all of which allow the user to specify precision.

Character Strings and Binary Data Types

A list of some character strings and binary data types is given in Fig. 5.8.

Data Type	Abbreviation	Description	Comments
CHARACTER(n)	CHAR(n)	Fixed length character string of length n (max n = 255 bytes). Padded on right.	CHAR(5) may be 'India' but not 'Kerala'
CHARACTER VARYING(n)	VARCHAR(n)	Variable length character string up to length n (max n = 2000).	VARCHAR(10) may be 'India' or 'Kerala'
CHARACTER LARGE OBJECT(n)	CLOB(n)	Variable length character string (usually large)	May be a book. n specified in Kbytes, Mbytes or Gbytes.
BINARY VARYING (n)	VARBINARY(n)	Variable length character string up to length n	n can be between 1 and 8000.
BINARY LARGE OBJECT(n)	BLOB(n)	Variable length binary string (usually large)	May be a photo. n specified in Kbits, Mbits or Gbits.

Figure 5.8 SQL string and binary data types

As shown above, quotes must be used when dealing with character strings. Comparison of character strings in SQL queries is explained later.

BLOBs may be used for large multimedia objects, for example, photos. CLOBs may be used for objects like a book. BLOBs as well as CLOBs may not be compared with others and may not be part of a relation's primary key and may not be used in DISTINCT, GROUP BY and ORDER BY clauses.

Date and Time Data Types

We now list the date and time data types in Fig. 5.9. In addition, data types TIMESTAMP and INTERVAL are also available.

Data Type	Description	Example	Comments
Date	YEAR, MONTH and DAY in the format YYYY-MM-DD. Less than comparison may be used.	2010-05-25 or 25-May-2010	Year varies from 0001 to 9999.
Time	HOUR, MINUTE and SECOND in the format HH:MM:SS. Less than comparison may be used.	22:14:37 (fraction of a second may also be represented)	Hour varies from 00 to 23.
TIMESTAMP(n)	Used to specify time of an event in the format YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND	The format is YYYY-MM-DD-HH:MM:SS[.s.]	The length is 26.
INTERVAL	Used to represent time intervals like 9 days or 20 hours and 37 minutes.	Can be YEAR-MONTH or DAY-TIME.	

Figure 5.9 SQL date and time data types

Other complex data types have also been defined by SQL. These include user defined types (UDTs) and structures.

5.4 DATA MANIPULATION—SQL DATA RETRIEVAL

As noted above, the SELECT command is used for retrieving information from a database. The basic structure of the SELECT command is as given in Fig. 5.10.

The SELECT clause specifies what column values are to be retrieved and is therefore equivalent to specifying a projection by listing one or more column names. There can often be confusion between the SELECT clause and the relational algebra operator called selection. The selection operator selects a number of rows from a relation given a condition that the rows satisfy while the SQL SELECT clause is similar to the relational algebra projection operator as it specifies the columns that need to be retrieved. A SELECT clause is mandatory in all SQL queries.

The FROM clause includes a list of one or more table names that are needed to answer the query. If more than one table is specified, a join operation must be specified. Having more than one table without specifying a join is rarely useful. A FROM clause is mandatory in all SQL queries.

The condition in the WHERE clause in an SQL query is optional. The clause provides qualification for the rows that are to be selected. If a WHERE clause is not specified, there is no qualification and the whole table is selected. Normally the WHERE clause is the heart of an SQL query. It specifies the condition(s) that identify the rows the user wishes to retrieve. It is therefore like the condition in the relational algebra selection

```
SELECT something_of_interest
FROM table(s)
WHERE condition holds
```

Figure 5.10 The Basic SELECT Command in SQL

operator. The condition specified in the WHERE clause is either true or false for each row in the table. The rows for which the condition is true are selected and the rows for which the condition is false are rejected. A number of other optional clauses may follow the WHERE clause. These may be used to virtually group the rows in the table and to specify group conditions and to order the table retrieved if necessary. These are discussed later in this chapter.

The result of each SQL query is a table and may therefore be used like any other table. There is however one major difference between a base table and a table that is retrieved from an SQL query. The table retrieved by an SQL query may have duplicates while the rows in a base table are unique.

SQL is case insensitive although we will continue to use upper case for the SQL reserve words like SELECT, FROM and WHERE which could well be written as select, from and where. Also, it should be noted that all table names and column names in a query are one word and do not have spaces within them. The names must be unique. Usually each query ends with a semicolon but the user should check the documentation of the DBMS that is being used to ensure correct notation.

5.4.1 SQL Aliases

A table or a column may be given another name in a SQL query by using an alias. This can be helpful if there are long or complex table or column names which may be converted to short names using aliases, in particular in complex queries using several tables. Also, for some reasons, we may want to change the columns' names in the output from the names used in the base tables. SQL aliases may then be used. A column alias in the column list of the SELECT statement may be specified by using the AS keyword.

The syntax of table name aliases is as shown in Fig. 5.11.

```
SELECT column_name(s)
FROM table_name AS alias_name
```

Figure 5.11 Alias used in the FROM clause in SQL

The syntax for column names is as shown in Fig. 5.12.

```
SELECT column_name AS alias_name
FROM table_name
```

Figure 5.12 Alias used in the SELECT command in SQL

SQL column aliases are used to make the output of an SQL query easy to read and more meaningful. An example is shown in Fig. 5.13.

```
SELECT Team1 AS HomeTeam
FROM Match
```

Figure 5.13 Alias used in the SELECT command in SQL

A query like the one above may be used to change the output of the query. It will now have the heading *HomeTeam* instead of *Team1*.

We now present a number of examples to illustrate different forms of the SELECT command.

5.4.2 Simple Queries—Selecting Columns and Rows

We will first present some simple queries that involve only one table in the database. The simplest of such queries includes only a SELECT clause and a FROM clause.

Retrieving the Whole Table using*

- (Q1) Print the *Player* table.

Figure 5.14 shows how this query can be formulated in SQL.

When the WHERE clause is omitted, all the rows of the table are selected.

```
SELECT *
FROM Player
```

Figure 5.14 SQL for Query Q1

The asterisk following SELECT indicates that all columns of the table specified in the FROM clause are to be retrieved. We could have written all the column names but the wild card * provides a convenient way to print them all. If all columns are not required then we can formulate a query like the next one.

The result of this query is very similar to the table *Player* given in Table 5.2 at the beginning of this chapter although the ordering of the rows cannot be predicted. As noted in the last chapter, the relational model does not define an ordering of the rows and the ordering in a result is system dependent.

Using Projection to Retrieve Some Columns of All Rows from One Table

- (Q2) Find the IDs and first and last names of all players.

Figure 5.15 shows how this query can be formulated in SQL.

The above query is equivalent to specifying a projection operator in relational algebra to the table *Player* such that only columns *PlayerID*, *FName*, and *LName* are retrieved.

```
SELECT PlayerID, FName, LName
FROM Player
```

Figure 5.15 SQL for query Q2

The result of this query is presented in Table 5.5.

Retrieving All Information about Certain Rows from One Table

Often we may not want information about all the rows. Instead we need information only about some rows based on specific criteria. The criteria may be specified by using the WHERE clause.

- (Q3) Find all the information from table *Player* about players from India.

Figure 5.16 shows how this query can be formulated in SQL. Since we are only interested in players from India, the WHERE clause is used to specify the condition.

Table 5.5 Result of query Q2

PlayerID	FName	LName
89001	Sachin	Tendulkar
90001	Brian	Lara
95001	Ricky	Ponting
96001	Rahul	Dravid
96002	Herschelle	Gibbs
92001	Shane	Warne
95002	Shaun	Pollock
99003	Michael	Vaughan
92003	Inzamam	Ul-Huq
94004	Stephen	Fleming
93002	Heath	Streak
90002	Anil	Kumble
93003	Gary	Kirsten
95003	Jacques	Kallis
94002	Chaminda	Vaas
92002	Muthiah	Muralitharan
97004	Daniel	Vettori
25001	MS	Dhoni
23001	Yuvraj	Singh
96003	Saurav	Ganguly
99002	Adam	Gilchrist
24001	Andrew	Symonds
99001	Brett	Lee
91001	Sanath	Jayasuriya
21001	Virender	Sehwag
98001	Shahid	Afriди
98002	Harbhajan	Singh
27001	Praveen	Kumar
27002	Ishant	Sharma

Only those rows that have the *Country* column value equal to India are selected. All columns of these rows are presented to the user. Note that when an attribute value is text, the value in the WHERE clause is surrounded by single quotes.

```
SELECT *
FROM Player
WHERE Country = 'India'
```

Figure 5.16 SQL for query Q3

The result of this query is all the details of Indian players and is the relation given in Table 5.6.

Table 5.6 Result of query Q3

PlayerID	LName	FName	Country	YBorn	BPlace	FTest
89001	Tendulkar	Sachin	India	1973	Mumbai	1989
96001	Dravid	Rahul	India	1973	Indore	1996
90002	Kumble	Anil	India	1970	Bangalore	1990
25001	Dhoni	MS	India	1981	Ranchi	2005
23001	Singh	Yuvraj	India	1981	Chandigarh	2003
96003	Ganguly	Saurav	India	1972	Calcutta	1996
21001	Sehwag	Virender	India	1978	Delhi	2001
98002	Singh	Harbhajan	India	1980	Jalandhar	1998
27001	Kumar	Praveen	India	1986	Meerut	NULL
27002	Sharma	Ishant	India	1988	Delhi	2007

In the query posed in Fig. 5.16, it is possible to use “WHERE Country \neq ‘India’” if all players who are not from India were to be retrieved. Many other conditions may be used in the WHERE clause as discussed later.

- (Q4) Find all the information from table *Player* about players from India who were born after 1980.

Figure 5.17 shows how this query may be formulated in SQL.

In this query we are not only interested in players from India but also those born after 1980. The WHERE clause is used to specify both these conditions.

Only those rows that have the *Country* column value equal to India and players born after 1980 are selected.

The result of this query is all the details of Indian players who were born after 1980 and is the relation given in Table 5.7.

```
SELECT *
FROM Player
WHERE Country = 'India'
AND YBORN > 1980
```

Figure 5.17 SQL for query Q4

Table 5.7 Result of query Q4

PlayerID	LName	FName	Country	YBorn	BPlace	FTest
25001	Dhoni	MS	India	1981	Ranchi	2005
23001	Singh	Yuvraj	India	1981	Chandigarh	2003
27001	Kumar	Praveen	India	1986	Meerut	NULL
27002	Sharma	Ishant	India	1988	Delhi	2007

Retrieving Certain Columns of Certain Rows from One Table

- (Q5) Find details of matches that have been played in Australia.

First we note a weakness of our example data model. There is no way to obtain sufficient detail about each ODI match as we do not include a variety of useful information, for example, which ODI matches were played as part of the “2007-8 ODI Series between Australia and India”. Also in the last few years many triangular ODI series have been held and some matches have been held in neutral venues like Dubai, Belfast or Kuala Lumpur. It is thus difficult to identify where each ODI match was played but we do so for the example database by assuming that the *Team1* name is also the name of the country where the match is being played.

Figure 5.18 shows how this query may be formulated in SQL.

Again, a WHERE clause specifies the condition that *Team1* is Australia.

One way of looking at the processing of the above query is to think of a pointer going down the table *Match* and at each row asking the question posed in the WHERE clause. If the answer is yes, the row being pointed at is selected, otherwise rejected. Since we have not included the name of the country where a match is being played in this example database, as noted above, we have to assume that if *Team1* is Australia then the match is being played in Australia.

The result of this query providing details of matches played in Australia is given below in Table 5.8.

Table 5.8 Result of query Q5

<i>Team1</i>	<i>Team2</i>	<i>Ground</i>	<i>Date</i>
Australia	India	Melbourne	10/2/2008
Australia	India	Sydney	2/3/2008
Australia	India	Brisbane	4/3/2008

- (Q6) List matches played in which India or Australia was *Team1*.

The condition in this query is slightly more complex. Figure 5.19 shows how this condition can be formulated in SQL.

In this query also, match information is retrieved when *Team1* column value is either Australia or India. Any number of values may be included in the list that follows IN.

The result of query Q6 above is given in Table 5.9.

Constants like ‘Australia’ or ‘India’ are often called a *literal tuple*. A literal tuple of course could have a number of values. For example, it could be

WHERE {2675, Australia, India, Melbourne, 10/2/2008, Team1} IN Match

```
SELECT Team1, Team2, Ground, Date
FROM Match
WHERE Team1 = 'Australia'
```

Figure 5.18 SQL for query Q5

```
SELECT Team1, Team2, Ground, Date
FROM Match
WHERE Team1 = IN {'Australia','India'}
```

Figure 5.19 SQL for query Q6

Table 5.9 Result of query Q6

<i>Team1</i>	<i>Team2</i>	<i>Ground</i>	<i>Date</i>
India	England	Delhi	28/3/2006
India	Australia	Mohali	29/10/2006
India	West Indies	Nagpur	21/1/2007
India	West Indies	Vadodara	31/1/2007
India	Sri Lanka	Rajkot	11/2/2007
India	Australia	Mumbai	17/10/2007
India	Pakistan	Guwahati	5/11/2007
Australia	India	Melbourne	10/2/2008
India	Sri Lanka	Adelaide	19/2/2008
Australia	India	Sydney	2/3/2008
Australia	India	Brisbane	4/3/2008

The set of columns in the list before IN must match the list of columns in the list that follows IN for the set membership test carried out by IN to be legal (the list in the example above being a table).

```
SELECT Team1, Team2, Ground, Date
FROM Match
WHERE Team1 = 'Australia'
      OR Team1 = 'India'
```

The above query in Fig. 5.19 is equivalent to the following SQL query shown in Fig. 5.20.

Figure 5.20 SQL for query Q6

SQL allows some queries to be formulated in a number of ways. We will see further instances where a query can be formulated in two or more different ways.

The above query could use aliases if the headings in Table 5.9 are to be changed. For example,

```
SELECT Team1 AS HTeam, Team2 AS Visitors, Ground, Date
FROM Match
WHERE Team1 = 'Australia'
      OR Team1 = 'India'
```

Figure 5.21 SQL for query Q6 using aliases for column names

Removing Duplicates from the Information Retrieved

- (Q7) Find IDs of all players that have bowled in an ODI match in the database.

Figure 5.22 shows how this query may be formulated in SQL. A WHERE clause is not required since we want IDs of all players in the table *Bowling*.

The above query is a projection of the table *Bowling*. Only the *PID* column has been selected and DISTINCT is used to remove duplicate values.

```
SELECT DISTINCT (PID)
FROM Bowling
```

Figure 5.22 SQL for query Q7

The result of query *Q7* is given in Table 5.10.

DISTINCT may also be used when the query retrieves more than one column. The next query presents an example illustrating this fact.

- (*Q8*) Find names of teams and grounds where India has played an ODI match outside India.

We need to make a number of assumptions in formulating this query. We assume that if *Team2* is India then India is playing outside India. We also assume that India could have played the same country at the same ground more than once (for example, Australia at Melbourne) and therefore we should remove the duplicates.

The query may now be formulated as in Fig. 5.23 in SQL.

Obviously the list obtained by specifying *DISTINCT (Team1, Ground)* would be smaller than that obtained by specifying *(Team1, Ground)* without the *DISTINCT* clause although our database is so small that it had only one duplicate, playing Sri Lanka in Colombo.

The result of query *Q8* is the teams and grounds on which India has played ODI matches outside India as given in Table 5.11.

Table 5.10 Result of query Q7

PID
99001
24001
23001
94002
92002
91001
23001
98002

```
SELECT DISTINCT (Team1, Ground)
FROM Match
WHERE Team2 = 'India'
```

Figure 5.23 SQL for query Q8

Table 5.11 Result of query Q8 formulated in Fig. 5.23

Team1	Ground
Pakistan	Peshawar
Pakistan	Rawalpindi
West Indies	Kingston
Sri Lanka	Colombo
South Africa	Cape Town
England	Southampton
Australia	Melbourne
Australia	Sydney
Australia	Brisbane
Pakistan	Karachi

- (*Q9*) Find IDs of all players that batted in match 2755.

To formulate this query, we require a *WHERE* clause that specifies the condition for rows in the table *Batting* such that the rows are for match 2755. Figure 5.24 shows how this query may be formulated in SQL.

This query also involves a selection as well as a projection. Note that the *MatchID* attribute is being treated as a string.

```
SELECT PID
FROM Batting
WHERE MatchID = '2755'
```

Figure 5.24 SQL for query Q9

*image
not
available*

*image
not
available*

- WHERE C1 OR C2—Each selected row must satisfy either the condition C1 or the condition C2 or both conditions C1 and C2.
- AND and OR operators may be combined, for example, in a condition like *A AND B OR C*. In such compound conditions the AND operation is done first. Parentheses could be used to ensure that the intent is clear, for example, *(A AND B) OR (C)*. A more complex example may help: *A AND B OR C AND D OR E AND F* would be processed as *(A AND B) OR (C AND D) OR (E AND F)* and will be true if any one or more of the conditions *A AND B*, *C AND D*, *E AND F* is true.
- WHERE NOT C1 AND C2—Each selected row must satisfy condition C2 but must not satisfy the condition C1.
 - WHERE *A* IN—Each selected row must have the value of *A* in the list that follows IN. NOT IN may be used to select all that are not in the list that follows NOT IN.
 - WHERE *A* operator ANY—Each selected row must satisfy the condition for any of the list that follows ANY. Any of the following operators may be used in such a clause: greater than ($>$), less than ($<$), less than equal (\leq), greater than equal (\geq) and not equal (\neq).
 - WHERE *A* operator ALL—Each selected row now must satisfy the condition for each of the list that follows ALL. Any of the following operators may be used in such a clause: greater than ($>$), less than ($<$), less than equal (\leq), greater than equal (\geq) and not equal (\neq).
 - WHERE *A* BETWEEN *x* AND *y*—Each selected row must have value of *A* between *x* and *y* including values *x* and *y*. NOT BETWEEN may be used to find rows that are outside the range *(x, y)*.
 - WHERE *A* LIKE *x*—Each selected row must have a value of *A* that satisfies the string matching condition specified in *x*. The expression that follows LIKE must be a character string and enclosed in apostrophes, for example ‘Delhi’.
 - WHERE *A* IS NULL—Each selected row must satisfy the condition that *A* is NULL, that is, a value for *A* has not been specified.
 - WHERE EXISTS—Each selected row be such that the subquery following EXISTS does return something, that is, the subquery does not return a NULL result.
 - WHERE NOT EXISTS—Each selected row be such that the subquery following NOT EXISTS does not return anything, that is, the subquery returns a NULL result.

5.4.4 Queries Involving More than One Table—Using Joins and Subqueries

Many relational database queries require more than one table because each table in the database is about a single entity. For example, player information including player name is in table *Player* while batting and bowling performances are provided in tables *Batting* and *Bowling*. Therefore any time we wish to retrieve information about the batting or bowling performance of a player by using the player’s name (and not the *Player ID*) we have to use two tables as shown in the examples that follow.

There are two possible approaches to using more than one table. One approach is to use a join. The other approach is to use one or more subqueries as shown in the next few examples. When using a join, most queries below only consider an inner equi-join.

Subqueries

A subquery is a query that is a part of another query. Subqueries may also include subqueries and this may continue to any number of levels. The queries presented below use subqueries in the WHERE clause but subqueries may also be used in the FROM clause instead of a table name as we will show later.

When a subquery is used in the WHERE clause, it normally tests for two types of conditions. The first type of condition usually tests a condition on the result from the subquery. For example, it may use formulation like “*PID IN <subquery>*” or “*PID NOT IN <subquery>*”. The second type of condition is like “*EXISTS <subquery>*” or “*NOT EXISTS <subquery>*” to test that a non-null result is returned by the subquery or a null result is returned. We illustrate these conditions now.

- (Q12) Find Match IDs of all matches in which Tendulkar batted.

As noted above, this query needs to use two tables as the player name is given only in the table *Player* and the batting performance is given only in the table *Batting*. We will use a subquery to obtain the *PID* of Tendulkar and use that information to obtain *Match IDs* of matches that he has played in.

This query is a simple use of a subquery. The subquery only returns a single constant which is compared with another value in the WHERE clause. If the comparison's result is true then the row in the outside query is selected otherwise it is not.

In a query like this in which the subquery returns only one value, *PID IN* could be replaced by *PID =* but if we are not certain that the result of the subquery will be a constant then it is best to use the IN operator.

Figure 5.28 shows how this query may be formulated in SQL.

The above formulation of the query has a subquery (or a *nested query*) associated with it. The subquery feature is very useful and as noted earlier, a subquery may itself have a subquery within it and queries may be nested to any level.

The result of the above query is given in Table 5.15. Due to very limited data in our database only one ODI is found.

It may be appropriate to briefly discuss how such a query is processed. The subquery is processed first resulting in a new table (in the present case a single column table with only one row). We have of course assumed that the last name Tendulkar is unique in the database. This is unlikely to be true if the last name of the player was Singh or some other common name. This resulting table is then used in the outer query in which the WHERE clause becomes true if the *PID* value of the player in the *Batting* table is in the set of *PIDs* that is returned by the subquery. Of course, in this case there is only one member in the set returned by the subquery. The IN operation in the WHERE clause tests for set membership. The clause may also use construct NOT IN instead of IN when appropriate.

Another format for the IN condition is illustrated in the following query.

- (Q13) Find the match information of all matches in which Dhoni has batted.

```
SELECT MatchID
FROM Batting
WHERE PID IN
    (SELECT PlayerID
     FROM Player
     WHERE Lname = 'Tendulkar')
```

Figure 5.28 SQL for query Q12

Table 5.15 Result of query Q12

MatchID
2689

Now we show how a subquery may itself use a subquery in its WHERE clause.

This query must involve the three tables *Match*, *Batting* and *Player* as the match information is in table *Match*, the player names are in table *Player* and batting information is available only in table *Batting*. In this query we use a subquery that itself uses a subquery within it. The innermost subquery obtains the *Player ID* of Dhoni, then the next level subquery is used to obtain the *Match ID* of matches he has batted in and then use those *Match IDs* in the outermost query to obtain information about the matches.

```
SELECT MatchID, Team1, Team2, Ground, Date
FROM Match
WHERE MatchID IN
    (SELECT MatchID
     FROM Batting
     WHERE PID IN
         (SELECT PlayerID
          FROM Player
          WHERE LName = 'Dhoni'))
```

Figure 5.29 SQL for query Q13

Figure 5.29 shows how this query may be formulated in SQL.

This query results in the last subquery (or innermost subquery) being processed first. This subquery involves a selection of table *Player* which results in a table with only one row that is the *Player ID* of Dhoni. This result is then used to find a list of *Match IDs* from table *Batting* in which Dhoni has batted. These *Match IDs* are then used to find match information from the table *Match*.

Note that each subquery is executed only once and its result is used as a constant or a relation in the next outer query.

The result of the query is only two matches in our example database. The results are presented in Table 5.16.

Table 5.16 Result of query Q13

MatchId	Team1	Team2	Ground	Date
2689	Australia	India	Brisbane	4/3/2008
2755	Sri Lanka	India	Colombo	27/8/2008

We now look at how these queries can be formulated using a join. Additional queries that use subqueries are presented following the discussion of join.

Join Queries

It is possible to combine rows from two tables (or any number of tables) listed in the FROM clause by specifying the join condition(s) in the WHERE clause. We illustrate join by reformulating the last two queries.

The two queries, *Q12* and *Q13*, involving subqueries may be reformulated using the join operator as given in Figs 5.30 and 5.31 respectively.

Reformulation of query *Q13* is given in Fig. 5.31.

```
SELECT MatchID
FROM Batting, Player
WHERE PID = PlayerID
AND LName = 'Tendulkar'
```

Figure 5.30 SQL query using a join for query Q12

It is worth discussing how the join operator is used in the above queries. When the FROM clause specifies more than one table name, the query processor (conceptually) forms a Cartesian product of those tables and

applies the join condition specified in the WHERE clause. Any other conditions in the WHERE clause are then also applied. In practice, the algorithm used is not that simple (and inefficient). Fast algorithms, to be discussed in Chapter 8, are used for computing joins.

In both joins in Figs 5.30 and 5.31, the join condition is an equality condition and therefore equi-join is used in both SQL formulations.

We should also note that, in queries like those in Figs 5.30 and 5.31, it may sometimes be necessary to qualify some of the column names if the same name is used in more than one table in the FROM clause. For example, if the column *MatchID* was called by the same name in *Match* and *Batting* then one of the equalities in the WHERE clause will have to be *Match.MatchID = Batting.MatchID*. Such qualifications are needed only when there is a chance of ambiguity.

As noted earlier, it is possible to give aliases to some table names (for example, *Batting* may be given an alias *b1*) when names are long or more than one instance of the same table is used in a query.

- (Q14) Find the player ID of players who have made a century in each of the ODI matches 2755 and 2689.

The table *Batting* has a separate row for every batsman in each innings. We can therefore either use a subquery or use a join. We use a join of table *Batting* with itself.

Figure 5.32 shows how this query may be formulated in SQL.

There are no players that have scored a century in our database and therefore the result is null.

This query is presented to illustrate that sometimes we may need to join a table to itself. In this query we have joined the table *Batting* to itself such that the join column is *PID*. In addition we have ensured that the first score is from the Match 2755 and the second score is from Match 2689 and both scores are 100 or more. We then retrieve the *Player IDs*.

The resulting join table in the query above would have rows such that each row has two scores of a player from the two ODI matches and both scores are 100 or more.

The query formulation in Fig. 5.32 also shows, as discussed earlier, that a table may be given an alias if necessary. In this case one instance of table *Batting* is given the name *b1* while the other is given the name *b2*.

More Subqueries

- (Q15) Find IDs of players that have both bowled and batted in the ODI match 2689.

This query needs to access both the tables *Batting* and *Bowling*. We will use a subquery formulation in which the subquery finds player IDs of those players that bowled in match 2689. The outer subquery then finds player IDs of those players that batted in match 2689 if they are also in the list of players that bowled in that match. Note that the inner subquery is executed only once.

```
SELECT Team1, Team2, Ground, Date
FROM Match, Batting, Player
WHERE Match.MatchID = Batting.MatchID
AND PlayerID = PID
AND LName = 'Dhoni'
```

Figure 5.31 SQL query using a join for query Q13

```
SELECT PID
FROM Batting b1, Batting b2
WHERE b1.PID = b2.PID
AND b1.MatchID = 2755
AND b2.MatchID = 2689
AND b1.NRuns > 99
AND b2.NRuns > 99
```

Figure 5.32 SQL for query Q14

Figure 5.33 shows how this query may be formulated in SQL.

As noted above, the subquery finds the player IDs of those players that have bowled in Match 2689 while the outside subquery finds from this list of bowler player IDs those players that have also batted.

The result of this query is given in Table 5.17.

This query could also be formulated as an intersection of the two queries, one that finds the player IDs of all the bowlers and the other that finds the player IDs of all the batsmen. The intersection then gives the player IDs of those players that are in both player ID lists.

- (Q16) Find IDs of players that have either bowled or batted (or did both) in the ODI match 2689.

This query is similar to the last query, again requiring both tables *Batting* and *Bowling*. In this case, a subquery formulation is presented in which the subquery finds player IDs of all those players that bowled in match 2689. The outer query then finds player IDs of all those players that batted in match 2689 or if they are on the list of players that bowled in that match. Note that the inner subquery is executed only once.

Figure 5.34 shows how this query may be formulated in SQL.

```
SELECT PID
FROM Batting
WHERE MatchID = '2689'
OR PID IN
  (SELECT PID
   FROM Bowling
   WHERE MatchID = '2689')
```

Figure 5.34 Incorrect SQL for query Q16

Using UNION, INTERSECT and MINUS (or EXCEPT)

The query Q16 may also be written using the UNION operator as given in Fig. 5.35.

```
SELECT PID
FROM Batting
WHERE MatchID = '2689'

UNION

(SELECT PID
 FROM Bowling
 WHERE MatchID = '2689')
```

Figure 5.35 SQL for query Q16

```
SELECT PID
FROM Batting
WHERE MatchID = '2689'
AND PID IN
  (SELECT PID
   FROM Bowling
   WHERE MatchID = '2689')
```

Figure 5.33 SQL for query Q15

Table 5.17 Result of query Q15

PID
99001
24001
23001
98002

The result of this query is given in Table 5.18.

Figure 5.35 SQL formulation of the query would produce duplicates in the result because some players would appear in both the lists, the list of bowlers for match 2689 and the list of batsmen for match 2689. We should note that when UNION is used the data type of the columns in the first SELECT command must match the data type of the corresponding column in the second SELECT command. In the above query there is no difficulty as we are retrieving only one column which obviously matches.

Can this query be formulated in another way; perhaps using the table match in addition to tables *Batting* and *Bowling*?

What would the query become if UNION was replaced by INTERSECT or MINUS in Fig. 5.35?

Let us first consider replacing UNION with INTERSECT as shown in Fig. 5.36.

```
SELECT PID
FROM Batting
WHERE MatchID = '2689'

INTERSECT

(SELECT PID
FROM Bowling
WHERE MatchID = '2689')
```

Figure 5.36 SQL query using INTERSECT

Set intersection of any two sets results in those members of the two sets that are common to both sets. Therefore the query in Fig. 5.36 results in giving us a list of *PIDs* for players who have batted as well as bowled in Match 2689. The result therefore is the same as Table 5.17.

Let us now use EXCEPT, which is the same as MINUS, in Fig. 5.37.

```
SELECT PID
FROM Batting
WHERE MatchID = '2689'

EXCEPT

SELECT PID
FROM Bowling
WHERE MatchID = '2689'
```

Figure 5.37 SQL query using EXCEPT

Table 5.18 Result of query Q16

<i>PID</i>
89001
98002
23001
25001
99002
95001
24001
99001
27001

Set difference of any two sets, results in those members of the first set that are not in the second set. Therefore the query in Fig. 5.37 results in giving us a list of *PIDs* for players who batted but did not bowl in Match 2689. This is reformulated in the next query and its result is the same as in Table 5.19.

- (Q17) Find IDs of players that have batted in match 2689 but have not bowled.

Figure 5.38 shows another way this query may be formulated in SQL. Given the difficulty in formulating the query Q16, we must formulate this one carefully.

This query is equivalent to the query in Fig. 5.37 using EXCEPT to select those players that are selected by the outer query (i.e., those that have batted in the match) as long as they are not in the list of players that are returned by the subquery, that is, the list of players that bowled in that match. This has been done by using the “*PID NOT IN*” in the WHERE clause. Note that the subquery is only executed once.

The result of this query is given in Table 5.19.

```
SELECT PID
FROM Batting
WHERE MatchID = '2689'
AND PID NOT IN
(SELECT PID
FROM Bowling
WHERE MatchID = '2689')
```

Figure 5.38 SQL for query Q17

Table 5.19 Result of query Q17

<i>PID</i>
89001
25001
99002
95001
27001

5.4.5 Using Subquery in the FROM Clause

A subquery may also be used in the FROM clause. We illustrate this use by an example given in Fig. 5.39.

- (Q18) Find the *Match IDs* of matches in which Sachin Tendulkar has played.

This subquery in Fig. 5.39 uses a subquery in the FROM clause which returns a table as its result and is therefore treated like a virtual table. In this particular case, the table returned has only one element which is the *PlayerID* of Sachin Tendulkar. This subquery table must be given a name which is ST in this query. We can now use this name in the WHERE clause.

```
SELECT MatchID
FROM Batting, (SELECT PlayerID
               FROM Player
               WHERE LName = 'Tendulkar') ST
WHERE PID = ST.PlayerID
```

Figure 5.39 Using a subquery in the FROM Clause

This use of subquery in the FROM clause is helpful in answering a variety of queries that use multiple aggregate functions as we will show later in this chapter.

We are now ready to illustrate the use of different forms of the WHERE clause in example queries that follow.

- (Q19) Find Match IDs of those matches in which player 27001 bats and makes more runs than he made at every match he played at Brisbane.

To formulate this query we first find all the scores of the player 27001 when he batted in Brisbane. This requires two subqueries, the innermost subquery and the middle subquery, since batting performance is given in table *Batting* and the name of the ground is available only in the table *Match*. Once the Brisbane scores are available, the outermost query finds Match IDs of matches where he scored more.

Figure 5.40 shows how this query may be formulated in SQL.

As noted earlier, when ALL is used in the WHERE clause as in Fig. 5.40 above, the value in the outer query must satisfy the specified condition with all the rows that are returned by the subquery. In this particular query the outermost query condition *NRuns* > ALL would be true only if the value of attribute *NRuns* is larger than the largest value of *NRuns* returned by the middle subquery.

It should also be noted that the outermost query will scan the whole table *Batting* including the matches played in Brisbane but none of the Brisbane scores would meet the condition.

The result of this query is given in Table 5.20.

- (Q20) Find IDs and scores of players who scored less than 75 but more than 50 in Colombo.

Once again we need to use a subquery because the information about the grounds is given in table *Match* while the batting information is in *Batting*. We will use the clause WHERE something BETWEEN x AND y although we could also use two WHERE conditions such as WHERE something \geq x AND something \leq y.

Figure 5.41 shows how this query may be formulated in SQL.

```
SELECT MatchID
FROM Batting
WHERE PID = '27001'
AND NRuns > ALL
  (SELECT NRuns
   FROM Batting
   WHERE PID = '27001'
   AND MatchID IN
     (SELECT MatchID
      FROM Match
      WHERE Ground = 'Brisbane'))
```

Figure 5.40 SQL for query Q19

Table 5.20 Result of query Q19

MatchID
2755

```
SELECT PID, NRuns
FROM Batting
WHERE NRuns BETWEEN 51 AND 74
AND MatchID IN
  (SELECT MatchID
   FROM Match
   WHERE Ground = 'Colombo')
```

Figure 5.41 SQL for query Q20

Note that we are looking for scores between 51 and 74 and not between 50 and 75. This is because $(A \text{ BETWEEN } x \text{ AND } y)$ has been defined to mean $A \geq x$ and $A \leq y$.

The result of this query is given in Table 5.21.

Table 5.21 Result of query Q19

PlayerID	NRuns
25001	71
91001	60

- (Q21) Find the player IDs of those players whose date of first test match (*FTest*) is not given in the database.

As noted earlier, SQL provides a special command for checking if some information is missing as shown in Fig. 5.42 below.

```
SELECT PlayerID
FROM Player
WHERE FTest IS NULL
```

Figure 5.42 SQL for query Q21

It should be noted that the column value being NULL is very different than it being zero. The value is NULL only if it has been defined to be NULL.

The result of this query is given in Table 5.22.

Table 5.22 Result of query Q21

PlayerID
27001

5.4.6 Correlated Subqueries and Complex Queries

A number of queries above have used subqueries. The subqueries used were needed to be evaluated only once. Such subqueries are called *noncorrelated subqueries*. There is another type of subqueries that are more complex and are called *correlated subqueries*. In these subqueries the value retrieved by the subquery depends on a variable (or variables) that the subquery receives from the outer query. A correlated subquery thus cannot be evaluated once and for all since the outer query and the subquery are related. It must be evaluated repeatedly; once for each value of the variable received from the outer query.

The correlated subqueries are most often used to check for existence or absence of matching rows in the parent table and the related table in the subquery. A correlated subquery therefore always refers to one or more columns from a table in the outer query. We illustrate such queries in examples ahead.

- (Q22) Find the names of players who batted in match 2689.

The query *Q8* required finding the player IDs of players who batted in a match. This query is a little bit different since in this query we wish to find the names (not player IDs) of players who have batted. We now need to use the two tables *Player* and *Batting* because player names are available only in the table *Player* while batting information is given in table *Batting*. In this query we look at each player in the table *Player* and then check, using a subquery, if there is a batting record for him in the table *Batting*.

Figure 5.43 shows how this query may be formulated in SQL.

The WHERE EXISTS clause returns true if the subquery following the EXISTS returns a non-null result. Similarly a WHERE NOT EXISTS clause returns true only if the subquery following the NOT EXISTS returns nothing, that is, a null table.

This query produces Table 5.23 as result.

Table 5.23 Result of query Q22

FName	LName
Sachin	Tendulkar
M. S.	Dhoni
Yuvraj	Singh
Adam	Gilchrist
Andrew	Symonds
Brett	Lee
Praveen	Kumar
Ricky	Ponting
Harbhajan	Singh

Again, the above query can be formulated in other ways. One of these formulations uses a join. Another uses a different subquery. Using the join, we obtain the SQL query given in Fig. 5.44.

The join table would have rows only for players that have batted in match 2689 since a selection has been applied after the join.

- (Q23) From the example database, find the player IDs of players who have scored more than 30 in every ODI match that they have batted.

SQL provides no direct way to check that for a given player all his batting or bowling performances satisfy some condition like all scores being above 30 but we can obtain the same information by reformulating the question. Instead of asking for players that have scored more than 30 every time they have batted, we ask for players that have batted but never scored less than 31.

```
SELECT FName, LName
FROM Player
WHERE EXISTS
  (SELECT *
   FROM Batting
   WHERE PlayerID = PID
   AND MatchID = '2689')
```

Figure 5.43 SQL for query Q22

```
SELECT FName, LName
FROM Player, Batting
WHERE PlayerID = PID
AND MatchID = '2689'
```

Figure 5.44 SQL for query Q22

Reformulating the question makes the query somewhat complex. The SQL query below formulates the query as “find the *Player IDs* of players that have no score of less than 31 in any ODI innings that they have batted”!

Figure 5.45 shows how this query may be formulated in SQL.

Note that the subquery in Fig. 5.45 cannot be executed only once since it is looking for rows in *Batting* that satisfy the condition *NRuns* < 31 for each player in the table *Batting* whose *PID* has been supplied to the subquery by the outside query. The subquery therefore must be evaluated repeatedly; once for each value of the variable *PID* received by the subquery. The query is therefore using a correlated subquery.

Note that the condition in the WHERE clause will be true only if the subquery returns a null result and the subquery will return a null result only if the player has no score which is less than 31.

Note that we have used aliases in Fig. 5.28 to label two instances of *Batting* as *b1* and *b2*. Also the heading of the query output has been changed to *PlayerID* by using an alias in the SELECT clause.

The result of this query is presented in Table 5.24.

It is worth considering why we used the table *Batting* in the outside query. Should we have used the table *Player* in it? The reason for using the table *Batting* in the outside query is that we want to first find a player that has batted and then check if the player has ever scored less than 31. We do not need to check a player that has never batted according to our example database.

Some further explanation might be helpful. The outer query looks at each row of the table *Batting* and then checks if the player qualifies. The checking is done in the subquery which checks if the players in Table *b1* have ever scored below 31.

We also wish to note that the above query formulation has a weakness that an alert reader would have already identified. The above query would retrieve *PID* of a player that qualifies for each value of *NRuns* that is greater than 30 that the player has. Therefore if a player has batted five times and has scored more than 30 every time, his ID will be retrieved five times in the result. *DISTINCT* may be used to remove such duplicates. Another approach would be to use the table *Player* instead of table *Batting b1* in the outside query but this leads to another problem that we discussed earlier.

- (Q24) Find the names of all players that have batted in all the ODI matches in Melbourne that are in the database.

This is a complex query that uses a correlated subquery because of the nature of SQL as it does not include the universal quantifier (*forall*) that is available in relational calculus. Once again we need to reformulate the question we are asking and this time it becomes quite ugly with a double negative. The reformulated question is “Find the names of players for whom there is no ODI match played in Melbourne that they have not batted ”! The SQL query that solves it is shown in Fig. 5.46.

```
SELECT PID AS PlayerID
FROM Batting b1
WHERE NOT EXISTS
  (SELECT *
   FROM Batting b2
   WHERE b1.PID = b2.PID
   AND NRUNS < 31)
```

Figure 5.45 SQL for query Q23

Table 5.24 Result of query Q23

PlayerID
25001
80001



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

This is not a correlated query since the subquery does not use a variable from the outer query and is executed only once.

The result of the query *Q27* is presented in Table 5.27.

Ishant Sharma is the youngest player having been born in 1988.

We note that we have used the equality comparison “*YBorn* =” just outside the subquery. Technically this is comparing an atomic attribute value of *YBorn* with the result of a subquery which only returns a table as the result, like all queries. In this query however it is clear that the result of the subquery is an aggregation MAX and therefore an atomic value. Even if more than one row has the same MAX value, only a single value will be returned which may be compared with an attribute value.

When we need to find the row with an attribute value that is the largest or smallest, the functions MAX and MIN are used but SQL does not allow us to formulate the query in a straightforward manner as that given in Fig. 5.51.

This SQL query is illegal although some systems will produce an output that will include all values of *LName* combined with the value of MAX(*YBorn*). The reason it is illegal is that when an aggregate operation is used in the SELECT clause then only aggregation operations may be used except when the GROUP BY clause is used. When the system finds the value of MAX(*YBorn*), it only looks at the *YBorn* column and values of other attributes corresponding to rows with the MAX value are then not available. We therefore must first find what the MAX(*YBorn*) value is and then find the row which has the attribute value equal to that MAX value.

Further queries using the built-in functions are presented after we have considered a mechanism for grouping a number of rows having the same value of one or more specified column(s).

5.4.8 Using the GROUP BY and HAVING Clauses

So far we have considered relatively simple queries that use aggregation functions like MAX, MIN, AVG and SUM. These functions are much more useful when used with GROUP BY and HAVING clauses which divide a table into virtual tables and apply qualifications on those virtual tables.

GROUP BY and HAVING clauses allow us to consider groups of records together that have some common characteristics (for example, players from the same country) and compare the groups in some way or extract information by aggregating data from each group. The group comparisons also are usually based on using an aggregate function.

The general form of a GROUP BY query is as shown in Fig. 5.52.

The SELECT clause must include column names that are in the GROUP BY column list (other column names are not allowed in the SELECT clause) and aggregate functions applied to those columns. These results of aggregate functions should normally be given a column name for output. The HAVING clause must

Table 5.27 Result of query *Q27*

Youngest_Player
Sharma

```
SELECT LName, MAX(YBorn)
FROM Player
```

Figure 5.51 An illegal formulation of query *Q27*

```
SELECT something_of_interest
FROM table(s)
WHERE condition_holds
GROUP BY column_list
HAVING group_condition
```

Figure 5.52 The basic SELECT command in using GROUP BY



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```

SELECT PID AS PlayerID, AVG(NRuns) AS AveRuns
FROM Batting
WHERE MatchID IN
  (SELECT MatchID
   FROM Match
   WHERE TeamI = 'Australia')
GROUP BY PID
  
```

Figure 5.56 SQL for query Q31

This query first finds all rows in *Batting* that are for matches played in Australia by using the subquery. Then the rows are grouped by *PID* and the average score for each player is computed. Unfortunately our example database has batting information from only one ODI that was played in Australia (although *Match* table has information on matches 2675 and 2688 also) and therefore the average scores are the same as the score in that match.

The result of the query is given in Table 5.31.

- (Q32) Find the ID of players that had a higher average score than the average score for all players when they played in Sri Lanka.

Figure 5.57 shows how this query may be formulated in SQL.

Table 5.31 Result of query Q31

PlayerID	AveRuns
89001	91
23001	38
25001	36
99002	2
95001	1
24001	42
99001	7
27001	7
98002	3

```

SELECT PID AS PlayerID, AVG(NRuns) AS AveRuns
FROM Batting
GROUP BY PID
HAVING AVG(NRuns) >
  (SELECT AVG(NRuns)
   FROM Batting
   WHERE MatchID IN
     (SELECT MatchID
      FROM Match
      WHERE TeamI = 'Sri Lanka')
   GROUP BY PID)
  
```

Figure 5.57 SQL for query Q32

This query first finds the average score for all players that have batted in Sri Lanka and then the outermost query selects those players that have average scores that are greater than that score.

The result of query Q32 is given in Table 5.32.

There was only one match in Sri Lanka so the average in the middle query are equal to the number of runs scored in that match.

Table 5.32 Result of query Q32

PlayerID	AveRuns
25001	53.5
91001	60.0
89001	91.0
24001	42.0

Multiple Aggregate Functions

In some situations, more than one function may be required. Let us illustrate it by an example.

- (Q33) In the query Q28, we found the number of players in the database from each country. Now we find the average number of players from this list in Table 5.28.

Once we have counted the number of players from each country, as in query Q28, the result is a table to which one may be tempted to apply another function to compute the average number of players.

Figure 5.58 shows how one might formulate this query. The formulation is however incorrect.

How do we then formulate this query?

One approach is to use a subquery in the FROM clause. This subquery then produces the table given in Table 5.28. Once we have that table, we may compute the average as shown in Fig. 5.59.

In the subquery, the table *Player* is divided into groups, each group with the same *Country* value. The number of rows in each group is counted resulting in the table *NP* in the FROM clause. Now we can find the average of the *NPlayers* column of this table.

The result of the above query Q33 is given in Table 5.33.

Later in the chapter we will show how this query may be formulated using a view.

```
SELECT AVG(COUNT(*)) AS NAvePlayers
FROM Player
GROUP BY Country
```

Figure 5.58 An illegal SQL query for Q33

```
SELECT AVG(NPlayers) AS NAvePlayers
FROM
  (SELECT Country, COUNT(*) AS NPlayers
   FROM Player
   GROUP BY Country) NP
```

Figure 5.59 Legal SQL for query Q33

Table 5.33 Result of query Q33

NAvePlayers
3.2

5.4.9 Outer Join

The concept of outer join was discussed in the last chapter. Further details are now presented including some examples in SQL to illustrate the concept.

We will use two new tables. Figure 5.60 lists the top 10 batsmen in ODI matches and Fig. 5.61 lists the top 10 batsmen in Test matches (both based on the total number of runs scored and current on 1st January 2010). The cricket database we have been using does not illustrate the outer join as well as these tables.

The best ODI batsmen in the world on 1st January 2010 are given in Fig. 5.60.

The best test batsmen in the world on 1st January 2010 are given in Fig. 5.61.

As noted earlier the natural join command finds matching rows from the two tables that are being joined and rejects rows that do not match. For example, if we needed to find the names, the total number of runs scored and the number of centuries scored by each player in ODI matches and Test matches we would join *bestODIBatsmen* and *bestTestBatsmen* and then carry out a projection. The SQL query is given in Fig. 5.62.

Player	Span	Matches	Innings	Runs	Ave	Strike Rate	100s
SR Tendulkar	1989-2010	440	429	17394	44.71	85.90	45
ST Jayasuriya	1989-2010	444	432	13428	32.43	91.22	28
RT Ponting	1995-2010	330	321	12311	43.19	80.50	28
Inzamam-ul-Haq	1991-2007	378	350	11739	39.52	74.24	10
SC Ganguly	1992-2007	311	300	11363	41.02	73.70	22
R Dravid	1996-2010	339	313	10765	39.43	71.17	12
BC Lara	1990-2007	299	289	10405	40.48	79.51	19
JH Kallis	1996-2010	295	281	10409	45.25	72.01	16
AC Gilchrist	1996-2008	287	279	9619	35.89	96.94	16
Mohammed Yousuf	1998-2009	276	261	9495	42.96	75.30	15

Figure 5.60 The table *bestODIbatsmen*

Player	Span	Matches	Innings	TRuns	Ave	T100s
SR Tendulkar	1989-2010	162	265	12970	54.72	43
BC Lara	1990-2006	131	232	11953	52.88	34
RT Ponting	1995-2010	140	236	11550	55.26	38
AR Border	1978-1994	156	265	11174	50.56	27
SR Waugh	1985-2004	168	260	10927	51.06	32
R Dravid	1996-2010	137	237	11256	53.60	28
JH Kallis	1995-2010	133	225	10479	54.57	32
SM Gavaskar	1971-1987	125	214	10122	51.12	34
GA Gooch	1975-1995	118	215	8900	42.58	20
Javed Miandad	1976-1993	124	189	8832	52.57	23

Figure 5.61 The table *bestTestbatsmen*

```
SELECT ORuns, 100s, Player, TRuns, T100s
From bestODIbatsmen AS b1, bestTestbatsmen AS b2
WHERE b1.Player = b2.Player
```

Figure 5.62 SQL query involving a join

The result of this query is given in Table 5.34.

This result of course only gives information about players that are in both the tables given in Figs 5.60 and 5.61. No information is provided about players that are in one table but not the other. The outer join provides information about rows that match in the two tables and also about rows that do not match.

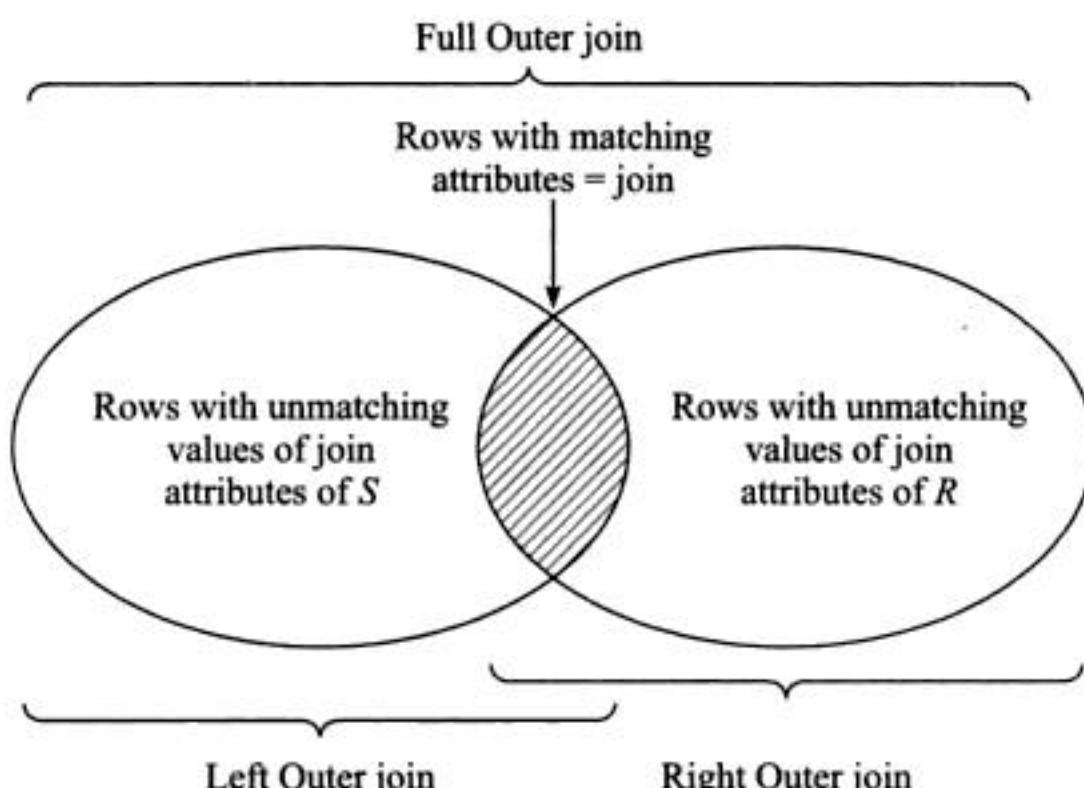
Table 5.34 Result of the inner join query

<i>ORuns</i>	<i>100s</i>	<i>Player</i>	<i>TRuns</i>	<i>T100s</i>
17394	45	SR Tendulkar	12970	43
12311	28	RT Ponting	11550	38
10405	19	BC Lara	11953	34
10765	12	R Dravid	11256	28
10409	16	JH Kallis	10479	32

Consider the Fig. 5.63 which explains the concept of inner and outer joins. The figure shows the distribution of join attribute values for two relations R and S . The rows for which the join attribute values match are shown in the middle and these rows form the natural join of the two tables. The remaining rows of the two relations are rows that did not find a matching row from the other relation. There are therefore three groups of rows from the two relations:

- Rows from both relations that have matching join attribute values and are therefore combined to form the (inner) natural join of R and S .
- Rows from relation R that did not match rows from S .
- Rows from relation S that did not match rows from R .

These three regions are shown in Fig. 5.63. It shows that the left outer join of tables R and S has all the rows of relation R including those that are in the natural join as well as those that are not. The right outer join has all the rows of relation S including those that are in the natural join as well as those that are not. Finally, the full outer join of the join has all the rows of R and S as well as those that form the natural join. We now illustrate these concepts using the example.

**Figure 5.63** Inner join and outer joins

- (Q34) Find the left outer join of the tables in Figs 5.60 and 5.61 and find the total number of runs scored, total number of centuries, and the strike rate in the ODIs for each player.

Consider the left outer join between the two tables *bestODIBatsmen* and *bestTestbatsmen*. It is formulated in SQL using the FROM clause as given in Fig. 5.64. We find the total number of runs scored in both types of the game and the total number of centuries scored in the query in Fig. 5.64.

```
SELECT ORuns+TRuns AS TR, 100s+T100s AS TC, SR, Player
From bestODIBatsmen AS b1 LEFT OUTER JOIN bestTestbatsmen AS b2
ON b1.Player = b2.Player
```

Figure 5.64 SQL query involving a left outer join

The result of this query Q34 is given in Table 5.35 which gives the total number of runs (TR), total number of centuries (TC) and strike rate for players that have made it to the list of top ten batsmen in both ODI matches and test matches but it also shows those players that appear in the list of top 10 ODI batsmen but do not in the list of top 10 test batsmen. Since they do not appear in both tables, some of the information for them is given as NULL. These entries did not appear in the result of a natural join in Table 5.34.

Table 5.35 Result of the left outer join query in query Q34

TR	TC	SR	Player
30364	88	85.90	SR Tendulkar
23861	66	80.50	RT Ponting
22358	53	79.51	BC Lara
22021	40	71.17	R Dravid
20888	48	72.01	JH Kallis
NULL	NULL	91.22	ST Jayasuriya
NULL	NULL	74.24	Inzamam-ul-Haq
NULL	NULL	73.70	SC Ganguly
NULL	NULL	96.94	AC Gilchrist
NULL	NULL	75.30	Mohammed Yusuf

Similarly we can carry out a right outer join of the two tables which will give us a list of players who are in both tables as well as those that are in the test batsmen table but not in the ODI table. We leave it as an exercise for the reader to find the right outer join.

- (Q35) Find the full outer join of the tables in Figs 5.60 and 5.61 and find the number of runs scored in ODIs and tests, total number of centuries in ODIs and tests, and the strike rate in the ODIs for each player.

The full outer join is illustrated in the query in Fig. 5.65 which is formulated once again using the FROM clause.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Modifying Existing Data

Every database will need modification of data from time to time; the data may have been input incorrectly or may be out-of-date. For example, an employee address or salary may have changed. The UPDATE command is used to carry out modifications.

The basic form of UPDATE command is as given in Fig. 5.69.

Table is the name of the table to be modified. SET Newvalues provides a list of new values for each table column specified while the WHERE condition selects the rows that are to be modified. An UPDATE statement may be used without a WHERE clause in which case the column values are modified for all the rows of the table. This is useful only infrequently.

- (Q37) Modify the number of centuries made by Tendulkar in Tables 5.61 to 51.

Figure 5.70 shows a case of modifying existing data as required in Q37.

- (Q38) Modify the number of matches and innings played by Tendulkar to 177 and 290, respectively.

We are modifying some more data in Table 5.61. This is shown in Fig. 5.71.

Deleting Data

Similar to UPDATE, all or some selected rows of a table may be deleted by a command illustrated in Fig. 5.72.

- (Q39) Delete Match 2689 from the table *Match*.

Figure 5.72 shows how this delete operation to delete a row corresponding to match 2689 may be formulated in SQL.

The format of the DELETE command is very similar to that of the SELECT command. The DELETE command goes down the table *Match* row by row and deletes rows that meet the condition *MatchID* = '2689'. DELETE may therefore be used to either delete one row or a group of rows that satisfy the given condition.

It should be understood that DELETE commands like that in Fig. 5.72 and the one in Fig. 5.73 below are likely to have side effects if the rows deleted by these commands are being referenced by rows in the other tables.

- (Q40) Delete all bowling records of Brian Lara.

Once again the query is similar to some we have seen before. We use a subquery to find the player ID of Brian Lara and then delete all rows for that *Player ID* from the table *Bowling*.

```
UPDATE Table
SET Newvalues
[WHERE condition]
```

Figure 5.69 SQL for UPDATE of rows

```
UPDATE bestTestbatsmen
SET T100s = 51
WHERE Player = "SR Tendulkar"
```

Figure 5.70 SQL for UPDATE of row for query Q37

```
UPDATE bestTestbatsmen
SET Matches = 177,
Innings = 290
WHERE Player = 'SR Tendulkar'
```

Figure 5.71 SQL for UPDATE of more than one column

```
DELETE Match
WHERE MatchID = '2689'
```

Figure 5.72 SQL for Delete Q39



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Another example is the following view definition that provides information about all the bowlers in ODI match 2689. Figure 5.75 shows how this can be created in SQL.

```
CREATE VIEW Bowling2689 (PID, FName, LName, Country, NOvers, NWickets)
AS SELECT PlayerID, FName, LName, Country, NOvers, NWickets
FROM Player, Bowling
WHERE PlayerID = PID
AND MatchID = '2689'
```

Figure 5.75 Creating a view for match 2689

Note that we can use column names for the view that are different than the column names in the base tables.

5.5.2 Querying Views

As noted earlier, views may be used in retrieving information as if they were base tables although views do not exist physically. When a query uses a view instead of a base table, the DBMS retrieves the view definition from meta-data and uses it to compose a new query that would use only the base tables. This query is then processed. Let us consider some examples.

- (Q41) Find the names of all players who have scored centuries.

Figure 5.76 shows how this query may be formulated in SQL using a view. In this case the view *Batsmen* defined in Fig. 5.74 is being used.

- (Q42) Find the names of bowlers that have taken five wickets or more in an ODI match 2689.

Figure 5.77 shows how this query may be formulated in SQL using a view. In this case the view *Bowling2689* defined in Fig. 5.75 is being used.

The above queries on views are transformed by the system by using the view definitions and then processed. The transformation is relatively easy as it only requires the view name to be replaced by the tables' names in the FROM clause of the view definition. A WHERE clause of the view definition is included in addition to the WHERE clause specified in the query on the view.

After transformation, the queries in Figs 5.76 and 5.77 would look like what is given in Figs 5.78 and 5.79.

```
SELECT FName, LName
FROM Batsmen
WHERE Score ≥ 100
```

Figure 5.76 Using the view *Batsmen*

```
SELECT FName, LName
FROM Bowling2689
WHERE NWickets ≥ 5
```

Figure 5.77 Using the view *Bowling2689*

```
SELECT FName, LName
FROM Player, Batting
WHERE PlayerID = PID;
WHERE NRuns ≥ 100
```

Figure 5.78 Query in Fig. 5.76 after transformation of the view definition



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

- Can views be updated?
- Can rows be inserted or deleted in a view in spite of views being virtual tables?

The answer to the first question is yes. There is no difficulty in using views and base tables in the same query as the query is going to be transformed into a query that uses base tables anyway.

The answer to the second and third question is maybe! Essentially if the view definition is such that a row in a view can directly identify one or more rows in the base tables then the views may be updated and rows be inserted in the view. It means the view must have a primary key of one or more base tables. The following points are worth noting about modifying data through views:

- A view to be modified must not contain any aggregate functions although a subquery used in the view may.
- The view to be modified must not use the duplicate removal DISTINCT.
- The view must not include calculated columns.
- A view may be deleted in a similar fashion as deleting a table.

Further discussion of this topic is beyond the scope of this book.

5.6 USING SQL IN PROCEDURAL PROGRAMMING LANGUAGES—EMBEDDED SQL AND DYNAMIC SQL

We have so far covered the interactive use of SQL in which the SQL queries are submitted from a terminal, executed and results returned to the terminal (sometime called *Direct SQL*). Although SQL provides powerful features to build programs, SQL is sometimes inadequate for writing application programs. Embedding SQL commands to access and manipulate the database in a conventional programming language (called the *host language*) like C or C++ allows power of SQL to be combined with the facilities of a programming language, for example, looping, to be used together to build application programs that use a database.

5.6.1 Embedded SQL

Use of embedded SQL is common in some application development environments. Embedded programming is useful when a previously written program in a procedural language needs to access a database or because SQL is not providing the facilities, for example, it has no looping facilities like IF..THEN..ELSE. On the other hand using SQL in a procedural programming language like C or C++ also creates some problems because the procedural languages are not equipped to deal with collections of multiple rows as a single operand. To overcome this difficulty, a facility is needed for stepping through a collection of rows returned by a SQL query, one row at a time. This is done by introducing the concept of a *cursor*.

It is possible to use SQL from within a program written in almost any of the conventional programming languages. Since conventional programming languages know nothing about SQL, no database table names may be used in a host language program outside the embedded SQL commands. An SQL query in a host

*image
not
available*

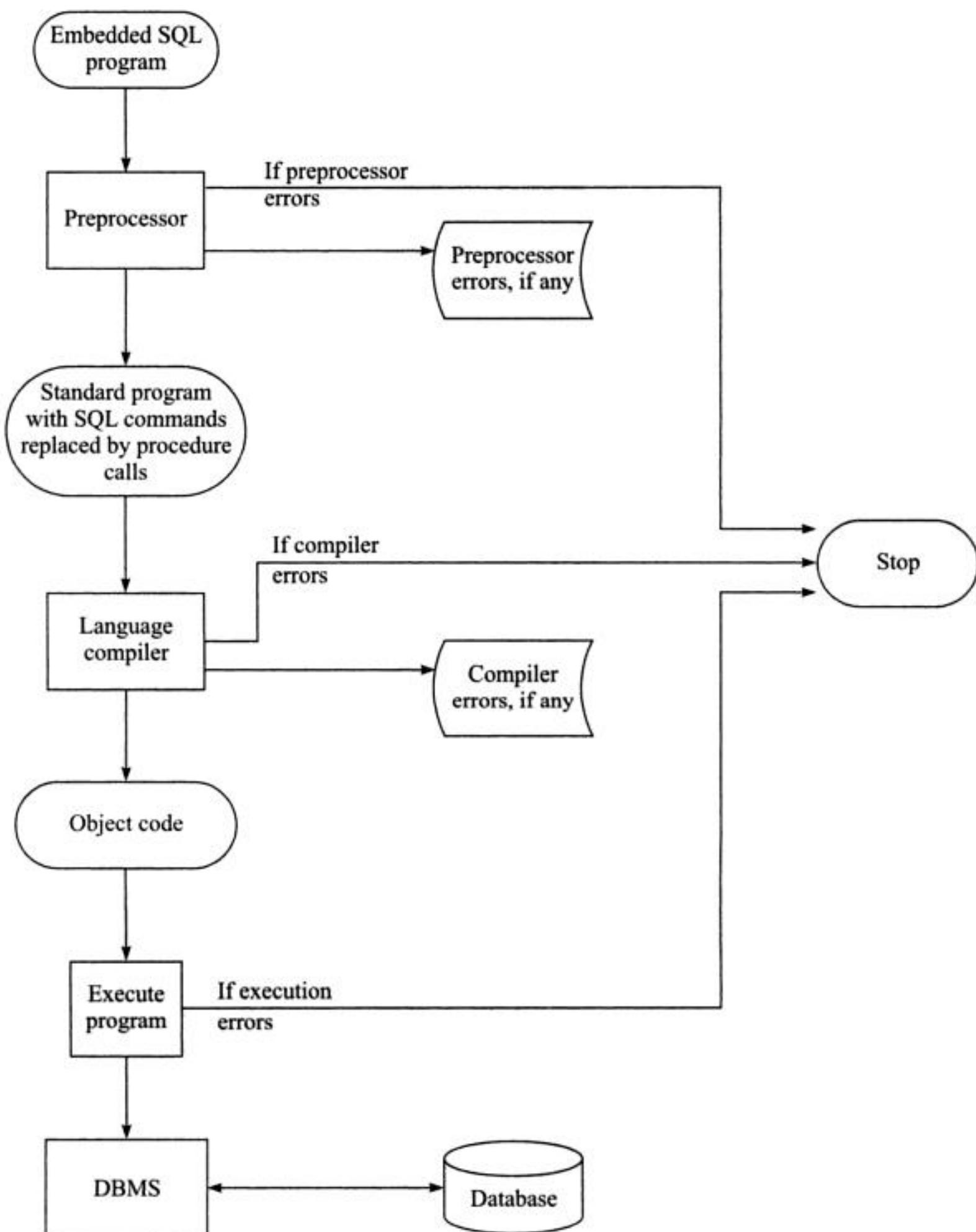


Figure 5.83 Preprocessing and compilation of an embedded SQL program

The host variables declared here are used in the SQL queries embedded in the host language program. Note that these variables are called shared variables and they are prefixed by a colon (:) when used.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

*image
not
available*

*image
not
available*



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



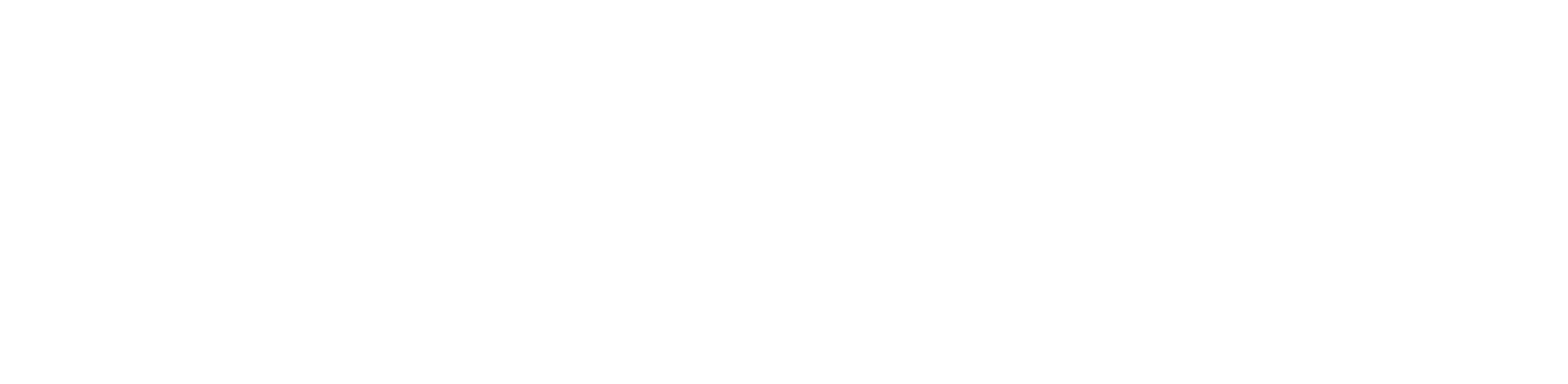
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



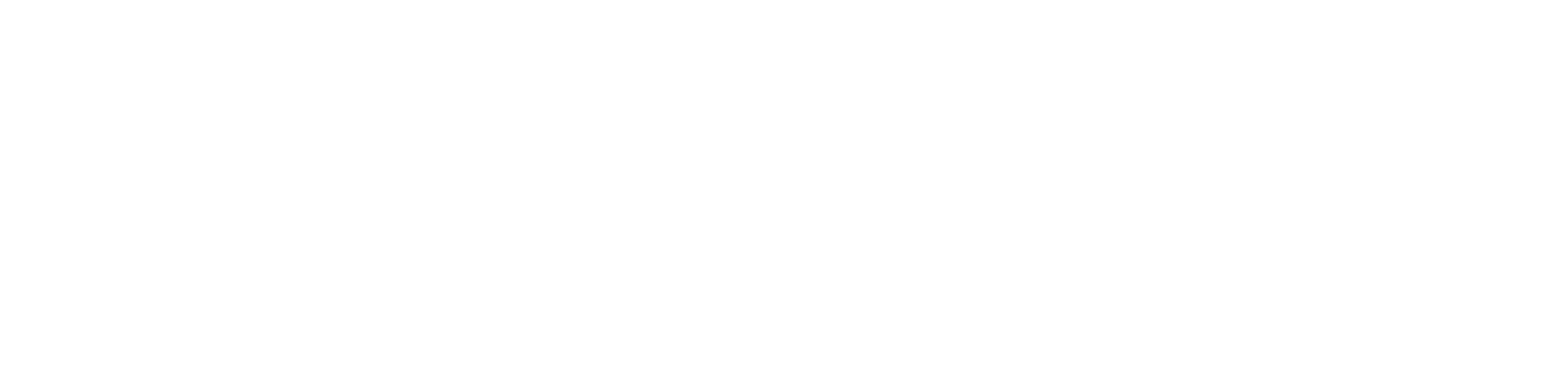
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



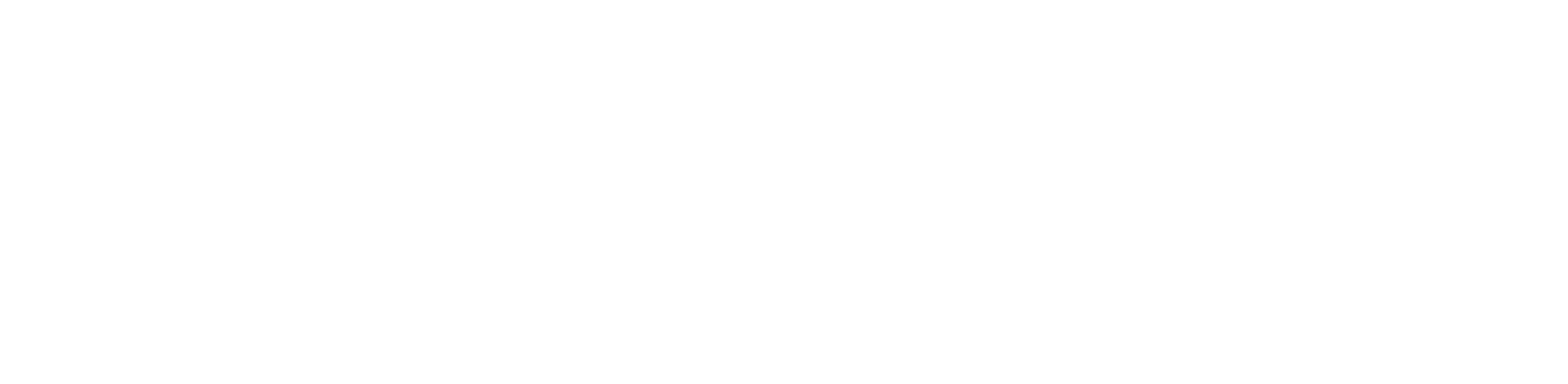
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



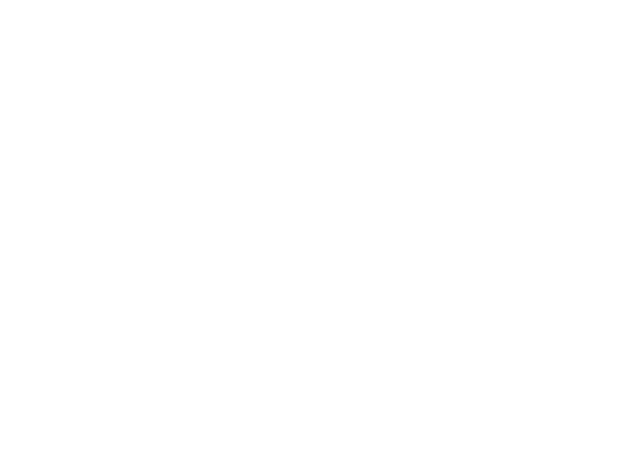
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



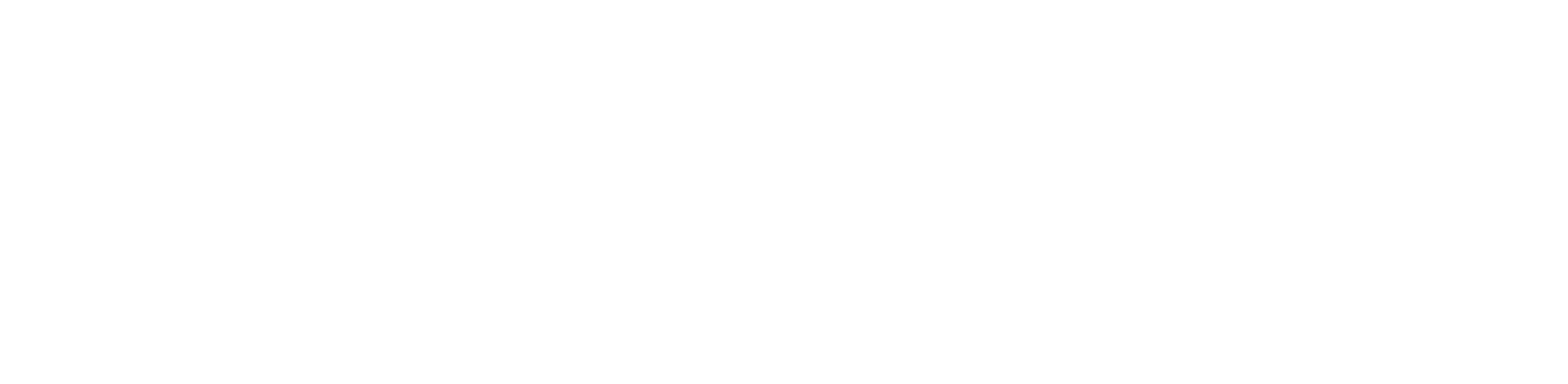
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



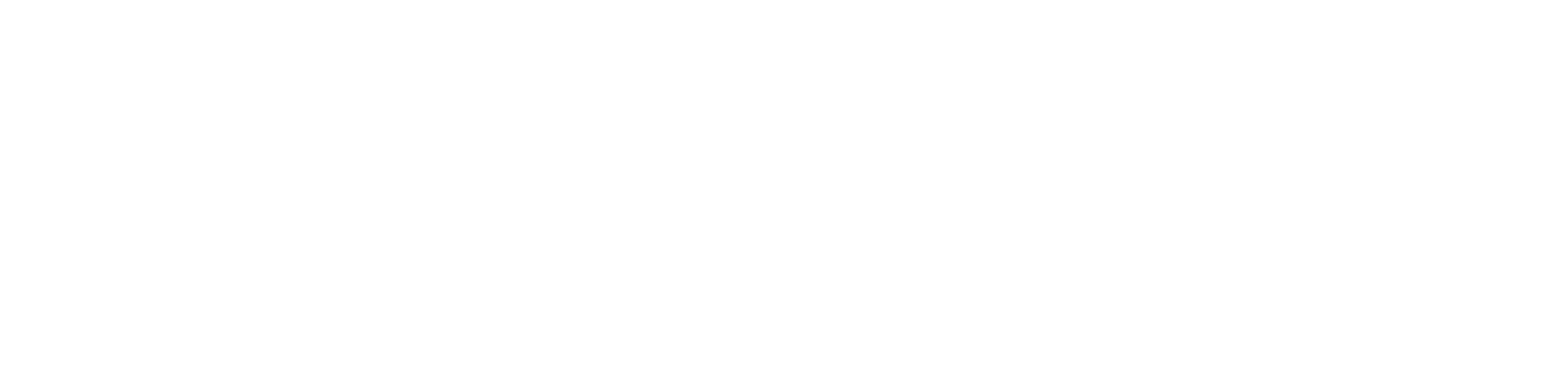
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



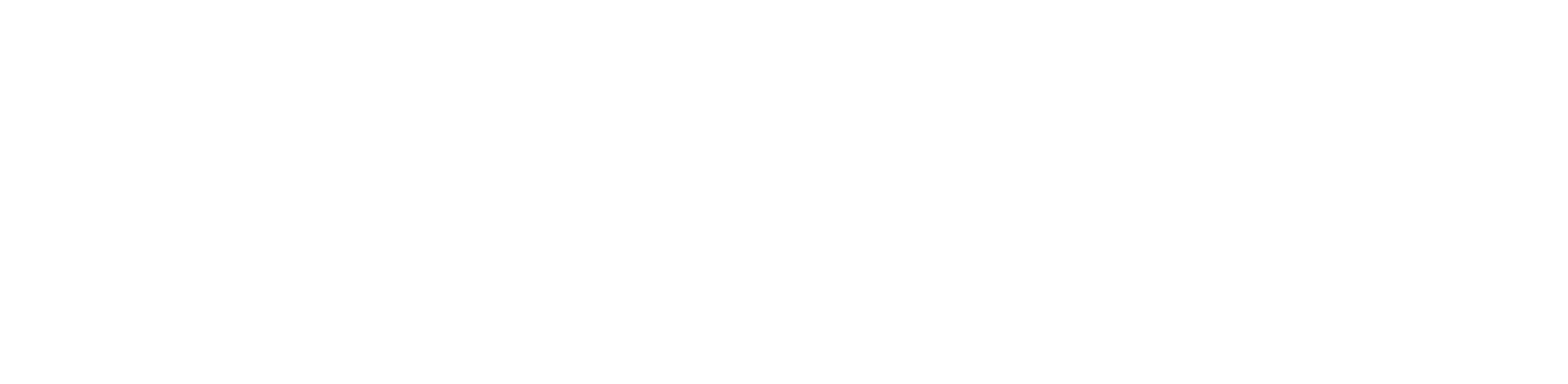
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

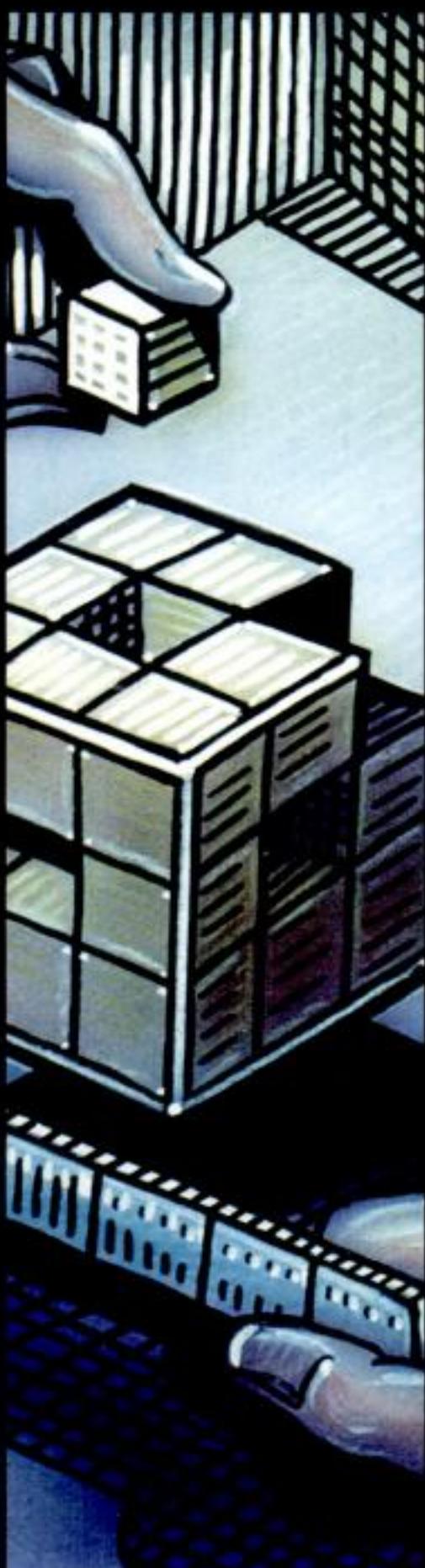


You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.



You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

DATABASE MANAGEMENT SYSTEMS



This book provides simple and comprehensive explanation of fundamentals of database management systems. It focuses on building database applications by emphasizing on concepts that are the foundation of database processing. This book is intended to be a complete text for undergraduate and graduate level database management courses offered across a range of academic disciplines such as computer science, information systems, and business management.

Highlights

- Concepts like Relational Model, ER Model, etc., elucidated through a running example of a cricket database, especially formulated for Indian students
- In-depth coverage of Transaction Management and Concurrency, Query Processing, Distributed Databases, and Backup and Recovery
- Discussion of new technologies like Mobile Databases and Cloud Computing
- Student friendly example (cricket database) running through all the chapters
- Strong pedagogical features:
 - ◆ 342 Review Questions along with Section References
 - ◆ 348 Short Answer Questions
 - ◆ 409 Multiple Choice Questions with Answers
 - ◆ 246 Exercises
 - ◆ 56 Lab Assignments
 - ◆ 43 Projects

The online learning centre for this book can be accessed at
<http://www.mhhe.com/gupta/dbms>

Visit us at : www.tatamcgrawhill.com

The McGraw-Hill Companies

ISBN-13: 978-0-07-107273-1

ISBN-10: 0-07-107273-X



9 780071 072731



Higher Education

Copyrighted material