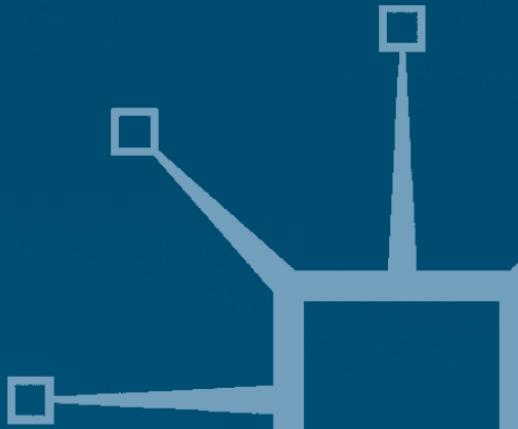


Database Systems

Third Edition

Paul Beynon-Davies



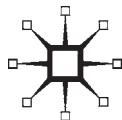
DATABASE SYSTEMS
THIRD EDITION

DATABASE SYSTEMS

THIRD EDITION

Paul Beynon-Davies

palgrave
macmillan



© Paul Beynon-Davies 1996, 2000, 2004

All rights reserved. No reproduction, copy or transmission of this publication may be made without written permission.

No paragraph of this publication may be reproduced, copied or transmitted save with written permission or in accordance with the provisions of the Copyright, Designs and Patents Act 1988, or under the terms of any licence permitting limited copying issued by the Copyright Licensing Agency, 90 Tottenham Court Road, London W1T 4LP.

Any person who does any unauthorised act in relation to this publication may be liable to criminal prosecution and civil claims for damages.

The author has asserted his right to be identified as the author of this work in accordance with the Copyright, Designs and Patents Act 1988.

First edition 1996

Second edition 2000

Reprinted twice

Third edition 2004

Published by

PALGRAVE MACMILLAN

Hounds Mills, Basingstoke, Hampshire RG21 6XS and

175 Fifth Avenue, New York, N.Y. 10010

Companies and representatives throughout the world

PALGRAVE MACMILLAN is the global academic imprint of the Palgrave Macmillan division of St. Martin's Press, LLC and of Palgrave Macmillan Ltd. Macmillan® is a registered trademark in the United States, United Kingdom and other countries. Palgrave is a registered trademark in the European Union and other countries.

ISBN 1-4039-1601-2

This book is printed on paper suitable for recycling and made from fully managed and sustained forest sources.

A catalogue record for this book is available from the British Library.

Typeset by Cambrian Typesetters, Frimley, Camberley, Surrey

10 9 8 7 6 5 4 3 2 1
13 12 11 10 09 08 07 06 05 04

Printed and bound in China



CONTENTS

PREFACE TO THE THIRD EDITION	ix
PART 1 FUNDAMENTALS	1
1 Database Systems as Abstract Machines	3
2 Data and Information	19
3 Database, DBMS and Data Model	30
4 Database Systems, ICT Systems and Information Systems	48
5 Database Systems and Electronic Business	62
6 Data Management Layer	76
PART 2 DATA MODELS	87
7 Relational Data Model	89
8 Object-Oriented Data Model	113
9 Deductive Data Model	125
10 Post-Relational Data Model	143
PART 3 DATABASE MANAGEMENT SYSTEMS (DBMS) – INTERFACE	155
11 SQL – Data Definition	157
12 SQL – Data Integrity	168
13 SQL – Data Manipulation	177
PART 4 DATABASE DEVELOPMENT	193
14 Database Development Process	195
15 Requirements Elicitation	208

16	Entity–Relationship Diagramming	219
17	Object Modelling	246
18	Normalisation	269
19	Physical Database Design	292
20	Database Implementation	309
PART 5	PLANNING AND ADMINISTRATION OF DATABASE SYSTEMS	321
21	Strategic Data Planning	323
22	Data Administration	334
23	Database Administration	343
PART 6	DATABASE MANAGEMENT SYSTEMS (DBMS) – TOOLKIT	355
24	DBMS-Toolkit – End-User Tools	357
25	DBMS-Toolkit – Application Development Tools	365
26	DBMS-Toolkit – Database Administration Tools	375
PART 7	DATABASE MANAGEMENT SYSTEMS – KERNEL	381
27	Data Organisation	383
28	Access Mechanisms	395
29	Transaction Management	403
30	Other Kernel Functions	418
PART 8	DATABASE MANAGEMENT SYSTEMS – STANDARDS AND COMMERCIAL SYSTEMS	425
31	Post-Relational DBMS – SQL3	427
32	Object-Oriented DBMS – ODMG Object Model	436
33	Microsoft Access	446
34	ORACLE	457
35	O ₂ DBMS	467
PART 9	TRENDS IN DATABASE TECHNOLOGY	475
36	Distributed Processing	477
37	Distributed Data	486
38	Parallel Databases	498
39	Complex Data	512

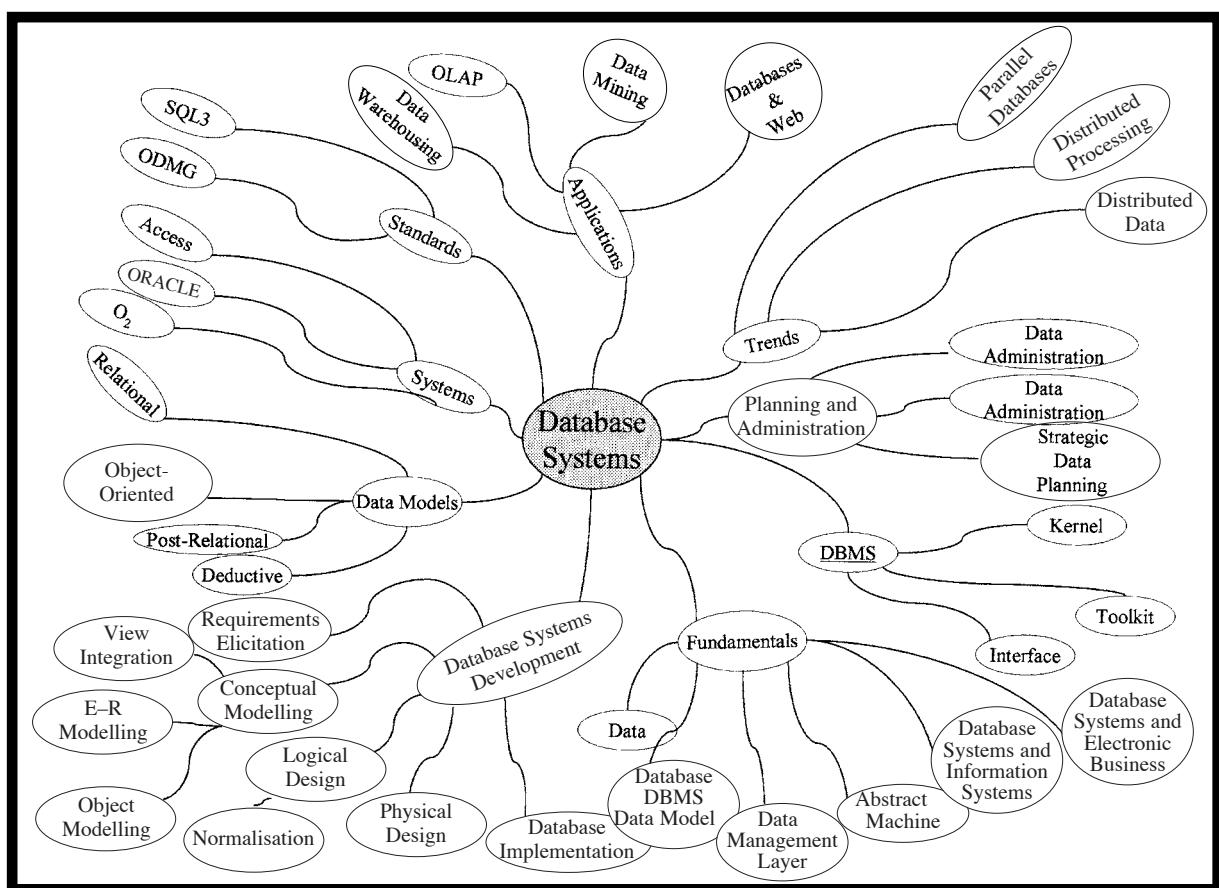
PART 10 APPLICATIONS OF DATABASE SYSTEMS	525
40 Data Warehousing	527
41 On-Line Analytical Processing	539
42 Data Mining	547
43 Databases and the Web	554
BIBLIOGRAPHY	567
GLOSSARY AND INDEX	572



PREFACE TO THE THIRD EDITION

Perhaps the most valuable result of all education is the ability to make yourself do the thing you have to do, when it ought to be done, whether you like it or not.

Thomas H. Huxley (1825–95)



 MISSION

The main aim of this work is to provide one readable text of essential core material for further education, higher education and commercial courses on database systems. The current volume is designed to form a consolidated, introductory text on modern database technology and the development of database systems.

It is undoubtedly true that database systems hold a prominent place in most contemporary approaches to the development of information systems. It is this practical emphasis on the use of database systems for information systems work that distinguishes the current volume from other texts on the subject. This work therefore forms a companion volume to *Information Systems: An Introduction to Informatics in Organisations* (Beynon-Davies, 2002) and to *e-Business* (Beynon-Davies, 2003), also published by Palgrave Macmillan. The current version has been revised to reflect the successful style and organisation of the *Information Systems* text.

The text is built from the author's experiences of consultancy in this area, as well as in running a number of academic and commercial courses on database technology and database development for several years.

 CHANGES TO THE THIRD EDITION

Two types of changes have been made to the third edition: presentational and content. In terms of presentation, the structure of the chapters has been reorganised and the parts resequenced to make for a better educational experience. In terms of content, the third edition has been expanded considerably. This expansion has enabled particular topics to be covered in more depth:

- The part on database system fundamentals has been extended with a number of chapters, particularly with a chapter defining the concept of data and a chapter considering the important place of database systems in the informatics infrastructure for electronic business
- A new part has been added which extends the discussion on the SQL database sublanguage
- Chapters on the ORACLE DBMS and the Microsoft Access DBMS have been updated
- New chapters have been included on the database development process and requirements elicitation
- The part on trends has been reorganised around four chapters: distributed processing, distributed data, parallelism and complex data. This latter chapter now includes coverage of semi-structured data and XML
- The chapter on database systems and the Web has been updated



Many chapters have been updated with developments since 2000, and the structure of a number of chapters changed to provide a more coherent presentation of key topics. The sequencing of parts has been particularly changed in response to requests for a more natural flow of topics. All chapters have been revised to align with the successful style of *Information Systems*, including examples, Case Studies and spider diagrams (see below).



PARTS

The text is organised into a number of parts:

- **Part 1: Fundamentals.** The introductory chapters set the scene for the core of the text. We begin with a description of the key features of a database system. We then move on to define some key concepts and illustrate the importance of database technology to contemporary information systems and electronic business. We close this part with an example of some of the key features of the data management layer of an information and communications technology (ICT) system
- **Part 2: Data Models.** Part 2 explores a number of contemporary architectures for database systems. Because of its current dominance, particular emphasis is placed on the relational data model, although developments in object-oriented data management explain the extended coverage offered. We also consider another data model that is significant because of its association with ‘intelligent’ applications – the deductive data model. We close this part with a consideration of a hybrid data model – the extended-relational, object-relational or post-relational data model – which is important because of its popularity among contemporary DBMS
- **Part 3: DBMS Interface.** Part 3 addresses the issue of the standard interface to DBMS – structure query language (SQL). We consider how SQL may be used to declare database schemas, maintain data in such schemas and implement integrity for such schemas
- **Part 4: Database Development.** Part 4 represents a discussion of the major elements of database systems development. We begin with a description of the development process and the toolkit needed by the database developer. This is followed by a description of the features of requirements elicitation. In terms of conceptual modelling we discuss the techniques of entity–relationship diagramming and object modelling. In terms of logical modelling we discuss the technique of normalisation. We conclude with a review of a number of issues involved in the physical design of databases and database implementation
- **Part 5: Planning and Administration of Database Systems.** The issue of planning and administering data in organisations is considered in this part. We first consider the issue of using corporate data models to plan for database systems. This is followed by a consideration of the importance of administering data



within organisations. We conclude with a review of the database administration function

- *Part 6: DBMS – Toolkit.* In this part we consider the toolkit of products that can be used in association with DBMS. The toolkit is divided into end-user tools, database administration tools and application development tools
- *Part 7: DBMS – Kernel.* This part considers some of the critical functions of the kernel of a database management system (DBMS): data organisation, access mechanisms, query management, transaction management, database security and database recovery
- *Part 8: DBMS – Standards and Commercial Systems.* Part 7 discusses the architectures and facilities available in three contemporary DBMS – one relational (Microsoft Access), one post-relational (ORACLE) and one object-oriented (O_2). We also consider developments among the SQL3 and ODMG standards for DBMS
- *Part 9: Trends in Database Technology.* This part considers some of the developing infrastructure for databases. It consists of four chapters which look at areas having a significant effect on the functionality of database systems: the issues of distributing data and processing, parallelism and handling semi-structured data
- *Part 10: Applications of Database Systems.* This part discusses four contemporary applications for database systems: data warehousing, OLAP, data mining, and databases and the Web



PRESENTATION

The presentation of the book is deliberately designed to be used within some structured learning activity. Each part of the book begins with a section summarising the key elements of the database systems domain considered. Each chapter in the book is designed as a learning unit and aims at imparting a number of key concepts in the area of database systems. The structure of each chapter consists of:

- A set of learning outcomes
- A series of sections. Each section provides a description of a key concept and an example of the concept, where relevant
- A Case Study is presented at the end of most chapters and is used to illustrate some real-world expression of the concepts discussed
- A summary of key points
- A set of activities for further consideration. Many activities are designed for readers to attempt to apply a concept discussed in the chapter to some of their own experience or understanding
- A set of references

Each part of the book and chapter now includes a spider diagram. Spider diagrams were originally developed by Tony Buzan (1982) as an effective method of note-taking within study which exploits the natural ability of humans to associate. The idea of a spider diagram is simply to relate concepts together using free-form lines. The centre of the diagram is used to locate the orienting concept. Hence the spider diagram at the start of this Preface details the structure of the entire book. Associated concepts are drawn radiating outwards. Any concept on a spider diagram may act as an orienting concept on its own spider diagram. In this way, most of the detailed structure of the book can be observed in the set of spider diagrams.

Spider diagrams are primarily used as a method of summarising the organisation of information contained in this work. However, we hope that the student of the area will also find them useful as a revision aid, and the lecturer or instructor will find them useful as a way of summarising key elements of the domain.



WEB-SITE AND FEEDBACK

A Web-site, including a teaching pack for lecturers/instructors, has been produced to accompany the book and can be accessed at

www.palgrave.com/resources

The author is keen to receive feedback on the current text. Any comments or suggestions should be addressed to the author (p.beynon-davies@swansea.ac.uk).



SUGGESTED ROUTES THROUGH THE MATERIAL

Below we indicate some ways in which the material may be used in educational modules and courses. These are meant to be purely suggestions.

INTRODUCTORY MODULE IN DATABASE SYSTEMS

An introductory module in database systems should provide a basic understanding of database concepts and an appreciation of database development issues. For this purpose the following chapters are suggested: *four chapters in fundamentals; relational data model; SQL; access; normalisation; E-R diagramming*.

INTERMEDIATE MODULE IN DATABASE SYSTEMS

An intermediate module should impart an understanding of the architecture of a DBMS and issues pertaining to physical design and database administration. For this purpose the following chapters are suggested: *kernel chapters; toolkit chapters; ORACLE; database administration; physical design; database implementation*.



ADVANCED MODULE IN DATABASE SYSTEMS

Advanced modules in database systems should provide an awareness of alternative approaches to handling data and developing database systems over and above that provided by the relational data model and relational databases. For this purpose the following chapters are suggested: *OO data model; deductive data model; post-relational data model; OODBMS; object modelling; strategic data planning; data administration; trends.*



ACKNOWLEDGEMENTS

My thanks to Rebecca Mashayekh at Palgrave Macmillan for encouragement and support in producing this book. My thanks also to a number of anonymous reviewers for their helpful suggestions.



TRADEMARK NOTICE

AccessTM and SQL ServerTM are trademarks of Microsoft

O₂TM is a trademark of O₂ Technology

ORACLETM is a trademark of the Oracle Corporation



REFERENCES

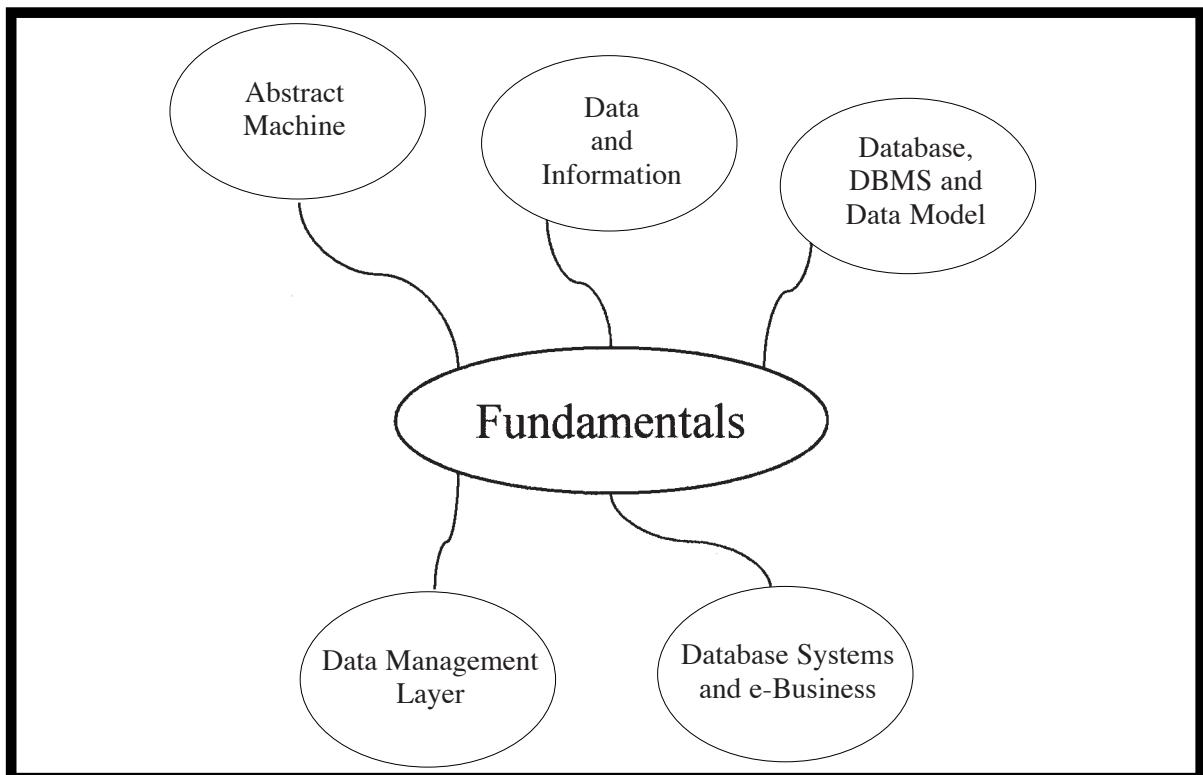
- Beynon-Davies, P. (2002). *Information Systems: An Introduction to Informatics in Organisations*. Basingstoke, Palgrave (formerly Macmillan).
- Beynon-Davies, P. (2003). *e-Business*. Basingstoke, Palgrave Macmillan.



FUNDAMENTALS

Learn the fundamentals of the game and stick to them. Band-Aid remedies never last.

Jack Nicklaus (1940–)



This part introduces the fundamental material on database systems and sets it in the context of information system applications. We begin with a description of the key features of a database system, considering it in terms of an abstract machine (Chapter 1). We then move on to define the concept of data using ideas from the discipline of semiotics (Chapter 2). Chapter 3 makes the critical distinction between a database, a database management system and a data model. Chapter 4 illustrates the importance of database technology to contemporary information systems and ICT systems. This sets the scene for a consideration of the role of database systems within the modern phenomenon of electronic business (Chapter 5). We conclude with an examination of the data management layer of an ICT system (Chapter 6).

CHAPTER 1

DATABASE SYSTEMS AS ABSTRACT MACHINES

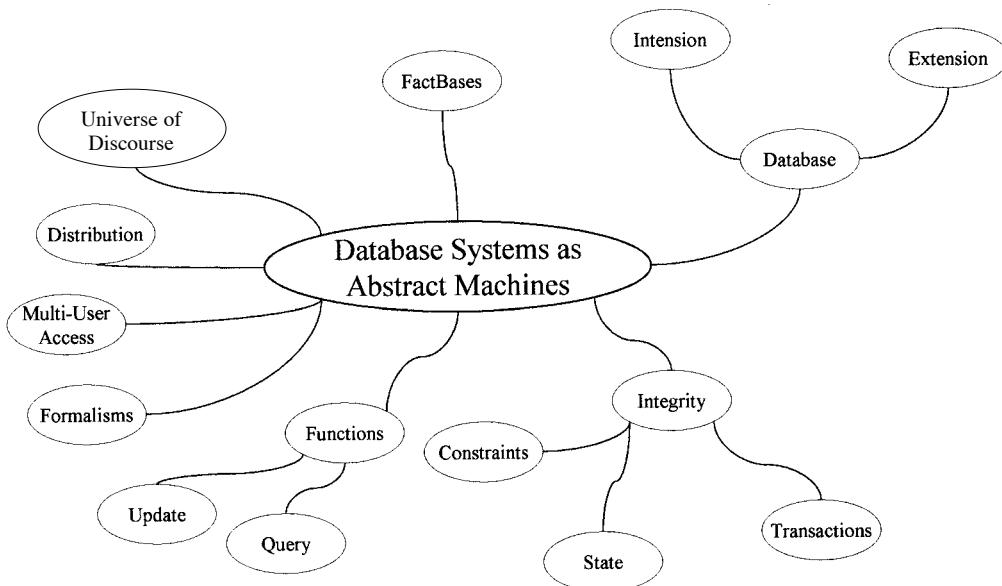
Machines take me by surprise with great frequency.

Alan Turing (1912–54)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define the idea of a universe of discourse
- Distinguish between the intension and extension of a database
- Define the concept of data integrity and integrity constraints
- Discuss the issues of transactions, states and update functions
- Relate the idea of a data model
- Introduce the multi-user and distributed nature of contemporary database systems



1.1 INTRODUCTION

In this chapter we discuss in an informal manner the idea of a database as an abstract machine. An abstract machine is a model of the key features of some system without any detail of implementation. The objective of this chapter is to describe the major components of a database system without introducing any formal notation, or introducing any concepts of representation, development or implementation. All of the concepts introduced in this chapter are elaborated upon in further chapters.

Most modern-day organisations have a need to record data relevant to their everyday activities. Many modern-day organisations choose to organise and store some of this data in an electronic database.

Example ➔ Take for instance a university. Most universities need to record data to help in the activities of teaching and learning. Most universities need to record, among other things:

- what students and lecturers they have
- what courses and modules they are running
- which lecturers are teaching which modules
- which students are taking which modules
- which lecturer is assessing against which module
- which students have been assessed in which modules

Various members of staff at a university will be entering data such as this into a database system. For instance, data relevant to courses and modules may be entered by administrators in academic departments, course leaders may enter data pertaining to lecturers, and data relevant to students, particularly their enrolments on courses and modules, may be entered by staff at a central registry.

Once the data is entered into the database it may be utilised in a variety of ways. For example, a complete and accurate list of enrolled students may be used to generate membership records for the learning resources centre, it may be used as a claim to educational authorities for student income or as an input into a timetabling system which might attempt to optimise room utilisation across a university campus.

In this chapter we will unpack the fundamental elements of a database system using this example of an academic database to illustrate concepts.

1.2 (c) UNIVERSE OF DISCOURSE

A database is a model of some aspect of the reality of an organisation (Kent, 1978). It is conventional to call this reality a *universe of discourse* (UoD), or sometimes a domain of discourse. A UoD will be made up of classes and relationships between classes. The classes in a UoD will be defined in terms of their properties or attributes.

Example → Consider an academic setting such as a university. The UoD in this case might encompass, among other things, modules offered to students and students taking modules.

Modules and students are examples of things of interest to us in a university. We call such things of interest classes or entities. We are particularly interested in the phenomenon that students take offered modules. This facet of the UoD would be regarded as a relationship between classes.

A class or entity such as module or student is normally defined as such because we wish to record some information about occurrences of the class. In other words, classes have properties or attributes. For instance, students have names, addresses and telephone numbers; modules have titles and credit points.

A database of whatever form, electronic or otherwise, must be designed. The process of database design is the activity of representing classes, attributes and relationships in a database. This process is illustrated in Figure 1.1.

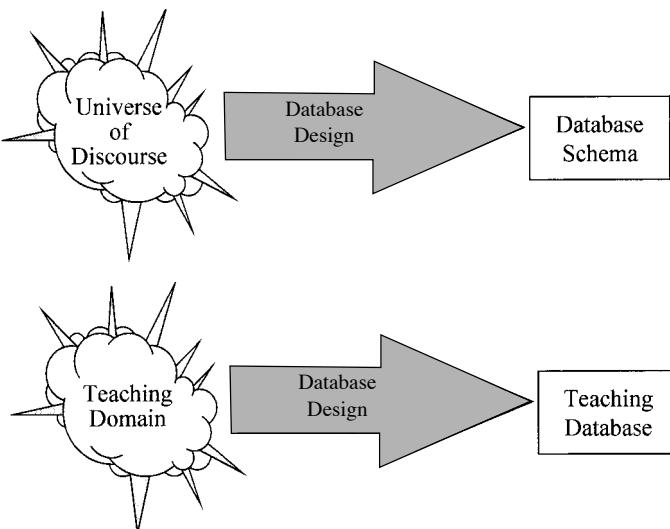


Figure 1.1 Database design.

1.2.1 FACTBASES

A database can be considered as an organised collection of data which is meant to represent some UoD. Data are facts. A datum, a unit of data, is one symbol or a collection of symbols that is used to represent something. Facts by themselves are meaningless. To prove useful they must be interpreted. Information is interpreted data. Information is data placed within a meaningful context. Information is data with an assigned semantics or meaning.

Example → Consider the string of symbols 43. Taken together these symbols form a datum. Taken together however they are meaningless. To turn them into information we have to supply a meaningful context. We have to interpret them. This might be that they constitute a student number, a student's age, or the number of students taking a module. Information of this sort will contribute to our knowledge of a particular domain, in this case university administration. It might add, for instance, to our understanding of the total number of students taking the modules in a particular university department or school.

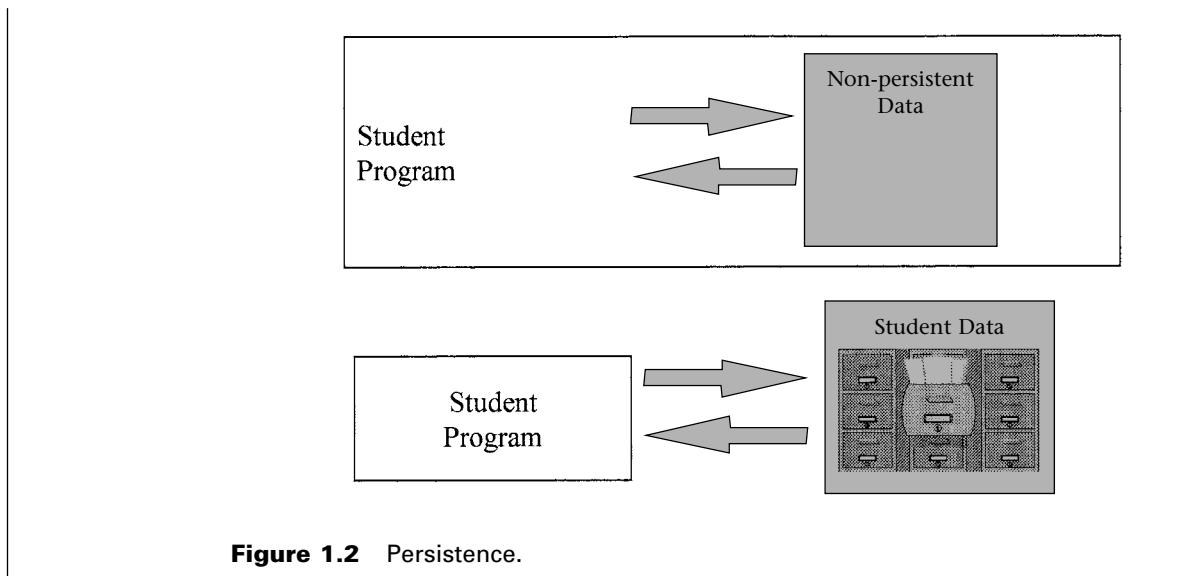
A database can be considered as a collection of facts or positive assertions about a UoD, such as *relational database design is a module* and *John Davies takes relational database design*. Usually negative facts, such as what modules are not taken by a student, are not stored. Hence, databases constitute 'closed worlds' in which only what is explicitly represented is regarded as being true.

A database is said to be in a given state at a given time. A state denotes the entire collection of facts that are true at a given instant in time. A database system can therefore be considered as a factbase which changes through time.

1.2.2 PERSISTENCE

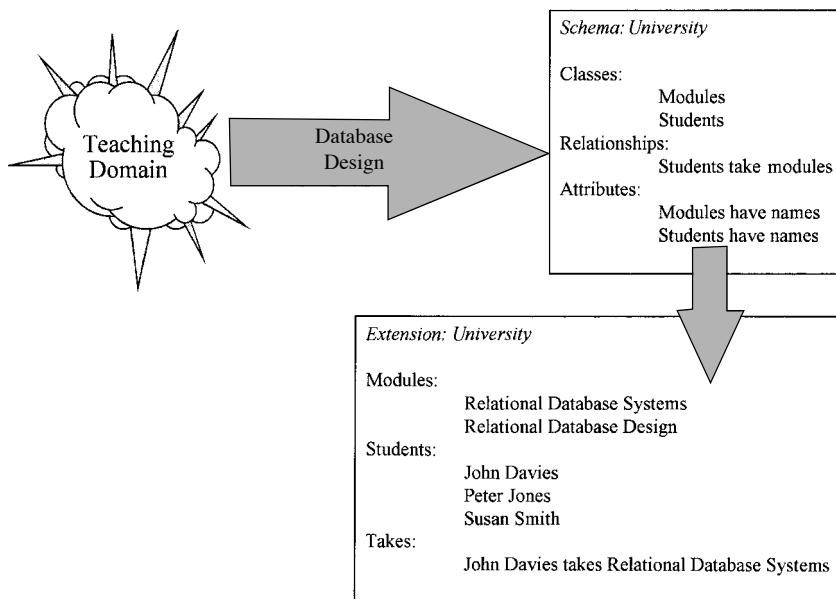
Data in a database is described as being persistent. By persistent we mean that the data is held for some duration. The duration may not actually be very long. The term persistence is used to distinguish more permanent data from data which is more transient in nature. Hence, product data, account data, patient data and student data would all normally be regarded as examples of persistent data. In contrast, data input at a personal computer, held for manipulation within a program, or printed out on a report, would not be regarded as persistent, as once it has been used it is no longer required.

Example → In Figure 1.2 a program for manipulating student data will typically store data in terms of variables. The data in such 'placeholders' will normally be lost on program shutdown. Hence, this is an example of non-persistent data. In contrast, consider a program which reads and writes data to and from an external data file. The data in such a file persists beyond the running of the program.

**Figure 1.2** Persistence.

1.2.3 INTENSIONAL AND EXTENSIONAL PARTS

A database is made up of two parts: an intensional part and an extensional part. The intension of a database is a set of definitions which describe the structure or organisation of a given database. The extension of a database is the total set of data in the database. The intension of a database is also referred to as its schema. The activity of developing a schema for a database system is referred to as database design. This is illustrated in Figure 1.3.

**Figure 1.3** Intension and extension.

Example → Below we define, in an informal manner, the schema relevant to a university database:

Schema: University

Classes:

Modules – courses run by the institution in an academic semester

Students – people taking modules at the institution

Relationships:

Students take modules

Attributes:

Modules have names

Students have names

In the schema we have identified classes of things such as *modules*, relationships between classes such as *students take modules*, and properties of classes such as *modules have names*. The process of developing such definitions which form the key elements of the schema of a database is considered in detail in Chapters 16 and 17.

A brief extension for the academic database might be:

Extension: University

Modules:

Relational Database Systems

Relational Database Design

Students:

John Davies was born on 1 July 1985

Peter Jones was born on 1 January 1985

Susan Smith was born on 1 April 1986

Takes:

John Davies takes Relational Database Systems

Here we have provided specific instances of classes such as *relational database systems*, of relationships such as *John Davies takes relational database systems* and of attributes such as *John Davies was born on 1 July 1985*. In fact, we have also implicitly defined some instances of attributes for the class modules. We have provided a list of two so-called identifiers for the instances of the class modules, in this case *relational database systems* and *relational database design*. Identifiers are special attributes which serve to uniquely identify instances of classes.

1.3 INTEGRITY

When we say that a database displays integrity we mean that it is an accurate reflection of its UoD. The process of ensuring integrity is a major feature of modern information systems. The process of designing for integrity is a much neglected aspect of database development.

Example  In terms of our academic database, integrity means ensuring that our database provides accurate responses to questions such as: how many students are currently enrolled on the module Relational Database Systems?

Integrity is an important issue because most databases are designed, once in use, to change. In other words, the data in a database will change over a period of time. If a database does not change, i.e. it is only used for reference purposes, then integrity is not an issue of concern.

It is useful to think of database change as occurring in discrete rather than continuous time. In this sense, we may conceive of a database as undergoing a number of state-changes, caused by external and internal events. Of the set of possible future states feasible for a database some of these states are valid and some are invalid. Each valid state forms the extension of the database at that moment in time. Integrity is the process of ensuring that a database travels through a space defined by valid states.

Example  Hence, the extension of the database in section 1.2.3 represents a state. Integrity involves determining whether a transition to the state below is a valid one. That is, integrity involves answering questions such as: is it valid to add another fact to our database which relates John Davies to Relational Database Design?

Extension: University

Modules:

Relational Database Systems

Relational Database Design

Students:

John Davies

Peter Jones

Susan Smith

Takes:

John Davies takes Relational Database Systems

John Davies takes Relational Database Design



In this case, it probably would be a valid transition because this relationship did not exist in the previous state of the database.

1.3.1 REDUNDANCY

Consider another transition. We wish to add the assertion *John Davies takes Relational Database Systems* to our extension. This is not a valid transition. As we shall see, databases are designed to minimise redundant storage of data. In a database we attempt to store only one item of data about objects or relationships between objects in our UoD. Ideally, a database should be a repository with no redundant facts. Clearly adding this assertion to our extension will replicate the relationship between John Davies and Relational Database Systems.

1.3.2 TRANSACTIONS

The events that cause a change of state are called transactions in database terms. A transaction changes a database from one state to another. A new state is brought into being by asserting the facts that become true and/or denying the facts that cease to be true. Hence, in our example we might want to enrol Peter Jones in the module relational database design. This is an example of a transaction.

Transactions of a similar form are called transaction types.

Example → A transaction type that might be relevant to the university database system might be:

Enrol Student in Module

1.3.3 INTEGRITY CONSTRAINTS

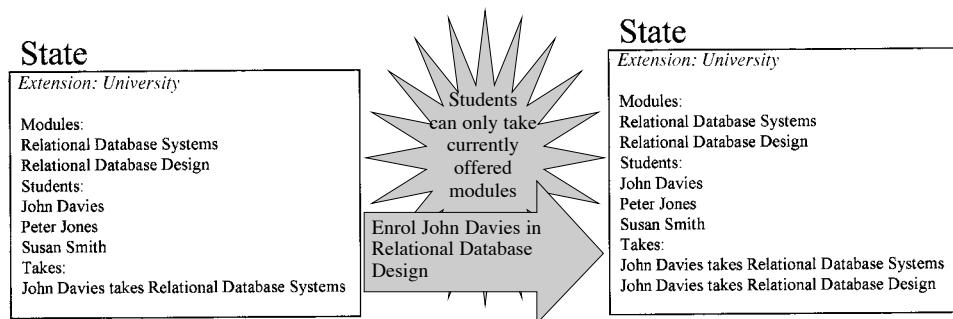
Database integrity is ensured through integrity constraints. An integrity constraint is a rule which establishes how a database is to remain an accurate reflection of its UoD.

Constraints may be divided into two major types: static constraints and transition constraints (see Figure 1.4). A static constraint, sometimes known as a 'state invariant', is used to check that an incoming transaction will not change a database into an invalid state. A static constraint is a restriction defined on states.

Example → An example of a static constraint relevant to our UoD might be: *students can only take currently offered modules*. This static constraint would prevent us from entering the following fact into our current database.

John Davies takes Deductive Database Systems

since Deductive Database Systems is not a currently offered module.

**Figure 1.4** States, transactions and constraints.

In contrast, a transition constraint is a rule that relates given states of a database. A transition is a state transformation and can therefore be denoted by a pair of states. A transition constraint is a restriction defined on a transition.

Example → An example of a transition constraint might be: *the number of modules taken by a student must not drop to zero during a semester*. Hence, if we wished to remove a fact relating John Davies to a particular module from our database we should first check that this would not cause an invalid transition.

1.4 DATABASE FUNCTIONS

Most data held in a database is there to fulfil some organisational need. To perform useful activity with a database we need two types of function: update and query functions. Update functions cause changes to data. Query functions extract data.

1.4.1 UPDATE FUNCTIONS

Transactions may initiate update functions. Update functions change a database from one state to another.

Example → Update functions that might be relevant to the university database might be:

Update Functions:

- Initiate Semester
- Offer module
- Cancel module
- Enrol student on course



Enrol student in module

Transfer student between modules

We expand on the update function *Transfer student between modules* in the next section.

1.4.2 FUNCTIONS AND CONSTRAINTS

The idea of an update function and an integrity constraint are interrelated. An update function will usually have a series of conditions associated with it. Such conditions represent integrity constraints. There will also be a series of actions associated with the update function. These will specify what should happen if the conditions are true.

Example ➔ We might specify one of the update functions above as follows:

Update Function: Transfer student X from module 1 to module 2

Conditions:

Student X takes module 1

Student X does not take module 2

Module 2 is offered

Actions:

Deny that student X takes module 1

Assert that student X takes module 2

Here, X, module 1 and module 2 are placeholders for values. Hence, we might initiate a transaction using this update function by assigning the value John Davies to X, Relational Database Systems to module 1 and Relational Database Design to module 2. In this case, of course, the transaction will fail because the second condition of the update function is false in this case. This is because student (John Davies) does take (Relational Database Design) currently.

The action of a given update function may cause the conditions of another update function to become true. In this case, the latter update function can be said to have 'triggered' a state-change. In this way a whole chain of 'internal' transactions may be propagated from an 'external' transaction.

Example → Suppose we specified a cancel module update function in the following way:

Update Function: Cancel Module 1

Conditions:

Module 1 exists

Module 1 roll < 10

Actions:

Deny that module 1 exists

Also suppose that a transaction generated by the transfer student function succeeds in reducing the roll of module 1 to 9 students. Here, of course, both conditions of the cancel module function are satisfied and a transaction will be initiated to remove the fact that the module exists.

1.4.3 QUERY FUNCTIONS

The other major type of function used against a database is the query function. This does not modify the database in any way, but is used primarily to check whether a fact or group of facts holds in a given database state. As we shall see, query functions can also be used to derive data from established facts.

Example → Query functions that might be relevant for the current UoD might be:

Is course X being offered?

Is student X taking course Y?

The simplest form of query function is inherently associated with a given fact. We use such a function to check whether a given fact holds or not in our database.

Example → The query:

Does John Davies take relational database design?

would return true or yes. Whereas the query:

Does Peter Jones take relational database systems?

would return false or no.



However, most useful database queries will return a range or set of values, rather than true or false.

Example Administrators in a university might want answers to the following questions:

Which students take relational database design?

What modules are we presently offering?

In this sense the words *students* and *modules* in the queries above would be place-holders for values returned from the database.

1.5

FORMALISMS

Every database system must use some representation formalism. Patrick Henry Winston has defined a representation formalism as being '*a set of syntactic and semantic conventions that make it possible to describe things*' (Winston, 1984). The syntax of a representation specifies a set of rules for combining symbols and arrangements of symbols to form statements in the representation formalism. The semantics of a representation specify how such statements should be interpreted; that is, how we can derive meaning from them.

In database terms the idea of a representation formalism corresponds with the concept of a data model. A data model provides for database developers a set of principles by which they can construct a database system.

Part 2 is devoted to considering a number of different data models. This is undertaken because each formalism or data model has its strengths and weaknesses. Hence, the relational data model offers an extremely flexible way of representing facts but does not offer a convenient way of representing the intricacies of update functions. In contrast, object-oriented databases offer a powerful notation for representing update functions in database systems, but currently suffer from a lack of uniformity among implementations.

1.6

MULTI-USER ACCESS

A database can be used solely by one person or one application system. In terms of organisations however, many databases are used by multiple users. Sharing of common data is a key characteristic of most database systems. In such situations we speak of a multi-user database system.

By definition, some way must exist in a multi-user database of handling situations where a number of persons or application systems want to access the

same data at effectively the same time. This is known as the issue of concurrency.

Example  Consider a situation where one user is enrolling student John Davies on module Relational Database Systems. At the same time another user is removing the module Relational Database Systems from the current offering. Clearly, in the time it takes the first user to enter an enrolment fact, the second user could have denied the module fact. The database is left in an inconsistent state.

In any database system some system must therefore be provided to resolve such conflicts of concurrency. This issue is discussed in Chapter 29.

1.6.1 VIEWS

Part of the reason that data in databases is shared is that a database may be used for different purposes within one organisation. For instance, the academic database described in this chapter might be used for various purposes in a university such as recording student grades or timetabling classes. Each distinct user group may demand a particular subset of the database in terms of the data it needs to perform its work. Hence administrators in academic departments will be interested in items like student names and grades while a timetabler will be interested in rooms and times. This subset of data is known as a view.

In practice, a view is merely a query function that is packaged for use by a particular user group or program. It provides a particular window into a database and is discussed in detail in Chapter 23.

1.7 DISTRIBUTION

In contemporary database systems the extension and the intension of the database may be distributed across numerous different sites. In a distributed database system one effective schema is stored in a series of separate fragments and/or copies (replicates) (see Figure 1.5).

Example  In an academic domain data about students may be held as fragments relevant to each school or department in a university. A replicate of some or all of this data may be held for central administration purposes.

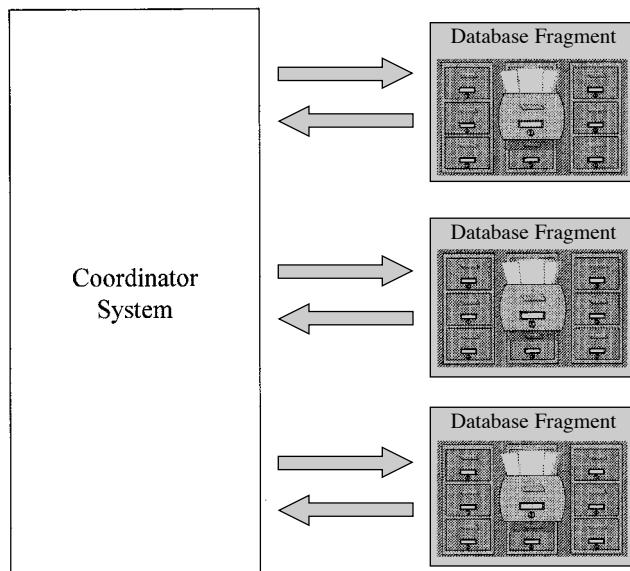


Figure 1.5 Distributing data.

1.8

CASE STUDY: SCHEMA FOR A UNIVERSITY DATABASE

It is possible to provide a more extensive schema for a university database system as follows:

Schema: University

Classes:

- Modules – courses run by the institution in an academic semester
- Students – people taking modules at the institution
- Lecturers – academic staff at the university

Relationships:

- Students enrol on courses
- Students enrol on modules
- Students are assessed on modules
- Lecturers teach modules
- Modules have prerequisite modules

Attributes:

- Modules have identifiers, names and levels
- Students have identifiers, names, addresses and telephone numbers
- Lecturers have identifiers, names, statuses and salaries

Update Functions:

Initiate Semester

Offer module, Cancel module, Create module prerequisite

Enrol student on course, Enrol student in module

Transfer student between modules, Transfer student between courses

Create lecturer, Assign lecturer to Module

Update Assessment

This schema is revisited in Part 2 when we consider the way in which a number of data models can be used to represent the intension described in this chapter.

1.9  SUMMARY

In this chapter we have discussed in a relatively informal manner the major components of a database system. We summarise the discussion below:

- A database is a model of some universe of discourse (UoD)
- A database consists of two parts: an intensional part and an extensional part
- The intensional part (schema) of a database defines the structure of the extensional part
- The process of developing the schema for some database is database design
- The extensional part of a database consists of a collection of positive assertions
- Integrity is the problem of ensuring that a database remains an accurate reflection of its UoD
- Integrity is ensured through the application of integrity constraints
- There are two types of integrity constraint: static and transition constraints
- A static constraint is a restriction defined on a database state
- A transition constraint is a rule that relates given states of a database
- To perform useful activity with a database two types of function are needed: update and query functions
- Update functions cause changes to the state of a database
- Query functions are mainly used to check whether a fact or group of facts holds in a given database state
- To implement the elements of a database system we need some formalism (data model)



- Data normally needs to be shared between numerous different users. A database system therefore needs to provide concurrency. Sharing data for different uses requires the concept of a view
- Data is frequently distributed within a database system

1.10

ACTIVITIES

1. Define a UoD in informal terms for some organisation known to you.
2. Declare a brief schema for the UoD of an organisation known to you.
3. Think of an example of a database where integrity is not a major issue.
4. Assume the initial state of the database is as follows:

State 1:

Relational Database Design is a module

Relational Database Systems is a module

John Davies is a student

John Davies takes Relational Database Design

Is the transition to state 2 below a valid one given the constraint defined above in section 1.3.3?

State 2:

Relational Database Design is a module

Relational Database Systems is a module

John Davies takes Relational Database Design

John Davies takes Object-Oriented Database Systems

Assume we add facts about lecturers to our database. What update functions might be relevant?

5. What constraints might we attach to the update function offer module?
6. Write an enrol student on module function in terms of conditions and actions.
7. Assuming the entity or class Lecturer has been added to the schema, define a useful query of relevance.

1.11

REFERENCES

- Kent, W. (1978). *Data and Reality*. Amsterdam, North-Holland.
Winston, P.H. (1984). *Artificial Intelligence*. Reading, MA, Addison-Wesley.

CHAPTER 2

DATA AND INFORMATION

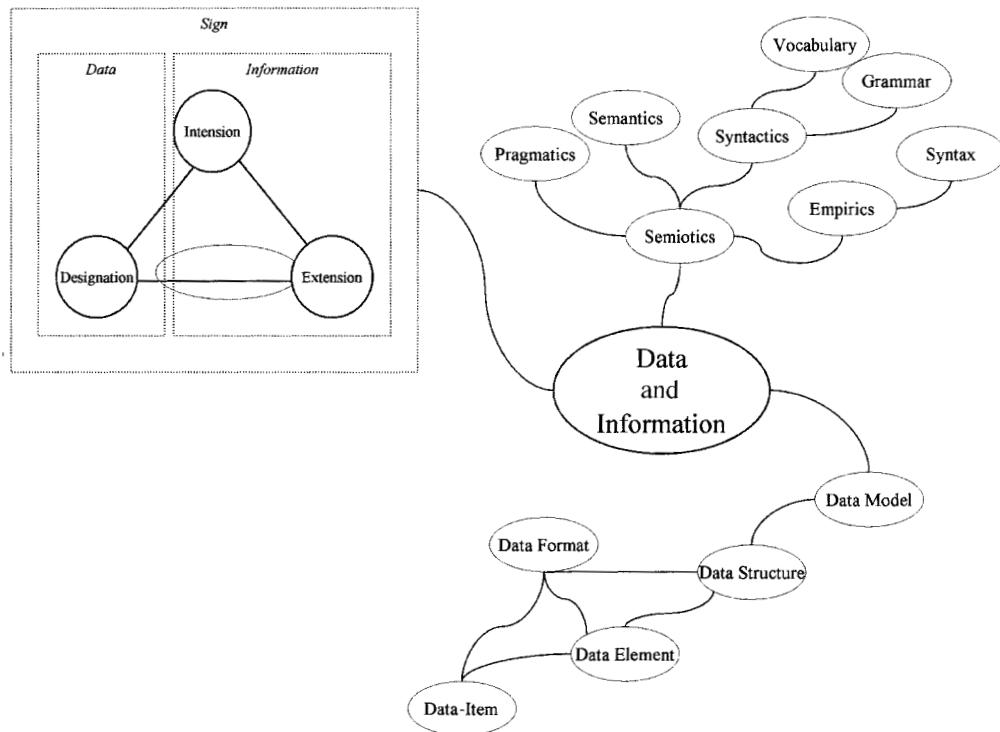
It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts.

Sir Arthur Conan Doyle (1859–1930)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Distinguish between data and information in terms of the concept of a sign
- Describe the four layers of a sign
- Distinguish between syntax and semantics, and explain the relevance of this distinction to the concept of a database schema



**2.1****INTRODUCTION**

In Chapter 1 we defined data as facts or positive assertions about some domain of discourse. According to this definition a database is a logically organised collection of facts about some domain. A datum – a unit of data – is a symbol or a set of symbols which is used to represent something. This relationship between symbols and what they represent is the essence of what we mean by information. Hence, information is interpreted data – data supplied with semantics. In this chapter we use the concept of a sign and the discipline which studies signs – semiotics – to help us more clearly explain the distinction between data and information.

Much of database work, and indeed information systems work, takes place without an examination of this critical distinction between data and information. Hence, readers may skip this chapter without fear of being unable to develop effective database systems. However, an understanding of this distinction is important for a number of reasons:

- It has a bearing on the nature of database development. We shall argue that requirements specification techniques such as entity–relationship diagramming (Chapter 16) are sign-systems
- It helps us understand more clearly the place of database systems within ICT systems and information systems more generally (Chapter 4)
- It helps us understand the critical place of data in modern business and the need for effective planning (Chapter 21) and administration (Chapter 23) of data.

2.2**SIGNS**

The concept of information is an extremely vague one, open to many different interpretations (Stamper, 1989). One conception popular in the computing literature is that information results from the processing of data: the assembly, analysis or summarisation of data. This conception of information as analogous to chemical distillation is useful, but ignores the important place of human interpretation in any understanding of information.

We shall argue that both data and information are embodied in the concept of a sign and that hence signs and sign-systems are the fundamental stuff of database systems.

A sign is anything that is significant. In a sense, everything that humans do is significant to some degree. The world within which humans find themselves is resonant with sign-systems. A sign-system is any organised collection of signs. Everyday spoken language is probably the most readily accepted and complex example of a sign-system. Signs however exist in most other forms of human activity, since signs are critical to the process of human communication and understanding.



Example ➔ Humans communicate through non-verbal as well as verbal sign-systems. We colloquially refer to such non-verbal communication as 'body language'. Hence, humans can impart a great deal in the way of information by facial movements and other forms of bodily gesture. Such gestures are also signs.

Note the link between the words *sign* and *significant* in English. These words clearly have the same root. The concept of the significance of signs cannot be divorced from people. Different people find different things significant. Many such differences in interpretation are due to differences in the context and culture of communication.

2.3 (c) SEMIOTICS

Semiotics or semiology is the study of signs or more precisely of sign-systems. Signs and sign-systems can be considered in terms of four interdependent levels (Stamper, 1989): pragmatics, semantics, syntactics and empirics. These constitute the four main branches of semiotics.

2.3.1 PRAGMATICS

Pragmatics is the study of the general *context and culture* of communication or the shared assumptions underlying human understanding. For communication to occur between human beings signs must exist in a context of shared understanding. As we shall see, there must be agreed expectations among a group of people about the symbols and the referents or concepts they signify. Pragmatics is the study of such mutual understanding. Much of pragmatics can be considered as the study of culture – the common expectations underlying human communicative behaviour in a particular context.

2.3.2 SEMANTICS

Semantics is the study of the *meaning* of signs – the association between signs and behaviour. Semantics can be considered as the study of the link between symbols and their referents or concepts, particularly the way in which signs relate to human behaviour embodied in norms.

2.3.3 SYNTACTICS

Syntactics is the study of the logic and *grammar* of sign systems. Syntactics is devoted to the study of the physical form rather than the content of signs.

2.3.4 EMPIRICS

Empirics is the study of the *physical characteristics* of the medium of communication. Empirics is devoted to the study of communication channels and their characteristics, e.g. sound, light, electronic transmission etc.

2.4 SEMANTICS

The relationship between information and data is located in the area of meaning – semantics. Semantics is the study of what signs refer to. Communication involves the use and interpretation of signs. When we communicate, the sender has to externalise his or her intentions in terms of some signs. In face-to-face conversation this will involve the use of linguistic signs. The receiver of the message must interpret the signs. In other words, he or she must assign some meaning to the signs of the message. Semantics is concerned with this process of assigning meaning to signs.

A simple model of semantics is one in which a sign can be broken down into three component parts, which are frequently referred to collectively as the meaning triangle (Sowa, 1984) (see Figure 2.1):

- *Designation*. The designation of a sign refers to the symbol (or collection of symbols) by which some concept is known. The designation of a sign is sometimes referred to as the signifier – that which is signifying something

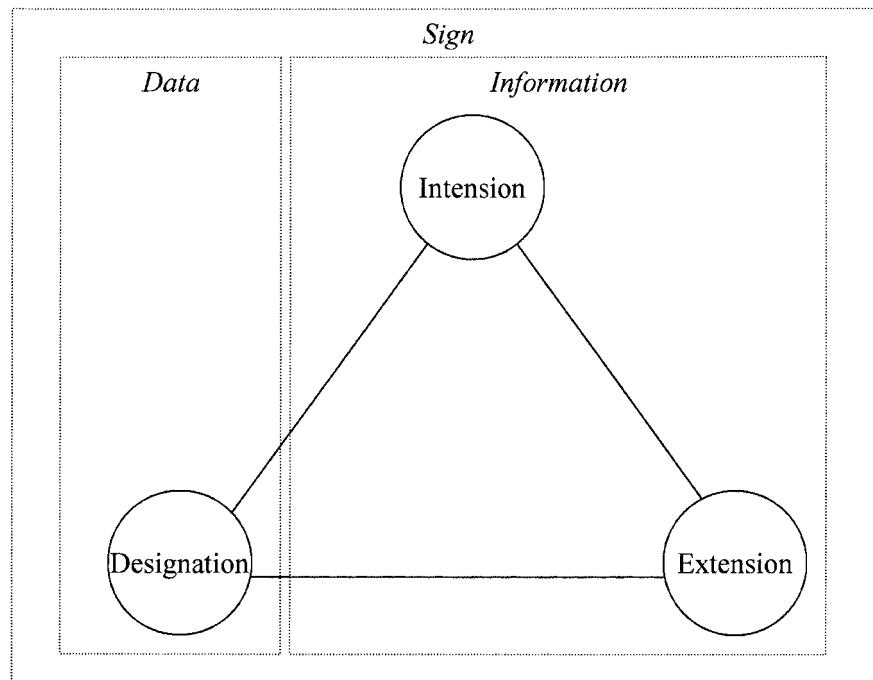


Figure 2.1 The meaning triangle.

- *Extension.* The extension of a sign refers to the range of phenomena that the concept in some way cover. The extension is sometimes known as the referent or the signified – that which is being signified
- *Intension.* The intension of a sign is the collection of properties that in some ways characterise the phenomena in the extension. The intension is the idea of significance

The designation or symbols are equivalent to data in the classic language of information systems. A datum, a single item of data, is a set of symbols used to represent something. Information particularly occurs in the 'stands-for' relation between the designation and its intension, and the intension and its extension.

Examples ➔ In a manufacturing database system the symbols 43 constitute the designation. A possible extension is a collection of products. The intension may be the quantity of a product sold.

The symbols M and F might be significant in some informatics context. To speak of information we must supply some intension and/or extension for the symbols. M might have as its extension the male population; F might have as its extension the female population. Taken together, the meaning of these symbols is supplied by the intension of human gender.

The meaning triangle should not be taken to mean that signs have an inherent meaning. A sign can mean whatever a particular social group chooses it to mean. The same sign may mean different things in different social contexts. As interpreters of signs humans are extremely proficient at assigning the correct interpretation for a sign in a particular context.

Examples ➔ In the Welsh language the same verb *dysgu* (pronounced dusgey) is used both for *to teach* and *to learn*. Hence the same sentence – *Rydw i'n dysgu* – can mean either *I am learning* or *I am teaching* depending on the context supplied usually by elements such as the rest of the conversation.

Take the sign 030500. In a given context we interpret this as a date. However, in the UK we would interpret the significance of the sequence of digits differently from those in the USA. In the UK the first two digits represent the day of the month. In the USA the first two digits represent the month.

**2.5****SYNTACTICS**

The most important category of sign-system used in human communication is that of language. Syntactics concerns itself with all these representational aspects of language. All languages consist of the following elements:

- *Vocabulary*. A complete list of the terms of a language
- *Grammar*. Rules that control the correct use of a language
- *Syntax*. The operational rules for the correct representation of terms and their use in the construction of sentences of the language

We may distinguish between two broad categories of languages: natural languages and formal languages:

- *Natural languages*. These are languages used for everyday communication and include languages such as English, French and Welsh. Natural languages have evolved over time in particular linguistic communities, and indeed continue to evolve. As such they tend to be extremely rich and complex in terms of vocabulary, grammar and syntax
- *Formal languages*. These are artificial languages normally constructed for some defined purpose. As such they tend to have much simpler vocabularies, grammar and syntax. Work in the information systems area abounds with a variety of such formal languages

Example → All the existing programming languages such as C++ or Java are examples of formal languages.

Some of the most well-developed examples of formal languages are those developed in formal logic, including propositional and predicate logic. These languages are useful as ways of representing restricted universes of discourses and developing reasoning about their properties.

Example → Many of the formal languages used to build database systems, such as structured query language (SQL) (Part 3), are founded in formal logic.

2.6**SCHEMAS**

The term universe of discourse (UoD), or domain of discourse, is sometimes used to describe a context within which a group of signs (usually linguistic terms) is used continually by a social group or groups. For work in the area of information systems it is important to develop a detailed understanding of the

structure of these terms. In database work this structure is known as a schema. A schema is an attempt to develop an abstract description of some UoD, usually in terms of a formal language such as logic.

In classic Aristotelian logic the intension of a sign is a collection of properties that may be divided into two groups: the defining properties that all phenomena in the extension must have; and the properties that the phenomena may or may not have. The nature of the defining properties is such that it is objectively determinable whether or not a property has certain phenomena. In classical logic, such properties are designated by predicates (Chapter 9). Hence, defining properties in this view determines a sharp boundary between membership and non-membership of a sign.

A database schema is made up of a series of signs. A schema is therefore a representation of some reality. Although there are other interpretations of the link between intension and extension, in traditional terms, signs in a schema adhere to an Aristotelian interpretation. In a database we must be able to provide an unambiguous procedure for determining whether a phenomenon is a member of a sign or not.

2.7 DATA STRUCTURES, DATA-ITEMS, DATA ELEMENTS AND DATA FORMATS

To build a schema we must have a system of signs or a language we can use. Such a language must have an agreed vocabulary, grammar and syntax. The formal language used for defining schemas is generally described as a data model. A data model establishes a set of principles for organising data. This set of principles that define a data model may be divided into three major parts:

- *Data definition.* A set of principles concerned with how data is structured
- *Data manipulation.* A set of principles concerned with how data is operated upon
- *Data integrity.* A set of principles concerned with determining which states are valid for a database

In general terms, the syntax of data definition in any data model tends to use a hierarchy of data-items, data elements and data structures (Tsitchizris and Lochovsky, 1982).

- A data-item is the lowest-level of data organisation. A data-item is the atomic construct in any data model. In other words, that construct that cannot be divided any further
- A data element is a logical collection of data-items
- A data structure is a logical collection of data elements



Example Hence, in the relational data model (Chapter 7), a data-item corresponds to a column of a table. A data element corresponds to a row of a table, and the one and only data structure in the relational data model is the table or relation.

Some of the semantics of data-items, data elements and data structures in a data model is provided by the concept of a data format. The format of a data-item tends to be referred to as a data type. Some of the meaning of a given data-item is embodied in its data type. A data type declares the range of valid operations that is possible on a data-item.

Example We might declare a given data-item to be a person's age and this data-item to be declared on an integer data type. This means it is possible to conduct the range of arithmetic operations on a person's age.

Most data needed in commercial information systems is what we might call standard data. Standard data is defined in terms of a number of standard data types. A data type acts as a categorisation of data and defines not only the format of some data-item but also the allowable set of operations we may perform on the data in the item. There are a number of standard data types used by most information systems applications. These include:

- *Text.* Strings of symbols made up of characters from the alphabet and a range of other characters
- *Numbers,* including integer, decimal and real numbers
- *Units of time,* including dates, seconds, minutes and hours

Example Clearly it is important for a computer system to know the type of data it is dealing with. For instance, it makes sense to add two integers such as 1 and 2 together. It makes very little sense to add the character A to the character B. Hence, a number data type will allow addition but a text data type will disallow it.

Information systems are now being used to capture, store and manipulate far more complex data types than the standard data types discussed above. This is to enable such systems to handle different media. Such complex data types include:

- *Images.* Graphics and photographic images
- *Audio.* Various forms of sound data
- *Video.* Various forms of moving image



Example ➔ These forms of complex or multi-media data need different coding schemes or formats to enable the representation of this data in digital form. For instance, images are coded in terms of pixels. A pixel is a picture element. A high-resolution computer monitor can be considered as a 1024 by 768 grid in which each cell of the grid is a pixel. To represent an image we therefore need to store, for each pixel, its colour in the image. For a complete monitor over 700,000 pixels are needed.

Information systems are now also required to handle complex data structures as well as complex data-items or elements. Organisations want to be able to define the structure of documentation that they use for communication. A document such as an invoice or a contract is a complex data structure in that it may be made up of text, numeric data and images. Also different organisations may use different formats for such documents. Hence, there is pressure on organisations in the same industrial sector to develop standards for such documentation. Such standards are being specified using a formal language known as XML – extensible markup language. XML is a variant of the Internet language HTML and is discussed in more detail in Chapter 39.

2.8 CASE STUDY: UNIVERSITY MODULE DESCRIPTIONS

Information is important to a university for successful operation and will include information about students, staff and modules. A module description is a critical sign-system in this environment. As an organisational communication a module description can be analysed on a number of levels:

- *Pragmatics* – the study of the general *context and culture* of communication or the shared assumptions underlying human understanding. Key assumptions surrounding the use of module descriptions are that the educational experience can be packaged or chunked in terms of modules, and delivered and assessed in discrete units. Effectively modules become the building blocks of the higher education experience. They are the educational products delivered by universities
- *Semantics* – the study of the *meaning* of signs or sign-systems. Within a university setting, students and staff tend to treat module descriptions as informal contracts. They define particularly what is to be covered in terms of the delivery of the educational experience. They also define the likely mechanism to be used for assessment of students
- *Syntactics* – the study of the logic and *grammar* of sign-systems. At the syntactic level a module description can be considered as a data structure. A standard format for module descriptions will normally be formulated at a particular university and will include data elements such as a module code, module name, level, learning outcomes, syllabus, assessment pattern and



suggested reading. Data-items within such elements are likely to store characters of written text as key symbols

- *Empirics* – the study of the *physical characteristics* of the medium of communication. A module description may be made available in a physical form on paper as part of a student or module handbook. In the modern university it may also be made available electronically over a university intranet and may be stored within a database system for easy retrieval and maintenance

2.9



SUMMARY

- A database can be considered as an organised collection of data which is meant to represent some universe of discourse
- Data are facts. A datum, a unit of data, is one symbol or a collection of symbols that is used to represent something
- Data by itself is meaningless. To prove useful data must be interpreted. Information is interpreted data. Information is data placed within a meaningful context. Information is data with an assigned semantics or meaning
- The distinction between data and information is embodied in the concept of a sign
- Semiotics or semiology is the study of signs, or more precisely of sign-systems
- Signs and sign-systems can be considered in terms of four interdependent levels: pragmatics, semantics, syntactics and empirics
- The term universe of discourse (UoD), or domain of discourse, is sometimes used to describe a context within which a group of signs (usually linguistic terms) is used continually by a social group or groups. For work in the area of information systems it is important to develop a detailed understanding of the structure of these terms. In database work this structure is known as a schema. A schema is an attempt to develop an abstract description of some UoD, usually in terms of a formal language
- To build a schema we must have a system of signs or a language we can use. Such a language must have an agreed vocabulary, grammar and syntax. The formal language used for defining schemas is generally described as a data model
- In general terms, the syntax of data definition in any data model tends to use a hierarchy of data-items, data elements and data structures

2.10



ACTIVITIES

1. Analyse some organisational communication known to you in terms of the levels of semiotics. For instance, in what ways can a student grade for an assessment be treated as a sign? What does it signify in terms of pragmatics, semantics, syntactics and empirics?

2. After reading Part 3, attempt to detail some of the vocabulary, grammar and syntax of the formal language SQL. For instance, what is the syntax of the SELECT statement?
3. Use the meaning triangle to analyse an item of data – order number, debit from a bank account – in terms of the concept of a sign. For instance, what is the designation, intension and extension of an order number?

2.11 REFERENCES

- Sowa, J.F. (1984). *Conceptual Structures: Information Processing in Mind and Machine*. Reading, MA, Addison-Wesley.
- Stamper, R.K. (1989). *Information in Business and Administrative Systems*. London, Batsford.
- Tsitchizris, D.C. and F.H. Lochovsky (1982). *Data Models*. Englewood Cliffs, NJ, Prentice-Hall.

CHAPTER 3

DATABASE, DBMS AND DATA MODEL

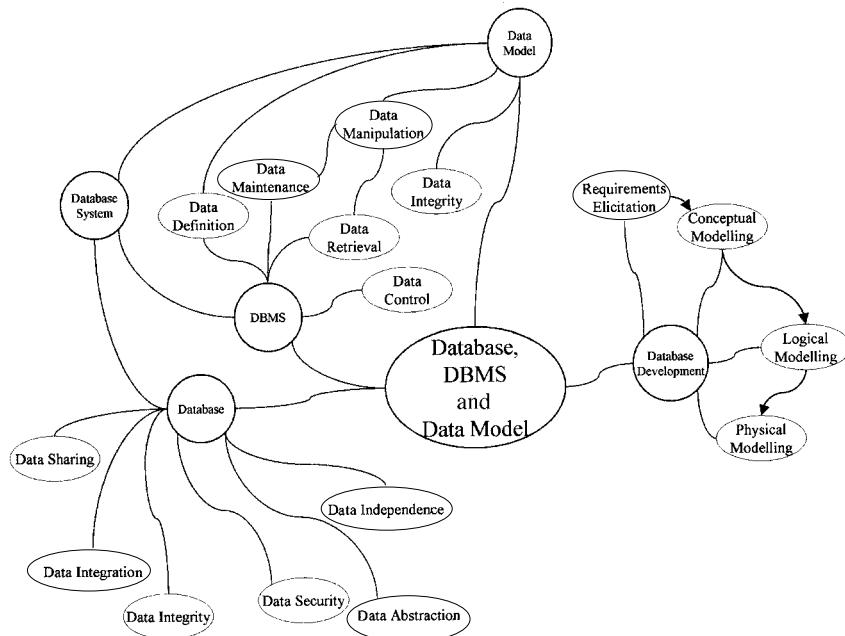
Any sufficiently advanced technology is indistinguishable from magic.

Arthur C. Clarke (1917–)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Establish the need for database systems
- Provide a definition for the concepts of a database, a DBMS and a data model
- Distinguish between data model as architecture and data model as blueprint
- Provide a brief history of data management
- Introduce the process of database system development



3.1 INTRODUCTION

In this chapter we extend the discussion of Chapters 1 and 2 by establishing an initial definition for some of the major concepts which are discussed in the chapters which follow: database, database management system, data model. By a database system we normally mean a database running under some DBMS, both database and DBMS conforming to the principles of some data model. All three concepts of database, DBMS and data model contribute to an understanding of the process of database development.

3.2 ⚡ PIECEMEAL SCENARIO

It is conventional to discuss the need for database systems by portraying a scenario in which an organisation initially does not use database systems.

When organisations first began to use computers they naturally adopted a piecemeal approach to computer systems development. One manual system at a time was analysed, redesigned and transferred onto the computer with little thought to its position within the organisation as a whole. This piecemeal approach was necessitated by the difficulties experienced in using a new and more powerful organisational tool.

Example Consider the situation illustrated in Figure 3.1. In some university the academic registry has developed an information system to store data about students. The personnel department has developed an information system to store data about academic and administrative staff. Each school and department in the university has developed an information system to store data about modules. Each system is independent of the others – it has been developed piecemeal.

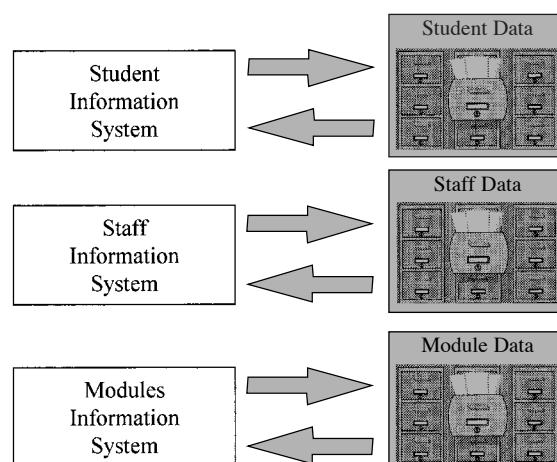


Figure 3.1 Piecemeal scenario.



This approach, by definition, produces a number of separate information systems, each with its own program suite, its own files, and its own inputs and outputs. As a result of this:

- The systems, being self-contained, do not represent the way in which the organisation works, i.e. as a complex set of interacting and interdependent systems
- Systems built in this manner often communicate outside the computer. Reports are produced by one system, which then have to be transcribed into a form suitable for another system. These so-called 'workarounds' proliferate inputs and outputs, and create delays
- Information obtained from a series of separate files is less valuable to employees because it does not provide a complete picture of the activity of an organisation. For example, a sales manager reviewing outstanding orders may not get all the information he needs from the sales system. He may need to collate information about stocks from another file used by the company's stock control system
- Data may be duplicated in the numerous files used by different information systems in the organisation. Hence, personnel may maintain data similar to that held by payroll. This creates unnecessary maintenance overheads and increases the risk of inconsistency

Note, although we have implicitly presented the piecemeal scenario described above as a historical one, in practice many so-called 'legacy' database systems in organisations adhere to many of the characteristics of piecemeal development. Legacy database systems are those applications which have been produced some time in the past to support some form of organisational activity but which are still critically important to the organisation.

Example ➔

Let us contrast the piecemeal approach with an integrated data scenario. Suppose we consider the data used in running a supermarket store – part of a larger national chain of supermarkets.

In each supermarket a number of checkouts operate electronic point-of-sale (EPOS) equipment. This equipment allows checkout staff to record sales by simply scanning bar-codes on products. Details of each sale are transmitted electronically to the store's database.

The sales data automatically updates data held on the current shelf levels of products. If the shelf level of a particular product falls below a certain amount a report is generated for store staff to replenish the shelves from the supermarket's available stock. In replenishing stock on the shelves staff update the database. If the stock level of a product held at the store falls below a limit an order is generated automatically and sent electronically to the central supplies division.



Data held within the database about sales, shelf and stock levels is also used by store management to decide on marketing strategies – which goods to promote, how to site products on shelves etc.

In this example at least three effective uses are being made of the same integrated collection of data: the collection of customer transaction data, the management of stock and the marketing of goods.

3.3 (c) DATABASES AND DATABASE MANAGEMENT SYSTEMS

Because of the many problems inherent in the piecemeal approach, it is nowadays considered desirable to maintain a single, ideally centralised pool of organisational data, rather than a series of separate files. Such a pool of data is known as a database.

It is also considered desirable to integrate the systems that use this data around a piece of software which manages all interactions with the database. Such a piece of software is known as a database management system or DBMS.

A database in manual terms is analogous to a filing cabinet, or more accurately to a series of filing cabinets. A database is a structured repository for data. The overall purpose of such a repository is to maintain data for some set of organisational objectives. Traditionally, such objectives fall within the domain of administration. Most database systems are built to retain the data required for the running of the day-to-day activities of an organisation. Hence, in a university a database system will be needed for such activities as recording assessments of students on modules.

Structure normally implies some logical division, usually hierarchical. Hence we speak of a database as being a collection of data structures. A collection of data structures containing information on students at a university, for instance, would normally constitute a database. Each data structure in a database is in turn also a hierarchical collection of data. In manual terms, such data structures would be folders hung in a filing cabinet. Each folder in the cabinet consists of a series of cards of information, for example, on each student. Each card is further divided into a series of data-items such as StudentNo, StudentName, DateOfBirth etc. Within each data-item a specific value is written.

3.4 (c) PROPERTIES OF A DATABASE

The question of structure is therefore of fundamental importance to a database system. However, the term database also usually implies a series of related properties: data sharing, data integration, data integrity, data security, data abstraction and data independence.



3.4.1 DATA SHARING

Data stored in a database is not usually held solely for the use of one person. A database is normally expected to be accessible by more than one person, perhaps at the same time.

Example → Hence a student database might be accessible by members of not only academic but also administrative staff. In our supermarket example, sales data is accessible by the stock control and management information systems.

3.4.2 DATA INTEGRATION

Shared data brings numerous advantages to the organisation. Such advantages, however, only result if the database is treated responsibly. One major responsibility of database usage is to ensure that the data is integrated. This implies that a database should be a collection of data which, at least ideally, has no redundant data. Redundant data is unnecessarily duplicated data. A data value is redundant when an attribute has two or more identical values. A data value is redundant if you can delete it without information being lost.

Example → In the past, separate files of student information for instance might have been maintained by different academic and administrative departments of a university with many fields in common. The aim of a database system would be to store one logical item of data in one place only. Hence, one student record would be accessible to a range of information systems.

3.4.3 DATA INTEGRITY

Another responsibility arising as a consequence of shared data is that a database should display integrity. In other words, the database should accurately reflect the universe of discourse that it is attempting to model. This means that if relationships exist in the real world between objects represented by data in a database then changes made to one partner in such a relationship should be accurately reflected in changes made to other partners in that relationship.

Example → Hence, if changes are made to the information stored on a particular university degree course, then relevant changes should be made to the information stored on modules and perhaps students of that course.

3.4.4 DATA SECURITY

One of the major ways of ensuring the integrity of a database is by restricting access – in other words, securing the database. The main way this is done in

contemporary database systems is by defining in some detail a set of authorised users of the whole, or more usually parts, of the database.

Example A secure system would be one where the finance department has access to information used for the collection of student fees but is prohibited from changing the fee levels of given students.

3.4.5 DATA ABSTRACTION

A database can be viewed as a model of reality. The information stored in a database is usually an attempt to represent the properties of some objects in the real world. Hence, for instance, an academic database is meant to record relevant details of university activity. We say relevant, because no database can store all the properties of real-world objects. A database is therefore an abstraction of the real world (Chapter 1).

Example Although a university database might record data concerning a student's academic achievement, it is unlikely to store data about the student's hair-style or TV-watching habits.

3.4.6 DATA INDEPENDENCE

One immediate consequence of abstraction is the idea of buffering data from the processes that use such data. The ideal is to achieve a situation where data organisation is transparent to the users or application programs which feed off data. If, for instance, a change is made to some part of the underlying database, no application programs using affected data should need to be changed. Also, if a change is made to some part of an application system then this should not affect the structure of the underlying data used by the application.

These properties amount to desirable features of the ideal database. As we shall see, properties such as data independence are only partly achieved in current implementations of database technology.

3.5 WHAT IS A DATABASE MANAGEMENT SYSTEM?

A Database Management System (DBMS) is an organised set of facilities for accessing and maintaining one or more databases. A DBMS is a shell which surrounds a database or series of databases and through which all interactions take place with the database. The interactions catered for by most existing DBMS fall into four main groups (see Figure 3.2):

- *Data definition* – defining new data structures for a database, removing data structures from the database, modifying the structure of existing data

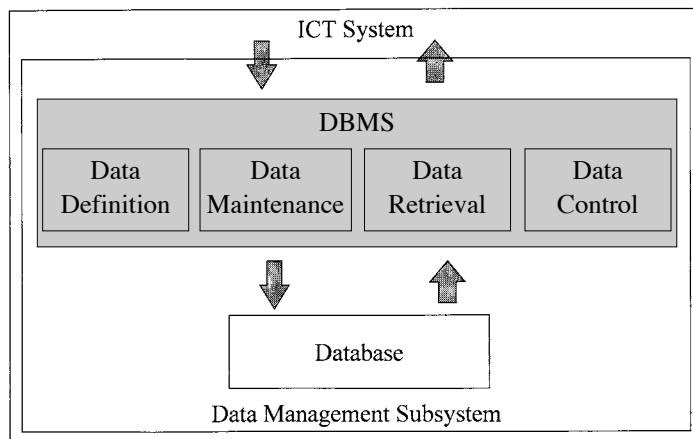


Figure 3.2 Functions of a DBMS.

- *Data maintenance* – inserting new data into existing data structures, updating data in existing data structures, deleting data from existing data structures
- *Data retrieval* – querying existing data by end-users and extracting data for use by application programs
- *Data control* – creating and monitoring users of the database, restricting access to data in the database and monitoring the performance of databases

3.6

DATA MODELS

Every database, and indeed every DBMS, must adhere to the principles of some data model. However, the term data model is somewhat ambiguous. In the database literature the term is used in a number of different senses, two of which are the most important: that of an architecture for data, and that of an integrated set of data requirements.

3.6.1 DATA MODEL AS ARCHITECTURE

The term data model is used to refer to a set of general principles for handling data (Tsitchizris and Lochovsky, 1982). Here, people talk of the relational data model, the hierarchical data model or the object-oriented data model. It is this definition of the term data model that we shall explore in Part 2.

This set of principles that define a data model may be divided into three major parts:

- *Data definition* – a set of principles concerned with how data is structured
- *Data manipulation* – a set of principles concerned with how data is operated upon

- *Data integrity* – a set of principles concerned with determining which states are valid for a database

Data definition involves defining an organisation for data: a set of templates for the organisation of data. Data manipulation concerns the process of how the data is accessed and how it is changed in the database. Data integrity is very much linked with the idea of data manipulation in the sense that integrity concerns the idea of what are valid changes and invalid changes to data.

Any database and DBMS must adhere to the tenets of some data model.

Example  Hence, in the relational data model (Chapter 7), data definition involves the concept of a relation, data manipulation involves a series of relational operators, and data integrity amounts to two rules – entity and referential integrity.

Note that by the data integrity part of a data model we are describing only those rules that are inherently part of the data model. A great deal of other integrity constraints or rules will have to be specified by additional means, i.e. using mechanisms not inherently part of the data model.

3.6.2 DATA MODEL AS BLUEPRINT

The term data model is also used to refer to an integrated, but implementation-independent, set of data requirements for some application (Howe, 1981). Here, we might speak of the order-processing data model, the accounts-receivable data model, or the student-admissions data model. A data model in this sense is an important component part of any information systems specification (Beynon-Davies, 1998). We shall explore the development of such application data models in Part 4.

3.6.3 TYPOLOGY OF DATA MODELS

We may make a distinction between three generations of architectural data model (Brodie *et al.*, 1984):

- *Primitive data models*. In this approach objects are represented by record-structures grouped in file-structures. The main operations available are read and write operations over records
- *Classic data models*. These are the hierarchical, network and relational data models. The hierarchical data model is an extension of the primitive data model discussed above. The network is an extension of the hierarchical approach. The relational data model is a fundamental departure from the hierarchical and network approaches (Chapter 7)
- *Semantic data models*. The main problem with classic data models such as the relational data model is that they maintain a fundamental record-orientation.



In other words, the meaning of the information in the database – its semantics – is not readily apparent from the database itself. Semantic information must be consciously applied by the user of databases using the classic approach. For this reason, a number of so-called semantic data models have been proposed. Semantic data models (SDMs) attempt to provide a more expressive means of representing the meaning of information than is available in the classic models. In many senses the post-relational data model described in Chapter 10 and the object-oriented data model described in Chapter 8 can both be regarded as semantic data models.

3.7

DATABASE SYSTEM

In this text we use the term database system to include the concepts of a data model, DBMS and database. To build a database system we must use the facilities of some DBMS to define suitable data structures – the schema. We also use the DBMS to define integrity constraints. We then populate such data structures using the manipulative facilities of the DBMS. The DBMS will enforce integrity on any manipulative activity.

Both the DBMS and the resulting database must adhere to the principles of some data model. In practice, there is usually some divergence between the abstract ideas specified in a data model and the practical implementation of ideas in a DBMS. This is particularly evident in the area of relational database systems where the relational data model (Chapter 7) is not completely represented by any existing DBMS.

3.8

A BRIEF HISTORY OF DATA MANAGEMENT

Database systems are tools for data management and form a branch of information and communications technology. Data management is an ancient activity and the use of ICT for such activity is a very recent phenomenon. Gray (1996) provides a useful categorisation of data management into six historical phases. We have adapted this somewhat to take account of recent developments:

3.8.1 MANUAL RECORD MANAGERS (4000 BC–AD 1900)

Human beings have been keeping data records for many thousands of years. Interestingly, some of the earliest recorded uses are for the representation of economic information. For example, the first known writing describes the royal assets and taxes in Sumeria dating to around 4000 BC. Although improvements were made in the recording medium with the introduction of paper, little radical change was made to this form of manual data management for six thousand years.



3.8.2 PUNCHED-CARD RECORD MANAGERS (1900–55)

Automated data management began with the invention of the Jacquard loom circa 1800. This technology produced fabric from patterns represented on punched-cards. In 1890, Herman Hollerith applied the idea of punched-cards to the problem of producing the US census data. Hollerith formed a company (later to become International Business Machines – IBM) to produce equipment that recorded data on such cards and was able to sort and tabulate the cards to perform the census. Use of punched-card equipment and electro-mechanical machines for data management continued up until the mid-1950s.

3.8.3 PROGRAMMED RECORD MANAGERS (1955–70)

Stored program computers were developed during the 1940s and early 1950s to perform scientific and military calculations. By 1950 the invention of magnetic tape caused a significant improvement in the storage of data. A magnetic tape of the time could store the data equivalent to 10,000 punched-cards. Also, the new stored-program computers could process many hundreds of records per second using batch processing of sequential files. The software of the day used a file-oriented record-processing model for data. Programs read data as records sequentially from several input files and produced new files as output. A development of this approach was batch transaction processing systems. Such systems captured transactions on card or tape and collected them together in a batch for later processing. Typically these batches were sorted once per day and merged with a much larger data-set known as the master file to produce a new master file. This master file also produced a report that was used as a ledger for the next day's business.

Batch processing used the computers of the day very efficiently. However such systems suffered from two major shortcomings. First, errors were not detected until the daily run against the master file, and then they might not be corrected for several days. Second, the business had no way of accurately knowing the current state of its data because changes were only recorded in the batch run, which typically ran overnight.

3.8.4 ON-LINE NETWORK DATA MANAGERS (1965–80)

Applications such as stock-market trading and travel reservation could not use the data supplied by batch processing systems because it was typically at least one day old. Such applications needed access to current data. To meet such a need many organisations began experimenting with the provision of on-line transaction databases. Such databases became feasible with developments in tele-processing monitors which enabled multiple terminals to be connected to the same central processing unit (CPU). Another key development was the invention of random access storage devices such as magnetic drums and disks. This, in turn, led to improvements in file structures with the invention of indexed-sequential files. This enabled data management programs to read a



few records off devices, update them, and return the updated records for inspection by users.

The indexed-sequential file organisation led to the development of a more powerful set-oriented record model. This structure enabled applications to relate two or more records together in hierarchies. This hierarchical data model evolved into a more flexible representation known as the network data model.

Managing set-oriented processing of data became so commonplace that the COBOL programming language community chartered a database task group (DBTG) to define a standard data definition and manipulation language for this area. Charles Bachman, who had built a prototype data navigation system at General Electric (called the Integrated Data Store – IDS), received a Turing award for leading the DBTG effort. In his Turing lecture, Bachman described this new model of data management in which programs could navigate among records by traversing the relationships between them.

The DBTG model crystallised the concept of schemas and data independence through the definition of a three-level architecture for database systems (DBTG, 1971) (Chapter 6). These early on-line systems pioneered solutions to running many concurrent update activities against a database shared among many users. This model also pioneered the concept of transactions and the management of such transactions using locking mechanisms and transaction logs (Chapter 29).

A person named Goodrich saw the work that was being done at GEC and decided to port IDS across onto the new IBM system 360 range of computers. John Cullinane entered into a marketing agreement with Goodrich. This was the beginning of a company named Cullinane, later Cullinet, which established the IDMS DBMS as the dominant force of network DBMS on IBM mainframes in the 1960s, 1970s and 1980s.

3.8.5 RELATIONAL DATA MANAGEMENT (1980–95)

Despite the success of network databases, many software practitioners felt that navigational access to data was too low-level and primitive for effective application building.

In 1970 an IBM scientist, Dr E.F. Codd, outlined the relational data model which offered higher-level definition and manipulation of data, and published an influential paper on database architecture entitled ‘A relational model for large shared data banks’ (Codd, 1970). Researchers at IBM used the material in Codd’s early publications to build the first prototype relational DBMS called System/R. This was emulated at a number of academic institutions, perhaps the foremost example being the INGRES research team at the University of Berkeley, California.

During the 1970s and early 1980s relational databases gained their primary support from academic establishments. The commercial arena was still dominated by IDMS-type databases. In 1983, however, IBM announced its first relational database for large mainframes – DB2. Since that time, relational databases have grown from strength to strength.



Many people claim that relational systems received their strongest impetus from the large number of PC-based DBMS created during the 1980s. Although it is undoubtedly true that many of these packages such as dBase II and its derivatives were relational-like, there is some disagreement over whether they truly represented relational DBMS. The most popular current example of this class of DBMS is Microsoft Access (Chapter 33).

The relational model was influential in stimulating a number of important developments in database technology. For example, developments in client-server computing, parallel processing and graphical user interfaces were built on the bedrock supplied by the relational data model.

The idea of applying parallel computer architectures to the problems of database management have something of the order of twenty years of history. Only comparatively recently however has the idea been treated seriously in the development of commercial database systems. Chapter 38 provides an overview of this area and illustrates how many modern relational DBMS are offering parallel processing capabilities.

3.8.6 MULTIMEDIA DATABASES (1995–99)

Databases are now being used for storing richer data types – documents, images, voice and video data, as storage engines for the Internet and Intranet (Chapter 39), and offer the ability to merge procedures and data (Chapter 31).

Traditionally, a database has simply stored data. The processing of data was accomplished by application programs working outside, but in cooperation with, a DBMS. One facet of many modern relational DBMS is their ability to embed processing within the database itself. This has a number of advantages, particularly in its suitability for client-server architectures (Chapter 36).

In recent years many claims have been made for new types of database systems based on object-oriented ideas. A number of commercial object-oriented DBMS have now become available. Many of the relational vendors are also beginning to offer object-oriented features in their products, particularly to enable handling complex data. This issue of object-relational systems is considered in Chapter 10.

3.8.7 WEB-BASED SYSTEMS (1999–)

We would add a further phase to the typology outlined above – that of Web-based systems. The World-Wide-Web, or Web for short, is a set of technology standards for managing and disseminating information over the Internet. Web-based standards have become dominant in the definition of front-end ICT systems. Contemporary databases provide the data for such systems and hence a great deal of activity has occurred in suggesting various approaches for integrating DBMS with Web-sites (Chapter 43). Certain Web-based standards, such as XML, are also proving significant in attempts to integrate diverse database systems (Chapter 39).



3.9

DATABASE DEVELOPMENT

Producing a database model is an important part of the process of database development. Database development is generally a process of modelling. It is a process of successive refinement through three levels of model: conceptual models, logical models and physical models. A conceptual model is a model of the real world expressed in terms of data requirements. A logical model is a model of the real world expressed in terms of the principles of some data model. A physical model is a model of the real world expressed in terms of the constructs of some DBMS such as tables, and access structures such as indexes.

Hence there are three core stages to any database development task: conceptual modelling, logical modelling and physical modelling (see Figure 3.3).

Some people see conceptual modelling as a stage headed by a requirements elicitation. This process involves eliciting the initial set of data and processing requirements from users. The conceptual modelling stage can be thought of as comprising two substages: view modelling, which transforms the user requirements into a number of individual user views, and view integration which combines these schemas into a single global schema. Most conceptual models are built using constructs from the semantic data models. In Chapter 16 we shall use the extended entity–relationship model to illustrate the process of

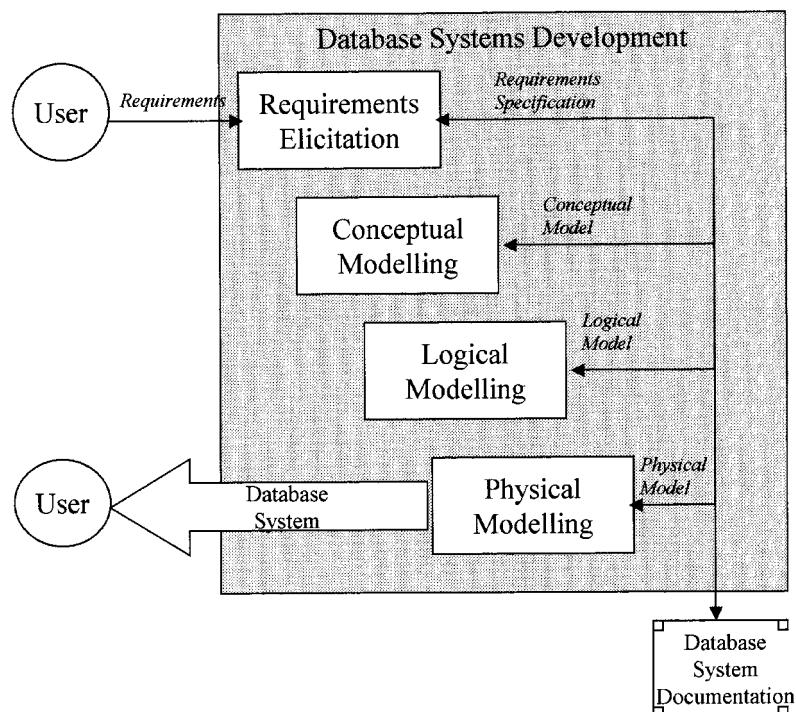


Figure 3.3 Database systems development.

building conceptual models. The process of logical modelling is concerned with determining the contents of a database independently of the exigencies of a particular physical implementation. This is achieved by taking the conceptual model as input, and transforming it into the architectural data model supporting the target database management system (DBMS). This is usually the relational data model. In Chapter 7 we shall outline the key concepts of the relational data model. Physical modelling involves the transformation of the logical model into a definition of the physical model suitable for a specific software/hardware configuration. This is usually some schema expressed in the data definition language of SQL. In Part 3 we illustrate the primary components of this modern-day database sublanguage.

3.9.1 DATA ANALYSIS

The term data analysis is frequently used in the context of database work. Data analysis is a term generally reserved for conceptual and logical modelling. There are two complementary approaches to conducting data analysis: normalisation and entity–relationship modelling. Normalisation is a technique based upon the work of Codd (Codd, 1970). Sometimes referred to as a bottom-up database design technique, normalisation involves the transformation of data subject to a range of data-maintenance problems into a form free from such problems (Chapter 18). Entity–relationship modelling is sometimes known as a top-down design technique (Chapter 16). Entity–relationship modelling involves representing some universe of discourse in terms of entities and relationships. Object modelling is discussed in this work as an extension of entity–relationship modelling. It is particularly seen as a recent attempt to use the constructs of top-down data analysis to model system behaviour or dynamics (Chapter 17).

3.10 CASE STUDY: GORONWY GALVANISING

To understand the importance of database systems to effective data management, consider the following case which describes a paper-based information system in a manufacturing company.

Goronwy Galvanising Ltd is a small company specialising in treating steel products such as lintels, crash barriers, palisades etc., produced by other manufacturers. Goronwy is a subsidiary of a large multi-national whose primary business is the production of various alloys. The multi-national maintains ten plants on similar lines to Goronwy around the UK. Each plant is relatively autonomous in terms of managing its day-to-day business. Head office coordinates administrative activities such as accounting, finance and the dissemination of information.

Galvanising, in very simplistic terms, involves dipping steel products into baths of molten zinc to provide a rustproof coating. Untreated steel products are described as being ‘black’. Treated steel products are referred to as ‘white’. There is a slight gain in weight as a result of the galvanising process.



Goronwy mainly processes steel for a major manufacturer known as Blackheads. More than 80% of their business is with Blackheads, which places a regular set of orders with Goronwy. Other manufacturers, such as Pimples, order galvanised steel on an irregular basis.

The staff at Goronwy consists of a plant manager, a production controller, an office clerk, 3 shift foremen and 50 shopfloor workers. The plant remains open 24 hours per day, seven days a week. Most of the production workers, including the foremen, work a shift-pattern.

Black products are delivered to Goronwy on large trailers in bundles referred to as batches. Each trailer may be loaded with a number of different types of steel product and each batch is labelled with a unique job number. Each trailer is given its own delivery advice note detailing all the associated jobs on the trailer.

Goronwy mainly processes lintels for the major steel manufacturer Blackheads. The advice note supplied with lintels is identified by an advice number specific to this manufacturer. Each job is identified on the advice note by a job number generated by Blackheads. Below we present a typical advice note received from this company detailing all the black material on a particular trailer.

Blackheads Steel Products		Delivery Advice			
Advice No.: A3137	Date: 20/01/2003	Customer Name: Goronwy Galvanising		Instructions: Galvanise and Return	
Order No.	Description	Product Code	Item Length	Number	Weight (Tonnes)
13/1193G	Lintels	UL150	1500	20	
44/2404G	Lintels	UL150	15000	20	
70/2517P		UL135	1350	20	
23/2474P		UL120	1200	16	
Haulier: International 5	Received in Good Order:				

On arrival at the galvanising plant the black material is unpacked and checked for discrepancies with the data on the advice note. Two major types of discrepancies occur:

- A count discrepancy occurs when the number of steel items actually found is less than or greater than the number indicated
- A non-conforming black discrepancy arises when some of the material is unsuitable for galvanising. For instance, a steel lintel may be bent

Both types of discrepancy are written as annotations to the advice note.

When all the material has been checked the advice note is given to the production controller. He and the office clerk transcribe details, including any discrepancies, from the advice note to a job sheet. A sample job sheet is provided below.

Job Sheet					
Order No.	Description	Product Code	Item Length	Order Qty	Batch Weight
13/1193G	Lintels	UL150	1500	20	
Count Discrepancy	Non-Conforming Black	Non-Conforming White	Non-Conforming No Change		
Galvanised	Despatch No.	Despatch Date	Qty Returned	Weight Returned	
Y					

The job sheet is passed down to the shopfloor where the shift foreman uses it to record details of processing. Most jobs will pass through processing smoothly. The steel items will be placed on racks, dipped in the zinc bath and left to cool. The site foreman will then check the condition of each job. If all items have galvanised properly he will put a Y for yes in the galvanised box on the job sheet and pass it back to the production controller.

Occasionally some of the items will not have galvanised properly. Such items will then be classed as non-conforming white and the number of such items placed in the appropriate box on the job sheet. Non-conforming white items will typically be re-galvanised at some later date.

When the company has treated a series of jobs, the production controller will issue a despatch note and send it down to the shopfloor. Workers will then stack the white material on trailers ready to be returned to the appropriate manufacturers. Each trailer for despatch will have an associated despatch note detailing the white material on the trailer.

Partial despatches may be made from one job. This means that the trailer of white material need not correspond to the trailer of original black material originally supplied to Goronwy. Hence the data on delivery advices need not



correspond precisely with the data on despatch advices. A typical despatch advice is represented below.

Goronwy Galvanising		Delivery Advice					
Advice No.:	Date:	Customer Name and Address: Blackheads					
Order No.	Description	Product Code	Item Length	Order Qty	Batch Weight	Returned Qty	Batch Weight
13/1193G	Lintels	UL150	1500	20	150	20	150
44/2404G	Lintels	UL150	15000	20	150	20	150
70/2517P		UL135	1350	20	130	20	135
23/2474P		UL120	1200	16	100	14	82
Driver	Received by:						

It should be apparent that this information system would benefit from a database system for a number of reasons including:

- *Data sharing*. The data is shared between a number of people, particularly production controllers and shift foremen
- *Data integration*. Currently a considerable amount of time is taken in transferring data from one collection of forms, such as delivery advices, onto other forms, such as job sheets
- *Data integrity*. Human errors in entering the wrong data onto any of the three forms in this information system can have considerable repercussions for the efficiency of the galvanising process

3.11 SUMMARY

- Figure 3.4 illustrates the key concepts described in this chapter. At the centre of the figure lies the shared collection of data: the database
- Surrounding the database we have the shared collection of facilities: the DBMS
- Both the DBMS and database adhere to the tenets of some data model. The DBMS is used to manage all interactions with the database by end-users or application programs
- The components of database, DBMS and data model all contribute to our definition of a database system
- Database development is the process of representing the data needed to support some organisational activity in some chosen DBMS

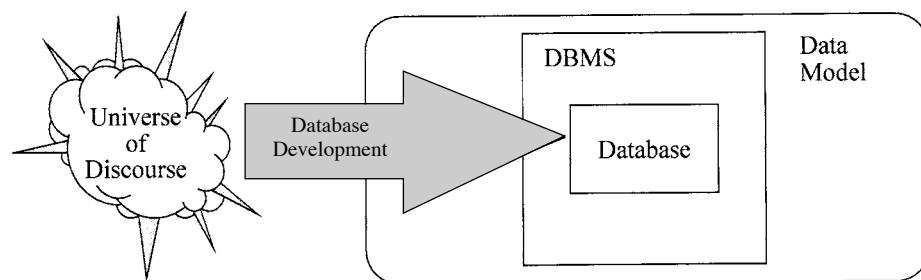


Figure 3.4 Database, DBMS and data model.

3.12 (c) ACTIVITIES

1. Provide a list of applications in an organisation known to you. Identify the areas of replicated data in such applications.
2. Describe and define the key properties of a database system. Give an organisational example of the benefits of each property.
3. Conduct an investigation to identify the named databases in an organisation known to you.
4. Examine a DBMS known to you and identify the DBMS functions.
5. Describe the key phases of the database development process.
6. Distinguish between logical and physical database design.

3.13 (p) REFERENCES

- Beynon-Davies, P. (1998). *Information Systems Development: An Introduction to Information Systems Engineering*. Basingstoke, Macmillan (now Palgrave).
- Brodie, M.L., J. Mylopoulos and T.W. Schmidt (Eds) (1984). *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*. Berlin, Springer-Verlag.
- Codd, E.F. (1970). A relational model for large shared data banks. *Comm. of ACM* **13**(1): 377–87.
- DBTG (1971). *Report of the Codasyl Database Task Group*, ACM.
- Gray, J. (1996). The evolution of data management. *IEEE Computer* 38–46.
- Howe, D. (1981). *Data Analysis for Database Design*. London, Edward Arnold.
- Tsitchizris, D.C. and F.H. Lochovsky (1982). *Data Models*. Englewood Cliffs, NJ, Prentice-Hall.

CHAPTER 4

DATABASE SYSTEMS, ICT SYSTEMS AND INFORMATION SYSTEMS

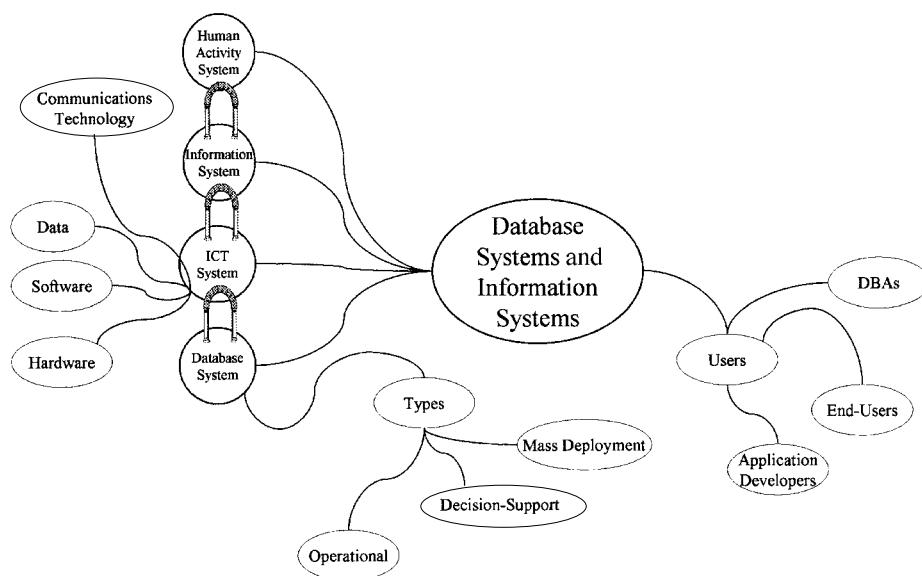
It is impossible to design a system so perfect that no one needs to be good.

T.S. Eliot (1888–1965)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define the concept of a system and its applicability to systems of technology, information and human activity
- Describe the place of a database system within an ICT system
- Distinguish between various levels of a database model
- Describe a number of types of user for a database system





4.1 INTRODUCTION

Databases are used in the larger context of ICT systems and of information systems (IS). The aim of this chapter is to describe some of this context. First we define the concept of a system. Then we review the distinction between three types of system of concern to us: systems of information and communications technology, systems of information and systems of human activity. This helps us to position database systems in their rightful place within modern electronic organisations. We conclude with an examination of some modern types of database system in the context of this system hierarchy.

4.2 SYSTEM

Systems thinking is fundamental to the modern discipline of organisational informatics: the application of information, information systems and ICT in organisations (Beynon-Davies, 2002).

A system might be defined as a coherent set of interdependent components which exists for some purpose, has some stability, and can be usefully viewed as a whole. Systems are generally portrayed in terms of an input–process–output model existing within a given environment. The environment of a system might be defined as anything outside a system which has an effect on the way the system operates. The inputs to the system are the resources it gains from its environment or other systems. The outputs from the system are those things which it supplies back to its environment or other systems. The process of the system is the activity which transforms the system inputs into system outputs. Most organisations are open systems in that they are affected by their environment and other systems.

4.3 INFORMATION SYSTEM

An information system is a system which provides information for some organisation or part of an organisation. An information system is a system of communication between people. Information systems are systems involved in the gathering, processing, distribution and use of information. In a sense, information systems constitute sign manipulation systems.

Certain information systems may have evolved within human activity systems over a prolonged period of time. However, in most modern organisations information systems have been rationally designed. Information systems have to be designed in the sense that the key features of such systems need to be determined prior to the construction and the implementation of the systems. Such key features or properties are critical ways in which we can assess the worth or success of some information system.

Traditionally, design features of an information system fall into one of two categories: functionality – what the system does; usability – how the system is

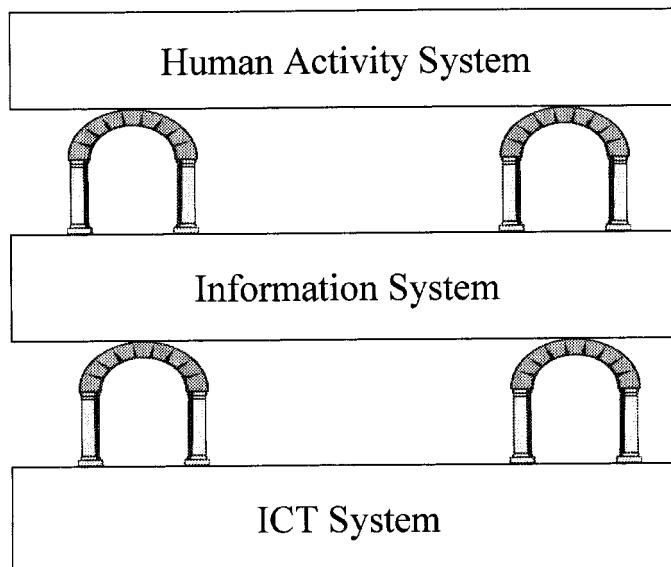


Figure 4.1 ICT, IS and human activity systems.

used. One should note that both functionality and usability are inherently related to the place of the information system within the context of some human activity system. Hence to functionality and usability we should add utility. Utility is an important but neglected feature of an information system. Utility concerns the contribution the IS makes to supporting the human activity of some organisation.

Information systems support human activity systems and are supported by ICT systems (Figure 4.1).

4.4

HUMAN ACTIVITY SYSTEM

The idea of a system has been applied in many fields as diverse as physics, biology and electronics. The class of systems to which computing is generally applied has been referred to as human activity systems (Checkland, 1987). Such systems have an additional component added to the input-process-output model described above: people. Human activity systems consist of people, conventions and artefacts designed to serve human needs. Every human activity system will have one or more information systems. The purpose of such information systems is to support and enable the effective management of the human activity system. Organisations are normally made up of a number of human activity systems and hence normally need a number of information systems to work effectively.



Example ➔ In the Case Study at the end of the chapter we describe the current electoral process in the UK as a human activity system which relies very much on a manually based information system. In a later chapter we discuss the potential of ICT systems to improve aspects of information-handling in support of this human activity system.

4.5 INFORMATION AND COMMUNICATIONS TECHNOLOGY (ICT) SYSTEM

Information and communications technology is any technology used to support information gathering, processing, distribution and use. ICT provides a means of constructing aspects of information systems, but is distinct from information systems. Modern ICT consists of hardware, software, data and communications technology:

- *Hardware*. This comprises the physical (hard) aspects of ICT consisting of processors, input devices and output devices
- *Software*. This comprises the non-physical (soft) aspects of information technology. Software is essentially programs – sets of instructions for controlling computer hardware
- *Data*. This constitutes a series of structures for storing data on peripheral devices such as hard disks. Such data is manipulated by programs and transmitted via communication technology
- *Communication technology*. This forms the interconnective tissue of information technology. Communication networks between computing devices are essential elements of the modern ICT infrastructure of organisations

Examples ➔

- Hardware includes devices such as keyboards (input), processing units and monitors (output)
- Software includes operating systems, programming languages and office packages
- Data is normally stored in databases managed by a database management system
- Communications technology includes such components as cabling, transmitters and routers

An information and communications technology system is a technical system. Such systems are frequently referred to as examples of 'hard' systems in the sense that they have a physical existence. An ICT system is an organised collection of



hardware, software, data and communications technology designed to support aspects of some information system. An ICT system has data as input, manipulates such data as a process and outputs manipulated data for interpretation within some human activity system.

Example → Take an order processing ICT system. In such a system the data entered will describe the properties of orders. The manipulation comprises the processing of orders probably in relation to other data collected such as that on customers. The manipulated data in the system constitutes the processed orders.

The order processing ICT system will support that activity concerned with the effective sales of products or services to customers. Without effective and efficient performance of this activity the organisation is unlikely to survive in its marketplace.

4.6

LAYERS OF AN ICT SYSTEM

It is useful to consider an ICT system as being made up of a number of subsystems or layers (Figure 4.2):

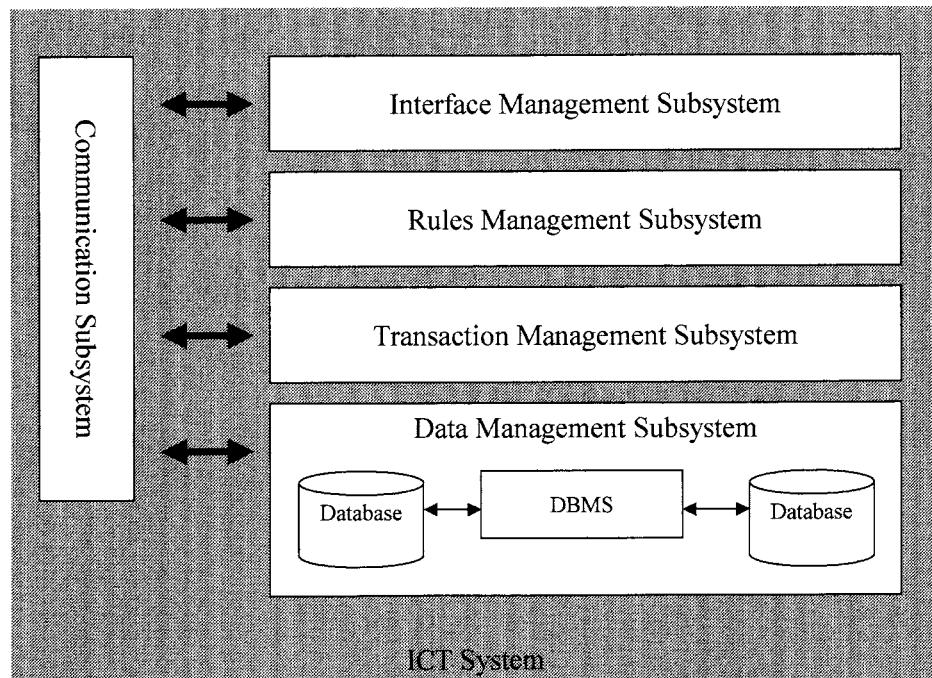


Figure 4.2 Layers of an ICT system.



- *Interface subsystem.* This subsystem is responsible for managing all interaction with the end-user. Hence, it is frequently referred to as the user interface
- *Rules subsystem.* This subsystem manages the application logic in terms of a defined model of business rules
- *Transaction subsystem.* This subsystem acts as the link between the data subsystem and the rules and interface subsystems. Querying, insertion and update activity is triggered at the interface, validated by the rules subsystem and packaged as units (transactions) that will initiate actions (responses or changes) in the data subsystem
- *Data subsystem.* This subsystem is responsible for managing the underlying data need by an application

In the contemporary ICT infrastructure each of these parts of an application may be distributed on different machines, perhaps at different sites. This means that each part usually needs to be stitched together in terms of some communications backbone. For consistency, we refer to this facility here as the communications subsystem.

In terms of the data management layer, aspects of the processing of this layer (Chapter 36) or aspects of the data (Chapter 37) may be distributed.

Historically, the four parts of a conventional ICT application were constructed using one tool, the high-level or third generation programming language (3GL). A language such as COBOL was used to declare appropriate file structures (data subsystem), encode the necessary operations on files (transaction subsystem), validate data processed (rules subsystem) and manage the terminal screen for data entry and retrieval. However, over the last couple of decades there has been a tendency to use a different, specialised tool for one or more of these layers. For instance:

- Graphical user interface tools have been developed as a means of constructing sophisticated user interfaces
- Fourth generation languages have been developed as a means of coding business rules and application logic
- Transaction processing systems have been developed to enable high throughput of transactions
- Database management systems have been developed as sophisticated tools for managing multi-user access to data
- Communications are enabled by a vast array of software supporting local area and wide area communications

The key conclusions to be drawn are that information systems support human activity systems and consequently cannot be designed without an effective understanding of the context of human activity. ICT systems are critical components of contemporary information systems. Database systems are critical elements of ICT systems.

4.7 Socio-technical system

Most systems in organisations are examples of socio-technical systems. A socio-technical system is a system of technology used within a system of activity. Information systems are primary examples of socio-technical systems. Information systems generally consist of ICT systems used within some human activity system. They therefore bridge between ICT and human activity. Part of the human activity will involve the use of the ICT system. Such use occurs at the human-computer interface. The information provided by the ICT system will also drive decision-making, leading to further action within the organisation.

This relationship between the three levels of system is illustrated in Figure 4.3.

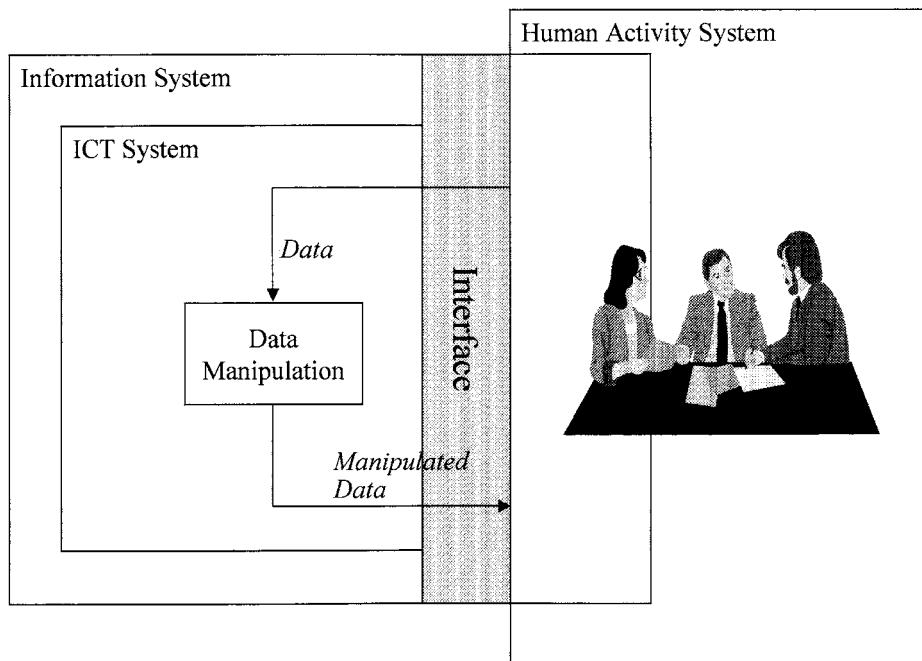


Figure 4.3 Socio-technical system.

Example In a video store the human activity system is generally involved in hiring out copies of video-cassettes or DVDs. To support hiring activity, staff need to have an information system which tells them who are valid members, what copies they have hired out and what copies they have in stock. An ICT system is employed to record details of copies in stock, membership details, details of particular hiring and details of payments made.

Functionality is primarily a feature of the ICT system. It is a property concerned with the efficiency and effectiveness of the technical artefact. Usability is a feature of the interface between the ICT system and the information system. At its most basic it is an assessment of how easy it is to enter data and retrieve data from the ICT system. Utility is primarily a feature of the relationship between the information system and the human activity system. It constitutes some assessment of the contribution the information system makes to human activity, particularly to improvements in decision-making.

4.8 DATA MODELS AND INFORMATION SYSTEMS PLANNING

Because of its central importance for organisations, data has to be planned for and administered. The major tool employed in data planning and data administration is the application data model.

Application data models can be produced on at least three levels (see Figure 4.4):

- *Corporate data models* – specifying data requirements for a whole organisation
- *Business area data models* – specifying data requirements for a particular business area
- *Application data models* – specifying data requirements for a particular information systems application

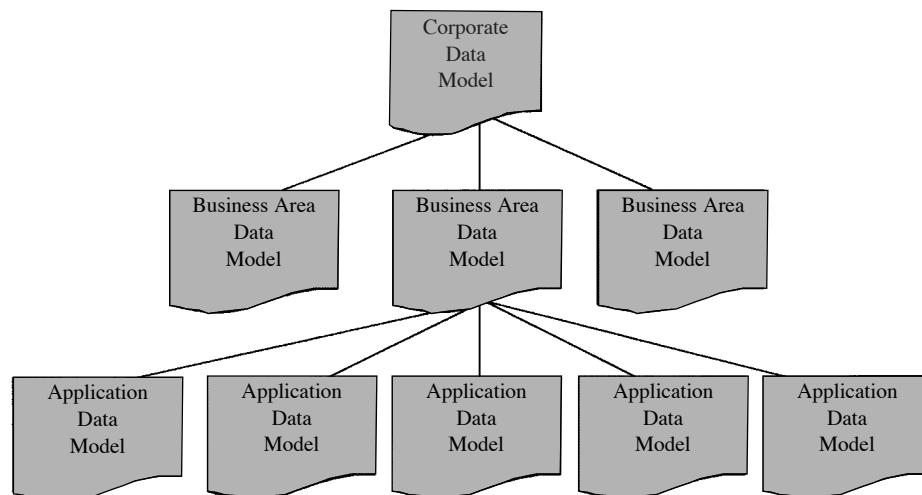


Figure 4.4 Corporate, business area and application database models.



Example → Consider the case of a university setting (Figure 4.5). Traditionally, to support the activities of the university, given application data models may have been produced to document the data requirements relevant to a particular information system such as a library information system or a student enrolment system.

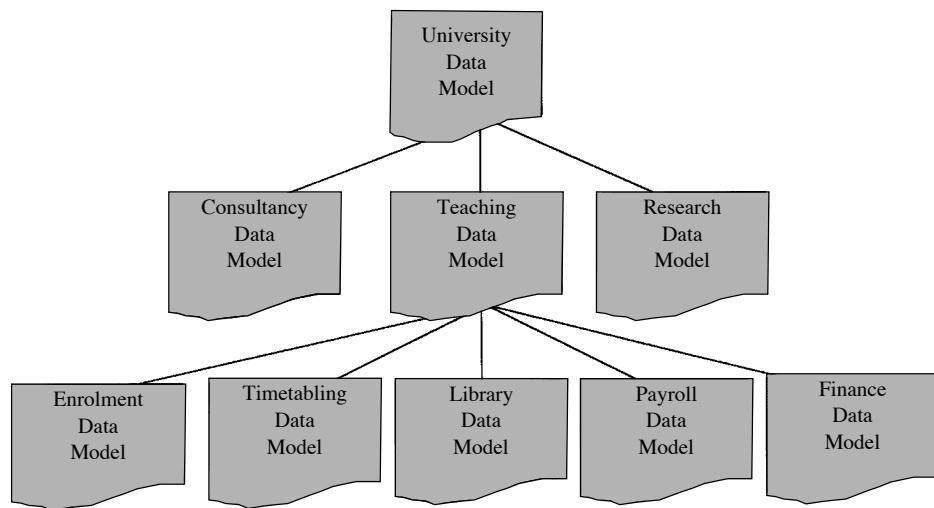


Figure 4.5 Example in university setting.

At a higher level a data model may be produced to integrate the data requirements relevant to a particular business area. In a university setting, for instance, we might develop a data model relevant to the key 'business' areas of teaching, research and consultancy.

At the highest level the university may have developed a corporate data model to integrate the data requirements for the entire organisation. This may either be a unification of the business area data models or a summary of their key elements (Chapter 21).

The development of application, business area and corporate data models may be critical elements of planning for information systems. Information systems planning is the process of defining an informatics infrastructure for some organisation, and developing strategic and operational plans for the delivery of this infrastructure.

An informatics infrastructure can be envisaged as having three inherent layers corresponding to the distinctions raised between information, information systems, and information and communications technology described above:

- *Information infrastructure.* This consists of activities involved in the collection, storage, dissemination and use of information within the organisation

- *Information systems infrastructure.* This consists of the information systems needed to support organisational activity in the areas of collection, storage, dissemination and use
- *ICT infrastructure.* This consists of the hardware, software, data, communication facilities and ICT knowledge and skills available to the organisation

4.9 CONTEMPORARY DATABASE TYPES

As we have indicated above, databases are arguably the most important contemporary component of an ICT system. In such systems databases serve three primary purposes: as operational tools, as tools for supporting decision-making and as tools for deploying data around the organisation (Figure 4.6).

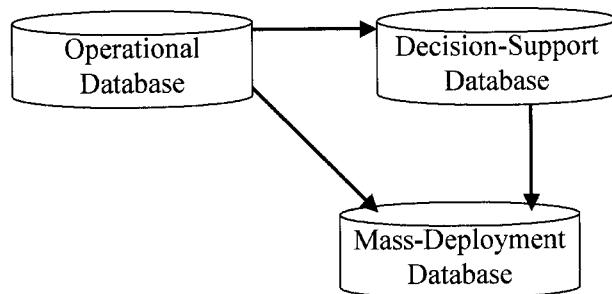


Figure 4.6 Operational, decision-support and mass-deployment databases.

4.9.1 OPERATIONAL DATABASES

Such databases are used to collect operational data. Operational databases are used to support standard organisational functions by providing reliable, timely and valid data. The primary usages of such databases include the creating, reading, updating and deleting of data – sometimes referred to as the CRUD activities.

Example → In terms of a university, an operational database will probably be needed to maintain an ongoing record of student progression.

4.9.2 DECISION-SUPPORT DATABASES

Such databases are used as data repositories from which to retrieve information for the support of organisational decision-making. Such databases are read only databases. They are designed to facilitate the use of query tools or custom applications.



Example ➔ In terms of a university, a decision-support database may be needed to monitor recruitment and retention patterns among a student population.

4.9.3 MASS-DEPLOYMENT DATABASES

Such databases are used to deliver data to the desktop. Generally such databases are single-user tools running under some PC-based DBMS such as Microsoft Access. They may be updated on a regular basis either from production or decision-support databases.

Example ➔ In terms of a university, a mass-deployment database will be needed by each lecturer to maintain an ongoing record of student attendance at lectures and tutorials.

Ideally we would like any database system to fulfil each of these purposes at the same time. However, in practice, medium to large-scale databases can rarely fulfil all these purposes without sacrificing something in terms of either retrieval or update performance. Many organisations therefore choose to design separate databases to fulfil production, decision-support and mass-deployment needs, and to build necessary update strategies between each type.

4.10

USERS OF DATABASE SYSTEMS

It is possible to describe three common types of users of database systems – end-users, database administrators (DBAs) and systems developers. Each type of user will demand a different type of interface to the database system:

- *End-users.* Most users in organisations will be unlikely to know they are using a database system. This is because they will be using a database system indirectly through some ICT system. Some sophisticated end-users will be able to access database systems directly through employing end-user tools such as visual query interfaces (Chapter 24)
- *Database administrators/data administrators.* DBAs will be concerned with creating and maintaining databases for various applications. Hence, most DBMS provide a range of tools for DBAs, particularly in the area of data control (Chapter 25)
- *System developers.* Developers of ICT systems will need to integrate database systems with the wider functions of the ICT system. Various tools such as application programming interfaces are available for this purpose (Chapter 26)



4.11 CASE STUDY: ELECTRONIC VOTING

The UK is a parliamentary democracy and hence is reliant on an effective electoral system – a human activity system. General elections are held after Parliament has been dissolved either by Royal Proclamation or because the maximum term of five years for a government term of office has been reached. The decision as to when a general election is to be held is taken by the Prime Minister.

For parliamentary elections the UK is divided into 659 constituencies – 18 in Northern Ireland, 40 in Wales, 72 in Scotland and 529 in England. For so-called general elections to Parliament, the UK currently employs a first-past-the-post system, sometimes described as a single member plurality system. In this system each voter uses a single poll card to cast a single ballot for one constituency candidate and each constituency elects a single Member of Parliament on the basis of the majority of the votes cast. Each candidate in the general election generally represents a single political party, and the party with the most seats in Parliament (not necessarily the most votes) will usually become the next government of the nation.

The key inputs into the electoral system are ballot poll cards and ballot papers provided by the key agents, voters. The key outputs are a set of election results provided for each constituency. The key control process is one of electoral monitoring which establishes guidance on expected electoral practice and monitors the actual election to determine any deviation from such practice. The environment of the electoral system is the political system of the UK.

The electoral process relies on an information system for managing the collection, storage and manipulation of votes. Interestingly, the system has remained relatively unchanged since the ballot act of 1872 and hence relies on very little modern ICT. The information system is thus a paper-based system and the key activities are described below:

- *Registration of candidates.* Candidates in a UK general election must be over 21 and must register for election for a given constituency
- *Registration of voters.* Registration used to be possible only at set times during the year. Nowadays a person can register to vote at any time prior to the holding of an election. To vote in a UK general election a person must be a citizen of the UK, be over the age of 18 and not excluded on grounds such as being in prison, detained in a psychiatric hospital or a member of the House of Lords. Normally voters would be expected to attend in person at a specified polling booth in order to vote. If they are able to supply a valid reason, a person may be entitled to appoint a proxy (some other person) to attend for them. Postal voting was introduced for the first time in the UK general election of 2001. In this election some 1.4 million people out of a total electorate of 44 million voted in this manner
- *Production of electoral list and correspondence.* Each local authority in the UK is tasked with maintaining an electoral register. From this register each



authority needs to produce an electoral list for each of the designated polling stations in its area. Electors receive various items of documentation through the post, the main item being a polling card detailing the name and address of the voter plus the date of the election and address of the designated polling station. Interestingly, the poll card also contains a serial number which can be used to track against electoral lists

- *Authentication.* Voters typically turn up at their indicated polling station. They produce their poll card for inspection and this is checked against the electoral list. If the elector is correctly authenticated in this manner, he or she is handed a ballot paper on which the candidates for the constituency are listed. Each ballot paper is stamped by polling staff to indicate that it has been issued appropriately
- *Voting.* The elector enters a polling booth and chooses one entry against the list of candidates by placing an 'X' in an appropriate box. The ballot paper is then folded and posted in a sealed ballot box
- *Transfer of ballot boxes.* Most elections in the UK are held on a weekday (typically a Thursday) and voters are only allowed to vote during the set period of 0700 to 2200 hours on that day. At the end of the voting period all ballot boxes are collected and taken to a central counting centre
- *Validation.* The total number of persons crossed off against the electoral list is usually cross-checked against the total number of ballots cast for each ballot box/polling station
- *Counting.* A team of workers then count the ballots by hand into piles of 50 by candidate. Recounts are normally only ordered if candidate totals are close. Then only the bundles are normally counted unless candidates request that bundles be checked
- *Publishing.* Results are announced at each counting centre and communicated to national electoral headquarters

4.12



SUMMARY

- An information system is a system which provides information for some organisation or part of an organisation
- Information systems support human activity systems. Human activity systems consist of people, conventions and artefacts designed to serve human needs
- Information technology provides means of constructing aspects of modern-day information systems or ICT
- An information technology system is made up of a number of subsystems or layers: interface subsystem, rules subsystem, transaction subsystem and data subsystem
- Data models can be produced on at least three levels: corporate data models, business area data models and application data models

- In contemporary applications database systems tend to support three main purposes: production databases, decision-support databases and mass-deployment databases
- Three main types of users – end-users, database administrators and system developers – demand different interfaces to the database system

4.13 ACTIVITIES

1. In terms of the academic domain discussed in Chapter 1, give an example of a datum and a piece of information relevant to the domain.
2. Examine some report known to you in terms of the distinctions between data, information and knowledge.
3. What human activity systems are relevant to a university domain?
4. Identify some information systems of relevance to a university domain.
5. Investigate an organisation known to you and determine whether it has developed either a corporate data model or a series of business area data models.
6. Describe the ways in which data modelling contributes to the IS planning process, and illustrate with a description of its relevance to a university domain.
7. Distinguish between a production database and a decision support database. What sort of production databases are relevant to a university domain? What decision support databases are relevant?

4.14 REFERENCES

- Beynon-Davies, P. (2002). *Information Systems: An Introduction to Informatics in Organisations*. Basingstoke, Palgrave (formerly Macmillan).
- Checkland, P. (1987). *Systems Thinking, Systems Practice*. Chichester, John Wiley.

CHAPTER 5

DATABASE SYSTEMS AND ELECTRONIC BUSINESS

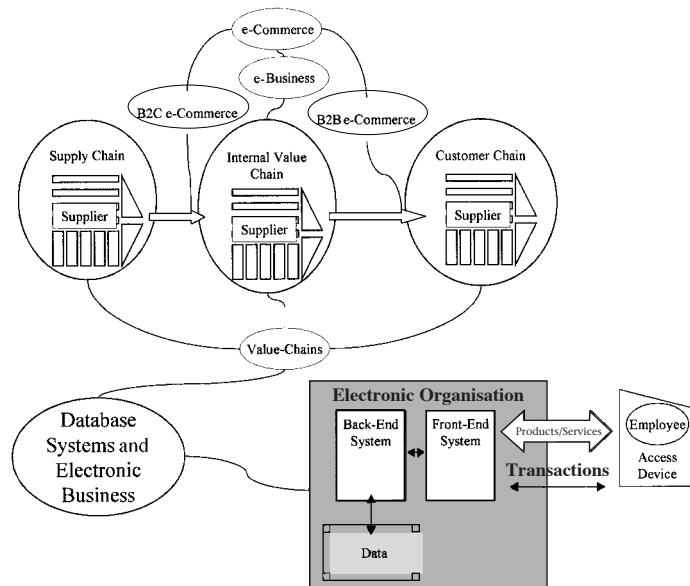
I find it rather easy to portray a businessman. Being bland, rather cruel and incompetent comes naturally to me.

John Cleese (1939-)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define electronic business and electronic commerce
- Explain how ICT aids restructuring of the internal and external value-chains of businesses
- Describe the typical ICT infrastructure of the business
- Identify the place of databases and database systems in the ICT infrastructure





5.1 INTRODUCTION

Much has been recently published on the topics of e-Business and e-Commerce. Business can either be considered as an entity or as the set of activities associated with a commercial organisation. Electronic business or e-Business might be defined as the utilisation of ICT to support all the activities of business. Commerce constitutes the exchange of products and services between businesses, groups and individuals. Commerce or trade can hence be seen as one of the essential activities of any business. Electronic commerce or e-Commerce focuses on the use of ICT to enable the external activities and relationships of the business with individuals, groups and other businesses.

Database systems have a crucial part to play in the technical infrastructure required to support e-Business. First, we consider the ways in which e-Business is redefining the value-chains of business. Second, we examine the technical infrastructure for e-Business. Third, we examine the place of database systems within this infrastructure.

5.2 e-BUSINESS, e-COMMERCE AND I-COMMERCE

The organisational activity within the electronic economy is generally referred to as electronic business (e-Business) or electronic commerce (e-Commerce) (Beynon-Davies, 2003). Electronic business is a superset of electronic commerce. In turn e-Commerce can be considered a superset of Internet commerce or I-Commerce:

- *e-Business*. Business can either be considered as an entity or as the set of activities associated with a commercial organisation. We treat e-Business as the utilisation of ICT to support all the activities of business
- *e-Commerce*. Commerce constitutes the exchange of products and services between businesses, groups and individuals. Commerce or trade can hence be seen as one of the essential activities of any business. e-Commerce focuses on the use of ICT to enable the external activities and relationships of the business with individuals, groups and other businesses
- *I-Commerce*. Internet commerce is the use of Internet technologies to enable e-Commerce. Such technologies are becoming the key standards for intra and inter-organisational communication

These distinctions allow us to delineate the use of ICT to enable communication and coordination between the internal stakeholders of the business (intra-business e-Business), by using ICT to enable communication and coordination with external actors (e-Commerce).

Examples  Many businesses have been using ICT to improve the efficiency and effectiveness of internal activities for some three to four decades. Many large companies have been using ICT to enable the flow of transactions with their business partners.



Recently, with the rise of the Internet, electronic commerce has become available to all classes and sizes of company.

Many companies are now using ICT to connect disparate corporate information systems. This enables management to gain an accurate picture of the day-to-day performance of the organisation. Companies are also selling goods and services on a global scale using corporate Web-sites.

5.3 VALUE-CHAINS AND e-BUSINESS

Fundamentally, e-Business and e-Commerce focus around organisational value-chains. Porter (1985) offers a template for considering an organisation's key human activity systems. This is a generic model of an organisation known as the value-chain. In this view, organisations are seen as social institutions that deliver value to stakeholders through defined activities. An organisation's value-chain is a series of interdependent activities that delivers value (typically products or services) to stakeholders such as customers.

Porter's notion of the value-chain focuses on the internal processes of some organisation. Two other chains of value critical to the competitive environment assume significance for most organisations: the supply chain and the customer chain. They are both chains in the sense that both customers and suppliers will typically be other organisations. Hence an economic system will be composed of a complex network of such chains.

The trend to use ICT to restructure aspects of the internal value-chain of organisations has been ongoing for a number of decades. Recently, increasing interest has been expressed in using ICT to re-engineer aspects of the organisation customer and supply chain. This is illustrated in Figure 5.1.

5.4 RESTRUCTURING THE INTERNAL VALUE-CHAIN

Intra-business e-Business refers to the changes to the internal value-chain that may be affected by information and communication technology. Internet technologies and standards have been used to build corporate intranets. These are internal Internets only accessible to authorised users within the company.

Examples Internet standards have been particularly used to:

- *Standardise interfaces.* Web browsers have now become one of the most popular ways of accessing interfaces to infrastructure information systems
- *Share knowledge.* The creation and maintenance of corporate intranets have enabled organisations to share knowledge across the enterprise
- *Integrate information systems.* Corporate intranets have been used as ways of integrating information across diverse infrastructure systems

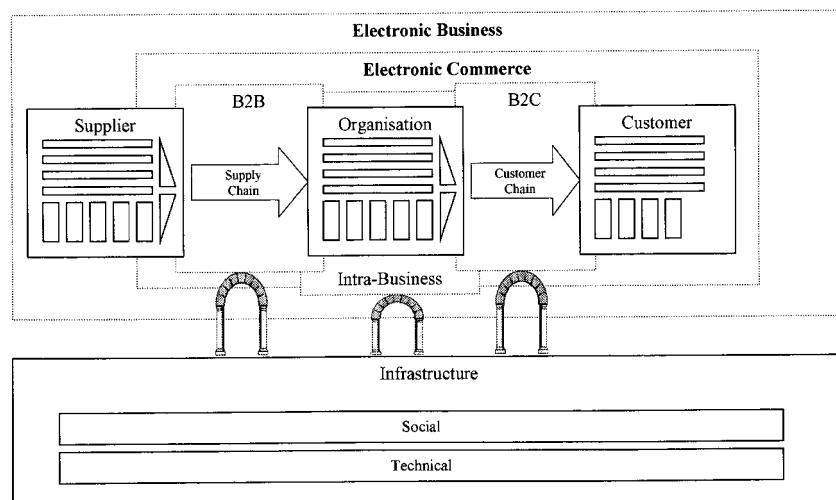


Figure 5.1 e-Business and e-Commerce.

5.5 RESTRUCTURING THE SUPPLY CHAIN

ICT has supported the supply chain for a number of decades in that businesses have used electronic records to transfer documentation in terms of standards in the area of electronic data interchange or EDI. More recently the Internet and its associated technologies have been used to transfer data between companies engaged in a trading relationship.

The form of electronic commerce associated with restructuring the supply chain is generally referred to as business-to-business (B2B) e-Commerce.

Examples I-Commerce is being used to impact upon the supply chain in ways that include:

- *Procurement.* Companies are setting up Web-sites for suppliers to bid against particular requirements. Intermediary organisations are also offering supply-side services such as e-mails and Internet auctions
- *Invoicing.* Extensible markup language (XML) – a version of HTML (HyperText Markup Language) – is being piloted as a replacement to Electronic Data Interchange (EDI) standards for electronic document transfer between organisations (Chapter 39). Using this technology, standard documents such as invoices can be rapidly transferred between organisations, and it is likely that it will provide a medium for organisations of all kinds to conduct their business transactions electronically in the near future
- *Inventory.* Some companies have established relationships with their suppliers that provide a form of access to their extranet. An important application in this area is enabling suppliers to access up-to-the-minute information on the state of an organisation's inventory of a product it supplies. If stock levels are low, it can automatically resupply the organisation concerned

**5.6****RESTRUCTURING THE CUSTOMER CHAIN**

It is only relatively recently that ICT has impacted significantly upon the customer chain, as it is only comparatively recently with the rise of technologies underlying the Internet (I-Commerce) that direct connections between customers and businesses have been made possible.

The form of electronic commerce associated with restructuring the customer chain is generally referred to as business-to-customer or business-to-consumer (B2C) e-Commerce.

Examples

→ I-Commerce is being used to impact upon the customer chain in ways which include:

- *Advertising*. Most medium-to-large companies now have a presence on the Internet. This ranges from merely putting up information about an organisation to more sophisticated versions of on-line product and service catalogues
- *Marketing*. Companies are now contacting potential customers via e-mail rather than the traditional forms of telephone and postal services
- *Sales*. Organisations have begun to offer the facility to purchase their goods and services over the Internet. Sophisticated versions of this form of application will even accept payments for such goods and services over the Internet

5.7**INTRANET AND EXTRANET**

Internet technology, such as WWW (World-Wide-Web), has become particularly popular because of the way it has established readily available 'open' standards for electronic communication. It is therefore not surprising that many organisations are beginning to use Internet technology for communications applications internal to the organisation. An intranet involves using Internet technology within the context of a single organisation. At its most basic, it involves setting up a Web service for internal communications and coordination. At its most sophisticated, it involves using Web interfaces to core corporate applications such as corporate-wide database systems. Typically an intranet is connected to the wider Internet through a so-called firewall. The firewall constrains the types of information that can be passed into and out of an organisation's intranet.

Whereas an intranet resides behind a firewall and is only accessible to the members of an organisation, an extranet provides a certain level of access to an organisation's Web-based information to outsiders. Extranets are becoming particularly popular as they enable electronic connections to be made to an organisation's customers and suppliers.

5.8 ICT INFRASTRUCTURE FOR ELECTRONIC BUSINESS

Organised activity of whatever form requires infrastructure. Infrastructure consists of systems of social organisation and technology that support human activity. The arches on Figure 5.1 are meant to indicate that technical and social infrastructure support activity in key areas of e-Business.

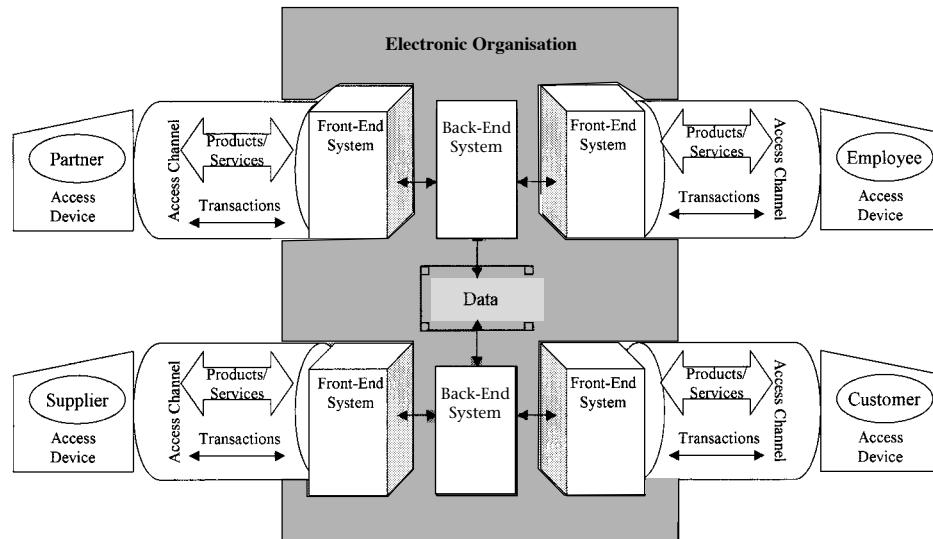


Figure 5.2 The technical infrastructure for e-Business.

Figure 5.2 illustrates the key components of the ICT or technical infrastructure for e-Business. The key message being promoted is that ICT is an enabler for organisational change focused around the redesign of the delivery of services and products to key stakeholders – customers, suppliers, partners and employees. Hence ICT is seen to offer the potential for more effective and efficient delivery of value along supply, customer and internal value-chains.

ICT is being promoted within both the public and private sector as a means of improving the efficiency, effectiveness and efficacy of the delivery of services and products to internal and external stakeholders.

A key distinction is essential between products, services and transactions. Products and/or services (represented as broad arrows on Figure 5.2) are typically the value delivered to the stakeholder. Hence, products and/or services are typically the end-points of processes or human activity systems undertaken by organisations (Checkland, 1987).

Information is needed to support most human activity systems, particularly in terms of transactional data. Transactions (indicated as narrow arrows on Figure 5.2) are ways of recording the delivery of products and services, and hence are the essential raw data for modelling and evaluating organisational performance. Performance management systems cannot work effectively within organisations without transactional data. Reducing the costs associated



with the administration of transactions is also seen as the typical way of introducing cost savings with ICT.

The objective of redesigning human activity systems with ICT to support electronic delivery of products and services typically involves:

5.8.1 INVESTIGATING AND IMPLEMENTING VARIOUS ACCESS MECHANISMS FOR DIFFERENT STAKEHOLDERS

In terms of interaction with the customer, face-to-face contact and telephone conversation are two of the most commonly used mechanisms for accessing an organisation's services and/or products. However, with an eye on the longer-term, organisations both in the public and private sectors are either implementing or investigating access mechanisms that allow customers to interact with the organisation remotely using ICT. An access mechanism typically is composed of some access device and an access channel. Typical access devices being supported are the Internet-enabled personal computer (PC) and interactive digital television (iDTV). There are a number of advantages to both the organisation and its external stakeholders in promoting use of such access mechanisms. For instance, an organisation may be able to provide access to its services and products 24 hours a day, 365 days a year.

5.8.2 PROVIDING EFFECTIVE DELIVERY OF INTANGIBLE GOODS AND SERVICES

Certain goods are primarily information-based or intangible in nature. Key examples here are software and music. As such they are prime candidates for electronic delivery. This enables certain organisations to replace traditional physical distribution channels with electronic distribution. In this scenario the access channel on Figure 5.2 also becomes a distribution channel.

Many services are also intangible in nature. Key examples here are insurance, legal advice, news reports and monetary transfers. One would expect that such business areas would be prime candidates for electronic service delivery.

5.8.3 CONSTRUCTING FRONT-END ICT SYSTEMS TO MANAGE CUSTOMER INTERACTION

The technological infrastructure for modern front-end systems relies on two critical technologies: the Internet and the Web.

Currently, the Internet is a set of interconnected computer networks distributed around the globe and can be considered on a number of levels. The base infrastructure of the Internet is composed of packet-switched networks and a series of communication protocols. On this layer runs a series of applications such as electronic mail (e-mail) and more recently the World-Wide-Web – the Web for short. The Web is effectively a set of standards for the representation and distribution of hypermedia documents over the Internet. A hypermedia document is made up of a number of chunks of content such as text, graphics



and images connected together with associative links called hyperlinks. The Web has become a key technology for constructing the front-end ICT systems of organisations.

One of the most critical of such front-end ICT systems is the organisation Web-site. The term *Web-site* is generally used to refer to a logical collection of Web documents normally stored on a Web server. Such sites now constitute major ways of providing electronic delivery to customers.

Because of the increasing use of technologies such as the Internet and the Web by stakeholders such as customers, major investment is currently being undertaken by companies to increase levels of interactivity on their Web-sites. The aim for many companies is to provide fully transactional Web-sites in which customers can undertake a substantial proportion of their interaction with an organisation on-line.

5.8.4 RE-ENGINEERING OR CONSTRUCTING BACK-END ICT SYSTEMS

Effective back-end ICT infrastructure is critical to organisational success. The back-end ICT infrastructure of the organisation will particularly manage the operational data of the organisation. Hence, database systems are critical to back-end infrastructure.

A key focus within the e-Business agenda is on re-engineering service delivery around the customer. This requires the effective integration and inter-operability of back-end ICT systems. Hence, for example, when customers enter personal details, such as their name and address, into one system this information should ideally be available to all other systems that need such data.

5.8.5 ENSURING FRONT-END/BACK-END ICT SYSTEMS INTEGRATION

To enable fully transactional Web-sites, the information presented to the user needs to be updated dynamically from back-end databases. Also, the information entered by customers needs to update company information systems effectively. This demands integration and interoperability of front-end and back-end systems within the ICT infrastructure.

5.8.6 ENSURING SECURE TRANSACTIONS ALONG COMMUNICATION CHANNELS

For effective e-Commerce people must trust electronic delivery. A major part of such trust is focused in ensuring the privacy of electronic data held in ICT systems and the transactions flowing between ICT systems, particularly payments. A number of technologies now exist to ensure such security, including encryption and digital certificates.

Example In the Case Study at the end of this chapter we consider how each of these principles of electronic service delivery is applicable to local government organisations.



5.9 APPLICATIONS OF DATABASE SYSTEMS IN THE ICT INFRASTRUCTURE

The discussion in the previous section should make clear that database systems play a primary role in the back-end ICT infrastructure of e-Business. We examine some of the typical applications of database systems in this role in this section.

5.9.1 DATA WAREHOUSES/DATA MARTS

A data warehouse (Chapter 40) is a type of contemporary database system designed to fulfil decision-support needs. However, a data warehouse differs from a conventional decision-support database in a number of ways:

- A data warehouse is likely to hold far more data than a decision-support database
- The data stored in a warehouse is likely to have been extracted from a diverse range of ICT systems, only some of which may utilise database systems
- A data warehouse is designed to fulfil a number of distinct ways (dimensions) in which users may wish to retrieve data

A data mart is a small data warehouse. A data mart is also likely to store data representing a particular business area rather than representing data applicable to the entire organisation.

5.9.2 ON-LINE ANALYTICAL PROCESSING AND DATA MINING

Two application areas are normally discussed in association with data warehousing and data marts:

- *On-line analytical processing.* Modern database applications, such as market analysis and financial forecasting, require access to large databases for the support of queries which can rapidly produce aggregate data. Such applications are frequently called analytical on-line processing applications, or OLAP for short. OLAP systems use multi-dimensional data structures to store data and relationships. OLAP (Chapter 41) comprise the dynamic synthesis, analysis and consolidation of large volumes of multi-dimensional data
- *Data mining.* To gain benefit from a data warehouse, the data patterns resident in the large data sets characteristic of such applications need to be extracted. As the size of a data warehouse grows the more difficult it is to extract such data using the conventional means of query and analysis. Data mining (Chapter 42) involves the use of automatic algorithms to extract such data. Data mining comprises the process of extracting hidden patterns from large databases and using it to make decisions critical to some organisation



Example Consider the case of a supermarket chain. At the operational level, sales data is recorded in each supermarket in the chain at checkouts using electronic point-of-sale devices. This data allows administrative staff to record the amount of each type of product sold and, in turn, triggers decisions as to the amount of stock to reorder.

This sales data may be a major data source for a data warehouse, perhaps sited at the headquarters of the supermarket chain. This enables headquarters staff to monitor nationally or perhaps internationally its sales performance in certain areas, and helps staff make decisions as to what it should be selling and for what price.

However, the sales data is likely to be combined with data from other sources such as customer data, collected perhaps through the supermarket chain's loyalty card scheme. Associating sales data with customer data in its data warehouse may provide the chain with important information about the purchasing patterns of its customers. This may enable the chain to proactively plan activities in relation to particular customer groups or in terms of new business opportunities such as financial services.

5.9.3 DATABASE SYSTEMS AND THE WEB

Database systems have been impacted upon by developments in Internet technology through so-called Web-enabled database applications (Chapter 43). Users access such applications through a Web browser on their desktop system. Browser software accesses and displays Web pages sited on a Web server identified by a universal resource locator (URL). A Web page is basically a document, a text file with HTML (HyperText Markup Language) codes inserted. HTML commands instruct the browser how to display the specified text file.

A user requests a service by specifying the URL. The specified Web page is then transferred across the network to the desktop, allowing the browser to display the page.

Traditional Web pages are static entities in the sense that links between textual information is hard-coded into the text. There are hence numerous problems experienced in maintaining many interconnected Web pages.

For this reason there are many modern developments in dynamic Web pages. In a dynamic Web page, data on the Web page is dynamically updated by a database. This makes for the easier development and maintenance of larger-scale Web applications.

5.9.4 UNIVERSAL SERVERS

Traditional database applications support structured data such as numbers or character strings. Newer applications such as Web-enabled database applications demand the ability to store and manipulate more complex data types such as image data, audio data and video data.



Many contemporary DBMS are hence casting themselves as so-called universal servers. The universal server approach involves the extension of DBMS to support both traditional and non-traditional data. Non-traditional data is generally supported through user-defined data types (UDTs) and user-defined functions (UDFs). UDTs, also known as abstract data types, define non-standard data structures. UDFs permit the users of a database to alter and manipulate UDTs.

One of the more recent activities in this area is the use of database systems to store semi-structured data. Semi-structured data (Chapter 39) lies between the poles of structured and unstructured data. In structured data the structure is provided by some schema. Unstructured data lacks any schema and the structure is not apparent from any repetition of instances of such data. Semi-structured data also lacks a schema but structure is apparent from the way in which data is presented. Hence, the meaning of any data in a set of such data is determined by its position.

5.10

CASE STUDY: ELECTRONIC LOCAL GOVERNMENT IN THE UK

In the UK, local government is organised as a three-tier system: unitary authorities, county authorities and district authorities. Unitary authorities were introduced during the mid-1990s and take on the role of both county and district councils in delivering services to their local community.

Unitary authorities are responsible for a wide range of services, including education, environmental health, planning, housing, personal social services, waste management and disposal, highways, libraries, recreation, cemeteries and crematoria. Authorities also assume some overseeing functions in relation to the fire and police services.

The UK Prime Minister announced in 1997 that by 2002, 25% of dealings with government should be able to be carried out by the public electronically. These targets were later revised in the Modernising Government White Paper. The UK government has now set a target that by 2005, all (100%) government services that can be delivered electronically will be delivered electronically.

It is estimated that a typical unitary authority will have 700 different services. Some of these services will be primarily information-based. Major examples here are maintaining a land and electoral register and collecting revenues such as the council tax. For a number of such information-based services national standards and systems are being developed and promoted among authorities within the UK. These include:

- *Local Authorities Secure Electoral Register* (LASER). The LASER project aims to provide electoral registers that are joined up, maintained and managed locally, and can then be accessible on a national level to authorised users. LASER aims to draw together all of the locally held registers of electors and make them available nationally to support e-Voting
- *National Land Information Service* (NLIS) The NLIS is an on-line, one-stop shop that delivers land and property-related information from source data



providers to conveyancers and home buyers. At present, NLIS provides access to information held and maintained by local authorities, the Land Registry and the Coal Authority. Negotiations are now under way towards including searches relating to the water service companies and the Environment Agency

- *National Land and Property Gazetteer* (NLPG). The NLPG is a single, comprehensive, up-to-date list of addresses. Local authorities are essential participants because they start the process by naming streets and numbering buildings. The NLPG requires local authorities to convert their existing lists of addresses into a fully consistent national information system, held electronically, constructed to common standards, and based on unique property reference numbers for each property or piece of land. The NLPG is reliant on authorities producing and maintaining Local Land and Property Gazetteers (LLPGs) to a common standard

Other services, such as waste disposal, although not information-based, will nevertheless rely on effective and efficient transmission of information to stakeholders. Hence, for example, effective waste disposal is reliant on the provision of accurate collection times to authority customers.

The objective of expanding electronic service delivery (ESD) among local authorities in the UK will involve:

- *Investigating and implementing various access mechanisms and channels for different stakeholders.* Consultations recently conducted by authorities suggest relatively low levels of uptake of ESD are to be expected in the short-term. However, with an eye on the longer-term, most authorities are either implementing or investigating multiple-channel access centres that allow customers to interact with the authority using devices such as the Internet-enabled personal computer (PC) and even interactive digital television (iDTV). There is some evidence of increasing interest in access to government services through ICT (KPMG, 2002) that lends support to this strategy
- *Re-engineering or constructing front-end systems to manage ESD.* Major investment is currently being undertaken by authorities to increase levels of interactivity on their Web-sites. The aim for many authorities is to provide fully transactional Web-sites designed around so-called life episodes such as registering births and deaths and adding names to an electoral register
- *Re-engineering or constructing back-end systems to deliver the data needed by front-end systems.* To enable fully transactional Web-sites, the information presented needs to be updated dynamically from back-end databases. Also, the information entered by customers needs to update authority information systems effectively. Here, initiatives like LASER are critical
- *Re-engineering being not just technological change but also organisational change.* A key focus within the electronic local government agenda is on re-engineering service delivery around the customer. Hence, for example, when



customers enter personal details, such as their name and address, into one system for council tax purposes, this information should ideally be available to all other systems that need such data

5.11**SUMMARY**

- Business can either be considered as an entity or as the set of activities associated with a commercial organisation. Commerce constitutes the exchange of products and services between businesses, groups and individuals. Commerce or trade can hence be seen as one of the essential activities of any business
- Electronic business or e-Business might be defined as the utilisation of information and communication technologies to support all the activities of business
- Electronic commerce or e-Commerce focuses on the use of ICT to enable the external activities and relationships of the business with individuals, groups and other businesses
- Internet commerce or I-Commerce is the use of Internet technologies to enable e-Commerce
- Intra-business e-Business refers to the changes to the internal value-chain that may be affected by information and communication technology
- The form of electronic commerce associated with restructuring the supply chain is generally referred to as business-to-business (B2B) e-Commerce
- The form of electronic commerce associated with restructuring the customer chain is generally referred to as business-to-customer or business-to-consumer (B2C) e-Commerce

5.12**ACTIVITIES**

1. In a company or a public-sector organisation (such as a higher education institution) known to you, attempt to identify when it started to use ICT and for what purpose.
2. Sales and accounting systems are frequently integrated in business. Identify another example of two back-end business systems in a private or public-sector organisation that you would expect to be integrated. For instance, what back-end systems would one expect to find in a university?
3. Identify the customer and supply chains in a public-sector organisation such as the Inland Revenue or Benefits Agency.
4. Does your organisation (for instance, a university) have an intranet? If so, what is it used for? If not, would an intranet be useful and for what purpose?
5. Attempt to map the technical infrastructure for e-Business described in this chapter onto an organisation known to you, such as a university.

5.13  REFERENCES

- Beynon-Davies, P. (2003). *e-Business*. Basingstoke, Palgrave Macmillan.
- Checkland, P. (1987). *Systems Thinking, Systems Practice*. Chichester, John Wiley.
- KPMG (2002). *Is Britain on Course for 2005?* London, KPMG Consulting.
- Porter, M.E. (1985). *Competitive Advantage: Creating and Sustaining Superior Performance*. New York, Free Press.

CHAPTER 6

DATA MANAGEMENT LAYER

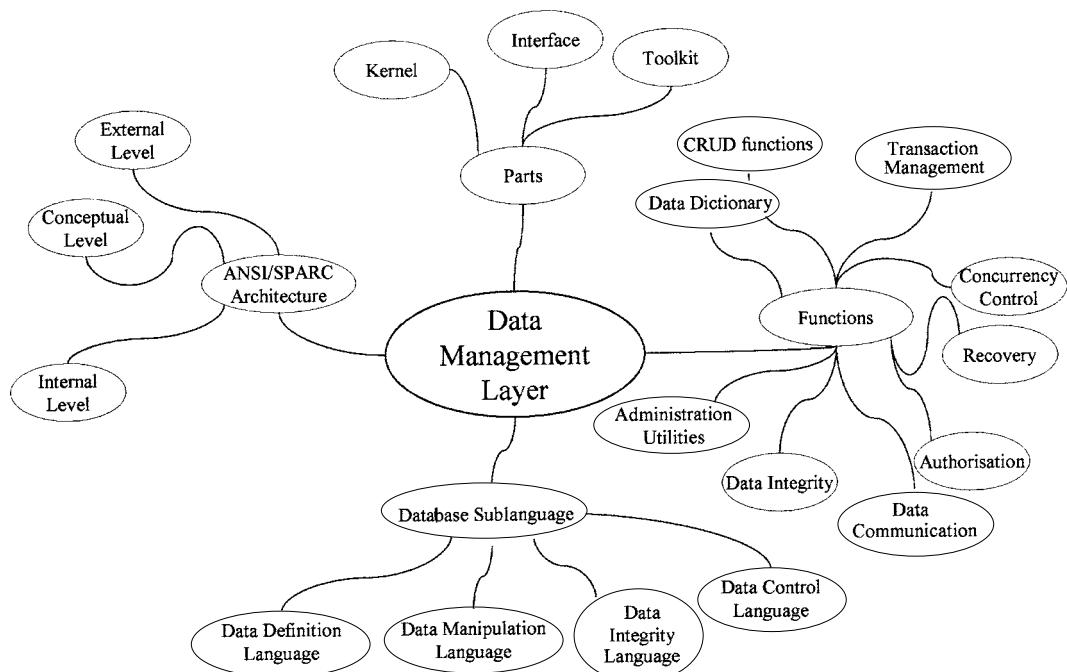
Wherever you come near the human race there's layers and layers of nonsense.

Thornton Wilder (1897–1975)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Describe the three-level architecture of a DBMS
- Highlight the key functions in a DBMS
- Provide an overview of the interface to a DBMS
- Provide an overview of the structure of the kernel of a DBMS



6.1 INTRODUCTION

In this chapter we examine the key features of the data management layer of an information system. The data management layer is normally made up of the facilities provided by some DBMS (Ramakrishnan and Gehrke, 2003). We examine an architecture proposed for the data management layer and some of the key functions of a DBMS in terms of the distinctions between DBMS kernel, interface and toolkit.

6.2 THE ANSI/SPARC ARCHITECTURE

A DBMS can be considered as a buffer between application programs, end-users and a database designed to fulfil features of data independence. In 1975 the American National Standards Institute Standards Planning and Requirements Committee (ANSI-SPARC) proposed a three-level architecture for this buffer (ANSI, 1975). This architecture identified three levels of abstraction. These levels are sometimes referred to as schemas or views (see Figure 6.1):

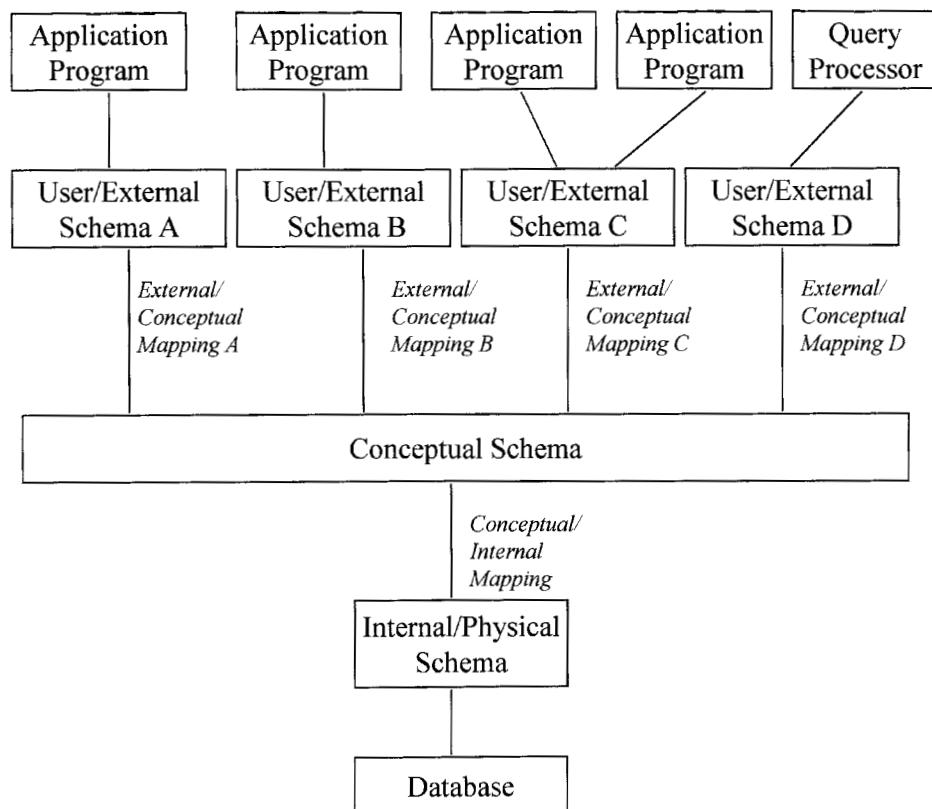


Figure 6.1 ANSI/SPARC architecture.



- *The external or user level.* This level describes the users' or application program's view of the database. Several programs or users may share the same view
- *The conceptual level.* This level describes the organisations' view of all the data in the database, the relationships between the data and the constraints applicable to the database. This level describes a logical view of the database – a view lacking implementation detail
- *The internal or physical level.* This level describes the way in which data is stored and the way in which data may be accessed. This level describes a physical view of the database

Each level is described in terms of a schema – a map of the database. The three-level architecture is used to implement data independence through two levels of mapping: that between the external schema and the conceptual schema, and that between the conceptual schema and the physical schema. Data independence comes in two forms:

- *Logical data independence.* This refers to the immunity of external schemas to changes in the conceptual schema. For instance, we may add a new data item to the conceptual schema without impacting on the external level
- *Physical data independence.* This refers to the immunity of the conceptual schema to changes made to the physical schema. For instance, we may change the storage structure of data within the database without impacting on the conceptual schema

6.3

KERNEL, INTERFACE AND TOOLKIT

In Chapter 3 we made the analogy between a DBMS and a filing cabinet. However, modern DBMS are a lot more than computerised filing cabinets. It is useful to distinguish between the idea of a kernel, the idea of a toolkit and the idea of an interface in the context of DBMS (see Figure 6.2).

6.3.1 KERNEL

By DBMS kernel we mean the central engine which operates core data management functions such as most of those defined below. These functions are discussed in more detail in Part 7.

6.3.2 TOOLKIT

By DBMS toolkit we mean the vast range of tools which are now either packaged as part of a DBMS or are provided by third-party vendors. For instance, spreadsheets, fourth generation languages, performance monitoring facilities etc. Various parts of the database toolkit are discussed in Part 6.

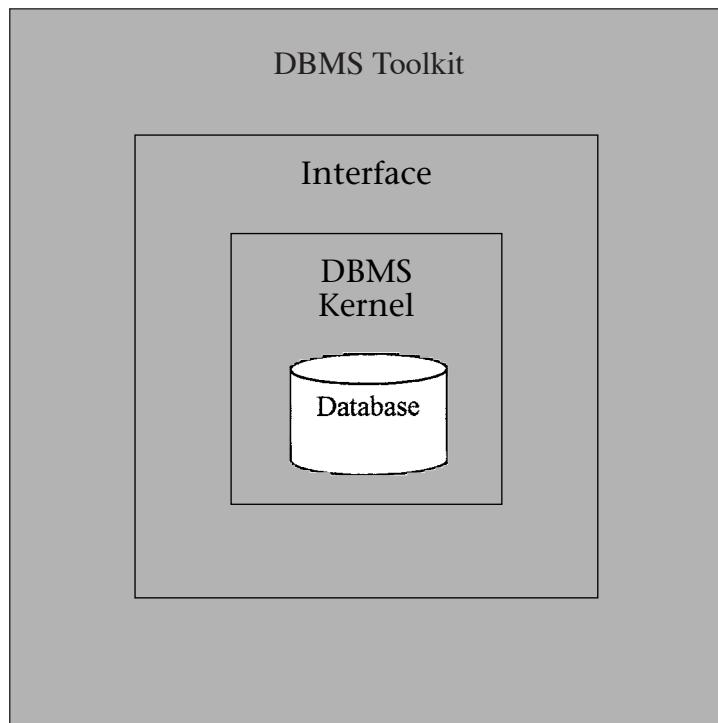


Figure 6.2 Kernel, interface and toolkit.

6.3.3 INTERFACE

Interposing between the kernel and toolkit we must have a defined interface. The interface is a standard language which connects a tool such as a fourth generation language with kernel functions. This is examined in Part 3.

6.4 FUNCTIONS OF A DBMS

There are a number of key functions that any DBMS must support.

6.4.1 CRUD FUNCTIONS

A DBMS must enable users to create data structures, to retrieve data from these data structures, to update data within these data structures and to delete data from these data structures. These functions are known collectively as CRUD activities – Create, Read, Update and Delete.

6.4.2 DATA DICTIONARY

A DBMS must support a repository of meta-data – data about data. This repository is known as a data dictionary or system catalog. Typically this data dictionary will



store data about the structure of data, relationships between data-items, integrity constraints expressed on data, the names and authorisation privileges associated with users. The data dictionary effectively implements the three-level architecture of a DBMS.

6.4.3 TRANSACTION MANAGEMENT

A DBMS must offer support for the concept of a transaction and must manage the situation of multiple transactions impacting against a database. A transaction is a series of actions which access or cause changes to be made to the data in a database.

6.4.4 CONCURRENCY CONTROL

A DBMS must enable many users to share data in a database – to access data concurrently. The DBMS must ensure that if two transactions are accessing the same data then they do not leave the database in an inconsistent state.

6.4.5 RECOVERY

The DBMS must ensure that the database is able to recover from hardware or software failure which causes the database to be damaged in some way.

6.4.6 AUTHORISATION

The DBMS must provide facilities for the enforcement of security. Generally the DBMS must support the concept of an authorised user of a database and should be able to associate with each user privileges associated with access to data within the database and/or facilities of the DBMS.

6.4.7 DATA COMMUNICATION

A DBMS must be able to integrate with communications software running in the context of an ICT system. This is particularly important to enable the connection of toolkit software with the kernel of a DBMS.

6.4.8 DATA INTEGRITY

Data integrity is that property of a database which ensures that it remains an accurate reflection of its universe of discourse. To enable this the DBMS must provide support for the construct of an integrity constraint. The DBMS must be able to enforce such constraints in the context of CRUD activities.

6.4.9 ADMINISTRATION UTILITIES

The DBMS should ensure that there are sufficient facilities available for the administration of a database. Such facilities will include (Chapter 26):

- Facilities for importing data into a database from other data sources
- Facilities for exporting data from a database into other data sources
- Facilities for monitoring the usage and operation of a database
- Facilities for monitoring the performance of a database and for enhancing this performance

6.5 THE INTERFACE

The DBMS interface will comprise a database sublanguage. A database sublanguage is a programming language designed specifically for initiating DBMS functions. Such a database sublanguage will contain one or more of the following parts:

6.5.1 DATA DEFINITION LANGUAGE (DDL)

The data definition language is used to create structures for data, delete structures for data and amend existing data structures. The DDL updates the meta-data held in the data dictionary.

6.5.2 DATA MANIPULATION LANGUAGE (DML)

The data manipulation language is used to specify commands which implement the CRUD activities against a database. The DML is the primary mechanism used for specifying transactions against a database.

6.5.3 DATA INTEGRITY LANGUAGE (DIL)

The data integrity language is used to specify integrity constraints. Constraints specified in a DIL are written to the data dictionary.

6.5.4 DATA CONTROL LANGUAGE (DCL)

The data control language is that part of the database sublanguage designed for use by the database administrator. It is particularly used to define authorised users of a database and the privileges associated with such users.

The dominant example of such a database sublanguage currently is the structured query language or SQL (Part 3). Such a sublanguage is frequently used in association with some application-building tools such as a fourth generation language (4GL) to implement an ICT system (Chapter 4).

6.6 THE KERNEL

Figure 6.3 illustrates the fact that most DBMS are pieces of software that sit on top of some native operating system. Because of this, the DBMS needs to interact with

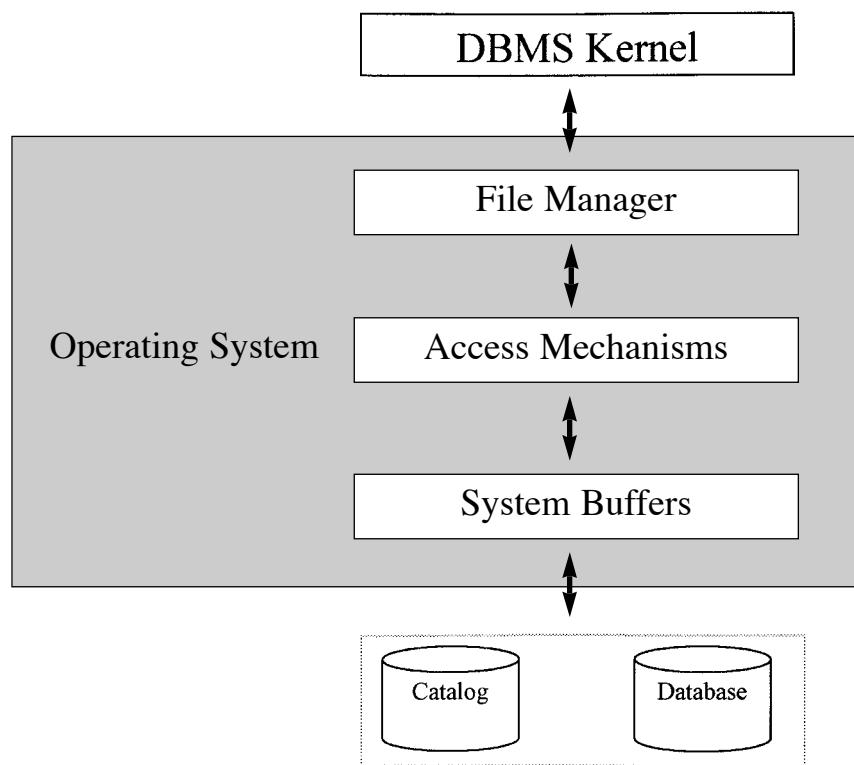


Figure 6.3 Interaction between DBMS and operating system.

the operating system, particularly to implement and access the database and system catalog which are usually stored on disk devices. Three elements of the operating system are critical to this interaction:

- *File manager*. The file manager of the operating system translates between the data structures manipulated by the DBMS and the files on disk. To perform this process it establishes a record of the physical structures on disk (Chapter 27)
- *Access mechanisms*. The file manager does not manage the physical input and output of data directly. Instead it interacts with appropriate access mechanisms established for different physical structures (Chapter 28)
- *System buffers*. The reading and writing of data are normally conducted in terms of the system buffers in the operating system. These are temporary structures for storing data and for manipulating data

Modern DBMS are highly complex pieces of software. The actual architecture of a DBMS will vary among products. However, it is possible to abstract from the variety of implementations a number of generic software components available in DBMS (see Figure 6.4):

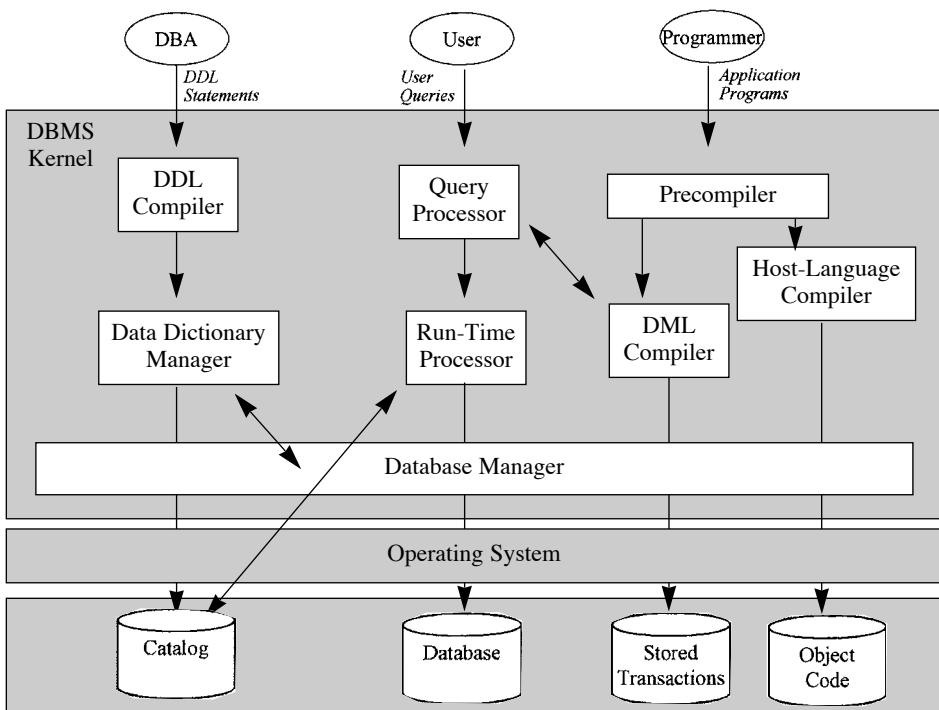


Figure 6.4 DBMS kernel.

- **Database and system catalog.** The database and system catalog store the data and meta-data of an application
- **Operating system.** Access to disk devices is primarily controlled by the host operating system which schedules disk input and output
- **Database manager.** A higher-level data manager module in the DBMS controls access to the DBMS information stored on disk. All access to database information is performed under the control of the database manager. This database manager may use lower-level data management functions provided by the host operating system in its file management system or may use its own routines. Such functions will be involved with transferring data to and from main memory and disk
- **DDL compiler.** The DDL compiler takes statements expressed in the Data Definition Language and updates the system catalog. All DBMS modules that need information about database objects must access the catalog
- **Run-time processor.** The run-time processor handles retrieval or update operations expressed against the database. Access to disk goes through the data manager
- **Query processor.** The query processor handles interactive queries expressed in a data manipulation language (DML) such as SQL. It parses and analyses a query before generating calls to the run-time processor



- *Precompiler, DML compiler, host-language compiler.* The precompiler strips DML commands from an application program written in some host language. These commands are then passed to the DML compiler which builds object code for database access. The remaining parts of the application program are sent to a conventional compiler, and the two parts are linked to form a transaction which can be executed against the database (Chapter 30)

6.7

THE DATABASE MANAGER

The database management module of a DBMS will enact the following functions:

- *Authorisation control.* The database manager must check the authorisation of all attempts to access the database
- *Processing commands.* The database manager must execute the commands of the database sublanguage
- *Integrity checking.* The database manager must check that every operation that attempts to make a change to the database does not violate any defined integrity constraints
- *Query optimisation.* This function determines the optimal strategy for executing any queries expressed against a database
- *Managing transactions.* The database manager must execute any transactions against a database
- *Scheduling transactions.* This function ensures the operation of concurrent access of transactions against a database
- *Managing recovery.* This function is responsible for the commitment and aborting of transactions expressed against a database
- *Managing buffers.* This function is responsible for transferring data to and between the main memory and the secondary storage

6.8

CASE STUDY: THE DATA MANAGEMENT LAYER OF UTILITY COMPANY SYSTEMS

In February 2003 it was reported that 50,000 British customers had been waiting for up to a year for their electricity bills to arrive. The key reason was that electricity supply companies have not developed common data standards for the data management layer of their ICT systems following deregulation of this market in 1998. British Gas was reported to have lost £13 million as a result of this problem.

There are some 30 companies supplying electricity in the UK, each of which employs a number of subcontracting companies to collect, store and aggregate customer data that includes:

- Details of customers and tariffs
- Data held about electricity meters and meter readings
- Data on the history associated with particular meters and readings
- Schedule dates for meter readings

Hence there are dozens of companies involved in the collection, storage and processing of customer data. Unfortunately, there is no common data format for such data. Also, much of this data is held in legacy information systems constructed before deregulation of the industry. Because of such inconsistencies in data management layers, significant costs are associated with transferring data between systems because mismatches frequently have to be resolved manually.

6.9 SUMMARY

- The ANSI/SPARC architecture distinguishes between the external level, the conceptual level and the internal level
- A DBMS must support the following functions: CRUD functions, data dictionary, transaction management, concurrency control, recovery, authorisation, data communication, data integrity and administration utilities
- We may distinguish between three parts of a DBMS: kernel, interface and toolkit
- The DBMS kernel provides all the central functions such as CRUD functions and concurrency control
- The DBMS toolkit comprises the range of tools that can be used in association with a database
- The DBMS interface comprises a standard language for accessing kernel functions

6.10 ACTIVITIES

1. In terms of a DBMS known to you, identify the functions listed in this chapter.
2. What database sublanguage does the DBMS selected for activity 1 use? Determine the features of the offered language.
3. Write a simple data entry program to store data in a file of simple student records composed of the name and address for each student.
4. Make some changes to the program of activity 3, such as the ability to enter the student's date of birth. Identify the consequent changes needed to the data file.



5. Make some changes to the data file of activity 4, such as identifying each student record by a unique student identifier. Identify the consequent changes needed to the data entry program.
6. Detail some of the functionality of a program to allow a user to construct a simple data dictionary.

6.11

- ANSI (1975). ANSI/X3/SPARC Study Group on Data Base Management Systems: Interim Report, FDT. *ACM SIGMOD Bulletin* 7(2).
- Ramakrishnan, R. and J. Gehrke (2003). *Database Management Systems*. Boston, MA, McGraw-Hill.

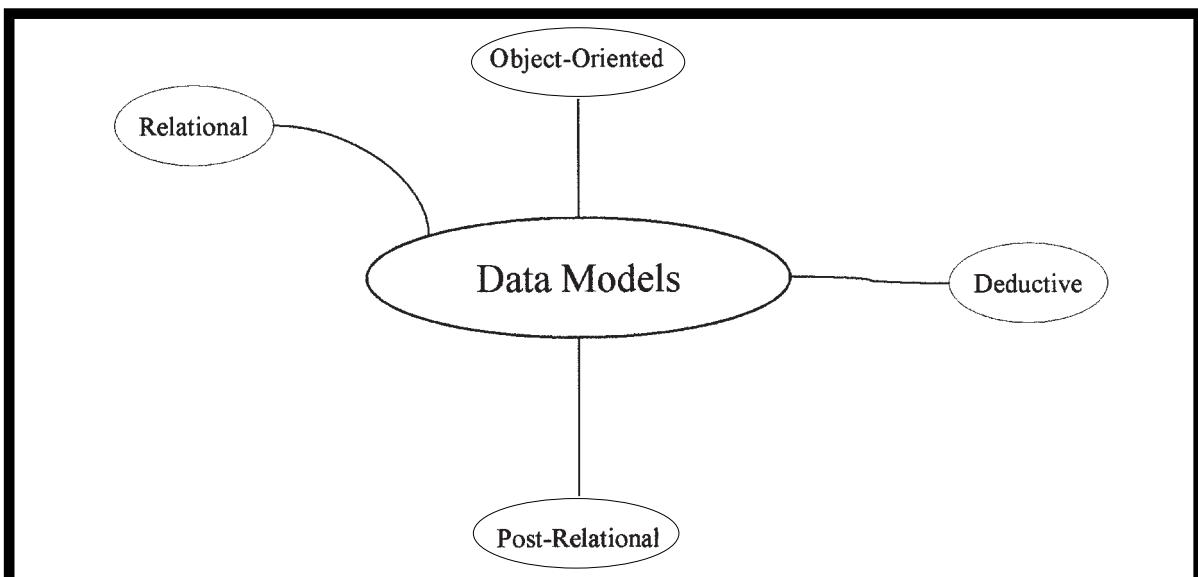


PART 2

DATA MODELS

*No man is an island, entire of itself,
Every man is a piece of the continent, a part of the main.*

John Donne



Part 2 is devoted to the issue of data management architectures, or what we have referred to in Part 1 as a data model. An understanding of key data models is important for the following reasons:

- A data model allows us to treat a database as an abstract machine. In other words, we can concentrate on the principles of design divorced from an immediate concern with implementation. We can get the data organised suitably before building the technology
- Data models constitute formal languages for defining data structures declaring integrity and for manipulating data. A data model is a mechanism for specifying the schema of some database
- Data models establish the principles underlying DBMS



In this part we consider a number of alternative data models – relational, object-oriented, deductive and post-relational. Each data model has its own strengths and weaknesses in terms of commercial applications.

By far the largest chapter is devoted to the relational data model. This is for two reasons. First, the relational data model is unusual in having a uniform, theoretical foundation. Second, database systems based upon this data model are currently the most popular in commerce and industry.

We also consider two advanced data models: the object-oriented data model and the deductive data model. The object-oriented data model is particularly important because of its ability to capture complex processing within the remit of a database system. The deductive data model is important in that it permits the incorporation of a number of ‘intelligent’ features into database systems.

We conclude Part 2 with a brief examination of a number of features popular in contemporary DBMS that can be seen as extensions to the relational data model in that they incorporate certain features that may be interpreted as having an object-oriented foundation. We consider this collection of features very loosely as a data model and refer to it as the post-relational data model.

A loose chronology for the development of the data models discussed in this part of the book is illustrated in Figure P2.1. Semantic data models are addressed in the chapter on the post-relational data model.

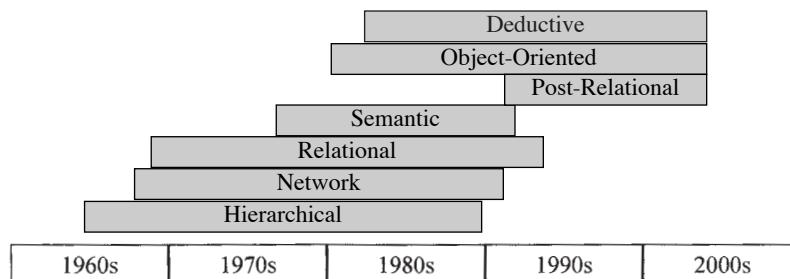


Figure P2.1 A chronology of data models.

It is tempting to describe the development of data models as an evolutionary one. For instance, the relational data model can be seen to build upon some of the foundations established by two data models which held sway in the commercial world for a number of years during the 1970s and 1980s – the hierarchical and network models. In turn, the object-oriented data model can be seen to have developed some of its features upon foundations established by the relational data model. However, in practice, of course, the picture is not so clear-cut. As an example of this, the relational model was in fact specified before any true specification emerged for the network data model.

One should also note that the chronology of data models does not correspond precisely with the chronology of DBMS founded in these data models. An example here is the way in which, although the relational data model was specified in the early 1970s, it took some fifteen years or so for the model to start to experience a commercial impact in terms of actual DBMS.



RELATIONAL DATA MODEL

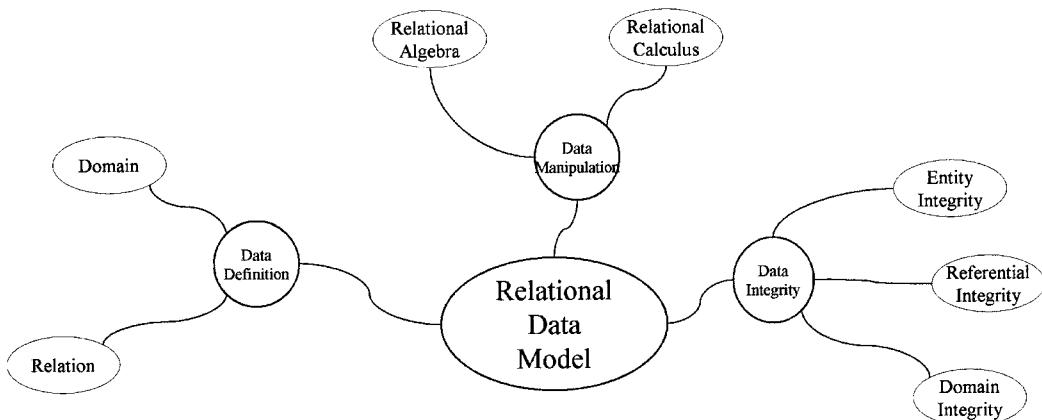
No hatred is so bitter as that of near relations.

Cornelius Tacitus (AD 55–117)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Describe the history of the relational data model
- Describe the key features of data definition in the relational data model
- Describe the key features of data manipulation in the relational data model
- Describe the key features of data integrity in the relational data model



**7.1****INTRODUCTION**

The relational data model is unusual in being largely due to the efforts of one man, E.F. Codd. In 1970 E.F. Codd published a seminal paper which laid the foundation for probably the most popular of the contemporary data models. From 1968 to 1988 Codd published more than 40 technical papers on the relational data model. He refers to the total content of the pre-1979 papers as Version 1 of the Relational Data Model (Codd, 1990).

Early in 1979, Codd presented a paper to the Australian Computer Society at Hobart, Tasmania entitled 'Extending the relational database model to capture more meaning'. The paper was later to be published in the ACM Database Transactions (Codd, 1979). The extended version of the relational data model discussed in this paper Codd named RM/T (T for Tasmania).

Early in 1990 Codd published a book entitled *The Relational Model for Database Management: Version 2* (Codd, 1990). One of the main reasons Codd cited for publishing this work is that he believed that many vendors of DBMS products with the relational stamp had failed to understand the implications not only of RM/T but also of many aspects of Version 1 of the model. We consider the issue again in Chapter 11.

In this chapter we concentrate on describing the major elements of Version 1 of the data model. A discussion of elements of RM/T and Version 2 are included in appropriate sections.

In Chapter 10 we discuss a number of extensions that have been added to the relational data model to increase its functionality. Most of the key players in the DBMS market are built on the foundations of the relational data model but include a number of what we have called post-relational extensions.

7.1.1 OBJECTIVES

Codd has always maintained that there were three problems that he wanted to address with his theoretical work. First, he maintained that data models prior to his (see Chapter 6) treated data in an undisciplined fashion. His model was proposed as a disciplined way of handling data using the rigour of mathematics, particularly set theory. As a result of this increased rigour Codd expected two consequences to follow. Codd believed that his ideas would enhance the concept of program – data independence. He also expected to improve programmer productivity. As evidence of this, in 1982, when Codd received the ACM Turing award, the title of his acceptance paper was 'Relational database: a practical foundation for productivity' (Codd, 1982).

7.2**DATA DEFINITION**

A database is effectively a set of data structures for organising and storing data. In any data model, and consequently in any DBMS, we must have a set of principles for exploiting such data structures for information systems applications

within organisations. Data definition is the process of exploiting the inherent data structures of a data model for a particular organisational application.

7.2.1 RELATIONS

There is only one data structure in the relational data model – the relation. Because the idea of a relation is modelled on a mathematical construct, a relation is a table which obeys a certain restricted set of rules:

- Every relation in a database must have a distinct name
- Every column in a relation must have a distinct name within the relation
- All entries in a column must be of the same kind. They are said to be defined on the same domain
- The ordering of columns in a relation is not significant
- Each row in a relation must be distinct. In other words, duplicate rows are not allowed in a relation
- The ordering of rows is not significant
- Each cell or column/row intersection in a relation should contain only a so-called atomic value. In other words, multiple-values are not allowed in the cells of a relation

Example  The tables below conform to all of these rules and hence constitute relations:

Modules

moduleName	level	courseCode	staffNo
Relational Database Systems	1	CSD	244
Relational Database Design	1	CSD	244
Deductive Databases	3	CSD	445
Object-Oriented Databases	3	CSD	445
Distributed Databases	2	CSD	247

Lecturers

staffNo	staffName	status
244	Davies T	L
247	Patel S	SL
445	Evans R	PL

In contrast, the table below is not a relation because the *moduleName* column contains multiple values:



Courses	
courseCode	moduleName
CSD	Relational Database Systems
	Relational Database Design
	Deductive Databases
	Object-Oriented Databases
	Distributed Database Systems
BSD	Intro to Business
	Basic Accountancy

Codd originally borrowed from the terminology of mathematics to denote elements of his data model. Columns of tables are known as attributes. Rows of tables are known as tuples. The number of columns in a table is referred to as the table's degree. The number of rows in a table is referred to as the table's cardinality. Hence the cardinality of the modules relation above is 5 and the degree of the relation is 4.

Primarily because most commercial DBMS that refer to themselves as relational use the more colloquial terms table, row and column, we shall tend to conform to this practice both in this chapter and throughout the text.

7.2.2 PRIMARY KEYS

Each relation must have a primary key. This is to enforce the property that duplicate rows are forbidden in a relation. A primary key is one or more columns of a table whose values are used to uniquely identify each of the rows in a table.

In any relation there may be a number of *candidate keys*; that is, a column or group of columns which can act in the capacity of a unique identifier. The primary key is chosen from one of the candidate keys.

Example ➔

Consider the relation named Lecturers above. The attributes staffNo, staffName and status are currently candidate keys since the values in these columns are currently unique. We know in practice, of course, that as the size of the lecturers table grows, we may store information about more than one lecturer named Evans R, and more than one lecturer is likely to be a senior lecturer – an SL. This leaves staffNo – a unique code for each lecturer of a university – as the only practicable primary key.



Any candidate key, and consequently any primary key, must have two properties. It must be unique, and it must not be null – a special character being used to indicate a missing or incomplete datum. First, by definition any candidate key must be a unique identifier. Hence, there can be no duplicate values in a candidate or primary key column. Second, we must have a primary key value for each row in a table. In other words, we cannot have a null (non-existent) value within a primary key column or columns.

7.2.3 DOMAINS

The primary unit of data in the relational data model is the data-item. Such data-items are said to be non-decomposable or atomic. A set of such data-items of the same type is said to be a domain. Domains are therefore pools of values from which actual values appearing in the columns of a table are drawn.

Examples ➔ Examples of data-items include a staff number such as 244, a lecturer name such as S Patel or a student's date of birth such as 1/7/1986.

Suppose there are four members of staff in our institution with the staff numbers 244, 386, 534 and 222. The domain of staff numbers is the set of all possible staff numbers, in our case {244, 386, 534, 222}.

7.2.4 FOREIGN KEYS

Foreign keys are the means of interconnecting the data stored in a series of disparate tables. A foreign key is a column or group of columns of some table which draws its values from the same domain as the primary key of some related table in the database.

Example ➔ In our university example staffNo is a foreign key in the modules table. This column draws its values from the same domain as the staffNo column – the primary key of the lecturers table. This means that when we know the staffNo of some lecturer we can cross-refer to the lecturers table to see, for instance, the status of that lecturer.

7.2.5 NULL VALUES

A special character is used in relational systems to indicate incomplete or unknown data – the character null. This character, which is distinct from zero or space, is particularly useful in the context of primary–foreign key links.

However, the concept of null values is not without controversy. Codd maintains that the introduction of null values into relational systems turns conventional two-valued logic (true, false) into a three-valued logic (true, false, unknown). In normal two-valued logic if question 1 is true and question 2 is true the combined question is also true. In three-valued logic if question 1 is



true and question 2 is unknown then the truth-value of the combined question is unknown. This introduces additional complications into query-processing in relational systems which some commentators (e.g. Chris Date) believe is unnecessary.

7.2.6 SYNTAX FOR DATA DEFINITION

There is no agreed syntax for expressing the structure of relations. Here, we express a relational schema as a series of templates made up of relation names, attributes and primary and foreign keys declarations

Example → The two templates below define the current schema for the academic database:

Domains

```
moduleNames: CHARACTER(15)
levels: INTEGER{1,2,3}
courseCodes: CHARACTER(3)
staffNos: INTEGER
statuses: CHARACTER(6): {L,SL,PL,Reader,Prof,HOD}
staffNames: CHARACTER(20)
```

Relation Modules

Attributes

```
moduleName: moduleNames
level: levels
courseCode: courseCodes
staffNo: staffNos
Primary Key moduleName
Foreign Key staffNo references Lecturers
```

Relation Lecturers

Domains

Attributes

```
staffNo: staffNos
staffName: staffNames
status: statuses
Primary Key staffNo
```

Note how we have separated domain definitions from relation definitions. We have assumed some predefined domains: character and integer – and also specified two domains using sets – lists contained in curly brackets. For instance, in the case of the column *level* we are allowing only the values 1, 2 or 3 as valid values.

7.2.7 THE BRACKETING NOTATION

It is frequently convenient to use a shorthand notation for a full relational schema definition. This is known as the bracketing notation. We list a suitable



mnemonic name for the table first. This is followed by a list of data-items or column names delimited by commas. It is conventional to list the primary key for the table first and underline this data-item. If the primary key is made up of two or more attributes, we underline all the component data-items.

Example The two relations for our university example would look as follows:

Modules(moduleName, level, courseCode, staffNo)

Lecturers(staffNo, staffName, status)

7.3 DATA MANIPULATION

In this section we discuss the second of the three parts of the relational data model, that part dealing with data manipulation.

Data manipulation has four aspects:

- How we input data into a relation
- How we remove data from a relation
- How we amend data in a relation
- How we retrieve data from a relation

When Codd first proposed the relational data model, by far the most attention was devoted to the final aspect of data manipulation – data retrieval. Retrieval in the relational data model is conducted using a series of operators known collectively as the relational algebra.

7.3.1 RELATIONAL ALGEBRA

The relational algebra is a set of eight operators. Each operator takes one or more relations as input and produces one relation as output. The three main operators of the algebra are restrict, project and join. Using these three operators most of the manipulation required of relational systems can be accomplished.

The additional operators – product, union, intersection, difference and division – are modelled on the traditional operators of set theory. Figure 7.1 illustrates the operators of the relational algebra.

There is no standard syntax for the operators of the relational algebra. We therefore use here a notation designed more for the purposes of explanation than for rigour.

7.3.2 RESTRICT

Restrict is an operator which takes one relation as input and produces a single relation as output. Restrict can be considered a ‘horizontal slicer’ in

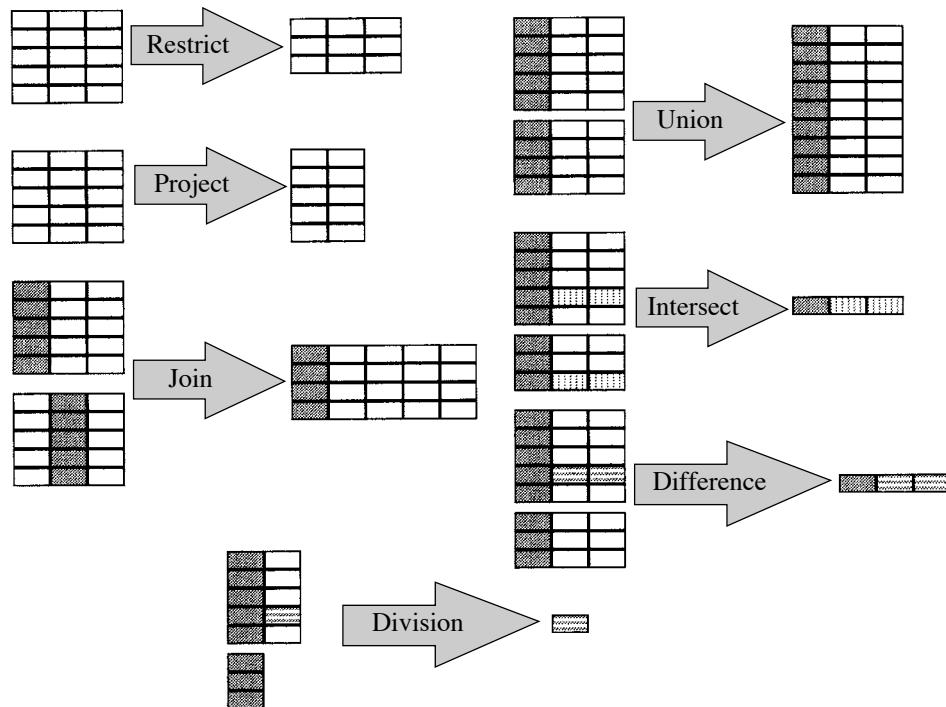


Figure 7.1 The relational algebra.

that it extracts rows from the input relation matching a given condition and passes them to the output relation. Our syntax for the restrict operator is as follows:

RESTRICT <table name> [WHERE <condition>] → <result table>

Example ➔

RESTRICT Modules WHERE level = 1 → R1

This statement would produce the following relation as output:

R1

<i>moduleName</i>	<i>level</i>	<i>courseCode</i>	<i>staffNo</i>
Relational Database Systems	1	CSD	244
Relational Database Design	1	CSD	244

Note that we need to provide a name, in this case R1, for the output relation. R1 must adhere to all the rules described in section 7.2.1.



7.3.3 PROJECT

The project operator takes a single relation as input and produces a single relation as output. Project is a ‘vertical slicer’ in that it produces in the output relation a subset of the columns in the input relation. The syntax for the project operator is as follows:

PROJECT <table name> [<column list>] → <result table>

Example	PROJECT Modules(moduleName) → R1
	R1
	moduleName
	Relational Database Systems
	Relational Database Design
	Deductive Databases
	Object-Oriented Databases
	Distributed Databases

7.3.4 PRODUCT

Joins are based on the relational operator product, a direct analogue of an operator in set theory known as the Cartesian product. Product takes two relations as input and produces as output one relation composed of all the possible combinations of input rows. Product is a little-used operator in practice because of its potential for generating an ‘information explosion’. Hence a product of lecturers with modules will list all possible lecturer-module combinations. However, product is the base from which the join operator is derived. The syntax of the product operator is given below:

PRODUCT <table 1> WITH <table 2> → <result table>

Example	PRODUCT Lecturers WITH Modules → R1
----------------	-------------------------------------

7.3.5 EQUI-JOIN

The join operator takes two relations as input and produces one relation as output. A number of distinct types of join have been identified. Probably the most commonly used is the natural join, a development of the equi-join.

The equi-join operator is a product with an associated restrict. In other words, we combine two tables together but only for records where the values match in the join columns of two tables. We shall assume that the primary key of one table and the foreign key of the other table form the default join



columns. (Actually, there is nothing preventing us from performing an equi-join across any two common columns between two relations.) Hence an equi-join of Lecturers with Modules will produce the table R1 below. The syntax is:

EQUIJOIN <table 1> WITH <table 2> → <result table>

Example → EQUIJOIN Lecturers WITH Modules → R1							
R1							
moduleName	level	course Code	staff No	staff No	staff Name	status	
Relational Database Systems	1	CSD	244	244	Davies T	L	
Relational Database Design	1	CSD	244	244	Davies T	L	
Deductive Databases	3	CSD	445	445	Evans R	PL	
Object-Oriented Databases	3	CSD	445	445	Evans R	PL	
Distributed Databases	2	CSD	247	247	Patel S	SL	

Here we have joined Lecturers and Modules, but only produced a row in R1 where a lecturer's staffNo value matches a module's staffNo value.

7.3.6 NATURAL JOIN

Note that an equi-join does not remove the duplicate join column in the resulting table above – staffNo appears twice in R1. A natural join removes one of these join columns. The natural join operator is a product with an associated restrict followed by a project of one of the join columns. Hence a natural join of Lecturers with Modules will produce the table R1 below:

JOIN <table 1> WITH <table 2> → <result table>

Example → JOIN Lecturers WITH Modules → R1						
R1						
moduleName	level	course Code	staff No	staff Name	status	
Relational Database Systems	1	CSD	244	Davies T	L	
Relational Database Design	1	CSD	244	Davies T	L	
Deductive Databases	3	CSD	445	Evans R	PL	
Object-Oriented Databases	3	CSD	445	Evans R	PL	
Distributed Databases	2	CSD	247	Patel S	SL	



7.3.7 OUTER JOINS

Natural joins are certainly the most common type of join used in practice. Another type of join is however important for commercial database processing: the outer join.

Example ➔ Suppose we extend our academic database as below. Note that some modules now temporarily have no lecturer (indicated by null), and some lecturers do not currently have any modules:

Modules

moduleName	level	courseCode	staffNo
Relational Database Systems	1	CSD	244
Relational Database Design	1	CSD	244
Deductive Databases	3	CSD	445
Object-Oriented Databases	3	CSD	445
Distributed Databases	2	CSD	247
Database Development	2	CSD	null
Data Administration	2	CSD	null

Lecturers

staffNo	staffName	status
244	Davies T	L
247	Patel S	SL
124	Smith J	L
145	Thomas P	SL
445	Evans R	PL

A natural join is designed to produce a result from two input relations R and S of only those rows in R that have matching rows in S, and vice versa. In the case above, a natural join will list all modules with lecturers.

A related set of operators known as outer joins has been proposed for use when we want to keep all rows in R or S or both in the result, whether or not they have matching rows in the other relation. There are three types of outer join: left outer joins, right outer joins and two-way outer joins (sometimes called full outer joins).

A left outer join preserves unmatched rows in the first table expressed in a join.



Examples → In our case, a left outer join will list all modules with or without lecturers as below:

RLeft

moduleName	level	course Code	staff No	staff Name	status
Relational Database Systems	1	CSD	244	Davies T	L
Relational Database Design	1	CSD	244	Davies T	L
Deductive Databases	3	CSD	445	Evans R	PL
Object-Oriented Databases	3	CSD	445	Evans R	PL
Distributed Databases	2	CSD	247	Patel S	SL
Database Development	2	CSD	null	null	null
Data Administration	2	CSD	null	null	null

Note that those modules with no lecturers have had the staffNo, staffName and status attributes padded with nulls.

A right outer join will list all lecturers with or without modules as below.

Example →

RRight

moduleName	level	course Code	staff No	staff Name	status
Relational Database Systems	1	CSD	244	Davies T	L
Relational Database Design	1	CSD	244	Davies T	L
Design Deductive Databases	3	CSD	445	Evans R	PL
Object-Oriented Databases	3	CSD	445	Evans R	PL
Distributed Databases	2	CSD	247	Patel S	SL
null	null	null	124	Smith J	L
null	null	null	145	Thomas P	SL

A two-way outer join will list all modules with or without lecturers and all lecturers with or without modules, as below.

R2way						
moduleName	level	course Code	staff No	staff Name	status	
Relational Database Systems	1	CSD	244	Davies T	L	
Relational Database Design	1	CSD	244	Davies T	L	
Deductive Databases	3	CSD	445	Evans R	PL	
Object-Oriented Databases	3	CSD	445	Evans R	PL	
Distributed Databases	2	CSD	247	Patel S	SL	
null	null	null	124	Smith J	L	
null	null	null	145	Thomas P	SL	
Database Development	2	CSD	null	null	null	
Data Administration	2	CSD	null	null	null	

7.3.8 UNION

Union is an operator which takes two compatible relations as input and produces one relation as output. By compatible is meant that the tables have the same structure – the same columns defined on the same domains.

Example ➔ Suppose we wished to union a table of lecturers information with a table of administrators data. We might do this as follows:												
Administrators												
<table border="1"> <thead> <tr> <th>staffNo</th> <th>staffName</th> <th>status</th> </tr> </thead> <tbody> <tr> <td>1001</td> <td>Davies P</td> <td>Clerk</td> </tr> <tr> <td>1024</td> <td>Jones S</td> <td>Senior Clerk</td> </tr> <tr> <td>445</td> <td>Evans R</td> <td>PL</td> </tr> </tbody> </table>	staffNo	staffName	status	1001	Davies P	Clerk	1024	Jones S	Senior Clerk	445	Evans R	PL
staffNo	staffName	status										
1001	Davies P	Clerk										
1024	Jones S	Senior Clerk										
445	Evans R	PL										

<table 1> UNION <table 2> → <result table>

Lecturers UNION Administrators → R1

**R1**

staffNo	staffName	status
1001	Davies P	Clerk
1024	Jones S	Senior Clerk
244	Davies T	L
247	Patel S	SL
445	Evans R	PL

Note that although Evans R appears in both the Lecturers and Administrators table, he only appears once in R1. This is because, since R1 is a relation, it cannot have duplicate rows.

7.3.9 INTERSECTION

Intersection is fundamentally the opposite of union. Whereas union produces the combination of two sets or tables, intersection produces a result table which contains rows common to both input tables.

<table 1> INTERSECTION <table 2> → <result table>

Example ➔

Lecturers INTERSECTION Administrators → R1

Applying intersection to the relations used in the previous section would give us a relation containing staff who were both lecturers and administrators.

R1

staffNo	staffName	status
445	Evans R	PL

7.3.10 DIFFERENCE

In most operators of the relational algebra, the order of specifying input relations is insignificant. A union of table 1 with table 2, for instance, is exactly the same as a union of table 2 with table 1. Using difference, in contrast, the order of specifying the input tables does matter.

Example ➔

Hence

Lecturers DIFFERENCE Administrators → R1

will produce all lecturers who are not administrators

R1

staffNo	staffName	status
244	Davies T	L
247	Patel S	SL

while

Administrators DIFFERENCE Lecturers → R1

will produce all administrators who are not lecturers

R1

staffNo	staffName	status
1001	Davies P	Clerk
1024	Jones S	Senior Clerk

7.3.11 DIVISION

Division or divide takes two tables as input and produces one table as output. One of the input tables must be a binary table (i.e. a table with two columns). The other input table must be a unary table (i.e. a table with one column). The unary table must also be defined on the same domain as one of the columns in the binary table.

The fundamental idea of division is that we take the values of the unary table and check them off against the associated column in the binary table. Whenever all the values in the unary table match with the same value in the binary table then we output a value to the output table.

Example ➔ Suppose that we maintain a table with a list of dates on which particular modules are taught:

ModuleDays

Module Name	Day
Relational Database Systems	19/12/2003
Relational Database Design	12/12/2003
Relational Database Design	19/12/2003
Object-Oriented Database Systems	20/12/2003
Object-Oriented Database Systems	20/12/2003

We might want to find a common date on which both relational database systems and relational database design is taught. Hence, our unary table would be:

**PairedModules****Module Name**

Relational Database Systems

Relational Database Design

If we divide the relation *ModuleDays* by the relation *PairedModules* we would get the following output relation:

R1**Day**

19/12/2003

7.3.12 A PROCEDURAL QUERY LANGUAGE

The relational algebra as described above is a procedural query language. This means that to extract information from a database using the algebra we must specify a sequence of operators in which the result from each step in the sequence constitutes all or part of the input to the next step.

Example → Consider the following query: *List all module names taken by T Davies*. There are a number of ways we can implement this query using the algebra. Two possible solutions are given below:

```
RESTRICT Lecturers WHERE staffName = 'Davies T' → R1
JOIN R1 WITH Modules ON staffNo → R2
PROJECT R2(moduleName) → R4
```

```
JOIN Lecturers WITH Modules ON staffNo → R1
RESTRICT R1 WHERE staffName = 'Davies T' → R2
PROJECT R2(moduleName) → R4
```

The relational algebra is a procedural query language because it demonstrates the property of closure. In other words, the output from the application of one relational operator can be passed as input to another relational operator, and so on. As we shall see some query languages, of which SQL (Part 3) is an example, do not demonstrate this property of closure.

The property of closure means that we can rewrite any sequence of algebraic statements as one nested expression, in that the result of one statement can be passed as a parameter to the next statement, and so on. This will be particularly useful when we come to specify additional constraints.



Example Below we provide a nested version of the query above.

```
PROJECT(JOIN Modules WITH
      (RESTRICT Lecturers WHERE staffName = 'Davies T')
      ON staffNo (moduleName) → R1
```

7.3.13 RELATIONAL CALCULUS

The relational calculus is an alternative to the relational algebra as far as the manipulative part of the relational data model is concerned. The main difference between the algebra and calculus is that whereas the algebra may be described as procedural or prescriptive, the calculus is non-procedural or declarative. By this we mean that in the calculus we write an expression which specifies what is to be retrieved rather than how to retrieve it.

The algebra and calculus are however identical in the sense that it can be shown that any retrieval specified in the algebra can also be specified in the calculus, and vice versa. The relational calculus is a language based on a branch of formal logic known as the predicate calculus. There are two variants of the relational calculus, one known as the tuple relational calculus, the other the domain relational calculus. The tuple relational calculus has formed the basis for SQL (Chapter 11), and the domain relational calculus has formed the basis for QBE (Query By Example) interfaces (Chapter 24). We shall concentrate on a brief description of the tuple relational calculus in this section.

An expression in the tuple relational calculus has the following basic form:

```
RANGE OF <tuple variable> IS <table name>
GET <tuple variable>.<attribute name>
INTO <result table>
WHERE <conditional statement>
```

A tuple variable is some named placeholder which is said to range over a particular database relation.

Example For instance:

```
RANGE OF l IS Lecturers
GET l.status
INTO R1
WHERE l.status = SL
```

This query declares a tuple variable l to range over the table Lecturers. It places all tuples matching the condition in the result table R1.

A statement in the calculus can include logical connectives (and, or, not) and the quantifiers (the existential quantifier *Forsome*, and the universal



quantifier Forall). Forsome, sometimes read as *there exists*, has the meaning that in a given set of tuples there is at least one tuple which satisfies a given condition.

Example The statement:

```
RANGE OF l IS Lecturers
GET l.status
INTO R1
WHERE Forsome l (l.status = SL)
```

has exactly the same result as the first statement.

Forall has the meaning that in a given set of tuples exactly all tuples satisfy a given condition.

Examples The statement:

```
RANGE OF m IS Modules
GET m.level
INTO R1
WHERE Forall m (m.level)
```

will retrieve all the levels associated with modules. This enables quite complex queries to be built as one statement, such as:

```
RANGE OF l IS Lecturers
RANGE OF m IS Modules
GET l.staffName
INTO R1
WHERE Forall m Forsome l (l.staffNo = m.staffNo)
```

which retrieves the names of all those lecturers who take all modules. In our case, of course, this would produce the empty relation as a result.

7.3.14 MAINTENANCE OPERATIONS ON RELATIONS

There are three basic maintenance operations needed to support retrieval activities specified in the algebra or calculus: insert, delete and update. We illustrate a suggested syntax for these operations below:

```
INSERT (<value>,<value>,...) INTO <table name>
DELETE <table name> WITH <condition>
UPDATE <table name> WHERE <condition> SET <column name> = <value>
```

Example ➔ INSERT ('Distributed Database Systems', 2,'CSD',247,'Jones S','SL') INTO Modules
 DELETE Modules WITH level = 1
 UPDATE Modules WHERE level = 1 SET level = 2

Whenever we apply any update operations we must ensure that none of the integrity constraints specified in a relational schema are violated. It is to this issue of integrity that we now turn.

7.4 DATA INTEGRITY

When we say a person has integrity we normally mean we can trust what that person says. We assume, for instance, a close correspondence between what that person says he or she did and what he or she actually did.

When we say a database has integrity we mean much the same thing. We have some trust in what the database tells us. There is a close correspondence between the facts stored in the database and the real world it models. Hence, in terms of our university database we believe that the fact – Jones is a lecturer – is an accurate reflection of the workings of our organisation.

Integrity is enforced via integrity rules or constraints. In the relational data model there are two inherent integrity rules: entity integrity and referential integrity. They are inherent because every relational database should demonstrate these two aspects of integrity.

7.4.1 ENTITY INTEGRITY

Entity integrity concerns primary keys. Entity integrity is an integrity rule which states that every table must have a primary key and that the column or columns chosen to be the primary key should be unique and not null.

A direct consequence of the entity integrity rule is that duplicate rows are forbidden in a table. If each value of a primary key must be distinct, no duplicate rows can logically appear in a table. Entity integrity is enforced by adding a primary key clause to a schema definition as, for instance, in the Lecturers relation in section 7.2.1.

7.4.2 REFERENTIAL INTEGRITY

Referential integrity concerns foreign keys. The referential integrity rule states that any foreign key value can only be in one of two states. The usual state of affairs is that the foreign key value refers to a primary key value of some table in the database. Occasionally, and this will depend on the rules of the business, a foreign key value can be null. In this case we are explicitly saying that either there is no relationship between the objects represented in the database or that this relationship is unknown.



Example ➔ Take the case of our university database. We observe only one foreign key – staffNo in Modules. By examining the Modules table we can verify that each Module has a staffNo value, and that each staffNo value corresponds to a value existing in the Lecturers table. In this case we can say that the relationship between module and lecturer is mandatory. Every module is assigned to a lecturer. No module record has a null staffNo value. We enforce this by adding a not null clause to the foreign key definition as below:

Relation Modules

Attributes

moduleName: moduleNames
level: levels
courseCode: courseCodes
staffNo: staffNos

Primary Key moduleName

Foreign Key staffNo **References** Lecturers **Not Null**

Domains

moduleNames: character
levels: {1,2,3}
courseCodes: character
staffNos: integer

However, suppose the rules of this organisation change. The university must now allow for the fact that new modules may not have an assigned lecturer for a certain period of time. For such modules it is decided that their staffNo value will be null until they are assigned to a lecturer (as in section 7.2.1). In this case, we assume that null is the default state for foreign keys and hence we do not add a not null clause to our foreign key declaration.

Maintaining referential integrity in a relational database is not simply a case of defining when a foreign key should be null or when it should not. It also entails defining propagation constraints. A propagation constraint details what should happen to a related table when we update a row or rows of a target table.

For every relationship between tables in our database we should define how we are to handle deletions of target and related tuples. Three possibilities exist:

- *Restricted delete.* This is the cautious approach. It means we forbid the deletion of the target row until all module rows for that key have been deleted from the related table
- *Cascades delete.* This is the confident approach. If we delete a target row all associated rows in the related table are deleted as a matter of course
- *Nullifies delete.* This is the median approach. If we delete a target row we set the foreign key values of relevant rows in the related table to null



Example ➔ In our university database if we delete a row from the Lecturers table (our target table) we must decide what should happen to related rows in the Modules table. Three possibilities exist:

- *Restricted delete.* This means we forbid the deletion of the lecturer row until all module rows for that lecturer have been deleted
- *Cascades delete.* If we delete a lecturer row all associated module rows are deleted as a matter of course
- *Nullifies delete.* If we delete a lecturer row we set the staff numbers of relevant modules to null

Relation Modules

Attributes

moduleName: moduleNames

level: levels

courseCode: courseCodes

staffNo: staffNos

Primary Key moduleName

Foreign Key staffNo **References** Lecturers **Not Null**

Delete Restricted

Update Cascades

In the definition above we have forbidden lecturer rows from being deleted until all associated module rows have been deleted. We have also enforced the constraint that any changes made to the staffNo within the Lecturers table should be reflected in the Modules table.

7.4.3 ADDITIONAL INTEGRITY

Not all aspects of integrity appropriate to a particular application can be represented using inherent integrity. This means that additional integrity must be implemented via other mechanisms.

We can add aspects of additional integrity to a relational schema by including a series of constraint definitions. Such definitions can be written as relational algebra or relational calculus expressions. We choose to write them as nested expressions of the relational algebra.

Examples ➔ For instance, although we can ensure that every module has a requisite lecturer by placing a not null clause against the foreign key staffNo, we cannot inherently represent the fact that every lecturer should have responsibility for at least one module. To do this, we must write a constraint as follows:

Constraint (Project Lecturers(staffNo)) Difference (Project Modules(staffNo)) is empty



In effect we are saying that any maintenance activity should trigger the running of this query. This constraint may then be added to the relation definition for Lecturers.

Another constraint might be to ensure that every level 3 module should be taught by a principal lecturer (PL). We write the following constraint to enforce this:

Constraint (Restrict Lecturers Where status <> 'PL') Intersection (Restrict Modules Where level = 3) is empty

7.5

FORMAL NOTATION FOR THE RELATIONAL ALGEBRA

In this chapter we have deliberately used a rather informal notation for the syntax of the relational algebra. A more formal notation for expressing operations in this formal language is presented in the table below.

Operator	Syntax	Definition
Restrict	The restrict operator transforms a single relation R into resulting tuples matching the specified condition (predicate)	$\sigma_{\text{predicate}}(R)$
Project	The project operator transforms a single relation R into a subset consisting of specified attributes a_1, \dots, a_n	$\Pi a_1, \dots, a_n(R)$
Union	The union operator produces output from two relations R and S containing all the tuples of R or S, or both R and S. Duplicate tuples are removed from the output	$R \cup S$
Difference	The difference operator produces output from two relations R and S in which output tuples exist in R but not S	$R - S$
Intersection	The intersection operator produces output from two relations R and S in which output tuples exist both in R and S	$R \cap S$
Natural join	A natural join is an equi-join of two relations R and S. One instance of each of the common attributes in the output relation is projected out	$R >< S$

7.6

CASE STUDY: A RELATIONAL SCHEMA FOR A UNIVERSITY DATABASE

In this chapter we introduced two relations from a database of interest to a university. This was clearly heavily simplified from an actual case. Below we present a more complete version of this schema expressed in the bracketing

notation. Primary keys are underlined. Foreign keys are tagged with an asterisk – *.

```
Courses(courseNo, courseName, deptName, courseLeader)
CourseEnrolment(courseNo*, studentNo*, enrolmentDate, feePaid)
Modules(moduleNo, moduleName, level, courseCode)
ModuleEnrolment(moduleNo*, studentNo*, enrolmentDate)
ModuleAssessment(moduleNo*, studentNo*, assessmentNo, grade)
Lecturers(staffNo, staffName, status, salary)
Students(studentNo, studentName, gender, studentTermAddress, studentTermTelNo,
studentHomeAddress, studentHomeTelNo)
Teaches(staffNo*, moduleNo*)
Prerequisites(moduleNo*, moduleNo*)
```

Note that we have introduced a different primary key for the relation Modules. We have also introduced a link relation (Chapter 16) to connect modules with lecturers. The primary key of this relation – ModuleEnrolment – contains two foreign keys from the linked relation. A number of other link relations exist in this schema – CourseEnrolment, ModuleEnrolment, ModuleAssessment, Teaches and Prerequisites. This last relation is interesting in that it builds a hierarchy of prerequisite conditions between modules. In other words, you cannot take module A without first having completed module B successfully.

The reason why moduleEnrolment and moduleAssessment are two separate relations will be explained in the chapter on normalisation (Chapter 18).

7.7 SUMMARY

- The relational data model has one data structure, the relation. The relation is a restricted table
- Relations are defined in terms of attributes, tuples, primary keys, foreign keys and domains
- A special character exists in relational systems for representing incomplete or missing information – null
- The manipulative part of the relational data model has some eight operators bundled together as the relational algebra
- The relational calculus is an alternative but equivalent formalism for the manipulative part of the relational data model
- The model also has two inherent integrity rules: entity and referential integrity

7.8 ACTIVITIES

1. Create a relational schema to hold information about insurance policies and policy holders. Insurance policies have properties policyNo, startDate, premium,



renewalDate, policyType. Policy holders are characterised by holderNo, holderName, holderAddress and holderTelno.

2. What would be suitable primary keys in this database?
3. How do we relate insurance policy information with policy holder information?
4. The insurance company want to enforce the business rule – every policy must have a holder. How do we enforce this rule in our database?
5. Declare a suitable domain for policy type.
6. Write a relational algebra statement to insert new holder information into the database.
7. Write a statement to update existing holder information in the database.
8. Write a statement to delete existing holder information from the database.
9. Write a relational algebra statement to list all standard life policies.
10. Write a statement to list the holder numbers and names of all persons holding standard life policies.
11. Write a constraint to enforce the fact that every policy holder should hold at least one policy in the database.
12. Given the implications of activity 4 and activity 11, are inner and outer joins relevant to this insurance database?

7.9



REFERENCES

- Codd, E.F. (1979). Extending the relational database model to capture more meaning. *ACM Trans. on Database Systems* **4**(4): 397–434.
- Codd, E.F. (1982). Relational database: a practical foundation for productivity. *Comm. of ACM* **25**(2).
- Codd, E.F. (1990). *The Relational Model for Database Management: Version 2*. Reading, MA, Addison-Wesley.

CHAPTER

8

OBJECT-ORIENTED DATA MODEL

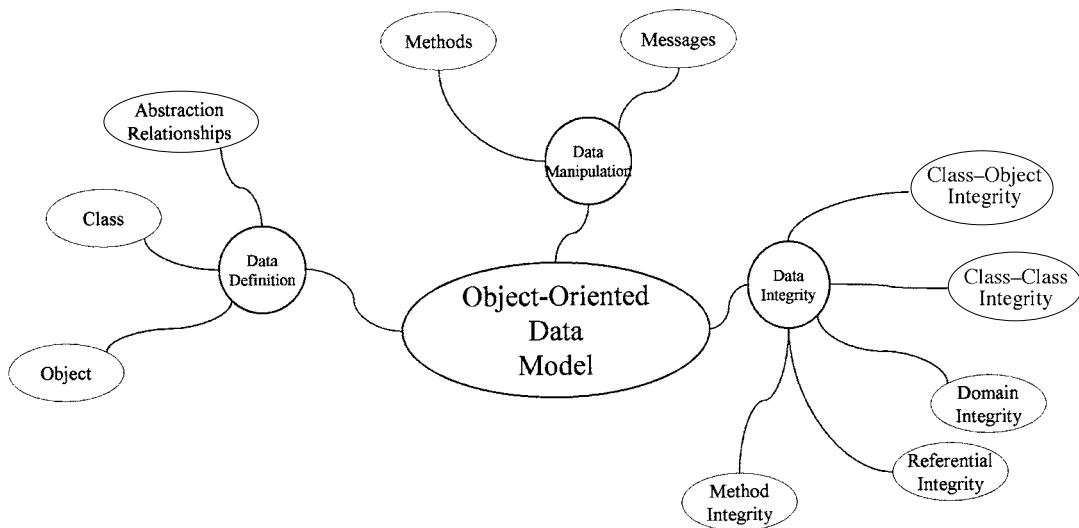
I never saw an ugly thing in my life: for let the form of an object be what it may – light, shade, and perspective will always make it beautiful.

John Constable (1776–1837)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Explain the evolution of the object-oriented (OO) data model from semantic data models
- Introduce an approach to object-orientation founded in extensions to the relational data model
- Describe the key features of the OO data model in terms of data definition, data manipulation and data integrity



**8.1**

INTRODUCTION

An OVUM report published in 1988 predicted that database systems adhering to an object-oriented (OO) data model as opposed to a relational data model would overtake relational database systems by the mid-1990s (Ovum, 1988). Although this clearly has not happened, there is no doubt that object-oriented concepts are influencing the development of information systems in general and database systems in particular. As evidence of this many relational DBMS vendors are beginning to offer OO features, and SQL3 (Chapter 31) includes some features of object-orientation in what was originally a relational database sublanguage. However, there is some confusion about what it actually means to be object-oriented. The present chapter attempts a brief discussion of this issue in terms of database systems.

The term object-oriented is used in a number of different senses within contemporary computing. The term was first applied to a group of programming languages descended from a Scandinavian invention known as SIMULA. SIMULA was the first language to introduce the concept of an abstract data type (Chapter 10), an integrated package of data structures and procedures. However, probably the best known object-oriented programming language is SMALLTALK – a programming environment developed at the Xerox Park Research Institute.

It is only comparatively recently that the term object-oriented has been applied to database systems. The main difference between object-oriented programming languages and databases is that object-oriented database systems require the existence of persistent objects (Chapter 1). In object-oriented programming languages, objects exist only for the span of program execution. In object-oriented database systems, objects have a life in secondary storage over and above the execution of programs.

There is still some debate about whether object-oriented database systems offer a better alternative to database management than relational database systems. There is no doubt that an object-oriented approach seems particularly well-suited to certain non-standard applications such as geographical information systems or computer-aided design. However, a question remains as to the use of object-orientation in standard commercial applications for databases.

Having said this, many vendors of relational DBMS have included facilities such as user-defined data types, triggers and stored procedures and described these as object-oriented features. We address this issue of so-called object-relational systems in Chapter 10.

8.1.1 SEMANTIC DATA MODELS

In Chapter 3 we made a distinction between the classic data models, such as the network and relational data models, and the semantic data models. The classic data models represent schemas using the concepts of record structures and links between record structures. The semantic data models are so named because they attempt to provide a richer set of constructs with which to build

schemas. Such constructs, if maintained, allow the database developer to incorporate within a database system more of the meaning or semantics of some application domain.

The semantic data models were originally developed to address some of the inherent weaknesses of the relational data model, such as:

- The relational data model is held to provide poor representation of 'real-world' entities. This is because of the fragmentation of an entity or classes into numerous relations through the process of normalisation (Chapter 18)
- The relational model is said to suffer from semantic overloading. This is because the data model uses one construct for both entities and relationships
- The relational data model suffers from the difficulty of managing complex objects. The relational constraint of atomic values means that hierarchical/nested structures cannot be accommodated easily

During the 1970s and 1980s a great number of alternative models to the relational model were proposed. Many of these approaches are surveyed in a paper by Hull and King (1987). One of the most popular groups of semantic data models is based around the entity–relationship approach of Chen (1976). This approach is the topic of Chapter 16. Most of the semantic data models extend Chen's basic approach with a number of abstraction mechanisms, the most popular being generalisation and aggregation. An example from this group of data models is SDM developed by Hammer and McCleod (1981). Generalisation and aggregation, as we shall see, form the basis for many object-oriented approaches.

There is little agreement also concerning the difference between an object-oriented data model and many of the semantic data models such as SDM. Perhaps a possible distinction can be made using a structural/behavioural framework. Semantic data models are usually seen as mechanisms for producing structural abstraction. In contrast, object-oriented data models are geared more towards providing behavioural abstraction. In other words, semantic data models are geared more towards the representation of data while object-oriented data models are geared more towards the manipulation of data (King, 1988). In this sense, some people have questioned whether an OO data model is not something of a misnomer, since it contains a process as well as data component.

8.2

EXTENDED RELATIONAL APPROACHES

Two alternative approaches to object-orientation in the context of databases have been proposed. The first is to replace the relational data model with a semantically richer data model more clearly suited to the needs of object-orientation. This is the topic of the current chapter. The second alternative is to exploit the capabilities of the relational model and to extend them with object-oriented facilities. This is the topic of Chapter 10.

8.3 (c) COMPONENTS OF AN OO DATA MODEL

In this chapter we consider a more radical solution to the problem of introducing object-orientation into the database arena. However, unlike the relational data model, no one person has defined the critical elements underlying OO database systems. Nevertheless, there is a developing consensus emerging as to the key elements of an OO data model (Atkinson *et al.*, 1990). A number of groups have attempted to develop standards for object data management. For instance, the Object Database Management Group (ODMG) was formed in 1991 by members of a limited number of companies, the objective being to develop a set of standards for object data management that would encourage portability of applications (Eaglestone and Ridley, 1998).

Because there is no readily agreed syntax for an OO data model, we use here a simplified syntax designed to display the general features of a possible model. It is very loosely based on features of the ODMG model (Chapter 32) and the facilities of the OODBMS described in Chapter 35.

8.4 (c) DATA DEFINITION

An OO database is made up of objects and object classes linked via a number of abstraction mechanisms.

8.4.1 OBJECTS

An object is a package of data and procedures. Data is contained in attributes of an object. Procedures are defined by an object's methods. Methods are activated by messages passed between objects.

An OO data model must provide support for object identity. This is the ability to distinguish between two objects with the same characteristics. The relational data model is a value-oriented data model. It does not inherently support the notion of a distinct identifier for each object in the database. Hence, in the relational data model two identical tuples or rows must fundamentally refer to the same object. In an object-oriented database two identical records can refer to two distinct objects via the notion of a unique system-generated identifier.

All objects must demonstrate the property of encapsulation. This is the process of packaging together of both data and process within a defined interface and controlled access across that interface. Encapsulation is implicit in the definition of an object since all manipulation of objects must be done via defined procedures attached to objects. The metaphor of an object as a gene or egg has been used to emphasise its character as containing all the information needed for it to behave appropriately in a given environment.



8.4.2 OBJECT CLASSES

An object class is a grouping of similar objects. It is used to define the attributes, methods and relationships common to a group of objects. Objects are therefore instances of some class. They have the same attributes and methods. In other words, object classes define the intension of an OO database – the central topic of database design. Objects define the extension (sometimes referred to as the extent) of an OO database – the central topic of database implementation.

8.4.3 ABSTRACTION MECHANISMS

An object data model supports two mechanisms which allow the database builder to construct hierarchies or lattices of object classes. Two such abstraction mechanisms are normally discussed: generalisation and aggregation.

Implicit in the construction of object classes is the support for the abstraction mechanism of generalisation. This allows us to declare certain object classes as subclasses of other object classes. For instance, *Manager*, *Secretary* and *Technician* might all be declared subclasses of an *Employee* class. Likewise, *SalesManager*, *ProductionManager* etc. would all be declared subtypes of the *Manager* class. Aggregation is the process by which a higher-level object is used to group together a number of lower-level objects. For instance, a *Car* entity might be built up of an assembly of wheels, chassis, engine etc.

An OO data model must also supply some means of linking objects to object classes. In this case an object class is said to classify an object, or in contrast an object is said to instantiate an object class.

8.4.4 INHERITANCE

Associated with the idea of a generalisation hierarchy is the process of inheritance. There are two major types of inheritance: structural inheritance and behavioural inheritance.

In structural inheritance a subclass is said to inherit the attributes of its superclass. Hence, suppose an employee class has attributes name, address, salary, then a subclass such as manager will also be defined by such attributes. In behavioural inheritance a subclass is said to inherit the methods of its superclass. Hence, if the employee class has a method calculatePay, then a subclass such as manager will also be characterised with this method.

We may also distinguish between single and multiple inheritance. If an OO database displays single inheritance then a class may be the subclass of only one superclass. Hence, a manager could not both be an employee and a customer. If an OO database displays multiple inheritance then a class may inherit the attributes or methods associated with more than one superclass. In this case, managers could be both employees and customers.



8.4.5 DEFINING A SCHEMA

Defining a schema for an OO database means defining object classes.

Example ➔ We define three simple object classes for a university database below:

```

CREATE CLASS Student
SUPERCLASS: object
ATTRIBUTE studentName: CHARACTER,
ATTRIBUTE termAddress: CHARACTER,
ATTRIBUTE dateOfBirth: DATE,
ATTRIBUTE dateOfEnrolment: DATE

CREATE CLASS Undergraduate
SUPERCLASS: Student
RELATIONSHIP Course memberOfCourse INVERSE courseEnrolment
RELATIONSHIP SET(Module) memberOfModule INVERSE Module moduleEnrolment

CREATE CLASS Postgraduate
SUPERCLASS: student
RELATIONSHIP Department assignedTo INVERSE Department registers

CREATE CLASS Assessment
ATTRIBUTE studentDetails: Student,
ATTRIBUTE moduleDetails: Module

```

The schema definition is made up of the following elements:

- Each class is defined by a unique name in the schema: Student, Undergraduate, Postgraduate, Assessment
- Three classes are defined on superclasses. The Student class is said to form a subclass of a system class (meta-class) called object
- The attributes of the class Student have been defined on pre-established data types integer, character and date. These are said to be simple attributes since they take a literal value and are defined on a standard data type such as integer
- The subclasses Undergraduate and Postgraduate contain no attribute declarations, but contain relationship declarations for memberOf and assignedTo. For instance, the Undergraduate class is declared as being related to the Module class. It is implicitly defined as a one-to-many link from Undergraduate to Module since the relationship is implemented as a set of Module identifiers. Both relationships have inverses defined. The inverse indicates a name for the traversal of the relationship from the linked class. Hence, the inverse relationship from Module to Undergraduate is defined as moduleEnrolment. This traversal will be defined in the Module class and is also likely to be one-to-many

- The class Assessment might be considered one in which the concept of aggregation was important. We might define an assessment as being made up of student and module details. The two attributes in this class are complex attributes since they contain collections and/or references – values that refer to other objects. Here the attributes are defined as references to specific instances of other classes

8.5 DATA MANIPULATION

Data manipulation in the OO data model is accomplished by passing messages to methods defined on objects.

8.5.1 METHODS

Methods come in four major forms:

- Constructor methods.* These operations create new instances of a class
- Destructor methods.* These operations allow the user to remove unwanted objects
- Accessor methods.* These operations yield properties of objects
- Transformer methods.* These operations yield new objects from existing objects

Examples In the case of a banking application the following methods might be appropriate:

- Constructor methods.* We would need a method for creating a new instance of a bank account given values for attributes such as accountNumber, customer details etc.
- Destructor methods.* This type of method would allow users to remove existing bank accounts
- Accessor methods.* A typical accessor method might be the calculation of service charges for a given bank account
- Transformer methods.* Examples of such methods might be debit and credit operations on a bank account

We may also distinguish between public methods, which are visible to the user and to other objects, and private methods, which are used mainly to perform internal operations for a class.

Methods would normally be assigned to classes at the definition stage.



Example ➔ CREATE CLASS Module

```
ATTRIBUTE moduleName: STRING,
ATTRIBUTE level: INTEGER,
ATTRIBUTE roll: INTEGER,
RELATIONSHIP Course givenOn,
RELATIONSHIP List(Lecturer) taughtBy,
METHOD increaseRoll(amount: INTEGER)
```

The transformer method declaration illustrated here constitutes only the method's 'signature'. In other words, it merely indicates the name of the method, and the name and types of the arguments. The body of a method would be defined using a standard programming language.

8.5.2 MESSAGES

Messages need to contain the name of a method, the object identifier (or some other way of accessing objects) and the appropriate parameters for the method.

Example ➔ A message to initiate the increaseRoll method declared above might look like:

```
Module where moduleName = relationalDatabaseSystems increaseRoll(30)
```

8.5.3 CREATING AND MANIPULATING A DATABASE

In this section we give two examples of the use of methods for data manipulation.

Examples ➔ Suppose we wish to create a student object. We might write a message:

```
Student create(studentName: 'Johns S', termAddress: '14, BelleVue Ave, Cardiff', dateOfBirth: 07/3/1977, date Of Enrolment: 1/10/1998)
```

We might delete objects using the folowing statement:

```
Student where dateOfEnrolment > 01/01/1990 delete)
```

Here we are passing a condition as a parameter to the delete method.



Inherent integrity in the OO model arises from the relationship between objects and classes. Since all access to objects must take place via methods,



additional integrity can be implemented via constraint definitions embedded in methods.

8.6.1 INHERENT INTEGRITY

An object-oriented data model can display:

- *Class-class integrity*. A superclass should not be deleted until all associated subclasses have been deleted
- *Class-object integrity*. A class should not be deleted until all associated objects have been deleted
- *Domain integrity*. Attributes are defined on pre-established classes such as integer, user classes such as Lecturer or sets of object identifiers
- *Referential integrity*. Since classes can be related to other classes via relationships, referential integrity issues similar to those discussed for relations exist in the OO data model

- Examples** ➔
- *Class-class integrity*. Hence a student class should not be deleted until undergraduate and postgraduate are deleted
 - *Class-object integrity*. Hence, the class undergraduate should only be deleted if all objects like 'Steve Austin' and 'Steve Jones' have been deleted
 - *Domain integrity*. For example, a course attribute of a class module might be defined on the set of course IDs
 - *Referential integrity*. Hence a lecturer should not be deleted who has outstanding modules in the database

8.6.2 ADDITIONAL INTEGRITY

Additional integrity can be added to an OO data model by adding constraints to the body of a method.

- Example** ➔ Suppose we attach the following method to the Student class:

```
enrolStudent(courseCode: Course)
```

The body of this method might be expressed in pseudo-code as:

```
body enrolStudent(courseCode: Course)
conditions
  student exists
  courseCode exists
  roll of course not exceeded
```

**actions**

create new instance of Enrolment class
update Student dateOf Enrolment

The conditions of this method enforce a number of constraints before performing any action.

8.7**CASE STUDY: CLASS SCHEMA FOR ACADEMIC DATABASE**

To illustrate how we can use two different data models for specifying the same universe of discourse we present a simplified class schema for the academic database presented in the case in Chapter 7. We have only included the class name and any generalisation or association relationships in the schema.

```

CREATE CLASS Department
SUPERCLASS Object
RELATIONSHIP SET(PostgraduateStudent) registers INVERSE PostgraduateStudent
assignedTo

CREATE CLASS Course
SUPERCLASS Object
RELATIONSHIP SET(UndergraduateStudent) courseEnrolment INVERSE
UndergraduateStudent memberOfCourse
RELATIONSHIP Lecturer courseLeader INVERSE Lecturer Leads

CREATE CLASS Module
SUPERCLASS Object
RELATIONSHIP SET(UndergraduateStudent) moduleEnrolment INVERSE
UndergraduateStudent memberOfModule
RELATIONSHIP LIST(Lecturer) taughtBy INVERSE Lecturer teaches
RELATIONSHIP SET(Module) prerequisite

CREATE CLASS Student
SUPERCLASS Object

CREATE CLASS UndergraduateStudent
SUPERCLASS Student
RELATIONSHIP Course memberOfCourse INVERSE courseEnrolment
RELATIONSHIP SET(Module) memberOfModule INVERSE Module moduleEnrolment

CREATE CLASS PostgraduateStudent
SUPERCLASS Student
RELATIONSHIP Department assignedTo INVERSE Department registers

CREATE CLASS ModuleAssessment
ATTRIBUTE studentDetails:Student
ATTRIBUTE moduleDetails:Student

```

CREATE CLASS Lecturer
 RELATIONSHIP Set(Module) Teaches INVERSE Module taughtBy
 RELATIONSHIP Course Leads INVERSE Course courseLeader

8.8 SUMMARY

- An OO database is made up of objects and object classes linked via a number of abstraction mechanisms
- An object is a package of data and procedures. Data is contained in attributes of an object. Procedures are defined by an object's methods. Methods are activated by messages passed between objects
- An object class is a grouping of similar objects. It is used to define the attributes, methods and relationships common to a group of objects. Objects are therefore instances of some class
- An object data model supports two abstraction mechanisms: generalisation and aggregation
- There are four types of methods: constructor, destructor, accessor and transformer
- There are four types of inherent integrity: class–class, class–object, domain and referential integrity
- Additional integrity can be defined in constraints associated with methods

8.9 ACTIVITIES

1. Declare the classes discussed in section 8.4.3 (employee, manager etc.) as an OO schema.
2. Declare a range of appropriate methods for the schema.
3. Declare some integrity constraints for the schema.
4. Describe the main differences between the OO data model and the relational data model.
5. Investigate in what ways the OO data model is similar to aspects of the hierarchical and network data models.
6. Declare the schema in section 7.6 as an OO schema.
7. What methods might be appropriate for this schema?

8.10 REFERENCES

Atkinson, M., D. DeWitt, D. Maier, F. Bancilhon, K. Dittrich and S. Zdonik (1990). The object-oriented database system manifesto. In W. Kim, J.-M. Nicolas and S. Nishio (Eds), *Deductive and Object-Oriented Databases*. North-Holland, Elsevier Science.



- Chen, P.P.S. (1976). The entity–relationship model: towards a unified view of data. *ACM Trans. on Database Systems* **1**(1): 9–36.
- Eaglestone, B. and M. Ridley (1998). *Object Databases: An Introduction*. Maidenhead, Berkshire, UK. McGraw-Hill.
- Hammer, M. and D. McCleod (1981). Database description with SDM: a semantic database model. *ACM Trans. on Database Systems* **6**(3): 351–86.
- Hull, R. and R. King (1987). Semantic database models: survey, applications and research issues. *ACM Computing Surveys* **19**(3): 201–60.
- King, R. (1988). My cat is object-oriented. In W. Kim and F. Lochovsky (Eds), *Object-Oriented Languages, Applications and Databases*. Reading, MA, Addison-Wesley.
- Ovum (1988). *The Future of Databases*. London, Ovum Press.

CHAPTER 9

DEDUCTIVE DATA MODEL

The real problem is not whether machines think but whether men do.

B.F. Skinner (1904–90)

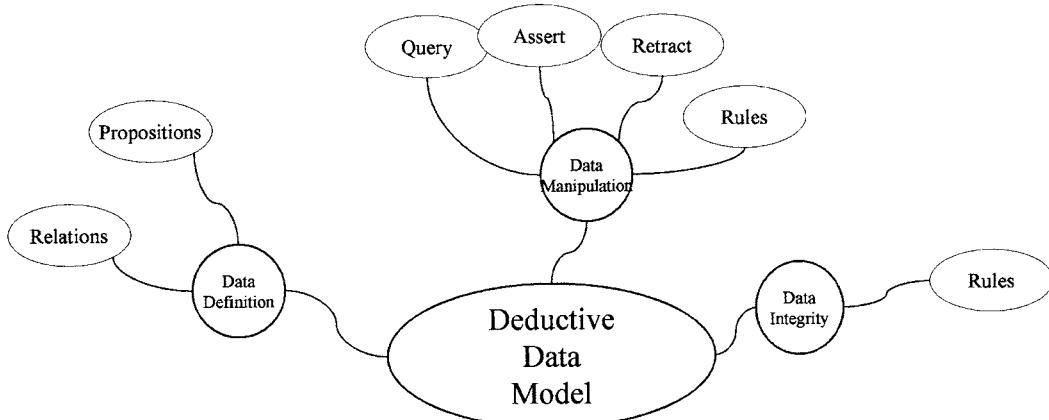
The grand aim of all science is to cover the greatest number of empirical facts by logical deduction from the smallest number of hypotheses or axioms.

Albert Einstein (1879–1955)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Discuss the genesis of the deductive data model in formal logic
- Describe the major elements of the deductive data model
- Explain why deductive databases are sometimes described as ‘intelligent’ databases



**9.1**

INTRODUCTION

The deductive data model fundamentally involves the application of formal logic to the problems of data definition, data manipulation and data integrity. Logic can act as a formal basis for expressing in one uniform manner these three aspects of a data model. Logic also provides the capacity to extend the idea of a conventional database with deductive facilities.

Research on the relationship between databases and formal logic dates back at least to the late 1970s. Codd's early work on the relational data model was heavily motivated by formal logic, particularly the development of the relational algebra and the relational calculus. The principal stimulus for the interest in this relationship seems to have been the publication of a landmark paper by Reiter (1984). In this paper Reiter characterised the traditional perception of databases as being model-theoretic. He contrasted this with a proof-theoretic view of databases. It is to this proof-theoretic view of databases that the term deductive databases is normally applied. This paper built upon the earlier work published by Reiter and others in a collected volume edited by Gallaire and Minker (1978).

Most of the work on deductive databases has been done using various variants of standard Prolog known collectively as Datalog. Prolog is an Artificial Intelligence language designed to implement aspects of formal logic as a programming environment. Datalog is fundamentally a variant of Prolog specifically designed for database work. This chapter constitutes a tutorial description of Datalog. For a more formal discussion of Datalog see Gardarin and Valduriez (1989).

9.2

DATA DEFINITION IN DATALOG

In this section we describe the basic data structures of the deductive data model. We assume that a database sublanguage based on formal logic can be built on the basis of the data model.

9.2.1 PROPOSITIONS AND RELATIONS

The fundamental data elements in the deductive data model are propositions. Propositions may combine to form data structures called relations.

Example Let us suppose that we want to represent the following assertions relevant to the university database first introduced in Chapter 1:

Relational Database Systems is a module
John Davies is a student
John Davies is taking Relational Database Systems

In Datalog we represent such assertions as:

```
module(relationalDatabaseSystems).
student(johnDavies).
takes(johnDavies, relationalDatabaseSystems).
```

Within logic the terms ‘module’, ‘takes’ and ‘student’ are called predicates, while the terms ‘relationalDatabaseSystems’ and ‘johnDavies’ are called arguments. Predicates are written first followed by arguments. Arguments are enclosed in brackets, delimited by commas. The whole assertion is concluded with a period.

Predicates and arguments form a collection known as a proposition, and each proposition may take only one of two possible values, namely, true or false.

Example  Hence, each of the following propositions has a truth-value.

```
module(relationalDatabaseSystems).
module(relationalDatabaseDesign).
```

In the area of databases we assume that if a proposition is written with concrete arguments then it is assumed to be true. A collection of propositions with the same predicate is known as a relation.

9.2.2 CONSTANTS AND VARIABLES

Each argument in a proposition can be either a constant or a variable. A constant indicates a particular individual or class of individuals. A variable is a placeholder in the sense that it indicates that such an individual or class of individuals remains unspecified.

Example  Hence, one interpretation of the following proposition would be *there is some entity X that is a student*:

```
student(X).
```

In Datalog, it is conventional to write constants beginning with a lower-case letter. Variables begin with an upper-case letter.



Example Hence, the following are constants:

johnDavies
hywelEvans

while the following are variables:

Student
Lecturer

When a variable becomes filled with or bound to a constant, that variable is said to be instantiated.

Example We might therefore instantiate the variable X in the proposition student(X) with the constant johnDavies.

A proposition or assertion with all constants for arguments is said to be a ground proposition or clause.

Example Hence, the proposition below is a ground proposition:

student(johnDavies).

The extension of a deductive database consists primarily of ground propositions or clauses.

9.2.3 CONNECTIVES

Individual propositions are often referred to as being ‘atomic’. By this is meant that the internal structure and meaning of such propositions are not determined by the logic. They are in a sense the primitive material from which we build. Such atomic propositions may however be combined by the use of logical connectives with special symbols. These include:

\wedge (AND), \vee (OR), \neg (NOT) and \Rightarrow IMPLIES

Example Thus we might have the following compound proposition:

lecturer (hywelEvans) \wedge teaches(hywelEvans, relationalDatabaseSystems).

which reads, *Hywel Evans is a lecturer and Hywel Evans teaches Relational Database Systems.*

Because the early Prolog systems could not handle symbols such as \wedge , \vee , \Rightarrow , these connectives are usually written in the following way:

- \wedge as , (comma)
- \vee as ; (semicolon)
- \Rightarrow as :- (colon followed by a hyphen)

Example  Hence, the statement above would be rewritten:

```
lecturer (hywelEvans), teaches(hywelEvans, relationalDatabaseSystems).
```

9.3 DATA MANIPULATION IN DATALOG

The uniformity of the deductive database approach can be demonstrated by the way in which there exists a seamless join between data definition and data manipulation.

9.3.1 ASSERT AND RETRACT

We add assertions to our database using the in-built function assert. We deny assertions by using the in-built function retract.

Example  Hence, to initialise the university database we might issue the following assert statements:

```
assert(module(relationalDatabaseSystems))
assert(student(johnDavies))
assert(takes(johnDavies, relationalDatabaseSystems))
```

To remove reference to johnDavies from our database we issue the following statement:

```
retract(student(johnDavies))
```

Note that we assume that assert and retract modify propositions stored in a persistent store.

9.3.2 BASIC QUERY FUNCTION

To issue a basic query we merely write an atomic proposition with constants for arguments. The system then sees if it has an assertion which matches in the database. If it does, it replies *true* or *yes*. If it does not, it replies *false* or *no*.



Example ➔ Suppose we wished to see if hywelEvans is a lecturer. We would issue the following statement as a query:

```
lecturer(hywelEvans)
```

If an assertion with this exact form is found in the database then the system would respond *true*.

9.3.3 QUERIES WITH VARIABLES

Let us now assume that we wish to list all the lecturers in our database. To do this we need to write a proposition with a variable as an argument.

Example ➔ The simplest form of query of this type would be:

```
lecturer(X)
```

where X is a variable.

In a Prolog system, the inference engine would look in its program and match on the first assertion of this form – i.e. with lecturer as predicate and a single argument. It then binds the constant it finds to X and displays the result.

Example ➔ `lecturer(hywelDavies)`

The Prolog engine would then respond 'Next Solution'. If the user replies 'yes' it then attempts to make the next possible binding. This process continues until the engine runs out of bindings, or the user responds 'no'.

In Datalog we assume that the engine will not return one solution, but it will return a set of solutions by default. Hence, to perform practicable queries in Datalog we need to assume the presence of some function such as *forall*. Bundling a query within this function will return the entire set of base assertions matching the query.

Example ➔ Hence, the statement:

```
forall(lecturer(X))
```

will produce all the possible lecturer names in our database.

9.4 DATA INTEGRITY IN DATALOG

Enforcing data integrity in the deductive data model means writing rules. Rules can also be used to produce virtual data, and handle recursive information.

9.4.1 THE IMPLIES CONNECTIVE

The IMPLIES connective (section 9.2.3) is particularly important in the sense that it allows us to construct rules of the form:

```
takes(Student, Module) ∧ teaches(Lecturer, Module)
⇒ assesses (Lecturer, Student).
```

That is, *if a student takes a module and a lecturer teaches that module, then that lecturer assesses that student*. This could be rewritten using Datalog symbols as:

```
assesses (Lecturer, Student) :-
    takes(Student, Module),
    teaches(Lecturer, Module).
```

Note that the conclusion of the rule, the rule head, is written first followed by an implies connective (:-), followed in turn by the body of the rule, a series of conditions delimited with commas.

9.4.2 UPDATE FUNCTIONS

In Chapter 1 we identified the following update function as being of relevance to the academic database:

Offer Module – assert that the module is offered.

Update functions are written as rules in Datalog. Hence offer module might be written in its simplest form as:

```
offerModule(X):-assert(module(X))
```

Hence, if we issued a statement such as:

```
offerModule(relationalDatabaseDesign)
```

to the Datalog engine the system would look for a matching predicate. X would then be instantiated to relationalDatabaseDesign. Finding the rule above it would proceed to execute the rule body. This would add the assertion:

```
module(relationalDatabaseDesign).
```

to the database.

9.4.3 INTEGRITY CONSTRAINTS

We have already mentioned that integrity constraints are rules. In fact, integrity constraints are usually associated with update functions.



Example ➔ Suppose we wish to implement the following update function and its associated constraints:

Enrol Student in Module – if the module is offered and the person is a student, then assert that the student takes the module.

This might be implemented as follows:

```
enrolStudentModule(S, M):-
    student(S),
    module(M),
    assert(takes(S,M)).
```

For this rule to work then all the conditions of the rule have to be true.

Example ➔ Hence, if we issued the statement:

```
enrolStudentModule(terryConnolly, relationalDatabaseDesign)
```

the system would not add an assertion to the database. It would not do this because the first condition, namely that S be a student, is currently false.

9.4.4 QUERIES ON A DATABASE WITH RULES

Introducing rules into our database causes a small complication in the way in which queries are handled.

Example ➔ Suppose our database (deductive) looks as follows:

```
module(relationalDatabaseSystems).
student(johnDavies).
lecturer(hywelEvans).
teaches(hywelEvans, relationalDatabaseSystems).
takes(johnDavies, relationalDatabaseSystems).

assesses(Lecturer, Student) :-
    takes(Student, Module),
    teaches(Lecturer, Module).
```

Suppose also that we run the following query against our database:

```
forall(assesses(L, S))
```

In other words, we wish to find all combinations of assessment relationships. The Datalog engine would look in the database for the predicate `assesses`. It finds no assertions for this but finds the predicate attached to a rule. It proceeds to execute

the conditions of the rule body in sequence from first condition to last condition. The first thing it does is try to instantiate `takes(Student, Module)`. This it can do with the values `johnDavies` for `Student` and `relationalDatabaseSystems` for `Module`. It then tries to instantiate the variable `Lecturer` in `teaches(Lecturer, Module)`. This it can do with the value `hywelEvans`. Since both conditions have been set to true, the rule head is true and the values `johnDavies` for `S` and `hywelEvans` for `L` can be displayed for the user. The engine then unravels the instantiations and starts the process again.

In a sense, querying on a database with rules produces virtual or derived data from the database.

9.5 DATALOG AND THE RELATIONAL DATA MODEL

A database which contains a collection of positive assertions is equivalent to a conventional database – a factbase. A database which contains both facts and rules is said to be a deductive database. It is deductive because we can use the rules to deduce virtual data – data not actually stored in the factbase.

Datalog without rules is essentially similar to that of the relational model (Chapter 7). Predicates in Datalog denote relations. However, in Datalog attributes are not named. Reference to a column is only through its position among the arguments of a given predicate.

Introducing rules into a database means that there are now two ways in which a relation can be defined. A predicate whose relation is declared explicitly by assertions is said to be part of the extensional database. A predicate defined by rules is said to be part of the intensional database.

In general the deductive data model is more powerful than the relational data model. The power of a deductive database over a conventional relational database can be demonstrated by considering the issue of recursive query processing.

9.5.1 RECURSION

Recursion occurs naturally in many domains having hierarchical structures – inventories, schedules, part hierarchies, routes.

However, conventional database sublanguages such as SQL (Part 3) are too weak to express recursion effectively. In such cases, data needs first to be retrieved from the database into an application program in which recursion is performed.

One advantage of applying formal logic to databases is that it allows recursive information to be explicitly specified in databases.



Example ➔ Suppose we wish to generate an organisational hierarchy based on a manages relation. The database is given below:

```
manages(jSmith, fJones).
manages(fJones, tDavies).
manages(jSmith, pEvans).
```

The first argument of the relation specifies the employee. The second argument specifies the employees' manager. To produce an organisational hierarchy we would include the following rules in our database:

```
manager(E, M) :- manages(E, M).
manager(E, M) :- manager(E, N), manages(N, M).
```

The first rule allows us to find someone's immediate superior. The second rule defines the fact that a manager of a manager is also a manager, and so on. Running a query such as `forall (manager(X,Y))` will recursively generate all the managerial relationships.

9.6

DIFFERENCES BETWEEN DATALOG AND PROLOG

The syntax of Datalog as described in previous sections looks very much like standard Prolog. So what are the differences between the programming language and the abstract or ideal database sublanguage? Some of the differences are presented below:

- *Set processing.* In Prolog results are returned one at a time. This is implicitly as we have described it in section 9.3.3. In Datalog results should be returned in sets. In other words, we assume the existence of built-in predicates such as `forall`
- *Order insensitivity.* Processing in Prolog is affected by the order of the rules or facts in a program and by the order of predicates in the body of a rule. A Prolog programmer frequently uses order sensitivity to improve the performance of a program. Datalog is a database sublanguage and as such should ideally be non-procedural. In other words, the execution of database queries should be insensitive to the order of retrieval of predicates
- *Special predicates.* Prolog programmers control the execution of programs through special predicates (e.g. the `cut`). This reduces the declarative nature of the language. Datalog does not include any comparable predicates
- *Function symbols.* Prolog has function symbols, which are typically used for building recursive functions and complex data structures. Base Datalog does not contain any function symbols, although a number of extensions to Datalog have been proposed, particularly for handling complex objects

Hence, syntactically Datalog is very similar to pure Prolog. Functionally they are different. Datalog is non-procedural, set-oriented, with no order sensitivity, no special predicates and no function symbols.

9.7 EXTENSIONS TO PURE DATALOG

The Datalog syntax described above corresponds to a restricted subset of first order logic and is often referred to as pure Datalog. Several extensions to pure Datalog have been proposed. Some of the most important extensions are built-in predicates, negation and complex objects.

9.7.1 BUILT-IN PREDICATES

Built-in predicates are expressed by special predicate symbols such as $>$, $<$, \geq and \leq , \neq , each of which has a predefined meaning. These symbols can appear in the conditions of a rule and are usually written in infix notation.

Example  For instance:

```
sibling(X,Y):- parent(Z,X), parent(Z,Y), X <> Y.
```

Here the symbol \neq is taken to mean *not equal to*.

9.7.2 NEGATION

In pure Datalog the negation operator \neg is not allowed. Normally the database is expected to adhere to the *closed world assumption*. This means that negative facts can be inferred from a set of positive Datalog clauses.

In general, a given fact about some universe of discourse (UoD) implies a large number of other facts. For example, knowing that John Smith teaches module relational database systems only (positive information) leads one to infer that John Smith does not teach modules relational database design, object-oriented databases etc. (negative information). This allows one to answer not only the query – Does John Smith teach module relational database systems? – but also queries of the form – Does John Smith teach module relational database design?

In the relational data model the usual semantic assumption made is that if a tuple does not exist in a relation then it exists in the complement of that relation. Thus the relation *teaches(Lecturer, Module)* represents all module offerings and *NOT(teaches(Lecturer, Module))* lists all modules that are not taught by the corresponding lecturers. The complement of a relation is usually not made explicit for reasons of efficiency.

Two analogous assumptions are made in deductive database systems: *the closed world assumption* and *negation as failure*. The closed world assumption merely



asserts that the only things held to be true about a UoD are the positive assertions about this UoD – all else is held to be false. The negation as failure assumption asserts that if a statement Q cannot be proved to be true, $\text{not}(Q)$ should be taken to be true. Hence, if one cannot prove that John Smith teaches module relational database systems then it should be concluded that John Smith does not teach relational database systems – failure to prove something equates to negation.

The closed world assumption however does not allow the use of negative facts to allow us to deduce further facts. In real life it is frequently important to express rules whose premises contain negative information.

Example ➔ For instance:

If X is a student and X is not a postgraduate student then X is an undergraduate student.

In pure Datalog such a rule would not be allowed. In extended Datalog such a rule might be written:

```
undergraduate(X):-student(X),  $\neg$ postgraduate(X).
```

9.7.3 COMPLEX OBJECTS AND FUNCTIONS

The ‘objects’ handled by pure Datalog programs correspond to the tuples of relations in that all attribute values are atomic. Datalog can be extended to handle complex objects, mainly by the inclusion of function symbols and set constructors.

Example ➔ We might construct a series of facts representing the complex object person as follows:

```
person(name(joe,bloggs),dob(5,march,1958),children({john,paul,george})).  
person(name(joe,smith),dob(5,march,1958),children({john,susan})).  
person(name(jill,jones),dob(5,april,1960),children({bill,sarah})).
```

Here name, dob and children are function symbols, the symbols {} are set constructors. In this scheme, variables may represent atomic objects or compound objects. This is similar to the idea of nested relations (Chapter 10). Hence the following rule produces the last names of all persons having the same first name and date of birth:

```
similar(X,Y):-person(name(Z,X),B,C),person(name(Z,Y),B,D).
```

By applying this rule we can infer the fact, $\text{similar}(\text{bloggs},\text{smith})$.

The question of what is intelligence has of course been an important one for philosophers and psychologists for hundreds of years. No concensus definition



has yet been reached. Many people have portrayed the concept of intelligence as a continually moving target. This target has been moving faster of late with the rise of a particularly mechanistic discipline devoted to the study of intelligence: artificial intelligence (AI).

One major hypothesis lies at the heart of all work in AI: the physical symbol system (PSS) hypothesis (Newell and Simon). Put simply, this hypothesis states that a physical symbol system has the necessary and sufficient means for generating intelligent action. So what is a physical symbol system?

A physical symbol system consists of a set of entities, called symbols, which are physical patterns that can occur as components of another type of entity called an expression (or symbol structure). Thus a symbol structure is composed of a number of instances of symbols which are related in some physical way. At any instant of time the system will contain a collection of these symbol structures. Besides these structures, the system also contains a collection of processes that operate on expressions to produce other expressions: processes of creation, modification, reproduction and destruction. A physical symbol system is a machine that produces through time an evolving collection of symbol structures. Such a system exists in a world of objects wider than just these symbol structures themselves (Newell and Simon, 1976).

The important thing about this definition is that this hypothesis cannot be proven or disproved on logical grounds. We can only test the truth of the hypothesis by empirical validation; that is, by building physical symbol systems that generate intelligent action.

Of course we are still left with the question of what is intelligent action. It is extremely difficult to delineate the exact characteristics of intelligent action. The idea of the Turing test frequently discussed in this context only takes us so far. Many maintain that the idea of a computer system generating intelligence is nonsensical. For instance, Winograd and Flores (1986) maintain that the idea of intelligent action cannot be divorced from context and human interpretation. In this sense, they place key emphasis on the last sentence in the above quote: *that a PSS exists in a complex, ever-changing world wider than itself*.

9.8.1 DATABASE SYSTEMS AND PHYSICAL SYMBOL SYSTEMS

In Chapter 1 we discussed the essential features of a conventional database system. Such a system clearly meets many of the features of a physical symbol system. However, a conventional database system would not normally be described as being intelligent. So what distinguishes an intelligent from a non-intelligent database? This question is best answered in terms of the idea of a representation formalism.

Every database system must use some representation formalism. A representation formalism is 'a set of syntactic and semantic conventions that make it possible to describe things' (Winston, 1984). In database terms the idea of a representation formalism corresponds with the concept of a data model (Tsitchizris and Lochovsky, 1982). A data model is an architecture for data. A



data model is a formalism for implementing some or all of the abstract features of a database system described in Chapter 1.

Each database formalism has its strengths and weaknesses. Consider probably the most popular data model of the present day – the relational data model. The relational model offers an extremely flexible way of representing facts and querying such facts but does not offer a convenient way of representing the intricacies of integrity constraints and update functions. Integrity constraints and update functions are normally implemented in application programs which sit on top of a relational database.

In contrast, deductive databases, the topic of this chapter, offer a powerful notation for representing facts, integrity constraints, queries and update functions. However, they currently suffer from a lack of efficient implementations for large factbases.

Both relational and deductive databases are forms of PSS. A pure relational database would not normally be described as being intelligent. A deductive database would normally be seen as an example of an intelligent database. One part of the definition of an intelligent database is therefore involved with considering how certain formalisms offer a more powerful and uniform set of facilities for implementing the component parts of a database system.

9.9

INTELLIGENT DATABASES AND BUSINESS APPLICATIONS

In the sections above we have implicitly defined one of the important characteristics of an intelligent database as being its ability to store rules and use such rules to provide an active dimension. An intelligent database no longer sits there receiving data from its environment; it can react to its environment.

We would probably make the claim that an intelligent database system is an important tool for improving the productivity of information systems development and maintenance (Beynon-Davies, 2002). Take, for instance, the example of integrity management. Some people have estimated that as much as 50–80% of a standard application is made up of integrity management. Traditionally, each program that accesses a database has to replicate this integrity management.

An intelligent database offers the possibility of storing integrity constraints centrally within the database itself. This is likely to mean:

- Replication of coding is reduced
- Information systems are likely to be smaller
- Information systems are likely to be more maintainable
- Better performance is likely to result in a cooperative processing environment

However, deductive databases are not the only approach to building intelligent database systems. It is useful to distinguish between two approaches to intelligent databases: evolutionary and revolutionary approaches.

Evolutionary approaches involve melding together existing technology or extending existing technology. For instance, one approach to building deductive databases is to meld a Prolog system to a relational DBMS.

Many of the in-built procedure and trigger mechanisms being offered by relational vendors can be cast in this light (Chapter 10), as can the current attempts to extend relational technology to handle complex objects. The current development of SQL3 (Chapter 31) is important in this context.

Revolutionary approaches propose new architectures. Datalog, for instance, is a proposal which unifies features of logic programming with databases. People are even attempting to extend Datalog to produce deductive object-oriented systems.

9.10 CASE STUDY: A DEDUCTIVE SCHEMA FOR A UNIVERSITY DATABASE

Below we present a more complete example of the deductive schema alluded to throughout this chapter for a university application. Comments are provided between the symbols /* and */.

Each entity in the schema is presented as a series of ground clauses – a proposition in which all the arguments are constants. The set of ground clauses with the same predicate constitutes a relation. Hence, there are three clauses in the *CourseEnrolment* relation below. The meaning of each attribute is defined by its position in the list of arguments for each predicate. Hence, the second argument in the *CourseEnrolment* relation always stands for a *studentNo*.

```
/*Courses – courseCode, courseName, deptName, courseLeader*/
courses('ISD', 'Information Systems Development', 'Computer Studies', '345').
/*Course enrolments – courseNo, studentNo, enrolmentDate, feePaid*/
courseEnrolment('ISD', '200001', '2002', 'Y').
courseEnrolment('ISD', '200002', '2002', 'Y').
courseEnrolment('ISD', '200003', '2000', 'Y').
/*Modules – moduleNo, moduleName, level, courseCode*/
modules('CS101', 'Relational Database Systems', 'I', 'ISD').
modules('CS102', 'Relational Database Design', 'I', 'ISD').
modules('CS301', 'Deductive Databases', '3', 'ISD').
/*Module enrolments – moduleNo, studentNo, year, semester enrolmentDate*/
moduleEnrolment('CS101', '200001', '2002', '1').
moduleEnrolment('CS101', '200002', '2002', '1').
moduleEnrolment('CS102', '200001', '2002', '2').
moduleEnrolment('CS102', '200002', '2002', '2').
/*Module assessments – moduleNo, studentNo, assessmentNo, grade*/
moduleAssessment('CS101', '200001', '1', '80%').
moduleAssessment('CS101', '200001', '2', '60%').
moduleAssessment('CS101', '200002', '1', '65%').
/*Lecturers – staffNo, staffName, status, salary*/
```



```

lecturers('234', 'Davies T', 'L', '20000').
lecturers('237', 'Patel S', 'SL', '27500').
lecturers('345', 'Evans R', 'PL', '35500').
/*Students – studentNo, studentName, gender, studentTermAddress, studentTermTelNo,
studentHomeAddress, studentHomeTelNo*/
students('200001', 'Morgan P', 'M', '11 Bronwyn St.', '562854', '12 Candice St., Liverpool',
'678923').
students('200002', 'Mirani F', '11 Mair Rd.', '564354', '12 Temple St., Reading', '228943').
students('200003', 'Smith T', 'M', '11 Bronwyn St.', '562854', '12 Canterbury Ave., Leeds',
'442223').
/*Teaches – lecturerNo, moduleNo*/
teaches('234', 'CS101').
teaches('234', 'CS102').
teaches('345', 'CS301').
/*Prerequisites – moduleNo, moduleNo*/
prerequisites('CS301', 'CS101').
prerequisites('CS301', 'CS102').

```

A number of basic update functions might also be offered. We provide a fuller version of one of the update functions defined in the text.

```

enrolStudentModule(Student, Module, Year, Semester):-
    students(Student,A,B,C,D,E,F),
    modules(Module,X,Y,Z),
    NOT(moduleEnrolment(Student,Module,Year,Semester)),
    assert(moduleEnrolment(Student,Module,Year,Semester)).

```

9.11



SUMMARY

- A database system is a physical symbol system. Different data models have different capacities for fulfilling intelligent action. By intelligence in this context most people refer to the active nature of such databases
- Logic as a data model consists of data structures, operators and integrity rules all represented in the same uniform way, namely as axioms in a logic language. In this chapter we have discussed Datalog as such a logic language and have illustrated the key features of this approach
- The Datalog discussed in this chapter bears a close resemblance to the logic programming language Prolog. The main difference is that we assume that Datalog demonstrates persistence
- The notion of intelligence is a slippery concept. Newell and Simon define intelligence in terms of a physical symbol system (PSS). A PSS has the necessary and sufficient means for generating intelligent action
- Both relational and deductive databases can be considered as physical symbol systems; but only deductive databases have been referred to as being ‘intelligent’

- Because deductive databases contain an inferential component they are frequently discussed in the context of intelligent database systems. Intelligent databases have the possibility of significantly improving the productivity of information systems development and maintenance
- The idea of building intelligence into database systems will also prove important in contributing to the management of a number of other emergent properties of database systems: distribution, complex data and multiple media

9.12 ACTIVITIES

1. Write Datalog statements for the assertions: a share is an investment; a stock is an investment.
2. Write a Datalog statement to add deductive databases as a module to our database.
3. Given our academic database, how would the Datalog engine respond to the following query?
`takes(johnDavies, relationalDatabaseSystems)`
4. Given our current academic database, what would happen to the following query?
`module(X)`
5. Write a rule to implement the fact that a lecturer of a module is also the author of that module.
6. Given the database described in the chapter, run through the process of solving the query:
`teaches(L, relationalDatabaseSystems).`
7. Implement the update functions described in section 1.4 for the university database in Datalog.
8. Given the rule described in section 9.5.1, list all the results of running the query:
`manager(X, Y).`

9.13 REFERENCES

- Beynon-Davies, P. (2002). *Information Systems: An Introduction to Informatics in Organisations*. Basingstoke, Palgrave (formerly Macmillan).
- Gallaire, H. and G. Minker (1978). *Logic and Databases*, New York, Plenum Press.
- Gardarin, G. and P. Valduriez (1989). *Relational Databases and Knowledge Bases*. Reading, MA, Addison-Wesley.
- Newell, A. and H.A. Simon (1976). Computer science as empirical inquiry: symbols and search. *Comm. of ACM* 19(3).



- Reiter, R. (1984). Towards a logical reconstruction of relational database theory. In M.L. Brodie, J. Mylopoulos and J.W. Schmidt (Eds), *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*. New York, Springer-Verlag.
- Tsitchizris, D.C. and F.H. Lochovsky (1982). *Data Models*. Englewood Cliffs, NJ, Prentice-Hall.
- Winograd, T. and F. Flores (1986). *Understanding Computers and Cognition: A New Foundation for Design*. Norwood, NJ, Ablex Publishing.
- Winston, P.H. (1984). *Artificial Intelligence*. Reading, MA, Addison-Wesley.

CHAPTER 10

POST-RELATIONAL DATA MODEL

It's always useful to know where a friend-and-relation is, whether you want him or whether you don't.

Rabbit, *Pooh's Little Instruction Book*, inspired by A.A. Milne

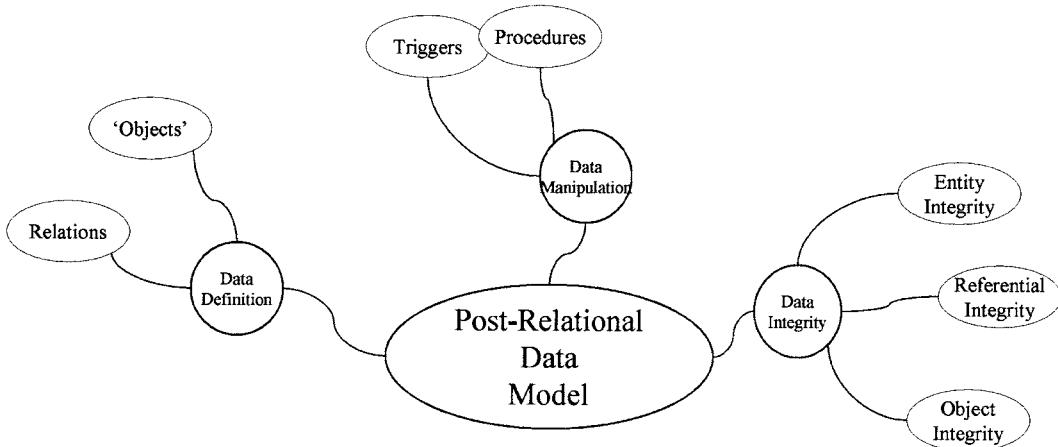
Hell, there are no rules here – we're trying to accomplish something.

Thomas A. Edison (1847–1931)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Distinguish between the relational and the post-relational data model
- Describe some of the key features of the post-relational data model: triggers, procedures and abstract data types



10.1 (C) INTRODUCTION

A number of extensions to the relational data model have been proposed in the three decades or so since its invention. Many of these extensions have been implemented in commercial DBMS. What is termed the post-relational data model here is not strictly a data model in that no coherent theory has been developed. Nevertheless it is useful to discuss it here in terms of a set of mechanisms found in many contemporary DBMS. Such a data model is also referred to by the terms extended-relational and object-relational data model. In Chapter 18 we discuss how the proposed SQL3 standard addresses many of these features. In Chapter 34 we also consider how the ORACLE DBMS supports some of these features.

In the first half of the chapter we consider two extensions to the data definition part of the relational data model: abstract data types and nested relations. In the second half of the chapter we consider two constructs – triggers and stored procedures – that have been used both for data manipulation and data integrity purposes. The incorporation of these features into a relational DBMS provides it with the ability to handle complex objects and behaviour. Hence many of the DBMS with these features have termed themselves object-relational systems.

10.2 (C) THE RELEVANCE OF OBJECT-RELATIONAL SYSTEMS

Michael Stonebraker (1996) has proposed a simplistic but useful characterisation of the relevance of different types of data management approach (see Figure 10.1):

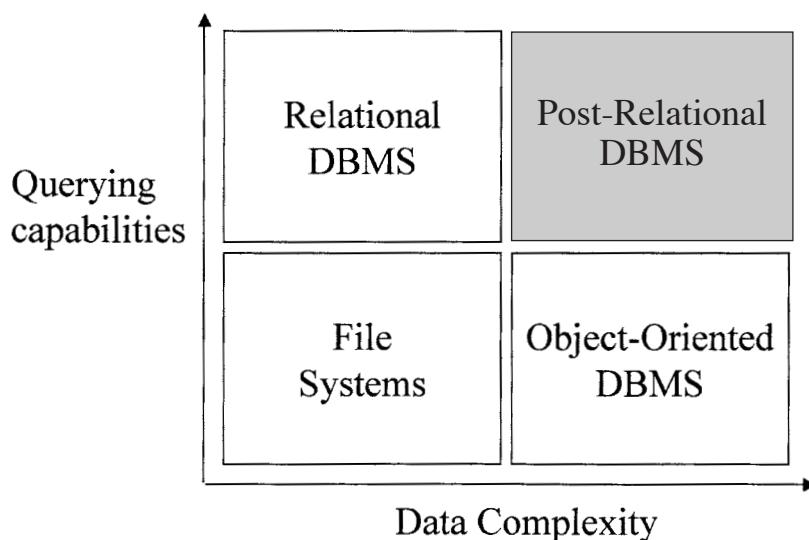


Figure 10.1 The place of the post-relational data model.

- In file systems the data is relatively simple in terms of structure and there is no real requirement for querying the data
- In relational systems the data is structurally simple but there is a heavy demand for querying
- In object-oriented database applications the data is usually inherently complex but there is usually no significant demand for querying the data
- In object-relational or post-relational systems the data may be complex but there is also an associated need for complex querying

This categorisation clearly portrays post-relational systems as an attempt to build on the strengths of both the relational and object-oriented (OO) approaches to managing data. It particularly seeks to extend the proven ability of relational DBMS to handle the data relevant to standard commercial information systems with the ability to manage complex data characteristic of OO DBMS.

10.3 DATA DEFINITION

In this section we consider how the concepts of an abstract data type and a nested relation can be added to the relational model to improve its object orientation.

10.3.1 ABSTRACT DATA TYPES

Most existing relational database systems provide only a limited number of data types for the user. For example, SQL-based systems (Part 3) provide integer, character, date and real data types, among others. These are usually sufficient for most standard data processing applications. However, for other applications, such as office automation, geographical information systems (Chapter 39) or computer aided design, these data types are not sufficiently rich. An office automation system, for instance, might want a document data type and a geographical information system might want a polygon data type. One solution to this problem is to extend the range of primitive data types offered to the user by the DBMS. However, the vast number of data types needed to support the infiltration of computing into more and more areas is probably too large to make this a practicable solution. A better approach is therefore to make the DBMS infinitely extendable by the user through user-defined and implemented abstract data types (ADT), sometimes called user-defined types (UDTs) (Figure 10.2).

An ADT or UDT is a type of object that defines a domain of values and a set of operations designed to work with these values. The objective of an ADT is to provide a characteristic known as information hiding. Implementation details of the ADT are hidden from higher levels of an application system. If the implementation of an ADT is changed, the upper layers of a system are

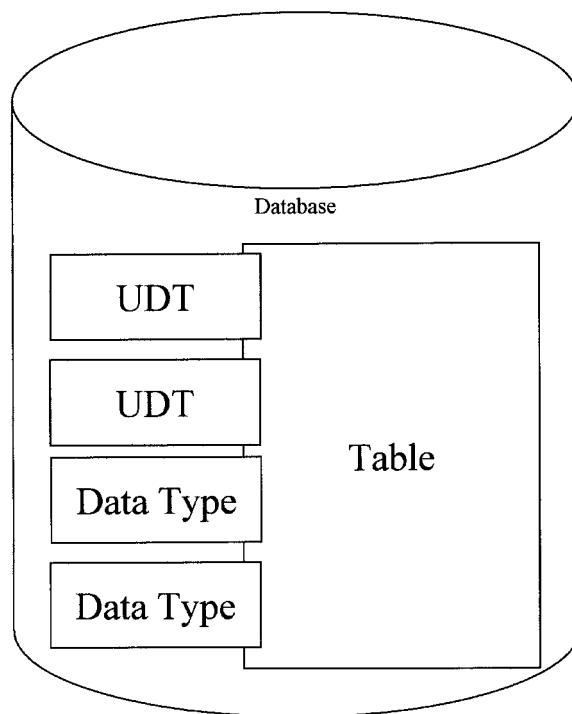


Figure 10.2 User-defined types.

unaffected since they communicate with the ADT only through a set of abstract operations composing its interface.

Example ➔ Suppose we have a clinical information system designed to store the written materials of a consultant psychiatrist. In such a system, a patient note might be a new ADT defined by the system designer. Associated with this new data type might be the operation count_note. This would count the number of words in a consultation note. Assuming that the number of words per consultation would give the psychiatrist some idea of the importance of the record, a query which extracted the health service numbers of patients seen after 30 June 2002 and ordered the results in descending order of the size of their consultation record might prove useful.

10.3.2 NESTED RELATIONS

Nested relations, sometimes known as NF 2 (non-first normal form) relations, have been proposed as a means of handling complex objects. A complex object can be defined as an object having a hierarchical structure whose attributes can be atomic or indeed complex objects themselves. In other words, an object can have any internal complexity we wish to define.

Example To continue with the psychiatry example, suppose that PatientHistory was a relation in our database. Such a nested relation might be represented as follows:

PatientHistory						
	<i>patientNo</i>	<i>patientName</i>	<i>DOB</i>	<i>consultation</i>		
				<i>date</i>	<i>time</i>	<i>treatment</i>
						<i>treatmentCode</i> <i>duration</i>
2312	Poulson J		7/4/70	1/2/2003	09:00	I/XS 30
				7/2/2003	09:30	A/XS 30
4377	Gearing P		4/3/73	1/2/2003	10:00	L/TP 70
				2/2/2003	10:00	L/TP 30
				3/2/2003	10:00	L/TP 47

Like normal relations, nested relations have a list of column names. However, a column name in a nested relation may refer to a group of attributes, each of which may also refer to a group, and so on. Hence, consultation in the table above refers to the group of attributes date, time and treatment. The attribute treatment in turn refers to the group treatmentCode and duration.

To operate on such relations we need two in-built operations: nest and unnest. Nest takes a relation and an attribute name (or composite attribute) and returns a corresponding nested relation.

Example Hence, if we supplied the relation Modules and the attribute courseCode we would obtain a nested relation with two attributes: courseCode and a composite group (moduleName, level, staffNo).

R1			
<i>courseCode</i>			
	<i>moduleName</i>	<i>level</i>	<i>staffNo</i>
CSD	Relational Database Systems	1	234
	Relational Database Design	1	234
	Deductive Databases	3	347
	Object-Oriented Databases	3	347
	Distributed Database Systems	2	237
BSD	Intro to Business	1	123
	Basic Accountancy	1	147

Unnest is the opposite of nest. Applying unnest to the table above would produce the original relation.

10.4 PASSIVE AND ACTIVE DATABASE SYSTEMS

Many of the extensions to the relational data model have been proposed to enable more functionality to be placed within the database itself. This functionality was described in abstract terms in Chapter 1 as a set of update functions associated with the database. To refresh, update functions can be used to encapsulate both constraints and events that cause changes to the state of some database.

A database with embedded update functions may be referred to as an active database. It is active in the sense that it may perform operations traditionally performed in application systems external to the database. This idea of an active database may be contrasted with the traditional idea of a database as a passive repository for data. This distinction is illustrated in Figure 10.3.

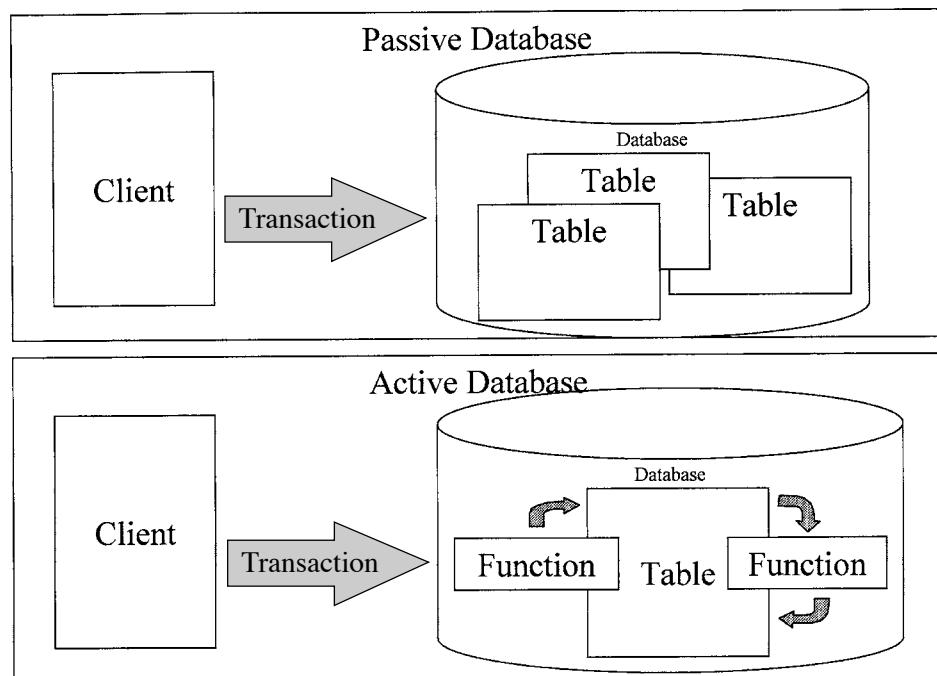


Figure 10.3 Passive and active databases.

Triggers and stored procedures are two mechanisms available for implementing integrity constraints and update functions and embedding these directly within the database.

10.4.1 TRIGGERS

Triggers are units of logic embedded in a database. Triggers are event-driven (see Figure 10.4). The logic is triggered by transactions which seek to update the database. A trigger is inherently associated with a single table within the database and is activated when insertions, updates or deletions are commanded against rows in that table. Such events are said to 'fire' a trigger.

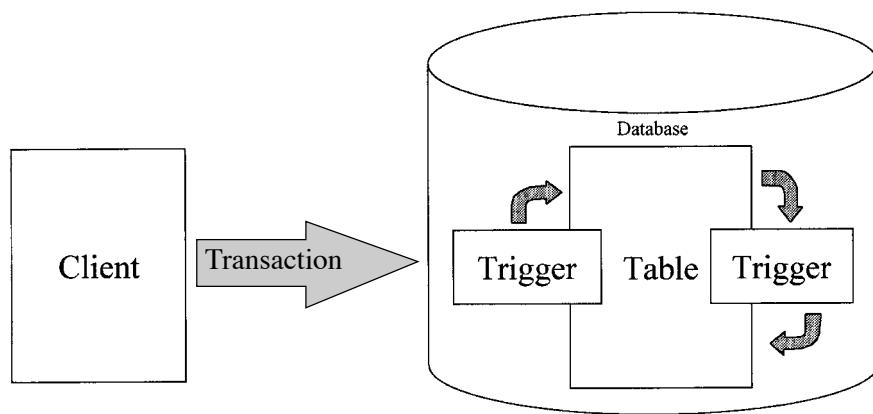


Figure 10.4 Triggers.

Triggers are used to perform a number of different database functions:

- Validating input data and maintaining complex integrity constraints: certain integrity constraints may not be declared using the inherent integrity mechanisms of a given data model, and triggers may be used to program such constraints
- Use as alerters – in other words, indicating actions that have been performed on the database to particular users
- Maintaining audit information for database administrators
- Supporting distribution actions such as the automatic replication of data across a network

Examples ➔

- *Validating input data.* For instance, in terms of the academic database we might use a trigger to check that a lecturer teaches no more than eight modules in an academic session
- *Use as alerters.* Hence, a trigger might be used to flag to school administrators when a particular module is full



- *Maintaining audit information for database administrators.* Triggers might be used by a DBA to record information about updates made to a particular table in the database
- *Supporting distribution actions.* A trigger may be used to automatically update a desktop system with information off the central student information system at regular intervals

10.4.2 STORED PROCEDURES

Stored procedures are also units of logic embedded in the database. However, unlike triggers, stored procedures are relatively autonomous units of logic (see Figure 10.5). A stored procedure may be called from an application program, from another stored procedure or from a trigger.

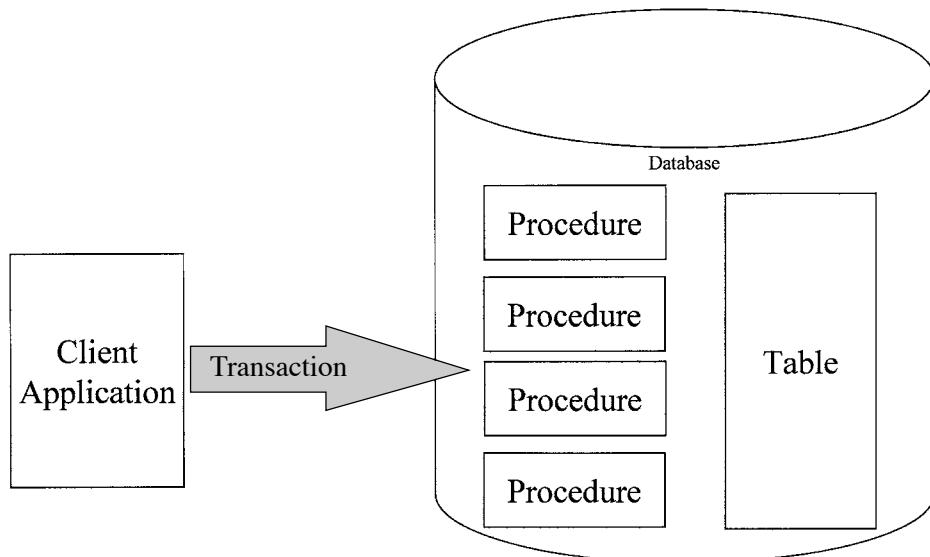


Figure 10.5 Stored procedures.

Example ➔

In terms of our academic database, stored procedures might then be used to: offer modules, cancel modules, enrol students on courses, enrol students in modules or transfer students between modules. In other words, all of the update functions defined for our academic database.

10.4.3 ADVANTAGES AND DISADVANTAGES OF TRIGGERS AND PROCEDURES

There is some debate about the appropriate use of triggers and stored procedures.

Advantages

Triggers and procedures have been developed to fulfil an active database need in contemporary applications. There are hence clear advantages to the use of these constructs such as:

- *Centralisation, standardisation and reuse of application logic.* The advantages of centralising replicated data (Chapter 3) also apply to replicated processes or code
- *Siting processing appropriately on the database server.* In client-server architectures (Chapter 36), siting applications on powerful servers can reduce the traffic along communication lines
- *Improving the productivity of application building.* The reuse of centralised database logic can enhance the productivity of the development of information systems

Disadvantages

Triggers and procedures are rather recent inventions in database history. The database development industry is still learning about the appropriate use of these constructs. Some possible disadvantages are associated with the modern use of triggers and procedures:

- There is some evidence that a system built substantially out of triggers and stored procedures may be difficult to manage. It may prove difficult to design, implement and perform administration tasks with active databases
- There are certain problems of transparency. Moving functionality to the database has the tendency of hiding functionality from the user. The database, not the user, controls much of the activity in an active database
- There is currently no standard syntax for writing stored procedures or triggers. Hence, there is the inherent concern of being tied-in to a particular DBMS vendor if an organisation chooses to write most of its application logic as triggers and procedures
- Using triggers and stored procedures normally has a performance implication. As the number of triggers increases then the processing overhead increases. For this reason, many systems have mechanisms for enabling and disabling triggers when performance proves an issue

10.5 TABLES AS OBJECTS

It is possible to treat post-relational extensions to the relational data model in a uniform way as having elements of an object-based approach. Note we call it an object-based, not an object-oriented approach. We can treat a table with extensions as an object but we usually cannot treat it in the same way as objects defined in Chapter 8.

A table with its associated attributes, constraints, triggers and procedures can be mapped on to the object-oriented concepts discussed in Chapter 8. Attributes of a table can map to the attributes of an object. Nested relations also allow all the attributes of an object to be kept together, not fragmented through normalisation. Stored procedures and triggers can, with a little imagination, be envisaged as the methods of the associated object. Tables can also be included in generalisation hierarchies with subtables inheriting the attributes, constraints, triggers and procedures of its supertable.

10.6 CASE STUDY: STORED PROCEDURES IN AN ACADEMIC DATABASE

Stored procedures can be used to implement some or all of the update functions against a database. In terms of a database system used at a university (see Case Study of Chapter 7), the following update functions might be appropriate. We have expressed these functions in a sort of pseudo-code of conditions and actions, similar in nature to the rules of the deductive data model (Chapter 9).

```

OfferModule(ModuleCode M, ModuleName N, . . .)
  Conditions: NOT(Modules(M));
  Actions: INSERT INTO Modules(M,N, . . .)

CancelModule(ModuleCode M)
  Conditions: Modules(M);
  Actions: DELETE Modules(M)

RegisterStudentDetails(StudentNo S, StudentName N, . . .)
  Conditions: NOT(Students(S));
  Actions: INSERT INTO Students(S,N, . . .)

EnrolStudentOnCourse(StudentNo S, CourseCode C)
  Conditions: Students(S); Courses(C); NOT (CourseEnrolment(S,C));
  Actions: INSERT INTO CourseEnrolment(S,C)

EnrolStudentOnModule(StudentNo S, ModuleNo M, CourseCode C)
  Conditions: Students(S); Modules(M); Courses(C);
  NOT (ModuleEnrolment(S,M)); CourseEnrolment(S,C);
  Actions: INSERT INTO ModuleEnrolment(S,M)

TransferStudentBetweenCourses(StudentNo S, CourseCode C1, C2)
  Conditions: Students(S); Courses(C1); Courses(C2);
  CourseEnrolment(S,C1); NOT (CourseEnrolment(S,C2));
  Actions: DELETE CourseEnrolment(S,C1); INSERT INTO CourseEnrolment(S,C2)

```

TransferStudentBetweenModules(StudentNo S, ModuleNo M1, M2)*Conditions:* Students(S); Modules(M1); Modules(M2);

ModuleEnrolment(S,M1); NOT (ModuleEnrolment(S,M2));

Actions: DELETE ModuleEnrolment(S,M1); INSERT INTO ModuleEnrolment(S,M2)**10.7** SUMMARY

- The post-relational or extended relational data model incorporates the following extensions to the relational data model: user-defined types, nested relations, triggers and stored procedures
- An abstract data type or user-defined data type is a type of object that defines a domain of values and a set of operations designed to work with these values
- Nested relations have been proposed as a means of handling objects having a hierarchical structure whose attributes can be atomic or indeed complex objects themselves
- Triggers are event-driven units of logic embedded in a database inherently associated with a table
- Stored procedures are autonomous units of logic embedded in the database

10.8 ACTIVITIES

1. Define what is meant by the term post-relational database system.
2. What features of a database can be said to make it active?
3. Distinguish between a stored procedure and a trigger.
4. In terms of an application known to you, investigate how much of it could be implemented using triggers and procedures.

10.9 REFERENCE

Stonebraker, M. (1996). *Object-Relational DBMSs: The Next Great Wave*. San Francisco, CA, Morgan Kauffman.

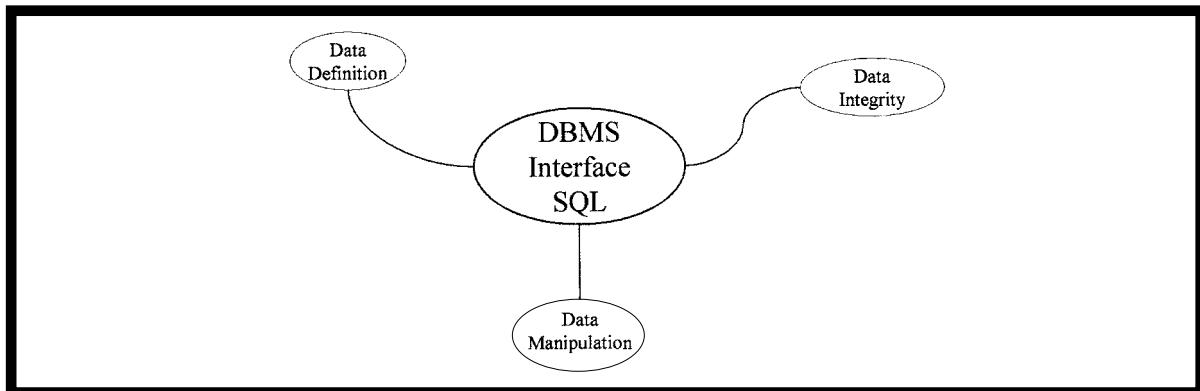
PART

3

DATABASE MANAGEMENT SYSTEMS (DBMS) – INTERFACE

Life is a foreign language; all men mispronounce it.

Christopher Morley (1890–1957)





In this part we discuss the common standard interface to most contemporary DBMS – SQL. Structured Query Language is a standard database sublanguage. A database sublanguage is a formal language for managing databases. In this part we consider the three major parts of this database sublanguage – data definition, data integrity and data manipulation – corresponding to the three parts of a data model discussed in Chapter 3. The issues of using SQL for data control and as an embedded language for use in application development are deferred to further parts.

- *Data definition.* Chapter 11 considers the various ways in which relational data structures, elements and items may be defined to form relational schemas
- *Data integrity.* Chapter 12 considers various approaches to implementing entity, referential and additional integrity using SQL
- *Data manipulation.* Chapter 13 devotes most attention to the issue of data retrieval using the Select command. It also considers ways of implementing the other CRUD operations using SQL

Figure P3.1 illustrates the position of the interface in relation to the DBMS kernel and toolkit.

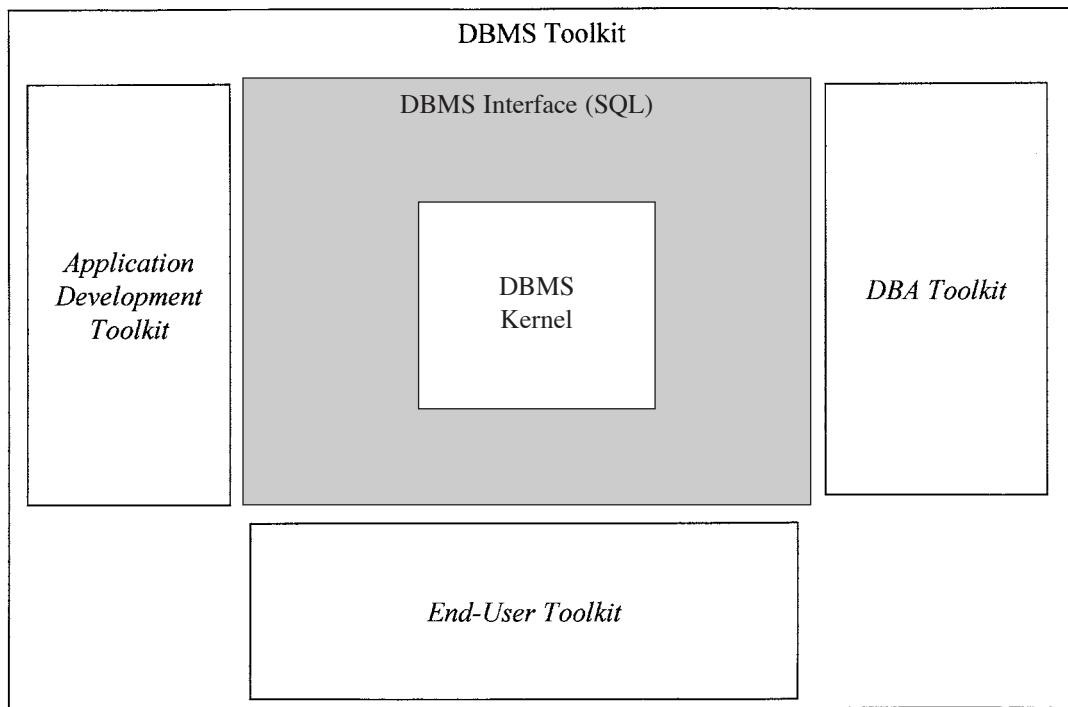


Figure P3.1 Kernel, interface and toolkit.

CHAPTER 11

SQL – DATA DEFINITION

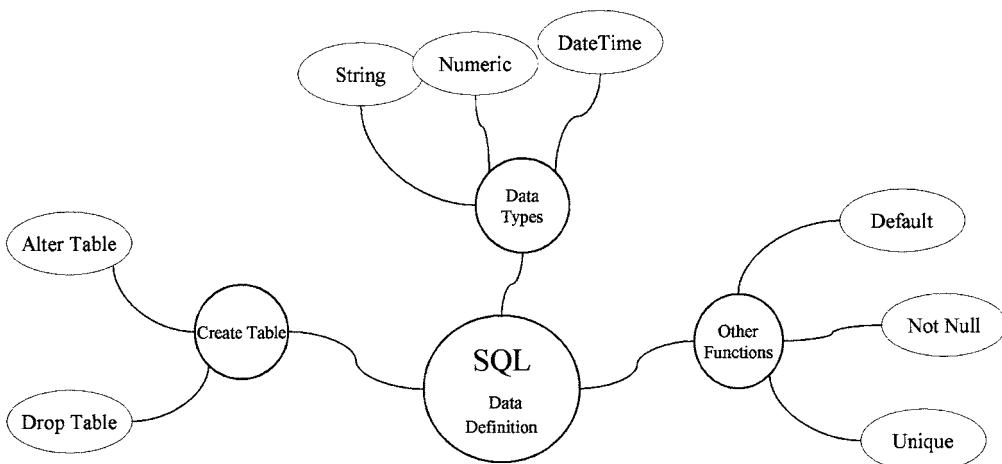
England and America are two countries separated by a common language.

George Bernard Shaw (1856–1950)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Describe the history of the standard database sublanguage SQL
- Define the common data types available in SQL
- Describe the key features of the create table statement
- Discuss mechanisms for defining schemas in SQL
- Discuss standardisation of system tables among DBMS



11.1 (C) INTRODUCTION

In the three chapters in this part of the book we contrast the theory of the relational data model as described in Chapter 7 against the contemporary practice of relational database systems. Contemporary practice is primarily centred around a language known as SQL (Structured Query Language). SQL was originally designed as a query language based on the relational calculus. However, the current specification of SQL makes it a lot more than simply a query language, and it is more accurately described as being a database sublanguage. It is a programming language but not a full application development language: a formal language designed specifically for database work. This database sublanguage has become the standard interface to relational, and non-relational, DBMS.

11.2 (C) THE HISTORY OF SQL

SQL has its origins in work done at the IBM research laboratory in San Jose, California during the early 1970s (Melton and Simon, 1993). Here a prototype implementation of relational concepts named System/R was built. This early relational DBMS embodied a language then known as SEQUEL – the name was changed later to SQL for legal reasons. This is why many people still refer to the SQL language as ‘sequel’.

During the years 1973 to 1979 IBM researchers published a great deal of material about the development of System/R in academic journals. This period was characterised by intense discussion about the validity of relational DBMS at conferences and seminars both in the USA and in Europe. IBM however was undoubtedly slow to see the commercial relevance of relational systems. It fell to the ORACLE Corporation, founded in 1977, to first exploit successfully in the commercial world the ideas underlying the relational data model.

ORACLE was, and is, an SQL-based RDBMS. Many other vendors also produced systems that support SQL. For these reasons, in 1982 the American National Standards Committee gave its database committee (X3H2) the remit to develop a standard Relational Database Language (RDL). This committee finally produced a definition for a standard SQL syntax in 1986 primarily based on the IBM and ORACLE dialects of SQL. The International Standards Organisation followed suit with a publication of much the same standard in 1987 (ISO, 1987). This standard is also known as SQL1. The original ANSI document specifies two levels for SQL1: level one and level two. Level two is the complete SQL language. Level one is a subset of level two originally intended to act as the intersection of existing implementations.

Following its publication, a number of criticisms were made of the ANSI/ISO standard, most notably by database personalities such as E.F. Codd (Codd, 1988a, b) and C. Date (Date, 1987). Many people viewed the standard as being the lowest common denominator among implementations. Others saw the language as having more serious defects, particularly in its ability to address relational constructs.



In response to some of these criticisms, an addendum to the standard was published in 1989 by ANSI, primarily addressing a number of integrity enhancement features. Much of the material in this addendum was included in a working draft of a proposed second version to the standard, also published by ANSI in 1989. ISO, working in close collaboration with ANSI, published a document in the same year entitled ‘Database Language SQL with Integrity Enhancement’ (ISO, 1989).

In 1992 both ANSI and ISO brought out a complete specification for an extended version of SQL known as SQL2 – sometimes referred to as SQL92 (ISO, 1992). Two subsets were specified for this standard: entry level and intermediate level. Entry SQL2 is fundamentally the same as SQL1 with integrity enhancement. Additional features are incorporated in the Intermediate level. This is the standard fundamentally addressed in this chapter. Further substantial enhancements to the SQL2 standard were agreed and a version known as SQL3 was produced in 1999 (ISO, 1999a,b). We briefly review some of the proposed features of SQL3 in Chapter 31.

Thus, there is not one standard, but at least three (see Figure 11.1). This means that any given implementation of SQL may support all or part of the three versions of the standard. At the time of writing, most implementations support SQL1 and SQL89 but none currently support all the features of SQL2, although a number of DBMS support a significant number of features. This is one of the reasons why most vendor implementations are best regarded as dialects of the SQL standard. In other words, they find common ground in many respects, usually formulated around some definition of the core or level one of the SQL1 standard and the integrity enhancement feature of SQL89. In other respects they differ by not adhering to either the SQL1 or later standards in certain areas (data types is a good example). Also, some implementations offer additional facilities not included in the standard.

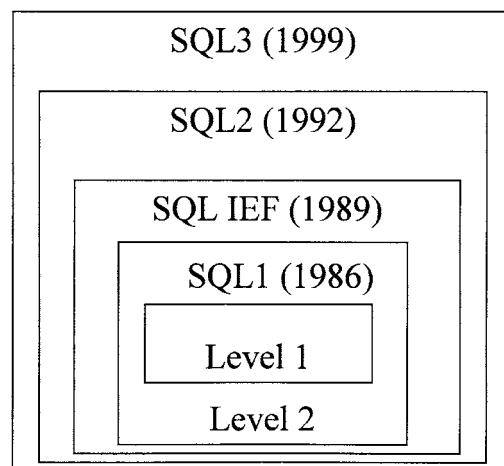


Figure 11.1 ISO/ANSI standards.

11.3 UNIVERSITY DATABASE

In the chapters in this part we shall use an extended version of the university database first introduced in Chapter 7 to illustrate the key features of data definition, data manipulation and data integrity in SQL. The relevant tables are displayed below:

Modules

moduleName	level	courseCode	staffNo
Relational Database Systems	1	CSD	234
Relational Database Design	1	CSD	234
Deductive Databases	3	CSD	345
Object-Oriented Databases	3	CSD	345
Distributed Database Systems	2	CSD	237
Intro to Business	1	BSD	123
Basic Accountancy	1	BSD	145

Lecturers

staffNo	staffName	status	deptName	salary
234	Davies T	L	Computer Studies	20000.00
237	Patel S	SL	Computer Studies	27500.00
345	Evans R	PL	Computer Studies	35500.00
123	Smith J	L	Business Studies	20000.00
145	Konstantinou P	SL	Business Studies	27500.00

11.4 A DATABASE SUBLANGUAGE

SQL is a standard database sublanguage for relational database systems (Chapter 7). In its latest standard – SQL3 – it is also a standard database sublanguage for post-relational database systems (Chapter 10). In this chapter we concentrate on those parts of the language devoted to fulfilling aspects of the relational data model. Post-relational features are covered in Chapter 31.

The SQL2 standard can be divided into four major parts:

- *Data definition.* Those commands concerned with declaring the structure of schemas and tables. Data definition is the topic of the current chapter
- *Data manipulation.* Those commands concerned with retrieving data and updating data in tables. This is the topic of Chapter 13
- *Data integrity.* Much of the idea of data integrity is attached to data definition in SQL. For convenience we consider those commands concerned with declaring entity, referential and domain integrity in Chapter 12
- *Data control.* Those commands concerned with administering the database are considered as part of the database administrator's toolkit in Chapter 26

There is also one command primarily concerned with the management of performance – Create Index – which is covered in Chapter 28.

11.5 THE CREATE TABLE STATEMENT

Data definition involves facilities for maintaining schemas and tables. To form a table using SQL the user needs to specify four components:

- Name of the table
- Name of each of the columns in the table
- Data type of each column
- Maximum length of each column

These four items are formulated together in a CREATE TABLE command having the following basic format:

```
CREATE TABLE <table name>
  (<column name> <data type> (<length>),
   <column name> <data type> (<length>),
   ..           ..           ..    )
```

Example → The statement below, for instance, demonstrates how the modules table might be created using some of the data types defined in the SQL2 standard and described in the next section:

```
CREATE TABLE Modules
  (moduleName CHARACTER(15),
   level SMALLINT,
   courseCode CHARACTER(3),
   staffNo INTEGER)
```

11.6 DATA TYPES

Data types act in part as a definition for domains (Chapter 7); they define certain properties concerning the allowable values for a column. Every data value within a column must be of the same type. The SQL standard defines some fifteen data types organised into the following groups:

11.6.1 STRING TYPES

String data consists of a sequence of characters from some defined character set such as ASCII or EBCDIC.



- *CHARACTER(N)*. A character string of fixed length. When a string less than the specified length (N) is input, spaces are added to the end of the string. Can be abbreviated to CHAR
- *CHARACTER VARYING(N)*. A character string of minimum length 1 and maximum length determined by the system. When a string less than the specified length (N) is input, only the actual length is stored. Can be abbreviated to VARCHAR
- *BIT(N)*. Bit strings are used to define a sequence of binary digits and are primarily used for graphics and sound data
- *BIT VARYING(N)*. Variable-length bit strings

Example ModuleName *CHARACTER(15)* specifies a fixed-length character string of 15 characters for the name of a module.

11.6.2 NUMERIC TYPES

There are two major types of numeric data: exact numeric data such as INTEGER and approximate numeric data such as REAL.

- *NUMERIC(M,N)*. A synonym for decimal
- *DECIMAL(M,N)*. A decimal number of length M with N places after the decimal point. Can be abbreviated to DEC
- *INTEGER*. An integer between the maximum system-defined values. Can be abbreviated to INT
- *SMALLINT*. An integer between a smaller range of system-defined values
- *FLOAT*. A number stored in floating-point representation
- *REAL*. A synonym for float
- *DOUBLE PRECISION*. A synonym for float

Example StaffNo *INTEGER* declares this identifier to be a standard integer.

11.6.3 DATETIME TYPES

Datetime data is used to define points in time to a specific level of accuracy. INTERVAL is not strictly a datetime data type as it is used to specify periods of time.

- *DATE*. System-defined dates
- *TIME*. System-defined times

- *TIMESTAMP*. Dates and times, including allowance for fractions of a second
- *INTERVAL*. Intervals between dates

Implementations differ in their support for these data types.

Example ➔ We do not use any such data types in our sample database. However, suppose we wished to add a start date for a lecturer. This might be defined as startDate DATE.

11.7 (C) NOT NULL AND UNIQUE

Any column in a table can be specified as being NOT NULL. This means that the user is then unable to enter null values into that column. The default specification for a column is null. That is, null values are allowed in a column.

Any column can also be defined as being UNIQUE. This clause prohibits the user from entering duplicate values into a column. The combination of NOT NULL and UNIQUE can be used to define the characteristics of a primary key (Chapter 7).

Example ➔ For instance:

```
CREATE TABLE Modules
(moduleName CHARACTER(15) NOT NULL UNIQUE,
level SMALLINT,
courseCode CHARACTER(3),
staffNo INTEGER)
```

11.8 (C) DEFAULT VALUES

A clause can be added to a column definition specifying the value a column should take in response to incomplete information being entered by the user.

Example ➔ For instance, a DEFAULT <value> specification can be added to the level column for modules indicating that the default level should be 1.

```
CREATE TABLE Modules
(moduleName CHARACTER(15) NOT NULL UNIQUE,
level SMALLINT DEFAULT 1,
courseCode CHARACTER(3),
staffNo INTEGER)
```


11.9
DROP TABLE

Table definitions can be created and table definitions can be deleted. To remove a table from the database we use the following command:

```
DROP TABLE <table name>
```

Example ➔ For instance:

```
DROP TABLE Modules
```

11.10
MODIFYING TABLES

To reiterate some of the discussion of Chapter 6, the ideal of data independence dictates that a database administrator should be allowed to modify the structure of a database without impacting on the users or application programs which access this database. In practice, SQL-based products support only a limited form of data independence. The database administrator is allowed to:

- Add an extra column to a table
- Drop a column from a table
- Modify the maximum length of an existing column
- Add a new table constraint (see Chapter 12)
- Drop a table constraint (see Chapter 12).
- Set a default for a column
- Drop a default for a column

Each operation is specified using the ALTER TABLE command.

Examples ➔ For instance:

```
ALTER TABLE Lecturers
ADD COLUMN roomNo SMALLINT
```

```
ALTER TABLE Lecturers
ALTER COLUMN staffName VARCHAR(20)
```

```
ALTER TABLE Lecturers
DROP COLUMN staffName
```

```
ALTER TABLE Modules
ALTER level SET DEFAULT 3
```

11.11 THE SYSTEM TABLES

Whenever a table is created, altered or dropped, updates occur to a set of system tables. These tables constitute a meta-database for use by the database administrator. However, the SQL1 and SQL89 standards do not define the structure of this meta-database. This means that the names given to the system tables, and indeed the schema of the system tables, are different in each relational DBMS. SQL2 has attempted to address this problem by introducing the idea of standard views of the system tables (Chapter 26).

11.12 SCHEMAS

In SQL2 each user may use many named schemas. A schema is defined using a *create schema* statement which allows tables and views to be declared part of the schema.

Example → We might create an academic schema with the modules table as a part and declare it for access by a user name AccDiv:

```
CREATE SCHEMA Academic AUTHORITY AccDiv
CREATE TABLE Modules(...)
```

Once a schema is declared, further tables or views may be defined as part of it using CREATE TABLE or CREATE VIEW (Chapter 26) statements. The name given to a table must be unique within a schema, but the same name can occur across schemas. To resolve naming conflicts a table name may need to be qualified by a schema name, either explicitly as in Academic.Modules, or implicitly by using the concept of a default schema. A default schema is associated with each session that a user has access to a database. Normally this is defined at logon time. However, a SET SCHEMA statement can be used to change the default schema.

Example → For instance:

```
SET SCHEMA Administration
```

Schemas can be dropped using the DROP SCHEMA statement. For instance:

```
DROP SCHEMA Academic
```

11.13 CATALOGS

A catalog is a named group of schemas. However, there is no create catalog statement defined in the standard. This is implementation-dependent. Schema names must be unique within a catalog and naming conflicts must be resolved by a similar process of qualification as for tables. In other words, the full name of a schema object is made up of three components: the catalog name, the schema name and the table name. Each catalog should contain a schema called INFORMATION_SCHEMA which defines a number of views for system tables with names such as TABLES and COLUMNS, providing access to information on defined tables and columns in the data dictionary.

11.14 CASE STUDY: INSURANCE POLICIES

Suppose we are an insurance company. Our business primarily revolves around insurance policies – effectively a financial product assuring risk between the holder of the policy and some events such as death, building damage or car accident. A basic part of the database of the insurance company is likely to be organised in the following fashion:

```
Policies(policyNo, holderNo, startDate, premium, renewalDate, policyType)
PolicyHolders(holderNo, holderName, holderAddress, holderTelno)
```

We may create two SQL data structures for this database in the following fashion:

```
CREATE TABLE Policies
  (policyNo CHARACTER(4) NOT NULL UNIQUE,
   holderNo CHARACTER(4) NOT NULL,
   startDate DATE,
   premium DECIMAL(8,2),
   renewalDate DATE,
   policyType CHARACTER(10))
CREATE TABLE PolicyHolders
  (holderNo CHARACTER(4) NOT NULL UNIQUE,
   holderName CHARACTER(20),
   holderAddress CHARACTER(50),
   holderTelNo CHARACTER(12))
```

11.15 SUMMARY

- SQL has become the dominant sublanguage for interacting with databases. A number of versions of an ANSI and ISO standard for SQL have been produced. The standard SQL3 addresses two major issues: enhanced relational capabilities and object-orientation. This is the topic of Chapter 31
- SQL has a data definition part, a data manipulation part and a data integrity part. There is also the issue of data control which is addressed in Chapter 26

- Data definition in SQL centres around the CREATE TABLE statement
- A column of a table can be defined on a number of existing data types
- A NOT NULL statement may be used to prohibit the entry of null values into a column
- A UNIQUE statement may be used to ensure unique values for a column
- Tables may be dropped from a database and certain aspects of their structure may be altered
- SQL2 permits the definition of schemas and catalogs

11.16 ACTIVITIES

1. Produce CREATE TABLE statements for the Goronwy Galvanising case specified in Chapter 3.
2. What data types are appropriate for each column in these tables and why?
3. Where is NOT NULL relevant?
4. Where is UNIQUE relevant?
5. Where is DEFAULT relevant?

11.17 REFERENCES

- Codd, E.F. (1988a). Fatal flaws in SQL. *Datamation* August: 45–8.
 Codd, E.F. (1988b). Fatal flaws in SQL: part 2. *Datamation* Sept.: 71–4.
 Date, C.J. (1987). Where SQL falls short. *Datamation* May: 83–6.
 ISO (1987). *Database Language SQL*. ISO/IEC 9075: 1987. Geneva, International Standards Organisation.
 ISO (1989). *Database Language SQL with Integrity Enhancement*. ISO/IEC 9075: 1989. Geneva, International Standards Organisation.
 ISO (1992). *Database Language SQL*. ISO/IEC 9075: 1992. Geneva, International Standards Organisation.
 ISO (1999a). *Database Language SQL – Part 2: Foundation*. ISO/IEC 9075–2. Geneva, International Standards Organisation.
 ISO (1999b). *Database Language SQL – Part 2: Persistent Stored Modules*. ISO/IEC 9075–4. Geneva, International Standards Organisation.
 Melton, J. and A.R. Simon (1993). *Understanding the New SQL: A Complete Guide*. New York, Morgan Kaufmann.

CHAPTER 12 →

SQL – DATA INTEGRITY

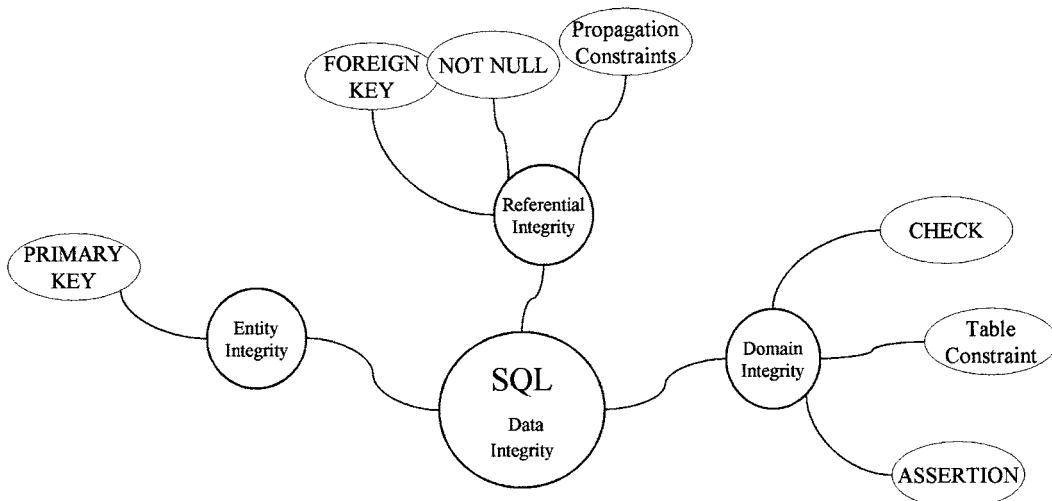
Integrity without knowledge is weak and useless, and knowledge without integrity is dangerous and dreadful.

Samuel Johnson (1709–84)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Describe the implementation of entity integrity using SQL
- Relate the implementation of referential integrity using SQL
- Describe the implementation of domain integrity using SQL
- Discuss the implementation of table constraints using SQL



12.1 INTRODUCTION

In this chapter we discuss the range of SQL commands available for implementing the inherent integrity of the relational data model – entity, referential and domain integrity. SQL3 (Chapter 31) contains commands which can be used to implement additional integrity in either a procedural or a declarative manner.

Up until the early 1990s most self-styled relational DBMS were deficient in implementing the inherent integrity of the relational data model. However, SQL89 (ISO, 1989) offers a means of declaring primary and foreign keys and declarations for domains. Most DBMS have now implemented these facilities. However, only some DBMS have implemented commands for declaring domains explicitly.

12.2 ENTITY INTEGRITY

Entity integrity concerns primary keys. Entity integrity is an integrity rule which states that every table must have a primary key and that the column or columns chosen to be the primary key should be unique and not null.

In SQL89 we can add a primary key specification to a *create table* statement. There are two ways of doing this. First, we can include a clause next to the appropriate column.

Example 

```
CREATE TABLE Modules
  (moduleName CHARACTER(15) PRIMARY KEY,
   level SMALLINT,
   courseCode CHARACTER(3),
   staffNo NUMBER(5))
```

This declares moduleName to be the primary key of the Modules table.

Second, we may add a clause at the end of the table definition.

Example 

```
CREATE TABLE Lecturers
  (staffNo NUMBER(5),
   staffName VARCHAR(15),
   status VARCHAR(10),
   deptName VARCHAR(20),
   salary DECIMAL(7,2),
   PRIMARY KEY (staffNo))
```

This clause declares staffNo to be the primary key of the Lecturers table.



If two or more columns of a table act as a compound primary key then this may be declared as a sequence delimited with commas.

Example → Suppose we wished to implement the table with the following structure:

```
ModuleAssessment(moduleNo, studentNo, assessmentNo, grade)
```

We would declare the primary key to be:

```
PRIMARY KEY (moduleNo, studentNo).
```

It is important to recognise that if a primary key clause is added to a table definition then the clauses unique and not null as described in Chapter 11 do not have to be used to enforce these properties of a primary key.

12.3

REFERENTIAL INTEGRITY

Referential integrity concerns foreign keys. The referential integrity rule states that any foreign key value can only be in one of two states. The usual state of affairs is that the foreign key value refers to a primary key value of some table in the database. Occasionally, and this will depend on the rules of the business, a foreign key value can be null. In this case we are explicitly saying that either there is no relationship between the objects represented in the database or that this relationship is unknown.

Referential integrity is achieved in SQL89 via foreign key specifications. The foreign key specified below identifies the associated table.

Example →

```
CREATE TABLE Modules
  (moduleName CHARACTER(15),
   level SMALLINT,
   courseCode CHARACTER(3),
   staffNo NUMBER(5),
   PRIMARY KEY (moduleName)
   FOREIGN KEY (staffNo REFERENCES Lecturers))
```

Here the staffNo column is declared as a foreign key to the primary key of the Lecturers table – the column staffNo.

```
CREATE TABLE Lecturers
  (staffNo NUMBER(5),
   staffName VARCHAR(15),
   status VARCHAR(10),
   deptName VARCHAR(20),
   salary DECIMAL(7,2),
   PRIMARY KEY (staffNo))
```

By examining the Modules table in Chapter 7 we can verify that each Module has a staffNo value, and that each staffNo value corresponds to a value existing in the Lecturers table. In this case we can say that the relationship between module and lecturer is mandatory. Every module is assigned to a lecturer. No module record has a null staffNo value.

Example → To enforce this refinement of our foreign key specification we need to add a not null clause to the foreign key definition as below:

```
CREATE TABLE Modules
  (moduleName CHARACTER(15),
   level SMALLINT,
   courseCode CHARACTER(3),
   staffNo NUMBER(5) NOT NULL,
   PRIMARY KEY (moduleName)
   FOREIGN KEY (staffNo REFERENCES Lecturers))
```

12.4 (C) PROPAGATION CONSTRAINTS

Maintaining referential integrity in a relational database is not simply a case of defining when a foreign key should be null or not. It also entails defining propagation constraints. A propagation constraint details what should happen to a related table when we update a row or rows of a target table (Chapter 7).

Example → The foreign key also has two propagation constraints defined. The specifications below make no action to a module's staffNo if an associated lecturer record is attempted to be deleted. It also specifies that any change made to the staff number of a lecturers record should be reflected in all relevant module rows.

```
CREATE TABLE Modules
  (moduleName CHARACTER(15),
   level SMALLINT,
   courseCode CHARACTER(3),
   staffNo NUMBER(5) NOT NULL,
   PRIMARY KEY (moduleName)
   FOREIGN KEY (staffNo REFERENCES Lecturers)
   ON DELETE NO ACTION
   ON UPDATE CASCADE)
```

Propagation options (called referential actions) in SQL2 (ISO, 1992) are:

- **NO ACTION.** This corresponds to the restricted option discussed in Chapter 7. It rejects the delete action on the table containing the primary key reference by the foreign key. This is the default setting in the absence of a specified ON DELETE constraint



- **CASCADE.** This corresponds to the cascades option discussed in Chapter 7. It causes automatic deletion of associated rows in the foreign key table following deletion of the row or rows in the primary key table
- **SET DEFAULT.** This causes deletion of the row in the primary key table followed by setting the foreign key values to the default specified on the foreign key
- **SET NULL.** This corresponds to the nullifies option specified in Chapter 7. Following deletion of rows in the primary key table it sets the associated foreign key values to null

12.5

DOMAIN INTEGRITY

As we mentioned in Chapter 7, domains are an important concept in the relational data model. SQL2 (ISO, 1992) offers a weak form of support for domains. A certain degree of domain integrity can be specified using data types. A check clause can also be added to enforce valid updates. The format of the CHECK command is:

`CHECK <search condition>`

Hence any search condition that may be used in a SELECT command (Chapter 13) can be used in a CHECK.

Example → We can enforce that a value entered into the level column is within a specified set or that staff numbers lie within a given range, as below:

```

CREATE TABLE Modules
  (moduleName CHARACTER(15),
  level SMALLINT,
  courseCode CHARACTER(3),
  staffNo NUMBER(5),
  PRIMARY KEY (moduleName)
  FOREIGN KEY (staffNo identifies Lecturers)
  ON DELETE NO ACTION
  ON UPDATE CASCADE,
  CHECK (level IN 1,2,3))

CREATE TABLE Lecturers
  (staffNo NUMBER(5),
  staffName VARCHAR(15),
  status VARCHAR(10),
  deptName VARCHAR(20),
  salary DECIMAL(7,2),
  PRIMARY KEY (staffNo),
  CHECK (staffNo BETWEEN 10 AND 9999))

```

In SQL2 a domain represents a set of permissible values. A domain is defined in a schema and is identified by a domain name. The primary purpose of a domain is to constrain the set of valid values that can be stored in a column.

A domain definition may specify a data type, a domain constraint and a default clause. Domains are declared using a CREATE DOMAIN statement.

Example → CREATE DOMAIN Levels AS INTEGER(1),
DEFAULT 1,
CHECK (VALUE BETWEEN 1 AND 3)

Domain names can then be used instead of data types in column definitions.

Example → level Levels

An ALTER DOMAIN statement is also defined which can be used to add a new constraint, set a new default value or drop a constraint or default value from an existing domain definition. There is also a DROP DOMAIN command.

12.6 TABLE CONSTRAINTS

A table constraint is a named constraint clause which forms part of a table definition. The advantage of naming a constraint in this manner is that such a constraint can be dropped from a table at some later date using the ALTER TABLE command (Chapter 11). Table constraints can also be added to a table using this command.

Example → CREATE TABLE Modules
(moduleName CHARACTER(15),
level Levels,
courseCode CHARACTER(3),
staffNo NUMBER(5)
CONSTRAINT StaffQuotaCheck
CHECK(NOT EXISTS(SELECT staffNo
FROM Modules
GROUP BY staffNo
HAVING COUNT(*)>6)),
PRIMARY KEY (moduleName)
FOREIGN KEY (staffNo identifies Lecturers)
ON DELETE NO ACTION
ON UPDATE CASCADE)



This table constraint (named StaffQuotaCheck) runs a check on the Modules table to make sure that the member of staff is not teaching more than six modules before allowing entry of a new staffNo into a Modules row.

12.7 ASSERTIONS

A constraint can be named and defined independently from any table or domain. In this case, the constraint is known as an assertion.

Example We can declare an assertion (which we name StaffNoCheck) independently of the Lecturers table to perform a check against any updates made of this table:

```
CREATE ASSERTION StaffNoCheck
CHECK (staffNo BETWEEN 10 AND 999)
```

12.8 CONSTRAINT VIOLATION

Having the facility to name constraints means that every time an SQL statement inserts, updates or deletes a row of a table there exists the possibility that a constraint might be violated. SQL89 requires that constraints should be checked for violation at the end of the execution of each statement. SQL2 allows the capability of checking constraints on completion of a transaction. If a constraint is checked after each statement it is said to be in immediate mode. If it is checked at the end of a transaction then it is said to be in deferred mode.

Each constraint definition can hence include a specification of whether the constraint is DEFERRABLE or NOT DEFERRABLE. The initial mode of a constraint can be specified as either INITIALLY DEFERRED or INITIALLY IMMEDIATE. This mode can then be changed during processing by a SET CONSTRAINTS statement which specifies that a list of named constraints be deferred or immediate.

12.9 CASE STUDY: INSURANCE POLICIES

We can add integrity to the schema produced in Chapter 11 in the following manner:

```
CREATE TABLE Policies
(policyNo CHARACTER(4) PRIMARY KEY,
holderNo CHARACTER(4) NOT NULL,
startDate DATE,
```

```

    premium DECIMAL(8,2),
    renewalDate DATE,
    policyType CHARACTER(10)
    FOREIGN KEY holderNo REFERENCES PolicyHolders
    ON UPDATE CASCADE)
CREATE TABLE PolicyHolders
    (holderNo CHARACTER(4) PRIMARY KEY,
    holderName CHARACTER(20),
    holderAddress CHARACTER(50),
    holderTelNo CHARACTER(12))

```

We might also create a domain for the policyType column in the following manner:

```

CREATE DOMAIN PolicyTypes AS CHARACTER(10),
DEFAULT 'life',
CHECK (VALUE IN ('car', 'contents', 'buildings', 'life'))

```

It is probably also important to enforce business rules such that a policy's startDate should be less than a policy's renewalDate. This we might do using an assertion or a table constraint. For instance:

```

CREATE ASSERTION DateCheck
CHECK (NOT EXISTS(SELECT policyNo FROM Policies WHERE renewalDate >=
startDate))

```

12.10 SUMMARY

- Data integrity primarily concerns the specification of inherent integrity
- Inherent integrity in the relational data model means entity integrity, referential integrity and domain integrity
- Entity integrity is implemented in SQL through the PRIMARY KEY CLAUSE
- Referential integrity is implemented in SQL through the FOREIGN KEY CLAUSE
- A number of propagation constraints are defined for referential integrity in SQL
- Domain integrity can be specified using CHECK, table constraints and/or assertions
- Named constraints can be deferrable

12.11 ACTIVITIES

1. Declare primary keys for the full academic schema discussed in the Case Study of Chapter 7.
2. Declare foreign keys for this schema.



3. The university wants to enforce the business rule – all students must have paid their fees before they can enrol on a course. How do we enforce this rule in the schema?
4. Declare suitable domains for a number of the columns in this schema using the create domain statement.

12.12

REFERENCES

- ISO (1989). *Database Language SQL with Integrity Enhancement*. ISO/IEC 9075: 1989. Geneva, International Standards Organisation.
- ISO (1992). *Database Language SQL*. ISO/IEC 9075: 1992. Geneva, International Standards Organisation.

CHAPTER 13 →

SQL – DATA MANIPULATION

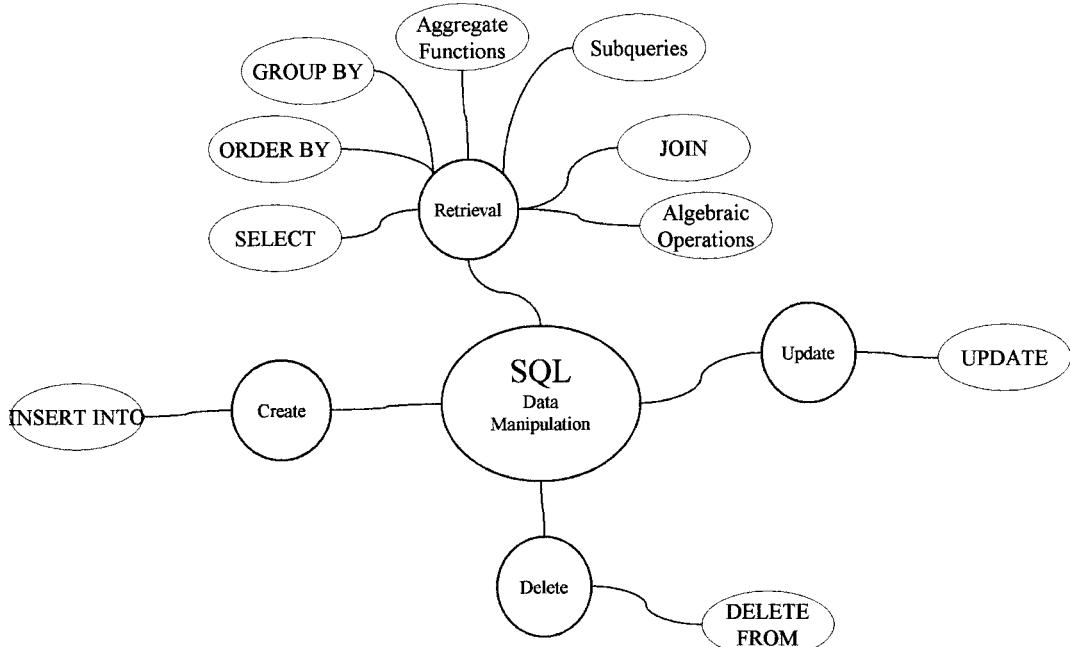
Expletive deleted

Submission of recorded presidential conversations to the House of Representatives by President R.M. Nixon, 30 April 1974

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Describe the key features of the Select statement
- Describe the key commands for insertion, update and deletion of data in SQL



13.1 INTRODUCTION

Data manipulation concerns the CRUD activities associated with a database:

- *Create* – inserting rows into a table
- *Retrieval* – accessing data within tables
- *Update* – updating data within existing rows of tables
- *Delete* – deleting rows from tables

SQL was originally designed primarily as a means for extracting data from a database – as a retrieval or query language (Melton and Simon, 2002). Such extraction is accomplished through use of the Select command: a combination of the *restrict*, *project* and *join* operators of the relational algebra (Chapter 7). SQL also includes commands for the create, update and delete operations.

13.2 SIMPLE RETRIEVAL

Simple retrieval is accomplished by a combination of the *select*, *from* and *where* clauses (ISO, 1987):

```
SELECT <attribute1 name>, <attribute2 name>, ...
FROM <table name>
[WHERE <condition>]
```

The *select* clause indicates the table columns to be retrieved. The *from* clause defines the tables to be referenced. The *where* clause indicates a condition or conditions to be satisfied. For instance, the following command is a direct analogue of the relational algebra *restrict*. The asterisk (*) acts as a wildcard. That is, all the attributes in the Modules table are listed.

Example

```
SELECT *
FROM Modules
```

moduleName	level	courseCode	staffNo
Relational Database Systems	1	CSD	234
Relational Database Design	1	CSD	234
Deductive Databases	3	CSD	345
Object-Oriented Databases	3	CSD	345
Distributed Database Systems	2	CSD	237
Intro to Business	1	BSD	123
Basic Accountancy	1	BSD	145



Note that the example above has no *where* clause. The *where* clause is optional. If we omit the *where* clause then all the rows of a table are considered by the query. The addition of a *where* clause restricts the retrieval to a set of rows matching a given condition.

Example →

```
SELECT *
FROM Modules
WHERE moduleName = 'Deductive Databases'
```

<i>moduleName</i>	<i>level</i>	<i>courseCode</i>	<i>staffNo</i>
Deductive Databases	3	CSD	345

If we specify column names, then the SQL *select* becomes a form of relational algebra project.

Example →

For instance:

```
SELECT level
FROM Modules
```

<i>level</i>
1
1
3
3
2
1
1

Note however, that since this table has duplicate values, it is not a relation. Duplicate values are legal in SQL, but they are illegal as far as the relational data model is concerned. To produce a true relational response to the query above we must add the keyword *distinct* to the *select* clause. This removes duplicate entries in a table.



Example ➔ For instance:

```
SELECT DISTINCT level
FROM Modules
```

<i>level</i>
1
2
3

13.3 ORDER BY

In Chapter 7 we discussed how there is no explicit ordering of rows in a relation. Ordering of data can therefore only be produced as a result of applying some processing to a relation. To produce a sorted list as output we add the *order by* clause to the *select* statement.

Example ➔ For instance:

```
SELECT *
FROM Modules
WHERE courseCode = 'CSD'
ORDER BY level
```

<i>moduleName</i>	<i>level</i>	<i>courseCode</i>	<i>staffNo</i>
Relational Database Systems	1	CSD	234
Relational Database Design	1	CSD	234
Distributed Database Systems	2	CSD	237
Deductive Databases	3	CSD	345
Object-Oriented Databases	3	CSD	345

The default order is ASCII or EBCDIC ascending. To produce the list in descending order we add the keyword DESC to an order by clause.

Example ➔ For instance:

```
SELECT *
FROM Modules
WHERE courseCode = 'CSD'
ORDER BY level DESC
```

13.4 GROUP BY

To undertake aggregate work, such as computing the average salary of lecturers in a particular department, we use the *group by* clause.

Example ➔

```
SELECT deptName, avg(salary), count(*)
FROM Lecturers
GROUP BY deptname
```

deptName	avg(salary)	count(*)
Computer Studies	27666.00	3
Business Studies	23750.00	2

This statement effectively organises the data into groups based on the specified column or group of columns and allows aggregate processing against each group, in this case computing the average salary of each group and counting the number of elements in each group.

A *group by* clause can also have its own ‘where’ qualifier – GROUP BY HAVING.

Example ➔ Hence, the following statement retrieves only those departments in our database having more than two lecturers:

```
SELECT deptName, count(*)
FROM Lecturers
GROUP BY deptname
HAVING count(*) > 2
```

deptName	count(*)
Computer Studies	3



13.5

AGGREGATE FUNCTIONS

Five aggregate functions are defined in the standard:

- *COUNT*. Returns the number of values in a column
- *SUM*. Returns the sum of the values in a column
- *AVG*. Returns the average of the values in a column
- *MIN*. Returns the minimum value of the set of values in a column
- *MAX*. Returns the maximum value of the set of values in a column

We have already seen the use of *avg* and *count* above. The statement below demonstrates the application of *max* and *min*.

Example

```
SELECT deptname, max(salary), min(salary)
FROM Lecturers
GROUP BY deptName
```

deptName	max(salary)	min(salary)
Computer Studies	35500.00	20000.00
Business Studies	27500.00	20000.00

We can change the name of the output column by using an AS clause.

Example

We can change the aggregate column headings into more meaningful names as follows:

```
SELECT deptname, max(salary) AS Maximum_Salary, min(salary) AS Minimum_Salary
FROM Lecturers
GROUP BY deptName
```

deptName	Maximum_Salary	Minimum_Salary
Computer Studies	35500.00	20000.00
Business Studies	23500.00	16500.00

13.6

SUBQUERIES

The word *structured* in Structured Query Language originally referred to the ability to nest queries in select statements. A nested query is called a



subquery. Subqueries can be nested to an arbitrary number of levels. SQL evaluates the subqueries in order from the innermost query to the outermost query. The result from the innermost query is passed as a value to the outermost query.

We may distinguish between three types of subquery in terms of the type of result returned by the subquery:

- *Scalar subquery*. This returns a single value (row/column intersection) as a result. Hence, scalar subqueries tend to be used with operators such as $>$, $<$, \neq , $=$
- *Row subquery*. This returns multiple columns from a single row
- *Table subquery*. This returns a table (multiple rows and columns) as a result. A table subquery can be used with an operator such as IN. IN expects a set of values to return

Example ➔ To find out who makes more money than Jones, we would write a scalar subquery such as:

```
SELECT staffNo, staffName
FROM Lecturers
WHERE salary >
  (SELECT salary
   FROM Lecturers
   WHERE staffName = 'Jones S')
```

SQL is frequently criticised for its redundancy. By this is meant that there is frequently more than one way of expressing a given ‘natural’ query in SQL. For instance, it is frequently the case that many natural queries expressed using subqueries can equally well be expressed by using a join (Chapter 7).

Example ➔ An example of a table query using IN might be to find out all the module names of modules taught by senior lecturers:

```
SELECT moduleName
FROM modules
WHERE staffNo IN
  (SELECT staffNo
   FROM Lecturers
   WHERE status = 'SL')
```



13.7

CORRELATED SUBQUERIES

In the previous examples the subquery was executed once. The resulting value was then used in the *where* clause of the outermost query. A subquery can however execute repeatedly. In this case it produces a series of values to be matched against the results of the outermost query.

Example → Consider the following request: *List the staff name, department name and salary of all lecturers who earn more than the average salary of their department.*

```
SELECT staffName, deptName, salary
FROM Lecturers L
WHERE salary >
  (SELECT avg(salary)
   FROM Lecturers
   WHERE L.deptname = deptname)
```

The letter L in the clause where L.deptname = deptname acts as a reference from the subquery to the outermost query. Thus L.deptname refers to lecturers in the outermost query. Deptname refers to lecturers in the subquery. This is effectively like manipulating two copies of the same table. One copy is used to calculate average salaries, the other is used as the basis of comparison for each lecturer.

Hence, a column or table within the context of a query can be given another name known as an alias, such as in the case of L for lecturers above. An alias is hence appended immediately after the table name.

13.8

ANY AND ALL

The keywords *any* and *all* can be used with any table subqueries that return a single column of numbers. If the subquery is preceded by the keyword *all* then the condition will only be true if satisfied by all the values returned from the subquery. If the subquery is preceded by the keyword *any* then the condition will be true if it is satisfied by one or more of the values returned from the subquery. The ISO standard allows the keyword *some* to be used in place of *any*.

Example ➔ An example of a table query using ANY might be to find all those staff whose salary is larger than at least one member of staff in the Computer Studies department:

```
SELECT staffName
FROM lecturers
WHERE salary > SOME
    (SELECT salary
     FROM Lecturers
     WHERE deptName = 'Computer Studies')
```

13.9 ⚡ JOINS

In a relational database data is held in a set of fragmented but related tables. The data is related through common data held in the primary key and foreign key columns of tables. Joins are the way in which data held in such separate but related tables are brought together in one set so that particular retrieval operations can be performed on the whole set of data.

SQL performs relational joins by indicating common attributes in the *where* clause of a *select* statement. The particular condition (or conditions) used to specify a join is then known as a join condition. The join condition normally specifies the link between a primary key and a foreign key.

Example ➔ The *select* statement below extracts data from the *Lecturers* and *Modules* tables:

```
SELECT staffName, salary, moduleName
FROM Lecturers L, Modules M
WHERE L.staffNo = M.staffNo
```

In the example above the join condition is *L.staffNo = M.staffNo*. This indicates the link between the primary key of the *Lecturers* table and the foreign key held in the *Modules* table.

13.10 ⚡ EXISTS AND NOT EXISTS

The keywords *exists* and *not exists* are designed to be used specifically with subqueries. They are described as Boolean functions in the sense that they return either a true or a false result. *Exists* is set to true only if there exists at least one row in the result from a subquery. *Not exists* is set to true only if the result is the empty set.



Example → A query to find all those modules taught by computer staff could be written as:

```
SELECT moduleName
FROM modules M
WHERE EXISTS
  (SELECT *
   FROM Lecturers L
   WHERE M.staffNo = L.staffNo and deptName = 'Computer Studies')
```

13.11

UNION, INTERSECT AND EXCEPT

These three operators implement the union, intersection and difference operators of the relational algebra (Chapter 7).

The *union* operator essentially constitutes a way of combining the results of two compatible queries.

Example → The query below produces a result combining information about computer studies degree modules (courseCode = 'CSD') with those in the electrical engineering department (courseCode = 'EED'):

```
SELECT moduleName, level
FROM Modules
WHERE courseCode = 'CSD'
UNION
SELECT moduleName, level
FROM Modules
WHERE courseCode = 'EED'
```

The *intersect* operator constitutes a way of finding those rows common to the results from two queries.

Example → The query below produces a result detailing those modules that are run both under the computer studies programme and under the electrical engineering programme:

```
SELECT moduleName, level
FROM Modules
WHERE courseCode = 'CSD'
INTERSECT
SELECT moduleName, level
FROM Modules
WHERE courseCode = 'EED'
```

The *except* operator produces a result of all those rows in one result that are not present in the other result.

Example → The query below produces a result detailing those modules that are run under the computer studies programme and not under the electrical engineering programme:

```
SELECT moduleName, level
FROM Modules
WHERE courseCode = 'CSD'
DIFFERENCE
SELECT moduleName, level
FROM Modules
WHERE courseCode = 'EED'
```

13.12 ⚡ ALGEBRAIC OPERATIONS IN SQL2

The main rationale for producing SQL2 (ISO, 1992) as a replacement for SQL1 has been to remove unnecessary restrictions and generalise operations within the database sublanguage as much as possible. This is evident in the way the relational algebraic operators *join*, *union*, *intersect* and *except* (difference) are used in the new standard.

Example → A join operator can be used in a FROM clause:

```
SELECT moduleName, staffName
FROM Lecturers NATURAL JOIN Modules
```

This will join the two tables on the primary–foreign key references stored in the table definitions, assuming that the join column has the same name in the different tables. If the name is different, for instance staffNo in one table and staffCode in another table, then we would need to rewrite the query as something like the example below.

Example →

```
SELECT moduleName, staffName
FROM Lecturers L JOIN Modules M
ON L.staffNo = M.staffCode
```

SQL2 also declares standard syntax for outer joins.



Examples Left, right and two-way outer joins would be specified in the following ways in SQL2:

```

SELECT *
FROM Lecturers L LEFT OUTER JOIN Modules M
ON L.staffNo = M.staffCode

SELECT *
FROM Lecturers L RIGHT OUTER JOIN Modules M
ON L.staffNo = M.staffCode

SELECT moduleName, staffName
FROM Lecturers L FULL OUTER JOIN Modules M
ON L.staffNo = M.staffCode

```

13.13

INSERT INTO, UPDATE AND DELETE FROM

The type of data manipulation we have devoted primary attention to in this chapter has been passive in nature. In other words, we have assumed a database of facts which we have endeavoured to ask questions of.

There are however three other data manipulation commands in SQL which have a more active role. In other words, they are designed to enact changes to the state of the database itself. These commands are *insert into*, *update* and *delete from*.

Note that the end-user would not normally be expected to use these commands to manipulate data. Instead, these commands would be issued from a more user-friendly interface to a database system.

13.13.1 THE *INSERT INTO* COMMAND

The simplest form of *insert into* is used to add an extra row to a given table.

Example The statement:

```

INSERT INTO Modules
VALUES ('Intro to Management',1,'BSD',123)

```

will add an extra row to the modules table.

The order in which values are listed in the *insert into* statement should correspond to the order in which the columns were originally specified for the table in the create table statement. If we wish to list the values in some order other than that originally specified, or omit some columns from the insertion, then we must add a list of column names to the insert statement.

Example ➔ For instance:

```
INSERT INTO Modules (level, courseCode, staffNo, moduleName)
VALUES (2,'CSD', 237,'Information Systems Development')
```

A variation of the *insert into* command allows us to add multiple rows to a table. This is normally used as a means of loading up a given table with the results of some query.

Example ➔ Suppose we wish to create a table of lecturers from the Computer Studies department. This we might do as follows:

```
CREATE TABLE ComputerLecturers
(staffNo NUMBER(5),
staffName VARCHAR(15),
status VARCHAR(10),
salary DECIMAL(7,2))

INSERT INTO ComputerLecturers(staffNo, staffName, status, salary)
SELECT staffNo, staffName, status, salary
FROM Lecturers
WHERE deptName = 'Computer Studies'
```

13.13.2 THE UPDATE COMMAND

The *update* command is used to modify the contents of one or more rows of a table. The relevant rows are specified by an optional *where* clause and the change or changes to be made to the rows are specified by a *set* clause.

Example ➔ The following command will increase the salary of all PL (principal lecturer) lecturers by 10%:

```
UPDATE Lecturers
SET salary = salary * 1.1
WHERE status = 'PL'
```

13.13.3 THE DELETE FROM COMMAND

The *delete from* command is used to remove rows from a table. The actual rows to be deleted are specified by a *where* clause similar to that used in the *select* statement.

Example ➔ DELETE FROM Lecturers
WHERE deptName = 'Computer Studies'

13.14 CASE STUDY: INSURANCE POLICIES

In Chapters 11 and 12 we established the structure of the insurance company's database to be as follows:

```
Policies(policyNo, holderNo, startDate, premium, renewalDate, policyType)
PolicyHolders(holderNo, holderName, holderAddress, holderTelno)
```

Suppose we wish to insert new holder information into the PolicyHolders table of this database. This would be done in the following manner:

```
INSERT('4324', 'P. Beynon', 'Anywhere Drive', '081434692') INTO PolicyHolders
```

We could then update this row in the following way:

```
UPDATE PolicyHolders WHERE holderNo = '4324' SET holderTelNo = '071434692'
```

We also might delete a particular policy holder in the following way:

```
DELETE PolicyHolders WHERE holderNo = '4324'
```

A query to retrieve all standard life policies might be:

```
SELECT * FROM Policies WHERE policyType = 'life'
```

Likewise a statement to list the holder numbers and names of all persons holding standard life policies would be:

```
SELECT holderNo, holderName
FROM Policies P, Holders H
WHERE P.holderNo = H.holderNo AND policyType = 'life'
```

13.15 SUMMARY

- Data manipulation consists of the CRUD activites against a database
- Data manipulation in SQL centres around the SELECT statement
- Data can be ordered and grouped for application of aggregate functions
- The structure in SQL refers to the concept of a subquery
- Natural and outer joins can be performed using SQL
- SQL2 has specified a syntax for the relational algebraic operations

13.16  ACTIVITIES

1. In terms of the academic database discussed in the Case Study of Chapter 7, write a statement to insert new student data into the database.
2. Write a statement to update existing Lecturer data in the database.
3. Write a statement to delete existing Modules data from the database.
4. Write a *select* statement to list all students taking the module Relational Database Systems.
5. Write a *select* statement using a subquery to list all the students taught by lecturer 'Jones S'.
6. Write a *select* statement using a join to list all the students taught by lecturer 'Jones S'.
7. Write a statement to list the student numbers and names of all persons taking level 3 modules in the Computer Studies department.
8. Generate a query using one or more of the aggregate functions with the appropriate use of a group by clause.

13.17  REFERENCES

- ISO (1987). *Database Language SQL*. ISO/IEC 9075: 1987. Geneva, International Standards Organisation.
- ISO (1992). *Database Language SQL*. ISO/IEC 9075: 1992. Geneva, International Standards Organisation.
- Melton, J. and A.R. Simon (2002). *SQL: 1999: Understanding Relational Language Components*. San Francisco, CA, Morgan Kaufmann.

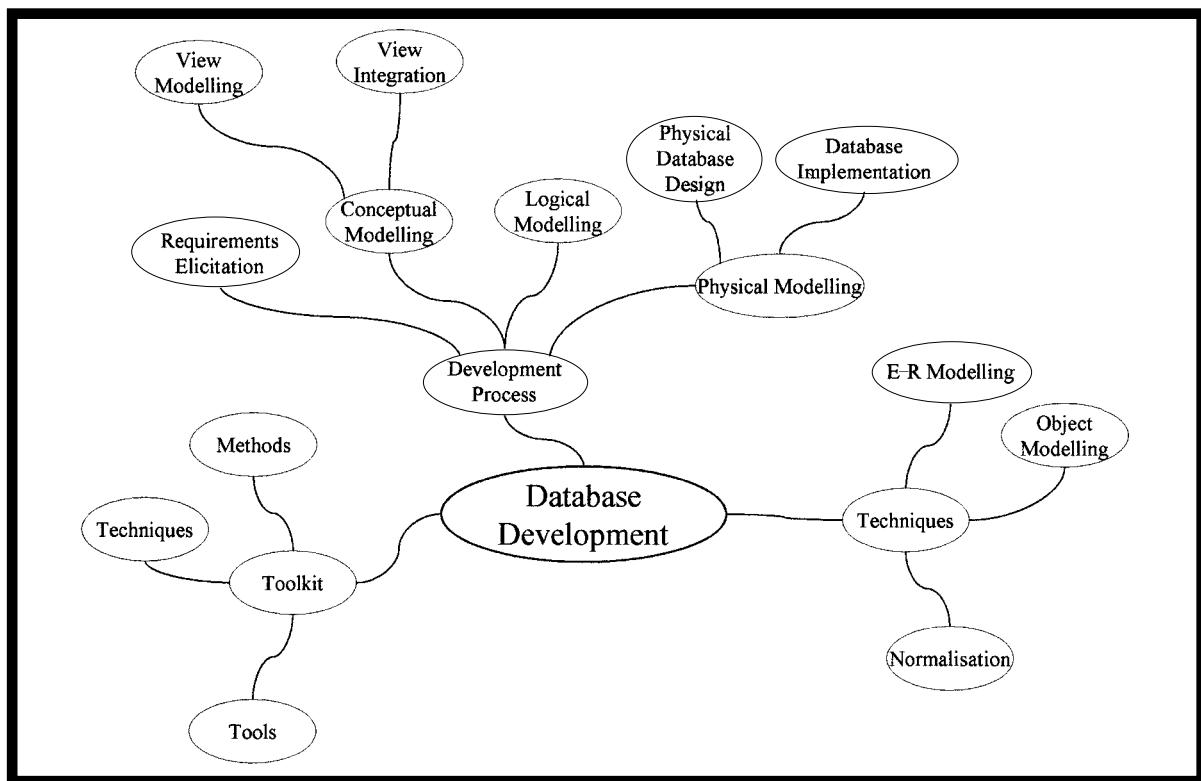
PART

4

DATABASE DEVELOPMENT

Inventions reached their limit long ago, and I see no hope for further development.

Julius Frontinus, 1st Century AD





Database development is that process concerned with producing a database system to meet the needs of a particular organisation. In this part we consider some of the important elements of database development.

- *Development process and toolkit.* The development of database systems is normally undertaken as a subprocess within the development of some information system. Chapter 14 describes some of the critical features of the database development process. It particularly divides the development process into four major stages: requirements elicitation, conceptual modelling, logical modelling and physical modelling. These stages are headed by the process of requirements elicitation. The chapter also describes some of the components of the database development toolkit – methods, techniques and tools
- *Requirements elicitation.* Chapter 15 describes features of this process and reviews a number of important elicitation techniques
- *Conceptual modelling.* Because of its critical importance, two chapters are devoted to conceptual modelling techniques – Entity–Relationship Diagramming (Chapter 16) and Object Modelling (Chapter 17)
- *Logical modelling.* Chapter 18 is devoted to detailing the most used logical modelling technique – normalisation
- *Physical modelling.* The physical modelling stage may be divided into the process of physical database design and the process of database implementation. Chapter 19 covers the crucial issues in the physical design stage and Chapter 20 covers some important issues of implementation

CHAPTER 14

DATABASE DEVELOPMENT PROCESS

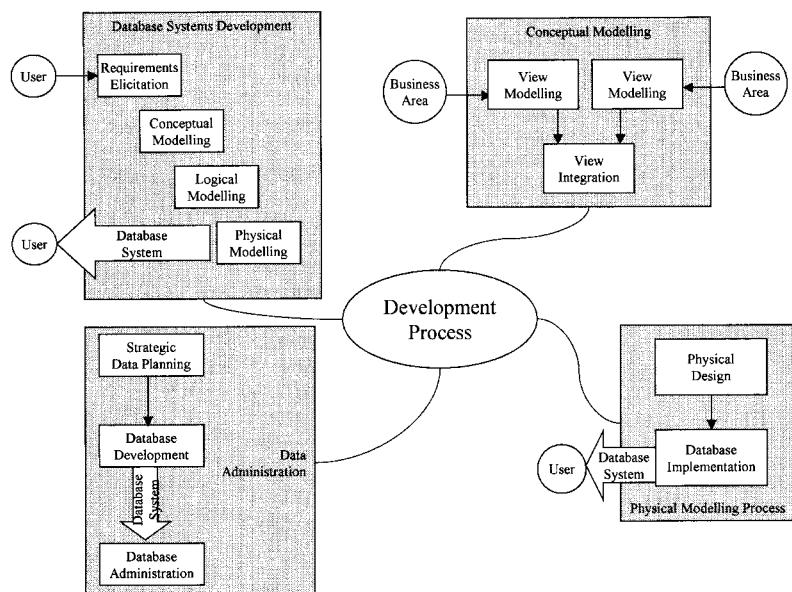
*It is best to do things systematically, since we are only human,
and disorder is our worst enemy.*

Hesiod (circa 800 BC)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define the key stages of the database development process
- Describe the key features of conceptual modelling
- Outline the key features of logical modelling
- Describe the key features of physical modelling
- Relate the key elements of the database development toolkit



14.1 DEVELOPMENT PROCESS AND TOOLKIT

Database development normally occurs within the context of information systems development. Information systems development is a key organisational process for many organisations (Beynon-Davies, 2002). Since it is a process it can be considered as a system in itself (Figure 14.1). The key inputs into the process are information and communications technology resources and developer resources. Information and communications technology resources may comprise systems construction tools or software packages. Developer resources include not only people but a toolkit which includes methods, techniques and tools available to the developer.

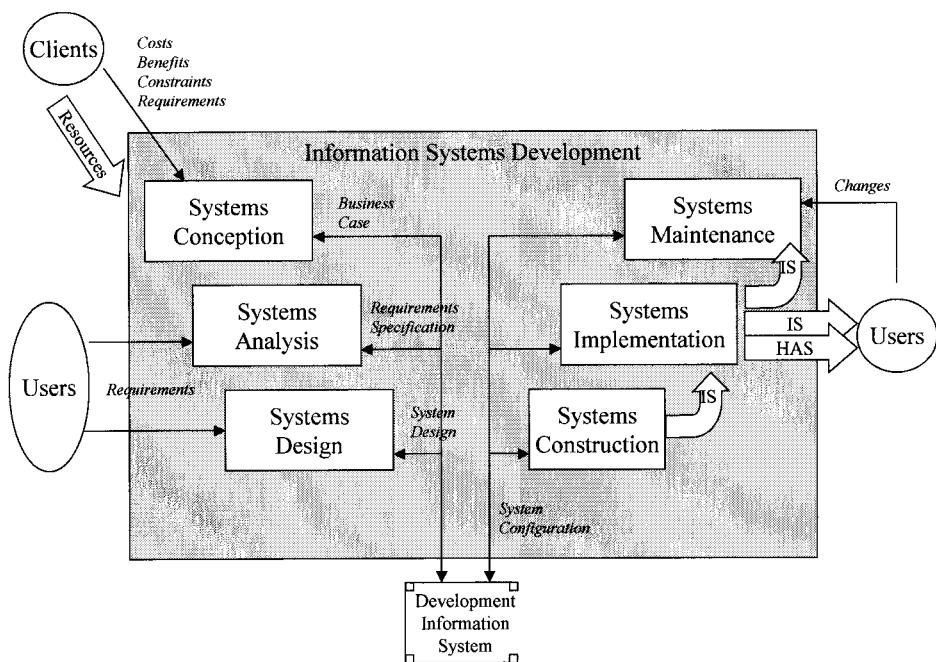


Figure 14.1 The information systems development process.

The key outputs are an information system (IS) and some associated human activity system. Information systems in themselves are socio-technical systems. They include both an information and communications technology system and a system of use. Ideally the human activity system and the information system should be designed in parallel. The ICT system produced may be a bespoke system or a configured/tailored software package.

A number of key activities are involved in the development process sometimes referred to collectively as the information systems development lifecycle. Such activities include systems conception, systems analysis, systems design, systems construction, systems implementation and systems maintenance.

A development information system is essential to ensure the effective and efficient operation of the human activity system which is the development process. Any reasonable-sized development project will need a consequent information system to support communication between teams of developers, to feed up into a project management and an informatics management process and to communicate with other stakeholder groups such as users and business managers. Such an information system consists of both system documentation and project documentation. System documentation acts as a model of the developing system; project documentation acts as a model of the development process. The key point here is that all information systems in a sense model elements of other systems. Such elements may be artefacts or organisational activities.

The development process is normally organised in terms of projects – organised units of activity consisting of teams with a defined goal and a finite duration. It is also normally undertaken by a suborganisation – the informatics service.

14.2 DATABASE DEVELOPMENT PROCESS

Figure 14.2 illustrates a simplified model of the database development process abstracted from a range of actual development approaches used for database systems (Teorey, 1994).

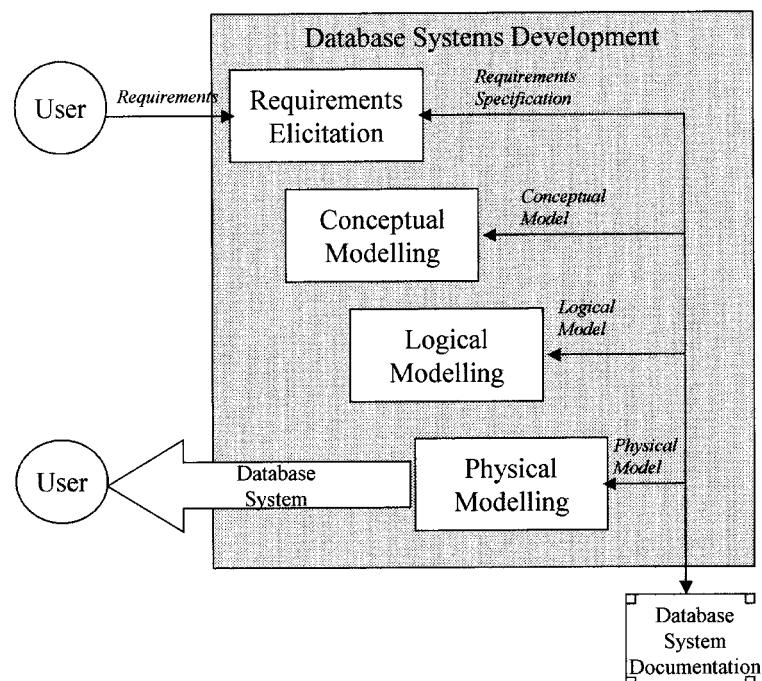


Figure 14.2 The database development process.



The process is divided into four main stages: requirements elicitation, conceptual modelling, logical modelling and physical modelling. The techniques used in the development process naturally divide into three categories: those concerned with conceptual modelling, those concerned with logical modelling and those concerned with physical modelling. The key input into the development process is a set of requirements for a database system; the key output is the final database and associated functions. Various forms of modelling will also input into the documentation accompanying a database system.

14.2.1 REQUIREMENTS ELICITATION

Requirements elicitation involves establishing the key technical requirements for a database system usually through formal and informal interaction between developers and organisational stakeholders such as users. In general terms it involves establishing the structure of data needed and the use of the data in some information systems context. One key aspect of requirements elicitation is the determination of the scope of the ‘universe of discourse’ (UoD) to be covered by a proposed database system. This is the topic of Chapter 15.

14.2.2 CONCEPTUAL MODELLING

Conceptual modelling involves building a model of the real world expressed in terms of the data requirements established (Silberschatz *et al.*, 2002). There are a number of alternative approaches to conceptual modelling. Chapters 16 and 17 cover two of the most popular approaches. We first discuss the well-established technique of entity–relationship (E–R) diagramming. This technique can be used to construct an entity model – a representation of a UoD in terms of entities, relationships and attributes. Then we discuss a series of extensions to the E–R technique which allows us to construct an object model of our chosen UoD – a representation expressed in terms of object classes. The output from the conceptual modelling process on the diagram in Figure 14.2 will therefore either be an entity model or an object model.

In Figure 14.3 we illustrate the fact that, particularly for large database development efforts, conceptual modelling may consist of two subprocesses: view modelling and view integration. View modelling involves developing an application data model for a particular business area. View integration involves taking a number of these distinct views and producing an integrated data model for some organisation.

14.2.3 LOGICAL MODELLING

Logical modelling involves constructing a model of the real world expressed in terms of the principles of some data model. Because of its popularity we focus on the relational data model in our discussion of logical modelling. Both Chapters 16 and 17 discuss how either an entity or an object model may be mapped to a relational schema. In Chapter 18 we consider the logical database

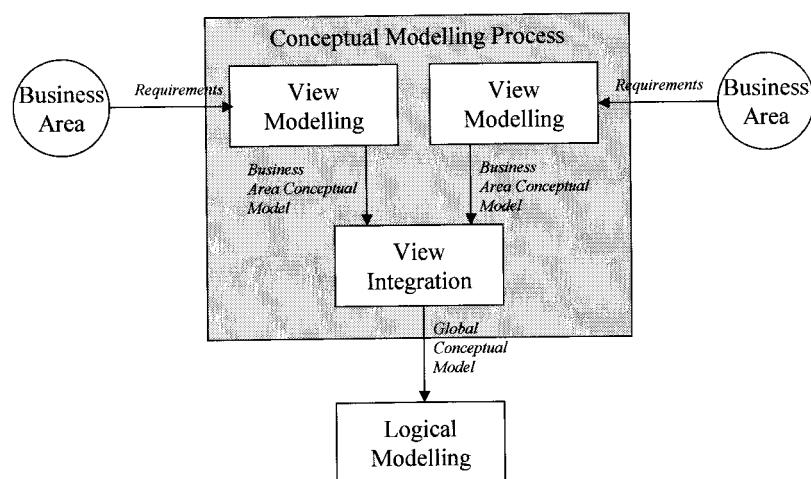


Figure 14.3 View modelling and view integration.

design technique of normalisation. This technique enables us to construct a relational schema free from update problems. The output from the logical modelling process on the diagram in Figure 14.2 is therefore likely to be a relational schema.

14.2.4 PHYSICAL MODELLING

Physical modelling involves constructing a model of the real world expressed in terms of data structures and access mechanisms available in a chosen DBMS. This phase in database development is illustrated in Figure 14.4. Physical

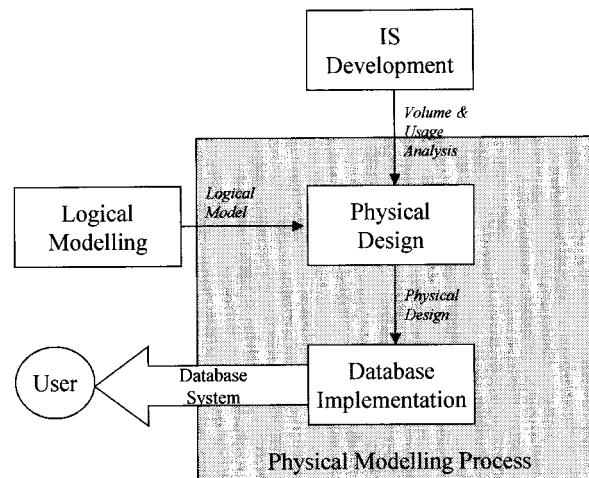


Figure 14.4 Physical modelling.



modelling involves two distinct subprocesses: physical database design and database implementation. Physical database design comprises the process of annotating a logical model with information pertaining to a particular application such as volume and usage information. This physical design process is the topic of Chapter 19. The output from the physical database design process is a database implementation plan. Database implementation involves taking the output from physical database design and implementing the design decisions contained in the plan in a chosen DBMS. This is the topic of Chapter 20.

14.3 KEY ACTIVITIES

We summarise the key activities involved in the database development process below:

Key Phase	Subphase	Activities	Alternative Activity
Requirements analysis			
Conceptual modelling	View modelling	Defining entities/objects Defining relationships and constraints on relationships Defining attributes Defining abstraction mechanisms Defining behaviour	
	View integration	Identifying communalities among views Producing a global conceptual model Accommodating the conceptual model to a relational schema	
Logical modelling	Normalisation	Producing a 3NF schema through non-loss decomposition Reconciling the normalised schema with the schema produced from conceptual modelling	Producing a 3NF schema through a dependency analysis

Key Phase	Subphase	Activities	Alternative Activity
Physical modelling	Physical database design	Volume analysis Usage/transaction analysis Integrity analysis Control/security analysis Distribution analysis	
	Database implementation	Selecting the DBMS Creating the physical schema Establishing storage structures and associated access mechanisms Adding indexes Denormalisation Defining users and privileges Tuning in terms of the chosen DBMS Building integrity constraints	

14.4 RELATIONSHIP WITH THE IS DEVELOPMENT LIFE-CYCLE

Figure 14.5 illustrates the correspondence between the standard model of the IS development life-cycle and the database development life-cycle. The traditional requirements elicitation and specification activities of the IS development process take place within systems analysis. This corresponds to the data requirements elicitation phase of the database development process plus

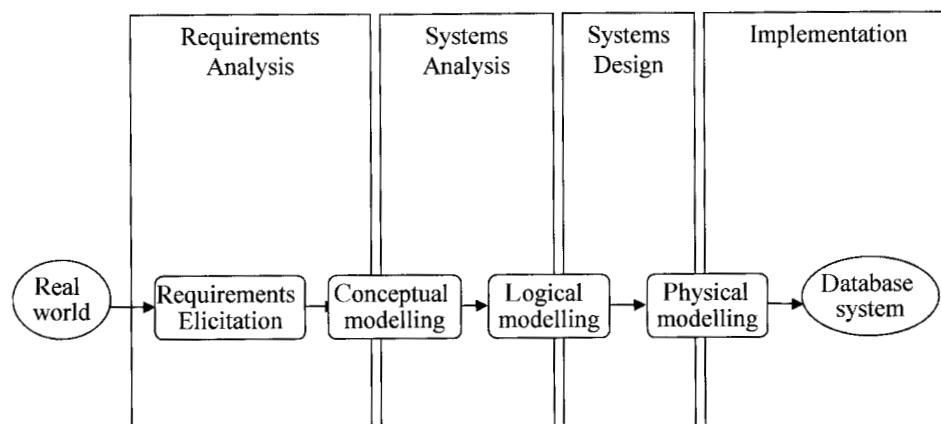


Figure 14.5 Relationship between the database development process and the IS development life-cycle.



conceptual modelling. Conceptual modelling, because of its key role in documenting data requirements at a high-level, also overlaps with the traditional idea of systems analysis. Logical modelling corresponds mostly to systems design in the sense of specifying logical schema and the incorporation of physical design decisions. Finally, physical design of database systems is one of the important processes relevant to modern-day information systems implementation.

Systems conception is the process of building a business case for an information system. This will overlap with the processes of strategic data planning and data administration referred to in the next section. Also, systems maintenance – the process of servicing an information system once in use – will correspond to the process of database administration.

14.5



DEVELOPMENT CONTEXT

A database development project occurs within a continuous flux of information systems development. Formulation of the need for a particular database project is likely to be formulated within a larger planning effort which will identify priorities for data areas and systems. This is the topic of strategic data planning covered in Chapter 21. Strategic data planning is one of the important activities of the data administration function covered in Chapter 21. One of the key inputs from strategic data planning into the database development process is likely to be elements of a corporate data model, a global plan for data requirements across the organisation. Also, following implementation, database administration needs to be put into place to maintain an effective and efficient database system. The context for a database development project is illustrated in Figure 14.6.

14.6



DATABASE DEVELOPMENT TOOLKIT

Humans are *Homo Habilis* – man the toolmaker. We make tools in order to aid in the construction of artefacts and to extend our physical and mental grasp. Therefore, to undertake any development effort the information systems or database system developer needs a toolkit. Such a toolkit will consist of methods, techniques and tools for supporting the activities of the development process.

Methods, techniques and tools are what we might call supporting ‘technology’ for information system development in general and database system development in particular. The term technology is used here in its broadest sense to refer to any form of device that aids the work of some person or group of persons.

- *Methods.* These constitute frameworks which prescribe, sometimes in great detail, the tasks to be undertaken in a given development process. Methods are used to guide the whole or a major part of the development process

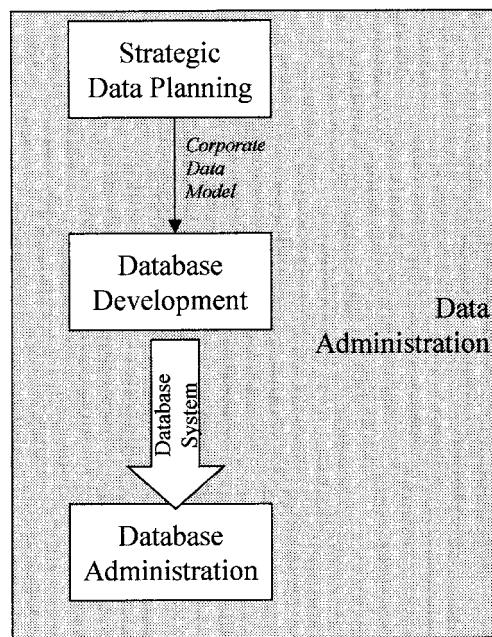


Figure 14.6 The context for database development.

- *Techniques*. These form the component parts of methods in that they constitute particular ways of undertaking given parts of the process. Techniques are normally used to guide activity within one phase of the development process
- *Tools*. By tools we primarily mean here available hardware, software, data and communication technology for engaging in some part of the information systems development process or its set of associated external activities. Tools are frequently used to support the application of particular techniques

Development methods comprise frameworks which prescribe development activity. An information systems development or database development method might be defined as being made up of the following primary components:

- A model of the development process
- A set of techniques
- A documentation method associated with these techniques. This consists of a notation for representing constructs and some principles for relating representations of different techniques
- Some indication of how the techniques chosen, along with the documentation method, fit into the model of the development process



Examples ➔ In a sense, most of this chapter has been devoted to describing a database development method. This method represents a hybrid of and a simplification of available database development methods.

Examples of techniques used in this method are entity–relationship diagramming.

Examples of tools include the DBMS itself and an application programming interface.

14.7 CASE STUDY: DEVELOPMENT OF A RESEARCH DATABASE SYSTEM

In Chapter 15 we describe a project (see Case Study) to produce a research database system for a UK university. The key activities engaged in on this project can be mapped onto the phases of the database development process:

Requirements elicitation. Data and processing requirements for the database system were gathered through a number of techniques, including documentary analysis, interviews and a joint requirements planning workshop. Incremental prototypes were produced throughout the development project.

Conceptual modelling. One main entity–relationship diagram was produced for the database system as illustrated in Figure 14.7.

Logical modelling. From the E–R diagram in Figure 14.7 a relational schema was produced. This was checked against a couple of normalisation exercises conducted on data-sets derived from existing documentation. For instance, a number of research grant documents used by the financial department of the university were analysed using determinancy diagramming to determine the following data requirements:

- A given research grant may have a number of grant holders – members of academic staff
- One of the grant holders is nominated as the principal grant holder against the research grant
- A given member of academic staff may hold more than one research grant at any one time
- A research grant is from one nominated funding body
- A given funding body may provide a number of research grants to the university at any one time

The schema is presented below in the bracketing notation:

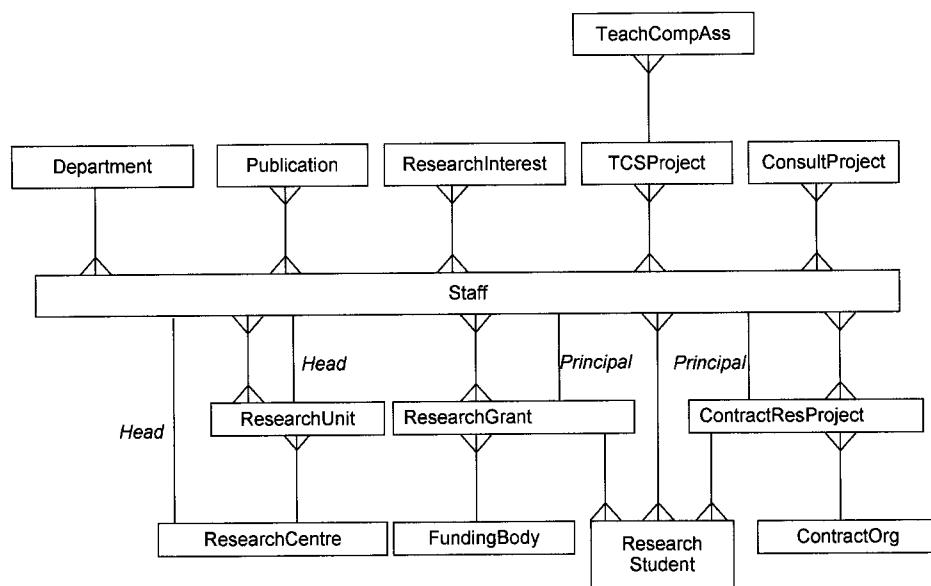


Figure 14.7 E-R diagram for research database system.

```

ResearchCentre(centreName, centreHead, ...)
ResearchUnit(unitName, unitHead, centreName, ...)
Staff(staffNo, departmentName, ...)
Publications(publicationNo, ...)
ResearchGrants(grantNo, fundingBody, principalApplicant, ...)
FundingBodies(fundingBody, ...)
ResearchInterests(interestNo, InterestDescription, ...)
ResearchStudents(studentNo, grantNo, contractNo, ...)
TCSProjects(TCSNo, ...)
ConsultancyProjects(consultNo, ...)
ContractResearchProjects(contractNo, contractOrg, principalHolder, ...)
ContractOrganisations(contractOrg, ...)
MembershipUnit(staffNo, unitName, ...)
Publishing(staffNo, publicationNo, ...)
GrantHolding(staffNo, grantNo, ...)
StaffInterests(staffNo, interestNo, ...)
GrantHolding(staffNo, grantNo, ...)
TCSHolding(staffNo, TCSNo)
ContractResearchHolding(staffNo, contractNo, ...)
ConsultHolding(staffNo, consultNo, ...)

```

Physical modelling. The final prototype for the system was produced in Microsoft Access and placed on a database server accessible from a number of client machines based in Academic registry. It was initially produced solely for use by five persons in Academic registry. Eventually the system was upsized



onto an SQL Server DBMS with a consequent larger server. The system is now available across the campus network through the University's intranet.

14.8

SUMMARY

- Systems development of whatever form consists of a development process and an associated development toolkit
- A development process consists of a series of activities for producing an ICT system
- A development toolkit consists of methods, techniques and tools
- The database development process is divided into four main stages: requirements elicitation, conceptual modelling, logical modelling and physical modelling
- Requirements elicitation involves establishing the key technical requirements for a database system usually through formal and informal interaction between developers and users
- Conceptual modelling involves building a model of the real world expressed in terms of the data requirements established
- Logical modelling involves constructing a model of the real world expressed in terms of the principles of some data model
- Physical modelling involves constructing a model of the real world expressed in terms of data structures and access mechanisms available in a chosen DBMS
- The database development toolkit consists of methods, techniques and tools

14.9

ACTIVITIES

1. In what respect is the database development process a human activity system (Chapter 4)?
2. Attempt to detail some of the key benefits of following a given method for database development.
3. Consider a project to produce a database system for Goronwy Galvanising (Chapter 3). After reading Chapter 15 and from the Case Study description, attempt to identify some of the fundamental data requirements for such a database system.
4. After reading Chapter 16, attempt to develop an E-R diagram from the description of the existing data used to support the galvanising process.
5. After reading Chapter 18, attempt to conduct a normalisation exercise on the despatch advices form.
6. Attempt to specify the relational schema for the Goronwy Galvanising application using SQL data definition statements.

14.10  REFERENCES

- Beynon-Davies, P. (2002). *Information Systems: An Introduction to Informatics in Organisations*. Basingstoke, Palgrave (formerly Macmillan).
- Silberschatz, A., H.F. Korth and S. Sudarshan (2002). *Database System Concepts*. Boston, MA, McGraw-Hill.
- Teorey, T.J. (1994). *Database Modelling and Design: The Fundamental Principles*. San Mateo, CA, Morgan Kaufmann.

CHAPTER 15 →

REQUIREMENTS ELICITATION

There is no doubt that the first requirement for a composer is to be dead.

Arthur Honegger (1892–1955)

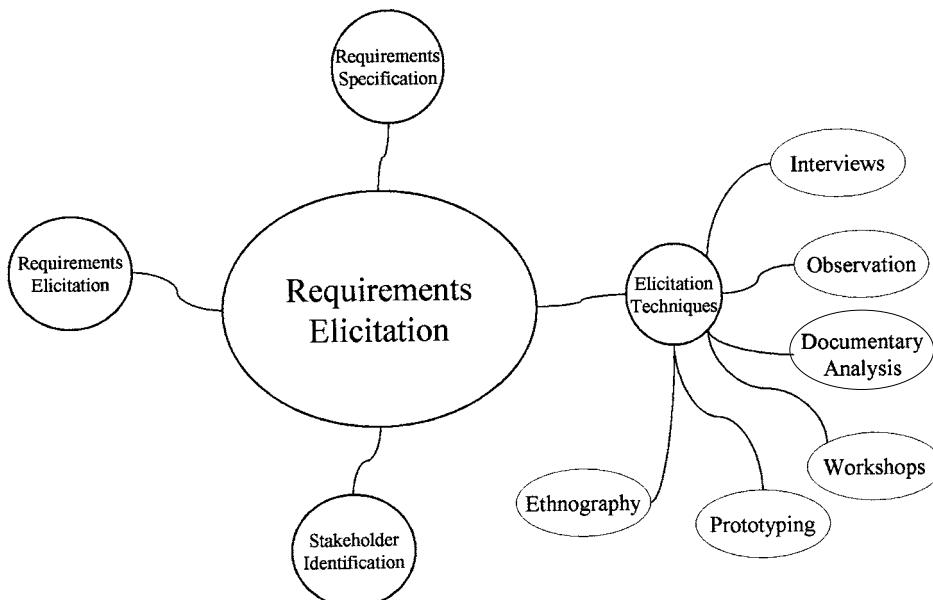
Confidence in nonsense is a requirement for the creative process.

Anonymous

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define the difference between requirements elicitation and requirements specification
- Discuss the importance of stakeholder identification and participation
- Relate some of the features of a requirement
- Discuss the differences between requirements elicitation and requirements capture
- Describe some of the key elicitation techniques



15.1  INTRODUCTION

Systems analysis involves two primary and interrelated activities – requirements elicitation and requirements specification. Both sets of activities demand different techniques. Requirements elicitation demands approaches for identifying requirements. Requirements specification demands techniques for representing requirements. Systems analysis benefits from forms of stakeholder participation, especially in the elicitation phase.

In development projects, requirements specification is conducted using one of the established conceptual modelling approaches such as entity–relationship (E–R) diagramming (Chapter 16). Therefore, we concentrate on the process of requirements elicitation in this chapter.

15.2  STAKEHOLDER IDENTIFICATION AND PARTICIPATION

One of the first things that must be done in any information systems project is to identify the relevant stakeholders. A stakeholder group is any social group within and without the organisation that potentially may influence the successful use and impact of some ICT system.

In terms of defining the proposed shape of some ICT system the following stakeholder groups will prove significant:

- *Clients.* Clients normally equate to managerial groups within organisations. They will normally set the key organisation objectives for the information system
- *End-users.* Users are particularly important for setting the key functionality and usability requirements for an information system
- *Customers.* The customers of some organisation may also be an important stakeholder group in the sense that they assess the worth of the activity supported by the information system. Many front-end ICT systems now directly interface with customers. In this sense, customers are the users of such systems
- *Regulators.* Certain organisations and group may have an important influence on setting regulatory regimes which the system has to satisfy

The importance of each stakeholder group to the successful trajectory of the project should be determined. Ideally, representatives of the most important stakeholder types will form part of the project organisation and will particularly contribute to the requirements elicitation process.

Example In terms of an ICT system for research management at a university we might identify the following stakeholder groups:

- *Producers.* A team of two programmers and one analyst
- *Clients.* Management layers in the university – vice-chancellor, pro-vice-chancellors, deans of faculty and heads of school
- *Users.* Research coordinators in each academic school and members in Academic Registry
- *Customers.* Funding agencies and bodies
- *Regulators.* National and regional government

15.3 REQUIREMENT

Elicitation is concerned with system requirements. A requirement is any desired feature of an information system in general or database system in particular. Requirements are traditionally portrayed as unproblematic in the sense that they can be captured without too much difficulty from the stakeholder community. However, requirements have the following features:

- Requirements may vary depending on the stakeholder group. Requirements are not objective phenomena, they are relative to a particular stakeholder's perspective
- Requirements may be in conflict. Analysis involves attempting to achieve some inter-subjective agreement among stakeholder groups about requirements
- Requirements must be frozen at some point in order to construct an information technology system – an artefact. However, requirements are likely to change over time. Part of the reason for the maintenance of systems and databases is because requirements change

Example In terms of a system to manage research information for some university, the following may be a list of proposed requirements for the system:

- The system must have the ability to capture information about all journal research papers produced by members of staff at the university
- The system must have the ability to capture information about research student completions
- The system must have the ability to capture information about all research grants submitted to external research bodies
- The system must have the ability to provide information on active research staff in the university

- The system should enable management in the university to monitor the research performance of academic departments
- The system must have the ability to capture information about research income achieved by the university
- The system should enable academic departments to monitor their research activity
- The system must have the ability to capture information about all research papers given at conferences by members of staff of the university
- The system should be able to generate information for use in reports provided for external agencies: prospectuses, research reports, brochures etc
- The system must have the ability to capture information about research student supervision
- The system must have the ability to capture information about all books or chapters in books produced by members of staff at the university
- The system should enable academic research coordinators to maintain records of research students, publications, grants etc.
- The system must have the ability to capture information about research student progression

In the classic software engineering literature a distinction is made between so-called functional and non-functional requirements:

- Functional requirements are expected features of some information system
- Non-functional requirements are constraints set on the systems development project

The set of functional and non-functional requirements establishes the scope of the information system.

- Example** ➔ A set of non-functional requirements for this research information system might be:
- Needs to be built in a maximum of six months in order to be able to perform an internal research assessment exercise
 - To be initially built in registry and used by 6+ staff there
 - Eventually to be used by research coordinators/administrators, possibly by active researchers in departments
 - One programmer and one analyst project manager available for the project
 - One development machine available

15.4 Requirements Elicitation and Requirements Specification

Requirements elicitation is that process devoted to the identification of requirements. Unfortunately this process is sometimes referred to as requirements capture, thus implying that requirements are objective constructs that can be collected and represented in an unproblematic way. The features of requirements described above clearly indicate that requirements cannot be captured. They are inter-subjective constructs. Hence requirements elicitation necessarily involves a process of negotiation between various stakeholder groups in organisations. The differences between these two models of the requirements elicitation process are illustrated in Figure 15.1.

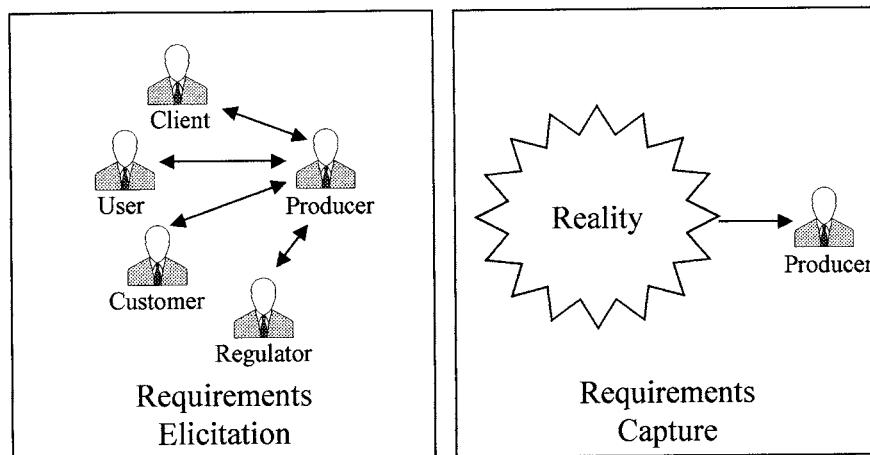


Figure 15.1 Requirements elicitation and requirements capture.

Requirements elicitation is the precursor to requirements specification. Requirements specification in database terms equates to conceptual modelling and logical modelling. Both requirements elicitation, conceptual modelling and logical modelling are reliant on the process of abstraction.

In the real world we are confronted by a vast range of phenomena. Abstraction is the process of recognising similarities between phenomena and ignoring differences. An abstraction is therefore a group of phenomena covered by similar properties. Other commonly used words for an abstraction are concept and sign (Chapter 2).

Conceptual modelling is therefore the process of identifying appropriate concepts and linking them with real-world phenomena. Conceptual modelling is the process of developing a schema for some application. A database schema is made up of a series of concepts. A schema is therefore a representation of some reality. Having said this, there is a tendency in the literature to assume that the process of developing a conceptual model using some notation such as object modelling is unproblematic. The major philosophical position taken

in much of information systems development is that of objectivism (Dahlbohm and Mathiassen, 1993). The assumption is that there is some objective reality, elements of which can be captured and represented (hence the use of terms such as requirements *capture*). Although this position works well in relation to the physical world where the objects can be seen, touched and smelt, this position is problematic when applied to the social world, of which organisational reality is a part. A more appropriate philosophical position in relation to this world might be described as a subjectivist interpretive position. The assumption here is that organisational reality is not a given; reality is socially constructed in the process of human interaction.

In this view, conceptual modelling should not be seen as a straightforward, unproblematic process. Instead, it is more accurately viewed as a process of inter-subjective negotiation between the database developer and numerous interest groups within organisations (Beynon-Davies, 1992).

In Chapters 16 and 17 we discuss two of the most commonly used techniques for requirements specification in the database area: E-R diagramming and object modelling. In Chapter 18 we discuss the most commonly used technique for logical modelling.

15.5 Requirements Specification as Modelling

Requirements specification as a process is a process of modelling. A requirements specification as an artefact is a model – a conceptual or logical model. Any model can be seen as a sign-system (Chapter 2) – a collection of signs used by some social group. The construction of such sign-systems is the process of modelling and is undertaken for three reasons:

- *Communication*. The primary use for a model is as a medium of communication between some group of persons
- *Representation*. A model is used to represent common understandings about some real-world phenomena among this group of persons
- *Abstraction*. Modelling generally implies some form of simplification of the real world. The modeller uses a model to focus on what are seen to be the important features of some real-world situation (see above)

Although modelling is an essential element of information systems work, there is not agreement about the most appropriate way of modelling systems in general or information and ICT systems in particular. However, the database systems area is unusual in the sense that there is some agreement about appropriate ways of forming conceptual and logical models. In the chapters which follow we portray at a high-level some of the key elements of the most accepted approaches to database systems modelling.

Any modelling of whatever form needs three elements:

- *Constructs*. By constructs we mean the component elements of the modelling approach



- *Notation.* By notation we mean the form of representation employed for the constructs within the modelling approach. Such a notation can be textual, graphical and/or mathematical. Graphical notations are probably the most frequently employed in information systems work because of their ease of use
- *Principles of construction.* By principles we mean the formal and informal rules for correctly constructing models

Example In entity–relationship modelling, the constructs include entities and relationships. A possible notation is to draw entities as boxes and relationships as lines. A principle of construction is that every relationship should have a defined cardinality.

15.6 ELICITATION TECHNIQUES

A number of techniques exist for requirements elicitation. Many of such techniques are used in common with those for social science and information systems research. This is not surprising because elicitation primarily involves gathering people's ideas, opinions and attitudes in relation to ICT systems.

Requirements elicitation techniques include interviews, observation, documentary analysis, workshops, prototyping and ethnography.

15.6.1 INTERVIEWS

Interviews are certainly the most commonplace requirements elicitation technique. Interviews constitute either formal or informal discussions with representatives of key stakeholder groups. Formal interviews are structured conversations in which the questions are determined beforehand. They are hence frequently referred to as structured interviews. Informal interviews are a form of discussion in which the questions are formulated within the flow of the interview itself. These can be semistructured, in which the developer uses a checklist of key points to organise the conversation, or unstructured, in which the developer poses questions within the conversation itself dependent on its flow.

The record of an interview may be recorded in an unobtrusive way as interview notes or in an obtrusive way as an audio-recording. An audio-record may later be transcribed for easier analysis.

15.6.2 OBSERVATION

Frequently what people describe in interviews only partly reveals the reality of human activity systems and information systems. Hence interviews will frequently be used with other elicitation techniques such as observation to



'triangulate' requirements. In other words, to check that the requirements produced through one elicitation technique match those from some other technique.

Observation usually involves being present in work settings recording the detailed work behaviour of people. It may also involve the close observation of potential work groups using current ICT systems of various forms and involve obtrusive data capture techniques such as video-recording.

15.6.3 DOCUMENTARY ANALYSIS

Most organisations generate large amounts of documentation which may be useful to the developer particularly if the intention of a database system is in some way to automate aspects of data capture, data storage and data retrieval. Hence, documents are a valuable resource for indicating the data that needs to be stored within some information system and the type of reports that need to be generated from the information system.

15.6.4 WORKSHOPS

Workshops constitute sessions in which developers and representatives of stakeholder groups get together in a structured situation to formulate requirements for systems. Workshops are controlled environments for the negotiation of requirements and typically take place away from the normal worksetting of both business users and developers. Two types of workshops are familiar in development projects: joint requirements planning workshops and joint application design workshops. The former clearly are events for requirements elicitation or specification. The latter are events particularly for logical or physical design.

15.6.5 PROTOTYPING

Frequently stakeholder groups may not be able to formulate what they require until they see some representation of a proposed innovation. Prototyping involves building versions of particular parts of a system to demonstrate to stakeholder representatives in order to obtain their feedback. It is possible to distinguish between a range of different prototyping techniques:

- *Throwaway prototype.* A throwaway prototype is used to test out some part of a system but then discarded. Such prototypes may be low-technology (flip-charts, cardboard mockups etc.) or high-technology prototypes involving the use of some fast development tools
- *Incremental prototype.* An incremental prototype is one which, by incremental refinement, will form the whole of or part of a delivered system
- *Evolutionary prototype.* Graham (1989) makes the useful distinction between incremental development and incremental delivery. An incremental prototype



appears to comprise a complete system that is developed incrementally; an evolutionary prototype forms part of a proposed system which is planned to be delivered incrementally. Hence, all phases of development are conducted for the increment delivered, including coding and testing, moving on to the next increment.

All three types of prototyping impinge on requirements elicitation. Incremental and evolutionary prototypes impinge on other development activities such as design and implementation.

15.6.6 ETHNOGRAPHY

Ethnography does not constitute one technique but a set of techniques for requirements elicitation. It is seen to be particularly appropriate for gaining access to the tacit or hidden knowledge underlying work practices. Ethnography frequently involves observers participating in some work setting and attempting to empathise with stakeholders to gain an in-depth appreciation of both explicit and tacit work processes. Ethnography has only comparatively recently been used to gather rich representations of work situations as a precursor to systems design.

15.7

CASE STUDY: RESEARCH DATABASE SYSTEM

This project was conducted in the context of a UK university. The university utilises piecemeal information systems to support the core organisational processes of teaching, research and consultancy. The specific project described here emerged from problems experienced with data collection and manipulation of data for the university's last research assessment exercise submission. The project was initiated with the intention of developing a research database system which would obviate this problem in the future.

The university has an internal ICT systems department which has been down-sized over the last few years. Consequently, no major IS development takes place in-house. Although the development effort on this project was reasonably small-scale, the ICT systems department seemed reluctant to take on this project because, among other reasons, it felt that the requirements were not agreed by the diverse stakeholders within the university.

The system was designed to act as a centralised resource for research information such as publications, grants, research students etc. It was intended that the system be used to continually monitor research performance of academic units and to be a key input into the future research strategy for the university.

The development team consisted of two developers and two user representatives. One developer acted in the role of project manager/analyst, the other as a programmer. An initial Joint Requirements Planning (JRP) workshop was conducted with a limited number of user representatives. This enabled scoping of the project within the six months available. Three key phases for the project were planned, terminating in a user review which was open to all stakeholders in the university. After the third phase, a period of some three weeks was spent in consolidation work (documentation), testing and training.

An initial JRP session was run, followed by three formal Joint Application Design workshops. The project was undertaken over a six-month period and was staffed by a team of 4–24 people (4 core members, consisting of 2 developers and 2 users; 10–20 other stakeholders were periodically involved). The team was skilled in the use of the development environment (RDBMS) and was knowledgeable about the business issues. All project activity took place in an open-plan office situated at the users' site. One of the developers was solely focused on the project during the project period but the users interleaved their project involvement with other work. The system produced (a research database system) was of medium background complexity and displayed a high level of interactivity. Explicit use was made of incremental prototyping throughout the project.

The most interesting feature of this project was the large number of stakeholders affected by the system. Virtually every academic unit and the majority of administrative units had some impact on the system in terms of either supplying data to the system or needing to pull data off the system. Therefore, user review sessions were as much a forum for informing and consulting with diverse stakeholders as they were opportunities for design.

15.8 SUMMARY

- Systems analysis involves two primary and interrelated activities – requirements elicitation and requirements representation
- Requirements elicitation is that process devoted to the identification of requirements
- Requirements specification is that process concerned with the representation of requirements
- A requirement is any desired feature of an information technology system
- Requirements may vary depending on the stakeholder group. Requirements are not objective phenomena. Requirements elicitation involves attempting to achieve some inter-subjective agreement among stakeholder groups about requirements
- Requirements are likely to change over time. Part of the reason for adaptation of systems is because requirements change
- Requirements must be frozen at some point in order to construct an information technology system – an artefact
- Conceptual modelling is still subject to the important philosophical characteristics of organisational reality

15.9 ACTIVITIES

1. Develop a requirements elicitation strategy for a database development project in a key area known to you.

2. After reading Chapter 16, consider in what respect an E-R diagram might constitute a prototype.
3. Distinguish between the constructs, notation and principles of some conceptual technique such as E-R diagramming (Chapter 16).

15.10 REFERENCES

- Beynon-Davies, P. (1992). The realities of database design: the sociology, semiology and pedagogy of database work. *Journal of Information Systems* **2**: 207–20.
- Dahlbohm, B. and L. Mathiassen (1993). *Computers in Context*. London, NCC/Blackwell.
- Graham, I. (1989). Incremental development: review of nonmonolithic life-cycle development models. *Information and Software Technology* **31**(1): 7–20.

CHAPTER 16

ENTITY-RELATIONSHIP DIAGRAMMING

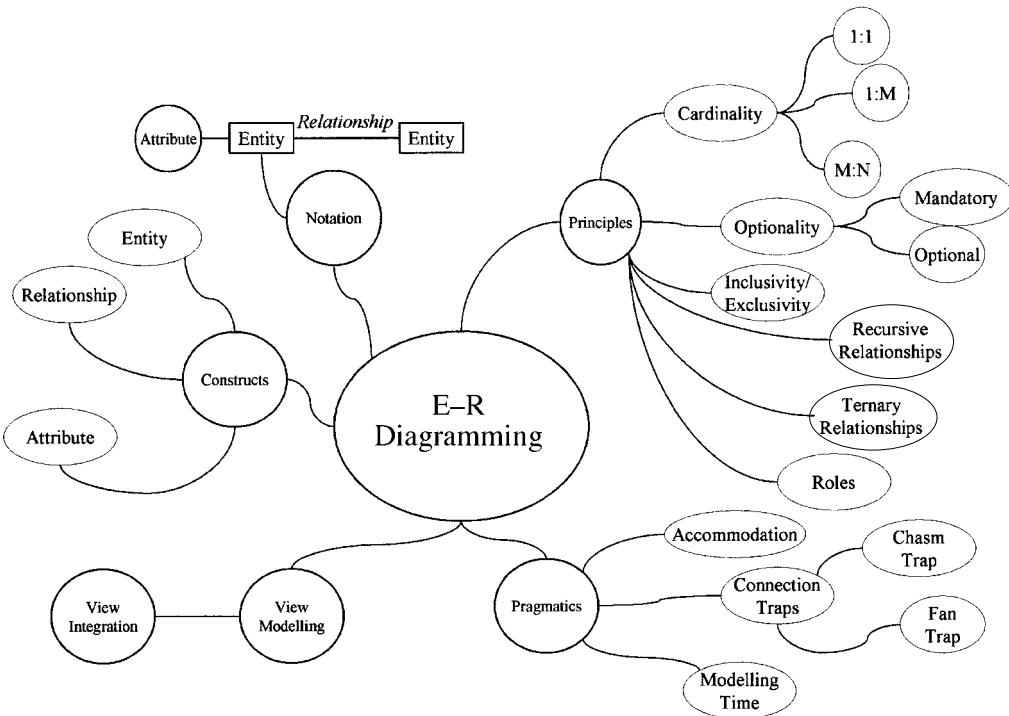
Life is just one damned thing after another.

Elbert Hubbard (1856–1915)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Describe the key elements of the entity–relationship approach
- Discuss a number of extensions to the entity–relationship approach
- Consider some of the pragmatics involved in producing entity models
- Consider the issue of view integration



16.1 INTRODUCTION

In this chapter we describe in some detail an approach to conceptual modelling based around a diagrammatic technique known as entity–relationship (E–R) diagramming. In the logical modelling approach (Chapter 18) of normalisation we must have a data-set in place before we can begin the process of logical database design. The aim of logical modelling is to divide this data-set into logical groupings.

In contrast, conceptual modelling is data analysis in the abstract. There need be no data-set in place which the data analyst can examine. Instead, he or she has to build a model of the ‘things of interest’ to an organisation. Such a model is usually referred to as a conceptual data model (Chapter 3).

In practice, database developers normally do more conceptual data modelling than they do normalisation. This is because conceptual data modelling can cover more ground than normalisation. Normalisation is generally used as a means of validating or verifying critical areas of an applications data model.

16.2 DATA MODELS

In this chapter we assume that a data model is represented using the constructs of the E–R approach. In this approach, a given universe of discourse is represented using an entity model: a model built up of entities, relationships and attributes.

16.2.1 ENTITIES

An entity may be defined as a thing which an organisation recognises as being capable of an independent existence and which can be uniquely identified. An entity is an abstraction from the complexities of some domain. When we speak of an entity we normally speak of some aspect of the real world which can be distinguished from other aspects of the real world.

An entity may be a physical object such as a house or a car, an event such as a house sale or a car service, or a concept such as a customer transaction or order. Although the term entity is the one most commonly used, following Chen (1976) we should really distinguish between an entity and an entity-type. An entity-type is a category. An entity, strictly speaking, is an instance of a given entity-type. There are usually many instances of an entity-type.

Example  Hence, Lecturer is an entity-type whereas Paul Beynon-Davies is an instance of this entity-type. Paul Beynon-Davies is an entity.

Because the term entity-type is somewhat cumbersome, most people tend to use the term entity as a synonym for this term. We shall conform to this practice.

It must be remembered however that whenever we refer to an entity we normally mean an entity-type.

Entities are by their very nature interesting things because entities are normally used to define logical data groupings. In conventional information systems jargon the term logical data grouping normally means a file. One rule of thumb to apply in identifying suitable entities for a given application is therefore the following:

If you need to store data about many properties of some thing, then that thing is likely to be an entity.

16.2.2 RELATIONSHIPS

A relationship is some association between entities. In this section we shall concentrate on binary relationships. That is, associations between two entities. In section 16.4 we shall introduce other *N*-ary relationships. That is, relationships between one, three, four or *N* entities.

In the E-R approach, more than one relationship can exist between any two entities.

Example → The entities *House* and *Person* can be related by *ownership* and/or by *occupation*. In theory, having identified a set of say 6 entities, up to 15 relationships could exist between these entities. In practice, it will usually be obvious that many entities are unrelated.

Furthermore, the object of the E-R approach is to document only so-called direct relationships. For instance, direct relationships exist between the entities *Parent* and *Child* and between *Child* and *School*. The relationship between *Parent* and *School* is indirect; it exists only by virtue of the *Child* entity (Shave, 1981).

Note however that we must be clear about what we are attempting to represent.

Example → If the relationship between *Parent* and *Child* is one of *parenting*, the relationship between *Child* and *School* is one of child *attends* school, and the relationship between *Parent* and *School* is one of parent *chooses which* school to send child to, then the parent–school relationship is probably correctly labelled as an indirect relationship. Consider however the case of the relationship between *Parent* and *School* being parent is *governor* of school. This is no longer an indirect relationship. It cannot be represented by the two relationships parent–child and child–school. Not every parent of a child is a school governor. This is actually an example of a so-called connection trap (section 16.5.3).



16.2.3 ATTRIBUTES

As a real-world aspect, an entity is characterised by a number of properties or attributes. Values assigned to attributes are used to distinguish one entity from another.

Example Hence, `deptName` and `location` are both attributes which define the entity `Department`.

We also choose one or more attributes to act as identifiers for instances of an entity. In terms of relational schema the entity identifier will eventually turn into the primary key (see below).

Example Hence, the attribute `deptName` is the likely identifier for the entity `department`, in that it helps us to uniquely distinguish between instances of this entity.

16.2.4 SEMANTIC MODELLING

The data needed to support a given information system does not usually fall irrevocably into one of the three categories entity, relationship and attribute.

Example A classic example is the data needed to be stored on marriages. Marriage could be regarded as an entity with attributes such as date, place, and names of bride and groom. It could similarly be regarded as the attribute marital status associated with the entity `Person`. Finally, it could be represented as a relationship between the entities `Man` and `Woman`.

One of the tasks of the data modeller is to decide which of these viewpoints is the most important for the information system under consideration. Hence, top-down data analysis is frequently referred to as semantic modelling (Date, 2000). The aim is to represent data as it is perceived in the organisation under consideration (Klein and Hirschheim, 1987).

16.2.5 NOTATION

Data models are usually mapped out as entity–relationship diagrams (E–R diagrams). The end-product of entity–relationship diagramming is a graphical model of the entities and relationships in a particular domain of discourse.

There is unfortunately no standard notation for E–R diagramming. A number of notations used in database development practice are indicated in

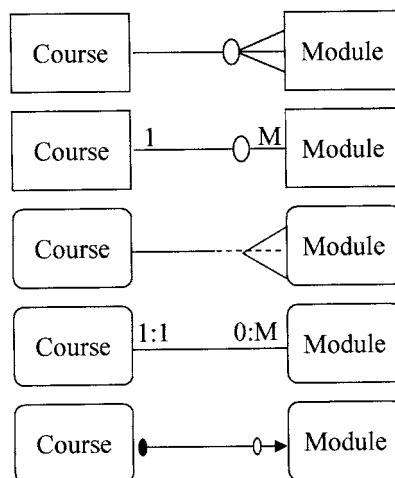


Figure 16.1 Variation in E-R modelling notation.

Figure 16.1. Each of the diagrams in this figure specify the same set of data requirements, namely:

- A course contains a number of modules
- A module belongs to one and only one course
- A course must contain at least one module
- A module does not need to belong to a given course

In this work we shall employ a notation based upon a set of standards proposed by Martin (Martin and McClure, 1985). Our major intent has been to provide a consistent practice for moving between conceptual modelling and logical modelling.

An entity is represented on the diagram by a rectangular box in which is written a meaningful name for the entity (Figure 16.2A). Note that we label the entity with a singular noun.

Example We speak of an order and not of orders, a patient and not of patients.

This is because an entity represents a category of something. There is only one instance of a category.

A relationship between entities is represented by drawing a line (sometimes labelled) between the relevant boxes on the diagram (Figure 16.2B). In some notations labels are placed on the relationship lines. This is a useful technique in resolving ambiguity. There are however a number of problems with this convention:



- It is frequently difficult to think of a meaningful label for relationships
- Most relationships are best represented by verbs. Verbs however usually imply some direction. Hence the relationship between person and grade might be read as person *is graded by* grade in one direction and grade *grades* person in the opposite direction. This is cumbersome to represent on a diagram

In this text we shall therefore adopt the practice of labelling relationships where necessary with a simple numbering convention. We assign to each relationship on our entity model a unique number. This number is then used to reference an entry in a data dictionary. We shall delay the discussion of data dictionaries until section 16.7.

An attribute is represented by a circle or oval attached by a line to the appropriate entity. We use circles as symbols because the attributes on E-R diagrams correspond to the data-items on determinancy diagrams (Chapter 18). The entity identifier is also underlined (Figure 16.2C).

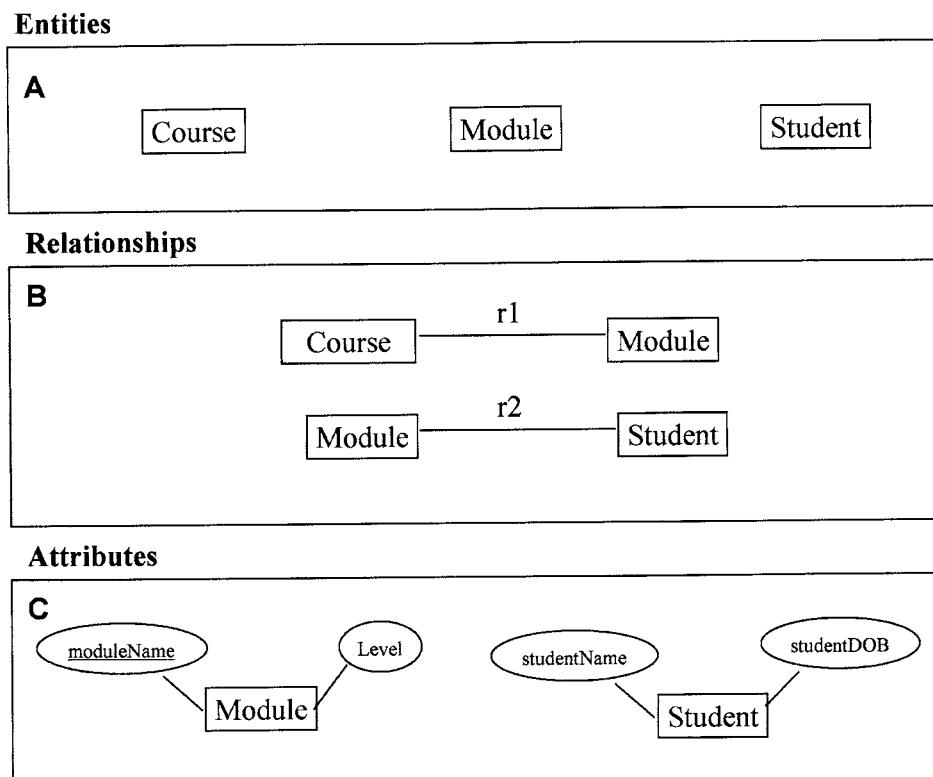


Figure 16.2 Constructs in the E-R approach.

16.3 RULES DEFINING A RELATIONSHIP

There are two sets of rules which we need to add to any relationship. These rule-sets have different names in different approaches to E-R diagramming. We shall refer to these rule-sets as cardinality and participation.

16.3.1 CARDINALITY

Cardinality (or degree) concerns the number of instances involved in a relationship. A relationship can be said to be either a 1:1 (one-to-one) relationship, a 1:M (one-to-many) relationship, or a M:N (many-to-many) relationship.

Expressing the cardinality of a relationship means expressing two assertions about some universe of discourse.

Example → The relationship between a Bankaccount entity and a Customer entity can be said to be one-to-one (1:1) if it can be defined in the following way:

- A bank account is held by at most one customer
- A customer may hold at most, one bank account

In contrast, the relationship between Bankaccount and Customer is one-to-many (1:M) if it is defined as:

- A customer holds many bank accounts
- A bank account is held by at most one customer

Finally, we are approaching a realistic representation of the relationship when we describe it as being many-to-many (M:N). That is:

- A customer holds many bank accounts
- A bank account may be held by many customers

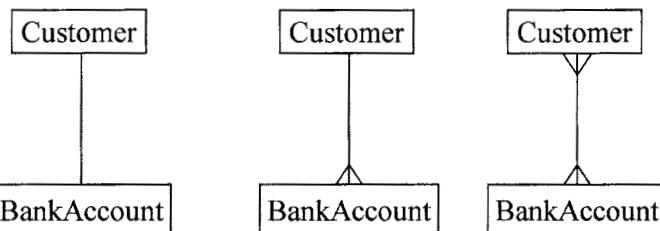
There are a number of competing notational devices available for portraying the cardinality of a relationship. We choose to represent cardinality by drawing a crow's foot on the many end of a relationship (see Figure 16.3A).

16.3.2 PARTICIPATION

Participation (or optionality) concerns the involvement of entities in a relationship. Participation is about exceptions to the rule. An entity's participation is optional if there is at least one instance of an entity which does not participate in the relationship. An entity's participation is mandatory if all instances of an entity must participate in the relationship. We assume that the default participation is mandatory. If the participation is optional we add a circle (an 'O' for optional) alongside the relevant entity (see Figure 16.3B).



Cardinality

A

Optionality

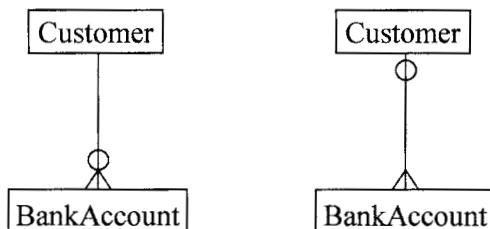
B

Figure 16.3 Cardinality and optionality.

Example Hence in the assertions given below the entity Lecturer has mandatory status while the entity Department is optionally involved in the relationship.

Every lecturer must be employed within a department
A department may exist without any lecturers

16.3.3 MIN/MAX CARDINALITY

In some approaches to building entity models the participation of a relationship is modelled using cardinality. A minimum and a maximum cardinality are stated for each end of an association. The minimum cardinality corresponds to a specification of participation. This is the approach taken in the notation for object modelling described in Chapter 17.

Example ➔ For instance,:

Employee worksFor Department
(1,M) (0,1)

This indicates that the minimum number of employees working for a department is 1, indicating that employee is in fact mandatory in this relationship. The minimum cardinality for department is zero, indicating that this object is optional in the relationship.

16.3.4 INSTANCE DIAGRAMS

The concepts of cardinality and participation can best be understood by using occurrence or instance diagrams. These diagrams illustrate how occurrences or instances of entities relate. The circles or ovals on the diagrams are meant to represent sets of instances. Each entity therefore is represented as a set of instances. A line drawn between instances of two sets indicates a specific instance of a relationship.

Examples ➔ Hence, in Figure 16.4 the line drawn between 'Computer Studies' and '234' indicates that lecturer 234 is employed by the Computer Studies department.

Note that the cardinality of the relationship in Figure 16.4 is 1:M. The department 'Computer Studies', for instance, has two lecturers associated with it. The entity Lecturer has mandatory participation while Department has optional participation. Mechanical Engineering, for instance, is not associated with any lecturers.

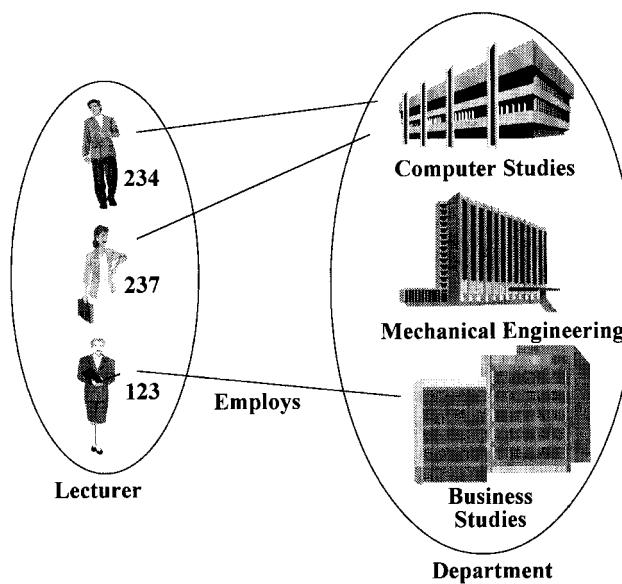


Figure 16.4 A one-to-many relationship as an instance diagram.



In Figure 16.5 Lecturer to Student is a many-to-many relationship. Lecturer 234, for example, teaches students 34698 and 37798. The participation of Lecturer is optional; Student is mandatory.

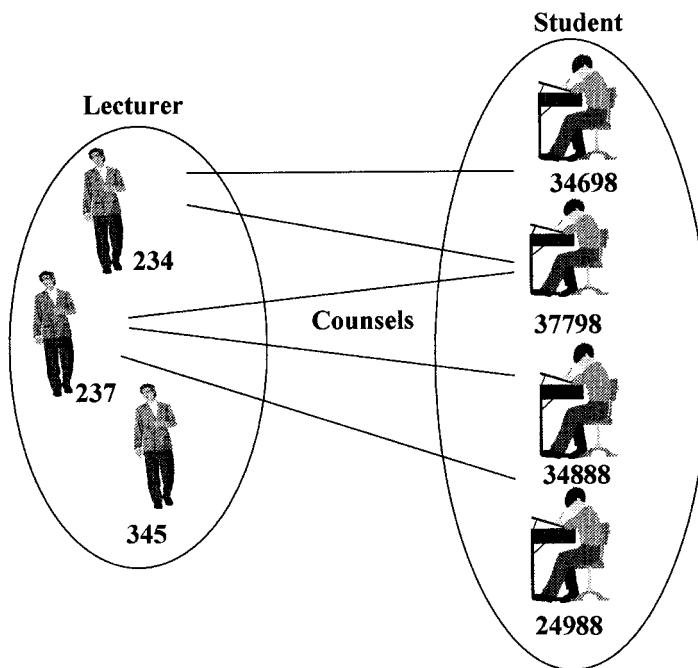


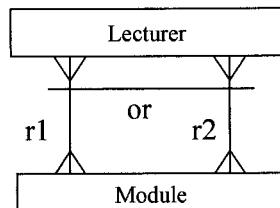
Figure 16.5 A many-to-many relationship as an instance diagram.

16.3.5 INCLUSIVITY/EXCLUSIVITY

As is evident from the discussion above, it is relevant to speak of sets of instances making up a relationship. Indeed, we can extend the notion of participation to a consideration of sets of relationships rather than single relationships.

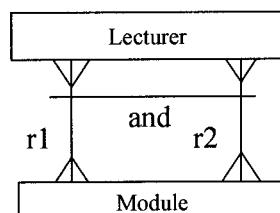
Examples → Suppose we have an E-R diagram as in Figure 16.6A. Here, a module is prepared by a lecturer, and a lecturer teaches a module. The module entity cannot participate in both relationships at the same time – it cannot be both being prepared and being presented. In this case we say that these two relationships are mutually exclusive.

Consider another case. Figure 16.6B illustrates a situation in which a lecturer teaches a module and a module is examined by a lecturer. In this case, we would say that these two relationships are inclusive. A lecturer cannot give a module without examining or assessing it.

Exclusivity**A**

r1 – teaches, is taught by

r2 – prepares, is prepared by

Inclusivity**B**

r1 – teaches, is taught by

r2 – examines, is examined by

Figure 16.6 Inclusivity and exclusivity.

We can represent exclusivity and inclusion by drawing a line between the appropriate relationships and writing the words (AND) for inclusion and (OR) for exclusivity next to the line.

16.4 EXTENSIONS TO DATA MODELLING

A number of extensions have been proposed to the basic E-R approach. In this section we consider the idea of recursive relationships and roles. A number of specially defined relationships or abstraction mechanisms have also been proposed as extensions. We defer discussing this issue until Chapter 17.

16.4.1 RECURSIVE RELATIONSHIPS

In conventional entity–relationship diagrams the relationships are all binary, that is, we diagram two entities and a relationship or a set of relationships between these entities. It is possible however for relationships to be unary. In other words, a relationship may involve only a single entity. Unary relationships are frequently described as being recursive in that they relate entities of the same type.



Example → Figure 16.7 details a recursive relationship called prerequisites which applies to the entity module. A module may have a number of prerequisite modules; a given module may also be a prerequisite for numerous other modules.

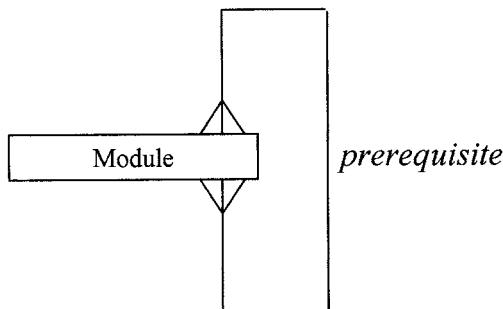


Figure 16.7 Recursive relationships.

16.4.2 TERNARY RELATIONSHIPS

A ternary relationship is one which associates three or more entities together.

Example → The relationship skill_used in Figure 16.8 associates the entities employee, skill and project.

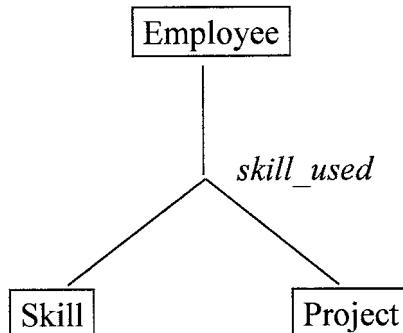


Figure 16.8 Ternary relationships

Ternary relationships are only used when they cannot be decomposed into a series of binary relationships (Teorey, 1994). Ternary relationships cannot be decomposed into binary relationships if the relation used to represent the relationship is in fifth normal form (Chapter 18).

Example → If we attempt to fragment the relation skill_used in Figure 16.8, for instance, into any two relations we lose valuable information.

16.4.3 ROLES

An entity participating in a relationship with another entity may act in two or more different roles within such a relationship.

Example → In the *manages* relationship in Figure 16.9A the Employee entity acts in two different ways – as a manager, and as a subordinate. These two different perspectives on the same entity are known as the entity's roles. An employee may act in the role of a manager in one relationship, and in the role of worker in another relationship.

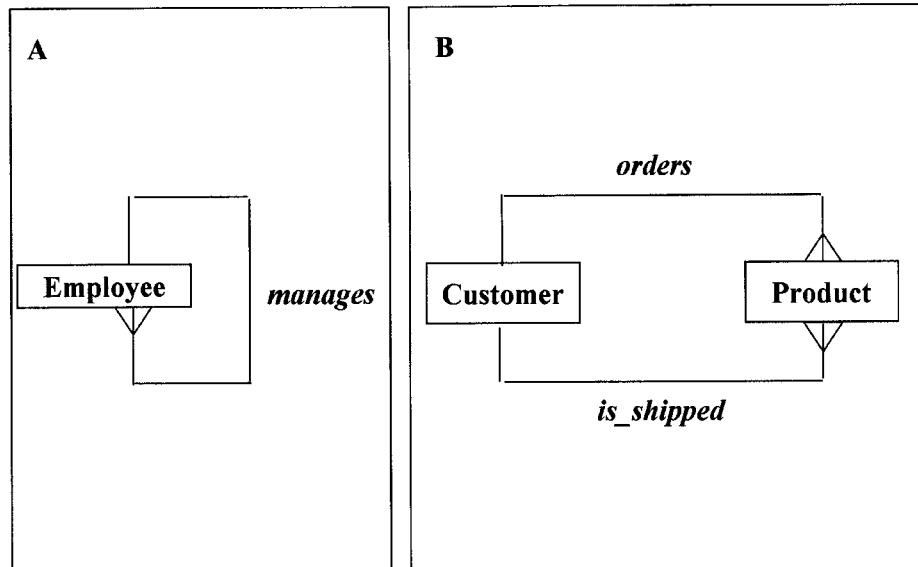


Figure 16.9 Roles.

Roles however are not solely of relevance to unary relationships. They are equally of relevance to binary relationships.

Example → The two distinct relationships between customers and products indicated in Figure 16.9B, for instance, mean that the Customer entity must act in two different roles towards products. A customer is both a person who orders products, and a person who receives products.

16.5 (C) PRAGMATICS OF E-R MODELLING

In the previous sections we have considered the important component parts of the E-R approach to database modelling. In this section we consider some important facets of the practice of E-R modelling.

16.5.1 STRONG AND WEAK ENTITIES

An entity is said to be a strong entity (or entity-type) if its existence does not depend on the existence of some other entity. In contrast, a weak entity depends on the existence of some other entity in a universe of discourse.

Example → In a university database the entities Student and Lecturer are both strong entities. However, the entity grade is likely to be a weak entity since it depends on the prior existence of both a Student entity and a Module entity.

Identifying weak entities is important because instances of such entities cannot be uniquely identified by attributes of the entity itself. They must acquire some identifying properties from the strong entities with which they are associated.

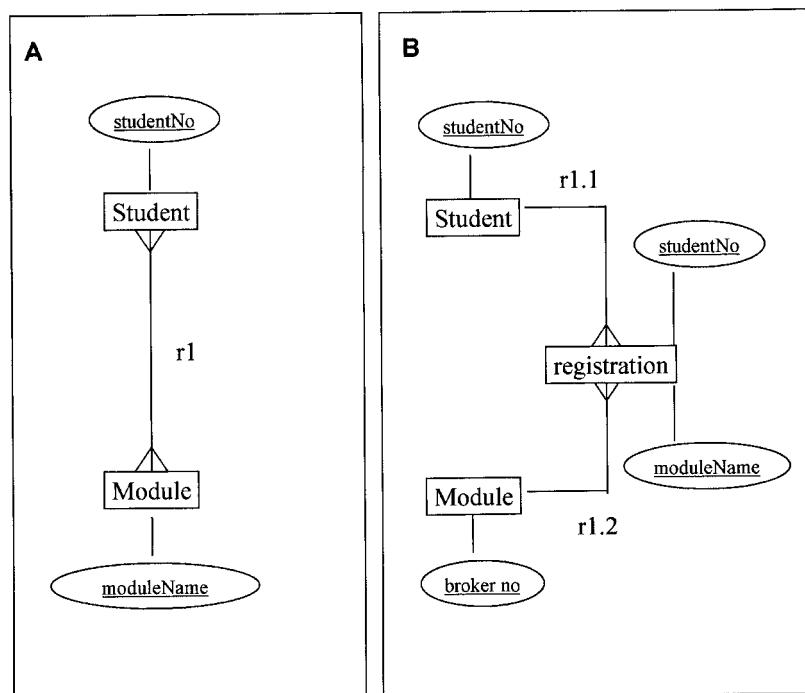
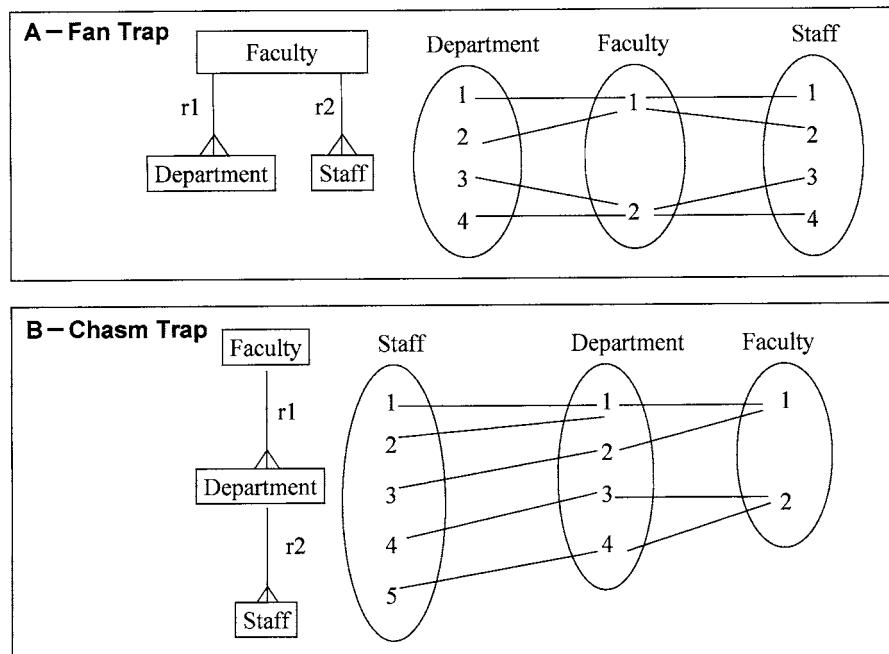
16.5.2 SIMPLIFYING MANY-TO-MANY RELATIONSHIPS

To simplify the process of accommodating to a relational schema we shall recommend decomposing any many-to-many relationships (Figure 16.10A) in a conceptual data model into two one-to-many relationships. To decompose a relationship we introduce a link entity. This entity cross-refers between instances of one entity and instances of another entity.

Example → Hence, in Figure 16.10B we introduce a registration link entity to connect students with modules. Note that the many ends of the relationships always appear at the link entity. Student and Module are both strong entities in this relationship. Registration is a weak entity because it relies on the prior existence of both Student and Module.

16.5.3 CONNECTION TRAPS

Figures 16.11 and 16.12 illustrate a number of potential pitfalls in entity modelling. These pitfalls are known as connection traps because they make invalid assumptions about the connection between entities (Howe, 1981).

**Figure 16.10** Simplifying many-to-many relationships.**Figure 16.11** Connection traps.

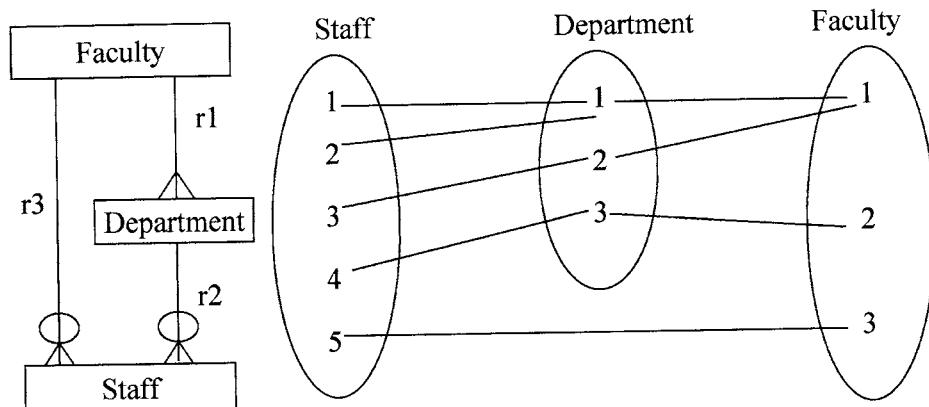


Figure 16.12 Faculty staff.

Example → Figure 16.11A illustrates a type of connection trap known as a fan trap, so called because it may occur when two 1:M relationships fan out from the same entity. For instance, the entity model on Figure 16.11A documents the following assertions about an academic application:

- A faculty has many departments
- Every department belongs to at most one faculty
- A department has many staff
- A member of staff belongs to at most one department

The database designer assumed that this organisation was sufficient to tell him which staff belongs to which departments. As we see from the associated instance diagram however this assumption is incorrect. Although we can tell from the entity model which staff belongs to which faculty and which department belongs to which faculty, the link between staff and departments is ambiguous.

Figure 16.11B illustrates a representation for the same data model which overcomes the fan trap. The instance diagram seems to indicate that the query *which staff work for which departments* is clearly answerable from this revised model.

This revised model may however be subject to a further problem. What if we have staff in our university who are employed by faculties rather than by departments? In other words, the participation of staff in r2 is optional. Our data model would give us an incorrect answer as it assumes that all staff must be employed by departments. This is known as a chasm trap. A chasm trap exists if an entity model suggests that a relationship exists between all instances of two entities when this is not the case.

Example To avoid this misinterpretation we have to introduce an additional relationship into our diagram between staff and faculty (see Figure 16.12) and clearly indicate that both r2 and r3 are optional. Note that what defines a fan trap or a chasm trap is determined by the semantics of the application. Figure 16.11B, for instance, would be perfectly reasonable as a representation of some applications where the business rules prohibit faculty staff.

16.5.4 MODELLING TIME

Most database systems exist within a temporal dimension. This is because most database systems handle events – real-world entities that must be timestamped. In this section we discuss some techniques for including past and future time in our database design.

Example Consider the case where a university wants to build a database recording details of the courses it runs and details of the students which enrol on courses. If we are only interested in current enrolment then the entity model in Figure 16.13A is sufficient. However, a more realistic situation is approached when the university decides it wishes to store information about past enrolments for management information purposes. We now wish to record which students have enrolled in which courses over a period of say five years. We make the relationship between course and student a many-to-many relationship (Figure 16.13B).

This structure however is equally capable of handling future events. Suppose the university wishes to extend the use of its database to schedule future courses. The only modification necessary to our entity model is to make course and student optional in the appropriate relationships. In other words, we wish to allow course details to be recorded prior to places being filled. Likewise we wish to record details of students prior to their enrolment on a particular course.

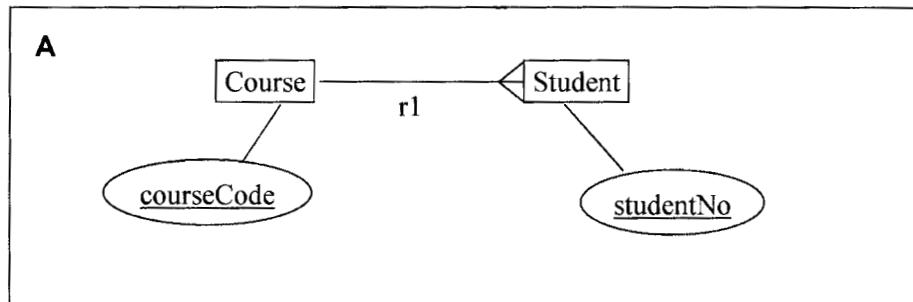


Figure 16.13A Modelling time.

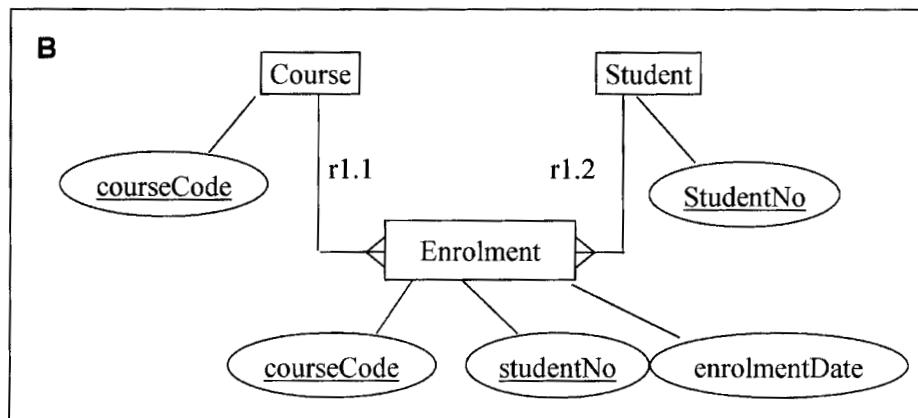


Figure 16.13B Modelling time.

16.6 ACCOMMODATION TO RELATIONAL SCHEMA

In the same way as a determinacy diagram needs to be accommodated to a relational schema, so an E-R diagram has to be accommodated. Fortunately, the process of accommodating an E-R diagram to a relational schema is relatively straightforward.

Example

- For each entity on our diagram we form a table. It is useful to take each entity name and make it a plural to distinguish relations from entities. Hence from the entity model in Figure 16.14 we arrive at two tables: Courses and Modules
- The identifying attribute of the entity becomes the primary key of the table. This means that **courseCode** and **moduleName** become the primary keys of Courses and Modules respectively
- All other attributes of an entity become non-key attributes of the table. **courseName** is a non-key attribute of Courses. **Level** is a non-key attribute of Modules
- For each one-to-many relationship, post the primary key of the one table into the table representing the many end of the relationship. Hence we post **courseCode** into the Modules table
- Optionality on the many end of a relationship tells us whether the foreign key representing the relationship can be null or not. If the many end is mandatory the foreign key cannot be null. If the foreign key is optional the foreign key can be null. In Figure 16.14A **courseCode** cannot be null. In Figure 16.14B **courseCode** can be null

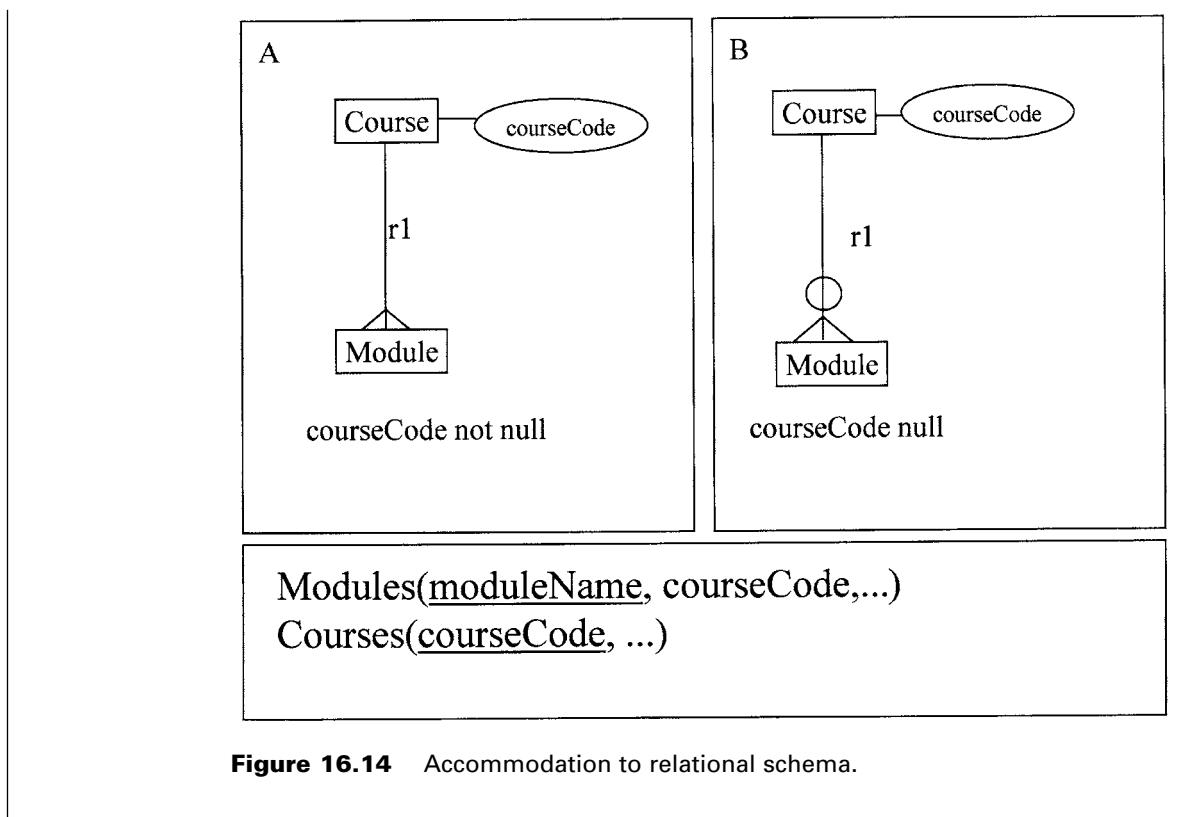


Figure 16.14 Accommodation to relational schema.

A larger example of accommodation is provided in the Case Study at the end of the chapter.

Note that we have not considered many-to-many or one-to-one relationships in the accommodation process described above. Many-to-many relationships we assume will be broken down into one-to-many relationships (section 16.5.2). One-to-one relationships can normally be handled as a single table. In other words, we take both entities in a 1:1 relationship and feed the attributes into one table structure.

A ternary relationship, such as the one described in section 16.4.2, would mean creating one table for the ternary relationship with a compound key composed of the identifiers of all the participating entities.

For each role (section 16.4.3) representing a one-to-many relationship we need a distinct foreign key.

A one-to-many recursive relationship (section 16.4.1) would be accommodated to one table with a foreign key which is effectively the primary key. A many-to-many recursive relationship would need to be broken down into two one-to-many relationships in the normal way. The link entity will then act as a set of cross-reference instances between the instance of the original entity.



16.7

DATA DICTIONARIES

In Chapter 1 we defined a data model as any model of the real world built out of entities (classes), relationships and attributes. A data model however can be represented in a number of different ways. The most popular representation is certainly as an entity–relationship diagram. One of the main problems with an E–R diagram however is that as it gets more detailed, the efficacy of the technique starts to diminish. As more entities, relationships and attributes are drawn, diagrams become less easy to understand and manipulate. For this reason, it is usual for practising data analysts to exploit an alternative mechanism for documenting applications data models – the data dictionary.

16.7.1 REPRESENTING ENTITIES

Each entity in the data dictionary needs to be defined in terms of a description, identifying attributes and other attributes.

Example A student entity might be defined as below:

Entity: student

Description: Any person enrolled on a course at the university

Identifying Attribute(s): studentNo

Other Attributes: studentName, termAddress, homeAddress, termTelNo, homeTelNo

16.7.2 REPRESENTING RELATIONSHIPS

Equally, each relationship in our data model needs to be documented in the dictionary. Relationships are labelled with unique identifiers. This allows us to supply a number of convenient names for the relationship, read in each possible direction. For each entity in the relationship we document its name, participation and cardinality. We also provide space for a brief description which may help us to disambiguate references at a later date.

Example Relationship ID: r1

Relationship Name(s): enrolls, isEnrolledOn

Description: Associates students with the courses they are currently enrolled upon

Participating Entity: student

Cardinality: many

Optionality: mandatory

Participating Entity: course

Cardinality: many

Optionality: optional

16.7.3 REPRESENTING ATTRIBUTES

As we approach a more detailed specification we may need to document information about attributes. All this information will be usefully applied in the definition of domains in the relational model.

Example  **attribute:** lecturerStatus
Description: allowable status assigned to a lecturer
Type: character {L, SL, PL, Reader, Prof, HOD}

16.7.4 DOCUMENTING INTEGRITY CONSTRAINTS

A number of integrity constraints are inherent to the E-R data model itself.

Examples  Suppose we describe the relationship between the entity lecturer and the entity department as being one-to-many from department to lecturer. We also add that both entities are mandatory in the employs relationship. Doing this means that we are explicitly documenting a so-called existence constraint. In other words, a department cannot exist without any lecturers.

Now take the following constraint:

No lecturer should earn more than his head of department

There is currently no mechanism available for expressing this constraint in the graphic notation of the E-R approach. This is therefore referred to as an additional constraint.

Because of the very diverse nature of additional constraints it is unlikely that we will ever arrive at a satisfactory graphic notation for expressing such constructs. Instead, it is more practical to capture such constraints in a data dictionary. Such constraints may either be added to entity, relationship or attribute information, or stored separately.

16.8 VIEW INTEGRATION

In the sections above we have discussed the process of building a single conceptual data model for an application. However, the process described is a simplification of the data modelling process, particularly for large-scale data modelling. In Chapter 14 we have indicated that the process of conceptual modelling is usually broken up into two stages: view modelling and view integration.

View modelling is the process of transforming individual user requirements into a conceptual schema (or view). A view is an implementation-independent



map of data requirements constructed using the principles underlying some approach, the most popular of which is the E-R approach.

However, in any reasonably large database design project the development of a conceptual schema cannot be undertaken in one pass. This is because:

- The size of the design task usually means that a team of data analysts, each perhaps covering a different organisational area, will be engaged on the project
- For any particular information area there will usually be a number of different perspectives on data requirements. It may be important to capture each such perspective as a conceptual schema before resolution

The consequence of this is that for such projects a number of schemas are likely to result from the view modelling phase. View integration is the process of combining such individual views into a global, unified view which encapsulates the data requirements from all of the input views. An example of the type of project where view integration is particularly applicable is described in Beynon-Davies (1994).

16.8.1 DIFFERENCES IN VIEWS

Let us consider the simple case of two data analysts each developing views for the same universe of discourse. There are a number of ways in which two or more views drawn from the same universe of discourse may be different:

- *Synonyms*. Probably the most commonplace difference arises when the same real-world entity is named differently in two or more conceptual schemas – so-called synonyms
- *Homonyms*. In contrast, there may be the situation where the same label is used in two or more schemas, but the label actually describes two distinct real-world entities – so-called homonyms
- *Modelling flexibility*. Because of the inherent flexibility of the approach, a given real-world situation can be represented in more than one way using the constructs of E-R diagramming
- *Errors*. There is scope within any modelling technique for making errors. The advantage of a graphic technique such as E-R diagramming is that these errors are made more explicit

Examples ➔ In a university environment the same entity might be called a lecturer in one schema and an academic staff member in another. In contrast, we may have the entity student appearing in two schemas. However, in one schema it is used to refer to undergraduate students while in another schema the term is used to refer to postgraduate students.

In terms of modelling flexibility, the concept of a student's address might be documented as an entity or an attribute. One data analyst sees addresses as important things of interest and breaks an address down into the component attributes of streetNumber, streetName, town, county and postCode. Another data analyst sees addresses to be of lesser importance and decides to represent them as a simple string.

An example of errors being introduced is one where we may have the situation in which one data analyst has represented the relationship between students and projects as one-to-one. Perhaps a more flexible situation is represented by the data analyst documenting a many-to-many relationship.

16.9 CASE STUDY: VIEW INTEGRATION IN A UNIVERSITY SETTING

Let us now illustrate how some of these techniques are applicable within a small example of view modelling and view integration.

Two conceptual data models are illustrated in Figure 16.15. Both models represent attempts at documenting the same application – academic teaching. Each model however has been derived from a separate requirements elicitation exercise, perhaps with different groups in a university.

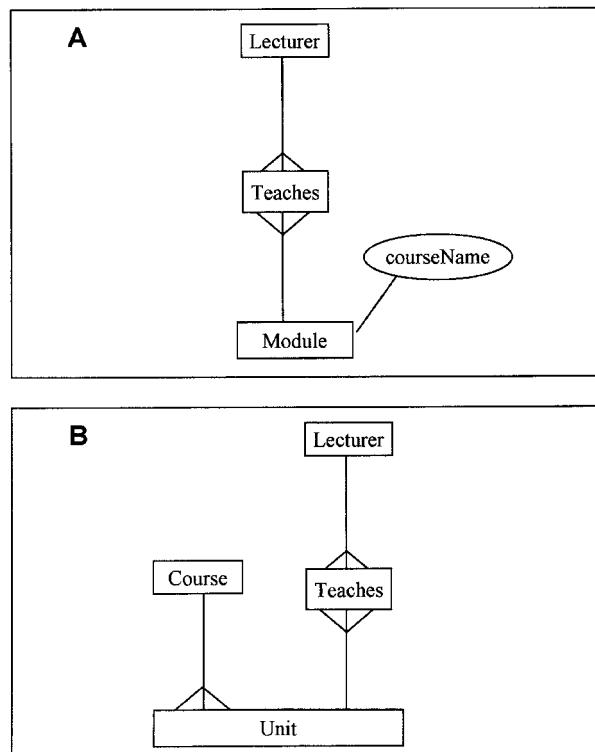


Figure 16.15 Two entity models.



The first logical step we might take is to attempt to identify synonyms in the two entity models. Examining each entity in turn we find that the *unit* entity in model B is a synonym for the *module* entity in A. Finding module to be a more general label than unit we choose to rename the entity in B module. Note that in the process of identifying synonyms we would also be checking for homonyms. We would ask ourselves questions like: is title the same attribute in A as it is in B?

The next step we might take is to merge elements in our data models that are currently represented by different E-R constructs. Consider, for example, the *courseName* attribute in A. Because we consider information about courses to be of fundamental importance to the academic setting, we feel that it should be promoted to the status of an entity. This brings it into line with the construction of data model B.

This means that an integrated view for this database might look as in Figure 16.16.

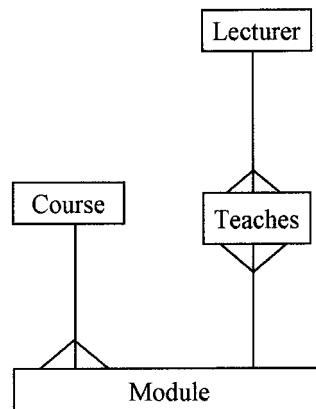


Figure 16.16 Integrated view.

A complete entity model for the entire universe of discourse might look like that in Figure 16.17.

Following the rules discussed in section 16.6 we may accommodate this entity model to a relational schema in the following manner:

```

Courses(courseNo, ...)
CourseRegistrations(courseNo, studentNo, ...)
Students(studentNo, ...)
Modules(moduleNo, ...)
ModuleAssessments(moduleNo, studentNo, ...)
ModuleRegistrations(moduleNo, studentNo, ...)
Lecturers(staffNo, ...)
Teaches(staffNo, moduleNo, ...)
Prerequisites(moduleNo, moduleNo, ...)
  
```

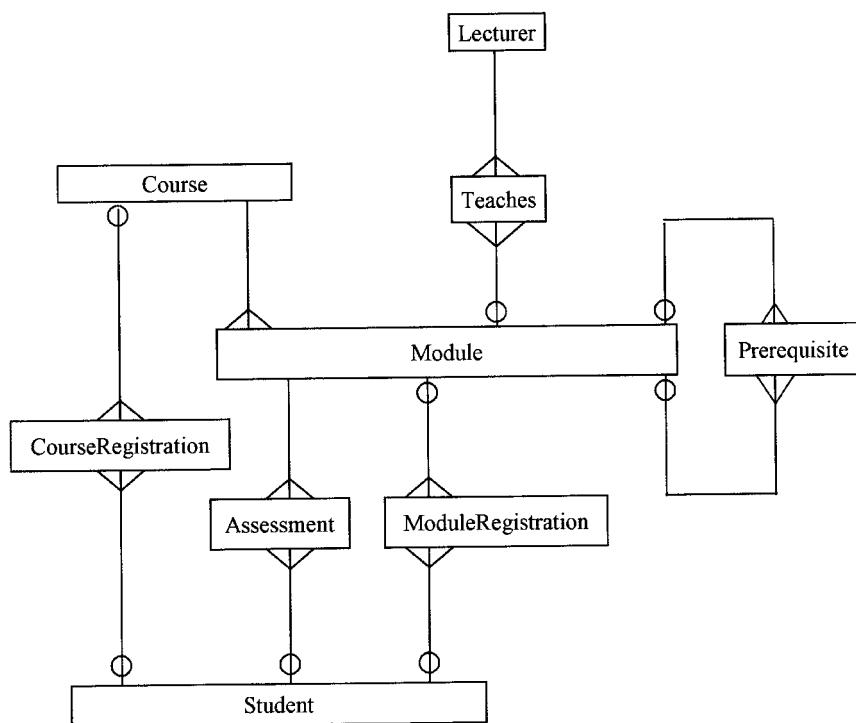


Figure 16.17 Complete entity model for the university domain.

16.10 SUMMARY

- E-R diagramming is a top-down approach to data analysis. Originally proposed as a data model, E-R diagramming is now normally used as a conceptual modelling technique
- There are three basic constructs in the E-R data model: entities, relationships and attributes
- An entity may be defined as a thing which the enterprise recognises as being capable of an independent existence and which can be uniquely identified
- A relationship is some association between entities. Relationships are characterised by two sets of rules: cardinality rules and optionality rules
- An entity is characterised by a number of properties or attributes
- A number of extensions have been proposed to the E-R approach: unary and ternary relationships, and roles. A number of other extensions are addressed in Chapter 25
- There is no agreed notation for representing entity models. A number of graphic notations may be used. An entity model may also be represented in the form of a data dictionary



- Schema modelling (sometimes known as view modelling) is the process of transforming individual user requirements into a conceptual schema (or view). Schema integration (sometimes known as view integration) is the process of combining individual schemas into a global, unified schema which encapsulates the data requirements from all of the input schemas

16.11

ACTIVITIES

1. Produce an E-R diagram which represents the following domain of criminal court cases. Each judge has a list of outstanding cases over which he will preside. Only one judge presides per case. For each case one prosecuting counsel is appointed to represent the Department of Public Prosecutions. Cases are scheduled at one Crown Court for an estimated duration from a given start date. A case can try more than one crime. Each crime can have one or more defendants. Each defendant can have one or more defending barristers. If a crime has multiple defendants, each defendant can have one or more defence counsel defending. Defendants may have more than one outstanding case against them.
2. Accommodate the diagram produced for activity 1 to a relational schema.
3. Produce an E-R diagram which represents the following domain of insurance. A policy-holder may have a number of policies with the insurance company. Each policy is given a policy number and relates to a single policy-holder. The company has a range of insurance products and may put together a range of products to form a policy. Examples of motor products are: third party, fire, theft, accident damage, windscreen cover etc. Brokers sell policies for commission and any one policy may have commission payable to more than one broker. Claims are made against policies. A claim relates to only one policy and each claim is classified according to one of six claim types. The company's products are grouped by business area, i.e. life, motor, marine etc. Any particular product belongs to only one business area. The company holds information on clubs and associations, for promotions and selective mailings. A holder may belong to a number of different associations. In order to limit risk the company may place all or part of a policy with reinsurers. All or part of a single policy may be placed with a number of different reinsurers.
4. Accommodate the diagram produced for activity 3 to a relational schema.
5. Produce an E-R diagram which documents the entities and relationships involved in a conceptual data dictionary (section 16.7).

16.12

REFERENCES

Beynon-Davies, P. (1994). Information management in the British National Health Service: the pragmatics of strategic data planning. *International Journal of Information Management* **14**(2 (April)): 84–94.

- Chen, P.P.S. (1976). The entity–relationship model: towards a unified view of data. *ACM Trans. on Database Systems* **1**(1): 9–36.
- Date, C.J. (2000). *An Introduction to Database Systems*. Reading, MA, Addison-Wesley.
- Howe, D. (1981). *Data Analysis for Database Design*. London, Edward Arnold.
- Klein, H.K. and R.A. Hirschheim (1987). A comparative framework of data modelling paradigms and approaches. *Computer Journal* **30**(1): 8–14.
- Martin, J. and C. McClure (1985). *Diagramming Techniques for Analysts and Programmers*. Englewood Cliffs, NJ, Prentice-Hall.
- Shave, M.J.R. (1981). Entities, functions and binary relations: steps to a conceptual schema. *Computer Journal* **24**(1).
- Teorey, T.J. (1994). *Database Modelling and Design: The Fundamental Principles*. San Mateo, CA, Morgan Kaufmann.



OBJECT MODELLING

The goal of all inanimate objects is to resist man and ultimately defeat him.

Russell Baker (1925–)

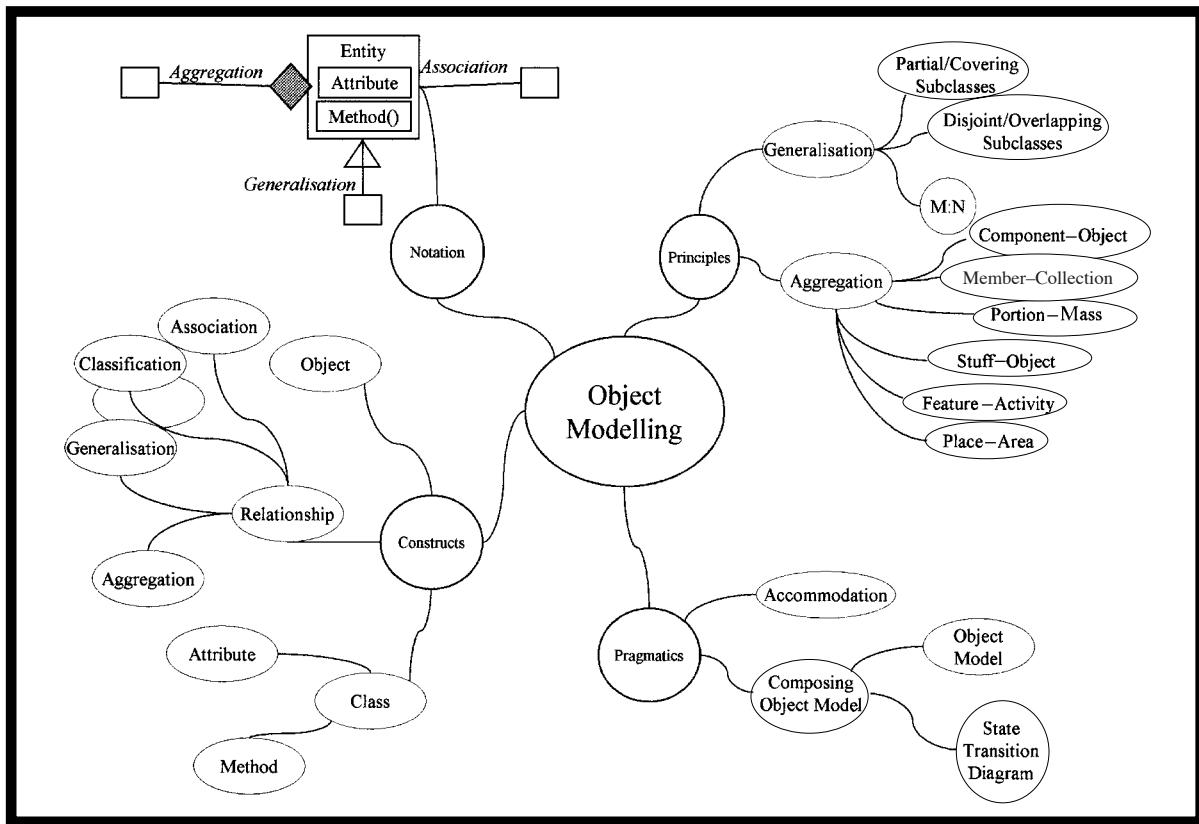
The creation of something new is not accomplished by the intellect but by the play of instinct acting from inner necessity. The creative mind plays with the objects it loves.

Carl Jung (1875–1961)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Discuss the links between object modelling and entity–relationship diagramming
- Discuss the key constructs of object modelling
- Practise some of the key principles of object modelling
- Describe the process of composing an object model



17.1 INTRODUCTION

Object-oriented (OO) analysis has definitely had an impact upon systems analysis, systems design, programming, and also database systems. The main aim of this chapter is to portray how conceptual modelling as applied to database systems can move relatively painlessly into the domain of OO. We will discuss how OO analysis, an approach primarily directed at the building of applications in procedural or object-oriented languages, is equally relevant to the development of database systems.

Database modelling using the E-R approach has, as we have described in Chapter 16, much in common with object modelling (Blaha *et al.*, 1988). In this chapter we consider some of the common threads between entity–relationship and object-oriented analysis and discuss some proposals for extending entity–relationship modelling into an object modelling approach.

17.2 A UNIVERSE OF DISCOURSE

To provide some concrete material for our discussion we shall use the universe of discourse below, modelled loosely on the activities of the UK stock market.



Example → The stock market is a market for the purchase and sale of securities. Securities come in two major forms: stocks and shares. A stock, sometimes known as a gilt-edged security or gilt, is a security with an associated interest rate. The most important type of stock is government bonds. Shares are a type of security which pay no interest, but pay a dividend to shareholders at regular intervals. Shares are normally issued by companies to raise capital.

All securities are initially placed on the stock market at an issue price: a price in pence per share. Once a share has been issued it can be traded: that is, bought and sold on the stock market.

Stocks can have variable interest rates or fixed interest rates. Fixed interest rate stock is sometimes called debenture or loan capital. Similarly, shares can offer fixed or variable dividends. Fixed dividend shares are sometimes known as preference capital; variable dividend shares are known as equity capital.

Persons or institutions which deal in securities on the stock market are known as financial intermediaries. There are two main types of financial intermediary: brokers and market makers.

Securities are bought from certain registered marketmakers. Marketmakers are normally financial institutions that opt to deal in a limited number of securities. The collection of such securities held by a marketmaker is known as the marketmaker's 'book'.

Each marketmaker will define the state of each type of share it holds in terms of two prices: the offer price and the bid price. The offer price is the price at which a marketmaker is willing to sell a share: the price at which an investor will buy. The bid price is the price a marketmaker is willing to pay for a share: the price at which an investor can sell to him. The difference between the two prices is known as the marketmaker's 'spread'. Different marketmakers will quote different spreads on shares depending on the state of their book. For example:

Allied Metals

	Marketmaker A	MarketMaker B
Offer	102	103
Bid	100	101

Brokers act as intermediaries between investors and marketmakers. Marketmakers act as intermediaries between corporations and brokers. Brokers purchase securities on behalf of investors, and/or sell securities to marketmakers on behalf of an investor. On both such transactions brokers normally charge a commission, generally a percentage of each share price. Under the recent reorganisation of the market, marketmakers may also act in the capacity of brokers. However, if a marketmaker deals with itself, in its role as marketmaker it must equal the best price offered on the market.

To conduct a deal, an investor issues a broker with an order specification. Such a specification normally includes:

- The name of the security
- Whether the order is to purchase or to sell securities
- The size of an order
- The time for which the order is to remain outstanding. Day orders should terminate at the end of the day's trading; open orders remain in force until filled by a broker or cancelled by an investor
- Type of order. A market order indicates that the broker should get the best price at the time; a limit order specifies a price below which the broker should buy or above which the broker should sell; a stop order is the reverse of a limit order

17.3 OBJECT MODELS

In Chapter 16 we defined an entity as being something of interest to an organisation which has an independent existence. This abstract definition can serve equally well for defining objects. The major difference between entities and objects lies in the way we apply these constructs in the modelling process. Entities are primarily static constructs. An entity model gives us a useful framework for painting the structural detail of a database system. In other words, an entity model usually translates into data structures and relationships between data structures in some database system.

Objects however have a more ambitious purpose (Blaha and Premerlani, 1997). An object is designed to encapsulate both a structural and a behavioural aspect. In other words, an object model gives us a means for designing not only a database structure but also how that database structure is to be used. An object model not only models data structures, it also models constraints and transactions. The most natural implementation medium for object models is therefore a DBMS which exploits object-oriented principles as described in Chapter 8.

The easiest way to begin to build an object model is to exploit some of the inherent strengths of entity–relationship modelling and extend them with structural and behavioural abstractions. An object model is a data model plus structural and behavioural abstractions. An object model is a set of data requirements expressed using the object-oriented concepts discussed in Chapter 8.

Objects and classes

It is important to reiterate the distinction made in Chapter 8 between objects and object classes: an object is an instance of something; an object class is a grouping of similar objects.



Example Hence, Paul Beynon-Davies is an object, Lecturer might be an appropriate object class.

17.4

STRUCTURAL ABSTRACTION

Three types of relationships or relations can occur in an object model: association relationships, generalisation relationships and aggregation relationships. Association relationships were considered in Chapter 16. Relationships on an E-R diagram are relationships of association. In this chapter we consider generalisation and aggregation relationships. These relationships are sometimes referred to as abstraction relationships or mechanisms because they provide means of hiding detail in abstraction hierarchies.

17.5

GENERALISATION

Generalisation is the process of extracting common features from a group of object classes and suppressing the detailed differences between object classes. In practice, generalisation allows us to declare certain object classes as subclasses of other object classes.

Example

For instance, *Stock* and *Share* might be seen as subclasses of a *Security* object. Likewise, *Debenture* and *VariableStock* might be considered subclasses of *Stock*. In this way a generalisation hierarchy may be built up in which classes further up the hierarchy cover more real-world phenomena than classes lower down the hierarchy.

17.5.1 INHERITANCE

The important consequence of this process of generalisation is that objects lower down the generalisation hierarchy are said to take on the attributes and relationships of objects higher up the hierarchy.

Example

Hence, debentures will take on the *interestRate* attribute of Stocks, and both stocks and shares will take on the class *Security*'s relationship with the class *Investor*.

17.5.2

GENERALISATION, CARDINALITY AND OPTIONALITY

A generalisation relationship is always characterised by the same optionality and cardinality. At the superclass level, the optionality is always optional and



the cardinality is always one. At the subclass level, optionality is always mandatory and cardinality is always one.

Example → In other words, if *Security* generalises *Share* then every *Share* is a *Security* but not every *Security* is a *Share*.

17.5.3 GENERALISATION LATTICES AND MULTIPLE INHERITANCE

Many applications of the concept of generalisation do not fall into neat hierarchies. In such cases we speak of a generalisation lattice. In other words, a given object class may be a subclass of more than one superclass. Such an object class is then said to be subject to multiple inheritance. It inherits the attributes and relationships from more than one superclass.

Example → A *MarketMaker* class could be said to be a subclass of both an *Investor* class and a *FinancialIntermediary* class.

17.5.4 SPECIALISATION

The opposite of generalisation is referred to as specialisation. Specialisation is the process of creating a new object class by introducing additional detail to the description of an existing object class.

Example → For instance, *OpenOrder* is a specialisation of *Order*. Open orders are orders that do not need to be date-stamped.

17.5.5 GENERALISATION AND CLASSIFICATION

A distinction is sometimes made between generalisation and classification. Generalisation can be considered as the process of extracting from one or more object classes the description of a more general class. Classification involves grouping objects that share common characteristics into a class object. The opposite process to classification is said to be instantiation.

Examples → For instance, the statement BT1 and BG1 are shares indicates classification. BT1 would be said to be an instance of a share.



17.5.6 ISA AND AKO

The distinction between classification and generalisation can be expressed as the difference between an object to object class relation and object class to object class relation. This is similar to a distinction sometimes made in the Artificial Intelligence literature between ISA (Is-A) and AKO (A-Kind-Of) relations (Brachman, 1983):

- ISA relates objects to object classes.
- AKO relates an object class to another object class.

Examples When we state *BT1 is a share* we are defining a particular financial instrument as being an instance of the class Share. When we state that *shares are a kind of security* we are defining a Share to be a specialisation of a Security.

17.5.7 PROPERTIES OF GENERALISATION AND CLASSIFICATION RELATIONS

In formal terms, generalisation relations are transitive, irreflexive and anti-symmetric:

- *Transitive*. If A AKO B and B AKO C then A AKO C. This is really part of the definition of inheritance
- *Irreflexive*. A not AKO A
- *Anti-symmetric*. If A AKO B then B not AKO A

Examples *Transitive* – if programmers are computing staff and computing staff are a kind of employee, then programmers are employees.
Irreflexive – employees are not a kind of employee.
Anti-symmetric – if computing staff is a kind of employee, then employees are not a kind of computing staff.

Hence, AKO relations are usually understood to define strict partial ordering. Partial ordering relations structure a semantic space in a clear hierarchical fashion. Since ISA relations define links between objects and object classes it does not make sense to talk of the transitivity, or symmetry, of these relations.

17.5.8 PARTIAL AND COVERING SUBCLASSES

It is useful to make a distinction between partial and covering subclasses. If subclasses are partial then other subclasses can be included. If subclasses are covering, then no further subclasses are permitted.



Example Hence, if we regard *Broker* and *MarketMaker* as partial subclasses of *FinancialIntermediary*, then other subclasses are possible. If these subclasses are covering then brokers and marketMakers would be the only type of financialIntermediary permitted on the UK stock market.

17.5.9 DISJOINT AND OVERLAPPING SUBCLASSES

Disjoint subclasses do not overlap. However, we can conceive of real-world situations where the concepts embodied in object classes do overlap. If all subclasses in an object model are disjoint we have a strict hierarchy of classes. If some are overlapping we have a lattice structure. Overlapping classes are another facet of multiple inheritance.

Examples Hence, Share and Stock are disjoint subclasses of Security. A Security cannot be both a share and a stock. Broker and MarketMaker are two overlapping subclasses of financial intermediary since marketMakers can act as brokers.

17.5.10 TEXTUAL NOTATION

Generalisation and classification can be written as simple assertions linking objects and object classes via the keywords ISA and AKO.

Example For instance:

Share AKO Security
Stock AKO Security
BT1 ISA Share
BG1 ISA Share

We assume that unless stated explicitly, subclasses are assumed to be disjoint. Overlapping subclasses can be represented by assertions such as that in the following example.

Example Broker Overlaps MarketMaker

We further assume that the default mode for subclasses is covering. A superclass with partial subclasses can be represented by an assertion such as that in the following example.

**Example**

? AKO FinancialIntermediary

This implies that there is some unspecified set of subclasses for the class FinancialIntermediary.

17.5.11 GRAPHICAL NOTATION

A number of different graphical notations have been proposed for representing object models. One recent development having a significant impact on the area of OO analysis and design is the attempt by three 'gurus' of the field, Grady Booch (Booch), James Rumbaugh (OMT) and Ivar Jacobson (OOSE), to develop a Unified Modelling Language (UML) for OO analysis and design. Because of its developing use as an open standard we use UML as the graphical notation in this chapter (Booch *et al.*, 1999; Jacobson *et al.*, 1999).

In UML, generalisation is indicated by a line between subclass and superclass with a triangle placed at the head of the line next to the superclass. Disjoint generalisation is represented by the labels *disjoint* or *overlapping* expressed in brackets and placed next to the triangle. Partial generalisation is indicated by the keywords *incomplete* or *complete* expressed in a similar way. Hence, the way in which the subclasses of Order are drawn in Figure 17.1 indicates that these are disjoint, covering subclasses.

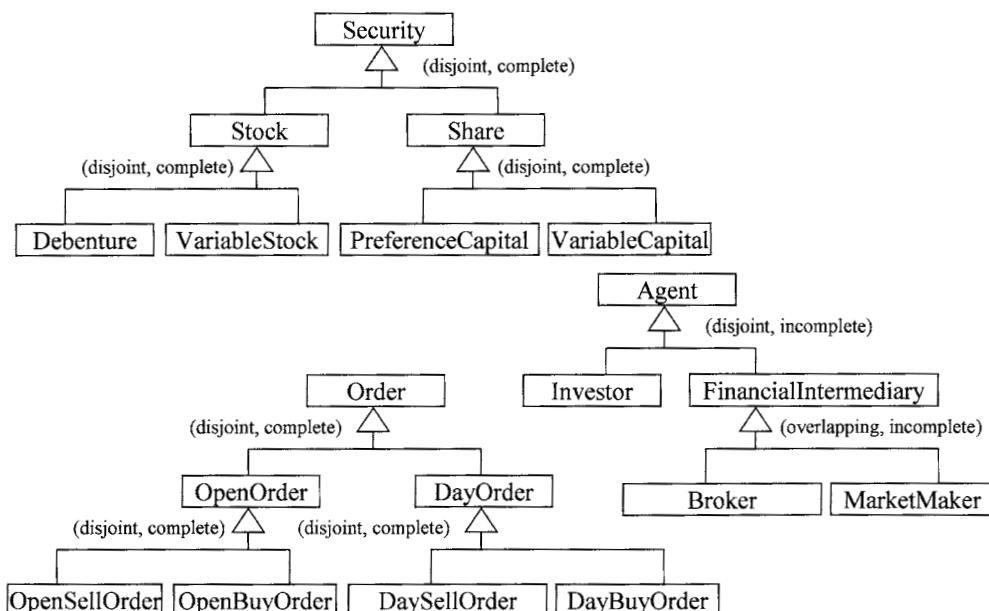


Figure 17.1 Generalisation in stock market model.

17.5.12 ACCOMMODATING GENERALISATION HIERARCHIES TO RELATIONAL SCHEMA

Clearly, the relational model has no concept of a generalisation hierarchy. Generalisation therefore has to be represented by appropriate design of a relational schema. There are a number of choices for accommodating a superclass–subclass relationship to the relational data model:

- Create one table. Include all the attributes of the superclass plus all the attributes of each subclass in the table
- Create one table for each subclass. Include all the attributes for the superclass in each subclass table
- Create one table for each subclass and one table for the superclass

Example  Hence, considering the classes Security, Stock and Share in Figure 17.1 the various options are expressed in the bracketing notation below:

- Securities(securityName, securityType, issuePrice, dividend, interestRate)
- Stocks(securityName, issuePrice, interestRate); Shares(securityName, issuePrice, dividend)
- Securities(securityName, issuePrice); Stocks(securityName, interestRate); Shares(securityName, dividend)

The third option is probably the most flexible in that new subclasses can be added easily. The first and second options reduce the amount of joining in generalisation operations and hence are likely to be more computationally efficient in relational terms.

17.6 AGGREGATION

An aggregation relationship occurs between a whole and its parts (Smith and Smith, 1977). An aggregation is an abstraction in which a relationship between objects is considered a higher-level object. This makes it possible to focus on the aggregate while suppressing low-level detail. For example, a FinancialPortfolio can be considered as an aggregate of securities, insurance policies and savings accounts. A country can be considered as an aggregate of regions which are aggregates of counties which are aggregates of districts, and so on. Hence, an aggregation hierarchy can be assembled.

The opposite of aggregation is decomposition; that is, the process of decomposing an object class into its constituent parts.

17.6.1 DIFFERENCE BETWEEN AGGREGATION AND GENERALISATION

It is possible to distinguish between aggregation and generalisation in the following way. If two classes are defined in terms of a generalisation relationship then both subclass and superclass effectively refer to the same thing.



Example Hence if we state that BT1 ISA Share then we are also saying that BT1 is also a Security. When two classes are defined in terms of an aggregation relationship they refer to different things. A share is a distinct thing from a FinancialPortfolio.

We shall elaborate on the types of aggregation relation below.

17.6.2 OBJECT CLASSES AND ATTRIBUTES

An object class could be considered as an aggregation of attributes. Similarly an object could be considered as an aggregation of its properties. We prefer to consider the relation between an object class and an attribute to be an instance of an attribution relation.

Example Hence, to state that a person can be characterised in terms of his or her height we write:

Person HASA height

More usually the term aggregation is used to refer to an object class which collects together a series of other object classes. An object class is seen to be a component part of some other object class.

Example It is nonsensical to say that the height of a person is a component part of the person. It is sensible to say that Personality is a component part of a person.

17.6.3 MERONYMIC RELATIONS

Winston *et al.* (1987) call the range of semantic relations between the parts of things and the wholes which they comprise meronymic relations, after the Greek word ‘meros’ for part. They distinguish six major types of meronymic relationship, not all of which are exploited in commercial object modelling:

- Component-object
- Member-collection
- Portion-mass
- Stuff-object
- Feature-activity
- Place-area



These meronymic relations are distinguished in three major ways:

- *Functionality* – whether the relation of the part to the whole is functional, or not. Functional parts are restricted by their function in terms of their spatial or temporal location
- *Homeomerous* – parts are the same kind of thing as their whole, while non-homeomerous parts are different from their wholes
- *Separation* – parts can in principle be separated from their whole, while inseparable parts cannot

Examples ➔

- Functionality – a handle of a cup can only be placed in a limited number of positions if it is to function as a handle
- Homeomerous – a slice of a pie is still a pie, but trees are different entities from forests
- Separation – a handle can in principle be separated from a cup, but steel cannot be separated from a bike

Component–Object

These are functional, separable but non-homeomerous relations.

Examples ➔

- A handle is part of a cup
- Wheels are part of cars
- Chapters are parts of books

In such cases an object is divided into its components in terms of some organised pattern or structure. Each component part has a function which can be separated from the whole. Each component is also in some way different from the whole.

Member–collection

These are separable but non-functional and non-homeomerous relations.

Examples ➔

- A tree is part of a forest
- A juror is part of a jury
- The ship is part of the fleet



Each part in this type of relationship is not the same as the whole and does not play a specific function in terms of the whole. The parts can be clearly separated out from the whole. Membership in the collection is determined purely on the basis of spatial proximity or social connection, unlike the case of a generalisation relation where membership is determined on the basis of similarity.

Portion–mass

These are homeomerous, separable but non-functional relations.

- | | |
|-------------------|--|
| Examples ➔ | The slice is part of a pie
A yard is part of a mile |
|-------------------|--|

The parts here are similar in nature to their wholes but can be separated out from the whole. The parts do not have any functional relationship to their whole. The portion–mass relation forms the basis for the idea of standard measure and the arithmetic operations of addition, multiplication and division.

Stuff–object

These are non-homeomerous, non-functional and non-separable relations.

- | | |
|-------------------|---|
| Examples ➔ | The bike is partly steel
Water is partly hydrogen
A martini is partly alcohol |
|-------------------|---|

This particular relation expresses the idea that a particular type of substance constitutes a portion of the total stuff of which an object is made. However, the portion cannot be easily separated out from the substance and will be different in isolation from the substance. In commercial object modelling, this type of meronymic relation would usually be represented as an attribution relation.

Feature–activity

These are functional, non-homeomerous and non-separable relations.

- | | |
|-------------------|--|
| Examples ➔ | Paying is part of shopping
Ovulation is part of the menstrual cycle |
|-------------------|--|



This idea is similar to the AI concept of a script in which a complex activity is broken down into particular subactivities or features. This form is frequently exploited in the hierarchical specification of methods.

Place-area

These relations are non-functional, homeomerous and non-separable. They are used to relate areas to places or locations within them.

- | | |
|-------------------|--|
| Examples → | The Everglades are part of Florida
An oasis is part of a desert |
|-------------------|--|

Places are not parts by virtue of any functional contribution to the whole. Every location shares common features with all other areas, but no location can be separated from the area of which it is a part.

17.6.4 PROPERTIES OF AGGREGATION RELATIONS

Like generalisation relations, aggregation relations are transitive, irreflexive and anti-symmetric:

- *Transitive.* If A is part of B and B is part of C then A is part of C
- *Irreflexive.* A is not a part of A
- *Anti-symmetric.* If A is a part of B then B is not a part of A

Hence, aggregation relations are usually understood to define strict partial ordering. Partial ordering relations structure a semantic space in a clear hierarchical fashion.

17.6.5 TEXTUAL NOTATION

An aggregate object can be specified as a series of assertions with the relation type `partOf`.

- | | |
|-------------------|---|
| Examples → | RailwayStation partOf RailwayLine
Wheel partOf Car
Receptionist partOf Company
Demonstration partOf Presentation |
|-------------------|---|



17.6.6 GRAPHICAL NOTATION

Graphically we may depict aggregation as a series of lines or a forked line between whole or parts. A filled diamond is also placed next to the aggregate class. Figure 17.2 illustrates a sample aggregation relationship.

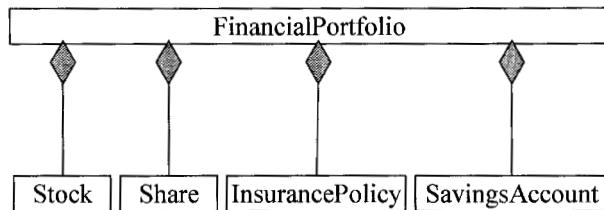


Figure 17.2 Aggregation.

17.6.7 ACCOMMODATING AGGREGATES TO RELATIONAL SCHEMA

Clearly, the relational model has no concept of an aggregate except in the sense of aggregating a group of related attributes together in a relation. The idea of nested relations discussed in Chapter 10 is an attempt to extend the relational model to include aggregation. In the pure relational approach however, aggregation can be simulated by creating a separate table for each of the component classes and a table for the aggregate class. A foreign key can then be posted from the aggregate class table into each of the component tables.

Example → For instance, a *FinancialPortfolio* aggregate can be expressed as follows relationally:

```

Portfolios(portfolioNo, portfolioDate, ...)
Policies(policyNo, portfolioNo, ...)
Securities(securityName, portfolioNo, ...)
  
```

17.7 BEHAVIOURAL ABSTRACTION

Two major types of behavioural mechanism need to be documented in the design of a database system: constraints and transactions. Constraints define allowable states of a database. Transactions are the means of causing changes to a database.

Constraints and transactions are therefore interdependent. Constraints determine the acceptable behaviour of transactions. Transactions are the means of activating constraints.

This interdependence is reflected in an object model. Transactions are



represented usually by messages between objects. Constraints are generally embedded in the methods pertaining to objects.

17.7.1 GRAPHIC NOTATION

We turn an entity into an object class by defining its interface in terms of attributes and methods. This makes for a box divided into three parts: the class name, a list of attributes and the names of the most important methods defined for the object. In Figure 17.3 we represent the classes Stock and Share. Two methods are defined for each class: issueNewStock() and changeInterestRate(); issueNewShare() and changeDividend(). Rounded brackets merely indicate that we expect to pass a number of parameters to a defined method.

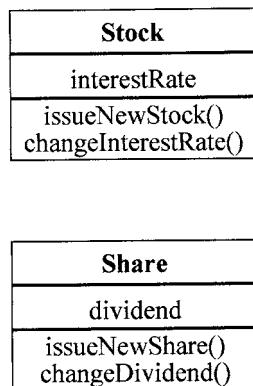


Figure 17.3 Object classes.

In a truly OO manner, transactions impact upon data (attributes) through messages. Hence, a basic form of transaction would involve sending a message to a class and invoking a method of that class such as changeInterestRate.

Methods may also be used to incorporate integrity constraints. Hence, the issueNewShare method might perform a check to see that no share currently exists with the specified name.

17.7.2 TEXTUAL NOTATION

We can express the fact that a class has given methods in the following way:

<Class name> method <Method name>

Examples ➔

Share method issueNewShare()
Share method changeDividend()



Most of the methods needed by commercial database systems can be expressed in terms of a series of conditions and actions working on appropriate parameters.

Example → For instance, the issueNewShare method can be expressed as:

issueNewShare(shareName, shareType, marketMaker, dividend, issueOffer,
issueBid, issueAmount)

Conditions:

marketMaker exists

shareName does not exist

infer shareType ako share

Actions:

assert new object with shareName and dividend

17.8

OBJECT LIFE HISTORIES

In the previous section we illustrated how we can expand an E-R diagram to contain behavioural information. However, an object model of this form can only tell the developer the names of the major methods contained within each object. It gives us no idea of how methods relate both within a given class and between classes. In this section we look at one technique which begins to address this issue. A technique already widely used in conventional structured analysis, the entity life history has been adapted to the needs of object modelling.

17.8.1 ENTITY LIFE HISTORIES

One way to begin behavioural abstraction is to collect the set of events that characterise the typical life history of an object class. This is an adaptation of the idea of an entity life history (Beynon-Davies, 1998).

The entity life history (ELH) technique was originally designed to extend the available database design techniques such as E-R diagramming and normalisation (Robinson, 1979). Such data analysis techniques concentrate almost exclusively on a static view of the information system being modelled. ELHs, in contrast, were developed as a technique for the explicit modelling of system dynamics (Rosenquist, 1982). In this sense, the ELH may be seen as a competitor to the Data Flow Diagram and associated process specification techniques. However, the ELH is more closely linked to the modelling ideas underlying data analysis.

17.8.2 ENTITIES IN FLUX

From a traditional data modelling point of view, the entities on an E-R diagram are usually thought of as static or structural concepts. An entity model represents a time-independent slice of reality.



An entity may however be considered from a contrasting point of view. An entity is realistically in a state of flux. It is the subject of a large number of processes or events which change its state. In the stock market, for instance, the entity *share* may proceed through a number of different states: it is first issued, then it is traded, finally it is withdrawn from the market. An ELH is a diagrammatic technique for charting the usage of a particular entity by the processes or events making up an information system. An ELH is a technique which relates events to states.

17.8.3 EVENTS

An event is something that happens at a point in time. For conceptual purposes however we assume that an event has no duration. Events may be related in that one logically precedes or follows another. Events may also be unrelated, in which case we say they are concurrent. In the stock market, a typical event might constitute a deal made between an investor and a financial intermediary. An event concurrent with this one might be the issue of a new share.

Every event is unique. However, we may group events into event classes based on common features. Event classes are organised hierarchically in the same way as we organise object classes hierarchically.

Examples For instance, *Student Andrew James enrols on Module relationalDatabaseSystems* and *Student P Davies enrols on Module relationalDatabaseDesign* are both instances of the event class *Student enrolledOn Module*.

17.8.4 STATES

A state is a moment in the life of an object. When people talk of states they are really referring to state classes. A state class is an abstraction of the values of objects and the relationships of objects. A state is the response of an object to input events. A state represents the interval of time between two events impacting upon an object. Events represent points in time; states represent periods of time. A state therefore has duration; it occupies an interval of time.

Example The state of a *BankAccount* might be characterised as being within balance or overdrawn. It is within balance if the current balance of the account is greater than zero, and overdrawn otherwise. Various debits from a *BankAccount* may cause it to become overdrawn.



17.8.5 STATE TRANSITION DIAGRAMS

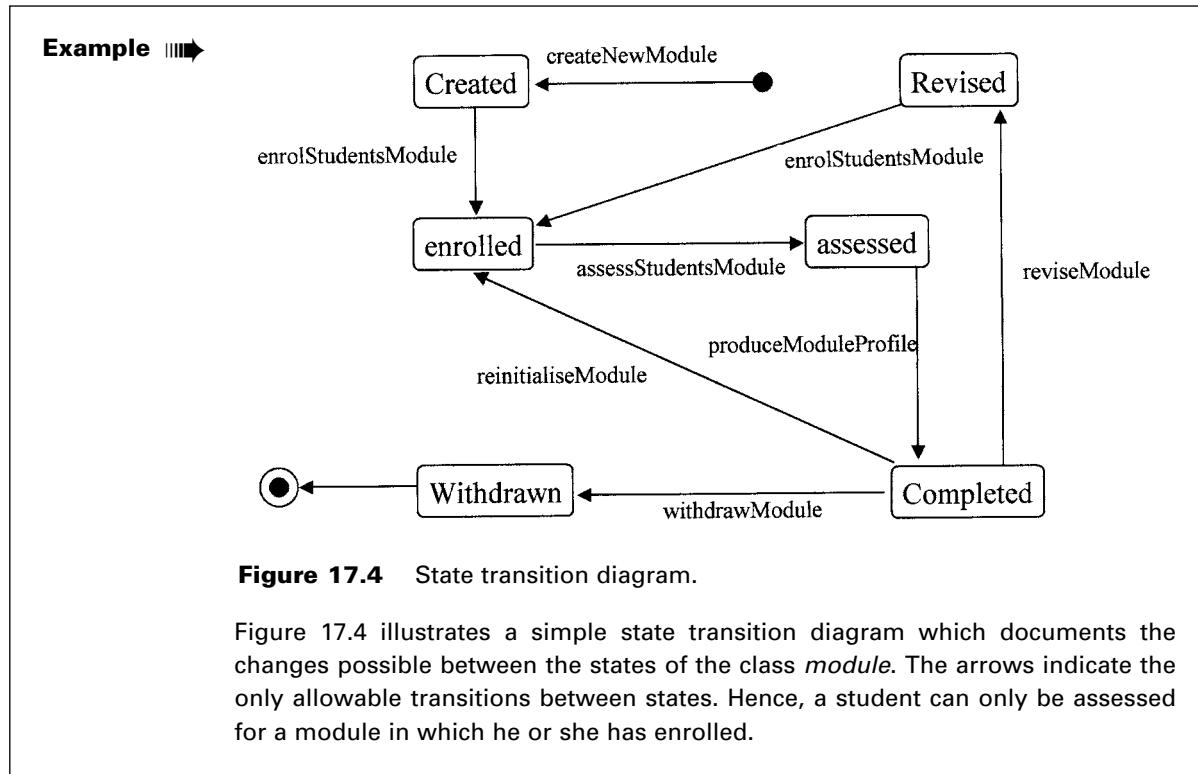
To model the behaviour of objects we need some notation which relates states and events in some form of history. In this section we shall illustrate the use of the notation of state transition diagrams for this purpose.

A state transition diagram documents a view of the world in which each object class in a database system can be regarded as a finite state machine. A finite state machine is a hypothetical mechanism which can be in one of a finite number of discrete states at one time. Events occur at discrete points in time and cause changes to the machine's state.

Note the assumption that change is not continuous. Change is modelled as a series of discrete events. This assumption is in fact implicit in the OO paradigm via its adoption of the ideas of methods and messages. It is also implicit in the use of the database concept of a transaction.

In UML, a state transition diagram is made up of the following components:

- States are drawn as labelled rounded boxes
- A double circle denotes a final state
- A circle with a dotted arrow pointing to it denotes the initial state
- An arc between two states denotes an event. The arc represents a change of state in the direction of the arrowhead



17.9 CASE STUDY: COMPOSING AN OBJECT MODEL FOR THE STOCK MARKET

It is impracticable for an object model to be specified on a single diagram. We would expect an object model to be specified as a series of related parts. For instance, Figure 17.5 represents an extended E-R diagram for the stock market universe of discourse. This sets the context for the object model. It details the major classes in the domain, the major association relationships and one example of generalisation.

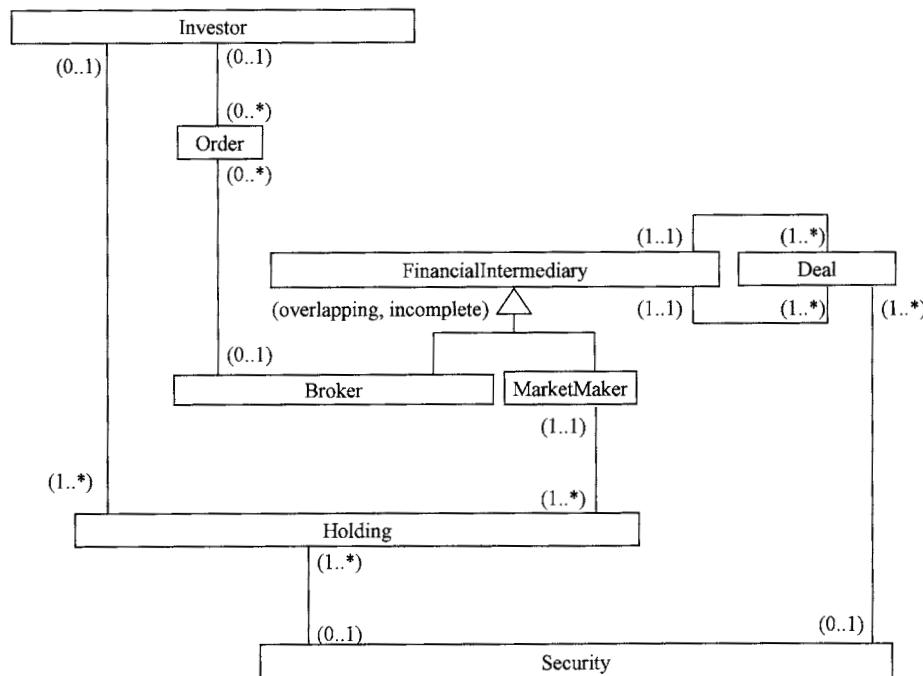


Figure 17.5 Stock market object model.

The association relationships have cardinality and optionality expressed in a different way from that discussed in Chapter 16. The two figures in brackets at the end of each relationship represent the lower bound and the upper bound of cardinality for the entity in the relationship. Hence, the following meaning applies to the notation:

- (0..*) – (zero or more)
- (1..*) – (1 or more)
- (1..1) – (1 and only one)

Figure 17.5 extends this information by illustrating the fact that generalisation is relevant to a number of the entities on the context diagram. Figure 17.6 provides a series of examples of classes with associated methods.



Holding	Deal	Broker
Attributes: holdingId offerPrice bidPrice amountHeld Operations: addSecurityHolding() changeOfferPrice() changeBidPrice() incrementHolding() decrementHolding() listBook() listSpread() lowestOfferPrice() lowestBidPrice()	Attributes: dealId dealPrice dealAmount Operations: marketMakerDeal() brokerDealBuy() brokerDealSell() listSales() listPurchases()	Attributes: brokerName Operations: addBroker() removeBroker() updateBrokerBalance()

Figure 17.6 Sample stock market classes.

Below we illustrate some sample templates for a number of the methods in the model.

addBroker(broker, commission, balance)

Conditions: broker does not exist

Actions: assert new broker with commission and balance

removeBroker(broker)

Conditions: broker exists

Actions: remove broker with commission and balance

/ strike a deal between two marketmakers */*

marketMakerDeal(securityName, buyer, seller, amount)

Conditions: securityName is a security; buyer and seller are marketMakers;

seller holds securityName; amount <= amountHeld by seller

Actions: decrement seller holding; increment seller balance;

increment buyer holding; decrement seller balance; record deal

Finally, Figure 17.7 represents a simple state transition diagram for the class *share*.

17.10

SUMMARY

- Object modelling is the extension of E-R diagramming with two types of extensions: structural abstraction and behavioural abstraction
- Structural abstraction incorporates the relationships of generalisation and aggregation
- Behavioural abstraction incorporates the concepts of methods and messages
- Object models provide a new dimension for database design. They allow the integration of process analysis with the proven traditions of data analysis

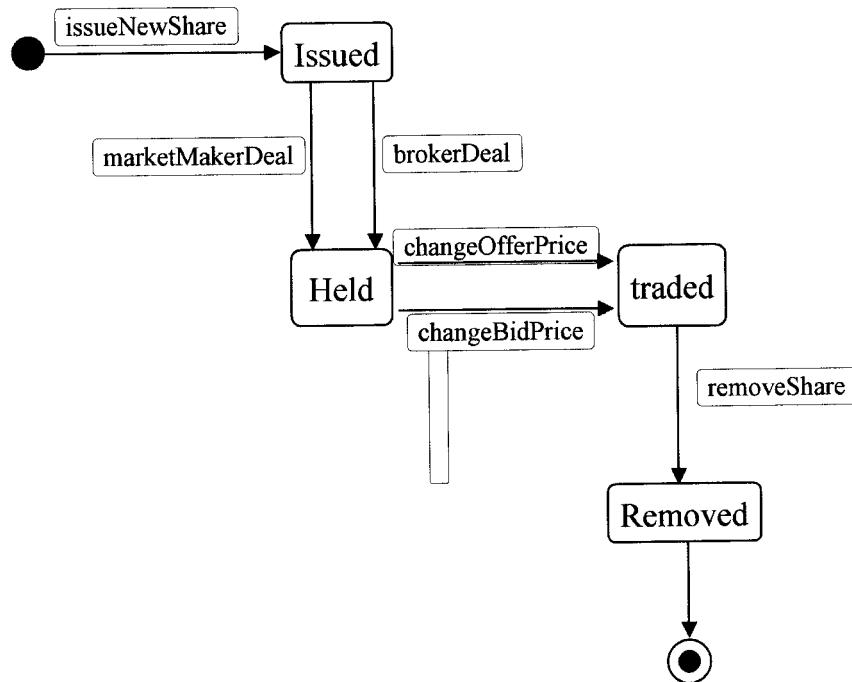


Figure 17.7 State transition diagram for share.

- Object models can be used to develop the newer OO databases, the more conventional databases such as relational systems, and hybrids such as OO extensions to relational systems

17.11 ACTIVITIES

- Indicate how generalisation relationships might apply to modules, students and lecturers.
- What aggregation relationships might be identifiable in a university domain?
- Identify suitable methods for the classes Student, Module and Lecturer.
- Specify each method in terms of conditions and actions.
- Attempt to draw state transition diagrams for the classes Student and Lecturer.
- In section 17.2 a marketmaker's book is defined. How might this construct be modelled in terms of object modelling?
- Express a complete specification for the stock market scenario as an object model.

17.12  REFERENCES

- Beynon-Davies, P. (1998). *Information Systems Development: An Introduction to Information Systems Engineering*. Basingstoke, Macmillan (now Palgrave).
- Blaha, M. and W. Premerlani (1997). *Object-Oriented Modelling and Design for Database Applications*. Englewood Cliffs, NJ, Prentice-Hall.
- Blaha, M.R., W.J. Premerlani and J.E. Rumbaugh (1988). Relational database design using an object-oriented methodology. *Comm. of ACM* **31**(4): 414–27.
- Booch, G., J. Rumbaugh and I. Jacobson (1999). *The Unified Modelling Language User Guide*. Reading, MA, Addison-Wesley.
- Brachman, R.J. (1983). What ISA is and isn't: an analysis of taxonomic links in semantic networks. *Computer* **16**(10): 30–6.
- Jacobson, I., G. Booch and I. Rumbaugh (1999). *The Unified Software Development Process*. Reading, MA, Addison-Wesley.
- Robinson, K.A. (1979). The entity/event modelling method. *Computer Journal* **22**: 270–81.
- Rosenquist, C.J. (1982). Entity life-cycle models and their applicability in information systems development. *Computer Journal* **25**: 307–15.
- Smith, J.M. and D.C.P. Smith (1977). Database abstractions: aggregation and generalisation. *Transactions of Database Systems* **2**(2).
- Winston, M., R. Chaffin and D. Herrmann (1987). A taxonomy of part–whole relations. *Cognitive Science* **11**: 417–44.

CHAPTER 18

NORMALISATION

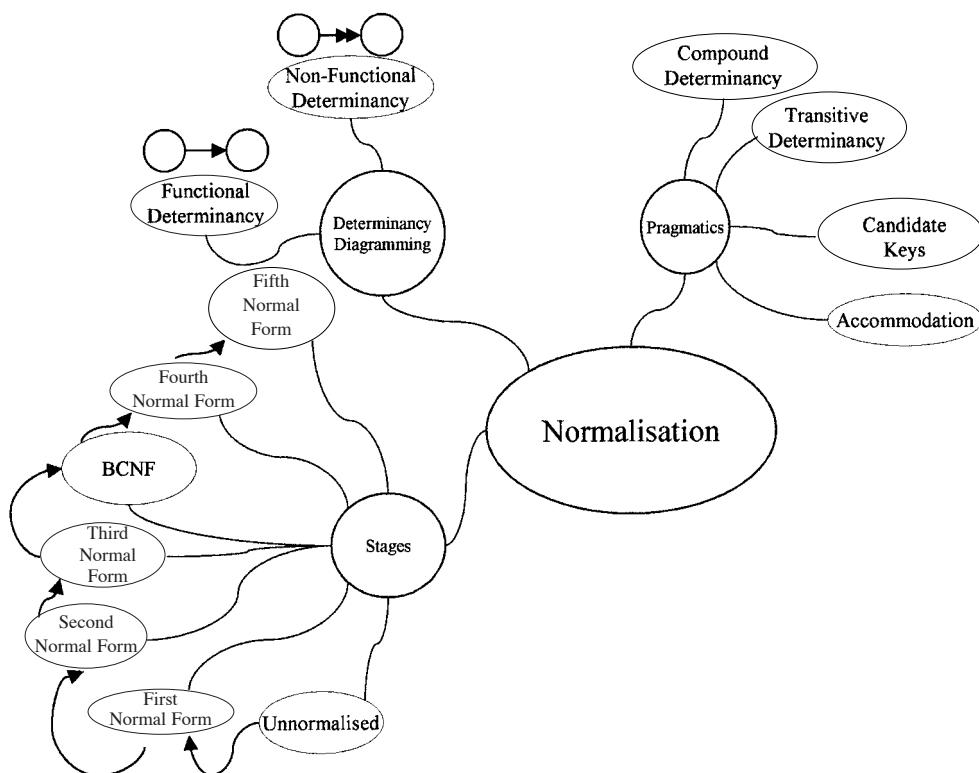
Success is more a function of consistent common sense than it is of genius.

An Wang

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Describe why normalisation is important
- Relate the five normal forms
- Develop a diagram based upon determinancies
- Accommodate a determinancy diagram to a relational schema



18.1 INTRODUCTION

In his seminal paper on the relational data model, E.F. Codd formulated a number of design principles for a relational database (Codd, 1970). These principles were originally formalised in terms of three normal forms: first normal form, second normal form and third normal form. The process of transforming a database design through these three normal forms is known as normalisation. By the mid-1970s third normal form was shown to have certain inadequacies and a stronger normal form, known as Boyce–Codd normal form (BCNF), was introduced (Codd, 1974). Subsequently Fagin introduced fourth normal form and indeed fifth normal form (Fagin, 1977; 1979).

In this chapter we consider Codd's original ideas on normalisation while also describing a graphic technique used for designing fully normalised schema. We particularly emphasise the use of normalisation as a logical modelling technique for database design.

18.2 WHY NORMALISE?

Suppose we are given the brief of designing a database to maintain information about students, modules and lecturers in a university. An analysis of the documentation used by the administrative staff gives us the following sample data-set with which to work. If we pool all the data together in one table as below, a number of problems, sometimes called file maintenance anomalies, would arise in maintaining this data-set.

Modules						
moduleName	staffNo	staffName	student	student No	ass Grade	ass Type
Relational Database Systems	234	Davies T	34698	Smith S	B3	cwk1
Relational Database Systems	234	Davies T	34698	Smith S	B1	cwk2
Relational Database Systems	234	Davies T	37798	Jones S	B2	cwk1
Relational Database Systems	234	Davies T	34888	Patel P	B1	cwk1
Relational Database Systems	234	Davies T	34888	Patel P	B3	cwk2
Relational Database Design	234	Davies T	34698	Smith S	B2	cwk1
Relational Database Design	234	Davies T	34698	Smith S	B3	cwk2
Deductive Databases	345	Evans R	34668	Smith J	A1	exam

- What if we wish to delete student 34668? The result is that we lose some valuable information. We lose information about deductive databases and its associated lecturer. This is called a deletion side-effect
- What if we change the lecturer of deductive databases to V Konstantinou? We need to update not only the staffName but also the staffNo for this module. This is called an update side-effect

- What if we admit a new student on to a module, say studentNo 38989? We cannot enter a student record until a student has had at least one assessment. This is known as an insertion side-effect

The size of our sample file is small. One can imagine the seriousness of the file maintenance anomalies mentioned above multiplying as the size of the file grows. The above structure is therefore clearly not a good one for the data of this enterprise. Normalisation is a formal process whose aim is to eliminate such file maintenance anomalies.

18.3 (c) STAGES OF NORMALISATION

Normalisation is carried out in the following steps:

- Collect the data-set – the set of data-items
- Transform the unnormalised data-set into tables in first normal form
- Transform first normal form tables to second normal form
- Transform second normal form tables to third normal form

Occasionally, the data may still be subject to anomalies in third normal form. In this case, we may have to perform further steps:

- Transform third normal form to Boyce–Codd normal form
- Transform third normal form to fourth normal form
- Transform fourth normal form to fifth normal form

The process of transforming an unnormalised data-set into a fully normalised (third normal form) database is frequently referred to as a process of non-loss decomposition. This is because we continually fragment our data structure into more and more tables (through relational projects, see Chapter 7) without losing the fundamental relationships between data-items.

18.3.1 DETERMINANCY/DEPENDENCY

Normalisation is the process of identifying the logical associations between data-items and designing a database which will represent such associations but without suffering the file maintenance anomalies discussed in section 18.2. The logical associations between data-items that point the database designer in the direction of a good database design are referred to as determinant or dependent relationships. Two data-items, A and B, are said to be in a determinant or dependent relationship if certain values of data-item B always appear with certain values of data-item A.

Determinancy/dependency also implies some direction in the association. If data-item A is the determinant data-item and B the dependent data-item then the direction of the association is from A to B and not vice versa.

There are two major types of determinancy or its opposite dependency: functional (single-valued) determinancy, and non-functional (multi-valued)



determinacy. We introduce here the concept of functional determinancy. Non-functional determinancy is discussed below.

Data-item B is said to be functionally dependent on data-item A if for every value of A there is one, unambiguous value for B. In such a relationship data-item A is referred to as the determinant data-item, while data-item B is referred to as the dependent data-item. Functional determinancy is so-called because it is modelled on the idea of a mathematical function. A function is a directed one-to-one mapping between the elements of one set and the elements of another set.

Example → In a university personnel database, staffNo and staffName are in a functional determinant relationship. StaffNo is the determinant and staffName is the dependent data-item. This is because for every staffNo there is only one associated value of staffName. For example, 345 may be associated with the value J. Smith. This does not mean to say that we cannot have more than one member of staff named J. Smith in our organisation. It simply means that each J. Smith will have a different staffNo. Hence, although there is a functional determinancy from staffNo to staffName the same is not true in the opposite direction – staffName does not functionally determine staffNo.

Staying with the personnel information, staffNo will probably functionally determine departmentName. For every member of staff there is only one associated department or school name that applies. A member of staff cannot belong to more than one department or school at any one time.

18.3.2 UNNORMALISED DATA-SET TO FIRST NORMAL FORM

The data-set represented in tabular form in section 18.2 is said to be an unnorma-lised data-set. This can be seen, for instance, if we choose the data-item moduleName as the key of this data-set and underline it to indicate this. Realistically, if we remove redundant information, we should represent the information as follows:

Unnormalised data set

Modules

<u>moduleName</u>	staffNo	staffName	student No	student Name	ass Grade	ass Type
Relational Database Systems	234	Davies T	34698	Smith S	B3	cwk1
			37798	Jones S	B1	cwk2
			34888	Patel P	B2	cwk1
					B1	cwk1
Relational Database Design	234	Davies T	34698	Smith S	B3	cwk2
					B2	cwk1
Deductive Databases	345	Evans R	34668	Smith J	A1	exam

This does not logically represent a relation as defined in Chapter 7. A given cell of the table for the attributes studentNo, studentName, assGrade and assType contains multiple values. Examining the table above we see that studentNo, studentName, assGrade and assType all repeat with respect to moduleName.

A relation is in first normal form if and only if every non-key attribute is functionally dependent upon the primary key

The attributes studentNo, studentName, assGrade and assType are clearly not functionally dependent on our chosen primary key moduleName. The attributes staffNo and staffName clearly are. This means that we form two tables: one for the functionally dependent attributes, and one for the non-dependent attributes. We declare a compound of moduleName, studentNo and assType to be the primary key of this second table.

First normal form tables

Modules				
<u>moduleName</u>	<u>staffNo</u>	<u>staffName</u>		
Relational Database Systems	234	Davies T		
Relational Database Design	234	Davies T		
Deductive Databases	345	Evans R		

Assessments				
<u>moduleName</u>	<u>studentNo</u>	<u>ass Type</u>	<u>student Name</u>	<u>ass Grade</u>
Relational Database Systems	34698	cwk1	Smith S	B3
Relational Database Systems	34698	cwk2	Smith S	B1
Relational Database Systems	37798	cwk1	Jones S	B2
Relational Database Systems	34888	cwk1	Patel P	B1
Relational Database Systems	34888	cwk2	Patel P	B3
Relational Database Design	34698	cwk1	Smith S	B2
Relational Database Design	34698	cwk2	Smith S	B3
Deductive Databases	34668	exam	Smith J	A1

18.3.3 FIRST NORMAL FORM TO SECOND NORMAL FORM

To move from first normal form to second normal form we remove part-key dependencies. This involves examining those tables that have a compound key and for each non-key data-item in the table asking the question: can the data-item be uniquely identified by part of the compound key?

A relation is in second normal form if and only if it is in first normal form and every non-key attribute is fully functionally dependent on the primary key



Take, for instance, the table named Assessments. Here we have a three-part compound key moduleName, studentNo and assType. We ask the question above for each of these data-items in relation to the non-key data-items studentName and assGrade. Clearly we need all the items of the key to tell us what the assessment grade is. ModuleName however has no influence on the studentName. StudentNo alone determines studentName. Hence, we break out the determinant and dependent data-items into their own table. This leads to a decomposition of the tables as follows:

Second normal form tables

Modules			
<i>moduleName</i>	<i>staffNo</i>	<i>staffName</i>	
Relational Database Systems	234	Davies T	
Relational Database Design	234	Davies T	
Deductive Databases	345	Evans R	

Assessments			
<i>moduleName</i>	<i>studentNo</i>	<i>ass Type</i>	<i>ass Grade</i>
Relational Database Systems	34698	cwk1	B3
Relational Database Systems	34698	cwk2	B1
Relational Database Systems	37798	cwk1	B2
Relational Database Systems	34888	cwk1	B1
Relational Database Systems	34888	cwk2	B3
Relational Database Design	34698	cwk1	B2
Relational Database Design	34698	cwk2	B3
Deductive Databases	34668	exam	A1

Students	
<i>StudentNo</i>	<i>Student Name</i>
34698	Smith S
37798	Jones S
34888	Patel P
34668	Smith J

18.3.4 SECOND NORMAL FORM TO THIRD NORMAL FORM

To move from second normal form to third normal form we remove inter-data dependencies. To do this we examine every table and ask of each pair of non-key

data-items: is the value of data-item A dependent on the value of data-item B, or vice versa? If the answer is yes we split off the relevant data-items into a separate table.

A relation is in third normal form if and only if it is in second normal form and every non-key attribute is non-transitively dependent on the primary key

The only place where this is relevant in our present example is in the table called Modules. Here, staffNo determines staffName. StaffName is hence transitively dependent on moduleName. StaffNo is therefore asking to be a primary key. Hence, we create a separate table to be called Lecturers with staffNo as the primary key. This is illustrated below:

Third normal form tables

Modules	
<u>moduleName</u>	<u>staffNo</u>
Relational Database Systems	234
Relational Database Design	234
Deductive Databases	345

Lecturers	
<u>staffNo</u>	<u>staffName</u>
234	Davies T
345	Evans R

Assessments			
<u>moduleName</u>	<u>studentNo</u>	<u>ass Type</u>	<u>ass Grade</u>
Relational Database Systems	34698	cwk1	B3
Relational Database Systems	34698	cwk2	B1
Relational Database Systems	37798	cwk1	B2
Relational Database Systems	34888	cwk1	B1
Relational Database Systems	34888	cwk2	B3
Relational Database Design	34698	cwk1	B2
Relational Database Design	34698	cwk2	B3
Deductive Databases	34668	exam	A1



Students	
<i>studentNo</i>	<i>student Name</i>
34698	Smith S
37798	Jones S
34888	Patel P
34668	Smith J

18.4 THE NORMALISATION OATH

A useful mnemonic for remembering the rationale for normalisation is the distortion of the legal oath presented below:

1. No Repeating,
2. The Data-Items Depend Upon The Key,
3. The Whole Key,
4. And Nothing But The Key,
5. So Help Me Codd.

Line 5 simply reminds us that the techniques were originally developed by E.F. Codd in the 1970s. Line 2 states that all data-items in a table must depend solely upon the key. Line 1 indicates that there should be no repeating groups of data in a table. Line 3 indicates that there should be no part-key dependencies in a table. Finally, line 4 reminds us that there should be no inter-data dependencies in a table. The only dependency should be between the key and other data-items in a table.

18.5 DETERMINANCY DIAGRAMMING

Classic normalisation is described as a process of non-loss decomposition. The decomposition approach starts with one (universal) relation – the unnormailised data-set. File maintenance anomalies such as insertion, deletion and update anomalies are gradually removed by a series of projections. Non-loss decomposition is therefore a design process guaranteed to produce a data-set free from file maintenance problems. It does however suffer from a number of disadvantages, particularly as a practicable database design technique. Firstly, it requires all of the data-set to be in place before the process can begin. Secondly, for any reasonably large data-set the process is extremely time-consuming, difficult to apply and prone to human error.

The remaining part of this chapter will therefore concentrate on describing a contrasting approach to normalisation which uses a graphical notation. The

main advantage of the determinancy diagramming technique is that it provides a mechanism for designing a database incrementally. One does not need a complete data-set in hand to begin the process of design. The data analyst can begin work with a small collection of central data-items. Around this core, data-items can be continuously added until all the dependencies are fully documented.

Once documented, the determinancy diagram can be transformed into a relational schema in a couple of easy steps.

18.5.1 DETERMINANCY DIAGRAMS

We shall refer to a diagram which documents the determinancy or dependency between data-items as a determinancy or dependency diagram. Data-items are drawn on a determinancy diagram as labelled ovals, circles or bubbles. Functional dependency is represented between two data-items by drawing a single-headed arrow from the determinant data-item to the dependent data-item. For example, Figure 18.1A represents a number of functional dependencies as diagrams.

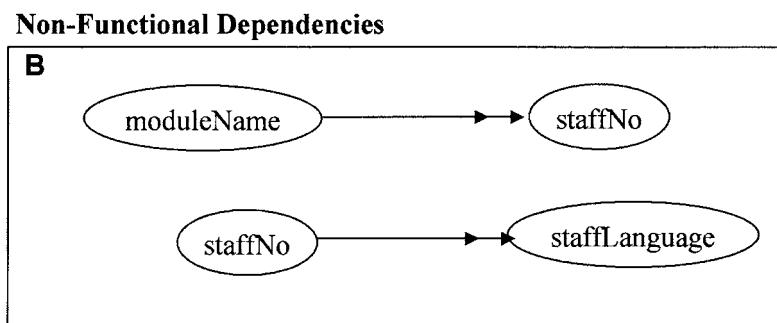
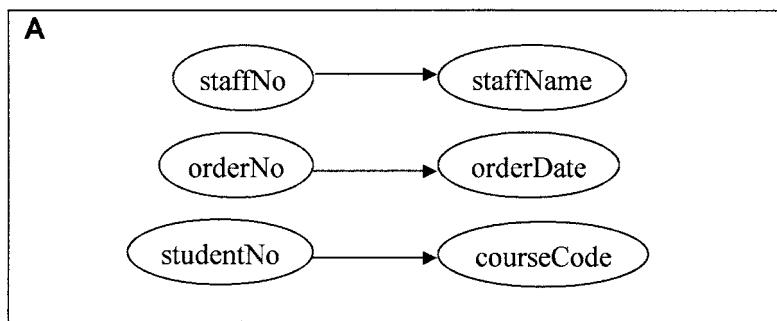


Figure 18.1 Functional and non-functional dependencies.

18.5.2 NON-FUNCTIONAL (MULTI-VALUED) DETERMINANCY

Not all dependencies can be modelled in terms of functions. It is for this reason that we need to introduce the idea of a non-functional dependency. Data-item



B is said to be non-functionally dependent on data-item A if for every value of data-item A there is a delimited set of values for data-item B. The mapping is no longer functional because it is one-to-many.

Example

Let us assume that the university maintains a list of languages relevant to the organisation. The university wishes to record which members of staff have which language skills. Clearly the relationship between staffNos and languages is not a functional determinancy. Many staff members may just have one language, but some will have two or more languages. Also, each language, particularly in the case of European languages such as English, French and German, is likely to be spoken by more than one staff member.

Therefore staffNo and staffLanguage are in a non-functional or multi-valued determinancy. In other words, for every staffNo we can identify a delimited set of language codes which apply to that staff member.

Multi-valued or non-functional dependency is indicated by drawing a double-headed arrow from the determinant to the dependent data-item. Figure 18.1B represents two non-functional relationships as determinancy diagrams.

18.6

PRAGMATICS OF DETERMINANCY DIAGRAMMING

Dependencies between any two data-items may be diagrammed as A to B or B to A, but not both. Frequently, we may find that what is a single-valued dependency in one direction is a multi-valued dependency in the opposite direction. For example, take staffNo and departmentName. In the direction staffNo to departmentName it is a functional dependency. In the direction departmentName to staffNo it is a multi-valued dependency. In such situations we always choose the direction of the functional dependency. This makes the eventual relational schema a lot simpler. It reduces, as we shall see, the number of compound keys required.

If, however, a functional or non-functional dependency exists in both directions then we choose either.

Example

staffNo might functionally determine extensionNo and extensionNo in turn functionally determines staffNo. We hence may choose either staffNo or extensionNo as our determinant. Our choice is however likely to be influenced by how many other dependencies arise from the data-item. staffNo is more likely to do more work for us in this context. More data-items are likely to be dependent upon staffNo than upon extensionNo.

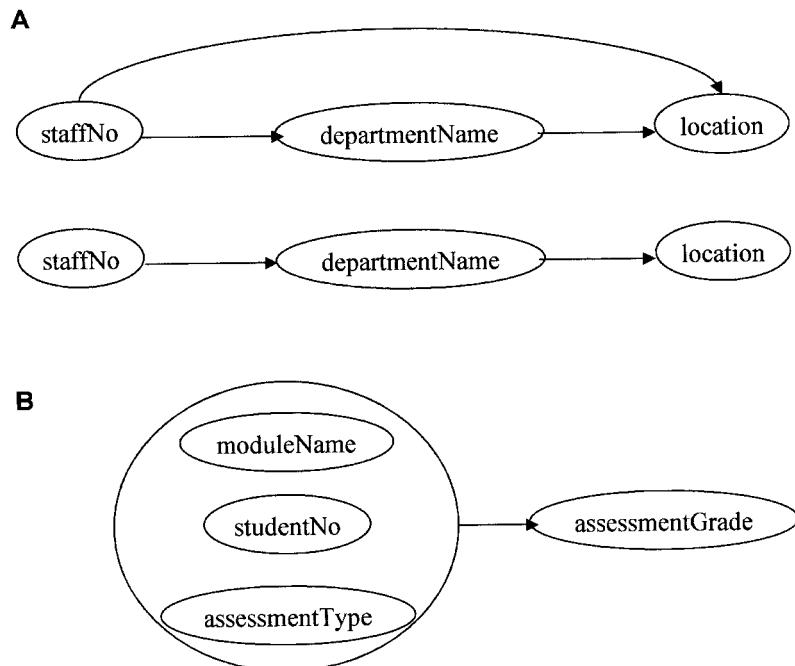


Figure 18.2 Transitive and compound determinancy.

18.6.1 COMPOUND AND TRANSITIVE DETERMINANCY

Figure 18.2A documents a transitive dependency. A functional dependency exists from staffNo to departmentName, from departmentName to location, and from staffNo to location. Any situation in which A determines B, B determines C and A also determines C can usually be simplified into the chain A to B and B to C. Identifying these so-called transitive determinancies can frequently simplify complex determinancy diagrams and indeed is an important part of the process of normalisation (section 18.3.4).

Frequently, one data-item is insufficient to fully determine the values of some other data-item. However, the combination of two or more data-items gives us a dependent relationship. In such situations we call the group of determinants a compound determinant. A compound determinant is drawn as an enclosing bubble around two or more data-item bubbles. Hence, in Figure 18.2B we need studentNo, moduleName and assessmentType to functionally determine assessmentGrade. The functional dependency is drawn from the outermost bubble.

18.6.2 MINIMAL SET OF FUNCTIONAL DEPENDENCIES

In practice, even if we restrict ourselves to specifying the functional dependencies between data-items, the complete set of such dependencies can be very



large. In the discussion above we introduced some heuristics or rules of thumb which help us prune some dependencies because they are trivial in the sense that they can be inferred from other dependencies – transitive dependencies are a case in point. The objective is to determine the minimal set of dependencies for a relation from which the complete set of dependencies can be inferred.

It can be shown that a set of rules known as Armstrong's axioms can be used to determine new functional dependencies from existing dependencies. Hence, these rules can be used to compute the so-called closure of X, where X represents a set of functional dependencies and the closure of X represents the complete set of functional dependencies implied by X. These rules are included below, where the symbol ' \Rightarrow ' represents the implies connective and A, B and C represent attributes of some relation (Chapter 7):

- *Reflexivity.* If B is a subset of A, then $A \Rightarrow B$
- *Augmentation.* If $A \Rightarrow B$ then $A,C \Rightarrow B,C$
- *Transitivity.* If $A \Rightarrow B$ and $B \Rightarrow C$ then $A \Rightarrow C$

Four other rules can be derived from Armstrong's axioms which simplify the task of computing the complete set of functional dependencies:

- *Self-determination.* $A \Rightarrow A$
- *Decomposition.* If $A \Rightarrow B,C$ then $A \Rightarrow B$ and $A \Rightarrow C$
- *Union.* If $A \Rightarrow B$ and $A \Rightarrow C$ then $A \Rightarrow B,C$
- *Composition.* If $A \Rightarrow B$ and $C \Rightarrow D$ then $A,C \Rightarrow B,D$

We apply these rules systematically to a set of dependencies to determine its closure. They can also be used to determine a minimal set of dependencies from a large set of dependencies for a relation. These rules can also be implemented in deductive systems such as Datalog and used to automatically produce a logical design for a relational database from a sample data-set. This idea forms the basis for facilities available in some CAISE tools for database design (Chapter 26).

18.7

ACCOMMODATION

In this section we examine the process of transforming a determinancy diagram into a set of table structures or relational schema – a process frequently known as accommodation.

18.7.1 ACCOMMODATING FUNCTIONAL DEPENDENCIES

Suppose we are given the determinancy diagram in Figure 18.3A. We transform this diagram into a set of table structures by applying the rule:

Every functional determinant becomes the primary key of a table. All immediate dependent data-items become non-key attributes of the table

This is frequently referred to as the Boyce–Codd rule, after its inventors.

18.7.2 DRAWING BOUNDARIES

A useful intermediate step to take in the accommodation process is to draw boundaries around data-items of a determinancy diagram that form elements of table structures. The number of determinants (ovals with arrows emerging) should indicate the number of tables required. We have two determinants in Figure 18.3A, hence we need two tables. We also include within a boundary all the immediate dependent data-items (ovals with arrows converging). We hence draw two boundaries for our example as illustrated in Figure 18.3B. Note that foreign keys are easily identified on determinancy diagrams. They are usually ovals with arrows both emerging and converging.

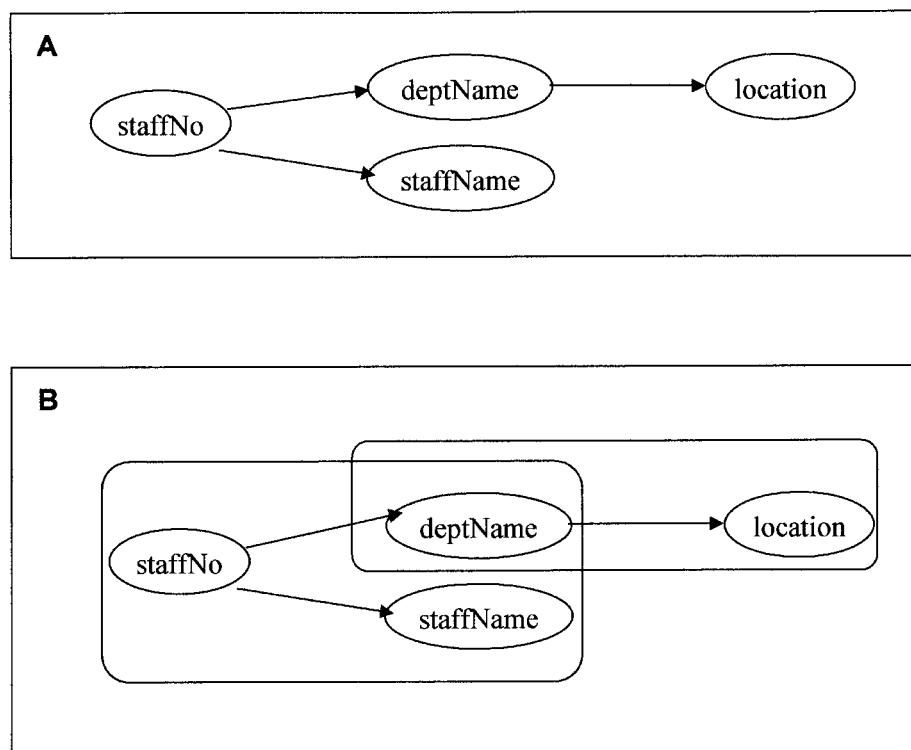


Figure 18.3 A simple determinancy diagram.

Applying the functional dependency rules to the diagram in Figure 18.3A gives us the following table structures in the bracketing notation:



`Staff(staffNo, staffName, deptName ...)`
`Departments(deptName, location ...)`

18.7.3 CANDIDATE KEYS

A determinacy diagram is useful in identifying candidate keys. A candidate key is any data-item that can act in the capacity of a primary key for a table. In determinacy diagramming terms, candidate keys are represented by competing determinants. Consider the diagram in Figure 18.4A. Here both staffNo and national insurance number (NINo) determine staffName independently. In this case, staffNo and NINo are candidate keys. We choose one of these to be the actual key of the table and make the other a dependent data-item in the table.

The Boyce–Codd rule described in section 18.7.1 should actually be phrased as *every functional determinant becomes a candidate key for a relation*. In other words, in the face of a number of candidate keys for a relation we choose one of these to be the primary key.

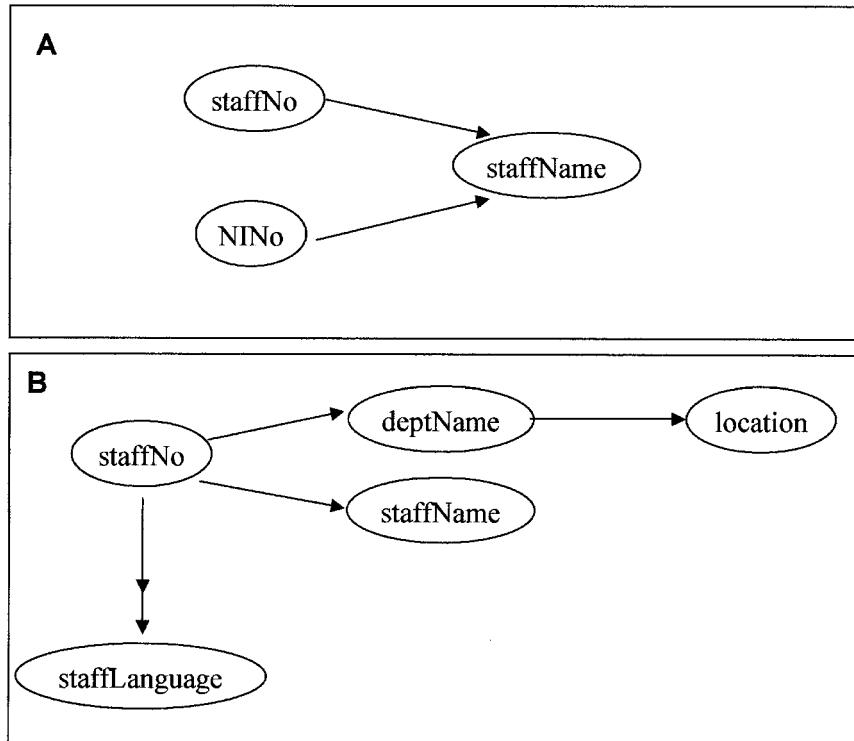


Figure 18.4 Non-functional dependencies and candidate keys.

18.7.4 ACCOMMODATING NON-FUNCTIONAL DEPENDENCIES

We accommodate non-functional dependencies by applying the following rule:

Every non-functional determinant becomes part of the primary key of a table

That is, we make up a compound key from the determinant and dependent data-items in a non-functional relationship.

Example ➔ Suppose we add the data-item staffLanguage to our data-set in section 18.7.2 (see Figure 18.4B). We now need two tables to represent the functional dependencies as above. We also need a table to record the multi-valued dependency between staffNo and staffLanguage. This gives us three table-structures as below:

```
Staff(staffNo, staffName, departName, ...)  
Departments(deptName, location, ...)  
Languages(staffNo, staffLanguage, ...)
```

18.8 DETERMINANCY DIAGRAMMING AND THE NORMAL FORMS

In this section we shall connect up the discussion on non-loss decomposition with our discussion of determinancy diagramming. The determinancy diagram for the data-set decomposed in the early section of this chapter is illustrated in Figure 18.5.

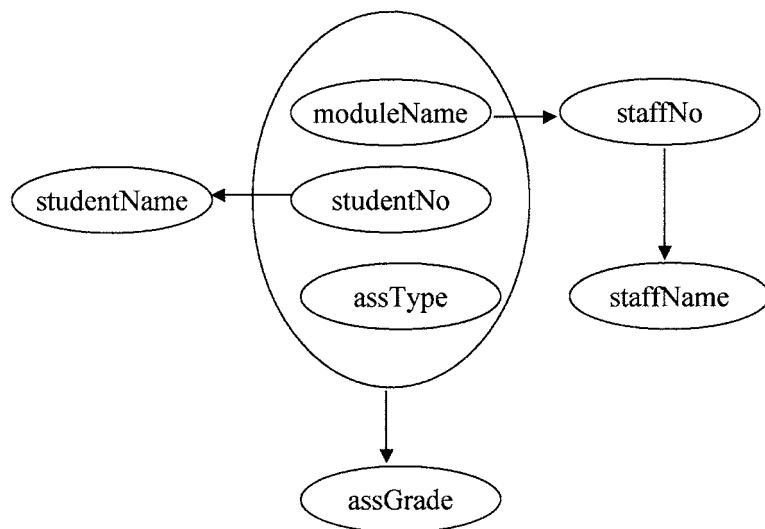


Figure 18.5 Determinancy diagram for the academic schema.

First, second, third normal forms and BCNF are all about functional dependencies. Fourth and fifth normal forms are about non-functional dependencies.



First normal form concerns repeating groups. Since functional dependencies document one-to-many relationships between data they are an explicit representation of repeating groups. Second normal form concerns part-key dependencies. Here we are segmenting functional dependencies out of a compound key. Third normal form concerns inter-data dependencies. Here we are identifying determinants within the non-key attributes of a table.

18.9



BCNF

BCNF is a stronger normal form than third normal form, designed to cover anomalies that arise when there is more than one candidate key in some set of data requirements.

Example

Suppose we have introduced a scheme of majors and minors into our degree schemes at a university. The business rules relevant to that part of this domain covering majors are listed below:

- Each student may major in several areas
- A student has one tutor for each area
- Each area has several tutors but a tutor advises in only one area
- Each tutor advises several students in an area

A diagram incorporating all these business rules is given in Figure 18.6.

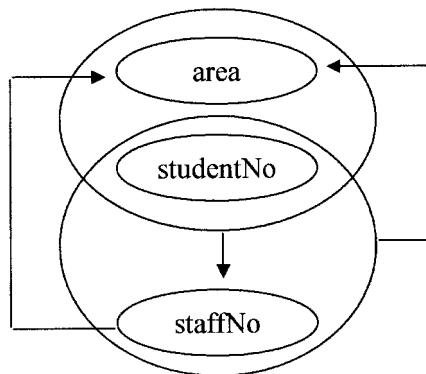


Figure 18.6 Determinacy diagram for BCNF problem.

On the basis of these business rules, a schema is produced in third normal form, represented in the bracketing notation below:

Majors(studentNo, area, staffNo)

A set of sample data is provided in the table below.

Majors

StudentNo	Area	StaffNo
123456	Computer Science	234
234567	Information Systems	345
123456	Software Engineering	456
234567	Graphic Design	678
345678	Information Systems	567

This schema is in third normal form because there are no partial dependencies and no inter-data dependencies. However, anomalies will still arise when we come to update this relation. For instance:

- Suppose student 123456 changes one of his or her majors from computer science to information systems. Doing this means that we lose information about staffNo 234 tutoring on computer science. This is an update anomaly
- Suppose we wish to insert a new row to establish the fact that staffNo 789 tutors on computer science. We cannot do this until at least one student takes this area as his or her major. This is an insertion anomaly
- Suppose student 345678 withdraws from the university. In removing the relevant row we lose information about staffNo 567 being a tutor in the area of information systems. This is a deletion anomaly

These anomalies occur because there are two overlapping candidate keys in this problem. R.F. Boyce and E.F. Codd identified this problem and proposed a solution in terms of a stronger normal form known as BCNF. A relation is in BCNF if every determinant is a candidate key. The schema above can be converted into BCNF in one of two ways. The two schemas are presented in bracketing notation below:

Schema 1:
StudentTutors(studentNo, staffNo)
TutorAreas(staffNo, area)

Schema 2:
StudentTutors(studentNo, area)
TutorAreas(staffNo, area)

18.10**FOURTH NORMAL FORM**

To move from BCNF to fourth normal form we look for tables that contain two or more independent multi-valued dependencies. Multi-valued dependencies are fortunately scarcer than part-key or inter-data dependencies. Our university problem discussed in section 18.2 does not contain any such dependencies. We therefore examine here an example adapted from Kent (1983).



Suppose we wish to design a personnel database for the Commission of the European Community, which stores information about an employee's skills and the languages an employee speaks. An employee is likely to have several skills (eg. typing, word-processor operation, spreadsheet operation), and most employees are required to speak at least two European languages. Our first attempt at a design for this system might aggregate all the data together in one table as below:

EUEmployees		
employee	No	skilllanguage
0122443	Typing	English
0122443	Typing	French
0122443	Dictation	English
0221133	Typing	German
0221133	Dictation	French
0332222	Typing	French
0332222	Typing	English

However, we wish to add the restriction that each employee exercises skill and language use independently. In other words, typing as a skill is not inherently linked with the ability to speak a particular language.

Under fourth normal form these two relationships should not be represented in a single table as in Figure 18.7A. This is evident when we draw the determinancy diagram as in Figure 18.7B. Having two independent multi-valued dependencies means that we must split the table into two as below:

EUSkills	
employeeNo	skill
0122443	Typing
0122443	Dictation
0221133	Typing
0221133	Dictation
0332222	Typing

EULanguages	
employeeNo	language
0122443	English
0122443	French
0221133	German
0221133	French
0332222	French

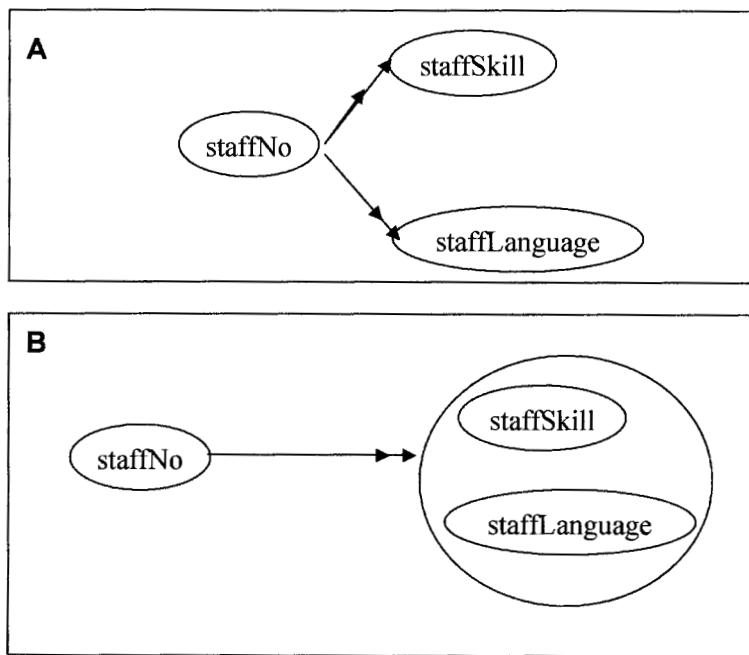


Figure 18.7 Fourth normal form.

18.11 (c) FIFTH NORMAL FORM

Generally speaking, a fourth normal form table is in fifth normal form if it cannot be non-loss decomposed into a series of smaller tables. Consider the table below which stores information about automobile agents, automobile companies and automobiles:

Outlets		
agent	company	automobile
Jones	Ford	Car
Jones	Vauxhall	Van
Smith	Ford	Van
Smith	Vauxhall	Car

If agents represent companies, companies make products, and agents sell products, then we might want to record which agent sells which product for which company (Figure 18.8). To do this we need the structure above. We cannot decompose the structure because although agent Jones sells cars made by Ford and vans made by Vauxhall, he does not sell Ford vans or Vauxhall cars. Fifth

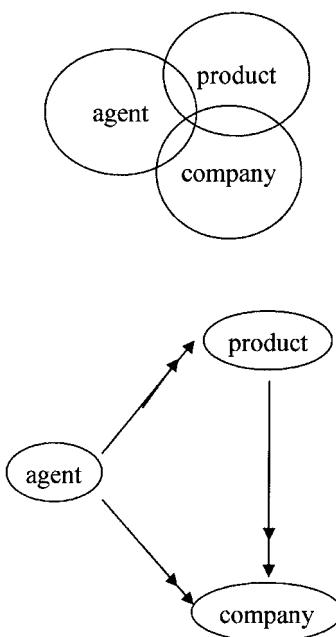


Figure 18.8 Fifth normal form.

normal form concerns interdependent multi-valued dependencies, otherwise known as join dependencies.

18.12

CASE STUDY: TIMETABLING AT A UNIVERSITY

Assume we wish to extend our database to include data pertaining to academic timetabling of staff and modules into rooms within a university. Examining the current documentation available on timetables has provided us with the following fully normalised data-set:

Timetable

Module Name	Date	Time	StaffNo	Room
Relational Database Systems	12/12/2003	15:00	234	J206
Relational Database Systems	19/12/2003	15:00	234	J206
Relational Database Systems	19/12/2003	12:00	234	J206
Relational Database Design	12/12/2003	15:00	345	J207
Relational Database Design	19/12/2003	13:00	345	J206
Relational Database Design	12/12/2003	15:00	345	J207
Object-Oriented Database Systems	20/12/2003	12:00	234	J208
Object-Oriented Database Systems	20/12/2003	12:00	234	J206

A third normal form database for this data-set can be represented in the bracketing notation as follows:

Timetable(date, time, room, staffNo, moduleName)

This relation is in third normal form because staffNo and moduleName are functionally dependent on the compound primary key – date, time and room. The attributes staffNo and moduleName are also dependent on the whole of this key, and staffNo and moduleName are not determinants in themselves.

However, this database is not in BCNF because in BCNF every determinant in a relation is a candidate key. From the dependency diagram in Figure 18.9 we see that the group of data-items – date, time and staffno – are not represented in this structure.

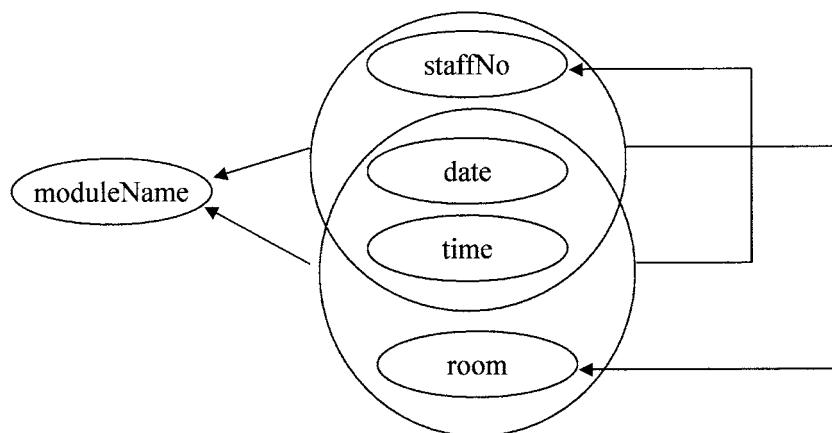


Figure 18.9 Determinacy diagram of the timetable.

To produce a BCNF version of this structure we need to create two relations as below:

TeachingEvents(date, time, room, moduleName)

StaffEvents(date, time, staffNo, room)

18.13 SUMMARY

- Normalisation is the process of producing a schema not subject to file maintenance anomalies
- We have discussed two complementary approaches to bottom-up data analysis: non-loss decomposition and determinancy diagramming
- Non-loss decomposition is a step-by-step approach to producing a fully normalised schema



- Determinancy diagramming is a graphic approach reliant on an accommodation stage to produce a logical relational schema
- Normalisation is normally used as a means of validating the results of critical areas of top-down data analysis

18.14
ACTIVITIES

1. Draw a determinancy diagram for each of the following types of marriage: monogamy — one man marries one woman; polygamy — one man can marry many women, but one woman marries only one man; polyandry — one woman can marry many men, but one man marries only one woman; group marriage — one man can marry many women, one woman can marry many men.
2. Accommodate each dependency diagram in activity 1 above into a relational schema.
3. Produce third normal form table-structures from the table below:

Cinemas									
Film No.	Film Name	Cinema Code	Cinema Name	Town	Population	Man. No.	Man. Name	Film Takings	
25	Star Wars	BX	Rex	Cardiff	300000	01	Jones	900	
25	Star Wars	KT	Rialto	Swansea	200000	03	Thomas	350	
25	Star Wars	DJ	Odeon	Newport	250000	01	Jones	800	
50	Jaws	BX	Rex	Cardiff	300000	01	Jones	1200	
50	Jaws	DJ	Odeon	Newport	250000	01	Jones	400	
50	Jaws	TL	Rex	Bridgend	150000	02	Davies	300	
50	Jaws	RP	Grand	Bristol	350000	04	Smith	1500	
50	Jaws	HF	State	Bristol	350000	04	Smith	1000	
30	Star Trek	BX	Rex	Cardiff	300000	01	Jones	850	
30	Star Trek	TL	Rex	Bridgend	150000	02	Davies	500	
40	ET	KT	Rialto	Swansea	200000	03	Davies	1200	
40	ET	RP	Grand	Bristol	350000	04	Smith	2000	

4. Take any receipt produced by a major supermarket chain and conduct a normalisation exercise.
5. Take any electricity or gas bill and conduct a normalisation exercise on the data represented in the bill.
6. Generate a set of fully normalised tables from the following unnormalised table:

Operating		Schedule					
Doctor No.	Doctor Name	OpNo.	opDate	opTime	Patient No.	Patient Name	Admission Date
18654	Smith	AA1234	04/02/1999	08:30	2468	Davies M.	20/01/1999
18654	Smith	BA1598	04/02/1999	10:30	3542	Jones D.	11/01/1999
18654	Smith	FG1965	04/02/1999	16:00	1287	Evans I.	25/12/1999
18654	Smith	AA1235	13/02/1999	14:00	2468	Davies M.	20/01/1999
13855	Evans	LP1564	13/02/1999	14:00	4443	Beynon P.	05/01/1999
18592	Jones	PP900	15/02/1999	14:00	2222	Scott I.	04/01/1999
18592	Jones	BA1598	04/02/1999	10:30	3542	Jones D.	11/01/1999
18592	Jones	FG1965	04/02/1999	16:00	1287	Evans I.	25/12/1999

18.15 REFERENCES

- Codd, E.F. (1970). A relational model for large shared data banks. *Comm. of ACM* **13**(1): 377–87.
- Codd, E.F. (1974). Recent investigations into relational database systems. *Proc. IFIP Congress*.
- Fagin, R. (1977). Multi-valued dependencies and a new normal form for relational databases. *ACM Trans. on Database Systems* **2**(1).
- Fagin, R. (1979). Normal forms and relational database operators. *ACM SIGMOD Int. Symposium on the Management of Data* 153–60.
- Kent, W. (1983). A simple guide to five normal forms in relational database theory. *Comm. of ACM* **26**(2).



CHAPTER 19 →

PHYSICAL DATABASE DESIGN

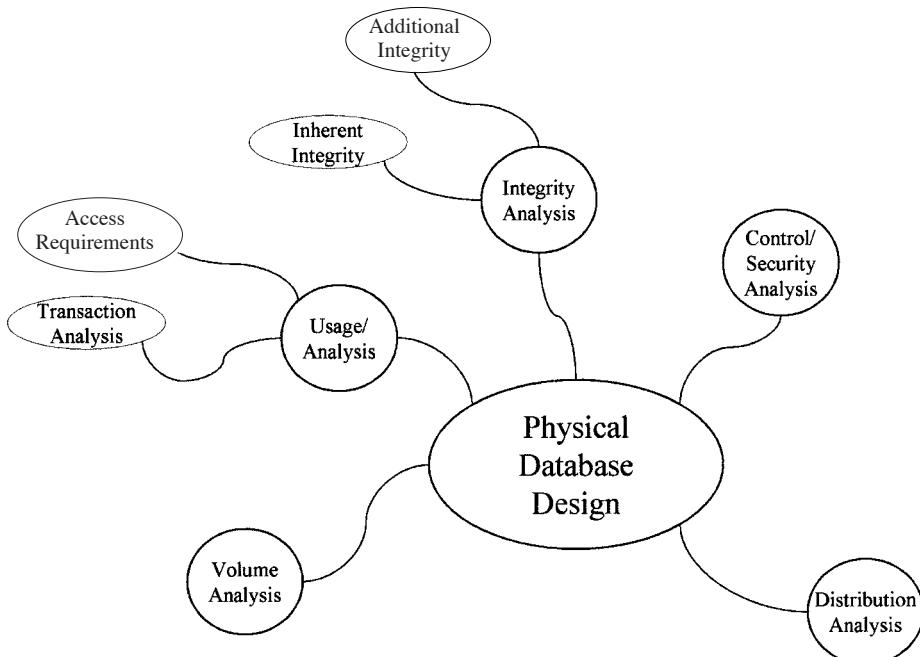
See first that the design is wise and just: that ascertained, pursue it resolutely; do not for one repulse forego the purpose that you resolved to effect.

William Shakespeare (1564–1616)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Distinguish the process of physical database design from that of logical database design
- Indicate the alignment between physical design and models of the IS development process
- Describe the major activities of physical database design



19.1 INTRODUCTION

Logical database design is the process of constructing a business data model, that is, a model of the business data requirements applying to some organisation or part of it. Physical database design involves taking the results from the logical design process, fine-tuning them against the usage, performance and storage requirements of some application and then making certain implementation decisions in terms of a chosen DBMS.

With the rise of the relational data model the emphasis has changed in the database literature. Prior to the relational data model the emphasis in texts on database design tended to be on physical database design. Nowadays, the emphasis is clearly directed at logical database design. This is primarily because some success has been achieved in formalising logical design. Physical design, in comparison, is relatively unformalised. This is because the natures of the two exercises are clearly different. Logical database design is about implementation independence. This makes a general analysis of the area and the establishment of some semi-formal approaches feasible. Physical database design, in contrast, is about implementation dependence. You cannot conduct a physical database design exercise effectively unless you know in some detail how your application is meant to work, on what hardware and operating system the application is to run, and what facilities are provided by your chosen DBMS. Given these constraints, fewer generalities are possible.

Nevertheless, physical database design is still of extreme importance to designers and indeed users of database systems. This chapter therefore provides an overview of some of the most important aspects of the physical design of databases. We concentrate on the physical design of post-relational databases, although similar principles apply to classic, object-oriented or even deductive databases. The next chapter considers how we utilise the information generated in physical database design to implement a database system.

The main output from the logical database design process is the logical data model. In most contemporary database design projects this model will be expressed as a relational schema. Therefore, in this chapter we first review the process of logical database design using yet another example of a distinct universe of discourse.

19.2 UNIVERSITY SHORT COURSES

In this section we provide a short Case Study to illustrate the process of realistic database development. We shall concentrate here on the development of a relational database system.

Example A university decides to offer a series of short courses to industry and commerce. It is particularly interested in offering courses in information technology of some three to four days' duration. It decides to set up a commercial arm, USC (University Short Courses), to develop, market and administer such courses.



Each short course is developed and maintained by one member of university staff, known as the course manager. A course may however be presented by a number of different lecturers, besides the course manager, depending on the popularity of the course.

USC holds course presentations both at a specially prepared site on the university campus and at commercial and industrial sites. The former are described as being scheduled presentations, while the latter are described as being on-site presentations.

Students on scheduled courses are likely to come from a number of different organisations. Students on on-site courses obviously come from one organisation. A number of companies use USC courses as part of their in-house training schedule.

USC wants to build a database system to help it to perform the administration of short courses more efficiently. Besides conventional insertion, update and deletion of data, a range of other functions must be performed by the intended system:

- When a person telephones to register for a particular session, USC staff need to check the number of persons already registered. The number of people registered for a presentation should not be greater than the course limit
- At any time prior to a particular presentation USC staff want to query the database to find out how many people have registered for the presentation
- On a regular basis the system needs to notify USC staff of those presentations that have fewer than four students registered for them. Four students is a break-even point for costing a presentation. If fewer than four persons register, then the presentation will be postponed and run at a later date
- Two weeks before a presentation, staff needs to ring each student to confirm attendance
- One week before each presentation, staff needs to check that the presentation fee has been paid. An attendance list then needs to be printed for each presentation
- USC needs to keep track of which lecturers are qualified to teach which courses. Lecturers have to be assigned to scheduled courses at least six months ahead of a presentation
- The manager of USC wants to produce statistics on the average number of students per course, to give him some idea of the popularity of each course. He is also particularly interested in finding out whether courses sell better at particular times of the year
- USC also wants to maintain an up-to-date mailing list so that it can send regular brochures of new and updated courses to potential customers
- The university is establishing links with sister universities in Europe, the USA and the Far East. USC wants to design for the possibility of a distributed database capacity within a year or so

19.3 LOGICAL DATABASE DESIGN

The first task is obviously to develop a data model for the intended database system. This process is illustrated in Figure 19.1.

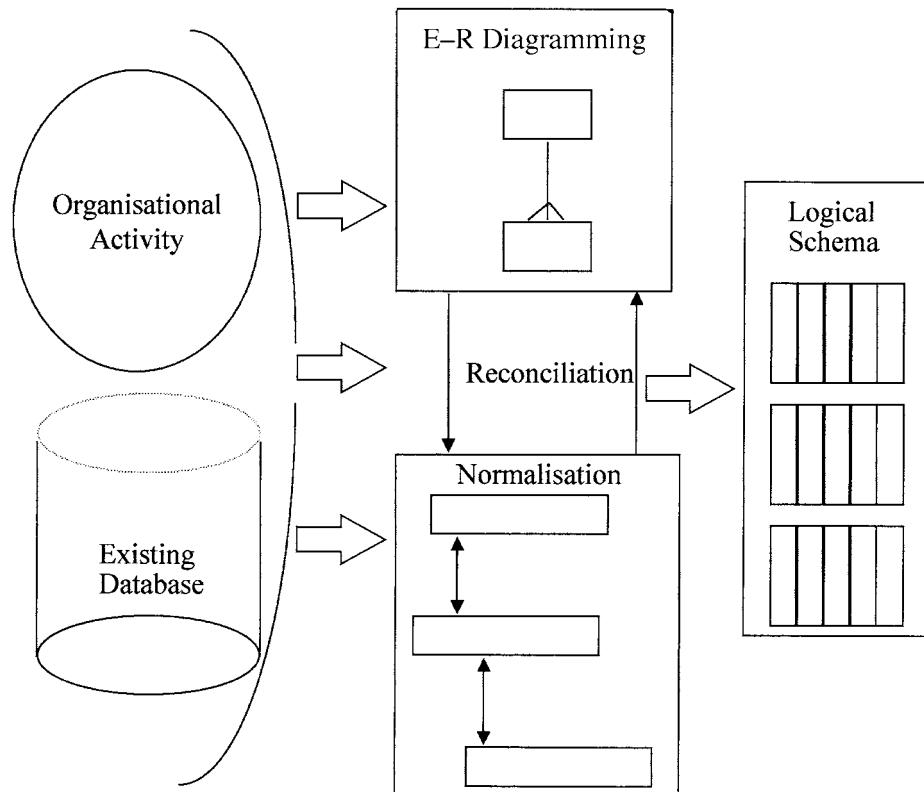


Figure 19.1 Reconciliation of logical and conceptual modelling.

19.3.1 THE LOGICAL DATABASE DESIGN PROCESS

The first element of this figure indicates that logical database design will normally involve both eliciting and documenting new requirements for a database application, and incorporating a range of existing data requirements perhaps gleaned from existing ICT applications. The second element of the figure indicates that two complementary approaches can be taken to logical database design. In a conventional database project the data analysts will conduct a great deal of data modelling. For critical aspects of an application the analysts will also conduct some normalisation. The results of these two approaches may have to be reconciled. The output of this reconciliation process will be a logical schema.



Example → Figure 19.2 illustrates the main entities and relationships derived from the business rules of University Short Courses. A logical schema produced from this diagram is expressed in the bracketing notation and the more detailed notation of Chapter 7.

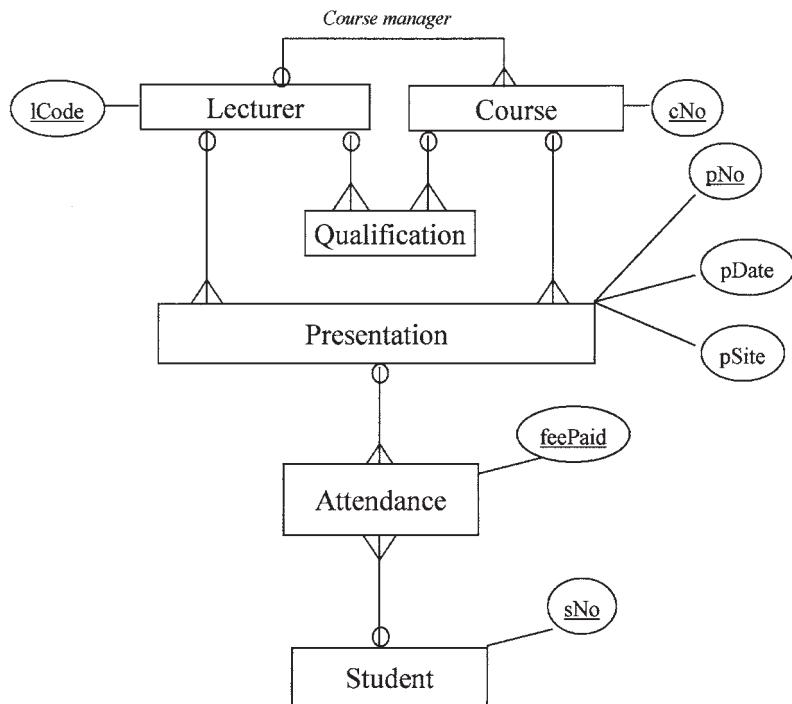


Figure 19.2 USC entity model.

Courses(courseNo, courseName, courseManager, courseDuration)
 Lecturers(lecturerCode, lecturerName, IHomeAddress, IWorkA
 IHomeTelNo, IWorkTelNo)
 Students(studentNo, studentName, studentAddress, studentTelNo)
 Presentations(pNo, courseNo, pDate, pSite, lecturerCode)
 Attendance(studentNo, pNo, feePaid)
 Qualifications(lecturerCode, courseNo)

Domains

dates: julianDates
 lecturerCodes: integer(3)
 personNames: character(30)
 studentNos: integer(6)
 Addresses: character(50)
 TelephoneNumbers: character(17) format <(99999) (99999999)>
 courseNos: integer(3)
 courseNames: character(30)

```

courseDuration: integer(1) {1,2,3,4}
sites: character(10) {university, hotel, on-site}
feePaid: character(1) {Y,N}
presentationNos: integer(5)

```

Relation Courses**Attributes**

```

courseNo: courseNos
courseName: courseNames
courseManager: lecturerCodes
courseDuration: courseDurations

```

Primary Key courseNo**Foreign Key** courseManager **References** Lecturers **Not Null****Delete Restricted****Update Cascades****Relation Lecturers****Attributes**

```

lecturerCode: lecturerCodes
lecturerName: personNames
lHomeAddress: addresses
lWorkAddress: addresses
lHomeTelNo: telephoneNumbers
lWorkTelNo: telephoneNumbers

```

Primary Key lecturerCode**Relation Students****Attributes**

```

studentNo: studentNos
studentName: personNames
studentAddress: addresses
studentTelNo: telephoneNumbers

```

Primary Key studentNo**Relation Presentations****Attributes**

```

presentationNo: presentationNos
pSite: sites
courseNo: courseNos
pDate: dates
lecturerCode: lecturerCodes
lHomeTelNo: telephoneNumbers
lWorkTelNo: telephoneNumbers

```

Primary Key lecturerCode**Foreign Key** courseNo **References** Courses **Not Null****Delete Nullifies****Update Cascades****Foreign Key** lecturerCode **References** Lecturers

**Delete Restricted**
Update Cascades**Relation** Attendance**Attributes**

studentNo: studentNos

pNo: pNos

feePaid: feePaids

Primary Key studentNo, pNo**Foreign Key** studentNo **References** Students**Delete Restricted****Update Cascades****Foreign Key** pNo **References** Presentations**Delete Restricted****Update Cascades****Relation** Qualifications**Attributes**

lecturerCode: lecturerCodes

courseNo: courseNos

Primary Key lecturerCode, courseNo**Foreign Key** lecturerCode **References** Lecturers**Delete Restricted****Update Cascades****Foreign Key** courseNo **References** Courses**Delete Restricted****Update Cascades**

We have assumed that studentNames and lecturerNames are all personNames. Note that we have not specified the null characteristics of the foreign keys in attendance and qualifications. This is because both foreign keys are part of the primary key, and no part of a primary key may be null.

19.4**PHYSICAL DATABASE DESIGN PROCESS**

A logical database model provides insufficient information to enable effective physical database design. Usually, the following activities need to be performed as part of physical design:

- Volume analysis
- Usage/transaction analysis
- Integrity analysis
- Control/security analysis
- Distribution analysis

The output from the physical database design process is an implementation plan for the database. This process is illustrated in Figure 19.3.

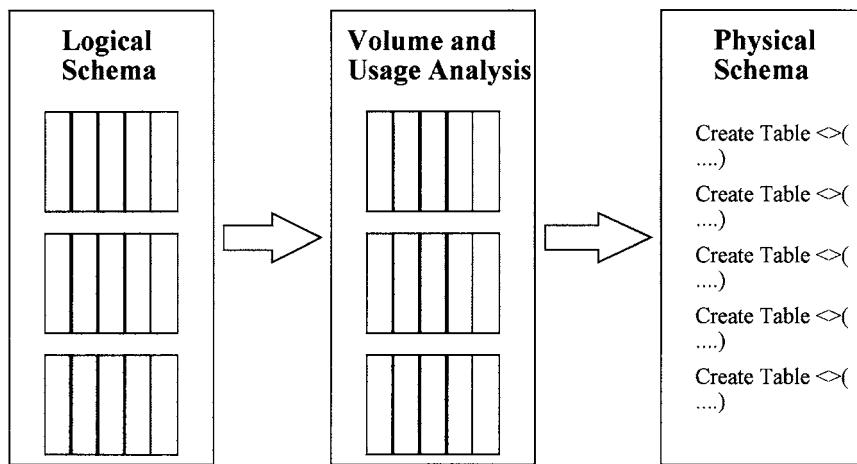


Figure 19.3 Physical database design.

This plan will be composed of the following elements:

- Data structures declared in a suitable data definition language
- Indexes declared on the data structures
- Clustering data where appropriate
- A set of inherent integrity constraints expressed in some data definition language and a set of additional integrity constraints expressed in some data integrity language
- A distribution strategy for the database system, including a plan for distributing data
- A set of queries optimised for running on some database
- A set of defined users

In this chapter we concentrate on the activities appropriate to physical database design. In the next chapter we look more closely at the issues of database implementation.

19.5 (c) VOLUME ANALYSIS

One of the first steps we need to take in moving from logical to physical database design is to establish estimates as to the average and maximum number of instances per entity in our database. An estimate of the maximum possible number of instances per entity is useful in deciding upon realistic storage requirements. An estimate of how many instances are likely to be present in



the system on average also gives us a picture of the model's ability to fulfil access requirements.

Example  The table below summarises some provisional sizing estimates for the USC database. Using the column sizes established in the schema above it is relatively straightforward to translate entity sizing into table sizing. For each relation described in our database the relevant calculations are:

Table	Rows Max	Rows Avg	Column Size	Table Max	Table Avg
Courses	100	80	37	3,700 Chars	2,960
Lecturers	50	40	107	5,350 Chars	4,280
Students	1,000,000	300,000	83	83,000,000	24,900,000
Presentations	100,000	40,000	52	5,200,000	2,080,000
Qualifications	100	80	6	600	480
Attendance	5,000,000	1,000,000	13	65,000,000	13,000,000
Total				103,206,320	39,987,720

19.6 USAGE ANALYSIS

Usage analysis requires that we identify the major transactions required for a database system. Transactions are considered here as being made up of a series of insertions, updates, retrievals or deletions, or a mixture of all four. In the next section we will consider a number of retrieval transactions.

Example  Below we give a sample list of insertion, update and delete transactions of relevance to USC:

- Register a new student on a presentation
- Add new presentation details
- Purge all lapsed presentations before a certain date
- Assign a lecturer to a presentation
- Record details of a possible student

Sweet (1985) makes a useful distinction between two kinds of entities or tables in a database application: volatile and non-volatile. The difference between the

average number of instances and maximum number of instances estimated for an entity gives us a rough idea of the volatility of an entity or table.

Example → Imagine a classic stock control database. In this database there is a supplier table of 10,000 rows; 100 new suppliers are added to this table each month and 100 inactive rows are purged from the table in the same period. The table's monthly volatility – turnover divided by population – is therefore 1%. Compare this with a table of purchase orders of 1,000 records, where 1,000 orders are added each month and the same number closed. The monthly volatility of this table is 100%.

If this calculation is conducted for each table in a database and the results plotted on a histogram, two distinct peaks will be seen. One, such as suppliers, centred close to 1%, and one, such as purchase orders, centred close to 100%. Volatility appears to be a binary characteristic of tables.

The classic non-volatile table is the reference table. In terms of USC, Lecturers and Courses are two tables of this nature. The state of such tables changes relatively infrequently. Few, if any, update transactions impact on these tables in say a period of six months. In contrast, Students and Attendance are classic volatile tables. In the same period, many students and attendance rows will be added and consequently many purged.

As we shall see, the volatility of a table has a clear impact on its expected update and retrieval performance.

Example → We therefore mark its expected volatility against each of the tables in the USC database, as below:

Table	Volatility
Courses	low volatility
Lecturers	low volatility
Students	high volatility
Presentations	high volatility
Qualifications	low volatility
Attendance	high volatility

19.7 TRANSACTION ANALYSIS

Transaction analysis involves analysing and documenting the set of critical transactions that is expected to impact against some database system. Ideally,



transaction analysis will not be viewed as conventional information systems analysis and design in the sense that these activities should document the major data management activities expected in an application.

For each transaction we need to determine:

- The expected frequency
- The data structures accessed by the transaction as well as the type of access: query, insert, update and delete
- Any constraints placed on the transaction, such as whether a transaction must be completed within a given time period

We are unlikely to be able to analyse all transactions using the above criteria as a guideline. A useful heuristic is to consider examining the 20% of critical transactions. For such critical transactions it is useful to produce so-called transaction maps.

19.7.1 TRANSACTION MAPS

Figure 19.4 illustrates a sample transaction map produced for one of the queries identified as important to run against the UCS database – *Tell me how many students have presently booked for a particular presentation*. This map illustrates the stages of the transaction (represented by numbered arrows) against a subset of the USC model. The entry point is at the presentation entity; then the path goes to the attendance entity; then it proceeds to the student entity.

List Students Against Presentation

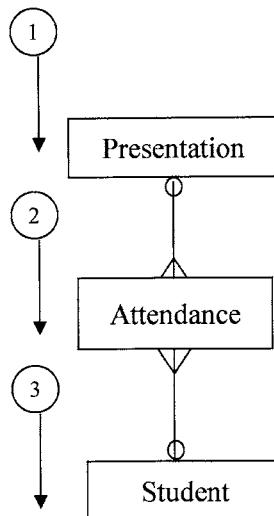


Figure 19.4 An example transaction map.

Example An analysis of each transaction step is entered in the table below:

Step	Type of Access	No. of References Per Transaction	Per Period
1	Read	1	10
2	Read	20	100
3	Read	1	10
Total		22	120

The type of access to each entity is recorded. Since this is a retrieval transaction all these steps are read transactions. **List Students Against Presentation** requires only one presentation instance per transaction. In step 2, for each presentation instance we establish that 20 attendance instances on average are accessed. In step 3, for each attendance instance one student instance needs to be accessed.

We also establish that this particular transaction has a peak volume of 10 transactions per hour (period). Therefore the number of accesses per period for step 1 of the transaction is likely to be 10. For step 2 the expected number of accesses is likely to be 100. Finally, for step 3 the number of accesses per period is 10.

Totalling the number of references or accesses per transaction provides us with some idea of the expected processing requirements for this transaction.

19.8 ACCESS REQUIREMENTS

The size and volatility of a table have a clear impact on what access we can provide to data. The larger the table, the more necessary it is to build access structures into the data. The more volatile the table however, the greater the penalty we pay for such access structures.

Example We list below seven common queries or retrieval transactions to be run on the USC database. These queries would be run on the database time and time again by staff at USC. They therefore form suitable candidates for packaged reports. A major objective of physical database design is to ensure that the access performance of such reporting functions is satisfactory. The frequency with which each of these reports is run on the database will usually provide the database designer with sufficient information to decide upon the degree of access performance required in each case. For instance, the first query might be run several times in every hour of every working day by members of USC. Retrieval performance is therefore of the utmost importance. In contrast, the fourth query is probably run only once a quarter. This is to allow managers at USC to assess the performance of scheduled as compared



to on-site courses. It would therefore be sensible to run this query as an overnight process. Retrieval performance in this case would not be of critical importance. We list below the expected frequency of each of these reports:

- Tell me how many students have presently booked for a particular presentation (run many times a day)
- Give me a list of which lecturers qualified to teach course X are giving course X on the week of Y (run once a day)
- Produce a schedule of what lecturers are giving what presentations for a particular course (run once a month)
- Produce an analysis of courses by number of on-sites and number of university courses (run once a quarter)
- Produce a list of previous students by company for use as a mailing list (run once every six months)
- Produce a total of attendance for each course from the previous year's schedule (run once per year)
- Give me a list of which lecturers are qualified to teach what courses (run once a year)

19.9

INTEGRITY ANALYSIS

Classic data analysis provides us with a logical database design which indicates appropriate data structures and a set of inherent integrity constraints. For example, three types of inherent constraints should be documented in a relational schema (Chapter 7): entity, referential integrity and domain constraints.

Example *Entity integrity constraints:* lecturerCode is the primary key of Lecturers, and hence lecturerCode must be unique and cannot be null.

Referential integrity constraints: every presentation must have an associated Courses record identified by courseNo. In this business, courseNo in Presentations cannot be null. Other examples are: do not assign a non-existent instructor to a session; do not delete course until all appropriate presentations are deleted; do not delete instructor until all appropriate presentations are deleted.

Domain constraints: cno in courses should be number(3) and taken from the set {<course numbers>}, or values for the column site in Presentations must be taken from the set {university, hotel, on-site}.

Additional constraints are usually needed by any application. An important class of additional constraint is the so-called transition constraint. Such

constraints document valid movements from one state of the database to another.

Example → An example of a transition constraint of relevance to USC would be: *remove a presentation if the number of people booked < 4, 2 weeks before due date*. Another example might be: *remove a lecturer from qualifications if he has not given at least three presentations of a course in any one year*.

Static constraints involve checking that a transaction will not move the database to an invalid state.

Example → Instances of static constraints of relevance to the USC database might be:

- Lecturer assigned to a presentation must be qualified to teach the course
- The duration of a presentation should equal the duration of a course
- The number of students booked for a presentation \leq course limit

Any constraint such as enforcing referential integrity is expensive in update performance terms. Every time a new attendance record is inserted, for instance, a check will need to be made against the Students file and the Presentations file. Constraints such as cascading updates are even more expensive in update performance terms. For instance, a transaction updating a course record might fire off a whole series of updates in the presentations file.

The output from integrity analysis should be a documented set of additional integrity constraints.

19.10 SECURITY/CONTROL ANALYSIS

Security is a large issue involving securing buildings and rooms, hardware, operating systems and DBMS. Many organisations now employ special persons to deal with information security issues. In this section we concentrate on the issue of database security. However, there is little point in securing a DBMS and database by itself. To take an example, most DBMS run on top of native operating systems and place their database tables within operating system files. Hence one must secure the operating system against attack by unwanted persons as well as securing the tables within the database.

Database security is normally assured by using the data control mechanisms available under a particular DBMS. Data control comes in two parts: preventing unauthorised access to data, and preventing unauthorised access to the facilities of a particular DBMS. Database security will be a task for the Database



Administrator (Chapter 23) – normally conducted in collaboration with the organisation's security expert.

To exercise control over data and privileges we first need to build a profile of the users or user groups that we expect to access the database. This should be possible as a result of our usage analysis. For each such user group we then need to build a profile of data and facilities privileges appropriate to their expected level of access.

19.11

DEFINING PERFORMANCE

When relational DBMS products first emerged into the commercial arena they were heavily criticised for being poor performers. The relational data model was consequently attacked not only about its ability to support transaction-processing applications, but also in the area of decision support, where relational databases were traditionally held to be strong.

Many of these early criticisms were however misdirected. They ignored the crucial fact that the relational data model is a logical data model. It is an abstract machine and as such is devoid of physical implementation concerns. Performance is therefore not a critical issue for the relational data model. Performance is a critical issue for relational database systems and for their post-relational cousins.

But what is performance? The definition and implementation of performance are usually a balancing act. By performance do we mean access performance or update performance? Do we need a clean-cut data model amenable to change, or is our application stable enough to allow the reintroduction of redundancy?

If we set fast access to data as a priority we may have to sacrifice data entry speed. By transforming data structures to meet the needs of fast access we may have to sacrifice some aspects of a clean-cut business data model.

We began this chapter by highlighting that while generalities are possible in the area of logical database design, few such generalities are possible in the area of physical design. This is because of the inherent implementation-dependent nature of physical design.

This is particularly true of the concept of performance. Performance is a relativistic concept. It is application-specific. In conducting any physical design exercise it is therefore extremely important to construct a mission statement for the proposed application. This mission statement should contain the following items:

- A *volume analysis* – estimates of the maximum and average number of instances per entity. Also, a volatility classification for each entity in our data model
- A *usage analysis* – a prioritised list of the most important update and retrieval transactions expected to impact on the applications data model. As a minimum we should indicate the expected frequency of each of these transactions
- An *integrity analysis* – inherent integrity constraints, particularly referential integrity constraints, can be written onto the data model. The most

important domain and additional constraints can be specified in an associated data dictionary

19.12 CASE STUDY: ACCESS CONTROL AGAINST THE USC DATABASE

In terms of the USC database we might establish three user groups: managers, administrators and lecturers. For each such user group we then need to build a profile of data and facilities privileges appropriate to their expected level of access.

- *Administrators.* Different users among the administrators at USC might be established and given different forms of access to the database. For instance, portfolio administrators may be given update rights against courses, lecturers and qualifications. Scheduling and bookings staff may be given update privileges against students, attendance and presentations
- *Lecturers.* We might establish that lecturers should have access to the presentations they are scheduled to teach, they should not have update access to any of this data, and they only need to be able to log on and query a view built across the lecturer/presentations tables
- *Managers.* Such persons will not normally want update access against any aspect of the database. However, they will probably want retrieval access to all the objects in the database

19.13 SUMMARY

- Physical database design follows on from logical database design. Usually the following activities need to be performed as part of physical design: volume analysis, usage/transaction analysis, integrity analysis, control/security analysis and distribution analysis
- A volume analysis establishes estimates as to the average and maximum number of instances per entity in our database
- A usage analysis identifies the major transactions required for a database system. This will include a volatility analysis and an analysis of access requirements
- An integrity analysis will identify and specify the inherent and additional integrity constraints needed to be supported by the application
- A security analysis will determine the security and data control requirements for the database application

19.14 ACTIVITIES

1. Generate the USC schema as a series of ANSI SQL CREATE TABLE statements.
2. Build a complete mission statement for the USC database.



3. Indicate precisely the implementation decisions you would take in the light of the mission statement in activity 2.
4. Generate the stock market schema (Chapter 17) as a series of ANSI SQL CREATE TABLE statements.
5. Build a complete mission statement for the stock market database. You will have to generate your own requirements.
6. Indicate precisely the implementation decisions you would take in the light of this mission statement.

19.15

REFERENCE

Sweet, F. (1985). A 14-part series on process-driven database design. *Datamation* Jan.–Nov.



DATABASE IMPLEMENTATION

Neither snow, nor rain, nor heat, nor gloom of night stays these couriers from the swift completion of their appointed rounds.

Herodotus (484–430 BC)

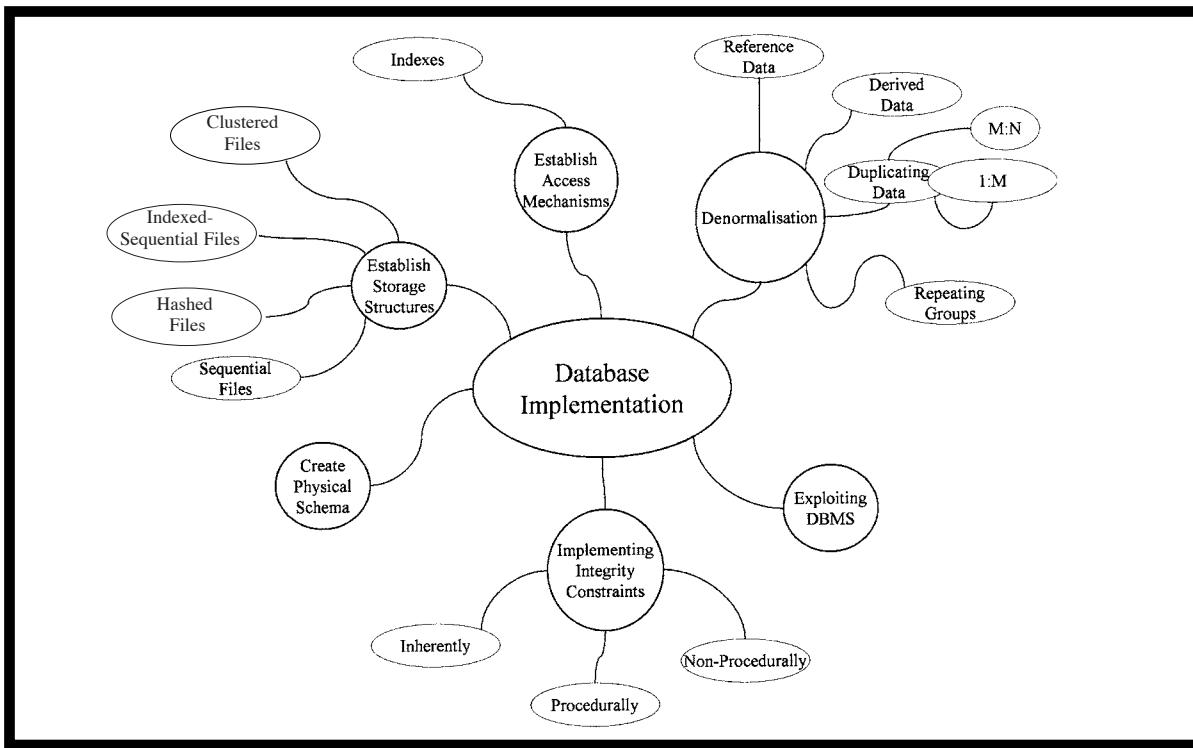
Now here, you see, it takes all the running you can do, to keep in the same place.

Lewis Carroll, *Through the Looking Glass*

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define the process of database implementation
- Describe a list of common database implementation decisions
- Discuss options in the establishment of storage-related structures and access mechanisms
- Discuss the process of denormalisation and various options used to implement integrity constraints



20.1 INTRODUCTION

Physical database design involves taking the results from the logical design process and fine-tuning them against the usage, performance and storage requirements of some application. Database implementation involves using the strategies developed as output from physical database design to make decisions as to constructing the database system using the mechanisms or features of some DBMS.

20.2 INPUTS TO AND OUTPUTS FROM PHYSICAL DESIGN

To refresh, the main outputs from physical database design are:

- Volume analysis
- Usage/transaction analysis
- Integrity analysis
- Control/security analysis
- Distribution analysis (Chapter 37)

The main outputs from database implementation are:

- Data structures declared in a suitable data definition language
- Indexes declared on the data structures
- Clustering data where appropriate
- A set of inherent integrity constraints expressed in some data definition language and a set of additional integrity constraints expressed in some data integrity language
- A distribution strategy for the database system, including a plan for distributing data (Chapter 37)
- A set of queries optimised for running on some database
- A set of defined users

20.3 CREATING THE PHYSICAL SCHEMA

Creating the physical schema involves implementing the data structures and inherent integrity constraints in a database sublanguage such as SQL.

Example → In our case, we use the *create table* command of SQL2 to create the major elements of the USC schema (Chapter 19).

```

CREATE TABLE Courses(
    courseNo INTEGER(3),
    courseName CHARACTER(30),
    courseManager INTEGER(3),
    courseDuration INTEGER(1),
    PRIMARY KEY(courseNo)
    FOREIGN KEY(courseAuthor) REFERENCES Lecturers NOT NULL
    ON DELETE NO ACTION
    ON UPDATE CASCADE)

CREATE TABLE Lecturers(
    lecturerCode INTEGER(3),
    lecturerName CHARACTER(30),
    HomeAddress CHARACTER(50),
    IWorkAddress CHARACTER(50),
    IHomeTelNo CHARACTER(17),
    IWorkTelNo CHARACTER(17),
    PRIMARY KEY(lecturerCode))

CREATE TABLE Students(
    studentNo INTEGER(6),
    studentName CHARACTER(30),
    studentAddress CHARACTER(50),

```



```

studentTelNo CHARACTER(17),
PRIMARY KEY (studentNo))

CREATE TABLE Presentations(
presentationNo INTEGER(5),
pSite CHARACTER(10),
courseNo INTEGER(3),
pDate DATE,
lecturerCode INTEGER(3),
IHomeTelNo CHARACTER(17),
IWorkTelNo CHARACTER(17),
PRIMARY KEY (lecturerCode)
FOREIGN KEY (courseNo) REFERENCES Courses NOT NULL
ON DELETE NO ACTION
ON UPDATE CASCADE
FOREIGN KEY (lecturerCode) REFERENCES Lecturers
ON DELETE NO ACTION
ON UPDATE CASCADE)

CREATE TABLE Attendance(
studentNo INTEGER(6),
pNo INTEGER(5),
feePaid CHARACTER(1),
PRIMARY KEY (studentNo, pNo)
FOREIGN KEY (studentNo) REFERENCES Students
ON DELETE NO ACTION
ON UPDATE CASCADE
FOREIGN KEY (pNo) REFERENCES Presentations
ON DELETE NO ACTION
ON UPDATE CASCADE)

CREATE TABLE Qualifications(
lecturerCode INTEGER(3),
courseNo INTEGER(3),
PRIMARY KEY (lecturerCode, courseNo)
FOREIGN KEY (lecturerCode) REFERENCES Lecturers
ON DELETE NO ACTION
ON UPDATE CASCADE
FOREIGN KEY (courseNo) REFERENCES Courses
ON DELETE NO ACTION
ON UPDATE CASCADE)

```

20.4



DATABASE IMPLEMENTATION DECISIONS

Below, we list the five major options available to the relational database developer in fine-tuning his or her application for performance:

- Establishing storage structures and associated access mechanisms
- Adding indexes
- Denormalisation
- Exploiting your DBMS
- Integrity constraints

In the next few sections we illustrate some of the heuristics for implementation decisions based upon information provided in an annotated data model.

20.5 (c) ESTABLISHING STORAGE-RELATED ACCESS MECHANISMS

There are four major mechanisms available in relational systems for storing data on secondary storage in a way designed for access (Chapter 27):

- Sequential files
- Hashed files
- Indexed-sequential files
- Clustered files

In this section we discuss a number of heuristics or rules of thumb that may be used to decide upon appropriate storage structures.

20.5.1 SEQUENTIAL FILES

Sequential files and the associated sequential access mechanism are recommended for:

- Small tables
- Medium-to-large tables when more than 20% of rows need to be accessed to satisfy some request
- Any tables which are accessed by low-priority queries or which are suitable for batch processing

Example → Our volume and usage analysis should direct us to the areas within our application where sequential access is appropriate. Our volume analysis will locate the small tables in our application such as lecturers and courses. A detailed usage analysis will tell us how many rows are likely to be accessed in a transaction as well as the expected frequency of transactions. Hence we might find for instance that because query 4 in Chapter 19 (*produce an analysis of courses by number of on-sites and number of university courses*) accesses most of the rows in the presentations file and only happens irregularly, sequential access is a perfectly valid approach for this transaction. Another key use for sequential files is when data needs to be bulk-loaded into a table prior to being indexed.



20.5.2 HASHED FILES

Hashed files are useful in situations in which it is possible to identify one overarching access method based on one key value. A hashed file would normally be imposed on the major access path into a file and therefore, in relational systems, will normally be built on primary keys.

Hashed files are not recommended for situations in which:

- The rows of a table need to be accessed on a range of values of the hash key
- Rows need to be retrieved on an attribute other than the hash key
- When access is required in terms of a pattern-match against the hash key or on the basis of part of the hash key

20.5.3 INDEXED-SEQUENTIAL FILES

Indexed-sequential, sometimes known as indexed-sequential access mechanism (ISAM), files are a more versatile structure than hash files in that they support retrievals based on an exact key match, pattern-matching and range searches. However, because in an ISAM file the index is created when the file is created, then retrieval performance of an ISAM file deteriorates as the file is updated.

20.5.4 CLUSTERED FILES

Clusters will be built to effectively enforce pre-joins on data.

Example

In the USC database it may be important to build a cluster on presentations and students to help run query 1. In terms of any other form of access over and above the pre-join, clustered files tend to offer poor retrieval performance. Given their interleaved structure they also carry something of an update performance overhead.

20.6**ADDING INDEXES**

An index is a mechanism for improving access to data without changing the underlying storage structure. It is useful to distinguish between two major types of index: primary indexes and secondary indexes.

As a bare minimum, we should always index on primary and foreign keys. It is usual to create unique indexes on primary keys. Many RDBMS now do this automatically. Since most effective and meaningful joins occur via foreign keys, indexing on these data-items can radically improve the performance of such joins.

Although it is theoretically possible to index on all the data-items in a table, this is rarely done in practice, the major reason being that although indexes can radically improve access performance, they have a negative effect on

update performance. In other words, every time we add a record to a table, each index on that table has to be updated. Hence, the more indexes we have on a table, the longer it takes to insert an additional record.

We always index on primary and foreign keys. We sometimes index on other non-key attributes of a table, usually when the attribute is extremely important in the running of some report.

Example → Hence we might build an index on presentationDate, for instance, to help us to retrieve all presentations due to start on a particular date.

In general we build indexes on medium-to-large tables to facilitate access to a small percentage of rows in a table.

20.7 DENORMALISATION

The main problem with a fully normalised database is that it is usually made up of many tables. To perform useful queries, such tables have to be reconstituted via expensive join operations. Also updates frequently have to be performed across more than one table.

One obvious way of improving retrieval or update performance is therefore to step back from a fully normalised database and introduce some controlled redundancy. Control however is the operative word. By reintroducing redundancy we are likely to boost retrieval performance at the expense of more complex integrity maintenance (Rodgers, 1989). In this section we consider a number of alternative ways of conducting denormalisation.

20.7.1 REFERENCE DATA

A good example of a case for denormalisation is where reference data is employed in a one-to-many relationship.

Example → Consider the case of the Presentations and Lecturers tables in the USC database. We see that there are very few rows in the Lecturers table. It might therefore be possible to add the Lecturers data to the Presentations file. Doing this however means that if for some reason we wish to change the details of a particular lecturer, we now have to update multiple rows in the presentations file.

20.7.2 DERIVED DATA

Another example of denormalisation is the storage of derived data.



Example → Consider the case of a simple order-processing database with the following normalised schema:

```
Orders(oNo, oDate, cno)
OrderLines(oNo, pNo, qty)
Products(pno, pDesc, pPrice)
```

If we frequently find that we need to calculate the total line price and total order price from this database, it may make sense to store the derived data directly as in the modified schema below:

```
Orders(oNo, oDate, cNo, totalOrderPrice)
OrderLines(oNo, pNo, qty, linePrice)
Products(pno, pdesc, pprice)
```

20.7.3 DUPLICATING DATA IN MANY-TO-MANY RELATIONSHIPS

Suppose we need to run the query: *give me the list of lecturers who are qualified to teach X*. Assuming that X equals a course name then the query needs to join the lecturers, courses and qualifications tables together. One way of improving the performance of this relation is to include the course name and lecturer name attributes in the intermediate or link table *qualifications*.

20.7.4 DUPLICATING NON-KEY ATTRIBUTES IN ONE-TO-MANY RELATIONSHIPS

Consider the query: *who is teaching a particular presentation?* Assuming we only wish to know the name of the person here, we could include lecturer name as an attribute of the presentation table.

20.7.5 REINTRODUCING REPEATING GROUPS INTO SCHEMA

An example of reintroducing repeating groups into a schema might be combining company and student data together to rapidly answer such queries as providing a list of students from particular companies.

As a general rule, denormalisation should only be attempted when all other techniques fail. Volume and volatility analyses frequently highlight where denormalisation may be appropriate. In general, low volatile, small tables are likely candidates for denormalisation. Denormalising large, high volatile tables should be avoided. This is because normalised schemas are designed to prevent update problems. Volatile tables are subject to large amounts of update and hence need the benefits offered by normalisation.

20.8 (c) IMPLEMENTING INTEGRITY CONSTRAINTS

Specifying constraints in design terms for a given application is relatively straightforward. Implementing constraints however is somewhat more involved. There are three major ways of implementing integrity constraints:

20.8.1 INHERENTLY

If the data model underlying a DBMS is sufficiently rich, much work can be accomplished merely using the integrity mechanisms supported by the data model. In a DBMS defined on the relational data model, for instance, we should at least be able to specify entity, referential and domain integrity (Chapters 7 and 12).

20.8.2 PROCEDURALLY

Much of the present-day implementation of integrity constraints occurs outside the domain of a particular architectural data model. Most existing application programs that interface with databases implement constraints procedurally in third generation languages such as C or COBOL, or fourth generation languages such as PL/SQL (Chapter 24).

20.8.3 NON-PROCEDURALLY

A third strand of constraint management has emerged. Many people see the logical place for implementing integrity constraints as being the data dictionary or system catalog associated with the RDBMS. Constraints held in a central data dictionary offer similar advantages to data held in a central database. Constraints are maintained independent of application programs, the aim being to manage integrity rules on a corporate level. The constructs of triggers and stored procedures are directed at this.

However, currently it is true to say that implementing constraints procedurally is far more efficient than implementing constraints, whether inherent or additional, non-procedurally or declaratively. Just as we have to balance update performance against access performance, so must we balance update performance against the need for integrity. We must also judge whether the benefits of integrity independence outweigh the clear performance advantages of implementing constraints procedurally.

20.9 (c) SELECTING AND EXPLOITING A DBMS

Selecting a DBMS for your application can affect the performance of your application in a number of ways. For instance, product selection may affect performance in terms of query-handling. Some systems compile all the queries run on the database and store for reuse the access plans generated. Other systems



interpret queries at run-time. As a general rule, compiled queries execute faster than interpreted queries. This is because the generation of an access plan need only be done once. However, if your database is volatile or the major form of access is of an *ad-hoc* nature, then interpreted queries are probably more flexible. The overhead involved in compilation of many one-off queries will far outweigh any performance advantage gained.

20.10**CASE STUDY: IMPLEMENTING QUERIES FOR THE USC DATA SYSTEM**

In Chapter 19 we described a series of common queries that we would expect to run against the USC database. Examples of how we might implement each of these queries in SQL are presented below:

Tell me how many students have presently booked for a particular presentation

```
SELECT pNo, COUNT(*)
FROM Attendance
GROUP BY pNo
```

Give me a list of which lecturers qualified to teach course X are giving course X on the week of Y

```
SELECT p.lecturerCode, p.courseNo, p.presentationNo, pDate
FROM Qualifications Q, Presentations P
WHERE Q.courseNo = P.courseNo
AND courseNo = %1
AND pDate = %2
```

Produce a schedule of what lecturers are giving what presentations for a particular course

```
SELECT lecturerCode, courseNo, presentationNo, pDate
FROM Presentations
WHERE courseNo = %1
ORDER BY courseNo
```

Produce me an analysis of courses by number of on-sites and number of university courses

```
SELECT courseNo, pSite, Count(*)
FROM Presentations
GROUP BY pSite
```

Produce a list of previous students by company for use as a mailing list

```
SELECT studentName, studentAddress, StudentTelNo
FROM Students S, Attendance A, Presentations P
WHERE S.studentNo = A.studentNo
AND A.pNo = P.presentationNo
AND pDate < SYSDAT
ORDER BY studentName
```

Produce a total of attendance for each course from the previous year's schedule

```
SELECT C.courseNo, courseName, count(*)
FROM Attendance A, Presentations P, Courses C
WHERE C.courseNo = P.courseNo
AND P.presentationNo = A.pNo
AND pDate BETWEEN %1 AND %2
GROUP BY C.courseNo
```

Give me a list of which lecturers are qualified to teach what courses

```
SELECT courseNo, L.lecturerCode, courseName
FROM Qualifications Q, Lecturers L, Courses C
WHERE Q.LecturerCode = L.lecturerCode
AND Q.cNo = C.courseNo
ORDER BY C.courseNo, L.lecturerCode
```

20.11 SUMMARY

- Database implementation follows on from physical database design
- Database implementation involves the following activities: creating the physical schema, establishing storage structures and associated access mechanisms, adding indexes, conducting denormalisation, where appropriate, exploiting the facilities of the chosen DBMS and implementing integrity constraints
- Creating the physical schema involves implementing the data structures in the chosen database sublanguage
- Storage structures for the database objects are established using the options of the chosen DBMS to meet the needs of usage
- Indexes are used to improve retrieval performance and will be guided by the analysis of access requirements
- Denormalisation of the physical schema may be undertaken to improve performance. Denormalisation should be used with care
- Knowing about the facilities of the chosen DBMS and how to use these to fine-tune the database is important
- Appropriate ways are chosen from the following to implement inherent and additional integrity constraints: inherently, procedurally or non-procedurally

20.12 ACTIVITIES

1. Produce a physical schema for Goronwy Galvanising described in the Case Study of Chapter 3.
2. Conduct a tentative volume analysis on the schema of activity 1 using some reasonable assumptions.



3. Develop a suitable storage strategy for the schema of activity 1, including where indexes will be built.
4. Assess whether denormalisation might be appropriate for this schema.
5. Determine the degree to which integrity constraints will be implemented inherently, procedurally and non-procedurally for this schema.

20.13

REFERENCE

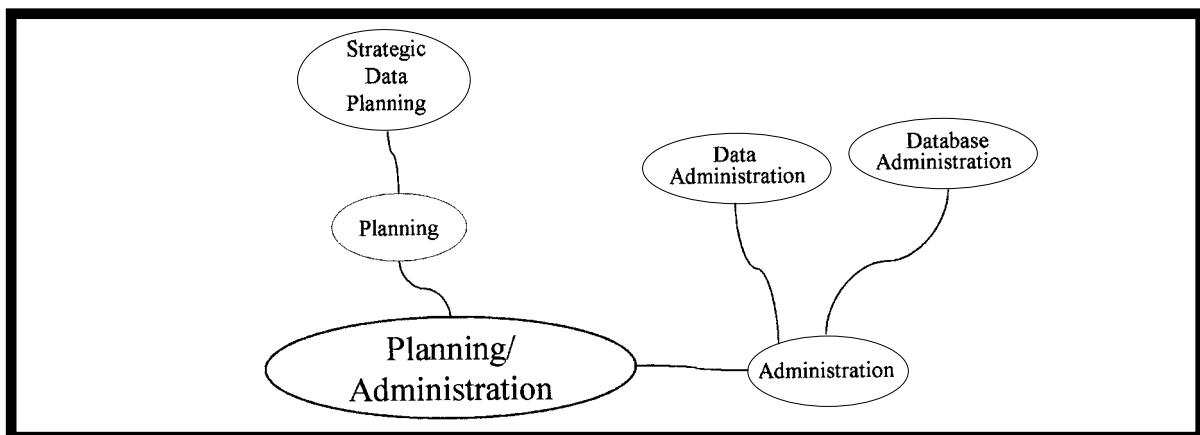
Rodgers, U. (1989). Denormalisation: why, what and how? *Database Programming and Design* 2(12): 46–53.



PLANNING AND ADMINISTRATION OF DATABASE SYSTEMS

Our memories are card indexes consulted, and then put back in disorder by authorities whom we do not control.

Cyril Connolly (1903–74)





Database systems have become so important to modern organisations that significant activity is now devoted to planning for, monitoring and administering such systems. In this part we discuss three important planning and managerial activities relevant to database systems: strategic data planning, data administration and database administration.

- *Strategic data planning* is the process of developing corporate data models and using such models for the planning and management of application development (Chapter 21)
- *Data administration* is that function concerned with the management, planning and documentation of the data resource of some organisation (Chapter 22)
- *Database administration* involves the technical implementation of database systems, managing the database systems currently in use, and setting and enforcing policies for their use (Chapter 23)

CHAPTER 21 →

STRATEGIC DATA PLANNING

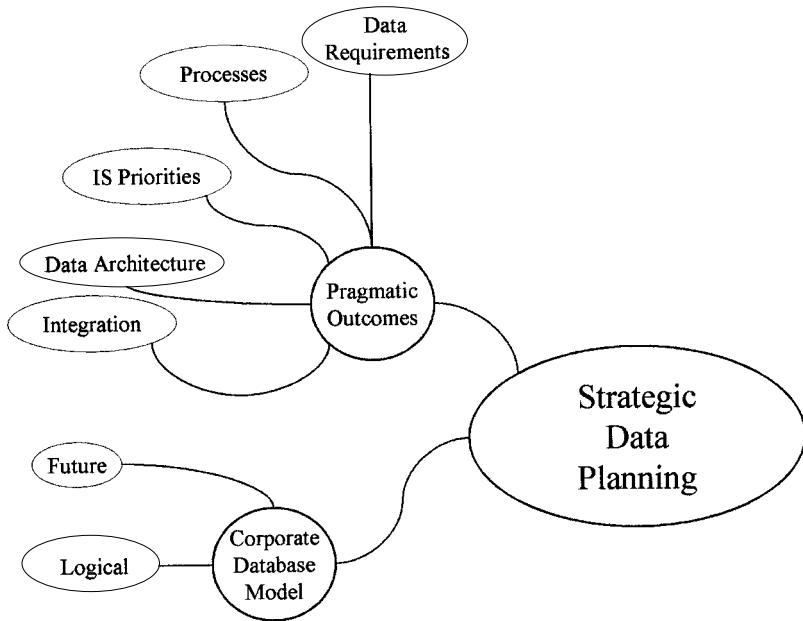
When we are planning for posterity, we ought to remember that virtue is not hereditary.

Thomas Paine (1737–1809)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Distinguish between corporate data modelling and application data modelling
- Discuss the pragmatics of strategic data planning




21.1


INTRODUCTION

To review, database development, the process of representing some universe of discourse (UoD) as a database system, can generally be said to consist of the following phases: requirements elicitation, conceptual modelling, logical modelling and physical modelling. Conceptual modelling is normally further divided into two substages: view modelling, a process that transforms user requirements into a conceptual model; and view integration, a process that combines individual views into a global conceptual model.

It thus seems to have become established practice that the appropriate way to develop database systems is to engage in a prior exercise in application data modelling. Data modelling is the process of developing an implementation-independent map of database requirements. Most data modelling is therefore conducted to aid in the development of some application system. In this sense, the term data model is used to describe a blueprint for the data requirements of a particular application.

Having said this, Hudson (1991) usefully classifies data models on two dimensions: a logical–physical dimension and a current–future dimension. Most application data models attempt to document data structures to support new systems – future data models. Data models can also be used to document existing ‘legacy’ systems – current data models. Logical models are designed to support the organisational view of data; physical models are designed to support the technical view of data (Figure 21.1). The activity devoted to the construction of future/logical data models is generally referred to as strategic data planning. This activity is the topic of the current chapter.

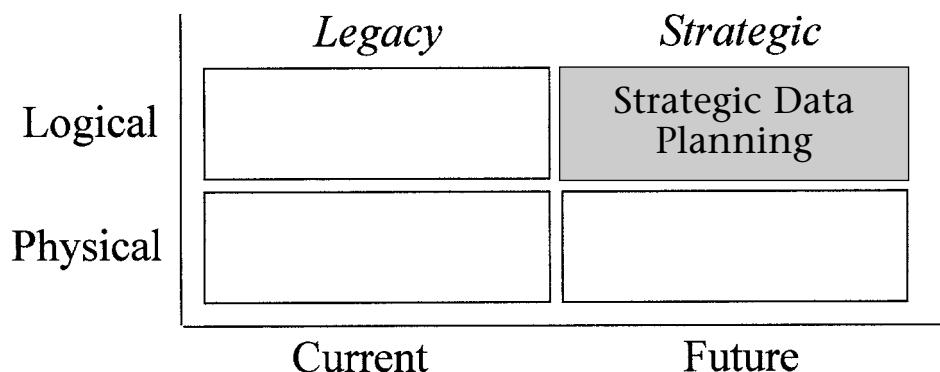


Figure 21.1 Data model types.

21.2


CORPORATE DATA MODELLING

Prefixing the term data modelling with the word corporate would tend to suggest an elevation of this practice to the level of the whole organisation. A

corporate data model (CDM), using this perspective, should form a map of the data requirements of the whole or a substantial part of an organisation. This differs from an application data model which is produced to support a specific information systems project (Beynon-Davies, 1997).

Perhaps part of the reason for the scarcity of material on the topic of CDM may be because of a commonly held assumption that developing a corporate data model is no different an activity from that of developing an applications data model. As we shall see, however, whereas the techniques applied may be similar, the specific problems involved in corporate data modelling relate to the much higher level of abstraction demanded by this activity as compared to developing an applications data model.

The idea of a CDM as a model of large scope is clearly the emphasis in methodologies such as Information Engineering, in which corporate data modelling forms a key activity within an information strategy planning phase of development (Braithwaite, 1992). The idea of CDM in association with modelling corporate processes also seems to underlie the idea of enterprise modelling and business objects. CDM is also significant in the idea of information (resource) management and the discipline of data administration which has been floated since the early 1980s.

However, there appear to be some differences in the way in which particular organisations construct such a map. One approach is to build what amounts to a map of very large scale. On such a map only the key features of the data terrain are delineated. The intention of such a map seems to be to act as an aid in controlling the systems development process in such organisations. This we shall call a type 1 CDM. Another approach is to build a map to a smaller scale but in a schematic manner. A minimum data-set is provided for key organisational functions. The key rationale for this type of corporate data model seems to be to act as a means of standardising data capture and usage across what may be an ill-structured environment in information terms. This we shall call a type 2 CDM.

21.3 EXISTING LITERATURE ON CDM

Hudson (1991) briefly describes an attempt to develop both current and future corporate database models at Sara Lee Corp, a large clothing and food consortium based in Chicago, USA. She particularly documents the importance of CASE technology to her CDM project, permitting the analysis of some 135 entities in three months. She states that 'a future logical corporate data model can play a significant role in effectively guiding long-range applications development and information use.' She adds however that the creation of such a model is a labour-intensive process that needs the support of CASE tools.

Sussman (1993) documents a Canadian case study in CDM – the process of building a CDM for the municipal activities of the city of Scarborough, Ontario. Although the paper is particularly concerned with the problem of integrating spatial data for Geographical Information Systems work into a



CDM (Chapter 39), Sussman makes some interesting comments about the process of building a CDM in general. The Scarborough CDM seems to have been developed as part of a larger information systems planning effort. Part of the initial analysis found that at least 125 data-sets were employed in the municipality, which, in total, required 28 man years per annum to maintain. In this context, the main aim of the CDM was to integrate diverse data-sets, and as a consequence produce a municipal data model independent of activities, computer technology or the organisational structures of departments (the idea that data is 'owned' by departments). Sussman sees a CDM as encouraging the perception of data as a city-wide resource that helps promote the sharing of data. The key use of the CDM seems to be in the planning of application development.

The UK government's centre for information systems, the CCTA, published a handbook on corporate database modelling in the mid-1990s (CCTA, 1994). It defines a corporate database model as being in very broad terms: 'a set of definitions for things of significance to the organisation, about which information needs to be held, and the relationships between them.' It describes a corporate data model as being a progression through three levels of detail:

- *Strategic tier* – a model of major business concepts
- *Architectural tier* – more detailed model including additional attributes and volume information
- *Application tier* – an aggregate of more detailed models from which applications are built

The CCTA sees the main reason for building a CDM to be to harmonise the data requirements across a number of diverse application systems. The idea is to increase the integration and interoperability of information systems through data sharing.

The CCTA maintains that corporate data modelling should be accomplished by a small team of analysts and identifies two critical success factors for corporate data modelling:

- High level of support within the organisation for what must be seen to be a valuable activity
- Strong end-user involvement with the modelling

Because of the importance of these factors it suggests that corporate data modelling should first be conducted as a pilot study on a subject area that has significant potential for data sharing. It also sees it as extremely important that, because of their overarching scope, the management and control of a CDM should be outside the framework of normal application system-development projects. Some formal structure should be set in place for delimiting ownership, responsibility and change control. A meta-model has been constructed by the CCTA which serves as a framework for determining the minimal component parts of a CDM.

21.4 CASE STUDY: STRATEGIC DATA PLANNING IN THE NATIONAL HEALTH SERVICE

The UK National Health Service (NHS) is one example of an organisation that has engaged in a number of strategic data planning initiatives. In this section we present a brief review of some of these initiatives and come to some conclusions as to the pragmatics of these exercises in the NHS (Beynon-Davies, 1994).

The idea of corporate data modelling in the NHS dates back to the late 1970s. In 1980, a body known as the NHS/DHSS (Department of Health and Social Security) Steering Group on Health Services Information was set up under the chairmanship of Mrs Edith Korner. The remit of this body was to identify a minimum data-set which could be used routinely for management purposes in every district health authority (DHA). The eventual aim was to produce management information from data usefully collected at the operational level. A number of working groups were set up, each with a substantial membership of NHS personnel. Each group was given the job of building a data model for a specific area of the health service: hospital clinical activity, patient transport services, health services manpower, activity in hospitals and the community, services in the community and health services finance.

All the working groups conducted their data modelling in a similar fashion:

- The data requirements of a DHA and its management team were identified
- The additional data needs of regional authorities and central government departments were identified
- Definitions for the data-items were agreed
- Initial recommendations were published and field-tested in four DHAs
- Consultation was encouraged through formal publications and informal seminars
- Recommendations were finalised, likely resource consequences were discussed and a timetable for implementation was generated

Over the following five years the Steering Group published six reports and one supplement which collectively became incorporated within a document known as the Korner Data Model Report (KDMR).

The KDMR contained entity–relationship (E–R) diagrams of each of the areas covered by the reports, together with detailed definitions of entities and attributes. Various changes to the KDMR have occurred since 1985, and new datasets have been added. In 1989, the post-Korner data model was published by a newly formed body known as the NHS Information Management Centre, and renamed the Minimum Data Set Model (MDSM).

The purpose of the MDSM was ‘to define for health authorities those data which they have all agreed to collect consistently. The model provided a tool for assessing ready-made systems, for creating new ones and for evaluating the impact of proposals for change.’



The MDSM documentation constituted two large volumes. Volume 1 included fifty or so detailed E-R diagrams divided into the following groups: general, hospital, community, manpower, finance, patient transport and estates. Associated with each uniquely coded diagram there was an overview which primarily described the attributes associated with each entity on a diagram. A brief activity analysis was also included against each entity model, detailing how some of the standard reports of the health service may be produced.

Volume 2 included a series of organised entity and attribute lists. This volume effectively constituted a data dictionary made up of the following components:

- An alphabetic entity list of some 500 entities in which each entity was referenced against its occurrence in each of the entity models
- An alphabetic attribute list of some 1,000 attributes in which attributes were referenced against their occurrence in entities
- An alphabetic entity list in which detailed textual definitions were given for each entity
- An alphabetic attribute list in which detailed formats were provided for each attribute

In 1991 version 2 of the MDSM was released. Whereas version 1 of the model constituted the minimum data-set required to be captured by a district health authority (a structural unit within the NHS), version 2 constituted the minimum data required to be captured by a health care provider, including the data to be passed on to purchasing organisations. These changes were clearly directed at the creation of an internal market within the NHS. Within a year of its publication the MDSM was described as being absorbed into, and superseded by, the NHS data dictionary. The data dictionary was described as recording the results of agreements and draft agreements about sets of data. It is meant to include data requirements from many more business areas than those included in the MDSM.

An E-R diagram simplified from one actually produced as part of the MDSM is illustrated in Figure 21.2.

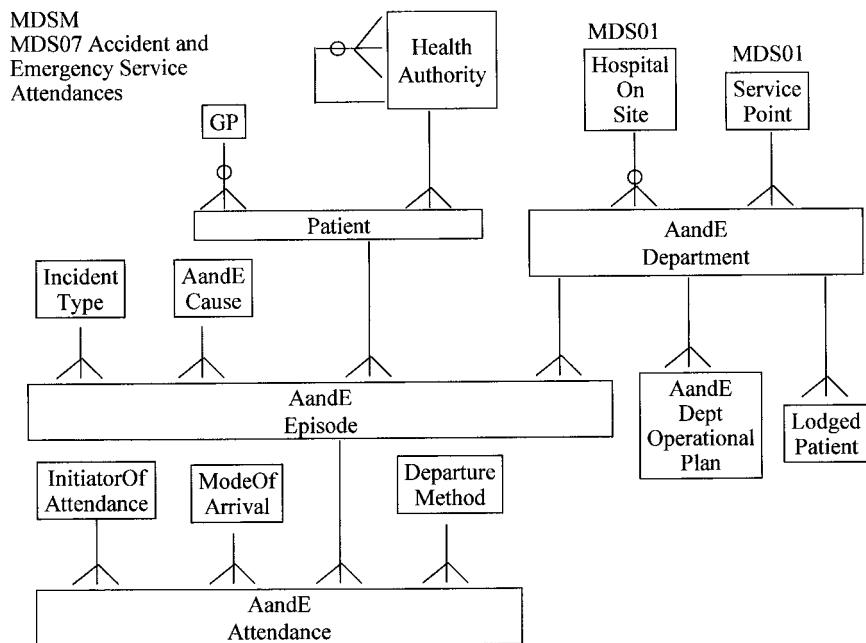
An extract from the associated data dictionary entry is illustrated in Figure 21.3.

21.5



THE PRAGMATICS OF STRATEGIC DATA PLANNING

Goodhue *et al.* (1992a) cite previous studies as questioning the value of strategic data planning (SDP) efforts, despite strong conceptual grounds for its use. They conducted an empirical study of strategic data planning in nine US companies. The authors identify five potential outcomes of SDP exercises from the literature:

**Figure 21.2** A data model from MDSM.

AandEAttendance (Accident and Emergency Attendance)

Health Authority Code

K-Primary Key
F-Foreign Key

Service Point Code

District Patient Number

Arrival Date

Arrival Time

First Attendance

Initiator of Attendance

F INITIATOR OF ATTENDANCE

Method of Departure

F DEPARTURE METHOD

Mode of Arrival

F MODE OF ARRIVAL

Start Date

F AandE Episode

Figure 21.3 A dictionary entry from MDSM.



- The implementation of integrated information systems
- The development of a corporate-wide data architecture
- The clear identification of information systems priorities
- The rethinking of an organisation's key processes
- The education and communication of data requirements throughout the organisation

Example → The strategic data planning conducted by the NHS is a good example of an organisation attempting to meet many of these objectives. The development of the MDSM was an initial attempt to standardise on an organisation-wide data architecture. The primary aim seems to have been to disseminate common data requirements throughout the organisation with the aim of at least unifying the processes of data capture throughout the NHS. As longer-term aims the MDSM was intended to facilitate better integration of information systems and to at least influence investment in information systems technology in the NHS:

Goodhue *et al.* (1992a) derive a number of lessons from their study of SDP efforts in the USA.

21.5.1 THE IMPLEMENTATION OF INTEGRATED INFORMATION SYSTEMS

Strategic data planning should be treated as a process of design as well as planning. For an SDP effort to be effective, data integration must be critical to strategic goals of the organisation as perceived by top management. Also, efforts to implement data integration need to balance global integration and local flexibility.

21.5.2 THE DEVELOPMENT OF A CORPORATE-WIDE DATA ARCHITECTURE

Goodhue *et al.* (1992a) come to the conclusion that it is not clear what the most appropriate form for a data architecture is. It is also not clear that an SDP is the most effective means to produce such architectures. An SDP project of too large a scope spells trouble. Also, to have an effect on data integration, architectures must be enforced and should be 'stolen', not invented.

21.5.3 THE CLEAR IDENTIFICATION OF INFORMATION SYSTEMS PRIORITIES

For a systems priorities goal, SDP does not narrow its scope fast enough before it begins the time-consuming process of modelling business functions and entities. In such a situation, creativity is swamped by the volume of detail.

21.5.4 THE EDUCATION AND COMMUNICATION OF DATA REQUIREMENTS THROUGHOUT THE ORGANISATION

In the projects Goodhue *et al.* (1992a) studied, new understanding seems difficult to communicate. They conclude that the cost of an SDP can probably not be justified by education and communication alone.

In a related research project, Goodhue *et al.* (1992b) considered the issue of data integration. Data integration is defined as the standardisation of data definitions and structures through the use of a common conceptual schema across a collection of data sources. This bears a resemblance to the type 2 CDM discussed in section 21.2. Data integration is implicitly assumed always to result in net benefits to the organisation. The authors question this view by using organisational information processing theory to construct a model of the costs as well as the benefits of data integration. This model is based on the central importance of mechanisms for handling uncertainty to organisations. Data integration is clearly one such mechanism for reducing uncertainty. However, a distinction is made between three major types of uncertainty:

- Uncertainty resulting from complex or non-routine subunit tasks
- Uncertainty resulting from unstable subunit task environments
- Uncertainty resulting from interdependence among subunits

They conclude that data integration can only satisfactorily address the last type of uncertainty. A distinction is also made between uncertainty and equivocality. Uncertainty is the absence of specific, needed information. Equivocality means that there are multiple, conflicting interpretations of a situation. Data integration can address aspects of uncertainty but not equivocality. Uncertainty may be reduced by a certain amount of information, equivocality can only be reduced by sufficiently rich information.

On the basis of these distinctions, the authors propose that the importance of data integration on the costs and benefits of information systems are mediated through three factors:

- *Increased ability to share information to address subunit interdependence.* All other things being equal, as the interdependence between subunits increases, the benefits of data integration will increase, and the amount of data integration in rational organisations should also increase
- *Reduced ability to meet unique subunit information requirements.* All other things being equal, as the differentiation between subunits increases, data integration will impose more and more compromise costs on local units; therefore the amount of data integration in rational firms should decrease. Also, all other things being equal, firms with increased data integration will experience greater bureaucratic delay in getting approval for changes affecting the data models used by individual subunits
- *Changes in the costs of information systems design and implementation.* All other things being equal, as the number and heterogeneity of subunit information



needs increase, the difficulty of arriving at acceptable design compromises increases, and the cost of the resulting design will increase more than linearly. Thus, rational firms will integrate less extensively when there are many heterogeneous subunits involved. As organisations face greater instability in their environments and their information requirements, the importance of this proposition will increase. In turbulent environments, firms with many heterogeneous subunits will be even less likely to integrate extensively, and firms with homogeneous subunits will be more likely to integrate extensively.

21.6**SUMMARY**

- Strategic data planning is the activity devoted to the construction of future/logical data models
- A corporate data model is a map of the data requirements of the whole or a substantial part of an organisation
- There are two main types of corporate data model: a type 1 CDM maps key features of the data terrain; a type 2 CDM provides a minimum data-set for key organisational functions
- Organisations generally conduct strategic data planning with one or more of the following objectives in mind: the implementation of integrated information systems; the development of a corporate-wide data architecture; the clear identification of information systems priorities; the rethinking of an organisation's key processes; and the education and communication of data requirements throughout the organisation
- Strategic data planning may be relevant only for certain forms of organisation. Generally speaking those organisations with homogeneous and interdependent subunits will benefit most from the data integration arising from strategic data planning

21.7**ACTIVITIES**

1. Examine an organisation known to you, such as a university. Determine whether it has a corporate data model. If it does, is it a type 1 or type 2 corporate data model?
2. If the organisation does not have a corporate data model, attempt to determine three or four of the most important entities to be included in such a model.
3. In terms of some organisation known to you, such as a university, use the data integration model produced by Goodhue *et al.* (1992b) to determine the importance of strategic data planning for the organisation.

21.8 REFERENCES

- Beynon-Davies, P. (1994). Information management in the British National Health Service: the pragmatics of strategic data planning. *International Journal of Information Management* **14**(2 (April)), 84–94.
- Beynon-Davies, P. (1997). The corporate data model: a study of organisational practice. *Journal of Systems and Information Technology* **1**(1 (March)), 47–63.
- Braithwaite, K.S. (1992). *Information Engineering: Analysis and Administration*. London, CRC Press.
- CCTA (1994). *Corporate Data Modelling*. London, HMSO.
- Goodhue, D.L., L.J. Kirsch, J.A. Quilliard and M.D. Wybo (1992a). Strategic data planning: lessons from the field. *MIS Quarterly* March: 11–34.
- Goodhue, D.L., M.D. Wybo and L.J. Kirsch (1992b). The impact of data integration on the costs and benefits of information systems. *MIS Quarterly* September: 293–311.
- Hudson, D.L. (1991). Approaches to corporate data model development at Sara Lee Corp. *Data Resource Management* **2**(3): 49–55.
- Sussman, R. (1993). Municipal GIS and the enterprise data model. *Int. Journal of Geographical Information Systems* **7**(4): 367–77.

CHAPTER 22 →

DATA ADMINISTRATION

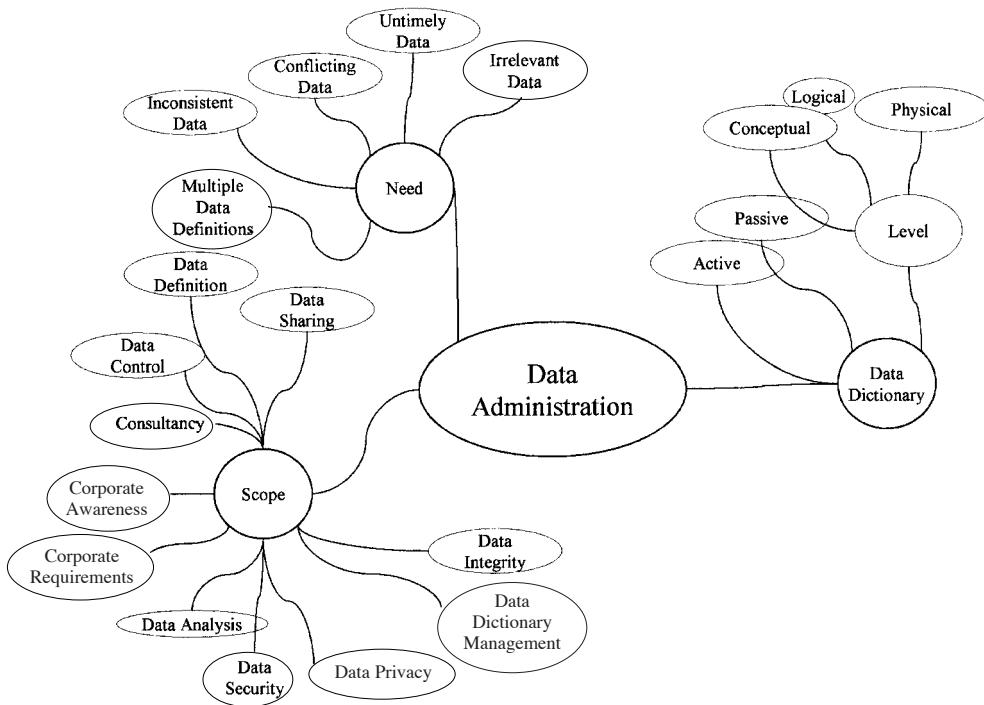
Tis hard to fight with anger, but the prudent man keeps it under control.

Democritus (460–370 BC)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define the concept of data administration
- Discuss the scope of the data administration function
- Relate the costs and benefits of having a data administration function
- Define the concept of a data dictionary
- Consider the issue of database security



22.1 INTRODUCTION

Database technology, because of its centrality in the modern informatics infrastructure, has stimulated the development of a large range of roles for servicing this technology. Two of the most commonplace roles are the data administrator and the database administrator.

Data administration is that function concerned with the management, planning and documentation of the data resource of some organisation (Gillenson, 1987). Data administration is concerned with the management of an organisation's meta-data, that is, data about data. It is a function which deals with the conceptual or business view of an organisation's data resource.

The key concept in the move towards data administration has been that data, like capital, personnel etc., should be treated as a manageable resource. In other words, data is a critical commodity in an organisation's attempt to compete in the open marketplace. In this sense, the data administration function is seen as a key part of the data and information management strategy of organisations (Beynon-Davies, 2002). The data administrator is also likely to be a key person in any technology concerned with data management such as data warehousing (Chapter 40).

Example Why is data such a critical resource for organisations? Consider the case of a university. Without data such as what students it has, what students are taking which modules and what grades have been achieved by students, a university is unable to operate effectively. However, consider the case in which:

- Different university departments or schools maintain their own distinctive collections of data with their own distinctive definitions for data-items
- Data is frequently missing or incomplete
- Data is frequently out-of-date
- There is incomplete knowledge among staff as to what data is collected, and where it is kept

In such situations staff may spend a substantial amount of its time resolving data problems. Such situations demonstrate the key need for an organisational function tasked with managing the data resource.

22.2 THE SCOPE OF DATA ADMINISTRATION

The scope of the data administration function will vary according to the organisation. There are however a core set of activities for which a data administration function is typically responsible:



- *Consultancy* – offering consultancy on all aspects related to an organisation's meta-data, particularly expertise in data analysis (Part 4)
- *Corporate awareness* – educating to increase awareness of the importance of data; also disseminating information on what data exists and for what purpose
- *Corporate requirements* – identifying corporate data requirements, particularly building a corporate data architecture which incorporates strategic planning
- *Data analysis* – coordinating the use of a standard data analysis methodology, and using such a methodology to develop business data models
- *Data control* – implementing standards for ensuring that access to data is controlled; also ensuring that suitable recovery procedures are in place
- *Data security* – ensuring that both technical and administrative controls are in place to protect against threats
- *Data definition* – implementing standards for the definition of data and controlling the medium for the recording and communication of such definitions
- *Data integrity* – implementing standard mechanisms for ensuring the integrity of an organisation's data; also documenting the rules for ensuring integrity
- *Data dictionary management* – promoting the use of a logical data dictionary (see below) and implementing standards for its control; also monitoring the use of and content of the data dictionary
- *Data privacy* – implementing procedures to ensure that the organisation complies with any legislation concerning national data regulation
- *Data sharing* – encouraging sharing of data across applications and promoting the idea of data that is independent of applications

22.3**THE NEED FOR DATA ADMINISTRATION**

The need for a data administration function arises from the problems of managing data within an organisation. A list of such problems is detailed below:

- A number of applications are developing within some organisations which use different definitions for the same data
- Data held by a number of diverse applications is inconsistent
- Decision-makers within an organisation receive conflicting data from different sources within that organisation

- Decision-makers receive data too late for it to be useful
- Decision-makers receive too much irrelevant data
- There are notable gaps in the data collected by an organisation
- Departments within an organisation have no clear idea why they collect certain data

In this view, data administration may be seen as an attempt to develop some order from the chaos of corporate information systems. However, data administration also involves planning the data required for future information systems. Data administrators would be key personnel in any strategic data planning effort (Chapter 21).

Example ➔ Consider the case of identifiers as data. A data administrator should ensure that identifiers for objects such as products, people, invoices and orders are designed to have a number of features:

- An identifier, as a matter of definition, should be uniquely associated with one, and only one, object in a database. Hence, every object in a database, such as an instance of a product, should have an identifier
- An identifier should be assigned immediately on creation of an object
- An identifier should not contain any details about the object it identifies. It should serve the sole function of identifying an object. The reason for this is that so-called non-mnemonic identifiers maintain stability over time

Consider the case where a company uses a three-digit code to identify its products. The first digit is used to indicate the warehouse where the product is stored. Now suppose the company decides to change its warehousing practice and moves all products of a particular type from one warehouse to another. This will necessitate changing all the product codes for the products moved.

A natural consequence of the discussion above is that it is important to administer and control the assignment and use of identifiers in organisations. It is usually not good practice to rely on identifiers supplied by external agencies. For example, suppose an organisation uses the delivery advice number from its supplier to identify different deliveries. If the supplier inadvertently sends two separate deliveries with the same advice number then the organisation's internal information systems are likely to suffer.

22.4 THE DIFFICULTIES OF DATA ADMINISTRATION

It is rare for a data administration function to be set up in an environment in which no applications systems yet exist. Most data administration departments



are set up as a response to some of the problems outlined above. In such a situation it is usually a significant task to reconcile data collected from diverse areas of the organisation. Many parts of the organisation will openly disagree about data requirements. Data administration is hence frequently a function which is shaped by the sociology, politics and economics of organisations.

22.5

THE PLACE OF THE DATA ADMINISTRATION FUNCTION

The concept of data administration crosses departmental and indeed organisational boundaries. The relationship between data administration and the structure of organisations is therefore significant.

In many organisations the data administration function has arisen naturally from the realms of the information systems department. Data administration then becomes another arm of information systems. However, because of its cross-departmental role, there is some argument for placing the data administration function outside the information systems department.

Many have argued that wherever it is placed, the data administration function cannot work effectively unless it has access to the management level of organisations. Some have even suggested that the data administrator should have a seat at executive board level.

22.6

DATA DICTIONARIES

The main tool of the database or data administrator is the data dictionary. In this section we first define what we mean by the term data dictionary, then we distinguish between different types of data dictionary.

22.6.1 WHAT IS A DATA DICTIONARY?

A data dictionary is a means for recording the meta-data of some organisation (Navathe and Kerschberg, 1986); that is, data about data. What constitutes meta-data is obviously determined by the stage of database development as discussed in Chapter 14. In this sense, we might usefully define three levels of data dictionary:

- Conceptual data dictionaries record meta-data at a very high level of abstraction. In Chapter 16 we discussed the use of a data dictionary to record the details of an entity–relationship model
- Logical data dictionaries are used to record data requirements independently of how these requirements are to be met. Logical meta-data is however at a slightly lower level of abstraction than conceptual meta-data
- Physical data dictionaries are used to record data structures; that is, they record meta-data relating to actual database or file structures. The system tables at the heart of a relational DBMS comprise a physical data dictionary

In terms of meta-data, therefore, we conventionally mean both data resources and data requirements. A data dictionary is a mechanism for recording the data resources and requirements of some organisation. This means that a data dictionary cannot be considered solely as a tool for the analysis and design of database systems. It is also an important implementation tool and an important function of what we might call the corporate information architecture. Conceptual and logical data dictionaries are usually the realm of the data administrator. Physical data dictionaries are the realm of the database administrator (Chapter 23).

22.6.2 ACTIVE AND PASSIVE DATA DICTIONARIES

The classic data dictionary is a passive repository for meta-data. In other words, the first data dictionaries were built as systems external to a database and DBMS. They were systems built to record the complexities of a given database structure for use by application development staff and database administrators (DBAs).

If we equate the term data dictionary with the set of system tables in a relational database, then a data dictionary in a relational system is an active repository. It is an inherent, internal part of the database system designed to buffer the user from the base tables.

As we have seen, many DBMS are beginning to extend the functionality of such active data dictionaries, particularly in the area of integrity management.

Traditionally, integrity issues have been external to the database system. Integrity has primarily been the province of application systems written in languages such as COBOL or C which interact with the database. Programs are written in such languages to validate data entered into the database and ensure that any suitable propagation of results occurs.

Many have argued however that the logical place for integrity is within the domain of the data dictionary under the control of the DBMS. The argument is that integrity cannot be divorced from the underlying database. Two or more application systems interacting with one database may enforce integrity differently or inconsistently. Hence, there should be only one central resort for monitoring integrity. Integrity should be the responsibility of the DBA. Mechanisms should be available therefore for the DBA to define and enforce integrity via the DBMS.

22.7 DATABASE SECURITY

Database security is the process of protecting a database from external threats. Database security is of primary concern to the data administrator because of the key value that data holds for modern organisations.

Threats include possibilities of theft and fraud, loss of confidentiality, loss of privacy, loss of integrity and loss of availability. Examples of each type of threat are given below:



- *Theft and fraud* – examples here include a person, not authorised to do so, updating corporate data with the aim of defrauding his or her employer. Another example is that of a hacker making an illegal entry into a database system and extracting corporate data without permission
- *Confidentiality* – an unauthorised person viewing information on confidential corporate policies and disclosing it to outside agencies
- *Privacy* – an unauthorised person viewing data on persons held by an organisation
- *Integrity* – a software virus causing corruption of data or loss of integrity through software or hardware failure
- *Availability* – a database system becoming unavailable because of natural disasters such as fire and flood, or disasters initiated by humans such as bomb attacks

The data administrator tasked with managing database security will need to plan a number of computer-based and non-computer-based measures for countering threats such as those listed above. Computer-based measures include:

- Implementing suitable authorisation in relation to an operating system on which the database system runs
- Implementing authorisation strategies that grant privileges to certain users and groups to access certain database objects via the chosen DBMS
- Setting up views into the database and granting access to such views to users and groups
- Taking regular backups of the whole of or part of a database
- Maintaining a log of all changes made to the database
- Utilising suitable strategies for encrypting sensitive data

The implementation of many of these computer-based strategies may be delegated to the DBA. Non-computer-based measures include:

- Establishing a security policy and plan
- Putting suitable personnel controls in place, such as separation of duties
- Positioning computer hardware in secure environments through physical access controls
- Securing copies of data and software in off-site, fireproof storage

The data administrator will need to liaise with other staff within the organisation, such as people managing the estate of the organisation, to execute these security strategies effectively.

22.8 CASE STUDY: DATA ADMINISTRATION IN A UNIVERSITY SETTING

Clearly, a university can be considered to be made up of a number of interdependent human activity systems (Chapter 4). A systems model of a university is likely to consist of at least three core processes. These are all organisational processes in that they are cross-organisational, have a defined set of customers, and link to the core competencies of the organisation.

- *Teaching*. It is important for a university to achieve suitable levels of enrolment by students on courses and to progress students through various undergraduate and postgraduate programmes
- *Research*. It is important for a university to achieve satisfactory levels of research funding from government and industry
- *Consultancy*. It is important for universities to achieve satisfactory levels of income from consultancy projects

Universities need to maintain accurate data about students, staff, courses, modules and finance in order to work effectively and efficiently. Such data supports the key organisational processes of teaching, research and consultancy:

- The teaching process needs accurate data about students, courses, modules, assessments and fees
- The research process needs accurate data about research outputs such as publications and research inputs such as grant income
- The consultancy process needs accurate data about projects and consultancy income

Data administration in such a university will be concerned with ensuring that data is not collected, stored and manipulated in a piecemeal manner by administrative and academic departments; in order to ensure this it is likely to enforce a series of common data definitions for such data across the organisation. This will enable sharing of data across the organisation.

The data administrator will also ensure that access to such data is regulated and that the privacy of sensitive data (such as that held about students) is preserved. He or she will also be critically involved in monitoring current data requirements across the organisation and constructing plans for data to fulfil future organisational strategy.

22.9 SUMMARY

- Database technology, because of its centrality in the modern IS architecture, has stimulated the development of a large range of roles for servicing this technology. Two of the most commonplace roles are the data administrator and the database administrator



- The data administrator is a high-level, corporate function
- As far as development is concerned, the data administrator will be involved in the analysis and design of a database system. Data administrators are also particularly involved in the management of the organisational data resource
- The main tool of the data administrator is the data dictionary, particularly conceptual and logical data dictionaries. A data dictionary is a means for recording the meta-data of some organisation
- Database security is a concern of the data administrator in association with the database administrator. Database security involves protecting a database from external threat by using computer-based and non-computer-based controls

22.10

ACTIVITIES

1. In terms of some organisation known to you, examine whether it has a data administration function. If it does, identify the functions undertaken by the data administration function. If it does not, examine the case for data administration.
2. Examine a DBMS known to you. See if you can determine the structure of the physical data dictionary used by the DBMS.

22.11

REFERENCES

- Beynon-Davies, P. (2002). *Information Systems: An Introduction to Informatics in Organisations*. Basingstoke, Palgrave (formerly Macmillan).
- Gillenson, M.L. (1987). The duality of database structures and design techniques. *Comm. of ACM* **30**(12): 1056–65.
- Navathe, S.B. and L. Kerschberg (1986). Role of data dictionaries in information resource management. *Information and Management* **10**: 21–46.

CHAPTER 23 →

DATABASE ADMINISTRATION

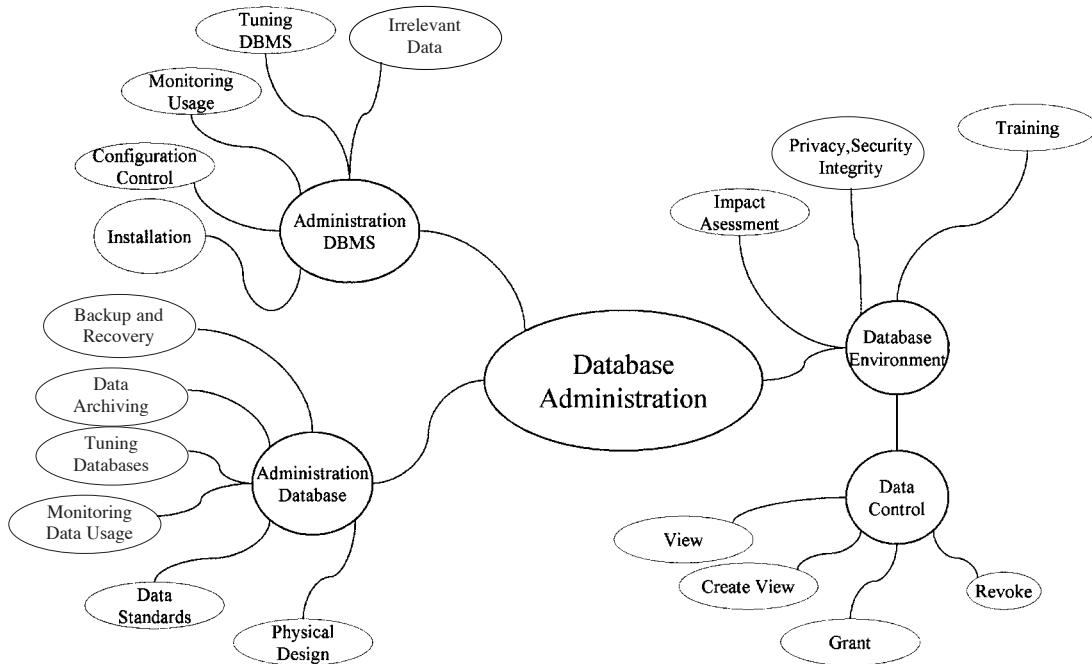
Control thy passions, lest they take vengeance on thee.

Epictetus (AD 55–AD 135)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define the concept of database administration
- Discuss the key features of the database administrator
- Discuss the key administration function of data control



**23.1**

The database administrator (DBA) is responsible for the technical implementation of database systems, managing the database systems currently in use, and setting and enforcing policies for their use. Whereas the data administrator (Chapter 22) works primarily at the conceptual level of business data, the DBA works primarily at the physical level. The place where the data administrator and the DBA meet is at the logical level. Both the data administrator and DBA must be involved in the system-independent specification and management of data.

The need for a specialist DBA function varies depending on the size of the database system being administered. In terms of a small desktop database system, the main user of the database will probably perform all DBA tasks such as taking regular backups of data. However, when a database is being used by many people and the volume of data is significant, the need for a person or persons who specialise in administration functions becomes apparent.

23.2

In many organisations, particularly small organisations, the DBA will be expected to undertake many data administration tasks. However, in general, the DBA can be said to have the following core responsibilities: administration of the database, administration of the DBMS and administration of the database environment.

23.2.1 ADMINISTRATION OF THE DATABASE

The DBA would normally be expected to engage in the following key activities in relation to administering a database or a series of databases:

- *Physical design* – whereas the data administrator will be concerned with the conceptual and logical design of database systems, the DBA will be concerned with the physical design and implementation of databases
- *Data standards and documentation* – ensuring that physical data is documented in a standard way such that multiple applications and end-users can access the data effectively
- *Monitoring data usage and tuning database structures* – monitoring live running against a database and modifying the schema or access mechanisms to increase the performance of such systems
- *Data archiving* – establishing a strategy for archiving of ‘dead’ data
- *Data backup and recovery* – establishing a procedure for backing-up data and recovering data in the event of hardware or software failure

23.2.2 ADMINISTRATION OF THE DBMS

The DBA would normally be expected to engage in the following key activities in relation to administering a DBMS:

- *Installation* – taking key responsibility for installing DBMS or DBMS components
- *Configuration control* – enforcing policies and procedures for managing updates and changes to the software of the database system
- *Monitoring DBMS usage and tuning DBMS* – monitoring live running of database systems and tailoring elements of the DBMS structure to ensure the effective performance of such systems

23.2.3 ADMINISTRATION OF THE DATABASE ENVIRONMENT

By administering the database environment, we mean monitoring and controlling the access to the database and DBMS by users and application systems. Activities in this area include:

- *Data control* – establishing user groups, assigning passwords, granting access to DBMS facilities, granting access to databases
- *Impact assessment* – assessing the impact of any changes in the use of data held within database systems
- *Privacy, security and integrity* – ensuring that the strategies laid down by data administration for data integrity, security and privacy are adhered to at the physical level
- *Training* – holding responsibility for the education and training of users in the principles and policies of database use

23.3 DATA CONTROL

One of the primary functions of the DBA is data control.

In any multi-user system, some person or persons have to be given responsibility for allocating resources to the community of users and monitoring the usage of such resources. In database systems, two resources are of pre-eminent importance: data and DBMS functions. Data control is the activity which concerns itself with allocating access to data and allocating access to facilities for manipulating data. Data control is normally the responsibility of the DBA.

In this section we shall examine some of the fundamental aspects of data control in relational database systems. First, we shall examine the concept of a relational view in SQL. Second, we shall discuss how DBAs manipulate the view concept in order to allocate access to data. Third, we shall examine the DBA's ability to allocate access to DBMS functions such as data definition.

23.4 VIEWS

A view is a virtual table. It has no concrete reality in the sense that no data is actually stored in a view. A view is simply a window into a database in that it presents to the user a particular perspective on the world represented by the database.

Example → For instance, suppose we have a base table of student data, a sample of which is given below:

Students							
student No	student Name	sex	Date of Birth	studentTerm Addr	student Tel	course Code	
1234	Paul S	M	01/03/1980	14, Hillside Tce	432568	CSD	
1235	Patel J	M	01/05/1981	28, Park Place	235634	CSD	
2367	Singh B	F	01/01/1974	10a, Pleasant Flats	345623	BSD	
3245	Conran J	M	01/08/1980	Rm 123, University Towers	NULL	CSD	
2333	Jones L	F	01/02/1981	9 Redhill St	222333	BSD	

We can identify three main interdependent uses for views in a database system: to simplify, to perform some regular function and to provide security.

23.4.1 SIMPLIFICATION

Suppose the students table above is defined for an entire university. Each department or school within the university however only wishes to see data relevant to its own courses. A number of views might then be created to simplify the data that each department or school sees.

Example → For instance, we might have two views established on the table above, one for the students of the school of computing and one for the students of the business school.

Computing Students					
student No	student Name	sex	studentTerm Addr	student Tel	
1234	Paul S	M	14, Hillside Tce	432568	
1235	Patel J	M	28, Park Place	235634	
3245	Conran J	M	Rm 123, University Towers	NULL	

Business Students				
student No	student Name	sex	studentTermAddr	studentTel
2367	Singh B	F	10a, Pleasant Flats	345623
2333	Jones L	F	9 Redhill St	222333

23.4.2 FUNCTIONALITY

There is an implicit edict in the relational model: store as little as possible. One consequence of this is that any data that can be derived from other data in the database is not normally stored.

Example → If, for instance, there is a requirement to regularly display the ages of students, we would not store both a student's age and his or her birthdate. A student's age can be derived from a person's birthdate. The view mechanism is the conventional means for performing such computations and displaying them to the user. For example, in the view below we display the sex and age profile of students. Age is a derived attribute in this view.

Students			
student No	student Name	sex	age
1234	Paul S	M	19
1235	Patel J	M	18
2367	Singh B	F	25
3245	Conran J	M	18
2333	Jones L	F	19

23.4.3 SECURITY

In a university we might want to restrict each department or school only to be able to update the data relating to its own students. A view declared on each department or school and attached to a range of user privileges can ensure this. Hence the two views above, *Computing Students* and *Business Students*, might be secured for access only by administrative staff of the school of computing and the business school respectively.

23.5 UPDATING THROUGH VIEWS

Retrieving information through views is a straightforward process. It involves merging the original query on the view with the view definition itself. File maintenance operations such as insertion, deletion and update are however more complex to accomplish via views, and hence deserve more consideration.

The main point is that not all views are updatable.

Example Consider for instance, two views declared on the students table, S1 and S2. S1 produces a view of all the columns in the base table but only for computing students; S2 is a vertical fragment of the base table:

S1

student No	student Name	sex	Date of Birth	studentTerm Addr	student Tel	course Code
1234	Paul S	M	01/03/1980	14, Hillside Tce	432568	CSD
1235	Patel J	M	01/05/1981	28, Park Place	235634	CSD
3245	Conran J	M	01/08/1980	Rm 123, University Towers		CSD

S2

studentName	Sex	Course Code
Paul S	M	CSD
Patel J	M	CSD
Singh B	F	BSD
Conran J	M	CSD
Jones L	F	BSD

In the case of S1, for instance, we can insert a new row into the view, delete an existing record from a view or update an existing field in the view. In the case of S2, problems arise. For example, suppose we wished to insert a record for a new student, Balshi J, M, BSD. We would have to insert the record *null, Balshi J, M, null, null, BSD* into the underlying students table. This attempt will fail since studentNo (the table's primary key) must not be null.

On closer examination, the main difference between S1 and S2 lies in the fact that S1 contains the primary key column of the base table and S2 does not.

This means that updates to S1 unambiguously update rows in a view. Updates to S2 are frequently ambiguous. S1 is hence said to be an updatable view whereas S2 is a non-updatable view.

23.6 THE CREATE VIEW STATEMENT

As stated previously, a view is a virtual table. It has no concrete reality in the sense that no data is actually stored in a view. A view is simply a window into a database in that it presents to the user a particular perspective on the world represented by the database. A view in SQL is a packaged *select* statement. The syntax is as follows:

```
CREATE VIEW <view name>
AS <select statement>
```

Example → The following two examples create the S1 and S2 views discussed in section 23.4.1:

```
CREATE VIEW S1 AS
SELECT *
FROM Students
WHERE courseCode = 'CSD'

CREATE VIEW S2 AS
SELECT studentName, sex, courseCode
FROM Students
```

When a view has been created, it can be queried in the same manner as a base table. For instance:

```
SELECT *
FROM S1
WHERE sex = 'F'
```

23.6.1 UPDATING THROUGH VIEWS IN SQL

In section 23.5 we discussed how certain views are not updatable and how certain other views are updatable. In this context, C. Date has made the distinction between three types of views (Date, 2000):

- Views which are updatable in theory and in practice
- Views which are updatable in theory, but not yet in practice
- Views which do not appear to be updatable in theory and therefore cannot be updatable in practice

What then defines the first type of view? As far as most SQL-based systems are concerned, the following conditions have to be satisfied:



- The *select* clause in a view cannot use a distinct qualifier or any function definition
- The *from* clause must specify just one table. No joins are allowed
- The *where* clause cannot contain a correlated subquery
- The *group by* and *order by* clauses are not allowed
- No *union* operator is permitted

23.7

DATA CONTROL COMMANDS FOR DBAs

In any database system, the DBA needs to be able to do three main things:

- Prevent would-be users from logging-on to the database
- Allocate access to specific parts of the database to specific users
- Allocate access to specific operations to specific users

In this section we shall examine SQL's capability for handling the second of these needs. Because the first and third functions are non-standard across DBMS we shall resort to examining the facilities available under the ORACLE RDBMS.

23.7.1 GRANT AND REVOKE

SQL provides two commands for controlling access to data. One command gives access privileges to users. The other command takes such privileges away.

To give a user access to a view or a table we use the grant command:

```
GRANT [ALL : SELECT : INSERT : UPDATE : DELETE ]
ON [<table name> : <view name>]
TO <user name>
```

Example ➔ For instance:

```
GRANT INSERT ON Modules TO pbd
GRANT SELECT ON Lecturers TO pbd
```

To take existing privileges away from a user we use the revoke command.

```
REVOKE [ALL : SELECT : INSERT : UPDATE : DELETE ]
ON [<table name> : <view name>]
FROM <user name>
```

Example ➔ For instance:

```
REVOKE INSERT ON Modules FROM pbd  
REVOKE SELECT ON Lecturers FROM pbd
```

23.7.2 GRANTING ACCESS VIA VIEWS

One of the ways of enforcing control in relational database systems is via the view concept.

Example ➔ Let us suppose, for instance, that we wish to give R. Evans the ability to look at and update the records of Lecturers in his department – Computer Studies. We do not however wish to give R. Evans the ability to delete Lecturer records or insert new Lecturer records. To enact these organisation rules we create a view:

```
CREATE VIEW Evans AS  
    SELECT *  
    FROM Lecturers  
    WHERE deptName =  
        (SELECT deptName  
        FROM Lecturers  
        WHERE staffName = 'Evans R')
```

Note that the employee record for R. Evans would be included in this view. If we now provide select and update access for R. Evans as below:

```
GRANT SELECT, UPDATE  
ON research  
TO REvans
```

this particular person will be able to change his own salary! There are a number of ways we can prevent this from happening by modifying the view definition. One possible solution is given below:

```
CREATE VIEW Evans AS  
    SELECT *  
    FROM Lecturers  
    WHERE deptName =  
        (SELECT deptName  
        FROM Lecturers  
        WHERE staffName = 'Evans R')  
    AND staffName <> 'Evans R'
```



23.7.3 DBA PRIVILEGES IN ORACLE

In the above discussion we have considered how SQL permits the DBA to specify access to parts of the database and how it permits the DBA to associate these allocations with particular users. However, at present SQL does not contain a definition of how users are declared to the database system in the first place, and also how such users are given access to various DBMS facilities. Assigning users and giving such users access privileges are two DBA functions which vary from product to product. To give the reader a flavour of the type of mechanisms available, we discuss here those available under the ORACLE RDBMS (Versions 6 through 8).

23.7.4 CONNECT, RESOURCE AND DBA

Most RDBMS differ in the levels of facilities or system privileges offered and what they are called. In this section we illustrate what is available under the ORACLE DBMS.

When a company receives a new ORACLE system it comes automatically installed with two 'super' users. The first thing the DBA does is change the default passwords for these user names. The second thing he or she does is start enrolling users into the system.

To declare an ORACLE user the DBA must normally supply three pieces of information: a distinct user name, a password and the level of privilege granted to the user.

There are three classes of ORACLE user: *connect*, *resource* and *dba*. A connect user is able to look at other users' data only if allowed by other users, perform data manipulation tasks specified by the DBA, and create views. Resource privilege allows the user to create database tables and indexes and grant other users access to these tables and indexes. Dba privilege is normally given to a chosen few. The superusers discussed above have dba privileges. Such privileges permit access to any user's data, and allow the granting and revoking of access privileges to any user in the database.

When a new user is enrolled into the system the database administrator grants the user one or more of these privileges using the command below:

```
GRANT {CONNECT : RESOURCE : DBA}
TO <username>
[IDENTIFIED BY <password>]
```

Example → For instance:

```
GRANT CONNECT, RESOURCE
TO pbd
IDENTIFIED BY crs
```

In a similar manner to granting database access, DBMS privileges can be revoked as below:

```
REVOKE {CONNECT : RESOURCE : DBA}
FROM <username>
```

Example ➔ For instance:

```
REVOKE CONNECT, RESOURCE
FROM pbd
```

23.8 CASE STUDY: DATA CONTROL OF THE ACADEMIC DATABASE

Assume we appoint a DBA to control the academic database below:

```
Courses(courseCode, courseName, deptName, courseLeader)
CourseEnrolment(courseNo, studentNo, enrolmentDate, feePaid)
Modules(moduleNo, moduleName, level, courseCode)
ModuleEnrolment(moduleNo, studentNo, enrolmentDate)
ModuleAssessment(moduleNo, studentNo, assessmentNo, grade)
Lecturers(staffNo, staffName, status, salary)
Students(studentNo, studentName, gender, studentTermAddress, studentTermTelNo,
studentHomeAddress, studentHomeTelNo)
Teaches(lecturerNo, moduleNo)
Prerequisites(moduleNo, moduleNo)
```

The DBA first has to establish the major user groups for this database. Five user groups are established – personnel administrators, teaching administrators, registry staff, academic staff and university management. The DBA establishes the following data control requirements for each such group:

- *Personnel administrators.* Lecturers [ALL]
- *Teaching administrators.* Courses, Modules, Prerequisites, Teaches, ModuleAssessment [ALL]
- *Registry staff.* Students, CourseEnrolment, ModuleEnrolment [ALL]
- *Academic staff.* View of Lecturer's Modules [SELECT]; View of Students on Lecturer's Module [SELECT]
- *Managers.* Entire database [SELECT]

23.9 SUMMARY

- The database administrator is a low-level, technical function
- In terms of development, the database administrator will be involved in the physical design of database systems. On the data management side, the database administrator will be particularly concerned with the issue of data control



- Data control is the activity devoted to controlling access to data and DBMS functions
- In relational database systems, the data aspect of data control revolves around the concept of a view. A view is a virtual table. Views can be used to create 'windows' into a database which can be allocated to specific users of the database
- The DBMS-side of data control varies from product to product. Most RDBMS however have facilities for assigning user names and passwords to users, and defining the overall privileges available to users
- ISO SQL has a limited range of DBA functions, particularly focused on granting access to data and revoking access to data. Most DBMS also have a range of non-standard functions for declaring users and passwords, fine-tuning database sizing, monitoring the performance of a database and fragmenting databases

23.10

ACTIVITIES

1. Examine an organisation known to you. Determine whether it employs database administrators. Examine the activities conducted by the DBAs in this organisation and compare them against the activities identified in this chapter.
2. Examine a DBMS known to you. Determine the mechanisms for enforcing data and system privileges in this DBMS.
3. Create the views discussed in section 23.4.2 using SQL.
4. Grant appropriate data access on these views using SQL.

23.11

REFERENCE

Date, C.J. (2000). *An Introduction to Database Systems*. Reading, MA, Addison-Wesley.

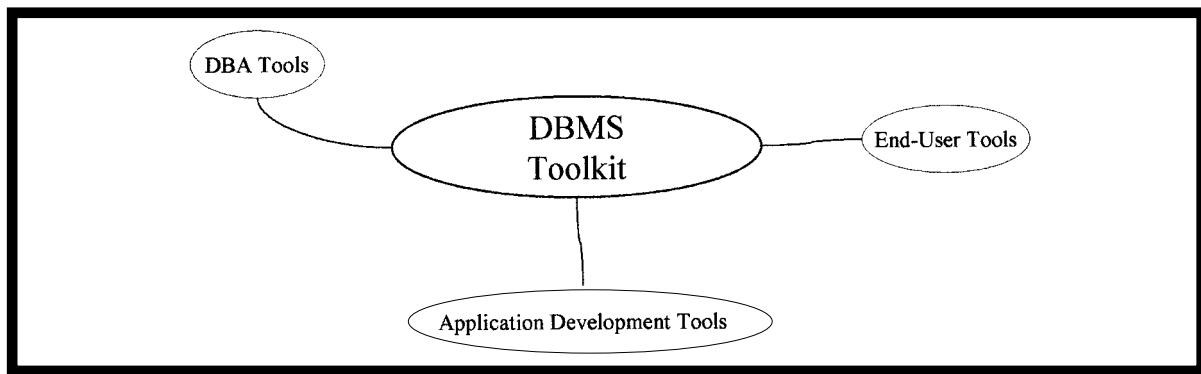


PART 6

DATABASE MANAGEMENT SYSTEMS (DBMS) – TOOLKIT

Engineering is not merely knowing and being knowledgeable, like a walking encyclopedia; engineering is not merely analysis; engineering is not merely the possession of the capacity to get elegant solutions to non-existent engineering problems; engineering is practising the art of the organized forcing of technological change . . . Engineers operate at the interface between science and society . . .

Dean Gordon Brown





This part discusses the panoply of toolkit products available for use in association with database systems. We devote a chapter each to three different collections of tools used in association with a DBMS: end-user tools, application development tools and database administration tools. These three types are clearly formed on the basis of three distinct user groups for database systems: end-users, application developers and database administrators. Figure P6.1 illustrates the relationship between toolkit, kernel and interface with a sample of tools in each area.

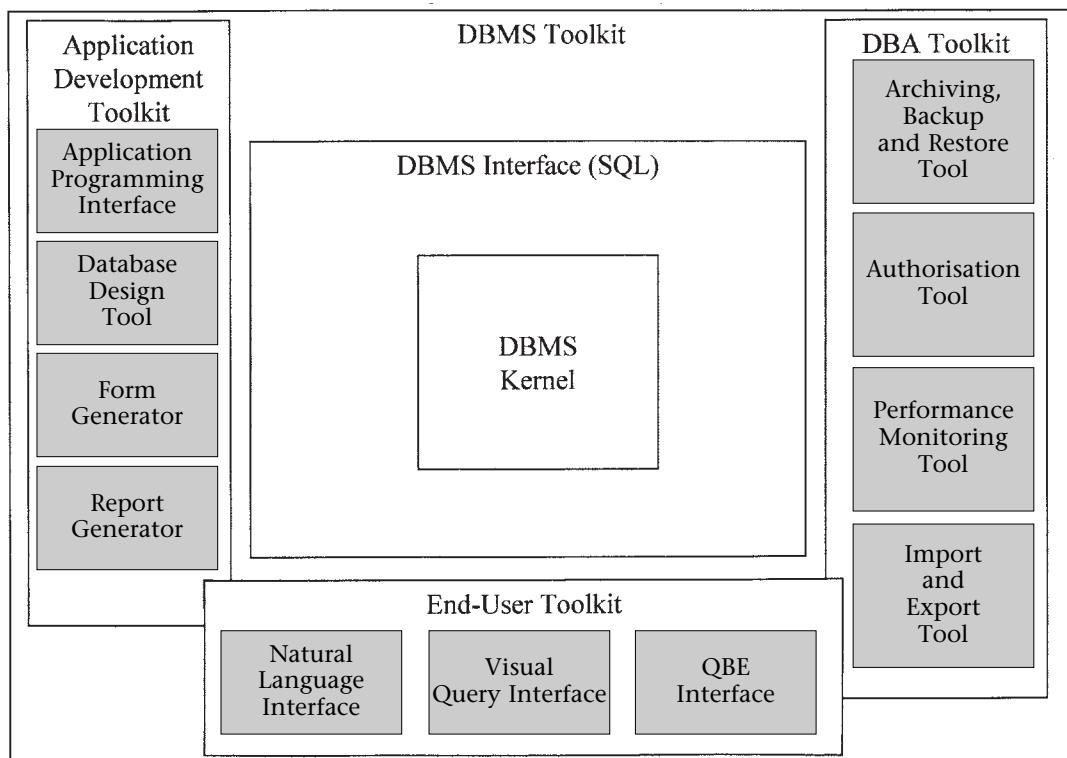


Figure P6.1 Kernel, interface and toolkit.

CHAPTER 24

DBMS-TOOLKIT – END-USER TOOLS

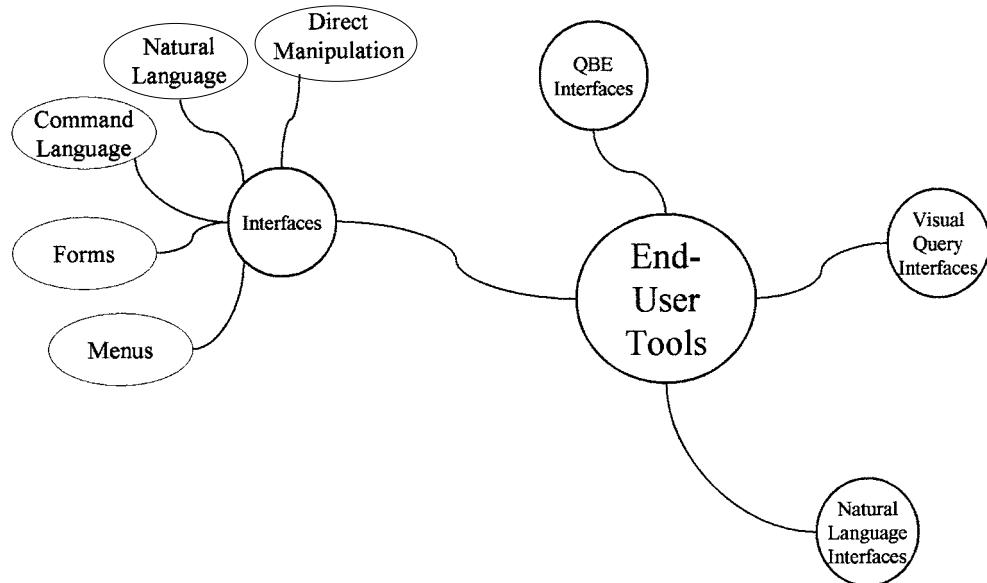
Why is it drug addicts and computer aficionados are both called users?

Clifford Stoll

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define the concept of a user interface
- Review the major forms of user interface
- Discuss the major end-user interfaces available in contemporary DBMS



24.1 INTRODUCTION

To review, modern-day relational DBMS can be considered as having three parts: a kernel, an interface and a toolkit. The kernel comprises the core DBMS functions such as storage and concurrency control. The interface comprises generally some database sublanguage such as SQL. Around this standard interface most vendors offer a range of additional software tools for accessing data, administering database systems and producing information systems. We call this collection the DBMS toolkit. Thus the concept of DBMS originally discussed in Chapter 3 is extending further outwards to encompass more and more areas of application building and use. In this and the subsequent two chapters we highlight the features and functionality available in three areas of such a toolkit: end-user tools, application development tools and database administration tools.

In this chapter we concentrate on tools for end-user use. By the end-user we mean the normal organisational user who needs access to the data in databases in order to perform his or her day-to-day activity. Generally we may make a distinction between naive and sophisticated end-users. Sophisticated users may be able to write their own SQL and hence will need an SQL interface of a type discussed in Chapter 13. Naive users may use a QBE, visual query or natural language interface.

24.2 THE INTERFACE

An essential function of most ICT systems is to interact with users. Even a batch process system has to have some mechanism for the user to launch and complete a process. No matter how well-built an information system is, its effectiveness is largely affected by its user interface. Hence, the utility of an information system is heavily affected by its usability.

A user interface can be seen as a collection of dialogues between the user and the ICT system. Each dialogue is made up of a series of messages between the user and the ICT system. It is useful to distinguish between three major aspects of such a dialogue:

- *Content* – this refers to the actual messages travelling between the user and the system
- *Control* – this refers to the way in which the user moves between aspects of one dialogue or from one dialogue to another
- *Format* – this refers to the actual layout of messages and data on input and output devices

We can think of dialogues in terms of the four areas of semiotics described in Chapter 2: pragmatics, semantics, syntactics and empirics. Format and control are largely aspects of the syntax and empirics of signs: the way in which signs are stored, transmitted and represented on the devices of some computer



system. Some aspects of control and certainly the matter of content impinge on the pragmatics and semantics of signs: how signs are given meaning within various layers of context.

24.2.1 TYPES OF INTERFACE

Schneiderman (1998) identifies five broad categories of interface:

- *Menus*. This interface consists of a displayed list of choices. The user selects an item from the list either by pressing some key combination or moving a cursor to the relevant choice, and selecting by a mouse click or by typing in some value and depressing the enter key. Menus are frequently used in database systems as a means of launching packaged queries
- *Forms*. These forms of interface are used for data entry and data retrieval purposes. A form is simply a set of fields laid out on a screen, or more readily these days within the context of a window. Each data entry field is normally labelled appropriately. Forms usually have some header area and some area for the display of error messages or prompts. Many modern application development tools produce default data entry forms from database schemas automatically
- *Command language*. In this form of interface the user enters statements in some formal language in order to carry out functions. Historically, command level interfaces were the first type of on-line interface. Operating systems such as MS-DOS and UNIX have command language interfaces. Many DBMS also offer command-level interfaces to SQL
- *Natural language*. So-called natural language interfaces are not truly natural language in the sense that they will not generally accept everyday English input as character strings. They are more accurately described as being restricted language interfaces in that a statement such as *Give me all the salaries of my employees* might be acceptable, whereas the statement *List my employee's salaries* may not. Such interfaces have achieved some success as front-ends to databases, particularly when employed with voice recognition technology
- *Direct manipulation*. This type of interface is generally associated with icon-based, windows environments. They are referred to as direct manipulation because the user causes events to happen by manipulating graphic objects via mouse-based actions. Such interfaces have now become dominant in most areas of information systems

To these five types of interface we may add two further types which are becoming prominent in contemporary systems:

- *Multi-media interface*. This is a direct extension of the direct manipulation interface described above in that it employs the same mechanisms for controlling input. The main difference is that rather than having simple menus and data entry forms, a full range of media types is used to build the



interface. Many interfaces on the World-Wide-Web (Chapter 43), for instance, now utilise images. Some companies have begun experimenting with the use of animation, video and audio within their interfaces to information systems

- *Virtual reality interface.* In some applications, such as those used in simulation, it is important for the user to experience a similar environment in terms of an interface to that available in the real world. Hence, for instance, aircraft simulators have been used for a number of years to train commercial and military pilots. Aspects of this form of interface have now become feasible on the desktop and some organisations are beginning to experiment with the use of such interfaces in the information systems domain. One example here might be an interface which emulates the physical arrangement of publications in a library. Users of such a system are able to select a given publication by traversing through the virtual space represented in the system

24.3

QUERY BY EXAMPLE INTERFACES

The idea of a query by example or QBE interface is generally attributed to the early work conducted by Moshe Zloof at IBM (Zloof, 1975). The basic idea is to provide a menu or forms-based interface to a database that is non-procedural in nature. QBE interfaces are primarily designed for use by the naive end-user of database systems.

Normally, the user first begins a dialogue with a QBE interface by selecting from a menu of tables. A table skeleton is then displayed for each of the tables selected. The user then fills in the blanks on the skeletons using constants or predefined commands. Sometimes function keys are used.

Examples

Figure 24.1 illustrates two examples of using an original QBE interface. In the first example the user has called up a skeleton for the Lecturers table. The 'P.' in the staffNo and staffName cells indicates that the user wants these attributes printed. The string 'Computer Studies' in the department cell indicates that the user wants a match on all lecturers in this particular department. This screen effectively implements the SQL statement:

```
SELECT staffNo, staffName
FROM Lecturers
WHERE department = 'Computer Studies'
```

The second example illustrates how we can use a QBE interface to update a database. Here a Modules skeleton has been called up. The string 'Relational Database Systems' indicates what we wish to match on. The string 'U.2' indicates that we want to change the level of this module to 2. This screen effectively implements the SQL command below:



```
UPDATE Modules
SET level = 2
WHERE moduleName = 'Relational Database Systems'
```

Lecturers

staffNo	staffName	status	deptName	salary
P.	P.		Computer Studies	

Modules

moduleName	level	courseCode	staffNo
Relational Database Systems	U.2		

Figure 24.1 Original QBE interface.

24.4

GRAPHICAL USER INTERFACES AND VISUAL QUERY

Query by example interfaces such as the one described above were first created as a step towards true graphical user interfaces (GUIs). Nowadays it is more usual for users to activate functions such as *print this column* through the use of buttons or some other direct manipulation device. For instance, this is the approach taken in DBMS such as Microsoft Access (Chapter 33).

Figure 24.2 illustrates how a query can be constructed using the Microsoft Access query builder. Here, two tables (modules and lecturers) are displayed in the upper half of the window. The relationship represented between these tables indicates that we have joined data from the two tables together. Thus we can select from the list of columns in the modules table and those in the lecturers table and add the chosen columns to our query represented by the bottom half of the window. In terms of the query under consideration here, we have chosen to output the staffNo and staffName from the lecturers table and the moduleName from the modules table. When executed this will produce a natural join of the data from the two tables.

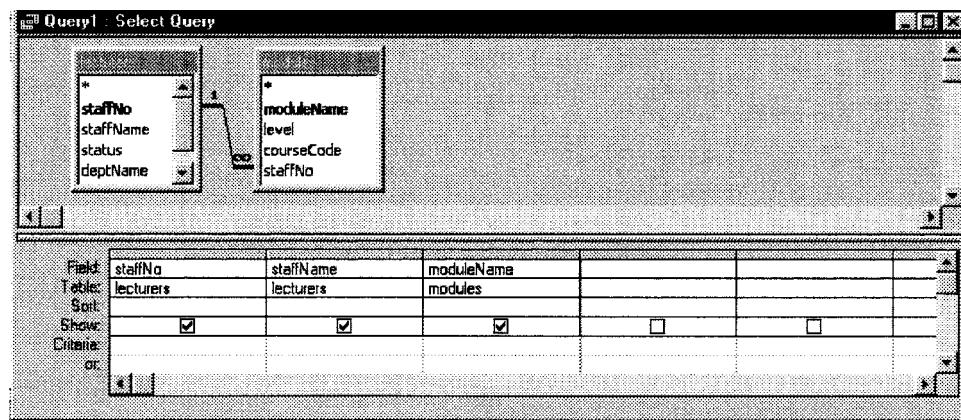


Figure 24.2 Microsoft Access query interface.

Yet another approach is to allow the user to specify his or her query in a visual manner. Some DBMS present a visual representation of either the whole or more usually part of a database for the user. The user can then directly indicate on this visual representation the data required.

In terms of managing the output from some query, many vendors are now offering the idea of scrollable cursors. A scrollable cursor is a window into which the results from a given query can be accessed. The user can scroll up and down through this window and print selected output as required.

Figure 24.3 illustrates a comparable facility in Microsoft Access. Here the results from the query specified in Figure 24.2 are displayed in a window for the user to examine.

staffNo	staffName	moduleName
145	Smith J	Intro to Business
145	Konstantinou P	Basic Accountancy
234	Davies T	Relational Database Systems
234	Davies T	Relational Database Design
237	Patel S	Distributed Database Systems
345	Evans R	Deductive Databases
345	Evans R	Object-Oriented Databases

Figure 24.3 A scrollable window in Microsoft Access.



24.5 NATURAL LANGUAGE INTERFACES

The normal method of extracting data from databases is via some formal language such as SQL. However, many people have made the point that a user wishing to extract data using a formal query language has to construct a query not only using knowledge of data but also using knowledge about the syntactical and semantic conventions of the formal language. Much of this problem can be overcome if the user interface exploits a more ‘natural’ form of communication – something akin to everyday English. The piece of software which conducts this form of dialogue with the user is normally known as a natural language interface (NLI) (Rich, 1984).

Example Suppose management of a university wishes to find out how many students are taking computer studies modules. Assuming the existence of a university database organised on the lines of that discussed in previous chapters, such a query might look in SQL as follows:

```
SELECT count(*)  
FROM enrolment E, modules M, courses C  
WHERE E.moduleCode = M.moduleCode  
AND M.courseCode = C.courseCode  
AND department = 'Computer Studies'
```

Alternatively, such a query might be expressed to a NLI as:

Count the number of students taking Computer Studies modules

The NLI would have to take this query and translate it into the syntax of the formal query language. This it normally does in three stages:

- *Lexical analysis.* This is the process of dividing up the input sentence into words. The list of words that a given NLI can understand is normally stored in a repository known as a lexicon
- *Syntactic analysis.* This is the process of assigning some grammatical structure to the input sentence. Such a structure is normally known as a parse
- *Semantic analysis.* This is the process of assigning some meaning to the input statement and determining what action should be taken

NLIs are more accurately referred to as restricted language interfaces. An NLI usually has to be ‘trained’ for a particular application. An NLI cannot ‘understand’ all English sentences, only a restricted set of everyday English sentences. It has to learn the appropriate terminology of the user group and how this terminology connects to underlying databases.

An NLI can be attached to a speech recognition system. This speech recognition system also has to be trained to associate an individual’s spoken words



with words in the lexicon of the NLI. This combined system will allow users to speak restricted English commands to the NLI interface and receive output which may also be converted into the spoken word.

24.6**SUMMARY**

- A user interface can be seen as a collection of dialogues between the user and the ICT system. There are a number of ways of building interfaces, including menus, forms, command language, direct manipulation and natural language interfaces
- There are a number of tools that can currently connect to a relational DBMS kernel via SQL. Each of the tools enables the user to build what is effectively an SQL query in an easier manner than writing SQL itself
- Such end-user interfaces include QBE, visual query and natural language interfaces
- The main problem with all such interfaces is that it usually proves difficult to specify complex queries using these approaches. For queries of any complexity the user eventually has to resort to the specialist SQL programmer
- There are, of course, many other tools we have not considered of use to the end-user, e.g. spreadsheets, word-processors, workflow software etc. With the rise particularly of the client-server architecture, the number and diversity of such products multiplies daily. All such tools will transparently build SQL queries and manage the results of such queries

24.7**ACTIVITIES**

1. Analyse some interface known to you in terms of the distinction between content, control and format.
2. In terms of some DBMS with which you are familiar, attempt to describe the range of interfaces available for the user.
3. Assess whether a DBMS such as ORACLE provides a QBE interface to users.

24.8**REFERENCES**

- Rich, E. (1984). Natural-language interfaces. *IEE Computer* **17**(9): 39–47.
Schneiderman, B. (1998). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Reading, MA, Addison-Wesley.
Zloof, M.M. (1975). Query by example. *Proc. NCC* **44**(May).

CHAPTER 25

DBMS-TOOLKIT – APPLICATION DEVELOPMENT TOOLS

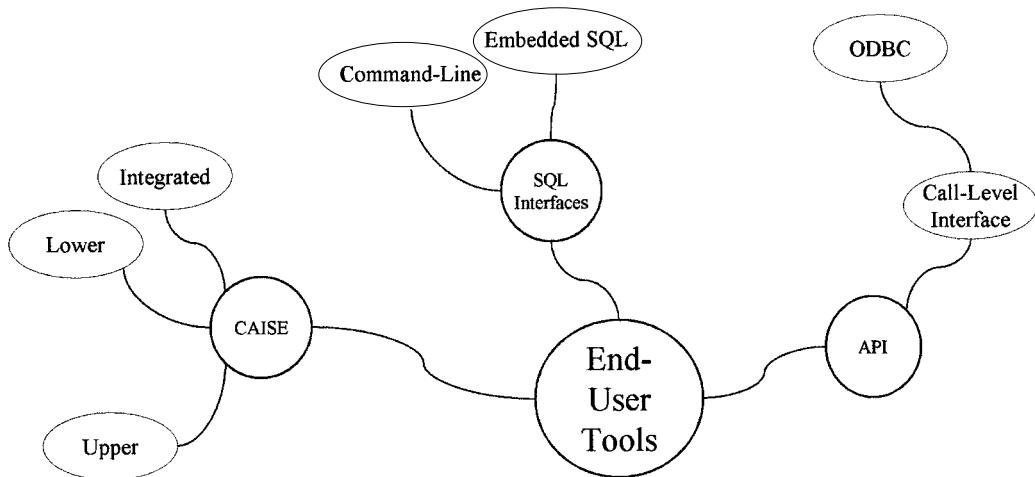
If the only tool you have is a hammer, you tend to see every problem as a nail.

Abraham Maslow

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Discuss command-line and embedded SQL interfaces
- Define the concept of an application programming interface
- Consider the use of SQL within a fourth generation language
- Discuss the relevance of computer aided information systems engineering tools to application development



**25.1****INTRODUCTION**

In this chapter we concentrate on tools for constructing ICT systems using the facilities provided by database systems. Hence, the main user we consider here is the application developer. We particularly concentrate on the use of SQL within the domain of third and fourth generation programming languages. We also look at the issue of computer aided information systems engineering (CAISE) tools and their relevance for database development.

The development of an ICT system is a human activity system (Chapter 14). The key output from this system is clearly an ICT system. The key inputs into the process are ICT resources and developer resources. ICT resources may comprise hardware, software, data and communications technology. Developer resources include not only people but a toolkit which includes methods, techniques and tools available to the developer. In this chapter we consider tools appropriate for constructing the data management layer of an ICT system and connecting this layer to other parts of an ICT application.

25.2**COMPUTER AIDED INFORMATION SYSTEMS ENGINEERING
(CAISE)**

Computer Aided Information Systems Engineering (CAISE) is a natural consequence of applying automation principles to the process of developing information systems. CAISE is founded in a model of the development process as a set of activities that operate on objects to produce other objects. The objects operated on may be documents, diagrams, data-structures and/or programs. The activities involved may be relatively formal, such as compiling a program, or informal, such as discussing alternative design proposals.

Three levels of CAISE tool can be identified:

- *Lower CAISE tools.* Sometimes referred to as back-end CAISE tools, these are high-level environments for application construction, testing and possibly maintenance. Examples include fourth generation languages (4GLs) and graphical user interface (GUI) builders
- *Upper CAISE tools.* Sometimes referred to as front-end CAISE tools, these are tools for supporting analysis and design work. Tools which support E-R diagramming and object-oriented analysis and design (see Chapter 16 and 17) are examples of this type
- *Integrated CAISE tools.* These tools attempt to integrate upper and lower CAISE by offering an integrated environment for supporting the entire life-cycle of the development

**Example** ➔

PC Select is an example of an integrated and relatively inexpensive CAISE tool. It offers a number of specific versions tailored for specific methodologies. For example, it has a version for the popular information systems development methodology SSADM (Structured Systems Analysis and Design Method) (Weaver *et al.*, 1998) – Select/SSADM. The facilities offered by this package fall mainly into the front-end CAISE category, although there are some facilities for things such as schema generation that fall into the back-end category. Select/SSADM particularly allows people to construct the standard documentation such as Logical Data Structures (effectively E-R diagrams), data flow diagrams and entity-life histories associated with the SSADM methodology. Without the support of a CAISE tool such as Select/SSADM, the production of SSADM deliverables such as E-R diagrams and entity-life histories would be a laborious and error-prone task. The use of Select/SSADM ensures that the SSADM diagrammatic conventions and rules are adhered to.

Select allows people to set up ‘projects’ which may contain all the documentation associated with a particular development project. A number of people can work concurrently within the domain of one particular project.

One way in which Select/SSADM can be used as a back-end facility is by automatically generating a file of SQL Data Definition Language statements from an E-R diagram. To do this an E-R diagram must first be extended by defining the primary/foreign keys associated with each entity and data-typing the other attributes of each entity. Having done this, the user can submit the design to a process that generates a file of SQL *Create table* and *Create index* statements.

In this chapter we particularly focus on that type of lower-end CAISE concerned with connecting conventional third and fourth generation languages to database systems.

25.3 SQL INTERFACES

When SQL was originally designed it was envisaged as an end-user query language. Since that time, most organisations are reluctant to release SQL directly to their end-user community. This is because whereas SQL is a language which is relatively easy to understand, it is not easily mastered. Writing SQL is a programming task and is hence best performed by those persons with some programming experience.

SQL can be accessed in a number of ways, two of which are defined in the ISO standard (Chapter 11). It can either be used interpretively as a command-line interface or it can be embedded within a host language such as COBOL, FORTRAN, PL/1, C or Pascal. A growing range of DBMS vendors are also closely-coupling SQL to 4GL offerings. However, little standardisation exists in this latter area.

25.4 COMMAND-LINE INTERFACES TO SQL

Command-line interfaces look somewhat dated in the age of graphical user interfaces (GUIs). However, they have a number of advantages as compared to GUI interfaces – they are quick to use and quick to execute. They are hence particularly effective as a means of prototyping database applications and it is for this reason that most industrial-strength DBMS retain such interfaces.

The normal way of experiencing SQL is in interpreted mode. In this section we look at how SQL works as a command-line interface. The first thing the user has to do is log-on to SQL from the operating system level.

Example → This he or she might do using a command such as the following:

```
$ SQL <username>/<password>
```

For instance:

```
$ SQL pbd/cymru
```

We know we are in the command-line interface when we get something like the following prompt:

```
SQL>
```

We can now type in a command-line such as the following:

```
SQL> SELECT *
  2  FROM Lecturers;
```

The interface responds with the following result:

staffNo	staffName	status	deptName	salary
234	Davies T	L	Computer Studies	16000.00
237	Jones S	SL	Computer Studies	23500.00
345	Evans R	PL	Computer Studies	26500.00
123	Smith J	L	Business Studies	16500.00
145	Thomas P	SL	Business Studies	23500.00

5 records selected

Note how we can spread a statement over several lines for ease of reading and for ease of editing. This is because most command-line interfaces maintain a small working memory known as the buffer. Anything we type in response to the SQL prompt gets stored in the buffer. The statement is not activated until we type a semicolon to terminate the statement or issue the command *run*.

Command-line interfaces are mainly used by the programmer to test SQL queries against databases or by the DBA who wants to quickly construct query



functions to run against the system catalog. They may also prove useful for the sophisticated end-user who wants to generate *ad-hoc* queries against a database.

25.5 EMBEDDED SQL

Embedded SQL is primarily for use by application developers. It is designed to simplify database input and output from application programs via a standard interface. This enables the process layer of an ICT system expressed in some 3GL to connect with an SQL-based DBMS which provides the data management layer for the application. In this section we introduce the concept of SQL as an embedded database language by discussing one simple example in a pseudo-host language modelled on the high-level language Pascal:

Example PROGRAM staffNumbers;
EXEC SQL BEGIN DECLARE SECTION;
 staffNo: integer;
 staffName: packed array(1..12) of char;
EXEC SQL END DECLARE SECTION;
BEGIN
 EXEC SQL DECLARE LectCurs CURSOR FOR
 SELECT staffNo, staffName
 INTO :staffNo, :staffName
 FROM Lecturers;
 WRITELN ('Staff Numbers and Names');
 WRITELN ('_____');
 EXEC SQL OPEN LectCurs;
 EXEC SQL FETCH LectCurs;
 WHILE SQLCODE = 0 and SQLCODE <> 100 DO
 WRITELN (staffNo,'',staffName);
 EXEC SQL FETCH LectCurs;
 EXEC SQL CLOSE LectCurs;
END

The program above prints the list of staff numbers and names in the Lecturers table. Embedded data language statements are preceded by the keywords EXEC SQL. The *declare* statement defines a pointer to the Lecturers table known as a cursor. When the open statement is activated, the *select* statement defined at the declare stage is executed. This creates a copy of all the staff numbers and names in the base table and places it in a workspace area. To retrieve an actual record from this workspace we use the *fetch* command. The fetch statement takes a staff number and places it in the program variables staffNo and staffName, both prefixed by a colon. *Sqlcode* is a system variable. It returns 0 if the fetch is executed successfully, non-zero otherwise. It returns 100 if no data



is returned. Hence we use it here to act as a terminating condition to traverse the staff numbers and names in the workspace.

The concept of a cursor is extremely important in that it acts as the cement between the inherently non-procedural nature of SQL and the inherently procedural nature of a third generation language (3GL) such as Pascal. There is in fact what is usually referred to as an impedance mismatch between SQL and 3GLs. SQL, being relational, works at the level of sets of data. It takes tables as input and produces tables as output. Third generation languages however work at the record level. They only process one record at a time. We therefore have to have some mechanism of translating from a table-based to a record-based approach. This is provided by the cursor concept.

Example → The program below illustrates how we use an embedded SQL program to update a base file. The program uses the idea of a cursor to update the base table positionally, that is, on the basis of the current position of the cursor in terms of base records.

```

PROGRAM NewYear;
EXEC SQL BEGIN DECLARE SECTION;
  studentLevel: integer;
EXEC SQL END DECLARE SECTION;
BEGIN
  EXEC SQL DECLARE StudentCurs CURSOR FOR
    SELECT studentLevel
    INTO :studentLevel
    FROM Students;
  EXEC SQL OPEN StudentCurs;
  EXEC SQL FETCH StudentCurs;
  WHILE SQLCODE = 0 and SQLCODE <> 100 DO
    EXEC SQL UPDATE Students
    SET studentLevel = :studentLevel + 1
    WHERE CURRENT OF studentCurs;
    EXEC SQL COMMIT WORK;
    EXEC SQL FETCH StudentCurs;
    EXEC SQL CLOSE StudentCurs;
END

```

In this program the cursor is first declared and given a name. Then the cursor is opened. The fetch command is used to get the next row from a cursor. SQLCODE is used here to terminate a loop which increments the student level for the current row. The 'where current of' clause indicates a positioned update, i.e. the current row of the cursor is updated. Note that each update is committed as a transaction to the database. Finally, when all work is completed successfully, the cursor is closed.

A program written in this hybrid manner is normally submitted to a pre-compiler. The *exec sql* keywords are important in enabling the precompiler to identify SQL commands from Pascal commands. The precompiler will take the SQL commands and create a database access module. It will also take the Pascal commands and produce a source program suitably modified with database access calls. This modified source is then compiled in the normal manner. Figure 25.1 illustrates this process.

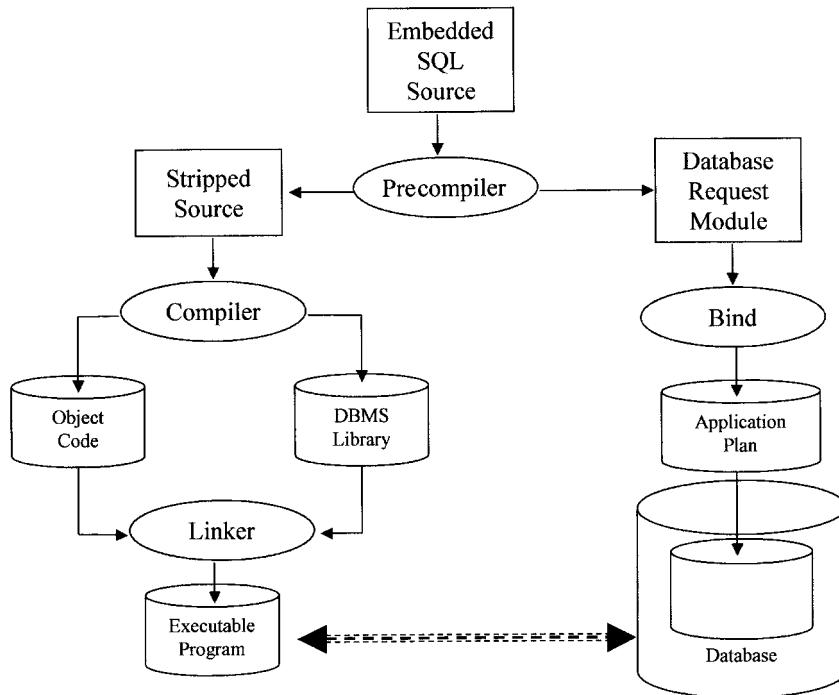


Figure 25.1 Using embedded SQL.

Embedded SQL actually comes in two forms within the standard: static embedded SQL and dynamic embedded SQL. Using static embedded SQL, the whole of some SQL statement must be specified when the embedded SQL source program is written. In dynamic embedded SQL, part of some SQL statement can be specified at run-time. In practice, this difference means that static embedded SQL does not allow host variables to stand for table names or column names (as in the second example in this section).

25.6

APPLICATION PROGRAMMING INTERFACES

Embedded SQL is one type of tool generally known as an application programming interface, or API for short. The other major type of API is a call-level interface. This type of interface has become extremely popular in client-server

architectures (Chapter 36). Specific variants of APIs for Web-based development are discussed in Chapter 43.

A call-level interface is a lower-level interface than embedded SQL. Rather than embedding SQL statements in application code we embed a series of standard subroutine calls that send messages to a given DBMS.

Microsoft, for instance, has a call-level interface – its Open Database Connection (ODBC) standard. This is basically an interface between a Microsoft Windows application and various DBMS. It is based on specifications for a call-level interface developed by the SQL Access group, a group of vendors which has produced a common specification for SQL. ODBC fundamentally amounts to a defined library of database access calls. It enables the application programmer to establish a connection with a database environment, to execute SQL and non-SQL statements and to retrieve results. The idea is that by defining a series of standard calls, different vendors can provide different drivers to their particular DBMS. Figure 25.2 illustrates this idea.

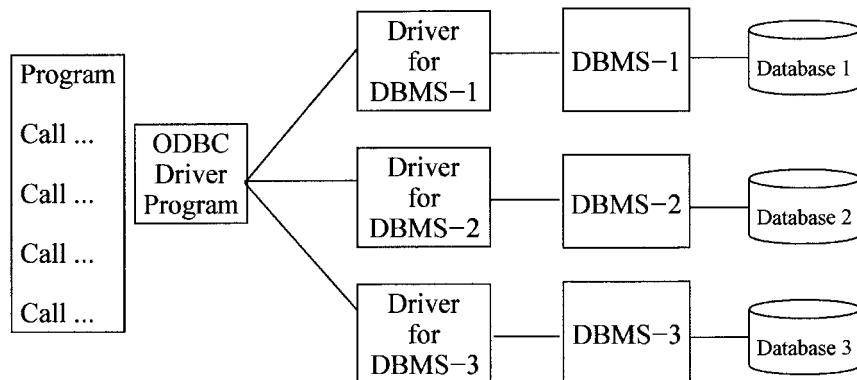


Figure 25.2 ODBC components.

25.6.1 COMPARISON OF EMBEDDED SQL AND CALL-LEVEL APIs

Both embedded SQL and the call-level API have advantages for the applications developer. Both forms of API are designed to create a degree of data independence (Chapter 6) between application code and database. They hence permit the easier maintenance of application code or database structures. In terms of embedded SQL, a specific advantage is the use of standard SQL and therefore there is no requirement for the application developer to learn another language. In terms of call-level APIs, no pre-compilation phase is required and it is argued that such approaches provide the developer with more fine-grained control over database operations. However, the major disadvantage lies in such APIs being *de facto* rather than *de jure* standards. Hence they are heavily reliant on vendor initiatives.

25.7 4GLs AND SQL

Among DBMS vendors there is a growing tendency to offer fourth generation language (4GL) solutions for solving application development problems. Two major types of 4GL tool are offered:

- A forms-based generator
- A high-level procedural language usually closely coupled to SQL

Normally these two offerings will be used interdependently. A series of data entry and retrieval forms will be built rapidly using the forms-based generator. Then a series of modules written in the 4GL will be attached at various points to the processing on the form.

A procedural 4GL frequently consists of the embedding of SQL syntax within standard programming constructs such as IF THEN ELSE, CASE, DO WHILE, REPEAT UNTIL etc. For instance, a piece of application logic is specified below in the syntax of ORACLES's PL/SQL 4GL product (Gennick, 2000). PL/SQL is discussed in more detail in Chapter 34.

Example →

```

DECLARE
    salary emp.sal%TYPE;
    manager emp.mgr%TYPE;
    lastname emp.ename%TYPE;
    starting_empno CONSTANT NUMBER(4):= 7902;

BEGIN
    SELECT sal, mgr
    INTO salary, manager
    WHERE empno = starting_empno;
    WHILE salary < 40000 LOOP
        SELECT sal, mgr, ename
        INTO salary, manager, lastname
        FROM emp
        WHERE empno = manager;
    END LOOP;
    INSERT INTO temp VALUES(null, salary, lastname);
    COMMIT;
END;

```

This example will find the first employee who has a salary greater than £40,000 and is higher than employee number 7902 in the chain of command. The %TYPE declarations serve to bind the data types of the host variables to the types declared in the system tables. This serves to provide a degree of data independence.

**25.8****SUMMARY**

- A number of different forms of interface are provided in SQL – GUIs, command-line interfaces, call-level interfaces and embedded SQL
- Command-line interfaces are historically the oldest and offer quick and effective execution of SQL commands for activities such as the testing of queries
- Embedded SQL represents a standard application programming interface for use with a range of third generation languages
- Call-level application programming interfaces are lower-level interfaces to DBMS
- Many CAISE tools can be used for front-end database work or for back-end database development

25.9**ACTIVITIES**

1. Identify a given CAISE tool and attempt to determine its relevance for creating the data management layer.
2. Discuss what you feel to be the major strengths and weaknesses involved in two different approaches to building application systems in call-level APIs and SQL.

25.10

Gennick, J. (2000). *Sams Teach Yourself PL/SQL in 21 Days*. New York, Sams Publishing.
Weaver, P.L., N. Lambrou and M. Walkley (1998). *Practical SSADM 4+*. London, Pitman.

CHAPTER 26

DBMS-TOOLKIT – DATABASE ADMINISTRATION TOOLS

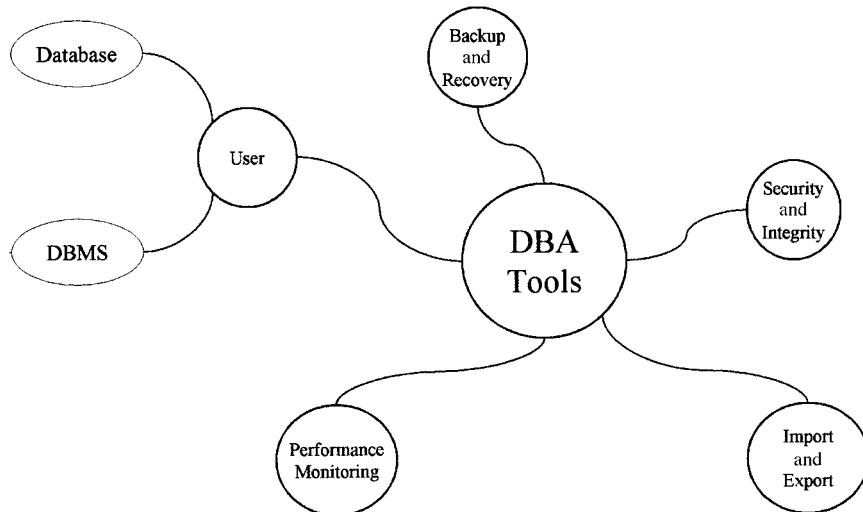
We shall not fail or falter; we shall not weaken or tire . . . Give us the tools and we will finish the job.

Sir Winston Churchill (1874–1965) BBC radio broadcast, 9 February 1941

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Describe some of the key user-related functions available to the database administrator
- Describe some of the key archiving, backup and recovery functions available to the database administrator
- Describe some of the key security and integrity functions available to the database administrator
- Describe some of the key performance monitoring functions available to the database administrator



26.1  INTRODUCTION

In any medium to large-scale database application, at least one person will be given the task of administering the database. These persons, called database administrators, are generally responsible for the technical implementation of database systems, managing the database systems currently in use, and setting and enforcing policies for their use.

Some of the key functions of the database administrator (DBA) are described in Chapter 23. Of the functions described, the following demand specialised tools (Loney and Theriault, 2002):

- Data archiving
- Data backup and recovery
- Data control
- Establishing user groups
- Assigning passwords
- Granting access to DBMS facilities
- Granting access to databases
- Monitoring data usage and tuning database systems
- Enforcing security and integrity

We focus on the DBMS tools available for supporting these functions in this chapter.

Most of the tools for performing these functions now provide both command-line and graphical user interfaces. However, because interfaces differ among DBMS we do not focus on the presentation of the tools but merely on their general functionality here.

26.2  STARTING, STOPPING AND TAKING A DATABASE OFF-LINE

Commands must be provided by the DBMS to start a database application and to stop a database application. This may be done at the start and end of each day or at some other period. Occasionally either the whole or a part of the database may need to be taken off-line, that is, still running but not accessible to any but DBA users. This may need to be done, for instance, to enable restructuring of the schema.

26.3  USER FUNCTIONS

A number of functions are usually provided to enable the DBA to specify precise usage profiles for a given database and DBMS.

26.3.1 ESTABLISHING USER GROUPS

Most DBMS now enable the DBA to enrol new users onto a database. Users are normally assigned unique user names. Some DBMS also allow the DBA to create user groups or ‘roles’ with defined names. This facility enables the DBA to specify a common profile for a collection of similar users.

Example ➔ For instance, in terms of the academic database discussed in previous chapters we might define two roles: Staff and Administrator.

Given that users or user groups have been specified in terms of user name, a number of properties can be attached to each user name or group.

26.3.2 ASSIGNING PASSWORDS TO USERS OR USER GROUPS

One of the most basic properties is the definition of a password against a user or user group. This enables some basic security to be built into a database system.

Example ➔ Hence we might assign the role Staff the preliminary password u23Z00 and the role Administrator the preliminary password Z54U00.

Individual user passwords may be changed by the respective user at any time. Role passwords are likely to be changed by the DBA or changed by a delegated member of the user group at periodic intervals.

26.3.3 GRANTING ACCESS TO DATA

This will normally be done using the SQL grant options described in Chapter 23. We briefly summarise these options below:

- The creation of views – windows into a database
- Granting the capability to insert data into specific tables or views to specific users or user groups
- Granting the capability to delete data from specific tables or views to specific users or user groups
- Granting the capability to update data in specific tables or views to specific users or user groups
- Granting the capability to retrieve data from specific tables or views to specific users or user groups

As well as being able to grant these functions to specific users or user groups, the DBA is able to revoke such privileges from specific users or groups.



Example ➔ In terms of our two defined roles we might define the following data access privileges:

- *Staff*. Read access to the staff record corresponding to the particular user. Read access to the students on modules assigned to the particular staff member. Write access to the assessments table corresponding to modules associated with a particular member of staff
- *Administrators*. Write access to the staff and students table. Read access to the assessments and modules tables

26.3.4 GRANTING ACCESS TO DBMS FACILITIES

The commands for granting access to data are relatively well-standardised. The commands for granting access to DBMS facilities vary among vendors. Those provided by the ORACLE DBMS are described in Chapter 34. Generally the DBMS will assign one or more levels of DBMS privileges to a specific user or user group. For instance, the DBMS may distinguish between users in terms of:

- The ability to access the data dictionary
- The ability to create users
- The ability to create tables and views
- The ability to create indexes
- The ability to grant and revoke access to tables and views

Generally, end-users will be assigned a level which permits them to access data in existing tables. In contrast, application developers will need the ability to create their own tables, views and indexes. At the highest level, one or more DBAs will be able to act as superusers in terms of a given DBMS.

Example ➔ Both the Staff and Administrator roles are likely to be classed as end-users and hence will be limited to read or write access to specified tables in the schema.



26.4 DATA ARCHIVING, BACKUP AND RECOVERY

A DBMS will normally provide a facility to allow backup copies of either the whole or part of a database to be made at regular intervals. The DBA must also be able to back-up the transaction log or journal (Chapter 29). Generally the DBA will be able to back-up the database and log without having to stop the database system. Typically, the backup copies are written to off-line storage media such as magnetic tape.

Backups of the database (including the data dictionary) and transaction log



may be used to restore the database system in the event of disastrous hardware or software failure. A DBMS normally provides recovery facilities for enabling this restoration process.

At longer intervals the DBA may need to purge the database of unused data. Such data may be archived for historical purposes. Facilities for compressing the data such that high volumes of data can be archived easily may be provided within a DBMS.

26.5 ENFORCING SECURITY AND INTEGRITY

Data integrity involves protecting the database from authorised users of the database. In contrast, data security involves protecting the database from unauthorised users of the database.

Security is normally enforced in a database system using the facilities of user enrolment and assigning privileges to defined users. Some DBMS now enable data to be encrypted, particularly in transmission. Encryption involves encoding the data using an algorithm that utilises encryption and decryption keys. Encrypted data cannot be read without access to the associated decryption key.

Integrity is enforced through integrity constraints. The DBMS will normally enable the DBA to report on active constraints, to drop, enable and disable constraints and to monitor the effect of constraints on update performance.

26.6 IMPORTING AND EXPORTING DATA

DBMS normally provide a number of tools for importing data (sometimes referred to as loading) from other systems into the database. Such data will frequently be stored in different original formats. This means that the data frequently needs to be run through a data conversion tool before import.

Tools will also be provided for exporting data from the database to other systems. Again, this may involve an associated data conversion process.

26.7 MONITORING DATA USAGE AND TUNING DATABASE SYSTEMS

A DBMS normally provides facilities for monitoring the performance of a database application. Such tools are used to ensure that the database performs to acceptable levels in terms of both retrieval and update. These tools enable the DBA to get information on the usage of the database, such as the execution of queries and the efficiency of concurrency control. The DBA can then use this information to tune the database application to provide better performance. This may be done by creating new indexes, by changing storage structures or by restructuring the schema.

The DBA will also need to occasionally maintain a database application. New requirements may lead to the need for redesigning aspects of the system. One example of this may be the need to adjust the storage capacity of a database.

26.8  SUMMARY

- Database administrators are responsible for the technical implementation of database systems, managing the database systems currently in use, and setting and enforcing policies for their use
- A range of tools is provided by contemporary DBMS to enable the administration process. These can be categorised into groups such as user-related functions, data archiving, data backup and recovery, data control, performance monitoring, enforcing security and integrity
- User-related functions include assignment of user groups and passwords, granting access to DBMS facilities and granting access to databases

26.9  ACTIVITIES

1. In terms of some DBMS known to you, investigate the range of user-related facilities provided for the database administrator.
2. In terms of some DBMS known to you, investigate the range of backup and recovery facilities provided for the database administrator.
3. In terms of some DBMS known to you, investigate the range of performance monitoring facilities provided for the database administrator.

26.10  REFERENCE

Loney, K. and M. Theriault (2002). *Oracle9i DBA Handbook*. New York, McGraw-Hill.

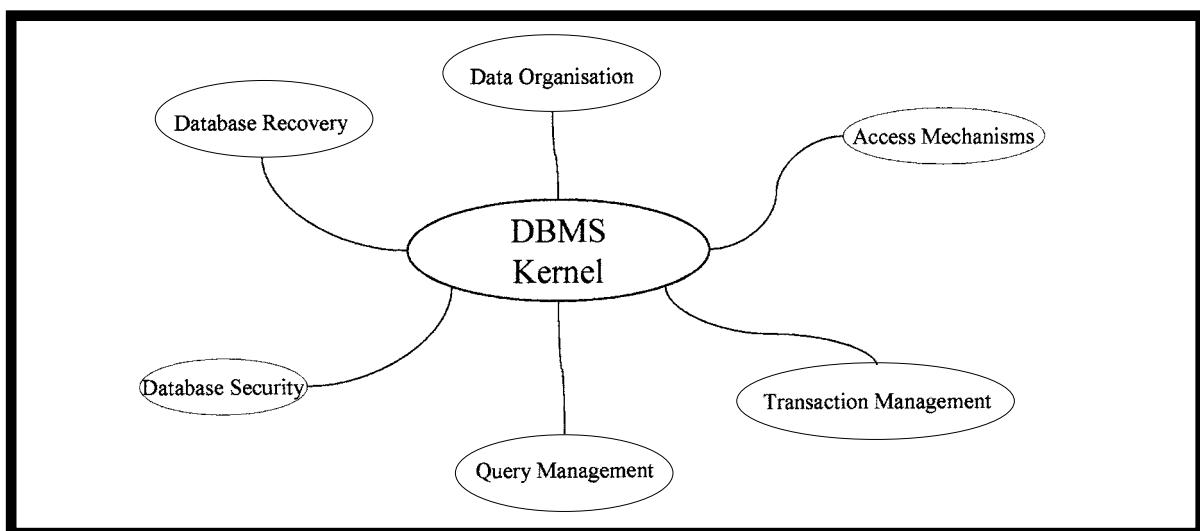
PART

7

DATABASE MANAGEMENT SYSTEMS – KERNEL

Good order is the foundation of all things.

Edmund Burke





In this part we consider some of the core functions of the kernel or engine of a DBMS originally referred to in Chapter 6:

- *Data organisation* (Chapter 27). In this chapter we consider some of the common ways of storing data at the physical level in database systems
- *Access mechanisms* (Chapter 28). In this chapter we consider a number of database mechanisms for improving retrieval performance to data
- *Transaction management* (Chapter 29). In this chapter we define the concept of a transaction and consider the important issues of concurrency and consistency control
- *Other kernel functions* (Chapter 30). In this chapter we consider the issues of dictionary management, query management, backup and recovery

In each chapter we particularly concentrate on the facilities available in current relational and post-relational DBMS.

CHAPTER 27

DATA ORGANISATION

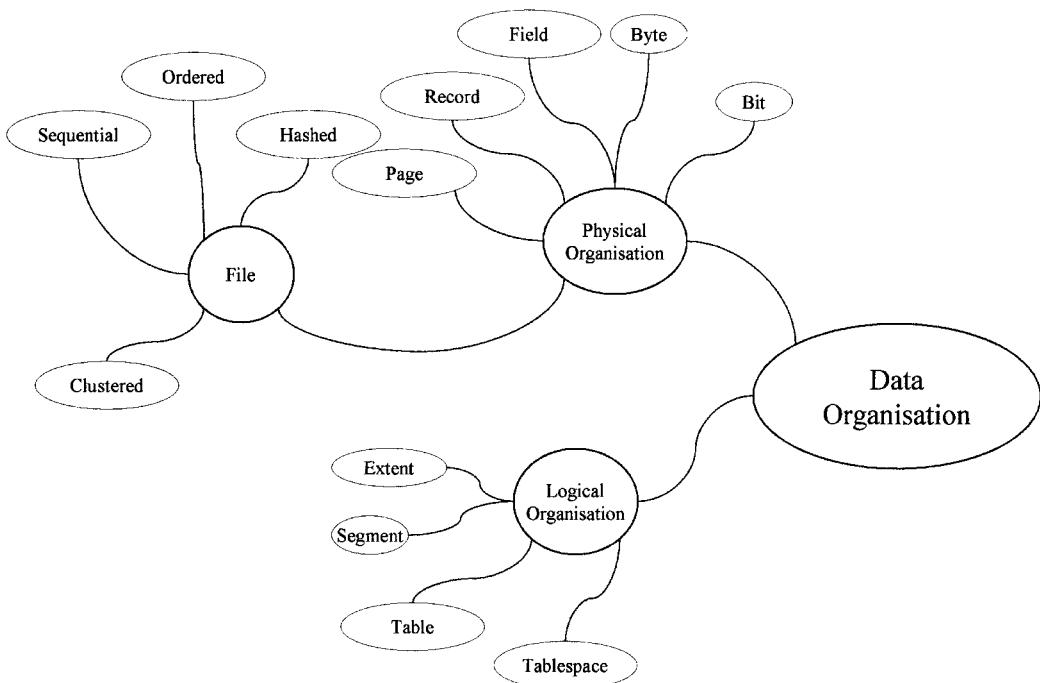
Drama is life with the dull bits cut out.

Alfred Hitchcock (1899–1980)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Distinguish between logical and physical organisation of data
- Describe the major features of the physical organisation of data
- Illustrate features of logical organisation and the mapping to physical organisation



**27.1**

INTRODUCTION

Data organisation (sometimes called file organisation) concerns the way in which data is structured on physical storage devices, the most important being disk devices. The idea of data organisation and access is inherently interlinked. In this chapter we shall discuss the main types of data organisation found in DBMS. We shall also discuss a type of data organisation known as a cluster, which is particularly commonplace in contemporary relational DBMS. In the next chapter we address the related issue of providing access to data.

27.2

LOGICAL AND PHYSICAL ORGANISATION

Data models as described in Part 2 are all forms of logical data organisation. For instance, relations or tables are logical data structures. A database organised in terms of any particular data model must be mapped onto the organisation relevant to a physical storage device. This form of data organisation is known as physical organisation (Ramakrishnan and Gehrke, 2003).

27.3

DISK ORGANISATION

Different storage devices will utilise different forms of physical organisation. In this chapter we particularly discuss those forms of physical organisation relevant to disk devices since disks are the most prevalent form of storage device available.

The collection of data making up a database must be physically stored on some computer storage medium. Computer storage can be divided into two categories: primary storage and secondary storage. Primary storage includes media that can be directly acted upon by the central processing unit (CPU) of the computer, such as main memory or cache memory. Primary storage usually provides fast access to relatively low volumes of data. Secondary storage cannot be processed directly by the CPU. It hence provides slower access than primary storage but can handle much larger volumes of data. Two of the most popular forms of secondary storage are magnetic disk and magnetic tape.

Most databases are too large to fit in main memory and are therefore usually stored on secondary storage devices such as magnetic disks. Main memory is referred to as volatile memory because the data is lost when the power supply is switched off. Secondary storage is referred to as non-volatile storage because it persists after power loss.

The most basic unit of data stored on disk is the bit (0 or 1). The capacity of some storage device is frequently expressed in terms of the maximum number of bytes it can store. A byte consists of 8 bits. The capacity of a disk is hence the number of bytes it can store. Disk capacities are normally described in terms of kilobytes (Kbytes: 1 thousand bytes), megabytes (Mbytes: 1 million bytes), gigabytes (Gbytes: 1 billion bytes), or terabytes (Tbytes: 1 trillion bytes).

Example → A typical floppy disk will store from Kbytes to Mbytes of data, hard disks for personal computers typically hold from Mbytes to Gbytes of data, and disk packs used in mini and mainframe systems typically store Gbytes to Tbytes of data.

Disks of whatever form are all constructed of thin circles of magnetic material protected by plastic covers. A disk is single-sided if it stores data on only one side of the circle (surface), double-sided otherwise. Disks are assembled into disk packs to increase storage capacity. Disk packs may have as many as 30 surfaces.

Data is stored on a disk surface in concentric circles. Each such circle is called a track. In a disk pack, tracks having the same diameter on different surfaces are said to form a cylinder. The number of tracks on a disk typically numbers in the hundreds. Each track stores Kbytes of information and is frequently divided into sectors that are hard-coded onto the disk. In contrast, a track will frequently be divided into a number of equal-sized blocks by the operating system during disk formatting. Blocks are hence soft-coded and separated by fixed-size areas known as inter-block gaps.

A disk is a random-access device. Data is transferred between main memory and disk in units of blocks (sometimes called pages). The hardware address of a block is composed of a combination of surface number, track number within surface and block number within track. This address is given to the disk input/output (I/O) hardware. If a read command is received by this hardware it copies the contents of a block to a buffer – a reserved area of main storage that holds one block. For a write command, the hardware copies the contents of the buffer to the disk block.

27.4 FILES, RECORDS AND FIELDS

In terms of a given operating system the key data structure is that of a file. Each file will be made of data elements known as records. Each record consists of a collection of related data values or data-items made up of a number of bytes and corresponding to a particular field. Each record in a file will be of a given record type. A record type amounts to a format for record. A file is a collection of such records. Usually all the records in a file will be of the same record type. Figure 27.1 illustrates the hierarchy of physical data structures, data elements and data-items.

Files and records have to be mapped onto blocks, since blocks are the physical unit of disk transfer. Usually, the record size will be less than the block size, meaning that many records will be stored in each block. Consequently a file will be made up of many blocks, frequently but not necessarily contiguous. Each file will normally have a file header. This will contain information used to determine the disk addresses of file blocks and record type information. Figure 27.2 illustrates the relationship between files and pages.

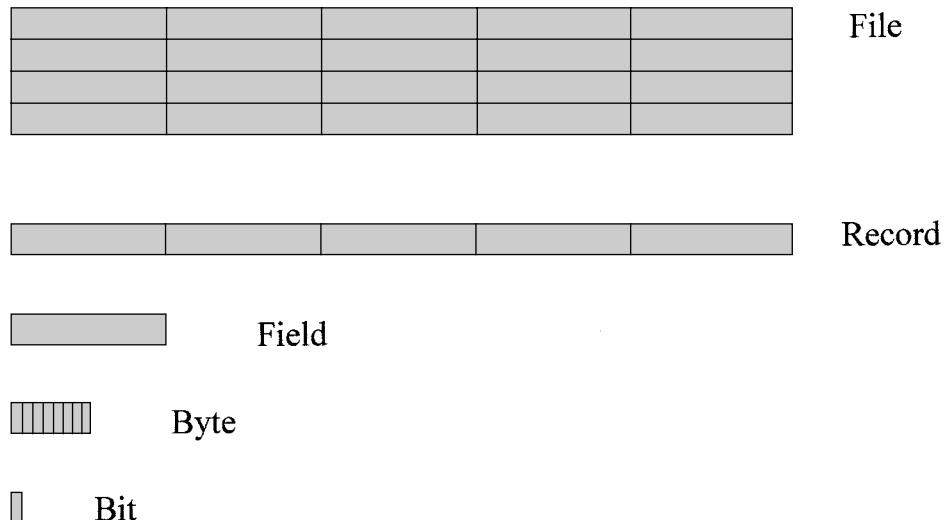


Figure 27.1 Physical organisation.

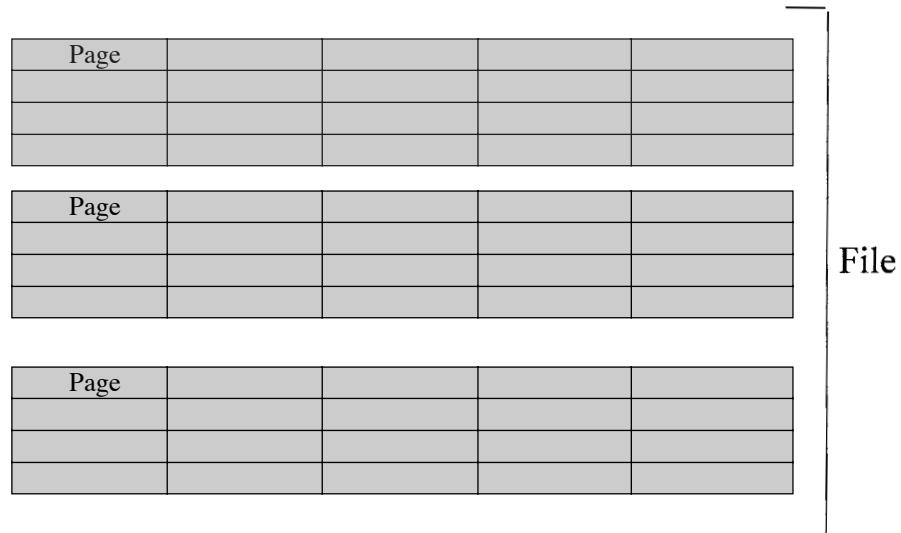


Figure 27.2 Pages.

The module of software concerned with the management of pages on disk is sometimes known as the disk manager. That module of software concerned with the management of files for the operating system is sometimes known as the file manager. When a file is created, the file manager needs to request a set of blocks from the disk manager. For instance, suppose we work in an environment in which the blocks are fixed at 4K. A 60K file will therefore need 15

blocks, and therefore the file manager requests a set of 15 blocks from the disk manager. The set of blocks is given a logical identifier – blocksetID – and each block within the set is assigned a logical ID, usually just its position within the block set. The file manager knows nothing of the physical storage of data on disk; it merely issues logical block requests (consisting of blocksetIDs and blockIDs) to the disk manager.

The disk manager translates the logical block requests from the file manager into physical blockIDs. The disk manager will usually maintain information about the correspondence between logical and physical blockIDs and will contain all the code used to control the disk device.

Figure 27.3 illustrates the process by which a DBMS communicates with an operating system's file manager which in turn issues requests to the disk manager. The DBMS has to translate requests against the constructs of a particular data model (e.g. tables) into requests for files from disk.

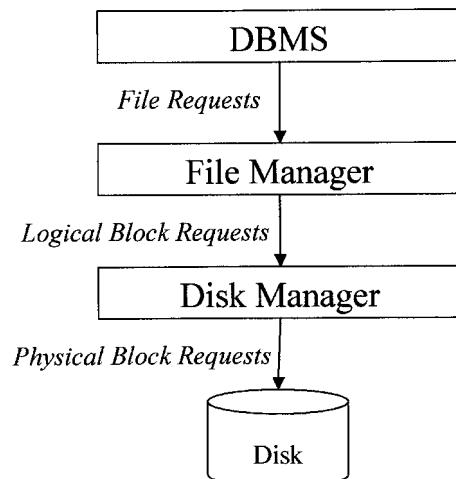


Figure 27.3 DBMS, file manager and disk manager.

The neat separation between the DBMS, file manager and disk manager discussed above does not always exist in practice. Some file managers are not particularly well-suited to the needs of database applications. In this case, the DBMS frequently bypasses the file manager and directly interacts with the disk manager to retrieve data.

27.4.1 TRANSFERRING DATA BETWEEN DISK AND CPU

To search for a record on disk, one or more blocks are copied into the main memory buffers. Programs then search for the desired record within the buffer, utilising information from the file header. If the address of the block that contains the record is not known, then programs have to perform a linear search of the records on disk. Hence, file organisation and accessing data or



input/output (I/O) activity are inherently interlinked. The objective of good physical database design (Chapter 19) is to minimise I/O activity.

27.5

PHYSICAL ORGANISATION

As the example above illustrates, the physical organisation of data can critically affect the performance of access to such data. Hence a number of different forms of file organisation have been used to address issues of access performance.

The main types of physical organisation are:

- Sequential files – sometimes known as serial files, heap files or piles
- Ordered files
- Hashed files

27.5.1 SEQUENTIAL FILES

The base form of file organisation is the sequential file. In this form of file organisation, records are placed in the file in the order in which they are inserted. A new record is simply added to the last block in the file. If there is insufficient space in the last block then a new block is added to the file. Therefore, insertion into a sequential file is very efficient.

Example Hence, suppose we have a sequential file of student records. As each student enrols with the university, a new student record is created and added to the end of the file.

However, searching for a record in a sequential file involves a linear search through the file, record by record. Hence, for a file of N records, $N/2$ records will be searched on average. This makes searching through a sequential file that is more than a few blocks long a slow process.

Another problem arises in relation to deletion activity performed against sequential files. To delete a record we first need to retrieve the appropriate block from disk. The relevant record is marked as deleted and then written back to disk. Because the deleted records are not reused, a database administrator normally has to reorganise a sequential file periodically to reclaim deleted space.

27.5.2 ORDERED FILES

The problems of maintenance associated with sequential files mean that most systems tend to maintain some form of ordered file organisation. In an ordered file the records are ordered on the basis of one or more of the fields – generally referred to as the key fields of the file.



Example ➔ If we stored student data as an ordered file then we might use student number as our key field.

Ordered files allow more efficient access algorithms to be employed to search a file. One of the most popular of such algorithms is the binary search or binary chop algorithm which involves a continuous half-wise refinement of the search space. This algorithm is described in Chapter 28.

Insertion and deletion are more complicated in an ordered file. To insert a new record we first have to find the correct position for the insertion in the key sequence. If there is space for the insertion in the relevant block then we can transfer it to main memory, reorder the block by adding the new record and then write the block back to disk. If there is insufficient space in the block then the record may be written to a temporary area known as overflow. Periodically, when the overflow area becomes full the DBA needs to initiate a merging of the overflow records with the main file.

27.5.3 HASHED FILES

Hashed files provide very fast access to records based on certain criteria. They are organised in terms of a hash function which calculates the address of the block that a record should be assigned to or accessed from.

A hashed file must be declared in terms of a so-called hash key. This means that there can be only one hashed order per file. Inserting a record into a hashed file means that the key of the record is submitted to a hash function. The hash function translates the logical key value into a physical key value – a relative block address. Because this causes records to be randomly distributed throughout a file – they are sometimes known as random files or direct files.

Hash functions are chosen to ensure even distribution of records throughout a file. One technique for constructing a hash function involves applying an arithmetic function to the hash key. For instance, suppose we use student code as the hash key for a student file. The hash function might involve taking the first three characters from a student code, converting these characters to an integer and adding the result to a similar integer conversion performed on the last three characters of the student code. The result of this addition would then be used as the address of the block into which the record should be placed.

Hashed files must have a mechanism for handling collisions. A collision occurs when two logical values are mapped into the same physical key value. Suppose we have a logical key value: 2034. We find that feeding this key value through the hash function computes the relative block address: 12 (in practice, this address would be a hexadecimal number). A little later we try to insert a record with the key value 5678. We find that this also translates to relative block address 12. This is no problem if there is space in the block for the record. As the size of the file grows, however, blocks are likely to fill up. If the file manager cannot insert a record into the computed block then a collision is said to have occurred.



One of the simplest schemes for handling collisions is to declare an overflow area into which collided records are placed. As the hashed file grows, however, the number of records placed in the overflow area increases. This can cause a degradation in access performance. For this reason, a number of more complex algorithms are now employed to handle collisions. Various types of hashed file are also available which dynamically resize themselves in response to update activity.

27.5.4 CLUSTERING

Clustering is a technique whereby the physical organisation of data reflects some aspect of the logical organisation of data. We may distinguish between two types of cluster: intra-file clustering and inter-file clustering.

Intra-file clustering involves storing data in order of some key value whereby each set of records having the same key value is organised as a cluster. For instance, we might decide to cluster a students table in terms of a department or school code. Hence all computing students would be stored in a cluster, all humanities students would be stored as a cluster, and so on.

Inter-file clustering involves interleaving the data from two or more tables.

Example → The table below, for instance, is a clustered version of the Lecturers and Modules tables with which we are familiar.

**Cluster:
Teaching**

StaffNo	StaffName	status	moduleName	level	courseCode
234	Davies T	L	Relational Database Systems	1	CSD
			Relational Database Design	1	CSD
345	Evans R	PL	Deductive Databases	3	CSD
			Object-Oriented Databases	3	CSD
237	Jones S	SL	Distributed Databases	2	CSD

The rationale for clustering Lecturers data with Modules data in this way is to improve the joining of Lecturer records with Module records. It must be remembered however that clustering, like indexing (Chapter 28), is a physical concern. All that matters in terms of the data model is that the user perceives the data as being organised in relational terms. How the data is stored on disk is outside the domain of the data model.

The process of creating a cluster will differ among DBMS. However, in general terms, the process will first involve defining the cluster:

```
CREATE CLUSTER <cluster name>
(<column> <datatype>, ...)
[optional sizing information]
```

Example ➔

```
CREATE CLUSTER Teaching
(staffNo NUMBER(5))
```

We have hence created a cluster named Teaching and declared staffNo to be the so-called cluster key. This will be used to organise the data in the cluster. Next we create the tables and assign each table to the cluster:

```
CREATE TABLE <table name>
(...)
cluster <cluster name> (<table column>)
```

Example ➔

```
CREATE TABLE Lecturers
(...)
CLUSTER Teaching (staffNo)

CREATE TABLE Modules
(...)
CLUSTER Teaching (staffNo)
```

It is valid to use clusters to implement established joins, i.e. stable access paths into your data. However, use of clusters can degrade other access paths, for example full table scans on single tables in a cluster.

27.6 ➔

CASE STUDY: ORACLE INTERNAL MANAGEMENT

In this section we provide an overview of the internal storage management structures available in the ORACLE DBMS to illustrate the mapping between logical and physical data organisation (Rolland, 1998).

ORACLE tables are assigned to tablespaces (Figure 27.4). An ORACLE database may comprise more than one tablespace. Hence, in Figure 27.4 the database is

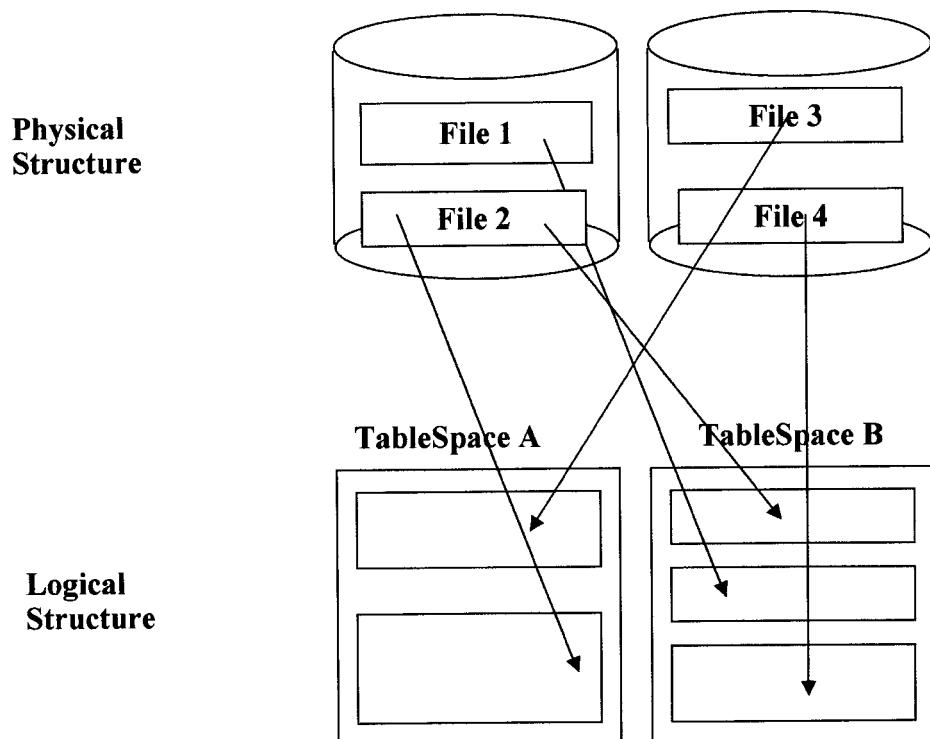


Figure 27.4 ORACLE physical and logical structures.

made up of two tablespaces: tablespace A and tablespace B. A tablespace is a logical data structure made up of one or more operating system files. In Figure 27.4, for instance, tablespace A is made up of file 2 and file 3.

There are a number of advantages which arise from this separation of logical and physical data structures:

- Tablespaces can be used to distribute I/O activity more effectively across a number of physical disks. This can be achieved, for example, by putting table data and index data in separate tablespaces
- Tablespaces can be used to set quotas against particular users of the database. When a table space is created, the DBA may set a space allocation against the tablespace
- Tablespaces can be taken off-line without disrupting the running of database activity, thus permitting more effective database administration

ORACLE uses two other logical structures known as segments and extents. A tablespace is divided up into a number of segments. A segment is a set of data blocks allocated for the storage of table or index data and can span more than one operating system file. A segment is made up of a number of extents. An extent is a set of contiguous space allocated within one operating system file.

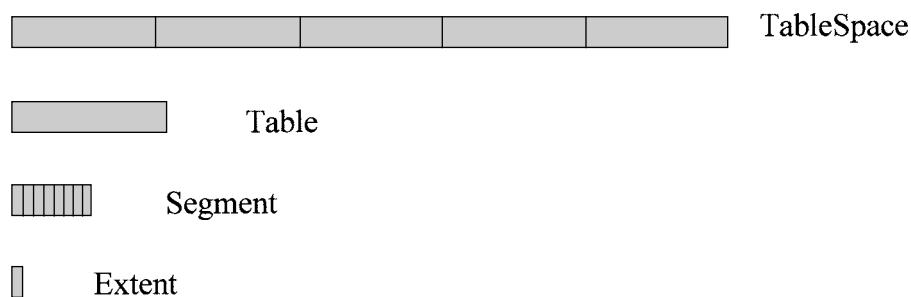


Figure 27.5 ORACLE logical data structures.

Extents are expressed in terms of bytes. The operating system will assign a number of blocks to each extent depending upon the size of the operating system block. This hierarchy of logical data structures is illustrated in Figure 27.5.

When a new table is created it is normally assigned to a default tablespace, unless otherwise specified. Also, on creation the table is assigned an initial extent of a default size, such as 1Mb. When rows are added to the table, the DBMS will allocate new data blocks to each extent until full. At such point the DBMS will automatically assign a new extent to the table. Both the default size of the initial extent and that of subsequent extents can be specified at the time of creating the table.

27.7 SUMMARY

- Data organisation concerns the way data is organised in physical storage devices
- Data models are forms of logical data organisation
- Database systems are heavily input/output intensive. Physical organisation can affect the efficiency of I/O activity
- Data is organised physically in terms of files, records, fields and blocks/pages
- The main types of file organisation are sequential files, ordered files and hashed files
- File organisation and access are intrinsically interlinked
- Clustering is an approach available in many contemporary DBMS for physically interleaving data on disk

27.8 ACTIVITIES

1. Investigate the forms of data organisation used in some DBMS known to you.
2. Determine how your chosen DBMS maps between logical and physical data structures.



3. Compare approaches used for clustering among a range of industrial-strength DBMS.

27.9 A circular icon containing a stylized spiral or swirl symbol, representing references.**REFERENCES**

- Ramakrishnan, R. and J. Gehrke (2003). *Database Management Systems*. Boston, MA, McGraw-Hill.
- Rolland, F.D. (1998). *The Essence of Databases*. Hemel Hempstead, UK, Prentice-Hall.

CHAPTER 28

ACCESS MECHANISMS

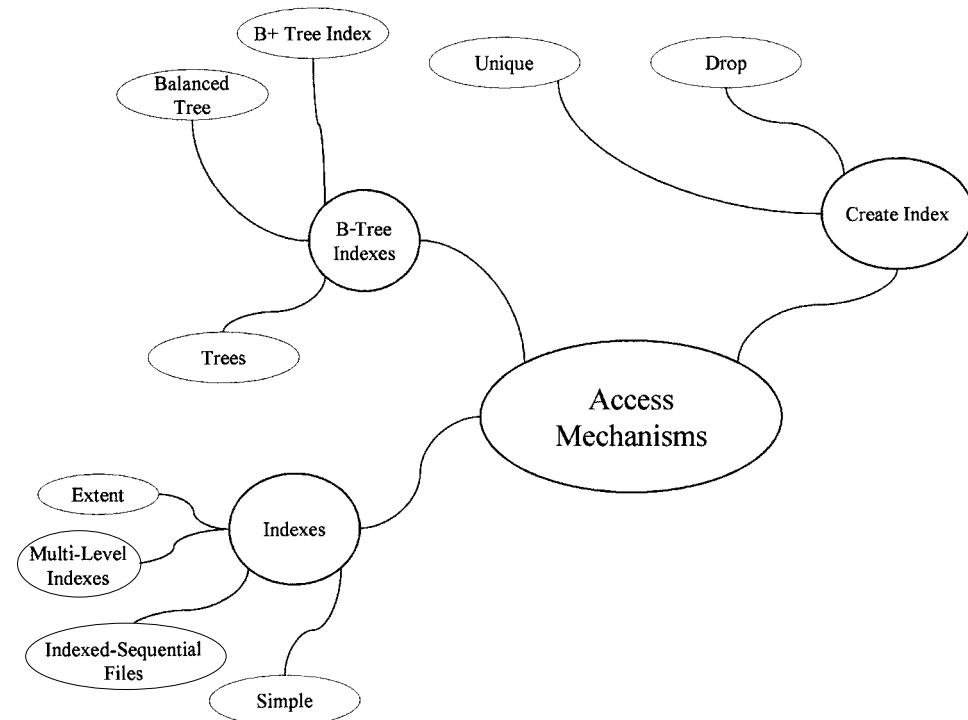
If you don't find it in the index, look very carefully through the entire catalogue.

Sears, Roebuck, and Co., *Consumer's Guide*, 1897

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Explain the concept of an access mechanism
- Define the concept of an index
- Describe a number of different types of access mechanism
- Explain the principle of a B-tree index
- Discuss the functionality of the SQL CREATE INDEX statement



28.1 INTRODUCTION

In the previous chapter we described a number of forms of access which are inherently related to underlying file organisations. Hence, for instance, sequential access is available for sequential files and hashed access is available for hashed files. In this chapter we discuss a form of access mechanism which is added to a database to improve performance without affecting the underlying storage structure. This is the index. We describe a number of different forms of index and explain how indexes are defined in SQL.

28.2 SIMPLE INDEXES

To simplify somewhat, the basic idea of an index can be regarded as a two-field file added to a database system. This file is known as the index file and is distinguished from the data file which stores the data. The first field of the index contains an ordered list of data values. These data values are defined on an index field or fields in the data file. The second field in a simple index file contains a list of block or page addresses for the records referenced by the index field.

Figure 28.1 illustrates a simple index. The key field in this case comprises a list of student numbers. These are the logical addresses to the records in the student file and are translated via the index into physical addresses indicating the pages on disk within which the records are stored.

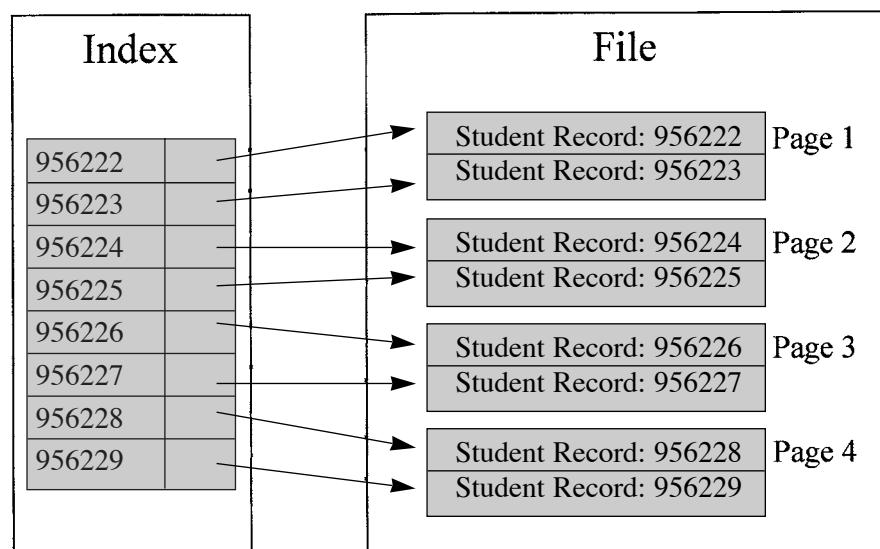


Figure 28.1 Simple index.

If the index field is defined on the key field, i.e. a field that is guaranteed to have unique values then it is called a primary index. If the index field is not defined on a key field then it is referred to as a secondary index. The index illustrated in Figure 28.1 is a primary key index.

Ideally, one would like to keep the index suitably small so that a major part of it can be held in main memory. Processing employing an algorithm such as binary chop (also referred to as binary search) can then be performed on the index. Binary chop is an efficient algorithm for searching an ordered list of values. A sample binary chop algorithm expressed in pseudo-code is presented below:

Binary Chop

```

first = 1; last = N; found = false; keepSearching = true
WHILE (last>=1 AND found = false AND keepSearching = true)
    I = (first + last)/2; {halve the search space}
    keyValue = readIndex(I);
    IF searchKey < keyValue THEN {key value is in lower half of search area}
        last = I - 1;
    ELSE IF searchKey > keyValue THEN {key value is in top half of search area}
        first = I + 1;
    ELSE IF searchKey = keyValue THEN {key value is found}
        found = true;
    ELSE {key value is not in index}
        keepSearching = false;
    
```

28.3 INDEXED SEQUENTIAL FILES

A sorted data file with a primary index defined on it is known as an indexed sequential file. This form of data structure permits the records either to be accessed sequentially or to be accessed directly via the index. The main problem with indexed sequential files lies in the need to update the index whenever new records are inserted into the data file.

28.4 MULTI-LEVEL INDEXES

As the size of an index grows so the search time associated with the index increases. Therefore, in practice, most indexes in DBMS are some form of multi-level index. A multi-level index consists of a number of levels of index. Figure 28.2 illustrates a two-level index. The values in level 2 of this index represent the highest key values of the pages in level 1.

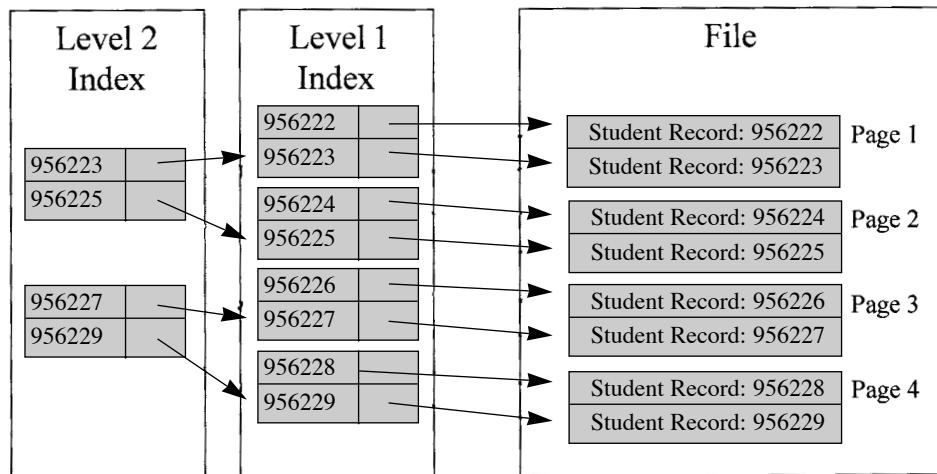


Figure 28.2 Multi-level index.

28.5

TREE STRUCTURES

Many DBMS use a structure known as a tree to store index values. A tree consists of a hierarchy of nodes in which every node, except the so-called root node, has one parent and zero or more child nodes. The nodes in the tree that have no children are known as the leaf nodes of the tree. The depth of a tree is the number of the levels between the root node and the leaf nodes of the tree. The depth may vary depending on the different paths between the root node and leaf nodes. The order of a tree is the maximum number of children allowed per parent node. A binary tree is one of order 2. Figure 28.3 illustrates a tree of order 2 and depth 2.

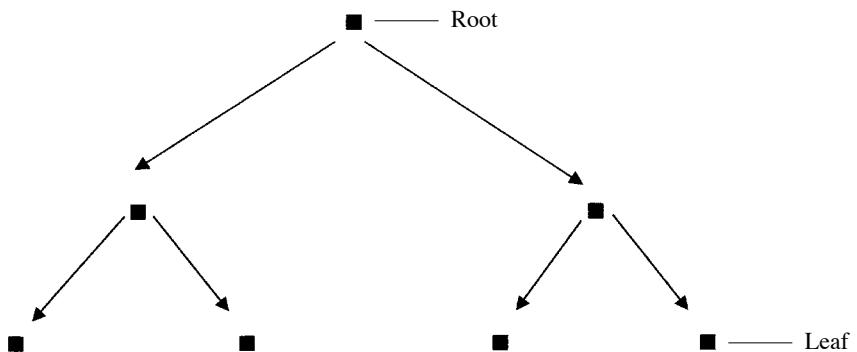


Figure 28.3 A sample tree.

28.5.1 BALANCED TREES

If all the paths in a tree are of the same depth then the tree is described as a balanced tree or B-tree (Comer, 1979). A B+ tree is a variation of a B-tree in which records are held only in the terminal nodes of the tree. Non-terminal nodes act as indexes to lower levels in the tree.

A B+ tree of order M is a tree with the following properties:

- No node has more than M children
- Every node, except the root node and terminal nodes, has at least $M/2$ children
- The root, unless the tree has only one node, has at least two children
- All terminal nodes appear on the same level, i.e. at the same distance from the root
- A non-terminal node with K children contains K – 1 records (keys). A terminal node contains at least $[M/2] - 1$ records and at most M – 1 records
- Terminal nodes are linked in order of key values

Example Suppose we have a simple file which we wish to index and which the logical key values are student numbers. A B-tree index for such a file is illustrated in Figure 28.4. Each record in the index now consists of one or more key values interspersed with pointers to other records in the tree. The tree is balanced in that all terminal nodes appear at the same distance from the root of the tree.

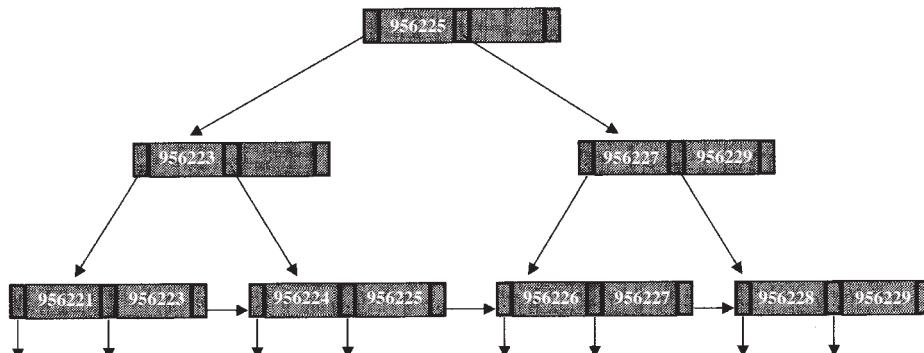


Figure 28.4 B+ tree.

28.5.2 SEARCHING B-TREES

When searching a B-tree for a key value (say 956224) we start at the root node. We search this node for the key value. If the key is not found (956224 does not match with 956225) a comparison with the keys of the node will identify the appropriate subtree to examine. If the appropriate pointer is null, then we are



at the lowest level of the tree and the key we are searching for is not present. If the pointer is not null then we read the node pointed to and repeat the process. In our case we need to search the left-most branch of the tree, since 956224 comes before 956225. 956224 does not match with the first key value in the next node (956223) so we inspect the next key value in this node (956225). We therefore need to take the mid-pointer in this node. This directs us the terminal node in which the search key is present.

28.5.3 UPDATING B-TREES

When inserting or deleting records from a data file, all B-tree indexes declared on the file will have to be updated as well. Inserting a key value involves first finding the appropriate place in the tree for the key and then updating the tree. However, since a B-tree has to remain balanced, any insertion or deletion activity may cause a reorganisation of the tree structure.

To take a simple example, suppose we wish to insert the record with the key value 956222 into the file described above. This means we have to update the index. Presently there is no space for this key value in the tree. This means that we have to reorganise the index into the structure illustrated in Figure 28.5. Note how we have had to introduce a new terminal node to handle this new key value.

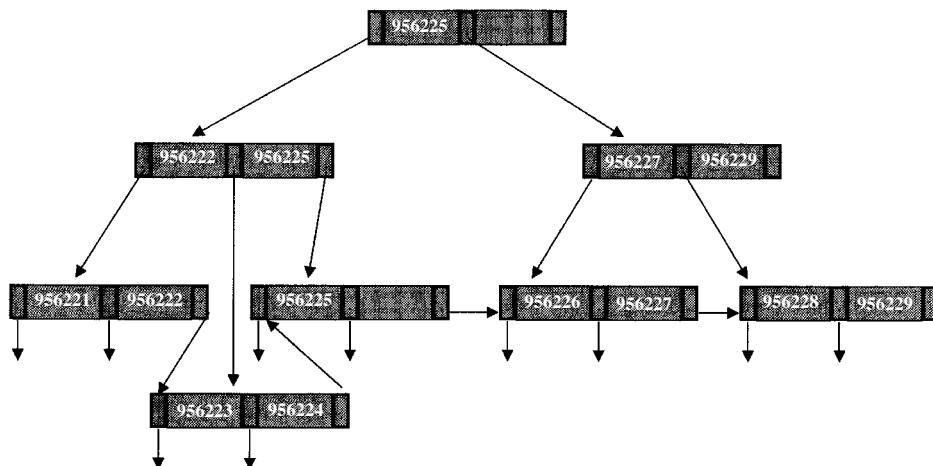


Figure 28.5 An updated B+ tree index.

28.6



CASE STUDY: THE SQL CREATE INDEX STATEMENT

In Codd's early papers on the Relational Data Model, little attention was paid to questions of performance, particularly retrieval performance. In any relational database system, however, the question of how long it takes to gain a response to a query is of fundamental importance.

The main way of achieving retrieval performance advantages in SQL-based systems is by adding indexes to a database. To build an index in SQL we use the CREATE INDEX statement:

```
CREATE INDEX <index name>
ON <table name> (<column name>)
```

Example → For instance, we might create an index on the courseCode column of the table modules. We would do this as follows:

```
CREATE INDEX MInd
ON Modules (courseCode)
```

The keyword UNIQUE may be added to a CREATE INDEX statement. Such an index will only store distinct values. This facility can be used to force users only to enter unique values in the column or columns of a table.

One of the main strengths of relational databases is that indexes can be created and deleted with relative abandon. To delete an index we use the following command:

```
DROP INDEX <index name>
```

Example → For instance:

```
DROP INDEX MInd
```

It is noteworthy that indexes take up additional storage in a database system. Frequently, database designers forget to include indexes in their sizing estimates (Chapter 19). Also, it is significant that index information will be stored in the system catalog and is used particularly to perform the query optimisation process (Chapter 30).

28.7 SUMMARY

- Access mechanisms are added to databases to improve retrieval performance
- The most commonplace form of access mechanism is the index
- The most popular form of index in contemporary DBMS is the B+ tree index
- SQL has a CREATE INDEX and DROP INDEX command

28.8  ACTIVITIES

1. Investigate whether a DBMS known to you uses indexes.
2. How can indexes be created using this DBMS?
3. Does the DBMS use B-tree indexes?
4. Does the DBMS use the SQL CREATE INDEX statement?

28.9  REFERENCE

Comer, D. (1979). The ubiquitous B-tree. *ACM Computing Surveys* **11**(2): 121–38.

CHAPTER 29

TRANSACTION MANAGEMENT

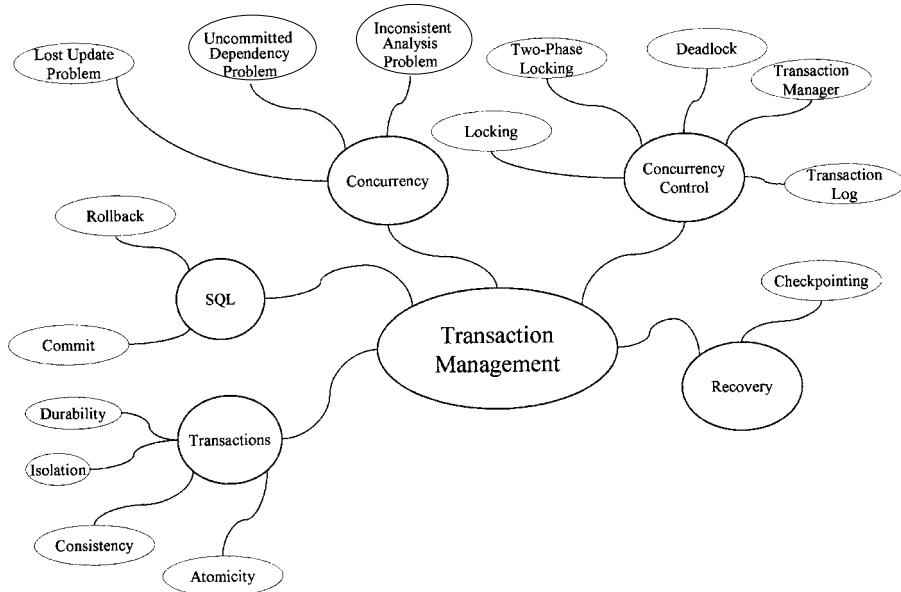
He who every morning plans the transaction of the day and follows out that plan, carries a thread that will guide him through the maze of the most busy life. But where no plan is laid, where the disposal of time is surrendered merely to the chance of incidence, chaos will soon reign.

Victor Hugo (1802–85)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define the concept of base integrity
- Discuss the concept of a transaction
- Discuss the problems of concurrency control and the traditional solution of locking
- Review the issue of database consistency and the traditional solution in terms of transaction logging and recovery



**29.1**

In Chapter 3 we distinguished between two major types of integrity: inherent and additional integrity. These two types of integrity are concerned with the logical state of some database. The overall objective of both inherent and additional integrity is to make sure that some database is an accurate reflection of the real world that it is attempting to represent.

Ensuring integrity is also however a concern of the physical dimension of a database system, particularly in multi-user environments. The kernel of a DBMS has to concern itself with the problems of hardware and software failure or the problems of distributing a scarce resource among multiple users. We might call this type of integrity base integrity to distinguish it from semantic integrity of which inherent and additional integrity are forms.

In this chapter we examine one of the most important aspects of base integrity: transaction management. Transaction management involves ensuring concurrent access to a database, and ensuring the consistency of this database in a multi-user environment.

29.2

In a multi-user database system the procedures that cause changes to a database or retrieve data from the database are called transactions. A transaction may be defined as a logical unit of work (Elmasri and Navathe, 2000). It normally corresponds to some coherent activity performed by an individual, group, organisation or software process on a database. In practice it may constitute an entire program, a set of commands or a single command which accesses or updates some database.

Example

Consider a database maintained as part of an airline reservation system. A transaction here might consist of a travel agency booking several connecting flights for a customer. This transaction must therefore:

- Read a number of specified data structures from the database to obtain the dates, times, flight numbers and seat availability information
- Check that the proposed group of flights has a seat available
- If a seat is available on each flight, reserve seats, otherwise report failure

To remain an accurate reflection of its real-world domain, any transaction such as the one above should demonstrate the properties of atomicity, consistency, isolation and durability (sometimes abbreviated to ACID):

- *Atomicity*. Since a transaction consists of a collection of actions, the DBMS should ensure that either all the transaction is performed or none of it is performed
- *Consistency*. All transactions must preserve the consistency and integrity of the database. Operations performed by an updating transaction, for instance, should not leave the database in an inconsistent or invalid state (section 1.3)
- *Isolation*. While a transaction is updating shared data, that data may be temporarily inconsistent. Such data must not be made available to other transactions until the transaction in question has finished using it. The transaction manager must therefore provide the illusion that a given transaction runs in isolation from other transactions
- *Durability*. When a transaction completes, the changes made by the transaction should be durable. That is, they should be preserved in the case of hardware or software failure

Example 

- *Atomicity* – in our example, only if a seat is available on all the required flights in a person's proposed journey is the booking confirmed
- *Consistency* – hence in an airline system we should not be able to overbook a flight
- *Isolation* – in terms of our example, the data structures containing information about connecting flights should not be available for update by other transactions until this transaction has finished using the information
- *Durability* – in the case of the airline booking, we must ensure that if some failure occurs during the process of executing the booking, the system can restore the system to the state that existed before the transaction began executing

In the sections that follow, we consider these properties in more detail, and the consequences they have for transaction management.

29.2.1 TRANSACTIONS IN SQL-BASED SYSTEMS

In most SQL-based products, a transaction is simply a sequence of SQL statements that is packaged as a single entity. Relational DBMS maintain consistency by ensuring either that all the SQL statements in a transaction complete successfully or that none do. In other words, if a transaction is made up of a debit of money from one bank account and a credit to another bank account, then either both updates succeed or both fail.

In SQL the statements COMMIT and ROLLBACK are used to delineate transactions. COMMIT makes permanent changes to a database. ROLLBACK undoes all changes made in an unsuccessful transaction.



Example Hence, a transaction to enrol a student in a module might involve two statements, one to update a registration table, the other to increment the roll of a module:

```
INSERT INTO Registration (studentNo, moduleName)
    VALUES (34698,'Relational Database Systems')
UPDATE Modules
    SET roll = roll + 1
COMMIT
```

If for some reason a system failure occurred after the first update, issuing a ROLLBACK command will take the database back to its previous state. Conversely, if we are happy that our transaction has completed successfully, the COMMIT command will make the updates permanent. This ensures the property of atomicity.

Some SQL commands such as *create table* and *create index* usually automatically commit when they are executed. For most other commands, the default is not to commit. This system-wide edict can be changed however in many DBMS by a command that sets automatic commitment. This has the effect of committing all changes immediately after the execution of each SQL statement.

29.3



CONCURRENCY

It is more than likely that the airline system described in section 29.2 is a multi-user database system. In other words, the data about flight bookings is probably shared between a whole range of terminals, perhaps spread geographically among major airports and travel agencies. An immediate consequence of the data-sharing property of database systems is that mechanisms must be provided for handling shared or concurrent access. When several transactions are executed effectively in parallel on a shared database, their execution must be synchronised.

Some of the problems introduced by concurrency are considered below (Connolly and Begg, 2002).

29.3.1 THE LOST UPDATE PROBLEM

Suppose we have a file of relevance to the airline system described above. In this file each record corresponds to a particular flight. Record (flight) 21 within this file currently has the number of seats booked (a field named seats) set to 10. This is the state at time $T = 0$. At time $T = 1$, Transaction A accesses flight record 21. Transaction B however also accesses record 21 at time $T = 2$. Transaction A increments the number of seats booked on record 21 by 1 at time $T = 2$ and writes this back to the database at time $T = 3$. Transaction B increments the number of seats by 5 at time $T = 3$ and writes back this change at time $T = 4$.

The problem is that Transaction A's update has been lost since Transaction B's update has overwritten it. This means that the value of seats is 15 rather than the true value of 16. This anomaly is normally referred to as the *lost update problem*. It is one example of the need for concurrency control. This problem is illustrated in the table below.

Time	Transaction A	Transaction B	Record 21
T = 0			Seats = 10
T = 1	Read record 21 (Seats = 10)		Seats = 10
T = 2	Seats = Seats + 1	Read record 21 (Seats = 10)	Seats = 10
T = 3	Write record 21 (Seats)	Seats = Seats + 5	Seats = 11
T = 4	Commit	Write record 21 (Seats)	Seats = 15
T = 5		Commit	Seats = 15

The lost update problem can be avoided if the transaction manager module within the DBMS prevents Transaction B from reading the value of the field in the record being updated until after Transaction A's update has completed. In other words, the transaction manager must ensure the isolation of transactions.

29.3.2 THE UNCOMMITTED DEPENDENCY PROBLEM

The uncommitted dependency problem occurs as the result of one transaction viewing the intermediate results of another transaction before that transaction has committed. In the example below we use exactly the same transactions as in the previous example, but change the time at which Transaction B starts to T = 4. Also, for some reason, this transaction fails to complete successfully and rolls back at T = 5. However, since Transaction B starts at T = 4, it reads the updated value of the field *seats* and proceeds to increment this value by 5. Hence the final value for this field, 16, is incorrect. This process is illustrated in the table below.

Time	Transaction A	Transaction B	Record 21
T = 0			Seats = 10
T = 1	Read record 21 (Seats = 10)		Seats = 10
T = 2	Seats = Seats + 1		Seats = 10
T = 3	Write record 21 (Seats)		Seats = 11
T = 4		Read record 21 (Seats = 11)	Seats = 11
T = 5	Rollback	Seats = Seats + 5	Seats = 10
T = 6		Write record 21 (Seats)	Seats = 16
T = 7		Commit	Seats = 16

This anomaly can be avoided by preventing Transaction B from reading the value of a field until after the decision is made to commit or rollback Transaction A.



29.3.3 THE INCONSISTENT ANALYSIS PROBLEM

The two problems discussed above are examples caused by problems of updating. However, transactions that solely read from a database can also suffer problems, particularly if they read partial results of incomplete transactions. This type of problem is sometimes known as the *dirty read* or the *unrepeatable read*.

Example Consider the case below. Transaction B is attempting to build a summation of the current state of seat utilisation across three flights (records 21, 22, and 23). However, while it is executing, Transaction A is updating two of the relevant flights. This causes the final total of 44 to be incorrect, as illustrated below.

Time	Transaction A	Transaction B	Record 21	Record 22	Record 23	Sum
T = 0			Seats = 10	Seats = 15	Seats = 20	
T = 1		Sum = 0	Seats = 10	Seats = 15	Seats = 20	0
T = 1	Read record 21 (Seats = 10)	Read record 21 (Seats = 10)	Seats = 10	Seats = 15	Seats = 20	0
T = 2	Seats = Seats + 1 (Seats)	Sum = Sum + Seats (Seats = 15)	Seats = 10	Seats = 15	Seats = 20	10
T = 3	Write record 21 (Seats)	Read record 22 (Seats = 15)	Seats = 11	Seats = 15	Seats = 20	10
T = 4	Read record 23 (Seats = 20)	Sum = Sum + Seats (Seats = 20)	Seats = 11	Seats = 15	Seats = 20	25
T = 5	Seats = Seats - 1		Seats = 11	Seats = 15	Seats = 20	25
T = 6	Write record 23 (Seats)		Seats = 11	Seats = 15	Seats = 19	25
T = 7	Commit	Read record 23 (Seats = 19)	Seats = 11	Seats = 15	Seats = 19	25
T = 8		Sum = Sum + Seats (Seats = 19)	Seats = 11	Seats = 15	Seats = 19	44
T = 9		Commit	Seats = 11	Seats = 15	Seats = 19	44

This problem can be avoided by preventing Transaction B from reading the seats field of records 21 and 23 until after Transaction B has performed its updates.



Some of the main problems associated with the concurrent access to a database by a set of transactions have been discussed above. The primary objective of concurrency control is to prevent such interference between transactions from occurring. Concurrency control aims for the objective of serialisability.

A transaction comprises a set of reads and writes against some database. The sequence of reads and writes relevant to a set of concurrent transactions is known as a schedule. A serial schedule is one in which each transaction is executed consecutively, thus prohibiting any interleaving of operations between transactions. In a serial schedule, any one transaction must complete before the next transaction is allowed to begin. A serial schedule will always guarantee that, when executed, it never leaves the database in an inconsistent state.

However, in a multi-user DBMS there may be many transactions that can effectively execute in parallel without interfering with one another. For example, a set of transactions that accesses entirely different parts of a database can be executed concurrently without any interference. A schedule is said to be a non-serial schedule when the operations from a set of concurrent transactions are interleaved. Concurrency control aims for the objective of so-called serialisability; that is, to identify those non-serial schedules that may behave as serial schedules in that they produce a database state that could be produced by serial execution.

A non-serial schedule is said to be *correct* if it produces the same result as some serial execution. A correct non-serial schedule is described as being serialisable. In practice, DBMS do not test for the serialisability of a schedule directly; instead they employ protocols that have been established to produce serialisability.

There are two major protocols employed: locking and timestamping. Both locking and timestamping are generally referred to as pessimistic approaches to ensuring serialisability in that they are based on the premise that conflict between transactions is a common occurrence and therefore that certain transactions should be delayed in their access to data. In contrast, optimistic methods are based on the premise that conflict is rare and hence that transactions should normally proceed unhindered, checking being performed before a transaction commits.

29.4.1 LOCKING

We consider the pessimistic mechanism of locking because it is the most commonplace approach in contemporary DBMS. When a transaction locks a data-item it is saying to other transactions 'I am doing something with this data-item, wait until I have completed my task'. The net result of applying locks is to serialise access to limited resources. We can distinguish between two main types of locks:

- *Read locks*. A read lock gives only read access to data and prevents any other transaction from updating the locked data. Any number of transactions may hold a read lock on a data item and thus they are sometimes referred to as shared locks. A transaction applies this type of lock when it wishes to query a file but does not want to change it. Also, it does not wish other transactions to change it while it is looking at it



- *Write locks.* A write lock gives both read and write access to a data-item. It also prevents any other transaction from reading from or writing to a data-item. For this reason it is often referred to as an exclusive lock

Locks are normally used in the following way:

- Any transaction that wishes to access a data-item must first lock the data-item. If the access is merely read only, a read lock is requested. If the transaction wishes to both read and update the data-item, then a write lock is requested
- The DBMS first checks to see if the data-item is locked by another transaction. For this purpose the DBMS needs to maintain a lock table. If it is locked, the DBMS determines whether the lock requested is compatible with that set on the data-item. If a read lock is requested on a data-item already having a read lock on it, then the request is allowed. In all other cases, the requesting transaction must wait
- If, of course, the data-item is not currently locked then the DBMS allows access and puts a lock against it
- A transaction continues to hold a lock until either it releases the lock during its execution or the transaction terminates

A DBMS normally maintains a locking scheme by updating a lock table held in main memory. Imposing a write lock on a data-item can have a serious effect on the amount of concurrency in a database system. The degree of seriousness frequently depends on the granularity of the data-item. Locks can usually be declared at the level of files, pages, records or even fields within records. If the locked data-items are large, then the performance of update activity can possibly degrade. If the locked data-items are at the level of fields, then the size of the lock table held by the DBMS and the amount of processing conducted on the lock table can degrade performance.

29.4.2 TWO-PHASE LOCKING

Using locks by themselves does not guarantee serialisability of transactions. To guarantee such serialisability the DBMS must enforce an additional protocol known as two-phase locking. This involves dividing the life of every transaction into two phases: a growing phase, in which it acquires all the locks it needs to perform its work, and a shrinking phase, in which it releases its locks. During the growing phase it is not allowed to release any locks, and during the shrinking phase it is not allowed to acquire any new locks. This does not mean that all locks be acquired simultaneously since a transaction normally will engage in the process of acquiring some locks, conducting some processing, then going on to acquire more locks, and so on. The two-phase locking protocol merely involves enforcing two rules:

- A transaction must acquire a lock on a data-item before performing any processing on that data-item
- Once a transaction releases a lock it is not permitted to acquire any new locks

Below we demonstrate how two-phase locking prevents the three anomalies associated with concurrent transactions discussed above.

29.4.3 THE LOST UPDATE PROBLEM

In the table below, Transaction A requests a write lock on record 21 at time $T = 1$. Transaction B attempts a write lock at time $T = 2$. It is forced to wait because Transaction A has a write lock on the record. It waits until time $T = 5$ when Transaction A unlocks the record. The final state of record 21 after these two transactions have executed leaves the field *seats* at the correct value of 16.

Time	Transaction A	Transaction B	Record 21
$T = 0$			Seats = 10
$T = 1$	Write Lock (record 21)		Seats = 10
$T = 2$	Read record 21 (Seats = 10)	Write Lock (record 21)	Seats = 10
$T = 3$	Seats = Seats + 1	Wait	Seats = 10
$T = 4$	Write record 21 (Seats)	Wait	Seats = 11
$T = 5$	Unlock (record 21)	Wait	Seats = 11
$T = 6$	Commit	Read record 21 (Seats 11)	Seats = 11
$T = 7$		Seats = Seats + 5	Seats = 11
$T = 8$		Write record 21 (Seats)	Seats = 16
$T = 9$		Unlock (record 21)	Seats = 16
$T = 10$		Commit	Seats = 16

29.4.4 THE UNCOMMITTED DEPENDENCY PROBLEM

In the table below, Transaction A requests a write lock on record 21. It then proceeds to attempt to update the record. At time $T = 3$ Transaction B attempts a write lock on record 21. It is prevented from doing this because of the write lock that Transaction A holds on this record. At time $T = 5$ Transaction A rolls back, causing the database to return to its previous value for the *seats* field. The rollback terminates the transaction, allowing transaction B to be granted access to the record. After completing its update, Transaction B leaves the database in a consistent state.



Time	Transaction A	Transaction B	Record 21
T = 0			Seats = 10
T = 1	Write Lock (record 21)		Seats = 10
T = 1	Read record 21 (Seats = 10)		Seats = 10
T = 2	Seats = Seats + 1		Seats = 10
T = 3	Write record 21 (Seats)	Write Lock (record 21)	Seats = 11
T = 4	Unlock (record 21)	Wait	Seats = 11
T = 5	Rollback	Wait	Seats = 10
T = 6		Read record 21 (Seats = 10)	Seats = 10
T = 7		Seats = Seats + 5	Seats = 10
T = 8		Write record 21 (Seats)	Seats = 15
T = 9		Commit	Seats = 15

29.4.5 THE INCONSISTENT ANALYSIS PROBLEM

The table below illustrates a solution to the inconsistent analysis problem. To prevent this problem, Transaction A precedes all its reads by write locks. Also, Transaction B must precede its reads with read locks. This strategy ensures that the summation transaction must wait until update activity has completed.

Time	Transaction A	Transaction B	Record 21	Record 22	Record 23	Sum
T = 0			Seats = 10	Seats = 15	Seats = 20	
T = 1		Sum = 0	Seats = 10	Seats = 15	Seats = 20	0
T = 2	Write Lock (21)		Seats = 10	Seats = 15	Seats = 20	0
T = 3	Read Seats = 10	Read Lock (21)	Seats = 10	Seats = 15	Seats = 20	0
T = 4	Seats = Seats + 1	Wait	Seats = 10	Seats = 15	Seats = 20	0
T = 5	Write Seats	Wait	Seats = 11	Seats = 15	Seats = 20	0
T = 6	Write Lock (23)	Wait	Seats = 11	Seats = 15	Seats = 20	0
T = 7	Read Seats = 20	Wait	Seats = 11	Seats = 15	Seats = 20	0
T = 8	Seats = Seats - 1	Wait	Seats = 11	Seats = 15	Seats = 20	0
T = 9	Write Seats	Wait	Seats = 11	Seats = 15	Seats = 19	0
T = 10	Commit, Unlock (21, 22)	Wait	Seats = 11	Seats = 15	Seats = 19	0
T = 11		Read Seats = 11	Seats = 11	Seats = 15	Seats = 19	0
T = 12		Sum = Sum + Seats	Seats = 11	Seats = 15	Seats = 19	11
T = 13		Read Lock (22)	Seats = 11	Seats = 15	Seats = 19	11
T = 14		Read Seats = 15	Seats = 11	Seats = 15	Seats = 19	11
T = 15		Sum = Sum + Seats	Seats = 11	Seats = 15	Seats = 19	26
T = 16		Read Lock (23)	Seats = 11	Seats = 15	Seats = 19	26
T = 17		Read Seats = 19	Seats = 11	Seats = 15	Seats = 19	26
T = 18		Sum = Sum + Seats	Seats = 11	Seats = 15	Seats = 19	45
T = 19		Commit, Unlock (21, 22, 23)	Seats = 11	Seats = 15	Seats = 19	45

29.4.6 DEADLOCK

A data-item need not be locked for the entire duration of a transaction. It may simply be locked during data access. This can increase the level of concurrency in a system but increases the likelihood of deadlock occurring.

Deadlock, sometimes known as deadly embrace, is best described by a simple example. Consider two transactions, Transaction A and Transaction B:

- At time T1 Transaction A write locks record 1
- At time T2 Transaction B write locks record 2
- At time T3 Transaction 1 requests a write lock on record 2. It must wait since Transaction 2 has locked record 2
- At time T4 Transaction 2 requests a write lock on record 1. It must also wait

Time	Transaction A	Transaction B
T = 0		
T = 1	Write Lock (1)	
T = 2	Read record 1 (Seats)	Write Lock (2)
T = 3	Write Lock (2)	Read record 2 (Seats)
T = 4	Wait	Write Lock (1)
T = 5	Wait	Wait
T = 6	Wait	Wait
T = 7	Wait	Wait

At this point neither transaction can proceed. – they are said to be *deadlocked*.

There are a number of strategies that can be employed to prevent deadlock. One strategy is to force all transactions to perform all their locks prior to execution. This strategy however tends to have a detrimental effect on performance, as large transactions continuously wait to execute. More effective strategies in database systems involve detecting deadlock after it has occurred and resolving such deadlock.

29.5 THE TRANSACTION MANAGER AND TRANSACTION LOG

The transaction manager (Silberschatz *et al.*, 2002) is a module within the kernel of a DBMS that handles the throughput of transactions against a database. One of its primary responsibilities is ensuring that a transaction either completes successfully or fails. A successful transaction is one in which all the operations of the transaction have completed successfully. A failed transaction is one in which one or more of the related operations within the transaction have not completed successfully.

One of the major functions of the transaction manager is to record the execution of transactions. The place where these records of execution are stored is normally referred to as the transaction log, or sometimes as the transaction journal. The major steps in the execution of a transaction are:

- Start the transaction – input transaction into the transaction manager
- Log the transaction – record initial details of the transaction in log
- Fetch database records



- Log before image of transaction – that is, the values before processing the transaction
- Compute new values
- Log after image of transaction – that is, the values after processing
- Log commitment
- Write new records to database

The transaction log is a file that contains data about the updates affecting a database. Such data may comprise both transaction records and checkpoint records.

Transaction records may contain:

- An identifier for the log record
- An identifier for the transaction
- A time for the transaction operation
- An operation type such as the start of the transaction, insert, update, delete, abort or commit
- An identifier for the data-item affected
- In the case of updates and deletions, the value of the data-item before the change (before image)
- In the case of insertions or updates, the value of the data-item after the change (after image)
- Pointers to previous and next log records associated with a particular transaction
- Checkpoint records (see section 29.6.2)

Example ➔ A segment of a transaction log is illustrated below:

Log ID	Transaction	Time	Operation Type	Data Item	Before Image	After Image	Previous Pointer	Next Pointer
1	Trans 1	13:10	Start				0	2
2	Trans 1	13:11	Update	Flight record 21 (Seats)	(old)	(new)	1	6
3	Trans 2	13:14	Start				0	4
4	Trans 2	13:15	Insert	Flight record		(new)	3	6
5	Trans 2	13:16	Delete	Flight record 45	(old)		5	8
6	Trans 1	13:17	Commit				2	0
7		13:18	Checkpoint	Trans 2				
8	Trans 2	13:18	Commit				5	0

In the case of total system failure, the transaction manager will use the transaction log to recover. Therefore, because of its importance, the transaction log must be included in any strategy for backup and recovery.

Because of the interaction of concurrency and consistency, SQL2 defines a number of so-called isolation levels that can be set for given transactions. This allows the user to control the balance between concurrency and consistency. Hence, a user might set a transaction to have an isolation level of *read uncommitted*. This will basically tell the transaction manager to allow the type of read problem discussed in section 29.3 (the lost update or dirty read problem), to the benefit of increased levels of concurrency.

29.6 RECOVERY

Recovery is the process of ensuring that a database can achieve a consistent state in the event of failure. The basic unit of recovery in a database system is the transaction. The module of the DBMS tasked with handling recovery must ensure that transactions display the properties of atomicity and durability. This process is complicated by the fact that the management of update activities against a database is not a single-step process. Frequently the writing of changes to a database will involve the steps of reading data from disk into a database buffer, updating data in main memory and writing the database buffer back to disk.

The process of transferring data from buffers in main memory to secondary storage is known as flushing. Flushing may occur when triggered by a command such as a *commit*, may occur automatically when the buffer becomes full or may be set to occur at periodic intervals. The latter approach is known as force-writing the buffer.

Failure can occur either when writing the data to buffers or while flushing the buffers. If a transaction issued a *commit* but failure occurred before flushing completes, then the changes to the database will not be made permanent. In this case the recovery module will need to redo the transaction updates. If the transaction has not committed when failure occurs, then the recovery module has to undo any changes made to the database to ensure atomicity of transactions.

29.6.1 RECOVERY FACILITIES

Most DBMS will provide the following recovery facilities:

- A mechanism which enables backup copies of either the whole or part of a database
- A facility which logs transactions and changes to the database (see above)
- A facility for checkpointing updates to a database
- A facility which permits the system to restore the database to a consistent state after hardware and software failure

The next section considers the issue of checkpointing.



29.6.2 CHECKPOINTING

Checkpointing is a technique used to limit the amount of searching that needs to be performed against a log file in order to carry out recovery effectively. Checkpointing involves force-writing database buffers to secondary storage at predetermined intervals. At such intervals:

- All log records in main memory will be written to secondary storage
- All modified data in the buffers will be written to secondary storage
- A checkpoint record will be written to the log indicating all active transactions at the time of the checkpoint

29.6.3 RECOVERING FROM FAILURE

In the case of failure that has caused extensive damage to the database, recovery will involve using the last backup copy of a database in association with the log file to reapply committed transactions. In the case where some failure has left the database in an inconsistent state, we do not need to utilise a backup copy. Instead, we can use the before and after images in the log file to restore the database to a consistent state.

There are a number of different ways in which recovery can be undertaken in the latter situation. One approach, known as the deferred update protocol, involves updating a database only after transactions have reached their commit point. In the event of failure, uncommitted transactions do not need to be considered because they will not have modified the database. However, committed transactions may need to be redone because their updates may not have modified the database.

29.7



SUMMARY

- Transaction management involves ensuring concurrent access to a database, and ensuring the consistency of this database in a multi-user environment
- In a multi-user database system, the procedures that cause changes to a database or retrieve data from the database are called transactions
- Any transaction should demonstrate the properties of atomicity, consistency, isolation and durability
- In SQL the statements COMMIT and ROLLBACK are used to delineate transactions
- An immediate consequence of the data-sharing property of database systems is that mechanisms must be provided for handling shared or concurrent access. A number of problems are introduced by concurrency, including the lost update problem, the uncommitted dependency problem and the inconsistent analysis problem
- Concurrency control aims for the objective of so-called serialisability
- Locking is a pessimistic approach to ensuring serialisability

- The transaction manager is a module within the kernel of a DBMS that handles the throughput of transactions against a database. One of the major functions of the transaction manager is to record the execution of transactions. The place where these records are stored is normally referred to as the transaction log
- Recovery is the process of ensuring that a database can achieve a consistent state in the event of failure. The basic unit of recovery in a database system is the transaction
- Checkpointing is a technique used to increase the efficiency of recovery. It involves force-writing database buffers to secondary storage at predetermined intervals

29.8 ACTIVITIES

1. Examine a DBMS known to you and try to determine what approach to concurrency control it uses.
2. If it uses a locking scheme, determine the granularity of the scheme.
3. Try to determine the structure of its transaction log.
4. Does it use checkpointing as an approach?

29.9 REFERENCES

- Connolly, T.M. and C.E. Begg (2002). *Database Systems: A Practical Approach to Design, Implementation, and Management*. Harlow, UK, Addison-Wesley.
- Elmasri, R. and S.B. Navathe (2000). *Fundamentals of Database Systems*. Reading, MA, Addison-Wesley.
- Silberschatz, A., H.F. Korth and S. Sudarshan (2002). *Database System Concepts*. Boston, MA, McGraw-Hill.

CHAPTER 30 ➔

OTHER KERNEL FUNCTIONS

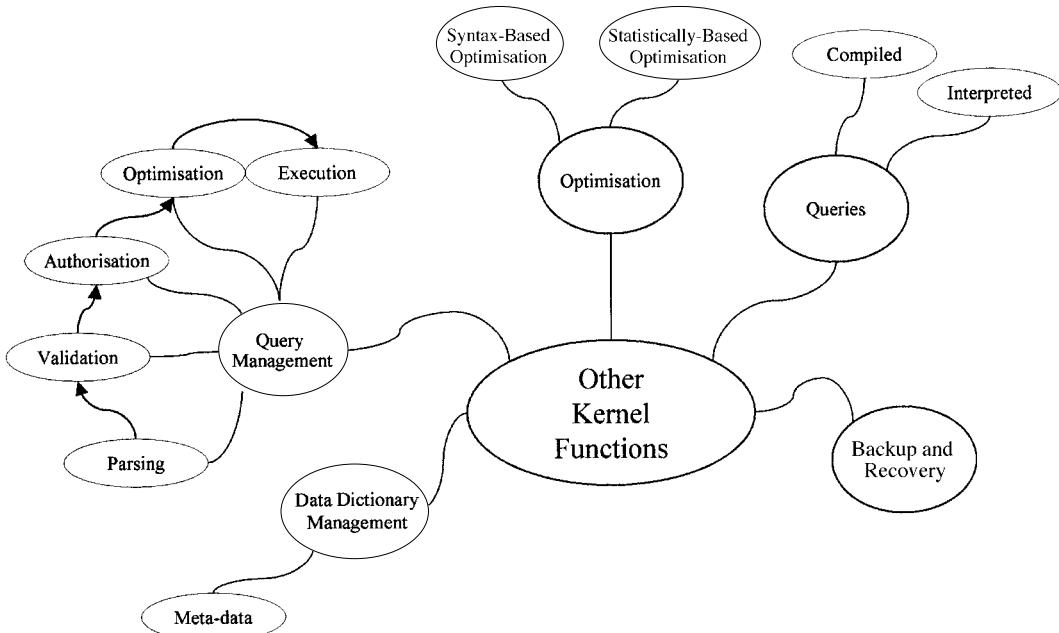
I keep the subject of my inquiry constantly before me, and wait till the first dawning opens gradually, by little and little, into a full and clear light.

Isaac Newton (1642–1727)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Discuss the structure of the physical data dictionary
- Discuss the key features of the query management function of a DBMS
- Describe the process of query optimisation
- Review backup and recovery facilities available in DBMS



30.1 INTRODUCTION

In this chapter we discuss three other functions of the kernel of a DBMS. First, we briefly review the structure of the physical data dictionary at the heart of the DBMS. Second, we discuss the query management module of the DBMS and the associated period of query optimisation. Finally, we briefly review the backup and recovery facilities available in DBMS.

30.2 DATA DICTIONARY MANAGEMENT

The system catalog or data dictionary is at the heart of a DBMS. When a DBMS is first configured on some hardware, the DBA will initially create groups of users, provide passwords for such users, and declare data and facilities privileges for each user group (Chapter 23). All this information will be written to the data dictionary.

When users issue their data definition statements, this information will be stored in the data dictionary. Also, when users maintain the data in the database or retrieve data from the database, all this activity will be done using information stored in the data dictionary.

The basic information that is stored in a data dictionary for a relational DBMS includes:

- A description of base relations, including relation names, column names, column data types and null characteristics of columns
- Primary-key and foreign-key declarations, including propagation constraints
- A description of views (Chapter 26)
- Declarations of user groups and authorisations
- Information about indexes, sizes of files, file structures and clusters (Chapter 27)
- Post-relational systems will also store information about procedures, triggers and constraints (Chapter 10)

Figure 30.1 illustrates some of the key elements of the structure of a data dictionary as an entity–relationship diagram (Chapter 16).

It is conventional in relational DBMS to store this meta-data in a set of tables which can be queried using the same DBMS software as is used to query base tables. Update activity against the system tables is however usually controlled by forcing such activity to use a limited set of data definition commands. The DBA is not normally allowed to create or delete a system table.

Each RDBMS catalog tends to be specific to that product. Hence, although each RDBMS may store functionally the same information in their catalogs, the structure of each catalog is different. Attempts are however ongoing to develop a standard for the system catalog of relational DBMS (Chapter 11). The main objective of this work is to enhance the connectivity of diverse DBMS.

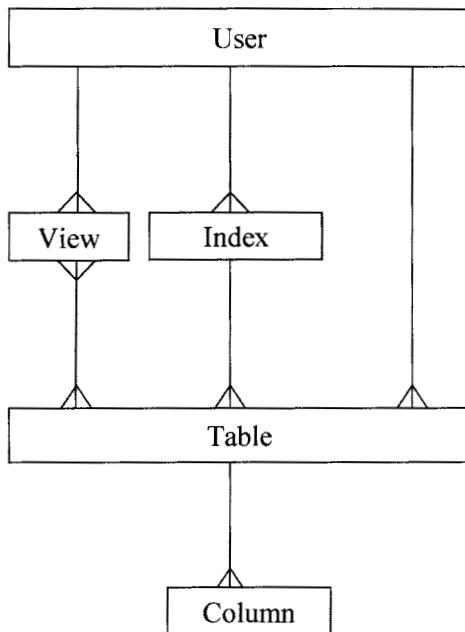


Figure 30.1 Structure of a data dictionary.

30.3 QUERY MANAGEMENT

When a query is expressed, probably in a DML such as SQL (Chapter 13) via some interface (Chapter 24) to a DBMS, that query goes through a number of stages:

- First the query is checked to see if it is syntactically correct. This is frequently referred to as the parsing stage
- Then the query is validated against information stored in the data dictionary. For instance, checks are made to see if the names of columns and tables are appropriate. Checks are also made to ensure that users have the appropriate access privileges
- Then the query is optimised. In other words, a piece of software translates the user query into an execution plan. Such a plan will probably be produced to take advantage of appropriate data structures and indexes declared for the given database
- The query is executed
- The results of the query or error messages are returned to the user

This process is illustrated in Figure 30.2.

In relational systems such a process can be considered as a means of turning a non-procedural query (perhaps expressed in SQL) into a procedural

implementation (which can be considered, at least theoretically, as a series of relational algebraic operations).

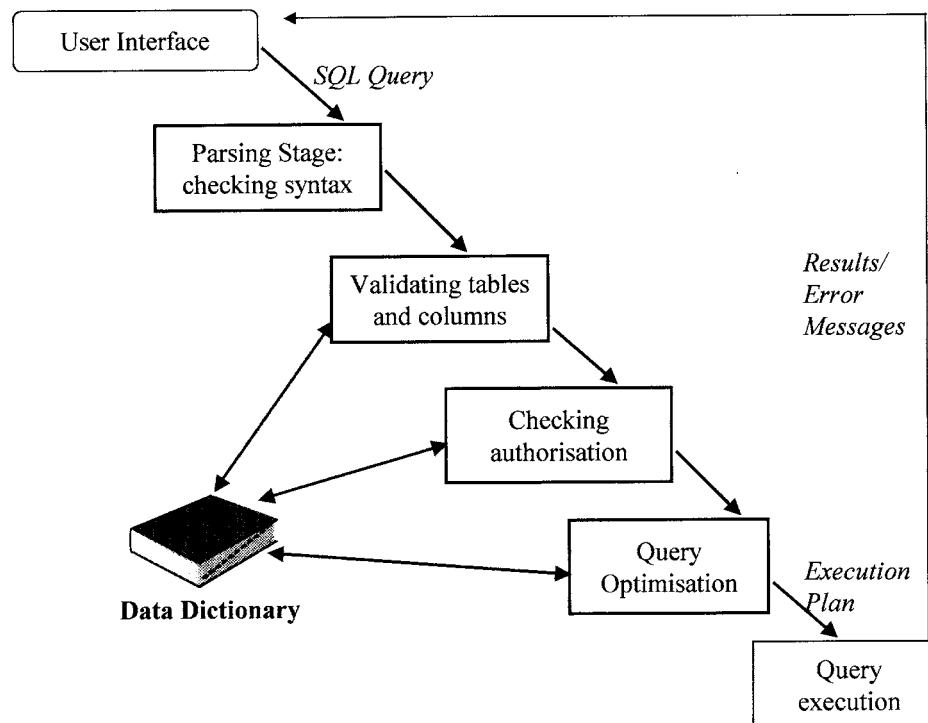


Figure 30.2 Processing a query.

30.3.1 THE NEED FOR OPTIMISATION

The problem of query optimisation can be illustrated using a simple example (Järke and Koch, 1984). Suppose we have two tables from the academic database, a students table and a moduleEnrolment table. The moduleEnrolment table stores details of which students have enrolled for which modules. Given these tables, the query '*What students are currently enrolled on the relational database systems module (a module code RDS)?*' might be expressed in SQL as:

```

SELECT studentName
FROM Students S, ModuleEnrolment M
WHERE S.studentNo = M.studentNo
AND M.moduleCode = 'RDS'
    
```

There are two basic ways in which this query can be implemented:

1. a. Form a combined record of students and enrolment information.
b. Extract from the resulting table those records having moduleCode 'RDS'.



2. a. Extract from the enrolment table those records having moduleCode 'RDS'.
- b. Match with the students table for the student names.

Let us assume, for the sake of simplicity, that the database contains 100 student records and 1,000 enrolment records. Further assume that of the 1,000 enrolment records only 50 are for moduleCode 'RDS'. Using some performance measure, such as the number of record I/Os to achieve each query implementation, it is clear that the second method is much more efficient than the first strategy:

1. a. Read 1,100 records and write $100 \times 1,000$ records back to disk.
b. Read 100,000 records and produce a table in main memory of some 50 records from which the student names can be produced.
Total = $1,100 + 100,000 + 100,000 = 201,100$ I/Os.
2. a. Read 1,100 records and form a table in main memory of some 50 records.
b. Read 100 records to produce a join of enrolment data with students data.
c. Produce the student names from the records in main memory.
Total = $1,100 + 100 = 1,200$ I/Os

30.3.2 THE PROCESS OF OPTIMISATION

The data-retrieval facilities of SQL are close to what we might expect from a non-procedural query language. By this we mean that the user can express a retrieval request in SQL largely in terms of what he or she requires from the database. The user does not need to get heavily involved in traversing or navigating through the database.

However, since computers are procedural machines, any statement expressed in non-procedural terms must eventually be translated into some procedural form. But, for any one non-procedural query there are usually numerous different ways of achieving the same results procedurally. Hence, some mechanism must choose between the various procedural alternatives in terms of some notion of the best transformation for the task. Such a translation mechanism is known as a query optimiser.

In general terms there are two classes of optimisers used in current RDBMS: syntax-based (sometimes called heuristics-based) optimisers and statistically-based (cost-based) optimisers.

Syntax-based optimisers choose an execution plan on the basis of the syntax of the SQL statement. It will come to its decision on the basis of such things as the form and order of the conditions in the *where* clause. This means that the same logical query expressed in two slightly different ways is likely to experience two distinct responses in terms of retrieval performance.

In contrast, statistically-based optimisers force queries to undergo a prior examination stage. The result of this stage is a conversion of the query into a base or canonical form. This means that two syntactically different but semantically equivalent queries issued to this form of optimiser will be converted to the same base form. The optimiser then chooses an execution plan based on



definitions about tables, columns and indexes in the data dictionary together with statistics about such things as the sizes of files.

One comment we could make about this distinction is that for persons knowledgeable about their optimiser it is possible to get very good performance from a syntax-based optimiser. However, for persons unaware of the idiosyncrasies of their optimiser it is also possible to get very bad performance from a syntax-based optimiser.

Using a statistically-based optimiser the user cannot directly affect the performance of the optimiser. However, a statistically-based optimiser will use more machine resources than would a syntax-based optimiser. Also, a statistically-based optimiser is only as good as the statistics maintained by the DBMS. Because of the overhead involved in generating such statistics, most systems tend to force the DBA to periodically command the update of statistics held in the catalog.

30.3.3 COMPILED AND INTERPRETED QUERIES

In some DBMS, queries are compiled. In other words, execution plans are produced via a compilation phase and stored for reuse. Other DBMS take an interpreted approach. All queries are interpreted at run-time.

As a general rule, compiled queries execute faster than do interpreted queries. This is because in the case of a repeated query the compilation need not be done again. However, if the database is subject to many queries of an *ad-hoc* nature, then interpreted queries are probably more flexible since they are not subject to a prior compilation phase.

30.4 BACKUP AND RECOVERY

In the case of a disaster, such as disk corruption, the database administrator (DBA) must recover from previously backed-up data. The DBA will usually have established a strategy of taking copies of either the whole or parts of the database periodically on some inexpensive storage medium such as magnetic tape. The DBA will also back-up the transaction log. To conduct such copying, the DBA will use one of the backup utilities of the DBMS.

In the case of a catastrophic failure, the latest backup copy can be reloaded from tape to disk and the system restarted.

Some backup utilities demand that the normal DBMS processing halt while backups are made. Because creating a full backup of a database usually takes a long time, the DBA will probably run this as an overnight process.

Other DBMS supply incremental backup utilities. These allow the DBA to take backups while normal processing continues.

30.5 SUMMARY

- The data dictionary is at the heart of a DBMS. Being a database, it has a structure and stores the meta-data relevant to some database



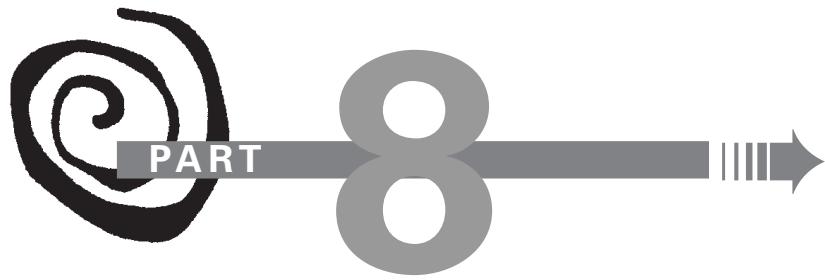
- Query management involves the syntactic checking of a query, query optimisation and query execution
- Query optimisation is the process of determining the optimally most efficient execution plan of a query
- DBMS either compile queries or interpret them at run-time
- Backup and recovery facilities are important tools for database administration

30.6A circular icon containing a stylized spiral or swirl symbol.
ACTIVITIES

1. Determine the structure of the system catalog of a DBMS known to you.
2. Examine the query manager of a DBMS known to you and determine whether it uses a syntax-based or statistically-based optimiser.
3. What sort of backup and recovery facilities are offered by a DBMS known to you?

30.7A circular icon containing a stylized spiral or swirl symbol.
REFERENCE

Jarke, M. and J. Koch (1984). Query optimisation in database systems. *ACM Computing Surveys* **16**: 111–52.

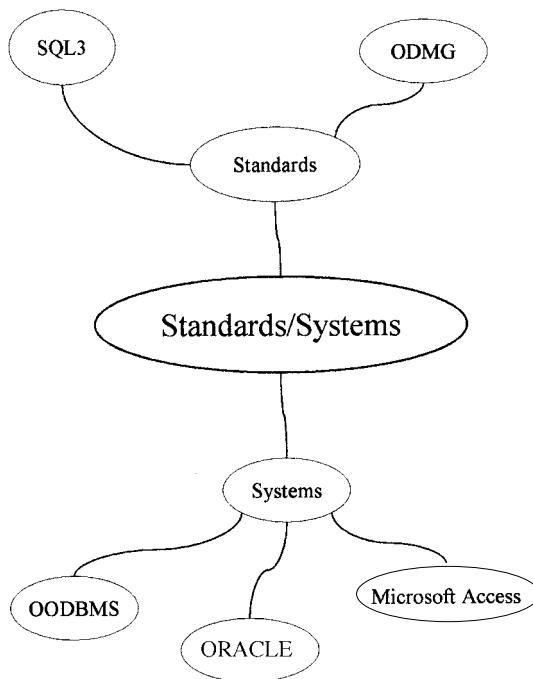


DATABASE MANAGEMENT SYSTEMS – STANDARDS AND COMMERCIAL SYSTEMS

It is common sense to take a method and try it. If it fails, admit it frankly and try another.

But above all, try something.

Franklin D. Roosevelt (1882–1945)



In this part we examine two important standards and a representative set of existing database management systems that are used in information systems practice.

SQL3 in the latest version of the SQL standard. It is a key attempt to standardise the syntax of post-relational features in many contemporary DBMS with a foundation in the relational data model. The ODMG standard is an attempt to standardise the syntax of OODBMS.

We also provide an overview of the structure and facilities of three contemporary DBMS. The three DBMS are chosen because of their popularity and because they are representative of the marketplace in terms of including features discussed in previous chapters. We particularly wish to include one representative desktop DBMS, one ‘industrial-strength’ post-relational DBMS and one DBMS from the object-oriented domain.

Microsoft Access is a DBMS originally created for the desktop market and designed to be used by non-sophisticated database users. ORACLE is a DBMS with a long history, being one of the first commercial relational DBMS. Its contemporary versions offer a number of post-relational or extended relational features defined in the SQL3 standard. O₂ is a system which began life as an academic research project. Nowadays it is a commercial OODBMS which adheres to the ODMG object model standard.

The table below provides a brief comparison of the three systems:

DBMS	Data Model	Primary Market	Genesis
Access	Relational	Desktop	Early 1990s
ORACLE	Post-relational	Enterprise	Late 1970s
O ₂	Object-oriented	Enterprise	Late 1980s

CHAPTER 31 →

POST-RELATIONAL DBMS – SQL3

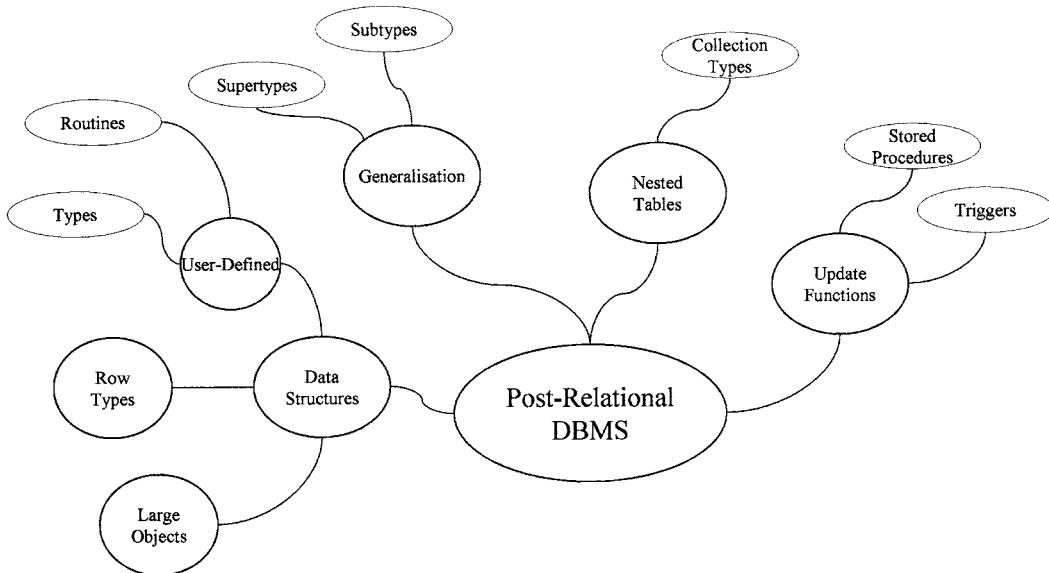
The nice thing about standards is that there are so many of them to choose from.

Andrew S. Tanenbaum

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Distinguish between relational and post-relational DBMS
- Describe some of the key features of SQL3



31.1 INTRODUCTION

Relational DBMS support the relational data model. In recent times a number of extensions have been introduced into contemporary DBMS to improve the active nature of database systems built using these tools. This class of DBMS, known variously as post-relational, object-relational, sometimes extended-relational systems, is based on a superset of the relational data model (Chapter 7). However, there is generally a lack of standards in this area. Each DBMS tends to offer subtly different facilities with different syntax. The development in the area of the SQL3 standard (ISO 1999a, b) is one attempt to encourage such standardisation. We concentrate on this initiative in this chapter.

31.2 SQL3 FEATURES

SQL3 was published as a full working standard in 1999 (Melton and Simon, 2002). Because of the delay experienced in producing the standard, some of the features discussed here may be deferred until SQL4. In this chapter we examine the following features proposed in the SQL3 standard:

- Row types
- User-defined types
- User-defined routines
- Reference types
- Collection types
- Support for large objects
- Stored procedures
- Triggers

Many of these features defined in SQL3 can be mapped onto the constructs of ADTs, complex objects (nested relations), stored procedures and triggers defined in Chapter 10.

31.3 ROW TYPES

A row type consists of a sequence of field names with their associated data types. Row types can be used to allow complete rows to be stored in variables, passed as arguments to procedures and returned as values from function calls. The construct of a row type enables a certain amount of complex structuring to be included within a table.

Example In the table below we define the postCode attribute as of type ROW. postCode is itself made up of two attributes: an area identifier and a location identifier. We further define a termAddress attribute to be made up of a sequence of column definitions including that of postCode:

```
CREATE TABLE students (
    studentNo INTEGER(6),
    studentName CHARACTER(20),
    termAddress ROW(houseNo INTEGER(2),
                    street VARCHAR(20),
                    town VARCHAR(20),
                    postCode ROW(areaID VARCHAR(4), locID VARCHAR(4)))
```

31.4 USER-DEFINED TYPES

SQL3 allows the definition of abstract data types (ADT) which are called user-defined data types or UDTs. A UDT definition consists of a number of attribute definitions and routine (function, procedure) declarations. A UDT is encapsulated in the sense that the definition of attributes and functions is only visible in terms of its interface, not in terms of its implementation. There is also a proposal that attributes and routines be classified in terms of the following tags familiar in C++ programs:

- Public components are visible to authorised users of the UDT
- Private components are visible only within the definition of the UDT
- Protected components are visible within their own UDT and within the specifications of all subtypes of the UDT

Example In the example below two attributes and two functions are defined for the studentType UDT. The first function is an example of a so-called *observer* function in that it returns a value from the UDT. The second function is an example of a so-called *mutator* function in that a value can be set for an attribute or attributes in the UDT.

```
CREATE TYPE studentType AS (
    PUBLIC
        studentNo INTEGER(6) NOT NULL,
        studentName VARCHAR(20) NOT NULL,
    FUNCTION studentName (S studentType) RETURNS VARCHAR
        RETURN S.fName
    END,
```



```

FUNCTION studentName (S studentType, newValue VARCHAR)
RETURNS studentType
BEGIN
    S.fName = newValue;
    RETURN S;
END)

```

31.5

USER-DEFINED ROUTINES

The functions defined in the UDT above are examples of user-defined routines. User-defined routines (UDR) define methods for manipulating data. UDRs may be defined as part of a UDT or separately as part of a database schema.

UDRs may be a procedure, function or iterative routine specified in some standard programming language such as C++ or they may be defined in terms of an extended version of SQL.

A procedure is invoked from an SQL CALL statement and has input and output parameters defined. A function returns a value and has one output parameter designated with usually a number of input parameters. An iterative routine is used to compute aggregate values.

31.6

SUBTYPES AND SUPERTYPES

SQL3 allows UDTs to be specified as subtypes of a given supertype using the UNDER clause.

Example

Hence, we might define an undergraduate UDT to be a subtype of the studentType class as below:

```
CREATE TYPE undergraduate UNDER studentType AS (
```

Multiple inheritance is supported in SQL3 in that a UDT can have more than one supertype. The names of UDRs may also be overloaded. This enables a UDT to redefine a method inherited from a supertype.

To maintain compatibility with SQL2 the CREATE TABLE statement still has to be used in SQL3, even if the table comprises one UDT.

Example ➔ Hence, to create a table using the studentType UDT we could do this in one of two different ways:

```
CREATE TABLE students (
    info STUDENTTYPE,
    PRIMARY KEY studentNo);

CREATE TABLE students OF studentType (
    PRIMARY KEY studentNo);
```

31.7 REFERENCE TYPES

SQL2 adheres to the relational model in specifying relationships between tables in terms of primary-key and foreign-key declarations. In SQL3 the construct of a reference type is introduced which can be used to define relationships between row types. Reference types are particularly useful in enabling the idea of object identity to be emulated in a post-relational form. A reference type has a type REF and is specified in the base table using the keywords VALUES ARE SYSTEM GENERATED.

Example ➔ In the example below we have created an object identifier in the person table. This table is referenced by the type studentType in terms of its nextOfKin attribute. This relationship is then inherited by the students table which is defined on the studentType.

```
CREATE TABLE person OF personType (
    objectID REF(personType) VALUES ARE SYSTEM GENERATED);

CREATE TYPE studentType UNDER personType AS (
    PUBLIC
        studentNo INTEGER(6) NOT NULL,
        studentName VARCHAR(20) NOT NULL,
        nextOfKin REF(personType)
    )

CREATE TABLE students OF studentType (
    PRIMARY KEY studentNo);
```

31.8 COLLECTION TYPES

SQL3 and proposals in SQL4 allow the definition of collection types. A collection type can be used to store multiple values in a column and can be used to nest tables within tables. The following collection types are available in SQL3:



- ARRAY – a one-dimensional array with a maximum number of elements
- LIST – an ordered collection that allows duplicates
- SET – an unordered collection that does not allow duplicates
- MULTISET – an unordered collection (sometimes referred to as a bag) that does allow duplicates

Example → We might modify the definition of a student type as below to include a nextOfKin attribute which allows a number of values to be stored, in this case, a SET of person identifiers.

```
CREATE TYPE studentType UNDER personType AS (
PUBLIC
    studentNo INTEGER(6) NOT NULL,
    studentName VARCHAR(20) NOT NULL,
    nextOfKin SET(personType)
)
```

31.9 STORED PROCEDURES

A number of new statements have been added to SQL3 to make the language computationally complete. These statements form the SQL Persistent Stored Module (SQL/PSM) language and can be collected together in a block with its own variables. Statements include:

- An assignment statement which allows the result of an SQL value to be assigned to a local variable, an attribute of a UDT or a column
- Conditional statements such as IF . . . THEN . . . ELSE and CASE
- Looping statements such as FOR, WHILE and REPEAT UNTIL
- A statement that permits a CALL to another procedure and a RETURN statement that allows an SQL value expression to be used as a return value

31.10 TRIGGERS

SQL3 introduces the concept of a trigger and defines a syntax for triggers, as below:

```
CREATE TRIGGER <trigger name>
BEFORE : AFTER <trigger event> ON <table name>
[REFERENCING <old or new values alias list>]
[FOR EACH {ROW : STATEMENT}]
[WHEN <trigger condition>]
<trigger body>
```



The elements of this syntax are explained below:

- Each trigger can be given a unique name – <trigger name>
- Each trigger is declared on a given database table – <table name>
- Triggering events include insertion, deletion and update of a table
- A trigger has an associated timing: it can be fired BEFORE the event occurs or AFTER the event occurs
- The trigger body can consist of a number of SQL statements but not transaction statements, such as COMMIT or ROLLBACK, or connection statements, such as CONNECT or DISCONNECT
- The trigger body can be executed in one of two ways: a row-level trigger (FOR EACH ROW) is executed for each row affected by the event; a statement level trigger (FOR EACH STATEMENT) is the default and is executed only once for each event
- The old or new values alias list allows the body to refer to an old or new row in the case of a row-level trigger or an old or new table in the case of an AFTER trigger

31.11 SUPPORT FOR LARGE OBJECTS

Contemporary databases need to store high-volume data in the columns of a table. For instance, a database may contain graphics or textual data. SQL3 defines three special data types for this type of data:

- Binary large object (BLOB) – a binary string that does not have a character set
- Character large object (CLOB) and national character large object (NCLOB) – both of which are character strings

Traditionally BLOBS are stored in databases as byte streams and the database has no information as to the internal structure of the BLOB. This means that the DBMS is unable to query or perform operations on BLOBS. In SQL3 large objects can be operated upon using standard string operations such as concatenation operators, substring operators, length and position functions.

Example  Below we include an example of how a column might be defined with a BLOB datatype:

```
ALTER TABLE staff  
ADD COLUMN studentPicture BLOB(10M)
```

Here we have added a column storing the image associated with a particular student having a declared size of ten megabytes.

**31.12****CASE STUDY: INSURANCE POLICIES**

We can use certain features of SQL3 to enrich the semantics of the schema specification for the insurance policies database introduced in Chapter 11. For instance, we may introduce a row type for the holder's address and a number of subtypes for the policy type.

```

CREATE TABLE PolicyHolders
  (holderNo CHARACTER(4),
   holderName CHARACTER(20),
   dateOfBirth DATE,
   holderAddress ROW(houseNo INTEGER(2),
                     street VARCHAR(20),
                     town VARCHAR(20),
                     postCode ROW(areaID VARCHAR(4), locID VARCHAR(4))),
   holderTelNo CHARACTER(12),
   PRIMARY KEY (holderNo))
CREATE TYPE PolicyType AS (
  PUBLIC
  policyNo CHARACTER(4),
  holderNo CHARACTER(4))
CREATE Table Policies OF policyType(
  currentPremium DECIMAL(8,2),
  PRIMARY KEY (policyNo)
  FOREIGN KEY (holderNo REFERENCES PolicyHolders))
CREATE TABLE LifePolicies UNDER Policies
  (startDate DATE)
CREATE TABLE CarPolicies UNDER Policies
  (regNo CHARACTER(10),
   engineCapacity INTEGER,
   renewalDate DATE)

```

The tables lifePolicies and carPolicies inherit the attributes of the Policies table which is declared to be of type PolicyType.

31.13**SUMMARY**

- Many contemporary DBMS are based on a post-relational, object-relational or extended-relational data model. This is a superset of the relational data model
- The current full standard for SQL, SQL3, includes a specification for a number of features found in post-relational DBMS: row types, user-defined types, user-defined routines, reference types, collection types, large objects, stored modules and triggers



31.14 ACTIVITIES

1. Detail how you might use subtypes and supertypes from SQL3 to define aspects of the stock market schema discussed in the Case Study of Chapter 17.
2. Use an SQL3 row type to define a delivery advice, as discussed in the Case Study of Chapter 3.
3. Use SQL3 syntax to define the stored procedures in the Case Study of Chapter 10.



31.15 REFERENCES

- ISO (1999a). *Database Language SQL – Part 2: Foundation*. ISO/IEC 9075–2. Geneva, International Standards Organisation.
- ISO (1999b). *Database Language SQL – Part 2: Persistent Stored Modules*. ISO/IEC 9075–4. Geneva, International Standards Organisation.
- Melton, J. and A.R. Simon (2002). *SQL: 1999: Understanding Relational Language Components*. San Francisco, CA, Morgan Kaufmann.



CHAPTER 32 →

OBJECT-ORIENTED DBMS – ODMG OBJECT MODEL

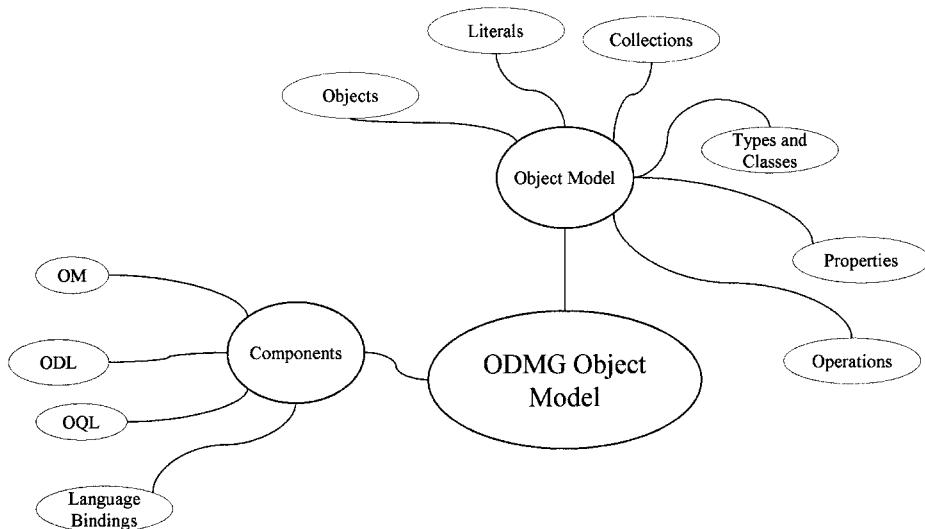
I paint objects as I think them, not as I see them.

Pablo Picasso (1881–1973)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Discuss the history of the ODMG object model
- Describe the key elements of the ODMG object model
- Illustrate schema definition using the object definition language



32.1 INTRODUCTION

A number of software vendors have got together to form the Object Database Management Group (ODMG) and to specify a standard for object-oriented DBMS (OODBMS). To do this, the group has specified a standard syntax and semantics for OODBMS. The intention is that this syntax and semantics should be portable across implementations of OODBMS. The first version of the ODMG standard was released in 1993 (Cattell, 1994), the second version was released in 1997 (Eaglestone and Ridley, 1998). The third and current version was launched in 2000 (Cattell, 2000).

32.2 THE OBJECT-ORIENTED DATABASE MANAGEMENT SYSTEMS MANIFESTO

The ODMG standard can be seen to be a reaction to an attempt to develop a specification of mandatory features for OO databases published in 1990 (Atkinson *et al.*, 1990). The OODBMS manifesto proposes thirteen mandatory features for an OODBMS. The first eight rules detail the importance of an OODBMS conforming to OO principles. The last five rules specify that an OODBMS must support a number of classic DBMS features:

- *Complex objects.* Complex objects must be supported. It must be possible to build complex objects by applying constructors to basic objects, such as set, tuple and list
- *Object identity.* Object identity must be supported. All objects must have a unique identity that is independent of the values of its attributes
- *Encapsulation.* Encapsulation must be supported. Strictly speaking, encapsulation requires that programmers only have access to objects through their defined interfaces. In many practical DBMS the enforcement of encapsulation is not as important as the need for *ad-hoc* query
- *Object classes.* The construct of object classes must be supported. The schema in an OODBMS must comprise a set of classes
- *Inheritance.* Inheritance of methods and attributes must be supported
- *Dynamic binding.* Dynamic binding must be supported. To support overriding, the DBMS must bind method names to logic at run-time – this is known as dynamic binding
- *Complete DML.* The DML must be computationally complete. The DML of the OODBMS should be a general-purpose programming language
- *Extensible set of data types.* The set of data types must be extensible. The user must be able to build new data types from predefined types
- *Data persistence.* Data persistence must be provided. Data must persist after the application is terminated and the user should not need to explicitly initiate persistence



- *Very large databases.* The DBMS must be capable of managing very large databases. An OODBMS should have a number of mechanisms enabling effective management of secondary storage such as indexes
- *Concurrent access.* The DBMS must provide concurrent access to data
- *Recovery facilities.* The DBMS must be able to recover from hardware and software failure
- *Query management.* The DBMS must provide a simple way of querying data

32.3

COMPONENTS OF THE STANDARD

The major components of the ODMG standard are as follows:

- *The object model (OM).* This defines a common data model to be supported by ODMG-compliant DBMS. It is based upon a data model originally produced by the Object Management Group (OMG)
- *Object specification languages.* Two such languages are defined – the object definition language (ODL) and the object interchange format (OIF). ODL is a specification language used to define object types that conform to the object model. OIF is a specification language used to dump the state of an object database to a set of files or load from a set of files into an object database
- *The object query language (OQL).* This is a declarative query language for querying and maintaining object databases. It is based upon the structured query language SQL (Part 3)
- *A number of language bindings (C++, Smalltalk, Java).* In effect these permit a programmer to read from and write to the same database from different programming languages

We concentrate mainly on the first two components in this chapter.

32.4

THE OBJECT MODEL

The object model (OM) defines the following constructs for modelling:

- *Objects and literals.* The basic modelling constructs or primitives are objects and literals
- *Types.* Objects and literals are classified into types. All elements of a given type have the same set of properties and common behaviour
- *Properties.* The state of an object is defined by the values associated with its set of properties. The properties of an object may be either an attribute of an object or a relationship between objects

- *Operations.* The behaviour of an object is defined by a set of operations (methods) associated with an object. Operations may have a list of input and output parameters, each with a specified type
- *Schemas.* A database is defined by a schema specified in the object definition language (ODL). An object database contains instances of the types defined in a database's schema

32.4.1 OBJECTS

Objects are defined against a type hierarchy consisting of the major abstract types: atomic objects, collection objects and structured objects. Under collection types and structured types a number of instantiable types are defined. They are instantiable in the sense that they are the only types that can be used as base types in the database. Each object has an object identifier and may have a number of names defined for it by users.

32.4.2 LITERALS

A set of literal types is defined including atomic, structured, collections or null. The value of a literal's properties may not change and hence literals may be used as traditional data types:

- *Atomic.* These are literals that are not created by applications and typically consist of numbers and character data types such as long, short, char and string
- *Structured.* These are literals that have a fixed number of elements each of which has a variable name and can contain either a literal value or an object. Structured literals include such data types as date, interval and time

32.4.3 COLLECTIONS

Collections may be collection objects or collection literals. The only difference between the two is that collection objects have identity. For instance, we may use a collection object to define the set of students in a school. We may also use a collection literal to define addresses as data structures.

The object model defines five types of collection objects:

- *Set* – unordered collections that do not allow duplicates
- *Bag* – unordered collections that allow duplicates
- *List* – ordered collections that allow duplicates
- *Array* – one-dimensional arrays of varying length
- *Dictionary* – unordered sequence of keys and values with no duplicate keys



32.4.4 TYPES AND CLASSES

The object model distinguishes between types and classes.

A type has one specification and one implementation. There are two aspects to a type: a specification and one or more implementations. The specification details the external characteristics of the type: the operations that can be invoked on its instances, the properties that can be accessed and any exceptions that can be raised by specific operations. A type's implementation constitutes the internal details of the type and is determined by a language binding.

The combination of the type specification and one implementation is said to be a class. A class definition is a specification that defines the abstract behaviour and abstract state of an object type.

The abstract behaviour of an object type is specified in its interface definition. The interface definition specifies its supertypes, its extent and its keys. The set of all instances of a given type is its extent. A key uniquely identifies the instances of an object type.

Example The class Module defines both the abstract behaviour and abstract state of the Module objects.

```
CLASS Module { . . . };
```

32.4.5 SUBTYPES AND SUPERTYPES

The ODMG object model includes the notion of supertyping and subtyping and the associated process of inheritance. This form of relationship implements a generalisation–specialisation relationship (Chapter 17).

Example In the example below SeniorLecturer is a subtype of Professor and Professor is a subtype of Employee:

```
INTERFACE Employee { . . . };
INTERFACE Lecturer: Employee { . . . };
INTERFACE SeniorLecturer : Lecturer { . . . };
```

This means that an instance of SeniorLecturer conforms to all the behaviours specified in the SeniorLecturer interface. It also inherits the operations of both the Lecturer and Employee interfaces.

The colon in the definitions above defines the inheritance of behaviour between object types. The object model also defines an EXTENDS relationship for the inheritance of state and behaviour.



Example → In this example a class Person might have attributes such as name and date of birth which are inherited by the class EmployeePerson. Since EmployeePerson is also declared to be a subtype of Employee, it would inherit all the operations of this interface.

```
CLASS Person { . . . };
CLASS EmployeePerson EXTENDS Person : Employee { . . . };
```

32.4.6 PROPERTIES

The OM defines two types of property: attributes and relationships. A user accesses and manipulates the state of the instances of some class through its properties.

An attribute is defined on a single object type and takes as its value a literal or an object identifier.

Example → We may define the attributes of the class *module* as follows:

```
ATTRIBUTE STRING moduleCode;
ATTRIBUTE STRING moduleName;
ATTRIBUTE SHORT level;
ATTRIBUTE SHORT roll;
```

All relationships in the object model are binary. In other words, relationships are defined between two types. Each such relationship has a specific cardinality: one-to-one, one-to-many or many-to-many. In the interface definition for a class, traversal paths are defined for each relationship. Traversal paths are declared in pairs, one for each direction of traversal.

Example → Lecturer *teaches* Module and Module *is taught by* Lecturer are the two traversal paths for the relationship *teaches*. The teaches traversal would be defined in the declaration for the Lecturer type. The taught by traversal would be defined in the declaration for the Module type.

To indicate that two traversal paths apply to the same relationship, the keyword INVERSE is used. On the many side of relationships the objects are unordered (set or bag) or ordered (list).



Example ➔ A one-to-many relationship between a Lecturer type and a Module type would be defined in the following way:

```
CLASS Lecturer {
    ...
    RELATIONSHIP SET<Module> Teaches INVERSE Module::TaughtBy;
    ...
};

CLASS Module {
    ...
    RELATIONSHIP Lecturer TaughtBy INVERSE Lecturer::Teaches;
    ...
};
```

The two traversal paths specified above indicate that a given lecturer teaches a set of possible modules while a given module is taught by at most one lecturer.

Referential integrity of relationships is automatically maintained by the DBMS. Hence, if an object that participates in a relationship is deleted, then any traversal path to that object is also deleted.

32.4.7 OPERATIONS

The instances of an object type have behaviour specified in a set of operations. The object type definition defines a signature for the operation which includes the operation name, parameters and type of each parameter, and the types of values returned.

Example ➔ For instance, we might define a signature for an operation relevant to the class *Module* as below:

```
VOID increaseRoll(IN SHORT amount);
```

This operation increases the roll of a Module instance by a set amount.

The parameters of an operation are listed in rounded brackets. Each parameter has a name and a type. A parameter can also be designated as an input parameter (IN), output parameter (OUT) or both (INOUT).

An operation can return an object as its value. If an operation does not return a value, the operation is prefixed with the keyword VOID.

The object model specifies built-in operations for adding (FORM) and deleting (DROP) instances of relationships and managing the integrity constraints associated with relationships.



Example ➔ Hence, given the relationships specified for the class Module as above, two further operations will be defined for this class:

```
ATTRIBUTE moduleTaughtBy;
VOID drop_module(IN Module aModule);
VOID form_module(IN Module aModule);
```

32.5 OBJECT DEFINITION LANGUAGE AND OBJECT QUERY LANGUAGE

ODL corresponds to the DDL of a conventional DBMS. It constitutes a language for defining object types in a database. It allows developers to specify the attributes, relationships and operation signatures but does not permit the specification of operations. We have implicitly used parts of ODL in illustrating some of the major constructs of the object model above.

The object query language (OQL) can be used to provide access to an OO database. It can also be used directly to maintain data in object databases, although this would normally be achieved through the operations specified against objects in the schema. It uses a syntax closely modelled on SQL2 and can be used as a standalone query language or as an embedded query language in C++, Java or Smalltalk. Examples of the use of OQL are provided in Chapter 35.

32.6 CASE STUDY: SPECIFYING THE ACADEMIC DATABASE IN ODL

Below we specify part of the schema relevant to the academic database in ODL:

```
CLASS Module
(EXTENT Modules; KEY moduleCode)
{
/* Define Attributes */
    ATTRIBUTE STRING moduleCode;
    ATTRIBUTE STRING moduleName;
    ATTRIBUTE SHORT level;
    ATTRIBUTE SHORT roll;
    ATTRIBUTE moduleTaughtBy;
    ATTRIBUTE modulePresentedOn;
/* Define Relationships */
    RELATIONSHIP SET <Lecturer> TaughtBy INVERSE Lecturer::Teaches;
    RELATIONSHIP Course PresentedOn INVERSE Course::Presents;
/* Define Operations */
    VOID increaseRoll(IN SHORT amount);
}
CLASS Student
```



```

(EXTENT Students; KEY sno)
{
/* Define Attributes */
ENUM {male, female} sex;
STRUCT studentAddress {SHORT houseNumber, STRING street, STRING area, STRING
city, STRING postCode};
STRUCT studentName {STRING fName, STRING lName};
ATTRIBUTE studentAddress termAddress;
ATTRIBUTE studentAddress homeAddress;
ATTRIBUTE studentName sName;
ATTRIBUTE sex studentSex;
ATTRIBUTE SHORT roll;
ATTRIBUTE DATE DOB;
/* Define Relationships */
RELATIONSHIP SET <Module> enrolledOnModule INVERSE Module::HasEnrolled;
RELATIONSHIP SET<Module> assessedOnModule INVERSE Module::HasAssessed;
/* Define Operations */
SHORT studentAge(IN DATE DOB);
}

CLASS UndergraduateStudent:Student
(EXTENT UndergraduateStudents; KEY sno)
{
/* Define Attributes */
ATTRIBUTE SHORT sNo;
/* Define Relationships */
RELATIONSHIP SET <Course> registeredOnCourse INVERSE Course::HasRegistered
}

CLASS PostStudent:Student
(EXTENT PostgraduateStudents; KEY sno)
{
/* Define Attributes */
ATTRIBUTE SHORT sNo;
/* Define Relationships */
RELATIONSHIP SET<Lecturer> supervisedBy INVERSE Lecturer::Supervises
}

```

In this schema we have four class definitions for object types: Module, Student, UndergraduateStudent and PostStudent. UndergraduateStudent and PostStudent are subtypes of the object type Student. For each class an extent (collection of instances) is defined. For instance, the extent of the class Module is modules. Each class also has a key defined. For example, each instance of the class module is distinguished from other instances by the value moduleCode. The state of each class is defined in terms of a number of attribute definitions. Each class has defined relationships with other classes. The behaviour of each class is defined in terms of operations.

32.7 SUMMARY

- The ODMG standard is an attempt to specify a standard syntax and semantics of OODBMS. It can be seen as a reaction to the OO database manifesto
- The ODMG has the following components: an object model, an object definition language, an object query language and a set of language bindings
- The object model defines the set of modelling constructs
- The ODL constitutes a DDL for object databases
- The OQL constitutes a DML for object databases

32.8 ACTIVITIES

1. Specify the class Lecturer in ODMG syntax.
2. Determine the degree to which a specific OODBMS known to you satisfies the principles of the OO database manifesto.
3. Determine the degree to which a given OODBMS known to you is ODMG compliant.

32.9 REFERENCES

- Atkinson, M., D. DeWitt, D. Maier, F. Bancilhon, K. Dittrich and S. Zdonik (1990). The object-oriented database system manifesto. In W. Kim, J.-M. Nicolas and S. Nishio (Eds), *Deductive and Object-Oriented Databases*. North-Holland, Elsevier Science.
- Cattell, R.G.G. (1994). *The Object Database Standard: ODMG-93: Release 1.1*. San Mateo, CA, Morgan Kaufmann.
- Cattell, R.G.G. (Ed.) (2000). *The Object Database Standard: ODMG Release 3.0*. San Mateo, CA, Morgan Kaufmann.
- Eaglestone, B. and M. Ridley (1998). *Object Databases: An Introduction*. Maidenhead, Berkshire, UK, McGraw-Hill.

CHAPTER 33 →

MICROSOFT ACCESS

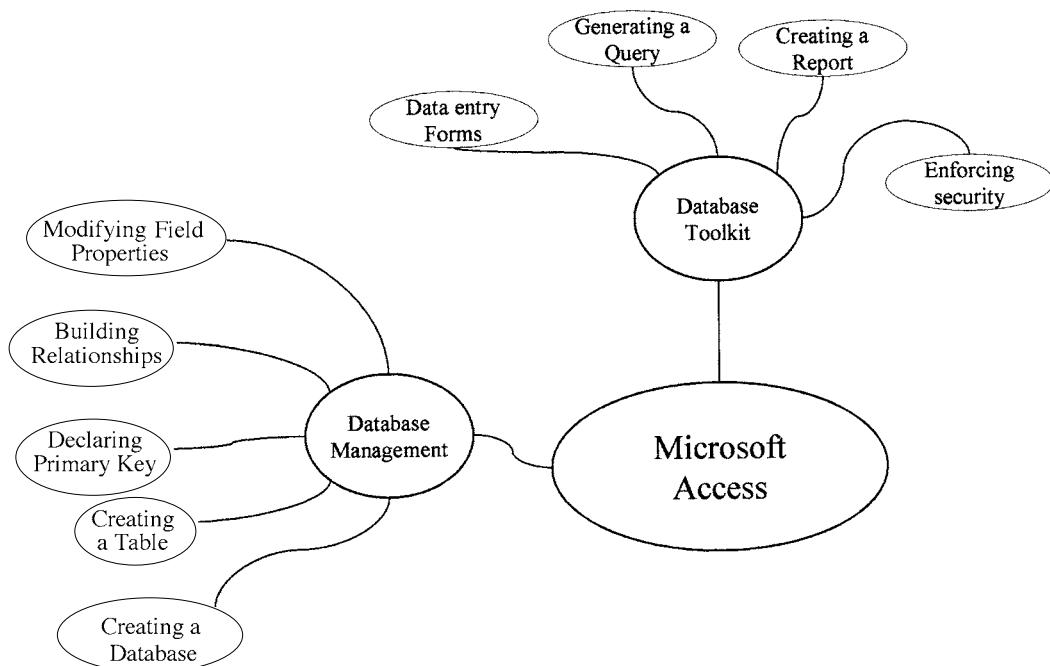
Computer Science is no more about computers than astronomy is about telescopes.

E.W. Dijkstra

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Describe the features of the Microsoft Access DBMS
- Provide a brief tutorial introduction to using Microsoft Access



33.1 INTRODUCTION

Microsoft Access was designed originally as a desktop DBMS by Microsoft to compete with other DBMS in this category at the time. The first version of this product was brought out by Microsoft in 1990. Since that time the Access DBMS has come to dominate the desktop DBMS market and forms an important element of Microsoft's Office suite of products. The current version of Access is Access 2002 (O'Leary, 2002).

Microsoft Access is a DBMS designed for use on personal computers. Access provides facilities for creating tables, queries, forms and reports. Database applications can be created using the Microsoft Access Macro Language or the Microsoft Visual Basic for Applications language. Access can be run as a stand-alone system on a PC or as a multi-user system across a network of PCs. However, when used as a multi-user environment, Access suffers from some key limitations. For instance, it lacks a fully-formed method of transaction management as available in such systems as ORACLE, has key limitations in terms of the volume of information it can manage satisfactorily and suffers from the lack of certain constructs essential for good database administration, such as a system catalog. For these reasons, Access tends to be used mainly for low-volume, small-to-medium complexity systems with a defined set of no more than twenty users. When applications grow beyond these levels then many organisations prefer to use Microsoft's enterprise level DBMS SQL Server, often in association with Access front-ends.

The facilities of Access can be divided into two groups: those devoted to data management (the database kernel) and those devoted to application development (the database toolkit). Access has proven popular as a rapid application development tool. Its ability to construct both database schemas and application components such as data entry screens quickly makes it an effective tool for prototyping information systems. In many projects Access is used to build early versions of systems as a means of encouraging organisational learning. Once requirements have stabilised, systems are then frequently up-sized onto Microsoft's enterprise level DBMS, SQL Server, or Access is used as a front-end to other 'industrial-strength' DBMS such as ORACLE.

Microsoft Access also provides a number of so-called wizards – facilities which enable the user to quickly create standard database system objects such as tables, forms and queries.

In this chapter we review some of the main features of Access in the process of constructing a simple database application designed to store data about research publications.

33.2 DEFINING SCHEMAS IN ACCESS

A database schema can be defined in Microsoft Access using a graphical user interface. This interface generates commands in SQL which are executed by the Access DBMS kernel.



33.2.1 CREATING A DATABASE

When the Microsoft Access DBMS is started the Microsoft Access dialog box is displayed. This dialog asks whether the user wishes to create a blank database, use the database wizard or open an existing database (Figure 33.1).

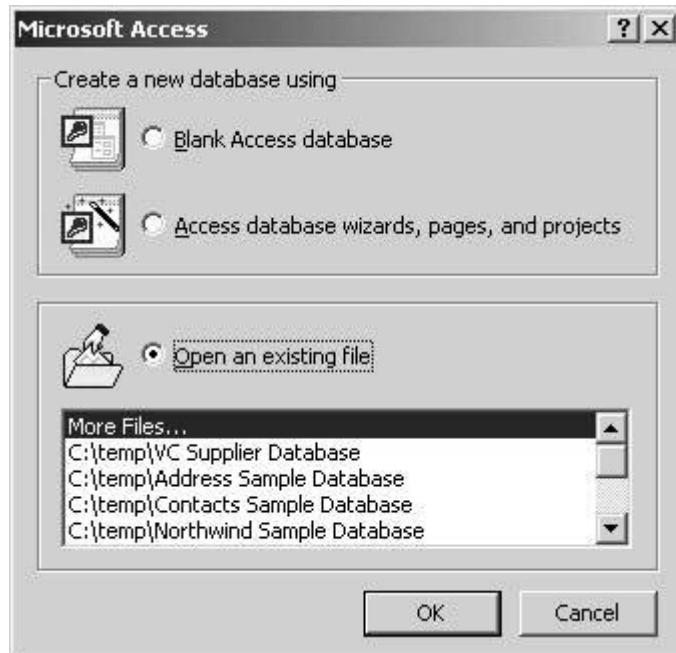


Figure 33.1 Microsoft access dialog box.

The database wizard is a software module which leads the user through the steps of creating simple table structures. In this section we concentrate on creating tables without using the database wizard.

Having provided a name for the database, the database window should appear.

33.2.2 CREATING A TABLE

There are a number of ways of building tables in Microsoft Access. Here, we concentrate on constructing a table using a design view.

The database window allows you to choose to create a new table, form, query etc. Suppose the user selects *table* and clicks on *New*. The next dialog asks the user what view he or she wants to work within. Suppose we select *design view*.

The user now has to enter the names of the columns for a table under field name and a data type for each column (text, number etc.) (Figure 33.2). The description is optional.

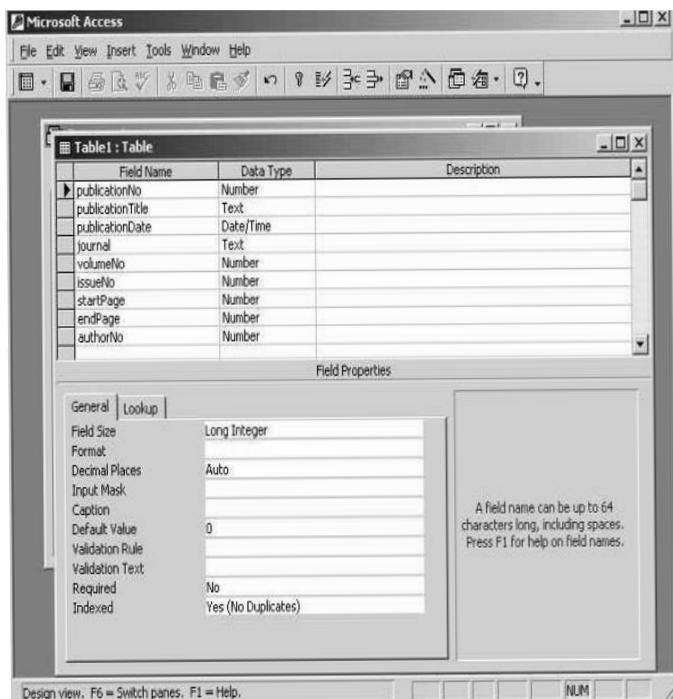


Figure 33.2 Design view for table.

Example For instance, we might enter the following column names and data types for a table storing data pertaining to research publications:

- publicationNo, Number
- publicationTitle, Text
- publicationDate, Date/Time
- Journal, Text
- VolumeNo, Number
- IssueNo, Number
- StartPage, Number
- EndPage, Number
- AuthorNo, Number

33.2.3 DECLARING A PRIMARY KEY

Once all the columns have been entered into the design view a primary key for the table should be selected, for example, *publicationNo*. This column or group



of columns is specified as the primary key by clicking on the key icon in the menu bar. A key symbol then appears alongside this row in the view, indicating that publicationNo has been specified as the primary key for this table.

33.2.4 SAVING TABLES

To save the structure of a table the user must click on the disk icon in the menu bar. When prompted for the table name, we might type in *publications*. Then we close the design view and we should see *publications* entered as a table name.

33.2.5 BUILDING RELATIONSHIPS

To illustrate how relationships are built we need to create another table using the steps above, this time with the following structure:

- AuthorNo, Number
- AuthorName, Text
- AuthorTitle, Text

We give it the name *Authors*.

To relate two tables together using primary and foreign keys, the user needs to click *relationships* on the toolbar. The user should then be prompted with a *show table* dialog in which both authors and publications are listed.

We select authors and click add. This table structure is then added to the relationships window. Next we do the same with publications and choose close. Both tables should now be present in the window.

To create a relationship between two tables we drag the attribute that we want related from one table across to the related field in the other table. For instance, we might click *authorNo* in publications and drag the field over to the authors table. A dialogue should appear asking the user whether he or she wishes to create the relationship. This dialog box also enables the user to specify referential integrity and a number of associated propagation constraints. For instance, if we click on the referential integrity box and cascades update and cascades delete options, the DBMS will enforce a number of rules against this relationship:

- Whenever an authorNo is entered in the Publications table, a check is made in the Authors table to see that the relevant authorNo exists
- If an authorNo is changed in the Authors table then this is reflected in the Publications table (cascades update)
- If an author is deleted from the Authors table then all relevant publications are deleted from the Publications table (cascades delete)

If we choose to create this relationship a line should then appear in the relationships layout window linking the primary key of the authors table with the foreign key in the publications table (Figure 33.3).

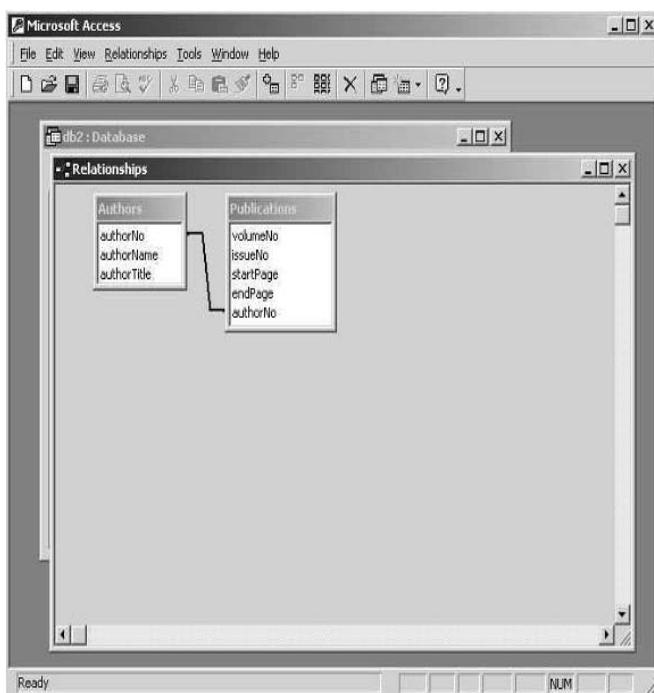


Figure 33.3 Relationships window showing relationship.

We then need to exit the relationships layout window and save the relationships.

33.2.6 MODIFYING FIELD PROPERTIES

Access provides facilities for adding constraints to a table through the field properties option of the table design view. Constraints are associated with field properties such as:

Field Property	Description	Example
Field Size	For text, number and auto-number (counter) data types, field size sets the maximum size for data entered against this field	publicationNo – 10
Format	This property permits the user to customise the way data is displayed and printed	publicationDate – short date – DD/MM/YY
Decimal Places	When using numbers, this property specifies the number of decimal places to be displayed	volumeNo – 0
Input Mask	An input mask contains the type of characters to be entered against each character of a field	isbnNo – 0-000-00000-0 0 – digit; L – letter



Caption	Permits a fuller description or title to be displayed for the field in reports	publicationTitle – Publication Title
Default Value	A value automatically entered when a new row is created	volumeNo – 1; issue No – 1
Validation Rule	Allows the user to specify domain constraints	volumeNo – >=1 OR <=100
Required	Allows the user to specify not nulls against fields	publicationNo – Yes; startPage – No
Indexed	Sets an index against a field	authorNo in publications – Yes

33.3**CREATING A DEFAULT DATA ENTRY/RETRIEVAL FORM IN MICROSOFT ACCESS**

As well as being able to specify database schemas visually, Microsoft Access allows users to quickly create data entry and retrieval screens for a given database. Hence, a degree of application development can be conducted through the medium of Access.

33.3.1 WHAT IS A FORM?

A form is a user interface construct used to enter, display or modify data stored in a database. Within Access, a form can be based on one or more tables. In this exercise we will build a form based on a single table and concentrate on the use of a form wizard to build a default form.

33.3.2 CREATING A DEFAULT FORM

Assuming the user has selected the previous database created above, the user now needs to choose *forms* off the database window and select *new*. This time we shall choose *form wizard* and select *Authors* as the table on which the default form will be built.

The user should then be requested which fields he or she wants on the form. Select all (double arrow) and click *next*. When you are asked which layout you would like, choose *columnar* and click *next*. Then you are requested to specify a style for the form; specify *standard* and click *next*. Finally you are asked to specify a title; type in *Authors*. Then click on *finish*. You should then be presented with a sample form.

On closing the form you will be asked for a name for the form; type in *Authors*.

33.3.3 USING A FORM

Enter some fictional data through the authors form. Then choose right-arrow at the bottom of the form. This will cause the data to be entered into the database. Try entering at least three fictional records into the authors table and navigating through them using the arrow buttons on the bottom of the form.

Now try creating a default form based on the Publications table following the same steps as above.

Next try entering fictional data into the Publications table.

Try entering an author number into this table which does not currently exist in the authors table. You should get an error message saying that referential integrity has been violated.

33.3.4 MODIFYING A FORM

Select *forms* off the database window and choose the Publications form. Then select *design view* for the form. The design view displays a number of controls. Access refers to any object in a form as a control. A control can be a field, a calculation, a piece of text, a graph or a picture. The current form should have a text box comprising the form header. In the form detail there are a number of control boxes composed of two parts: an attached label which describes a field in the database (these should currently be the column names in the Publications table) and a text box which is the field itself. Try changing the following elements on the form:

- Change the title of the form to *Authors Data Entry* by selecting the form header and modifying the text
- Change the labels of the fields by selecting a particular control box in the form detail and overtyping the label
- Resize author title by selecting the appropriate text box and dragging the box boundaries outwards. You may need to resize the design view to do this

Then close and save the form.

33.4 GENERATING A QUERY AND A REPORT IN MICROSOFT ACCESS

Queries can also be generated quickly in Access using a query wizard.

33.4.1 WHAT IS A QUERY?

A query is a question asked of a database. Microsoft Access uses SQL as its base language for representing queries. Users of Access can however build queries without needing to write any SQL directly. It is interesting that a form can be based on a query as well as on an underlying table. This is similar to the concept of a relational view.

Queries can also be quickly turned into reports for printing or displaying on the screen.

33.4.2 CREATING A SIMPLE QUERY USING THE QUERY WIZARD

Select *queries* off the database window. Then select the query wizard. In response to the request to specify fields, choose all the fields in the Publications



table. Click *next*. You are requested to specify a title. Type in *Simple Publications Query* and click *finish*. The results of the query should then be presented.

33.4.3 RUNNING A QUERY

From the database window select tables/queries and choose the simple publications query. Then view the results of the query in datasheet view.

33.4.4 MODIFYING A QUERY

Now choose design view for this query. You will see the current list of fields included in the query and on what basis they are sorted. Cut startPage and endPage from the query. Close the query and save it. Reopen it in datasheet view.

33.4.5 CREATING A REPORT USING THE REPORT WIZARD

Choose *reports* off the database window and select *new*. Choose *reports wizard* and select *authors* as the table on which the report will be built. Then click OK.

You should then be requested which fields you want on the report. Select all (double arrow). Click next. Then you will be asked to specify any grouping levels. Ignore by clicking on next. Then you are asked to specify the sort order. Select descending order on author name. Next you are asked which layout you would like. Choose tabular. Click next. Then the user is requested to specify a title style for the form. Specify corporate. Click next. Finally you are asked to specify a title. Type in *Authors report*. Then click on *finish*. You should then be presented with a sample report.

33.4.6 RUNNING A REPORT

Select *reports* off the database view. Then double-click on the Authors report. The result of the report should then be produced in a window.

33.4.7 MODIFYING A REPORT

Select the *authors* report. Choose *design view*. In this view you are able to edit the components on the report.

Modify the title of the report to Authors Report. Also modify the labels for each of the fields again. Save the report and run the report again.

33.5



SECURITY IN MICROSOFT ACCESS

Microsoft Access has two levels of security. The whole database can be secured by setting a password which is activated when opening the database. A more

sophisticated form of security can be enacted by enrolling users as members of groups. By default, Access provides two groups – users and administrators. However, further groups can be added. Permissions are set against users and/or groups, such as the ability to be able to read a particular table or amend particular database forms.

33.6 MULTI-USER SUPPORT

As we mentioned in the introduction, Access can be used in a multi-user environment. The main ways in which this can be achieved are listed below:

- *File-server*. An Access database can be placed on a server within an organisation's network so that each user running a copy of the application on their workstation can gain access to data
- *Client-server*. In a previous version this was achieved via linking Access tables through an ODBC driver to databases such as those stored under SQL Server. In Access 2000 an Access project file can be created which can store items like forms and reports on the client but link to a remote database through OLE DB (object linking and embedding for databases)
- *Replication*. Changes to an Access application can be replicated from a central application (known as the Design Master) to copies in numerous locations across a network. This solution enables the mass deployment of data via Access databases
- *Data access pages*. Access 2000 introduces a special type of Web page known as a data access page. This can be used for viewing or updating data over some intranet and is stored as an object in an Access database

33.7 SUMMARY

- Microsoft Access is a desktop DBMS
- The facilities of Access can be divided into two groups: those devoted to data management (the database kernel) and those devoted to application development (the database toolkit)
- Data management facilities include the creation of tables, relationships and queries
- Application development facilities include the production of forms and reports

33.8 ACTIVITIES

1. Run through the steps of producing the schema, forms and reports described in this chapter.



2. Consider modifying the schema to handle the following elements: research grants, research students and various types of publications such as journal papers, conference papers and books.
3. Modify the interface to produce the necessary data entry forms and reports for these new aspects of the schema.

33.9

REFERENCE

O'Leary, T. (2002). *Microsoft Access 2002*. New York, McGraw-Hill.

CHAPTER 34 → ORACLE

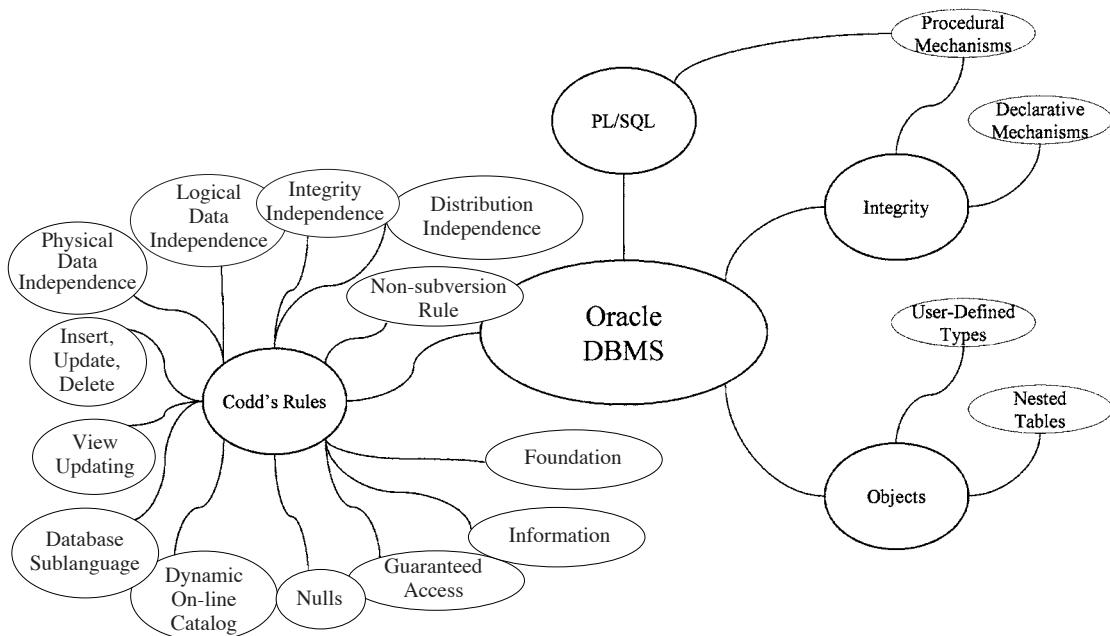
The folly of mistaking a paradox for a discovery, a metaphor for a proof, a torrent of verbiage for a spring of capital truths, and oneself for an oracle, is inborn in us.

Paul Valery (1871–1945)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Describe some of the history of the ORACLE DBMS
- Define Codd's rules for relational DBMS
- Describe the core features of the ORACLE DBMS
- Illustrate some of the key object-relational features of ORACLE



34.1 INTRODUCTION

The ORACLE relational DBMS was the first commercial version of relational technology when it was first released in 1977. From this beginning ORACLE has achieved a predominant position among DBMS vendors. It is possible to distinguish a number of distinct stages in the development of the ORACLE DBMS:

- *The pioneer phase (1977–84).* In this first phase ORACLE established itself as an ‘industrial strength’ relational DBMS. However, during the 1970s and early 1980s relational DBMS took second position in the commercial marketplace to hierarchical and network DBMS. This reflects the development of relational technology in general
- *The management information (MIS) phase (1984–88).* By the mid 1980s relational DBMS were beginning to make an impact, particularly because of their ability to support decision support systems (DSS) and management information systems (MIS)
- *The on-line transaction processing (OLTP) phase (1988–92).* In the late 1980s relational DBMS started to address a perceived failing, their ability to support high-volume transaction-based systems
- *The universal server phase (1992–97).* During the 1990s ORACLE had positioned itself as an object-relational or post-relational DBMS (Chapters 10 and 31), as a DBMS able to handle both standard and non-standard data
- *The Internet support phase (1997–2002).* ORACLE released version 8 of its DBMS in 1997. This version extended the post-relational features of the product and improved its performance and scalability. In 1999 the company released version 8i which was designed to support Internet applications
- *The data warehousing phase (2002–).* The current version of the DBMS is ORACLE 9i (Abbey *et al.*, 2002). A particular focus of this product is support for large data warehousing (Chapter 40) applications

Not surprisingly, ORACLE comprises a set of data management products rather than one DBMS. Many of such products are priced separately. The vendor also provides a range of databased application packages for supporting key business applications such as financials.

34.2 CODD'S RULES FOR RELATIONAL DBMS

During the mid-1980s there was growing concern about the large number of DBMS claiming to be relational. In response to this, Codd published a number of rules which attempted to define key features of what a relational DBMS should offer (Codd 1985a, b). The rules are listed below:

- *Rule 0 – Foundation rule.* Any system that is advertised as, or claimed to be, a relational DBMS must be able to manage databases entirely through its relational capabilities
- *Rule 1 – The information rule.* All information in a relational database is represented explicitly at the logical level and in exactly one way – by values in tables
- *Rule 2 – The guaranteed access rule.* 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
- *Rule 3 – Systematic treatment of nulls.* Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in a fully relational DBMS for representing missing or not applicable information in a systematic way, independent of data type
- *Rule 4 – Dynamic on-line catalog based on the relational model.* The database description is represented at the logical level in the same way as ordinary data, so that authorised users can apply the same relational language to its interrogation as they apply to regular data
- *Rule 5 – Comprehensive data sublanguage rule.* A relational system may support several languages and various modes of interaction. However, there must be at least one language whose statements are expressible, through some well-defined syntax, as character strings and that is comprehensive in supporting all the following: data definition, view definition, data manipulation (interactive and by program), integrity constraints, authorisation and transaction boundaries
- *Rule 6 – View updating rule.* All views that are theoretically updatable are also updatable by the system (Chapter 23)
- *Rule 7 – High-level insert, update and delete.* The capability of handling a base relation or a derived relation as a single operation applies not only to the retrieval of data but also to the insertion, update and deletion of data
- *Rule 8 – Physical data independence.* Application programs and interface activities remain logically unimpaired whenever any changes are made in either storage representations or access methods
- *Rule 9 – Logical data independence.* Application programs and interface activities remain logically unimpaired when information-preserving changes of any kind that theoretically permit unimpairment are made to the base tables
- *Rule 10 – Integrity independence.* Integrity constraints specific to a relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs



- *Rule 11 – Distribution independence.* A relational DBMS has distribution independence. This means that users should not be aware that a database has been distributed
- *Rule 12 – Non-subversion rule.* 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 language (multiple-records-at-a-time)

ORACLE as a DBMS scores quite highly on this scorecard – probably with a score of 10 out of 12. It uses tables as its primary data structure, utilises primary and foreign key constructs, uses null characters to represent incomplete or missing data, offers an on-line catalog, uses SQL as its defined interface and permits the storage of integrity constraints in the system catalog. However, like most contemporary DBMS, it does not enable full data or distribution independence.

In recent years however, ORACLE, like many other relational DBMS, has included additional features to those originally specified by Codd. This is particularly true of the ability to incorporate object-based features within a relational database system. It is for this reason that we refer to ORACLE as a post-relational DBMS.

34.3

THE ORACLE KERNEL, INTERFACE AND TOOLKIT

As we discussed earlier, modern-day DBMS generally come in three parts: kernel, interface and toolkit.

The kernel of the ORACLE DBMS comprises a relational database engine with standard features such as concurrency control, transaction management, dictionary and query management. Various options for running the kernel are included in the vendor's product-set, such as a parallel server (Chapter 38), a database server in a three-tier client-server architecture (Chapter 36) and a universal server for documents, audio and video files (Chapter 39). These kernel options can be accessed through a version of the SQL standard.

Various functions are also bundled as tools in the ORACLE DBMS. For instance:

- *Database administration.* ORACLE provides the DBA with functions necessary to create, maintain, start, stop and recover a database
- *Application development.* ORACLE provides the means of integrating SQL with standard programming languages such as C, FORTRAN and COBOL. It also provides development tools for constructing data entry forms, reports and graphs
- *Additional integrity.* Recent versions of ORACLE have included facilities for writing stored procedures and triggers, creating nested tables and defining user-defined types

We particularly focus on the latter features of ORACLE within this chapter.

34.4  PL/SQL

PL/SQL is ORACLE's version of a procedural fourth generation language with embedded SQL (Chapter 25). PL/SQL was first introduced with version 6 of ORACLE. It was developed with the aim of permitting the construction of complex database operations completely within the database server. This means that the server does not need to pass back control to a process after every SQL operation.

PL/SQL functions can be implemented in non-procedural development tools such as ORACLE forms. PL/SQL functions can also be packaged as stored programs on the server. PL/SQL is also used in the definition of database triggers (Chapter 10).

PL/SQL is a block-oriented language. Every PL/SQL block consists of the following three components:

- *Declarative part.* Contains the names and data types of variables and constants as well as cursor declarations
- *Executable part.* Contains SQL DML statements embedded in control constructs such as IF-THEN-ELSE. May also contain commands for transaction management
- *Exception handling part.* An exception is an error condition defined either by the user or by the system. Such exceptions can be trapped by a series of declared 'exception handlers'

Example  Below we give an example of a simple PL/SQL program:

```

DECLARE
deptCodeConst lecturers.deptCode%TYPE := 'CS';
totalRecs := number(3);
BEGIN
SELECT count(*) INTO totalRecs
FROM lecturers
WHERE deptCode = deptCodeConst;

IF totalRecs > 0 THEN
update lecturers
set deptCode = null
WHERE deptCode = deptCodeConst;
ENDIF;

COMMIT;

EXCEPTION
WHEN no_data_found THEN
INSERT INTO error_log1 VALUES(sysdate, deptCodeConst);
WHEN others

```



```

err_code := sqlcode;
err_text := sqlerrm;
INSERT INTO error_log1 VALUES(sysdate, err_code, err_text);

END;

```

The PL/SQL block above first counts the number of computer studies lecturers. If there are lecturers for this department, it then proceeds to set their department code to null.

34.5

SEMANTIC INTEGRITY

ORACLE 7 introduced a number of facilities for embedding semantic integrity within a relational database. These facilities can be broadly distinguished in terms of whether they constitute procedural or declarative mechanisms.

34.5.1 DECLARATIVE MECHANISMS

The declarative mechanisms constitute variants of SQL89 syntax for primary-key, foreign-key and propagation constraints as well as check, unique and default (Chapter 11). Each such constraint can be given a name and specified either at CREATE TABLE time or separately using an ALTER TABLE command. This is a variant of the SQL2 CREATE ASSERTION command (Chapter 12).

Examples ➔ The examples below illustrate the process of declaring constraints in ORACLE:

```

CREATE TABLE Modules
  (moduleName CHAR(15),
   level NUMBER(1) CONSTRAINT m_level_1 DEFAULT(1) NOT NULL,
   courseCode CHAR(3),
   staffNo NUMBER(5),
   CONSTRAINT m_pk PRIMARY KEY (moduleName),
   CONSTRAINT m_fk FOREIGN KEY (staffNo) REFERENCES Lecturers)
   ON DELETE RESTRICT
   ON UPDATE CASCADE)

```

```
ALTER TABLE Modules ADD CONSTRAINT m_level_2 CHECK (level between 1 and 3)
```

The first example, if issued to an ORACLE DBMS, will create the modules table with appropriate primary and foreign keys. Note that the primary-key and foreign-key clauses have been given a name via a CONSTRAINT command. This is useful in that any infringement of such constraints will then throw up the name of the constraint in any error message. The second example illustrates how a constraint may be added to an existing table declaration. Here a CHECK clause is being used to ensure that the level attribute lies within a given range of values.

34.5.2 PROCEDURAL MECHANISMS

Integrity is maintained procedurally in ORACLE primarily via database triggers. A trigger is a procedure written in PL/SQL which is directly assigned to a database table and which is automatically activated by specific actions on that table. For example, suppose we have an enrolments table which records which students have enrolled for which modules. Inserting an enrolment record into this table may activate a trigger which automatically updates the current roll of a particular module.

An ORACLE trigger is characterised in terms of triggering statements (insert, update or delete), the trigger timing (before or after) and the trigger type (statement or row trigger). The trigger timing specifies whether the trigger should be executed before or after the triggering statement. The trigger type determines whether a trigger should be fired once per statement regardless of the number of rows affected or each time that row is modified.

Triggers are created using the CREATE TRIGGER command. A database trigger can contain any number of PL/SQL statements, SQL DML operations or calls to other PL/SQL programs. Triggers can also make use of special IF constructs that allow structuring of PL/SQL code so that it can handle multiple events within one trigger.

Triggers are mainly used to:

- Enforce complex business rules that cannot be enforced by integrity constraints
- Compute derived column values
- Audit database changes
- Implement advanced security requirements

Examples ➔ For instance:

- Ensure that a lecturer must be employed for at least three years before he can teach at MSc level
- Compute an increment point based upon the start date and status of a lecturer
- Record all changes made to the students table in a journal
- Ensure that a student evaluation cannot be changed by a lecturer

Below we define the general structure of a trigger definition:

```
CREATE TRIGGER <trigger name>
BEFORE/AFTER trigger timing
INSERT OR
UPDATE [OF <column1>, <column2>, ...] OR trigger event
DELETE
```



```

ON <table name>
{FOR EACH ROW} trigger type
{WHEN <condition>} trigger restriction
<PL/SQL code>
END;

```

Examples → Below we give an example of a statement level trigger and a row level trigger:

```

CREATE TRIGGER lecturer_delete_trig
AFTER DELETE
ON Lecturers
BEGIN
IF TO_CHAR(sysdate, 'HH24') > 13 THEN
    RAISE_APPLICATION_ERROR(-20010,'Not allowed to delete a lecturer now');
END IF;
END;

```

This trigger raises an application error if we attempt to delete a lecturers row after 1 o'clock.

```

CREATE OR REPLACE TRIGGER student_update_row
BEFORE UPDATE
OF totalGrant
ON Students
FOR EACH ROW
BEGIN
IF :new.totalGrant < :old.totalGrant THEN
:new.feeCompensation := :old.totalGrant - :new.totalGrant;
END IF;
END;

```

Assuming the structure of the students table is as follows:

```
Students(studentNo, studentName, totalGrant, feeCompensation)
```

this trigger updates the feeCompensation column depending on the relationship between the value for totalGrant currently stored and that being entered.



ORACLE supports the construct of an object type and closely follows the SQL3 syntax for specifying such types. Object types can be used in ORACLE for such purposes as:

34.6.1 SPECIFYING USER-DEFINED TYPES

A user-defined type is a customised data type defined for using in a particular set of applications.

Example  Suppose we wish to create an address as a specialised data type. We might specify it as follows:

```
CREATE OR REPLACE TYPE addressType AS OBJECT (
    HouseNumber INTEGER(4),
    Street VARCHAR2(50),
    Town VARCHAR2(50),
    County VARCHAR2(50),
    PostCode VARCHAR2(10))
```

We can then use this in *create table* statements or PL/SQL blocks as a valid data type.

34.6.2 CREATING NESTED TABLES

Object types can be used to nest tables within tables.

Example  Suppose we wish to nest line-item information within an orders table. This might be specified as follows:

```
CREATE OR REPLACE TYPE itemType AS OBJECT (
    ItemID INTEGER,
    Qty INTEGER)
CREATE OR REPLACE TYPE lineItems AS TABLE OF itemType

CREATE TABLE Orders (
    OrderID INTEGER PRIMARY KEY,
    OrderDate DATE,
    LineItem lineItems)
```

34.7 SUMMARY

- ORACLE is a relational DBMS with a robust kernel for medium to large-scale database applications
- ORACLE uses SQL as its database sublanguage
- ORACLE offers a range of tools for database administration and application development



- ORACLE provides a procedural fourth generation language with embedded SQL known as PL/SQL
- ORACLE provides a range of ways of implementing inherent and additional integrity including the use of triggers and stored procedures
- ORACLE 8, the current version of the DBMS, offers a number of SQL3 compliant object-relational features

34.8
ACTIVITIES

1. Explain what the following two statements are doing:

```
ALTER TABLE lecturers ADD CONSTRAINT L_status_1 CHECK (status IN ('L','SL','PL'));
ALTER TABLE Modules ADD CONSTRAINT L_staffNo_1 CHECK (staffNo BETWEEN 100 AND 999);
```

2. Write a constraint that the salaries of relevant staff should be in the following ranges:

status	min. salary	max. salary
L	£15,000	£20,000
SL	£20,000	£28,000
PL	£27,000	£34,000

Hint: *CHECK((salary BETWEEN 15000 AND 20000 AND status = 'L') OR ...)*

3. Explain what you think the following trigger definition is doing:

```
CREATE TRIGGER lecturer_del_cascade
AFTER DELETE ON lecturers
FOR EACH ROW
BEGIN
    DELETE FROM modules
    WHERE modules.staffNo = :old.staffNo;
END;
/
```

34.9
REFERENCES

- Abbey, M., M. Corey and I. Abramson (2002). *Oracle9i: A Beginner's Guide*. London, McGraw-Hill.
 Codd, E.F. (1985a). Does your DBMS run by the rules? *Computerworld* 49–64.
 Codd, E.F. (1985b). Is your DBMS really relational? *Computerworld* 1–9.

CHAPTER 35 →

O₂ DBMS

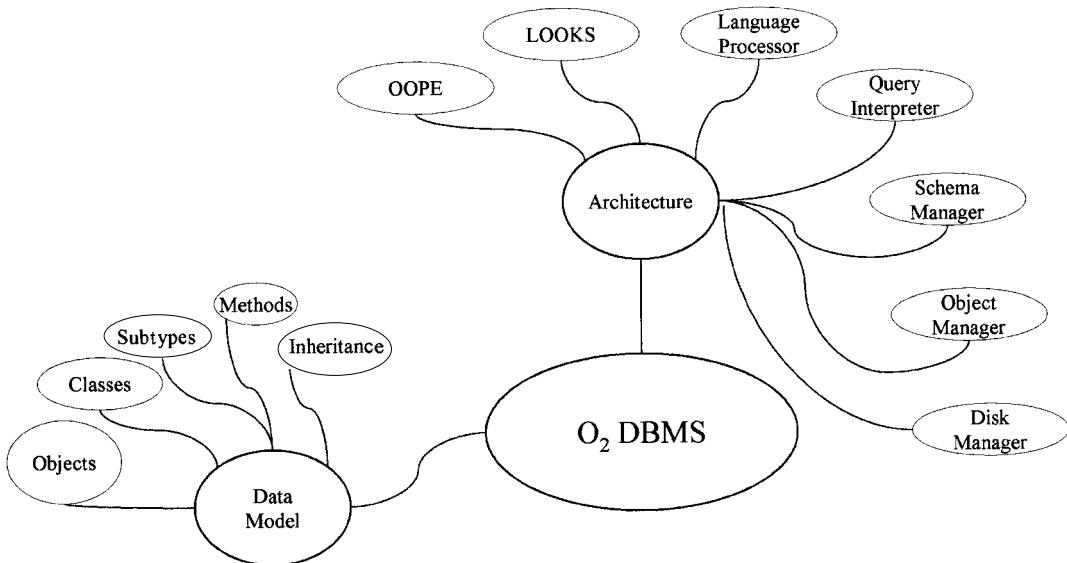
I believe in getting into hot water; it keeps you clean.

G.K. Chesterton (1874–1936)

LEARNING OUTCOMES

At the end of the chapter the reader will be able to:

- Describe a number of different approaches to implementing an OODBMS
- Discuss some of the key features of the O₂ DBMS



35.1 INTRODUCTION

An OODBMS is a piece of software designed to implement object-oriented constructs as described in Chapters 8 and 17. Object-orientation represents the amalgam of behaviour as traditionally implemented via programming languages and data as traditionally implemented in database systems. Hence, there are a number of strategies that may be used for implementing an OODBMS:

- *Extended programming languages.* One approach is to extend an OO programming language, such as Smalltalk, C++ or Java, with database capabilities. This is the approach taken by the Gemstone DBMS
- *Extensible libraries.* Another approach is to provide extensible OODBMS libraries. This differs from the approach above in that rather than extending the language, the system merely provides class libraries to support DBMS characteristics such as persistence. This is the strategy employed by the Ontos, Versant and Object Store DBMS
- *Embedded constructs.* This approach involves embedding OO database language constructs in a conventional host language. This is the approach taken by the O₂ DBMS described in this chapter. O₂ provides embedded extensions for the programming language C
- *Extended database sublanguage.* This approach involves extending an existing database sublanguage with OO facilities. This is the approach taken in the SQL3 standard (Chapter 31)
- *Novel data model.* This involves developing a novel data model and database sublanguage. This is the approach that is taken in much of the semantic data modelling literature (Chapter 10)

We concentrate on the third of these strategies in this chapter by reviewing the key features of the O₂ DBMS.

35.2 HISTORY

The O₂ system is an object-oriented DBMS which was designed and implemented between 1986 and 1989 by Altair, a French, non-profit-making body consisting of a number of research organisations (Bancilhon *et al.*, 1992). A commercial product was launched in 1991. The aim of the O₂ project was to develop a complete DBMS which could be seen as an advance on relational technology. The current version of O₂ is version 5.0.

35.3 O₂ DATA MODEL

O₂, being a DBMS that adheres to the object-oriented paradigm, utilises many of the concepts discussed in Chapter 8. As a member of the Object Database Management Group, O₂ is also ODMG compliant (Chapter 32).

A database system created using O₂ consists of objects that have identity and that encapsulate data and behaviour. Objects are defined in terms of classes organised in multiple inheritance hierarchies.

Examples → A number of examples of O₂ objects are given below:

```
i0: TUPLE(moduleName: 'Relational Database Systems',
level: 1,
roll: 0,
courseCode: i2,
staffNo: LIST(i1, i5))

i1: TUPLE(staffNo: 4324,
staffName: 'Beynon-Davies P',
status: 'PL')

i2: TUPLE(courseCode: 'CSD',
courseName: 'Computer Studies Degree')
```

These objects are made up of pairs of identifiers and values. i0, i1, i2 etc. are system-generated identifiers. The object i0 has a value of type tuple which in turn is made up of identifiers, attribute names and values. Note that some of these values are identifiers to other objects.

35.4 CLASSES

Objects are instances of a class. Classes in O₂ are characterised by having a name, a type and methods. Methods define the external interface to an object.

The type of a class describes the structure of the values encapsulated by objects. Types are constructed using a number of atomic types and constructors such as *tuple* and *list*.

Example → TUPLE(moduleName: STRING,
level: INTEGER,
roll: INTEGER,
courseCode: Course,
staffNo: LIST(Lecturer))

defines the type of the class module. Note that domains can be atomic types such as string and integer or instances of a class such as *course* and *lecturer*.

A class is created in O₂ by using the add class command.

Example → ADD CLASS module
 TUPLE(moduleName: STRING,
 level: INTEGER,
 roll: INTEGER,
 courseCode: Course,
 staffNo: LIST(Lecturer))

35.5 PERSISTENCE

Objects are made persistent in O₂ by defining names as ‘handles’ for applications. Defining a name for an object involves using data definition commands such as add name.

Example → ADD NAME relationalDatabaseSystems: module

Here we are persistently associating the name *relationalDatabaseSystems* with the class *module*. The name *relationalDatabaseSystems* is then a global name for an object in the module class. If we then associate this name with the object *i0* declared in section 35.3 then not only the values of *i0* but also the objects *i1*, *i2* and *i5* will be made persistent, since any object or value referred to by a named object is made persistent.

35.6 METHODS

Methods are defined in two stages. First the developer must specify a method’s ‘signature’ to O₂ using the add method command.

Example → ADD METHOD increaseRoll(amount: integer) IN CLASS module

This method will be used to increment the module roll by a given integer amount. The body of a method is defined in an application language such as CO₂ – the standard programming language C supplemented with an object-oriented layer.

35.7 SUBTYPES AND INHERITANCE

An inheritance hierarchy can be defined in O₂ using a variant of the add class command.

Example ➔ ADD CLASS partTimeLecturer INHERITS academicStaff, partTimeStaff

Here a part-time lecturer class is defined as inheriting all the attributes and methods of the academic staff and part-time staff classes.

35.8 DATA MANIPULATION

Data maintenance is conducted in O₂ via methods. Methods are implemented in an applications language such as CO₂.

Example ➔ The method below implements the increaseRoll function defined as a signature in section 35.6:

```
BODY increaseRoll(amount: integer) IN CLASS module
co2 {*self.roll += amount;}
```

Here the value of the object to which the method is applied is written *self. The += operator increases the value of the roll attribute by the given amount. In O₂ you apply a method to an object by sending a message.

Example ➔ The message

```
[relationalDatabaseSystems increaseRoll(30)]
```

will activate the method declared above and increase the roll for relationalDatabaseSystems by 30.

Within CO₂, objects are created using the command *new*.

Example ➔ relationalDatabaseDesign = new(module).

To assign values to such a new object we would write:

```
*relationalDatabaseDesign = tuple(moduleName: 'Relational Database Design',
level: 2,
roll: 0,
courseCode: CSD,
staffNo: LIST(234))
```

35.9 QUERY LANGUAGE

O_2 also has an in-built query language known as OQL (object query language). The syntax of OQL is SQL-like.

Examples  Some OQL queries are provided below:

```
SELECT m.moduleName
FROM m IN module
WHERE level = 1
```

This query extracts the names of all instances in the class module that are level 1.

```
TUPLE(roll: relationalDatabaseSystems.roll, level: relationalDatabaseSystems.level)
```

Here we are extracting the current roll and level of the object named relationalDatabaseSystems.

```
SELECT TUPLE(module: m.moduleName, lecturer: l.staffName)
FROM m IN module, l in lecturer
WHERE l.status = 'PL' AND m.roll > 30
```

This query will retrieve names of Principal Lecturers (PL) that are teaching modules with more than 30 students.

35.10 APPLICATIONS PROGRAMMING

An O_2 application program is made up of programs and global variables. An application program differs from a method in that it is not attached to a class and contains transaction management commands. A program's body is written in CO_2 . A global variable is used to store temporary data that exists during the session that an application is running.

Applications are added to an O_2 database using the add application command.

Example  ADD APPLICATION moduleAdministration
 Variable
 accessDate: Date
 PROGRAM
 moduleInformation(modules: LIST(module)),
 updateStaff().

35.11  O_2 ARCHITECTURE

The O_2 system is made up of seven modules:

- *OOPE* – provides facilities necessary for the development of applications; it particularly emphasises graphical construction of applications
- *LOOKS* – an interface generator, responsible for managing the screen, the display of objects and interactions with the object manager; it allows straightforward interfaces and user dialogs to be built quickly
- *Language processor* – handles DDL commands, updates the schema via the schema manager and compiles methods
- *Query interpreter* – handles queries written in OQL and executes them with reference to the schema and object managers
- *Schema manager* – responsible for managing information about classes and methods
- *Object manager* – converts between objects in the O₂ data model and their disk representation
- *Disk manager* – responsible for disk I/O

O₂ is normally installed on a client–server configuration (Chapter 36). A workstation and server version of O₂ exist, each having an identical interface but different functionality. The workstation version is single-user and memory-resident. The server version is multi-user and disk-oriented. Workstation systems will primarily be involved in running OOPE, LOOKS and/or the query processor. The server system is particularly concerned with running the disk, schema and object manager.

35.12 SUMMARY

- O₂ is a commercial OODBMS and constitutes an offshoot of a French research project
- O₂ adheres to the ODMG OODBMS standard
- O₂ schemas can be expressed in terms of objects, classes and methods
- Applications can be created using CO₂, OOPE and LOOKS

35.13 ACTIVITIES

1. Create a class for the class Lecturer using O₂ syntax.
2. Write an OQL query that lists all the lecturers having PL status.

35.14 REFERENCE

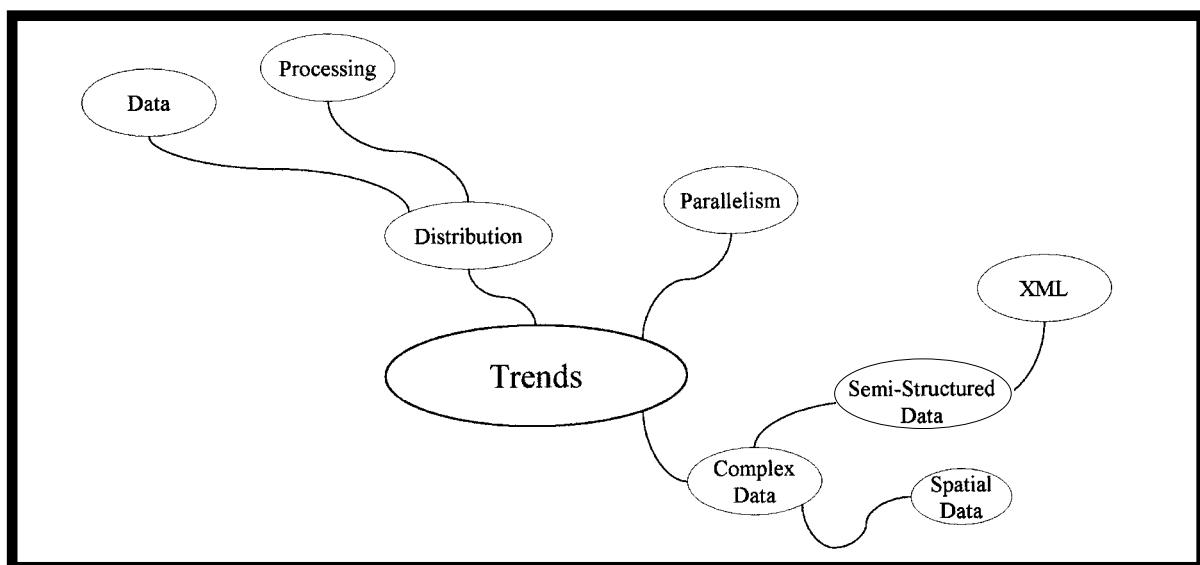
Bancilhon, F., C. Delobel and P. Kanellakis (Eds) (1992). *Building an Object-Oriented Database System: The Story of O₂*. San Mateo, CA, Morgan Kaufmann.

PART 9

TRENDS IN DATABASE TECHNOLOGY

I derive no pleasure from talking with a young woman simply because she has regular features.

Henry David Thoreau (1817–62)





Because databases are such a significant part of modern-day information systems, it is impossible in the space we have available to discuss all the possible developments for this technology. Instead, we choose to concentrate here on three strands which are already determining the functionality of future database systems: distribution, parallelism and handling multiple media.

Chapter 36 considers the issue of distributing processing across a network at different sites. This trend of distribution is a key feature of the push towards making the network the focus of information system applications. Chapter 37 continues this theme in considering the issue of distributing data.

Parallelism is effectively the idea of building computer technology from collections of processors and memories. Parallel computer technology has been applied to the problems of database management for a number of years and has proven its effectiveness in significantly improving the efficiency with which high volumes of transactions impact against a database. This important issue is the topic of Chapter 38.

As information systems have impinged on more and more areas of organisational life, so the concept of a database has met with more and more complex application domains. Chapter 39 is devoted to the issue of handling spatial and semi-structured data.

CHAPTER 36 →

DISTRIBUTED PROCESSING

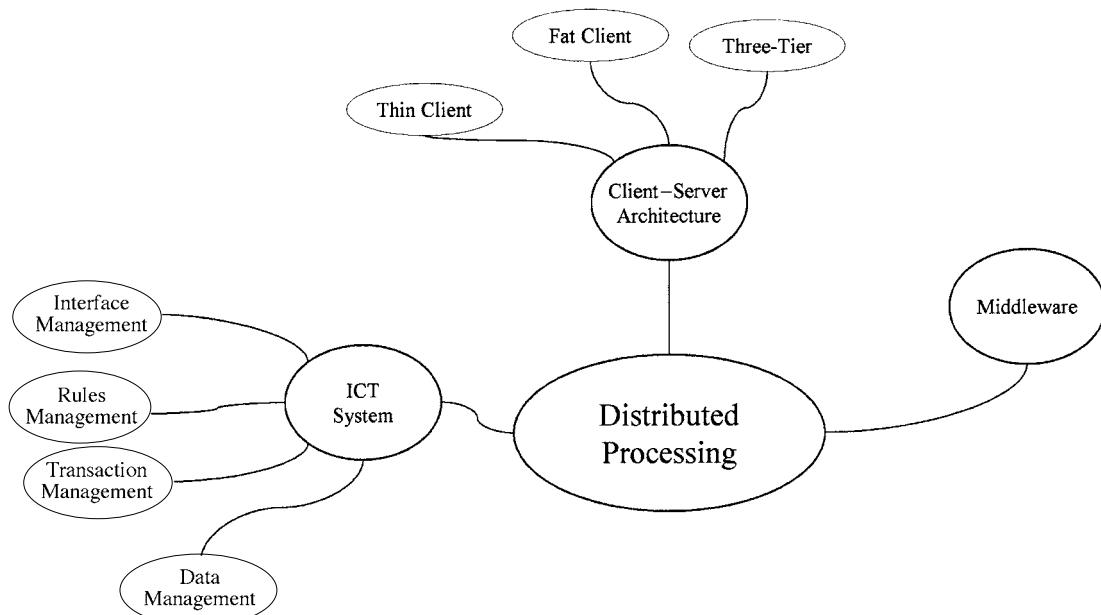
The future is here. It's just not widely distributed yet.

William Gibson (1948–)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define the layers of an ICT system
- Discuss some of the major options in distributing processing in a database system
- Relate some of the activities involved in building client-server database systems



36.1 INTRODUCTION

It is possible to make the distinction between two dimensions of distributed database systems: distributed data and distributed processing. Chapter 37 covers the issue of distributed data. In this chapter we provide an overview of the topic of distributed processing.

Distribution can also be discussed in terms of distribution of functions or processing. Figure 36.1 illustrates a system characterised by distributed processing. Here, we also have four sites connected by a communications network. However, in this configuration only site 1 stores any data. Sites 2, 3 and 4 act as clients to this database, perhaps running particular information system applications. It is for this reason we include client-server database systems within our discussion of distributed database systems. Although most current client-server database systems do not effectively distribute data, they do distribute functionality.

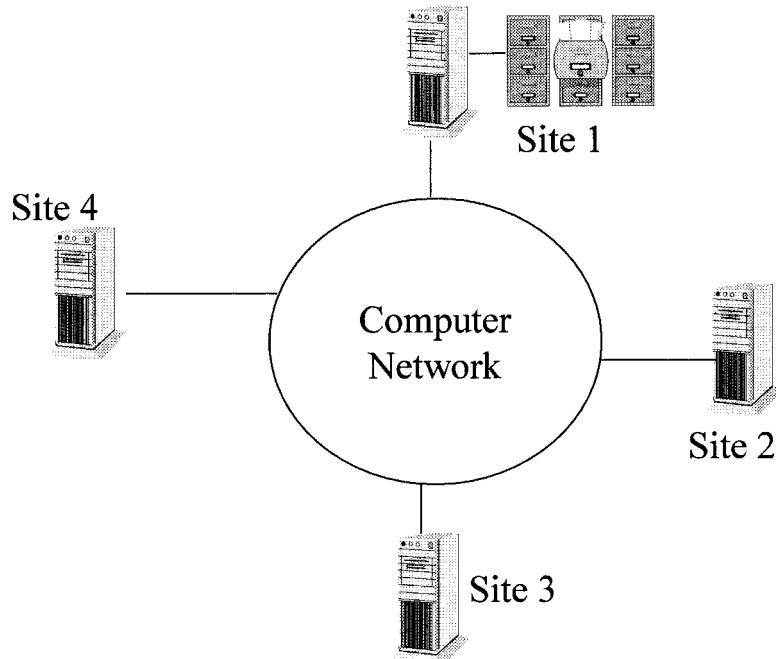


Figure 36.1 An example of distributed processing.

36.2 LAYERS OF AN INFORMATION AND COMMUNICATIONS TECHNOLOGY SYSTEM (ICT)

In terms of software it is useful to consider an ICT system as being made up of a number of subsystems or horizontal layers (Figure 36.2). The layers define the major component parts of processing in an ICT system:

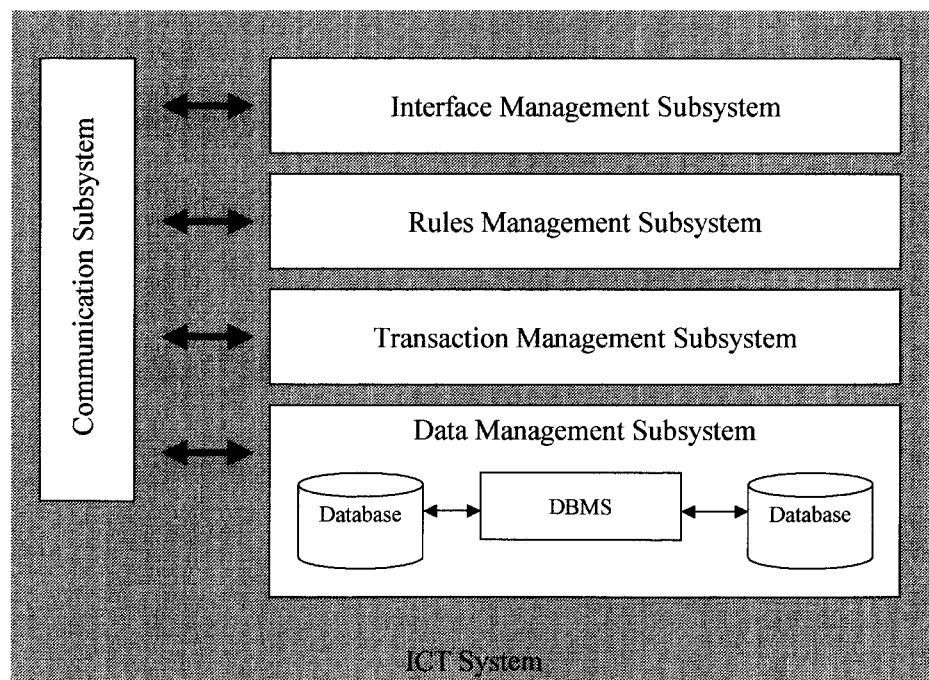


Figure 36.2 Layers of an ICT system.

- *Interface subsystem.* This subsystem is responsible for managing interaction with the end-user. It is generally referred to as the user interface
- *Rules subsystem.* This subsystem manages the application logic in terms of a defined model of business rules. In a database application these will comprise functions which primarily enforce both inherent and additional constraints on data
- *Transaction subsystem.* This subsystem acts as the link between the data subsystem and the rules and interface subsystems. Querying, insertion and update activity are triggered at the interface, validated by the rules subsystem and packaged as units (transactions) that will initiate actions (responses or changes) in the data subsystem
- *Data subsystem.* This subsystem is responsible for managing the underlying data needed by an application. This includes the management of concurrency, storage and security. We described this as the data management layer in Chapter 6

In the contemporary ICT infrastructure each of these parts of an application may be distributed on different machines, perhaps at different sites. This means that each part usually needs to be stitched together in terms of some communications backbone. For consistency, we refer to this facility here as the communications subsystem.



Example ➔ Consider an ICT system for storing research publications in a university:

- One part of the interface will be a data entry form to enter details of a journal publication
- One of the rules or constraints used to validate data may be that the date entered for the publication must be less than or equal to today's date
- A key transaction will be the update function involved in the entry of new publication data into the system
- Part of the data management layer will have data structures for the storage of publication data

36.3



CLIENT-SERVER ARCHITECTURE

The parts of a conventional ICT or database application are normally distributed in terms of some form of client–server architecture. Client–server is a software architecture in which two processes interact as superior and subordinate. The client process always initiates requests, and the server process always responds to requests. Theoretically, client and server can reside on the same machine. Typically however they run on separate machines linked by some form of communications network, usually a local area network.

Many different types of server exist, for example:

- Mail servers
- Print servers
- File servers
- Database (SQL) server

Most people tend to equate the term client–server with a network of PCs or workstations connected by a network to a remote machine running a database server.

In practice, a client–server database system generally refers to a local area network of personal computers (PCs). At least one of these PCs is dedicated to serve the database needs of the others, which act in a client capacity. The database is held on the server. The user interface and application development tools are held on the client machines.

The server in this configuration is either set up as a file server or SQL server. In a file server situation, an SQL query expressed by a client will issue a request to the server for the appropriate files needed by the query. The client will perform the query and extract the relevant data. In an SQL server situation, the SQL statement will travel down the communication line from client to server. The server then executes the query and sends back only the extracted data.

Clearly, because of the reduced communication traffic, most DBMS now offer SQL server facilities.

36.4 (c) DISTRIBUTION PATTERNS

In Figure 36.3 the diagram is divided vertically into client computers and server computers. Clients request services from server computers. The figure illustrates a number of different distribution patterns for ICT systems.

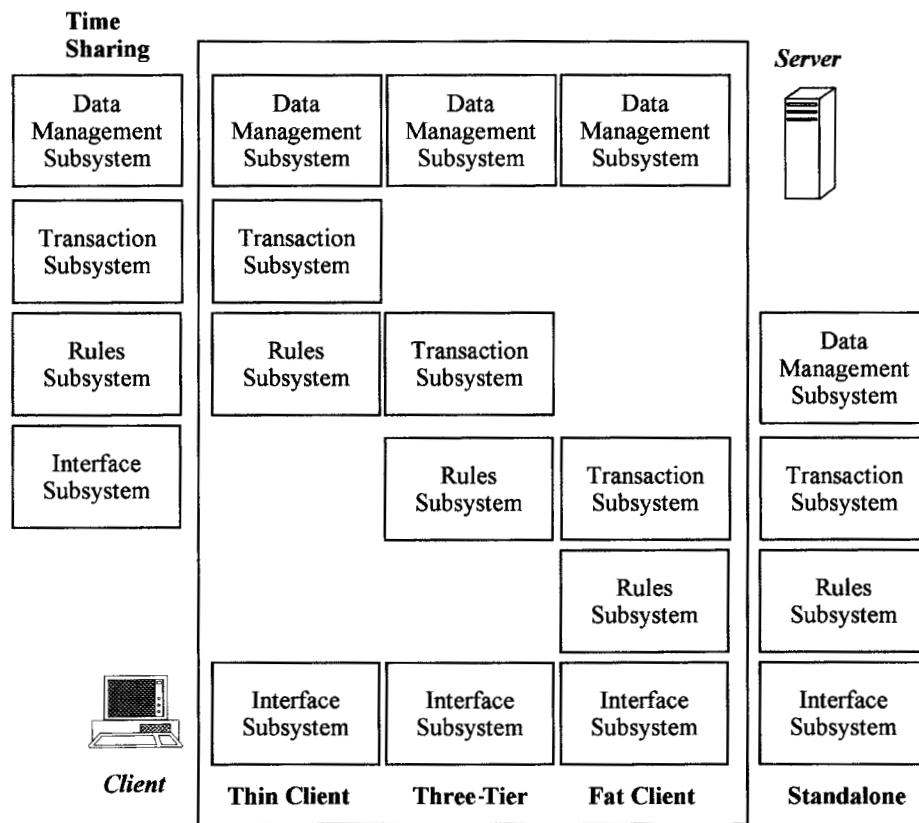


Figure 36.3 Distribution options.

36.4.1 TIME-SHARING

In the first phase of information technology systems, each of the component layers of such a system was located on one machine. Large mainframe systems ran the data management, transaction management, rules management and much of the interface management functions of an application. Connection to such systems was via so-called 'dumb' terminals. They were referred to as 'dumb' because they contained very little inherent functionality and primarily



enabled operators to control systems via character-based interfaces – interfaces demanding users to type in commands. Most of the processing on such systems was conducted in batch mode, i.e. the processing of vast amounts of transactions in sequence with very little direct user input. Specialist data entry staff conducted data entry. Retrieval of information was available through paper reports produced from the system.

36.4.2 FAT CLIENTS

Over time, more and more functionality has been placed on the clients. Technological developments enabled the user interface layer and some of the rules management layer to be located on so-called ‘intelligent’ terminals. They were referred to as ‘intelligent’ because they were able to take some of the processing off the centralised mainframe or minicomputer. This advance enabled the development of on-line systems. Such systems enabled users in the workplace to enter data directly into the information system and in certain respects to query the data in the system. The rise of on-line systems enabled the development of management information system (MIS) and decision support system (DSS) applications.

With the rise of personal computers, much more of the functionality of the information technology system began to be placed on desktop machines. The sophistication of graphical user interfaces meant that more power needed to be available on the desktop to run such interfaces effectively. With the rise of software for the desktop, slices from the total functionality of an information technology system could be built using application development tools available for the desktop.

36.4.3 THREE-TIER

Many current applications run on three-tier client–server architectures. The first tier runs the user-interface layer, the second tier runs the application logic and the third tier runs the transaction and data management functions. In a Web-based approach (Chapter 43) the client will run a browser, and the middle tier will comprise a Web server that will interact with a database server. Modern systems may extend this three-tier architecture to an N -tier architecture – meaning that more than three layers may be involved in the distribution of processing.

36.4.4 THIN CLIENTS

Some discussion has recently occurred within the computing fraternity over the way in which Internet technology may cause a profound change from ‘fat’ clients to ‘thin’ clients. The corporate PC is currently a fat client. It requests information from a server and then processes and presents it at the client end using software resident on the PC. The network computer or NC is a thin client. In its extreme form, only a minimum of software (usually a Web

browser) will be resident on the client. Applications will be resident on and accessed from the server. This means that each user will have a desktop system that looks like a PC but which has no secondary storage devices. Consequently, NCs, or so the argument goes, are likely to be considerably less expensive than a PC. Also, the network administrator will only need to buy and maintain one copy of each software application on the server.

36.5 MIDDLEWARE

Figure 36.4 illustrates the fact that another collection of software will normally need to exist in a client–server database system. This class of software has come to be known as middleware. Middleware software provides ‘transparent’ linkage between client application and server data. It makes server data appear local to the application on the client. Middleware will use communications software to exchange messages between client and server.

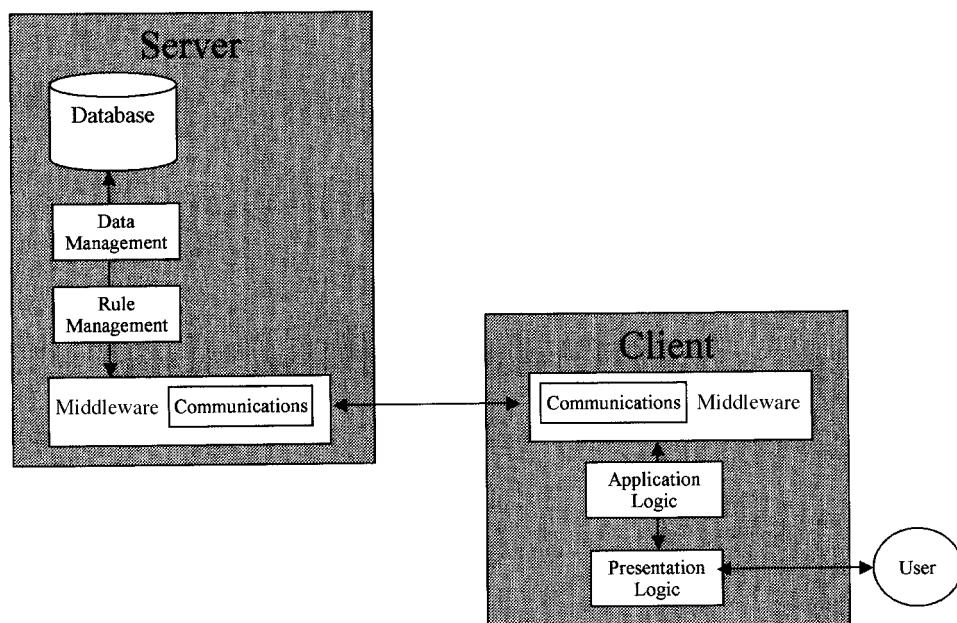


Figure 36.4 Middleware.

Middleware may be offered by third-party vendors, be part of some vendor attempts at standardisation such as ODBC, or be specific to a DBMS vendor such as ORACLE's SQL*NET.

Example → Application programming interfaces, such as embedded SQL or ODBC (Chapter 25), are key examples of important middleware products in the database systems area.

**36.6****BUILDING A SIMPLE CLIENT-SERVER DATABASE SYSTEM**

Because of the physical separation of client and server, such a system is subject to a number of constraints arising from the marriage of processing with communication. In designing an effective client-server database system one should seek to:

- Offload as much logic, particularly presentation logic, as possible at the client end. This serves to accelerate feedback to the user and reduce the load on the server
- Place some business rules enforcement, particularly that involved in type checking of input, at the client
- Minimise traffic on the communication line by constraining the number and size of requests from the client
- Implement important business rules on the server, particularly those relevant to the entire business or business area

36.7**CASE STUDY: AN ATM SYSTEM**

Consider, for example, a simple application that emulates many of the features of an ATM (automatic telling machine) for a bank.

At the client end we would implement a number of screens with associated dialogue. This would first check that a number entered by the customer matched an identifier on record. This would enable the customer to select one of two options off a menu: retrieve account balance or withdraw funds. If the customer chose to withdraw funds then the program would check to see if the amount entered was a valid number.

At the server end we will have implemented tables such as a customer and account table. We will have also specified rules such as a customer should not be able to overdraw to a degree greater than his or her overdraft limit. We would also need to have specified two transactions: one to check an entered ID against a recorded ID (T1), the other to update the account balance (T2).

This would mean that the type of messages that would travel between client and server would be:

- From client: Run transaction T1 to check this ID
- From server: Result of T1 – true or false
- From client: Run transaction T2 to update balance
- From server: Result of T2 – success or error due to constraint

36.8  SUMMARY

- Distribution is normally discussed solely in terms of the fragmentation and replication of data. Distribution can also be discussed in terms of distribution of functions or processing
- Client–server database systems may be included in the definition of distributed database systems
- The various parts of an ICT system may be distributed at different parts of the network
- Middleware is critical to the effective operation of the client–server architecture

36.9  ACTIVITIES

1. In terms of an application system known to you, attempt to distinguish between its various layers.
2. Determine whether the data management layer of such an application is resident on one or more specialised servers.
3. Determine whether client machines in the network are thin or fat clients.



CHAPTER 37 →

DISTRIBUTED DATA

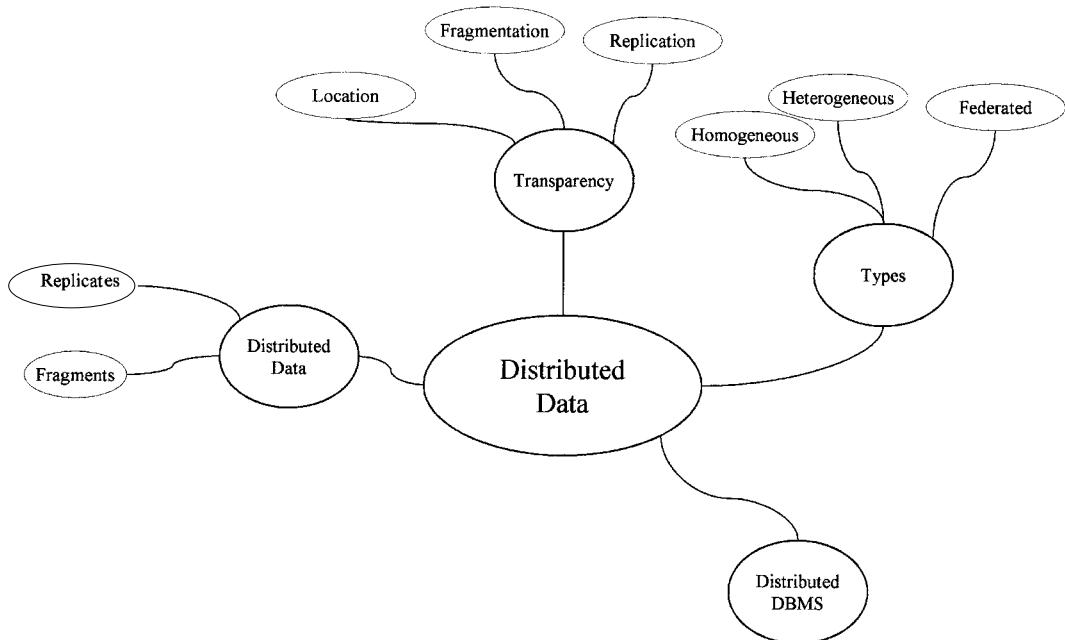
I propose getting rid of conventional armaments and replacing them with reasonably priced hydrogen bombs that would be distributed equally throughout the world.

Idi Amin

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define the key objectives of a distributed database system
- Discuss some of the advantages of distributed database systems
- Describe the major types of distributed database system
- Discuss some of the problems in building distributed database systems



37.1 INTRODUCTION

The 1990s were portrayed by many as the era of the distributed database system (Özsu and Valduriez, 1991). We begin this chapter by describing some of the major objectives, advantages and disadvantages of distributed database systems. This leads us to discuss some of the major types of distributed database system. We conclude with a brief discussion of the issue of distributed database design.

37.2 DISTRIBUTED DATA

A distributed database system is a database system which is fragmented or replicated on the various configurations of hardware and software, located usually at different geographical sites within an organisation. Distribution is normally discussed solely in terms of the fragmentation and replication of data. A data fragment constitutes some subset of the original database. A data replicate constitutes some copy of the whole or part of the original database. Figure 37.1 illustrates the idea of a distributed database system. Four sites are connected by a communications network. Sites 1, 2 and 4 each run a database. Each database at these sites stores fragments and/or replicates from the global distributed database system. Site 3 stores no data but accesses the data stored at one or more of the other sites over the communications network.

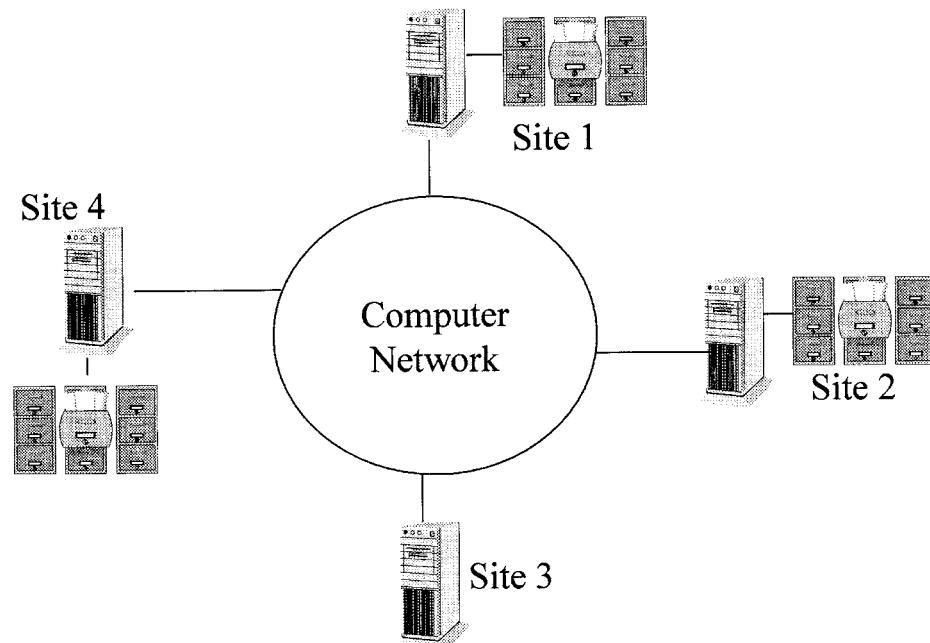


Figure 37.1 An example of a distributed database system.



Example → Normally we would expect a distributed database system to be a representation of a single business data model. For example, suppose we construct a data model which reflects the practice of human resource management within our organisation. We might decide to implement this data model as a distributed system such that our Edinburgh office maintains Scottish personnel data, our Cardiff office maintains Welsh personnel data and our London office maintains English personnel data. However, the data at all three regions would be brought together periodically to provide a national picture.

37.3
FRAGMENTATION AND TRANSPARENCY

Sometimes it is useful to distinguish between horizontal and vertical fragments. In relational terms a horizontal fragment amounts to a subset of the rows of a table. The scenario above could be considered as a series of three horizontal fragments of human resource information. In contrast, a vertical fragment is a subset of the columns of a table. A vertical fragment might amount to the personnel number, date of birth and salary of each employee. Horizontal fragmentation is illustrated in Figure 37.2.

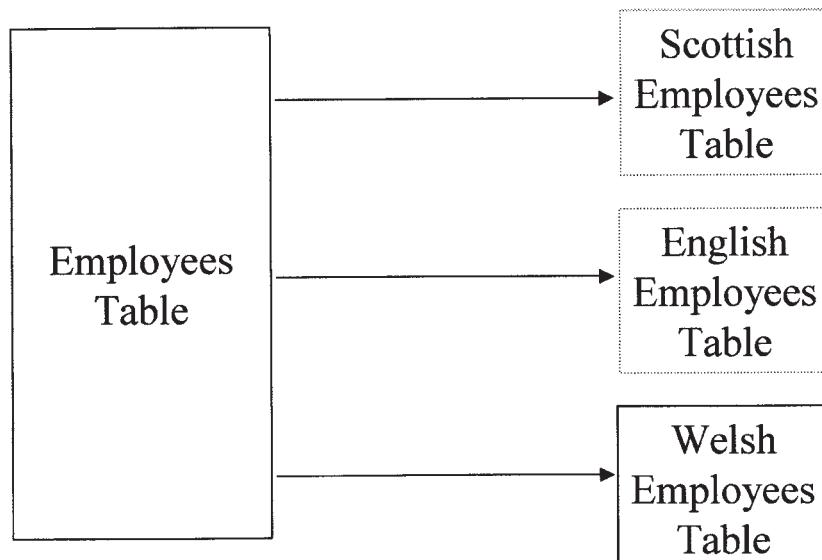


Figure 37.2 Fragmentation example.

Figure 37.3 displays a number of possible replicates from the same example.

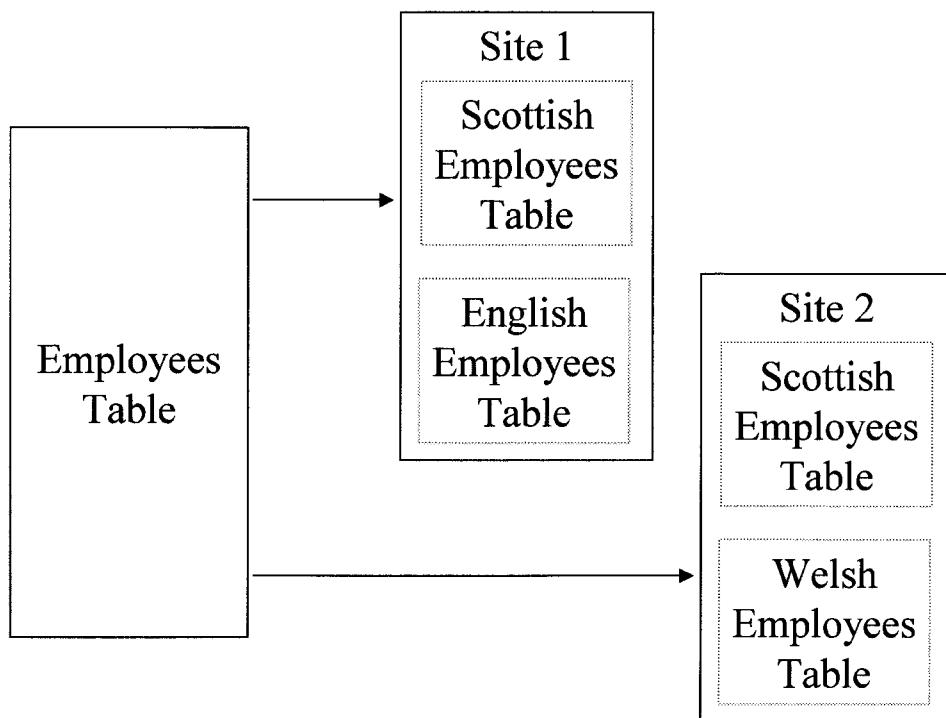


Figure 37.3 Replication example.

Distribution of data should not be apparent to the user. In other words, any distributed database system should ideally display three types of transparency:

- *Location transparency.* Users do not need to be aware at what sites data is located. Location independence is desirable because it simplifies user programs and interface activities. Data can migrate from site to site without invalidating any of those programs or activities. Data may be migrated around the network in response to changing usage or performance requirements
- *Fragmentation transparency.* Users do not need to be aware of how data is fragmented
- *Replication transparency.* Users should not need to be aware of how data is replicated. Replication is desirable because performance is better if applications can operate on local copies and availability is better so long as at least one copy remains available for retrieval purposes

Example ➔ In terms of our example distributed database system, the three types of transparency are important because of the following:



- *Location transparency.* A manager wishing to find the total number of employees at the Scottish subsidiary need not be aware that he is querying a remote database
- *Fragmentation transparency.* In terms of our human resources example, there is one logical personnel database but three physical fragments. Therefore, a manager running a query in London should not need to be aware that to produce the aggregate salary bill for the company all three sites – London, Cardiff and Edinburgh – need to be interrogated
- *Replication transparency.* In our human resources database, information about the departmental structure of the company is held as a copy at each site. When periodically this data needs to be updated, the user need not directly know that three sites are effectively updated

37.4
FOUNDATION RULE

The three types of transparency described above are designed to fulfil the overall objective of a distributed database system, which is to look exactly like a centralised database system to the user (Date, 1987). This is known as the foundation rule for distributed database systems. A number of characteristics of a distributed database system arise from the need to fulfil these three types of transparency:

- *Local autonomy.* The sites in a distributed database system are autonomous to the maximum extent possible
- *No reliance on a central site.* No reliance on a central site is a corollary of the foundation rule. Also, it is desirable because reliance on a central site could prove a performance bottleneck and the entire system is vulnerable to failure at the central site

37.5
RELATIONAL DATABASE SYSTEMS AND DISTRIBUTION

In many senses, the push towards distribution in the database arena would not have been possible if it were not for the rise of the relational database approach. Relations or tables are particularly easy to fragment and replicate. The process of fragmenting and replicating relations merely involves using the operators of the relational algebra or relational calculus.

Vertical fragmentation is performed by projecting out columns from a table with primary keys. Reconstituting vertical fragments involves performing joins using the original keys. Horizontal fragmentation is performed by restricts. Reconstituting horizontal fragments involves performing unions.

In practice, some combination of horizontal and vertical fragmentation will be performed using the SQL *Select* command (see Figure 37.4).

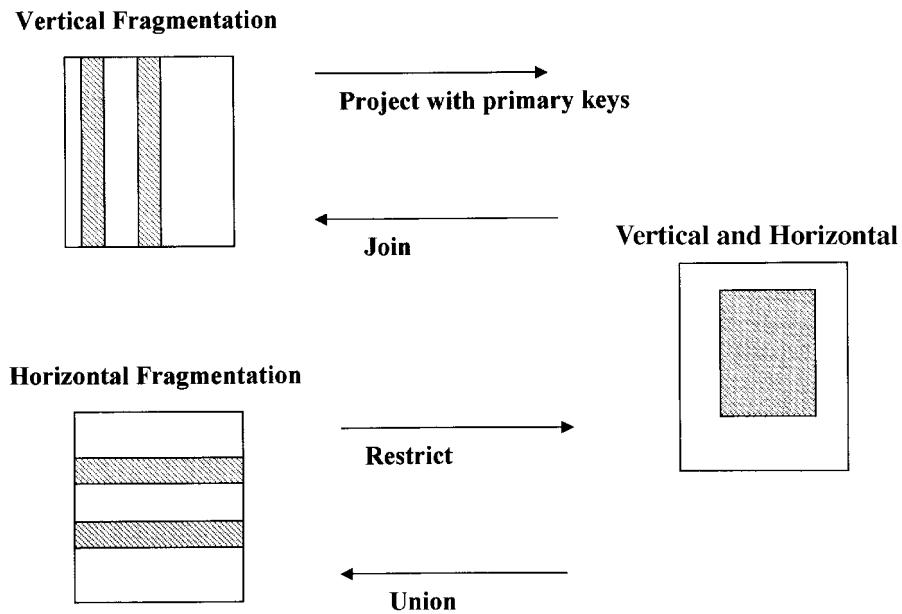


Figure 37.4 Fragmentation.

37.6 ADVANTAGES OF DISTRIBUTED DATA

Distributed database systems are much more complex than centralised systems because of the added factor of communication lines. In general terms, central processor activity is much faster than input/output (I/O) activity and I/O activity in turn is much faster than communications activity. Managing the traffic along communication lines effectively is therefore a major objective of the distributed DBMS.

Because of these added complications, distributing data is not easy. Why then do organisations consider the construction of distributed database systems? Some reasons are given below:

- *Emulating organisational structure.* It may be important to emulate the geographical distribution of an organisation in terms of its data systems. If the organisation is national or multi-national, the computing systems will have to emulate the distribution of departments, subsidiaries etc.
- *Greater control.* Greater control over data may occur if data is devolved to the places where it is needed
- *Greater reliability.* Keeping replicates of data may increase the reliability of systems. Siting data where it is needed increases a system's availability
- *Better performance.* Performance can actually increase if distribution is judiciously applied. If most queries go to a local small database, rather than a



large central one, then both update and retrieval access are likely to be improved

- *Easier growth.* A distributed environment may enhance the organisation's ability to expand its data infrastructure

Example ➔

- *Organisational structure.* In our personnel example it may be important to provide three database systems in each of England, Scotland and Wales because of the way in which the organisation is structured in terms of these three UK regions
- *Greater control.* If we devolve responsibility for Scottish data to Scottish employees, only these staff in the organisation can be given update access to operational data
- *Greater reliability.* In major high street banks, large database systems handling customer transactions have to run 24 hours a day, seven days a week. In such situations, the business cannot afford the system to go down. It is therefore commonplace to effectively run two database systems – one in the foreground doing the work, the other in the background emulating all the activity. The data in these systems is therefore effectively replicated
- *Better performance.* In terms of our personnel example we can ensure fast retrieval performance to Scottish data if the Scottish database is sited locally
- *Easier growth.* Databases are likely to grow astronomically in size in the modern commercial world. Increased storage capacity can be added to the organisation network to cope with these growing demands

37.7 ➔

TYPES OF DISTRIBUTED DATABASE SYSTEM

We shall distinguish between three major types of distributed database system (Ceri and Peligatti, 1984): :

- A homogeneous distributed database system
- A heterogeneous database system
- A federated (multi-database) database system

37.7.1 HOMOGENEOUS SYSTEMS

A homogeneous distributed database system is one in which data is distributed across two or more systems, each system running the same DBMS (e.g. ORACLE). Frequently, this type of distributed database system will be running on the same types of hardware under the same operating system with the same DBMS. This is illustrated in Figure 37.5.

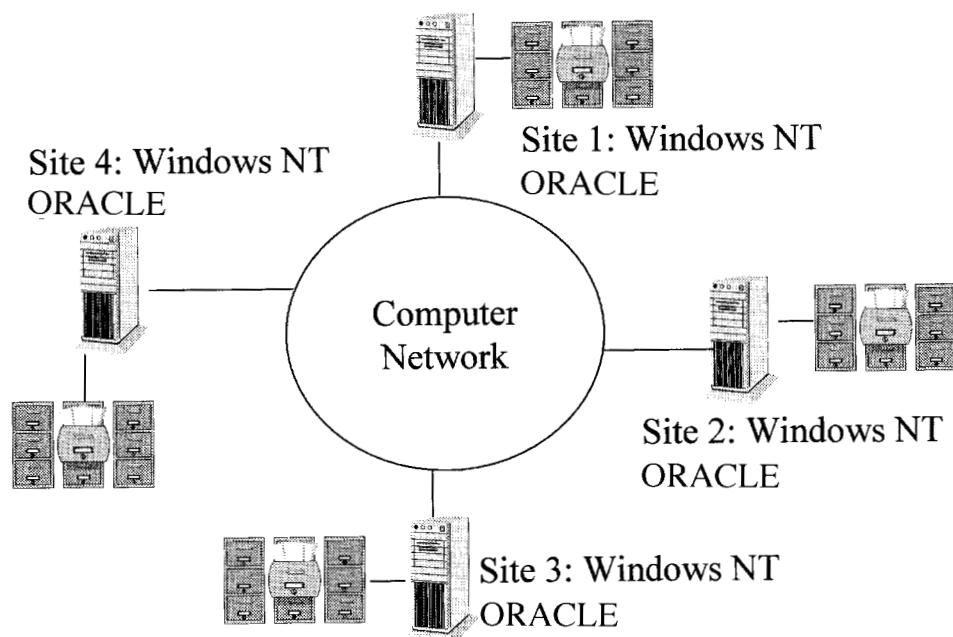


Figure 37.5 Homogeneous distributed database system.

37.7.2 HETEROGENEOUS SYSTEMS

A heterogeneous distributed database system is one in which the hardware and software configuration is diverse. One site might be running ORACLE under Windows NT, another site Informix under UNIX, and yet another site Ingres under Windows NT. Currently the major way in which heterogeneity is achieved is via the concept of a gateway. A gateway is an interface from one DBMS to another, usually supplied by a specific DBMS vendor.

37.7.3 FEDERATED SYSTEMS

The idea of a federated, sometimes referred to as a multi-database, distributed database system is similar to the political model of a federation. A country such as Switzerland is made up of a number of autonomous political units. Occasionally such units will come together to make decisions at the national level.

A federated database system is made up of a number of relatively independent, autonomous databases. Occasionally we may wish to bring some or all of these separate databases together to give us a more global picture.

Although some aspects of federation are beginning to emerge with the push towards open systems interconnection and the idea of a standard for remote database access, multi-database systems are still very much an ideal to which a number of vendors are working towards. Developments in areas such as XML



(Chapter 39) may encourage open transmission of data between diverse database systems.

37.8

DISTRIBUTED DBMS

To support the objectives of distributed DBMS as discussed in section 37.3, a distributed DBMS has to be a much more sophisticated piece of software than has a centralised DBMS. For instance:

- The system catalog of a distributed database has to be more complex. For instance, it has to store details about the location of fragments and replicates
- Concurrency problems are multiplied in distributed systems. The problems of propagating updates to a series of different sites are very involved
- A query optimiser in a true distributed system should be able to utilise information about the structure of the network in deciding how best to satisfy a given query
- To ensure a robust system, the distributed DBMS should not be located solely at one site. Software as well as data need to be distributed

It is useful to talk of two phases of implementing distributed databases. In the first phase we distribute queries between sites but update only to a single site. In the second phase we not only distribute queries, we also distribute transactions between sites. The latter scenario is clearly the more technically challenging of the two. Most existing distributed database systems are in phase 1. Very few organisations seem to have solved all of the problems associated with phase 2 applications (Özsu and Valduriez, 1992).

37.9

DISTRIBUTED DATABASE DESIGN

The ability to distribute data and processing among different nodes in a network is now a commonplace feature of modern DBMS. The design for distributed database systems is therefore an important aspect of modern physical database design (Chapter 19).

Distributed database design can be seen as a variant of physical database design. The first stage in distributed database design is to develop a data model in the same way as for a centralised system. The requirements of distribution may then mean that some changes need to be made to the data model. The second stage is to decide how to fragment and replicate the logical schema. A data fragment represents a horizontal and/or vertical fragment of tables in the database. The main aim of distributed database design is to reduce communications traffic. We wish to locate data close to its point of use. Hence a consideration of the topology of the communications network will influence the distributed design. This process is illustrated in Figure 37.6.

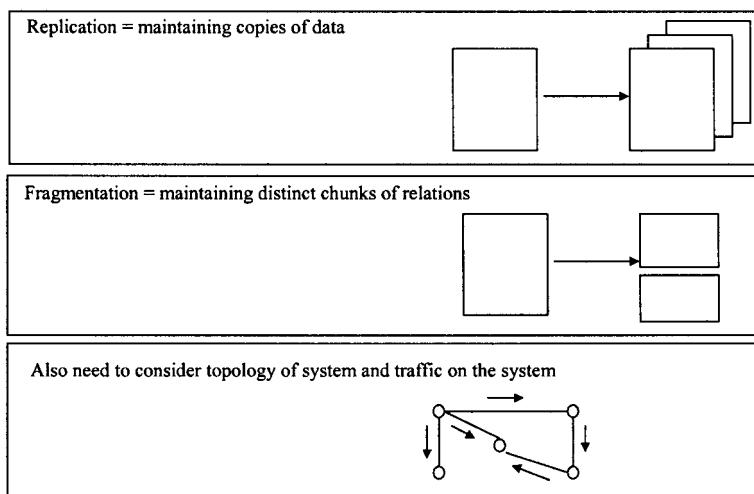


Figure 37.6 Distributed database design.

37.10 CASE STUDY: DISTRIBUTING THE USC SCHEMA

To provide an example of distributed database design, suppose the USC company (Chapter 19) is doing so well that it decides to offer its courses at four sites within the UK: Cardiff, Edinburgh, London and Belfast. Eventually it aims to offer its courses in mainland Europe, and perhaps the USA. At each site that runs its courses, USC intends to locate a database system. Each database system will form a part of the global USC database.

To help us decide on an appropriate distribution strategy we can use the idea of volatility as discussed in Chapter 19. We have already determined the volatility of each table in the USC schema using volume and usage information. In general, we would wish to fragment large, volatile tables and replicate small, non-volatile tables. In the USC case, for instance, we would fragment students, presentations and attendance, probably on the basis of location. If a location attribute is not currently part of the schema then this clearly needs to be added. In other words, Edinburgh would store Scottish student information, London English student information and so on.

As far as replication is concerned, we would wish to replicate lecturers, courses and qualifications. Such information is relatively static in the sense that it will only be changed relatively infrequently. It therefore makes sense to update each site's copy of this information on a periodic basis.

37.11 SUMMARY

- A distributed database system is a database system in which data is fragmented or replicated on the various configurations of hardware and software located usually at different geographical sites within an organisation



- Distribution is normally discussed solely in terms of the fragmentation and replication of data. A data fragment constitutes some subset of the original database. A data replicate constitutes some copy of the whole or part of the original database
- Distribution can also be discussed in terms of distribution of functions or processing
- The overall objective of a distributed database system is to look exactly like a centralised database system to the user: location, fragmentation and replication transparency
- Relations or tables are particularly easy to fragment and replicate
- There are three major types of distributed database system: homogeneous distributed database system, heterogeneous database system and federated (multi-database) database system
- A homogeneous distributed database system is one in which data is distributed across two or more systems, each system running the same DBMS
- A heterogeneous distributed database system is one in which the hardware and software configuration is diverse
- A federated database system is made up of a number of relatively independent, autonomous databases
- A distributed database design will determine any requirements for distributing the data or processing in the database application
- Some issues arising in DDBMS: the system catalog of a distributed database has to be more complex; concurrency problems are multiplied in distributed systems; a query optimiser in a true distributed system should be able to utilise topological information in deciding how best to satisfy a given query; to ensure a robust system the distributed DBMS should not be located solely at one site; and software as well as data need to be distributed

37.12**ACTIVITIES**

1. Assume that a university runs over a number of campuses, some outside the UK. Determine a suitable distribution strategy for the academic database discussed in the Case Study of Chapter 7.
2. XML is being proposed as a vehicle for federated database systems. Investigate.
3. Compile a list of the disadvantages of distributing data.

37.13**REFERENCES**

Ceri, S. and G. Peligatti (1984). *Distributed Database Principles and Systems*. New York, McGraw-Hill.

- Date, C.J. (1987). Twelve rules for a distributed database. *Computerworld* 2–5.
- Özsu, M.T. and P. Valduriez (1991). *Principles of Distributed Database Systems*. Englewood Cliffs, NJ, Prentice-Hall.
- Özsu, M.T. and P. Valduriez (1992). Distributed database systems: where are we now? *Database Programming and Design* 42(1).

CHAPTER 38 →

CHAPTER

38



PARALLEL DATABASES

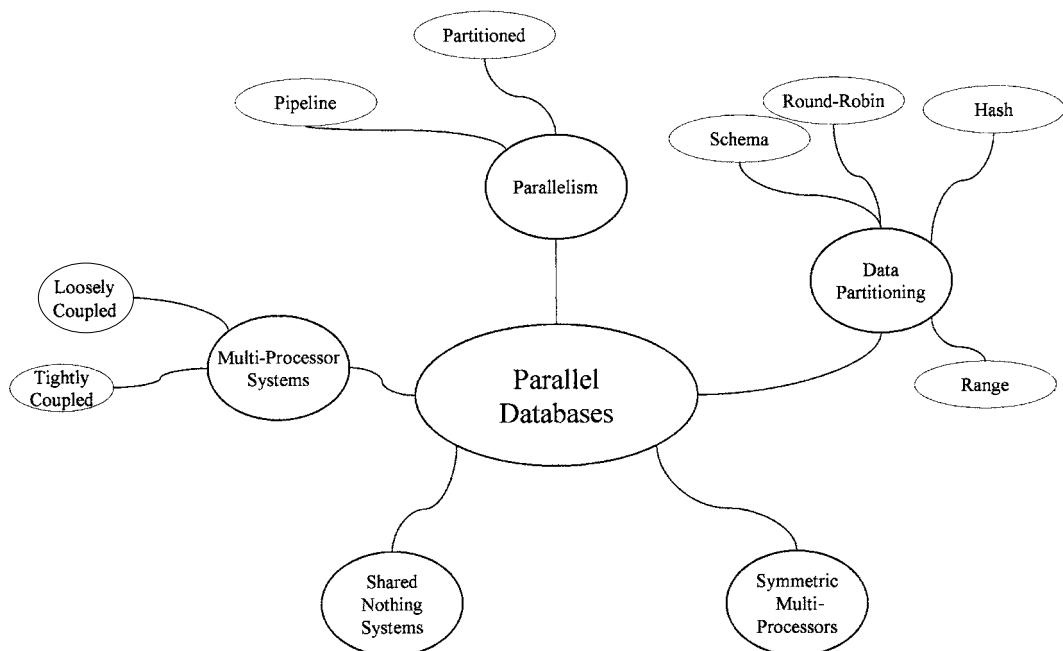
I am into parallel monogamy.

Seen on a button

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Discuss the distinction between conventional computer architectures and parallel computer architectures
- Describe a number of alternative parallel computer architectures
- Indicate the importance of parallel computers to database systems



38.1 HISTORICAL BACKGROUND

A computer can conveniently be considered to be made up of three parts, excluding any man-machine interfaces such as keyboard, screen, printers etc. These three parts are the central processor, the main memory and the backing store. A simple representation of such a system is shown in Figure 38.1.

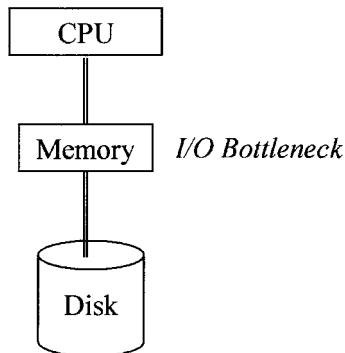


Figure 38.1 Basic computer architecture.

In the forty or so years since the first commercially produced computers became available the central processor has increased steadily in performance and since the development of very-large-scale integration (VLSI) technology it has also decreased dramatically in both size and cost. The first processors could obey instructions at the rate of approximately one every millisecond while the fastest of today's computers can attain a rate that is a million times greater than this.

The main memory and backing store have also developed in a dramatic fashion, both in speed and capacity. However, of more importance is the change in the capability of the processor to address these two levels of memory and the speed at which it can move data between them.

Early computers addressed words in the main store, and the number of bits in the address of the instruction generally reflected the maximum number of words that could be sensibly built or afforded. In the first computer in 1951 there were only 10 address bits, permitting a maximum of 1024 words. The number of address bits had steadily increased to about 20 by the 1960s as memory technology advanced. The backing store was addressable as blocks of either fixed or variable size and the computer obeyed instructions which permitted these blocks to be transferred between the two levels of the store. The need for a backing store is self-evident, with the limited size of the main store, but the major difference, by a factor of 1000, is the access times of the two levels.

In 1960 the concept of virtual memory was developed in which the main store and some or all of the backing store were each divided into fixed-sized blocks called pages. The address length permitted the addressing of at least as



many locations as were physically available, and pages were automatically moved on demand between the two levels of the store. Although this development greatly simplified the task of the programmer, it did not remove the long delays that occurred when the data required had to be moved from the backing store to the main store.

Today, almost all computers use virtual addressing, with the addressing of bytes rather than words and an address length typically of 64 to 128 bits. As an address length of 32 bits permits a program to have up to four gigabytes of addressable data, it might be thought that this would be more than sufficient; however this is not the case and, as will be discussed later, the development of multi-computer systems has led to the implementation of 64-bit address computers.

38.2

THE SPECIAL PROBLEMS OF COMMERCIAL DATA PROCESSING

Early computers were developed for scientific or engineering applications and it has been these areas which have driven the demand for ever more powerful computers. The large disparity between access time to the main store and the disk backing store has always been a problem and is generally referred to as the input/output (I/O) bottleneck. While the difference in access times was a difficulty for scientific computation, it was not critical, as there is generally a large amount of computation performed on the data transferred to the main store before it has to be replaced by more data from backing store. This is not the case in commercial data processing where generally the situation is totally reversed, with relatively little computation performed on data brought from backing store but with large amounts of data having to be transferred. Commercial data processing did not, therefore, benefit to anything like the same extent as scientific computing from the rapid increase in computing power. In fact, as the decrease in disk access times was only a fraction of the increase in computing power, the effect of the I/O bottleneck steadily became more severe.

In an attempt to overcome this difficulty, there has been considerable research into the development of high-capacity data stores with low access times to replace the conventional moving head disk. This research resulted in magnetic bubble memories, charge coupled device memories and head per track disks. None of these developments proved to be commercially viable as a replacement to the standard disk and it was finally accepted by the mid-1980s that special-purpose database computers were not the way forward for data processing.

38.3

TODAY'S COMPUTERS

Developments in computing technology have led to high-performance PCs and workstations with computing power in excess of most of the previous

generation of mainframes. Even though the I/O gap on these systems is relatively worse than that on the mainframes that they have replaced, the enormous improvement in the price/performance ratio means that these systems have been widely used for database applications and have resulted in a large increase in the use of database software by individuals and small-to-medium organisations. They are not, however, capable of supporting the number of users or the massive amounts of data that large organisations require, and until recently such organisations have been restricted to using larger and larger mainframes.

In the mid-1980s relational database systems were just beginning to appear in the data processing marketplace; today they dominate. The CPU and I/O demands of relational database management systems serving large numbers of simultaneous users or searching terabyte databases have made it difficult, if not impossible, for the manufacturers of conventional mainframes to produce systems capable of satisfying the needs of these users.

Relational databases are ideally suited to client-server configurations (DeWitt and Grey, 1992). A typical computer configuration could consist of PCs or small workstations as clients communicating via a high bandwidth local area network to a server system which supports the database. While this approach overcomes the computational requirements, it does nothing to solve the I/O bottleneck, which is still present in the server.

38.4

MULTI-PROCESSOR SYSTEMS (PARALLEL COMPUTING)

The technology which has resulted in the widespread availability of PCs and workstations has also made it economically sensible to build parallel computers of the type classified as multiple instruction multiple data, or MIMD, systems. Two types of MIMD systems can be identified: tightly coupled and loosely coupled. These are illustrated in Figure 38.2.

Tightly coupled systems are also called shared memory systems and, more recently, symmetric multi-processors (SMP). In these systems each processor has to be capable of addressing the whole of the shared memory, the maximum size of which is therefore limited by the number of address bits in the CPU. As CPUs typically have 64 address bits this limitation has not, until recently, inhibited the design and production of shared memory systems.

Loosely coupled systems are arrays of nodes where each node is a computer capable of running a program to process the data from its own store. The array also possesses an interconnect network which permits the sending or receiving of data to or from other nodes in the array.

In both configurations there are a large number of I/O channels connected to the shared memory or the distributed memories, each channel capable of supporting several small computer system interface (SCSI) disks. It is this aspect of the design that provides the potential to overcome the bottleneck.

Parallel computers provide a path to the required levels of performance for commercial applications, and an important reason for this is the widespread

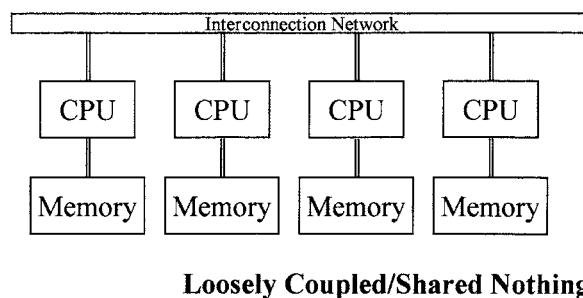
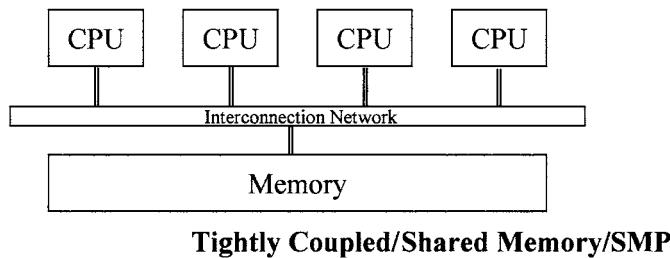
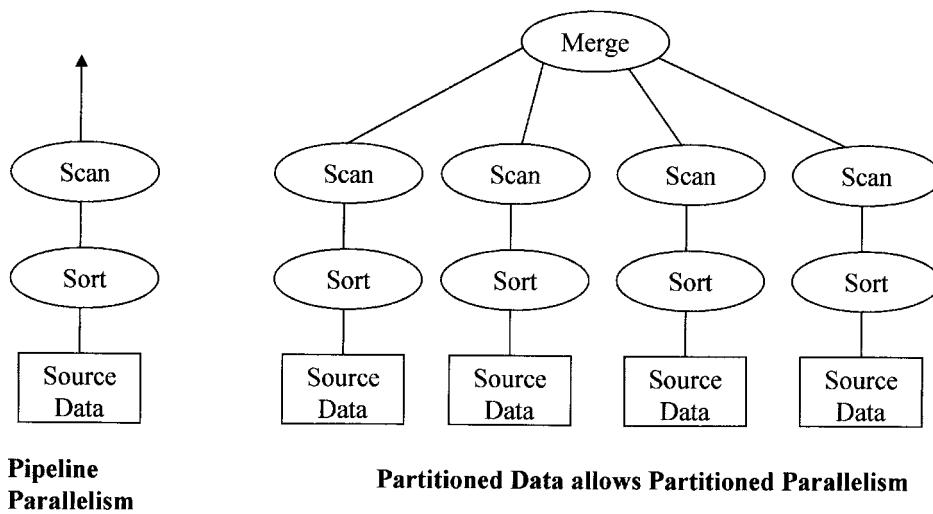


Figure 38.2 Tightly and loosely coupled systems.

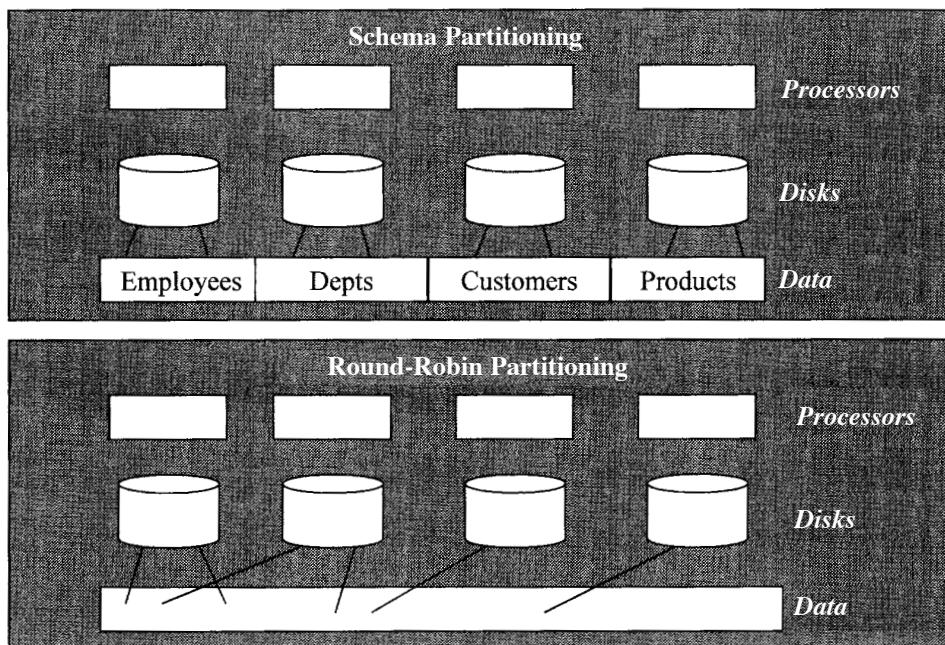
adoption of relational databases. As stated by DeWitt and Grey (1992), relational queries are ideally suited to parallel execution, as they consist of uniform operations applied to uniform streams of data. Each operator produces a new relation, so the operators can be composed into highly parallel data flow graphs. By streaming the output of one operator into the input of another operator, the two operators can work in series, giving pipeline parallelism. By partitioning the input data among multiple processors and memories, an operator can often split into many independent operators, each working on part of the data. This partitioned data and execution give partitioned parallelism (Figure 38.3).

Experience over the past few years strongly suggests that parallel computing is the only way forward for relational databases serving large numbers of simultaneous users and/or searching terabyte databases.

If the multiple disks are to remove the I/O bottleneck, then the data will have to be partitioned across the disks to permit the concurrent transfer of multiple data streams to the multiple processors and thus permit parallel computing. Four data partitioning schemes have been identified by Ferguson (1993), as shown in Figures 38.4 and 38.5. The first, schema partitioning, makes a simple division of data. In Figure 38.4, the schema approach shows four typical areas: Employees, Departments, Customers and Products. Another simple approach is to implement tuples along the fragments in a round-robin fashion (Figure 38.4). This is the partitioned version of the classic entry sequence file. Round-robin partitioning is excellent if all the partitions want to access the relation by sequentially scanning all of it on each query. The

**Figure 38.3** Pipeline and partitioned parallelism.

problem with round-robin partitioning is that applications frequently want to associatively access tuples, meaning that the application wants to find all the tuples having a particular attribute value.

**Figure 38.4** Schema and round-robin partitioning.



The remaining two approaches in Figure 38.5 are the ones most frequently used in modern systems. Hash partitioning is ideally suited for applications that want only sequential and associative access to data. Tuples are placed by applying a hashing function to an attribute in each tuple. The function specifies the placement of the tuple on a particular disk. Associative access to the tuples with the specific attribute values can be directed to a single disk, avoiding the overhead of starting queries on multiple disks. The final method, range partitioning, clusters tuples with similar attributes together in the same partition. It is suitable for sequential and associative access and also for clustering data. Range partitioning can be based on lexicographic order, but any other clustering algorithm is possible. The problem with range partitioning is that it risks data skew, where the data is placed in one partition, and execution skew, where all the execution occurs in one partition. Hashing and round-robin are less susceptible to these skew problems. Range partitioning can minimise skew by using non-uniformly partitioning criteria.

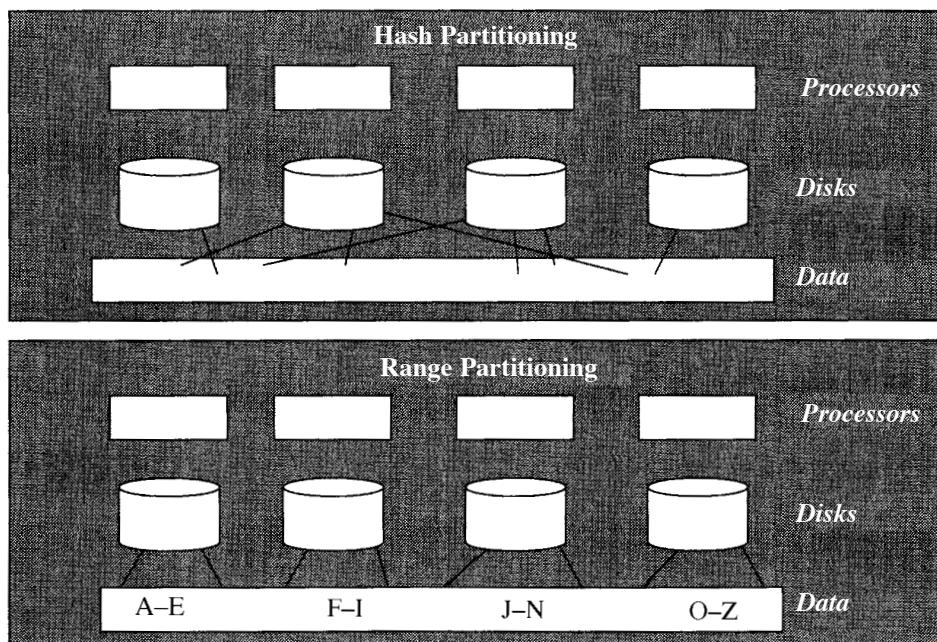


Figure 38.5 Hash and range partitioning.

38.5 SYMMETRIC MULTI-PROCESSORS (SMP)

These have many processors attached to a shared memory which is in turn attached via multiple channels to a large number of disks (Figure 38.6). As in any multi-processor system, the computing power can be increased by the addition of more processors. In an SMP this increases the load on the interconnection bus to the shared memory, and the capacity of this bus is the limitation on

the maximum number of processors in the system. By putting large cache memories between each processor and the interconnect bus, the number of accesses to main memory can be significantly reduced and it is this reduction which has made larger SMP systems viable.

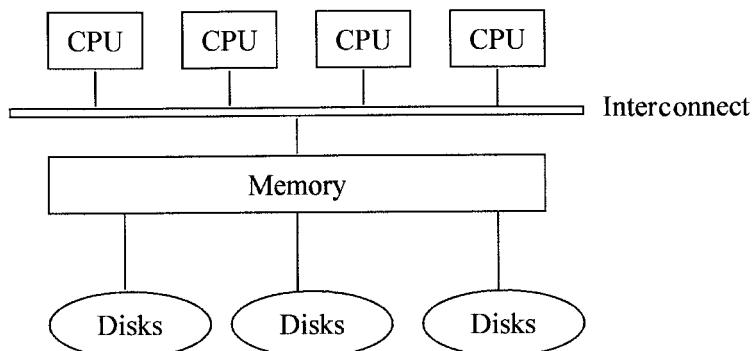


Figure 38.6 Shared memory system.

Shared memory is a considerable advantage from a programming point of view and, provided that the application is not one which results in low hit rates in the cache memories, SMP systems can be very effective. The most suitable applications are generally accepted to be:

- Parallel transactions where many SQL queries are each running on different processors at the same time
- Departmentalised, easily partitioned applications that access by key
- Random access applications that have minimal contention for the same blocks
- Applications that update disjoint data or update the same data at different times

Applications for which they are not particularly suited are:

- High-volume update from different instances affecting the same blocks
- *Ad-hoc* or decision support applications requiring full table scans

They are considered to work best for on-line transaction processing workloads, although some of the most recent SMP systems have very large capacities and considerable power and claim to be appropriate platforms for both data warehousing (Chapter 40) and data mining (Chapter 42).

The present and anticipated bandwidth of the CPU/memory connect bus will permit SMP architectures to scale to larger numbers of CPUs. The I/O bottleneck has been more or less completely removed by the use of many high bandwidth channels to hundreds of disks.

38.6 ◎ SHARED NOTHING SYSTEMS

The most widely used version of this type of system has been produced by Teradata since 1978, based on commodity microprocessors, disks and memories. Teradata systems act as SQL servers to client programs operating on conventional computers. Teradata systems may have over 1,000 processors and many thousands of disks. Many systems have been installed with over 100 processors and hundreds of disks. These systems demonstrate near linear speed-up and scale-up on relational queries. They far exceed the speed of traditional mainframes in their ability to process terabyte databases.

Example → Figure 38.7 illustrates the design of the NCP 3600 Teradata database management system. In this system, SQL messages from the clients are connected via applications processors to the Y-net which is a dual redundant tree-shaped interconnection. The processors below the Y-net can be either Intel 486 or Intel 586 models.

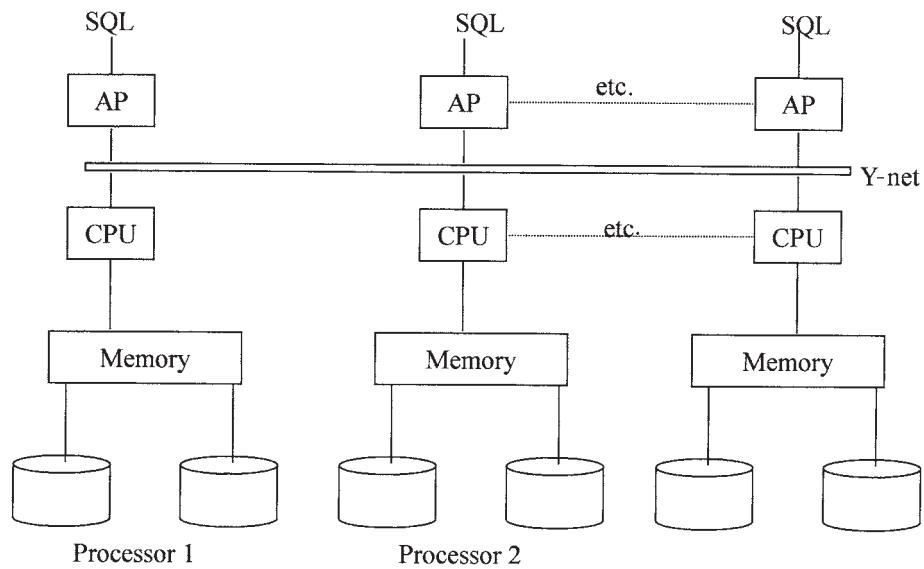


Figure 38.7 Teradata DBMS.

The advantage of the shared nothing architecture is that there is no contention for resources and no interference. This leads to highly linear scale-up to hundreds of processors. Larger tasks can be tackled by a divide and conquer approach. There is little traffic through the interconnect and the design is a perfect match for relational operators. Every operator results in another table and several operators in one query can be done in parallel. The shared nothing architecture is well suited to parallel query.



The principal feature of the Teradata system is the full exploitation of shared nothing architecture by the DBMS optimiser. There are three levels of parallelism in the Teradata DBMS:

- Multi-statement request executed in parallel
- Multiple steps within each SQL statement executed in parallel
- Parallel processing within each step of an SQL statement

The system also ensures automatic even distribution of rows across all disks, with the following benefits:

- Part of a table on each disk
- Each part can be worked on in parallel
- No reorganisation and high availability, which is critical to managing large data volumes

This organisation ensures efficient use of the hardware as each processor joins, sorts, scans projects, updates, deletes, inserts etc. at the same time as all the other processors.

Shared nothing architectures are most suitable for the following:

- Parallel queries – complex SQL queries are broken up into smaller steps and each run on different processors in parallel
- Information warehouse applications scan millions or billions of detailed data rows
- *Ad-hoc* GUI generated SQL and DSS applications
- Complex analytical queries and applications that are time-series oriented
- High-volume set level updates across whole tables
- Applications with very large data volumes
- Applications that require linear scalability and speed-up as the database grows

38.7 INTERCONNECTED ARRAYS

In the development of parallel computers for scientific applications there has been a concentration on interconnected arrays of microcomputers with this array connected via a large number of channels to SCSI disks. Some of these systems have been applied to commercial data processing including the Meiko Computing surface.

This system can have a large number of nodes. In the first version, each node was a T800 transputer with four megabytes of memory. Each node is connected to the four adjacent (north, south, east and west) nodes in the array. When



data needs to be transferred, this is done by means of messages which go between the two nodes concerned via the shortest route through the network. The process of receiving and passing on a message at any intermediate node does not affect the computational activity of that node, as the message-passing is performed by the message-passing hardware associated with that node. This system has been used for several database applications with a high level of success. In a more recent version, the transputers have been replaced by Sun processors with 64 megabytes of memory.

38.8**GLOBALLY DISTRIBUTED SHARED VIRTUAL MEMORY**

A variation of the array of computers approach with message-passing is an array of computers with data movement between nodes achieved by a variation of demand paging. The whole system has a single virtual address space and, when an address is requested by a processor, it first checks to see whether it is currently resident in a page of its own memory. If the address is not in its own memory then it is sent to the other nodes in turn, and when the location of the address is found the page containing it is transferred to the node which generated the request. Obviously, the success of such a design is determined by the frequency with which the address requests result in page movements from the remote nodes. Too many of these page movements is the equivalent of 'page thrashing' in a conventional computer, caused when the fast memory is too small to hold the current working set. The effect is also the same; the system spends most of its time waiting for pages to be moved and does very little effective computation.

The principle of locality which made conventional virtual memory systems efficient also applies in this case and the pages of data migrate to the store of the node using them. The traffic on the interconnect highway is low and these systems have linear scale-up comparable with that of other shared nothing architectures.

Example

Two systems using this approach have been produced commercially. The KSR1 made by Kendall Square Research of Boston was designed to have up to 1024 nodes, although the biggest system ever produced only had 256. As each node has 32 megabytes of memory, the total real store of the system was 8 gigabytes. The system therefore needs more than 32 address bits and the early versions used 48 bits of the ultimate maximum of 64. A more recent system has been produced by Convex. This uses Hewlett Packard PA RISC chips, which have a 64-bit address, as the nodes.

The KSR system was highly successful in a range of commercial data processing activities. A specific advance was the development of a technique of query decomposition to be used with ORACLE to speed up the processing of complex queries to large databases.

The query decomposer works in conjunction with the underlying ORACLE RDBMS to greatly speed the execution of decision support queries. Such queries are often expensive to execute, requiring minutes or even hours of processing time to sweep through relatively large portions of the database. The query decomposer leverages the parallelism of the KSR1 computer to reduce query execution time to seconds.

In order to maximise parallel reads from disk for decision support queries, all large database tables are partitioned physically over multiple disks by the database administrator, using ORACLE DDL commands. There may be tens or even hundreds of partitions for a given table, which can be populated randomly or via a hash function that scatters tuples into small clusters. These partitioning techniques were originally developed to spread out on-line transaction processing (OLTP) 'hotspots', or frequently accessed portions of tables, but they are well suited to supporting the query decomposer.

When an application tool or user first submits a query to ORACLE, it is intercepted by the query decomposer at a common processing point, as shown in Figure 38.8. Queries that will not benefit from decomposition are simply passed through to ORACLE unchanged; those that will are transformed. Based primarily on the data access strategy selected by the ORACLE query optimiser, the query decomposer generates a number of subqueries to match the underlying physical data partitions of one of the partitioned tables, called the 'driving' table. Decisions made automatically include the number of subqueries, the choice of the driving table, the minor query transformations to handle aggregate functions and the method of combining query results. Each subquery looks very much like the original query, with minor transformations that include an additional condition to restrict it to just one partition.

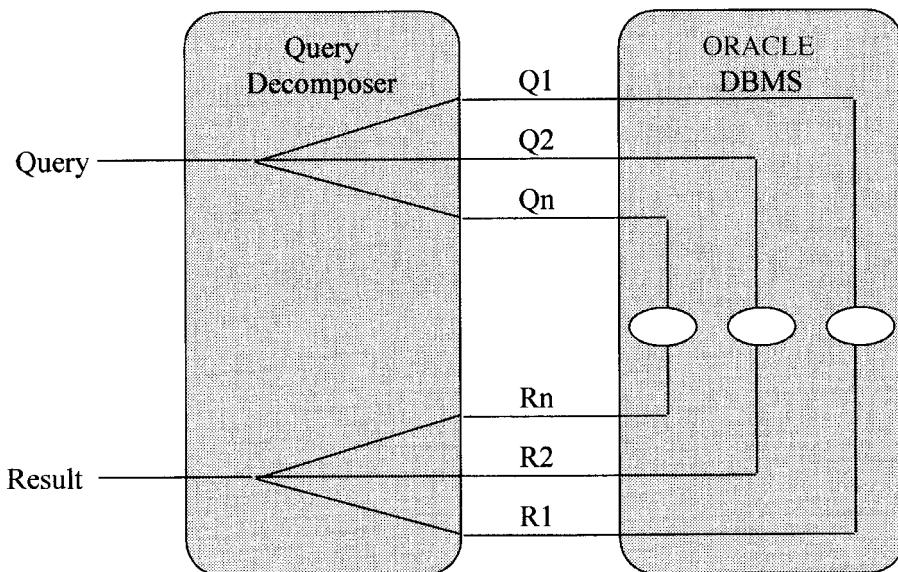


Figure 38.8 Query decomposition.



The subqueries are submitted in parallel to ORACLE over multiple, coordinated connections established by the query decomposer so that they see a consistent view of the database. Each subquery is executed either by a dedicated or shared server process. All of the subqueries are executed in parallel, each accessing its own part of the driving table, and doing further joins as required. Since partitions are approximately equal in size, initial partition scans are unaffected by data skew. Common lookup tables and index pages are only brought in once from disk, and are shared among subsequent subqueries when needed.

Subquery results are combined inside the query decomposer, and returned to the application, tool or user which submitted the original query. If the original query called for grouped or sorted data, so will the subqueries, and the query decomposer will correctly combine their results. This effectively gives the user a parallel sorting capability for decision support queries. Aggregate functions such as maximum, count and average are also correctly computed when subquery results are combined.

From the viewpoint of an application developer or end-user, the query decomposition process is transparent, other than the substantial speed-up in performance. The query decomposer is applied automatically to appropriate queries (although it can be turned off if desired). Queries do not need to be modified, and existing ORACLE applications do not need to be rewritten to use this powerful facility.

38.9



SUMMARY

- Conventionally a computer can be said to be made up of three parts: CPU, memory and backing store. Improvements in the performance of each of these elements have increased computational power and storage capacity
- Business computing, as compared to scientific computing, is more subject to the I/O bottleneck. This is particularly critical in the area of querying and maintaining terabyte databases. Parallel computing is one way of addressing this bottleneck
- PC technology makes it possible to build reasonably priced parallel machines of the multiple instruction multiple data (MIMD) type. Tightly-coupled, shared memory or symmetric multi-processing (SMP) machines are the most commonplace. Here, a number of separate processors share a memory (normally 2–30 CPUs)
- Relational queries are ideally suited to parallel execution
- To facilitate parallel processing, data must be partitioned across disks. Four main techniques exist: schema, round-robin, hash and range partitioning
- Shared memory systems tend to suit applications in which large-volume, concentrated querying is occurring. They tend not to be effective for mass, concentrated update from multiple users
- Closely-coupled or shared nothing systems (sometimes referred to as massively parallel processors or MPP) are less commonplace. Here, each processor has its

own memory. Processors communicate via message-passing. These systems are particularly good at handling mass update and mass querying of data, as in data warehousing applications running against terabyte databases.

- Most current, large-scale DBMS have versions for SMP architectures; some have been adapted to MPP. Query decomposition becomes an issue. A number of DBMS will run against shared nothing machines in the background.
- Many DBMS are looking to parallel processing to support large multi-media servers for the WWW.

38.10 ACTIVITIES

1. In terms of a DBMS known to you, determine which approach to parallelism is taken, if any.
2. Determine the types of commercial database activities most suitable for parallel execution.

38.11 REFERENCES

- DeWitt, D. and J. Grey (1992). Parallel database systems: the future of high performance database systems. *CACM* **35**(6).
- Ferguson, M. (1993). Parallel query and transaction processing with relational databases. *UKOUG Conference*, Bournemouth, UK.

CHAPTER 39 →

COMPLEX DATA

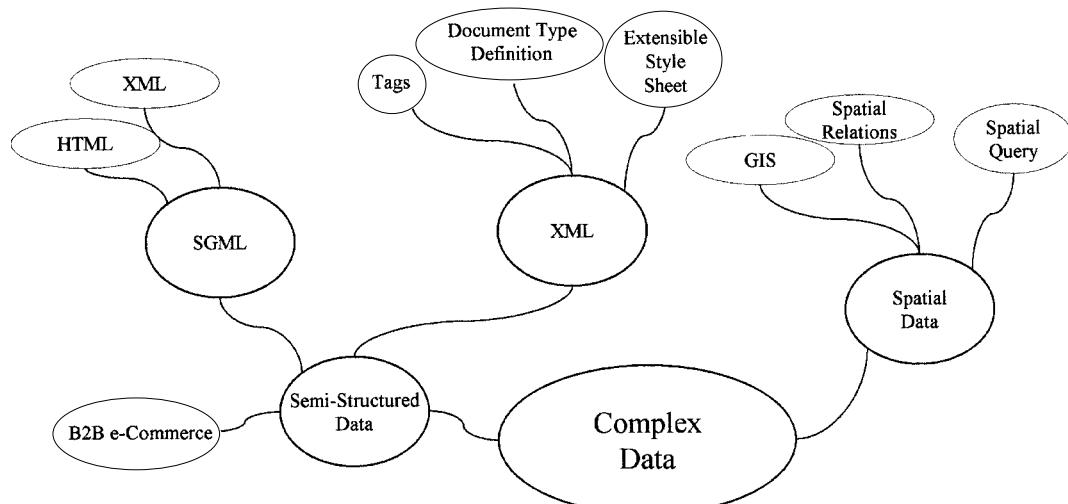
It is a very sad thing that nowadays there is so little useless information.

Oscar Wilde (1854–1900)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define what is meant by complex data
- Describe the concept of semi-structured data and the significance of XML
- Describe the concept of spatial data and the importance of database systems to GIS



39.1 INTRODUCTION

In previous chapters we have discussed how modern database systems have been extremely effective at providing efficient solutions for handling standard commercial data. By standard commercial data we mean data subject to a limited range of data types such as integer, character etc. Such data is said to be structured in the sense that a defined format embedded in the data type can be assigned to data-items and data-elements. However, as the realms of information systems impinge on more and more areas of human activity, so the idea of applying database technology to the problems of handling more complex data has become significant. Such complex data types include:

- *Images* – graphics and photographic images
- *Audio* – various forms of sound data
- *Video* – various forms of moving image

Information systems are now also required to handle complex data structures as well as complex data-items or elements. In this chapter we consider two examples of such complex data structures: spatial data and semi-structured data.

Organisations want to be able to define the structure of documentation that they use for communication. A document such as an invoice or a contract is a complex data structure in that it may be made up of text, numeric data and images. Also different organisations may use different formats for such documents.

The most significant challenge for business has been to define standards for the storage and transmission of electronic documentation such as invoices. Hence databases need to adapt to handle this semi-structured data. There is pressure on organisations in the same industrial sector to develop standards for such documentation. These standards are being specified using a formal language known as XML – extensible markup language.

A type of information system known as geographical information systems (GIS) is now a significant part of the infrastructure of many public-sector and private-sector organisations. GIS need to store and manipulate spatial data, and databases have been used as significant technologies in this respect.

39.2 SEMI-STRUCTURED DATA

Semi-structured data lies between the poles of structured and unstructured data. In structured data the structure is provided by some schema. Unstructured data lacks any schema and the structure is not apparent from any repetition of instances of such data. Semi-structured data also lacks a schema but structure is apparent from the way in which data is presented. Hence, the meaning of any data in a set of such data is determined by its position.

Semi-structured data is important because of the following reasons:



- Web sources tend to be composed of semi-structured data
- On many occasions it may be appropriate to treat such sources as databases
- Much documentation is semi-structured

39.3 ELECTRONIC DOCUMENTATION

In any regular trading relationship, four main packages of information flow between buyer and seller (Figure 39.1):

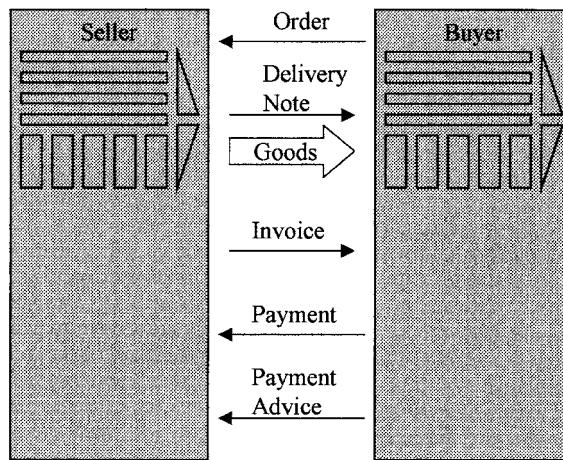


Figure 39.1 Major transactions between buyer and seller.

- The buyer sends an order to the seller
- The seller sends the goods and a delivery note at some later time to the buyer
- The seller follows up the delivery note with an invoice which is sent to the buyer
- The buyer makes payment against the invoice to the seller and sends a payment advice to the seller

Order, delivery note, invoice, payment and payment advice are all transactions flowing between buyer and seller. Traditionally, such transactions would be structured as paper documents that would be sent between companies. In more recent years, such transactions are likely to be electronic, involving data flow between companies.

Example In a supermarket chain an example of an order would be for a quantity of potatoes from some agricultural producer. The delivery note would detail the actual number of sacks of potatoes delivered to the distribution warehouse of the supermarket chain.

39.4 SGML

Electronic documents (such as invoices and delivery notes) are made up of two forms of data: data which represents content, and data which describes to applications how the document is to be presented on such media as the printed page and the PC screen-structure. In terms of electronic documents, content normally comprises text and graphics. Structure is represented by a set of embedded tags that indicate how the content is to be presented. This process of tagging text with extra structural information is known as marking-up and the set of tags for doing this a markup language. In the 1960s work began on developing a generalised markup language for describing the formatting of electronic documents. This work became established in a standard known as the standard generalised markup language or SGML.

SGML in fact constitutes a meta-language – a language for defining other languages. Hence, SGML can be used to define a large set of markup languages. Tim Berners-Lee used SGML to define a specific language for hypertext documents known as hypertext markup language (HTML). HTML (Chapter 43) is a standard for marking-up or tagging documents that can be published on the Web, and can be made up of text, graphics, images, audio-clips and video-clips. Documents also include links to other documents either stored on the local HTML server or on remote HTML servers. HTML documents are also referred to as ‘pages’.

Example Below we include a very simple document expressed in HTML

```
<HTML>
<TITLE>Database Systems</TITLE>
<H1>Database Systems</H1>
<H2>Paul Beynon-Davies</H2>
</HTML>
```

The text between angled brackets constitutes tags. Each piece of text is preceded by a start-tag and succeeded by an end-tag. A forward slash precedes an end-tag. The *HTML* tags indicate the start and end of the document. The tag *Title* provides a name for the page. The tags *H1* and *H2* indicate that a first-level and second-level heading should be displayed respectively.

39.5 XML

One of the main advantages of HTML is its simplicity. This enables it to be used effectively by a wide user community. However, this simplicity is also one of its disadvantages. Sophisticated users want to define their own tags, particularly for functionality involved with the exchange of data. Extensible markup language or XML (W3C, 2000) was developed by the World-Wide-Web Consortium in 1998. The key feature of XML is that it is extensible. This means that new markup tags can be created by users for the exchange of data.

Like HTML, XML is another restricted descendant of SGML. Whereas HTML is used to define how the data in a document is to be displayed, XML can be used to define the structure of a document. XML allows system developers to define a common format for standard business documents such as invoices and receipts. Systems can then be developed to enable the transmitting and receiving of standard electronic documents.

39.5.1 ADVANTAGES OF XML

XML is important for the following reasons:

- XML is reasonably simple; the definition of the language comprises fewer than 50 pages
- XML is not limited to text markup; its extensibility means that it can be applied to marking-up other complex data such as sound, images and video
- Since XML describes the structure of data, it could potentially be used to define the schema of diverse databases. Hence, it could become a useful mechanism for defining the structure of data in heterogeneous database systems (Chapter 37)
- XML is platform and vendor-independent; this makes it easier to develop interoperable systems using the standard
- XML encourages reuse; the extensible nature of XML means that libraries of definitions can be constructed and reused by other applications
- XML makes data self-describing; this enables more efficient processing of data by front-end and back-end ICT systems

39.5.2 OVERVIEW OF XML

An XML consists of a set of elements and attributes.

Elements or tags are the most common form of markup. The first element in an XML document must be a root element. The document must have only one root element but this element may contain a number of other elements.



Example ➔ Suppose your company is a coffee wholesaler. You might wish to create XML documents for the exchange of shipping information to your customers. An appropriate root element might therefore be the tag <PRODUCTDETAILSLIST>.

An element begins with a start-tag and ends with an end-tag.

Example ➔ The start-tag in our document for the root element would be <ProductDetailsList>. The corresponding end-tag would be </ ProductDetailsList>. Note that tags are case-sensitive in XML. Hence <PRODUCTDETAILSLIST> is a different tag from <ProductDetailsList>.

Elements can be empty, in which case they can be abbreviated to <EmptyElement/>.

Elements must be properly nested as subelements within a superior element.

Example ➔ The following XML element might be used to define a particular coffee product:

```
<ProductDetails ID='1234'>
    <ItemName>Kenya Special</ItemName>
    <CountryOfOrigin>Kenya Special</CountryOfOrigin>
    <WholeSaleCost>20.00</WholeSaleCost>
    <Stock>4000</Stock>
</ProductDetails>
```

Here we have a ProductDetails element with a number of subelements. Subelements such as ItemName, CountryOf Origin, WholeSaleCost and Stock are properly nested within ProductDetails.

In traditional database terms this would constitute a record in a products file. This record is made up of a number of data-items including an identifier for the product, the name of the item, the country of origin of the product, the cost of the product and the number of product items in stock.

Attributes are name–value pairs that contain descriptive information about an element. The attribute is placed inside the start-tag for the element and consists of an attribute name, an equality '=' sign and the value for the attribute placed within quotes.

Example ➔ In our coffee producer example the tag <ProductDetails ID= '1234'> contains the attribute ID and the value '1234'.



Within XML the ordering of elements is significant. However, the ordering of attributes is not significant.

Example → The two orders for the product information below would be regarded as different elements:

```
<ProductDetails ID='1234'>
  <ItemName>Kenya Special</ItemName>
  <CountryOfOrigin>Kenya Special</CountryOfOrigin>
  <WholeSaleCost>20.00</WholeSaleCost>
  <Stock>4000</Stock>
</ProductDetails>

<ProductDetails ID='1234'>
  <ItemName>Kenya Special</ItemName>
  <WholeSaleCost>20.00</WholeSaleCost>
  <Stock>4000</Stock>
  <CountryOfOrigin>Kenya Special</CountryOfOrigin>
</ProductDetails>
```

However, we might have represented this information as attributes of the element Product:

```
<Product ID="1234" ItemName="Kenya Special" CountryOfOrigin="Kenya" Stock ="400"
WholeSaleCost="20.00"/>
```

In this case the following element is regarded as being identical:

```
<Product ID="1234" ItemName="Kenya Special" WholeSaleCost="20.00" Stock ="400"
CountryOfOrigin="Kenya"/>
```

Two other mechanisms are important for an XML document:

- A document type definition (DTD). This defines the valid syntax for an XML document. It lists the names of all elements, which elements can appear in combination and what attributes are available for each type of element
- An extensible style sheet (XSL). A stylesheet is a definition which is used by a browser for the rendering of a document. An XSL allows the developer to specify the appropriate rendering for a given XML document



A geographical information system (GIS) is a special case of an information system. It is special in the sense that the key feature which distinguishes a GIS is its focus on spatial data and the modelling and analysis of such data.

Examples ➔ Consider a large supermarket chain. It maintains a decision support system (DSS) which is used to decide on the siting of new stores. Since the chain opens one new store every two weeks, this is a significant activity for the company. The DSS has two component parts: a digitised version of a small-scale map of the British Isles (spatial data); a database of demographic and customer data (attribute data). The DSS integrates the attribute data with the spatial data. Planners can then run 'what-if?' analyses against the GIS. A typical what-if? analysis might be: *if we site the store at this point on the map, how many people of these characteristics will be in commuting distance of the store?*

A number of the UK utility companies (gas, electricity, water and telecommunications) also use GIS. An electricity company, for instance, might use a GIS to store information about the physical location of its overground cables, power stations and substations. Associated with this spatial data might be some service history. Hence the GIS can be used to proactively produce a maintenance schedule for an electricity grid.

Finally, local government is beginning to exploit GIS technology in areas such as land registry. Here, the problem is to maintain precise data on which person owns what parcel of land. Also the system must be able to precisely record the exact boundaries of land-parcels for activities such as conveyancing.

39.7 SPATIAL DATA AND QUERIES

The concept of space might be defined as a set of objects to which may be attached associated attributes, together with a relation, or relations, defined on that set. A brief, and incomplete, typology of spatial objects might be:

- *Points* – objects represented by a single pair of locational coordinates
- *Lines* – sets of ordered or connected points
- *Areas* (also called zones or polygons) – collections of line segments that close to form discrete units

In terms of relations, the most common type of relation stored in a GIS is some form of metric spatial relation. A classic example of a metric spatial relation is the Euclidean distance between two points. Another example is the line-area relation, such as intersects. This might be of relevance to an application concerned with determining which areas a given rail route carrying hazardous cargoes should go through.

The ideas of spatial representation and querying spatial data are clearly inter-related.

**Examples** ➔ Examples of spatial queries include:

- What do we have at this point in a polygon?
- What do we have in this region?
- What do we have at this vacant place delimited by this polygon?
- What do we have within this distance from this object?
- What is the shortest path between these two points?

Clearly many of these types of query will need to integrate attribute with spatial information to provide results to the user of the GIS. Hence, a query such as *list me all the landowners whose subsoil has ore* will need to connect issues of landownership to the mineral characteristics of land-parcels.

The queries in the example above are all descriptive GIS queries. These are all questions of the *what?* and *where?* type. There is however another important type of GIS query – the analytical query. Here the user asks questions such as *why?* or *what-if?* For example, *why does this part of the river flood at periodic intervals?* These are questions which demand of a GIS the capability for spatial analysis.

Because of the complexity of the underlying data, standard query languages such as SQL do not have sufficient functionality to support many GIS operations. A number of systems have therefore extended the query language to incorporate spatial operators.



There are two contemporary approaches for implementing GIS in database technology: hybrid and integrated approaches.

In the hybrid approach, spatial information is stored separately from attribute information. Usually, digital cartographic data is stored in a set of direct access operating system files while attribute data is stored in a standard commercial RDBMS. GIS software manages linkages between the cartographic files and the DBMS data. The advantage of this approach is that spatial data can be stored in a form optimised for standard spatial analysis tasks. An example of this approach is the ARC/INFO system. In this system the spatial model is implemented in the ARC program and the attribute data model is implemented in INFO – a DBMS. Users of the system can access each component separately by means of the ARC macrolanguage and INFO commands respectively.

In the integrated approach, both map data and attribute data are stored in a database running under a DBMS. The GIS software then acts primarily as a query processor sitting on top of the database. One of the main problems of this approach is that although spatial data can, for instance, be represented in

a relational database, the standard relational operations are not highly effective for spatial analysis tasks. However, a number of advantages arise out of effective exploitation of standard DBMS functions, such as concurrency control, data security and data integrity. Some existing commercial RDBMS have begun to address the issue of handling non-standard data by introducing the idea of a binary large object or BLOB. BLOBs consist of large-volume data-items (images, audio, text etc.) that do not fit neatly into the standard RDBMS framework. Usually, BLOBs are stored separately from the conventional database but can be handled via the standard DBMS functions such as insert, update and selection. Querying on BLOB data on the basis of content is however usually not allowed using conventional mechanisms such as SQL.

39.9 OBJECT-ORIENTED DATABASES AND GIS

Because of some of the inherent problems in building contemporary integrated approaches to GIS databases, many people have proposed that an object-oriented (OO) data model is better suited to the problems of managing spatial data than classic data models such as the relational data model. This is because in an OO database, process information can be stored alongside attribute information. Potentially, then, spatial processing can be stored alongside spatial data.

Figure 39.2 represents a simple object model for a GIS application using the notation discussed in Chapter 17. Here a landParcel class has been declared as a subclass of the superClass polygon. This means that all the attributes and methods of the polygon class are inherited by the landParcel class. Hence it becomes possible to run a query against this database, which causes the land-parcel to be drawn or the area of the land-parcel to be displayed.

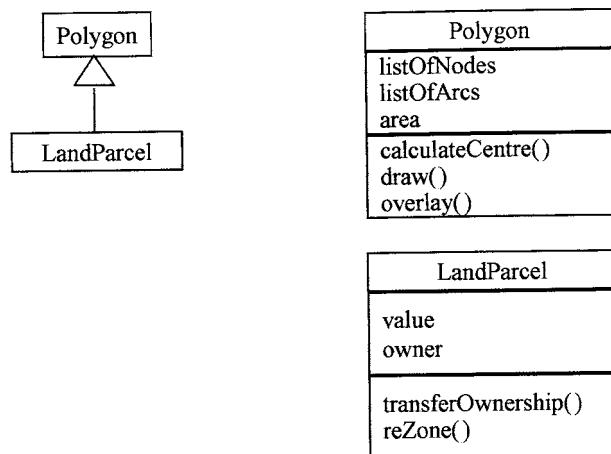


Figure 39.2 An object model for a simple GIS application.

39.10 DEDUCTIVE DATABASES AND GIS

Deductive databases have also been described as having considerable potential for the future development of GIS. Much of the potential value of deductive databases to GIS derives from being able to represent spatial and non-spatial relations explicitly in a database and therefore to facilitate the idea of intelligent processing of GIS information. In this section we briefly illustrate the application of deductive database concepts to a simple GIS application.

Spatial hierarchies appear frequently in GIS applications. An example of a spatial hierarchy is the UK administrative subdivisions of provinces, counties, districts and wards. In terms of the abstraction relations defined in Chapter 17, such hierarchies may be modelled using the *partOf* relation and its converse *aggregationOf*.

Examples  Hence we might declare a spatial schema in the following way:

```
partOf('Mid-Glamorgan', 'Wales').
partOf('Rhondda', 'Mid-Glamorgan').
partOf('Ystrad', 'Rhondda').
ako('Wales', 'Province').
ako('Mid-Glamorgan', 'County').
ako('Ystrad', 'District').
```

The relation *aggregateOf* could then be defined in the following way:

```
aggregateOf(A,B) :- partOf(B,A).
aggregateOf(A,B) :- partOf(C,A),
aggregateOf(C,B).
```

This inherently defines the transitive nature of the aggregation relation and can be implicitly used as a generator for a spatial hierarchy. For instance suppose we associate demographic data such as the number of people unemployed at the lowest level in our hierarchy. Then a simple rule can easily aggregate such statistics together at any level in the hierarchy.

We should note, however, that the *partOf* and *aggregateOf* relations as defined assume a strict spatial hierarchy. In practice, many spatial hierarchies contain overlapping regions. This means that the *partOf* and *aggregateOf* relations are not directly equivalent to their spatial brethren *contains* and *inside*.

39.11 CASE STUDIES

Leeds city council have introduced a spatial information system to simplify access to its street services and information. Council applications such as environmental health and planning are integrated with this system to enable the following:

- Reporting faulty street lamps
- Requesting land searches
- Searching for council access points

Securicor Cash Services have adopted XML technology to improve integration between two key information systems – the Contracts and Administration System and the Operational Management System. Such integration means that crucial changes from customers can be handled more speedily. The integration uses Microsoft's XML-based application integration software BizTalk. This converts data from an ORACLE database into a form acceptable to a Progress database.

39.12 SUMMARY

- Database systems have achieved a prominent place in the development of information systems. Database work is extending ever outwards to attempt to handle more and more complex application areas
- Semi-structured data lies between the poles of structured and unstructured data. In structured data, the structure is provided by some schema. Unstructured data lacks any schema and the structure is not apparent from any repetition of instances of such data
- Electronic documentation is the most significant example of semi-structured data. Databases need to adapt to handle semi-structured data through the development of standards using a formal language known as XML – extensible markup language
- There are two contemporary approaches for implementing GIS in database technology: hybrid and integrated approaches
- Integrated approaches demand a common representation for spatial and attribute relations

39.13 ACTIVITIES

1. Investigate some of the other common tags used in a HTML document.
2. In some trading relationship known to you, identify some common transactions/information flows.
3. Suppose we wanted to store data about the location of the Cyfartha Canal on a map of Mid Glamorgan. How might we do this in a deductive database?
4. Consider in what ways an amalgam of hypermedia, database and GIS technology is desirable.

39.14 REFERENCE

W3C (2000). *XML 1.0 2nd Edition*. World-Wide-Web Consortium.

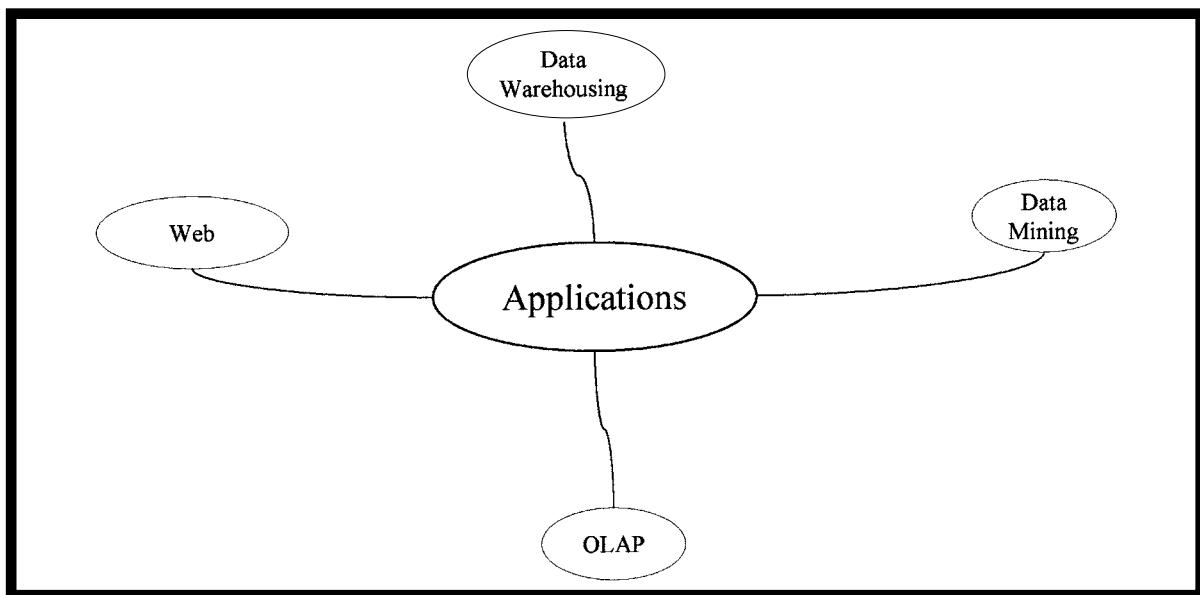
PART

10

APPLICATIONS OF DATABASE SYSTEMS

There are no such things as applied sciences, only applications of science.

Louis Pasteur (1822-95)





As we mentioned in Part 1, databases have now become the common core of most information systems applications within organisations. The traditional uses for databases have been to store operational data relevant to day-to-day organisational processes such as order-entry, stock control, accounting etc. From such operational data a vast range of what we might call management information can be gleaned. In the past, the approach taken to providing management information was to build specialised systems for capturing and presenting this information. More recently, interest has grown in using sophisticated software tools to meld information from a range of diverse systems and to analyse this data to identify patterns. This is where the interrelated technologies of data warehousing, on-line analytical processing (OLAP) and data mining have a part to play. The common thread through each of these technologies is the central place of the database system.

Data warehousing is the process of integrating operational data from diverse data sources together for the support of decision-support applications such as market analysis and financial forecasting. This is the topic of Chapter 40. On-line analytical processing comprises the support of queries which can rapidly produce aggregate data from the large volumes of data typical of data warehousing applications. This is the topic of Chapter 41. Data mining involves the use of automatic algorithms to extract patterns of data. This is the topic of Chapter 42.

The Internet and its associated technologies have begun to dominate the architecture of ICT systems. Organisations now tend to construct their front-end ICT systems using Internet and Web-based standards that interface to database systems. This is the issue of Chapter 43.

CHAPTER 40 →

DATA WAREHOUSING

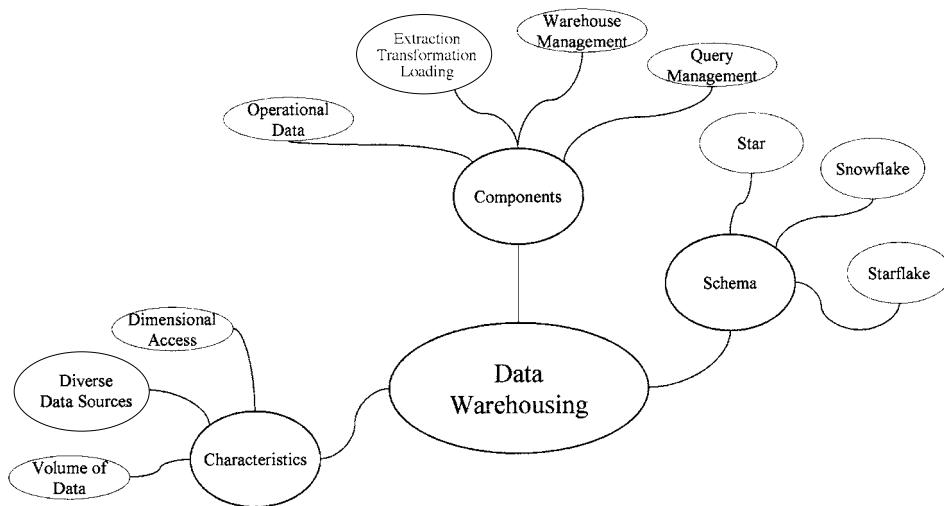
We can never tell what is in store for us.

Harry S. Truman (1884–1972)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define the concept of a data warehouse and that of a data mart
- Discuss some of the benefits and challenges of data warehousing
- Describe the steps needed to be taken in a data warehousing project
- Discuss the key components of a data warehouse
- Consider some of the issues involved in designing warehouse schemas



40.1 INTRODUCTION

Conventional database applications have been designed to handle high transaction throughput. Such applications are frequently called on-line transaction processing (OLTP) applications. The data available in such applications is important for running the day-to-day operations of some organisation. The data is also likely to be managed by relational or post-relational DBMS.

Contemporary organisations also need access to historical, summary data and access to data from other sources than that available through DBMS. For this purpose, the concept of a data warehouse has been created (Marakas, 2003). The data warehouse requires extensions to conventional database technology and also a range of application tools for on-line analytical processing (OLAP) (Chapter 41) and data mining (Chapter 42).

In this chapter we provide an overview of the concept of data warehousing. We describe some of the benefits and pitfalls associated with this construct. We also describe some of the necessary components of an architecture for data warehousing and some of the tools needed to construct data warehouses. Included in the discussion is also a brief examination of the issue of database design for data warehousing. Finally we consider the distinction between data warehouses and data marts.

40.2 DEFINITION

A data warehouse is a type of contemporary database system designed to fulfil decision-support needs (Chapter 4). However, a data warehouse differs from a conventional decision-support database in a number of ways:

- *Volume of data.* A data warehouse is likely to hold far more data than a decision-support database. Volumes of the order of over 400 gigabytes of data are commonplace
- *Diverse data sources.* The data stored in a warehouse is likely to have been extracted from a diverse range of application systems, only some of which may be database systems. These systems are described as data sources
- *Dimensional access.* A warehouse is designed to fulfil a number of distinct ways (dimensions) in which users may wish to retrieve data. This is sometimes referred to as the need to facilitate *ad-hoc* query

Inmon (2000) defines a data warehouse as being '*a subject-oriented, integrated, time-variant, and non-volatile collection of data used in support of management decision-making'*:

- *Subject-oriented.* A data warehouse is structured in terms of the major subject areas of the organisation such as, in the case of a university, students, lecturers and modules, rather than in terms of application areas such as enrolment, payroll and timetabling

- *Integrated.* A data warehouse provides a data repository which integrates data from different systems with data frequently in different formats. The objective is to provide a unified view of data for users
- *Time-variant.* A data warehouse explicitly associates time with data. Data in a warehouse is only valid for some point or period in time
- *Non-volatile.* The data in a data warehouse is not updated in real-time. Instead, it is refreshed from data in operational systems on a regular basis. A consequence of this is that the management of data integrity is not a critical issue for data warehouses

40.3 THE BENEFITS OF DATA WAREHOUSING

A data warehouse is seen to deliver three major benefits for organisations:

- A data warehouse provides a single manageable structure for decision-support data
- A data warehouse enables organisational users to run complex queries on data that traverses a number of business areas
- A data warehouse enables a number of business intelligence applications such as on-line analytical processing and data mining

The overall objective for a data warehouse is to increase the productivity and effectiveness of decision-making in organisations. This, in turn, is expected to deliver competitive advantage to organisations.

Example Consider the case of a supermarket chain. At the operational level, sales data is recorded in each supermarket in the chain at checkouts using electronic point-of-sale devices. This data allows administrative staff to record the amount of each type of product sold and, in turn, triggers decisions as to the amount of stock to reorder.

This sales data may be a major data source for a data warehouse perhaps sited at the headquarters of the supermarket chain. This enables headquarters' staff to monitor nationally or perhaps internationally its sales performance in certain areas, which helps it to make decisions as to what it should be selling and for what price.

However, the sales data is likely to be combined with data from other sources such as customer data, collected perhaps through the supermarket chain's loyalty card scheme. Associating sales data with customer data in its data warehouse may provide the chain with important information about the purchasing patterns of its customers. This may enable the chain to proactively plan activities in relation to



particular customer groups or in terms of new business opportunities such as financial services.

The components of this retail application are illustrated in Figure 40.1.

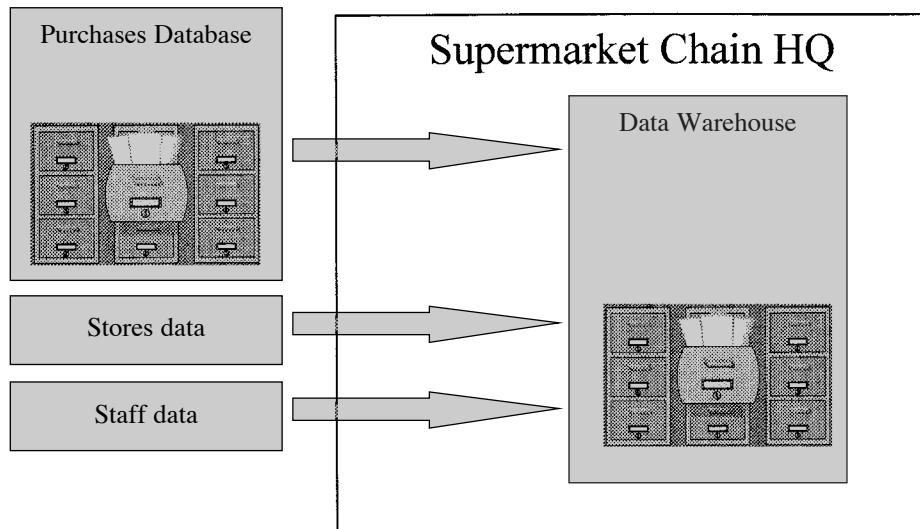


Figure 40.1 Data warehouses/data marts.

40.4 CHALLENGES OF DATA WAREHOUSING

Data warehousing projects are large-scale development projects. Typically a data warehousing project may take of the order of three years. Some of the challenges experienced in such projects are indicated below:

- Knowing in advance what the data users require, and determining the ownership and responsibilities in terms of data sources
- Selecting, installing and integrating different hardware and software required to set up the warehouse. The large volume of data needed in terms of a data warehouse requires large amounts of disk space. This means that estimation of storage volume is a significant activity
- Identifying, reconciling and cleaning existing production data and loading it into the warehouse. The diverse sources of data feeding a data warehouse introduces problems of design in terms of creating a homogeneous data store. Problems are also introduced in terms of the effort required to extract, clean and load data into the warehouse

- Managing the refresh or update process. Once a warehouse is established, a clear scheme of effectively managing the high volume of complex, *ad-hoc* queries needs to be produced

40.5 STEPS IN BUILDING A DATA WAREHOUSE

The key steps involved in a data warehousing project are outlined below (Inmon, 2000):

- Users specify information needs
- Analysts and users create a logical and physical design
- Sources of data are identified in operational systems, external sources etc.
- Source data is scrubbed, extracted and transformed
- Data is transferred and loaded into the warehouse periodically
- Users are given access to the warehouse data
- The warehouse is maintained in terms of changing requirements

Note that the first two stages are the same as those for conventional database development (Chapter 14). Also, identifying and managing data sources may be a key activity of the data administration function (Chapter 22).

40.6 COMPONENTS OF A DATA WAREHOUSE

Figure 40.2 illustrates some of the major components of a data warehouse:

- *Operational data.* Data for the warehouse may be sourced in a number of ways, e.g. from mainframe-based hierarchical or network databases, from relational databases and from data in proprietary file systems
- *Extraction, transformation and loading functions.* These ETL operations or functions are concerned with extracting data from source systems, transforming it into a suitable form and loading the transformed data into the data warehouse
- *Warehouse management.* A series of functions must be provided to manage the warehouse: consistency analysis, indexing, denormalisation, aggregation, backup and archiving
- *Query management.* The warehouse must perform a series of operations concerned with the management of queries for use by a variety of actors: reporting and query tools, OLAP tools or tools for data mining

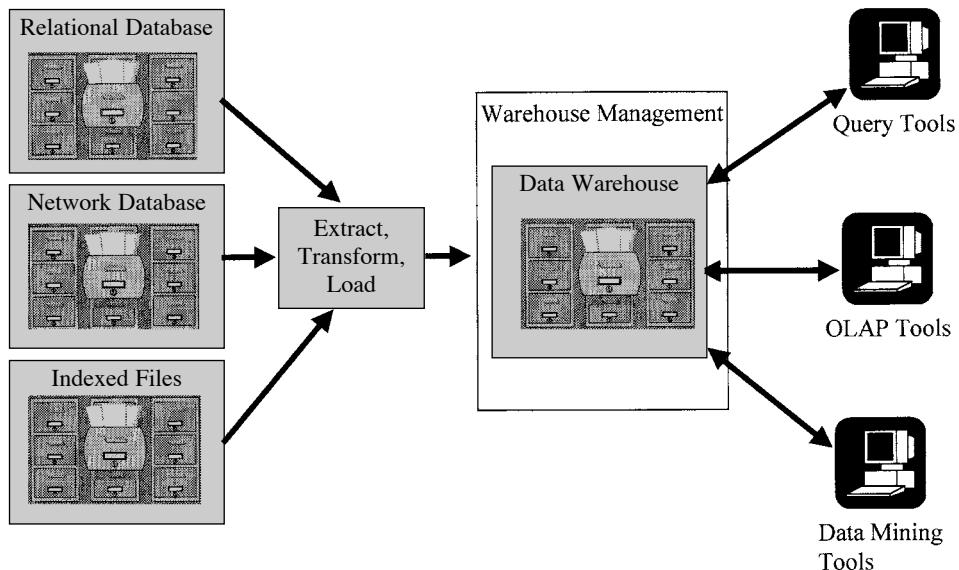


Figure 40.2 Components of a data warehouse.

Example ➔

Clickstream records are records of the behaviour of users of Web-sites. Capture and analysis of such data is seen to be critical to e-Commerce activity (Chapter 5) (Kahavi *et al.*, 2002). Such data needs to be integrated with demographic and other behavioural data in large data warehouses. OLAP and data mining applications (Chapters 41 and 42) may then be run to support strategic decision-making in companies.

40.7 EXTRACTION, TRANSFORMATION AND LOADING

As mentioned in the previous section, a number of tools exist to:

- Extract data from source systems such as database and file systems
- Perform necessary transformation of such data
- Load the transformed data into the data warehouse

Example ➔

Transformation of data may include cleansing data of rudimentary errors, summarising data in various ways, and converting and unifying various ways of coding data. For instance, source system 1 may use the codes 'M' for male and 'F' for female, while source system 2 may use the codes 1 and 2 for male and female. Transformation functions will involve converting both forms of coding into a common form acceptable to the data warehouse.



Such ETL tools effectively constitute middleware between source systems and the data warehouse, and may be built in a number of ways which include:

- *Code generators.* These create transformation programs based on data definitions provided for both source and target systems. The main problem is the large number of programs needed for all transformations required by a particular data warehousing application
- *Data replication tools.* These employ database triggers or a recovery log to capture transformations on a single data source from one system. Such changes are then applied to a copy of such data stored on a target system. Typically, the complexity of data transformations possible is limited
- *Dynamic transformation engines.* These are rule-driven engines that capture data from source systems at regular, predefined intervals. Transformations are applied and the data is loaded into a target environment

40.8 FORMS OF DATA IN A DATA WAREHOUSE

We may distinguish a number of different forms of data in a data warehouse:

- *Detailed data.* This comprises the detailed production data. Usually, detailed data is not stored on-line but is aggregated on a periodic basis. Sales data from the supermarket application described above is an example of detailed data
- *Summarised data.* Data in the warehouse is normally summarised or aggregated to speed up the performance of queries. The data may be lightly summarised or highly summarised. Summarised data needs to be updated periodically when the detailed data is refreshed. As an example, sales data may be summarised in terms of particular geographical areas, time-periods and/or product lines
- *Meta-data.* Data about data is needed to enable the extraction, transformation and loading processes by mapping data sources to the warehouse schema. Meta-data is also used to automate the production of summary data and to facilitate query management
- *Archive data.* Data needs to be periodically archived from the warehouse to prevent the database growing too large for its platform. Normally, this is done on the basis of some retention period established for data
- *Backup data.* Just as in conventional databases, detailed, summary and meta-data need to be backed up regularly in order to recover from failure

40.9 DATA MARTS

A data mart is a restricted data warehouse. It may be restricted in a number of ways:



- *Type of data.* A data mart may be limited to a particular type of input data such as that available from particular DBMS
- *Business area.* A data mart may be designed to store data representing a particular business area rather than representing data applicable to the entire organisation
- *Geographic area.* A data mart may be set up for one specific geographic area in terms of the activities of some organisation

Maintaining restrictions such as these means that whereas a data warehouse may store of the order of hundreds of gigabytes of data, a data mart may typically store something of the order of tens of gigabytes of data. This means that a data mart can usually be built more rapidly than a data warehouse and is easier to manage.

40.10



DESIGNING THE DATA WAREHOUSE SCHEMA

Designing a schema for a data warehousing application is a specialist case of database design. The methodology and techniques discussed in Part 4 are as relevant to data warehousing applications as they are to conventional database systems. However, two issues assume particular prominence in a data warehouse: the large volume of data and the issue of achieving satisfactory levels of retrieval performance. To provide satisfactory levels of performance frequently means designing specialised physical schemas for warehousing applications. In this section we consider three generic designs for warehousing schemas proposed by Anahory and Murray (1997).

40.10.1 STAR SCHEMAS

Star schemas are physical schemas that store what Anahory and Murray term ‘factual data’ in a central table surrounded by reference tables storing data relevant to particular dimensions needed for decision-support. The central factual data should be designed to store data generated by past events in relation to some organisational activity. Hence, it constitutes read-only data and means that the size of the fact table can be extremely large in comparison to that of the reference tables. Figure 40.3 illustrates a possible star schema relevant to the academic domain. At the centre we store ‘facts’ about the assessment of students. Three reference tables are associated with this central table, allowing us to analyse the factual data in terms of student characteristics, lecturer characteristics or assessment characteristics.

Each reference table may be a denormalised version of a number of other tables in order to improve retrieval performance. For instance, in the schema illustrated in Figure 40.3, the table assessment contains module information and the table staff contains school information.

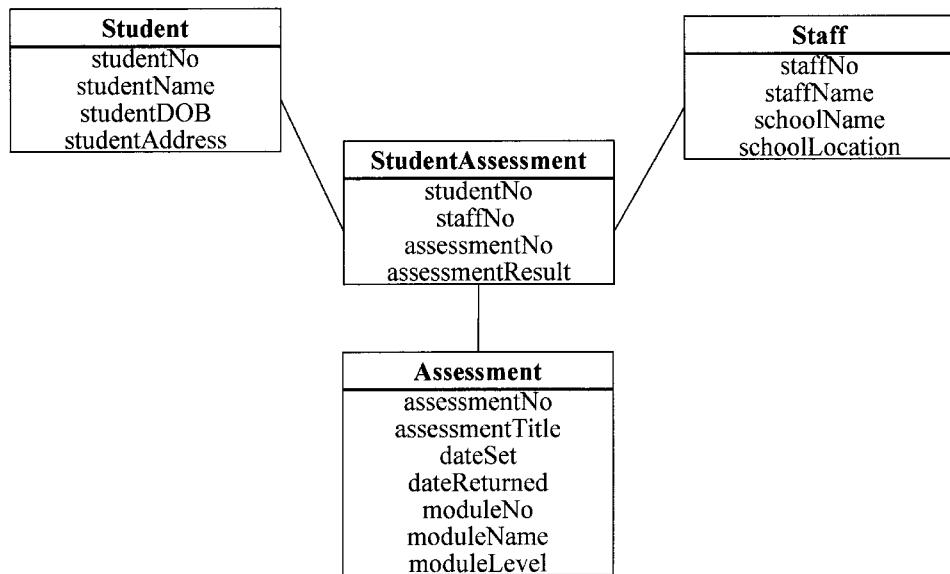


Figure 40.3 A star schema.

40.10.2 SNOWFLAKE SCHEMA

Snowflake schemas are a variation on the star schema. In a snowflake schema, each dimension can have a number of its own dimensions. This means that reference tables are not denormalised in a snowflake schema. Figure 40.4 illustrates a snowflake schema for the university application.

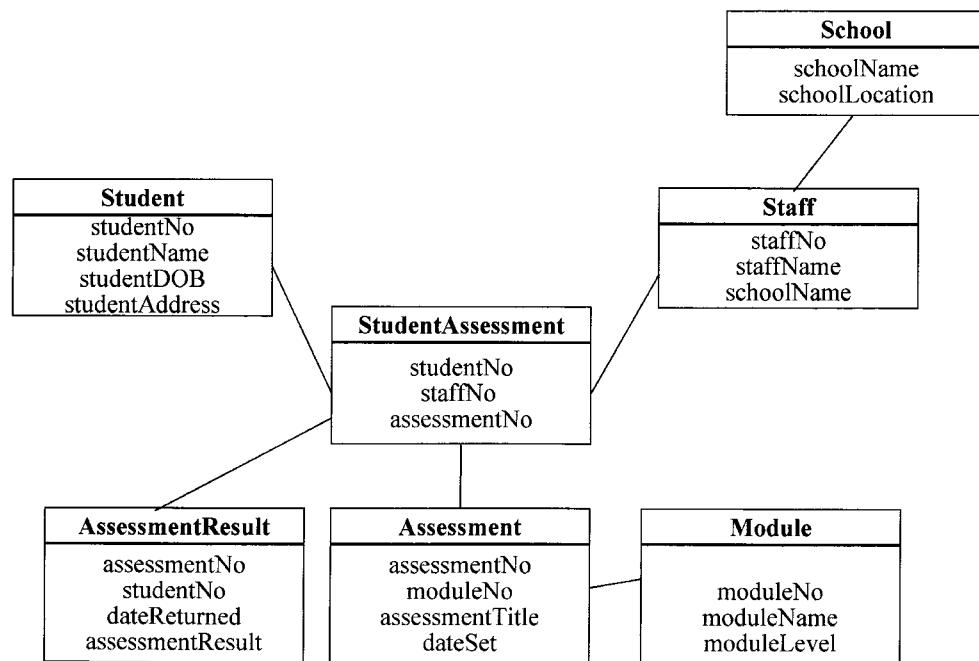
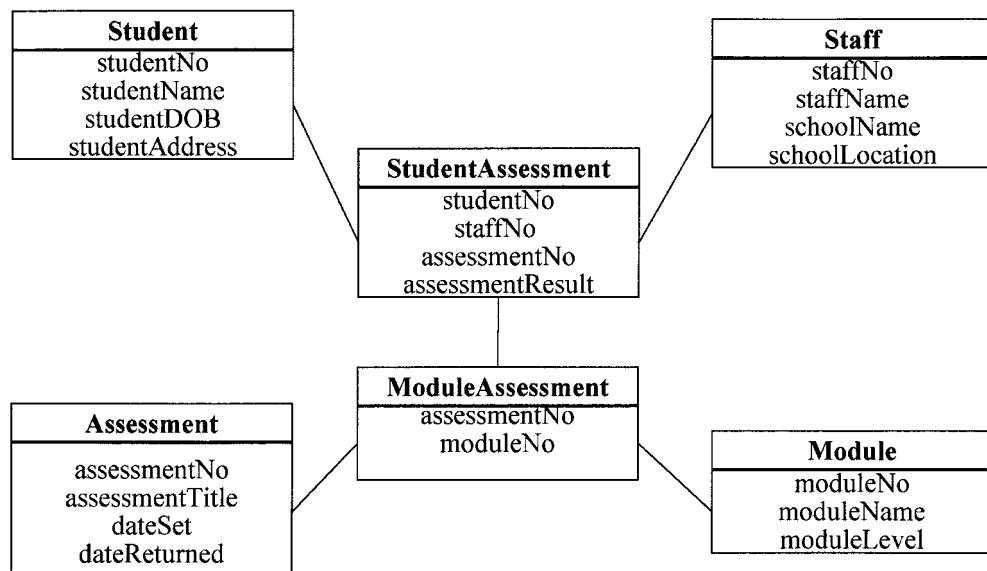
40.10.3 STARFLAKE SCHEMA

Starflake schemas occupy the middle ground between star and snowflake schemas. In a starflake schema, some of the reference tables will be denormalised and some will be normalised. Figure 40.5 provides an example of a starflake schema. In this schema the assessment information is now modelled on the lines of a normalised snowflake pattern, while the staff table is still denormalised.

40.11 CASE STUDY: ORACLE 9i

Oracle 9i Enterprise Edition describes itself as a DBMS specifically designed for data warehousing applications. The key features offered in this version, specifically pinpointed at data warehousing, include:

- *Summary management.* The DBMS is able to automatically maintain stored aggregates of data from base tables. These can be used to improve the performance of queries which attempt to summarise data on the basis of common

**Figure 40.4** A snowflake schema.**Figure 40.5** A starflake schema.

dimensions. The DBMS includes an advisor facility for the DBA which helps choose summary tables on the basis of database workload and usage statistics

- *Analytical functions.* The DBMS provides a number of added SQL functions for data warehousing work, including the ability to rank data and move aggregates across time dimensions
- *Bitmapped indexes.* Bitmapped indexes are particularly suitable for improving the performance of queries which include a wide range of criteria. The DBMS uses data compression technology to store such indexes efficiently
- *Advanced joins.* The DBMS supports advanced forms of joins such as partition joins and hash joins. Partition joins efficiently execute on tables that have been partitioned on the basis of a join key. Hash joins efficiently execute for operations that demand sorted data or parallel execution (Chapter 38)
- *Cost-based optimiser.* ORACLE's cost-based optimiser (Chapter 30) chooses the most effective execution strategy for a query based on detailed statistics held about the database. This can prove effective for queries against star schema
- *Resource management.* Effective management of resources is critical to data warehousing applications. The ORACLE DBMS uses the idea of a resource class to which can be assigned a defined set of resources and a defined group of users

40.12 SUMMARY

- A data warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data used in support of management decision-making
- A data mart is a small data warehouse, designed usually for a particular business area
- Data warehouses and data marts deliver decision-support applications in organisations
- Data warehousing projects are large-scale development projects
- The components of a data warehouse include: production data, extraction, transformation and loading functions, warehouse management functions and query management functions
- Three specialist forms of schema are relevant to data warehousing applications: star, snowflake and starflake schemas

40.13 ACTIVITIES

1. Consider the use of data warehousing in a university setting. What would constitute the operational data in this case?

2. For what purposes would a data warehouse be established in this domain?
3. What sort of data marts might be appropriate?

40.14 REFERENCES

- Anahory, S. and D. Murray (1997). *Data Warehousing in the Real World: A Practical Guide for Building Decision-Support Systems*. Harlow, UK, Addison-Wesley.
- Inmon, W.H. (2000). *Building the Data Warehouse*. New York, John Wiley.
- Kahavi, R., N.J. Rothlededer and E. Simoudis (2002). Emerging trends in business analytics. *Communications of the ACM* **45**(8): 45–53.
- Marakas, G.M. (2003). *Modern Data Warehousing, Mining, and Visualization: Core Concepts*. Upper Saddle River, NJ, Prentice-Hall.

CHAPTER 41

ON-LINE ANALYTICAL PROCESSING

Half of analysis is anal.

Marty Indik

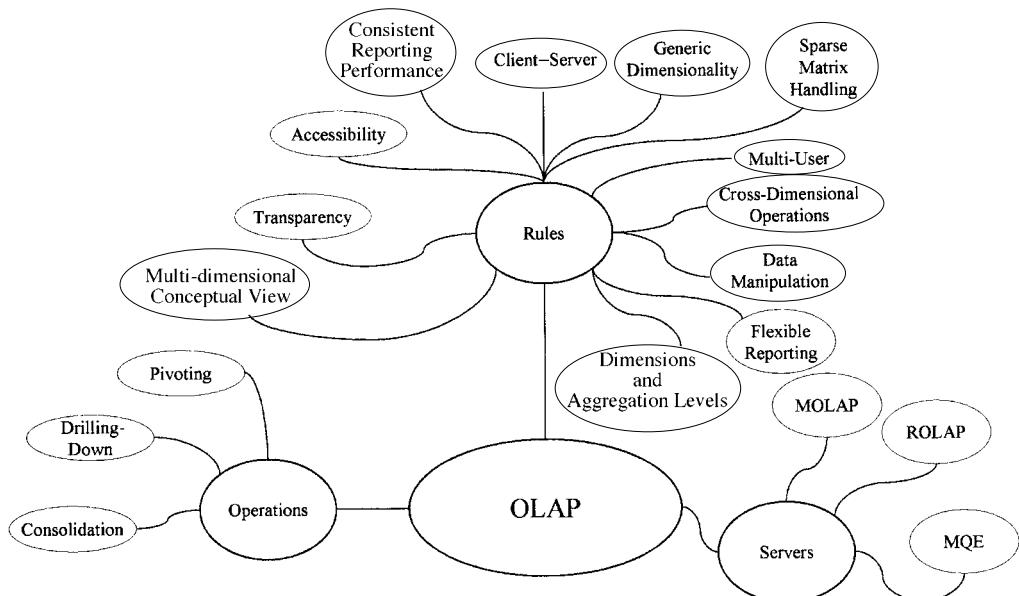
It requires a very unusual mind to undertake the analysis of the obvious.

Alfred North Whitehead (1861–1947)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define the concept of OLAP and the operations it supports
- Relate Codd's rules for OLAP tools
- Discuss the essential features of OLAP tools
- Describe different types of OLAP servers



41.1 INTRODUCTION

Modern database applications such as market analysis and financial forecasting require access to large databases for the support of queries which can rapidly produce aggregate data. Such applications are frequently called analytical on-line processing applications, or OLAP for short. OLAP is contrasted with on-line transaction processing, or OLTP.

The increasing need for supporting such applications has led to the development of specialised tools, used normally in association with data warehouses or data marts. This chapter provides a brief review of the area.

41.2 DEFINITION

OLAP is a term coined by Tedd Codd (Codd *et al.*, 1993) to define an architecture that supports complex analytical operations such as consolidation, drilling-down and pivoting:

- *Consolidation*. Also referred to as rolling-up, consolidation comprises the aggregation of data, such as modules data, being aggregated into courses data, and courses data being aggregated into schools or departments data
- *Drilling-down*. This is the opposite of consolidation and involves revealing the detail or disaggregating data, such as breaking down school-based data into that appropriate to particular courses or modules
- *Pivoting*. This operation, sometimes referred to as ‘slicing and dicing’, comprises the ability to analyse the same data from different viewpoints, frequently along a time axis. For example in the university domain, one slice may be the average degree grade per course within a school; another slice may be the average degree grade per age band within a school

To support these applications, OLAP applications tend to utilise special-purpose, multi-dimensional data structures.

41.3 MULTI-DIMENSIONAL DATA STRUCTURES

OLAP systems use multi-dimensional data structures to store data and relationships. Such data structures are frequently visualised as data cubes or data cubes of data cubes. Each side of a cube represents a data dimension of relevance to some organisation. The intersection of each dimension forms a cell. Within each cell a data value is stored. Each such data value may be derived from some other cube representing usually some lower levels of aggregation.

Example → Figure 41.1 illustrates the organisation of data in terms of part of a data cube for an academic application. Here, three dimensions are relevant: the degree level (MSc, BSc), the year and semester, and the course. In each cell of the cube the average examination mark is stored. Such a structure enables us to obtain answers to the following queries:

- The average examination mark per course (*pivot* on course), or the average examination mark per year/semester (*pivot* on year/semester), or the average examination mark per level of a course (*pivot* on level)
- The six distinct courses represented consist of all those offered by the School of Computing. Hence we might *consolidate* to the school level to answer a question about the average examination mark per school
- A given cube may consist of a hierarchical pre-aggregation of data. For instance, each examination average may be produced from a hierarchy of the module examination grades relevant to each course. This would enable us to *drill-down* to obtain the specific examination grades per module

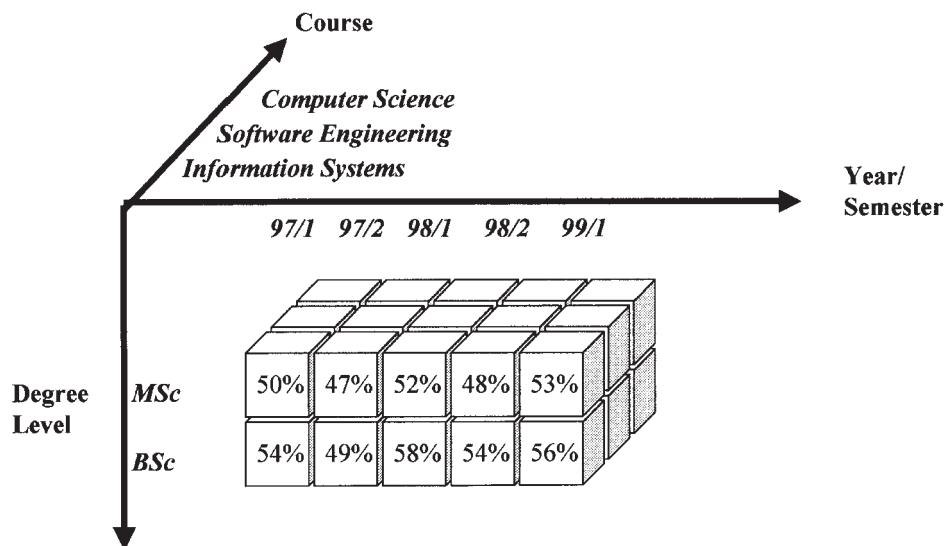


Figure 41.1 Multi-dimensional data.

41.4 SPARSITY

In terms of many applications, not all the cells in a data cube will contain data. In fact, in large applications with many dimensions, the vast majority of cells will contain no data. This is referred to as sparsity and means that a data cube is normally a sparse data structure.



Suppose we have a sales application in which the relevant dimensions are product type, customer, time of sale, location of sale, value of sale and quantity sold. In such a data cube, most of the cells will be empty because in any one period each customer will buy only a small proportion of the product range.

Because of this, OLAP servers will normally have the ability to store such data in compressed form that maximises space utilisation. Data that exists in a high percentage of the cells in a cube – so-called dense data – can be stored separately from sparse data – data in which a significant percentage of the cells of a cube is empty. This ability means that OLAP servers can both minimise physical storage capacity for data as well as making it possible to load more data into computer memory at any one time. This latter capability improves processing performance.

41.5

CODD'S RULES FOR OLAP TOOLS

In a similar manner to his twelve rules for relational DBMS, Codd has formulated twelve rules for selecting tools for OLAP:

- *Multi-dimensional conceptual view.* OLAP tools should provide a multi-dimensional model which corresponds with the user's views of the organisation. Such a model should also be easy-to-use
- *Transparency.* The technology used, the underlying database architecture and the various data sources should be transparent to the user
- *Accessibility.* The OLAP tool should be able to access data from sources in different formats – relational, non-relational and in terms of legacy systems
- *Consistent reporting performance.* The user should not perceive any degradation in performance when the number of dimensions, aggregations or the size of the database increases
- *Client-server architecture.* The OLAP tool should be able to work effectively in a client–server environment
- *Generic dimensionality.* Every dimension in the database must be equivalent in both structure and operational capabilities
- *Dynamic sparse matrix handling.* The OLAP tool should be able to adapt its physical organisation to optimise the handling of sparse matrix handling
- *Multi-user support.* The OLAP tool should be able to handle concurrent access to data
- *Unrestricted cross-dimensional operations.* The OLAP tool must be able to support dimensional hierarchies and automatically perform consolidation calculations within and across dimensions

- *Intuitive data manipulation.* Pivoting, drill-down and consolidation should be accomplished using classic graphical user interface operations such as point-and-click and drag-and-drop
- *Flexible reporting.* It must be possible to arrange rows, columns and cells in a fashion that meets the needs of particular users
- *Unlimited dimensions and aggregation levels.* The OLAP tool should not impose any restriction on the number of dimensions or aggregation levels in an analytical model

41.6 OLAP SERVERS

An OLAP application will normally reside on a specialised OLAP server, and the OLAP server will exist in the application layer of a three-tier client–server environment. The OLAP server uses multi-dimensional structures to store data and relationships. Three different types of OLAP tool may run on the OLAP server (Berson and Smith, 1997):

41.6.1 MULTI-DIMENSIONAL OLAP (MOLAP)

MOLAP tools use data structures founded in array technology and utilise efficient storage techniques to optimise disk management through sparse data management. Typically such tools demand a close coupling between the presentation and application layer of the three-tier architecture. Requests are issued from access tools in the presentation layer to the MOLAP server. The MOLAP server periodically refreshes its data from base relational and/or legacy systems (Figure 41.2).

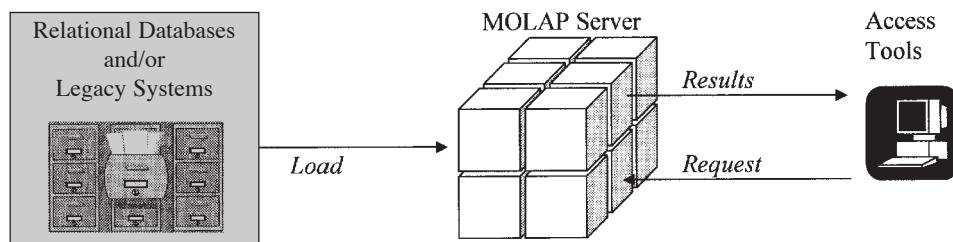


Figure 41.2 MOLAP server.

41.6.2 RELATIONAL OLAP (ROLAP)

Sometimes called multi-relational OLAP, this has been the fastest growing area of OLAP tools. ROLAP utilises connection to existing RDBMS products through the maintenance of a meta-data layer. This meta-data layer allows the creation of multiple multi-dimensional views of two-dimensional relations. Some



ROLAP products maintain enhanced query managers to enable multi-dimensional query, others emphasise the use of highly-denormalised database designs (typically a 'star' schema) to facilitate multi-dimensional access (Figure 41.3).

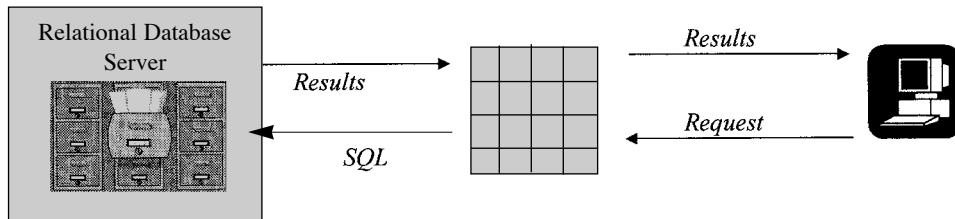


Figure 41.3 ROLAP server.

41.6.3 MANAGED QUERY ENVIRONMENT (MQE)

MQE systems are an intermediate form between ROLAP and OLAP servers. They provide limited capability to access either RDBMS directly or through using an intermediate MOLAP server. When interacting with the MOLAP server, data will be delivered in the form of a data cube to the desktop where it is stored and can be accessed locally (Figure 41.4).

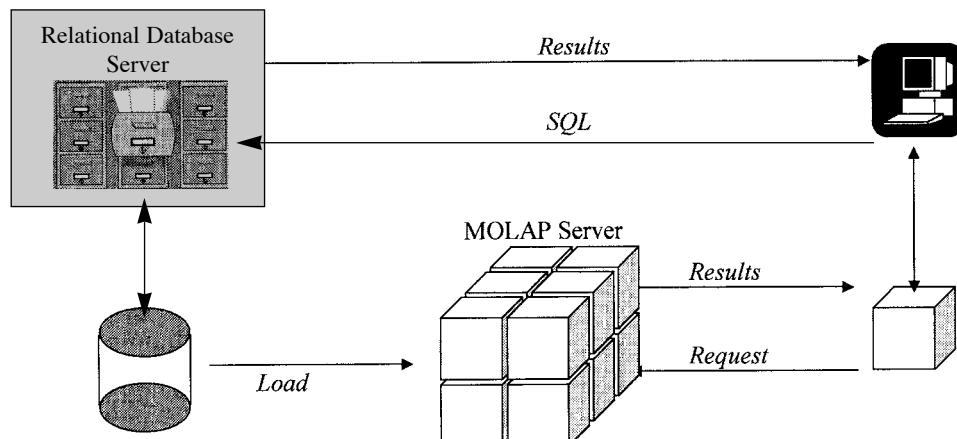


Figure 41.4 Managed query environment.



One of the key problems of a number of the standards for SQL discussed in Chapter 11 is that the language does not contain sufficient functionality for business analysts wishing to compute moving averages, cumulative sums and other statistical functions. For this reason, a number of additions to the

standard in this area have been proposed to support OLAP applications. A number of vendors of database technology, such as IBM and ORACLE, have been instrumental in this movement. To illustrate some of the principles of using SQL in this manner, consider the case in which we wish to show the monthly sales figures for branch office 001 along with monthly year-to-date figures for this branch. Assume that we have an appropriate database with a table called BranchQuarterlySales. In this table there are three columns – branchNo, quarter and quarterlySales. Further assume that one of the OLAP extensions to SQL is a CUME function which enables us to compute a running or cumulative total of a specified column's value. A relevant SQL query might then be:

```
SELECT quarter, quarterlySales CUME(quarterlySales) AS yearToDate
FROM BranchQuarterlySales
WHERE branchNo = '001'
```

The resulting output from this query might then look like:

quarter	quarterlySales	yearToDate
1	20000	20000
2	30000	50000
3	25000	75000
4	35000	110000

41.8 SUMMARY

- OLAP involves the multi-dimensional analysis of large volumes of data
- OLAP applications tend to utilise special-purpose multi-dimensional data structures. Such data structures are frequently visualised as cubes or cubes of cubes with each side of a cube being a dimension
- OLAP systems support complex analytical operations such as consolidation, drilling-down and pivoting
- Codd formulated twelve rules for selecting tools for OLAP
- There are three main categories of OLAP tool: multi-dimensional OLAP, relational OLAP and managed query environment

41.9 ACTIVITIES

1. Write a report analysing the applicability of OLAP technology to an organisation such as a university.
2. Identify a given OLAP product and determine its type.
3. Determine the key uses of OLAP technology in modern business.

41.10  REFERENCES

Berson, A. and S.J. Smith (1997). *Data Warehousing, Data Mining and OLAP*. New York, McGraw-Hill.

Codd, E.F., S.B. Codd and C.T. Salley (1993). *Providing OLAP (On-Line Analytical Processing) to User-Analysts: An IT Mandate*. Ann Arbor, MI, Arbor Software Corporation.

CHAPTER 42 →

DATA MINING

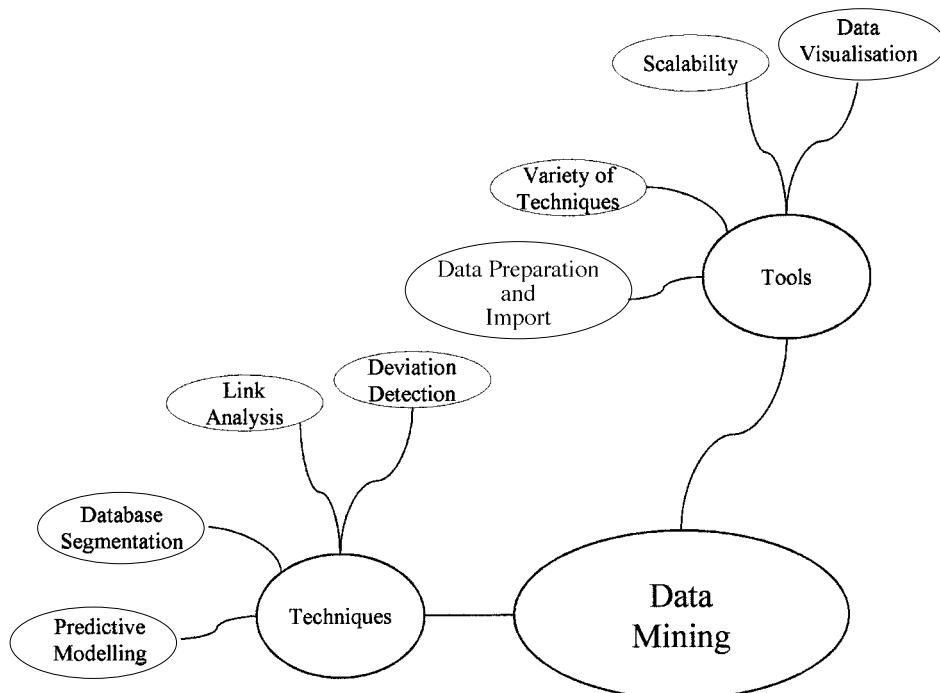
Aristotle maintained that women have fewer teeth than men; although he was twice married, it never occurred to him to verify this statement by examining his wife's mouths.

Bertrand Russell (1872–1970)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define the concept of data mining
- Discuss the essential features of data mining techniques



42.1 INTRODUCTION

Data mining is normally used in association with data warehouses and data marts. To gain benefit from a data warehouse or mart, the data patterns resident in the large data-sets characteristic of such applications need to be extracted. As the size of a data warehouse grows, the more difficult it is to extract such data using the conventional means of query and analysis. Data mining involves the use of automatic algorithms to extract this data.

42.2 DEFINITION

Data mining is the process of extracting previously unknown data from large databases and using it to make organisational decisions (Kantardzic, 2002). There are a number of features to this definition:

- Data mining is concerned with the discovery of hidden, unexpected patterns of data
- Data mining usually works on large volumes of data. Frequently, large volumes are needed to produce reliable conclusions in relation to data patterns
- Data mining is useful in making critical organisational decisions, particularly those of a strategic nature

Data mining began its life in specialist applications such as geological research (searching for natural resources) and meteorological research (weather forecasting). More recently it has been applied in a number of areas of industry and commerce.

Examples  Examples of such business applications are listed below:

- In retail chains, data mining has been used to identify the purchasing patterns of customers and associating this data with demographic characteristics such as the age and class profile of customers. Such patterns are useful in making decisions such as what products to sell in which retail stores and when
- In the insurance industry, data mining has been used to analyse the claims made against insurance policies and hence feed into actuarial decisions such as the pricing of particular policies
- In the finance industry, banks have used data mining techniques to identify fraudulent credit card use among its transaction data
- In the medical domain, data mining may be applied to identify successful medical treatments for particular medical complaints
- In direct-mail targeted marketing, retailers must be able to identify subsets of the population likely to respond to their promotions

42.3 DATA MINING OPERATIONS OR TECHNIQUES

Data mining is implemented in terms of a number of operations or techniques (Cabena *et al.*, 1997). There are four main operations associated with data mining: predictive modelling, database segmentation, link analysis and deviation detection. Particular operations are usually associated with particular applications. For instance, credit card fraud detection is likely to be implemented by predictive modelling, while profiling of customers is likely to be implemented by database segmentation.

Each operation may be implemented using a number of different techniques or algorithms. Since each particular technique has its own strengths and weaknesses, data mining tools may offer a choice of techniques to implement an operation:

42.3.1 PREDICTIVE MODELLING

Predictive modelling is an attempt to model the way in which humans learn. An existing data-set is analysed to form a model of its essential characteristics. This model is developed using a supervised learning approach consisting of two phases. The first, training, phase involves building a model using a large sample of data known as the training set. The second, testing, phase involves trying out the model on new data. Predictive modelling is associated with the techniques of classification and value prediction.

Classification involves determining a class for each row in a database. This may be done using decision trees or neural networks.

A decision tree (Beynon-Davies, 1998) represents a classification problem as a series of conditions. Each node represents a condition and each branch a specific response to a condition. The leaf nodes of the tree represent the range of classes into which a row might be classified. This tree will be induced from an analysis of the input data-set.

A neural network comprises a network of nodes. The nodes are divided up into three layers: an input layer, a processing layer and an output layer. Nodes in the input layer are activated depending on the characteristics of the data input into the network. The input nodes activate in turn the processing nodes, the 'strength' of the activation being dependent on weights assigned to the relationships between nodes. The processing nodes determine the relevance of applicable output nodes (the classes in the problem) based on these weights. The weights assigned to the relationships between nodes will be constructed in a phase in which the network 'learns' from a sample data-set.

Value prediction is a technique used to estimate a value associated with a database row. The technique generally uses the well-established statistical algorithms associated with linear regression. Linear regression attempts to build the best fit of a straight line through a plot of data.



Example → A fictitious example of a classification approach to data mining is illustrated in Figure 42.1. Here a pattern identified in a data-set of student applications for university courses is represented as a simple decision tree. The pattern induced from the existing data on student applications may be used in a number of ways. For example, it might be used to target marketing material at particular populations or it might be used to construct campaigns to address an obvious gender imbalance in applications for university courses.

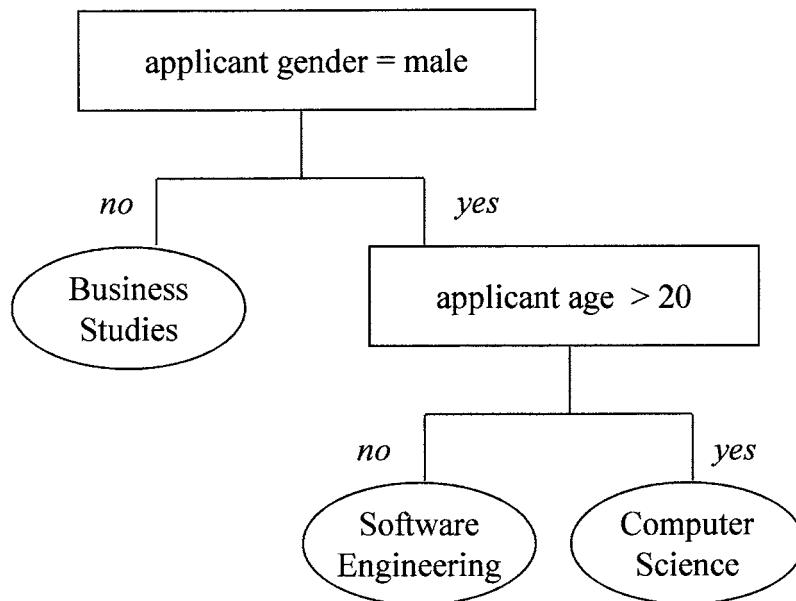


Figure 42.1 A tree induction example.

42.3.2 DATABASE SEGMENTATION

Database segmentation involves partitioning the database into a number of segments or clusters. Each segment comprises a set of rows having a number of properties in common. Segments are built and refined using a process of unsupervised learning.

Examples → An example of the application of this technique is in the area of bank note forgeries. A database might have been constructed in terms of observations recorded against a population of banknotes. Performing segmentation on this data-set allows us to cluster the banknotes into distinct sets in terms of dimensions such as size, colour etc. This may allow us not only to distinguish forgeries from legal tender, it may also allow us to distinguish particular types of forgery, perhaps associated with particular criminal groups.

Database segmentation may also be applied in market basket analysis (Witten and Frank, 2000). This is the use of association techniques to find groups of items associated with consumer transactions. For example, in a food supermarket a company gathers data about purchasing at checkouts. From conducting data mining on such data we determine that customers often purchase beer and disposable nappies at the same time. This may be because parents with young children typically stock up with both for the weekend. Such associations can be used for a number of purposes, such as planning store layouts and discounting of products sold together.

42.3.3 LINK ANALYSIS

Link analysis attempts to establish associations between individual records in the data-set. Such associations are frequently represented as rules in a technique known as associations discovery.

Example ➔ For instance, an analysis of data collected on student progression may identify a rule such as *if a computing student passes a first year programming module then in 40% of cases he or she will gain an upper second class degree.*

In a technique known as sequential pattern discovery, the associations are represented in terms of the presence of one set of data-items being followed by another set of data-items.

Example ➔ An example here might be the patterns identified in student module enrolments, allowing us to predict the likely take-up of final-year module options. Similarly, time sequence discovery identifies patterns between data which are time-dependent. For instance, a link analysis of customer purchasing patterns using this technique might find that within a certain time period after making a house purchase, buyers purchase items such as washing machines, refrigerators etc.

42.3.4 DEVIATION DETECTION

Deviation detection involves identifying so-called outliers in the population of data – i.e. those that deviate from some norm. Deviation detection can be detected by statistical and visualisation techniques such as linear regression. A classic example of deviation detection is that applicable to the quality control associated with manufacturing. Analysis of defect data associated with the production process may allow a company to identify reasons for the introduction of faults and may lead to improvements in production processes.

42.4 DATA MINING TOOLS

Data mining tools are likely to incorporate the following facilities:

- *Data preparation and import facilities.* The data mining tool should be capable of importing data from the target environment
- *Selection of data mining operations and techniques.* The data mining tool should provide a range of algorithms for pattern detection
- *Product scalability and performance.* The tool should be capable of handling sufficient volumes of data and should offer satisfactory levels of performance in terms of data manipulation operations
- *Facilities for data visualisation.* The tool should be capable of presenting a number of different graphical representations of patterns detected

42.5 CASE STUDY: DATA MINING OF MEDICAL DATA

In the 1990s, approximately 10% of the population of Singapore were diabetic. This is a medical condition with many side-effects such as increased risk of eye disease and kidney failure. Early detection of the condition and proper care management can significantly improve the health and longevity of sufferers.

To combat the disease, the government of Singapore introduced in 1992 a regular screening programme for diabetic patients in public hospitals. This programme generated a vast amount of data about patients, including patient details, clinical symptoms, eye-disease diagnoses and treatments. Over ten years of data has now been collected and made available for data mining. The aim is to use data mining to discover rules to enable physicians to better understand the association of the disease with particular population segments.

Before the data could be used it had to be cleaned of typographical errors, missing values and incorrect values. Rule generation algorithms applied to the data also produced too many rules to be practically used. Therefore, data pruning techniques had to be used to remove insignificant rules. Data visualisation software was also used to present the rules for easy manipulation by physicians.

42.6 SUMMARY

- Data mining is the process of extracting previously unknown data from large databases and using it to make organisational decisions
- There are four main operations associated with data mining: predictive modelling, database segmentation, link analysis and deviation detection
- Each operation may be implemented using different algorithms
- A particular software vendor may offer a number of algorithms to implement an operation

42.7 ACTIVITIES

1. Write a brief report identifying possible applications of data mining in an organisation such as a university.
2. For the business applications listed in section 42.2, identify the most appropriate data mining technique.

42.8 REFERENCES

- Beynon-Davies, P. (1998). *Information Systems Development: An Introduction to Systems Engineering*. Basingstoke, Macmillan (now Palgrave).
- Cabena, P., P. Hadjinian, R. Stadler, J. Verhees and A. Zanasi (1997). *Discovering Data Mining from Concept to Implementation*. Englewood Cliffs, NJ, Prentice-Hall.
- Kantardzic, M. (2002). *Data Mining: Concepts, Models, Methods and Algorithms*. Chichester, John Wiley.
- Witten, I.H. and E. Frank (2000). *Data Mining*. San Francisco, CA, Morgan Kaufman.

CHAPTER 43 →

DATABASES AND THE WEB

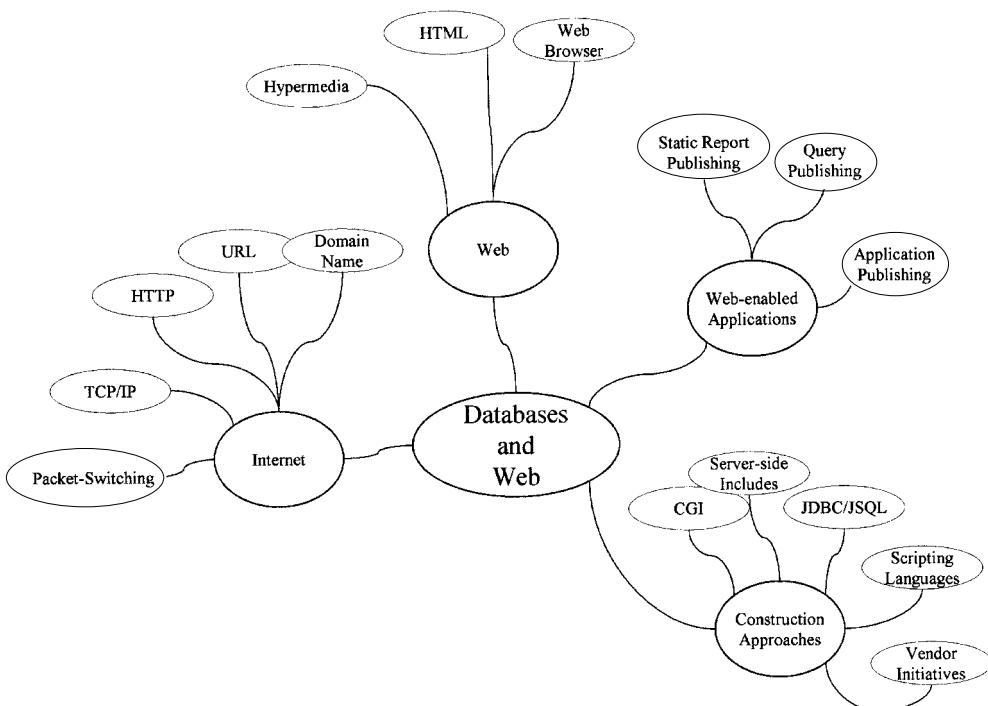
All men are caught in an inescapable network of mutuality.

Martin Luther King Jr (1929–68)

LEARNING OUTCOMES

At the end of this chapter the reader will be able to:

- Define the concept of the Internet
- Describe the key components of the Web
- Relate the place of database systems in Internet and Web applications
- Provide an overview of a number of different ways of building Web-enabled database applications





43.1 INTRODUCTION

The Internet – short for inter-network – began as a Wide Area Network in the USA, funded by its Department of Defense to link scientists and researchers around the world. It was initially designed primarily as a medium to exchange research data.

Currently the Internet is a set of interconnected computer networks distributed around the globe. The Internet can be considered on a number of levels. At the base level we have the technical infrastructure of the Internet which is composed of packet-switched networks and a series of communication protocols. On this layer runs a series of applications such as electronic mail (e-mail) and more recently the World-Wide-Web.

In this chapter we provide an overview of the Internet and the Web. We describe some of the ways in which these technologies are affecting the construction of ICT system applications. We also consider some of the reasons for the use of database systems within Internet/Web applications. Finally we conclude with a consideration of a number of ways in which Web-enabled database applications can be built.

43.2 INTERNET INFRASTRUCTURE

The technical infrastructure of the Internet consists of a number of components. These include:

- Packet-switched networks
- TCP/IP
- HTTP
- IP addresses
- URLs
- Domain names

43.3 PACKET-SWITCHED NETWORKS

The early computer networks were modelled on the local and long-distance telephone networks that dated back to the early 1950s. Computer networks tended to be composed of leased telephone lines. In these traditional telephone networks, a connection between a caller and the receiver was established through telephone switching equipment (both mechanical and computerised) selecting specific electrical circuits to form a single path. Once the connection was established, data travelled along the path. This is known as a circuit-switching network.

Circuit-switching works well for telephone communication but proves



expensive for data communication because of the need to establish a point-to-point connection for each pair of senders/receivers. Most computer networks therefore use a form of network technology known as packet-switching. In such a network the data in a message or file is broken up into chunks known as packets. Each packet is electronically labelled with codes that indicate the sender (origin) and receiver (destination) of the packet. Data travels along the network from computer to computer until it reaches its destination. Each computer in the network determines the best route forward for the packets it receives and must transmit. Computers that make these decisions are known as routers. The destination computer reassembles the packets into the original message.

There are a number of advantages to packet-switching networks for data communications. Long streams of data can be broken up into small, manageable chunks. This means that the packets can be distributed efficiently to balance the traffic across a wide range of possible transmission paths in a network.

43.4

TCP/IP

One of the key objectives of most computer networks is to achieve high levels of connectivity. Connectivity is the ability of computer systems to communicate with each other and to share data. For connectivity, standards must be defined to enable communication between sender and receiver. Such standards are embodied in communication software.

One approach to developing higher connectivity among systems is by using the idea of open systems. Open systems are built on public domain operating systems, user interfaces, application standards and networking standards. One of the oldest examples of an open systems model for communications is TCP/IP. This was developed by the US Department of Defense in 1972. The Transmission Control Protocol/Internet Protocol is the communications software model underlying the Internet. A protocol is a statement that explains how a specific networking task should be performed. TCP/IP divides the communication process into five layers of network tasks:

- *Application.* The application layer is that closest to the network user. The application layer provides data entry and presentation functionality to the end-user of the network
- *Transport/TCP.* This layer breaks application data up into TCP packets known as datagrams. Each packet consists of a header comprising the address of the sending computer, data for reassembling the original data and error-checking data
- *Internet protocol.* This layer receives datagrams from the TCP layer and breaks the packets down further. An IP packet contains a header with an address and carries TCP information and data in the body of the packet. The IP layer routes the individual packets from the sender to the receiver

- *Network.* This handles addressing issues, usually within the operating system, as well as providing an interface between the computer and the network. Each device on a network will normally have a unique ID (an IP number) assigned to it – represented in the network interface of each device
- *Physical.* This defines the basic characteristics of signal transmission along communication networks

Two different computer systems using TCP/IP are able to communicate with each other even though they may be based on different hardware and software platforms. Data sent from one computer passes down through the five layers. Once the data reaches the receiving computer it travels up through the layers. If the receiving computer finds a damaged packet it requests the sending computer to send again. This process is illustrated in Figure 43.1.

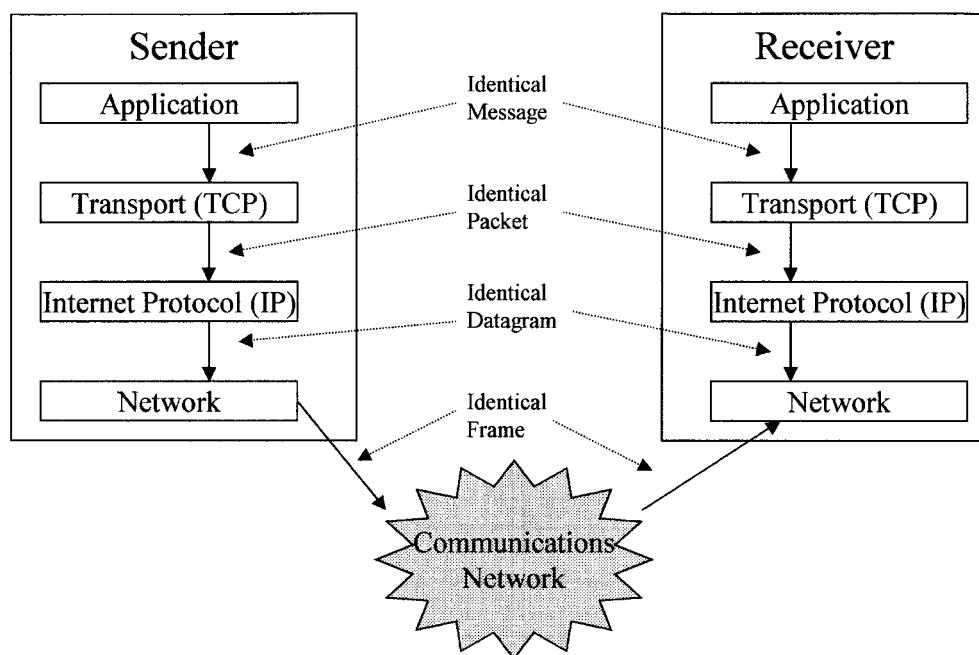


Figure 43.1 TCP/IP layers.

43.5 HYPertext Transfer Protocol (HTTP)

HTTP is an object-oriented protocol that defines how information can be transmitted between clients and servers. A HTTP transaction consists of the following phases:

- *Connection.* The client establishes a connection with a Web server
- *Request.* The client sends a request message to a Web server



- *Response.* The Web server sends a response to the client
- *Close.* The connection is closed by the Web server

HTTP is said to be a stateless protocol. This means that when a server provides a response and the connection is closed, the server has no memory of any previous transactions. This has the advantage of simplicity in that clients and servers can run with simple logic and there is little need for extra memory.

43.6



An IP address is the fundamental way of identifying uniquely a computer system on the Internet. An IP address is constructed as a series of up to four numbers, each delimited by a period. It is hence called a dotted quad. In a 32-bit IP address, each of the four numbers can range from 0 to 255. Generally the first of the four numbers identifies a computer network. The remaining numbers usually identify a node on this network.

Because of the explosion in Internet usage, more computers are being added to the global network. Hence, the 32-bit IP address will eventually run out of unique addresses. To offset this, a 128-bit IP address will be introduced globally.

Example

126.203.97.54 may be an IP address for a computer on the local area network at my university.

43.7



Internet users generally find IP addresses difficult to remember. Hence more mnemonic identifiers have been introduced which map to IP addresses.

A computer attached to the Internet and the HTML (hypertext markup language) documents resident on these computers are identified by universal resource locators (URLs). URLs can thus be used to provide a unique address for each document on the Web. Links between documents are activated by 'hotspots' in the document: a word, phrase or image used to reference a link to another document.

The syntax of a URL consists of at least two and as many as four parts. A simple two-part URL consists of:

- The protocol used for the connection (such as HTTP)
- The address at which a resource may be located on the host



Example ➔ In the URL below, the protocol – HTTP – is placed before the symbols `://`. The address after these symbols identifies a specific Web page on the host computer, in this case the home page of the University of Wales, Swansea:

`HTTP://www.swansea.ac.uk`

43.8 ⚡ DOMAIN NAMES

The ‘swansea’ in the URL in the previous example is short for ‘The University of Wales, Swansea’, the ‘ac’ for academic and the ‘uk’ for United Kingdom. This constitutes a so-called domain name, an agreed string of characters that may be used to provide some greater meaning to a URL. In practice, a domain name identifies and locates a host computer or service on the Internet. It often relates to the name of a business, organisation or service and must be registered in the same way as a company name.

A domain name is actually made up of three parts:

- *Subdomain*. This constitutes a provider of an Internet service. In this case it is University of Wales, Swansea
- *Domain type*. This suggests the type of provider. In this case it is ‘ac’ – indicating an academic institution based in the UK. The string ‘edu’ (short for education) is used more generally for an educational institution internationally
- *Country code*. Every country has its own specific code. For instance, ‘au’ is the code for Australia. If no country code is specified then the organisation is more than likely based in the USA

43.9 ⚡ COMPONENTS OF THE WWW

The World-Wide-Web, or Web for short (Berners-Lee, 1999) is the application (or series of applications) which is associated most readily with the Internet at the current time.

Figure 43.2 illustrates the primary components of the Web:

- Hypertext/Hypermedia
- HTML
- Web browsers

43.10 ⚡ HYPERTEXT/HYPERMEDIA

Vannevar Bush (1945) envisaged hypertext/hypermedia systems in the 1940s. In a ground-breaking paper Bush discussed the concept of a memex (memory

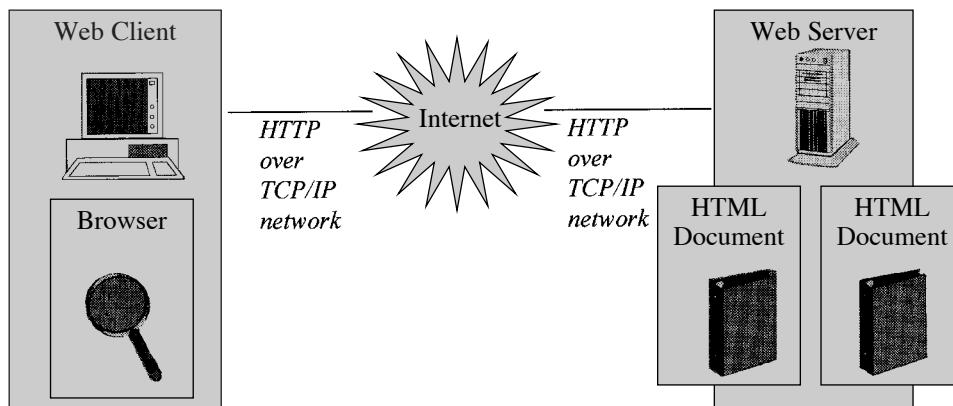


Figure 43.2 Components of the World-Wide-Web.

extender), a device capable of storing and retrieving information on the basis of content. In 1968, Douglas Englebart demonstrated the Augment system. Augment was an on-line working environment designed to augment the human intellect. It could be used to store and retrieve memos, research notes and other forms of documentation. Ted Nelson extended Bush's original idea in his Xanadu environment. Xanadu was designed to be an ever-expanding work-space that could be used to create and interconnect documents containing text, video, audio and graphics. Nelson actually coined the term hypertext and described it as being 'non-linear reading or writing.' A number of prominent hypermedia prototypes were developed during the 1980s. However, a software tool bundled with the Apple Macintosh computer did most to popularise the concept. In recent times hypertext and hypermedia form the bedrock for the Web.

Text can normally be organised in three major ways:

- *Linear text.* Linear text is exemplified in the format of the conventional novel. The reader is expected to start at the beginning and progress to the conclusion via a sequential reading of chapters
- *Hierarchical text.* Most textbooks and reports are organised hierarchically in terms of chapters, sections, subsections etc. The reader is able to access the text at various points in the hierarchy
- *Network text.* The dictionary or the encyclopaedia exemplifies network text. In a dictionary, each entry has an independent existence but is linked to a number of other entries via references

Hypertext is an electronic or on-line version of network text. A hypertext document is made up of a number of textual chunks connected together with associative links called hyperlinks. Hypermedia is a superset of hypertext. Here the nodes of the network may be various media, including text, graphics, audio and video.

43.11 (c) HYPERTEXT MARKUP LANGUAGE (HTML)

The WWW can be thought of as a collection of documents residing on thousands of servers or Web-sites around the world. Each such document contains content such as text and a set of embedded tags that indicate how the content is to be presented. This process of tagging text with extra presentational information is known as marking-up, and the set of tags for doing this a markup language. In the 1960s work began on developing a generalised markup language for describing the formatting of electronic documents. This work became established in a standard known as the standard generalised markup language, or SGML.

SGML in fact constitutes a meta-language – a language for defining other languages. Hence, SGML can be used to define a large set of markup languages. Tim Berners-Lee used SGML to define a specific language for hypertext documents known as hypertext markup language (HTML). HTML is a standard for marking-up or tagging documents that can be published on the Web, and can be made up of text, graphics, images, audio-clips and video-clips. Documents also include links to other documents either stored on the local HTML server or on remote HTML servers. HTML documents are also referred to as 'pages'.

Example → Below we include a very simple document expressed in HTML:

```
<HTML>
<TITLE>Information Systems</TITLE>
<H1>Information Systems</H1>
<H2>Paul Beynon-Davies</H2>
</HTML>
```

The text between angled brackets constitutes tags. Each piece of text is preceded by a start-tag and succeeded by an end-tag. A forward slash precedes an end-tag. The *HTML* tags indicate the start and end of the document. The tag *Title* provides a name for the page. The tags *H1* and *H2* indicate that a first-level and second-level heading should be displayed respectively.

A HTML document contains both content and tags. The document content consists of what is displayed on the computer screen. The tags constitute codes that tell the browser how to format and present the content on the screen.

The general form of this relationship between tags and content is

```
<tagname properties> content </tagname>
```

The tagname is taken from an established set of keywords established in the particular version of HTML. Certain tagnames have associated with them a



number of properties which serve to refine the meaning of a tag to a browser.

Examples ➔ In the tag <P align="right">, P is the tagname and acts as an abbreviation for the word paragraph. Consequently this tag is designed to be placed at the start of a chunk or paragraph of text. The word *align* is a property which can be assigned a number of values from a limited list. Here the value *right* specifies that the paragraph in question should be right-justified on the screen. An end-tag </P> will be placed at the end of the chunk of text.

The hyperlinks between pieces of text are established using anchor tags. The link can be to a textual element in the same document or to another document. This anchor tag has the form:

```
<A HREF="address">visible link text</A>
```

The letter A stands for anchor. HREF is a property which is used to specify the address of the document or piece of text to be linked to. The visible link text establishes what is displayed on the screen as a so-called hotspot. When you move the cursor over a hotspot, the cursor changes from an arrow to a pointing hand. This indicates that a click on the hotspot will transfer you to the text specified in the address.

The following tag embedded in a HTML document will establish a hyperlink to the author's current place of work:

```
<A HREF="http://www.swan.ac.uk">The University of Wales, Swansea</A>
```



To access the Web one needs a browser. This is essentially a program that lets the user read Web documents, view any in-built images or activate other media and hotspots. After the invention of the concept of the Web, the idea became established quickly in the scientific community. However, few people outside this community had software capable of reading HTML documents. In 1993 the first program that could read HTML documents and display them on a graphical user interface was written at the University of Illinois. It became known as Mosaic.

The commercial opportunities afforded by browsers soon became apparent and members of the Illinois team joined with one other to form the company Netscape Communications. Their key product, Netscape Navigator, became an immediate success. Microsoft soon entered the market with its Internet Explorer product and these two browsers still dominate this niche in the software market.

43.13 (c) THE PLACE OF DATABASE SYSTEMS IN INTERNET APPLICATIONS

Many contemporary Web-sites are repositories for files. In this case each Web document is stored in its own file. This is perfectly satisfactory for small Web-sites. As the amount of information provided on a Web-site grows, this file-based model proves difficult to manage.

For example, it is difficult to keep track of information contained in hundreds or thousands of separate Web documents. If changes are made to the structure of information contained in documents, such changes may affect hundreds of links from other documents.

Many organisations are now looking to Web-sites to make connections to their customers and suppliers. In terms of customers, the organisation may wish to provide a complete list of its products. In terms of suppliers, the organisation may wish to provide access to its production data. To support such applications, connections need to be made from Web servers to data contained in corporate databases.

The use of database systems to store both content and link information of relevance to Web information has therefore become a significant force over the last few years. We generally refer to the use of databases in this context as networked database applications.

43.14 (c) TYPES OF WEB-ENABLED DATABASE APPLICATIONS

A Web-enabled database application is a database application in which clients use a HTML-based browser and the server includes both an Internet server, a DBMS and a database. Several different types of such applications are emerging. Many of these are referred to as publishing applications because of the way in which Internet applications are organised around requests to an information source. We can distinguish between three main ways in which a database system may interact with Web-based services:

43.14.1 STATIC REPORT PUBLISHING

In this type of Web-enabled database application the DBMS generates a report, static form (a display only form), or response to a query in HTML format, and posts this onto the Web-site. The traffic in this type of application is one-way, from the server to the client. The user provides no data to the application.

43.14.2 QUERY PUBLISHING

In this type of Web-enabled database application a HTML form is generated containing text boxes in which the user may enter criteria for a query. Once



the form is submitted, a request is made to the DBMS which returns matching data or an error.

43.14.3 APPLICATION PUBLISHING

In this type of Web-enabled database application the interfaces (both data entry and reports) are all Web based. This is clearly a Web-based emulation of the traditional information technology system architecture. However, such applications currently suffer from three major problems:

- Internet applications usually suffer from low data transmission rates. This means that only applications having limited demands on data transmission can generally be shared
- Internet security currently suffers from limited security. Developments are occurring in areas like encryption but as yet many organisations are unhappy in using Internet technology to handle commercially sensitive data
- The stateless nature of Internet transactions means that the client needs to maintain some record of the status of a database session. This means that client applications generally have to be large and quite complex

Each of the networked database applications described above demand the use of dynamic Web pages. Traditionally, a HTML file stored as a document is an example of a static Web page. The content of the page only changes if a change is made to the document. In a dynamic Web page, the content is dynamically generated each time the page is accessed.

43.15



APPROACHES TO BUILDING WEB-ENABLED DATABASE APPLICATIONS

There are a number of approaches to building Web-enabled database applications including:

- *Common gateway interface (CGI)*. This constitutes a specification for how scripts communicate with Web servers. Scripts are programs stored on the Web server which are executed by the server, and the results of the execution are returned to the browser. Parameters can be passed to the script using the query string part of a URL
- *Server-side includes*. Normally Web servers do not examine the files that they send to browsers. Some servers are able to examine the files for so-called server-side includes. Such includes may command the server to execute some program and include the results within the document before returning it to the browser
- *JDBC, JSQL*. Java can be used in association with a number of defined application programming interfaces (APIs). For instance, Java Database

Connectivity (JDBC) is modelled on the Open Database Connectivity (ODBC) API. JSON is an extension to the embedded SQL standard proposed for use with Java

- *Scripting languages*, such as Javascript and Vbscript. Scripting languages enable the specification of functions within HTML documents. Javascript is an object-based scripting language originally developed by Netscape and Sun. Vbscript is a Microsoft scripting language whose functionality is similar to Javascript but which more closely resembles Microsoft's Visual Basic in terms of syntax
- *Vendor initiatives*, such as Microsoft's Active Server Pages. Active Server Pages (ASP) is a Microsoft-specific model for building Web pages on Web servers. An ASP can contain a combination of text, HTML tags, and script commands and output expressions. ASP files are requested using the '.asp' extension from the server. The Web server reads through the requested file, executes any commands and returns the dynamic HTML page to the browser

43.16 CASE STUDY: USING CGI

As indicated above, a Web server is a program running on a server machine that accepts requests from a Web browser and returns the results in the form of HTML documents. The browser and server communicate using the HTTP protocol. HTTP provides a number of features over and above the simple transfer of documents. One of the most important of these is the ability to execute programs with arguments supplied by the user, and deliver the results back in the form of HTML documents. When a Web server recognises a URL as pointing to a file, it returns the contents of the file. In contrast, when a URL points to a program, it executes the script and returns the output of the program back to the browser as a document.

This enables a Web server to act as an intermediary in an *N*-tier architecture (Chapter 36) between Web browser and application programs. The common gateway interface (CGI) standard defines how a Web server may communicate with application programs. In turn, the application programs will communicate with a database server through ODBC (Open Database Connectivity), JDBC or other protocols.

Consider the case of a customer wishing to order an item on an e-Commerce site. The customer navigates through the Web-site to an on-line order form. After completing the form, the customer clicks on a submit button. This initiates a request to the Web server which it interprets as a request to run an application program. This program (or programs) is likely to conduct a series of actions such as:

- Checking that the data on the order is correct
- Initiating a request to a database to check stock
- Initiating a request to another database to check customer details



Depending on the result of each of these actions, various responses are returned to the browser. For instance, if some data is missing, an error message is likely to be returned and the customer requested to correct the details.

43.17**SUMMARY**

- The Internet is a set of interconnected computer networks distributed around the globe
- The infrastructure of the Internet is built upon packet-switched networks, TCP/IP, HTTP, IP addresses, URLs and domain names
- The facilities provided by the World-Wide-Web are the ones most readily associated with the concept of the Internet at the current time
- The primary components of the Web comprise: hypertext transfer protocol (HTTP), hypertext markup language (HTML), universal resource locators and browsers
- Database systems are important in the context of managing large-scale Web applications through the use of dynamic pages
- Three types of applications for the use of database systems within Web applications are report publishing, query publishing and application publishing
- Different approaches for building networked database applications include: CGI, JDBC and Active Server Pages

43.18**ACTIVITIES**

1. Investigate the process by which domain names are assigned globally.
2. Investigate some of the other common tags used in a HTML document.
3. Identify three applications in a university setting corresponding to the three types of application for database systems within Web applications.
4. Determine the most popular approach for connecting Web servers to database servers.

43.19

Berners-Lee, T. (1999). *Weaving the Web: The Past, Present and Future of the World Wide Web by its Inventor*. London, Orion Business Publishing.

Bush, V. (1945). As we may think. *Atlantic Monthly* 176: 101–3.



BIBLIOGRAPHY

- Abbey, M., M. Corey and I. Abramson (2002). *Oracle9i: A Beginner's Guide*. London, McGraw-Hill.
- Anahory, S. and D. Murray (1997). *Data Warehousing in the Real World: A Practical Guide for Building Decision-Support Systems*. Harlow, UK, Addison-Wesley.
- ANSI (1975). ANSI/X3/SPARC Study Group on Data Base Management Systems: Interim Report, FDT. *ACM SIGMOD Bulletin* **7**(2).
- Apte, C., B. Liu, E.P.D. Pednault and P. Smyth (2002). Business applications of data mining. *CACM* **45**(8): 49–53.
- Atkinson, M., D. DeWitt, D. Maier, F. Bancilhon, K. Dittrich and S. Zdonik (1990). The object-oriented database system manifesto. In W. Kim, J.-M. Nicolas and S. Nishio (Eds), *Deductive and Object-Oriented Databases*. North-Holland, Elsevier Science.
- Avison, D.E. (1991). *Information Systems Development: A Database Approach*. Oxford, Alfred Waller.
- Bancilhon, F., C. Delobel and P. Kanellakis (Eds) (1992). *Building an Object-Oriented Database System: The Story of O₂*. San Mateo, CA, Morgan Kaufmann.
- Berners-Lee, T. (1999). *Weaving the Web: The Past, Present and Future of the World Wide Web by its Inventor*. London, Orion Business Publishing.
- Berson, A. and S.J. Smith (1997). *Data Warehousing, Data Mining and OLAP*. New York, McGraw-Hill.
- Beynon-Davies, P. (1992). The realities of database design: the sociology, semiology and pedagogy of database work. *Journal of Information Systems* **2**: 207–20.
- Beynon-Davies, P. (1994). Information management in the British National Health Service: the pragmatics of strategic data planning. *International Journal of Information Management* **14**(2 (April)): 84–94.
- Beynon-Davies, P. (1997). The corporate data model: a study of organisational practice. *Journal of Systems and Information Technology* **1**(1 (March)): 47–63.
- Beynon-Davies, P. (1998). *Information Systems Development: An Introduction to Information Systems Engineering*. Basingstoke, Macmillan (now Palgrave).
- Beynon-Davies, P. (2002). *Information Systems: An Introduction to Informatics in Organisations*. Basingstoke, Palgrave (formerly Macmillan).
- Beynon-Davies, P. (2003). *e-Business*. Basingstoke, Palgrave Macmillan.
- Blaha, M. and W. Premerlani (1997). *Object-Oriented Modelling and Design for Database Applications*. Englewood Cliffs, NJ, Prentice-Hall.
- Blaha, M.R., W.J. Premerlani and J.E. Rumbaugh (1988). Relational database design using an object-oriented methodology. *Comm. of ACM* **31**(4): 414–27.
- Bobrowski, S.M. (1998). *Oracle 8 Architecture*. Berkeley, CA, McGraw-Hill.
- Booch, G., J. Rumbaugh and I. Jacobson (1999). *The Unified Modelling Language User Guide*. Reading, MA, Addison-Wesley.
- Brachman, R.J. (1983). What ISA is and isn't: and analysis of taxonomic links in semantic networks. *Computer* **16**(10): 30–6.



- Braithwaite, K.S. (1992). *Information Engineering: Analysis and Administration*. London, CRC Press.
- Brodie, M.L., J. Mylopoulos and T.W. Schmidt (Eds) (1984). *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*. Berlin, Springer-Verlag.
- Bush, V. (1945). As we may think. *Atlantic Monthly*. 176: 101–3.
- Cabena, P., P. Hadjinian, R. Stadler, J. Verhees and A. Zanasi (1997). *Discovering Data Mining from Concept to Implementation*. Englewood Cliffs, NJ, Prentice-Hall.
- Cattell, R.G.G. (1994). *The Object Database Standard: ODMG-93: Release 1.1*. San Mateo, CA, Morgan Kaufmann.
- Cattell, R.G.G. (Ed.) (2000). *The Object Database Standard: ODMG Release 3.0*. San Mateo, CA, Morgan Kaufmann.
- CCTA (1994). *Corporate Data Modelling*. London, HMSO.
- Ceri, S. and G. Peligatti (1984). *Distributed Database Principles and Systems*. New York, McGraw-Hill.
- Checkland, P. (1987). *Systems Thinking, Systems Practice*. Chichester, John Wiley.
- Chen, P.P.S. (1976). The entity-relationship model: towards a unified view of data. *ACM Trans. on Database Systems* 1(1): 9–36.
- Clark, K.L. (1978). Negation as failure. In H. Gallaire and G. Minker (Eds), *Logic and Databases*. Oxford: OUP.
- CODASYL (1971). *Database Task Group Report*. New York, ACM.
- Codd, E.F. (1970). A relational model for large shared data banks. *Comm. of ACM* 13(1): 377–87.
- Codd, E.F. (1974). Recent investigations into relational database systems. *Proc. IFIP Congress*.
- Codd, E.F. (1979). Extending the relational database model to capture more meaning. *ACM Trans. on Database Systems* 4(4): 397–434.
- Codd, E.F. (1982). Relational database: a practical foundation for productivity. *Comm.of ACM* 25(2).
- Codd, E.F. (1985). Is your DBMS really relational? *Computerworld* 1–9.
- Codd, E.F. (1985). Does your DBMS run by the rules? *Computerworld* 49–64.
- Codd, E.F. (1988). Fatal flaws in SQL. *Datamation* August: 45–8.
- Codd, E.F. (1988). Fatal flaws in SQL: part 2. *Datamation*. Sept.: 71–4.
- Codd, E.F. (1990). *The Relational Model for Database Management: Version 2*. Reading, MA, Addison-Wesley.
- Codd, E.F., S.B. Codd and C.T. Salley (1993). *Providing OLAP (On-Line Analytical Processing) to User-Analysts: An IT Mandate*. Ann Arbor, MI, Arbor Software Corporation.
- Comer, D. (1979). The ubiquitous B-tree. *ACM Computing Surveys* 11(2): 121–38.
- Conklin, E.J. (1987). Hypertext: an introduction and survey. *IEEE Computer* 2(9): 17–41.
- Connolly, T.M. and C.E. Begg (2002). *Database Systems: A Practical Approach to Design, Implementation, and Management*. Harlow, UK, Addison-Wesley.
- Dahlbohm, B. and L. Mathiassen (1993). *Computers in Context*. London, NCC/Blackwell.
- Date, C.J. (1987). Twelve rules for a distributed database. *Computerworld* 2–5.
- Date, C.J. (1987). Where SQL falls short. *Datamation* May: 83–6.
- Date, C.J. (1997). *A Guide to the SQL Standard: A User's Guide to the Standard Database Language*. Reading, MA, Addison-Wesley.
- Date, C.J. (2000). *An Introduction to Database Systems*. Reading, MA, Addison-Wesley.
- DBTG (1971). *Report of the Codasyl Database Task Group*, ACM.
- DeWitt, D. and J. Grey (1992). Parallel database systems: the future of high performance database systems. *CACM* 35(6).
- Eaglestone, B. and M. Ridley (1998). *Object Databases: An Introduction*. Maidenhead, Berkshire, UK, McGraw-Hill.
- Elmasri, R. and S.B. Navathe (2000). *Fundamentals of Database Systems*. Reading, MA, Addison-Wesley.

- Fagin, R. (1977). Multi-valued dependencies and a new normal form for relational databases. *ACM Trans. on Database Systems* **2**(1).
- Fagin, R. (1979). Normal forms and relational database operators. *ACM SIGMOD Int. Symposium on the Management of Data* 153–60.
- Feldman, P. and D. Miller (1986). Entity model clustering: structuring a data model by abstraction. *Computer Journal* **29**(4).
- Ferguson, M. (1993). Parallel query and transaction processing with relational databases. *UKOUG Conference*, Bournemouth, UK.
- Frost, R.A. (1982). Binary-relational storage structures. *Computer Journal* **25**(3): 358–67.
- Frost, R.A. (1983). A step towards the automatic maintenance of the semantic integrity of databases. *Computer Journal* **26**(2): 124–33.
- Gallaire, H. and G. Minker (1978). *Logic and Databases*, New York, Plenum Press.
- Gallaire, H., J. Minker and J.-M. Nicolas (1984). Logic and databases: a deductive approach. *Computing Surveys* **16**(2): 153–85.
- Garcia-Molina, H., J.D. Ullman and J.D. Widom (2002). *Database Systems: The Complete Book*. Harlow, UK, Prentice-Hall.
- Gardarin, G. and P. Valduriez (1989). *Relational Databases and Knowledge Bases*. Reading, MA, Addison-Wesley.
- Gatrell, A.C. (1991). Concepts of space and geographical data. In D.J. Maguire, M.F. Goodchild and D.W. Rhind (Eds), *Geographical Information Systems: Principles*. Harlow, UK, Longman Scientific.
- Gennick, J. (2000). *Sams Teach Yourself PL/SQL in 21 Days*. New York, Sams Publishing.
- Gillenson, M.L. (1987). The duality of database structures and design techniques. *Comm. of ACM* **30**(12): 1056–65.
- Gillenson, M.L. (1991). Database administration at the crossroads: the era of end-user-oriented, decentralised data processing. *Journal of Database Administration* **2**(4): 1–11.
- Goodhue, D.L., L.J. Kirsch, J.A. Quilliard and M.D. Wybo (1992). Strategic data planning: lessons from the field. *MIS Quarterly* March: 11–34.
- Goodhue, D.L., M.D. Wybo and L.J. Kirsch (1992). The impact of data integration on the costs and benefits of information systems. *MIS Quarterly* September: 293–311.
- Graham, I. (1989). Incremental development: review of nonmonolithic life-cycle development models. *Information and Software Technology* **31**(1): 7–20.
- Grant, J. and J. Minker (1992). The impact of logic programming on databases. *Comm of ACM* **35**(3): 66–81.
- Gray, J. (1996). The evolution of data management. *IEEE Computer* 38–46.
- Hammer, M. and D. McCleod (1981). Database description with SDM: a semantic database model. *ACM Trans on Database Systems* **6**(3): 351–86.
- Harel, D. (1988). On visual formalisms. *Comm. of ACM* **31**(5): 514–29.
- Healey, R.G. (1991). Database management systems. In D.J. Maguire, M.F. Goodchild and D.W. Rhind (Eds), *Geographical Information Systems: Principles*. Harlow, UK, Longman Scientific.
- Howe, D. (1981). *Data Analysis for Database Design*. London, Edward Arnold.
- Hudson, D.L. (1991). Approaches to corporate data model development at Sara Lee Corp. *Data Resource Management* **2**(3): 49–55.
- Hull, R. and R. King (1987). Semantic database models: survey, applications and research issues. *ACM Computing Surveys* **19**(3): 201–60.
- Inmon, W.H. (2000). *Building the Data Warehouse*. New York, John Wiley.
- Inmon, W.H., J.D. Welch and K.L. Glassey (1997). *Managing the Data Warehouse*. New York, John Wiley.
- ISO (1987). *Database Language SQL*. ISO/IEC 9075:1987. Geneva, International Standards Organisation.
- ISO (1989). *Database Language SQL with Integrity Enhancement*. ISO/IEC 9075:1989. Geneva, International Standards Organisation.



- ISO (1992). *Database Language SQL*. ISO/IEC 9075:1992. Geneva, International Standards Organisation.
- ISO (1999). *Database Language SQL – Part 2: Foundation*. ISO/IEC 9075–2. Geneva, International Standards Organisation.
- ISO (1999). *Database Language SQL – Part 2: Persistent Stored Modules*. ISO/IEC 9075–4. Geneva, International Standards Organisation.
- Jacobson, I., G. Booch and I. Rumbaugh (1999). *The Unified Software Development Process*. Reading, MA, Addison-Wesley.
- Jarke, M. and J. Koch (1984). Query optimisation in database systems. *ACM Computing Surveys* **16**: 111–52.
- Jones, C.B. and L.Q. Luo (1994). Hierarchies and objects in a deductive spatial database. *6th Int. Symposium on Spatial Data Handling*, Edinburgh, 588–603.
- Kahavi, R., N.J. Rothlededer and E. Simoudis (2002). Emerging trends in business analytics. *Communications of the ACM* **45**(8): 45–53.
- Kantardzic, M. (2002). *Data Mining: Concepts, Models, Methods and Algorithms*. Chichester, John Wiley.
- Kent, W. (1978). *Data and Reality*. Amsterdam, North-Holland.
- Kent, W. (1983). A simple guide to five normal forms in relational database theory. *Comm. of ACM* **26**(2).
- King, R. (1988). My cat is object-oriented. In W. Kim and F. Lochofsky (Eds), *Object-Oriented Languages, Applications and Databases*. Reading, MA, Addison-Wesley.
- Klein, H.K. and R.A. Hirschheim (1987). A comparative framework of data modelling paradigms and approaches. *Computer Journal* **30**(1): 8–14.
- KPMG (2002). *Is Britain on Course for 2005?* London: KPMG Consulting.
- Laurini, R. and D. Thompson (1992). *Fundamentals of Spatial Information Systems*. San Diego, CA, Academic Press.
- Loney, K. and M. Theriault (2002). *Oracle9i DBA Handbook*. New York, McGraw-Hill.
- Maguire, D.J. (1991). An overview and definition of GIS. In D.J. Maguire, M.F. Goodchild and D.W. Rhind (Eds), *Geographical Information Systems: Principles*. Harlow, UK, Longman Scientific.
- Marakas, G.M. (2003). *Modern Data Warehousing, Mining, and Visualization: Core Concepts*. Upper Saddle River, NJ, Prentice-Hall.
- Martin, J. and C. McClure (1985). *Diagramming Techniques for Analysts and Programmers*. Englewood Cliffs, NJ, Prentice-Hall.
- Melton, J. and A.R. Simon (1993). *Understanding the New SQL: A Complete Guide*. New York, Morgan Kaufmann.
- Melton, J. and A.R. Simon (2002). *SQL: 1999: Understanding Relational Language Components*. San Francisco, CA, Morgan Kaufmann.
- Microcosm (1994). *Microcosm: A Technical Overview*. University of Southampton, UK.
- Navathe, S.B. and L. Kerschberg (1986). Role of data dictionaries in information resource management. *Information and Management* **10**: 21–46.
- Newell, A. and H.A. Simon (1976). Computer science as empirical inquiry: symbols and search. *Comm of ACM* **19**(3).
- O'Leary, T. (2002). *Microsoft Access 2002*. New York, McGraw-Hill.
- Oracle (1999). *Oracle 8i Administrator's Guide*. Oracle Corporation.
- Oracle (2000). *Oracle Internet Application Server 8i: Overview Guide*. Oracle Corporation.
- Ovum (1988). *The Future of Databases*. London, Ovum Press.
- Özsu, M.T. and P. Valduriez (1991). *Principles of Distributed Database Systems*. Englewood Cliffs, NJ, Prentice-Hall.
- Özsu, M.T. and P. Valduriez (1992). Distributed database systems: where are we now? *Database Programming and Design* **42**(1).
- Porter, M.E. (1985). *Competitive Advantage: Creating and Sustaining Superior Performance*. New York, Free Press.

- Rada, R. (1991). *Hypertext: From Text to Expertext*. London, McGraw-Hill.
- Ramakrishnan, R. and J. Gehrke (2003). *Database Management Systems*. Boston, MA, McGraw-Hill.
- Reiter, R. (1978). On closed world databases. In H. Gallaire and J. Minker (Eds), *Logic and Databases*. Oxford: OUP.
- Reiter, R. (1984). Towards a logical reconstruction of relational database theory. In M.L. Brodie, J. Mylopoulos and J.W. Schmidt (Eds), *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*. New York, Springer-Verlag.
- Rich, E. (1984). Natural-language interfaces. *IEE Computer* **17**(9): 39–47.
- Robinson, K.A. (1979). The entity/event modelling method. *Computer Journal* **22**: 270–81.
- Rodgers, U. (1989). Denormalisation: why, what and how? *Database Programming and Design* **2**(12): 46–53.
- Rolland, F.D. (1998). *The Essence of Databases*. Hemel Hempstead, UK, Prentice-Hall.
- Rosenquist, C.J. (1982). Entity life-cycle models and their applicability in information systems development. *Computer Journal* **25**: 307–15.
- Schneiderman, B. (1998). *Designing the User Interface: Strategies for Effective Human–Computer Interaction*. Reading, MA, Addison-Wesley.
- Shave, M.J.R. (1981). Entities, functions and binary relations: steps to a conceptual schema. *Computer Journal* **24**(1).
- Sikora, Z.M. (1997). *Oracle Database Principles*. London, Macmillan (now Palgrave).
- Silberschatz, A., H.F. Korth and S. Sudarshan (2002). *Database System Concepts*. Boston, MA, McGraw-Hill.
- Smith, H.C. (1985). Database design: composing fully normalised tables from a rigorous dependency diagram. *Comm. of ACM* **28**(8).
- Smith, J.M. and D.C.P. Smith (1977). Database abstractions: aggregation and generalisation. *Transactions of Database Systems* **2**(2).
- Sowa, J.F. (1984). *Conceptual Structures: Information Processing in Mind and Machine*. Reading, MA, Addison-Wesley.
- Stamper, R.K. (1973). *Information in Business and Administrative Systems*. London, Batsford.
- Stonebraker, M. (1996). *Object-Relational DBMSs: The Next Great Wave*. San Francisco, CA, Morgan Kauffman.
- Stonebraker, M. and J.M. Hellerstein (Eds) (1998). *Readings in Database Systems*. San Francisco, CA, Morgan Kaufmann.
- Sturner, G. (1995). *Oracle7: A User's and Developer's Guide*. London, International Thompson Publishing.
- Sussman, R. (1993). Municipal GIS and the enterprise data model. *Int. Journal of Geographical Information Systems* **7**(4): 367–77.
- Sweet, F. (1985). A 14-part series on process-driven database design. *Datamation* Jan.–Nov.
- Teorey, T.J. (1994). *Database Modelling and Design: The Fundamental Principles*. San Mateo, CA, Morgan Kaufmann.
- Tsitchizris, D.C. and F.H. Lochovsky (1982). *Data Models*. Englewood Cliffs, NJ, Prentice-Hall.
- W3C (1999). *HTML 4.01*. World-Wide-Web Consortium.
- W3C (2000). *XML 1.0 2nd Edition*. World-Wide-Web Consortium.
- Weaver, P.L., N. Lambrou and M. Walkley (1998). *Practical SSADM 4+*. London, Pitman.
- Winograd, T. and F. Flores (1986). *Understanding Computers and Cognition: A New Foundation for Design*. Norwood, NJ, Ablex Publishing.
- Winston, P.H. (1984). *Artificial Intelligence*. Reading, MA, Addison-Wesley.
- Winston, M., R. Chaffin and D. Herrmann (1987). A taxonomy of part–whole relations. *Cognitive Science* **11**: 417–44.
- Witten, I.H. and E. Frank (2000). *Data Mining*. San Francisco, CA, Morgan Kaufmann.
- Zloof, M.M. (1975). Query by example. *Proc. NCC* **44**(May).



GLOSSARY AND INDEX

Term	Definition	Pages
A		
Aborted Transaction	A <i>transaction</i> that does not complete successfully. No permanent changes are made to a database by an aborted transaction	405
Abstract Data Type (ADT)	An abstract data type or ADT is a type of <i>object</i> that defines a <i>domain</i> of values and a set of operations designed to work with these values	145–6
Abstract Machine	A model of the key features of some system without any implementation detail	4
Abstraction	The process of modelling ‘real-world’ concepts in a computational medium	24–5
Abstraction Mechanism	A mechanism for constructing hierarchies of <i>object classes</i>	117, 250
Access (Microsoft DBMS)	A popular <i>DBMS</i> originally produced for the desktop market by Microsoft	426, 446–56
Access Mechanism	An algorithm embodied in software established for accessing data	82, 313–15, 395–402
Accommodation	The process of producing a <i>logical model</i> from a <i>conceptual model</i>	232, 236–8, 255, 260, 280–3
Active Data Dictionary	That part of a <i>database system</i> that contains <i>meta-data</i> and is used to control <i>DBMS</i> operations	339
Active Database System	A <i>database system</i> with embedded <i>update functions</i>	148
Additional Integrity	The business rules relevant to some application domain that cannot be specified using the <i>inherent integrity</i> mechanisms of some <i>data model</i>	109–10, 121–2, 304–5, 460
After-Image Log	A history of the new or modified values that result when transactions are applied to a database	414, 416
Aggregate Functions (SQL)	A set of <i>SQL</i> functions that can operate over aggregations of data – Max, Min, Count, Avg	182



Term	Definition	Pages
Aggregation	An <i>abstraction mechanism</i> or process by which an object is used to group together a number of other objects	117, 255–60
AKO (A-Kind-Of)	AKO relates <i>object class</i> to object class. See also <i>Generalisation</i>	252
ANSI/SPARC Architecture	In 1975 the American National Standards Institute Standards Planning and Requirements Committee (ANSI-SPARC) proposed a three-level architecture for DBMS	77–8
API	See <i>Application Programming Interface</i>	
Application Data Model	A blueprint of data requirements	37, 55, 324
Application Development Toolkit	A set of tools for the application developer to build information systems in association with a database system	365–74
Application Programming Interface	A defined interface for connecting application programs to DBMS	371–2
Archiving	The process of moving unused data off a database into some form of storage	378–9
Armstrong's Axioms	A set of rules which can be used to compute new <i>functional dependencies</i> from existing dependencies	280
Assert	An in-built function within <i>Datalog</i>	129
Assertion	Some statement about the characteristics of some <i>UoD</i> . A command in SQL2	127, 174
Association	An <i>abstraction mechanism</i> . Instances of one <i>object</i> are linked to instances of another object	250
Atomicity	A property of a <i>transaction</i> . Atomicity ensures that a transaction either executes completely or is aborted	405
Attribute	A column in a <i>relation</i> . A property of an <i>object</i> or <i>entity</i>	5, 222, 441
Attribution	The process of relating properties to an <i>object</i>	256, 258
Authorisation	The facilities available for enforcing database security	80, 377–8
B		
B2B e-Commerce	The use of <i>e-Commerce</i> in the supply chain	65–6
B2C e-Commerce	The use of <i>e-Commerce</i> in the customer chain	66
Bachman, Charles	The creator of the Integrated Data Store (IDS) at General Electric – one of the first DBMS	40
Backing Store	See <i>Secondary Storage</i>	
Backup	The process of taking a copy of the whole or a part of some database for the purposes of <i>recovery</i>	344, 378–9 423
BCNF	See <i>Boyce–Codd Normal Form</i>	
Behavioural Abstraction	That part of an <i>object model</i> concerned with the specification of behaviour	249, 260–4



Term	Definition	Pages
Behavioural Inheritance	The <i>inheritance</i> of methods	117
Bit	An abbreviation of binary digit – one of the two digits (0 and 1) used in binary notation	384
BLOB	Binary Large Object. A byte stream used to store large objects in databases	433, 521
Block	A group of records treated as a unit on some secondary storage device. A term sometimes treated as a synonym for <i>page</i>	385
Bottom-up Data Analysis	The process of arriving at a logical schema via <i>normalisation</i>	43
Boyce–Codd Normal Form	A stronger normal form than <i>third normal form</i> . A BCNF schema is one in which every <i>functional determinant</i> is a <i>candidate key</i>	281, 284–5
Bracketing Notation	A shorthand notation for recording the structural detail associated with a relational schema	94–5
Browser	A program needed to access and read Web documents	562
B-Tree	A data structure used to build multi-level indexes	399–400
Business Area Data Model	A <i>data model</i> specified for some business area	55
Byte	A set of binary digits considered as a unit	384
C		
Call-level Interface	A type of <i>Application Programming Interface</i> in which subroutine calls to a DBMS library are embedded in some <i>Host Language</i>	371–2
Candidate Key	An attribute or group of attributes capable of acting as a <i>primary key</i>	92–3, 282
Cardinality	The number of attributes in a relation. The number of instances of an <i>entity</i> associated with some other entity.	92, 255, 226–7, 250–1
Chasm Trap	A <i>connection trap</i> . A misconception that the relationship between two entities can be generated by traversing other relationships in a <i>data model</i>	234–5
Checkpointing	Checkpointing is a technique used to limit the amount of searching that needs to be performed against a log file in order to carry out <i>recovery</i> effectively. Checkpointing involves force-writing database buffers to <i>secondary storage</i> at pre-determined intervals	416
Class	See <i>Object Class</i>	



Term	Definition	Pages
Classic Data Model	A term used to encapsulate the hierarchical, network and relational <i>data models</i> . Data models maintaining a fundamental record orientation	37
Classification	Classification involves grouping objects that share common characteristics into an <i>object class</i>	251
Client–Server	An applications architecture in which the processing is distributed between machines acting as clients and machines acting as servers	480–1
Closed World Assumption	The assumption that the only thing known about some <i>UoD</i> is represented in a database	6
Closure	A property of a <i>query language</i> in which the results from one query can be operated upon by another query	104
Clustered File	See <i>Clustering</i>	314, 390–1
Clustering	The process of storing logically related records close together on some <i>secondary storage</i> device	390–1
Clustering Index	An <i>index</i> used to help maintain the physical clustering of a stored file	391
CODASYL Data Model	An acronym for the Conference on Data Systems Languages, a committee that developed standards leading to the COBOL language. The Database Task Group (DBTG) sub-committee of CODASYL developed specifications on which the network data model is based. See <i>Network Data Model</i>	
Codd, E.F.	The creator of the relational data model	40, 270, 458
Codd's Rules	In the mid-1980s Codd proposed twelve rules for determining whether a DBMS was relational	458–60
Collection	An <i>ODMG</i> object model construct comprising aggregates of objects or literals	432, 439
Collection Type	An SQL3 construct allowing multiple values to be stored in columns and enabling <i>nested</i> tables	431–2
Column	See <i>Attribute</i>	
Command-line Interface	A user interface in which instructions are initiated through typed commands	368–9
Commit	A command used to make permanent in a database the results of some <i>transaction</i> . A keyword in <i>SQL</i>	405–6
Complex Data	A term used to refer to data subject to non-standard structure	512–23
Complex Object	An <i>object</i> with inherent hierarchical structure	136, 437
Compound Determinant	A <i>determinant</i> consisting of two or more attributes	279
Computer Aided Information Systems Engineering (CAISE)	The automation of information systems development	366–7



Term	Definition	Pages
Conceptual Data Dictionary	A <i>data dictionary</i> used to represent a <i>conceptual model</i>	338
Conceptual Level	See <i>Conceptual Schema</i>	
Conceptual Model	A model of the ‘real world’ expressed using constructs such as entities, relationships and attributes	42, 198
Conceptual Modelling	The process of developing a <i>conceptual model</i>	42, 198, 201
Conceptual Schema	A level of the <i>ANSI/SPARC architecture</i>	78
Concurrency	The process by which simultaneous access to a database is enabled for multiple users	15, 406–8
Concurrency Control	That facility provided by the <i>kernel</i> of a DBMS that enables concurrency	80, 408–13
Concurrent Transactions	Two or more transactions that overlap both in terms of time and the data they wish to access	406
Connection Trap	A set of design problems associated with E–R modelling. See also <i>Chasm Trap</i> , <i>Fan Trap</i>	232–5
Consistency	That property of a <i>transaction</i> which ensures that the changes caused by a transaction maintain the database as an accurate reflection of its real world domain	405
Constraint	See <i>Integrity Constraint</i>	
Corporate Data Model	A <i>data model</i> specified for an entire organisation	55, 324
Corporate Data Modelling	The process of developing a <i>corporate data model</i>	324–6
Correlated Subquery	A <i>subquery</i> that executes repeatedly, producing a series of values to be matched against the results of an outermost query	184
Cost-based Optimisation	See <i>Statistically-based Optimisation</i>	
Covering Subclass	Subclasses are covering if no other subclasses other than those specified are possible for a particular superclass	252–3
CPU	Central Processing Unit	499
CREATE INDEX	An <i>SQL</i> command for creating indexes on tables	400–1
CREATE TABLE	The main data definition command in <i>SQL</i>	161
CREATE VIEW	A standard command in <i>SQL</i> for constructing <i>view</i> definitions	349–50
CRUD functions	Create, Read, Update, Delete functions	79, 178
Cursor	A construct used to define both an <i>SQL</i> request and to iterate over rows in a result. A cursor is an embedded SQL construct used to solve the <i>impedance mismatch</i>	370
Customer Chain	The chain of activities that an organisation performs in the service of its customers	64, 66



Term	Definition	Pages
D		
Data	Sets of symbols	6, 19–28, 335
Data Abstraction	The process by which a database attempts to represent the properties of objects in the real world	35
Data Administration	Data administration is that function concerned with the management, planning and documentation of the data resource of some organisation	202, 334–42
Data Administrator	The person or persons tasked with <i>data administration</i> functions in some organisation	55, 58, 322, 335, 344
Data Analysis	The process involved in producing a <i>logical model</i> of some database system	43, 336
Data Analyst	A term frequently used to described the person or persons tasked with <i>requirements elicitation, conceptual modelling</i> and <i>logical database design</i>	336
Data Control	That part of a <i>data model</i> concerned with controlling access to databases and/or facilities of a DBMS	36, 305, 336, 345–53
Data Control Language	That part of a <i>database sublanguage</i> concerned with <i>data control</i>	81
Data Definition	That part of a <i>data model</i> concerned with defining data structures	25, 35, 36, 336
Data Definition Language (DDL)	That part of a <i>database sublanguage</i> concerned with <i>data definition</i>	81
Data Definition Language Compiler	That module of a DBMS concerned with updating the <i>system catalog</i>	83
Data Dictionary	A term used either to denote the <i>system tables</i> of some relational DBMS or to denote a more encompassing representation of the data used by some enterprise	79–81, 238–9, 328, 338–9, 419–20
Data Element	A logical collection of <i>data-items</i>	25
Data Format	A set of constraints defined on a data-item. See also <i>Data Type</i>	25
Data Independence	Buffering data from the processes that use such data	35, 372
Data Integration	A collection of data that displays no <i>redundancy</i>	34, 331–3
Data Integrity	That part of a <i>data model</i> concerned with specifying the business rules appropriate to some application domain	25, 34, 37, 80, 336, 379
Data Integrity Language (DIL)	That part of a <i>database sublanguage</i> concerned with <i>data integrity</i>	81
Data Management	The process of recording and manipulating data. Also used as a collective term for all the functions of a DBMS	38–41

Term	Definition	Pages
Data Management Layer	A layer of an <i>ICT System</i>	76–86
Data Manipulation	That part of a <i>data model</i> concerned with the manipulation of data in data structures	25, 36
Data Manipulation Language (DML)	That part of a <i>database sublanguage</i> concerned with <i>data manipulation</i>	81
Data Mart	A small <i>data warehouse</i>	70, 533–4
Data Mining	Data mining is the process of extracting previously unknown data from large databases and using it to make organisational decisions	70, 528, 547–53
Data Model	An architecture for data or a blueprint of data requirements for some application	14, 25, 36–8, 87–8
Data Modelling	See <i>Conceptual Modelling</i>	
Data Organisation	The way in which data is structured on physical storage devices	383–94
Data Security	The process of ensuring the security of data	34–5, 305, 336, 339–40, 379, 454–5
Data Sharing	The process of ensuring the sharing of data among two or more users or application systems	34, 336
Data Structure	An organised element of some database. A logical collection of <i>data elements</i>	25, 33
Data Subsystem	See <i>Data Management Layer</i>	53, 479
Data Type	A definition for the data appropriate to attributes or data-items	26, 161–3
Data Warehouse	A subject-oriented, integrated, time-variant and non-volatile collection of data used in support of management decision-making	70, 528–9
Data Warehousing	The process of building and managing data warehouses	527–38
Database	An organised pool of logically-related data	33
Database Administration	The activity of managing given database systems	202, 343–54
Database Administration Toolkit	The set of tools available to the <i>database administrator</i> for administering database systems	80–1, 375–80
Database Administrator	A person given responsibility for managing a particular database system or a limited number of database systems	58, 80–1, 322, 344, 376
Database Design	The process of modelling ‘real-world’ constructs in a database system	5, 7



Term	Definition	Pages
Database Development	The entire process of constructing some <i>database system</i>	42–3, 193–4, 324
Database Development Process	The process devoted to the development of database systems. A subprocess of the process of information systems development	195–207
Database Implementation	Taking the decisions of <i>physical database design</i> and implementing them in a chosen DBMS	199–200, 309–20
Database Management System	A suite of computer software providing the interface between users and a <i>database</i> or databases	33
Database Manager	A module of a DBMS concerned with controlling all access to <i>file manager</i> functions	83, 84
Database Segmentation	A <i>data mining</i> technique which involves partitioning a database into a number of segments	550–1
Database Sublanguage	A programming language designed specifically for initiating DBMS functions	158, 160
Database System	A term used to encapsulate the constructs of a <i>data model</i> , <i>DBMS</i> and <i>database</i>	38
Database Task Group (DBTG)	A part of the COBOL programming community chartered to define a standard <i>data definition</i> and <i>data manipulation</i> language	40
Data-item	The atomic construct of some <i>data model</i>	25
Datalog	A variant of <i>Prolog</i> designed for database work	125–42
Datum	A unit of <i>data</i>	6
DBA	See <i>Database Administrator</i>	
DBMS	See <i>Database Management System</i>	
DCL	See <i>Data Control Language</i>	
DDL	See <i>Data Definition Language</i>	
Deadlock	The state in which transactions have locked the data needed by each other	412–13
Decision Support Database	A <i>database</i> used to enable decision-making in some organisation	57
Decomposition	The opposite of <i>aggregation</i> . The process of breaking up an object into its component parts	255
Deductive Data Model	The <i>data model</i> based on formal logic	125–42
Deductive Databases	A database adhering to the <i>deductive data model</i>	126, 133, 522
Default Value	A value declared to be appropriate to be used in the absence of data for some given <i>column</i> in a table	163



Term	Definition	Pages
Degree	The number of <i>tuples</i> in a relation. Also a term used to denote the number of instances relevant to some relationship in the E–R approach	92
Denormalisation	The process of moving back from a fully-normalised data-set to prior normal forms to meet performance requirements	315–16
Dependency	An <i>association</i> between data-items	271–2
Dependency Diagram	See <i>Determinancy Diagram</i>	
Dependent Data-item	A <i>data-item</i> whose values are determined by some other data-item	271–2, 277
Designation	Refers to the symbol(s) by which some concept is known	22
Determinacy	See <i>Dependency</i>	
Determinancy Diagram	A diagram which documents the determinancies between data-items relevant to some application domain. Used in the process of <i>normalisation by synthesis</i>	277
Determinancy Diagramming	The process of producing a <i>determinancy diagram</i>	276–80
Determinant	A <i>data-item</i> which determines the values of some other data-item	371
Deviation Detection	A <i>data mining</i> technique that involves identifying outliers in the population of data	551
Difference	An operator in the <i>relational algebra</i> . A commutative operator that produces a result from two relations in the tuples in the resulting table that are not present in both input tables	101–2
Direct Access	Access to data based on a physical address	385
Disjoint Subclass	Subclasses are disjoint if they do not overlap	253
Disk	A <i>secondary storage</i> device	384–5, 500
Disk Manager	That part of an operating system which translates logical <i>block</i> requests into physical block identifiers	386
Distributed Data	The distribution of data in a database around a network	486–97
Distributed Database Design	The process of designing a distributed database system	494–5
Distributed Database System	A distributed database system is a <i>database system</i> which is fragmented or replicated on the various configurations of hardware and software located usually at different geographical sites within an organisation	478, 487
Distributed DBMS	A <i>DBMS</i> with facilities enabling the distribution of data and/or processing	494
Distributed Processing	The distribution of <i>DBMS</i> functions around a network	477–85



Term	Definition	Pages
Distribution	The distribution of processing or data in some <i>database system</i>	15–16, 478
Distribution Analysis	That part of <i>physical database design</i> concerned with the assignment of processing and/or data to different parts of a computer network. See <i>Distributed Database Design</i>	299
Division	An operator of the <i>relational algebra</i> . Divide takes two tables (one a unary table, the other a binary table) as input and produces one table as output. The resulting table contains tuples where a match is achieved between rows of the unary table and that of the binary table	103–4
DML	See <i>Data Manipulation Language</i>	
Documentary Analysis	A <i>requirements elicitation</i> technique that involves examination of existing organisational documentation	215
Domain	A pool of values from which the actual values represented in an <i>attribute</i> of a <i>relation</i> are taken	93, 173
Domain Integrity	An inherent integrity mechanism in the <i>relational data model</i> concerned with enforcing the concepts of domains	172–3, 304
Domain Name	An agreed string of characters used to add greater meaning to a <i>URL</i>	559
Domain Relational Calculus	A variant of the <i>relational calculus</i> which acts as the foundation for QBE interfaces	105
Duplicated Data	A data value is duplicated when an <i>attribute</i> has two or more identical values	10
Durability	The property of a <i>transaction</i> that ensures all changes caused by a transaction persist in time	405
E		
e-Business	Electronic Business. The conduct of business using information and communication technology. A superset of <i>e-Commerce</i>	62–74
e-Commerce	Electronic Commerce. The conduct of business-to-business commerce using ICT such as that supporting the Internet	63
Electronic Data Interchange (EDI)	A set of standards for the representation and transfer of electronic documentation	65
Electronic Delivery	The delivery of products and services through ICT	67–70
Embedded SQL	An <i>application programming interface</i> in which SQL commands may be embedded in some <i>host language</i> such as C	369–71
Empirics	Branch of semiotics concerned with the physical characteristics of the communication channel	22
Encapsulation	The packaging together of data and procedures within a defined interface	116, 437



Term	Definition	Pages
End-user	An organisational user of database applications	58
End-user Interface	A term used to describe the presentation layer of some information systems application	358–60
End-user Toolkit	The <i>DBMS</i> tools designed for use by end-users	357–64
Entity	Some aspect of the ‘real world’ which has an independent existence and can be uniquely identified	220–1, 249
Entity Integrity	An inherent rule of the <i>relational data model</i> . Every relation must have a <i>primary key</i> and a primary key should be unique and not null	107, 169–70
Entity Life Histories	A technique used to specify the key events affecting the ‘life’ of some <i>entity</i>	262
Entity Model	A model of the entities, relationships and attributes pertaining to some application	198, 220, 249
Entity Modelling	See <i>Entity–Relationship Diagramming</i>	
Entity Type	See <i>Entity</i>	220
Entity–Relationship Data Model	A <i>data model</i> , originally proposed by P.P.S. Chen, which utilises three primary constructs: entities, relationships and attributes	42
Entity–Relationship Diagramming	A technique for graphically representing a <i>conceptual model</i> using constructs from the <i>entity–relationship data model</i>	43, 198, 219–45
Equi-join	The equi-join operator is a <i>product</i> with an associated <i>restrict</i> . It combines two tables but only for rows where the values match in the join columns of two tables	97–8
E–R Diagramming	See <i>Entity–Relationship Diagramming</i>	
Ethnography	A <i>requirements elicitation</i> technique that involves immersion in some organisational setting	216
Event	An event is something that happens at a point in time to some defined <i>object</i>	262–3
Exclusivity	A term used in <i>E–R diagramming</i> to describe the situation in which two or more relationships are mutually exclusive	228–9
Export	The process of writing out data from a <i>database</i> in a format that other systems can use	379
Extended-Relational Data Model	See <i>Post-Relational Data Model</i>	115
Extension	A term used to describe the total set of the <i>data</i> in some <i>database</i>	7, 23
Extent	See <i>Extension</i>	
External Level	See <i>External Schema</i>	78
External Schema	A layer in the <i>ANSI/SPARC architecture</i> which describes data and relationships of interest to application systems or end-users	78



Term	Definition	Pages
Extranet	Allowing access to aspects of an organisation's intranet to accredited users	66
F		
Fact	A positive <i>assertion</i> about some <i>UoD</i>	
Factbase	A collection of facts or positive <i>assertions</i>	6, 133
Fan Trap	A <i>connection trap</i> . The incorrect assumption that an entity linking two other entities will act in the capacity of a data bridge between the entities	234
Federated Database System	A federated database system is made up of a number of relatively independent, autonomous databases	493–4
Field	An element of <i>physical data organisation</i> . A <i>record</i> is a collection of fields	385–6
Fifth Normal Form	Generally speaking, a <i>fourth normal form</i> table is in fifth normal form if it cannot be non-loss decomposed into a series of smaller tables	287–8
File	An element of <i>physical data organisation</i>	385–6
File Maintenance Anomalies	A series of problems occurring in unnormalised data-sets	270–1
File Manager	That module of some operating system that translates between the data structures supported by some <i>DBMS</i> and files on disk	82, 386–7
File Organisation	See <i>Data Organisation</i>	
Finite State Machine	A finite state machine is a hypothetical mechanism which can be in one of a finite number of discrete states at one time. Events occur at discrete points in time and cause changes to the machine's state	264
Firewall	A system used to secure an <i>intranet</i> from unauthorised access	66
First Normal Form	A <i>relation</i> is in first normal form if and only if every non-key attribute is functionally dependent upon the primary key	272–3
Foreign Key	An <i>attribute</i> of a relation which references the primary key of some other relation	93
Formal Language	An artificial language constructed for some defined purpose	24
Formalism	See <i>Representation Formalism</i>	
Fourth Generation Language (4GL)	A high-level programming language for application development. Most 4GLs differ from 3GLs in having highly developed data management functions	373
Fourth Normal Form	A <i>relation</i> is in fourth normal form if there are no independent multi-valued dependencies evident in the schema	285–7
Fragment	Some subset of a database <i>schema</i>	487



Term	Definition	Pages
Fragmentation Transparency	A <i>distributed database system</i> displays fragmentation transparency if users are not aware of how data is fragmented	489
Functional Determinancy (or Dependency)	Two data-items, A and B, are said to be in a functionally determinant or dependent relationship if the same value of data-item B always appears with the same value of data-item A	271–2
G		
Generalisation	An <i>abstraction mechanism</i> or process by which a higher-order object is formed to emphasise the similarities between lower-order objects	117, 250
Generalisation Hierarchy	A hierarchy of object classes linked by <i>generalisation</i> relationships and in which each subclass has only one superclass	250
Generalisation Lattice	A collection of object classes linked by <i>generalisation</i> relationships in which subclasses may have more than one superclass	251
Geographical Information System (GIS)	An information system centred around a core of <i>spatial data</i>	518–20
Gigabytes	1,000,000,000 bytes (billion)	384
Grammer	Rules that control the correct use of a language	24
GRANT	A command in <i>SQL</i> for granting data privileges to authorised users of some database	350–1
Graphical User Interface (GUI)	A user interface in which graphical objects such as icons are used to initiate instructions to the application	361–2
H		
Hash Function	See <i>Hashing</i>	389, 504
Hash Partitioning	Hash partitioning is a parallel partitioning strategy which involves placing tuples by applying a <i>hashing</i> function to an attribute in each tuple	504
Hashed File	A form of <i>physical organisation</i> in which records are stored in terms of some hash function	314, 389–90
Hashing	An algorithm for converting logical values to physical addresses	389, 504
Heterogeneous Distributed Database System	A heterogeneous <i>distributed database system</i> is one in which the hardware and software configuration is diverse	493
Heuristic-based Optimisation	See <i>Syntax-based Optimisation</i>	



Term	Definition	Pages
Hierarchical Data Model	A data model in which data structures are arranged in hierarchies of parent–child relationships	37
Homogeneous Distributed Database System	A homogeneous distributed database system is one in which data is distributed across two or more systems, each system running the same DBMS on the same types of hardware under the same operating system	492–3
Homonym	The same term used to denote dissimilar objects or entities	240
Host Language	A term used to refer to a traditional programming language such as C or COBOL in which data management commands can be embedded	369
HTML	Hypertext Markup Language. A standard for marking up documents to be published on the <i>WWW</i>	27, 71, 515, 561–2
HTTP	Hypertext Transfer Protocol. An object-oriented, stateless protocol that defines how information can be transmitted between client and server	557–8
Human Activity System	Systems consisting of people, conventions and artefacts designed to serve human needs	50–1
Hypermedia	Hypermedia is the approach to building information systems made up of nodes of various media (such as text, audio data, video data etc.) connected together by a collection of associative links	68, 559–60
Hypertext	A subset of <i>hypermedia</i> concentrating on the construction of loosely connected textual systems	560
I		
I-Commerce	Internet Commerce. The use of <i>Internet</i> technologies to enable e-Commerce	63
ICT System	A system of hardware, software, data and communication technology. The ICT component of some information system	51–3
Impedance Mismatch	The mismatch between record-at-a-time forms of processing characteristic of procedural programming languages and file-at-a-time forms of processing characteristic of relational DBMS	370
Import	The process in which data is read into a <i>database</i> from some external system	379
Inclusivity	In the E–R approach two relationships are said to be inclusive if one cannot participate in one <i>relationship</i> without participating in the other	228–9
Inconsistent Analysis Problem	A problem caused by the concurrency of <i>transactions</i> – transactions reading the partial results of incomplete transactions	408, 412



Term	Definition	Pages
Index	A mechanism which associates logical values with physical addresses	314–15, 396–7
Index Sequential File	A form of file structure in which the sequence of records is maintained by an <i>index</i>	39–40, 314, 397
Informatics Infrastructure	A specification of the information, information systems and information technology needed by some organisation	56–7
Information	<i>Data set in some meaningful context</i>	6, 19–28
Information and Communications Technology (ICT)	Technology that includes computers and communications designed to enable information systems	51
Information Management System (IMS)	One of the earliest <i>DBMS</i> developed at IBM	40
Information System	An information system is a system of communication between people. Information systems are systems involved in the gathering, processing, distribution and use of information	49–50
Information Systems Development	The process of producing some information system	196–7
Information Systems Planning	The process of defining some information systems architecture	56
Inherent Integrity	The integrity rules built into some data model	107–9, 121, 304
Inheritance	The process by which subclasses acquire the attributes, methods and relationships of its superclass	117, 250, 437
Instance Diagram	A representation of the instances of <i>entities</i> and the <i>relationships</i> between such instances	227–8
Instantiation	The process of defining an instance upon an <i>object class</i>	251
Integrated Data Store (IDS)	One of the earliest <i>DBMS</i> developed at General Electric	40
Integrity	Maintaining the logical consistency of some database. See also <i>Integrity Constraint</i>	9, 339, 404
Integrity Analysis	That part of the <i>physical database design</i> process concerned with the specification of integrity associated with some database application	304–5
Integrity Constraint	A rule for enforcing <i>integrity</i>	10–11, 12, 131–2, 239, 260, 317
Intension	The collection of properties that in some way characterise some phenomena represented as a sign. See also <i>Schema</i>	23, 25



Term	Definition	Pages
Interface (Database)	The intervening system between <i>toolkit</i> products and DBMS <i>kernel</i>	79, 81, 155–6
Interface Subsystem	A layer of an <i>ICT system</i>	53, 479
Internal Schema	A layer in the <i>ANSI/SPARC architecture</i> which describes the way in which data is stored and accessed from secondary storage devices	78
Internal Value-chain	That part of the <i>value-chain</i> concerned with internal activities	64–5
Internet	A set of interconnected global communication networks	68, 555
Interpreted Queries	Queries that are interpreted by the <i>query processor</i> of some DBMS at run-time	423
Interpreted SQL	A form of <i>SQL</i> in which commands are interpreted	368
Intersection	An operator of the <i>relational algebra</i> . Fundamentally the opposite of <i>Union</i> . Produces a result table which contains the tuples common to both input tables	101
Interviews	A <i>requirements elicitation</i> technique which involves a structured conversation between a developer and some stakeholder	214
Intranet	The use of <i>Internet</i> technology within a single organisation	64, 66
ISA	ISA relates objects to object classes. See also <i>Classification</i>	252
Isolation	The property of a <i>transaction</i> that ensures that changes made by one transaction only become available to other transactions after the transaction has completed	405
Isolation Level	A construct in SQL2 which allows the DBA to control the balance between <i>concurrency</i> and <i>consistency</i>	415

J

Join	An operator of the <i>relational algebra</i> which associates data between tables. The operator takes two relations as input and produces one relation as output. See also <i>Equi-join</i> , <i>Natural Join</i> , <i>Product</i>	97–101, 185, 187–8
------	--	-----------------------

K

Kernel	The core data management facilities of a <i>DBMS</i> such as concurrency control	78, 81–4, 381–2, 460
Key	That <i>field</i> or fields within a file used to identify records	388
Kilobyte	1000 bytes	384

L

Legacy Database System	A database application that was produced some time in the past but which is critically important to some organisation	32
------------------------	---	----



Term	Definition	Pages
Link Analysis	A <i>data mining</i> technique which establishes associations between records in a data-set	551
Location Transparency	A <i>distributed database system</i> displays location transparency if the user is not aware of data being distributed among multiple locations	489
Lock	A mechanism used in <i>concurrency control</i> . A lock can be set on some database element (such as a row) by a user or application system and prevents other users/systems from updating that element	409–10
Locking	A pessimistic form of <i>concurrency control</i>	409–13
Logical Data Dictionary	Logical data dictionaries are used to record data requirements independently of how these requirements are to be met	238–9, 338
Logical Data Independence	Immunity of <i>external schemas</i> to changes in a conceptual schema	78
Logical Database Design	That part of the <i>database design</i> process that produces a logical data model	293, 295–8
Logical Model	An implementation-independent model	42, 198–9, 298
Logical Modelling	The process of producing <i>logical models</i>	42, 198–9, 270
Logical Organisation	The form of <i>data organisation</i> defined in some data model	384
Lost Update Problem	A problem caused by <i>concurrency</i> of transactions – updates being lost in a series of transactions	406–7, 411

M

Main Memory	See <i>Primary Storage</i>	
Managed Query Environment (MQE)	A form of <i>OLAP</i> tool which allows access via ROLAP or MOLAP	544
Manual Record Manager	The earliest form of data management embodied in traditional forms of writing on physical media	38
Mass Deployment Database	A <i>database</i> used to deliver data to the desktop	58
MDSM	Minimum Data Set Model. A <i>corporate data model</i> produced for the UK National Health Service in the late 1980s.	327–8
Meaning Triangle	A simple model of <i>semantics</i> which includes the designation, extension and intension of a sign	22–3
Megabyte	1,000,000 bytes (million)	384
Meronymic Relation	The range of semantic relations between parts of things and their wholes – after the Greek word ‘meros’ for part	256–60



Term	Definition	Pages
Message	An object-oriented mechanism for activating <i>methods</i>	120, 471
Meta-Data	<i>Data</i> about data. A term particularly used to define the primary property of a <i>data dictionary</i>	336, 338
Method	A defined operation associated with an <i>object</i>	119–20, 261, 442–3, 469, 470
Middleware	That range of software which interposes between client software and server software	483–4
Multi-Database System	See <i>Federated Database System</i>	
Multi-Dimensional OLAP (MOLAP)	A type of <i>OLAP</i> tool which uses array data structures to optimise storage and retrieval on an OLAP server	543
Multi-Media	A term used to describe the handling of diverse and complex media (sound, video etc.) within computing applications	41
Multi-User Access	Access to databases by multiple users	14–15, 455
Multi-Valued Determinancy	Data-item B is said to be non-functionally dependent on data-item A if for every value of data-item A there is a delimited set of values for data-item B	277–8
Multiple Inheritance	Multiple inheritance occurs when a given object class may be a subclass of more than one superclass	251
Multiple Instruction Multiple Data (MIMD) Systems	Parallel architectures built out of cheap and readily available microprocessors and memory	501–4
N		
Natural Join	The natural join operator is a <i>product</i> with an associated <i>restrict</i> followed by a project of one of the join columns	98
Natural Language	A language that has evolved for use in everyday communication	24
Natural Language Interface (NLI)	An interface to a DBMS in which the user expresses queries in a restricted form of English or some other natural language	363–4
Negation	The process of denying an <i>assertion</i> or including a negative assertion	135–6
Nested Relation	An extension to the <i>relational data model</i> proposed for handling complex objects	146–8
Network Data Model	A <i>data model</i> in which data structures are arranged in network-like structures	40
Non-Functional Determinancy/ Dependency	See <i>Multi-Valued Determinancy</i>	277–8
Non-Loss Decomposition	The approach to <i>normalisation</i> in which a relation is decomposed into a number of other relations but without losing the fundamental associations between data-items	271

Term	Definition	Pages
Non-Procedural Query Language	A <i>query</i> language in which the user can specify what is required and not how to achieve the result	105
Normal Form	A stage in the <i>normalisation</i> process	43
Normalisation	Normalisation is the process of identifying the logical associations between data-items and designing a database which will represent such associations but without suffering from <i>file maintenance anomalies</i>	43, 269–91, 295
Normalisation Oath	No repeating, the data-items depend upon the key, the whole key and nothing but the key, so help me Codd	276
N-Tier Applications	Applications in which processing is distributed in terms of <i>N</i> tiers of functionality	482
Null	A special character used in relational systems to represent missing or incomplete information	93–4, 163
O		
O ₂	An OO DBMS adhering to the <i>ODMG</i> standard	426, 467–73
Object	Some real world thing which can be uniquely identified. A package of data and procedures	116, 152, 249, 439, 469
Object Class	An object class is a grouping of similar <i>objects</i>	5, 117, 249, 437, 440, 469–70
Object Life Histories	A variation of the <i>entity life history</i> technique applied to objects	262–4
Object Model	A model of some database expressed using <i>object-oriented</i> constructs	198, 249–50
Object Modelling	The process of developing an <i>object model</i>	43, 198, 215–16
Object-Oriented (OO)	A term applied to programming languages, design methods and database systems to mean providing support for constructs such as <i>objects</i> , <i>classes</i> , <i>generalisation</i> , <i>aggregation</i> etc.	247, 468
Object-Oriented Data Model	The application of <i>object-oriented</i> principles to data modelling	110–24, 521
Object-Oriented DBMS	A DBMS adhering to some or all the principles of the <i>OO data model</i>	436–45, 468
Object-Oriented DBMS Manifesto	A proposal produced in the early 1990s which attempted to define a list of mandatory features for <i>OODBMS</i>	437–8
Object-Relational Data Model	See <i>Post-Relational Data Model</i>	
Observation	A <i>requirements elicitation</i> technique involving observation of work activities relevant to the design of some ICT system	214–15



Term	Definition	Pages
ODBC	Open Database Connectivity. A standard call-level API	372
ODMG	Object Database Management Group	116, 437, 468
ODMG Object Definition Language	A standard <i>DDL</i> for object-oriented DBMS	438, 443
ODMG Object Model	A standard specification of constructs for OO databases	438–43
ODMG Object Query Language	A standard query language for OODBMS	438, 443, 472
OLAP	On-line Analytical Processing. A technology that supports complex analytical processing	70, 528, 539–46
OLTP	On-line Transaction Processing	70, 528
On-line Network Data Managers	A form of data management developed in the mid-1960s reliant on technologies such as tele-processing monitors and early DBMS	39–40
Operational Database	A database used to collect operational data	57
Optionality	See <i>Participation</i>	
ORACLE	The first commercial <i>relational DBMS</i>	426, 457–66, 508–10, 535–6
Ordered File	A form of physical data organisation in which records in a file are ordered in terms of one or more fields	388–9
Outer Join	A form of <i>join</i> in which we wish to keep all the rows in one of the input relations whether or not they match	99–101
Overlapping Subclass	Subclasses that share certain semantics	253
P		
Packet-switched Network	A form of network in which data is routed as data packets	555–6
Page	See <i>Block</i>	
Parallel Database System	Any database system utilising parallel technology	498–511
Partial Subclass	Subclasses are partial if other subclasses of a superclass are possible	252–3
Participation	The involvement of entities in a relationship	225–6, 250–1
Passive Data Dictionary	A <i>data dictionary</i> external to a database and DBMS built to record the complexities of a given database structure for use by application development staff and database administrators (DBAs)	339

Term	Definition	Pages
Performance	That characteristic of a <i>database system</i> concerned with the efficiency of execution and operation	306–7, 379
Persistence	Characteristic of <i>data</i> in which data is held for some duration	6, 437, 470
Physical Data Dictionary	Physical data dictionaries are used to record data structures. The set of <i>system tables</i> at the heart of a relational DBMS is a physical data dictionary	338
Physical Data Independence	The immunity of the <i>conceptual schema</i> to changes made to the physical schema	78
Physical Database Design	The process of producing an implementation plan for some <i>database</i>	292–308, 344
Physical Level	See <i>Internal Schema</i>	
Physical Model	An implementation model for some database	42, 199–200, 311–12
Physical Modelling	The process of producing a <i>physical model</i>	42, 199–200
Physical Organisation	That form of <i>data organisation</i> defined in <i>secondary storage devices</i>	384, 388–93
Physical Schema	See <i>Internal Schema</i>	
Physical Symbol System	A set of physical patterns that are manipulated to generate intelligent action	137–8
Piecemeal Development	The process of planning for and developing <i>database systems</i> on an individual basis without concern for aspects of integration	21–3
PL/SQL	Procedural Language/SQL – <i>ORACLE'S</i> procedural fourth generation language	461–4
Post-Relational Data Model	A term used to encapsulate some of the contemporary extensions to the relational data model such as <i>triggers</i> and <i>stored procedures</i>	143–53
Post-Relational DBMS	Contemporary DBMS adhering to the <i>post-relational data model</i>	427–35, 460
Pragmatics	The study of the general context and culture of communication	21
Precompiler	A software module used prior to the compilation process	84, 371
Predicate	A label applied to some property of a <i>UoD</i> . An element of a <i>Datalog</i> proposition	25, 127
Predictive Modelling	A <i>data mining</i> technique which attempts to learn from data	549–50
Primary Key	An <i>attribute</i> or group of attributes used to uniquely identify the tuples of a relation	92–3, 449–50
Primary Storage	Primary storage includes media that can be directly acted upon by the central processing unit (<i>CPU</i>) of the computer, such as main memory or cache memory. Primary storage usually provides fast access to relatively low volumes of data	384, 499



Term	Definition	Pages
Primitive Data Model	A set of <i>data models</i> in objects are represented as record-structures grouped in file-structures	37
Procedural Query Language	A <i>query language</i> in which a sequence of operations is expressed to satisfy some query	45
Product	Product takes two relations as input and produces as output one relation composed of all the possible combinations of input rows	97
Program-Data Independence	See <i>Data Independence</i>	
Programmed Record Managers	One of the earliest forms of data management developed on stored program computers	39
Project	An operator of the <i>relational algebra</i> . The project operator takes a single relation as input and produces a single relation as output with a subset of the columns from the input relation in the output relation	97
Prolog	Programming in Logic – an Artificial Intelligence language. See also <i>Datalog</i>	130, 134
Propagation Constraints	A constraint detailing what should happen between two tables linked by <i>referential integrity</i>	108–9, 171–2
Property	See <i>Attribute</i>	
Proposition	A fundamental data element in the <i>Deductive Data Model</i> . See also <i>Assertion</i>	126–7
Prototyping	A <i>requirements elicitation</i> technique that involves building part of a system to demonstrate to stakeholders	215–16
Punched-Card Record Managers	A form of data management based on the recording and manipulation of data punched onto card	39
Q		
Query	A retrieval operation expressed against some database	453
Query Decomposition	The process of breaking up a <i>query</i> for execution by a parallel architecture	508–10
Query Function	A function which retrieves <i>data</i> from a database	11, 13–14, 129–30
Query Language	That part of a <i>DML</i> which enables users to express queries against a database	81, 83
Query Manager	That module within a <i>DBMS</i> tasked with managing queries	420–3
Query Optimisation	The process of determining an optimal execution plan for a <i>query</i>	421–3
Query Optimiser	That part of a <i>DBMS</i> which performs <i>query optimisation</i>	422–3, 509
Query Processor	See <i>Query Manager</i>	



Term	Definition	Pages
Query-By-Example (QBE)	A type of <i>interface</i> to a database system originally developed by Moshe Zloof at IBM. The term is now generally used to refer to forms-based retrieval interfaces	360–1
R		
Random Access	See <i>Direct Access</i>	
Range Partitioning	A parallel technique involving clustering tuples with similar attributes together in the same partition	504
Record	A physical <i>data structure</i> composed of fields	385–6
Record Type	A collection of records used to store data relevant to some <i>entity</i>	385
Recovery	The process of copying backed-up data onto a database	80, 344, 379, 415–16
Recursion	The process of self-reference	133–4
Recursive Relationship	A <i>relationship</i> in which an <i>entity</i> is related to itself	229–30
Redundancy	See <i>Redundant Data</i>	10
Redundant Data	A data value is redundant if you can delete it without losing information	34
Reference Type	An SQL3 construct used to define relationships between <i>row types</i>	431
Referent	See <i>Extension</i>	
Referential Integrity	An inherent <i>integrity constraint</i> of the relational data model. Referential integrity states that a foreign key value must either refer to a primary key value somewhere in a database or be null	107–9, 170–1, 450–1
Relation	The primitive data structure in the <i>relational data model</i>	91–2
Relational Algebra	A candidate for the manipulative part of the <i>relational data model</i> . A <i>procedural query language</i>	95–105, 110
Relational Calculus	A candidate for the manipulative part of the <i>relational data model</i> based on the predicate calculus. A <i>non-procedural query language</i>	105–7
Relational Data Model	A <i>data model</i> originally created by E.F. Codd in the 1970s	40–1, 89–112, 133
Relational OLAP (ROLAP)	A form of <i>OLAP</i> tool that utilises conventional RDBMS technology	543–4
Relationship	An <i>association</i> between entities or objects	5, 221, 441–2, 450–1
Replicate	Some copy of the whole of or part of some <i>database</i>	487



Term	Definition	Pages
Replication	The process of producing <i>replicates</i>	489
Replication Transparency	A <i>distributed database system</i> displays replication transparency if users are not aware of there being multiple copies of data	489
Representation Formalism	A set of syntactic and semantic conventions that make it possible to describe things	14
Requirement	Any devised feature of some information system or <i>database system</i>	210–11
Requirements Elicitation	The process of eliciting <i>requirements</i> for some application from key stakeholders	42, 198, 208–18
Requirements Specification	The process of expressing the <i>requirements</i> for some application using the constructs of some representation formalism	208, 212–14
Restrict	An operator in the <i>relational algebra</i> which extracts rows from an input relation matching some condition	95–6
Retract	An in-built function in <i>Datalog</i>	129
REVOKE	A command in <i>SQL</i> for revoking data privileges	350
Role	The distinct meaning applied to a <i>relationship</i> between two entities	231
Rollback	The process of returning to a previous state prior to the execution of some <i>transaction</i>	405–6
Round-Robin Partitioning	The process of assigning <i>tuples</i> to partitions on the basis of entry sequence	502–3
Row	A collection of <i>attribute</i> -values in the relational data model	91
Row Type	An SQL3 construct comprising a sequence of column names with associated <i>data types</i>	428–9
Rule	See <i>Integrity Constraint</i>	
Rules Subsystem	A layer of an <i>information technology system</i>	53, 479
Run-time Processor	Part of a DBMS <i>kernel</i> that handles retrieval and update operations expressed against the database	83
<hr/>		
S		
Schema	A term used to refer to the <i>intension</i> of some database	7, 24–5, 165
Schema Partitioning	The assignment of <i>tuples</i> to partitions on the basis of schema elements	502
Second Normal Form	A relation is in second normal form if and only if it is in <i>first normal form</i> and every non-key attribute is fully functionally dependent on the primary key	273–4



Term	Definition	Pages
Secondary Storage	Secondary storage cannot be processed directly by the <i>CPU</i> . It hence provides slower access than primary storage but can handle much larger volumes of data. Two of the most popular forms of secondary storage are magnetic disk and magnetic tape	384
Security	See <i>Data Security</i>	
Security Analysis	That part of <i>physical database design</i> concerned with the specification of appropriate authorisation rights for a defined user community	305–6
Select	See <i>Restrict</i> . Also a command within SQL	178–88
Semantic Data Model (SDM)	That class of data models that were created with the aim of incorporating more semantics into a <i>data model</i>	37–8, 42, 114–15
Semantic Integrity	The business rules associated with some database application	462–4
Semantic Modelling	See <i>Top-down Data Analysis</i>	
Semantics	The study of the meaning of <i>signs</i>	14, 21, 22–3
Semiotics	The study of <i>signs</i>	21–2
Semi-Structured Data	Data which lacks a <i>schema</i> but in which structure is apparent from the way that data is presented	513–18
Sequential File	See <i>Sequential Organisation</i>	
Sequential Organisation	A form of data organisation in which records are placed in the file in the order in which they are inserted	313, 388
SGML	Standard Generalised Markup Language – a meta-language originally defined for the mark-up of documents	515
Shared Nothing Systems	A form of parallel architecture in which each processor has its own memory	506–7
Sign	Anything which is significant. Generally composed of a <i>designation</i> , concept and referent	20–1
Signifier	See <i>Designation</i>	
Single-Valued Determinancy	See <i>Functional Determinancy</i>	
Snowflake Schema	A variation on a <i>star schema</i> in which each dimension can have its own dimension	535
Socio-technical System	A system of technology used within a system of activity	54–5
Spatial Data	Data containing spatial attributes or relations	518–22
Specialisation	Specialisation is the process of creating a new <i>object class</i> by introducing additional detail to the description of an existing object class	251
SQL	See <i>Structured Query Language</i>	



Term	Definition	Pages
SQL – Data Definition	The data definition commands of <i>SQL</i>	157–67
SQL – Data Integrity	The data integrity commands of <i>SQL</i>	168–76
SQL – Data Manipulation	The data manipulation commands of <i>SQL</i>	177–91
Stakeholder	A group of persons within and without the organisation that may potentially influence the success of some <i>ICT system</i> and/or database system	209
Star Schema	A form of schema design utilised in <i>data warehousing</i> in which a central table of factual data is surrounded by tables of reference data	534–5
Starflake Schema	A middle-ground between a <i>star schema</i> and <i>snowflake schema</i>	535
State	A state is a moment in the life of an <i>object</i> . Also used to describe a moment in the life of some database	6, 7, 263
State Transition Diagram	A graphic technique for representing a <i>finite state machine</i>	264
Static Constraint	A restriction defined on states	10, 305
Statistically-based Optimisation	A form of <i>query optimisation</i> driven by statistics held in the data dictionary	422–3
Stored Procedure	A package of logic resident within some database system	150, 432
Strategic Data Planning	The process which identifies priorities for data systems, particularly using the construct of a <i>corporate data model</i>	55, 322, 323–33
Strong Entity	An <i>entity</i> whose existence does not depend on another entity	232
Structural Abstraction	That part of <i>object modelling</i> concerned with specifying relationships of <i>association, aggregation</i> and <i>generalisation</i>	249–60
Structural Inheritance	The <i>inheritance</i> of attributes and relationships between classes	250
Structured Query Language (SQL)	The standard <i>database sublanguage</i> for relational and non-relational access	43, 114, 158–9
Subquery	A <i>query</i> in SQL nested within another query	182–3
Subtype	An SQL3 construct allowing inheritance between tables. A subtype is defined on one or more supertypes	430–1, 440, 470–1
Supertype	An SQL3 construct allowing inheritance between tables	430–1, 440
Supply Chain	The chain of activities that an organisation performs in relation to its suppliers	64, 65–6
Symmetric Multi-Processor Architecture (SMP)	A form of parallel architecture in which an array of processors shares a common memory	501, 504–5
Synonym	An alternative name provided for a database construct	240

Term	Definition	Pages
Syntactics	That part of <i>semiotics</i> devoted to the study of the structure of signs and sign-systems	21, 24
Syntax	The operational rules for the correct representation of terms and their use in the construction of sentences of a language	14, 24
Syntax-based Optimisation	A form of <i>query optimisation</i> driven by the syntax of the input query	422
Synthesis	The approach to <i>normalisation</i> based around the process of synthesising a fully normalised schema from an analysis of dependencies	276
System	A coherent set of interdependent components which exists for some purpose, has some stability, and can be usefully viewed as a whole	49
System Buffer	Structure of some operating system used for storing and manipulating temporary data	82
System Catalog	See <i>System Tables</i>	83, 166
System Developer	Persons tasked with developing and maintaining <i>ICT systems</i>	58
System Tables	The meta-database at the heart of a relational DBMS	165, 339, 419–20
System/R	The first prototype relational <i>DBMS</i>	90

T

Table	See <i>Relation</i>	
TableSpace	A construct used in some <i>DBMS</i> to collect a series of related tables together	392
Tag	A code inserted into a document to indicate mark-up	515
TCP/IP	An open system's communication protocol	556–7
Terabyte	1,000,000,000,000 (trillion) bytes	384
Ternary Relationship	A relationship between three or more <i>entities</i>	230
Third Normal Form	A state in which all relations in a database are fully normalised	274–6
Three-Valued Logic	A modification of traditional two-valued logic (true, false) to accommodate the use of nulls in relational systems	93–4
Time (representation of)	Certain data models such as the <i>E-R data model</i> can be used to represent temporal semantics	235–6
Toolkit (Database)	The range of tools either packaged as part of some <i>DBMS</i> or provided by third-party vendors	78, 202–3, 355–6
Top-down Data Analysis	Generally equated with techniques such as E-R diagramming. <i>Data analysis</i> conducted from abstract constructs to schema definition	222



Term	Definition	Pages
Totality	See <i>Cardinality</i>	
Transaction	A logical unit of work. A transaction transforms a database from one consistent state to another consistent state	10, 67, 260, 300, 404–5, 514
Transaction Analysis	A <i>physical database design</i> process in which an analysis of likely transactions impacting upon a database is performed with the aim of designing the database to handle performance implications	301–3
Transaction Log	A stored history of the transactions that have impacted on a database over an established period of time	378, 413–14
Transaction Management	The process of managing multiple <i>transactions</i> impacting upon some database	80, 403–17
Transaction Maps	A transaction map documents the access issues associated with critical transaction types	302–3
Transaction Subsystem	A layer of an <i>information technology system</i> concerned with managing transactions	53, 479
Transition	A movement between two <i>states</i>	264
Transition Constraint	A rule relating given <i>states</i> of a database	11, 305
Transitive Determinancy	Any situation in which A determines B, B determines C and A also determines C	279
Trigger	An action that is automatically executed by a <i>DBMS</i> when the database achieves a certain state	149, 432–3, 463–4
Tuple	See <i>Row</i>	
Tuple Relational Calculus	A variant of the <i>relational calculus</i> on which languages such as <i>SQL</i> are founded	105
Two-Phase Locking	A protocol that guarantees that transactions are serialisable	410–12
U		
UML	Unified Modelling Language. An attempt to develop a standard syntax for <i>object modelling</i>	254
Uncommitted Dependency Problem	A problem caused by the concurrency of transactions – one transaction views the intermediate results of another transaction before that transaction has completed	407, 411
Union	An operator in the <i>relational algebra</i> . Union produces a melding of rows from two original input relations having the same structure	101
Universal Resource Locator	An identifier for a <i>WWW</i> document	71, 558–9
Universal Server	The use of database systems to support both traditional and non-traditional data	71–2



Term	Definition	Pages
Universe of Discourse (UoD)	Term used to described that aspect of the real world being addressed in a database development effort	5, 24–5
Unnormalised Data-set	A data-set subject to <i>update anomalies</i>	270–1
UoD	See <i>Universe of Discourse</i>	
Update Anomalies	See <i>File Maintenance Anomalies</i>	
Update Function	A standard, application-specific operation performed against some database application	11–12, 131
Usage Analysis	A <i>physical database design</i> process in which an analysis is made of likely patterns of usage particularly in terms of retrieval operations	300–1
User Level	See <i>External Schema</i>	
User Name	A name assigned to each user of a <i>DBMS</i>	377
User-Defined Function (UDF)	A function which permits certain users of the database to alter and manipulate <i>UDTs</i>	72
User-Defined Routine	An SQL3 construct that defines methods for manipulating data	430
User-Defined Type (UDT)	A <i>post-relational data model</i> feature. The ability to specify data types for specific, complex data handling. See also <i>abstract data type</i>	72, 429–30
V		
Value-chain	A series of interdependent activities by which an organisation delivers value	64
View	A virtual table or a window into a database. Implemented as a packaged query in <i>SQL</i> . Also a term used to refer to a <i>conceptual model</i> developed for a particular business area	15, 239, 346–9
View Integration	The process by which a number of application data models are integrated to form one uniform <i>data model</i>	42, 198, 239–41
View Modelling	The process of modelling the data <i>requirements</i> of a number of distinct areas of the organisation with the aim of later integration	42, 198, 239–41
Virtual Memory	A memory management scheme which utilises the idea of a virtual address space	499–500
Visual Query Interface	An <i>end-user interface</i> in which queries can be built visually	361–2
Vocabulary	A complete list of the terms of some language	24
Volatility	Term used to describe the degree to which given <i>data structures</i> are subject to update activity	300–1, 495
Volume Analysis	A <i>physical database design</i> process in which estimates are made as to the volume of data needed to be handled in a database	299–300

Term	Definition	Pages
W		
Weak Entity	An entity whose existence depends on the existence of another <i>entity</i>	232
Web-site	A logical collection of <i>WWW</i> documents normally stored on a <i>WWW server</i>	69
Word	See <i>Byte</i>	
Workshops	A <i>requirements elicitation</i> technique involving the structured interaction of developers and other stakeholders around the formulation of requirements	215
WWW	World-Wide-Web. A set of standards for <i>hypermedia</i> documentation. It now has become synonymous with the Internet	41, 66, 71, 554–66
X		
XML	Extensible Markup Language. A formal language used for the definition of <i>semi-structured data</i>	27, 41, 516–18

