Connolly, Thomas M. & Carolyn E. Begg
The Relational Model (Chapter 4) / from Database Systems: A Practical Approach to Design, Implementation, and Management
Boston ; London : Addison-Wesley, 2010. pp. 91-108.

DATABASE SYSTEMS

A Practical Approach to Design, Implementation, and Management

FIFTH EDITION

THOMAS M. CONNOLLY | CAROLYN E. BEGG

UNIVERSITY OF THE WEST OF SCOTLAND

Addison-Wesley

Boston San Francisco New York
London Toronto Sydney Tokyo Singapore Madrid
Mexico City Munich Paris Cape Town Hong Kong Montreal

Editor-in-Chief	Michael Hirsch	Marketing Coordinator	Kathryn Ferranti
Editorial Assistant	Stephanie Sellinger	Senior Manufacturing Buyer	Carol Melville
Managing Editor	Jeffrey Holcomb	Composition	Aptara®, Inc.
Production Supervisor	Heather McNally	Production Coordination	Shelley Creager, Aptara®, Inc.
Art Director	Linda Knowles	Copyediting	Nancy Kotary
Cover Designer	Beth Paquin, Joyce Cosentino Wells	Proofreading	Don Smith
Online Product Manager	Bethany Tidd	Indexing	Steve Rath
Marketing Manager	Erin Davis	Text Design	Susan Raymond

Access the latest information about Addison-Wesley Computer Science titles from our World Wide Web site: http://www.pearsonhighered.com/cs

Oracle Corporation for Figures 3.18, H.12, H.13, H.14, H.15, H.16, 20.8, 20.9, 20.10, 31.29, and 31.30 reproduced with permission; The McGraw-Hill Companies, Inc., New York for Figure 20.11, reproduced from BYTE Magazine, June 1997. Reproduced with permission. © by The McGraw-Hill Companies, Inc., New York, NY USA. All rights reserved; Figures 28.6 and 28.7 are diagrams from the "Common Warehouse Metamodel (CWM) Specification", March 2003, Version 1.1, Volume 1, formal/03-03-02. Reprinted with permission. Object Management, Inc. © OMG 2003; Screen shots reprinted by permission from Microsoft Corporation. Sun Corporation for Figure 3.10.

This interior of this book was composed in QuarkXpress 6.5.

Library of Congress Cataloging-in-Publication Data

Connolly, Thomas M.
  Database systems : a practical approach to design, implementation, and management / Thomas M. Connolly, Carolyn E. Begg.—5th ed.
    p. cm.
  Includes bibliographical references and index.
  ISBN-13: 978-0-321-52306-8
  ISBN-10: 0-321-52306-7
1. Database design. 2. Database management. I. Begg, Carolyn E. II. Title.
  QA76.9.D26C648 2010
  005.74—dc22

2008055021

Addison-Wesley
is an imprint of

PEARSON

www.pearsonhighered.com

ISBN-13: 978-0-321-52306-8
ISBN-10: 0-321-52306-7

1 2 3 4 5 6 7 8 9 10—EB—13 12 11 10 09

CHAPTER

4 The Relational Model

Chapter Objectives

In this chapter you will learn:

- The origins of the relational model.
- The terminology of the relational model.
- How tables are used to represent data.
- The connection between mathematical relations and relations in the relational model.
- Properties of database relations.
- How to identify candidate, primary, alternate, and foreign keys.
- The meaning of entity integrity and referential integrity.
- The purpose and advantages of views in relational systems.

The Relational Database Management System (RDBMS) has become the dominant data-processing software in use today, with estimated new licence sales of between US$6 billion and US$10 billion per year (US$25 billion with tools sales included). This software represents the second generation of DBMSs and is based on the relational data model proposed by E. F. Codd (1970). In the relational model, all data is logically structured within relations (tables). Each **relation** has a name and is made up of named **attributes** (columns) of data. Each **tuple** (row) contains one value per attribute. A great strength of the relational model is this simple logical structure. Yet behind this simple structure is a sound theoretical foundation that is lacking in the first generation of DBMSs (the network and hierarchical DBMSs).

We devote a significant amount of this book to the RDBMS, in recognition of the importance of these systems. In this chapter, we discuss the terminology and basic structural concepts of the relational data model. In the next chapter, we examine the relational languages that can be used for update and data retrieval.

Structure of this Chapter To put our treatment of the RDBMS into perspective, in Section 4.1 we provide a brief history of the relational model. In Section 4.2 we discuss the underlying concepts and terminology of the relational model. In Section 4.3 we discuss the relational integrity rules, including entity integrity and referential integrity. In Section 4.4 we introduce the concept of views, which are important features of relational DBMSs, although not a concept of the relational model *per se*.

Looking ahead, in Chapters 5–8 we examine SQL (Structured Query Language), the formal and *de facto* standard language for RDBMSs, and in Chapter 9 we examine QBE (Query-By-Example), another highly popular visual query language for RDBMSs. In Chapters 16–19 we present a complete methodology for relational database design. In Chapter 29, we discuss an extension to relational DBMSs to incorporate object-oriented features, and in particular examine the object-oriented features of SQL. In Appendix G, we examine Codd's twelve rules, which form a yardstick against which RDBMS products can be identified. The examples in this chapter are drawn from the *DreamHome* case study, which is described in detail in Section 11.4 and Appendix A.

4.1 Brief History of the Relational Model

The relational model was first proposed by E. F. Codd in his seminal paper "A relational model of data for large shared data banks" (Codd, 1970). This paper is now generally accepted as a landmark in database systems, although a set-oriented model had been proposed previously (Childs, 1968). The relational model's objectives were specified as follows:

- To allow a high degree of data independence. Application programs must not be affected by modifications to the internal data representation, particularly by changes to file organizations, record orderings, or access paths.
- To provide substantial grounds for dealing with data semantics, consistency, and redundancy problems. In particular, Codd's paper introduced the concept of **normalized** relations, that is, relations that have no repeating groups. (The process of normalization is discussed in Chapters 14 and 15.)
- To enable the expansion of set-oriented data manipulation languages.

Although interest in the relational model came from several directions, the most significant research may be attributed to three projects with rather different perspectives. The first of these, at IBM's San José Research Laboratory in California, was the prototype relational DBMS System R, which was developed during the late 1970s (Astrahan *et al.*, 1976). This project was designed to prove the practicality of the relational model by providing an implementation of its data structures and operations. It also proved to be an excellent source of information about implementation concerns such as transaction management, concurrency control, recovery techniques, query optimization, data security and integrity, human factors, and user interfaces, and led to the publication of many research papers and to the development of other prototypes. In particular, the System R project led to two major developments:

- the development of a structured query language called SQL, which has since become the formal International Organization for Standardization (ISO) and de facto standard language for relational DBMSs;
- the production of various commercial relational DBMS products during the late 1970s and the 1980s: for example, DB2 and SQL/DS from IBM and Oracle from Oracle Corporation.

The second project to have been significant in the development of the relational model was the INGRES (Interactive Graphics Retrieval System) project at the University of California at Berkeley, which was active at about the same time as the System R project. The INGRES project involved the development of a prototype RDBMS, with the research concentrating on the same overall objectives as the System R project. This research led to an academic version of INGRES, which contributed to the general appreciation of relational concepts, and spawned the commercial products INGRES from Relational Technology Inc. (now Ingres Corporation) and the Intelligent Database Machine from Britton Lee Inc.

The third project was the Peterlee Relational Test Vehicle at the IBM UK Scientific Centre in Peterlee (Todd, 1976). This project had a more theoretical orientation than the System R and INGRES projects and was significant, principally for research into such issues as query processing and optimization as well as functional extension.

Commercial systems based on the relational model started to appear in the late 1970s and early 1980s. Now there are several hundred RDBMSs for both mainframe and PC environments, even though many do not strictly adhere to the definition of the relational model. Examples of PC-based RDBMSs are Office Access and Visual FoxPro from Microsoft, InterBase from CodeGear, and R:Base from R:BASE Technologies.

Owing to the popularity of the relational model, many nonrelational systems now provide a relational user interface, irrespective of the underlying model. IDMS, the principal network DBMS, has become CA-IDMS from CA Inc. (formerly Computer Associates), supporting a relational view of data. Other mainframe DBMSs that support some relational features are Computer Corporation of America's Model 204 and Software AG's ADABAS.

Some extensions to the relational model have also been proposed; for example, extensions to:

- capture more closely the meaning of data (for example, Codd, 1979);
- support object-oriented concepts (for example, Stonebraker and Rowe, 1986);
- support deductive capabilities (for example, Gardarin and Valduriez, 1989).

We discuss some of these extensions in Chapters 27–29, as we discuss Object DBMSs.

4.2 Terminology

The relational model is based on the mathematical concept of a **relation**, which is physically represented as a **table**. Codd, a trained mathematician, used terminology taken from mathematics, principally set theory and predicate logic. In this section we explain the terminology and structural concepts of the relational model.

4.2.1 Relational Data Structure

Relation | A relation is a table with columns and rows.

An RDBMS requires only that the database be perceived by the user as tables. Note, however, that this perception applies only to the logical structure of the database: that is, the external and conceptual levels of the ANSI-SPARC architecture discussed in Section 2.1. It does not apply to the physical structure of the database, which can be implemented using a variety of storage structures (see Appendix F).

Attribute | An attribute is a named column of a relation.

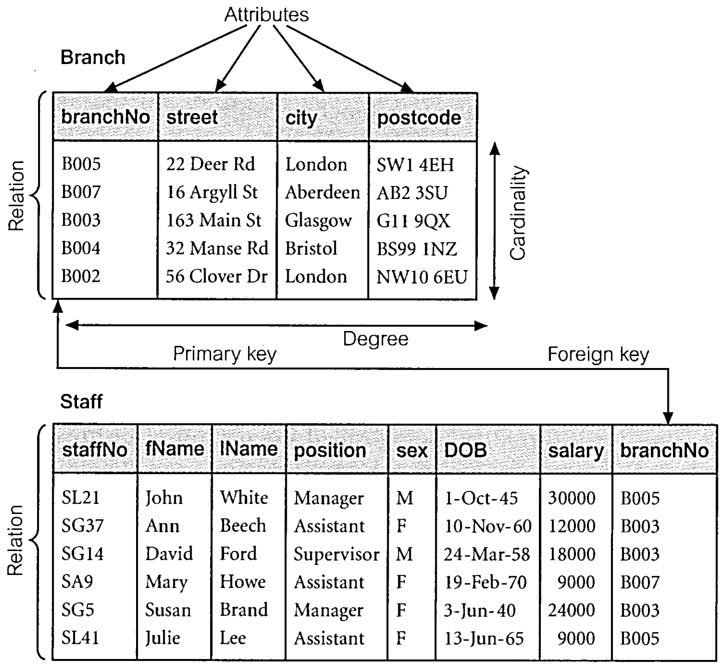
In the relational model, **relations** are used to hold information about the objects to be represented in the database. A relation is represented as a two-dimensional table in which the rows of the table correspond to individual records and the table columns correspond to **attributes**. Attributes can appear in any order and the relation will still be the same relation, and therefore will convey the same meaning.

For example, the information on branch offices is represented by the Branch relation, with columns for attributes branchNo (the branch number), street, city, and postcode. Similarly, the information on staff is represented by the Staff relation, with columns for attributes staffNo (the staff number), fName, lName, position, sex, DOB (date of birth), salary, and branchNo (the number of the branch the staff member works at). Figure 4.1 shows instances of the Branch and Staff relations. As you can see from this example, a column contains values of a single attribute; for example, the branchNo columns contain only numbers of existing branch offices.

Domain | A domain is the set of allowable values for one or more attributes.

Domains are an extremely powerful feature of the relational model. Every attribute in a relation is defined on a **domain**. Domains may be distinct for each attribute, or two or more attributes may be defined on the same domain. Figure 4.2 shows the domains for some of the attributes of the Branch and Staff relations. Note that at any given time, typically there will be values in a domain that do not currently appear as values in the corresponding attribute.

The domain concept is important, because it allows the user to define in a central place the meaning and source of values that attributes can hold. As a result, more



Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Figure 4.1 Instances of the Branch and Staff relations.

Attribute	Domain Name	Meaning	Domain Definition
branchNo	BranchNumbers	The set of all possible branch numbers	character: size 4, range B001–B999
street	StreetNames	The set of all street names in Britain	character: size 25
city	CityNames	The set of all city names in Britain	character: size 15
postcode	Postcodes	The set of all postcodes in Britain	character: size 8
sex	Sex	The sex of a person	character: size 1, value M or F
DOB	DatesOfBirth	Possible values of staff birth dates	date, range from 1-Jan-20, format dd-mmm-yy
salary	Salaries	Possible values of staff salaries	monetary: 7 digits, range 6000.00–40000.00

Figure 4.2 Domains for some attributes of the Branch and Staff relations.

information is available to the system when it undertakes the execution of a relational operation, and operations that are semantically incorrect can be avoided. For example, it is not sensible to compare a street name with a telephone number, even though the domain definitions for both these attributes are character strings. On the other hand, the monthly rental on a property and the number of months a property has been leased have different domains (the first a monetary value, the second an integer value), but it is still a legal operation to multiply two values from these domains. As these two examples illustrate, a complete implementation of domains is not straightforward, and as a result, many RDBMSs do not support them fully.

Tuple | A tuple is a row of a relation.

The elements of a relation are the rows or **tuples** in the table. In the Branch relation, each row contains four values, one for each attribute. Tuples can appear in any order and the relation will still be the same relation, and therefore convey the same meaning.

The structure of a relation, together with a specification of the domains and any other restrictions on possible values, is sometimes called its **intension**, which is usually fixed, unless the meaning of a relation is changed to include additional attributes. The tuples are called the **extension** (or **state**) of a relation, which changes over time.

Degree | The degree of a relation is the number of attributes it contains.

The Branch relation in Figure 4.1 has four attributes or degree four. This means that each row of the table is a four-tuple, containing four values. A relation with only one attribute would have degree one and be called a **unary** relation or one-tuple. A relation with two attributes is called **binary**, one with three attributes is called **ternary**, and after that the term ***n*-ary** is usually used. The degree of a relation is a property of the *intension* of the relation.

Cardinality | The cardinality of a relation is the number of tuples it contains.

By contrast, the number of tuples is called the **cardinality** of the relation and this changes as tuples are added or deleted. The cardinality is a property of the *extension* of the relation and is determined from the particular instance of the relation at any given moment. Finally, we define a relational database.

Relational database | A collection of normalized relations with distinct relation names.

A relational database consists of relations that are appropriately structured. We refer to this appropriateness as *normalization*. We defer the discussion of normalization until Chapters 14 and 15.

Alternative terminology

The terminology for the relational model can be quite confusing. We have introduced two sets of terms. In fact, a third set of terms is sometimes used: a relation may be referred to as a **file**, the tuples as **records**, and the attributes as **fields**. This terminology stems from the fact that, physically, the RDBMS may store each relation in a file. Table 4.1 summarizes the different terms for the relational model.

TABLE 4.1 Alternative terminology for relational model terms.

FORMAL TERMS	ALTERNATIVE 1	ALTERNATIVE 2
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

4.2.2 Mathematical Relations

To understand the true meaning of the term *relation*, we have to review some concepts from mathematics. Suppose that we have two sets, D_1 and D_2, where $D_1 = \{2, 4\}$ and $D_2 = \{1, 3, 5\}$. The **Cartesian product** of these two sets, written $D_1 \times D_2$, is the set of all ordered pairs such that the first element is a member of D_1 and the second element is a member of D_2. An alternative way of expressing this is to find all combinations of elements with the first from D_1 and the second from D_2. In our case, we have:

$$D_1 \times D_2 = \{(2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5)\}$$

Any subset of this Cartesian product is a relation. For example, we could produce a relation R such that:

$$R = \{(2, 1), (4, 1)\}$$

We may specify which ordered pairs will be in the relation by giving some condition for their selection. For example, if we observe that R includes all those ordered pairs in which the second element is 1, then we could write R as:

$$R = \{(x, y) \mid x \in D_1, y \in D_2, \text{and } y = 1\}$$

Using these same sets, we could form another relation S in which the first element is always twice the second. Thus, we could write S as:

$$S = \{(x, y) \mid x \in D_1, y \in D_2, \text{and } x = 2y\}$$

or, in this instance,

$$S = \{(2, 1)\}$$

as there is only one ordered pair in the Cartesian product that satisfies this condition. We can easily extend the notion of a relation to three sets. Let D_1, D_2, and D_3 be three sets. The Cartesian product $D_1 \times D_2 \times D_3$ of these three sets is the set of all ordered triples such that the first element is from D_1, the second element is from D_2, and the third element is from D_3. Any subset of this Cartesian product is a relation. For example, suppose we have:

$$D_1 = \{1, 3\} \quad D_2 = \{2, 4\} \quad D_3 = \{5, 6\}$$

$$D_1 \times D_2 \times D_3 = \{(1,2,5), (1,2,6), (1,4,5), (1,4,6), (3,2,5), (3,2,6), (3,4,5), (3,4,6)\}$$

Any subset of these ordered triples is a relation. We can extend the three sets and define a general relation on n domains. Let $D_1, D_2, \ldots, D_n$ be n sets. Their Cartesian product is defined as:

$$D_1 \times D_2 \times \ldots \times D_n = \{(d_1, d_2, \ldots, d_n) \mid d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n\}$$

and is usually written as:

$$\prod_{i=1}^{n} D_i$$

Any set of n-tuples from this Cartesian product is a relation on the n sets. Note that in defining these relations we have to specify the sets, or **domains**, from which we choose values.

4.2.3 Database Relations

Applying the previously discussed concepts to databases, we can define a relation schema.

Relation schema | A named relation defined by a set of attribute and domain name pairs.

Let $A_1, A_2, \ldots, A_n$ be attributes with domains $D_1, D_2, \ldots, D_n$. Then the set $\{A_1{:}D_1, A_2{:}D_2, \ldots, A_n{:}D_n\}$ is a relation schema. A relation R defined by a relation schema S is a set of mappings from the attribute names to their corresponding domains. Thus, relation R is a set of n-tuples:

$(A_1{:}d_1, A_2{:}d_2, \ldots, A_n{:}d_n)$ such that $d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n$

Each element in the n-tuple consists of an attribute and a value for that attribute. Normally, when we write out a relation as a table, we list the attribute names as column headings and write out the tuples as rows having the form $(d_1, d_2, \ldots, d_n)$, where each value is taken from the appropriate domain. In this way, we can think of a relation in the relational model as any subset of the Cartesian product of the domains of the attributes. A table is simply a physical representation of such a relation.

In our example, the Branch relation shown in Figure 4.1 has attributes branchNo, street, city, and postcode, each with its corresponding domain. The Branch relation is any subset of the Cartesian product of the domains, or any set of four-tuples in which the first element is from the domain BranchNumbers, the second is from the domain StreetNames, and so on. One of the four-tuples is:

{(B005, 22 Deer Rd, London, SW1 4EH)}

or more correctly:

{(branchNo: B005, street: 22 Deer Rd, city: London, postcode: SW1 4EH)}

We refer to this as a **relation instance**. The Branch table is a convenient way of writing out all the four-tuples that form the relation at a specific moment in time, which explains why table rows in the relational model are called "tuples". In the same way that a relation has a schema, so too does the relational database.

Relational database schema | A set of relation schemas, each with a distinct name.

If $R_1, R_2, \ldots, R_n$ are a set of relation schemas, then we can write the *relational database schema*, or simply *relational schema*, R, as:

$R = \{R_1, R_2, \ldots, R_n\}$

4.2.4 Properties of Relations

A relation has the following properties:

- the relation has a name that is distinct from all other relation names in the relational schema;
- each cell of the relation contains exactly one atomic (single) value;

- each attribute has a distinct name;
- the values of an attribute are all from the same domain;
- each tuple is distinct; there are no duplicate tuples;
- the order of attributes has no significance;
- the order of tuples has no significance, theoretically. (However, in practice, the order may affect the efficiency of accessing tuples.)

To illustrate what these restrictions mean, consider again the Branch relation shown in Figure 4.1. Because each cell should contain only one value, it is illegal to store two postcodes for a single branch office in a single cell. In other words, relations do not contain repeating groups. A relation that satisfies this property is said to be **normalized** or in **first normal form**. (Normal forms are discussed in Chapters 14 and 15.)

The column names listed at the tops of columns correspond to the attributes of the relation. The values in the branchNo attribute are all from the BranchNumbers domain; we should not allow a postcode value to appear in this column. There can be no duplicate tuples in a relation. For example, the row (B005, 22 Deer Rd, London, SW1 4EH) appears only once.

Provided that an attribute name is moved along with the attribute values, we can interchange columns. The table would represent the same relation if we were to put the city attribute before the postcode attribute, although for readability it makes more sense to keep the address elements in the normal order. Similarly, tuples can be interchanged, so the records of branches B005 and B004 can be switched and the relation will still be the same.

Most of the properties specified for relations result from the properties of mathematical relations:

- When we derived the Cartesian product of sets with simple, single-valued elements such as integers, each element in each tuple was single-valued. Similarly, each cell of a relation contains exactly one value. However, a mathematical relation need not be normalized. Codd chose to disallow repeating groups to simplify the relational data model.
- In a relation, the possible values for a given position are determined by the set, or domain, on which the position is defined. In a table, the values in each column must come from the same attribute domain.
- In a set, no elements are repeated. Similarly, in a relation, there are no duplicate tuples.
- Because a relation is a set, the order of elements has no significance. Therefore, in a relation, the order of tuples is immaterial.

However, in a mathematical relation, the order of elements in a tuple is important. For example, the ordered pair (1, 2) is quite different from the ordered pair (2, 1). This is not the case for relations in the relational model, which specifically requires that the order of attributes be immaterial. The reason is that the column headings define which attribute the value belongs to. This means that the order of column headings in the intension is immaterial, but once the structure of the relation is chosen, the order of elements within the tuples of the extension must match the order of attribute names.

4.2.5 Relational Keys

As stated earlier, there are no duplicate tuples within a relation. Therefore, we need to be able to identify one or more attributes (called **relational keys**) that uniquely identifies each tuple in a relation. In this section, we explain the terminology used for relational keys.

Superkey | An attribute, or set of attributes, that uniquely identifies a tuple within a relation.

A superkey uniquely identifies each tuple within a relation. However, a superkey may contain additional attributes that are not necessary for unique identification, and we are interested in identifying superkeys that contain only the minimum number of attributes necessary for unique identification.

Candidate key | A superkey such that no proper subset is a superkey within the relation.

A candidate key K for a relation R has two properties:

- **uniqueness**—in each tuple of R, the values of K uniquely identify that tuple;
- **irreducibility**—no proper subset of K has the uniqueness property.

There may be several candidate keys for a relation. When a key consists of more than one attribute, we call it a **composite key**. Consider the Branch relation shown in Figure 4.1. Given a value of city, we can determine several branch offices (for example, London has two branch offices). This attribute cannot be a candidate key. On the other hand, because *DreamHome* allocates each branch office a unique branch number, given a branch number value, branchNo, we can determine at most one tuple, so that branchNo is a candidate key. Similarly, postcode is also a candidate key for this relation.

Now consider a relation Viewing, which contains information relating to properties viewed by clients. The relation comprises a client number (clientNo), a property number (propertyNo), a date of viewing (viewDate) and, optionally, a comment (comment). Given a client number, clientNo, there may be several corresponding viewings for different properties. Similarly, given a property number, propertyNo, there may be several clients who viewed this property. Therefore, clientNo by itself or propertyNo by itself cannot be selected as a candidate key. However, the combination of clientNo and propertyNo identifies at most one tuple, so for the Viewing relation, clientNo and propertyNo together form the (composite) candidate key. If we need to take into account the possibility that a client may view a property more than once, then we could add viewDate to the composite key. However, we assume that this is not necessary.

Note that an instance of a relation cannot be used to prove that an attribute or combination of attributes is a candidate key. The fact that there are no duplicates for the values that appear at a particular moment in time does not guarantee that duplicates are not possible. However, the presence of duplicates in an instance can be used to show that some attribute combination is not a candidate key. Identifying a candidate key requires that we know the "real-world" meaning of the attribute(s) involved so that we can decide whether duplicates are possible. Only by using this semantic information can we be certain that an attribute combination is a candidate key. For example, from the data presented in Figure 4.1, we may think that a suitable candidate key for the Staff relation would be lName, the employee's surname. However, although there is only a single value of "White" in this instance of the Staff relation, a new member of staff with the surname "White" may join the company, invalidating the choice of lName as a candidate key.

Primary key | The candidate key that is selected to identify tuples uniquely within the relation.

Because a relation has no duplicate tuples, it is always possible to identify each row uniquely. This means that a relation always has a primary key. In the worst case, the entire set of attributes could serve as the primary key, but usually some smaller subset is sufficient to distinguish the tuples. The candidate keys that are not selected to be the primary key are called **alternate keys**. For the Branch relation, if we choose branchNo as the primary key, postcode would then be an alternate key. For the Viewing relation, there is only one candidate key, comprising clientNo and propertyNo, so these attributes would automatically form the primary key.

Foreign key | An attribute, or set of attributes, within one relation that matches the candidate key of some (possibly the same) relation.

When an attribute appears in more than one relation, its appearance usually represents a relationship between tuples of the two relations. For example, the inclusion of branchNo in both the Branch and Staff relations is quite deliberate and links each branch to the details of staff working at that branch. In the Branch relation, branchNo is the primary key. However, in the Staff relation, the branchNo attribute exists to match staff to the branch office they work in. In the Staff relation, branchNo is a foreign key. We say that the attribute branchNo in the Staff relation **targets** the primary key attribute branchNo in the **home relation**, Branch. These common attributes play an important role in performing data manipulation, as we see in the next chapter.

4.2.6 Representing Relational Database Schemas

A relational database consists of any number of normalized relations. The relational schema for part of the *DreamHome* case study is:

Branch	(branchNo, street, city, postcode)
Staff	(staffNo, fName, lName, position, sex, DOB, salary, branchNo)
PropertyForRent	(propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)
Client	(clientNo, fName, lName, telNo, prefType, maxRent, eMail)
PrivateOwner	(ownerNo, fName, lName, address, telNo, eMail, password)
Viewing	(clientNo, propertyNo, viewDate, comment)
Registration	(clientNo, branchNo, staffNo, dateJoined)

The common convention for representing a relation schema is to give the name of the relation followed by the attribute names in parentheses. Normally, the primary key is underlined.

The *conceptual model*, or *conceptual schema*, is the set of all such schemas for the database. Figure 4.3 shows an instance of this relational schema.

Figure 4.3 Instance of the *DreamHome* rental database.

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Client

clientNo	fName	lName	telNo	prefType	maxRent	eMail
CR76	John	Kay	0207-774-5632	Flat	425	john.kay@gmail.com
CR56	Aline	Stewart	0141-848-1825	Flat	350	astewart@hotmail.com
CR74	Mike	Ritchie	01475-392178	House	750	mritchie01@yahoo.co.uk
CR62	Mary	Tregear	01224-196720	Flat	600	maryt@hotmail.co.uk

PrivateOwner

ownerNo	fName	lName	address	telNo	eMail	password
CO46	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212	jkeogh@lhh.com	********
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419	cfarrel@gmail.com	********
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728	tinam@hotmail.com	********
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025	tony.shaw@ark.com	********

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-08	too small
CR76	PG4	20-Apr-08	too remote
CR56	PG4	26-May-08	
CR62	PA14	14-May-08	no dining room
CR56	PG36	28-Apr-08	

Registration

clientNo	branchNo	staffNo	dateJoined
CR76	B005	SL41	2-Jan-08
CR56	B003	SG37	11-Apr-07
CR74	B003	SG37	16-Nov-06
CR62	B007	SA9	7-Mar-07

4.3 Integrity Constraints

In the previous section, we discussed the structural part of the relational data model. As stated in Section 2.3, a data model has two other parts: a manipulative part, defining the types of operation that are allowed on the data, and a set of integrity constraints, which ensure that the data is accurate. In this section, we discuss the relational integrity constraints, and in the next chapter, we discuss the relational manipulation operations.

We have already seen an example of an integrity constraint in Section 4.2.1: because every attribute has an associated domain, there are constraints (called **domain constraints**) that form restrictions on the set of values allowed for the attributes of relations. In addition, there are two important **integrity rules**, which are constraints or restrictions that apply to all instances of the database. The two principal rules for the relational model are known as **entity integrity** and **referential integrity**. Other types of integrity constraint are **multiplicity**, which we discuss in Section 12.6, and **general constraints**, which we introduce in Section 4.3.4. Before we define entity and referential integrity, it is necessary to understand the concept of nulls.

4.3.1 Nulls

Null | Represents a value for an attribute that is currently unknown or is not applicable for this tuple.

A null can be taken to mean the logical value "unknown." It can mean that a value is not applicable to a particular tuple, or it could merely mean that no value has yet been supplied. Nulls are a way to deal with incomplete or exceptional data. However, a null is not the same as a zero numeric value or a text string filled with spaces; zeros and spaces are values, but a null represents the absence of a value. Therefore, nulls should be treated differently from other values. Some authors use the term "null value"; however, as a null is not a value but represents the absence of a value, the term "null value" is deprecated.

For example, in the Viewing relation shown in Figure 4.3, the comment attribute may be undefined until the potential renter has visited the property and returned his or her comment to the agency. Without nulls, it becomes necessary to introduce false data to represent this state or to add additional attributes that may not be meaningful to the user. In our example, we may try to represent a null comment with the value -1. Alternatively, we may add a new attribute hasCommentBeenSupplied to the Viewing relation, which contains a Y (Yes) if a comment has been supplied, and N (No) otherwise. Both these approaches can be confusing to the user.

Nulls can cause implementation problems, arising from the fact that the relational model is based on first-order predicate calculus, which is a two-valued or Boolean logic—the only values allowed are true or false. Allowing nulls means that we have to work with a higher-valued logic, such as three- or four-valued logic (Codd, 1986, 1987, 1990).

The incorporation of nulls in the relational model is a contentious issue. Codd later regarded nulls as an integral part of the model (Codd, 1990). Others consider this approach to be misguided, believing that the missing information problem is not fully understood, that no fully satisfactory solution has been found and, consequently, that the incorporation of nulls in the relational model is premature (see, for example, Date, 1995).

We are now in a position to define the two relational integrity rules.

4.3.2 Entity Integrity

The first integrity rule applies to the primary keys of base relations. For the present, we define a base relation as a relation that corresponds to an entity in the conceptual schema (see Section 2.1). We provide a more precise definition in Section 4.4.

Entity integrity | In a base relation, no attribute of a primary key can be null.

By definition, a primary key is a minimal identifier that is used to identify tuples uniquely. This means that no subset of the primary key is sufficient to provide unique identification of tuples. If we allow a null for any part of a primary key, we are implying that not all the attributes are needed to distinguish between tuples, which contradicts the definition of the primary key. For example, as branchNo is the primary key of the Branch relation, we should not be able to insert a tuple into the Branch relation with a null for the branchNo attribute. As a second example, consider the composite primary key of the Viewing relation, comprising the client number (clientNo) and the property number (propertyNo). We should not be able to insert a tuple into the Viewing relation with a null for the clientNo attribute, or a null for the propertyNo attribute, or nulls for both attributes.

If we were to examine this rule in detail, we would find some anomalies. First, why does the rule apply only to primary keys and not more generally to candidate keys, which also identify tuples uniquely? Second, why is the rule restricted to base relations? For example, using the data of the Viewing relation shown in Figure 4.3, consider the query "List all comments from viewings." This query will produce a unary relation consisting of the attribute comment. By definition, this attribute must be a primary key, but it contains nulls (corresponding to the viewings on PG36 and PG4 by client CR56). Because this relation is not a base relation, the model allows the primary key to be null. There have been several attempts to redefine this rule (see, for example, Codd, 1988, and Date, 1990).

4.3.3 Referential Integrity

The second integrity rule applies to foreign keys.

Referential integrity | If a foreign key exists in a relation, either the foreign key value must match a candidate key value of some tuple in its home relation or the foreign key value must be wholly null.

For example, branchNo in the Staff relation is a foreign key targeting the branchNo attribute in the home relation, Branch. It should not be possible to create a staff record with branch number B025, for example, unless there is already a record for branch number B025 in the Branch relation. However, we should be able to create a new staff record with a null branch number to allow for the situation where a new member of staff has joined the company but has not yet been assigned to a particular branch office.

4.3.4 General Constraints

General constraints | Additional rules specified by the users or database administrators of a database that define or constrain some aspect of the enterprise.

It is also possible for users to specify additional constraints that the data must satisfy. For example, if an upper limit of 20 has been placed upon the number of staff that may work at a branch office, then the user must be able to specify this general constraint and expect the DBMS to enforce it. In this case, it should not be possible to add a new member of staff at a given branch to the Staff relation if the number of staff currently assigned to that branch is 20. Unfortunately, the level of support for general constraints varies from system to system. We discuss the implementation of relational integrity in Chapters 7 and 18.

4.4 Views

In the three-level ANSI-SPARC architecture presented in Chapter 2, we described an external view as the structure of the database as it appears to a particular user. In the relational model, the word "view" has a slightly different meaning. Rather than being the entire external model of a user's view, a view is a **virtual** or **derived relation**: a relation that does not necessarily exist in its own right, but may be dynamically derived from one or more **base relations**. Thus, an external model can consist of both base (conceptual-level) relations and views derived from the base relations. In this section, we briefly discuss views in relational systems. In Section 7.4 we examine views in more detail and show how they can be created and used within SQL.

4.4.1 Terminology

The relations we have been dealing with so far in this chapter are known as base relations.

Base relation | A named relation corresponding to an entity in the conceptual schema, whose tuples are physically stored in the database.

We can define a view in terms of base relations:

View | The dynamic result of one or more relational operations operating on the base relations to produce another relation. A view is a *virtual relation* that does not necessarily exist in the database but can be produced upon request by a particular user, at the time of request.

A view is a relation that appears to the user to exist, can be manipulated as if it were a base relation, but does not necessarily exist in storage in the sense that the base relations do (although its definition is stored in the system catalog). The contents of a view are defined as a query on one or more base relations. Any operations on the view are automatically translated into operations on the relations from which it is derived. Views are **dynamic**, meaning that changes made to the base relations that affect the view are immediately reflected in the view. When users make permitted changes to the view, these changes are made to the underlying relations. In this section, we describe the purpose of views and briefly examine restrictions that apply to updates made through views. However, we defer treatment of how views are defined and processed until Section 7.4.

4.4.2 Purpose of Views

The view mechanism is desirable for several reasons:

- It provides a powerful and flexible security mechanism by hiding parts of the database from certain users. Users are not aware of the existence of any attributes or tuples that are missing from the view.
- It permits users to access data in a way that is customized to their needs, so that the same data can be seen by different users in different ways, at the same time.
- It can simplify complex operations on the base relations. For example, if a view is defined as a combination (join) of two relations (see Section 5.1), users may now perform more simple operations on the view, which will be translated by the DBMS into equivalent operations on the join.

A view should be designed to support the external model that the user finds familiar. For example:

- A user might need Branch tuples that contain the names of managers as well as the other attributes already in Branch. This view is created by combining the Branch relation with a restricted form of the Staff relation where the staff position is "Manager."
- Some members of staff should see Staff tuples without the salary attribute.
- Attributes may be renamed or the order of attributes changed. For example, the user accustomed to calling the branchNo attribute of branches by the full name Branch Number may see that column heading.
- Some members of staff should see property records only for those properties that they manage.

Although all these examples demonstrate that a view provides *logical data independence* (see Section 2.1.5), views allow a more significant type of logical data independence that supports the reorganization of the conceptual schema. For example, if a new attribute is added to a relation, existing users can be unaware of its existence if their views are defined to exclude it. If an existing relation is rearranged or split up, a view may be defined so that users can continue to see their original views. We will see an example of this in Section 7.4.7 when we discuss the advantages and disadvantages of views in more detail.

4.4.3 Updating Views

All updates to a base relation should be immediately reflected in all views that reference that base relation. Similarly, if a view is updated, then the underlying base relation should reflect the change. However, there are restrictions on the types of modification that can be made through views. We summarize here the conditions under which most systems determine whether an update is allowed through a view:

- Updates are allowed through a view defined using a simple query involving a single base relation and containing either the primary key or a candidate key of the base relation.
- Updates are not allowed through views involving multiple base relations.
- Updates are not allowed through views involving aggregation or grouping operations.

Classes of views have been defined that are **theoretically not updatable**, **theoretically updatable**, and **partially updatable**. A survey on updating relational views can be found in Furtado and Casanova (1985).

Chapter Summary

- The Relational Database Management System (RDBMS) has become the dominant data-processing software in use today, with estimated new licence sales of between US$6 billion and US$10 billion per year (US$25 billion with tools sales included). This software represents the second generation of DBMSs and is based on the relational data model proposed by E. F. Codd.
- A mathematical **relation** is a subset of the Cartesian product of two or more sets. In database terms, a relation is any subset of the Cartesian product of the domains of the attributes. A relation is normally written as a set of *n*-tuples, in which each element is chosen from the appropriate domain.
- Relations are physically represented as **tables**, with the rows corresponding to individual tuples and the columns to attributes.
- The structure of the relation, with domain specifications and other constraints, is part of the **intension** of the database; the relation with all its tuples written out represents an **instance** or **extension** of the database.
- Properties of database relations are: each cell contains exactly one atomic value, attribute names are distinct, attribute values come from the same domain, attribute order is immaterial, tuple order is immaterial, and there are no duplicate tuples.
- The **degree** of a relation is the number of attributes, and the **cardinality** is the number of tuples. A **unary** relation has one attribute, a **binary** relation has two, a **ternary** relation has three, and an ***n*-ary** relation has *n* attributes.
- A **superkey** is an attribute, or set of attributes, that identifies tuples of a relation uniquely, and a **candidate key** is a minimal superkey. A **primary key** is the candidate key chosen for use in identification of tuples. A relation must always have a primary key. A **foreign key** is an attribute, or set of attributes, within one relation that is the candidate key of another relation.
- A **null** represents a value for an attribute that is unknown at the present time or is not applicable for this tuple.

- **Entity integrity** is a constraint that states that in a base relation no attribute of a primary key can be null. **Referential integrity** states that foreign key values must match a candidate key value of some tuple in the home relation or be wholly null. Apart from relational integrity, integrity constraints include required data, domain, and multiplicity constraints; other integrity constraints are called **general constraints**.
- A **view** in the relational model is a **virtual** or **derived relation** that is dynamically created from the underlying base relation(s) when required. Views provide security and allow the designer to customize a user's model. Not all views are updatable.

Review Questions

4.1 Discuss each of the following concepts in the context of the relational data model:
(a) relation
(b) attribute
(c) domain
(d) tuple
(e) intension and extension
(f) degree and cardinality.

4.2 Describe the relationship between mathematical relations and relations in the relational data model.

4.3 Describe the differences between a relation and a relation schema. What is a relational database schema?

4.4 Discuss the properties of a relation.

4.5 Discuss the differences between the candidate keys and the primary key of a relation. Explain what is meant by a foreign key. How do foreign keys of relations relate to candidate keys? Give examples to illustrate your answer.

4.6 Define the two principal integrity rules for the relational model. Discuss why it is desirable to enforce these rules.

4.7 What is a view? Discuss the difference between a view and a base relation.

Exercises

The following tables form part of a database held in a relational DBMS:

Hotel	(<u>hotelNo</u>, hotelName, city)
Room	(<u>roomNo</u>, <u>hotelNo</u>, type, price)
Booking	(<u>hotelNo</u>, <u>guestNo</u>, <u>dateFrom</u>, dateTo, roomNo)
Guest	(<u>guestNo</u>, guestName, guestAddress)

where Hotel *contains hotel details and* hotelNo *is the primary key;*
Room contains room details for each hotel and (roomNo, hotelNo) forms the primary key;
Booking contains details of bookings and (hotelNo, guestNo, dateFrom) forms the primary key;
Guest contains guest details and guestNo is the primary key.

4.8 Identify the foreign keys in this schema. Explain how the entity and referential integrity rules apply to these relations.

4.9 Produce some sample tables for these relations that observe the relational integrity rules. Suggest some general constraints that would be appropriate for this schema.

4.10 Analyze the RDBMSs that you are currently using. Determine the support the system provides for primary keys, alternate keys, foreign keys, relational integrity, and views.

4.11 Implement the previous schema in one of the RDBMSs you currently use. Implement, where possible, the primary, alternate, and foreign keys, and appropriate relational integrity constraints.