

Adrienne Watt

SECOND EDITION

Database Design



Database Design - 2nd Edition

Database Design - 2nd Edition

ADRIENNE WATT AND WATT, ADRIENNE

NELSON ENG

BCCAMPUS
VICTORIA, B.C.



Database Design – 2nd Edition by Adrienne Watt and Nelson Eng is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/), except where otherwise noted.

© 2014 Adrienne Watt and Nelson Eng

The CC licence permits you to retain, reuse, copy, redistribute, and revise this book—in whole or in part—for free providing the authors are attributed as follows:

[Database Design – 2nd Edition](#) by Adrienne Watt and Nelson Eng is used under a [CC BY 4.0 International Licence](https://creativecommons.org/licenses/by/4.0/).

If you redistribute all or part of this book, you must include the following on the copyright notice page:

Download for free from the [B.C. Open Collection](#).

Sample APA-style citation:

This textbook can be referenced. In APA citation style, it would appear as follows:

Watt, A. and N. Eng. (2014). *Database Design – 2nd Edition*. Victoria, B.C.: BCcampus. Retrieved from <https://opentextbc.ca/dbdesign01/>.

Cover image attribution:

[Spiral Stairs In Milano old building downtown](#) by [Michele Ursino](#) is used under a [CC BY-SA 2.0 Generic License](https://creativecommons.org/licenses/by-sa/2.0/).

Visit [BCcampus Open Education](https://bccampusopeneducation.ca/) to learn about open education in British Columbia.

New to this edition are more examples, highlighted and defined key terms (both throughout and at the end of each chapter), and end-of-chapter review exercises. Two new chapters have been added on SQL, along with appendices that include a data model example, sample ERD exercises, and SQL lab with solutions.

This book was produced with Pressbooks (<https://pressbooks.com>) and rendered with Prince.

Contents

Preface	vii
About the Book	viii
Acknowledgements	ix
Chapter 1 Before the Advent of Database Systems	1
Adrienne Watt	
Chapter 2 Fundamental Concepts	6
Adrienne Watt & Nelson Eng	
Chapter 3 Characteristics and Benefits of a Database	9
Adrienne Watt	
Chapter 4 Types of Data Models	14
Adrienne Watt & Nelson Eng	
Chapter 5 Data Modelling	17
Adrienne Watt	
Chapter 6 Classification of Database Management Systems	23
Adrienne Watt	
Chapter 7 The Relational Data Model	27
Adrienne Watt	
Chapter 8 The Entity Relationship Data Model	33
Adrienne Watt	
Chapter 9 Integrity Rules and Constraints	49
Adrienne Watt & Nelson Eng	
Chapter 10 ER Modelling	61
Adrienne Watt	
Chapter 11 Functional Dependencies	67
Adrienne Watt	
Chapter 12 Normalization	73
Adrienne Watt	
Chapter 13 Database Development Process	83
Adrienne Watt	
Chapter 14 Database Users	91
Adrienne Watt	
Chapter 15 SQL Structured Query Language	93
Adrienne Watt & Nelson Eng	
Chapter 16 SQL Data Manipulation Language	105
Adrienne Watt & Nelson Eng	

Appendix A University Registration Data Model Example	127
Appendix B Sample ERD Exercises	132
Appendix C SQL Lab with Solution	135
About the Authors	142
Versioning History	144

Preface

The primary purpose of this text is to provide an open source textbook that covers most introductory database courses. The material in the textbook was obtained from a variety of sources. All the sources are found in at the end of each chapter. I expect, with time, the book will grow with more information and more examples.

I welcome any feedback that would improve the book. If you would like to add a section to the book, please let me know.

Adrienne Watt

About the Book

Database Design – 2nd Edition is a remix and adaptation based on Adrienne Watt's book *Database Design*. Works that are part of the remix for this book are listed at the end of each chapter. For information about what was changed in this adaptation, refer to the copyright statement in the copyright section or on the webbook homepage.

[BCcampus Open Education](#) began in 2012 as the B.C. Open Textbook Project with the goal of making post-secondary education in British Columbia more accessible by reducing student costs through the use of openly licenced textbooks and other OER. [BCcampus](#) supports the post-secondary institutions of British Columbia as they adapt and evolve their teaching and learning practices to enable powerful learning opportunities for the students of B.C. BCcampus Open Education is funded by the [British Columbia Ministry of Advanced Education, Skills & Training](#), and the [Hewlett Foundation](#).

Open textbooks are open educational resources (OER) created and shared in ways so that more people have access to them. This is a different model than traditionally copyrighted materials. OER are defined as teaching, learning, and research resources that reside in the public domain or have been released under an intellectual property license that permits their free use and re-purposing by others.¹ Our open textbooks are openly licensed using a [Creative Commons licence](#), and are offered in various e-book formats free of charge, or as printed books that are available at cost. For more information about this project, please see the [BCcampus Open Education](#) website. If you are an instructor who is using this book for a course, please fill out our [Adoption of an Open Textbook](#) form.

1. "Open Educational Resources," *Hewlett Foundation*, <https://hewlett.org/strategy/open-educational-resources/> (accessed September 27, 2018).

Acknowledgements

Adrienne Watt

This book has been a wonderful experience in the world of open textbooks. It's amazing to see how much information is available to be shared. I would like to thank Nguyen Kim Anh of OpenStax College, for her contribution of database models and the relational design sections. I would also like to thank Dr. Gordon Russell for the section on normalization. His database resources were wonderful. Open Learning University in the UK provided me with a great ERD example. In addition, Tom Jewet provided some invaluable UML contributions.

I would also like to thank my many students over the years and a special instructor, Mitra Ramkay (BCIT). He is fondly remembered for the mentoring he provided when I first started teaching relational databases 25 years ago. Another person instrumental in getting me started in creating an open textbook is Terrie MacAloney. She was encouraging and taught me think outside the box.

A special thanks goes to my family for the constant love and support I received throughout this project.

Nelson Eng

I would like to thank the many people who helped in this edition including my students at Douglas College and my colleague Ms. Adrienne Watt.

We would like to particularly thank Lauri Aesoph at BCcampus for her perseverance and hard work while editing the book. She did an amazing job.

Chapter 1 Before the Advent of Database Systems

ADRIENNE WATT

The way in which computers manage data has come a long way over the last few decades. Today's users take for granted the many benefits found in a database system. However, it wasn't that long ago that computers relied on a much less elegant and costly approach to data management called the file-based system.

File-based System

One way to keep information on a computer is to store it in permanent files. A company system has a number of application programs; each of them is designed to manipulate data files. These application programs have been written at the request of the users in the organization. New applications are added to the system as the need arises. The system just described is called the *file-based system*.

Consider a traditional banking system that uses the file-based system to manage the organization's data shown in Figure 1.1. As we can see, there are different departments in the bank. Each has its own applications that manage and manipulate different data files. For banking systems, the programs may be used to debit or credit an account, find the balance of an account, add a new mortgage loan and generate monthly statements.



Figure 1.1. Example of a file-based system used by banks to manage data.

Disadvantages of the file-based approach

Using the file-based system to keep organizational information has a number of disadvantages. Listed below are five examples.

Data redundancy

Often, within an organization, files and applications are created by different programmers from various departments over long periods of time. This can lead to *data redundancy*, a situation that occurs in a database when a field needs to be updated in more than one table. This practice can lead to several problems such as:

- Inconsistency in data format
- The same information being kept in several different places (files)
- *Data inconsistency*, a situation where various copies of the same data are conflicting, wastes storage space and duplicates effort

Data isolation

Data isolation is a property that determines when and how changes made by one operation become visible to other concurrent users and systems. This issue occurs in a concurrency situation. This is a problem because:

- It is difficult for new applications to retrieve the appropriate data, which might be stored in various files.

Integrity problems

Problems with *data integrity* is another disadvantage of using a file-based system. It refers to the maintenance and assurance that the data in a database are correct and consistent. Factors to consider when addressing this issue are:

- Data values must satisfy certain consistency constraints that are specified in the application programs.
- It is difficult to make changes to the application programs in order to enforce new constraints.

Security problems

Security can be a problem with a file-based approach because:

- There are constraints regarding accessing privileges.
- Application requirements are added to the system in an ad-hoc manner so it is difficult to enforce constraints.

Concurrency access

Concurrency is the ability of the database to allow multiple users access to the same record without adversely affecting transaction processing. A file-based system must manage, or prevent, concurrency by the application programs. Typically, in a file-based system, when an application opens a file, that file is locked. This means that no one else has access to the file at the same time.

In database systems, concurrency is managed thus allowing multiple users access to the same record. This is an important difference between database and file-based systems.

Database Approach

The difficulties that arise from using the file-based system have prompted the development of a new approach in managing large amounts of organizational information called the *database approach*.

Databases and database technology play an important role in most areas where computers are used, including business, education and medicine. To understand the fundamentals of database systems, we will start by introducing some basic concepts in this area.

Role of databases in business

Everybody uses a database in some way, even if it is just to store information about their friends and family. That data might be written down or stored in a computer by using a word-processing program or it could be saved in a spreadsheet. However, the best way to store data is by using *database management software*. This is a powerful software tool that allows you to store, manipulate and retrieve data in a variety of different ways.

Most companies keep track of customer information by storing it in a database. This data may include customers, employees, products, orders or anything else that assists the business with its operations.

The meaning of data

Data are factual information such as measurements or statistics about objects and concepts. We use data for discussions or as part of a calculation. Data can be a person, a place, an event, an action or any one of a number of things. A single fact is an element of data, or a *data element*.

If data are information and information is what we are in the business of working with, you can start to see where you might be storing it. Data can be stored in:

- Filing cabinets
- Spreadsheets
- Folders
- Ledgers
- Lists
- Piles of papers on your desk

All of these items store information, and so too does a database. Because of the mechanical nature of databases, they have terrific power to manage and process the information they hold. This can make the information they house much more useful for your work.

With this understanding of data, we can start to see how a tool with the capacity to store a collection of data and organize it, conduct a rapid search, retrieve and process, might make a difference to how we can use data. This book and the chapters that follow are all about managing information.

Key Terms

concurrency: the ability of the database to allow multiple users access to the same record without adversely affecting transaction processing

data element: a single fact or piece of information

data inconsistency: a situation where various copies of the same data are conflicting

data isolation: a property that determines when and how changes made by one operation become visible to other concurrent users and systems

data integrity: refers to the maintenance and assurance that the data in a database are correct and consistent

data redundancy: a situation that occurs in a database when a field needs to be updated in more than one table

database approach: allows the management of large amounts of organizational information

database management software: a powerful software tool that allows you to store, manipulate and retrieve data in a variety of ways

file-based system: an application program designed to manipulate data files

Exercises

1. Discuss each of the following terms:
 1. data
 2. field
 3. record
 4. file
2. What is data redundancy?
3. Discuss the disadvantages of file-based systems.
4. Explain the difference between data and information.
5. Use Figure 1.2 (below) to answer the following questions.
 1. In the table, how many records does the file contain?
 2. How many fields are there per record?
 3. What problem would you encounter if you wanted to produce a listing by city?
 4. How would you solve this problem by altering the file structure?

	PROJECT_CODE	PROJECT_MANAGER	MANAGER_PHONE	MANAGER_ADDRESS	PROJECT_BID_PRICE
▶	21-5Z	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	\$16,833,460.00
	25-2D	Jane D. Grant	615-898-9909	218 Clark Blvd., Nashville, TN 36362	\$12,500,000.00
	25-5A	George F. Dorts	615-227-1245	124 River Dr., Franklin, TN 29185	\$32,512,420.00
	25-9T	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	\$21,563,234.00
	27-4Q	George F. Dorts	615-227-1245	124 River Dr., Franklin, TN 29185	\$10,314,545.00
	29-2D	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	\$25,559,999.00
	31-7P	William K. Moor	904-445-2719	216 Morton Rd., Stetson, FL 30155	\$56,850,000.00

Figure 1.2. Table for exercise #5, by A. Watt.

Attribution

This chapter of *Database Design* (including its images, unless otherwise noted) is a derivative copy of [Database System Concepts](#) by Nguyen Kim Anh licensed under [Creative Commons Attribution License 3.0 license](#)

The following material was written by Adrienne Watt:

1. Introduction
2. Key Terms
3. Exercises

Chapter 2 Fundamental Concepts

ADRIENNE WATT & NELSON ENG

What Is a Database?

A *database* is a shared collection of related data used to support the activities of a particular organization. A database can be viewed as a repository of data that is defined once and then accessed by various users as shown in Figure 2.1.

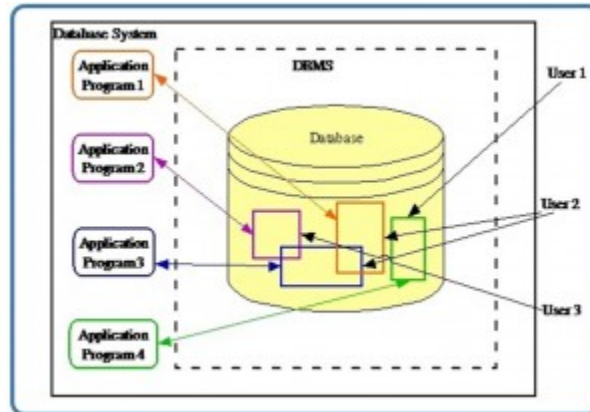


Figure 2.1. A database is a repository of data.

Database Properties

A database has the following properties:

- It is a representation of some aspect of the real world or a collection of *data elements* (facts) representing real-world information.
- A database is logical, coherent and internally consistent.
- A database is designed, built and populated with data for a specific purpose.
- Each data item is stored in a field.
- A combination of fields makes up a *table*. For example, each field in an employee table contains data about an individual employee.

A database can contain many tables. For example, a membership system may contain an address table and an individual member table as shown in Figure 2.2. Members of Science World are individuals, group homes, businesses and corporations who have an active membership to Science World. Memberships can be purchased for a one- or two-year period, and then renewed for another one- or two-year period.

Membership

ID EXPIRY DATE Prev Exp Stat Cat

Name Mrs. Res

Address

City Prov Country

Notes

Cards # Members #Years

FirstName	LastName	YYMM	G	BARCODE	V	DATE	TIME	F
Mickey	Mouse	0000	M	10000001	4	20130810	10:12:29	y
Minnie	Mouse	0000	F	10000002	4	20130810	10:12:29	y
Mighty	Mouse	0000	M	10000003	4	20130810	10:12:29	y
Door	Mouse	0000	F	10000004	4	20130810	10:12:29	y
Tom	Mouse	0000	M	10000005	4	20130810	10:12:29	y
King	Rat	0000	M	10000006	4	20130810	10:12:29	y
Man	Mouse	0000	M	10000007	4	20130810	10:12:29	y
Moose	Mouse	0000	M	10000008	4	20130810	10:12:29	y

Record: 1 of 1 | No Filter | Search

Figure 2.2. Membership system at Science World by N. Eng.

In Figure 2.2, Minnie Mouse renewed the family membership with Science World. Everyone with membership ID#100755 lives at 8932 Rodent Lane. The individual members are Mickey Mouse, Minnie Mouse, Mighty Mouse, Door Mouse, Tom Mouse, King Rat, Man Mouse and Moose Mouse.

Database Management System

A *database management system (DBMS)* is a collection of programs that enables users to create and maintain databases and control all access to them. The primary goal of a DBMS is to provide an environment that is both convenient and efficient for users to retrieve and store information.

With the database approach, we can have the traditional banking system as shown in Figure 2.3. In this bank example, a DBMS is used by the Personnel Department, the Account Department and the Loan Department to access the shared corporate database.

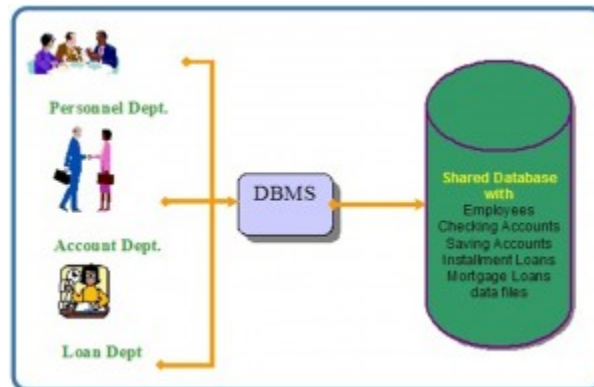


Figure 2.3. A bank database management system (DBMS).

Key Terms

data elements: facts that represent real-world information

database: a shared collection of related data used to support the activities of a particular organization

database management system (DBMS): a collection of programs that enables users to create and maintain databases and control all access to them

table: a combination of fields

Exercises

1. What is a database management system (DBMS)?
2. What are the properties of a DBMS?
3. Provide three examples of a real-world database (e.g., the library contains a database of books).

Attribution

This chapter of *Database Design* (including images, except as otherwise noted) is a derivative copy of [Database System Concepts](#) by Nguyen Kim Anh licensed under [Creative Commons Attribution License 3.0 license](#)

The following material was written by Nelson Eng:

1. Example under *Database Properties*
2. Key Terms

The following material was written by Adrienne Watt:

1. Exercises

Chapter 3 Characteristics and Benefits of a Database

ADRIENNE WATT

Managing information means taking care of it so that it works for us and is useful for the tasks we perform. By using a DBMS, the information we collect and add to its database is no longer subject to accidental disorganization. It becomes more accessible and integrated with the rest of our work. Managing information using a database allows us to become strategic users of the data we have.

We often need to access and re-sort data for various uses. These may include:

- Creating mailing lists
- Writing management reports
- Generating lists of selected news stories
- Identifying various client needs

The processing power of a database allows it to manipulate the data it houses, so it can:

- Sort
- Match
- Link
- Aggregate
- Skip fields
- Calculate
- Arrange

Because of the versatility of databases, we find them powering all sorts of projects. A database can be linked to:

- A website that is capturing registered users
- A client-tracking application for social service organizations
- A medical record system for a health care facility
- Your personal address book in your email client
- A collection of word-processed documents
- A system that issues airline reservations

Characteristics and Benefits of a Database

There are a number of characteristics that distinguish the database approach from the file-based system or approach. This chapter describes the benefits (and features) of the database system.

Self-describing nature of a database system

A database system is referred to as *self-describing* because it not only contains the database itself, but also *metadata* which defines and describes the data and relationships between tables in the database. This information is

used by the DBMS software or database users if needed. This separation of data and information about the data makes a database system totally different from the traditional file-based system in which the data definition is part of the application programs.

Insulation between program and data

In the file-based system, the structure of the data files is defined in the application programs so if a user wants to change the structure of a file, all the programs that access that file might need to be changed as well.

On the other hand, in the database approach, the data structure is stored in the system catalogue and not in the programs. Therefore, one change is all that is needed to change the structure of a file. This insulation between the programs and data is also called program-data independence.

Support for multiple views of data

A database supports multiple views of data. A *view* is a subset of the database, which is defined and dedicated for particular users of the system. Multiple users in the system might have different views of the system. Each view might contain only the data of interest to a user or group of users.

Sharing of data and multiuser system

Current database systems are designed for multiple users. That is, they allow many users to access the same database at the same time. This access is achieved through features called *concurrency control strategies*. These strategies ensure that the data accessed are always correct and that data integrity is maintained.

The design of modern multiuser database systems is a great improvement from those in the past which restricted usage to one person at a time.

Control of data redundancy

In the database approach, ideally, each data item is stored in only one place in the database. In some cases, data redundancy still exists to improve system performance, but such redundancy is controlled by application programming and kept to minimum by introducing as little redundancy as possible when designing the database.

Data sharing

The integration of all the data, for an organization, within a database system has many advantages. First, it allows for data sharing among employees and others who have access to the system. Second, it gives users the ability to generate more information from a given amount of data than would be possible without the integration.

Enforcement of integrity constraints

Database management systems must provide the ability to define and enforce certain constraints to ensure that users enter valid information and maintain data integrity. A *database constraint* is a restriction or rule that dictates what can be entered or edited in a table such as a postal code using a certain format or adding a valid city in the City field.

There are many types of database constraints. *Data type*, for example, determines the sort of data permitted in a field, for example numbers only. *Data uniqueness* such as the primary key ensures that no duplicates are entered. Constraints can be simple (field based) or complex (programming).

Restriction of unauthorized access

Not all users of a database system will have the same accessing privileges. For example, one user might have *read-only access* (i.e., the ability to read a file but not make changes), while another might have *read and write privileges*, which is the ability to both read and modify a file. For this reason, a database management system should provide a security subsystem to create and control different types of user accounts and restrict unauthorized access.

Data independence

Another advantage of a database management system is how it allows for data independence. In other words, the system data descriptions or data describing data (metadata) are separated from the application programs. This is possible because changes to the data structure are handled by the database management system and are not embedded in the program itself.

Transaction processing

A database management system must include concurrency control subsystems. This feature ensures that data remains consistent and valid during transaction processing even if several users update the same information.

Provision for multiple views of data

By its very nature, a DBMS permits many users to have access to its database either individually or simultaneously. It is not important for users to be aware of how and where the data they access is stored

Backup and recovery facilities

Backup and recovery are methods that allow you to protect your data from loss. The database system provides a separate process, from that of a network backup, for backing up and recovering data. If a hard drive fails and the

database stored on the hard drive is not accessible, the only way to recover the database is from a backup. If a computer system fails in the middle of a complex update process, the recovery subsystem is responsible for making sure that the database is restored to its original state. These are two more benefits of a database management system.

Key Terms

concurrency control strategies: features of a database that allow several users access to the same data item at the same time

data type: determines the sort of data permitted in a field, for example numbers only

data uniqueness: ensures that no duplicates are entered

database constraint: a restriction that determines what is allowed to be entered or edited in a table

metadata: defines and describes the data and relationships between tables in the database

read and write privileges: the ability to both read and modify a file

read-only access: the ability to read a file but not make changes

self-describing: a database system is referred to as self-describing because it not only contains the database itself, but also metadata which defines and describes the data and relationships between tables in the database

view: a subset of the database

Exercises

1. How is a DBMS distinguished from a file-based system?
2. What is data independence and why is it important?
3. What is the purpose of managing information?
4. Discuss the uses of databases in a business environment.
5. What is metadata?

Attribution

This chapter of *Database Design* is a derivative copy of [Database System Concepts](#) by Nguyen Kim Anh licensed under [Creative Commons Attribution License 3.0 license](#)

The following material was written by Adrienne Watt:

1. Introduction

2. Key Terms
3. Exercises

Chapter 4 Types of Data Models

ADRIENNE WATT & NELSON ENG

High-level Conceptual Data Models

High-level conceptual data models provide concepts for presenting data in ways that are close to the way people perceive data. A typical example is the entity relationship model, which uses main concepts like entities, attributes and relationships. An entity represents a real-world object such as an employee or a project. The entity has attributes that represent properties such as an employee's name, address and birthdate. A relationship represents an association among entities; for example, an employee works on many projects. A relationship exists between the employee and each project.

Record-based Logical Data Models

Record-based logical data models provide concepts users can understand but are not too far from the way data is stored in the computer. Three well-known data models of this type are relational data models, network data models and hierarchical data models.

- The *relational model* represents data as *relations*, or tables. For example, in the membership system at Science World, each membership has many members (see Figure 2.2 in Chapter 2). The membership identifier, expiry date and address information are fields in the membership. The members are individuals such as Mickey, Minnie, Mighty, Door, Tom, King, Man and Moose. Each record is said to be an *instance* of the membership table.
- The *network model* represents data as record types. This model also represents a limited type of one to many relationship called a *set type*, as shown in Figure 4.1.

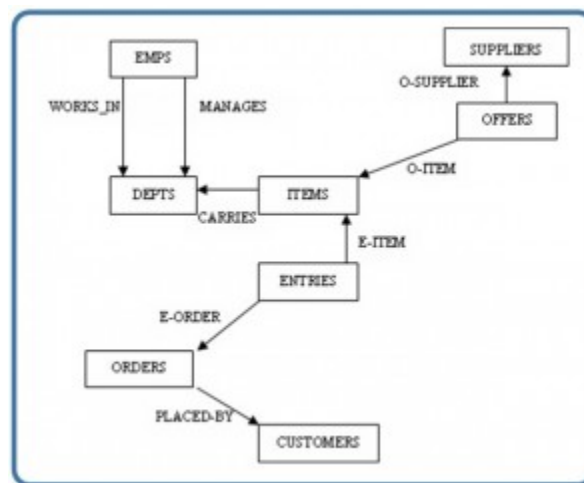


Figure 4.1. Network model diagram.

- The *hierarchical model* represents data as a hierarchical tree structure. Each branch of the hierarchy represents a number of related records. Figure 4.2 shows this schema in hierarchical model notation.

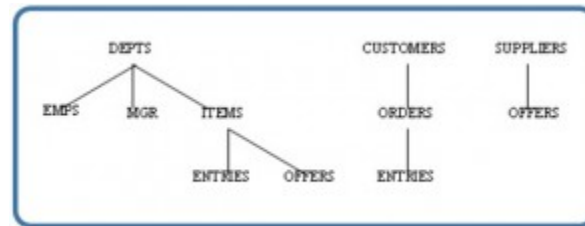


Figure 4.2. Hierarchical model diagram.

Key Terms

hierarchical model: represents data as a hierarchical tree structure

instance: a record within a table

network model: represents data as record types

relation: another term for table

relational model: represents data as relations or tables

set type: a limited type of one to many relationship

Exercises

1. What is a data model?
2. What is a high-level conceptual data model?
3. What is an entity? An attribute? A relationship?
4. List and briefly describe the common record-based logical data models.

Attribution

This chapter of *Database Design* is a derivative copy of [Database System Concepts](#) by Nguyen Kim Anh licensed under [Creative Commons Attribution License 3.0 license](#)

The following material was written by Adrienne Watt:

1. Key Terms
2. Exercises

Chapter 5 Data Modelling

ADRIENNE WATT

Data modelling is the first step in the process of database design. This step is sometimes considered to be a high-level and abstract design phase, also referred to as conceptual design. The aim of this phase is to describe:

- The data contained in the database (e.g., entities: students, lecturers, courses, subjects)
- The relationships between data items (e.g., students are supervised by lecturers; lecturers teach courses)
- The constraints on data (e.g., student number has exactly eight digits; a subject has four or six units of credit only)

In the second step, the data items, the relationships and the constraints are all expressed using the concepts provided by the high-level data model. Because these concepts do not include the implementation details, the result of the data modelling process is a (semi) formal representation of the database structure. This result is quite easy to understand so it is used as reference to make sure that all the user's requirements are met.

The third step is database design. During this step, we might have two sub-steps: one called *database logical design*, which defines a database in a data model of a specific DBMS, and another called *database physical design*, which defines the internal database storage structure, file organization or indexing techniques. These two sub-steps are database implementation and operations/user interfaces building steps.

In the database design phases, data are represented using a certain data model. The *data model* is a collection of concepts or notations for describing data, data relationships, data semantics and data constraints. Most data models also include a set of basic operations for manipulating data in the database.

Degrees of Data Abstraction

In this section we will look at the database design process in terms of specificity. Just as any design starts at a high level and proceeds to an ever-increasing level of detail, so does database design. For example, when building a home, you start with how many bedrooms and bathrooms the home will have, whether it will be on one level or multiple levels, etc. The next step is to get an architect to design the home from a more structured perspective. This level gets more detailed with respect to actual room sizes, how the home will be wired, where the plumbing fixtures will be placed, etc. The last step is to hire a contractor to build the home. That's looking at the design from a high level of abstraction to an increasing level of detail.

The database design is very much like that. It starts with users identifying the business rules; then the database designers and analysts create the database design; and then the database administrator implements the design using a DBMS.

The following subsections summarize the models in order of decreasing level of abstraction.

External models

- Represent the user's view of the database
- Contain multiple different external views

- Are closely related to the real world as perceived by each user

Conceptual models

- Provide flexible data-structuring capabilities
- Present a “community view”: the logical structure of the entire database
- Contain data stored in the database
- Show relationships among data including:
 - Constraints
 - Semantic information (e.g., business rules)
 - Security and integrity information
- Consider a database as a collection of entities (objects) of various kinds
- Are the basis for identification and high-level description of main data objects; they avoid details
- Are database independent regardless of the database you will be using

Internal models

The three best-known models of this kind are the relational data model, the network data model and the hierarchical data model. These internal models:

- Consider a database as a collection of fixed-size records
- Are closer to the physical level or file structure
- Are a representation of the database as seen by the DBMS.
- Require the designer to match the conceptual model’s characteristics and constraints to those of the selected implementation model
- Involve mapping the entities in the conceptual model to the tables in the relational model

Physical models

- Are the physical representation of the database
- Have the lowest level of abstractions
- Are how the data is stored; they deal with
 - Run-time performance
 - Storage utilization and compression
 - File organization and access methods
 - Data encryption
- Are the physical level – managed by the *operating system* (OS)
- Provide concepts that describe the details of how data are stored in the computer’s memory

Data Abstraction Layer

In a pictorial view, you can see how the different models work together. Let's look at this from the highest level, the external model.

The external model is the end user's view of the data. Typically a database is an enterprise system that serves the needs of multiple departments. However, one department is not interested in seeing other departments' data (e.g., the human resources (HR) department does not care to view the sales department's data). Therefore, one user view will differ from another.

The external model requires that the designer subdivide a set of requirements and constraints into functional modules that can be examined within the framework of their external models (e.g., human resources versus sales).

As a data designer, you need to understand all the data so that you can build an enterprise-wide database. Based on the needs of various departments, the conceptual model is the first model created.

At this stage, the conceptual model is independent of both software and hardware. It does not depend on the DBMS software used to implement the model. It does not depend on the hardware used in the implementation of the model. Changes in either hardware or DBMS software have no effect on the database design at the conceptual level.

Once a DBMS is selected, you can then implement it. This is the internal model. Here you create all the tables, constraints, keys, rules, etc. This is often referred to as the *logical design*.

The physical model is simply the way the data is stored on disk. Each database vendor has its own way of storing the data.

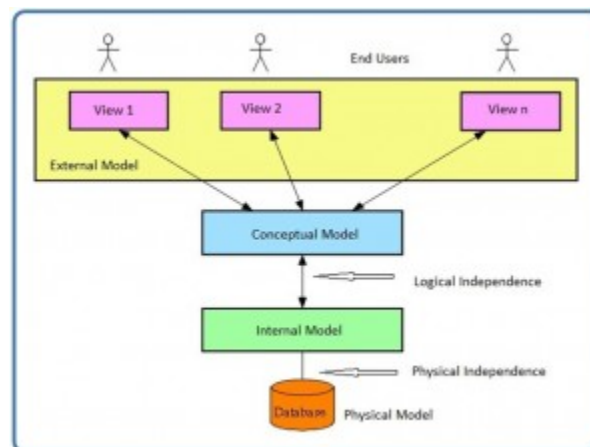


Figure 5.1. Data abstraction layers.

Schemas

A *schema* is an overall description of a database, and it is usually represented by the *entity relationship diagram (ERD)*. There are many subschemas that represent external models and thus display external views of the data. Below is a list of items to consider during the design process of a database.

- External schemas: there are multiple

- Multiple subschemas: these display multiple external views of the data
- Conceptual schema: there is only one. This schema includes data items, relationships and constraints, all represented in an ERD.
- Physical schema: there is only one

Logical and Physical Data Independence

Data independence refers to the immunity of user applications to changes made in the definition and organization of data. Data abstractions expose only those items that are important or pertinent to the user. Complexity is hidden from the database user.

Data independence and operation independence together form the feature of data abstraction. There are two types of data independence: logical and physical.

Logical data independence

A *logical schema* is a conceptual design of the database done on paper or a whiteboard, much like architectural drawings for a house. The ability to change the logical schema, without changing the *external schema* or user view, is called *logical data independence*. For example, the addition or removal of new entities, attributes or relationships to this *conceptual schema* should be possible without having to change existing external schemas or rewrite existing application programs.

In other words, changes to the logical schema (e.g., alterations to the structure of the database like adding a column or other tables) should not affect the function of the application (external views).

Physical data independence

Physical data independence refers to the immunity of the internal model to changes in the physical model. The logical schema stays unchanged even though changes are made to file organization or storage structures, storage devices or indexing strategy.

Physical data independence deals with hiding the details of the storage structure from user applications. The applications should not be involved with these issues, since there is no difference in the operation carried out against the data.

Key Terms

conceptual model: the logical structure of the entire database

conceptual schema: another term for logical schema

data independence: the immunity of user applications to changes made in the definition and organization of data

data model: a collection of concepts or notations for describing data, data relationships, data semantics and data constraints

data modelling: the first step in the process of database design

database logical design: defines a database in a data model of a specific database management system

database physical design: defines the internal database storage structure, file organization or indexing techniques

entity relationship diagram (ERD): a data model describing the database showing tables, attributes and relationships

external model: represents the user's view of the database

external schema: user view

internal model: a representation of the database as seen by the DBMS

logical data independence: the ability to change the logical schema without changing the external schema

logical design: where you create all the tables, constraints, keys, rules, etc.

logical schema: a conceptual design of the database done on paper or a whiteboard, much like architectural drawings for a house

operating system (OS): manages the physical level of the physical model

physical data independence: the immunity of the internal model to changes in the physical model

physical model: the physical representation of the database

schema: an overall description of a database

Exercises

1. Describe the purpose of a conceptual design.
2. How is a conceptual design different from a logical design?
3. What is an external model?
4. What is a conceptual model?
5. What is an internal model?
6. What is a physical model?
7. Which model does the database administrator work with?
8. Which model does the end user work with?
9. What is logical data independence?
10. What is physical data independence?

Also see *Appendix A: University Registration Data Model Example*

Attribution

This chapter of *Database Design* is a derivative copy of [Database System Concepts](#) by Nguyen Kim Anh licensed under [Creative Commons Attribution License 3.0 license](#)

The following material was written by Adrienne Watt:

1. Some or all of the introduction, degrees of data abstraction, data abstraction layer
2. Key Terms
3. Exercises

Chapter 6 Classification of Database Management Systems

ADRIENNE WATT

Database management systems can be classified based on several criteria, such as the data model, user numbers and database distribution, all described below.

Classification Based on Data Model

The most popular data model in use today is the relational data model. Well-known DBMSs like Oracle, MS SQL Server, DB2 and MySQL support this model. Other traditional models, such as hierarchical data models and network data models, are still used in industry mainly on mainframe platforms. However, they are not commonly used due to their complexity. These are all referred to as *traditional models* because they preceded the relational model.

In recent years, the newer *object-oriented data models* were introduced. This model is a database management system in which information is represented in the form of objects as used in object-oriented programming. Object-oriented databases are different from relational databases, which are table-oriented. Object-oriented database management systems (OODBMS) combine database capabilities with object-oriented programming language capabilities.

The object-oriented models have not caught on as expected so are not in widespread use. Some examples of object-oriented DBMSs are O2, ObjectStore and Jasmine.

Classification Based on User Numbers

A DBMS can be classification based on the number of users it supports. It can be a *single-user database system*, which supports one user at a time, or a *multiuser database system*, which supports multiple users concurrently.

Classification Based on Database Distribution

There are four main distribution systems for database systems and these, in turn, can be used to classify the DBMS.

Centralized systems

With a *centralized database system*, the DBMS and database are stored at a single site that is used by several other systems too. This is illustrated in Figure 6.1.

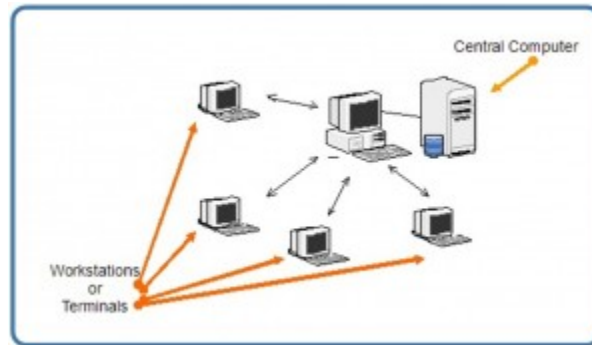


Figure 6.1. Example of a centralized database system.

In the early 1980s, many Canadian libraries used the GEAC 8000 to convert their manual card catalogues to machine-readable centralized catalogue systems. Each book catalogue had a barcode field similar to those on supermarket products.

Distributed database system

In a *distributed database system*, the actual database and the DBMS software are distributed from various sites that are connected by a computer network, as shown in Figure 6.2.

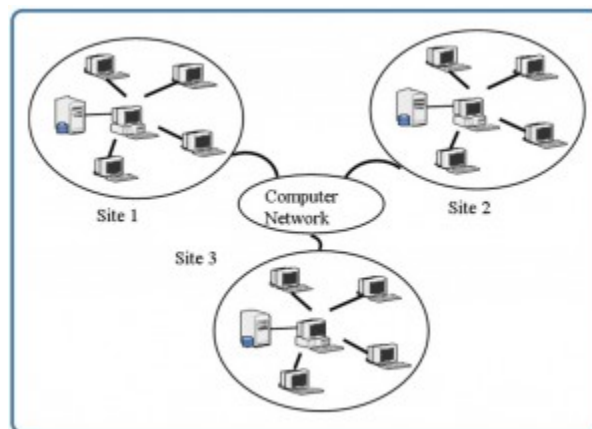


Figure 6.2. Example of a distributed database system.

Homogeneous distributed database systems

Homogeneous distributed database systems use the same DBMS software from multiple sites. Data exchange between these various sites can be handled easily. For example, library information systems by the same vendor, such as Geac Computer Corporation, use the same DBMS software which allows easy data exchange between the various Geac library sites.

Heterogeneous distributed database systems

In a *heterogeneous distributed database system*, different sites might use different DBMS software, but there is additional common software to support data exchange between these sites. For example, the various library database systems use the same machine-readable cataloguing (MARC) format to support library record data exchange.

Key Terms

centralized database system: the DBMS and database are stored at a single site that is used by several other systems too

distributed database system: the actual database and the DBMS software are distributed from various sites that are connected by a computer network

heterogeneous distributed database system: different sites might use different DBMS software, but there is additional common software to support data exchange between these sites

homogeneous distributed database systems: use the same DBMS software at multiple sites

multiuser database system: a database management system which supports multiple users concurrently

object-oriented data model: a database management system in which information is represented in the form of objects as used in object-oriented programming

single-user database system: a database management system which supports one user at a time

traditional models: data models that preceded the relational model

Exercises

1. Provide three examples of the most popular relational databases used.
2. What is the difference between centralized and distributed database systems?
3. What is the difference between homogenous distributed database systems and heterogeneous distributed database systems?

Attribution

This chapter of *Database Design* (including images, except as otherwise noted) is a derivative copy of [Database System Concepts](#) by Nguyen Kim Anh licensed under [Creative Commons Attribution License 3.0 license](#)

The following material was written by Adrienne Watt:

1. Key Terms
2. Exercises

Chapter 7 The Relational Data Model

ADRIENNE WATT

The relational data model was introduced by E. F. Codd in 1970. Currently, it is the most widely used data model.

The relational model has provided the basis for:

- Research on the theory of data/relationship/constraint
- Numerous database design methodologies
- The standard database access language called *structured query language (SQL)*
- Almost all modern commercial database management systems

The relational data model describes the world as “a collection of inter-related relations (or tables).”

Fundamental Concepts in the Relational Data Model

Relation

A *relation*, also known as a *table* or *file*, is a subset of the Cartesian product of a list of domains characterized by a name. And within a table, each row represents a group of related data values. A *row*, or record, is also known as a *tuple*. The columns in a table is a field and is also referred to as an attribute. You can also think of it this way: an attribute is used to define the record and a record contains a set of attributes.

The steps below outline the logic between a relation and its domains.

1. Given n domains are denoted by D_1, D_2, \dots, D_n
2. And r is a relation defined on these domains
3. Then $r \subseteq D_1 \times D_2 \times \dots \times D_n$

Table

A database is composed of multiple tables and each table holds the data. Figure 7.1 shows a database that contains three tables.

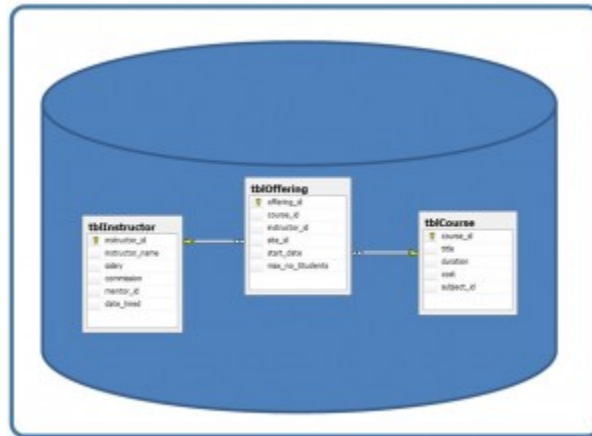


Figure 7.1. Database with three tables.

Column

A database stores pieces of information or facts in an organized way. Understanding how to use and get the most out of databases requires us to understand that method of organization.

The principal storage units are called *columns* or *fields* or *attributes*. These house the basic components of data into which your content can be broken down. When deciding which fields to create, you need to think generically about your information, for example, drawing out the common components of the information that you will store in the database and avoiding the specifics that distinguish one item from another.

Look at the example of an ID card in Figure 7.2 to see the relationship between fields and their data.

Field Name	Data
First Name	Isabelle
Family Name	Whelan
Nationality	British
Salary	109,900
Date of Birth	15 September 1983
Marital Status	Single
Shift	Mon, Wed
Place of issue	Addis Ababa
Valid until	17 December 2003

Figure 7.2. Example of an ID card by A. Watt.

Domain

A *domain* is the original sets of atomic values used to model data. By *atomic value*, we mean that each value in the domain is indivisible as far as the relational model is concerned. For example:

- The domain of Marital Status has a set of possibilities: Married, Single, Divorced.
- The domain of Shift has the set of all possible days: {Mon, Tue, Wed...}.

- The domain of Salary is the set of all floating-point numbers greater than 0 and less than 200,000.
- The domain of First Name is the set of character strings that represents names of people.

In summary, a domain is a set of acceptable values that a column is allowed to contain. This is based on various properties and the data type for the column. We will discuss data types in another chapter.

Records

Just as the content of any one document or item needs to be broken down into its constituent bits of data for storage in the fields, the link between them also needs to be available so that they can be reconstituted into their whole form. Records allow us to do this. *Records* contain fields that are related, such as a customer or an employee. As noted earlier, a tuple is another term used for record.

Records and fields form the basis of all databases. A simple table gives us the clearest picture of how records and fields work together in a database storage project.

Record ID	PubDate	Author	Title
1	26/07/1968	B. Pitt	Rights and Wrongs online
2	3/5/2000	A. Jolie	Networking for Change
3	27/02/1971	J. Carter	The Myth of Cyber Crimes
4	15/09/1983	I. Wheaton	Connecting the disconnected

Figure 7.3. Example of a simple table by A. Watt.

The simple table example in Figure 7.3 shows us how fields can hold a range of different sorts of data. This one has:

- A Record ID field: this is an ordinal number; its data type is an integer.
- A PubDate field: this is displayed as day/month/year; its data type is date.
- An Author field: this is displayed as Initial. Surname; its data type is text.
- A Title field text: free text can be entered here.

You can command the database to sift through its data and organize it in a particular way. For example, you can request that a selection of records be limited by date: 1. all before a given date, 2. all after a given date or 3. all between two given dates. Similarly, you can choose to have records sorted by date. Because the field, or record, containing the data is set up as a Date field, the database reads the information in the Date field not just as numbers separated by slashes, but rather, as dates that must be ordered according to a calendar system.

Degree

The *degree* is the number of attributes in a table. In our example in Figure 7.3, the degree is 4.

Properties of a Table

- A table has a name that is distinct from all other tables in the database.
- There are no duplicate rows; each row is distinct.
- Entries in columns are atomic. The table does not contain repeating groups or multivalued attributes.
- Entries from columns are from the same domain based on their data type including:
 - number (numeric, integer, float, smallint,...)
 - character (string)
 - date
 - logical (true or false)
- Operations combining different data types are disallowed.
- Each attribute has a distinct name.
- The sequence of columns is insignificant.
- The sequence of rows is insignificant.

Key Terms

atomic value: each value in the domain is indivisible as far as the relational model is concerned
attribute: principle storage unit in a database

column: see *attribute*

degree: number of attributes in a table

domain: the original sets of atomic values used to model data; a set of acceptable values that a column is allowed to contain

field: see *attribute*

file: see *relation*

record: contains fields that are related; see *tuple*

relation: a subset of the Cartesian product of a list of domains characterized by a name; the technical term for table or file

row: see *tuple*

structured query language (SQL): the standard database access language

table: see *relation*

tuple: a technical term for row or record

Terminology Key

Several of the terms used in this chapter are synonymous. In addition to the Key Terms above, please refer to Table 7.1 below. The terms in the Alternative 1 column are most commonly used.

Formal Terms (Codd)	Alternative 1	Alternative 2
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

Table 7.1. Terms and their synonyms by A. Watt.

Exercises

Use Table 7.2 to answer questions 1-4.

1. Using correct terminology, identify and describe all the components in Table 7.2.
2. What is the possible domain for field EmpJobCode?
3. How many records are shown?
4. How many attributes are shown?
5. List the properties of a table.

EMPLOYEE				
EMPID	EMPLNAME	EMPINIT	EMPFNAME	EMPJOBCODE
123455	Friedman	A.	Robert	12
123456	<u>Olanski</u>	D.	Delbert	18
123457	<u>Fontein</u>	G.	Juliette	15
123458	<u>Cruazona</u>	X.	Maria	18

Table 7.2. Table for exercise questions, by A. Watt.

Attribution

This chapter of *Database Design* (including images, except as otherwise noted) is a derivative copy of [Relational Design Theory](#) by Nguyen Kim Anh licensed under [Creative Commons Attribution License 3.0 license](#)

The following material was written by Adrienne Watt:

1. All or part of the sections on relations, tables, columns and degree
2. Key Terms
3. Exercises

Chapter 8 The Entity Relationship Data Model

ADRIENNE WATT

The *entity relationship (ER) data model* has existed for over 35 years. It is well suited to data modelling for use with databases because it is fairly abstract and is easy to discuss and explain. ER models are readily translated to relations. ER models, also called an ER schema, are represented by ER diagrams.

ER modelling is based on two concepts:

- Entities, defined as tables that hold specific information (data)
- Relationships, defined as the associations or interactions between entities

Here is an example of how these two concepts might be combined in an ER data model: Prof. Ba (entity) teaches (relationship) the Database Systems course (entity).

For the rest of this chapter, we will use a sample database called the COMPANY database to illustrate the concepts of the ER model. This database contains information about employees, departments and projects. Important points to note include:

- There are several departments in the company. Each department has a unique identification, a name, location of the office and a particular employee who manages the department.
- A department controls a number of projects, each of which has a unique name, a unique number and a budget.
- Each employee has a name, identification number, address, salary and birthdate. An employee is assigned to one department but can join in several projects. We need to record the start date of the employee in each project. We also need to know the direct supervisor of each employee.
- We want to keep track of the dependents for each employee. Each dependent has a name, birthdate and relationship with the employee.

Entity, Entity Set and Entity Type

An *entity* is an object in the real world with an independent existence that can be differentiated from other objects. An entity might be

- An object with physical existence (e.g., a lecturer, a student, a car)
- An object with conceptual existence (e.g., a course, a job, a position)

Entities can be classified based on their strength. An entity is considered weak if its tables are existence dependent.

- That is, it cannot exist without a relationship with another entity
- Its primary key is derived from the primary key of the parent entity
 - The Spouse table, in the COMPANY database, is a weak entity because its primary key is dependent on the Employee table. Without a corresponding employee record, the spouse record would not exist.

An entity is considered strong if it can exist apart from all of its related entities.

- Kernels are strong entities.
- A table without a foreign key or a table that contains a foreign key that can contain nulls is a strong entity

Another term to know is *entity type* which defines a collection of similar entities.

An *entity set* is a collection of entities of an entity type at a particular point of time. In an entity relationship diagram (ERD), an entity type is represented by a name in a box. For example, in Figure 8.1, the entity type is EMPLOYEE.

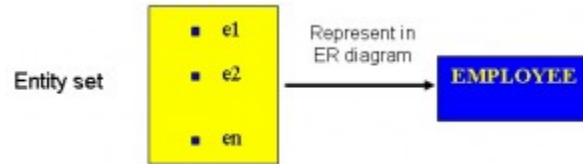


Figure 8.1. ERD with entity type EMPLOYEE.

Existence dependency

An entity's existence is dependent on the existence of the related entity. It is existence-dependent if it has a mandatory foreign key (i.e., a foreign key attribute that cannot be null). For example, in the COMPANY database, a Spouse entity is existence -dependent on the Employee entity.

Kinds of Entities

You should also be familiar with different kinds of entities including independent entities, dependent entities and characteristic entities. These are described below.

Independent entities

Independent entities, also referred to as kernels, are the backbone of the database. They are what other tables are based on. *Kernels* have the following characteristics:

- They are the building blocks of a database.
- The primary key may be simple or composite.
- The primary key is not a foreign key.
- They do not depend on another entity for their existence.

If we refer back to our COMPANY database, examples of an independent entity include the Customer table, Employee table or Product table.

Dependent entities

Dependent entities, also referred to as *derived entities*, depend on other tables for their meaning. These entities have the following characteristics:

- Dependent entities are used to connect two kernels together.
- They are said to be existence dependent on two or more tables.
- Many to many relationships become associative tables with at least two foreign keys.
- They may contain other attributes.
- The foreign key identifies each associated table.
- There are three options for the primary key:
 1. Use a composite of foreign keys of associated tables if unique
 2. Use a composite of foreign keys and a qualifying column
 3. Create a new simple primary key

Characteristic entities

Characteristic entities provide more information about another table. These entities have the following characteristics:

- They represent multivalued attributes.
- They describe other entities.
- They typically have a one to many relationship.
- The foreign key is used to further identify the characterized table.
- Options for primary key are as follows:
 1. Use a composite of foreign key plus a qualifying column
 2. Create a new simple primary key. In the COMPANY database, these might include:
 - Employee (EID, Name, Address, Age, Salary) – EID is the simple primary key.
 - EmployeePhone (EID, Phone) – EID is part of a composite primary key. Here, EID is also a foreign key.

Attributes

Each entity is described by a set of attributes (e.g., Employee = (Name, Address, Birthdate (Age), Salary).

Each attribute has a name, and is associated with an entity and a domain of legal values. However, the information about attribute domain is not presented on the ERD.

In the entity relationship diagram, shown in Figure 8.2, each attribute is represented by an oval with a name inside.

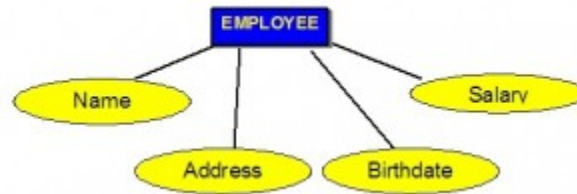


Figure 8.2. How attributes are represented in an ERD.

Types of Attributes

There are a few types of attributes you need to be familiar with. Some of these are to be left as is, but some need to be adjusted to facilitate representation in the relational model. This first section will discuss the types of attributes. Later on we will discuss fixing the attributes to fit correctly into the relational model.

Simple attributes

Simple attributes are those drawn from the atomic value domains; they are also called *single-valued attributes*. In the COMPANY database, an example of this would be: Name = {John} ; Age = {23}

Composite attributes

Composite attributes are those that consist of a hierarchy of attributes. Using our database example, and shown in Figure 8.3, Address may consist of Number, Street and Suburb. So this would be written as → Address = {59 + 'Meek Street' + 'Kingsford'}

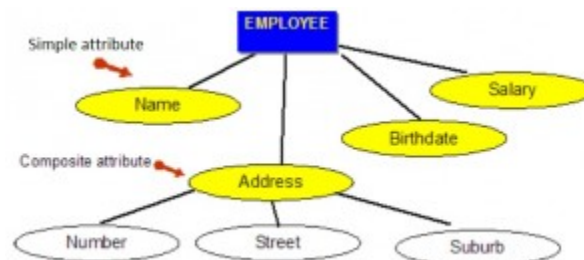


Figure 8.3. An example of composite attributes.

Multivalued attributes

Multivalued attributes are attributes that have a set of values for each entity. An example of a multivalued attribute from the COMPANY database, as seen in Figure 8.4, are the degrees of an employee: BSc, MIT, PhD.

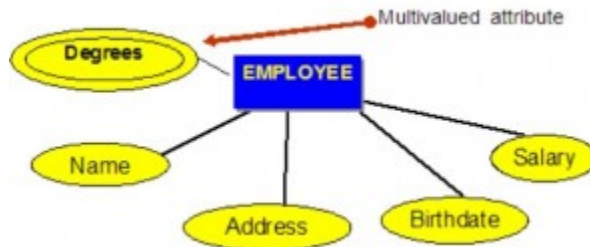


Figure 8.4. Example of a multivalued attribute.

Derived attributes

Derived attributes are attributes that contain values calculated from other attributes. An example of this can be seen in Figure 8.5. Age can be derived from the attribute Birthdate. In this situation, Birthdate is called a *stored attribute*, which is physically saved to the database.

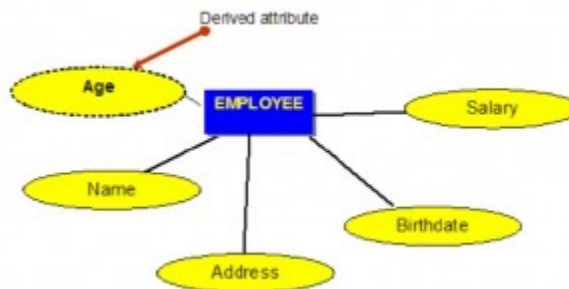


Figure 8.5. Example of a derived attribute.

Keys

An important constraint on an entity is the key. The key is an attribute or a group of attributes whose values can be used to uniquely identify an individual entity in an entity set.

Types of Keys

There are several types of keys. These are described below.

Candidate key

A *candidate key* is a simple or composite key that is unique and minimal. It is unique because no two rows in a table may have the same value at any time. It is minimal because every column is necessary in order to attain uniqueness.

From our COMPANY database example, if the entity is **Employee**(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID), possible candidate keys are:

- EID, SIN
- First Name and Last Name – assuming there is no one else in the company with the same name
- Last Name and DepartmentID – assuming two people with the same last name don't work in the same department

Composite key

A *composite key* is composed of two or more attributes, but it must be minimal.

Using the example from the candidate key section, possible composite keys are:

- First Name and Last Name – assuming there is no one else in the company with the same name
- Last Name and Department ID – assuming two people with the same last name don't work in the same department

Primary key

The primary key is a candidate key that is selected by the database designer to be used as an identifying mechanism for the whole entity set. It must uniquely identify tuples in a table and not be null. The primary key is indicated in the ER model by underlining the attribute.

- A candidate key is selected by the designer to uniquely identify tuples in a table. It must not be null.
- A key is chosen by the database designer to be used as an identifying mechanism for the whole entity set. This is referred to as the primary key. This key is indicated by underlining the attribute in the ER model.

In the following example, EID is the primary key:

Employee(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID)

Secondary key

A *secondary key* is an attribute used strictly for retrieval purposes (can be composite), for example: Phone and Last Name.

Alternate key

Alternate keys are all candidate keys not chosen as the primary key.

Foreign key

A *foreign key (FK)* is an attribute in a table that references the primary key in another table OR it can be null. Both foreign and primary keys must be of the same data type.

In the COMPANY database example below, DepartmentID is the foreign key:

Employee(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID)

Nulls

A *null* is a special symbol, independent of data type, which means either unknown or inapplicable. It does not mean zero or blank. Features of null include:

- No data entry
- Not permitted in the primary key
- Should be avoided in other attributes
- Can represent
 - An unknown attribute value
 - A known, but missing, attribute value
 - A “not applicable” condition
- Can create problems when functions such as COUNT, AVERAGE and SUM are used
- Can create logical problems when relational tables are linked

NOTE: The result of a comparison operation is null when either argument is null. The result of an arithmetic operation is null when either argument is null (except functions that ignore nulls).

Example of how null can be used

Use the Salary table (Salary_tbl) in Figure 8.6 to follow an example of how null can be used.

Salary_tbl

emp#	jobName	salary	commission
E10	Sales	12500	32090
E11	Null	25000	8000
E12	Sales	44000	0
E13	Sales	44000	Null

Figure 8.6. Salary table for null example, by A. Watt.

To begin, find all employees (emp#) in Sales (under the jobName column) whose salary plus commission are greater than 30,000.

- SELECT emp# FROM Salary_tbl
- WHERE jobName = Sales AND
- (commission + salary) > 30,000 -> E10 and E12

This result does not include E13 because of the null value in the commission column. To ensure that the row with the null value is included, we need to look at the individual fields. By adding commission and salary for employee E13, the result will be a null value. The solution is shown below.

- SELECT emp# FROM Salary_tbl
- WHERE jobName = Sales AND
- (commission > 30000 OR
- salary > 30000 OR
- (commission + salary) > 30,000 ->E10 and E12 and E13

Relationships

Relationships are the glue that holds the tables together. They are used to connect related information between tables.

Relationship strength is based on how the primary key of a related entity is defined. A weak, or non-identifying, relationship exists if the primary key of the related entity does not contain a primary key component of the parent entity. Company database examples include:

- Customer(**CustID**, CustName)
- Order(**OrderID**, CustID, Date)

A strong, or identifying, relationship exists when the primary key of the related entity contains the primary key component of the parent entity. Examples include:

- Course(**CrsCode**, DeptCode, Description)
- Class(**CrsCode**, **Section**, ClassTime...)

Types of Relationships

Below are descriptions of the various types of relationships.

One to many (1:M) relationship

A one to many (1:M) relationship should be the norm in any relational database design and is found in all relational database environments. For example, one department has many employees. Figure 8.7 shows the relationship of one of these employees to the department.

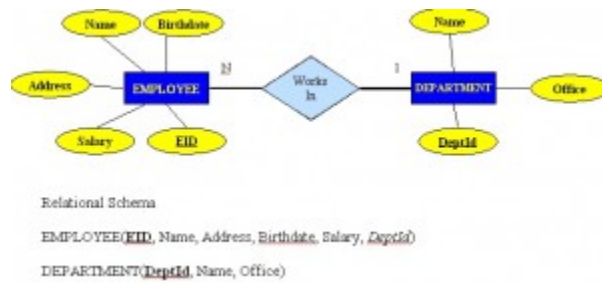


Figure 8.7. Example of a one to many relationship.

One to one (1:1) relationship

A one to one (1:1) relationship is the relationship of one entity to only one other entity, and vice versa. It should be rare in any relational database design. In fact, it could indicate that two entities actually belong in the same table.

An example from the COMPANY database is one employee is associated with one spouse, and one spouse is associated with one employee.

Many to many (M:N) relationships

For a many to many relationship, consider the following points:

- It cannot be implemented as such in the relational model.
- It can be changed into two 1:M relationships.
- It can be implemented by breaking up to produce a set of 1:M relationships.
- It involves the implementation of a composite entity.
- Creates two or more 1:M relationships.
- The composite entity table must contain at least the primary keys of the original tables.
- The linking table contains multiple occurrences of the foreign key values.
- Additional attributes may be assigned as needed.
- It can avoid problems inherent in an M:N relationship by creating a composite entity or bridge entity. For example, an employee can work on many projects OR a project can have many employees working on it, depending on the business rules. Or, a student can have many classes and a class can hold many students.

Figure 8.8 shows another another aspect of the M:N relationship where an employee has different start dates for different projects. Therefore, we need a JOIN table that contains the EID, Code and StartDate.

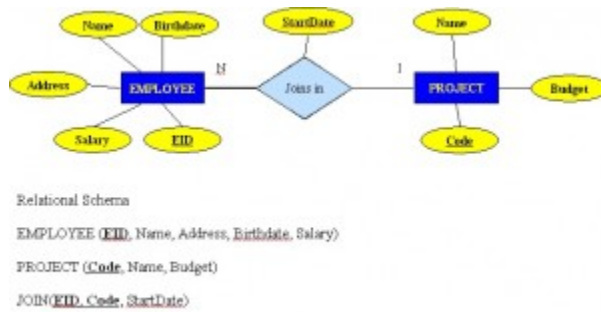


Figure 8.8. Example where employee has different start dates for different projects.

Example of mapping an M:N binary relationship type

- For each M:N binary relationship, identify two relations.
- A and B represent two entity types participating in R.
- Create a new relation S to represent R.
- S needs to contain the PKs of A and B. These together can be the PK in the S table OR these together with another simple attribute in the new table R can be the PK.
- The combination of the primary keys (A and B) will make the primary key of S.

Unary relationship (recursive)

A *unary relationship*, also called *recursive*, is one in which a relationship exists between occurrences of the same entity set. In this relationship, the primary and foreign keys are the same, but they represent two entities with different roles. See Figure 8.9 for an example.

For some entities in a unary relationship, a separate column can be created that refers to the primary key of the same entity set.

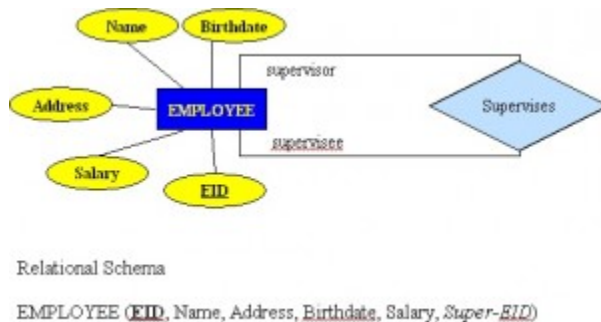


Figure 8.9. Example of a unary relationship.

Ternary Relationships

A *ternary relationship* is a relationship type that involves many to many relationships between three tables.

Refer to Figure 8.10 for an example of mapping a ternary relationship type. Note *n-ary* means multiple tables in a relationship. (Remember, N = many.)

- For each *n-ary* (> 2) relationship, create a new relation to represent the relationship.
- The primary key of the new relation is a combination of the primary keys of the participating entities that hold the N (many) side.
- In most cases of an *n-ary* relationship, all the participating entities hold a **many** side.

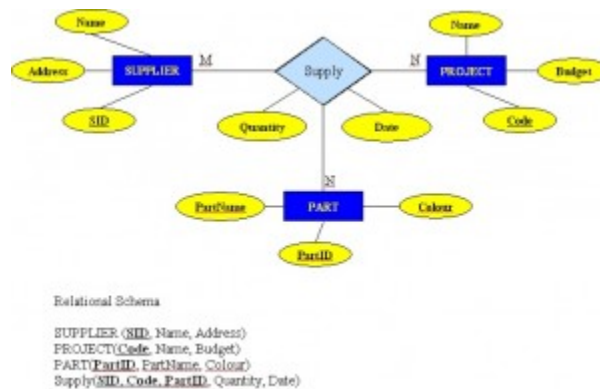


Figure 8.10. Example of a ternary relationship.

Key Terms

alternate key: all candidate keys not chosen as the primary key
candidate key: a simple or composite key that is unique (no two rows in a table may have the same value) and minimal (every column is necessary)

characteristic entities: entities that provide more information about another table

composite attributes: attributes that consist of a hierarchy of attributes

composite key: composed of two or more attributes, but it must be minimal

dependent entities: these entities depend on other tables for their meaning

derived attributes: attributes that contain values calculated from other attributes

derived entities: see *dependent entities*

EID: employee identification (ID)

entity: a thing or object in the real world with an independent existence that can be differentiated from other objects

entity relationship (ER) data model: also called an ER schema, are represented by ER diagrams. These are well suited to data modelling for use with databases.

entity relationship schema: see *entity relationship data model*

entity set: a collection of entities of an entity type at a point of time

entity type: a collection of similar entities

foreign key (FK): an attribute in a table that references the primary key in another table OR it can be null

independent entity: as the building blocks of a database, these entities are what other tables are based on

kernel: see *independent entity*

key: an attribute or group of attributes whose values can be used to uniquely identify an individual entity in an entity set

multivalued attributes: attributes that have a set of values for each entity

n-ary: multiple tables in a relationship

null: a special symbol, independent of data type, which means either unknown or inapplicable; it does not mean zero or blank

recursive relationship: see *unary relationship*

relationships: the associations or interactions between entities; used to connect related information between tables

relationship strength: based on how the primary key of a related entity is defined

secondary key an attribute used strictly for retrieval purposes

simple attributes: drawn from the atomic value domains

SIN: social insurance number

single-valued attributes: see *simple attributes*

stored attribute: saved physically to the database

ternary relationship: a relationship type that involves many to many relationships between three tables.

unary relationship: one in which a relationship exists between occurrences of the same entity set.

Exercises

1. What two concepts are ER modelling based on?
2. The database in Figure 8.11 is composed of two tables. Use this figure to answer questions 2.1 to 2.5.

DIRECTOR		
DIRNUM	DIRNAME	DIRDOB
100	<u>J.Broadway</u>	01/08/39
101	<u>J.Namath</u>	11/12/48
102	<u>W.Blake</u>	06/15/44

PLAY		
PLAYNO	PLAYNAME	DIRNUM
1001	Cat on a cold bare roof	102
1002	Hold the mayo, pass the bread	101
1003	I never promised you coffee	102
1004	Silly putty goes to Texas	100
1005	See no sound, hear no sight	101
1006	<u>Starstruck in Biloxi</u>	102
1007	Stranger in parrot ice	101

Figure 8.11. Director and Play tables for question 2, by A. Watt.

1. Identify the primary key for each table.
 2. Identify the foreign key in the PLAY table.
 3. Identify the candidate keys in both tables.
 4. Draw the ER model.
 5. Does the PLAY table exhibit referential integrity? Why or why not?
3. Define the following terms (you may need to use the Internet for some of these):
- schema
 - host language
 - data sublanguage
 - data definition language
 - unary relation
 - foreign key
 - virtual relation
 - connectivity
 - composite key
 - linking table
4. The RRE Trucking Company database includes the three tables in Figure 8.12. Use Figure 8.12 to answer questions 4.1 to 4.5.

TRUCK					
TNUM	BASENUM	TYPENUM	TMILES	TBOUGHT	TSERIAL
1001	501	1	5900.2	11/08/90	aa-125
1002	502	2	64523.9	11/08/90	ac-213
1003	501	2	32116.0	09/29/91	ac-215
1004		2	3256.9	01/14/92	ac-315

BASE				
BASENUM	BASECITY	BASESTATE	BASEPHON	BASEMGR
501	Dallas	TX	893-9870	J. Jones
502	New York	NY	234-7689	K. Lee

TYPE	
TYPENUM	TYPDESC
1	single box, double axle
2	tandem trailer, single axle

Figure 8.12. Truck, Base and Type tables for question 4, by A. Watt.

1. Identify the primary and foreign key(s) for each table.
2. Does the TRUCK table exhibit entity and referential integrity? Why or why not? Explain your answer.
3. What kind of relationship exists between the TRUCK and BASE tables?
4. How many entities does the TRUCK table contain ?
5. Identify the TRUCK table candidate key(s).

Customer		
CustID	CustName	AcctNo.
100	Joe Smith	010839
101	Andy Blake	111248
102	Sue Brown	061544

BookOrders			
OrderID	Title	CustID	Price
1001	The Dark Tower	102	12.00
1002	Incubus Dreams	101	19.99
1003	Song of Susannah	102	23.00
1004	The Time Traveler's Wife	100	21.00
1005	The Dark Tower	101	12.00
1006	Tanequil	102	15.00
1007	Song of Susannah	101	23.00

Figure 8.13. Customer and BookOrders tables for question 5, by A. Watt.

5. Suppose you are using the database in Figure 8.13, composed of the two tables. Use Figure 8.13 to answer questions 5.1 to 5.6.
 1. Identify the primary key in each table.

2. Identify the foreign key in the BookOrders table.
 3. Are there any candidate keys in either table?
 4. Draw the ER model.
 5. Does the BookOrders table exhibit referential integrity? Why or why not?
 6. Do the tables contain redundant data? If so which table(s) and what is the redundant data?
6. Looking at the student table in Figure 8.14, list all the possible candidate keys. Why did you select these?

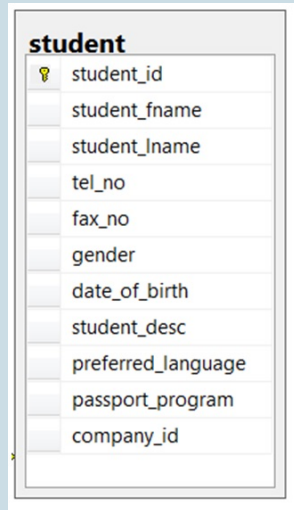


Figure 8.14. Student table for question 6, by A. Watt.

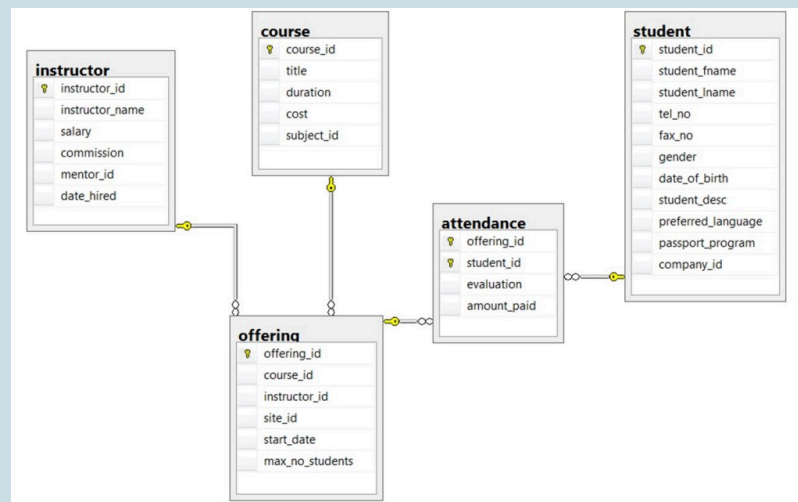


Figure 8.15. ERD of school database for questions 7-10, by A. Watt.

Use the ERD of a school database in Figure 8.15 to answer questions 7 to 10.

7. Identify all the kernels and dependent and characteristic entities in the ERD.

8. Which of the tables contribute to weak relationships? Strong relationships?
9. Looking at each of the tables in the school database in Figure 8.15, which attribute could have a NULL value? Why?
10. Which of the tables were created as a result of many to many relationships?

Also see *Appendix B: Sample ERD Exercises*

Attribution

This chapter of *Database Design* (including images, except as otherwise noted) is a derivative copy of [Data Modeling Using Entity-Relationship Model](#) by Nguyen Kim Anh licensed under [Creative Commons Attribution License 3.0 license](#)

The following material was written by Adrienne Watt:

1. Nulls section and example
2. Key Terms
3. Exercises

Chapter 9 Integrity Rules and Constraints

ADRIENNE WATT & NELSON ENG

Constraints are a very important feature in a relational model. In fact, the relational model supports the well-defined theory of constraints on attributes or tables. Constraints are useful because they allow a designer to specify the semantics of data in the database. *Constraints* are the rules that force DBMSs to check that data satisfies the semantics.

Domain Integrity

Domain restricts the values of attributes in the relation and is a constraint of the relational model. However, there are real-world semantics for data that cannot be specified if used only with domain constraints. We need more specific ways to state what data values are or are not allowed and which format is suitable for an attribute. For example, the Employee ID (EID) must be unique or the employee Birthdate is in the range [Jan 1, 1950, Jan 1, 2000]. Such information is provided in logical statements called *integrity constraints*.

There are several kinds of integrity constraints, described below.

Entity integrity

To ensure *entity integrity*, it is required that every table have a primary key. Neither the PK nor any part of it can contain null values. This is because null values for the primary key mean we cannot identify some rows. For example, in the EMPLOYEE table, Phone cannot be a primary key since some people may not have a telephone.

Referential integrity

Referential integrity requires that a foreign key must have a matching primary key or it must be null. This constraint is specified between two tables (parent and child); it maintains the correspondence between rows in these tables. It means the reference from a row in one table to another table must be valid.

Examples of referential integrity constraint in the Customer/Order database of the Company:

- Customer(**CustID**, CustName)
- Order(**OrderID**, CustID, OrderDate)

To ensure that there are no orphan records, we need to enforce referential integrity. An *orphan record* is one whose foreign key FK value is not found in the corresponding entity – the entity where the PK is located. Recall that a typical join is between a PK and FK.

The referential integrity constraint states that the customer ID (CustID) in the Order table must match a valid CustID in the Customer table. Most relational databases have declarative referential integrity. In other words, when the tables are created the referential integrity constraints are set up.

Here is another example from a Course/Class database:

- Course(**CrsCode**, DeptCode, Description)
- Class(**CrsCode**, **Section**, ClassTime)

The referential integrity constraint states that CrsCode in the Class table must match a valid CrsCode in the Course table. In this situation, it's not enough that the CrsCode and Section in the Class table make up the PK, we must also enforce referential integrity.

When setting up referential integrity it is important that the PK and FK have the same data types and come from the same domain, otherwise the relational database management system (RDBMS) will not allow the join. RDBMS is a popular database system that is based on the relational model introduced by E. F. Codd of IBM's San Jose Research Laboratory. Relational database systems are easier to use and understand than other database systems.

Referential integrity in Microsoft Access

In Microsoft (MS) Access, referential integrity is set up by joining the PK in the Customer table to the CustID in the Order table. See Figure 9.1 for a view of how this is done on the Edit Relationships screen in MS Access.

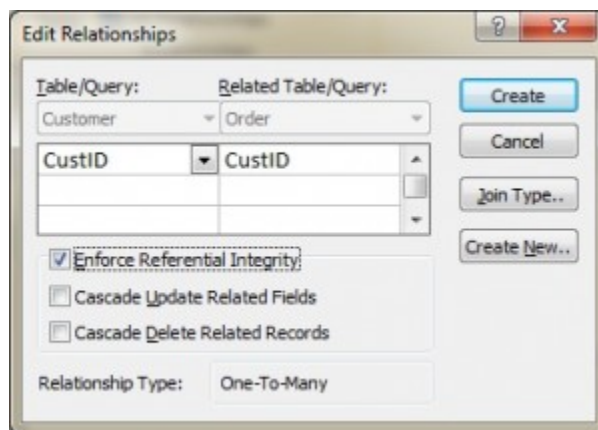


Figure 9.1. Referential access in MS Access, by A. Watt.

Referential integrity using Transact-SQL (MS SQL Server)

When using Transact-SQL, the referential integrity is set when creating the Order table with the FK. Listed below are the statements showing the FK in the Order table referencing the PK in the Customer table.

```
CREATE TABLE Customer
( CustID INTEGER PRIMARY KEY,
  CustName CHAR(35) )
```

```
CREATE TABLE Orders
( OrderID INTEGER PRIMARY KEY,
  CustID INTEGER REFERENCES Customer(CustID),
  OrderDate DATETIME )
```

Foreign key rules

Additional foreign key rules may be added when setting referential integrity, such as what to do with the child rows (in the Orders table) when the record with the PK, part of the parent (Customer), is deleted or changed (updated). For example, the Edit Relationships window in MS Access (see Figure 9.1) shows two additional options for FK rules: Cascade Update and Cascade Delete. If these are not selected, the system will prevent the deletion or update of PK values in the parent table (Customer table) if a child record exists. The child record is any record with a matching PK.

In some databases, an additional option exists when selecting the Delete option called Set to Null. In this is chosen, the PK row is deleted, but the FK in the child table is set to NULL. Though this creates an orphan row, it is acceptable.

Enterprise Constraints

Enterprise constraints – sometimes referred to as semantic constraints – are additional rules specified by users or database administrators and can be based on multiple tables.

Here are some examples.

- A class can have a maximum of 30 students.
- A teacher can teach a maximum of four classes per semester.
- An employee cannot take part in more than five projects.
- The salary of an employee cannot exceed the salary of the employee's manager.

Business Rules

Business rules are obtained from users when gathering requirements. The requirements-gathering process is very important, and its results should be verified by the user before the database design is built. If the business rules are incorrect, the design will be incorrect, and ultimately the application built will not function as expected by the users.

Some examples of business rules are:

- A teacher can teach many students.
- A class can have a maximum of 35 students.
- A course can be taught many times, but by only one instructor.
- Not all teachers teach classes.

Cardinality and connectivity

Business rules are used to determine cardinality and connectivity. *Cardinality* describes the relationship between two data tables by expressing the minimum and maximum number of entity occurrences associated with one occurrence of a related entity. In Figure 9.2, you can see that cardinality is represented by the innermost markings on the relationship symbol. In this figure, the cardinality is 0 (zero) on the right and 1 (one) on the left.

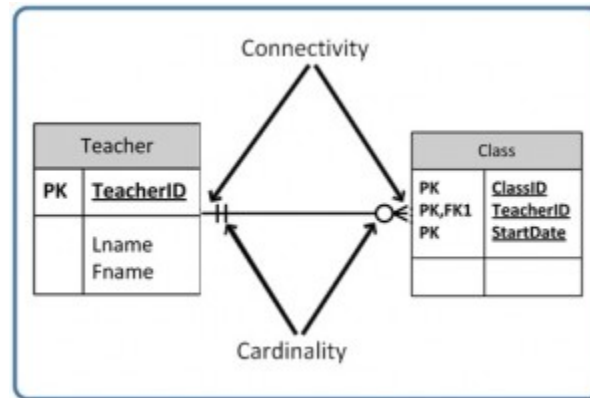


Figure 9.2. Position of connectivity and cardinality on a relationship symbol, by A. Watt.

The outermost symbol of the relationship symbol, on the other hand, represents the connectivity between the two tables. *Connectivity* is the relationship between two tables, e.g., one to one or one to many. The only time it is zero is when the FK can be null. When it comes to participation, there are three options to the relationship between these entities: either 0 (zero), 1 (one) or many. In Figure 9.2, for example, the connectivity is 1 (one) on the outer, left-hand side of this line and many on the outer, right-hand side.

Figure 9.3. shows the symbol that represents a one to many relationship.



Figure 9.3.

In Figure 9.4, both inner (representing cardinality) and outer (representing connectivity) markers are shown. The left side of this symbol is read as minimum 1 and maximum 1. On the right side, it is read as: minimum 1 and maximum many.



Figure 9.4.

Relationship Types

The line that connects two tables, in an ERD, indicates the *relationship type* between the tables: either identifying

or non-identifying. An *identifying relationship* will have a solid line (where the PK contains the FK). A *non-identifying relationship* is indicated by a broken line and does not contain the FK in the PK. See the section in Chapter 8 that discusses weak and strong relationships for more explanation.

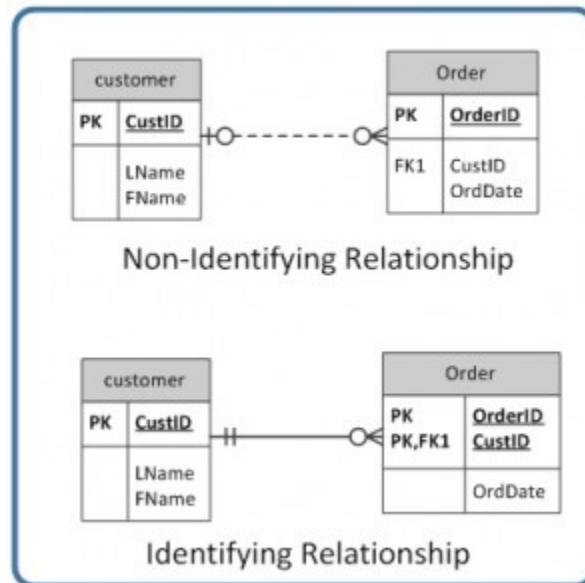


Figure 9.5. Identifying and non-identifying relationship, by A. Watt.

Optional relationships

In an *optional relationship*, the FK can be null or the parent table does not need to have a corresponding child table occurrence. The symbol, shown in Figure 9.6, illustrates one type with a zero and three prongs (indicating many) which is interpreted as zero OR many.



Figure 9.6.

For example, if you look at the Order table on the right-hand side of Figure 9.7, you'll notice that a customer doesn't need to place an order to be a customer. In other words, the **many side** is optional.

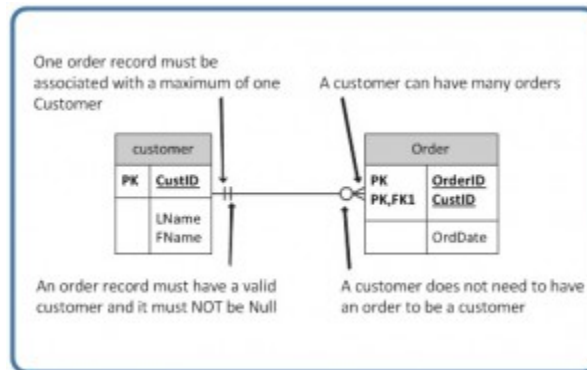


Figure 9.7. Example usage of a zero to many optional relationship symbol, by A. Watt.

The relationship symbol in Figure 9.7 can also be read as follows:

- Left side: The order entity must contain a minimum of one related entity in the Customer table and a maximum of one related entity.
- Right side: A customer can place a minimum of zero orders or a maximum of many orders.

Figure 9.8 shows another type of optional relationship symbol with a zero and one, meaning zero OR one. The **one** side is optional.



Figure 9.8.

Figure 9.9 gives an example of how a zero to one symbol might be used.

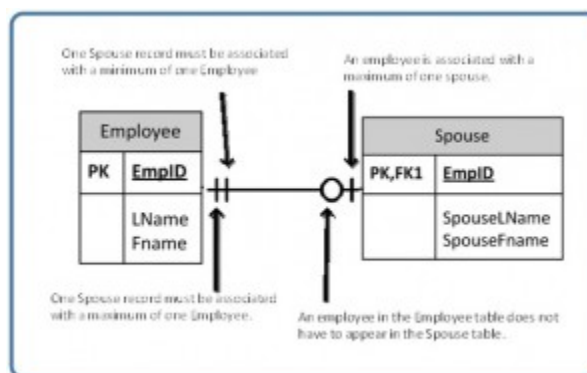


Figure 9.9. Example usage of a zero to one optional relationship symbol, by A. Watt.

Mandatory relationships

In a *mandatory relationship*, one entity occurrence requires a corresponding entity occurrence. The symbol for this relationship shows *one and only one* as shown in Figure 9.10. The one side is mandatory.



Figure 9.10

See Figure 9.11 for an example of how the one and only one mandatory symbol is used.

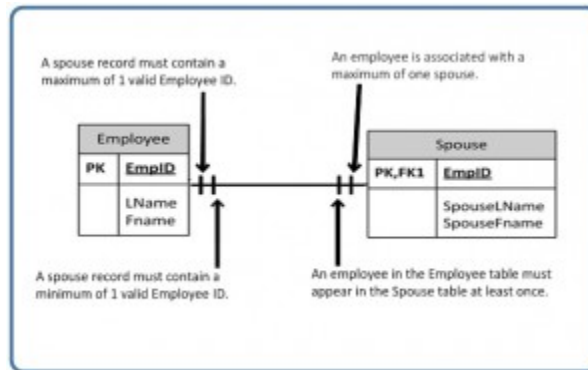


Figure 9.11. Example of a one and only one mandatory relationship symbol, by A. Watt.

Figure 9.12 illustrates what a one to many relationship symbol looks like where the **many side** is mandatory.



Figure 9.12.

Refer to Figure 9.13 for an example of how the one to many symbol may be used.

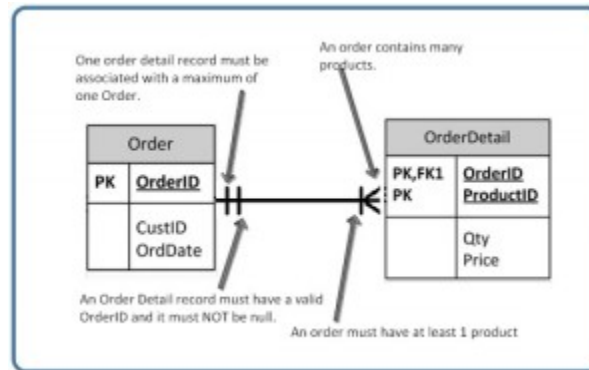


Figure 9.13. Example of a one to many mandatory relationship symbol, by A. Watt.

So far we have seen that the innermost side of a relationship symbol (on the left-side of the symbol in Figure 9.14) can have a 0 (zero) cardinality and a connectivity of many (shown on the right-side of the symbol in Figure 9.14), or one (not shown).



Figure 9.14

However, it cannot have a connectivity of 0 (zero), as displayed in Figure 9.15. The connectivity can only be 1.



Figure 9.15.

The connectivity symbols show maximums. So if you think about it logically, if the connectivity symbol on the left side shows 0 (zero), then there would be no connection between the tables.

The way to read a relationship symbol, such as the one in Figure 9.16, is as follows.

- The CustID in the Order table must also be found in the Customer table a minimum of 0 and a maximum of 1 times.
- The 0 means that the CustID in the Order table may be null.
- The left-most 1 (right before the 0 representing connectivity) says that if there is a CustID in the Order table, it can only be in the Customer table once.
- When you see the 0 symbol for cardinality, you can assume two things:
 1. the FK in the Order table allows nulls, and
 2. the FK is not part of the PK since PKs must not contain null values.

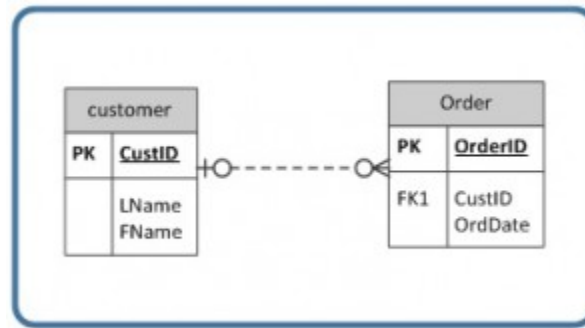


Figure 9.16. The relationship between a Customer table and an Order table, by A. Watt.

Key Terms

business rules: obtained from users when gathering requirements and are used to determine cardinality

cardinality: expresses the minimum and maximum number of entity occurrences associated with one occurrence of a related entity

connectivity: the relationship between two tables, e.g., one to one or one to many

constraints: the rules that force DBMSs to check that data satisfies the semantics

entity integrity: requires that every table have a primary key; neither the primary key, nor any part of it, can contain null values

identifying relationship: where the primary key contains the foreign key; indicated in an ERD by a solid line

integrity constraints: logical statements that state what data values are or are not allowed and which format is suitable for an attribute

mandatory relationship: one entity occurrence requires a corresponding entity occurrence.

non-identifying relationship: does not contain the foreign key in the primary key; indicated in an ERD by a dotted line

optional relationship: the FK can be null or the parent table does not need to have a corresponding child table occurrence

orphan record: a record whose foreign key value is not found in the corresponding entity – the entity where the primary key is located

referential integrity: requires that a foreign key must have a matching primary key or it must be null

relational database management system (RDBMS): a popular database system based on the relational model introduced by E. F. Codd of IBM's San Jose Research Laboratory

relationship type: the type of relationship between two tables in an ERD (either identifying or non-identifying); this relationship is indicated by a line drawn between the two tables.

Exercises

Read the following description and then answer questions 1-5 at the end.

The swim club database in Figure 9.17 has been designed to hold information about students who are enrolled in swim classes. The following information is stored: students, enrollment, swim classes, pools where classes are held, instructors for the classes, and various levels of swim classes. Use Figure 9.17 to answer questions 1 to 5.

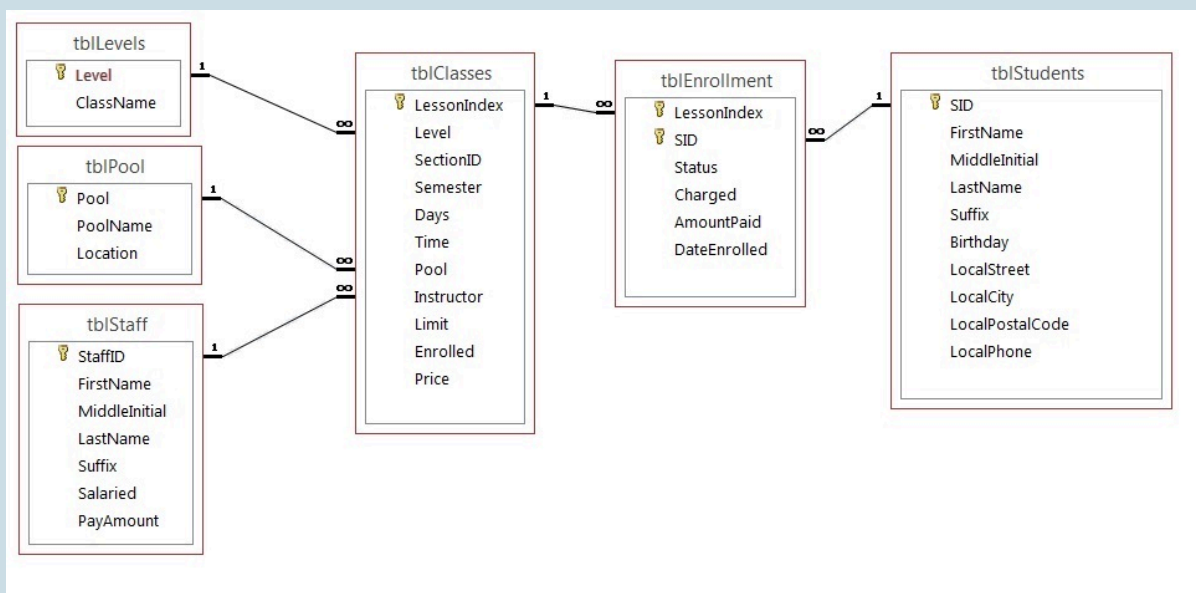


Figure 9.17. ERD for questions 1-5. (Diagram by A. Watt.)

The primary keys are identified below. The following data types are defined in the SQL Server.

tblLevels

Level – Identity PK

ClassName – text 20 – nulls are not allowed

tblPool

Pool – Identity PK

PoolName – text 20 – nulls are not allowed

Location – text 30

tblStaff

StaffID – Identity PK

FirstName – text 20

MiddleInitial – text 3

LastName – text 30

Suffix – text 3

Salaried – Bit

PayAmount – money

tblClasses

LessonIndex – Identity PK

Level – Integer FK

SectionID – Integer

Semester – TinyInt

Days – text 20

Time – datetime (formatted for time)

Pool – Integer FK

Instructor – Integer FK

Limit – TinyInt

Enrolled – TinyInt

Price – money

tblEnrollment

LessonIndex – Integer FK

SID – Integer FK (LessonIndex and SID) Primary Key

Status – text 30

Charged – bit

AmountPaid – money

DateEnrolled – datetime

tblStudents

SID – Identity PK

FirstName – text 20

MiddleInitial – text 3

LastName – text 30

Suffix – text 3

Birthday – datetime

LocalStreet – text 30

LocalCity – text 20

LocalPostalCode – text 6

LocalPhone – text 10

Implement this schema in SQL Server or access (you will need to pick comparable data types). Submit a screenshot of your ERD in the database.

1. Explain the relationship rules for each relationship (e.g., tblEnrollment and tblStudents: A student can enroll in many classes).
2. Identify cardinality for each relationship, assuming the following rules:

- A pool may or may not ever have a class.
 - The levels table must always be associated with at least one class.
 - The staff table may not have ever taught a class.
 - All students must be enrolled in at least one class.
 - The class must have students enrolled in it.
 - The class must have a valid pool.
 - The class may not have an instructor assigned.
 - The class must always be associated with an existing level.
3. Which tables are weak and which tables are strong (covered in an earlier chapter)?
 4. Which of the tables are non-identifying and which are identifying?

Image Attributions

Figures 9.3, 9.4, 9.6, 9.8, 9.10, 9.12, 9.14 and 9.15 by A. Watt.