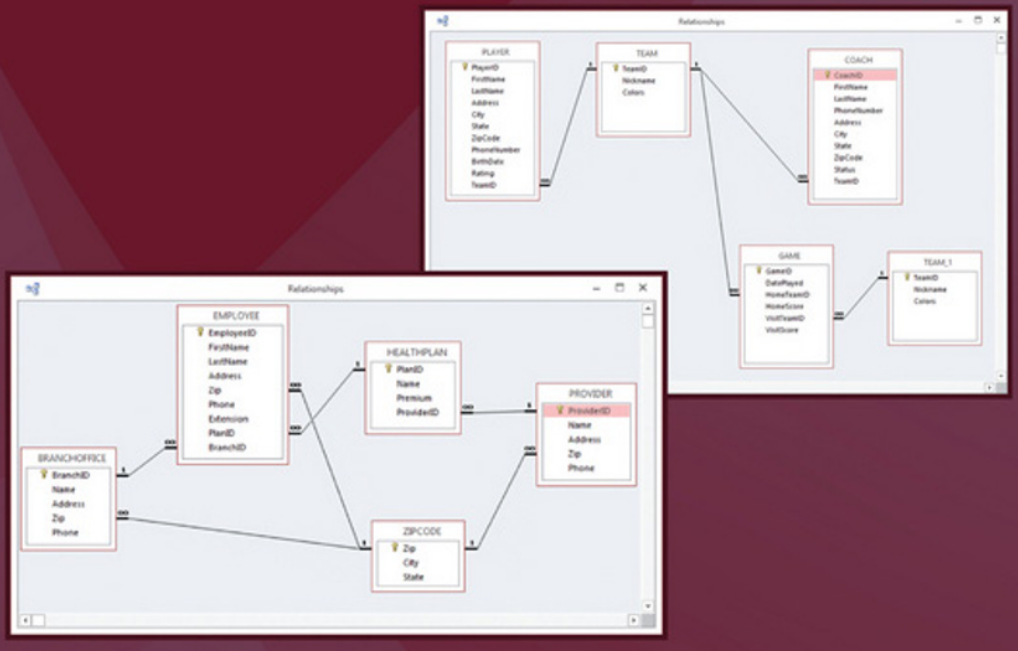
Jonathan Eckstein
Bonnie R. Schultz

INTRODUCTORY RELATIONAL DATABASE DESIGN FOR BUSINESS

WITH MICROSOFT ACCESS



WILEY

Introductory Relational Database Design for Business, with Microsoft Access

Introductory Relational Database Design for Business, with Microsoft Access

Jonathan Eckstein

MSIS Department
Rutgers Business School
United States

Bonnie R. Schultz

Schultz Writing Services
Princeton, New Jersey
United States

WILEY

This edition first published 2018

Registered Office(s)
John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, USA
John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, UK

Editorial Office
9600 Garsington Road, Oxford, OX4 2DQ, UK

For details of our global editorial offices, customer services, and more information about Wiley products visit us at www.wiley.com.

Wiley also publishes its books in a variety of electronic formats and by print-on-demand. Some content that appears in standard print versions of this book may not be available in other formats.

Library of Congress Cataloging-in-Publication Data

Names: Eckstein, Jonathan, author. | Schultz, Bonnie R., author.
Title: Introductory relational database design for business, with Microsoft Access / by Jonathan Eckstein and Bonnie R. Schultz.
Description: Hoboken : Wiley, 2017. | Includes bibliographical references and index. | Identifiers: LCCN 2017019748 (print) | LCCN 2017028117 (ebook) | ISBN 9781119329428 (pdf) | ISBN 9781119329442 (epub) | ISBN 9781119329411 (hardback)
Subjects: LCSH: Relational databases. | Microsoft Access. | BISAC: BUSINESS & ECONOMICS / Statistics. | COMPUTERS / Management Information Systems.
Classification: LCC QA76.9.D3 (ebook) | LCC QA76.9.D3 E325 2017 (print) | DDC 005.75/65–dc23
LC record available at https://lccn.loc.gov/2017019748

Cover Design: Wiley
Cover Image: Relationships Windows: Courtesy of Jonathan Eckstein

Set in 10/12pt Warnock by SPi Global, Pondicherry, India

10 9 8 7 6 5 4 3 2 1

Contents

Preface

Why Did We Write this Book?

This book arose from the first author's experience of teaching an undergraduate management information systems (MIS) course in the business school of Rutgers University in Piscataway, NJ, United States. This experience consisted of teaching 20 different sections in 12 different semesters, spread over a 20-year time span.

Rutgers' undergraduate New Brunswick business program's approach to teaching MIS differs from that of most business schools. Typically, MIS courses and textbooks stress superficial familiarity with dozens or even hundreds of aspects of information technology. The Rutgers approach, even before the first author arrived there, was different. At least two thirds of the course is spent achieving a relatively deep understanding of one of the most pervasive, durable, and persistent technologies in information technology: relational databases. Finding suitable textbooks was difficult, however. For some time, we used two books, one being a traditional MIS book and the other covering the Microsoft Access relational database product. This solution was expensive and not entirely satisfactory, and became less so over time. With each release of Access, the available Access books became increasingly focused on details of the user interface, and shied away from explaining the underlying design issues of how to structure databases. Giving such a book to somebody without solid prior experience in designing databases is like having somebody without a driver's license read the owner's manual of a feature-laden luxury car: while they might learn how to set the climate control to keep the passenger and driver at different temperatures, they would be no closer to being able to properly use the car for its fundamental task of transportation. Books specifically about database design also exist but are primarily aimed at computer science majors. They are overly abstract and too technical for business students just beginning to learn about information technology.

This book, which began as a set of class notes, takes a different approach. It develops an understanding of relational databases step by step, through numerous compact but realistic examples that gradually build in complexity. While readers will not necessarily gain enough experience to design large-scale organizational systems with hundreds or thousands of tables, they do get a thorough grounding in the technology and its applications, enough to build useful systems with dozens of tables. At every stage, the technology is presented through application examples from business, as well as other fields, giving the reader a chance to concretely think through the details and issues that often arise.

One may well ask, "why should one teach an introductory MIS course this way?" The main reasons are as follows:

- Relatively lasting hands-on knowledge of a pervasive and useful technology
- Acquisition of immediately marketable skills
- Development of analytical thinking and problem solving

The currently prevalent approach to teaching MIS stresses "buzzword"-level knowledge of numerous currently popular technologies. But without the foundation of hands-on application and problem solving, such material is quickly forgotten. Such knowledge may be useful for those in high-level decision-making positions, but by the time most undergraduate students might reach such positions, the knowledge will most likely be largely forgotten and outdated.

Relational databases are one of the most durable technologies in information systems. For decades, they have been the dominant way most organizations store most of their operational data. While databases have grown larger and data are being gathered at ever-increasing rates, the basic concepts and techniques of the technology have remained stable (much more stable, in fact, than procedural programming languages). Once one is comfortable with basic productivity software such as e-mail clients, word processors, spreadsheets, and presentation packages, there could scarcely be a more important or foundational technology to learn, even for manipulating data on one's own personal computer. By designing dozens of (albeit relatively simple) databases and formulating dozens of queries, students using this book acquire an understanding of relational databases in a way that should be more durable than knowledge acquired by memorizing facts or concepts.

Being able to understand and work with relational databases is a marketable skill that students can put to work at the beginning of their careers in almost any industry. While we first introduce queries using Microsoft Access' QBE (query-by-example) grid, most of this book's coverage of queries is through SQL (Structured Query Language), which is used with minor variations in nearly all relational database systems. We have received positive feedback from students who used earlier versions of this text distributed as class notes, to the effect that they were able to "hit the ground running" in jobs or internships

because they already understood how to formulate complex database queries in SQL. Superficial "survey" MIS courses do not provide such skills.

Designing a database is a highly analytical skill, involving breaking down a situation into its critical components such as things, people, and events, and clearly elucidating the relationships between these components. Learning such a skill develops the mind generally, fostering abilities in critical thinking and problem solving. Developing such abilities is an important component of any college education, regardless of students' fields of study. Just because a course is in a business school does not mean it should convey only facts – students in business programs deserve to develop their fundamental thinking skills just as much as (for example) majors in philosophy, mathematics, or chemistry. Such considerations motivate our approach of not teaching just facts and trends, but of also covering relevant material that helps students learn new ways of thinking and solving problems. Relational database design is an ideal vehicle for such mental development. Compared to other cognitively demanding IT-related skills like procedural computer programming, we have found that relational databases are relatively accessible and easily related to a wide range of nontrivial applications. The somewhat widespread notion that only computer scientists can or should design databases is simply not true. Almost any business student can learn how to design databases with up to a dozen or so tables, and for most people it is a much less frustrating means of cognitive development than learning, for example, Python or Java.

When embedded in packages such as Microsoft Access, relational database technology now allows the production of relatively sophisticated software applications with little or no computer programming in the traditional procedural sense. In fact, Access' Form, Report, Navigation, and Query features allow construction of professional-looking and useful applications without any "classical" programming whatsoever. Chapter 6 explores these abilities of Access, and its exercises provide a number of different mini-projects for student assignments. Being able to completely build such an application gives students a feeling of mastery and accomplishment.

This book uses Microsoft Access as a vehicle for learning about relational databases because it is widely available and relatively easy to use. But this is *not* "an Access book." We leave many features of Access uncovered and focus on basic skills that largely transfer to other relational database settings. Students need "hands-on" experience, and Access is simply the most logical vehicle to use. For more exhaustive coverage of the many "nooks and crannies" of Access, numerous books are already available. However, they all assume that their readers already know how to design a database.

When we teach MIS, we also cover some material not included in this book. In the course of a typical 28-class semester, we might have 6–7 lectures on other topics such as spreadsheets, network technology, security, and ethics. We chose not to include such material in this text because it is amply covered

in other textbooks, especially at the level of detail that only 6–7 classes permit. Instead, this book focuses on what is unique about our approach to teaching MIS. Instructors are encouraged to combine this book with other books, excerpts from other books, or their own notes and lectures on topics not covered here.

Finally, while this book was conceived as a textbook for undergraduate business students, it could also be used in other educational situations or even outside the context of a graded course, as a relatively "friendly" introduction to database technology. We are not aware of other books, textbooks or otherwise, that develop relational database technology in the incremental, example-rich manner that has proved effective at Rutgers over the past two decades.

1

Basic Definitions and Concepts

This chapter covers the following topics:

- Basic definitions and concepts in database technology
- The role of computers and network technology in helping run businesses and other organizations
- Common types of information processing systems in current use

Basic Terms and Definitions

There are some basic definitions and concepts that should provide useful context for understanding database design. Some of the terms we define are in common use but take on specific meaning in the information technology field.

Datum is a singular word, and *data* is its plural. A datum (sometimes called a "data item") is a "particle" of information like "12" or "Q."

Information refers to data that are structured and organized to be useful in making a decision or performing some task. Relational databases are currently the most common way data are organized into information; hence this book's focus on relational databases.

Knowledge denotes understanding or evaluating information. An example could be when Casleton Corporation analyzes its recruiting data and concludes that recruits from Driftwood College tend to have good performance evaluations only if their GPAs are at least 3.0. Based on this "knowledge," Casleton's managers might choose to screen applicants from Driftwood College by their GPAs, interviewing only those graduates with at least a 3.0 GPA.

For this book, we will focus on representing information within computer systems. Note, however, that knowledge can also be represented within computers. One common kind of knowledge representation (KR) within computers is part of the field of *artificial intelligence* (AI). One common business application of AI in business is in automated *business rules* systems. Another

Introductory Relational Database Design for Business, with Microsoft Access, First Edition.
Jonathan Eckstein and Bonnie R. Schultz.

recently popularized AI application is the "Siri" personal assistant on iPhones and iPads, or the similar "Google Voice" app on Android devices. Although its business uses are substantial and gradually expanding, we will not discuss AI, as relational database systems are simpler and far more ubiquitous.

Information systems consist of the ways that organizations store, move, organize, and manipulate/process their information. The components that implement information systems – in other words, *information technology* – consist of the following:

- Hardware – physical tools: computer and network hardware, but also low-tech objects such as pens and paper
- Software – (changeable) instructions for the hardware (when applicable; the simplest hardware does not need software)
- People
- Procedures – instructions for people
- Data/databases

Information systems existed before computers and networks – they just used relatively simple hardware that usually did not need software (at least as we know it today). For example, filing all sales receipts alphabetically by customer in a filing cabinet is a form of information system, although it is not electronic. Tax records kept on clay tablets by ancient civilizations were also a form of information system. Strictly speaking, this book is about an aspect of CBISs (computer-based information systems). Because of the present ubiquity of computers in information systems, we usually leave out the "CB," treating it as implicit.

Present-day CBISs have the following advantages over older, manual information systems:

- They can perform numerical computations and other data processing much more quickly, accurately, and cheaply than people.
- They can communicate very quickly and accurately.
- They can store large amounts of information quickly and cheaply, and information retrieval can often be very rapid.
- They can, to varying degrees, automate tasks and processes that previously required human labor.
- Information no longer needs to be "stuck" with particular things, locations, or people.

However, increasingly, automated systems can have drawbacks, such as the following:

- Small errors can have a much wider impact than in a less automated system. For example, in March 2003, a minor software bug in some airport data collection code – which programmers were aware of but considered too small to cause operational problems – grounded all aircraft in Japan for two days.

- Fewer people in the organization understand exactly how information is processed.
- Sometimes, malfunctions may go unnoticed. For example, American Airlines once discovered a serious bug in its "yield management" software only after reporting quarterly results that were significantly lower than expected. ("Yield management" refers to the process of deciding how many aircraft seats to make available for sale at different fare levels.)

Information architecture is the particular way an organization has arranged its information systems: for example, a particular network of computers running particular software might support a firm's marketing organization, while another network of computers running different software might support its production facilities, and so forth.

Information infrastructure consists of the hardware and software that support an organization's information architecture, together with the personnel and services dedicated primarily to maintaining and developing that hardware and software.

Application and *application program* (nowadays sometimes simply "app") are somewhat ill-defined terms but typically denote computer software and databases supporting a particular task or group of tasks. For example, a firm's human resource department might use one application to analyze benefit costs and usage, and another to monitor employee turnover.

A classic business IT problem is that applications, especially those used by different parts of an organization, may not communicate with one another effectively – for example, a new hire or retirement might have to be separately entered into both of the human resources systems described above because they do not communicate or share a common database.

Types of Information Systems

Particular information systems may be intended for use at one or more *levels* of an organization, as follows (Figure 1.1):

- The operational level – day-to-day operations and routine decisions. In an airline, for example, an operational decision is whether to cancel a particular flight on a particular day, or what type of aircraft to schedule on a particular flight during the summer flying season. Operational events that that might need to be recorded could include a customer scanning her boarding pass as she boards a flight, or an aircraft arriving at its destination gate.
- The strategic level – the highest-level, "big picture" decisions. In the example of an airline, whether to serve the Asia–US market, or whether to emphasize cost over service quality.

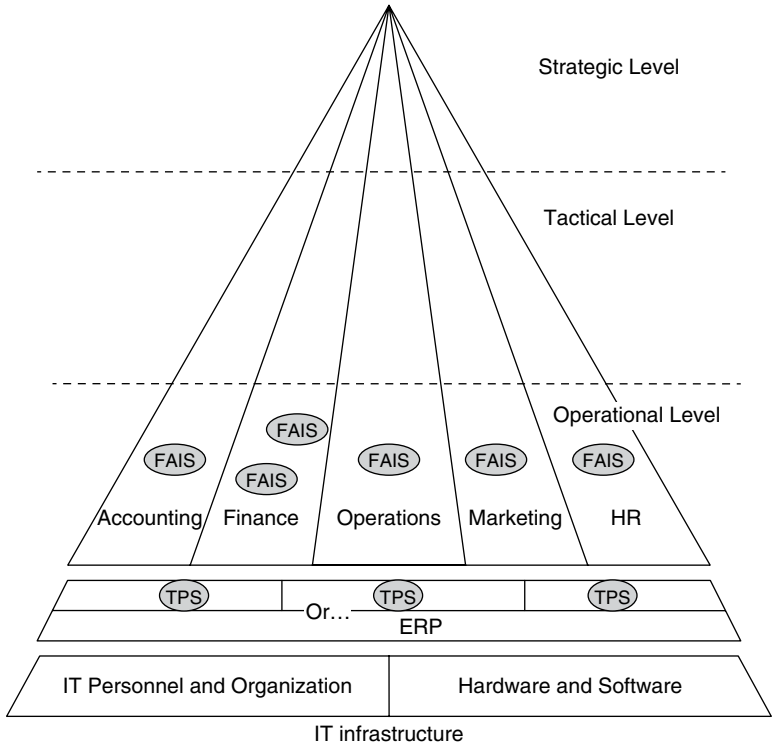


Figure 1.1 Information systems and the levels of an organization.

- The tactical level – decisions in between operational and strategic levels; for an airline, such a decision might be whether to increase or decrease service to a particular city.

In reality, the boundaries between these levels are typically somewhat indistinct: the levels form a continuous "spectrum." But labeling different segments of this spectrum as "levels" is useful conceptually.

Organizations are also typically divided into *functional areas*, meaning that different parts of the organization have different functions (that is, they do different things). These divisions vary by organization, but Figure 1.1 shows a fairly standard division into accounting, finance, operations, marketing, and human resources.

Transaction processing systems (TPSs) gather data about everyday business events in "real time" as they occur. Examples:

- You buy three items at a local store.
- A shipment of coffee beans arrives at a local distribution center.
- A passenger checks in for a flight.
- A package is unloaded from a FedEx or UPS aircraft.

Although only one of the above events is a transaction in the classical economic sense, from an information systems perspective all of these events are examples of *transactions* that may be immediately tracked by a TPS. Often, technology like barcodes and scanners makes tracking such transactions quicker, cheaper, and more detailed than if their associated data were to be keypunched manually. TPS systems are always operational-level systems, but they may also be used at other levels, or feed information to other systems at higher levels.

Functional area information systems (FAISs), also called *departmental information systems* (DISs), are designed to be operated within a single traditional functional department of an organization such as sales, human resources, or accounting. In the early days of CBIS, these were often the only kind of systems that were practical, because managing the data from more than one functional area would have required too much storage or computing power for a single system.

When an organization has multiple functional area systems, properly coordinating them becomes a potentially difficult issue. The systems may require overlapping data and can therefore become "out of sync" with one another. *ERP (enterprise resource planning) systems* are a relatively extreme reaction to the problem of poorly coordinated functional area systems, and are offered by vendors such as SAP and Oracle. They aim to support the entire organization's needs with essentially one single integrated system. They have enormous potential benefits but are also notoriously tricky and expensive to configure and install. Note that the only really meaningful word in the ERP acronym is "enterprise," denoting a system for the entire enterprise, and the reasons for "resource planning" in the acronym are historical. Such systems can perform resource planning but not particularly more than any other business function.

Some other common terms, some of which we will define in more detail later in the book, include the following:

- MIS – *management information system* – refers to a standard system that consolidates operational data into reports useful to managers.
- DSS – *decision support system* – refers to a system designed to help analyze and make specific kinds of decisions (at any level of the management hierarchy).
- ES – *expert system* – refers to a system that mimics the knowledge and behavior of human experts in particular domains, such as diagnosing problems with complicated equipment.
- EIS – *executive information system* – refers to a system that is designed to provide executives with information to assist them in making high-level (strategic or tactical) decisions.
- An *interorganizational system* (IOS) is a system that connects two organizations – for example, it may allow a company to automatically share inventory and backlog data with suppliers or customers.
- *Electronic commerce* or *e-commerce* refers to sales transactions in which at least one side of the transaction (buyer or seller), and perhaps both, is performed by a CBIS without direct human intervention.

2

Beginning Fundamentals of Relational Databases and MS Access

Microsoft Access is an example of *relational database* software, usually called a *relational database management system* (RDBMS). Access is just one of many relational database offerings in the software marketplace. Others include packages such as Oracle and Ingres. Some database software, such as MySQL, is available free of cost, while other packages are sold by commercial vendors such as Oracle and IBM.

All of these database packages are conceptually similar to MS Access. The greatest difference is in the scale of operation each package supports, in terms of both the volume of data and the number of simultaneous users. User interfaces also differ from package to package.

It is important to note that not all databases are relational. Some older technologies are still in use in the business environment, and other modern approaches exist, such as *object databases*. However, relational databases are by far the most commonly used today, especially in business applications, which is why this textbook focuses on them.

In relational databases, all data are kept in *tables*, also called *relations*. Most relational databases contain more than one table, but for now we will keep things simple and consider only a single table. A database with only one table is often called a *flat file* database.

Table 2.1 shows an example of a data table pertaining to students.

The *rows* of the table, also called *tuples* or *records*, correspond to things or events that we wish to store information about, such as people, orders, or products. In Table 2.1, each row corresponds to a student.

The *columns* of the table, also called *attributes* or *fields*, record various properties of the things or events being described. In this example, the attributes are the ID number, first name, last name, and zip code of each student.

Other kinds of software can store tables of data. For example, spreadsheet programs such as Microsoft Excel can store data tables. The biggest different between Excel and Access is in the way in which Access allows for relationships between multiple tables. However, other differences exist. For instance, each

Introductory Relational Database Design for Business, with Microsoft Access, First Edition.
Jonathan Eckstein and Bonnie R. Schultz.

Table 2.1 Example table of student data.

ID#	FirstName	LastName	ZipCode
14758993	Joseph	Ho	08765
23458902	Karen	Leigh	21678
89312199	Max	Saperstein	11572
90926431	Alex	Holmes	08743
82938475	Meera	Rajani	99371
19284857	Evan	Chu	34012

column in a relational database table has a *fixed datatype*. Here "datatype" refers to the kind of data being stored: for example, an amount of money, some other kind of number, or a character string like a person's name. In a relational database, every datum stored in a column must have the same datatype; that is, every entry in the column must be a percentage, or every entry must be a character string, and so forth. In spreadsheets, you can have data of different types within the same column. For example, a name might be stored in a particular cell, but another cell in the same column might contain a percentage.

Another difference is that in relational databases, one identifies columns by a user-specified attribute name (such as *ID#* or *FirstName* above) rather than by sequential letters (A, B, C,…), or column numbers as in a spreadsheet.

One more difference is that in spreadsheets, rows and columns are essentially symmetrical in their basic function. For example, it is no harder to add a column to a spreadsheet than it is to add a row. In relational database tables, rows and columns have fundamentally different roles. In relational databases, you can add or delete rows easily and quickly, whereas columns are largely static. Depending on the specific relational database software one is using, one might be able to add or delete columns in a table after it has been created, but if the table already contains a large amount of data, such an operation may be very time consuming and require significant computing resources.

Beginning Fundamentals of MS Access

Microsoft Access is both of the following:

- A relational database system
- A graphical, object-oriented software development environment (but not an object database or object-oriented database, which would imply a different, more flexible data-storage model)

To develop an Access application, one uses various tools and "wizards" to create, customize, and link "objects" to suit one's needs. It is also possible

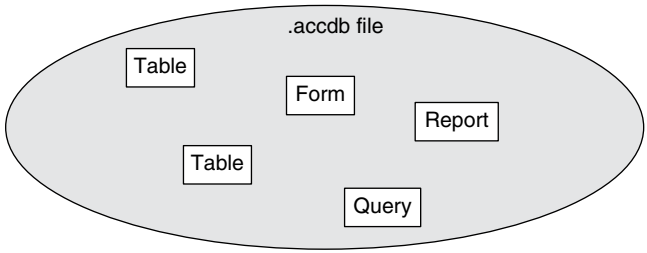


Figure 2.1 Objects within an Access file.

to write segments of computer programming code in the Visual Basic language and combine them with objects, but doing so is often unnecessary for simple Access applications. Thus, Access allows someone to build fairly sophisticated applications without engaging in classic text-based computer programming.

Access' most frequently used kinds of objects are *tables*, *forms*, *queries*, and *reports*. Access stores all the objects for a database in a single file with the type ".accdb" (or ".mdb" in earlier versions of Access).

Typically, Access keeps each database and its entire constituent objects inside a single operating-system file (Figure 2.1). For different database software or operating systems other than Microsoft Windows, the situation might be different: the database or even a single table might be spread across multiple operating-system files.

Access allows you to link objects in useful ways. We will start by examining the simplest such linkage, between a *table* and a *form*. Essentially, the table provides a way to store your basic data, and the form provides an alternative way to view that data on the screen. Information can flow in both directions between the table and form (Figure 2.2).

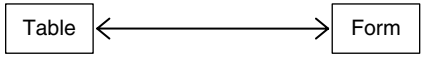


Figure 2.2 A table interacting with a form.

A "Hands-On" Example

Let us suppose we want to track information about students at a small college. We want to keep the following information for each student:

- ID number (9 digits)
- Name (First, Last, and Middle name or initial)
- Address, consisting of street address, city, state, and zip code
- Major
- Gender
- Birth date (question: why is it better to store a person's birth date than their age?)

- Whether or not the student is on financial aid (for the purposes of this example, a simple yes/no)
- Credits taken
- Grade points amassed

Note that if you take a three-credit class and get a B+, that means you get $3 \times 3.5 = 10.5$ grade points. Your GPA is the ratio of your grade points amassed to your total credits taken.

Let us create a database to store this information:

1) Open Microsoft Access from the "Start" or Windows menu at the bottom left of the screen (Figure 2.3).

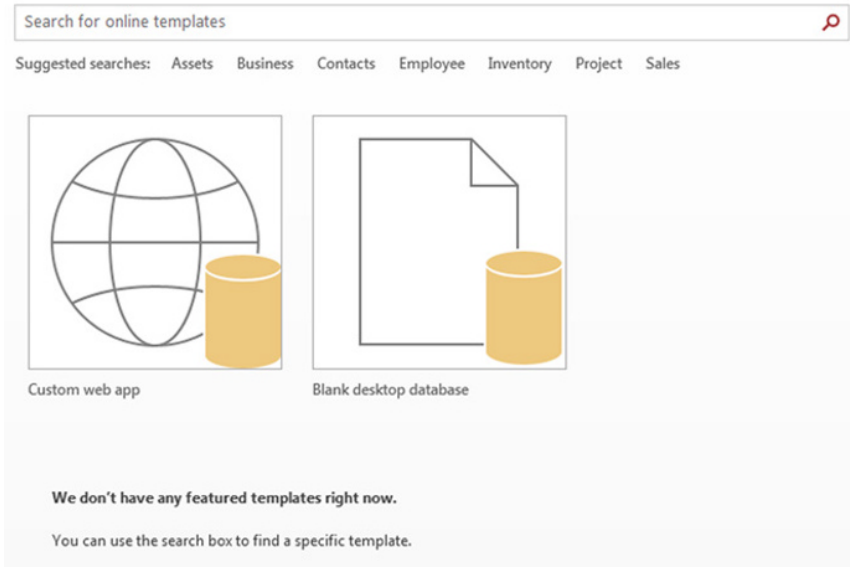


Figure 2.3 When Access opens.

2) Click the "Blank desktop database" icon ("Blank database" in earlier versions of Access).
3) In the resulting dialog box, provide a file name, for example, "students" (if Windows is configured not to display file types) or "students.accdb" (if Windows is configured to display file types).
4) Click "Create."

Access assumes that the first thing we want to do is to create a table. We see an empty table called "Table1." Now, we want to define what information resides in this table. This step is called *table design* (Figure 2.4).

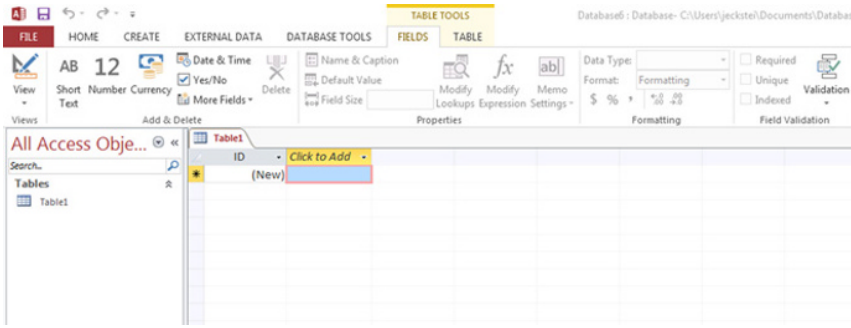


Figure 2.4 A new Access table.

5) Accordingly, we click the "View" button at the top left (make sure you have the "FIELDS" tab selected), and select "Design View."
6) A small dialog box appears requiring us to give the table a name – we call the table STUDENT and then click OK (Figure 2.5).

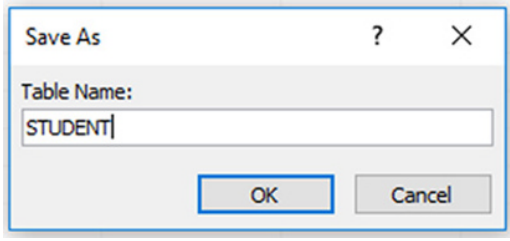


Figure 2.5 Naming a table in Access.

We now see a list of the attributes in the table (somewhat counterintuitively, the columns of the table appear as *rows* in this view; Figure 2.6).

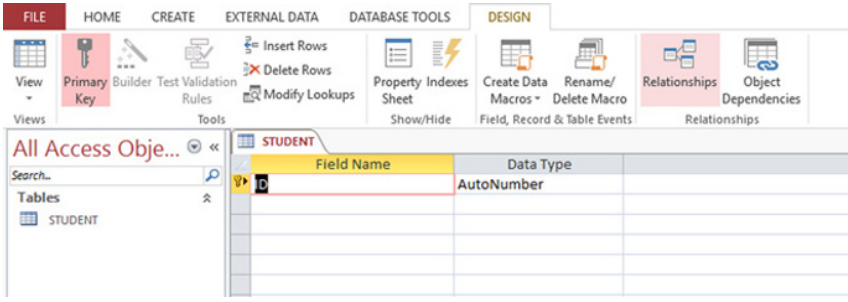


Figure 2.6 Table design view.

We now specify the columns in our table. In designing a database, you should keep in mind the following guidelines:

- *Try to plan for the future and include all the data you are likely to need.* In Access, it is possible to change attribute datatypes or add attributes later, but if the database is already large or many people are using it simultaneously, adding attributes or modifying datatypes could be slow or could disrupt the use of your system.
- *Have a separate field for each division of the data you anticipate needing.* As a general rule, it is much easier to put data together than to take them apart. For example, if we store student's names as three different fields (first, middle, and last names) then we can easily form a student's whole name by concatenating (placing end-to-end) the contents of these fields. If we just store the name as one large field, then certain tasks, such as sorting students by their first or last names, may become unnecessarily difficult and prone to error.
- *Avoid storing calculated fields.* If you have already stored the total credits and grade points amassed, you can easily calculate GPA. There is no need to redundantly store the students' GPAs. That will take up unnecessary space and create an opportunity for the fields of the database to become inconsistent with one another. In relational databases like Access, you should generally use table objects only for your "base" data. Calculations based on those data reside in other objects, such as queries, forms, and reports. This separation of base and calculated data is another way in which relational databases are different from spreadsheets, in which all "base" data coexist with calculations within the same two-dimensional grid of cells.

Let us now proceed with the design of our database:

1) Access has already provided a field called "ID," in keeping with the standard procedure of every table including a *primary key* attribute whose value uniquely identifies each row of a table. This is another difference from spreadsheets, which identify rows by simply numbering them sequentially. We will extensively discuss the choice and construction of primary keys later in this book.

   Now, each student should already have a unique student ID number, so that even if there were two students named "John Smith," each would have a different ID. Access defaults to a primary key field called "ID" with a datatype of "Autonumber," which means that Access will assign IDs automatically. Let us assume for this example that the college registrar has already assigned 9-digit ID numbers in a social-security-number-like format. We do not want to create our own different ID numbers, but would prefer to use the same ID numbers assigned by the registrar, in the same format. Therefore, we change the datatype of the ID field

to text, rather than Autonumber, by selecting "Short Text" (for Access 2013) or "Text" (for earlier versions) from the pull-down menu under "Data Type." Note that it is generally customary to use text fields to represent ID numbers and the like, which do not have any specific arithmetic meaning – for example, it makes no sense to add or subtract two students' ID numbers.

Versions of Access prior to 2013 had two kinds of text fields, "Text" for standard, fixed-length fields and "Memo" for potentially very long, free-form, variable-length fields. In Access 2013 the terminology for these two kinds of fields was changed to "Short Text" and "Long Text," respectively. This terminology can be a bit confusing because "Short Text" has the potential to be quite long (255 characters), and a "Long Text" field does not necessarily have to be long. In this textbook, we will use the term "Text" to refer to standard, "short" text fields, and we will not use "Long Text" fields in our exercises. Therefore, we will refer to "Short Text" fields simply as "Text" from this point forward.

In the "field properties" at the bottom of the screen, we enter "9" in "field size" (to set the ID length to nine characters) and "000\-00\-0000" under "input mask." This input mask allows you to enter only numbers (because of the "0" characters), and the placement of the "\-" characters causes the IDs to display in a social-security-number-like format. Note that the hyphens in the input mask are not actually stored in the database. Instead of just typing in the input mask, we can instead click on the "…" button in the input mask property, and select from commonly used input masks in the "input mask wizard" that then pops up. Finally, we can enter an explanation like "Student ID number from registrar" in the "description" area.

2) In the next row, we create a "FirstName" field by typing "FirstName". The datatype defaults to "Short Text" (or, equivalently, "Text" in older versions of Access), with a length of 255 characters. Here, we may select a shorter length, like 40. Here and below, we will select some reasonable lengths for text fields, but there is nothing magical or "best" about the lengths chosen.
3) Create a "MiddleName" text field; set the length to 20.
4) Create a "LastName" text field; set its length to 50.
5) Create a "StreetAddress" text field; set its length 80.
6) Create a "City" text field; set its length to 50.
7) Create a "State" text field, and set its length to 2 (assuming we will use standard postal two-letter codes for states).
8) Create a "ZipCode" text field, and set its length to 9 (for modern zip + 4 codes). We can click the "…" box on the right of "input mask," and after saving the table, select a standard mask for zip codes. Note that adding or multiplying zip codes makes no sense, so we store them as text.

9) Create a "Gender" text field; set its length to 1. We will just store "M" or "F" in this field. Later in this chapter we will see how to ensure that a user does not enter other letters.
10) Create a "Major" text field; set its length to 30 (later in this book, we will see how to allow only real majors to be entered in situation like this).
11) Create a "BirthDate" field. Access has a special datatype for dates and times, called "Date/Time," which can store combined dates/times with an accuracy of seconds. Select this datatype. Under "field properties," we can also select a format to use to display this information – for example, "short date." Note that it is much better to store a student's birth date, which is static, than their age, which would have to be periodically updated.
12) Create a "FinancialAid" field. Assume that we just want the database to remember whether the student has any financial aid. In this case, we have an example of a "yes/no" field; select the "Yes/No" datatype. We can set the format to "Yes/No" instead of the default "True/False" (this change affects only how the field is displayed).
13) Create a "Credits" field. Note that a student's tally of credits is a number on which it makes sense to perform addition and other arithmetic operations. Select the "number" datatype. In "field properties," select a "Field Size" of either "Long Integer" or "Integer" for the "field size" (an "Integer" means a whole number). Note that "Integer" can hold whole numbers in the range –32,768 through +32,767, which should be more than sufficient to hold a tally of credits. If the school were to allow fractions of credits, it would have to make this field a "Single" or "Double," datatypes that can hold numbers containing fractions.
14) Create a "GradePoints" field. Grade points amassed can be fractional, so select "number" and a field size of "Double," which can store arbitrary numbers including fractions. "Single" can store fractional values with about 6 digits of accuracy and "Double" about 14. "Double" is generally recommended over "Single" unless you anticipate having a gigantic database and need to worry about how much storage it will consume.
15) The table design process is now complete (Figure 2.7). Finally, we save the table design by clicking the small disk icon in the top left corner, next to the Access logo. Note that "saving" in Access saves an object within the overall Access file; in other Microsoft Office applications, "save" has the different meaning of saving the whole file. In Access, the database file as a whole is continuously saved.

    Next, we switch the "view" to "datasheet" instead of "design," and enter some data (we can just invent some information for now; Table 2.2).

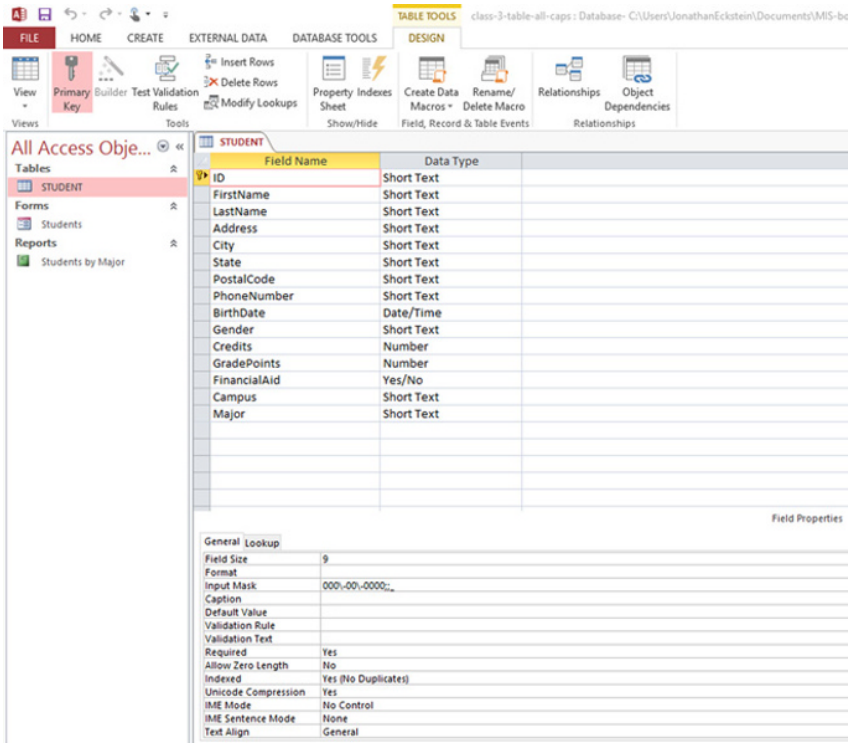


Figure 2.7 Completed Design View for the STUDENT table.

Introduction to Forms

We will now create a form to view our table in a more visually pleasing way.

With the table open, select the "Create" tab and push the "Form" button. Access then creates a form automatically linked to the STUDENT table. Note that we can enter data into the form, and it is immediately reflected in the table, and vice versa. The form first appears in "Layout View." If we choose "Form View," we can enter and change data.

We can now choose the "Design View" of the form and make changes. For example, we can change the font in the header, resize the text boxes, and so forth. To make changes to the widths and positions of the individual text boxes, we must first remove the "layout" that initially links them together. To remove this linking, choose the "arrange" tab, click the small handle on the upper left area of the form that looks like a box with a "+" sign in it, and click "Remove Layout."

Table 2.2 Sample data entered into the STUDENT table.

ID	First Name	Middle Name	Last Name	Street Address	City	State	Zip	Gender	Major	Birth Date	Financial Aid	Credits	Grade Points
547-89-9399	Bella	Q	Amati	393 West Boulevard	Piscataway	NJ	08854-	F	Marketing	12/22/1986	No	35	90
784-57-8483	Matthew	F	Short	23 Greene Circle	East Brunswick	NJ	08750-	M	Finance	3/12/1987	Yes	38	94
129-34-8900	Yu-Ping		Chen	177 Whitcomb Lane	Sparta	NJ	07768-	M	Accounting	4/4/1987	No	30	102.5

If you want to view forms and tables as sub-windows rather than tabs, select the "File" tab (or click the Office icon in Access 2007), click "Options," select the "Current Database" tab, and select "Overlapping Windows" under "Document Window Options." You then see a message stating "You must close and reopen the database for the specified option to take place." Close and reopen the database.

In Design View, our form should look approximately as in Figure 2.8.

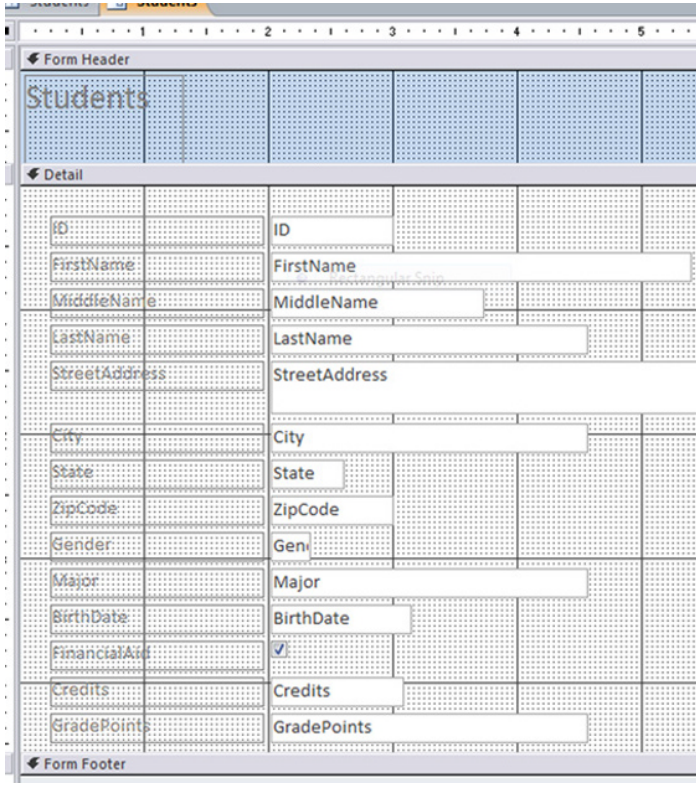


Figure 2.8 Design View of a form linked to the STUDENT table.

We can now add a box to display each student's GPA:

16) Select "Design View" again, if it is not already selected.
17) If desired, make space on the form by dragging the "Form Footer" boundary down.
18) In the "design" tab, click the "text box" tool ("ab |" near the left of the ribbon).
19) Draw the text box somewhere on the form.
20) After aligning the text box and its label, type "GPA" in the label.

- Next, view the properties of the text box (if you do not see them on the right of the screen, right-click the box and select "properties"). In these properties, perform the following operations:
  - Enter "=[GradePoints]/[Credits]" in "Control Source" – note that the names are case sensitive: "Credits" is not the same as "credits." The square brackets indicate to Access that it should use the values of fields in the linked table, in this case *Students*. Access should insert the brackets automatically if you type the field names correctly.
  - Select "Fixed" in the "Format" property and "3" for "Decimal Places."
  - Go back to "Form View," and observe the results (Figure 2.9). Note that GPA is computed from the other fields in the same record when you display the form, but is not stored in the database itself. Therefore, it is called a *computed field* or *calculated field*.

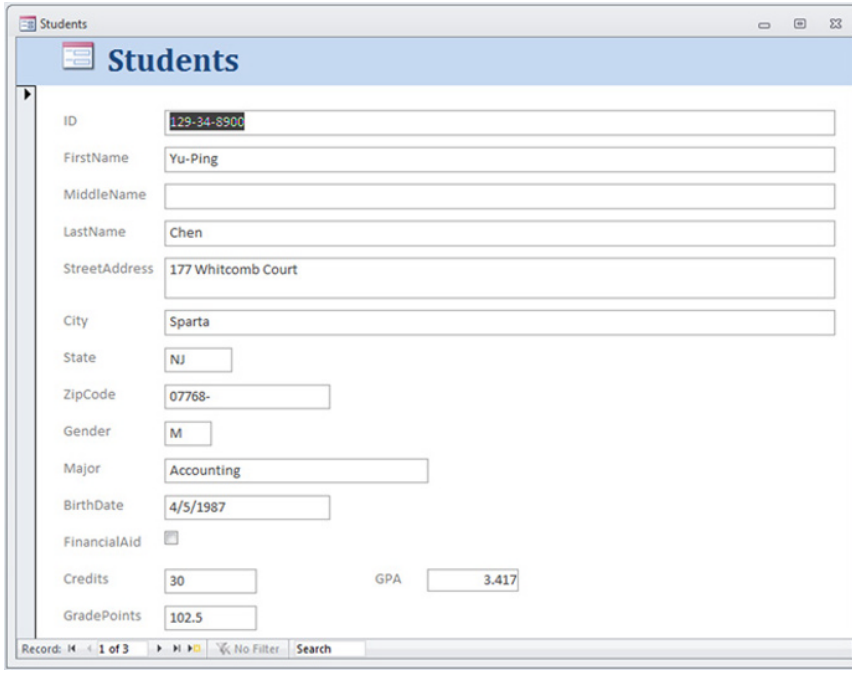


Figure 2.9 Completed student form with GPA display.

Another Method to Create Forms

We now present a method to create forms that gives the user a bit more control. To begin with, we download a slightly different version of the database we created earlier, `class-3-base.accdb`, from the book website.