

## Relational Databases



# Relational Databases

*RON MCFADYEN*



*Relational Databases by Ron McFadyen is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License, except where otherwise noted.*

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.



# Contents

Introduction	1
Ron McFadyen	
Preface	2
Ron McFadyen	
1. Relational Databases and Microsoft Access	5
Ron McFadyen	
2. Creating Tables	27
Ron McFadyen	
3. Creating Forms	53
Ron McFadyen	
4. Microsoft Access Queries	61
Ron McFadyen	
5. Relationships and Relationship Tools	81
Ron McFadyen	
6. Microsoft Access Queries – Advanced	95
Ron McFadyen	
7. Entity Relationship Modeling	144
Ron McFadyen	
8. Mapping an ERD to a Relational Database	180
Ron McFadyen	
9. Data Definition Language (DDL)	188
Ron McFadyen	
10. Normalization	196
Ron McFadyen	

Appendix A: Forms Involving Multiple Tables	245
Ron McFadyen	
Appendix B: Supertypes and Subtypes	249
Ron McFadyen	

# Introduction

RON MCFADYEN

Relational Databases and Microsoft Access

Ron McFadyen (2014-2017)

Cindy Miller (Revised: 2019)

This work is licensed under Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License.

# Preface

RON MCFADYEN

This text is a free introductory text that introduces Microsoft Access Office 365 and relational database design. The motivation is to support a second-year course on database systems which, to the student, is either a service course providing an introduction to database concepts, or, as a prerequisite for more advanced study in the field.

Various texts have been used with some success but were felt lacking for various reasons such as: (1) being workbook style with extensive tutorial lessons, (2) being too focused on a technology, (3) having design material that did not fit well with more advanced courses, and (3) being so expensive that some students opted not to purchase.

Our second-year course has no prerequisites and is taken by students from various disciplines. However, most students are registered in either a Computer Science major program or the Computer Science minor. Students who enroll in the course obtain: (1) a working knowledge of a personal database system (MS Access), (2) knowledge of SQL (primarily the Select statement) and (3) awareness of concepts and techniques necessary to database design.

Following this course, students can take third- and fourth-year courses in the database subject area. The coverage of Entity Relationship Modeling in those courses is based on the Chen notation – as is usual for academic texts. To be consistent with those higher level courses the same approach is used here.

It is our opinion that many students find normalization theory a difficult topic. Many presentations on normal forms are more complicated than necessary (e.g. some texts will give more than one definition of some normal forms). Our approach has been largely motivated by the writings of Chris Date. We have attempted to give

a suitable introduction to normalization theory for the beginning database student and to relate that material to other topics such as entity relationship diagrams.

Version 2.0 includes two appendices that cover a) creating forms that display data in a parent/child format where two tables are related via a one-to-many relationship, and b) entity-relationship modeling for supertypes and subtypes.

Version 3.0 includes revised Microsoft Access Office 365 content with improved accessibility to users supporting w3c accessibility standards.



# I. Relational Databases and Microsoft Access

RON MCFADYEN

A *database* is an organized collection of data. A database may be on paper or held in computer files such as spreadsheets or more formally in a software system known as a computerized database management system (for example DB2, db4o, IMS, MS Access, MS SQL Server, MySQL, Oracle, Sybase, Total, Versant). In this book, we focus on Relational databases and one specific relational database system: Microsoft Access.

There are many different commercial relational database systems and what you learn here will assist you in using those others. Because Microsoft Access is a workstation/personal system it is a convenient system for beginners.

## I.1 Relational Databases

Relational Databases were introduced by E. F. Codd in 1969<sup>1</sup>; Codd's 1970 paper<sup>2</sup> is considered one of the great papers in Computer Science.

We begin with a very small example: a database with one relation, the list of employees shown in figure 1.1. You should notice this looks

1. Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks, IBM Research Report, 1969.
2. A Relational Model of Data for Large Shared Data Banks, CACM 13, No. 6, June 1970.

just like a two-dimensional table of rows and columns. The name of the table is Employees, each column of the table has its own title, and each row has the same structure. Each row has a value for employee number, first name, last name, and gender. As tables of data appear in so many places (newspaper articles, textbooks, web pages, etc.) it is very likely you have seen and used this representation for data previously.

Employees			
Employee ID	First Name	Last Name	Gender
123	Joe	Smith	Male
333	Jim	Jones	Male
456	April	Smith	Female
842	Jenny	Jones	Female
777	Tom	Lee	Male

**Figure 1.1 A list of employees**

Let us assume the Employees table in figure 1.1 has one row for each employee who works for some hypothetical company. Data kept for each employee comprises their employee identification number, their first and last names, and their gender. Information structured in tables is very concise; at a glance, we can obtain useful information.

According to the database design methodology in *Information Modeling and Relational Databases*<sup>3</sup>, a database designer must be

### 3. Information modeling and relational databases, 2nd



able to express structured information as *verbalizations*. A verbalization that fits the information in one row of the Employees table is:

*Employee with ID ... has a first name ..., a last name ..., and is of ... gender*

In verbalizations like this, the ellipses are placeholders: we can use values from a single row to create complete statements that explain the meaning of a row. For example,

Employee with ID 123 has a first name Joe, a last name Smith, and is of Male gender

Employee with ID 333 has a first name Jim, a last name Jones, and is of Male gender

A similar approach to organizing knowledge about data appears in the literature on literacy. In the Journal of Reading several articles by Kirsch and Mosenthal discuss the organization of information and its conceptualization as document sentences. In *Building Documents by Combining Simple Lists*<sup>4</sup>, Kirsch and Mosenthal present an example based on information from The World Almanac and Book of Facts: 1980 (Newspaper Enterprise Association, p. 427). That data is reproduced in figure 1.2.

edition, by Terry Halpin and Tony Morgan; Morgan Kaufmann Publishers; ISBN -13 978-0-12-373568-3.

4. Irwin S. Kirsch and Peter B. Mosenthal. Building documents by combining simple lists. Journal of Reading, Vol. 33, No. 2, pp. 132-134.

Circulation of Leading U.S. Magazines	
Magazines	Circulation
TV Guide	19,547,763
Reader's Digest	18,094,192
National Geographic	10, 249,748
Better Homes & Gardens	8,007,202
Family Circle	7,611,578
Woman 's Day	7,535,855
McCall's	6,502,880

**Figure 1.2 Circulation of leading U.S. magazines**

A major point the authors make is that such information can be re-conceptualized as a series of simple document sentences formed from a basic *document sentence*. This document sentence expresses an understanding of the tabular data in natural language. The document sentence for figure 1.2 is:

*Magazine X has a circulation of Y.*

Kirsch and Mosenthal use variables (X and Y) to stand for data that comes from a table. Taking values from a row, we plug values for X and Y into the document sentence to obtain sentence instantiations:

TV Guide has a circulation of 19,547,763.

Reader's Digest has a circulation of 18,094,192.

National Geographic has a circulation of 10,249,748.

Better Homes & Gardens has a circulation of 8,007,202.

Family Circle has a circulation of 7,611,578.

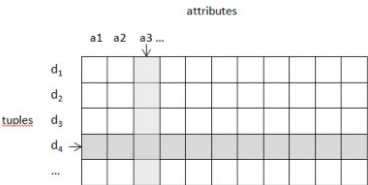
Woman's Day has a circulation of 7,535,855.

McCall's has a circulation of 6,502,880.

Document sentences and verbalization sentences are essentially

the same. Both sentences use natural language to express in words the meaning of tabular data. Whether one is designing databases or reading structured information, it can be useful for understanding to re-formulate data as statements in natural language.

Let us be a bit formal for a moment. Commercial *relational* database systems are systems where data is organized into *relations*. Figure 1.3 shows the general structure of a relation. We say a relation comprises a set of *tuples* where each tuple has the same number of *attribute values*, where each attribute value is taken from some corresponding *domain*, and where a domain represents a set of valid values for an attribute.



**Figure 1.3 General structure of a relation**

The Employees table in figure 1.1 can be considered a relation of 5 tuples where each tuple has 4 values drawn from each of the employee identifier, first name, last name, and gender domains.

Similarly, we can say the lists comprising the Circulation of leading U.S. Magazines in figure 1.2 can be considered a relation with 7 tuples each having 2 attribute values.

Relations are typically implemented in commercial databases as tabular structures comprising rows and a fixed number of columns. Everybody is familiar with tables as they are commonplace in textbooks, papers, magazines, etc. This simplicity of representation is one reason why relational databases have been very successful as repositories for important data.

# 1.1 Exercises

To design a database, a database engineer needs to find good representations of how an organization uses data. Good sources include: input forms, reports, web pages, etc. A challenge for database designers is to find these sources and interpret them.

- 1. Consider the following table example of product information sold by ABC Foods. Verbalize the information presented.

Product ID	Product Name	Unit Price	Units In Stock
1	Black Tea	\$2.00	44
2	Green Tea	\$3.00	33
3	Vegetarian Lasagne	\$10.00	20
4	Cajun Seasoning	\$11.00	29
5	Cranberry Sauce	\$21.00	0

- 2. Consider the following report example that the Human Resources department of ABC Foods must produce. Verbalize the information in that report.

Employee ID	First Name	Last Name	Department
1	John	Smith	Receiving
2	Lee	Daniels	Sales
3	April	Turner	Sales
4	Thomas	Trump	Marketing
5	Lee	Smith	Marketing

- Suppose the following input form example is used to enter contact information. Verbalize the information that is being collected.

## 1.2 Microsoft Access

Microsoft Access is a relational database system for workstations that run the Microsoft Windows operating system. Microsoft Access (MS Access) is an integrated Microsoft Office suite application. Microsoft Access is typically used by individuals for data they use personally, but in some situations, a single MS Access database may be used by a group of people or small department.

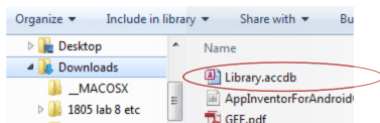
Microsoft Access databases are stored in a single file that has a

file suffix of “.accdb” or “.mdb”. Databases created using MS Access 2007 and later have a file suffix “.accdb”, and databases created using MS Access 2003 or earlier have a file suffix “.mdb”. We will be using databases where the files have names ending in “.accdb”. You need to use MS Access 2007 or later to open these databases.

This open resources will use Microsoft Office 365 – Access. If you are using a different Access version, your version may look slightly different. The basic Access functions applying to different versions of Access will be similar so you should be able to follow along without any problems.

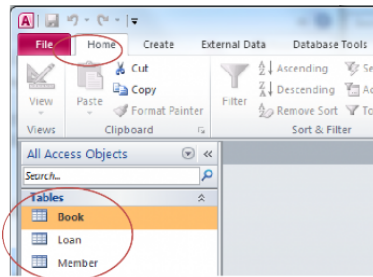
Our first sample database is in a file named *Library.accdb*; this database is available from the website associated with this text. Open the file *Library.accdb* included in your data files and then save the database as *Library.accdb* onto your storage device.

To use this database, you must first download the file containing the database, and then open the database by double-clicking the file name:



**Figure 1.4 Double-click the database file to open the sample database**

When you open this database you see a list of objects (figure 1.5) in the database; you will see three tables: Book, Loan, Member:



**Figure 1.5 The Home tab in MS Access shows you the table names**

Double-click a table name and MS Access opens the table in Datasheet View; you can see the contents of Book in figure 1.6. The datasheet view for a table is easily obtained, but it's not a particularly user-friendly way to view and manage data in a table. We will learn other ways of handling data with MS Access Forms. The Book table has three fields (i.e. attributes): callNo, title, author. When we view a table we see data organized into rows and columns. The data in one row corresponds to one book; if there are 11 books, then we have a table of 11 rows.

callNo	title	author
R CB 351 M255 1983	Atlas of medieval Europe	Donald Matthew
R HQ 1143 P58 1575	Medieval women	Elben Power
R PC 14 V48 1965	Medieval miscellany	Frederick Whithead
R QA 76.73 567C439 2004	Joe Celko's Trees and hierarchies in SQL for smarties	Joe Celko
R QA 76.73 567C439 1987	Joe Celko's SQL pointers & answers	Joe Celko
R QA 76.76 A65P76 2011	Programming Android	Zigurd R Mednieks
R QA 76.9 0204P55 2008	Information modeling and relational databases	T A Halpin
R QA 76.9 0204P48 1986	Data model patterns : conventions of thought	David Hay
R QA 76.9 0204C45 1999	Joe Celko's data & databases : concepts in practice	Joe Celko
R R 143 L45 2006	Medieval medicine and the plague	Lynne Elliott
R R 482 J35 1967	Medicine in medieval England	Charles H Talbot

**Figure 1.6 Datasheet View of a table**

The Book table contains one row for each book in the library. We can verbalize the content of a row as: *The book identified by call number ... is titled ... and is authored by ...*

Substituting actual values from rows we can make explicit statements such as:

- The book identified by call number PC 14 V48 1965 is titled *Medieval miscellany* and is authored by *Frederick Whitehead*
- The book identified by call number QA 76.76 A65P76 2011 is titled
- *Programming Android* and is authored by *Zigurd R Mednieks*

Knowing that books are identified by their call number and since the above statements use the conjunction 'and', the above verbalization can be expressed in an elementary form as:

- The book identified by call number ... is titled ...
- The book identified by call number ... is authored by ...

Each of these expressions is considered elementary because each states one fact about a specific book. We cannot make these statements any simpler.

Of course, we can now substitute values from the table and obtain:

- The book identified by call number PC 14 V48 1965 is titled *Medieval miscellany*
- The book identified by call number PC 14 V48 1965 is authored by *Frederick Whitehead*
- The book identified by call number QA 76.76 A65P76 2011 is titled *Programming Android*
- The book identified by call number QA 76.76 A65P76 2011 is authored by *Zigurd R Mednieks*

At this point, expressing verbalizations this way may seem trivial and unnecessary, but they do serve a purpose. These scenarios make it clear that the title and the author's name serve only to describe a book, and that the call number identifies the book. An aim of a database designer is to understand data requirements in terms



of these elementary forms. We'll have more to say about this in a later chapter.

Up to this point, we have learned how to

- open a Microsoft Access database;
- recognize that a database contains a number of tables;
- open a table to display a collection of rows and columns;
- verbalize the information in a table.

In the next section, we will examine the basic table features in Microsoft Access to insert, modify, and delete data records using our sample database named *Library.accdb*.

## 1.2 Exercises

Recall that an elementary verbalization is one where the verbalization cannot be simplified in any further way. Simpler statements would result in a loss of information.

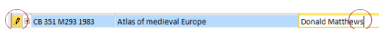
1. Rewrite the verbalization for the *Employees* table using elementary verbalizations.
2. Is the verbalization given for *Circulation of Leading U.S. Magazines* in elementary form?
3. What verbalizations apply to the *Loan* table in the *Library* database?
4. What verbalizations apply to the *Member* table in the *Library* database?
5. View the data in the *Loan* table. Each row in the table corresponds to a member borrowing a book. Notice how the call number field contains values that appear in the *Book* table and how the id field contains values that appear in the *Member* table. All rows have a value for the data borrowed field. Why would some of the date returned fields appear to have no value

at all?

6. The web site for these notes has a number of databases. Download the University database and examine its contents. This database contains information about departments and courses in a fictional university. Typically a university is organized into faculties which comprise departments and those departments offer courses. For instance many universities have a Faculty of Science which itself may contain departments such as Mathematics, Statistics, and Physics. Each of these departments will offer courses for students to take: Introduction to Calculus, Introduction to Statistics, Discrete Mathematics, etc.

## 1.2.1 Modifying Rows

With Microsoft Access, open the *Library.accdb* database file. Open the Book table in Datasheet view. With the cursor positioned in a row, try modifying the data recorded for that book. If you position the mouse cursor, you can change the value recorded for the book's call number, title or author. Try *doing this* – remember you can always download this database again if you wish to get back to what you started with. As you begin modifying a value (e.g. adding an 's' to make the last name Matthews) an editing symbol appears to the left of the row:



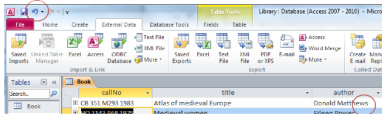
**Figure 1.7 Editing a row**

If you recognize that you are making a mistake you can undo your editing action by pressing the *Escape* key (keyboard).

To make your change permanent you must move the cursor to

another row for the update to be completed – when you do this, you will note the editing symbol disappears.

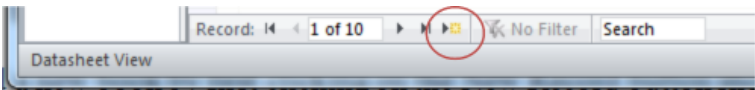
In some situations, you will find MS Access provides a formal Undo capability. Consider the following figure that shows an Undo icon in the upper left corner that appeared after changed Matthew to Matthews and moving the cursor to the next row:



**Figure 1.8 The Undo icon – click it and the last action is undone**

### 1.2.2 New Record Button to Add New Records

Try adding a new book record to the Book table. You can add a new book by first clicking on the *New Record* button shown near the bottom of the window:

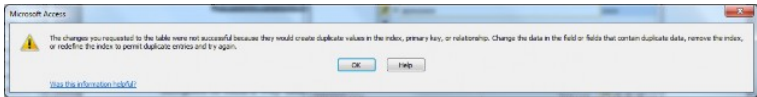


**Figure 1.9 Add new record button**

To complete your action you must type values for callNo, title, and author. As a first example use a call number that does not appear for any other book. As we will soon see the Book table is designed in such a way that each book must have a different call number. Your addition will be successful if your book is given a call number that no other book has. When you add a new row you must move the cursor out of the row for the addition to be completed.

As a second example try to add a new book, but this time, use a

call number that already appears in the table. In this case, MS Access will reject your new record. Try this and you will see a response similar to:



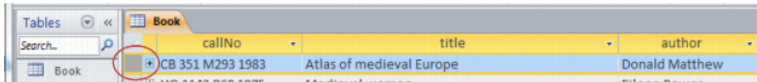
**Figure 1.10 MS Access message for duplicate primary key**

The important part of this message for us is the part that refers to *duplicate values* or *duplicate data*. When we try to add a row with the same call number as some other row MS Access refers to the duplicated call number value. Note that you can press the Escape key to remove the new row from the table display. Soon we discuss table design where you will see that the call number field is designed to be the *primary* key of the Book table.

Adding a row to a table is also referred to as inserting, or appending a row.

### 1.2.3 Deleting Rows

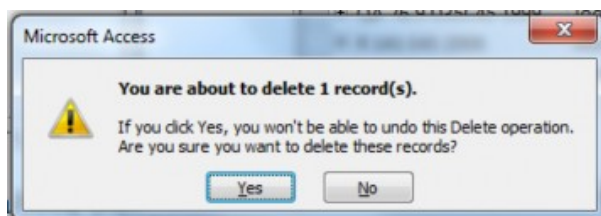
You can remove a record in the Book table by highlighting a row (click in the cell just to the left of a call number) and then press the Delete key on the keyboard:



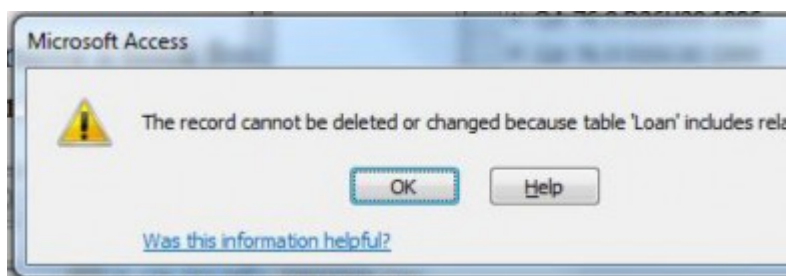
**Figure 1.11 Delete a record: select record, press delete**

When you press the Delete key, Microsoft Access will respond in one of two ways depending on whether or not there is an existing reference to the row you are trying to delete:

No, a reference to the book does not appear in the Loan table.



Yes, a reference to the book appears in the Loan table.



**Figure 1.12 Attempting to delete a record (row) in an Access table Datasheet View**

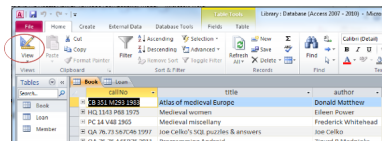
When you view the Loan table you are able to see the books that library members have taken out and whether or not a book has been returned. Rows in the Loan table have references to rows in the Book table and to rows in the Member table. The default action in MS Access is that deletion to this record is disallowed if there is a row in a table that has a similar reference to it. So we cannot delete a book if there is a Loan row referencing it.

We have briefly shown how to modify, add and delete data in tables. Next, we will introduce the design perspective for tables.

## 1.2.4 Table Design View

So far, we have been opening Access tables in Datasheet View where we can view and change data in rows of a table. When in Datasheet View, we can switch from datasheet view to *Design View* by clicking on the design icon located near the upper left hand corner (see

figure 1.13). When the Design View icon is clicked, the display changes: the icon becomes a Datasheet View icon and the display changes to reveal design information (see figure 1.14).



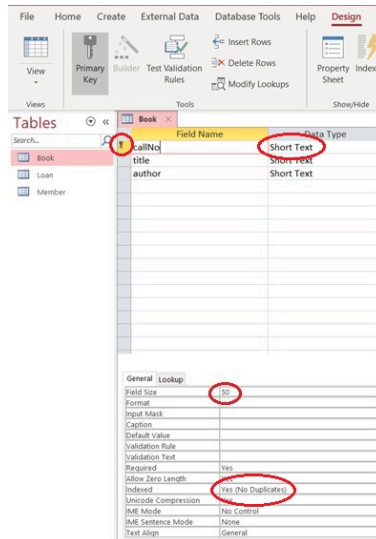
**Figure 1.13 Click on the Design View icon to switch to Design View**

When you click the Design View icon, you will see the display change as shown in figure 1.14. You will see the field names listed along with their *data type*. According to the field where the cursor is located, you see other *properties* for that field. Data types vary somewhat from one database system to another, but of course there are many similarities too. Properties are other characteristics that you can define for a field such as the maximum length of values stored for the field.

Generally, we want data in a database to be reasonable and correct. We can use data types and properties to achieve certain types of correctness. Consider the following integrity rules as rules we would like to enforce:

- Call numbers, titles, and authors are alphanumeric. Any text you can type on the keyboard is acceptable.
- Each call number must be unique (there can be no duplicates)
- Each book must have a title
- A value for call number must be no more than 50 characters long
- A value for title must be no more than 255 characters long
- A value for author must be no more than 255 characters long

The author field can be left out (it can be *null*). Now we discuss how these integrity rules are obtained in Table Design View.



**Figure 1.14 Table Design View**

In figure 1.14, the cursor is located on the callNo field; some properties of callNo are circled and discussed below:

- Beside the callNo field you can see the key icon. This means the callNo field is the *primary* key. A primary key is a unique identifier – every row in the table must have a unique value in that field. Every table should have a PK specified and there can be only one PK for a table. When a field is defined as the PK then a value must be provided in each and every row.
- The callNo field has a datatype of *Short Text* and a field size of 50. Any value you can type on the keyboard is acceptable but the overall length, number of characters, is restricted to at most 255.
- The callNo field is *indexed* and in this case no duplicates are allowed. The index constructed by MS Access is similar in

purpose to the index at the back of any book: the index allows MS Access to quickly locate a specified row. However, this index is different from that at the back of a book because it allows only one entry per indexed value (No *Duplicates* is specified for the *Indexed* property). Each call number is unique. As you move the cursor up and down you should note the following for this sample table: For title:

- The title field has a data type of *Long Text* with a field size of 255. A text field can comprise any combination of letters, digits, and punctuation. Any value entered by a user cannot exceed a gigabyte of text. Access forms and reports can only display 64,000 characters.
- A value is **required**. When entering data for some book, the user cannot omit the title.
- There is no *index* on title.

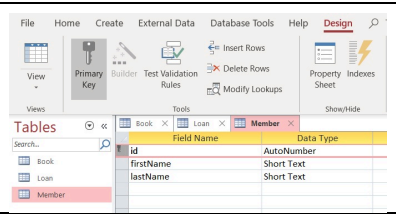
For author:

- The author field has a data type of *Long Text* with a field **size** of 255. A text field can comprise any combination of letters, digits, and punctuation. Any value entered by a user cannot exceed a gigabyte of text.
- A value is **not required**. When entering data for some student, the user can omit the author.
- There is no *index* on author.

Now, open the Member table and then the Loan table in Design view. Examine the properties of each field. For reference see figure 1.15.

### **Member Table**





Field	Data Type
id	Autonumber: MS Access will generate a unique number for each row. The user cannot enter id values.
firstName	Short Text
lastName	Short Text

**Loan Table**

Tools		Show/Hide
Book	Loan	Member
Field Name		Data Type
callNo		Short Text
id		Number
dateBorrowed		Date/Time
dateReturned		Date/Time
dateDue		Date/Time

Field	Data Type
callNo	Short Text
id	Number: numeric values must be entered by the user. This data type is a proper match to the Autonumber data type.
dateBorrowed	Date/Time: the user must enter a date
dateReturned	Date/Time
dateDue	Date/Time
fine	Currency: the user must enter values in dollars and cents.

**Figure 1.15 The fields of the Member and Loan tables**

Later, we will examine data types and properties in much more detail.

## 1.24 Exercises

Use your *Library.accdb* to complete the following:

1. Use Design View to add the following two fields to the Member table.  
*Gender*: Text field of length 25 to accommodate the values *male*, *female*, and *prefer not to disclose*. Make this a required field that is not indexed.  
*BirthDate*: a Date/Time field; required; not indexed. Switch back to Datasheet View (You must reply yes to the system prompt to save your changes). You should notice there are no values for *gender* nor *birthDate*.
2. Now enter values you deem appropriate in the *gender* and

*birthdate* fields for each member. Close the table and reopen it. You will see the values you entered are still there.

3. When new members join the library information about them must be entered into the Member table. Each member is given an id value automatically. Add new members to the library and note how MS Access will not let you enter id values; instead, MS Access generates those values for you – id values are generated sequentially. Close the table and then reopen the table to confirm your additions worked.
4. Typically, a library assesses a fine the user must pay if they keep a book out past the due date. As well the library needs to track the amount, if any, the member has paid. In this exercise, add two fields to the Loan table so we can keep track of fines that are assessed and the amount the member has paid.
  - Open the Loan table in design view and add two new fields named *fineAssessed* and *finePaid*. These fields must have a *Currency* data type. In lower pane of the Design view window, verify the Field Properties for *fineAssessed* and *finePaid* fields. Format property should contain a *Currency* data type and Decimal Places property contains *Auto*.
  - Save the Loan table and then view the rows of the table. There are no amounts for these fields.
  - Choose some row(s) in the Loan table and enter values for the *fineAssessed* and *finePaid* fields. Note the values you enter will appear as dollars and cents.
5. In exercise 3, you added a new member and in exercise 4 you added fields to the Loan table. Consider that the person you added now borrows a book and so a row must be entered into the Loan table. Enter such a row.
6. After successfully entering data for exercises 3, 4, and 5, you are aware of a member and a book for which there are references in the Loan table.

- View the Member table and try to delete that member, and then view the Book table and try to delete that book. These deletion attempts are unsuccessful because of the references to the Loan table.
- Now open the Loan table and find the loan record you entered in exercise 5. If you delete this row you will find that you are able to delete the member (provided you did not enter more loans for this person). These actions mirror the way in which data would typically be deleted from a database: if you want to delete a row you must first delete (or modify appropriately) any rows that reference it.

# 2. Creating Tables

RON MCFADYEN

The typical Microsoft Access database comprises several kinds of database objects such as tables, forms, queries and reports. Each table represents a kind of entity (persons, places, things, events, etc.), or relationship between entities. For instance, if we are keeping track of departments and courses at our University database then we should have two tables:

Department: to keep information about departments

Course: to keep information about courses.

For each department, suppose we need to know things such as department code, department name, location of the department (an office number), phone number for the department, and the name of the department's chair. Suppose departments can be identified by their department code (e.g. ACS) and by their department name (e.g. Applied Computer Science). Both of these fields are assigned by the University and each will be unique across departments. We will choose to use the department code as the primary key. We choose to use department code as the primary identifier for departments. We show Department with some sample data:

Department				
deptCode	deptName	deptLocn	deptPhone	chairName
ENGL	English	3D05	786-9999	April Jones
MATH	Mathematics	2R33	786-0033	Peter Smith
ACS	Applied Computer Science	3D07	786-0300	Simon Lee
PHIL	Philosophy	3C11	786-3322	Judy Chan
BIOL	Biology	2L88	786-9843	James Dunn

### **Figure 2.1 Department table**

Suppose the creation of the Course table keeps track of courses offered by the University and includes the fields: course number, title, description and credit hours. At the University, what is a course number? The ways of identifying courses varies from one institution to another. A common way is to display the department code followed by the course number (e.g. “ENGL 2221”; “ENGL 2221”) which comprises two fields: a department code and a course number. We will use this convention and must include department code as a field in the Course table. The combination of department code and course number serve as a unique identifier (i.e. together they comprise the primary key). We show this Course table structure with sample data as follows:

Course				
deptCode	courseNo	title	description	creditHours
ACS	1453	Introduction to Computers	This course will introduce students to the basic concepts of computers: types of computers, hardware, software, and types of application systems.	3
ACS	1803	Introduction to Information Systems	This course examines applications of information technology to businesses and other organizations.	3
ENGL	2221	The Age of Chaucer	This course examines a selection of medieval poetry and drama with emphasis upon Chaucer's Canterbury Tales.	6

PHIL	2219	Philosophy of Art	Through reading key theorists in the history of esthetics, this course examines some of the fundamental problems in the philosophy of art, including those of the definition and purpose of art, the nature of beauty, the sources of genius and originality, the problem of forgery, and the possible connection between art and the moral good	3
BIOL	4451	Forest Ecosystems Field Course	This is an intensive three-week field course designed to give students a comprehensive overview of forest ecology field skills.	2
BIOL	4931	Immunology	Immunology is the study of the defense system which the body has evolved to protect itself from external threats such as viruses and internal threats such as tumor cells.	3

---

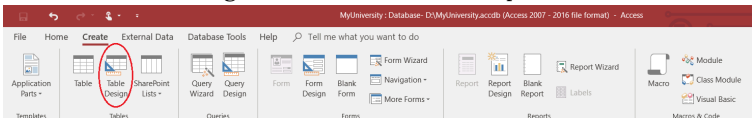


**Figure 2.2 Course table**

## 2.1 Using Design View to Create Access Tables

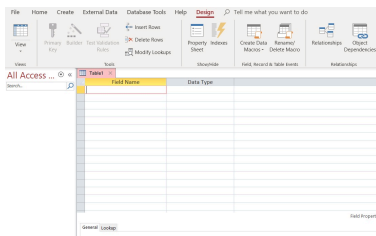
In this section, we will step through the process of creating a new table in a Microsoft Access database. From either the web page for these notes or as instructed by instructor, download and open the MyUniversity database (MyUniversity.accdb file).

1. Click on the *Create* tab (located in the Tables group) in the Ribbon. View the available table options by hovering the mouse over Table, Table Design and SharePoint Lists options.



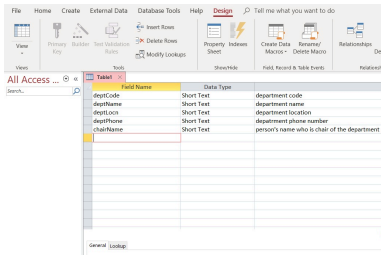
**Figure 2.3 Creating a new table using Microsoft Access – Table Design**

2. Click *Table Design* option to begin a process to create your new Department table. A new table object is inserted into your database and the new table will open in Design View. A blank table object will be displayed where you can enter field definitions for your table. A field definition comprises field name, data type, and description. You can also set the table's primary key. The Table design object is displayed as follows:






**Figure 2.4 Microsoft Access – Table Design View**

3. By referencing Figure 2-5, begin entering each field name and choosing the appropriate data type. The description column is optional and may contain a longer description of the field's contents. This description contains comments that may be useful for someone who is viewing the table. Once you have done this, you should have a table design that looks like:



**Figure 2.5 Microsoft Access – Department Table Design View**

4. Next, we need to select and set the primary key for the table. The primary key is a field(s) with unique values associated with the table. The deptCode field will be the primary key field for the Department table. Click the mouse in the cell to the left of the deptCode field name. Locate the Primary Key command in the Table Tools – Design tab and in the Tables group. Click the Primary Key

icon  on the toolbar. As an alternative, you can right-click using your mouse on the deptCode field and select the Primary Key  command. Access uses the primary key  icon to show the deptCode as the primary key for the Department table:

	Field Name	Data Type
	deptCode	Short Text
	deptName	Short Text
	deptLocn	Short Text

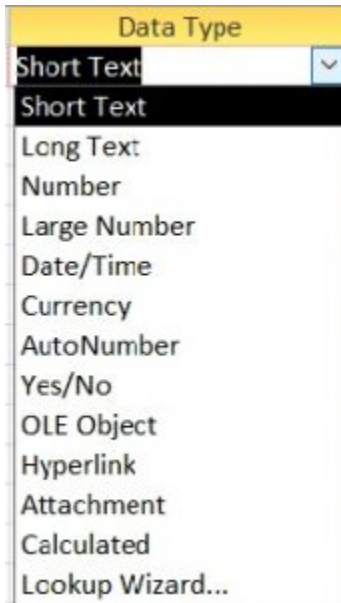
**Figure 2.6 Setting the Primary Key**

5. At this point, you should save your work by clicking the Save icon in the upper left hand corner of the form. You will be prompted to save the table. Save your table as “Department”.

You should still be in Design View for the Department table. Note that you can press the F1 function key on your keyboard to get help pertinent to the location of your mouse cursor. If your cursor is positioned on a field name and you press F1, you will see a window pop open that displays suggestions from Microsoft Access regarding how you should name fields. Try this. Before going any further, try pressing F1 in other locations on the table design view (i.e. Data Type and Description). We recommend that you read some of the information available to become more familiar with Microsoft Access.

## 2.1.1 Data Types

Microsoft Access provides several data types. We will discuss a few basic data types including Short Text, Long Text, Number, Date/Time, Currency, AutoNumber, Yes/No, Hyperlink, Attachment, Calculated, and Lookup Wizard.



**Figure 2.7 Microsoft Access Data Types**

### *Short Text*

If you specify that a field has the Short Text data type, Access will permit any alphanumeric characters to be placed in that field in a row of the table. This is a common choice when the data will not be used in calculations. The Short Text data type provides for values that have fewer than 256 characters. If you know that a maximum length is less than 255 characters, it would be appropriate to use the Field Size property (discussed in the next section) to limit the maximum length of a short text string.

### *Long Text*

A designer selects Long Text if the field will have alphanumeric data

longer than 255 characters. Long Text allows for a maximum length of 1 GB (gigabyte) of memory storage. You will need to remember that Access databases has a limitation of storing only 2 gigabytes of data. For example, consider the description field of the Course table. Could the description field be longer than 255 characters?

## *Number*

If a field is used for storing values that are used in numerical calculations then Number is the appropriate data type except for calculations involving money (e.g. Currency type). Access Number data types (e.g. quantity ordered) contain example Field Size properties available for Byte, Integer, Long Integer, Single, Double, and Decimal options. The Field Size property (properties are discussed later) can be used to limit the number of storage locations used per value.

## *Date/Time*

If a field contains date and or time values, then the Date/Time data type should be chosen. The Format property (discussed later) allows you to control how these values will appear to the user for date and/or time based data.

## *Currency*

If a field will contain monetary values, the Currency data type should be chosen. Currency data type provides for numeric calculations that are accurate to 15 digits to the left of the decimal and 4 digits to the right of the decimal. This data type can be used to prevent rounding off currency values during calculations.

## *AutoNumber*

If you choose AutoNumber, Microsoft Access will automatically generate the next value for you when a row is inserted into the table. By using the New Values property, you can arrange the numbers to be generated sequentially or randomly. Often control numbers for things like orders, invoices, registrations, etc. are numeric and we can leave it to the system to generate a next value for us.

## *Yes/No*

This data type can be used for data that has only one of two possible values including Yes/No, True/False, On/Off.

## *Hyperlink*

If you choose Hyperlink, you may store text/numeric values including website links (URLs) and email addresses.

## *Attachment*

This data type allows you to include embeddable objects to attach to your database records. Attachments may include a variety of different files including examples of documents, spreadsheets, digital images, graphs, tables, and other types of files you may use.

## *Calculated*

If you would like to create a calculation using fields from the same table, you can create a calculated field to store the result in a table.

## *Lookup Wizard*

Sometimes you need to restrict values to a list of known values or to values appearing as primary key values elsewhere in the database. An example of known values could be considered for the creditHours field. The creditHours field could contain a stored value list of 1, 2, 3, and 6. The Lookup Wizard allows you to create this data type with helpful steps to allow you to make the appropriate wizard choices for your lookup data values.

## **2.1 Exercises**

These exercises will refer to the Library database (as referenced in the previous chapter).

1. Open the Microsoft Access **Library** database file.
2. Open and evaluate the Member table. The id field was defined with the AutoNumber data type. Change the table view to Datasheet View. Experiment by adding new member records. You will note that id values increase by 1. Now try deleting the last two members that you added. If you add those members back in, what id values do they get? Are id values reused?
3. Consider the Book table. Change the table view to Design View. Add a field named *paperback* that can be used to indicate whether or not a book is a paperback. Choose the Yes/No datatype. Save the design and switch to datasheet view. Now

you will see how to enter such values. Microsoft Access provides a box that is to be checked, or not. You can select (a 'Yes') using the mouse or by using the space bar. You should experiment with this.

4. Consider the Member table. Previously you added a gender field. Open the Member table in design view and change the datatype for gender to be Lookup Wizard. The wizard will automatically present 3 successive popup windows where you will:
  - i) Specify that you are providing the lookup values;
  - ii) Enter the values (Male, Female, and NotDisclose);
  - iii) Specify that values are to be limited to your list.

Save the table. Change the Member table to Datasheet view so you can test the datatype you have just created. You will notice the user sees a drop down list containing Male and Female, and the user cannot enter or select an inappropriate value into the table.

## 2.1.2 Properties

While creating Microsoft Access tables, each field must have a declared data type as discussed above. According to the data type, Access will present to you a set of field properties that you can apply to the table design for your data and how data is stored in the database. We will discuss the following: Field Size, Format, Input Mask, Caption, Default Value, Validation Rule & Validation Text, Required, Indexed, Show Date Picker, and New Values.

As referenced earlier in MyUniversity database, department table stores department information. Departments can be identified by the department code and department name. When you open the Department table in Design View, you can view the table design grid (top portion of the window) containing the declared field names, data types, and descriptions. In the lower portion of the window,



you will notice the “Field Properties” section where you can specifically set the properties of a currently selected table field.

## *Field Size*

In the Department table, let’s focus on the deptCode field and the associated Field Size property. Suppose the University uses 3 and 4 character values for department codes. The deptCode field property can be set to have a Field Size of 4. The Field Size property enables you to set a maximum size limit on data entered in that column to limit the possibility that an end-user accidentally types a longer string of characters and thereby enters incorrect data. In this way, we can limit the kinds of data entry errors users make when they input data into the table and thus improve the overall quality of our database structure.

Data integrity is a serious issue for databases. Setting Field Size for Text data and Number data is a common thing to do. Often organizations will limit the data they collect for fields such as last name and first name (for example, 30 characters). If the data type is Number then values selected for Field Size are values such as Byte, Integer, Long Integer, etc. These kinds of values are associated with an increasing number of memory locations used per value. A selection of Byte restricts storage to 1 byte of memory (8 bits), and since the largest positive integer that can be stored in a byte is 255, the values stored in the field are forced to be in the range from 0 to 255. Further information is readily available if you use the F1 function key on Field Size for a Number data type.

## *Format*

The Format property is used to customize the way text, number, dates, and times are displayed to the end user. For instance,

selecting Medium Date causes values like January 14, 2013 to be displayed as 14-Jan-13; selecting Long Date results in the display January-14-13. See figure 2.8 for these format examples. If you have Text data (such as department code), you could display the text in uppercase letters by specifying the > symbol as the format code. Another interesting Format specification is @;None. If this is used and there is no value to display in the field, the word “None” will be displayed in the field to the user. In this final example, consider the field for the student identification number (SIN). You may have seen these displayed to users with hyphens between the 3rd and 4th digits and the 6th and 7th digits. If the SIN is a Text field of length 9, it can be displayed using a Format specification of @@@-@@@-@@@.

Value In Field	Format Property	Displayed as
michelle obama	>	MICHELLE OBAMA
January 17, 2019	Medium Date	17-Jan-19
January 17, 2019	Long Date	January-17-19
	@;None	None
786456789	@@@-@@@-@@@	786-456-789
	@@@-@@@-@@@;None	None

**Figure 2.8 Format Examples**

*Input Mask*

The Input Mask property allows the user to know how data will be entered into the table according to a specific pattern. This feature improves the overall quality and consistency of data entered in a database. When the cursor is in the Input Mask area, a ‘builder button’ appears. When you click this button, you will see a list of popular controls. If you were to choose the sample mask for a phone

number, you will see the control !(999) 000-0000 appear. As a result of this choice, the user must enter a 7-digit phone number with an optional 3-digit area code.

### *Caption*

The Caption property helps provide you with a descriptive heading for the defined field instead of simply displaying the field name. If there is no caption, the heading will display the Field Name for the data. Sometimes the field name is not what you want your users to see. Instead of displaying the field name, deptCode, as the heading above a list of department codes, you may prefer to use the words *Department Code*. To accomplish this, just enter the heading text in the Caption property for the field.

### *Default Value*

To help you expedite time when entering data into your database, you can include a default value for a table field. If there is a common value for a field, you should consider setting a default value which will automatically appear in the field for all new records. For example, if most courses are 3-credit hour courses then the value 3 can be set as the default value for all new courses.

### *Validation Rule & Validation Text*

When you enter data into a database, validation rules limit what data is entered into the field to improve data quality in the table. If a field has a validation rule then the rule is tested whenever the user enters data. If the test fails, the user is prompted with a message containing the validation text. To apply a validation rule, you could

enforce the number of credit hours to be less than 10 by entering the rule <10. The validation text to support the validation rule would display *Please enter a value between 0 and 10.*

## *Required*

You can make a field contain a Required property according to the mandatory information that needs be entered into the column in an Access table. Consider the deptName field in the Department table. If a user enters data for a new department then it is unreasonable for the deptName field to not have a value. To ensure there will be a required field in a table, we would select Yes for the Required property setting.

### Indexed

Microsoft Access automatically creates an index (unique – no duplicates) on a field that is the primary key. A unique index is a special internal data structure that Access builds to facilitate two things:

- (1) to ensure fast access to rows of data when the user specifies a value for such a field in a query, and
- (2) to ensure in the case of no duplicates that no two rows of the indexed table could have the same value for that field.

The index data structure is very similar to the index you see at the back of a book. An index comprises several entries where each entry has a value (a term used in the book) and a reference (a page number in a book). In the case of duplicates allowed, there can be several references (several pages where the term appears).

You may choose to have an index on any field. If a field could have duplicate values then you must choose an index that allows duplicates.

## *Show Date Picker*

If the data type is Date/Time, this selection option enables you to select a date using the date picker with a displayed calendar icon. In the table's Datasheet view, the calendar icon will display to the right of the chosen field where you would like to enter your date. Click the calendar icon, select your desired date, and the proper calendar control will display. This is a convenient tool for data entry to ensure the accuracy of a selected date from the displayed calendar icon.

## *New Values*

As you recall, Autonumber is a data type which will automatically generate incremented values as the primary key in the Datasheet View. In your Access table, click Design View and select the primary key field. If the table design data type is AutoNumber, you can choose New Values (Field Property) to specify whether the next value for the field will be the next highest integer or it will be a random integer.

## 2.1.2 Exercises

In this exercise, you will be using your **MyUniversity** database.

1. Open the *Department* table. In design view:
  - Set the deptCode field to have a length of 4. Use > (greater than symbol) as the display format to display deptCode text as uppercase text.
  - Set the length of the deptPhone field to be 10.
  - Choose the Phone Number input mask for the deptPhone field.

- Save the Department table.
  - Switch to Datasheet View for the Department table.
  - Use *Figure 2.1 Department Table* as a guide to enter data into the Department table.
2. Create a new *Course* table. In design view:
- Add the following attributes to include department code, course number, title, short description, and credit hours.
  - Set department code, course number, description and title fields with Short Text data types.
  - The credit hours field should have number data type with no decimal places..
  - Set the deptCode field to be Short Text with a length of 4 so that it matches the properties of deptCode in the Department table. Later, it will be important that the deptCode field in both Department and Course are defined the same.
  - Use *Figure 2.2 Course Table* as a guide and enter data into the Course table.
    - Note: Data entered into description field should accommodate Short Text data type. If there are additional characters entered in field, the data type should be changed to Long Text data type.

In this exercise, you will be using the Library database.

1. Consider the firstName and lastName fields in the Member table of the Library database.
  - Modify the caption for these fields to be First Name and Last Name respectively.
  - Save the table and reopen in Datasheet View. You will see these captions at the top of their respective columns.

2. The Loan table has fields that are defined with the Date/Time data type. Experiment applying different formats for these dates.
3. Consider the id field in the Member table of the Library database. In Design View, change the increment property of the id field to be *random* (instead of *increment*). This is a non-reversible action (but you can download the database later to get a fresh copy). Now add some new members. *What can you say about the id values that are assigned?*
4. Validation rules and validation text are important features to assist database users.
  - Consider the Loan table and its date fields. Microsoft Access has many built in functions one of which is Date() which always returns today's date.
  - To ensure that someone always enters a due date later than today, enter the following properties for the dateDue field:

Validation rule: >= Date()

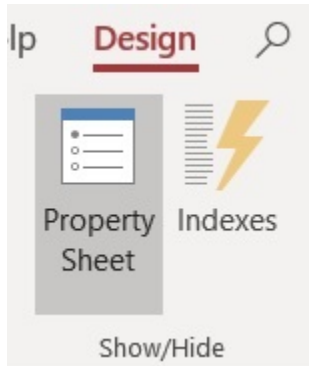
Validation text: Enter a future date.

In this situation, we are entering a field-level validation rule. These rules are useful when we can state a requirement independent of other fields. Test the effect of this validation rule by switching to Datasheet View and entering valid and invalid values for the due date.

- To ensure the date borrowed value is less than or equal to the date returned value, we need to construct a validation rule that involves two fields. Microsoft Access will not let you enter this rule at the field level; instead, such a rule must be specified at the table level. To enter a table level rule,
  - In Design View in the Loan table, click on the

dateBorrowed Field Name.

- In the Show/Hide group, click on the Property Sheet button to display the Loan table properties. The Property Sheet pane will be displayed on the right side of your window.



**Figure 2.9 Property Sheet button**

- Enter the the following table properties for the dateBorrowed field:

Validation rule: [dateBorrowed]<=[dateReturned]

Validation text: Date returned cannot be prior to date borrowed.

The square braces, [ ], that appear in the expression are required. These square brackets reference the field names in the Access table.

- Enter this rule and verify that it prevents the user from entering improper dates.



## 2.1.3 Primary Keys

This section will require that you have previously created the Department and Course tables in your MyUniversity database.

When designing and creating a database, every table should have a declared primary key field(s) which will contain unique values. The purpose of a primary key is to establish relationships between tables and key fields in a relational database. In the MyUniversity database:

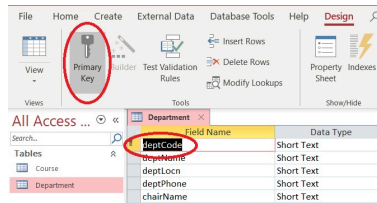
- The Department table has declared deptCode as the primary key.
- The Course table has a composite primary key – a key formed using two attributes: deptCode and courseNo.

To set a primary key, the table must be open in Design View. The primary key field is displayed as the first field in the table field names. You must select the field (or combination of fields) and then click the Primary Key icon. The deptCode is the primary key for the Department table. The Course table comprises two fields as its primary key which uniquely identifies each record. Since the primary key involves more than one field, this primary key is referred to as a *composite key*.

## 2.1.3 Exercises

In this exercise, you will continuous using your **MyUniversity** database.

1. Set the primary key for the Department table. With the Department table in Design View, select the deptCode field and then click on the Primary Key button. When done successfully, you will see the deptCode field with a key icon beside it:



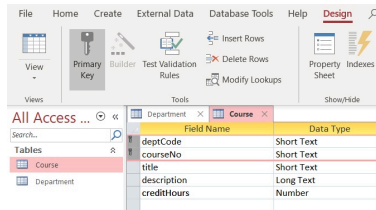
**Figure 2.10 Setting the Primary Key for the Department table**

If Microsoft Access rejects your primary key, you must examine the data values you previously entered for deptCode in the table. The error could occur due to duplicated record values. If this happens, you must view the table in Datasheet View and find the duplicated value and make necessary changes.

Once Access has accepted your primary key, you should open the table in Datasheet View and experiment: *How does Access respond if you try to create a new row with an existing primary key value?*

2. Set the composite primary key for the deptCode field and CourseNo field in the Course table. Begin by first selecting the first field key field in the table. While holding the CTRL (control) key down on the keyboard, select the other field. With both fields selected, click the Primary Key button:

- a. Select the deptCode (click the cell to the left of the deptCode field).
- b. Select the next field to be part of the PK. Hold down the CTRL (control key) on the keyboard. Click the courseNo field (and now release the CTRL key).
- c. Click the Primary Key icon and save your table. You will now see the key image beside both fields as in:

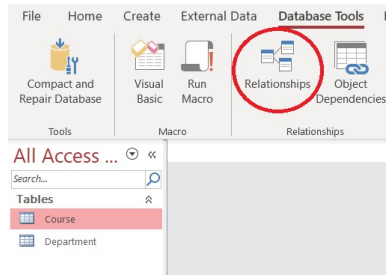


If Microsoft Access rejects your primary key, you must examine the values you previously entered for deptCode and courseNo for duplicate value(s). If this happens open the table in Datasheet View and examine the rows to find duplicate values of the combination {deptCode, courseNo}.

Once Access has accepted your primary key, you should open the table in Datasheet View and experiment: *How does Access respond if you try to create a new row with an existing primary key value?*

3. **Advanced:** Later on, we will discuss the different types of relationships between tables. Perhaps, you are curious on how relationships are created? The Department and Course tables are related to one another through the deptCode field. It is reasonable for us to expect that a deptCode value in a row of the Course table also appears in a row of the Department table. If we are recording a course for the mathematics department then we expect the database to have a corresponding row in the Department table. To ensure this is the case, a formal relationship between these two tables using the Relationships Tool can be created as follows:

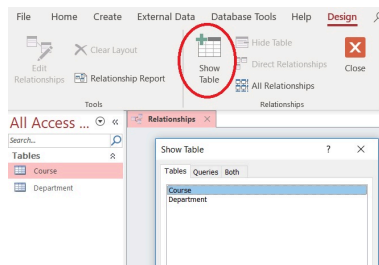
- a. Select the Database Tools tab, in the Relationships group, click the Relationships command:



b. The Relationships Tool opens and you will see a blank relationships diagram.

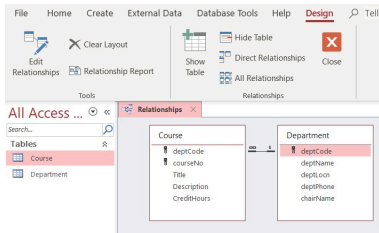


c. From the list of tables, select both Department and Course, and then click Add.

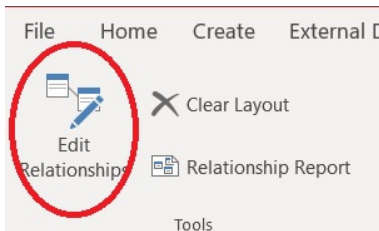


d. With both tables showing on the diagram:

- Select the deptCode primary key field in the Department table, drag deptCode field to the Course table, and release the mouse button above the deptCode field in the Course table.



- Verify the accuracy of the established relationships between the Department and Course tables.
- On the Ribbon, click on the *Edit Relationships* button and the Edit Relationships dialog box appears.



- Verify that the table names and associated field names are the common fields displayed for this relationship. If a field name in the Edit Relationships dialog box is not correct, click on the drop-down dialog box for the incorrect field

name, and select the appropriate field name displayed in the Table/Query list.

- If you follow these instructions, you will be able to enforce referential integrity (RI).
- To enforce referential integrity for the relationship between Department and Course tables, click on the Enforce Referential Integrity check box.

Edit Relationships

Table/Query:	Related Table/Query:
Department	Course
deptCode	deptCode

☒ Enforce Referential Integrity  
☐ Cascade Update Related Fields  
☐ Cascade Delete Related Records

Relationship Type: One-To-Many

Buttons: OK, Cancel, Join Type.., Create New..

- If referential integrity is enforced then it becomes impossible to have a row (record) in the Course table without a matching row in the Department table.
- If you attempt to create relationships in your database:
  - Examine if there is an existing relationship saved in the database.
  - If there is an existing relationship saved between two tables, the existing relationship will need to be removed before modification.

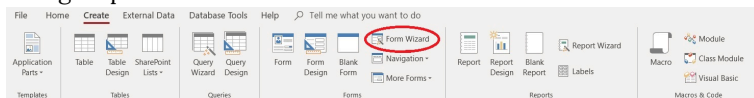
## 3. Creating Forms

RON MCFADYEN

For each table you should create a basic form that can be used to manage data for the table. Once forms have been created, your users will have a more user-friendly way of entering and managing data in your database. (Microsoft Access Datasheet View is not considered user-friendly).

### 3.1 Using The Form Wizard


There are many ways to create forms. We discuss the simplest approach here. In Microsoft Access, click the *Create* tab in the Forms group and then select the *Form Wizard*:



**Figure 3.1 Microsoft Access – Form Wizard**

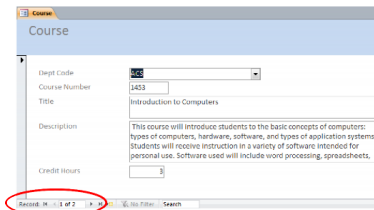
### Figure 3.1 Use Form Wizard

The Form Wizard steps you through a sequence of choices where you choose the tables/queries for the form, the fields to appear on the form, the layout for the form, the style, and the title to appear at the top of the form. At this point, you should create two forms in the MyUniversity database including one form for the Department table and one form for the Course table. As the wizard steps you through each case, you should select/enter:

1. All *fields* of the table to appear on the form (try clicking the  button when it shows to move the fields from the Available Fields column to Selected Fields column)
2. A *Columnar* layout
3. A style of your choice
4. A title of your choice

The last thing you do with the wizard is choose one of: *Open the form and view data*, or, to *Modify the form's design* – choose to open the form to view data. Note that data appears according to information you provided for data types and properties.

A major difference now is that the user will see just one row at a time. Notice the navigation buttons at the bottom of the form where the user can click to navigate through the rows of the table or to add a new row:



**Figure 3.2 Navigation buttons on a form**

## 3.2 Modifying the Form


You can make forms easier to use by adding buttons for common operations for:

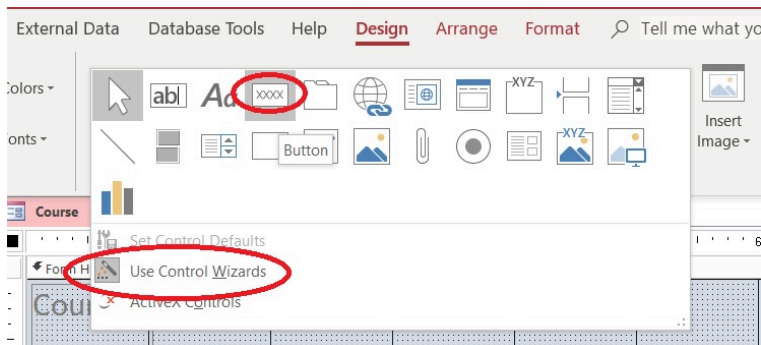
1. Adding a new row
2. Deleting the currently displayed row



### 3. Closing the form

#### 3.2.1 Adding A Button

To add buttons to an existing form, you must open the form in Design View. At this point make sure that *wizard* capabilities are available – if necessary select the *Use Control Wizards*:  (see below) to turn wizard capabilities on. To add a button you click the *Button* button:



**Figure 3.3 Use Control Wizards when adding controls to a form**

Next, you click a location on the form where you would like the control button to be placed. For example, click in the area labeled *Form Footer* and space will be added to the form's design to accommodate the button.

Because *Use Control Wizards* is on, the system will take you through a series of Command Button wizard prompts where you specify the nature of the button type. In this activity, you need to add three buttons on your form. Consider selecting/entering the following at the pertinent prompts:

Button	Category	Action	Text/ Picture
Add button	Record operations	Add new record	New Row
Delete button	Record operations	Delete a record	Delete Row
Close button	Form operations	Close form	Close

**Figure 3.4 Types of Button categories**

A Course form in Design View with three buttons in the Form Footer section:

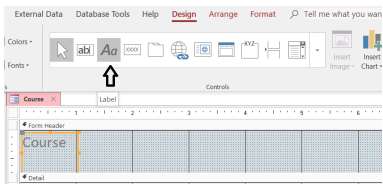
**Figure 3.5 Course form with three buttons in Form Footer**

Anytime after creating a button, you can switch to *Form View* to test your design. If some button is not working as you like then just switch back to *Design View*, delete the button and try again.

### 3.2.2 Adding A Label

A label is a control that holds text for display purposes only. By

default, Microsoft Access adds a label containing the table name in the Form Header area of the form.



**Figure 3.6 Course form with header text in Form Header**

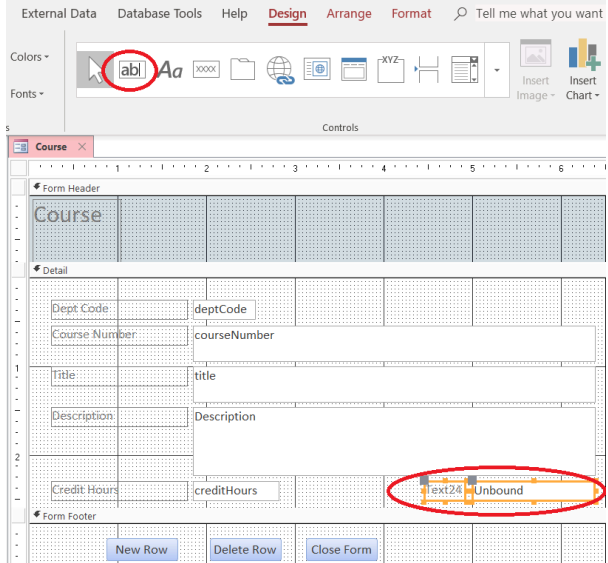
To add a label, you must click the Label control, then click (and drag for sizing) where you want the label placed. You can then type the content for the label and adjust its properties for formatting (e.g. font size, color, ...).

### 3.2.3 Adding a Calculated Field

A calculated field involves a calculation using existing fields which can be created in a table, query or form. In this example, we will be creating a calculated field in our form to determine the total number of student study hours for each credit hour. For instance, the expression will include multiplying the value of the credit hours (Course table) in a registered course times 2.5 study hours. The result of the expression will display the total number of hours a student should study for a course.

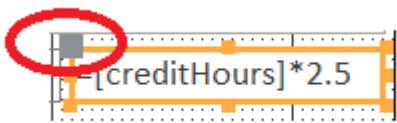
To add a control where a calculated value will be displayed in your form, you must click the Text Box control. Click (and drag for sizing) where you wish the control to be placed on the form. You will see two controls placed in the form: a label and a text box. For the label, click on the label and enter the text *Study hours*. In the text box, you

would click on the text box and enter a formula. The expression for the number of study hours formula would be:  $=[\text{creditHours}] * 2.5$  .



**Figure 3.6 Text Box control on a form**

Adjust the size and location of the controls as necessary. To do this can be a little tricky. To move a control you must select the control, and then click (and drag) the large dot in the control's upper left corner:



To resize a control you must position the mouse so you can see a resizing indicator:



## 3.2.4 Sample Course Form Solution

Dept Code	Course Number	Title	Description	Credit Hours	Study Hours	=[CreditHours]*2.5

**Figure 3.7 Creating a Study Hours calculated field**

## Exercises

1. Open the Orders database (see databases for these notes or instructions from your instructor). Create a new form named OrderDetails. Select all fields from the OrderDetails table to display in your form. Add a textbox on your form to calculate an extended price formula. Enter the text *Extended price* as the label. In the text box, enter an extended price formula calculating quantity times unitprice. Open your form in Form View and view the data to verify your calculated field displays properly.
2. Note that you can modify the properties of fields on a form. When you are in Design View for this form, you can right-click a the extended price field and select *Properties*. In the displayed Property Sheet pane, select the Format tab. Select the *Format* property for this calculated field, click on the drop-down menu, and choose *Currency* to modify this format property. Save your OrderDetails form.

3. Open the Library database and create individual forms of your choice or as directed by your instructor for each of the Book, Loan and Member tables.

## 3.3 Advanced Forms

We have discussed simple forms based on single tables above. To manage a complex database, a user will need to work with data that is obtained from more than one table. We will discuss Microsoft Access queries and relationships later in this course. Once you have worked through those subjects, we suggest that you read Appendix A: Forms Involving Multiple Tables.

# 4. Microsoft Access Queries

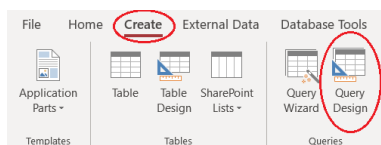
RON MCFADYEN

## Creating Queries

At this time, you have been able to create tables while entering data records into the database. With data now entered into the database, you can create a query that will display data from the tables in a database. A query is a saved Access object that is similar to asking the database questions which can display, sort, or filter this data into useful information.

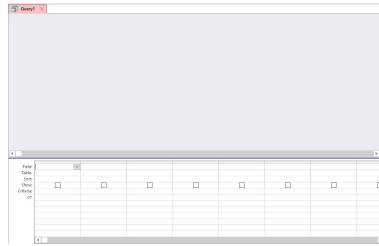
Queries are used for multiple purposes in a database environment. Queries can be used to view table data that contains criteria to restrict the information a user can see. A variety of different query types can be created to view, change, add and delete data. As a result, queries can be executed to form the basis of a Microsoft Access form and report.

We shall use the Library database for the following Access query examples. With Microsoft Access, you can create a query in multiple ways with the select query being the most basic type of Access query. We will start creating our first query examples using Query Design. To create an Access query, click the Create tab and then click the Query Design icon:



**Figure 4.1 Create a query**

As a result, Access opens a Query By Example (QBE) window that you use to specify components of a query:



**Figure 4.2 Query Design – QBE window**

This window comprises two areas: **Relationships** and **Grid**. The Relationships area (upper window pane) will display each table and fields that need to be accessed along with the identified relationships used between those tables.

The **Grid** area (lower window pane) is used to specify:

- fields and tables,
- sort fields,
- fields to be included in the results display,
- different types of criteria can be applied in the row to obtain specific query information,
- calculations,
- grouping of rows with similar values in the field list to display summary information.

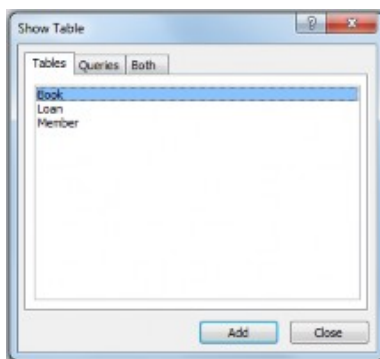
While using the Library database, we will start with simple query examples and progressively work our way to more complex scenarios.



## 4.1 Simple Query

The simplest query is one that displays all of the rows and columns in a table. In our first example, we want to list all of the books in the library. The process of creating the select query is as follows:

- Click on the Create tab (if necessary) and then click on the Query Design icon. Now, you can select the tables for your query in the Library database.
- If the Show Table dialog box has not displayed, choose the Show Table option in the query setup group.
- A Show Table window displays. From the list of tables, you must select the Book table. This task can be completed by double-clicking on the Book name or single click on Book and click on Add button.

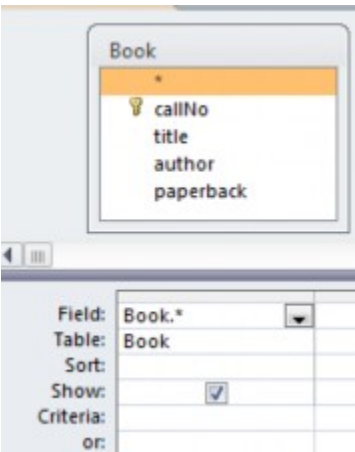


**Figure 4.3 QBE List of table names**

- Choose Close from the Show Table pop up window.

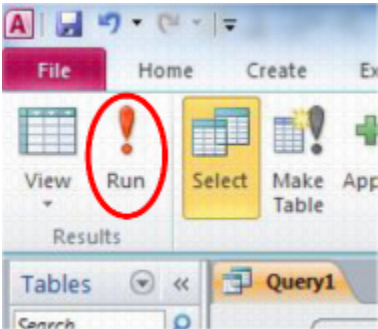
Microsoft Access displays the Book table and its fields in the

Relationships area. The first in this list is an \* which stands for *all attributes*. To *display all records in the Book table*, double-click the asterisk symbol (\*). This results will then be displayed as the following:



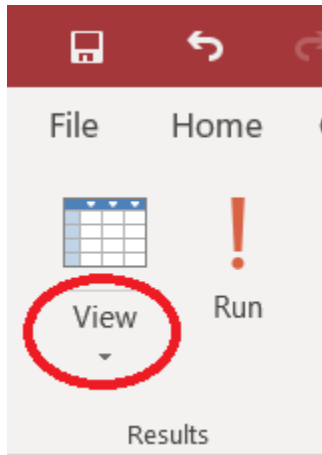
**Figure 4.4 Choosing all fields in the Book table**

We can run the query to test it and confirm the results that we expect: list all rows in Book. To run a query, click the Run (!) icon:



### Figure 4.5 Run a query

There are other views of a query. If you click the drop down just below the View icon:



### Figure 4.6 Several types of query views

You can see all the ways of viewing a query, including:

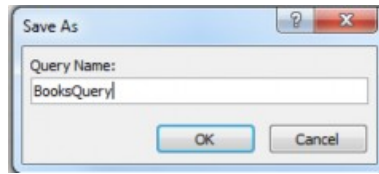
- Datasheet View
- Design View
- SQL View

You can also run a query by choosing *Datasheet View*. When developing a query, one often alternates between Datasheet View and Design View in order to obtain the desired query results. When you run a query, Access will retrieve the display requested table information. In this case, the results of running the query are:

callNo	title	author	paperback
CB 351 M293 1983	Atlas of medieval Europe	Donald Matthew	True
HQ 1143 P68 1975	Medieval women	Eileen Power	False
PC 14 V48 1965	Medieval miscellany	Frederick Whitehead	True
QA 76.73 S67C435 2004	Joe Celko's Trees and hierarchies in SQL for smarties	Joe Celko	False
QA 76.73 S67C46 1997	Joe Celko's SQL puzzles & answers	Joe Celko	True
QA 76.76 A65P76 2011	Programming Android	Zigurd R Mednieks	True
QA 76.9 D26H355 2008	Information modeling and relational databases	T A Halpin	True
QA 76.9 D26H39 1996	Data model patterns : conventions of thought	David Hay	True
QA 76.9 D35C45 1999	Joe Celko's data & databases : concepts in practice	Joe Celko	False
R 141 E45 2006	Medieval medicine and the plague	Lynne Elliott	False
R 487 T35 1967	Medicine in medieval England.	Charles H Talbot	False

**Figure 4.7 Displayed Query results**

Save your query by clicking on the Save button on the Quick Access Toolbar.



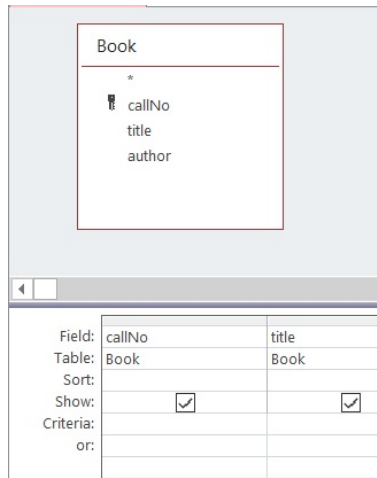
**Figure 4.8 Pop-Up dialog box for saving a query**

After saving the query, you can see the saved query name listed as an Access database object. This would save the definition of the query in the database. The results of the query are not stored or saved since it displays data records from the associated tables. The query, however, can be run any time by an end user. Whenever a user runs the query, the current contents of the Book table are accessed in this example.

## 4.2 Projection Query

Next, we will build a query that displays a subset of columns from a table. Projection queries display a subset of the fields in the table and is said to produce a vertical slice of the table.

While referencing the Book table, we need to display a listing of call number and title values. From the previous instructions, create a new query in Design View. In the Show Table dialog box, select the Book table and close dialog box. Double-click the callNo field and title field to be displayed in the grid.



**Figure 4.9 Query design showing specific fields**

The definition of the query is now complete. The grid area identifies the selected Book table fields. Both of these fields will be displayed because the check box has been selected in the Show row for these fields. Only fields with selected check boxes on the Show row are displayed in the results. Running this query yields:

callNo	title
CB 351 M293 1983	Atlas of medieval Europe
HQ 1143 P68 1975	Medieval women
PC 14 V48 1965	Medieval miscellany
QA 76.73 S67C435 2004	Joe Celko's Trees and hierarchies in SQL for s
QA 76.73 S67C46 1997	Joe Celko's SQL puzzles & answers
QA 76.76 A65P76 2011	Programming Android
QA 76.9 D26H355 2008	Information modeling and relational databas
QA 76.9 D26H39 1996	Data model patterns : conventions of thought
QA 76.9 D35C45 1999	Joe Celko's data & databases : concepts in pr
R 141 E45 2006	Medieval medicine and the plague
R 487 T35 1967	Medicine in medieval England.

Save the query in the Library database using a descriptive query name for the library callNo and title fields.

## 4.3 Criteria Added To Select Query

A select query is one of the most common types of Access queries. Similar to the previous query, select queries may include criteria to limit the subsets of displayed data.

Query designers have options to view all of the displayed records stored in a table or include criteria in a query to identify a limited number of records to include in the query results. In this query example, let us refer to the Book table in the Library database to apply query criterion. We want to create a query listing information about books where the paperback field has a value Yes. Requirements like this are placed on the criteria row of the pertinent field(s).

To develop this new query, we need to select the Book table and

then add fields to the query design grid. Include the callNo, title, author, and paperback fields from the Book table. In chapter 2, the paperback field was added to the Book table with a Yes/No data type. For the paperback field query criteria, we would also enter the value Yes on the criteria line. When the query is executed, Access will compare the paperback field values to determine whether or not the record contains the Yes record value. Save your query as paperbacksQuery.

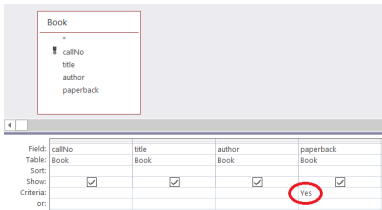


Figure 4.10 Query with selection criteria

When we run the query, we receive the following results listing paperbacks:

callNo	title	author	paperback
CB 351 M293 1983	Atlas of medieval Europe	Donald Matthew	<input checked="" type="checkbox"/>
PC 14 V48 1965	Medieval miscellany	Frederick Whitehead	<input checked="" type="checkbox"/>
QA 76 .73 S67C46 1997	Joe Celko's SQL puzzles & answers	Joe Celko	<input checked="" type="checkbox"/>
QA 76 .76 A659P56 2011	Programming Android	Eugene R. Mednick	<input checked="" type="checkbox"/>
QA 76 .9 D26H355 2008	Information modeling and relational databases	T A Halpin	<input checked="" type="checkbox"/>
QA 76 .9 D26H435 1996	Data model patterns : conventions of thought	David Hay	<input type="checkbox"/>

Figure 4.11a Query results with selection criteria

When a query runs, the query processor accesses the underlying table(s), and displays results where the data meets the criteria specified. For a query accessing a single table consider that the query processor is performing these actions:

For each row in the table:



- Retrieve the row from the database.
- Test the row to see if it meets the criteria specified. A query can compare values in a:
  - Text field by using quotation marks (ie: "Donald Matthew") as the criterion for the field.
  - Numeric field by placing the number (without quotes) as the criterion for the field.
- If the row meets the criteria, the fields marked for show will be displayed.

Select queries allow you to apply criteria (similar to formulas) for specific records you want to be displayed in the query results. The saved query object does not store data instead queries display data stored in your tables as a horizontal subset. Most queries are a combination of selection and projection which can be created from multiple tables. It is typically the case that queries can also be used as a data source for another query, form or report.

## LIKE Operator

Sometimes we need to retrieve information based on partial character comparison information. To find matching criterion, we can use the *Like* operator where we specify an appropriate pattern. These patterns are defined using one or more *wildcard* characters. By default our Microsoft Access databases use the ANSI-89 standard for special wildcard characters.

You can change the standard your database is using by examining and changing the Microsoft Access Options for Object Designers/Query Design.

The ANSI-89 wildcard characters are:

Wildcard Character	Matching criteria	Example
*	Matches any number of characters	Like "1*" matches all text strings that start with "1"
?	Matches any single character	Like "a?c" matches "aac", "abc", "acc", etc. but does not match longer strings such as "aacc" or "xabc"
#	Matches any single numeric character	Like "b#b" would match "b2b" and "b7b" but not "bam"
[]	Matches any single character within the brackets	Like "j[ai]m" matches "jim" and "jam" but not "jaim"
!	Used with [ ] when you do not want to match any of the enclosed characters	Like "b[!ao]b" matches "bim" and "bub" but not "bam" or "bob"
-	Used with [ ] to specify a range of matching characters (given in ascending sequence)	Like "b[0-9]b" would match to "b2b" but not to "bam" Like "b[a-c]b" would match "bab", "bbb", and "bcb"

**Figure 4.11b ANSI-89 Wildcard Characters**

## 4.4 Sorting Data In A Query

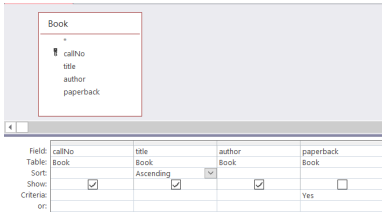
Sometimes an end user wants to view data in a particular field sort order. Fields can be sorted in ascending order (A-Z) or descending order (Z-A). Access will also allow queries to contain either a single sorted field or multiple sorted fields.

Let us extend the previous example to have books sorted in ascending order by title. Create another query similar to the paperbacksQuery. Now, place the cursor in the Sort line beneath

the title field. Click and select *ascending* from the drop-down menu choices. Save this query and run it.

From the query results, notice that all paperbacks records are displayed. Since they are all paperbacks, we will not need to display the paperback field in our final query results. In this query, we are seeing callNo, title and author in the Book table and having the paperback field displayed may be redundant information.

Access allows you to sort and hide fields in your query results. To hide the paperback field in our query, click the Show check box to turn Show off.



**Figure 4.12 Sorted query results with hidden field**

Save this query and run it to notice how the results are sequenced by title without displaying the paperback field.

## 4.4 Sorting Data In A Query Exercises

Use the Library database that is open for this query session.

Create and save the following queries:

1. List the titles of books in descending order.
2. List the titles of books written by Joe Celko.
3. List all members of the library.
4. List the members in sequence (ascending order) by last name.
5. List the members sequenced (ascending order) by last name

and then by first name. (If members have the same last name they appear on consecutive lines, and those lines are in sequence by first name.)

6. Which of the above are a) simple queries, b) selection queries, c) projection queries, d) both selection and projection queries?

## 4.5 AND Operator

Suppose, we want to apply multiple criteria using the Library database to list Celko's books in your Access query. There are two criteria a book must meet:

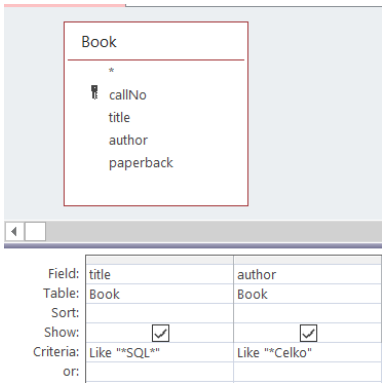
- Criteria 1: the author's name must end with "Celko".
- Criteria 2: "SQL" must appear in the title.

In this case, Criteria 1 must be true to display records containing "Celko" author names *and* Criteria 2 must be true to display books that have "SQL" in the title. The Access query design would use AND logic in the query Criteria row to support multiple criteria. When using QBE, we must place these two criteria on the same criteria row in order that Microsoft Access finds rows that match *both* criteria conditions.

We are also looking for titles that have the text "SQL" anywhere within the title. Access provides a way for us to define such a pattern. The asterisk (\*) wildcard character can be used in a text string that matches these specific letters or combination of letters. This wildcard character can be used to find "SQL" text that is located either at the beginning, end or in between the text title values. Refer to Chapter 6.2.1 Like Operator for additional ANSI-89 wildcard characters.

For Criteria 1, we need two wildcard characters. We specify the pattern that title must match: *Like* `"*SQL*"`.

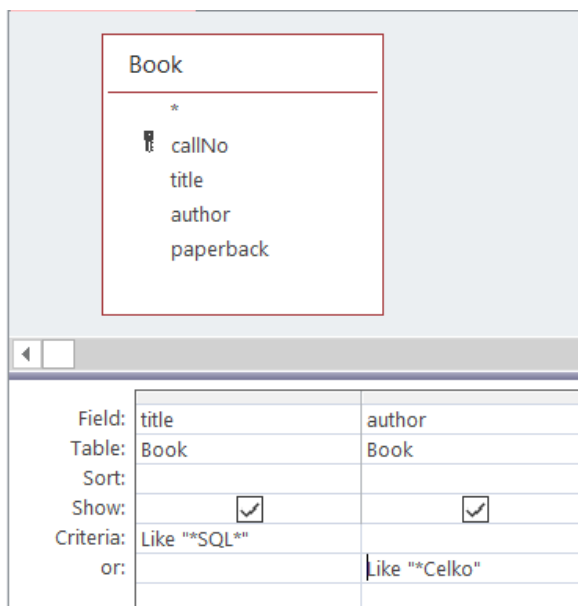
For Criteria 2, we specify the pattern that author must match: *Like “\*Celko”*.



**Figure 4.13 AND Criteria on one criteria line**

## 4.6 OR Operator

Instead of books with titles containing “SQL” **and** authored by Celko, suppose the end user wants a list of books with “SQL” in the title **or** where Celko is the author. In this situation, we place the criteria on separate QBE lines. Access finds records that matches any “or” criteria to be true. A row is selected for the result set if either or both of the criteria are true for a row.



**Figure 4.14 OR Criteria on different criteria line**

## 4.6 AND/OR Operator Exercises

Use the Library database that is open for this query session.

Create and save the following queries:

1. List the titles of books where the author name ends with "Celko".
2. List the titles of books where the author name ends with "Celko" and the text "data" appears in the title.
3. List the titles of books where the author name ends with "Celko" or the text "data" appears in the title.
4. List titles of books where the title contains the word "medieval".
5. List the titles of books where the title contains the words "medicine" and "medieval".

6. List the titles of books where the title contains the words “medicine” or “medieval”.

## 4.7 JOINS

From our queries, we have included criteria displaying a primary key field along with a selection of fields from a single table. By recalling basic table design concepts, each table has a defined primary key field. With the relationship between two tables, there is a defined one-to-many relationship between the key fields.

In the creation of queries, we will want to display data from one or more tables. These defined table relationships are represented by different types of query joins. If a query must be answered using data that appears in more than one table then the query requires a database join.

By continuing to use the Library database, suppose we wish to produce a list of member names and the call numbers of books they have borrowed. Important points about this query:

- The Loan table has the loan information we need
- The Member table has the member names we need.

Before we compose the query, consider how you would produce the results if you were to do this manually. If you had two listings showing the rows of each table in front of you on your desk, you could proceed as follows going through the Loan table listing row by row starting with the first row:

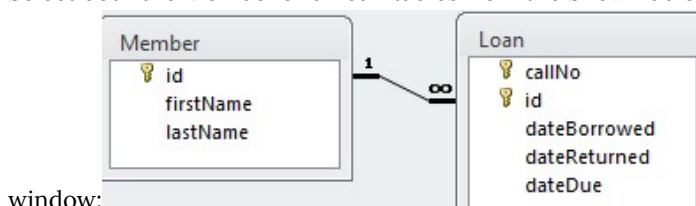
1. For the current row in Loan:
  - a) Write down the call number of this loan.
  - b) Let N stand for the value of the member's id for this loan.
  - c) Now look at the Member listing row by row starting with the first row:

- Examine the row to determine if the row is for member N.
  - If this is correct, write down the member's name beside the call number.
2. If there are more rows in Loan, advance to the next row and go back to step 1.

In the above algorithm, we have determined a member id at step 1. b) and we next look for a matching member id in step 1. c). For a human, the process is simple but tedious. We could say we are trying to go from a row in Loan to a row in Member based on rows having the same value for member id. In database terminology, we say we are *joining* Loan to Member based on a common value of member id. A tedious but well-defined task is something a computer can excel at, and fortunately, we can get the database system to do the job of *joining* rows, based on values of a common attribute, for us.

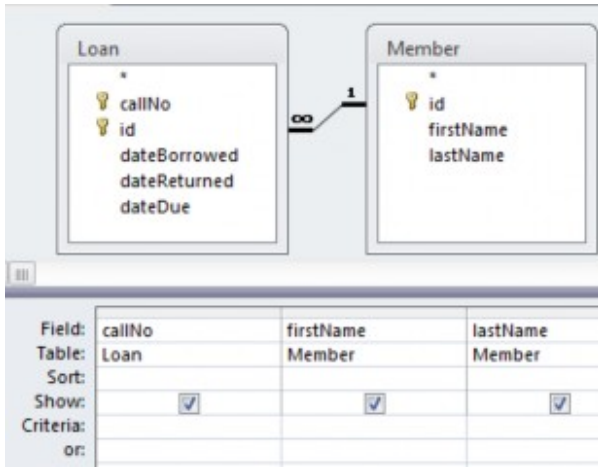
Construct this query as follows:

1. Create a new query.
2. Select both the Member and Loan tables from the Show Table



- window:
3. Note the line connecting the two tables. This is called a *relationships line* which causes Access to join pairs of rows. A row in Member is joined to a row in Loan where the two rows have the same value for id.
  4. Select the call number, first name, and last name fields by double-clicking them to obtain:





5. Run the query and you see the results:

callNo	First Name	Last Name
HQ 1143 P68 1975	John	Smith
QA 76.73 S67C46 1997	David	Martin
R 141 E45 2006	David	Martin
R 141 E45 2006	Betty	Freeman

## Chapter 4 – Query Exercises

Use the Library database that has been used for this query session. In each of the following exercises, the necessary data is in more than one table. Verify and/or specify a Join (default Inner Join type) is applied to your queries.

Create and save the following queries:

1. For each loan, show the title of the book and the date it was borrowed. Note that the title is in the Book table and the date borrowed is in the Loan table.
2. Modify the previous query to produce a listing that is in order by title and then by date.
3. Produce a list that shows for each loan the book title, the name

of the member who borrowed the book, and the dates the book was borrowed and then returned. Note: 3 tables are needed for this query.

4. Produce a list of members and the books they have taken out on loan. Include the member's last name, first name, and titles of the books. The information to be displayed is in 2 tables, but it is necessary to specify 3 tables for this query: Member joins to Loan Book joins to loan
5. Modify the previous query to produce a listing that is in order by last name and then by first name.
6. For member id 2, list the person's name and the titles borrowed.
7. Produce a list of book titles and member names for those books that are due back May 18, 2014.
8. Produce a list of book titles and member names for those books that have not been returned. In this case you must give the criteria for dateReturned as *null*. Null is a special keyword that represents no value.

# 5. Relationships and Relationship Tools

RON MCFADYEN

In this chapter, we will sharing information about database relationships and how relationships are defined from one table to another. The Relationships Tool is used to define relationships between tables based on common fields. Relationships defined using the Relationships Tool are important as they help ensure integrity of data and they provide us with default join criteria for queries involving more than one table.

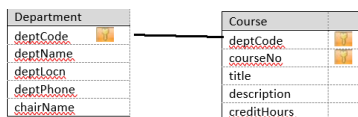
In this section, we will use the **University** and the **Library** databases in our examples.

Consider the University database that contains a Department table and a Course table. These two tables have the deptCode field in common:

In the Department table, deptCode is the primary key and is used to identify a specific department.

In the Course table, the deptCode field is a part of the primary key and indicates the department to which the course belongs.

To ensure that a row in Course is related to an existing row in Department, we can use the Microsoft Access -Relationships Tool to define a relationship between these two tables based on this common field. Using a diagram, we can illustrate this connection between these two tables:



### Figure 5.1 Displaying a relationship between two tables

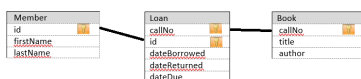
In this situation, we say that deptCode in Course is a *foreign key* referencing the deptCode field in Department.

Now, consider the Library database:

The Loan table has a callNo field as well as the Book table; the callNo field identifies a specific book.

The Loan table has an id field as well as the Member table; the id field identifies an individual member.

In the Library database, we can establish a relationship between the Loan table and the Book table based on the callNo field. A second relationship can be established between the Loan table and the Member table based on the id field. Using a diagram, we can illustrate these two relationships:



### Figure 5.2 Showing relationships involving three tables

The Loan table has two foreign keys identified as callNo and id:

The callNo field in Loan references the primary key (callNo) in Book.

The id field in Loan references the primary key (id) in Member.

## 5.1 Database Integrity In Relational Database Design

### Primary Key

Recall that a table's primary key (PK) is a field (possibly composite) that has unique values. Each record row has a PK value different from any other row in the table. Primary key is a field with a unique identifier. If a query were designed to retrieve a row of that table based on a value of the PK, then at most one row of the table will be retrieved.

### Foreign Key

A foreign key is a field (or combination of fields) in a table B that is associated with a primary key field in a table A through a relationship (A and B can be the same table). Data redundancy is eliminated by having a foreign key in one table related to a primary key in another table.

### Entity Integrity

When we define a primary key for a table, we are enforcing entity integrity. Entity integrity means that each row in the table is identifiable through its primary key. Microsoft Access requires a value for a primary key in a newly added row, and Access enforces uniqueness of those values.

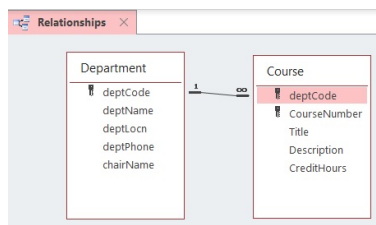
# Referential Integrity

Suppose we have two tables, table A and table B, where a relationship is defined between the primary key of table A and a foreign key in table B. We say referential integrity exists for this relationship if each row in table B has either:

- a foreign key (FK) that does not have a value at all (i.e. it is null) or
- a foreign key (FK) that has a value that exists as a primary key (PK) value in a record row in table A.

## 5.2 Relationships

Tables can be related through one-to-one, one-to-many, or many-to-many relationships. If you open the Access Relationships Tool for the University database, you will see the following diagram showing two tables and a one-to-many relationship:



**Figure 5.3 Access One-to-Many Relationship with Department and Course Tables**

There are two symbols on the relationship line which inform us

the table relationship is one-to-many for which there are two rules that are in place:

- For each department there will be zero or more courses for that department, and,
- Each course is for exactly one department.

To create a relationship in Microsoft Access, you must

- Open the Relationships Tool located in the Database Tools tab.
- Add the pertinent tables to the diagram if they are not there already
- Click, hold, and drag a field (normally this is the PK) of one table to the related field (to become a FK) in the other table.

You will be asked whether or not Referential Integrity is to be enforced. As a general rule-of-thumb, you should select Yes. There must be some exceptional circumstance that makes you select No.

Once relationships are established using the Relationships tool, they are used by Microsoft Access when you create queries. These relationships are then used as your default table joins.

## 5.2.1 One-To-Many Relationship

In relational databases, one-to-many relationships is one of the most common type of relationship between two tables. As described in the previous example, there is one record (also known as parent record) in a table which must be associated with one or more records (child or children records) in a second table to establish a one-to-many relationship. You will need to verify that both the PK field and FK field have the same data type to create this relationship even though these fields have different names.

In Access, you can create relationships by selecting the Relationships command and displaying your tables. You drag the

primary key (PK) field of one table to the other table. The primary key field in the first table must contain unique values. If the foreign key (located in the second table) does not have unique values, then you are creating a one-to-many relationship. The relationship between the two tables will be illustrated by then having Access display a relationship line.

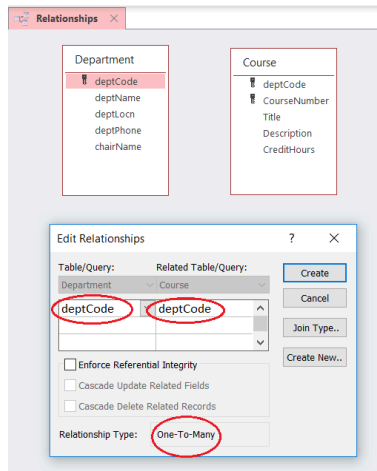
To summarize one-to-many relationships: For each row in the referenced table, there can be several related rows in the other table. For a primary key (PK) value, there can be many rows in the other table with that value stored as the foreign key (FK). Let's look at the following example to illustrate a one-to-many relationship using the University database.

## Practice Your Skills Example

The Department table and Course table are related through the common deptCode field. Both of these fields have the same data type declared in these tables. You can practice your skills in this exercise by creating the relationship between these two tables:

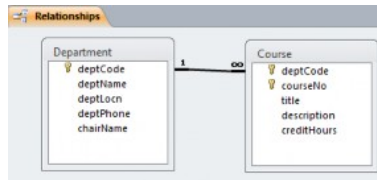
1. Before re-creating the relationship to practice your skills, you must first remove the current relationship between the Department and Course tables. Display the Access Relationships window.
2. Delete the existing relationships line (use your mouse to point and click on the relationship line, press delete, and follow through with the dialog box to delete the relationship).
3. Now, click and drag the deptCode field in the Department table and drop it on top of the deptCode field in Course table. On releasing the mouse, Microsoft Access will display the following dialog box:





**Figure 5.4 Defining One-to-Many Relationship**

4. At this point, Access is requesting the user to confirm the proper fields are being related. The user needs to make a choice regarding Referential Integrity and 'Cascade' options.
  - You should choose *Enforce Referential Integrity* in almost all cases as this helps reduce the chance of corrupting data.
  - We will not be currently discussing 'Cascade Update/Delete Related Fields' in this chapter.
  - Close the Relationship window.
  - From the above example: When the user clicks Create, Access shows the relationships line with 1 on the one side and an *infinity* symbol on the many side of the relationship:



**Figure 5.5 One-to-many relationship: department offers courses**

### 5.2.2 One-To-One Relationship

If you drag a primary key field of one table to another table, and if the foreign key has unique values (a unique index exists for it) then you are creating a one-to-one relationship. For each row in the first table, there can be at most one related row in the other table. A row in the referenced table has a primary key value that equals the foreign key value in at most one row of the referencing table.

### 5.2.3 Many-To-Many Relationship

If you create a relationship in Microsoft Access where both fields you associated (via the click, hold, and drag sequence) do not have unique values (i.e. neither have unique indexes) then Access creates an ‘indeterminant’ relationship. In this situation, a row in one table, A, may be related to multiple rows in the other table, B, and where a row in table B may be related to multiple rows in the table A.

This is not done very often and corresponds to a many-to-many relationship. Most database designers would avoid this in their database designs. If a database designer is faced with two tables, A and B, that are related via a many-to-many relationship, the

designer would likely introduce a third table, say C, where A and C will be related via a one-to-many relationship and similarly, B and C will be related via a one-to-many relationship.

Later in these notes, we discuss database design. We will see how many-to-many relationships can be decomposed into two one-to-many relationships.

## Exercises

### *Relationships*

For these exercises, use the *Company* database which does not have any Access relationships defined for the Employee and Department tables. The first few rows of Employee table and Department table data are as follows:

Employee				
empId	firstName	lastName	supervisor	dept
1	Tanya	Dickson		
2	Heidi	Herring	1	1
3	Hiroko	Hawkins	1	2
4	Emmanuel	Watkins	1	3
5	Oliver	Holt	2	1
6	Raphael	Delaney	3	2
7	Basia	Franks	2	1
8	Bruno	Pena	2	1

Department			
deptId	department	manager	phone
1	Marketing	2	(204) 999-4444
2	Human Resources	3	(204) 999-3333
3	Sales	4	(204) 999-2222

1. Consider the Employee and Department tables. Note: the Employee table has a field named *dept* which indicates the department where the employee works. The relationship can be stated:

- Each department has zero or more employees, and,
- Each employee works in at most one department.

Create a one-to-many “works in” relationship between Employee and Department. Enforce Referential Integrity between these tables.

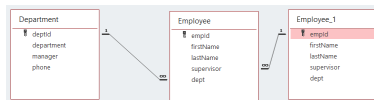
2. The Department table has a field named *manager* which indicates the employee who is the head of the department. The relationship is stated:

- Each department has one employee who manages that department, and,
- An employee may manage at most one department.

There is a unique index defined for the manager field and so you can create a one-to-one relationship “has manager” between Department and Employee.

- In the relationship window, add a **second** Employee table to the relationship window.
- There will be two (2) Employee tables on the diagram.
- Drag the PK empId field from Employee\_1 table to the supervisor field of the Employee table.
- Enforce Referential Integrity between these tables.
- Save the Relationships that you have created for your database structure.

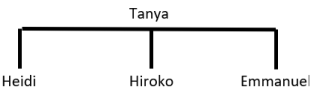
Note how Microsoft Access represents the relationships between these tables.



### Displayed relationships using MS Access – Database Tools

3. Consider the empID and the supervisor fields of Employee.  
Most employees report to someone – someone who is their supervisor. Only employee 1 does not report to anyone else.  
The *supervises* relationship can be stated:
  - An employee may supervise many other employees, and,
  - An employee reports to at most one other employee.
- a) Create the *supervises* relationship. If you are doing this exercise after Exercise 2 then your relationship diagram has 2 copies of the Employee table. You may proceed onto understanding the hierarchical reporting structure in Step C.
- b) If you are *not* doing this after exercise 2, then you must add Employee to the diagram twice so there are 2 copies of Employee on the diagram. Drag the PK empID field from Employee\_1 table to the supervisor field of the Employee table. Note how Access draws this diagram.
- c) Save the Relationships that you have created for your database structure.
- d) Open the *Employee* table in Datasheet View to view the data representing the hierarchical reporting structure. The supervisor field correlates with the empID field.
- e) The supervisor field is an implementation of a hierarchical reporting structure for our company. Use a piece of paper and draw the reporting structure for the company (for the data given at the start of these exercises).  
We have started this exercise showing the reporting structure for the first 4 employees. For example, Tanya is the supervisor for Heidi, Hiroko and Emmanuel. As you analyze

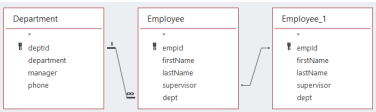
the data further, Heidi, Hiroko and Emmanuel would then supervisor additional employees in the database.



*Queries*

The following query exercises depend on the relationships diagram from the above exercises. When developing a query, you will see that MS Access will include relationships when you add tables in the upper pane of the Query Design window for a query.

Evaluate your table relationships carefully that your previously created. This will ensure that the tables included your queries will then have the correct one-to-many relationship between the Department and Employee tables. In these exercises, you will also need to create an inner join for Employee and Employee\_1 tables. The displayed join between these Employee tables will only display rows where the joined fields from both tables are equal.



**Upper Pane of the Query Design Window**

- 4. Create a query to list for each department, the name of the department and the name of its manager.
- 5. Create a query to list for each department, the name of the department and the names of its employees (the people who

work in the department). Sequence your results by department name.

6. Create a query to list for each department, the name of the department head and the names of the department's employees. Your query must list on each row of the result set the department name, the head's last name, and the last name of each employee. Sequence your results by department name, and within department by employee last name.
7. Create a query that lists each supervisor and the employees he/she is supervising. Your query must list, on each row of the result set, the last name of the supervisor and the last name of the supervised employee. Sequence the results by supervisor and within supervisor by employee.



# 6. Microsoft Access Queries – Advanced

RON MCFADYEN

Previously in Chapter 4 – Microsoft Access Queries, we learned how to construct simple queries using logical expressions including AND criteria and OR criteria to query different types of conditions. Now, we will examine more complex database query situations.

## 6.1 Logical Expressions

Sometimes we need to retrieve data based on multiple criteria which are expressed as logical expressions involving the logical operators *and*, *or*, and *not*. For example, a student using the University database might want to know which courses are offered by the Chemistry and Physics departments which are not 6 credit hour courses. The criteria can be restated with emphasis on logical operators:

- A course is a Chemistry course or a course is a Physics course, and
- The course has any value for credit hours but not 6.

The criteria applied in this example involves *and*, *or*, and *not*. Stating the requirements in our natural language may seem easy to understand. If we state these expressions in an Access Query By Example (QBE) design window, the expressions will require criterion placed in the applicable Criteria row(s).

Microsoft Access provides a way for us to specify the above using

the *Criteria* and *Or* lines in the Query design Grid. We will consider each of the operators And, Or, and Not.

## 6.1.1 AND Criteria

If one specifies multiple criteria on one line in the Access query design grid area, these criteria are identified with AND. For a row to contribute to the result of the query, the row must satisfy **all** the criteria which will result with fewer records being displayed.

### Example

Suppose we want a list of all ACS 3 credit hour courses. We need to obtain the rows in Course where the logical expression

(deptCode="ACS") **AND** (creditHours=3)

is true. We code this in QBE as:

Field	deptCode	CourseNumber	Title	Description	CreditHours
Table:	Course	Course	Course	Course	Course
Sort:					
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:	ACS				3
on:					

**Figure 6.1 Query design containing two expressions with AND criteria**

# 6.I.2 OR Criteria

While one specifies multiple criteria on alternative criteria lines in the Access query design grid area, these criteria are identified with OR. For a row to contribute to the result of the query, the row must satisfy at least one criteria to be true and then the row will be displayed.

If for some row either one or both of the sub- expressions evaluate to true, then the row will be selected for display. This will result with more records being displayed from your Access query.

## Example

In this example, we can use a combination of criteria identified with AND and OR. If one specifies multiple criteria on both the Criteria lines and OR lines, the criteria on each criteria line is ANDed, and those evaluations on alternative criteria lines are then ORed.

Suppose we need a list of all ACS courses that are 3 or 6 credit hour courses. Logically, we can express this as:

(deptCode="ACS" **AND** creditHours=3) OR  
(deptCode="ACS" **AND** creditHours=6)

We code this in QBE as:

Field:	deptCode	CourseNumber	Title	Description	CreditHours
Table:	Course	Course	Course	Course	Course
Sort:					
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:	"ACS"				3
or	"ACS"				6

**Figure 6.2 Two expressions that are ORed**

### 6.1.3 NOT Criteria

While AND and OR criteria compare expressions, the NOT logical operator negates a logical expression.

#### Example

To get a list of 3-credit hour courses, we would use a criteria of 3, but to list courses that are not 3 credit hours one could use the criteria: NOT 3, which, written in long form is:

NOT (creditHours = 3)

Coding this in QBE we have:

The screenshot shows the Microsoft Access Query Design View for a query named 'Course'. The design grid includes the Course table with fields: deptCode, CourseNumber, Title, Description, and CreditHours. The criteria row shows the condition 'Not 3' for the CreditHours field.

Field:	deptCode	CourseNumber	Title	Description	CreditHours
Table:	Course	Course	Course	Course	Course
Sort:					
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:					Not 3
or:					

**Figure 6.3 Using NOT**

## 6.1 Exercises

While using your MyUniversity database, create and save the following Access queries to:

1. List all courses either in the Mathematics department or Statistics department.
2. List all courses either in Mathematics or Statistics where the credit hours are greater than 1 for both courses.
3. Lists the titles of courses offered by the Chemistry department or courses offered by the Physics department that are not full courses (that is, they are not 6 credit hour courses).
4. List all 3 credit hour courses that are not ACS courses or all 6 credit hour courses that are not ACS courses.

## 6.2 Query Operators

We will present two additional query expression operators including LIKE and IN. LIKE is used for pattern matching of text values and IN is used to test for inclusion within a set.

### 6.2.1 LIKE Operator

Sometimes we need to retrieve information based on partial character comparison information. Consider someone using the University database and wanting to find courses where the course description contains the word “computer”. To find courses matching this criterion, we can use the *Like* operator where we specify an appropriate pattern. These patterns are defined using one or more *wildcard* characters. By default our Microsoft Access databases use the ANSI-89 standard for special wildcard characters.

Note: At some point, you may want to investigate the more recent ANSI-92 standard for wildcards. You can change the standard your database is using by examining and changing the Microsoft Access Options for Object Designers/Query Design.

The ANSI-89 wildcard characters are:

Wildcard Character	Matching criteria	Example
*	Matches any number of characters	Like "1*" matches all text strings that start with "1"
?	Matches any single character	Like "a?c" matches "aac", "abc", "acc", etc. but does not match longer strings such as "aacc" or "xabc"
#	Matches any single numeric character	Like "b#b" would match "b2b" and "b7b" but not "bam"
[]	Matches any single character within the brackets	Like "j[ai]m" matches "jim" and "jam" but not "jaim"
!	Used with [ ] when you do not want to match any of the enclosed characters	Like "b![ao]b" matches "bim" and "bub" but not "bam" or "bob"
-	Used with [ ] to specify a range of matching characters (given in ascending sequence)	Like "b[0-9]b" would match to "b2b" but not to "bam" Like "b[a-c]b" would match "bab", "bbb", and "bcb"

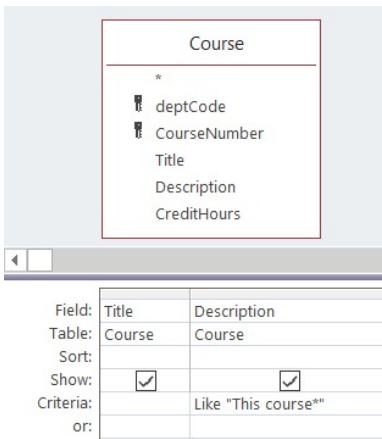
**Figure 6.4 ANSI-89 Wildcard Characters**

## Example

To list courses where the description begins with "This course", you

need a pattern where you specify that a text value begins with “This course” which can be followed by anything else: “*This course\**”.

In the Access query QBE grid, you enter the criteria for title: *Like “This course\*”* :



**Figure 6.5 Using LIKE**

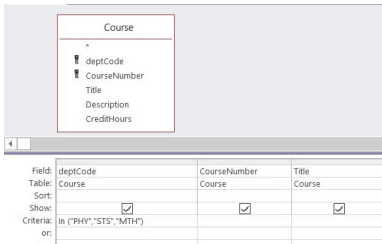
### 6.2.2 IN Operator

The IN operator can be used if you need to determine if a field value is in specific list of values. The list of values is a comma-separated list enclosed in parentheses; for example (1, 3, 6)

# Example

In the Access query QBE grid, use the IN operator in the University database. To list those courses offered by the Physics, Statistics and Mathematics departments you need a list of values: (“PHYS”, “STAT”, “MATH”)

Using QBE, we code IN (“PHY”, “STS”, “MTH”) in the criteria line:

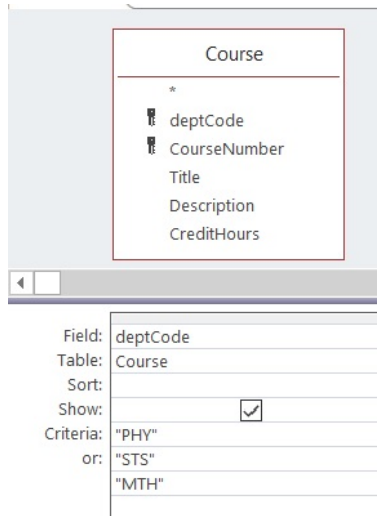


**Figure 6.6 Using IN**

To *exclude* those courses not offered by the Physics, Statistics and Mathematics departments, we code NOT IN (“PHY”, “STS”, “MTH”) in the criteria line.

Note: Using IN is equivalent to using three simple logical expressions that are ORed, and is a convenient way of expression if there are several values in the list:





**Figure 6.7 IN vs OR Operators**

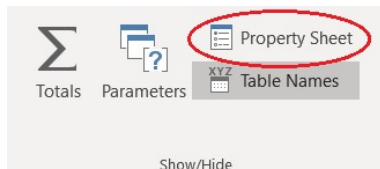
## 6.2 Exercises

Develop the following queries using your MyUniversity database to to:

1. List courses offered by *Physics* or *Applied Computer Science* where this course description contains the word *computer*.
2. List courses where the course description contains the word *computer* but where the course is not offered by the *Applied Computer Science* department.
3. List courses where the credit hours are 1, 3, 6 or 9.
4. List courses where the credit hours are not 1, not 3, not 6, and not 9.

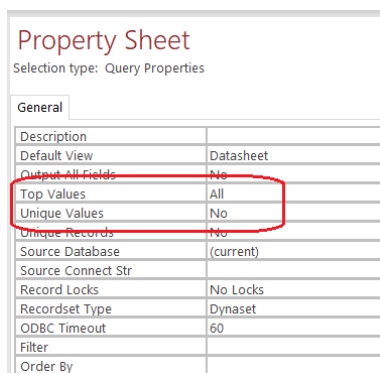
## 6.3 Query Properties

Continue using your MyUniversity database from the previous 6.2 Exercises. Re-open one of queries you created using the query operators. In the upper-right area of a query in Query Design View, you will see a button labeled *Property Sheet*.



**Figure 6.8-A Identifying Access Property Sheet button in Query design**

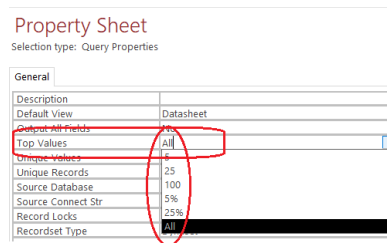
Click the Property Sheet button. In the Property Sheet, you will see properties for a field in the Grid, or, for the query itself, depending on where the cursor is located. Click the mouse in an open area in the Relationships Diagram (upper pane in Query design displaying the added table) and you will see properties for the query. Two query properties we will discuss include *Top Values* and *Unique Values*.



**Figure 6.8-B Query Sheet properties**

### 6.3.1 Top Values Properties

You can change the selection property for Top Values to display ALL records in a table or a partial subset of the records in a table. The default is ALL which results in all rows displayed when the query is executed, but you can also use this property to limit the number of displayed rows. As indicated below, you can select an option or manually type a specific number of rows such as 5, or a specific percentage of rows to be displayed when the query is executed.



**Figure 6.9 Setting the Top Values property**

### Example

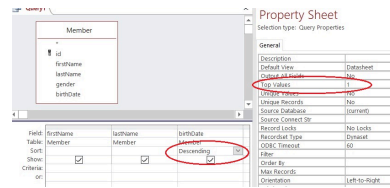
Open the Library database that you previously used in this text. Use the Library database where the Member table has one row per member. Sample data is shown below:

id	firstName	lastName	gender	birthDate
1	John	Smith	Male	15/05/1999
2	David	Martin	Male	06/08/2000
3	Betty	Freeman	Female	18/09/1997
4	John	Martin	Male	11/09/2000

**Figure 6.10 Sample Library Members**

*Scenario:* Suppose we wanted to know who is the youngest member. One way to find out is to sort the members by birthdate and then pick either the first or last row according to how you ordered them (descending or ascending).

*Solution:* Consider the following where the members are sorted in descending order by birthdate and then we list the first row by specifying **Top Values = 1**:



**Figure 6.11 Viewing the youngest member using the Top Values  
property = 1**

From the Member table data, the executed query produces the following result:

firstName	lastName	birthDate
John	Martin	11/09/2000

**Figure 6.12 Query returns one result row**

## 6.3.2 Unique Values Property

While creating your Microsoft Access query, there is a Unique Values property option. If the Unique Values property is set to Yes then Access will search for unique values in the field and eliminate duplicates rows from the result.

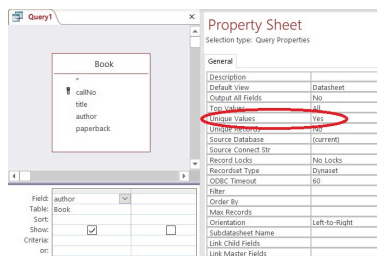
### Example

Suppose a librarian wants a list of authors from the Library database. If we use a query to list the authors but we do not set *Unique Values* to Yes then the result could show an author several times, once for each of his/her books. The following result set shows Jeo Celko listed 3 times:

author
Donald Matthew
Eileen Power
Frederick Whitehead
Joe Celko
Joe Celko
Zigurd R Mednieks
T A Halpin
David Hay
Joe Celko
Lynne Elliott
Charles H Talbot
*

**Figure 6.13 Query displaying duplicate records**

We can eliminate such duplicates by specifying *Unique Values* = Yes as in:



**Figure 6.14 Query Property Setting Unique Value to Yes**

Instead of 11 names being displayed, this query would only list the 9 different author names.

## 6.3 Exercises

1. Consider using the *Library* database. Answer the following questions by creating the following queries.
  1. Which member is the oldest?
  2. Which book was the first one to be taken out on loan?
  3. Which books have been taken out on loan? Any book listed should be listed only once – no duplicates.
2. Consider using your *MyUniversity* database.
  1. Create a query to list the department codes (with no duplicates) of departments that offer 6 credit hour courses.
  2. Modify your query to list the department names too.
3. Consider the *Company* database and its *Employee* table. Answer the following questions by creating the following queries.
  1. The *empId* field is assigned values sequentially starting at 1. What is the last *empId* value that was used? (What is the *empId* for the last employee added to the table?)
  2. Write a query to determine the name of the oldest employee.
  3. Write a query to list all of the employee last names. If at least two employees have the same last name then this list will be shorter than a list of employees.
  4. Suppose there is a field *hireDate* which holds the date when an employee was hired. Write a query to determine the name of the employee who was most recently hired?

## 6.4 Totals Query

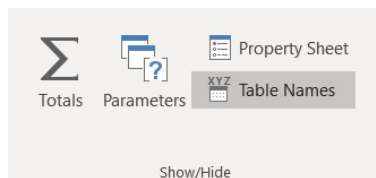
A Totals query allows you to summarize information in the database. When you summarize data from one or more tables, you are either:

- Producing summary data for the whole table, or
- Producing summary data for specific groups.

For instance, you may want to know:

- How many courses there are?
- The average of the credit hours?
- The number of courses in each department?

You can create this Totals query by applying the Microsoft Access aggregate Totals function. To create a Totals query, you begin by creating a simple query that retrieves all the attributes that will be needed to be summarized. Click the “Totals” icon button in the “Show/Hide” button group (upper-right hand corner of the Microsoft Access window):



**Figure 6.15 Totals icon button**

When you click the Totals icon, the QBE Grid will add the *Total* line to your query. You will now see “Group By” displayed for each field in the grid. You must choose from the available drop down options:





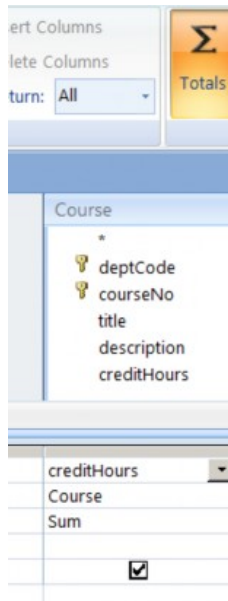
**Figure 6.16 Choices for Aggregate Total line**

For each field in the grid you choose one of:

- **“Group By”**: if the field is used for grouping
- An aggregate function: if the field is to be summarized using that function. We will consider the standard set including *sum*, *average*, *minimum*, *maximum*, *count*.
- **“Where”**: if the field has criteria to be used for selecting rows. Only rows satisfying the criteria contribute to the grouping and display of results.

## Example

The simplest type of totaling query displays an aggregate over an entire set of rows. Consider referencing the *University* database. For example, to sum the credit hours over all courses in the *MyUniversity* database one can use:



**Figure 6.17 Determining the total for one field over all rows**

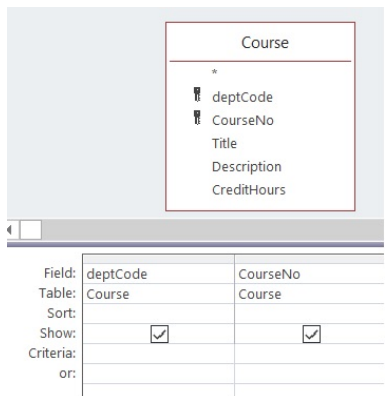
This query summarizes the entire table when executed. The result of this query is one line displaying a sum for all credit hours and courses.

## Example – Count

Typically, the use of the Totaling feature is more complicated. Consider the University database and that we now need to obtain a *count* of the number of courses offered by each department. Counting the number of items in a field is different from the previous Totals query using the Sum function.

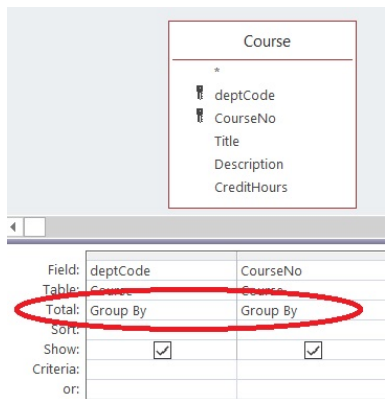
We begin with a query that lists the department code and any

other field in the Course table. (CourseNo is a good choice because it can never be null. Nulls are passed over when the counting of field values is performed). The query below lists the fields we need:



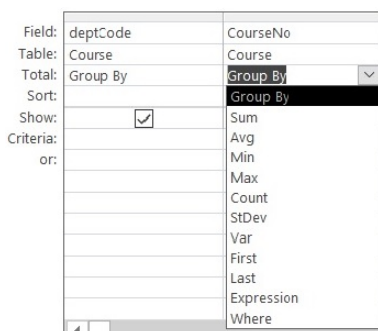
**Figure 6.18 Step 1: Identifying the needed Course table fields**

In the upper right-hand corner of Design View for queries, you must click the Totals icon. When you click the Totals icon a new line (Total line) is added to the grid:



**Figure 6.19 Step 2: Total line is added to the QBE grid**

By default, Microsoft Access sets each field up for grouping. To count the number of courses in each department, you must click in the Total area for courseNo and change from *Group By* to *Count*:



**Figure 6.20 Step 3: Choose the appropriate aggregate for the group**

Now you have a query that will show the value of each department code along with a count of the number of courses for the department. This query produces a count for one row per department.

### *To Review:*

Note: The first 5 aggregate function choices in Figure 6.20 are part of the SQL (structured query language) standard including: SUM, AVG, MIN, MAX, COUNT. They perform a sum, average, minimum, maximum or count over the values found within a group. When a *Totaling* query is executed, the following actions are performed by Microsoft Access:

1. Rows are retrieved from the underlying table(s): Recall that

when *Where* is specified in the *Total* line then there is criteria that must evaluate to true for a row to be part of this result.

2. The retrieved rows are organized into groups where the rows forming a group have the same value for the grouping field(s).
3. For each group, aggregates are evaluated.
4. A group can be eliminated from the results: If there is a criteria specified for a group and if the criteria evaluates to false, the group is excluded.

## 6.4 Exercises

Create and execute queries for the following exercises.

1. Use your *MyUniversity* database to consider the last example where the number of courses per department is listed. The sample database is small and so many departments have just 1 course. Modify the query to list results only for departments where there is more than 1 course. For this you must include a criteria >1 for the field where COUNT is specified:

courseNo
Course
Count
<input checked="" type="checkbox"/>
> 1

1 for the Course field." width="102" height="128">

2. Consider your *MyUniversity* database to create queries:

1. For each department list the department code and the largest value for credit hours.
2. For each department list the department code, department name, and the number of courses.
3. Consider using the *Library* database to create queries:
  1. List the number of books that have SQL in the title.
  2. List the number of members by gender.
  3. What is the total for fines?
4. Consider using the *Company* database to create queries:
  1. List the number of employees in each department.
  2. List departments that have more than 25 employees.
  3. For each employee who is a supervisor, list the supervisor name and the number of employees they supervise.
  4. Suppose the Employee table has a salary field holding an employee's salary. What is the average salary?

## 6.5 Parameter Query

If you need a query but the criteria will not be known until run-time, you use a parameter query. When we compare the previous select queries that have been created with parameter queries, they are very similar in design. A parameter query design uses square brackets [ ] on the Criteria line for a selected field which will allow the user to type inside the [ ] for a prompt when the user runs the query. When a user runs a parameter query, Microsoft Access will show the user the prompt and waits for the user to respond with a value for the parameter. Microsoft Access replaces parameters with the user-supplied values just before it executes the query.

# Example

Suppose a user using the University database needs a list of courses having a specific value for credit hours. The query below has a parameter in the criteria line for creditHours:

Course

\*

deptCode

CourseNo

Title

Description

CreditHours

Field:	deptCode	CourseNo	Title	Description	CreditHours
Table:	Course	Course	Course	Course	Course
Sort:					
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:					[Enter the value for credit hours: ]
or:					

Figure 6.21 Parameter query

When the query is run, the query is temporarily suspended. The user is prompted with the message as provided in the square braces [ ]. Once the user responds to the prompt, the running of the query continues with the value the user entered as the criteria value.

## 6.5 Exercises

1. Consider using your MyUniversity database to create a query to:
1. List all courses in a department (for which the user supplies the department code).

2. List all course titles where the user supplies both the department code and the credit hours. Note that two separate criteria, each with their own parameter, must be specified.
2. Consider the Company database to create a query to:
1. List the employees who manage a department where the

- department code is provided by the person running the query.
2. List all employees in some department where the department code is provided by the person running the query.
  3. Modify the employee data in the Company database so at least two employees have the same first and last names. Develop a query that lists all employees having a specific first name and last name that will be specified by the end user.
3. Consider the *Genealogy* database to:
1. Create a query with two parameters: a *start date* and an *end date*. The query will list all persons whose birth dates fall in the range from *start date* to *end date*.
4. Consider the *Library* database to:
1. Create a query to list books due on a specific date (a parameter).
  2. Create a query to list books written by a specific author (a parameter).

## 6.6 CrossTab Query

Standard Microsoft Access queries produce results with column headings. Crosstab queries are queries where results are displayed with both row and column headings similar to a spreadsheet. Crosstab queries can also utilize aggregate functions that can consolidate data into a group and displayed using row and column formatting styles.

We will limit our discussion to the use of the Crosstab Query Wizard for creating our crosstab queries with your *MyUniversity* database.



# Example

As an example, suppose we wish to display for each department a count of the number of 3 and 6 credit hour courses. The counts are to appear in matrix format where rows are labeled with department names and the columns appear with labels 3 and 6. Below is an outline of how the results should appear:

	3	6
Chemistry	16	7
Mathematic	22	11
...	...	...

**Figure 6.22 Query results to appear with row and column headings**

Crosstab queries have at least three fields: one field (department code) is used for row labels, another field (credit hours) is used for column labels, and one field (course number) is used with an aggregate function (Count).

We can begin by creating a simple query using the MyUniversity database that retrieves all the necessary values:

Course

deptCode

CourseNo

Title

Description

CreditHours

1

2

Department

deptCode

deptName

deptLocn

deptPhone

chairName

Field:	deptName	CourseNo	CreditHours
Table:	Department	Course	Course
Sort:			
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:			In (3,6)
or:			

**Figure 6.23 Query with required fields**

Next, we save the query (say Q1) and create a new query using the Crosstab query wizard. The wizard prompts for

- The table/query to use as the basis for the new crosstab query (the query just saved -> Q1)
- The field to use for row labels -> deptName
- The field to use for column labels -> creditHours
- The field and the aggregate function to use for summarizing data -> courseNo / Count

Running the query shows several columns: the department name (values in this column are the row labels), total over the remaining columns for the row, columns for credit hour values 3 and 6 (the column labels). For example:

CrosstabQuery				
Dept Name	Total Of courseNo	3	6	
Statistics	16	15	1	
Mathematics	18	6	12	

**Figure 6.24 Standard Crosstab Query results**

## 6.6 Exercises

While using your MyUniversity database, create and save the following Access query to:

1. Create and run the query to display for each department a count of the number of 3 and 6 credit hour courses.

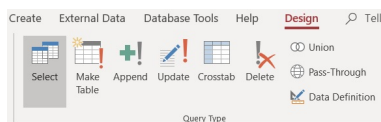
2. Modify the query so that credit hour values appear as row labels and department names appear as column labels.

## 6.7 Action Queries

Action Query is a category that Microsoft Access uses to distinguish queries that can modify the data in the database. We will discuss the query types including: *Make-Table*, *Append*, *Delete*, and *Update*.

To create an action query, one typically starts by creating a Simple Query that is subsequently changed (by clicking the pertinent button) to an Action Query type. You will notice that as you experiment running action queries, Microsoft Access gives a warning message asking you to confirm the changes the query will make to the database. It would be recommended to first make a backup copy of your database prior to making any database changes or modifying your data. The reason for the confirmation warning message is that you cannot click an Undo button to undo such changes as you can in other Microsoft Office applications. To undo a database action query, you would need to design and execute a compensating action query if you did not first make a copy of your database.

When you are in Design View for some query, you will see the buttons for changing the query type:

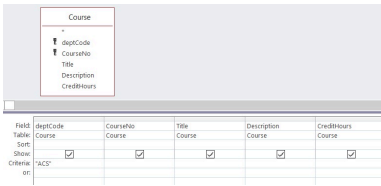


**Figure 6.25 Types of Action Queries**

# Make Table Query

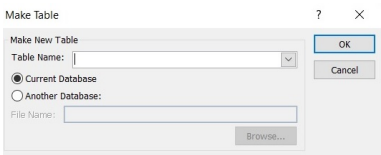
Make table queries are useful if you want to use existing data when you create a new table.

Consider the University database and suppose we need to create a table of ACS courses. We would start with a query that retrieves all ACS courses:



**Figure 6.26** Begin by creating a select query

Next, we change the query to a Make-Table Query by clicking the Make Table button. When you do this Microsoft Access will prompt you for the name for your new table:



**Figure 6.27** Prompt for table name for Make-Table query

The query does not run yet; you must either click the Run button or save the query and run it later. Each time you run the query, Microsoft Access will empty the table and insert rows into it.

## Append Query

Suppose you wish to add rows to an existing table. To do that you must use an Append query. To create an Append query, begin by creating a Simple query that lists the information you wish to see inserted to the table. Once you know the query retrieves the proper information, click the Append button and Microsoft Access will prompt you for the table name that should receive the new rows. After this, you can run the query from the Run button, or you can save the query and run it later.

## Delete Query

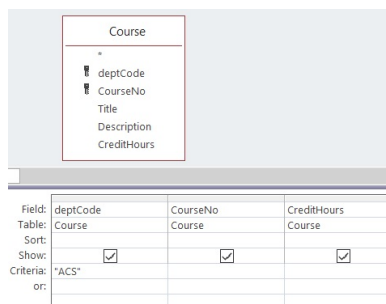
To remove entire rows from a table, you use a Delete query. As in the previous query types discussed, you can begin with a Simple query that retrieves the rows you wish to delete. Once the Simple query is working, you can change its type to Delete and run the query (or save it and run it later). Be careful with this delete query, a delete query can delete many rows in a single run.

## Update Query

The type of query used to modify existing rows in a table is the Update query. In order to create such a query, you should begin with a Simple query that retrieves the rows that are to be updated and then change the type to Update. When you change the type to Update, Microsoft Access will add a new row to the Grid area where you specify the new values for each field to be updated. The new value can be the result from a calculation.

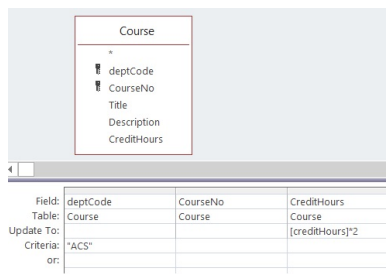
# Example

Suppose we wish to update the Course table so the credit hours are doubled for each ACS course. Continue using the University database. We begin with a Simple query to retrieve the primary (PK) field, the fields to be updated, and the fields needed for selection criteria purposes. In this case, we will need a Simple query to retrieve the department code, course number, and credit hours fields:



**Figure 6.28 Simple select query with criteria**

Next, we change the query type to Update and Microsoft Access modifies the Grid to include an *Update To* line. On that line, we enter an expression that generates the new values. To double the credit hours, we need the expression `[creditHours]*2`, as in:



**Figure 6.29 Update query with Update To line**

## 6.7 Exercises

While using your MyUniversity database, create and save the following Access queries:

1. Create a table of ACS courses, but name the new table ScienceCourses.
2. Does the table ScienceCourses have a primary key? If not, create one.
3. Run a delete query on ScienceCourses to delete all non 3-credit hour courses.
4. Append all 3-credit hour MATH courses to ScienceCourses.
5. Run an update query on ScienceCourses to double the credit hours of all 3-credit hour courses.

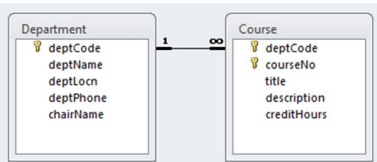
## 6.8 INNER And OUTER Joins

Whenever we use a query to retrieve data from two or more tables, the database query processor performs an operation called a *join*. In this section, we discuss inner joins, outer joins, and Cartesian products. We will also discuss some interesting special cases: self-join, anti-join, non-equi joins.

If we have previously established relationships between tables, and if we have more than one table in a query, then Microsoft Access will create *joins* based on those relationships. If necessary, we can alter, delete, or include new relationships.

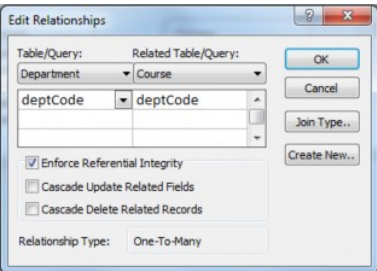
Microsoft Access creates joins where rows join if the join fields are equal in value; such joins are called *equi*-joins. If we create a query

for the University database and add the Department and Course tables to the relationships area of the query we have:



**Figure 6.30 Standard equi-join**

If you edit the relationship line (double-click it), you see the join properties:



**Figure 6.31 Join properties**

Here, we can see the join is based on the common attribute deptCode. If you click on the Join Type button, you will get information on the type of join used. You will see (as the following diagram shows) that Access has selected the first of three options:





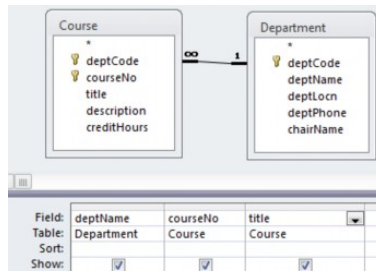
**Figure 6.32 Choosing inner join or outer join**

Joins can be further characterized as *inner* or *outer* joins. Option 1 is an inner join.

Options 2 and 3 are outer joins. One of these would also be called a *Left Outer Join* and the other a *Right Outer Join*. If you examine the SQL statement generated, you will see which is used. Left and Right choices are related to the textual expression of the SQL statement – which table name is leftmost/rightmost in the From clause.

## 6.8.1 INNER Join

All of the joins we have seen up to this point have been inner joins. For a row of one table to be included in the result of an inner join, the row must match a row in the other table. Because all joins so far have also been equi-joins, the matching is based on the values of the join fields of one table being equal to the values of the join fields of the other table. Consider the inner join between Department and Course based on deptCode:



**Figure 6.33 Inner join**

If the tables have the contents shown below:

Course				
Dept Code	Course Number	Title	Description	Credit Hours
ACS	1453	Introduction to Computers	This course will introduce students to the basic concepts of computers: types of computers, hardware, software, and types of application systems.	3
ACS	1803	Introduction to Information Systems	This course examines applications of information technology to businesses and other organizations.	3

Department				
Dept Code	Dept Name	Location	Phone	Chair
ACS	Applied Computer Science	3D07	(204) 786-0300	Simon Lee
ENG	English	3D05	(204) 786-9999	April Jones
MATH	Mathematics	2R33	(204) 786-0033	Peter Smith

**Figure 6.34 Table contents**

then the result of running the query is

Dept Name	Course Number	Title
Applied Computer Science	1453	Introduction to Computers
Applied Computer Science	1803	Introduction to Information Systems

**Figure 6.35 Query result**

In the above result, notice there is no result line for English or Mathematics. This is because in the sample data there were no rows in Course that joined to the English or Mathematics rows in Department. Both rows in Course have a value of “ACS” in the deptCode field and so they joined to the ACS row in Department.

This query demonstrates a distinguishing characteristic of the inner join: only rows that match other rows are included in the results.

## 6.8.1 Exercises

1. Consider the Library database to create a query that:
  1. Joins Loan and Member. List the member name and date due.
  2. Joins Loan and Book. List the book title and date due.
  3. Joins all three tables and lists the member name, book title, and date due.
2. Consider the two tables A and B below.

**Table A**

<b>X</b>	<b>Y</b>	<b>Z</b>
1	3	5
2	4	6
4	9	9

**Table B**

<b>X</b>	<b>Y</b>	<b>Q</b>
1	3	5
1	4	6
2	4	7
3	4	5

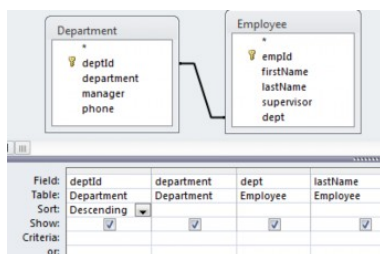
1. How many rows are in the result if A and B are joined based on the attribute X?

2. How many rows are in the result if A and B are joined based on both attributes X and Y?

## 6.8.2 OUTER JOIN

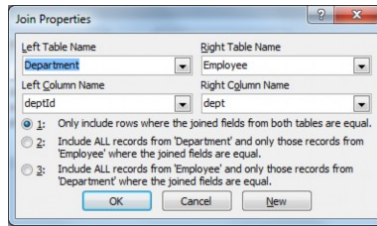
Consider the Company database to support this outer join information. Suppose we wanted to produce a report that lists each department and its employees, and must include *every* department. The two tables would be joined based on equal values of the dept id field. We want all departments and we know that an inner join will not include a department if there are no employees for the department to join to. To get *all* departments included when we are joining two tables, we must use an *outer* join.

Consider the query that is started below:



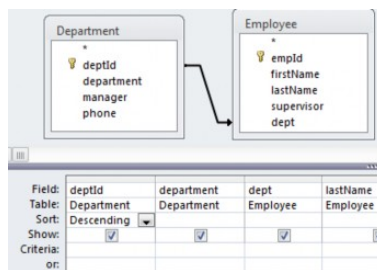
**Figure 6.36 Initial query**

By default the join is an inner join, but with Microsoft Access, you can get an outer join if you edit the relationship and specify either option 2 or option 3, as shown in the dialogue below:



**Figure 6.37 Default property is option 1**

By choosing option 2, your query will include all departments whether or not the department can join to an employee. If there is no employee for a department to join to, then the row is joined to a row of nulls. When you do this, notice the change in the relationship line – it is now a directed line; this is how Microsoft Access illustrates outer joins:



**Figure 6.38 Outer join – all rows of Department**

The first few rows of the result are:

deptId	department	dept	lastName
4	Special Operations		
3	Sales	3	Long
3	Sales	3	Craft
3	Sales	3	Watkins

**Figure 6.39 Query result**

Notice that the Special Operations department joined to a null row.

## 6.8.2 Exercises

1. Consider the Company database and list each department and the number of employees in the department.
2. Consider the Orders database.
  1. Create a query to list each customer and their orders (order id and order date). Are there any customers who have not placed an order?
  2. Modify the above query to list each customer and the number of orders they have placed (include all customers).
3. Consider the library database.
  1. Create a query that will list every book and the date it was borrowed. Include all books in your result.
  2. Create a query to list every library member and the dates they borrowed books. Include all members
  3. Try creating a query that will list books that have never been borrowed.
  4. Try creating a query to list any members who have not

borrowed a book.

### 6.8.3 Cartesian Product

Suppose you create a query, but without a join criteria. This is easily done by clicking on the relationship line and deleting it. When criteria for matching rows is not present, then each row of one table will join to each row of the other table.

This type of join is called a Cartesian Product and these can easily have very large result sets. If Department has 4 rows and Employee has 100 rows then the Cartesian Product has  $(4 \times 100 =)$  400 rows. Databases used in practice have hundreds, thousands, even millions of rows; a Cartesian Product may take a long, long time to run.

### Exercises

1. Consider the Sales database and its Store and Product tables. Construct a query to list the storeID and the productID. When you add Store and Product to the relationships area there is a line joining the two tables. Delete the join line. Run the query. Notice how many rows there are; the number of rows in the result set is the number of stores times the number of products.
2. Consider the Sales database and its Store, Product, and Sales tables. Suppose we want to obtain a list that shows for each store and product the total quantity sold. Note that the end user wants to see every store and product combination.

Hint: An approach you can use with Microsoft Access is to create two queries. The first of these performs a cross product of store and product (call this CP).

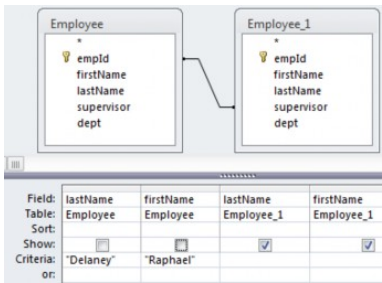


The second query is developed as a *join between the query CP and the table Sales*. CP is outer-joined to Sales in order that every combination of Store and Product is in the final result.

### 6.8.4 SELF-JOIN

A self-join, also called a recursive join, is a case where a table is joined to itself.

Consider the Company database and suppose we must obtain a list of employees who report to another employee named Raphael Delaney (i.e. List the employees Raphael Delaney supervises). To do this, we need to find the row in Employee for Raphael Delaney and then join that row to other rows of Employee where the supervisor field is equal to the empld field for Raphael. When we build the query in Microsoft Access, we simply add the Employee table to the relationships area twice. One copy of Employee will be named Employee\_1. Consider the following query:



**Figure 6.40 Self-Join**

Note the following:

- The criteria specifies the row in Employee will be that of

Raphael Delaney

- The join line connects supervisor to empId and so rows of Employee\_1 will be employees who report to Raphael.

## 6.8.4 Exercises

1. Consider the Genealogy database and develop queries to obtain:
  1. The father of Peter Chan.
  2. The mother of Peter Chan.
  3. The father and mother of Peter Chan.
  4. The children of Peter Chan.
  5. The grandchildren of Peter Chan.
2. Consider the Orders database and the Employee table.
  1. Write a query to list the employee who does not report to anyone.
  2. Write a query to list each employee and the number of employees they supervise.

## 6.8.5 Anti-Join

Suppose we need to list persons in our Company database that are not supervising anyone. One way of looking at this problem is to say we need to find those people that do not join to someone else based on the *supervises* relationship. That is, we need to find those employees whose employee id does not appear in the supervisor field of any employee.

To do this with Microsoft Access, we can construct a query that uses an outer join to connect an employee to another employee based on employeeID equaling supervisor, **but** where the supervisor

value is null. That is, we are looking for an employee who, in an outer join, does not join to another employee. See the query below:



**Figure 6.41 Anti-join query**

This query involves a join, specifically an outer join, and because it retrieves those rows that do not join, it is sometimes referred to as a special case – an *anti-join*.

## Exercises

1. Consider the Genealogy database and develop a query to find people that do not have any sons.
2. Consider the Genealogy database and develop a query to find people that do not have any daughters.

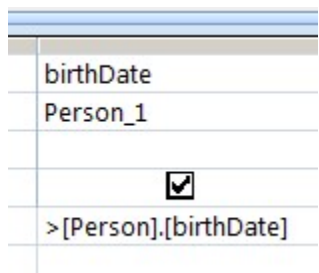
### 6.8.6 Non-Equi Join

A non-equi join is any join where the join criteria does not specify equals, “=”.

Suppose we wish to list all persons in the Genealogy database

who are younger than, say, Peter Chan. One approach to getting the results is to join the row for Peter Chan to another row in Person where the birthdate of Peter Chan is greater than the birthdate of the other person. This type of join would be a “greater than” join as opposed to an equi-join. Proceed in the following way:

1. Add Person to the relationships area twice so there is a Person table and a Person\_1 table. If there are any relationship lines delete them.
2. In the criteria line for Person fields: for firstName type “Peter” and for LastName type “Chan”.
3. In the criteria line for birthDate in Person\_1 type “>[Person].[birthDate]”



**Figure 6.42 Non-equi join**

In this way you are creating a “greater than” join.

3. Include attributes from Person\_1 to display these younger people.
4. Run your query.

## Exercises

1. Consider the genealogy database.
  - a) Run the example from above.
  - b) Modify the example to list those people who are older than Peter Chan.

## 6.9 SQL SELECT Statement

SQL is the standard language for relational database systems. There are variations of SQL that appear in Object-oriented database systems, and elsewhere. The study of SQL (structured query language) is very important and the knowledge gained here is useful in other database environments.

We will examine one SQL statement, the *Select* statement, used to retrieve data from a relational database. Other common data manipulation statements are the Insert, Update, and Delete used to modify or add data. Select, Insert, Update, and Delete all belong to the *Data Manipulation Language* (DML) subset of SQL. Another group of statements belong to the *Data Definition Language* (DDL) subset of SQL. DDL statements are used to create tables, indexes, and other structures and are discussed in a later section.

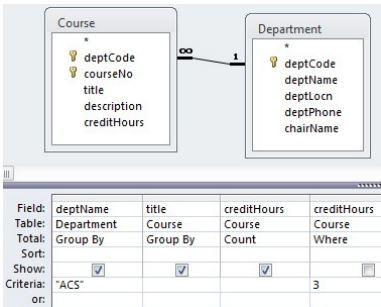
The general SQL **Select** statement syntax:

- Select** list of attributes or calculated results <sup>(1)</sup>
- From** list of tables with/without join condition <sup>(2)</sup>
- Where** criteria rows must meet beyond the join specifications <sup>(3)</sup>
- Group by** list of attributes for creating groups <sup>(4)</sup>
- Order by** list of attributes for ordering the results <sup>(5)</sup>
- Having** criteria groups must meet <sup>(6)</sup>

Each clause of the SQL statement has its counterpart in the Design View used by Access:

- (1) Attribute/calculated values are those for which Show is specified. If grouping is used, these must evaluate to a single value (group functions; grouping attribute) per group.
- (2) Tables that appear in the From clause are shown in the Relationships Area.
- (3) Specifications for the Where clause are found in the Criteria and Or rows.
- (4) Specifications for the Group By clause are made in the Totals row.
- (5) Specifications for sorting are made in the Sort row.
- (6) A Having clause specifies criteria that a group must meet to be included in the result. This clause is generated when you use an aggregate function with a criteria.

When you design a query, you can switch between various views including SQL View. You can easily confirm through examples how the SQL statement is generated from Design View. For example, consider the following query and its SQL expression below. Note how Microsoft Access has used names with dot-notation to fully specify fields and how Access has placed one criteria rows must meet in a Having clause.



```

SELECT      Department.deptName,      Course.title,
Count(Course.creditHours) AS CountOfcreditHours
FROM      Department      INNER      JOIN      Course      ON

```

```

Department.deptCode      =      Course.deptCode      WHERE
(((Course.creditHours)=3))
GROUP BY Department.deptName, Course.title HAVING
(((Department.deptName)="ACS"));

```

**Figure 6.43 QBE and SQL SELECT statements**

## Exercises

1. Consider any of the queries from a previous section. Translate the query into SQL manually and then compare your result to what you see when you view the query in SQL View.
2. Consider the following SQL statements and show how each statement would appear in Design View. You can confirm your result if you create a query, switch to SQL View, type the query statement and then switch to Design View. Unfortunately, if you make any syntax errors, Access will be unable to switch to Design View. Your database must contain the tables in the From clause.

Refer to the Orders.accdb database:

```

a)          SELECT          Products.ProductID,
Products.ProductName, Categories.CategoryName
FROM  Categories INNER JOIN Products ON
Categories.CategoryID = Products.CategoryID
WHERE  Categories.CategoryName = "Beverages"
AND Products.Discontinued = Yes;

```

Refer to the AutosSales.accdb database:

```

b) SELECT Auto.Year, Dealer.Name, Auto.Colour, A
uto.Price
FROM  Dealer INNER JOIN Auto
ON Dealer.DID = Auto.Did
WHERE Auto.Colour="blue" AND Auto.Price>10000;
c) SELECT Auto.Year, Dealer.Name, Auto.Colour, A

```

```

uto.Price
FROM Dealer RIGHT OUTER JOIN Auto ON
Dealer.DID = Auto.Did
WHERE Auto.Colour="blue" OR Auto.Price>10000;

```

## 6.10 SQL UNION AND UNION ALL

The Union and Union All operators merge the results of two or more queries that are given as SQL SELECT statements. With Microsoft Access, you must switch to SQL View to use Union/Union All

- UNION removes duplicates and sorts the results
- UNION ALL returns all values (includes duplicates) without sorting
- The output fields must be identical (number and type) for each SELECT. The syntax for UNION of two Select's :

Union	Union all
SQL SELECT Statement1	SQL SELECT Statement1
UNION	UNION ALL
SQL SELECT Statement2 ;	SQL SELECT Statement2 ;

**Figure 6.44 Union and Union ALL syntax**

Any number of SELECT statements can be united with UNION. A requirement for using UNION is that the queries are union-compatible. These queries must retrieve the same number of fields, and fields in the same position across the multiple SELECT clauses must be of matching types.



## Example

Consider the Employee table in the Company database. To list all names (first and last) in a single column, construct two queries: one to list the first names of employees and one to list the last names of employees.

These two queries can be combined to produce a single list of names. Now, in SQL view type and run:

### **Union Example**

*(sorted with no duplicates)*

```
SELECT firstname FROM Employee
```

```
UNION
```

```
Select lastname from Employee ;
```

## Exercises

1. Modify the above example so that duplicates are eliminated.

# 7. Entity Relationship Modeling

RON MCFADYEN

When designing a database, it is common practice for a database designer to develop an Entity Relationship model and to represent that model in a drawing, the entity relationship diagram (ERD). In this chapter, we discuss the concepts required to develop an ERD and the Peter Chen notation. Peter Chen introduced entity relationship modeling in his paper *The Entity-Relationship Model-Toward a Unified View of Data* (ACM Transactions on Database Systems, Vol. 1, No. 1, 1976). This paper can be found at <http://csc.lsu.edu/news/erd.pdf>; it is one of the most cited papers in the computer field, and has been considered one of the most influential papers in computer science. Another later paper published in *Software Pioneers: Contributions to Software Engineering* (2002) is *Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned* and can be found at [http://bit.csc.lsu.edu/~chen/pdf/Chen\\_Pioneers.pdf](http://bit.csc.lsu.edu/~chen/pdf/Chen_Pioneers.pdf).

Entity Relationship modeling is a process used to help us understand and document the informational requirements of a system as a logical or conceptual data model. When the model is complete, we then create a physical model in some database management system (DBMS); typically a relational DBMS, or relational database management system (RDBMS).

## 7.1 Introduction

In the entity relationship approach to modeling, we analyze system

requirements and classify our knowledge in terms of entities, relationships, and attributes.

## Entities

*Entities* are the things we decide to keep track of. For example, if one considers a system to support an educational environment, one is likely to decide that we need to keep track of students, instructors, courses, etc. Typically, entities are the people, places, things, and events that we need to remember something about.

Suppose we know of four student entities and two course entities. For example, consider four students (say John, Amelia, Lee, and April) and two courses (Introduction to Art and Introduction to History). We can illustrate these in a number of ways:

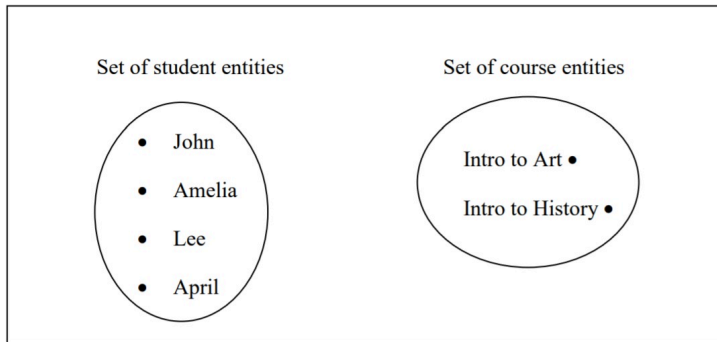
As tables of information:

Students		
Name	Id Number	Phone
John	184	283-4984
Amelia	337	838-3737
Lee	876	933-2211
April	901	644-3838

Courses		
Title	Course Number	Department
Introduction to Art	661	Art
Introduction to History	765	History

**Figure 7.1 Entities shown as rows in table**

Set of student entities



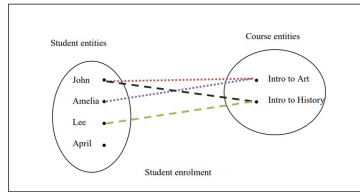
**Figure 7.2 Entities shown as sets**

## Relationships

Entities can be related to one another and so we use *relationships* to describe how entities relate to one another. Continuing with our educational example we know that students enroll in courses, and so this is one of the relationships we should know about. Suppose we have the two courses and four students listed previously. Suppose also that

- John and Amelia are enrolled in Introduction to Art
- John and Lee are enrolled in Introduction to History
- April is not enrolled in any course.

Below, we depict four instances of the *enroll-in* relationship by drawing a line from a student to a course. Each relationship pairs one student with one course.



**Figure 7.3 Relationships shown as lines connecting entities**

## Attributes

Entities and relationships have characteristics that describe them. For instance, the students in our example are described by the values for their name, id number, and phone number. As we look back, we can see there is a student named *John* whose id number is 184 and his phone number is 283-4984.

Courses are shown with a course title, a course number, and belong to a department. There is a course numbered 661 that is offered by the Art department and it is titled *Introduction to Art*.

If we consider the enroll-in relationship, we know there is a date when the student enrolled in the course and a final grade that was awarded to the student when the course was completed. For instance, we could have *John* enrolled in *Introduction to Art* on July 1, 2010 and was awarded an A+ on completion of that course.

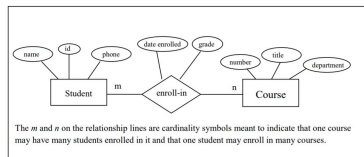
These characteristics that serve to describe entities and relationships are called *attributes*. We will be examining attributes in some detail. As we will see some attributes, such as student number, serve to distinguish one instance from another – each student has a student number distinct from any other student. Other attributes we consider to be purely descriptive, such as the name of a student – many students could have the same name.

# Notation

There are many notations in use today that illustrate database designs. In this text, as is done in many database textbooks, the Peter Chen notation is used; other popular notations include IDEF1X, IE and UML. There are many similarities, and so once you master the Peter Chen notation it is not difficult to adapt to a different notation.

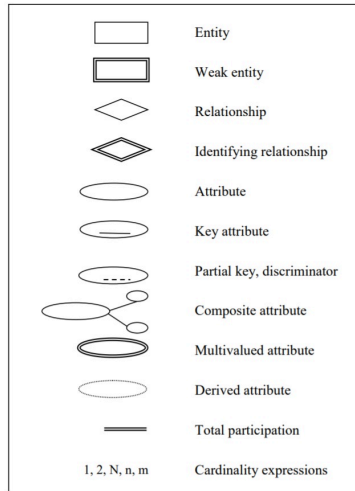
The following is an example of an ERD drawn using the Peter Chen notation. Note the following:

- Entity types are represented using rectangular shapes.
- Relationship types are shown with a diamond shape. Lines connect the relationship type to its related entity types with cardinality symbols (*m* and *n*).
- Attributes are shown as ovals with a line connecting it to the pertinent entity type or relationship type.



**Figure 7.4 An ERD in Chen notation**

The various symbols we use with the Peter Chen notation:



**Figure 7.5 Symbols used in the Chen notation**

## 7.2 Entities

*Entities* are the people, places, things, or events that are of interest for a system that we are planning to build. In the previous section, we considered there were several entities: four students and two courses.

In general, we find examples of entities when we think of people, places, things, or events in our area of interest:

People: student, customer, employee

Places: resort, city, country

Things: restaurant, product, invoice, movie, painting, book, building, contract

Events: registration, election, presentation, earthquake, hurricane

Entity sets are named collections of related entities. From our example, we have two entity sets:

- The Student entity set comprises at least the 4 student entities: John, Amelia, Lee, and April.
- The Course entity set comprises at least the 2 course entities: Introduction to Art and Introduction to History.

Entity sets are the collections of entities of one type. We consider an *Entity Type* to be the definition of the entities in such a set. A common convention is to name entity types as singular nouns and that, at least, the first letter is capitalized.

In an ERD entity, types are shown as named rectangular shapes. For example:



The Student and Department entity types shown above are drawn with a simple single-line border. This means that they are regular (or strong) entity types that are not existence-dependent on other entity types (see the next section).

## 7.2 Exercises

1. Consider your educational institution. Your educational institution needs to keep track of its students. How many student entities does the institution have? You have provided the institution with information about you. In your opinion,



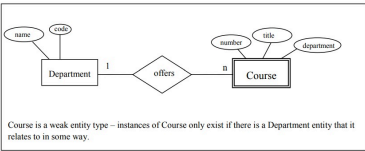
- what attributes describe these entities?
2. Consider your place of work. The Human Resources department in your company needs to manage information about its employees. How many employee entities are there? What attributes describe these entities?
  3. Consider your educational institution or place of work.
    1. What are some of the entity types that would be useful?
    2. What relationships exist that relate entity types to one another?
    3. What attributes would be useful to describe entities and relationships?
    4. Draw an ERD.

## 7.2.1 Weak Entities

Sometimes we know certain entities only exist in relationship to others. For example, a typical educational institution comprises a number of departments that offer courses. So we could have a History department, an Art department and so on. These departments would design and deliver courses that students would register for. In this framework, the courses exist in the context of a department, and the identifier for a course is typically a department code and course number combination. So the history course, Introduction to History, belongs to the History department and it would be known by the identifier HIST-765. HIST is a code representing the History department and 765 is a number assigned to the course; other departments could have a course with that same number, 765.

In these situations where the existence of an entity depends on the existence of another entity, we say the entity is a *weak* entity, and the corresponding entity type is a weak entity type. Weak entities often have identifiers that comprise multiple parts (such as department code and course number). Later on, we will see other

aspects of an ERD that relate to weak entity types. At this time, we should be aware that weak entity types are illustrated in an ERD with a double-lined rectangle:



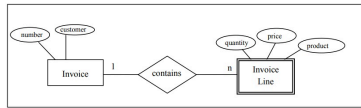
**Figure 7.6 Course as a weak entity type**

Often, when we purchase things the vendor provides, an invoice giving details of each item that is purchased (see the sample invoice below). Appearing on the invoice are detail lines specifying the product, the quantity and price. Invoice lines are things that exist only in the context of an invoice and so each invoice line is a *weak* entity; the invoice lines are existence-dependent on an invoice:

Winnipeg Retail Co		INVOICE	
104 Ave Street, Winnipeg Manitoba, R3B 2G9			
SOLD TO:		INVOICE NUMBER:	
John Smith		16357	
123 Anywhere		INVOICE DATE:	
Winnipeg, MB, R3B 2G9		February 20, 2013	
SHIPPED TO:		YOUR ORDER NO.:	
same as above		1784	
		TERMS:	
		Net 30	
		SALES REP:	
		Jim Jones	
QUANTITY	DESCRIPTION	UNIT PRICE	AMOUNT
15	pens	1.00	\$15.00
15	pencils	2.00	\$7.50
Total less taxes			\$22.50
PST			3.00
GST			2.25
			\$28.00

**Figure 7.7 Sample Invoice**

The following includes a few attributes to show how Invoice and Invoice Line could appear in an ERD.



**Figure 7.8 Invoices (regular entity type) and Invoice Lines (weak entity type)**

## 7.3 Attributes

Attributes are the characteristics that describe entities and relationships. For example, a Student entity may be described by attributes including:

- student number name
- first name
- last name
- address
- date of birth
- gender

An Invoice entity may be described by attributes including:

- invoice number
- invoice date
- invoice total

A common convention for naming attributes is to use singular nouns. Further, a naming convention may require one of:

- All characters are in upper case.
- All characters are in lower case.
- Only the first character is in upper case.
- All characters are lower case, but each subsequent part of a multi-part name has the first character capitalized

Using the last convention mentioned, some examples of attribute names:

- lastName *for* last name
- empLastName *for* employee last name
- deptCode *for* department code
- prodCode *for* product code
- invNum *for* invoice number

In practice, a naming convention is important, and you should expect the organization you are working for to have a standard approach for naming things appearing in a model. A substantial data model will have tens, if not into the hundreds, of entity types, many more attributes and relationships. It becomes important to easily understand the concept underlying a specific name; a naming convention can be helpful.

There are many ways we can look at attributes including whether they are atomic, composite, single-valued, etc. We consider these next.

### 7.3.1 Atomic Attributes

A simple, or **atomic**, attribute is one that cannot be decomposed into meaningful components. For example, consider an attribute for gender – such an attribute will assume values such as Male or Female. Gender cannot be meaningfully decomposed into other smaller components.

As another example, consider an attribute for product price. A sample value for product price is \$21.03. Of course, one could decompose this into two attributes where one attribute represents the dollar component (21), and the other attribute represents the cents component (03), but our assumption here is that such decompositions are not meaningful to the intended application or system. So, we would consider product price to be atomic because it cannot be usefully decomposed into meaningful components.

Similarly, an attribute for the employee's last name cannot be decomposed, because you cannot subdivide last name into a finer set of meaningful attributes.

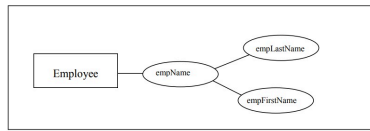
### 7.3.1 Exercises

1. Consider that a Human Resources system must keep track of employees. If we are only including atomic attributes, what attributes would you include for the employee's name. Some possibilities are first name, last name, middle name, full name.
2. In some large organizations where there are several buildings and floors, we see room numbers that encode information about the building, floor, and room number. For example, the room 3C13 stands for room 13 on the third floor of the Centennial building. Suppose we need to include Room in an ERD. How would you represent the room number given that you must include atomic attributes only?

### 7.3.2 Composite Attributes

Consider an attribute containing an employee name which is to represent an employee's complete name. For example, suppose an employee's name is John McKenzie; the first name is John and the last name is McKenzie. It is easy to appreciate that one user may only need employee last names, and another user may need to display the first name followed by the last name, and yet another user may display the last name, a comma, and then the first name. If it's reasonable for one to refer to the complete concept of employee name and to its component parts, first name and last name, then we can use a *composite* attribute. An attribute is *composite* if it comprises other attributes. To show that an attribute is composite

and contains other attributes, we show the components as attribute ovals connected to the composite as in:



**Figure 7.9 Composite attribute**

Attributes can be composite and some of its component attributes may be composite as well (see Exercise #3).

### 7.3.2 Exercises

1. How would you use a composite attribute to model a phone number?
2. Consider the previous exercise set. Show how we can include room number as a composite attribute that has multiple components.
3. Consider an address attribute. Show that this can be modeled as a multi-level composite attribute where the component attributes include street, city, province, country and where street includes apartment number, street number, street name.

### 7.3.3 Single-Valued Attributes

We characterize an attribute as being single-valued if there is only one value at a given time for the attribute.

Consider the Employee entity type for a typical business application where we need to include a gender attribute. Each

employee is either male or female, and so there is only one value to store per employee. In this case, we have an attribute that is single-valued for each employee. Single-valued attributes are shown with a simple oval as in all diagrams up to this point. In all of our examples so far, we have assumed that each attribute was single-valued.

### 7.3.3 Exercises

1. A college or university will keep track of several addresses for a student, but each of these can be named differently: for example, consider that a student has a mailing address and a home address. Create an ERD for a student entity type with two composite attributes for student addresses where each comprises several single-valued attributes.
2. Consider a marriage entity type and attributes marriage date, marriage location, husband, wife. Each marriage will only have one value for each of these attributes. Illustrate the marriage entity and its single-valued attributes in an ERD .

### 7.3.4 Multi-Valued Attributes

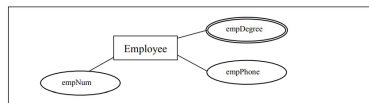
Now, suppose someone proposes to track each employee's university degrees with an attribute named empDegree. Certainly many employees could have several degrees and so there are multiple values to be stored at one time. Consider the following sample data for three employees: each employee has a single employee number and phone number, but they have varying numbers of degrees.

empNum	empPhone	empDegree
123	233-9876	
333	233-1231	BA, BSc, PhD
679	233-1231	BSc, MSc

**Figure 7.10 Employees – number, phone, degrees**

For a given employee and point in time, empDegree could have multiple values as is the case for the last two employees listed above. In this case, we say the attribute is *multi-valued*.

Multi-valued attributes are illustrated in an ERD with a double-lined oval.



**Figure 7.11 Employee degrees shown as multi-valued**

We can use multi-valued attributes to (at least) document a requirement, and at a later time, refine the model replacing the multi-valued attribute with a more detailed representation. The presence of a multi-valued attribute indicates an area that may require more analysis; multi-valued attributes are discussed again in Chapter 10.

## 7.3.4 Exercises

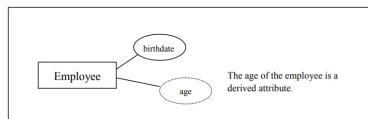
1. Consider the employee entity type.
  1. Suppose the company needs to track the names of



- dependents for each employee. Show the dependent name as a multi-valued attribute.
2. Modify your ERD to show empDependentName as a composite multi-valued attribute comprising first and last names and middle initials.
  2. Create an ERD that avoids the multi-valued attribute empDegrees in the previous example. Hint: Consider including another entity type and a relationship for keeping track of degrees.

### 7.3.5 Derived Attributes

If an attribute's value can be derived from the values of other attributes, then the attribute is derivable, and is said to be a *derived* attribute. For example, if we have an attribute for birth date then age is derivable. Derived attributes are shown with a dotted lined oval.



**Figure 7.12 Age is a derived attribute**

Sometimes an attribute of one entity type is derived from attributes from other entity types. Consider the attribute for the total of an Invoice. A value of InvTotal is derivable; it can be computed from invoice lines. Someone who implements a database and applications that access the database would need to decide whether the value of a derivable attribute should be computed when the entity is stored or updated versus computing the value (on-the-fly) when it is needed.

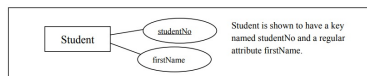
### 7.3.5 Exercises

1. Consider an educational environment where the institution tracks the performance of each student. Often this is called the students overall average, or overall grade point average. Is such an attribute a derived attribute? How is its value determined?
2. Consider a library application that needs to keep track of books that have been borrowed. Suppose there is an entity type Loan that has attributes bookID, memberID, dateBorrowed and dateDue. Suppose the due date is always 2 weeks after the borrowed date. Show Loan and its attributes in an ERD.

### 7.3.6 Key Attributes

Some attributes, or combinations of attributes, serve to identify individual entities. For instance, suppose an educational institution assigns each student a student number that is different from all other student numbers. We say the student number attribute is a *key attribute*; student numbers are *unique* and distinguish students.

In an ERD, keys are shown underlined:

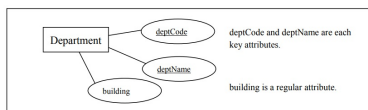


**Figure 7.13 Key attribute is underlined**

We define a key to be a minimal set of attributes that uniquely identify entities in an entity type. By minimal we mean that all of the attributes are required – none can be omitted. For instance, a typical key for an invoice line entity type would be the combination

of invoice number and invoice line number. Both attributes are required to identify a particular invoice line.

It is not unusual for an entity type to have several keys. For instance, suppose an educational institution has many departments such as Mathematics, Physics, and Computer Science. Each department is given a unique name and as well the institution assigns each one a unique code: MATH, PHYS, and CS. Both attributes would be underlined to show this in the ERD:



**Figure 7.14 Multiple key attributes**

### 7.3.6 Exercises

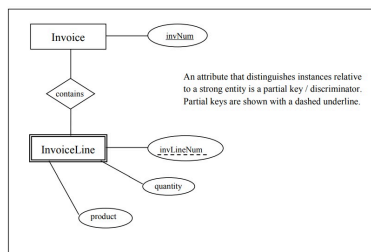
1. Suppose a company that sells products has a product entity type with the following attributes: prodNum, prodDesc, prodPrice. Suppose all three attributes are single-valued and that prodNum is a key attribute – each product has a different product number. Illustrate this information in an ERD.
2. Consider a banking application where each account is identified first by an account number and then by its type (Savings, Checking, and Loan). This scheme allows the customer to remember just one number instead of three, and then to pick a specific account by its type. Other attributes to be considered are the date the account was opened and the account's current balance. Draw an ERD for the entity type Account with the attributes account number, account type, date opened, current balance. What is the key of the entity type? Is there an attribute that is likely a derived attribute?

Show these attributes appropriately in the ERD.

### 7.3.7 Partial Key

Sometimes we have attributes that distinguish entities of an entity type from other entities of the same type, but only relative to some other related entity. This situation arises naturally when we model things like invoices and invoice lines. If invoice lines are assigned line numbers (1, 2, 3, etc.), these line numbers distinguish lines on a single invoice from other lines of the same invoice. However, for any given line number value, there could be many invoice lines (from separate invoices) with that same line number.

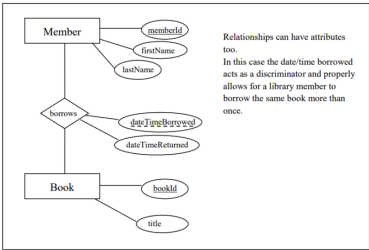
A *partial key* (also called a *discriminator*) is an attribute that distinguishes instances of a weak entity type relative to a strong entity. Invoice line number is a partial key for invoice lines; each line on one invoice will have different line numbers. Using the Peter Chen notation the discriminator attribute is underlined with a dashed line:



**Figure 7.15 Line number distinguishes lines on the same invoice**

Later when relationships are covered, it will be clearer that attributes for relationships can be discriminators too. Consider that a library has books that members will borrow. Any book could be

borrowed many times and even by the same member. However when a member borrows the same book more than once the date/time will distinguish those events. Consider the following ERD for this case:



**Figure 7.16 Relationship attribute as a discriminator**

### 7.3.7 Exercises

1. Consider an educational institution that has departments and where each department offers courses. Suppose departments are assigned unique identifiers and so deptCode is a key for the department entity type. Courses are identified within a department by a course number; course numbers are unique within a department but not across departments. So, History may have a course numbered 215, and English could have a course numbered 215 too. In order to identify a particular course we need to know the department and we need to know the course number. Illustrate an ERD including department and course entity types. Include attributes for the Department (department code and department name), and for Course (course number, title, and description).
2. Consider a company that owns and operates parking lots.

Develop an ERD with two entity types Parking Lot and Space and where:

- The address of a parking lot serves to identify the lot.
- Each space within a lot is rented at the same monthly rental charge.
- Each parking space is known by its number within the lot (within a lot these always start at 1).
- Each parking space is rented out to at most one vehicle. The vehicle's identifier must be recorded. The identifier comprises a province code and license plate number.

### 7.3.8 Surrogate Key

When a key specified for an entity is meaningless to the entity and to end-users (it doesn't describe any characteristic of an entity), the key is referred to as a *surrogate* key. A key that is not a surrogate key is often referred to as a *natural* key. Often a surrogate key is just a simple integer value assigned by the database system.

When database designs are implemented surrogate keys can be useful to simplify references from one table to another (referential integrity) and the associated joins when tables are referenced in queries.

### 7.3.8 Exercise

1. Assuming you have experience with some database system, what data type would you use for surrogate keys?

## 7.3.9 Non-Key Attributes

Non-key attributes are attributes that are not part of any key. Generally, most attributes are simply descriptive, and fall into this category. Determining key and non-key attributes is an important modeling exercise, one that requires careful consideration. Consider an Employee entity type that has attributes for first name, last name, birth date; these attributes would serve to describe an employee but would not serve to uniquely identify employees.

People may join an organization and their name is not likely unique for the organization; we expect many people in a large organization to have the same first name, same last name, and even the same combination of first and last name. Names cannot usually be used as a key.

However, names chosen for entities such as departments in an organization could be keys because of the way the company would choose department names – they wouldn't give two different departments the same name.

## 7.3.9 Exercises

1. Consider an employee entity type and its attributes, and decide which attributes are key attributes and which ones are non-key attributes. Illustrate with an ERD.
2. A birthdate attribute would appear for many entity types – for example students, employees, children. What is a birthdate likely to be: key or non-key?
3. Consider a library and the fact that books are loaned out to library members. Dates could be used heavily for the date a book was borrowed, the date the book was returned, and the due date for a book. Consider an entity type Loan that has attributes book identifier, member identifier, date borrowed,

date due, date returned. What combination of attributes would be a key? Which attributes are key attributes? Which attributes are non-key attributes?

### 7.3.10 Nulls

When a database design is implemented, one of the important things to know for each attribute of an entity type is whether or not that attribute must have a value. For example, when a book is borrowed from a library, the date the book is borrowed is known, but the returned date is not known. Sometimes you will not know the value of an attribute until a certain event occurs.

Consider an educational environment and when a student registers for a course. The date the student registers would be known, but the grade is yet to be determined.

When an entity is created but some attribute does not have a value we say it is *null*. Null represents the absence of a value; null is different from zero or from blank.

### 7.3.11 Domains

To complete the analysis for a database design, it is necessary to determine what constitutes a valid value for an attribute. A *domain* for an attribute is its set of valid values which includes a choice of datatype, but a full specification of domain is typically more than that.

For instance, analysis for student identifiers may lead one to state that a student identifier is a positive whole number of exactly 7 digits with no leading zeros. The analysis of requirements for person names may lead one to state that the values stored in a database for a first name, last name, or middle name will not be more than 50



characters in length, and that names will not have any spaces at the beginning or end.

For each attribute, one must determine its domain. More than one attribute can share the same domain. Knowing the underlying domains in your model is important. They help to complete your analysis, they are indispensable for coding programs, and they are useful for defining meaningful error messages.

Attribute domains are not usually shown in an ERD. Rather, domains are included in accompanying documentation which can be referred to when the database is being implemented.

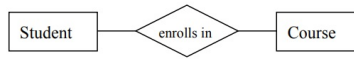
## 7.4 Relationships

Up to this point, we have made several references to the concept of relationship. Now, we will make our understanding of this concept more complete. A **relationship** is an association amongst entities. Relationships will have justification in business rules, in the way an enterprise manages its business.

There are several ways of classifying relationships, according to *degree*, *participation*, *cardinality*, whether *recursion* is involved, and whether or not a relationship is *identifying*.

### 7.4.1 Degree

We consider the *degree* as the number of entities that participate in the relationship. When we speak of a student enrolling in a course, we are considering a relationship (say, the *enroll in* relationship) where two entity types (Student and Course) are involved. This relationship is of degree 2 because each instance of the relationship will always involve one student entity and one course entity.



**Figure 7.17 Binary relationship involves two entity types**

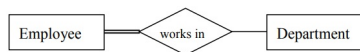
With binary relationships, there must be two defining statements we can express, one from the perspective of each entity type. In this case, our statements are:

- A student may enroll in any number of courses.
- A course may have any number of students enrolled.

Many database modeling tools only support binary relationships. However, there are situations where relationships of higher degree are useful. A relationship involving 3 entity types is called *ternary*; more generally, we refer to relationships with  $n$  entity types as *n-ary*. Our primary focus in this text is on binary relationships.

## 7.4.2 Participation

Suppose, we are designing a database for a company that has several departments and employees. Each employee must also be assigned to work in one department. We can define a *works in* relationship involving Department and Employee. Employees must participate in the relationship and we show this using a double line joining the diamond symbol to the Employee entity type.



**Figure 7.18 Employee must work in a department**

The double line stands for *total* or *mandatory* participation which means that instances of the adjacent entity type must participate in the relationship – in the case above, all instances of Employee must be assigned to some department. Any time we show a single line we are stating participation is *optional*; for the above we are saying that a department will have zero or more employees who work there.

## Cardinality

Cardinality is a constraint on a relationship specifying the number of entity instances that a specific entity may be related to via the relationship. Suppose we have the following rules for departments and employees:

- A department can have **several** employees that work in the department
- An employee is assigned to work in **one** department. From these rules, we know the cardinalities for the *works in* relationship and we express them with the cardinality symbols 1 and *n* below.



**Figure 7.19 One-to-many relationships are most common**

The *n* represents an arbitrary number of instances, and the 1 represents at most one instance. For the above works in relationship, we have

- a specific employee works in at most only one department, and
- a specific department may have many (zero or more) employees who work there.

$n$ ,  $m$ ,  $N$ , and  $M$  are common symbols used in ER diagrams for representing an arbitrary number of occurrences; however, any alphabetic character will suffice.

Based on cardinality, there are three types of binary relationships: *one-to-one*, *one-to-many*, and *many-to-many*.

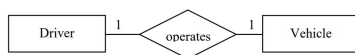
## One-to-One

One-to-one relationships have 1 specified for both cardinalities. Suppose, we have two entity types: Driver and Vehicle. Assume that we are only concerned with the current driver of a vehicle, and that we are only concerned with the current vehicle that a driver is operating. Our two rules associate an instance of one entity type with at most one instance of the other entity type:

a driver operates at most one vehicle, and

a vehicle is operated by at most one driver.

and so the relationship is one-to-one.



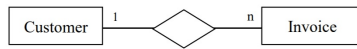
**Figure 7.20 One-to-one relationship**

## One-to-Many

One-to-many relationships are the most common ones in database

designs. Suppose, we have customer entities and invoice entities and:

- an invoice is for exactly one customer, and
- a customer could have any number (zero or more) of invoices at any point in time. Because one instance of an Invoice can only be associated with a single instance of Customer, and because one instance of Customer can be associated with any number of Invoice instances, this is a one-to-many relationship:



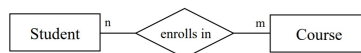
**Figure 7.21 One-to-many relationship**

## Many-to-Many

Suppose we are interested in courses and students and the fact that students register for courses. Our two rule statements are:

- any student may enroll in several courses,
- a course may be taken by several students.

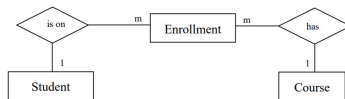
This situation is represented as a many-to-many relationship between Course and Student:



**Figure 7.22 Many-to-many relationship**

As will be discussed again later, a many-to-many relationship is implemented in a relational database in a separate relation. In a relational database for the above, there would be three relations: *one for Student, one for Course, and one for the many-to-many.* (Sometimes this 3rd relation is called an intersection table, a composite table, a bridge table.)

Partly because of the need for a separate structure when the database is implemented, many modelers will ‘resolve’ a many-to-many relationship into two one-to-many relationships as they are modeling. We can restructure the above many-to-many as *two* one-to-many relationships where we have ‘invented’ a new entity type called Enrollment:



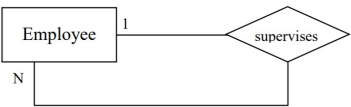
A student can have many enrollments, and each course may have many enrollments. An enrollment entity is related to one student entity and to one course entity.

**Figure 7.23 Many-to-many becomes two one-to-many relationships**

## 7.4.4 Recursive Relationships

A relationship is *recursive* if the same entity type appears more than once. A typical business example is a rule such as “an employee *supervises* other employees”. The *supervises* relationship is recursive; each instance of *supervises* will specify two employees, one of which is considered a *supervisor* and the other the *supervised*.

In the following diagram, the relationship symbol joins to the Employee entity type twice by two separate lines. Note the relationship is one-to-many: an employee may supervise many employees, and, an employee may be supervised by one other employee.

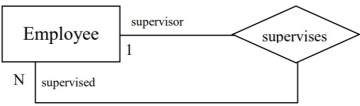


**Figure 7.24 Recursive relationship involving Employee twice**

With recursive relationships, it is appropriate to name the roles each entity type plays. Suppose we have an instance of the relationship:

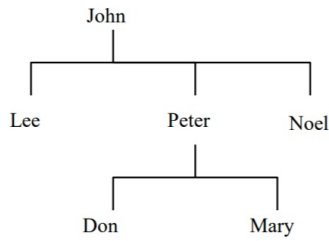
John *supervises* Terry

Then with respect to this instance, John is the *supervisor* employee and Terry is the *supervised* employee. We can show these two roles that entity types play in a relationship by placing labels on the relationship line:



**Figure 7.25 Recursive relationship with role names**

This one-to-many *supervises* relationship can be visualized as a hierarchy. In the following, we show five instances of the relationship: John supervises Lee, John supervises Peter, Peter supervises Don, Peter supervises Mary, and John supervises Noel.



**Figure 7.26 The supervising hierarchy**

In the above example, note the participation constraint at both ends of *supervises* is optional. This has to be the case because some employee will not be supervised, and, for some employees there are no employees they supervise.

Generally, recursive relationships are difficult to master. Some other situations where recursive relationships can be used:

- A person marries another person
- A person is the parent of a person
- A team plays against another team
- An organizational units report to another organizational unit
- A part is composed of other parts.

## 7.4.5 Identifying Relationships

When entity types were first introduced, we discussed an example where a department offers courses and that a course must exist in the context of a department. In that case, the Course entity type is considered a weak entity type as it is existence-dependent on Department. It is typical in such situations that the key of the strong entity type is used in the identification scheme for the weak



entity type. For example, courses could be identified as MATH-123 or PHYS-329, or as Mathematics-123 or Physics-329. In order to convey the composite identification scheme for a weak entity type, we specify the relationship as an *identifying* relationship which is visualized using a double-lined diamond symbol:



Additionally, in situations where we have an identifying relationship, we usually have

- a weak entity type with a partial key
- a weak entity type that must participate in the relationship (total participation) and so the ERD for our hypothetical educational institution could be:



**Figure 7.27 An identifying relationship**

Note the keys for the strong entity type appear only at the strong entity type. The identifying relationship tells one that a department key will be needed to complete the identification of a course.

## Exercises

1. Consider a company that owns and operates parking lots. Draw an ERD to include the following specifications. Each parking lot has a unique address (use the typical fields for addresses) and each parking lot has a certain number, say  $n$ , of parking spaces. Each space in a lot has a number between 1 and  $n$ . The cost of renting a parking space is the same for all spaces in a lot. The company rents individual spaces out to its customers. Each customer is identified by a driver's license id, has a first and last name. Each customer will identify possibly several cars that they will park in the space rented to them. For each car the company needs to know the year, make, model, color and its license plate number.
2. Modify your model from the previous question to allow for scrambled parking. By this we mean that a customer is rented a space in a lot, but the customer may park in any available space.
3. Draw an ERD involving employees and their dependents where each employee has a unique id number and where dependents of the same employee are numbered starting at 1. It may be rare, but we will allow for dependents of the same employee to have the same name and birthdates. Include typical attributes for an employee, and for a dependent include the birthdate, first and last names.
4. Draw an ERD for marriages between two people. For persons include birthdate, first name, last name, and a unique person id. Consider marriage to be a relationship between two people and suppose we want our model to allow for people to have more than one marriage. Use the date of the marriage as a discriminator.
5. Consider marriages again but now let marriage be an entity type. Suppose when people marry there is a marriage certificate that is granted by a government authority. Include

attributes applicable to a marriage.

6. Suppose we are modeling marriage as a relationship between two people. When, or under what circumstances, can we model this as a one-to-one relationship?
7. Draw an ERD that allows for marriages between possibly more than two people.
8. Consider the one-to-one *operates* relationship in this chapter. Modify the example so that drivers have attributes: driver license, name (which comprises first name and last name), and vehicles have attributes: license plate number, VIN, year, colour, make and model. Note that VIN stands for vehicle identification number and this is unique for each vehicle. Assume that each driver must be assigned to a vehicle.
9. Consider the *enroll in* relationship used in this chapter. Suppose we must allow for a student to repeat a course to improve their grade. Develop an ERD and include typical attributes for student, course, etc. We need to keep a complete history of all course attempts by students.
10. What problems arise if one makes the *supervises* relationship mandatory for either the supervising employee or the employee who is supervised?
11. Consider requirements for teams, players and games, and develop a suitable ERD. Each team would have a unique name, have a non-player who is the coach, and have several players. Each player has a first and last name and is identified by a number (1, 2, 3, etc.). One player is designated the captain of the team. Assume a game occurs on some date and time, and is played by two teams where one team is called the home team and the other team is called the visiting team. At the end of the game the score must be recorded.
12. Modify your ERD for the above to accommodate a specific sport such as curling, baseball, etc.
13. Consider an ERD for modelling customers, phones, and phone calls. Each customer owns one phone and so the phone number identifies the customer. Include other attributes such

as credit card number, first name, and last name for a customer. We must record information for each phone call that is made: for each call there is a start time, end time, and of course the phone number/customers involved.

14. Create an ERD suitable for a database that will keep genealogy data. Suppose there is one entity type Person and you must model the two relationships: *marries* and *child of*.
15. Develop an ERD to support home real estate sales. Consider there are several sales employees who list and sell properties. For each employee we need to know their name (first and last), the date they started working for this company, and the number of years they have been with the company. Each property has owners (one or more people), and may have certain features such as number of baths, number of levels, number of bedrooms. For each owner we must keep track of their names (first and last). Each property has an address; each address has the usual attributes: street (comprising apartment number, street number, street name), city, province, and postal code. A home is listed at a certain price and sold at possibly a different price. Of course, we need to track the names of the buyers, the date of a listing and the date of a sale.
16. Develop an ERD to keep track of information for an educational institution. Assume each course is taught by one instructor, and an instructor could teach several courses. For each instructor suppose we have a unique identifier, a first name, a last name, and a gender. Each course belongs to exactly one department. Within a department courses are identified by a course number. Departments are identified by a department code.
17. Develop an ERD to allow us to keep information on a survey. Suppose a survey will have several questions that can be answered true or false. Over a period of time the survey is conducted and there will be several responses.
18. Modify the ERD above to allow for surveys that have multiple choice answers.

19. Develop an ERD to support the management of credit cards. Each credit card has a unique number and has a customer associated with it. A customer may have several credit cards. The customer has a first name, last name, and an address. Each time a customer uses a credit card we must record the time, the date, the vendor, and the amount of money involved.
20. Modify the ERD for the above to accommodate the monthly billing of customers. Each month a customer receives a statement detailing the activity that month.
21. Develop an ERD to be used by a company to manage the orders it receives from its customers. Each customer is identified uniquely by a customer id; include the first name, last name, and address for each customer. The company has several products that it stocks and for which customers place orders. Each product has a unique id, unique name, unit price, and a quantity on hand. At any time a customer may place an order which will involve possibly many products. For each product ordered the database must know the quantity ordered and the unit price at that point in time. If the customer does this through a phone call then an employee is involved in the call and will be responsible for the order from the company side. Some orders are placed via the internet. For each order an order number is generated. For each order the database must keep track of the order number, the date the order was placed and the date by which the customer needs to receive the goods.

# 8. Mapping an ERD to a Relational Database

RON MCFADYEN

We use an Entity Relationship Diagram to represent the informational needs of a system. When we are convinced it is satisfactory, we map the Entity Relationship Diagram (ERD) to a relational database and implement it as a physical database. In general, relations are used to hold entity sets and to hold relationship sets. The considerations to be made are listed below. After we present the mapping rules, we illustrate their application in a few examples.

## 8.1 MAPPING RULES

To complete the mapping from an Entity Relationship Diagram (ERD) to relations, we must consider the entity types, relationship types, and attributes that are specified for the model.

### Entity Types

Each entity type is implemented with a separate relation. Entity types are either strong entity types or weak entity types.

#### 1. Strong Entities

Strong, or regular, entity types are mapped to their

own relation. The primary key (PK) is chosen from the set of keys available.

## 2. Weak Entities

Weak entity types are mapped to their own relation, but the primary key of the relation is formed as follows. If there are any identifying relationships, then the PK of the weak entity is the combination of the PKs of entities related through identifying relationships and the discriminator of the weak entity type. Otherwise, the PK of the relation is the PK of the weak entity.

# Relationship Types

The implementation of relationships involves foreign keys. Recall, as discussed in point 1) above. If the relationship is identifying, then the primary key of an entity type must be propagated to the relation for a weak entity type. We must consider both the degree and the cardinality of the relationship. In the following examples. examples 1 – 3 deal with binary relationships and example 4 concerns  $n$ -ary relationships.

## 1. Binary One-To-One

In general, with a one-to-one relationship, a designer has a choice regarding where to implement the relationship. One may choose to place a foreign key in one of the two relations, or in both. Consider placing the foreign key such that nulls are minimized. If there are attributes on the relationship, those can be placed in either relation.

## 2. Binary One-To-Many

With a one-to-many relationship, the designer must place a foreign key in the relation corresponding to the 'many' side of the relationship. Any other attributes defined for the relationship are also included on the 'many' side.

## 3. Binary Many-To-Many

A many-to-many relationship must be implemented with a separate relation for the relationship. This new relation will have a composite primary key comprising the primary keys of the participating entity types and any discriminator attribute, plus other attributes of the relationship if any.

## 4. $n$ -ary, $n > 2$

A new relation is generated for an  $n$ -ary relationship. This new relation has a composite primary key comprising the  $n$  primary keys of the participating entity types and any discriminator attribute, plus any other attributes. There is one exception to the formation of the PK: if the cardinality related for any entity type is 1, then the primary key of that entity type is only included as a foreign key and not as part of the primary key of the new relation.

# Attributes

All attributes, with the exception of derived and composite attributes, must appear in relations. You choose to include derived attributes if their presence will improve performance. In the following we consider attributes



according to whether they are simple, atomic, multi-valued, or composite.

4. Simple, atomic

These are included in the relation created for the pertinent entity type, many-to-many relationship, or  $n$ -ary relationship.

5. Multi-valued

Each multi-valued attribute is implemented using a new relation. This relation will include the primary key of the original entity type. The primary key of the new relation will be the primary key of the original entity type plus the multi-valued attribute. Note that in this new relation, the attribute is no longer multi-valued.

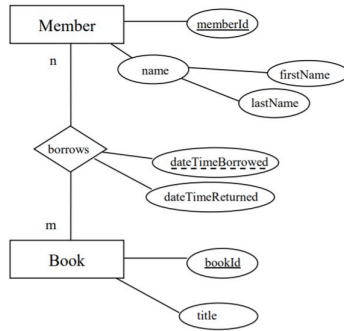
6. Composite and Derived attributes are not included.

The above constitutes the standard rules for mapping an ERD to relations. A designer may make other choices but one expects there would be good reasons for doing so.

## 8.2 Examples

### Example 1

Consider the ERD



The mapping rules lead to the relations:

Book	
<u>bookId</u>	title

Member		
<u>memberId</u>	firstName	lastName

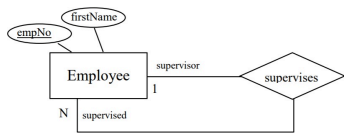
Borrow			
<u>memberId</u>	<u>bookId</u>	<u>dateTimeBorrowed</u>	dateTimeReturned

Notes:

- The Member relation does not have a composite attribute *name*.
- Since Borrows is a many-to-many relationship the Borrow relation is defined with a composite primary key {*memberId*, *bookId*, *dateTimeBorrowed*}.
- *memberId* in the Borrow relation is a foreign key referencing Member.
- *bookId* in the Borrow relation is a foreign key referencing Book.

# Example 2

Consider the ERD



The mapping rules lead to the relation:

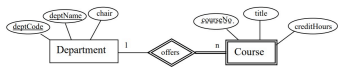
Employee		
<u>empNo</u>	firstName	supervisor

Notes:

- The attribute *supervisor* is a foreign key referencing Employee.
- A foreign key is placed on the ‘many’ side of a relationship and so in this case the foreign key references the employee who is the supervisor (the role name on the ‘one’ side); hence the name *supervisor* was chosen as the attribute name.

# Example 3

Consider the ERD



The mapping rules lead to the relations.

Department		
<u>deptCode</u>	deptName	chair

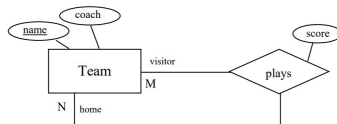
Course			
<u>deptCode</u>	<u>courseNo</u>	title	creditHours

Notes:

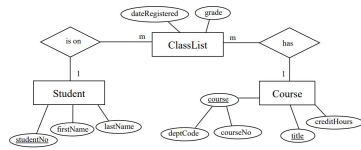
- *deptCode* was chosen as the primary key of Department.
- *deptName* is a key and so a unique index can be defined to ensure uniqueness.
- Since Course is a weak entity type and is involved in an identifying relationship, the primary key of Course is composite comprising {*deptCode*, *courseNo*}.
- *deptCode* in Course is a foreign key referencing Department.

## Exercises

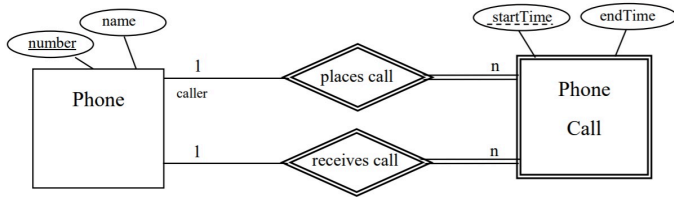
1. Map the ERD to relations.



2. Map the ERD to relations.



3. Map the ERD to relations.



# 9. Data Definition Language (DDL)

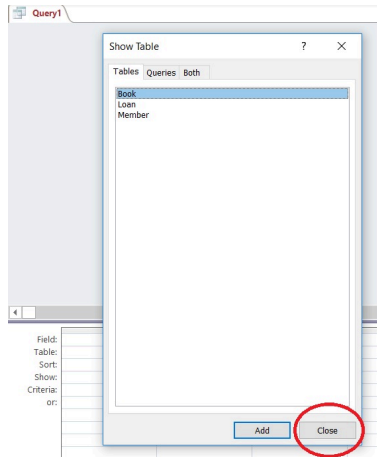
RON MCFADYEN

Many of the tools available for constructing entity relational diagrams (ERDs) are capable of generating data definition language (DDL) commands that are used for creating tables, indexes, and relationships. You can find many references easily to DDL. For instance, if you are interested try [http://en.wikipedia.org/wiki/Data\\_Definition\\_Language](http://en.wikipedia.org/wiki/Data_Definition_Language), or enter the phrase *Data Definition Language* in your favorite search engine.

## 9.1 Running DDL In Microsoft Access

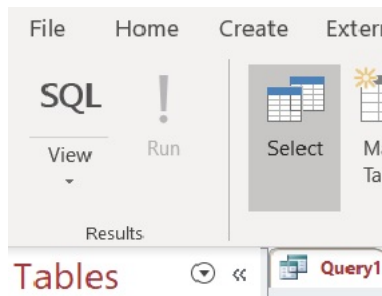
Most database systems provide a way for you to run data definition language commands. When such facility exists, it can be relatively easy to create and re-create databases from a file of DDL commands. One way to run DDL commands in Microsoft Access is through a query that is in SQL View. To run a DDL command, we follow these two steps:

- Open a database and choose to create a query, and then instead of adding tables to your query, you just close the Show Table window:



**Figure 9.1 Close the Show Table window with no tables selected**

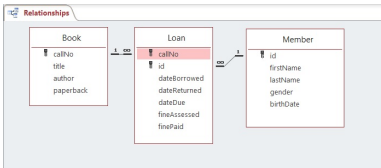
- Then, choose SQL View and you will be able to type a DDL command or paste one in :



**Figure 9.2 Choose SQL view for the query**

# 9.2 Example

In this chapter, we will creating tables and modifying tables for a new **Library-DDL** database using DDL. In Access, create a new blank database named **Library-DDL** database to apply your DDL skills. Suppose we require the three tables: Book, Patron, Borrow:



**Figure 9.3 Sample database to create**

The above diagram (produced from the Relationships Tool) represents the database *we wish to create* but where we will do so using DDL commands.

## 9.2.1 DDL Commands

We will illustrate three DDL commands (create table, alter table, create index) as we create tables and modify tables using the Library database.

CREATE TABLE	Create the Book table Create the Patron table with a primary key Create the Borrow table with a primary key and a foreign key referencing Patron
ALTER TABLE	Alter the Book table so it has a primary key Alter the Borrow table with a foreign key referencing Book Add an attribute named gender to Patron
CREATE INDEX	Create an index
DROP TABLE and DROP INDEX	Remove a table or index from the database



### Figure 9.4 Data Definition Commands

In some database environments, we can run more than one command at a time. The commands would be located in a file and would be submitted as a batch to be executed.

Before applying these DDL commands, verify that you have created a new blank Access database named **Library-DDL**. In the following, we will demonstrate SQL syntax commands supporting Microsoft Access and run one command at a time.

## 9.2.2 Creating and Modifying Database Tables

### Example 1

Consider the following create table command which is used to create a table named Book. The table has two fields: callNo and title.

```
CREATE TABLE Book
(
    callNo Text(50),
    title Text(100)
)
;
```

The command begins with the keywords CREATE TABLE. It's usual for keywords in DDL to be written in upper case, but it's not required to do so. The command is just text that is parsed and executed by a command processor. If humans are expected to read the DDL then the command is typically written on several lines as shown, one part per line.

## Example 2

Now consider the following CREATE TABLE command which creates a table and establishes an attribute as the primary key:

```
CREATE TABLE Patron
(
    PatronID Number NOT NULL PRIMARY KEY,
    lastName Text(50),
    firstName Text(50)
);
```

## Example 3

The primary key of Patron is the patronId field. Notice the data type is shown as Counter. After running this command you will be able to see that the Counter data type is transformed to AutoNumber.

Our last example of the create table command is one that creates a table, sets its primary key and also creates a foreign key reference to another table:

```
CREATE TABLE Borrow
(
    patronId Number,
    callNo Text(50),
    dateDue DATETIME,
    returned YESNO,
    PRIMARY KEY (patronId, callNo, dateDue), FOREIGN KEY
    (patronId) REFERENCES Patron
)
;
```

There are several things to notice in the above command:

- The primary key is composite and so it is defined in a separate PRIMARY KEY clause.

- The data type of patron id must match the data type used in the Patron table and so the data type is defined as Integer.
- The dateDue field will hold a due date and so its data type is defined as DATE/TIME.
- The returned field will hold a value to indicate whether or not a book has been returned and so its data type is defined as YES/NO.
- A row in the Borrow table must refer to an existing row in Patron and so we establish a relationship between Borrow and Patron using the FOREIGN KEY clause. After running this create table command you can see the relationship in Access by opening the Relationships Tool.

## Example 4

The Book table was created previously, but there is no specification for a primary key. To add a primary key, we use the ALTER TABLE command as shown below.

```
ALTER TABLE Book
ADD PRIMARY KEY (callNo)
;
```

## Example 5

Now that Book has a primary key, we can define the relationship that should exist between Borrow and Book. To do so, we use the ALTER TABLE command again:

```
ALTER TABLE Borrow
ADD FOREIGN KEY (callNo)
REFERENCES Book (callNo)
;
```

## Example 6

Notice that the Patron table does not have a gender attribute. To add this later on, we can use the ALTER TABLE command:

```
ALTER TABLE Patron ADD  
COLUMN gender Text(6)  
;
```

## Example 7

For performance reasons, we can add indexes to a table. DDL provides CREATE INDEX and DROP INDEX commands for managing these structures. To create an index for Patron on the combination last name and first name, we can execute:

```
CREATE INDEX PatronNameIndex ON Patron (LastName,  
FirstName);
```

## Example 8

To remove the above index, we need to identify the index by name:

```
DROP INDEX PatronNameIndex ON Patron;
```

## Example 9

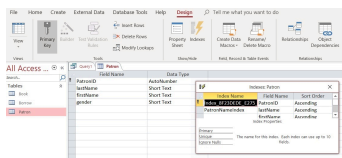
To remove a table, we use the DROP TABLE command.

```
DROP TABLE Person;
```

# DDL Exercises

Complete the following exercises using the Library database.

1. Try running the commands in Examples 1 through 3. After running each DDL statement, open the corresponding table in Design View and verify that the statement worked as intended.
2. Try running the commands in Examples 4 through 6. After running each DDL statement, open the corresponding table in Design View and verify that the statement worked as intended.
3. The effect of executing the commands in the first 6 examples can be accomplished by 3 create table commands. Example 9 shows a DROP TABLE command; use similar DROP commands to delete all the tables you created in Exercises 1 and 2. Now, write 3 create table commands that have the same effect as Examples 1 through 6. After running the DDL statements, open the Relationships Tool to verify your commands created the 3 tables and the 2 relationships.
4. Example 7 creates an index. Run this command in your database and then verify the index has been created. You can view index information by clicking the Indexes icon:



Notice that the (primary) index has a name that was generated by Microsoft Access.

5. Consider an ERD from the previous chapter. Write the DDL that could create the required relations.

# 10. Normalization

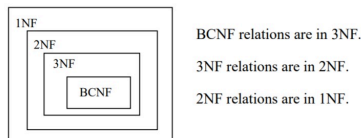
RON MCFADYEN

The theory of normal forms is concerned with the structure of relations in a relational database. There are several normal forms of which 1NF, 2NF, 3NF and Boyce-Codd (BCNF) are the most important for practical online transaction processing (OLTP) database design. Online transaction processing (OLTP) systems are used to run the day-to-day events of a business.

Normalization theory gives us a theoretical basis to judge the quality of a database and helps one understand the impact of some design decisions. In practice, Entity Relationship Modeling is the primary technique used for designing databases and experienced practitioners will typically develop BCNF relations as a result. Normalization can be applied by the practitioner to understand better the semantics behind some relations and possibly make some design modifications.

## Boyce-Codd (1NF – 3NF)

1NF, 2NF, 3NF and BCNF are acronyms for first normal form, second normal form, third normal form, and Boyce-Codd normal forms. There is a sequence to normal forms: 1NF is considered the weakest, 2NF is stronger than 1NF, 3NF is stronger than 2NF, and BCNF is considered the strongest of these four normal forms. Also, any relation that is in BCNF, is in 3NF; any relation in 3NF is in 2NF; and any relation in 2NF is in 1NF. This correspondence can be shown as:



Transactions are *units of work* designed to meet the goals of users. For instance in a banking environment, we would expect to find a deposit transaction, a withdrawal transaction, a transfer transaction, and a balance lookup transaction. A unit of work is a collection of database operations that are executed in their entirety or not at all. For example, if you are transferring money from one account to another, it's important for the integrity of accounts that the transfer be completely done, and never partly done. If a transfer transaction is partly done (say, because of a system failure) then accounts would be out of balance. A database environment has capabilities to back out partly executed transactions so the system can be back where it was prior to a failed transfer transaction. A banking system could have thousands of users and we expect transactions such as these to be correctly and efficiently executed. A normalized database is such that every relation is in at least 1NF, and preferably 3NF. Generally speaking, normalized databases lead to the most efficient designs for these types of transactions.

## Normalization

Normalization is a process that replaces a relation with other relations of a higher normal form. The process involves decomposing a relation into other relations in such a way as to preserve the original information and reduce redundancy of data. Reducing redundant data increases the number of relations, but makes the data easier to maintain. Later, we will provide examples of decomposition.

We say normalization is a process that *improves* a database design. The objective of normalization is sometimes stated: *to create relations where every dependency is on the key, the whole key, and*

*nothing but the key*<sup>1</sup>. A relation that is fully normalized is about a single concept such as a student entity type, a course entity type, and so on.

De-normalization is a process that changes relations from higher to lower normal forms, and hence generates redundant data in the *tuples* (rows/records) of a relation (table). If deemed necessary, this would be done to improve the performance (reduce the cost) of retrieving information from the database. The cost of querying de-normalized relations is generally less because fewer joins are required.

We consider higher normal forms to be better choices because the update semantics for data are simplified. By this, we mean that applications required to maintain the database are simpler to code and so they are easier to maintain. In the following, we discuss:

- Functional Dependencies
- Update Anomalies
- Partial Dependencies
- Transitive Dependencies
- Normal Forms

## 10.1 Functional Dependencies

To understand normalization theory (first, second, third and Boyce-Codd normal forms), we must understand what is meant by the term *functional dependency*. There is another type of dependency called a multi-valued dependency, but that is important to the understanding of higher normal forms not covered in this text. A

1. Kent, William. "A Simple Guide to Five Normal Forms in Relational Database Theory", Communications of the ACM 26 (2), Feb. 1983, pp. 120–125.



functional dependency is an association between two attributes. We say there is a **functional dependency** from attribute A to an attribute B if and only if for each value of A there can be at most one value for B. We can illustrate this by writing

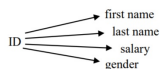
- A functionally determines B, or
- B is functionally determined by A, or
- by a drawing such as:  $A \longrightarrow B$

When we have a functional dependency from A to B we refer to attribute A as the **determinant**.

## Example 1

Consider a company collecting information about each employee such as the employee's identification number (ID), their first name, last name, salary and gender. As is typical, each employee is given a unique ID which serves to identify the employee. Hence for each value of ID, there is at most one value for first name, last name, salary and gender. Therefore, we have four functional dependencies where ID is the determinant; we can show this as a list or graphically:

ID  $\rightarrow$  first name  
ID  $\rightarrow$  last name  
ID  $\rightarrow$  salary  
ID  $\rightarrow$  gender



If you think about this case, there cannot be any other functional dependencies (FDs). For example, consider the gender attribute – we need to allow for more than one employee for a given gender, and so we cannot have a situation where gender functionally determines ID. So, gender  $\rightarrow$  ID cannot exist. Now consider the first name attribute. Again, we need to allow for more than one

employee to have the same first name and so first name cannot determine anything. Similarly for other attributes.

## Example 2

Recall the Department and Course tables introduced in Chapter 2 – sample data is shown below:

Department				
deptCode	deptName	deptLocn	deptPhone	chairName
ENGL	English	3D05	786-9999	April Jones
MATH	Mathematics	2R33	786-0033	Peter Smith
ACS	Applied Computer Science	3D07	786-0300	Simon Lee
PHIL	Philosophy	3C11	786-3322	Judy Chan
BIOL	Biology	2L88	786-9843	James Dunn

**Figure 2.1 Department table**

Course				
deptCode	courseNo	title	description	creditHours
ACS	1453	Introduction to Computers	This course will introduce students to the basic concepts of computers: types of computers, hardware, software, and types of application systems.	3
ACS	1803	Introduction to Information Systems	This course examines applications of information technology to businesses and other organizations.	3
ENGL	2221	The Age of Chaucer	This course examines a selection of medieval poetry and drama with emphasis upon Chaucer's Canterbury Tales.	6

PHIL	2219	Philosophy of Art	Through reading key theorists in the history of esthetics, this course examines some of the fundamental problems in the philosophy of art, including those of the definition and purpose of art, the nature of beauty, the sources of genius and originality, the problem of forgery, and the possible connection between art and the moral good	3
BIOL	4451	Forest Ecosystems Field Course	This is an intensive three-week field course designed to give students a comprehensive overview of forest ecology field skills.	2
BIOL	4931	Immunology	Immunology is the study of the defense system which the body has evolved to protect itself from external threats such as viruses and internal threats such as tumor cells.	3

---

**Figure 2.2 Course table**

Recall the primary keys (underlined above) of these two tables:

Table	Primary Key
Department	<u>deptCode</u>
Course	<u>deptCode</u> , <u>courseNo</u>

Consider the Department table where deptCode is the primary key. For each value of deptCode, there is at most one value for deptName, deptLocn, deptPhone, and chairName. You should agree the following functional dependencies exist:

deptCode  $\rightarrow$  deptName  
deptCode  $\rightarrow$  deptLocn  
deptCode  $\rightarrow$  deptPhone  
deptCode  $\rightarrow$  chairName

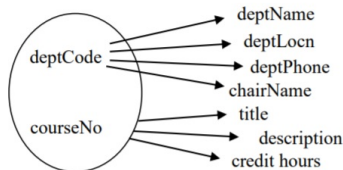
Each row of the Course table has one value for title, one value for description, and one value for credit hours. The primary key of Course is consists of two attributes, deptCode and courseNo.

The following functional dependencies exist for the Course table:

deptCode, courseNo  $\rightarrow$  title  
deptCode, courseNo  $\rightarrow$  description  
deptCode, courseNo  $\rightarrow$  credit hours

In this case, we have a determinant comprising two attributes; the determinant is composite.

We can draw the functional dependencies as:

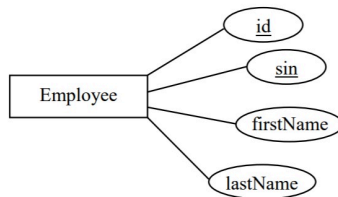


*Could there be other functional dependencies in this situation?*

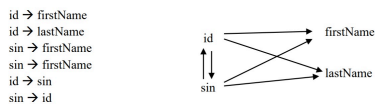
These examples demonstrate that there is a functional dependency from the primary key to each of the other attributes in a table.

### Example 3.

The following entity relational diagram (ERD) is shown in the Chen notation. There is one entity type named Employee that has 4 attributes. In this design, there are two keys (*id* and *sin*) and two descriptive attributes (*firstName* and *lastName*)



Each symbol in an ER diagram contains information about a model. From the above, we know there are two keys – *id* and *sin*. An *id* value, or a *sin* value, will uniquely identify an employee and so we have the six functional dependencies (FDs):



This example shows that an ER Diagram carries information that can be expressed in terms of functional dependencies.

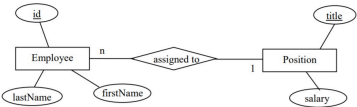
# Exercises

1. Consider the Product table below where productID is the PK. What FDs must exist in this table:

productID	description	unit price	quantity on hand
33	16 oz. can tomato soup	1.00	50
41	454 gram box corn flakes	4.50	39
45	Package red licorice	1.00	39
46	Package black licorice	1.00	50
47	1 litre 1% milk	1.99	25

2. Consider the ERD where the entity type *Employee* has one key attribute, *id*, and the entity type *Position* has one key attribute, *title*. As well the ERD shows a one-to-many relationship *assigned to* which can be expressed as:

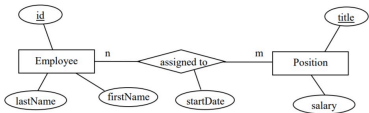
An employee is assigned to at most one position.  
A position can be assigned to many employees.



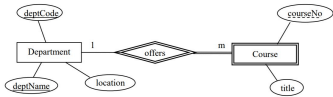
List the FDs that must be present.

3. Consider this ERD that is similar to the above but where the *assigned to* relationship is many-to-many, and

where *assigned to* has an attribute *startDate*. List the FDs that are present.



4. Consider the ERD below where *Department* has two keys *deptCode* and *deptName* – each department has a unique department code and has a unique department name. *Course* is a weak entity type with a partial key *courseNo*, and where *offers* is an identifying relationship.



List the FDs that must exist.

5. Consider the table with attributes A, B and C.



A	B	C
1	33	100
2	33	200
3	22	200
1	33	101
2	33	350
4	67	350
5	67	101

Suppose there are many more rows that are not shown.

a) Is there a functional dependency from B to A? Explain your answer.

b) The rows that are shown suggest there could be a functional dependency  $A \rightarrow B$ . Compose a database query that would list rows, if they exist, that are counterexamples to the functional dependency  $A \rightarrow B$ . Such a query would list rows in the table where two or more rows have the same value for A but different values for B.

## 10.1.2 Keys and Non-Keys

Before going further, we need to be clear regarding the concept of key. We define the **key** of a relation to be any minimal set of

attributes that uniquely identify tuples in the relation. We say minimal in order to eliminate trivial cases. Consider: If attribute  $k$  is a key and uniquely identifies a tuple then any combination of attributes that include  $k$  must also uniquely identify tuples. So, we restrict keys to be **minimal sets of attributes** that retain the property of unique identification. Further, we define **candidate keys** to be the collection of keys for a relation; a database designer must choose one of the candidate keys to be the primary key.

Additionally, we define **key attributes** to be those attributes that are part of a key, and **non-key attributes** are those attributes that are not part of any key.

### 10.1.3 Anomalies

An anomaly is a variation that differs in some way from what is considered normal. With regards to maintaining a database, we consider the actions that must occur when data is updated, inserted, or deleted. In database applications where these update, insert, and/or delete operations are common (e.g. OLTP databases), it is desirable for these operations to be as simple and efficient as possible.

When relations are not fully normalized, they exhibit update anomalies because basic operations are not as simple as possible. When relations are not fully normalized, some aspect of the relation will be awkward to maintain.

Consider the relation structure and sample data:

deptNum	courseNum	studNum	grade	studName
92	101	3344	A	Joe
92	115	7654	A	Brenda
81	101	7654	C	Brenda
92	226	3344	B	Joe

This relation is used for keeping track of students enrollments, the grade assigned, and (oddly) the student's name.

What must happen if a student's name were to change? We should want our databases to have correct information, and so the name may need to be changed in several records, not just one. This is an example of an *update anomaly* – the simple change of a student's name affects, not just one record, but potentially several in the database. The update operation is more complex than necessary, and this means it is more expensive to do, resulting in slower performance. When operations become more complex than necessary, there is also a chance the operation is programmed incorrectly resulting in corrupted data – another unfortunate consequence.

Consider the Course and Department tables again, but now consider that they are combined into a single table. Obviously, this is a table with a considerable redundancy – for each course in the same department, the department location, phone, and chair must be repeated.

Department_Course								
dept Code	dept Name	dept Location	dept Phone	chair Name	course No	title	description	credit Hours

The primary key of such a table must be {deptCode, courseNo}.

Consider for the following, however unlikely the situation seems, that the `Department_Course` table is the only table where department information is kept. Note that our point here is only to show, for a simple example, how redundancy leads to difficult semantics for database operations.

## Insert anomaly

Suppose the University added a new department but there are no courses for that department yet. How can a row be added to the above table? Recall that no part of a primary key can be null, and so we can't insert a row for a new department because we do not have a value for `courseNo`. This is an example of an insertion anomaly.

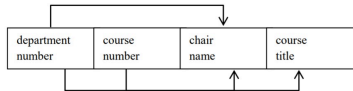
## Delete anomaly

Suppose some department is undergoing a major reorganization. All courses are to be removed and later on some new courses will be added. If we delete all courses then we lose all the information in the database for that department.

The previous discussion concerning anomalies highlights some of the data management issues that arise when a relation is not fully normalized. Another way of describing the general problem here, as far as updating a database is concerned, is that redundant data makes it more complicated for us to keep the data consistent.

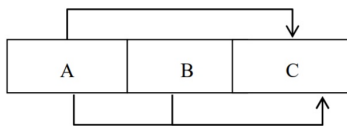
## 10.1.4 Partial Functional Dependencies

Consider a relation with department number, department chair name, course number and course title attributes. The combination {department number, course number} must be a key. The directed lines depict the FDs that are present.



Note the functional dependency of chair name on department number. If two or more rows in the relation have the same value for department number, they must have the same value for chair name. We say this redundancy is due to the FD of chair name on department number. Because chair name is a non-key attribute and is dependent on department number, a subset of a key, we call this dependency a **partial dependency**.

In general, if we have a composite key {A, B} and the dependencies below



we say that C is partially dependent on {A, B}.

## Exercises

1. Suppose each delivery of a course is called a section. In any one term, a course may have multiple sections and each section is assigned an instructor. Each course has a course title. Consider a Section relation where the PK is {dept number, course number, section number}. What FDs exist? Is there a partial dependency?

deptNo	courseNo	sectionNo	instructor	title
91	1906	001	J. Smith	Java I
91	1906	002	D. Grand	Java I
91	1910	001	J. Smith	Java II
91	1910	002	J. Daniels	Java II
53	1906	001	S. Farrell	History of the World
...	...	...	...	...

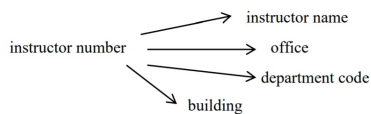
2. Consider a relation with attributes X, Y, Z, W where the only CK is {X,Y}, and where the FDs are  $\{X,Y\} \rightarrow Z$ ,  $\{X,Y\} \rightarrow W$ , and  $Y \rightarrow W$ . Is there a partial dependency?

### 10.1.6 Transitive Functional Dependencies

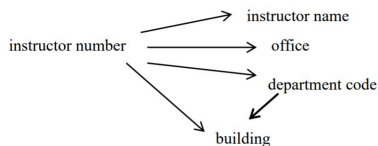
Consider a relation that describes a couple of concepts, say instructor and department, and where the building shown is the building where the department is located, and the attribute instructor number is the only key:

instructor number	instructor name	office	department code	building
33	Joe	3D15	B&A	Buhler
44	Joe	3D16	ACS	Duckworth
45	April	3D17	ACS	Duckworth
50	Susan	3D17	ACS	Duckworth
21	Peter	3D18	B&A	Buhler
22	Peter	3D18	MATH	Duckworth

As instructor number is the only key, we have the following FDs:



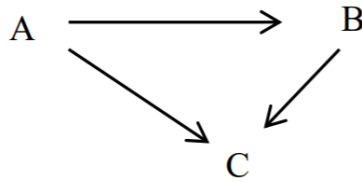
Suppose we also have the FD: department code determines building. Now our FD diagram becomes:



and we say the FD from instructor number to building is **transitive** via department code.

In general, if we have a relation with key A and functional

dependencies:  $A \rightarrow B$  and  $B \rightarrow C$ , then we say attribute A **transitively** determines attribute C.



**Figure 10.9 Non-key attributes and a transitive dependency**

**Note:** B and C above are non-key attributes. If we also had the functional dependency  $B \rightarrow A$  (and so A and B are candidate keys) then A does **not** transitively determine C.

## Exercises

1. Consider a relation that describes an employee including the province where the employee was born. Suppose the only key is employeeId and we have the attributes: name, birthDate, birthProvince, currentPopulation.



Employee				
employeeId	name	birthDate	birthProvince	currentPopulation
123	Joe	Jan 1, 1990	MB	1,200,000
222	Jennifer	Jan 5, 1988	SK	1,450,000
345	Jimmy	Feb 5, 1987	MB	1,200,000
...	...	...	...	...

What FDs would exist? Is there a transitive dependency?

- Consider a relation with attributes X, Y, Z, W where the only CK is X, and the FDs are  $X \rightarrow Y$ ,  $X \rightarrow Z$ ,  $X \rightarrow W$  and  $Y \rightarrow Z$ . Is there a transitive dependency?

## Normal Forms

The normal forms usually of interest to the database designer are 1NF, 2NF, 3NF and BCNF. There are more (higher) normal forms that we leave to follow-up courses. We discuss 1NF and BCNF; 2NF and 3NF are mentioned in our summary. 1NF is so important, it is actually a property of a relation; that is, to say something is a relation means that it is at least in 1NF. BCNF has a simple definition (compared to 2NF and 3NF) and is the usual objective of the designer.

If you understand 1NF and BCNF then you have good insight into the nature of relations that are easy to understand and maintain.

If you understand why a relation is not BCNF then you will know the source of its redundant data which is necessary in order to know how to properly maintain the data contained in the relation. In most practical cases when a relation is not BCNF, the reason will be related to partial or transitive dependencies. 2NF relations do not have partial dependencies, and 3NF relations do not have partial nor transitive dependencies.

## 10.2 FIRST NORMAL FORM (1NF)

We say a relation is in **1NF** if all values stored in the relation are **single-valued** and **atomic**. With this rule, we are simplifying the structure of a relation and the kinds of values that are stored in the relation.

### Example 1

Consider the following EmployeeDegrees relation.

- empNo is the PK
- Each employee has one first name and one salary
- Each employee has zero or more university degrees ... stored as a single attribute

EmployeeDegrees			
empNo	first name	salary	degrees
111	Joe	29,000	BSc, MSc
200	April	41,000	BA, MA
205	Peter	33,000	BEng
210	Joe	20,000	

This relation is **not** in 1NF because the degrees attribute can have multiple values. Below are two relations formed by splitting EmployeeDegrees into two relations – one relation has attributes empNo, first name, and salary and the other has empNo and degree. We say we have *decomposed* EmployeeDegrees into two relations and we have populated each with data from EmployeeDegrees. Each of these is in 1NF, and if we join them on empNo we can get back the information shown in the relation above.

Employee		
empNo	first name	salary
111	Joe	29,000
200	April	41,000
205	Peter	33,000
210	Joe	20,000

empNo is the PK.  
each employee has one name  
and one salary.

Degree	
empNo	degree
111	BSc
111	MSc
200	BA
200	MA
205	BEng

{empNo, degree} is the PK.  
degree is single-valued.

## Example 2

Consider the Student relation below. The name attribute comprises both first and last names and so its not atomic. Student is not 1NF.

Student – not in 1NF		
studentNo	name	gender
444	Jim Smith	m
254	Donna Jones	f
333	Peter Thomas	m
765	Jim Smith	m

If we modify Student so there are two attributes (say, first and last) then Student would be 1NF:

Student – in 1NF			
studentNo	first	last	gender
444	Jim	Smith	m
254	Donna	Jones	f
333	Peter	Thomas	m
765	Jim	Smith	m

If we can say that a relation (or table) is in 1NF then we are saying that every attribute is atomic and every value is single-valued. This simplifies the form of a relation.

It is very common for names to be separated out into two or more attributes. However, attributes such as birth dates, hire dates, etc. are usually left as a single attribute. Dates could be separated out into day, month, and year attributes, but that is usually beyond the needs of the intended system. Some would take the view that separating a date into 3 separate attributes is carrying the concept of normalization a little too far. Database systems do have

convenient functions that can be used to obtain a day, month, or year values from a date.

## 10.2 Exercises

1. Consider the relation below that holds information about courses and sections. Suppose departments have courses and offer these courses during the terms of an academic year. A section has a section number, is offered in a specific term (e.g. Fall 2016, Winter 2017) and a slot (e.g. 1, 2, 3, ...15) within that term. Each time a course is delivered, there is a section for that purpose. Each section of a course has a different number. As you can see, a course may be delivered many times in one term.

CourseDelivery		
deptNo	courseNo	delivery
ACS	1903	001, Fall 2016, 05; 002, Fall 2016, 06; 003, Winter 2017, 06
ACS	1904	001, Fall 2016, 12; 002, Winter 2017, 12
Math	2201	001, Fall 2016, 11; 050, Fall 2016, 15
Math	2202	050, Fall 2016, 15

Modify CourseDelivery to be in 1NF. Show the contents of the rows for the above data.

2. Chapter 8 covered mapping an ERD to a relational database. Consider the examples from Chapter 8; are the relations in

1NF?

## 10.3 Boyce-Codd Normal Form (BCNF)

Initial research into normal forms led to 1NF, 2NF, and 3NF, but later<sup>2</sup> it was realized that these were not strong enough. This realization led to BCNF which is defined very simply:

A relation R is in **BCNF** if R is in 1NF and every determinant of a non-trivial functional dependency in R is a candidate key.

BCNF is the usual objective of the database designer, and is based on the notions of candidate key (CK) and functional dependency (FD). When we investigate a relation to determine whether or not it is in BCNF, we must know what attributes or attribute combinations are CKs for the relation, and we must know the FDs that exist in the relation. Our knowledge of the semantics of a relation guides us in determining CKs and FDs.

Recall that a CK is an attribute, or attribute combination, that uniquely identifies a row. Also, recall a CK is minimal – no attribute can be removed without losing the property of being a key.

Recall that a FD  $X \rightarrow Y$  in a relation R means that for each row in the relation R that has the same value for X the value of Y must also be the same.

Recall that when we consider a FD  $X \rightarrow Y$  we refer to the left hand side, attribute X, as the determinant. We are concerned with minimal FDs – all attributes comprising the determinant are required for the FD property to hold. If  $X \rightarrow Y$  is a FD then the

2. Codd, E.F. (1974) —Recent Investigations in Relational Database Systems, Proceedings of the IFIP Congress, pp. 1017–1021.

determinant augmented with any other attribute is also a FD, but it would not be a minimal FD.

We consider a number of examples. The keep the examples simple and to the point, each relation involves very few attributes. This is of course unrealistic – in practice relations usually have many attributes. However, the examples illustrate one point each, and more attributes in the relations may cloud the issues. Each example begins with a relation that is in 1NF.

In general, when we determine the relation under consideration is not in BCNF, we obtain BCNF relations by decomposing the relation into two or more relations that are in BCNF. In this process, we say we take a projection of the original relation on a subset of its attributes and at the same time we eliminate any duplicate rows. An important property of the decomposition is that it must be lossless – the new relations will have attributes in common that can be used to join the new relations whereby we can realize the original relation. All rows of the original relation are obtained in the join, and no new or spurious rows are generated – we get back the original relation exactly.

In Example 1, we have a ‘good’ relation, one that is in BCNF. Hence, no decomposition is required. We discuss the CDs and FDs for the relation thereby knowing it is in BCNF.

Example 2 presents a relation that is not in BCNF. There is a type of redundancy present in its data. We illustrate how to decompose the relation into two relations that are each in BCNF. This example illustrates a type of dependency known as a partial functional dependency.

Example 3 presents another relation that is not in BCNF. There is a type of redundancy present in its data. We illustrate how to decompose the relation into two relations that are each in BCNF. This example illustrates a type of dependency known as a transitive functional dependency.

Our last example is a case where FDs involve overlapping candidate keys, and where FDs exist amongst attributes that make up CKs. There is a type of redundancy present which is not related

to 2NF and 3NF. BCNF gives us a theoretical basis for recognizing the source of the redundant data.

## Example 1

Consider the Employee relation below that depicts sample data for 5 employees. The semantics are quite simple: for each employee identified by a unique employee number, we store the employee's first name and last name.

Employee		
id	first	last
1	Joe	Jones
2	Joe	Smith
3	Susan	Smith
4	Abigail	McDonald
5	Abigail	McDonald

### *Candidate Keys*

The hypothetical company that uses this relation identifies employees by an identification number that is assigned by the Human Resources Department and they ensure each employee has a different id from every other employee. Clearly id is a candidate key. When an employee is hired they have a first and last name, and the company has no control over these names. As the sample data shows, more than one employee can have the same first name (id 1



and 2), can have the same last name (id 2 and 3), and can even have the same first and last names (id 4 and 5).

So, id is the only candidate key for this relation.

## *Functional Dependencies*

Since each row/employee has a unique identifier, it is easy to see there are two FDs for this relation:

id  $\rightarrow$  first

id  $\rightarrow$  last

There are no other FDs. For example, we cannot have first  $\rightarrow$  last. The sample data shows there can be many last names associated with any one first name.

These two FDs are minimal as the determinant, id, cannot be reduced at all.

## *BCNF?*

In this example, we have one candidate key, id, and this attribute is the determinant in all FDs. Therefore, **Employee relation is in BCNF**; it is a 'good' relation.

This relation has a 'nice' simple structure; there is one candidate key which is the determinant for every FD.

## **Example 2**

Consider the following relation named Enrollment:

Enrollment		
stuNum	courseId	birthdate
111	2914	Jan 1, 1995
113	2914	Jan 1, 1998
113	3902	Jan 1, 1998
118	2222	Jan 1, 1990
118	3902	Jan 1, 1990
202	1805	Jan 1, 2000

The semantics of this relation are:

- Each row represents an enrollment of a student in a course.
- A student is identified by their student number.
- A course is identified by a course identifier.
- A student can only enroll in a course once. Hence the combinations {stuNum,courseId} are unique.
- The birthdate column holds the date of birth for the student of that row. When the same student number appears in more than one row then the birthdate appears redundantly.
- A course can have many students registered in it

## *Candidate Keys*

It should be clear that several rows may exist for any given student number, and several rows may exist for any given course number. Also, since we cannot control when someone is born, there can be many rows for a value of birthdate. All this just means that no single attribute uniquely identifies a row and so no single attribute can

be a CK. Any CKs for this relation must be composite – comprising more than one attribute. It should be fairly clear, given the semantics of the relation, that the only attribute combination that is a CK is {stuNum,courseId}. For any given value of {stuNum, courseId} there can be at most one row.

## *Functional Dependencies*

This relation is quite simple in that there is just one FD: stuNum  $\rightarrow$  birthdate. If a specific student number appears in more than one row, the value stored for birthdate must be the same in all such rows.

## *BCNF?*

Enrollment has one CK: {stuNum, courseId}, and has one FD (stuNum  $\rightarrow$  birthdate) where the determinant is not a candidate key. Therefore, **Enrollment is not in BCNF**.

In this relation, we have an attribute that does not describe the whole key – it describes a part of the key. In normalization theory, the FD stuNum  $\rightarrow$  birthdate is called a **partial functional dependency** as its determinant is a subset of a candidate key.

When you think of the Enrollment relation now, you should consider that it is about two very different things:

1. Enrollment presents enrollment information.
2. Enrollment presents information about students (their birthdates).

# Decomposition

We now consider how Enrollment can be replaced by two relations where the new relations are each in BCNF. Above, we mentioned that Enrollment is about two very different things – what we need to do is arrange for two relations, one for each of these concerns.

Consider the following two relations as a decomposition of the above where we have placed information about enrollments in one relation and information about students in another relation. Note that these two relations have the stuNum attribute in common.

Enrollments	
stuNum	courseId
111	2914
113	2914
113	3902
118	2222
118	3902
202	1805

Students	
stuNum	birthdate
111	Jan 1, 1995
113	Jan 1, 1998
118	Jan 1, 1990
202	Jan 1, 2000

Enrollments and Students can be joined on stuNum to reproduce the exact information in Enrollment. Because we have not lost any information, and noting that the FD has been preserved, these two relations are equivalent to the one we started with.

- Enrollments has one candidate key: {stuNum,courseId}, and no FDs.

Therefore, **Enrollments is in BCNF**.

Students has one CK: stuNum, and has one FD: stuNum → birthdate.

Therefore, **Students is in BCNF**.

## Example 3

Consider the following relation named Course.

Course		
courseId	teacherId	lastName
2914	11	Smith
3902	22	Jones
3913	11	Smith
4902	33	Jones
4906	11	Smith
4994	22	Jones

The purpose of this relation is to record who is teaching courses. Note that a teacher's id and last name may appear in several rows – this information is repeated for each course the teacher is teaching. For example, teacher 11 (Smith) is teaching 3 courses (2914, 3913, 4906) and so we see the same id and last name in three rows.

The semantics of this relation are:

- Each course is identified by a course identifier.
- For each course there is one row.
- Each teacher is identified by a teacher identifier.
- Each course has one teacher, and so for each course one teacher Id is recorded.
- A teacher may teach several courses.
- A teacher's last name must be the same in every row where the teacher's Id appears. This point leads to redundant data in the relation.

## *Candidate Keys*

The semantics of the relation are that there is one row per course, and so a course id uniquely identifies a row; so, courseId is a candidate key. No other attribute or combination can be a candidate key for this relation.

## *Functional Dependencies*

It is stated there is one teacher per course and so for each courseId there is at most one teacherId, and so we have  $\text{courseId} \rightarrow \text{teacherId}$ . The opposite,  $\text{teacherId} \rightarrow \text{courseId}$ , does not hold for this relation since a teacher can teach more than one course.

Another FD that is present is  $\text{teacherId} \rightarrow \text{lastName}$ . This is because for each teacher there is a single last name. Note the opposite,  $\text{lastName} \rightarrow \text{teacherId}$  does not hold in this relation. The sample data shows multiple teachers who have the same last name.

Note that since  $\text{courseId} \rightarrow \text{teacherId}$  and  $\text{teacherId} \rightarrow \text{lastName}$ , it must be true we have the FD  $\text{courseId} \rightarrow \text{lastName}$ . For each course, we have one teacher and so one last name. For any value of course id, there will only be one value for teacher last name. In relational database theory, the FD  $\text{courseId} \rightarrow \text{lastName}$  is called a **transitive functional dependency** – lastName is dependent on courseId but this dependency is via teacherId.

## *BCNF?*

Hopefully, you agree the only FDs are these:

- $\text{courseId} \rightarrow \text{teacherId}$

- $\text{teacherId} \rightarrow \text{lastName}$
- $\text{courseId} \rightarrow \text{lastName}$

The only candidate key is  $\text{courseId}$ , and there is a FD,  $\text{teacherId} \rightarrow \text{lastName}$ , where the determinant is not a candidate key. Therefore, **Course is not BCNF.**

When you think of the Course relation now, you should see that it is about two very different things:

1. Course presents teacher information ( $\text{teacherId}$ ) for courses.
2. Course presents information about teachers (their last names).

## *Decomposition*

Course can be replaced by two relations where the new relations are each in BCNF. Above, we mentioned that Course is about two very different things – what we need to do is arrange for two relations, one for each of these concerns.

Consider the following two relations as a decomposition of the above where we have placed information about courses in one table and information about teachers in another table. These relations have a common attribute,  $\text{teacherId}$ .



Courses	
courseId	teacherId
2914	11
3902	22
3913	11
4902	33
4906	11
4994	22

Teachers	
teacherId	lastName
11	Smith
22	Jones
33	Jones

Courses and Teachers can be joined on teacherId to reproduce exactly the information in Course. Because we have not lost any information, and noting that the FD has been preserved as well, these two relations are equivalent to the one we started with.

- Courses has one candidate key: courseId. The only FD is courseId?teacherId. Therefore, **Courses is in BCNF**.
- Teachers has one candidate key: teacherId. There is one FD: teacherId?lastName. Therefore, **Teachers is in BCNF**.

## Example 4

This example uses a relation that contains data obtained from a 2011 Statistics Canada survey. Each row gives us information about the percentage of people in a Canadian province who speak a language considered their mother tongue<sup>3</sup>. The ellipsis “...” indicate there are more rows.

Province Language Statistics			
provCode	provName	language	percentMotherTongue
MB	Manitoba	English	72.9
MB	Manitoba	French	3.5
MB	Manitoba	non-official	21.5
SK	Saskatchewan	English	84.5
SK	Saskatchewan	French	1.6
SK	Saskatchewan	non-official	12.7
NU	Nunavut	English	28.1
...	...	...	...

3. Mother tongue refers to the first language learned at home in childhood and still understood by the person at the time the data was collected. The person has two mother tongues only if the two languages were used equally often and are still understood by the person.

The ProvinceLanguageStatistics relation has redundant data. In the rows listed above, we see that each province name and each province code appear multiple times.

## *Candidate Keys*

There can be more than one row for any province. For the combination of province and language, however, there can be only one row and so there are two composite candidate keys:

$\{\text{provCode}, \text{language}\}$   
 $\{\text{provName}, \text{language}\}$

## *Functional Dependencies*

Since province codes and province names are unique, we have the FDs:

$\text{provCode} \rightarrow \text{provName}$   
 $\text{provName} \rightarrow \text{provCode}$

For each combination of province and language, there is one value for percent mother tongue. We have FDs:

$\text{provCode}, \text{language} \rightarrow \text{percentMotherTongue}$   
 $\text{provName}, \text{language} \rightarrow \text{percentMotherTongue}$

## *BCNF?*

The first two FDs listed above have determinants that are subsets of candidate keys. Therefore, **ProvinceLanguageStatistics is not BCNF**.

The ProvinceLanguageStatistics relation has information about two different things:

- It has information about provinces (names/codes).
- It has information about mother tongues in the provinces.

## *Decomposition*

To obtain BCNF relations, we must decompose ProvinceLanguageStatistics into two relations. For example, consider Province and ProvinceLanguages below:

Province	
provCode	provName
MB	Manitoba
SK	Saskatchewan
NU	Nunavut
...	...

ProvinceLanguages		
provCode	language	percentMotherTongue
MB	English	72.9
MB	French	3.5
MB	non-official	21.5
SK	English	84.5
SK	French	1.6
SK	non-official	12.7
NU	English	28.1
NU	French	1.4
NU	non-official	69.6
...	...	...

These relations can be joined on provCode to produce exactly the information shown in ProvinceLanguageStatistics.

- Province has two composite keys (CKs): provCode and provName.
- There are two functional dependencies (FDs): provCode → provName and provName → provCode.  
Therefore, **Province is in BCNF**.
- ProvinceLanguages has one CK: {provCode,language}, and one FD: {provCode,language} → percentMotherTongue.  
Therefore, **ProvinceLanguages is in BCNF**.

## 10.4 Summary

We have discussed functional dependencies, candidate keys, 1NF and BCNF. BCNF is the usual objective of the database designer. When a relation is not BCNF then one or more of the following will be the source of redundancy in a relation:

- Partial dependencies
- Transitive dependencies
- Functional dependencies amongst key attributes.

**2NF:** 2NF involves the concepts of candidate key and non-key attributes. A relation is considered to be in **2NF** if it is in 1NF, and every **non-key** attribute is fully dependent on each candidate key.

In Example 2, we mentioned that stuNum  $\rightarrow$  birthdate was considered a partial functional dependency as stuNum is a subset of a candidate key. A 2NF relation does not contain partial dependencies.

**3NF:** 3NF involves the concepts of candidate key and non-key attributes. We say a relation is in **3NF** if the relation is in 1NF and all determinants of **non-key** attributes are candidate keys.

In Example 3, we mentioned that courseId  $\rightarrow$  lastName was considered a transitive dependency. LastName is dependent on teacherId which is not a candidate key. A 3NF relation does not have partial dependencies nor transitive dependencies.

**BCNF:** The definition of BCNF concerns FDs and CKs – there is no mention of non-key attributes. Hence, BCNF is a stronger form than 2NF or 3NF (a BCNF relation will be in 2NF and 3NF).

A database designer may decide to not normalize completely to BCNF. This is sometimes done to ensure that certain data can be retrieved without having to join relations in a query – when a join is avoided the data is typically retrieved more quickly from the database. This is often done in a data warehouse environment (outside the scope of these notes).

## 10.4 Exercises

In each of these exercises, consider the relation, CKs, and FDs. Determine if the relation is in BCNF. If not in BCNF, give a non-loss decomposition into BCNF relations. The last 5 questions are abstract and give no context for the relation nor attributes.

1. Identify the relationships of Player and the Player fields. Player has information about players for some sports league. While using the entities and fields found in Player, create a DBDL example of tables, fields, and key fields that are in first normal form, second normal form and third normal form. Convert this table to an equivalent collection of tables, fields and keys that are in first normal form. Represent your exercise answers in DBDL design from the database normalization phases explained in class. Player has attributes id, first, last, gender. Id is the only CK and the FDs are:

id  $\rightarrow$  first

id  $\rightarrow$  last

id  $\rightarrow$  gender

Player – sample data			
id	first	last	gender
1	Jim	Jones	Male
2	Betty	Smith	Female
3	Jim	Smith	Male
4	Lee	Mann	Male
5	Samantha	McDonald	Female

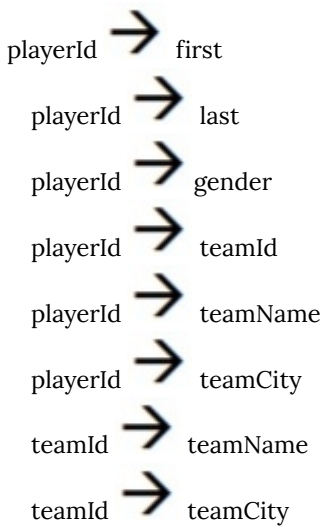
- Identify the relationships of Employee and fields for Employee. Employee has information about employees in some company. While using the entities and fields found in Player, create a DBDL example of tables, fields, and key fields that are in first normal form and second normal form and third normal form. Convert this table to an equivalent collection of tables, fields and keys that are in first normal form and second normal form and third normal form. Represent your exercise answers in DBDL design from the database normalization phases explained in class. Employee has attributes id, first, last, sin (social insurance number) where id and sin are the only CKs, and the FDs are:

id  $\rightarrow$  first  
 id  $\rightarrow$  last  
 sin  $\rightarrow$  first  
 sin  $\rightarrow$  last  
 id  $\rightarrow$  sin  
 sin  $\rightarrow$  id

Employee – sample data			
id	first	last	sin
1	Jim	Jones	111222333
2	Betty	Smith	333333333
3	Jim	Smith	456789012
4	Lee	Mann	123456789
5	Samantha	McDonald	987654321



3. Identify the relationships of Player and Player fields including PKs, CKs, and FDs. While using the entities and fields found in Player, create a DBDL example of tables, fields, and key fields that are in third normal form. Convert this table to an equivalent collection of tables, fields and keys that are in third normal form. Represent your exercise answers in DBDL design from the database normalization phases explained in class. Player contains information about players and their teams. Player has attributes playerId, first, last, gender, teamId, teamName, teamCity where playerId is the only CK and the FDs are:



Player – sample data						
playerId	first	last	gender	teamId	teamName	teamCity
1	Jim	Jones	M	1	Flyers	Winnipeg
2	Betty	Smith	F	5	OilKings	Calgary
3	Jim	Smith	M	10	Oilers	Edmonton
4	Lee	Mann	M	1	Flyers	Winnipeg
5	Samantha	McDonald	F	5	OilKings	Calgary
6	Jimmy	Jasper	M	99	OilKings	Winnipeg

4. Consider a relation Building which has information about buildings and floors. Identify the relationships of Building and Building fields including PKs, CKs, and FDs. While using the information in Building, create a DBDL example of tables and fields that are in third normal form. Convert this table to an equivalent collection of tables, fields, and key fields that are in third normal form. Represent your exercise answers in DBDL design from the database normalization phases explained in class. Building has attributes buildingCode, floor, numRooms, campus where {buildingCode,floor} is the only CK and the FDs are:

{buildingCode,floor} → numRooms  
 buildingCode → campus

Building – sample data			
buildingCode	floor	numRooms	campus
D3	3	15	Downtown – 3
C	2	5	Central
RP	1	20	Selkirk
D2	2	5	Downtown – 2
D1	1	20	Downtown – 1

5. Consider a relation Course which contains information about courses. While using the entities and fields found in Course, create a DBDL example of tables, fields, and key fields that are in third normal form. Convert this table to an equivalent collection of tables, fields and keys that are in third normal form. Represent your exercise answers in DBDL design from the database normalization phases explained in class. Course has attributes deptCode, deptName, courseNum, creditHours where {deptCode,courseNum} and {deptName,courseNum} are the only CKs. The FDs are:

{deptCode,courseNum}  $\rightarrow$  creditHours  
 {deptName,courseNum}  $\rightarrow$  creditHours  
 deptCode  $\rightarrow$  deptName  
 deptName  $\rightarrow$  deptCode

Course – sample data			
deptCode	deptName	courseNum	creditHours
Math	Mathematics	2101	3
Stat	Statistics	4002	3
Phy	Physics	3101	1
Stat	Statistics	4001	6
Math	Mathematics	2111	6

- Consider the relation Student Performance below which describes student performance in courses. While using the entities and fields found in Student Performance, create a DBDL example of tables, fields, and key fields that are in third normal form. Convert this table to an equivalent collection of tables, fields and keys that are in third normal form. Represent your exercise answers in DBDL design from the database normalization phases explained in class. The value stored in the gradePoint column is the grade point that corresponds to the grade received in a course. Assume that students are identified by their student number, and that courses are identified by their course id. Assume each student can take a course only once and so each row is uniquely identified by {stuNum, courseId}. Each student's overall gpa is stored – gpa is the average of gradePoint for all courses taken by a student.

Student Performance – sample data				
stuNum	courseId	grade	gradePoint	gpa
111	3030	C	2.0	2.0
113	3030	C	2.0	2.5
113	4040	B	3.0	2.5
118	2222	C	2.0	2.25
118	4040	C+	2.5	2.25
202	1188	B	3.0	3.0

- Consider Example 4. Is there another decomposition of ProvinceLanguageStatistics that leads to BCNF relations?
- Consider a relation R with attributes X, Y, W, Z where X is the only CK, and where there are FDs:

$X \rightarrow Y$   
 $X \rightarrow W$   
 $X \rightarrow Z$

- Consider a relation R with attributes X, Y, W, V where X and V are the only CKs, and where there are FDs:

$X \rightarrow Y$   
 $X \rightarrow W$   
 $V \rightarrow Y$   
 $V \rightarrow W$   
 $X \rightarrow V$   
 $V \rightarrow X$

10. Consider a relation R with attributes X, Y, W, V, Z where X is the only CK, and where there are FDs:

$$\begin{aligned}X &\rightarrow Y \\X &\rightarrow W \\W &\rightarrow Z \\W &\rightarrow V\end{aligned}$$

11. Consider a relation R with attributes A, B, C, D, E, F where {A,B} is the only CK, and where there are FDs:

$$\begin{aligned}\{A,B\} &\rightarrow C \\ \{A,B\} &\rightarrow D \\ A &\rightarrow E \\ A &\rightarrow F\end{aligned}$$

12. Consider a relation R with attributes A, B, C, D, E where {A,C} and {B,C} are the only CKs, and where there are FDs:

$$\begin{aligned}\{A,C\} &\rightarrow D \\ \{B,C\} &\rightarrow D \\ \{A,C\} &\rightarrow E \\ \{B,C\} &\rightarrow E \\ A &\rightarrow B \\ B &\rightarrow A\end{aligned}$$

# Appendix A: Forms Involving Multiple Tables

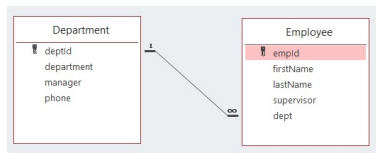
RON MCFADYEN

In Chapter 3, we created simple forms for single tables. A very useful form is one where the user can interact with data that comes from more than one table. We will consider how this can be done in cases where two tables are related by a one-to-many relationship.

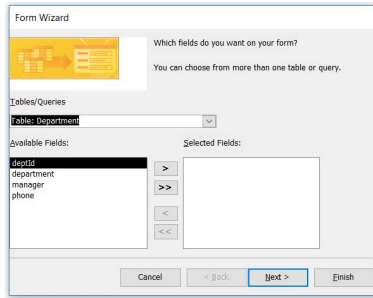
We will illustrate creating such a form using the Microsoft Access Form Wizard. As you will see, the Form Wizard will create a form and a *subform*. These two forms will have a connection established based on related fields: a primary key and a foreign key.

Consider using the Company database:

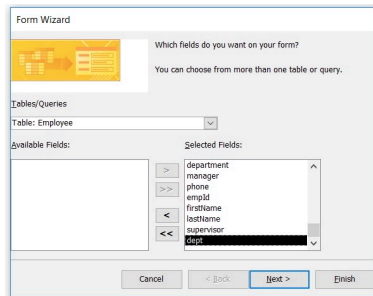
1. If the one-to-many relationship between Department table and Employee table does not exist, then create this now. Note that this is Exercise 1 in Chapter 5. After doing this you should have the relationship as shown:



2. Use the Create tab and create a form using the Form Wizard. Select all fields from the Department table:

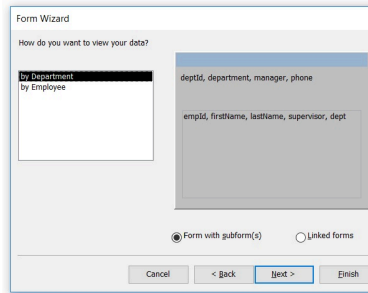


3. Do not click Next or Finish, instead choose the Employee table and select all of its fields and now the Selected Fields component shows fields from both tables:

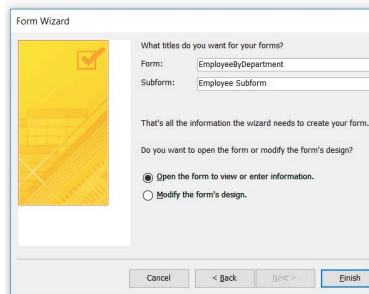


4. Now, click Next and Microsoft Access asks you how the data should be viewed:





5. We want the data displayed “by Department” and we want MS Access to use “Form with subform(s)” so you can select Next and Microsoft Access will let you choose a layout. Choose Datasheet Layout. Click Next and Access will ask you to name the form – name the form EmployeesByDepartment and name the subform EmployeesSubform:



6. Click Finish. Microsoft Access will display the finished form called EmployeesByDepartment – see below. Experiment with the form: notice the two sets of navigation buttons – one that controls the department being viewed, and the other that controls the view of the

department's employees.

empid	firstName	lastName	supervisor
2	Heidi	Herring	1
3	Oliver	Holt	2
7	Bruce	Franko	2
8	Bruno	Peris	2
9	Whitney	Christie	2
10	Leon	Harrison	2
11	Raya	Cook	2
12	Armenia	Decker	5
13	Kareem	Freeman	5
14	Arin	Stephens	5
15	Rogan	Clements	5
16	Mandy	Schmitt	5

## Exercises

1. Consider using the *University* database. Create a form to allow a user to view courses by department.
2. Consider using the *Library* database. There are two one-to-many relationships. Create a form to list the loan records for a book. Create another form to list the loan records for a member.
3. Consider using the *Orders* database. This database has several one-to-many relationships. Create appropriate forms to list
  1. A customer and the customer's orders;
  2. An order and its detail lines;
  3. A product and the order detail lines where the product is referenced;
  4. A category and the products belonging to the category.

# Appendix B: Supertypes and Subtypes

RON MCFADYEN

We have covered the basics of entity-relationship modeling and now we will extend our design capabilities to include *supertypes* and *subtypes*. It is often the case that an entity type has subgroupings that are useful to include in a data model. For instance, in a University environment, persons could be grouped into employees and students. Courses could be grouped into graduate courses and undergraduate courses.

Previously, we considered a Library database where one entity type was book; instances of book are loaned out to library members. A library could have many other kinds of things that it loans out such as videos and magazines. A more general thing the library loans out can be referred to as an item. Videos, magazines, and books can be considered subtypes of item.

We will consider only supertype and subtype hierarchies. Hierarchies arise when an entity type appears as a subtype of only one supertype. So we are disallowing cases where an entity type has two or more supertypes.

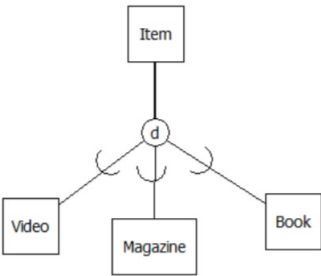
## B.1 Drawing Supertypes and Subtypes on the Entity Relationship Diagram

There are different ways that supertypes and subtypes can be shown on an Entity Relationship diagram (ERD or ER diagram). We will continue with the Peter Chen notation in this appendix. Between a supertype and its subtypes, we show a connection symbol (a circle) where one line is drawn from the supertype to the

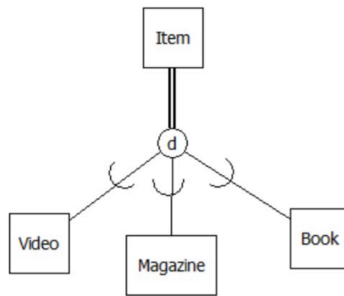
connection symbol and then lines are drawn from the connection symbol to each subtype.

A collection of related subtypes can be regarded as overlapping or disjoint. Subtypes are considered as disjoint if it is impossible for an instance of a supertype to be regarded as being an instance of more than one subtype. For example, a library item will be one of the subtypes (and only one). Subtypes are considered as overlapping if it is possible for an instance of a supertype to be regarded as being an instance of more than one subtype. An example of overlapping can exist with people in a university environment: it is possible that some person could be both an employee and a student at the same time. In our Peter Chen notation, we will use a “d” in the connection symbol to represent disjoint subtyping, and we will use “o” to represent overlapping.

In our notation, we also include an arc on each of the lines joining the connection symbol to the subtypes that implies “containment”. To illustrate the drawing technique, consider a library where items are loaned to members and where an item can be either a video, a magazine, or a book. Suppose also that an item belongs to exactly one (i.e. disjoint subtypes) of these subtypes. We can show this as:



We extend our notation once more. To indicate that a supertype **must** exist as one of its subtypes, we show total participation in subtyping by using a double line. For example, if we want to show that each item must be one of the subtypes:



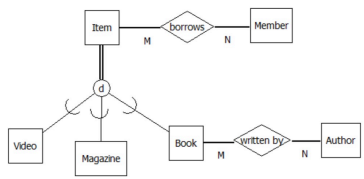
The double line from *Item* to the connection symbol shows total participation of *Item* in the subtyping: whenever there is an instance of an *Item*, then that item must also be one of the subtypes shown – a video, a magazine, or a book. If we did not specify total participation then we would be allowing an item to exist where that item is not a video, nor is it a magazine, nor is it a book. So, participation of a supertype in the subtyping is either total or optional. The converse is always true: if we have an instance of a subtype then that instance is an instance of the supertype. In the library model then, if we have an instance of book then that instance is of course an item.

## B.2 Supertypes, Subtypes AND Relationships

If a supertype participates in a relationship then all of its subtypes also participate in that relationship. We say that a supertype's relationships are inherited by its subtypes. The converse is not true: if the model specifies specifically that a subtype participates in a relationship, then its siblings (other entity types that are subtypes of the same supertype) and its supertype do not participate in that relationship.

As an example, consider that members can borrow items (i.e. any

item of any type) from the library but only books have authors. Our model can be extended as follows:



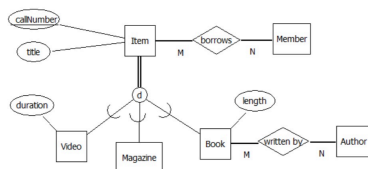
This model excludes the database from storing an author of a magazine or that a video has an author, but the model allows videos, magazines, and books to be borrowed by members.

### B.3 Supertypes, Subtypes and Attributes

All entity types including supertypes and subtypes can have attributes. Continuing with our library example suppose:

- All items have a call number and a title, and call number is a key (each item has a unique call number);
- Videos have a duration (time required to play);
- Books have a length (number of pages).

Just as subtypes inherit relationships, they also inherit any attributes of their supertype. We also have know that supertypes do **not** inherit the attributes of their subtypes. Attributes that are common to a supertype and its subtypes are only shown at the supertype. Consider our model now:

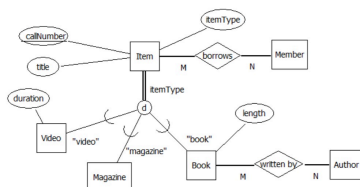


Our examples have been two-level hierarchies. In general, a hierarchy can be as many levels as the designer requires. For instance, books could be categorized as fiction and non-fiction and so book can be a subtype of item and at the same time a supertype of fiction and non-fiction.

### B.3.1 Discriminator Attributes

It is common for designers to introduce or discover an attribute such that its value can be used to explicitly determine the subtype an entity belongs to. For example, the item entity type can have an attribute, say itemType, which can have a value from the domain {"video", "magazine", "book"}. When this is done, the diagram must include the attribute of course, but additionally the attribute is shown as a discriminator attribute for subtyping purposes and the pertinent value for discriminating shown as well.

Below, you will see how these are laid out above and below the connection symbol.



This works well for disjoint subtyping, but not necessarily for overlapping subtypes. When overlap is possible, a designer may include a discriminator for each subtype, and so there are as many discriminator attributes as there are subtypes. Typically, this is a boolean-valued attribute. In the overlapping case, we not show discriminating values on the diagram.

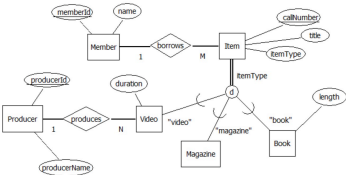
## B.4 Mapping Supertypes and Subtypes To A Relational Database

In chapter 8, we covered rules to be used when an ERD is mapped to a relational database. In this section, we add rules for mapping supertypes and subtypes to relations. There are three basic options a designer considers when mapping these structures to a database:

1. Create a relation for each entity type in the hierarchy.
2. Create relations for only the bottom-most entity types.
3. Create one relation to represent the whole hierarchy.

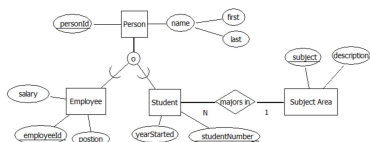
We use two examples to exhibit the mapping options; one where total participation is specified for the supertype and the other where participation is optional.

The previous library model is modified to show that an item can be out on loan to a member, and that one of the subtypes, video, is produced by a producer:





A university model applies where a person may be a student and/or an employee, and where students declare a major subject area:



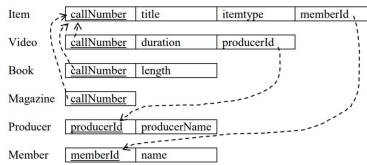
Regardless of the option selected for hierarchies, the rules for mapping an ERD to a relational database discussed previously (Chapter 8) still apply. We must apply rules regarding relationships and attributes consistently. For example, if any entity type in a hierarchy is involved in a one-to-many relationship we must ensure the proper use of foreign keys.

## B.4.1 Relations For All Entity Types

With this option, each entity type in a hierarchy is represented by its own relation. Important points here are that

- All relations representing entity types in the same hierarchy have the same primary key.
- The primary key of a subtype relation will also be a foreign key that references its supertype relation.
- Attributes of a supertype (except for the primary key) appear only in the relation that represents the supertype.

**Example:** The library model maps to the following relational design:



Note the foreign keys:

- Item has a foreign key referencing Member
- Video has a foreign key referencing Producer
- Each of Video, Book, and Magazine has a foreign key referencing Item. If a row exists in Video, Book, or Magazine then there must be a corresponding row in Item.

The tables are shown here with sample data. Note that

- Each row of Video, Book, and Magazine has a related row in Item
- Some items are out on loan to a member
- Each video has a producer

Item			
callNumber	title	itemType	memberId
MAG11	The Java Developer	magazine	2
QA123	Programming with Java	book	1
QA222	C++ Programming	book	
QV123	Fun with Java	video	1
QV222	The BlueJ IDE	video	

Video		
callNumber	duration	producerId
QV123	120	p111
QV222	45	p999

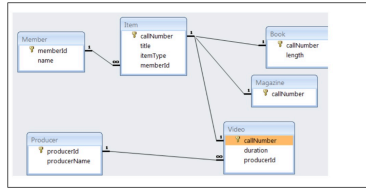
Book	
callNumber	length
QA123	456
QA222	605

Magazine	
callNumber	
MAG11	

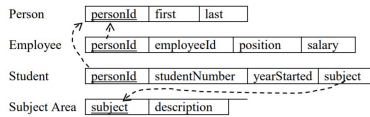
Producer	
producerId	producerName
p111	Sony
p999	Kent University

Member	
memberId	name
1	Joe Smith
2	Janet Lee

In the relationships diagram, note the one-to-one relationships between the supertype relation and each of its subtype relations:



**Example:** Now consider the university model. The relational design for this mapping option:



Since subtyping is optional in the university model, there can be a row in Person with no corresponding row in Employee or Student. A person does not have to exist as one of the subtypes.

Note the foreign keys:

- Student has a foreign key referencing SubjectArea
- Employee and Student have foreign keys referencing Person. If a row exists in Employee or Student then a corresponding row must exist in Person.

We will now show tables with some sample data and the relationships diagram.

A sample database is presented below. Note that person 2 is both a student and an employee, and that person 4 is neither a student nor an employee.

Person		
personid	first	last
1	Joe	Smith
2	Janet	Lee
3	Pat	Jones
4	Jack	Lee

SubjectArea	
subject	description
CS	Computer Science
ENG	English
HIST	History

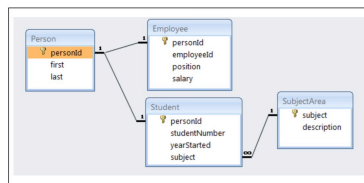
  

Student			
personid	studentNumber	yearStarted	subject
2	3012345	2009	CS
3	3054321	2010	ENG

Employee			
personid	employid	position	salary
1	101	manager	\$100,000.00
2	55	clerk	\$40,000.00

In the relationships diagram, note the relationships are one-to-one between the supertype relation and each of its subtype relations:

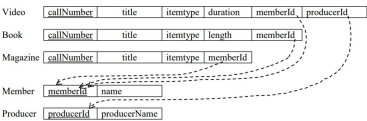


## B.4.2 Relations For Bottom-Most Entity Types

In this case, relations are created for only entity types that are at the “bottom” of the hierarchy. There are no relations created for a supertype. Important points here are that

- All relations derived from entity types in the same hierarchy will have the same primary key.
- No primary key value can be repeated (We have not seen how to handle this in MS Access. Further study of relational systems can include techniques that automate the checking for this kind of integrity constraint.)
- Attributes of a supertype must be included in each of its subtype relations.

**Example:** For the library model and since there is total participation in subtyping, this option works well. Every item will be stored in a relation, and each item is stored exactly once. The resulting design:



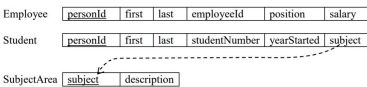
Note the foreign keys:

- Because there is no Item relation, each of Video, Book, and Magazine have foreign keys referencing Member.
- Video is the only relation with a foreign key referencing Producer.

An issue the designer should be aware of is that callNumbers across the three relations must be unique (call number is the primary key of Item). Further study of database systems is needed to know how this rule can be enforced.

It is left as an exercise for the student to create a database with sample data.

**Example:** Consider the university model. This approach (creating relations for bottom-most entity types) is not suitable for the university model because of the overlapping subtypes and because the participation in subtyping is not total. Applying the option we have:



If an entity exists in more than one subtype then such an entity

will have data stored redundantly in the database. In the design above if a person is both an employee and a student then that person's first and last names would be stored twice (in two different relations).

The Employee and Student relations are not sufficient to store Person data. The participation is optional and so a person may exist who is neither an employee nor a student; in such a case the data for the person cannot be stored!

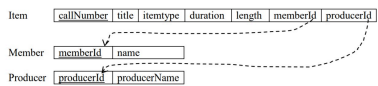
It is left as an exercise for the student to create a database with sample data.

### B.4.3 One Relation Representing The Whole Hierarchy

When this option is applied, one relation is created for a complete hierarchy. All attributes appearing in the hierarchy are placed in one relation. Note that the value of a discriminator attribute will enable the user to know easily the subtype of a particular entity. For our example models, when we map a hierarchy to a single relation we obtain very simple relational designs.

It is left as an exercise for the student to create databases with sample data.

**Example:** The library model maps to the following design



In the Item relation, the itemType attribute indicates if the row represents a video, a magazine, or a book. The memberId may have

a value if the item is out on loan. producerId can only have a value if itemType is “video”.

**Example:** When mapping a hierarchy to a single relation for the university model, the designer should include discriminator attributes that are boolean-valued with one discriminator attribute per subtype. Applying this option to the university model we have:



With this database, each person is stored at most once in the database. There is no duplicated data as with the previous mapping option.

If a person is neither an employee nor a student then the only attributes that can have values are: personId, employeeFlag, studentFlag, first and last – the others must be null. The values of employeeFlag and studentFlag would be false.

## Exercises

1. Consider the database designs illustrated in this appendix. Implement one or more of these and populate with data.
2. Consider the two designs used in the examples of this appendix. Combine these two designs by replacing Member with the Person hierarchy. Illustrate the relational structures when the model is mapped to a database. Choose mapping options for the hierarchies.
3. Consider the design you created in Exercise 2 but modify the one-to-many borrows relationship to be a many-to-many with attributes dateBorrowed and dateReturned where dateBorrowed is a discriminator for the relationship. Recall this discriminator is not the same as the discriminators suggested

for mapping supertypes and subtypes. Note that this modification to the library example will allow history to be recorded for the borrowing of items.

4. For Exercise 3, create the database and populate the database with sample data.
5. Create an ERD for a service station business that provides goods and services to its customers. Typically, a customer comes in with their vehicle and requests certain work to be performed. For example a customer may request an oil change and for a new set of four tires to be provided and installed.

The work items that can be performed or supplied can be of two types: a service (such as the oil change) and actual physical items (such as litres of oil). There will be several services that can be performed such as tire installation, changing oil, or fixing a flat tire. Each of these will have some cost to be charged to a customer. There are many concrete items that are supplied and charged to a customer such as fan belts, litres of oil, or tires – these are things that are kept in inventory. Consider creating a hierarchy for products (goods / services); make up reasonable attributes.

This service station has customers that fall into two groups: some are private individuals and others are businesses. Individuals will have a first name, last name, address and phone number. A business will have a business name, address, phone number and a contact person who has a first name and last name. Consider creating a hierarchy for customers.

The service station needs to keep track of all the goods and services it provides to its customers so that it has a historical record and knows what it has charged to each customer. Each visit to the service station by a customer will generate a work order that keeps track of the work that was done for the customer's vehicle. Vehicles have license plate numbers, and other attributes to describe them (make, model, color, ...). For each visit of a customer to the station, the system needs to know the date the visit occurred, the details of



the work performed and goods provided, and the total charge to the customer.