

# EXTENDED LEARNING MODULE A

## DESIGNING DATABASES AND ENTITY-RELATIONSHIP DIAGRAMMING

### Student Learning Outcomes

1. Identify how databases and spreadsheets are both similar and different.
2. List and describe the four steps in designing and building a relational database.
3. Define the concepts of entity class, instance, primary key, and foreign key.
4. Given a small operating environment, build an entity-relationship (E-R) diagram.
5. List and describe the steps in normalization.
6. Describe the process of creating an intersection relation to remove a many-to-many relationship.

## Introduction

As you learned in Chapter 3, databases are quite powerful and can aid your organization in both transaction and analytical processing. But you must carefully design and build a database for it to be effective. Relational databases are similar to spreadsheets in that you maintain information in two-dimensional files. In a spreadsheet, you place information in a cell (the intersection of a row and column). To use the information in a cell, you must know its row number and column letter. For example, cell C4 is in row 4 and column C.

Databases are similar and different. You still create rows and columns of information. However, you don't need to know the physical location of the information you want to see or use. For example, if cell C4 in your spreadsheet contained sales for Able Electronics (one of your customers), to use that information in a formula or function you would reference its physical location (C4). In a database, you simply need to know you want *sales* for *Able Electronics*. Its physical location is irrelevant—that's why we say that a **database** is a collection of information that you organize and access according to the logical structure of that information.

So you do need to carefully design your databases for effective utilization. In this Module, we'll take you through the process of designing and building a relational database, the most popular of all database types. A **relational database** uses a series of logically related two-dimensional tables or files to store information in the form of a database. There are well-defined rules to follow, and you need to be aware of them.

As far as implementation is concerned, you then just choose the DBMS package of your choice, define the tables or files and the relationships among them, and start entering information. We won't deal with the actual implementation in this Module. However, we do show you how to implement a database using Microsoft Access in Skills Module 2 on the CD-ROM accompanying this text.

Once you've implemented your database, you can change the information as you wish, add rows of information (and delete others), add new tables, and use simple but powerful reporting and querying tools to extract the exact information you need.

## Designing and Building a Relational Database

Using a database amounts to more than just using various DBMS tools. You must also know *how* to actually design and build a database. So, let's take a look at how you would go about designing a database. The four primary steps include:

1. Defining entity classes and primary keys
2. Defining relationships among entity classes
3. Defining information (fields) for each relation (the term *relation* is often used to refer to a file while designing a database)
4. Using a data definition language to create your database

Let's assume you own a small business and are interested in tracking employees by the department in which they work, job assignments, and number of hours assigned to each job. Below is a list of important business rules that you follow:

1. Each employee must be assigned to one and only one department.
2. A department may have many employees but also may not have any.

3. Each employee must be assigned to at least one job and can be assigned to many jobs.
4. A job can have many different employees assigned to it but a given job does not have to have any employees assigned to it.

Figure A.1 contains a sample Employee Report you would like to generate. Let's briefly look at it and review some of the business rules in action.

First, notice that each employee is assigned to only one department. For example, Mills is assigned to only department number 43. Second, notice that several employees are assigned to department number 43—these employees include Jones, Joslin, and Mills. Third, notice that each employee is assigned to at least one job and that several employees have more than one job. Finally, notice that more than one employee is assigned to the same job. For example, Jones, Mills, and Smith are assigned to the job “Acct” (its job number is 14).

Even before you begin the process of designing a database, it's important that you first understand the business rules. These business rules will help you define the correct structure of your database.

## STEP 1: DEFINING ENTITY CLASSES AND PRIMARY KEYS

The first step in designing a relational database is to define the various entity classes and the primary keys that uniquely define each record or instance within each entity class. An **entity class** is a concept—typically people, places, or things—about which you wish to store information and that you can identify with a unique key (called the primary key). A **primary key** is a field (or group of fields in some cases) that uniquely describes each record. Within the context of database design, we often refer to a record as an **instance**—an occurrence of an entity class that can be uniquely described.

From the Employee Report you want to generate, you can easily identify the entity classes *Employee*, *Job*, and *Department*. Now, you have to identify their primary keys. For most entity classes, you cannot use names as primary keys because duplicate names may exist. For example, you have two employees with the last name Jones. However, our sample report does show that each employee has a unique *Employee ID*. So, *Employee ID* should be the primary key for the *Employee* entity class.

Figure A.1

A Sample Report for Your Employee Database

Employee ID	Name	Department Num	Department Name	Num of Employees	Job Number	Job Name	Hours
1234	Jones	43	Residential	3	14 23	Acct Sales	4 4
2345	Smith	15	Commercial	1	14	Acct	8
6548	Joslin	43	Residential	3	23 46	Sales Admin	6 2
9087	Mills	43	Residential	3	23 14	Sales Acct	5 3
8797	Jones	69	Non-profit	1	39	Maint	8



If you consider *Department* as an entity class, you'll find three pieces of information describing it in the report—*Department Num*, *Department Name*, and *Num of Employees*. The logical choice for a primary key here is *Department Num*. Although each *Department Name* is unique, we would still suggest that you not use names.

Likewise, if you consider *Job* as an entity class, you'll find two pieces of information describing it—*Job Number* and *Job Name*. Again, we recommend that you use *Job Number* as the primary key.

For each of these entity classes, you can also easily identify instances. Smith is an instance of the *Employee* entity class, Residential is an instance of the *Department* entity class, and Sales is an instance of the *Job* entity class.

As you begin the process of designing a database, perhaps the easiest way to define entity classes and their primary keys is to do what we have just done—build a sample report that you would like to generate and use it find entity classes and their primary keys.

## STEP 2: DEFINING RELATIONSHIPS AMONG THE ENTITY CLASSES

The next step in designing a relational database is to define the relationships among the entity classes. To help you do this, we'll use an entity-relationship diagram. An **entity-relationship (E-R) diagram** is a graphical method of representing entity classes and their relationships. An E-R diagram includes five basic symbols:

1. A rectangle to denote an entity class
2. A dotted line connecting entity classes to denote a relationship
3. “|” to denote a single relationship
4. A circle to denote a zero or optional relationship
5. A “crow's foot” symbol to denote a multiple relationship

To use these symbols, you must first decide among which entity classes relationships exist. If you determine that two particular entity classes have a relationship, you simply draw a dotted line to connect them and then write some sort of verb that describes the relationship.

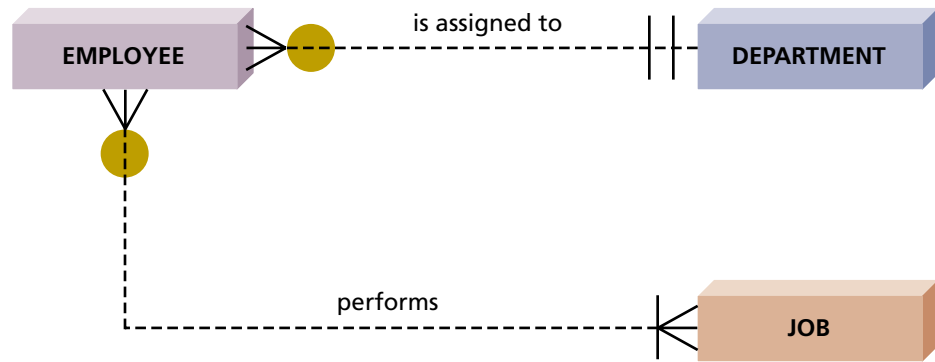
In Figure A.2, you can see the E-R diagram for your employee database. Let's take some time to explore it. To determine where the relationships exist, simply ask some questions and review your business rules. For example, is there a relationship between employees and departments? The answer is yes, because employees are assigned to departments. Likewise, employees are assigned jobs (another relationship). However, there is no logical relationship between departments and jobs. So we drew dotted lines between *Employee* and *Department* and between *Employee* and *Job*. We then added some verbs to describe the relationships. For example, an *Employee* is assigned to a *Department*, and an *Employee* performs a *Job*.

It should also make sense (both business and logical) when you read the relationships in reverse. To do this, simply flip the location of the nouns in the sentence and change the verb accordingly. For example:






- *Employee-Department*: An *Employee* is assigned to a *Department*.
- *Department-Employee*: A *Department* has an *Employee* assigned to it.
- *Employee-Job*: An *Employee* undertakes a *Job*.
- *Job-Employee*: A *Job* is performed by an *Employee*.

Each of the above statements makes logical sense, follows the relationships we identified in Figure A.2, and reflects the business rules given above. Again, we stress the

**Figure A.2**  
An Entity-Relationship  
Diagram



#### ENTITY-RELATIONSHIP DIAGRAM SYMBOLS

	Denotes entity class		Denotes relationship		Denotes multiple relationship
	Denotes single relationship		Denotes zero or optional relationship		

importance of using business rules. Technology (databases, in this instance) is a set of tools that you use to process information. So your implementations of technology should match the way your business works. If you always start by defining business rules and using those rules as guides, your technology implementations will hopefully mirror how your business works. And that's the way it should be.

Once you determine that a relationship does exist, you must then determine the numerical nature of the relationship, what we refer to as minimum and maximum *cardinality*. To describe this, you use a “|” to denote a single relationship, a circle to denote a zero or optional relationship, and/or a “crow’s foot” symbol to denote a multiple relationship. By way of illustration, let’s consider the portion of your E-R diagram in Figure A.3. To help you read the symbols, we’ve added coloured lines and arrows. Following the line marked “A,” you would read the E-R diagram as “An *Employee* is assigned to one *Department* at minimum and one *Department* at maximum.” So that part of the E-R diagram states that the logical relationship between *Employee* and *Department* is that an *Employee* is assigned to one and only one *Department*. This is exactly what business rules 1 and 2 above state.

Following the coloured line marked “B,” you would read the E-R diagram as, “A *Department* is not required to have any *Employees* assigned to it but may have many *Employees* assigned to it.” That statement again reinforces business rules 1 and 2.

Similarly, you can develop statements that describe the numerical relationships between *Employee* and *Job* on the basis of that part of the E-R diagram in Figure A.2. Those numerical relationships would be as follows:

- An *Employee* must be assigned to one *Job* at minimum and can be assigned to many *Jobs* (maximum).
- A *Job* might not have any *Employees* assigned to it but might have many *Employees* assigned to it.

Again, these statements reinforce business rules 3 and 4.

To properly develop the numerical relationships (cardinality) among entity classes, you must clearly understand the business situation at hand. That’s why it’s so important to write down all the business rules.

# TEAM WORK

## DEFINING RELATIONSHIPS AMONG ENTITY CLASSES

Let's continue on exploring how to design a relational database for your school and its offering of weekend seminars. If your group correctly completed the tasks

in the previous Team Work, you know that there are four entity classes. They include (along with their primary keys):

Entity Class	Primary Key
Seminar	Seminar 3-character identifier and number (e.g., Web101 and Web205)
Seminar Section	Seminar 3-character identifier, number, and section number (e.g., Web101-1 through Web101-5 and Web205-1 through Web205-4)
Student	Student ID (whatever your school happens to use)
Qualified Teacher	Teacher ID (your school may use the social insurance number)

Now, your task is to define among which of those entity classes relationships exist and then write some verbs that describe each relationship, just we did in Figure A.2. Below, we've drawn rectangles for each of

the four entity classes. Connect with dotted lines those rectangles between which relationships exist and write in appropriate verbs to describe the relationships.

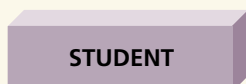
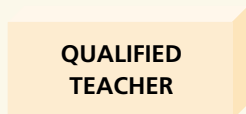
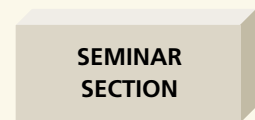
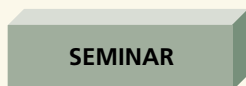
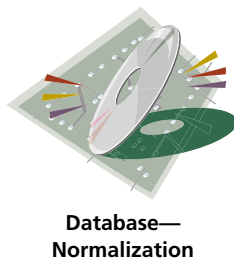
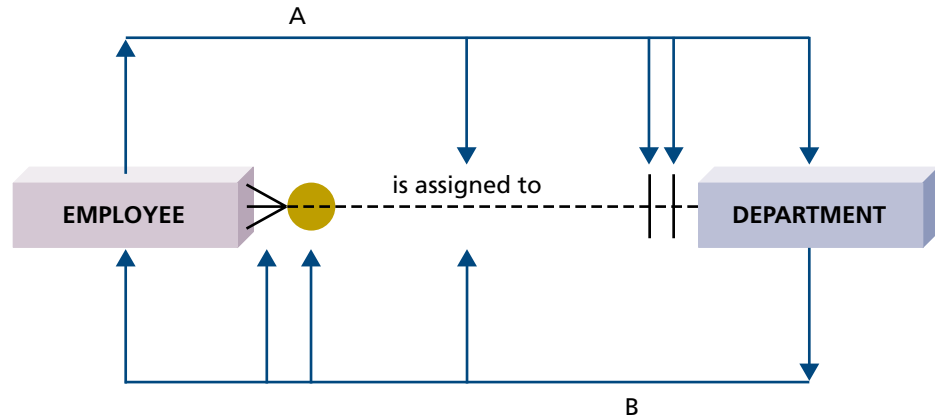


Figure A.3

Reading an Entity-Relationship (E-R) Diagram



After developing the initial E-R diagram, it's time to begin **normalization**—a process of ensuring that a relational database structure can be implemented as a series of two-dimensional relations (remember, *relations* are the same as files or tables). The complete normalization process is extensive but quite necessary for developing organizationwide databases. For our purposes, we will focus on the following three rules of normalization:

1. Eliminate repeating groups or many-to-many relationships.
2. Ensure that each field in a relation depends only on the primary key for that relation.
3. Remove all derived fields from the relations.

The first rule of normalization states that no repeating groups or many-to-many relationships can exist among the entity classes. You can find these many-to-many relationships by simply looking at your E-R diagram and noting any relationships that have a crow's foot on each end. If you look back at Figure A.2, you'll see that a crow's foot is on each end of the relationship between *Employee* and *Job*. Let's look at how to eliminate it.

In Figure A.4, we've developed the appropriate relationships between *Employee* and *Job* by removing the many-to-many relationship. Notice that we started with the original portion of the E-R diagram and created a new relation between *Employee* and *Job* called *Job Assignment*, which is an intersection relation. An **intersection relation** (sometimes called a **composite relation**) is a one you create to eliminate a many-to-many relationship. It's called that because it represents an intersection of primary keys between the first two relations. That is, an intersection relation will have a **composite primary key** that consists of the primary key fields from the two intersecting relations. The primary key fields from the two original relations now become foreign keys in the intersection relation. A **foreign key** is a primary key of one file (relation) that appears in another file (relation). When combined, these two foreign keys make up the composite primary key for the intersection relation.

For your employee database, the intersection relation *Job Assignment* represents which employees are assigned to each job. Here is how you would read the relationships between *Employee* and *Job Assignment* and *Job* and *Job Assignment* (see Figure A.5):

#### Employee-Job Assignment

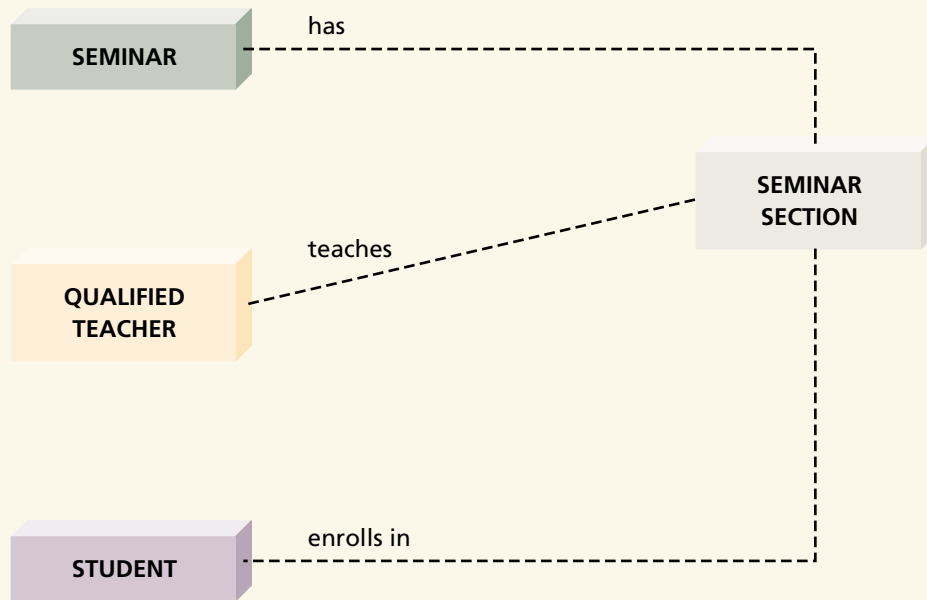
- From left to right: An *Employee* can have many *Job Assignments* and must have at least one *Job Assignment*.
- From right to left: An *Employee* found in *Job Assignment* must be found and can be found only one time in *Employee*.

# TEAM WORK

## DEFINING THE CARDINALITY AMONG ENTITY CLASSES

As we continue with the design of the relational database at your school, it's time to define the cardinality among the entity classes. If you correctly completed the

previous Team Work project, the relationships among the entity classes are as follows:

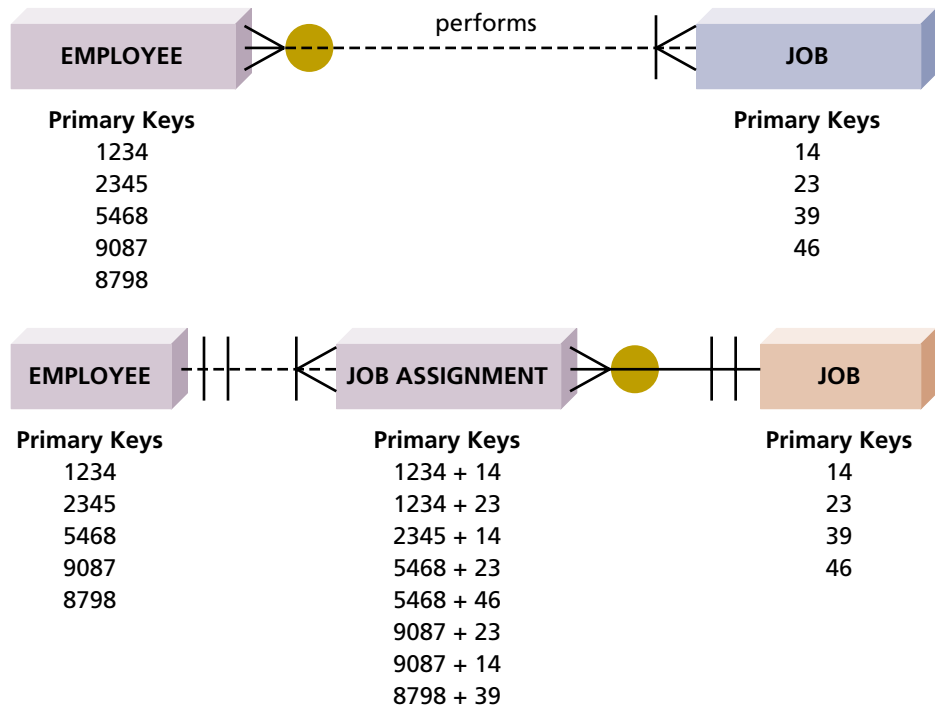


Your task is to create the numerical relationships by adding the cardinality symbols, described earlier, in the appropriate places. Once you do, complete the table

below by providing a narrative description of each numerical relationship.

Relationship	Narrative Description
Seminar-Seminar Section	
Seminar Section-Seminar	
Qualified Teacher-Seminar Section	
Seminar Section-Qualified Teacher	
Student-Seminar Section	
Seminar Section-Student	

**Figure A.4**  
Creating an Intersection  
Relation to Remove a  
Many-to-Many Relationship

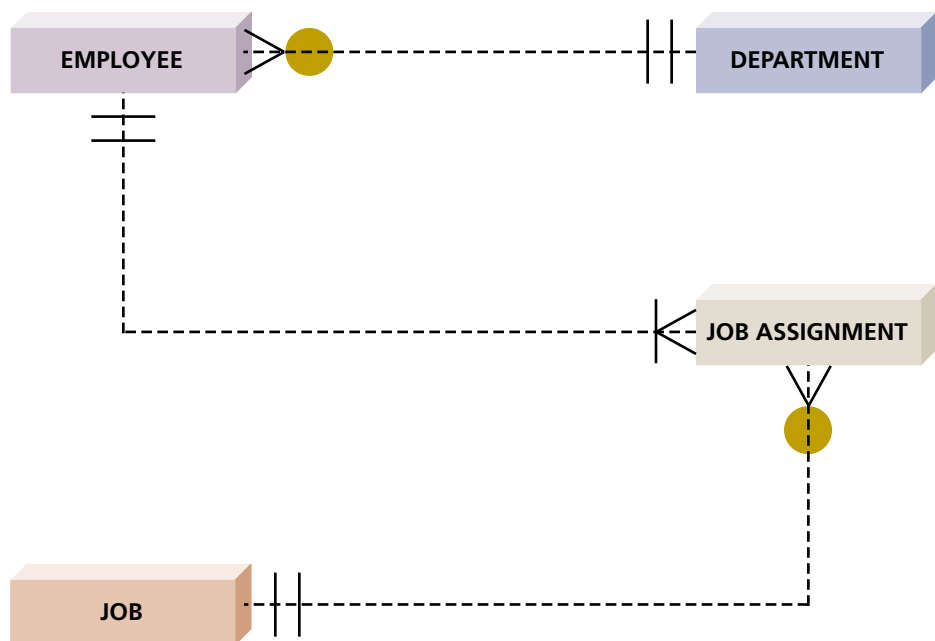


**Job-Job Assignment**

- From left to right: A *Job* can be found in many *Job Assignments* but might not be found in any *Job Assignments*.
- From right to left: A *Job* found in *Job Assignment* must be found and can be found only one time in *Job*.

If you compare the E-R diagram in Figure A.5 to the E-R diagram in Figure A.2, you'll notice that they are very similar. The only difference is that the E-R diagram in Figure A.5 contains an intersection relation to eliminate the many-to-many relationship between *Employee* and *Job*.

**Figure A.5**  
The Completed E-R  
Diagram for Your Employee  
Database



And removing many-to-many relationships is the most difficult aspect when designing the appropriate structure of a relational database. If you do find a many-to-many relationship, here are some guidelines for creating an intersection relation:

1. Just as we did in Figure A.5, start by drawing the part of the E-R diagram that contains a many-to-many relationship at the top of a piece of paper.
2. Underneath each relation for which the many-to-many relationship exists, write down some of the primary keys.
3. Create a new E-R diagram (showing no cardinality) with the original two relations on each end and a new one (the intersection relation) in the middle.
4. Underneath the intersection relation, write down some composite primary keys (these will be composed of the primary keys from the other two relations).
5. Create a meaningful name (e.g., *Job Assignment*) for the intersection relation.
6. Move the minimum cardinality appearing next to the left relation just to the right of the intersection relation.
7. Move the minimum cardinality appearing next to the right relation just to the left of the intersection relation.
8. The maximum cardinality on both sides of the intersection relation will always be “many” (the crow’s foot).
9. As a general rule, the new minimum and maximum cardinalities for the two original relations will be one to one.

### STEP 3: DEFINING INFORMATION (FIELDS) FOR EACH RELATION

Once you’ve completed steps 1 and 2, you must define the various pieces of information that each relation will contain. Your goal in this step is to make sure that the information in each relation is indeed in the correct relation and that the information cannot be derived from other information—the second and third rules of normalization.

In Figure A.6, we’ve developed a view of the relations based on the new E-R diagram with the intersection relation. To make sure that each piece of information is in the correct relation, look at each and ask, “Does this piece of information depend only on the primary key for this relation?” If the answer is yes, the information is in the correct relation. If the answer is no, the information is in the wrong relation.

Let’s consider the *Employee* relation. The primary key is *Employee ID*, so each piece of information must depend only on *Employee ID*. Does *Name* depend on *Employee ID*? Yes—so that information is in the correct relation. Does *Department Num* depend on *Employee ID*? Yes, because each employee’s department designation depends on the particular employee you’re describing. In this case, *Department Num* in the *Employee* relation is an example of a foreign key. What about *Department Name*? The answer here is no. The name for a particular department doesn’t depend on which employee is in that department.

So the question becomes “In which relation should *Department Name* appear?” The answer is in the *Department* relation, because *Department Name* depends on the primary key (*Department Num*) that uniquely describes each department. Therefore, *Department Name* should not be in the *Employee* relation, but rather in the *Department* relation.

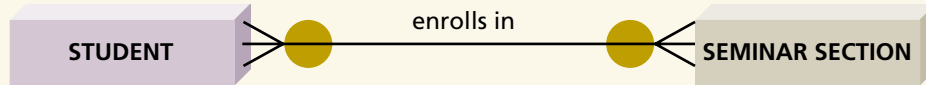
Now, take a look at the intersection relation *Job Assignment*. Notice that it includes the field called *Hours*. *Hours* is located in this relation because it depends on two things—the employee you’re describing and the job to which he or she is assigned. So *Hours* does depend completely on the composite primary key *Employee ID + Job Number* in the *Job Assignment* relation.

# TEAM WORK

## CREATING AN INTERSECTION RELATION

Back to work on the project for your school. If you completed the last Team Work project, you are now able to identify a many-to-many relationship between *Student* and *Seminar Section*. That is, a given *Seminar*

*Section* may have many *Students*, and a given *Student* can enroll in many different *Seminar Sections* (two to be exact). So that portion of your E-R diagram looks like this:



Your task is to eliminate the above many-to-many relationship by creating an intersection relation. As you do, we would encourage you to follow the guide-

lines given above. What did you name the intersection relation? What does the completed E-R diagram look like?

Figure A.6

A First Look at the Relations in Your Employee Database

EMPLOYEE RELATION

Employee ID	Name	Department Num	Department Name
1234	Jones	43	Residential
2345	Smith	15	Commercial
5468	Joslin	43	Residential
9087	Mills	43	Residential
8798	Jones	69	Non-profit

JOB RELATION

Job Number	Job Name
14	Acct
23	Sales
39	Maint
46	Admin

DEPARTMENT RELATION

Department Num	Department Name	Num of Employees
15	Commercial	1
43	Residential	3
69	Non-profit	1

JOB ASSIGNMENT RELATION

Employee ID	Job Number	Hours
1234	14	4
1234	23	4
2345	14	8
5468	23	6
5468	46	2
9087	23	5
9087	14	3
8798	39	8

*Hours* belongs in this relation because it depends on a combination of who (*Employee ID*) is assigned to which job (*Job Number*).

If you follow this line of questioning for each relation, you'll find that all other fields are in their correct relation. Now you have to look at each field to see whether you can derive it from other information. If you can, the derived information should not be stored in your database. When we speak of "derived" in this instance, we're referring to information that you can mathematically derive—counts, totals, averages, and the like. Currently, you are storing the number of employees (*Num of Employees*) in the *Department* relation. Can you derive that information from other information? The answer is yes—all you have to do is count the number of occurrences of each department number in the *Employee* relation. So, you should not store *Num of Employees* in your database (anywhere).

Once you've completed step 3, you've completely and correctly defined the structure of your database and identified the information each relation should contain. Figure A.7 shows your database and the information in each relation. Notice that we have removed *Department Name* from the *Employee* relation (following the second rule of normalization) and that we have removed *Num of Employees* from the *Department* relation (following the third rule of normalization).

Figure A.7

The Correct Structure of Your Employee Database

EMPLOYEE RELATION		
Employee ID	Name	Department Num
1234	Jones	43
2345	Smith	15
5468	Joslin	43
9087	Mills	43
8798	Jones	69

JOB RELATION		DEPARTMENT RELATION	
Job Number	Job Name	Department Num	Department Name
14	Acct	15	Commercial
23	Sales	43	Residential
39	Maint	69	Non-profit
46	Admin		

JOB ASSIGNMENT RELATION		
Employee ID	Job Number	Hours
1234	14	4
1234	23	4
2345	14	8
5468	23	6
5468	46	2
9087	23	5
9087	14	3
8798	39	8

## ON YOUR OWN

### CREATING THE FINAL STRUCTURE FOR YOUR SCHOOL

Now it's time for you to fly solo and try a task by yourself. If your group successfully completed the previous Team Work project, you should have a relational database model with five relations: *Seminar*, *Seminar Section*, *Student*, *Qualified Teacher*, and *Seminar Section Class Roll*. The first four you identified early on. The last is the result of eliminating the many-to-many relationship between *Seminar Section* and *Student*. We named this intersection relation *Seminar Section Class Roll*. We did so because it represents a list of students enrolling in all of the sections. If you group those students by section, then you get class rolls.

Your task is to complete each table below for each relation. That is, you must first fill in the column head-

ings by identifying what information belongs in each relation. Be sure to follow steps 2 and 3 of normalization. Then, we encourage you to add in some actual data entries.

A word of caution. You could easily identify hundreds of pieces of information that need to be present in this database. For *Qualified Teacher* for example, you could include birth date, rank, office hours, phone number, e-mail address, office location, employment starting date, and many more. Here, simply identify no more than five key pieces of information for each relation. And, by all means, identify the primary and foreign keys for each relation.

#### Seminar Relation


#### Seminar Section Relation

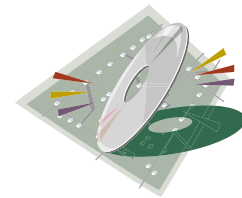

#### Student Relation


#### Qualified Teacher Relation


#### Seminar Section Class Roll Relation


## STEP 4: USING A DATA DEFINITION LANGUAGE TO CREATE YOUR DATABASE

The final step in developing a relational database is to take the structure you created in steps 1 to 3 and use a data definition language to actually create the relation. Data definition languages are found within a database management system. A **database management system (DBMS)** helps you specify the logical organization for a database and access and use the information within the database. This is the point at which we'll end this Extended Learning Module. But you shouldn't stop learning. We've created Skills Module 2 (on the CD-ROM accompanying this text) that takes you through the process of using the data definition language in Access to create the database we just designed. Keep learning.



Skills Module 2  
Database Design—  
Access

## Summary: Student Learning Outcomes Revisited

1. **Identify how databases and spreadsheets are both similar and different.** Databases and spreadsheets are similar in that they both store information in two-dimensional files. They are different in one key aspect—physical versus logical. Spreadsheets require that you know the physical location of information, by row and column. Databases, on the other hand, require that you know logically what information you want. For example, in a database environment you could easily request total sales for Able Electronics and you would receive that information. In a spreadsheet, you would have to know the physical location—by row and column—of that information.
2. **List and describe the four steps in designing and building a relational database.** The four steps in designing and building a relational database include:
  1. Defining entity classes and primary keys
  2. Defining relationships among entity classes
  3. Defining information (fields) for each relation
  4. Using a data definition language to create your database
3. **Define the concepts of entity class, instance, primary key, and foreign key.** An **entity class** is a concept—typically people, places, or things—about which you wish to store information and that you can identify with a unique key (called a primary key). A **primary key** is a field (or group of fields in some cases) that uniquely describes each record. Within the context of database design, we often refer to a record as an instance. An **instance** is an occurrence of an entity class that can be uniquely described. To provide logical relationships among various entity classes, you use **foreign keys**—primary keys of a file (relation) that also appear in another file (relation).
4. **Given a small operating environment, build an entity-relationship (E-R) diagram.** Building an E-R diagram starts with knowing and understanding the business rules that govern the situation. These rules will help you identify entity classes, primary keys, and relationships. You then follow the process of normalization, eliminating many-to-many relationships, ensuring that each field is in the correct relation, and removing any derived fields.
5. **List and describe the steps in normalization.** **Normalization** is the process of ensuring that a relational database structure can be implemented as a series of two-dimensional tables. The normalization steps include:
  1. Eliminate repeating groups or many-to-many relationships.
  2. Ensure that each field in a relation depends only on the primary key for that relation.
  3. Remove all derived fields from the relations.
6. **Describe the process of creating an intersection relation to remove a many-to-many relationship.** To create an intersection relation to remove a many-to-many relationship, follow these seven steps:
  1. Draw the part of the E-R diagram that contains a many-to-many relationship.
  2. Create a new E-R diagram with the original two relations on each end and a new one (the intersection relation) in the middle.

3. Create a meaningful name for the intersection relation.
4. Move the minimum cardinality appearing next to the left relation to just to the right of the intersection relation.
5. Move the minimum cardinality appearing next to the right relation to just to the left of the intersection relation.
6. The maximum cardinality on both sides of the intersection relation will always be “many.”
7. As a general rule, the new minimum and maximum cardinalities for the two original relations will be one and one.

## Key Terms and Concepts

---

composite primary key, 122

composite relation, 122

database, 116

database management system (DBMS), 129

entity class, 117

entity-relationship (E-R) diagram, 119

foreign key, 122

instance, 117

intersection relation (composite relation), 122

normalization, 122

primary key, 117

relational database, 116

## Short-Answer Questions

---

1. How are relational databases and spreadsheets both similar and different?
2. What are the four steps in designing and building a relational database?
3. What are some examples of entity classes at your school?
4. What is the role of a primary key?
5. What is the relationship between an occurrence and a record?
6. What is an entity-relationship (E-R) diagram?
7. What are the five basic symbols found in an E-R diagram?
8. How do business rules help you define minimum and maximum cardinality?
9. What is normalization?
10. What are the three major rules of normalization?
11. What is an intersection relation? Why is it important in designing a relational database?
12. Why must you remove derived information from a database?
13. What is a database management system (DBMS)?

## Assignments and Exercises

---

1. **DEFINING ENTITY CLASSES FOR THE MUSIC INDUSTRY.** The music industry tracks and uses all sorts of information related to numerous entity classes. Find a music CD and carefully review the entire contents of the jacket. List as many entity classes as you can find (for just that CD). Now, go to a music store and pick out a CD for a completely different music genre and read its jacket. Did you find any new entity classes? If so, what are they?
2. **DEFINING BUSINESS RULES FOR A VIDEO RENTAL STORE.** Think about how your local video rental store works. There are many customers, renting many videos, and many videos sit on the shelves unrented.

Customers can rent many videos at one time. And some videos are so popular that the video rental store keeps many copies. Write down all the various business rules that define how a video rental store works with respect to entity classes and their relationships.

3. **CREATING AN E-R DIAGRAM FOR A VIDEO RENTAL STORE.** After completing assignment 2 above, draw the initial E-R diagram on the basis of the rules you defined. Don't worry about going through the process of normalization at this point. Simply identify the appropriate relationships among the entity classes and define the minimum and maximum cardinality of each relationship. By the way, how many many-to-many relationships did you define?
4. **ELIMINATING A MANY-TO-MANY RELATIONSHIP.** Consider the following situation. At a small auto parts store, customers can buy many parts. And the same part can be bought by many different customers. That's an example of a many-to-many relationship. How would you eliminate it? What would you call the intersection relation? This one is particularly tough—you'll have to actually create two intersection relations to model this correctly.
5. **ELIMINATING A DERIVED FIELD.** If you think about it for a moment, your GPA is a derived field—it's a mathematical function of the grade points you've accumulated divided by the credits you've taken. So GPA should not be stored in your school's database. Given that it is not present, in what relations would the information be stored from which you could derive your GPA? What process would a DBMS have to go through to accurately calculate your GPA?
6. **DEFINING THE CARDINALITY AMONG TWO ENTITY CLASSES.** Consider the two entity classes of Student and Advisor at your school. How would you build an E-R diagram to show the relationship between these two entity classes? What is the minimum and maximum cardinality of the relationship?
7. **MODELLING YOUR SCHOOL'S BOOKSTORE.** Your school's bookstore maintains a vast inventory of books (and let's limit this exercise to just books). Consider that your school only orders new books from publishers. Books have a specific category (e.g., math, English, etc.). And books have at least one author and may have several. Focusing just on books, book categories, publishers, and authors, create a normalized relational database that would track key pieces of information. For each relation, define the primary key, any foreign keys, and a few pieces of important information. Did you find any many-to-many relationships? If so, how did you eliminate them?
8. **MAKING SOME CHANGES TO YOUR SCHOOL'S OFFERING OF WEEKEND SEMINARS.** Throughout this Module, you've been creating the correct structure for a database that will support your school's offering of weekend seminars on the Internet and Web. Now, you need to make a few changes. Implement the following business rules: (1) each student must take at least one seminar, (2) each seminar section must be assigned to a building and a room (some rooms won't have any seminar sections scheduled in them), and (3) each qualified teacher is assigned to only one academic department (and each academic department must have at least one qualified teacher in it). What does the new E-R diagram look like?
9. **MAKING SOME CHANGES TO YOUR EMPLOYEE DATABASE.** Throughout this module, you followed along with us as we created the correct structure for a database that will help you manage your employees, their departments, and their job assignments. Implement the following business rules: (1) each department must have one manager who is also an employee (some employees might not be managers) and (2) each employee has at least one of the following skills—team building, project management, and/or follow-up. What does the new E-R diagram look like?