Answers to Review Questions

1. **What two conditions must be met before an entity can be classified as a weak entity? Give an example of a weak entity.**

   To be classified as a weak entity, ***two conditions must be met***:
   1. The entity must be existence-dependent on its parent entity.
   2. The entity must inherit at least part of its primary key from its parent entity.

   For example, the (strong) relationship depicted in the text's Figure 4.10 shows a weak CLASS entity:
   1. CLASS is clearly existence-dependent on COURSE. (You can't have a database class unless a database course exists.)
   2. The CLASS entity's PK is defined through the combination of CLASS_SECTION and CRS_CODE. The CRS_CODE attribute is also the PK of COURSE.

   The conditions that define a weak entity are the same as those for a strong relationship between an entity and its parent. In short, the existence of a weak entity produces a strong relationship. And if the entity is strong, its relationship to the other entity is weak. (Note the solid relationship line in the text's Figure 4.10.)

   Keep in mind that whether or not an entity is weak usually depends on the database designer's decisions. For instance, if the database designer had decided to use a single-attribute as shown in the text's Figure 4.8, the CLASS entity would be strong. (The CLASS entity's PK is CLASS_CODE, which is not derived from the COURSE entity.) In this case, the relationship between COURSE and CLASS is weak. (Note the dashed relationship line in the text's Figure 4.8.) **However, regardless of how the designer classifies the relationship – weak or strong – CLASS is always existence-dependent on COURSE**.

2. **What is a strong (or identifying) relationship, and how is it depicted in a Crow's Foot ERD?**

   A strong relationship exists when en entity is existence-dependent on another entity and inherits at least part of its primary key from that entity. The Visio Professional software shows the strong relationship as a solid line. In other words, a strong relationship exists when a weak entity is related to its parent entity. (Note the discussion in question 1.)

3. Given the business rule "an employee may have many degrees," discuss its effect on attributes, entities, and relationships. (*Hint*: Remember what a multivalued attribute is and how it might be implemented.)

Suppose that an employee has the following degrees: BA, BS, and MBA. These degrees could be stored in a single string as a multivalued attribute named EMP_DEGREE in an EMPLOYEE table such as the one shown next:

EMP_NUM	EMP_LNAME	EMP_DEGREE
123	Carter	AA, BBA
124	O'Shanski	BBA, MBA, Ph.D.
125	Jones	AS
126	Ortez	BS, MS

Although the preceding solution has no obvious design flaws, it is likely to yield reporting problems. For example, suppose you want to get a count for all employees who have BBA degrees. You could, of course, do an "in-string" search to find all of the BBA values within the EMP_DEGREE strings. But such a solution is cumbersome from a reporting point of view. Query simplicity is a valuable thing to application developers – and to end users who like maximum query execution speeds. Database designers ought to pay some attention to the competing database interests that exist in the data environment.

One – *very* poor – solution is to create a field for each expected value. This "solution is shown next:

EMP_NUM	EMP_LNAME	EMP_DEGREE1	EMP_DEGREE2	EMP_DEGREE3
123	Carter	AA	BBA	
124	O'Shanski	BBA	MBA	Ph.D.
125	Jones	AS		
126	Ortez	BS	MS	

This "solution yields nulls for all employees who have fewer than three degrees. And if even one employee earns a fourth degree, the table structure must be altered to accommodate the new data value. (One piece of evidence of poor design is the need to alter table structures in response to the need to add data of an existing type.) In addition, the query simplicity is not enhanced by the fact that any degree can be listed in any column. For example, a BA degree might be listed in the second column, after an "associate of arts (AA) degree has been entered in EMP_DEGREE1. One might simplify the query environment by creating a set of attributes that define the data entry, thus producing the following results:

EMP_NUM	EMP_LNAME	EMP_AA	EMP_AS	EMP_BA	EMP_BS	EMP_BBA	EMP_MS	EMP_MBA	EMP_PhD
123	Carter	X				X			
124	O'Shanski					X		X	X
125	Jones		X						
126	Ortez				X		X		

This “solution” clearly proliferates the nulls at an ever-increasing pace.

The only reasonable solution is to create a new DEGREE entity that stores each degree in a separate record, this producing the following tables. (There is a 1:M relationship between EMPLOYEE and DEGREE. Note that the EMP_NUM can occur more than once in the DEGREE table. The DEGREE table’s PK is EMP_NUM + DEGREE_CODE. This solution also makes it possible to record the date on which the degree was earned, the institution from which it was earned, and so on.

Table name: EMPLOYEE

EMP_NUM	EMP_LNAME
123	Carter
124	O’Shanski
125	Jones
126	Ortez

Table name: DEGREE

EMP_NUM	DEGREE_CODE	DEGREE_DATE	DEGREE_PLACE
123	AA	May-1999	Lake Sumter CC
123	BBA	Aug-2004	U. of Georgia
124	BBA	Dec-1990	U. of Toledo
124	MBA	May-2001	U. of Michigan
124	Ph.D.	Dec-2005	U. of Tennessee
125	AS	Aug-2002	Valdosta State
126	BS	Dec-1989	U. of Missouri
126	MS	May-2002	U. of Florida

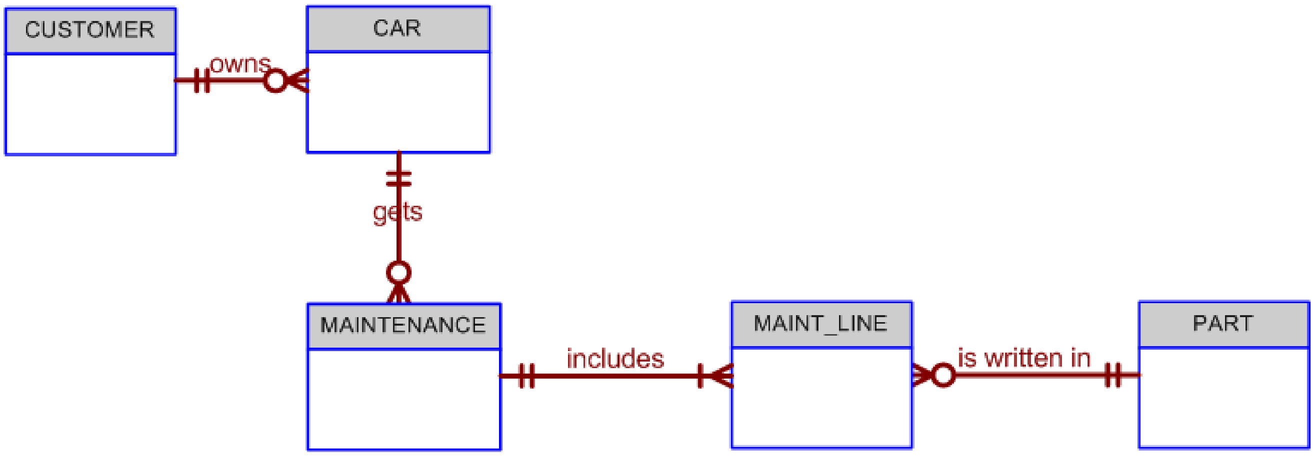
Note that this solution leaves no nulls, produces a simple query environment, and makes it unnecessary to alter the table structure when employees earn additional degrees. (You can make the environment even more flexible by naming the new entity QUALIFICATION, thus making it possible to store degrees, certifications, and other useful data that define an employee’s qualifications.)

4. What is a composite entity, and when is it used?

A composite entity is generally used to transform M:N relationships into 1:M relationships. (Review the discussion that accompanied Figures IM4.3 through IM4.5.) A composite entity, also known as a bridge entity, is one that has a primary key composed of multiple attributes. The PK attributes are inherited from the entities that it relates to one another.

5. **Suppose you are working within the framework of the conceptual model in Figure Q4.5.**

Figure Q4.5 The Conceptual Model for Question 5



Given the conceptual model in Figure Q4.5:

a. Write the business rules that are reflected in it.

Even a simple ERD such as the one shown in Figure Q4.5 is based on many business rules. Make sure that each business rule is written on a separate line and that all of its details are spelled out. In this case, the business rules are derived from the ERD in a "reverse-engineering" procedure designed to document the database design. In a real world database design situation, the ERD is generated on the basis of business rules that are written before the first entity box is drawn. (Remember that the business rules are derived from a carefully and precisely written description of operations.)

Given the ERD shown in Figure Q4.5, you can identify the following business rules:

1. A customer can own many cars.
2. Some customers do not own cars.
3. A car is owned by one and only one customer.
4. A car may generate one or more maintenance records.
5. Each maintenance record is generated by one and only one car.
6. Some cars have not (yet) generated a maintenance procedure.
7. Each maintenance procedure can use many parts.
   (Comment: A maintenance procedure may include multiple maintenance actions, each one of which may or may not use parts. For example, 10,000-mile check may include the installation of a new oil filter and a new air filter. But tightening an alternator belt does not require a part.)
8. A part may be used in many maintenance records.

(Comment: Each time an oil change is made, an oil filter is used. Therefore, many oil filters may be used during some period of time. Naturally, you are not using the *same* oil filter each time – but the part classified as "oil filter" shows up in many maintenance records as time passes.)

Note that the apparent M:N relationship between MAINTENANCE and PART has been resolved through the use of the composite entity named MAINT_LINE. The MAINT_LINE entity ensures that the M:N relationship between MAINTENANCE and PART has been broken up to produce the two 1:M relationships shown in business rules 9 and 10.

9. Each maintenance procedure generates one or more maintenance lines.
10. Each part may appear in many maintenance lines. (Review the comment in business rule 8.)

As you review the business rules 9 and 10, use the following two tables to show some sample data entries. For example, take a look at the (simplified) contents of the following MAINTENANCE and LINE tables and note that the MAINT_NUM 10001 occurs three times in the LINE table:

Sample MAINTENANCE Table Data

MAINT_NUM	MAINT_DATE
10001	15-Mar-2014
10002	15-Mar-2014
10003	16-Mar-2014

Sample LINE Table Data

MAINT_NUM	LINE_NUM	LINE_DESCRIPTION	LINE_PART	LINE_UNITS
10001	1	Replace fuel filter	FF-015	1
10001	2	Replace air filter	AF-1187	1
10001	3	Tighten alternator belt	NA	0
10002	1	Replace taillight bulbs	BU-2145	2
10003	1	Replace oil filter	OF-2113	1
10003	2	Replace air filter	AF-1187	1

b. Identify all of the cardinalities.

The Visio-generated Crow's Foot ERD, shown in Figure Q4.5, does not show cardinalities directly. Instead, the cardinalities are implied through the Crow's Foot symbols. You might write the cardinality (0,N) next to the MAINT_LINE entity in its relationship with the PART entity to indicate that a part might occur "N" times in the maintenance line entity or that it might never show up in the maintenance line entity. The latter case would occur if a given part has never been used in maintenance.

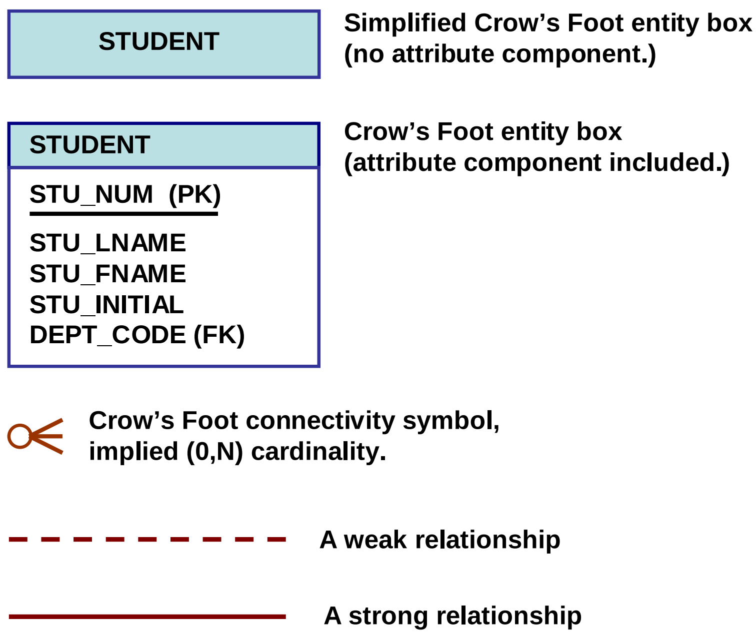
6. What is a recursive relationship? Given an example.

A recursive relationship exists when an entity is related to itself. For example, a COURSE may be a prerequisite to a COURSE. (See Section 4.1.10, "Recursive Relationships," for additional examples.

7. **How would you (graphically) identify each of the following ERM components in a Crow's Foot model?**

The answers to questions (a) through (d) are illustrated with the help of Figure Q4.7.

FIGURE Q4.7 Crow's Foot ERM Components



a. **an entity**

An entity is represented by a rectangle containing the entity name. (Remember that, in ER modeling, the word "entity" actually refers to the entity *set*.)

The Crow's Foot ERD – as represented in Visio Professional – does not distinguish among the various entity types such as weak entities and composite entities. Instead, the Crow's Foot ERD uses *relationship* types – strong or weak – to indicate the nature of the relationships between entities. For example, a strong relationship indicates the existence of a weak entity.

A composite entity is defined by the fact that at least one of the PK attributes is also a foreign key. Therefore, the Visio Crow's Foot ERD's composite and weak entities are not differentiated – whether or not an entity is weak or composite depends on the definition of the business rule(s) that describe the relationships. In any case, two conditions must be met before an entity can be classified as weak:
1. The entity must be existence-dependent on its parent entity
2. The entity must inherit at least part of its primary key from its parent entity.

b. the cardinality (0,N)

Cardinalities are implied through the use of Crow's Foot symbols. For example, note the implied (0,N) cardinality in Figure Q4.7.

c. a weak relationship

A weak relationship exists when the PK of the related entity does not contain at least one of the PK attributes of the parent entity. For example, if the PK of a COURSE entity is CRS_CODE and the PK of the related CLASS entity is CLASS_CODE, the relationship between COURSE and CLASS is weak. (Note that the CLASS PK does not include the CRS_CODE attribute.) A weak relationship is indicated by a dashed line in the (Visio) ERD.

d. a strong relationship

A strong relationship exists when the PK of the related entity contains at least one of the PK attributes of the parent entity. For example, if the PK of a COURSE entity is CRS_CODE and the PK of the related CLASS entity is CRS_CODE + CLASS_SECTION, the relationship between COURSE and CLASS is strong. (Note that the CLASS PK includes the CRS_CODE attribute.) A strong relationship is indicated by a solid line in the (Visio) ERD.

8. Discuss the difference between a composite key and a composite attribute. How would each be indicated in an ERD?

A composite key is one that consists of more than one attribute. If the ER diagram contains the attribute names for each of its entities, a composite key is indicated in the ER diagram by the fact that more than one attribute name is underlined to indicate its participation in the primary key.

A composite attribute is one that can be subdivided to yield *meaningful* attributes for each of its components. For example, the composite attribute CUS_NAME can be subdivided to yield the CUS_FNAME, CUS_INITIAL, and CUS_LNAME attributes. There is no ER convention that enables us to indicate that an attribute is a composite attribute.

9. What two courses of action are available to a designer when encountering a multivalued attribute?

The discussion that accompanies the answer to question 3 is valid as an answer to this question.

10. What is a derived attribute? Give an example.

A derived attribute is an attribute whose value is calculated (derived) from other attributes. The derived attribute need not be physically stored within the database; instead, it can be derived by using an algorithm. For example, an employee's age, EMP_AGE, may be found by computing the integer value of the difference between the current date and the EMP_DOB. If you use MS Access, you would use INT((DATE() – EMP_DOB)/365).
Similarly, a sales clerk's total gross pay may be computed by adding a computed sales commission to base pay. For instance, if the sales clerk's commission is 1%, the gross pay may be computed by
EMP_GROSSPAY = INV_SALES*1.01 + EMP_BASEPAY
Or the invoice line item amount may be calculated by
LINE_TOTAL = LINE_UNITS*PROD_PRICE

11. How is a relationship between entities indicated in an ERD? Give an example, using the Crow's Foot notation.

Use Figure Q4.7 as the basis for your answer. Note the distinction between the dashed and solid relationship lines, then tie this distinction to the answers to question 7c and 7d.

12. Discuss two ways in which the 1:M relationship between COURSE and CLASS can be implemented. (*Hint*: Think about relationship strength.)

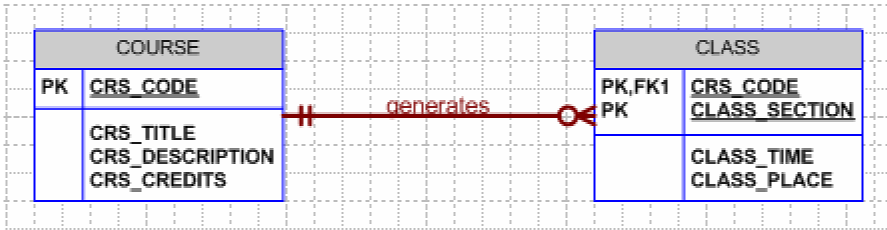
Note the discussion about weak and strong entities in questions 7c and 7d. Then follow up with this discussion:

The relationship is implemented as *strong* when the CLASS entity's PK contains the COURSE entity's PK. For example,

COURSE(**<u>CRS_CODE</u>**, CRS_TITLE, CRS_DESCRIPTION, CRS_CREDITS)
CLASS(**<u>CRS_CODE</u>**, **<u>CLASS_SECTION</u>**, CLASS_TIME, CLASS_PLACE)

Note that the CLASS entity's PK is CRS_CODE + CLASS_SECTION – and that the CRS_CODE component of this PK has been "borrowed" from the COURSE entity. (Because CLASS is existence-dependent on COURSE and uses a PK component from its parent (COURSE) entity, the CLASS entity is weak in this strong relationship between COURSE and CLASS. The Visio Crow's Foot ERD shows a strong relationship as a solid line. (See Figure Q4.12a.) Visio refers to a strong relationship as an *identifying* relationship.

Figure Q4.12a Strong COURSE and CLASS Relationship


COURSE
PK
CRS_CODE
CRS_TITLE
CRS_DESCRIPTION
CRS_CREDITS
generates
CLASS
PK,FK1
PK
CRS_CODE
CLASS_SECTION
CLASS_TIME
CLASS_PLACE

Sample data are shown next:

Table name: COURSE

CRS_CODE	CRS_TITLE	CRS-DESCRIPTION	CRS_CREDITS
ACCT-211	Basic Accounting	An introduction to accounting. Required of all business majors.	3
CIS-380	Database Techniques I	Database design and implementation issues. Uses CASE tools to generate designs that are then implemented in a major database management system.	3
CIS-490	Database Techniques II	The second half of CIS-380. Basic Web database application development and management issues.	4

Table name: CLASS

CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_PLACE
ACCT-211	1	8:00 a.m. – 9:30 a.m. T-Th.	Business 325
ACCT-211	2	8:00 a.m. – 8:50 a.m. MWF	Business 325
ACCT-211	3	8:00 a.m. – 8:50 a.m. MWF	Business 402
CIS-380	1	11:00 a.m. – 11:50 a.m. MWF	Business 415
CIS-380	2	3:00 p.m. – 3:50 a.m. MWF	Business 398
CIS-490	1	1:00 p.m. – 3:00 p.m. MW	Business 398
CIS-490	2	6:00 p.m. – 10:00 p.m. Th.	Business 398

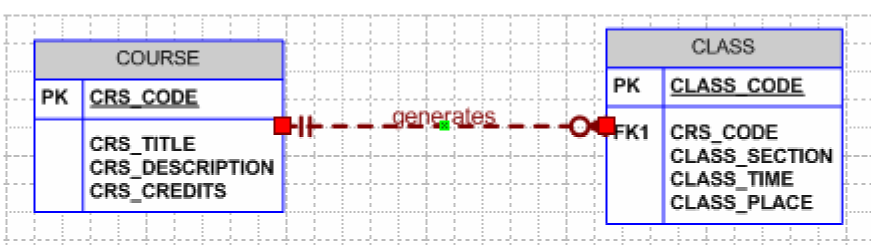
The relationship is implemented as *weak* when the CLASS entity's PK does not contain the COURSE entity's PK. For example,

COURSE(**<u>CRS_CODE</u>**, CRS_TITLE, CRS_DESCRIPTION, CRS_CREDITS)
CLASS(**<u>CLASS_CODE</u>**, CRS_CODE, CLASS_SECTION, CLASS_TIME, CLASS_PLACE)

(Note that CRS_CODE is no longer part of the CLASS PK, but that it continues to serve as the FK to COURSE.)

The Visio Crow's Foot ERD shows a weak relationship as a dashed line. (See Figure Q4.12b.) Visio refers to a weak relationship as a *non-identifying* relationship.

Figure Q4.12b Weak COURSE and CLASS Relationship



Given the weak relationship depicted in Figure Q4.13b, the CLASS table contents would look like this:

Table name: CLASS

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_PLACE
21151	ACCT-211	1	8:00 a.m. – 9:30 a.m. T-Th.	Business 325
21152	ACCT-211	2	8:00 a.m. – 8:50 a.m. MWF	Business 325
21153	ACCT-211	3	8:00 a.m. – 8:50 a.m. MWF	Business 402
38041	CIS-380	1	11:00 a.m. – 11:50 a.m. MWF	Business 415
38042	CIS-380	2	3:00 p.m. – 3:50 a.m. MWF	Business 398
49041	CIS-490	1	1:00 p.m. – 3:00 p.m. MW	Business 398
49042	CIS-490	2	6:00 p.m. – 10:00 p.m. Th.	Business 398

The advantage of the second CLASS entity version is that its PK can be referenced easily as a FK in another related entity such as ENROLL. Using a single-attribute PK makes implementation easier. This is especially true when the entity represents the "1" side in one *or more* relationships. **In general, it is advisable to avoid composite PKs whenever it is practical to do so**.

13. How is a composite entity represented in an ERD, and what is its function? Illustrate the Crow's Foot model.

The label "composite" is based on the fact that the composite entity contains at least the primary key attributes of each of the entities that are connected by it. The composite entity is an important component of the ER model because relational database models should not contain M:N relationships – and the composite entity can be used to break up such relationships into 1:M relationships.

Remind students to heed the advice given in the answer to the previous question. That is, **avoid composite PKs whenever it is practical to do so**. Note that the CLASS entity structure shown in Figure Q4.12b is far better than that of the CLASS entity structure shown in Figure Q4.12a. Suppose, for example, that you want to design a class enrollment entity to serve as the "bridge" between STUDENT and CLASS in the M:N relationship defined by these two business rules:

- A student can take many classes.
- Each class can be taken by many students.

In this case, you could create a (composite) entity named ENROLL to link CLASS and STUDENT, using these structures:

STUDENT(**STU_NUM**, STU_LNAME …………..)
ENROLL(**STU_NUM**, **CLASS_NUM**, ENROLL_GRADE ………)
CLASS(**CLASS_CODE**, CRS_CODE, CLASS_SECTION, CLASS_TIME, CLASS_PLACE)

Your students might argue that a composite PK in ENROLL does no harm, since it is not likely to be related to another entity in the typical academic database setting. Although that is a good observation, you would run into a problem in the event that might trigger a required relationship between ENROLL and another entity. In any case, you may simplify the creation of future relationships if you create an "artificial" single-attribute PK such as ENROLL_NUM, while maintaining the STU_NUM and CLASS_NUM as FK attributes. In other words:

ENROLL(**ENROLL_NUM**, STU_NUM, CLASS_NUM, ENROLL_GRADE ………)

The ENROLL_NUM attribute values can easily be generated through the proper use of SQL code or application software, thus eliminating the need for data entry by humans.

The use of composite vs. single-attribute PKs is worth discussing. Composite PKs are frequently encountered in composite entities and your students will see that MS Visio will generate composite PKs automatically when you classify a relationship as *strong*. Composite PKs are not "wrong" is any sense, but minimizing their use does make the implementation of multiple relationships simple … Simple is generally a good thing!

NOTE

Because composite entities are frequently encountered in the real world environment, we continue to use them in the text and in many of our exercises and examples. However, the words of caution about their use should be repeated from time to time and you might ask your students to convert such composite entities.

Let's examine another example of the use of composite entities. Suppose that a trucking company keeps a log of its trucking operations to keep track of its driver/truck assignments. The company may assign any given truck to any given driver many times and, as time passes, each driver may be assigned to drive many of the company's trucks. Since this M:N relationship should not be implemented, we create the composite entity named LOG whose attributes are defined by the end-user information requirements. In this case, it may be useful to include LOG_DATE, TRUCK_NUM, DRIVER_NUM, LOG_TIME_OUT, and LOG_TIME_IN.

Note that the LOG's TRUCK_NUM and DRIVER_NUM attributes are the driver LOG's foreign keys. The TRUCK_NUM and DRIVER_NUM attribute values provide the bridge between the TRUCK and DRIVER, respectively. In other words, to form a proper bridge between TRUCK and DRIVER, the composite LOG entity must contain at least the primary keys of the entities connected by it.

You might think that the combination of the composite entity's foreign keys may be designated to be the composite entity's primary key. However, this combination will not produce unique values over time. For example, the same driver may drive a given truck on different dates. Adding the date to the PK attributes will solve that problem. But we still have a non-unique outcome when the same driver drives a given truck twice on the same date. Adding a time attribute will finally create a unique set of PK attribute values – but the PK is now composed of four attributes: TRUCK_NUM, DRIVER_NUM, LOG_DATE, and LOG_TIME_OUT. (The combination of these attributes yields a unique outcome, because the same driver cannot check out two trucks at the same time on a given date.)

Because multi-attribute PKs may be difficult to manage, it is often advisable to create an "artificial" single-attribute PK, such as LOG_NUM, to uniquely identify each record in the LOG table. (Access users can define such an attribute to be an "autonumber" to ensure that the system will generate unique LOG_NUM values for each record.) Note that this solution produces a LOG table that contains two candidate keys: the designated primary key and the combination of foreign keys that *could* have served as the primary key.

While the preceding solution simplifies the PK definition, it does not prevent the creation of duplicate records that merely have a different LOG_NUM value. Note, for example, the first two records in the following table:

LOG_NUM	LOG_DATE	TRUCK_NUM	DRIVER_NUM	LOG_TIME_OUT	LOG_TIME_IN
10015	12-Mar-2014	322453	1215	07:18 a.m.	04:23 p.m.
10016	12-Mar-2014	322453	1215	07:18 a.m.	04:23 p.m.
10017	12-Mar-2014	545567	1298	08:12 a.m.	09:15 p.m.

To avoid such duplicate records, **you can create a unique index** on TRUCK_NUM + DRIVER_NUM + LOG_DATE + LOG_TIME_OUT.

Composite entities may be named to reflect their component entities. For example, an employee may have several insurance policies (life, dental, accident, health, etc.) and each insurance policy may be held by many employees. This M:N relationship is converted to a set of two 1:M relationships, by creating a composite entity named EMP_INS. The EMP_INS entity must contain *at least* the primary key components of each of the two entities connected by it. How many additional attributes are kept in the composite entity depends on the end-user information requirements.

14. What three (often conflicting) database requirements must be addressed in database design?

Database design must reconcile the following requirements:

a. Design elegance requires that the design must adhere to design rules concerning nulls, derived attributes, redundancies, relationship types, and so on.
b. Information requirements are dictated by the end users
c. Operational (transaction) speed requirements are also dictated by the end users.

Clearly, an elegant database design that fails to address end user information requirements or one that forms the basis for an implementation whose use progresses at a snail's pace has little practical use.

15. Briefly, but precisely, explain the difference between single-valued attributes and simple attributes. Give an example of each.

A single -valued attribute is one that can have only one value. For example, a person has only one first name and only one social security number.

A simple attribute is one that cannot be decomposed into its component pieces. For example, a person's sex is classified as either M or F and there is no reasonable way to decompose M or F. Similarly, a person's first name cannot be decomposed into meaningful components. (In contrast, if a phone number includes the area code, it can be decomposed into the area code and the phone number. And a person's name may be decomposed into a first name, an initial, and a last name.)

Single-valued attributes are not necessarily simple. For example, an inventory code

HWPRIJ23145 may refer to a classification scheme in which HW indicates Hardware, PR indicates Printer, IJ indicates Inkjet, and 23145 indicates an inventory control number. Therefore, HWPRIJ23145 may be decomposed into its component parts... even though it is single-valued. To facilitate product tracking, manufacturing serial codes must be single-valued, but they may not be simple. For instance, the product serial number TNP5S2M231109154321 might be decomposed this way:

TN = state = Tennessee
P5 = plant number 5
S2 = shift 2
M23 = machine 23
11 = month, i.e., November
09 = day
154321 = time on a 24-hour clock, i.e., 15:43:21, or 3:43 p.m. plus 21 seconds.

16. What are multivalued attributes, and how can they be handled within the database design?

The answer to question 3 is just as valid as an answer to this question. You can augment that discussion with the following discussion:

As the name implies, multi-valued attributes may have many values. For example, a person's education may include a high school diploma, a 2-year college associate degree, a four-year college degree, a Master's degree, a Doctoral degree, and various professional certifications such as a Certified Public Accounting certificate or a Certified Data Processing Certificate.

There are basically three ways to handle multi-valued attributes -- and two of those three ways are
bad:

1. Each of the possible outcomes is kept as a separate attribute within the table. This solution is undesirable for several reasons. First, the table would generate many nulls for those who had minimal educational attainments. Using the preceding example, a person with only a high school diploma would generate nulls for the 2-year college associate degree, the four-year college degree, the Master's degree, the Doctoral degree, and for each of the professional certifications. In addition, how many professional certification attributes should be maintained? If you store two professional certification attributes, you will generate a null for someone with only one professional certification and you'd generate two nulls for all persons without professional certifications. And suppose you have a person with five professional certifications? Would you create additional attributes, thus creating many more nulls in the table, or would you simply ignore the additional professional certifications, thereby losing information?

2. The educational attainments may be kept as a single, variable-length string or character field. This solution is undesirable because it becomes difficult to query the table. For example, even a simple question such as "how many employees have four-year college degrees?" requires string partitioning that is time-consuming at best. Of course, if there is no need to ever group employees by education, the variable-length string might be acceptable from a design point of view. However, as database designers we know that, sooner or later, information requirements are likely to grow, so the string storage is probably a bad idea from that perspective, too.

3. Finally, the most flexible way to deal with multi-valued attributes is to create a composite entity that links employees to education. By using the composite entity, there will never be a situation in which additional attributes must be created within the EMPLOYEE table to accommodate people with multiple certifications. In short, we eliminate the generation of nulls. In addition, we gain information flexibility because we can also store the details (date earned, place earned, etc.) for each of the educational attainments. The (simplified) structures might look like those in Figure Q4.16 A and B.

Figure Q4.16a The Ch04_Questions Database Tables

Database name: Ch04_Questions

Table name: EMPLOYEE

EMP_NUM	EMP_LNAME
215	Smith
216	Romero
217	Randall
218	Aarden

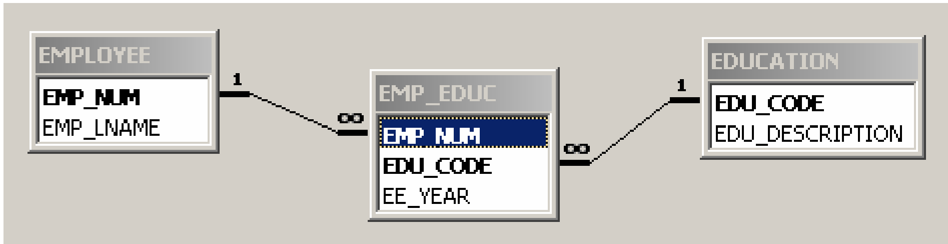
Table name: EMP_EDUC

EMP_NUM	EDU_CODE	EE_YEAR
215	B4	2002
215	HS	1991
215	JC	1994
216	B4	1989
216	CDP	2004
216	CNP	2002
216	HS	1985
217	HS	1992
217	JC	2000
218	B4	1990
218	CDP	2000
218	CNP	2005
218	HS	1984
218	M2	1995

Table name: EDUCATION

EDU_CODE	EDU_DESCRIPTION
B4	Bachelor's degree
CDP	Certified data Processing
CNP	Certified Network Professional
CPA	Certified Public Accountant
DR	Earned Doctorate degree
HS	Highschool diploma
JC	Junior College degree
M2	Master's degree

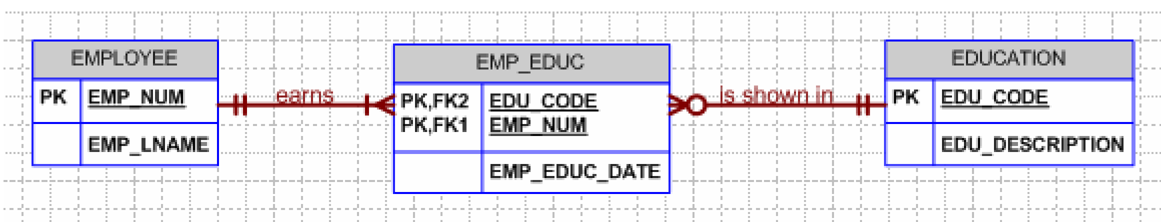
Figure Q4.16b The Ch04_Questions Relational Diagram



By looking at the structures shown in Figures Q4.16a and Q4.16b, we can tell that the employee named Romero earned a Bachelor's degree in 1989, a Certified Network Professional certification in 2002, and a Certified Data Processing certification in 2004. If Randall were to earn a Master's degree and a Certified Public Accountant certification later, we merely add another two records in the EMP_EDUC table. If additional educational attainments beyond those listed in the EDUCATION table are earned by any employee, all we need to do is add the appropriate record(s) to the EDUCATION table, then enter the employee's attainments in the EMP_EDUC table. There are no nulls, we have superb query capability, and we have flexibility. Not a bad set of design goals!

The database design on which Figures Q4.16a and Q4.16b are based is shown in Figure Q4.16c.

Figure Q4.16c The Crow's Foot ERD for the Ch04_Questions Database

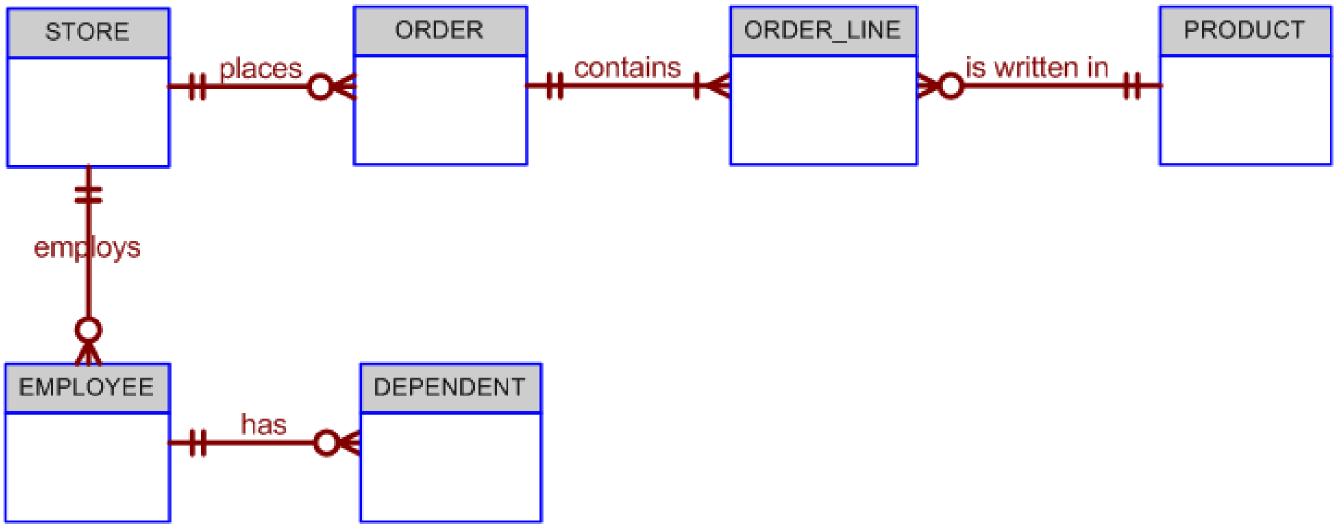


NOTE

Discuss with the students that the design in Figure Q4.16c shows that an employee must meet at least one educational requirement, because EMP_EDUC is not optional to EMPLOYEE. Thus each employee must appear at least once in the EMP_EDUC table. And, given this design, some of the educational attainments may not yet been earned by employees, because the design shows EMP_EDUC to be optional to EDUCATION. In other words, some of the EDUCATION records are not *necessarily* referenced by any employee. (In the original M:N relationship between EMPLOYEE and EDUCATION, EMPLOYEE must have been optional to EDUCATION.)

The final four questions are based on the ERD in Figure Q4.17.

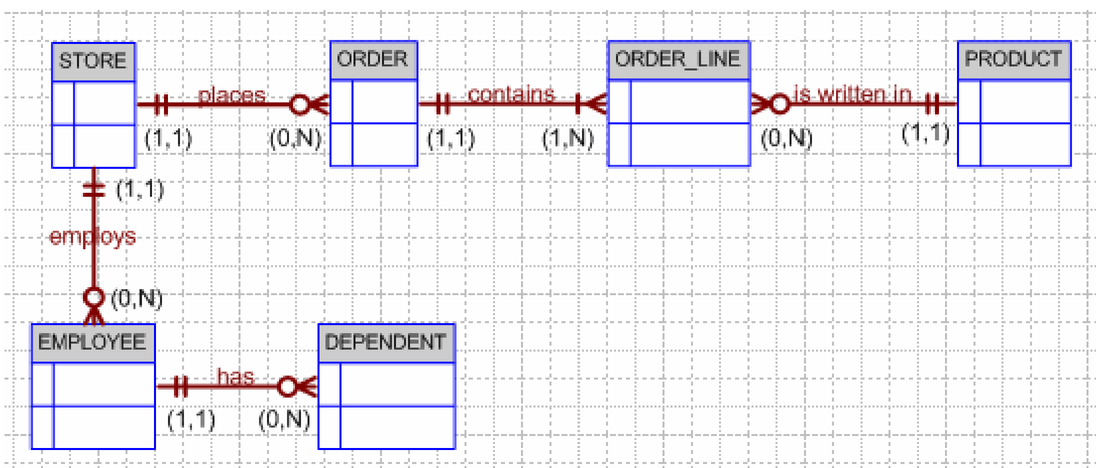
FIGURE Q4.17 The ERD For Questions 17–20



17. Write the ten cardinalities that are appropriate for this ERD.

The cardinalities are indicated in Figure Q4.17sol.

FIGURE Q4.17sol The Cardinalities



18. Write the business rules reflected in this ERD.

The following business rules are reflected in the ERD:

- A store may place many orders. (Note the use of "may" – which is reflected in the ORDER optionality.)
- An order must be placed by a store. (Note that STORE is mandatory to ORDER. In this ERD, the order environment apparently reflects a wholesale environment.)

- An order contains at least one order line. (Note that ORDER_LINE is mandatory to ORDER, and vice-versa.)
- Each order line is contained in one and only one order. (Discussion: Although a given item – such as a hammer – may be found in many orders, a *specific* hammer sold to a *specific* store is found in only one order.)
- Each order line has a specific product written in it.
- A product may be written in many orders. (Discussion: Many stores can order one or more specific products, but a product that is not in demand may never be sold to a store and will, therefore, not show up in any order line -- note that ORDER_LINE is optional to PRODUCT. Also, note that each order line may indicate more than one of a specific item. For example, the item may be "hammer" and the number sold may be 1 or 2, or 500. The ORDER_LINE entity would have at least the following attributes: ORDER_NUM, ORDLINE_NUM, PROD_CODE, ORDLINE_PRICE, ORDLINE_QUANTITY. The ORDER_LINE composite PK would be ORDER_NUM + ORDLINE_NUM. You might add the derived attribute ORDLINE_AMOUNT, which would be the result of multiplying ORDLINE_PRICE and ORDLINE_QUANTITY.)
- A store may employ many employees. (Discussion: A new store may not yet have any employees, yet the database may already include the new store information … location, type, and so on. If you made the EMPLOYEE entity mandatory to STORE, you would have to create an employee for that store before you had even hired one.)
- Each employee is employed by one (and only one) store.
- An employee may have one or more dependents. (Discussion: You cannot *require* an employee to have dependents, so DEPENDENT is optional to EMPLOYEE. Note the use of the word "may" in the relationship.)
- A dependent must be related to an employee. (Discussion: It makes no sense to keep track of dependents of people who are not even employees. Therefore, EMPLOYEE is mandatory to DEPENDENT.)

19. What two attributes must be contained in the composite entity between STORE and PRODUCT? Use proper terminology in your answer.

The composite entity must at least include the primary keys of the entities it references. The combination of these attributes may be designated to be the composite entity's (composite) primary key. Each of the (composite) primary key's attributes is a foreign key that references the entities for which the composite entity serves as a bridge.

As you discuss the model in Figure Q4.17sol, note that an order is represented by two entities, ORDER and ORDER_LINE. Note also that the STORE's 1:M relationship with ORDER and the ORDER's 1:M relationship with ORDER_LINE reflect the conceptual M:N relationship between STORE and PRODUCT. The original business rules probably read:

- A store can order many products
- A product can be ordered by many stores.

20. Describe precisely the composition of the DEPENDENT weak entity's primary key. Use proper terminology in your answer.

The DEPENDENT entity will have a composite PK that includes the EMPLOYEE entity's PK and one of its attributes. For example, if the EMPLOYEE entity's PK is EMP_NUM, the DEPENDENT entity's PK might be EMP_NUM + DEP_NUM.

21. The local city youth league needs a database system to help track children that sign up to play soccer. Data needs to be kept on each team and the children that will be playing on each team and their parents. Also, data needs to be kept on the coaches for each team. Draw the data model described below.

Entities required: Team, Player, Coach, and Parent.

Attributes required:

Team: Team ID number, Team name, and Team colors.

Player: Player ID number, Player first name, Player last name, and Player age.

Coach: Coach ID number, Coach first name, Coach last name, and Coach home phone number.

Parent: Parent ID number, Parent last name, Parent first name, Home phone number, and Home Address (Street, City, State, and ZIP Code).

The following relationships must be defined:

- **Team is related to Player.**
- **Team is related to Coach.**
- **Player is related to Parent.**

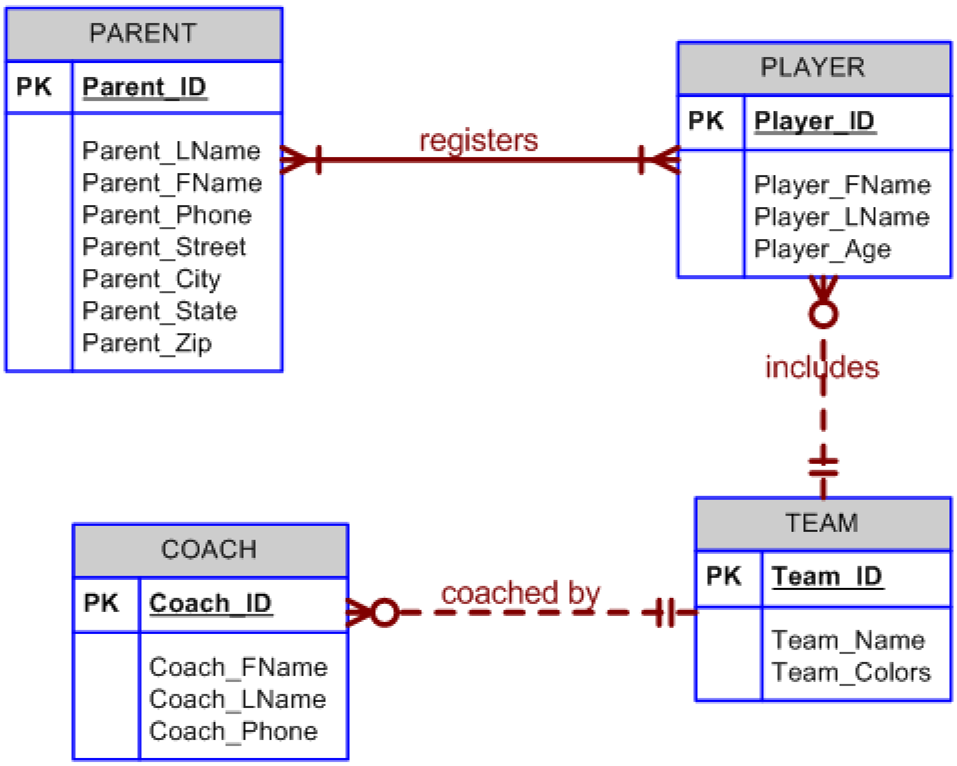
Connectivities and participations are defined as follows:

- **A Team may or may not have a Player.**
- **A Player must have a Team.**
- **A Team may have many Players.**
- **A Player has only one Team.**
- **A Team may or may not have a Coach.**
- **A Coach must have a Team.**
- **A Team may have many Coaches.**
- **A Coach has only one Team.**
- **A Player must have a Parent.**
- **A Parent must have a Player.**
- **A Player may have many Parents.**

- **A Parent may have many Players.**

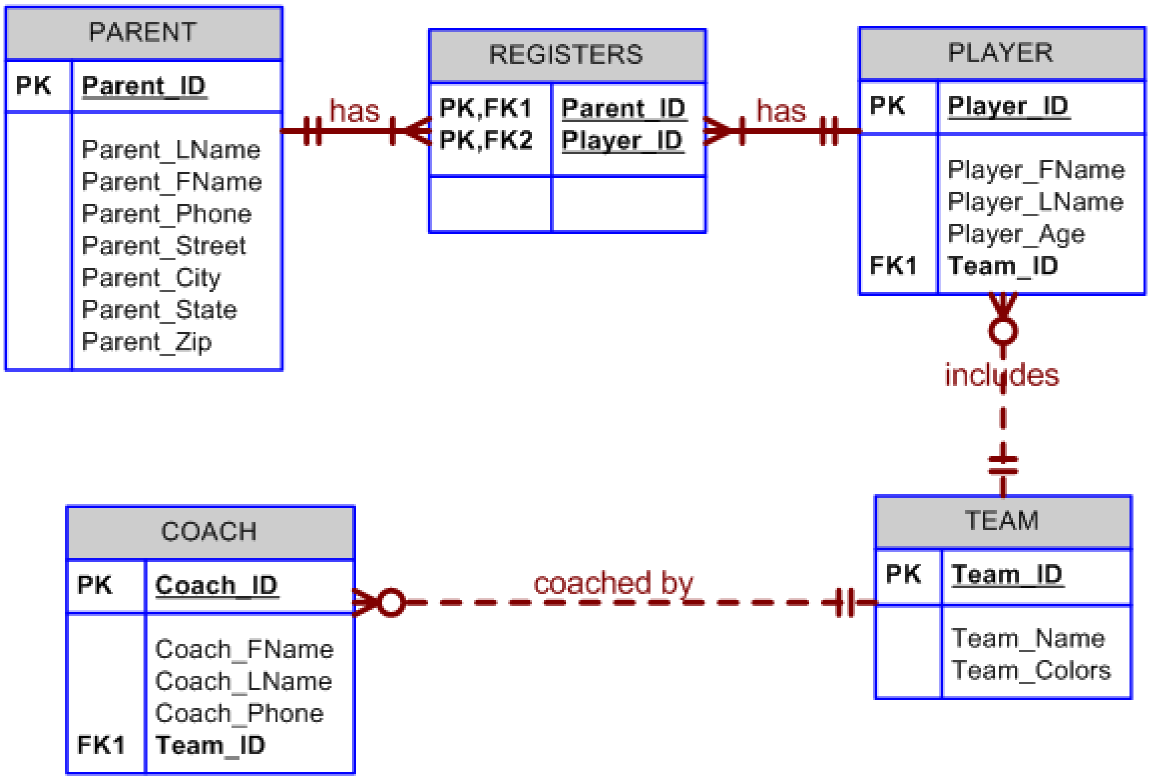
This is a great exercise in that it opens up possibilities for several discussion points. The conceptual ERD prior to placement of foreign keys and the resolution of the M:N relationship is shown in Figure Q4.21a.

FIGURE Q4.21a Conceptual ERD for Question 21



The most apparent issue that must be resolved is the M:N relationship. This is necessary so that foreign keys can be appropriately placed throughout the data model. The revised ERD with properly placed foreign keys is shown in Figure Q4.21b.

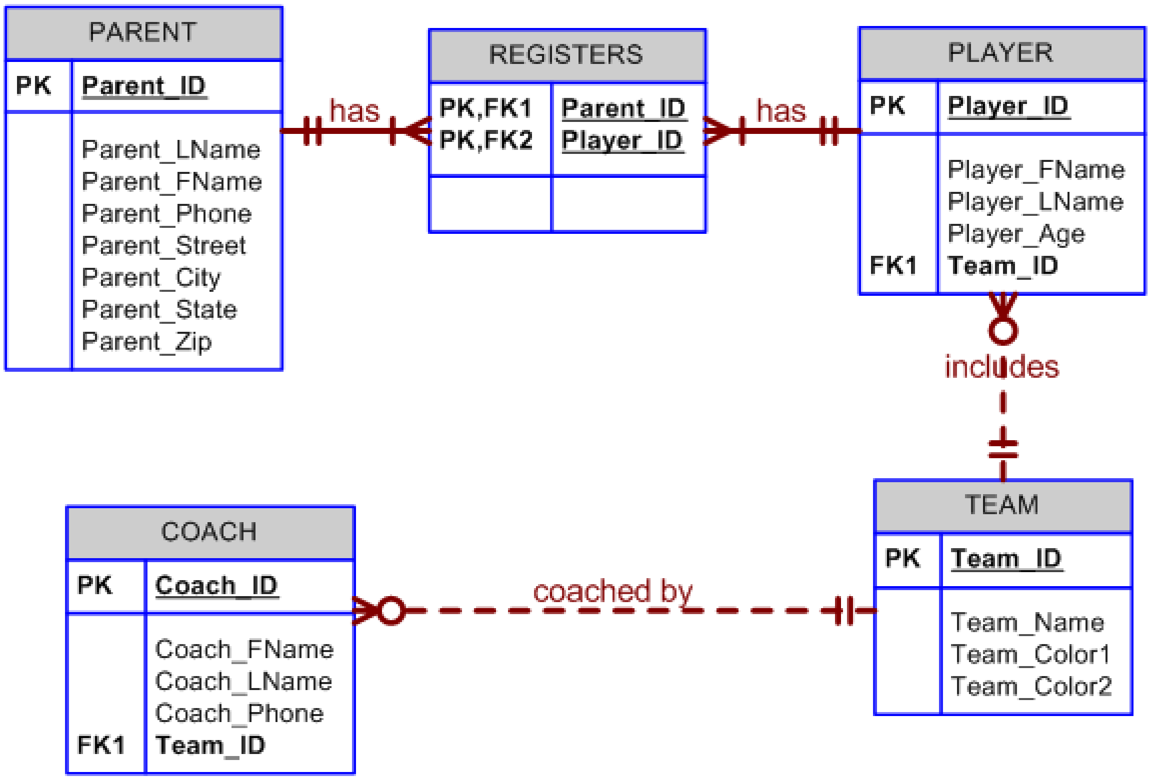
FIGURE Q4.21b ERD with foreign keys for Question 21



This solution, however, still leaves an interesting question about the Team_Colors attribute. What if teams have more than one color as is implied by the plural "colors" being used by the business users? Let's consider three options: 1) leave it as is (as if Team_Colors is a single-valued attribute), 2) create multiple attributes within the TEAM entity, or 3) create a new COLOR table.

Team_Colors may be left as a single attribute if it is determined through discussion with the business users that they are not concerned with dealing with the different colors individually. For example, they will never be interested to know how many teams have the color Blue as one of their team colors, then we may choose to implement the design as given above. However, if the users are interested, or foresee the possibility that at some time in the future they may become interested, in addressing the different colors for a given team individually, then we must modify the above design to accommodate this need. If we determine that all teams have the same number of colors, and no team now or in the future will ever have more than that number of colors, then we may modify the design by adding additional attributes in the TEAM entity. For example, if all teams, now and forever, will always have exactly two team colors then we may produce the design shown in Figure Q4.21c.

FIGURE Q4.21c ERD with two team colors for Question 21



This is a reasonable solution given the assurance that all teams now and forever will have exactly two team colors. A problem arises, however, if we cannot rely on that assurance. If some teams have fewer colors, then our design will lead to an increased number of nulls. If a team ever has more than two colors, we will have to modify the structure of the database after it has been built to add another team color attribute. This change in structure may require changes in the front-end applications so that they can properly address this new attribute. To avoid these potentially serious modifications in the future, we can re-design the database with a more robust structure that can handle any number of team colors without future modifications to the database or the front-end applications. The design with a separate table to handle the multi-valued Team_Colors attribute is shown in Figure Q4.21d.

FIGURE Q4.21d ERD with Color table for Question 21