**Chapter 3**

**The Relational Database Model**

**Discussion Focus**

Why is most of this book based on the relational database model? The answer to that question is, quite simply, that the relational database model has a very successful track record and it is the dominant database model in the market.

But why has the relational database model (RDM) been so successful? The Object Oriented database model (OODM) seemed to be poised to dislodge the RDM in the face of increasingly complex data that included video and audio … yet the OODM fell short in the database arena. However, the OODM’s basic concepts have become the basis of a wide variety of database systems analysis and design procedures. In addition, the basic OO approach has been adopted by many application generators and other development tools.

The OODM’s inability to replace the RDM is due to several factors. First, the large installed base of RDM-based databases is difficult to overcome. Change is often difficult and expensive, so the prime requisite for change is an overwhelming advantage of the change agent. The OODM advantages were simply not accepted as *overwhelming* and were, therefore, not accepted as cost-effective. Second, compared to the RDM, the OODM’s design, implementation, and management learning curves are much steeper than the RDM’s. Third, the RDM preempted the OODM in some important respects by adopting many of the OODM’s best features, thus becoming the extended relational data model (ERDM). Because the ERDM retains the basic modeling simplicity of the RDM while being able to handle the complex data environment that was supposed to be the OODM’s forte, you can have the proverbial cake and eat it, too.

The OODM-ERDM battle for dominance in the database marketplace seems remarkably similar to the one waged by the hierarchical and network models against the relational model almost three decades ago. The OODM and ERDM are similar in the sense that each attempts to address the demand for more semantic information to be incorporated into the model. However, the OODM and the ERDM differ substantially both in underlying philosophy and in the nature of the problem to be addressed. Although the ERDM includes a strong semantic component, it is primarily based on the relational data model’s concepts. In contrast, the OODM is wholly based on the OO and semantic data model concepts. The ERDM is primarily geared to business applications, while the OODM tends to focus on very specialized engineering and scientific applications. In the database arena, the most likely scenario appears to be an ever-increasing merging of OO and relational data model concepts and procedures.

Although the ERDM label has frequently been used in the database literature to describe the -- quite successful -- relational data model’s response to the OODM challenge, C. J. Date objects to the ERDM label for the following reasons (set forth in “Back to the Relational Future” [***www.dbpd.com/vault/9808date.html***](http://www.dbpd.com/vault/9808date.html)*)*:

* The useful contribution of the object model is its ability to let users define their own -- and often very complex -- data types. However, mathematical structures known as “domains” in the relational model also provide this ability. Therefore, a relational DBMS that properly supports such domains greatly diminishes the reason for using the object model. Given proper support for domains, relational data models are quite capable of handling the complex data encountered in time series, engineering design, office automation, financial modeling, and so on. Because the relational model can support complex data types, the notion of an “extended relational data model” or ERDM is “extremely inappropriate and inaccurate” and “it should be firmly resisted.” (The capability that is supposedly being extended is already there!)
* Even the label object/relational data model (O/RDM) is not quite accurate, because the relational data model’s domain is not an object model structure. However, there are already quite a few O/R products -- also known as **universal database servers** -- on the market. Therefore, Date concedes that we are probably stuck with the O/R label. In fact, Date believes that “an O/R system is in everyone’s future.” More precisely, Date argues that a true O/R system would be “nothing more nor less than a true relational system -- which is to say, a system that supports the relational model, with all that such support entails.”

C. J. Date concludes his discussion by observing that “We need do nothing to the relational model to achieve object functionality. (Nothing, that is, except implement it, something that doesn’t yet seem to have been tried in the commercial world.)”

Because C. J. Date is generally considered to be one of the world’s leading database thinkers and innovators, his observations cannot be easily dismissed. In any case, regardless of the label that is used to tag the relational data model’s growing capabilities, it seems clear that the relational data model is likely to maintain its database market dominance for some time. We believe, therefore, that our continued emphasis on the relational data model is appropriate.

**Answers to Review Questions**

|  |
| --- |
| MCj04315950000[1]**ONLINE CONTENT**  **Answers to selected Review Questions and Problems for this chapter are contained in the Premium Website for this book. The Premium Website also includes SQL script files (Oracle and SQLServer) for all of the data sets used throughout the book.** |

1. **What is the difference between a database and a table?**

A table, a logical structure that represents an entity set, is only one of the components of a database.

The database is a structure that houses one or more tables and metadata. The metadata are data about data. Metadata include the data (attribute) characteristics and the relationships between the entity sets.

1. **What does it mean to say that a database displays both entity integrity and referential integrity?**

*Entity integrity* describes a condition in which all tuples within a table are ***uniquely*** identified by their primary key. The unique value requirement prohibits a null primary key value, because nulls are not unique.

*Referential integrity* describes a condition in which a foreign key value has a match in the corresponding table or in which the foreign key value is null. The null foreign key “value” makes it possible ***not*** to have a corresponding value, but the matching requirement on values that are not null makes it impossible to have an invalid value.

1. **Why are entity integrity and referential integrity important in a database?**

Entity integrity and referential integrity are important because they are the basis for expressing and implementing relationships in the entity relationship model. Entity integrity ensures that each row is uniquely identified by the primary key. Therefore, entity integrity means that a proper search for an ***existing*** tuple (row) will always be successful. (And the failure to find a match on a row search will always mean that the row for which the search is conducted does not exist in that table.) Referential integrity means that, if the foreign key contains a value, that value refers to an existing valid tuple (row) in another relation. Therefore, referential integrity ensures that it will be impossible to assign a non-existing foreign key value to a table.

1. **What are the requirements that two relations must satisfy in order to be considered union-compatible?**

In order for two relations to be union-compatible, both must have the same number of attributes (columns) and corresponding attributes (columns) must have the same domain. The first requirement is easily identified be a cursory glance at the relations' structures. If the first relation has 3 attributes then the second relation must also have 3 attributes. If the first table has 10 attributes, then the second relation must also have 10 attributes. The second requirement is more difficult to assess and requires understanding the meanings of the attributes in the business environment. Recall that an attribute's domain is the set of allowable values for that attribute. To satisfy the second requirement for union-compatibility, the first attribute of the first relation must have the same domain as the first attribute of the second relation. The second attribute of the first relation must have the same domain as the second attribute of the second relation. The third attribute of the first relation must have the same domain as the third attribute of the second relation, and so on.

1. **Which relational algebra operators can be applied to a pair of tables that are not union-compatible?**

The Product, Join, and Divide operators can be applied to a pair of tables that are not union-compatible. Divide does place specific requirements on the tables to be operated on; however, those requirements do not include union-compatibility. Select (or Restrict) and Project are performed on individual tables, not pairs of tables. (Note that if two tables are joined, then the result is a single table and the Select or Project operator is performed on that single table.)

1. **Explain why the data dictionary is sometimes called "the database designer's database."**

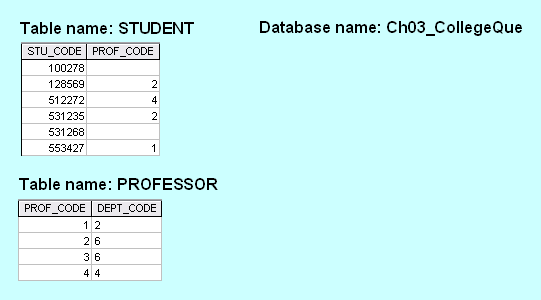
Just as the database stores data that is of interest to the users regarding the objects in their environment that are important to them, the data dictionary stores data that is of interest to the database designer about the important decisions that were made in regard to the database structure. The data dictionary contains the number of tables that were created, the names of all of those tables, the attributes in each table, the relationships between the tables, the data type of each attribute, the enforced domains of the attributes, etc. All of these data represent decisions that the database designer had to make and data that the database designer needs to record about the database.

1. **A database user manual notes that, “The file contains two hundred records, each record containing nine fields.” Use appropriate relational database terminology to “translate” that statement.**

Using the proper relational terminology, the statement may be translated to "the table -- or entity set -- contains two hundred rows -- or, if you like, two hundred tuples, or entities. Each of these rows contains nine attributes."

1. **Using the STUDENT and PROFESSOR tables shown in Figure Q3.8 to illustrate the difference between a natural join, an equijoin, and an outer join.**

**FIGURE Q3.8 The Ch03\_CollegeQue Database Tables**



|  |
| --- |
| **ONLINE CONTENT**  **All of the databases used in the questions and problems are found on the Premium Website for this book. The database names used in the folder match the database names used in the figures. For example, the source of the tables shown in Figure Q3.8 is the Ch03\_CollegeQue database. The Premium Website also includes SQL script files (Oracle and SQLServer) for all of the data sets used throughout the book.** |

The natural JOIN process begins with the PRODUCT of the two tables:

|  |  |  |  |
| --- | --- | --- | --- |
| **STU\_CODE** | **PROF\_CODE** | **PROF\_CODE** | **DEPT\_CODE** |
| 100278 |  | 1 | 2 |
| 128569 | 2 | 1 | 2 |
| 512272 | 4 | 1 | 2 |
| 531235 | 2 | 1 | 2 |
| 531268 |  | 1 | 2 |
| 553427 | 1 | 1 | 2 |
| 100278 |  | 2 | 6 |
| 128569 | 2 | 2 | 6 |
| 512272 | 4 | 2 | 6 |
| 531235 | 2 | 2 | 6 |
| 531268 |  | 2 | 6 |
| 553427 | 1 | 2 | 6 |
| 100278 |  | 3 | 6 |
| 128569 | 2 | 3 | 6 |
| 512272 | 4 | 3 | 6 |
| 531235 | 2 | 3 | 6 |
| 531268 |  | 3 | 6 |
| 553427 | 1 | 3 | 6 |
| 100278 |  | 4 | 4 |
| 128569 | 2 | 4 | 4 |
| 512272 | 4 | 4 | 4 |
| 531235 | 2 | 4 | 4 |
| 531268 |  | 4 | 4 |
| 553427 | 1 | 4 | 4 |

Next, a SELECT is performed on the PRODUCT generated in the first step to yield only the rows for which the PROF\_CODE values in the STUDENT table are matched in the PROF table. Because only the STUDENT table’s PROF\_CODE values 1, 2, and 4 yield matches in the PROFESSOR table, the SELECT yields the following output:

|  |  |  |  |
| --- | --- | --- | --- |
| **STU\_CODE** | **PROF\_CODE** | **PROF\_CODE** | **DEPT\_CODE** |
| 128569 | 2 | 2 | 6 |
| 512272 | 4 | 4 | 4 |
| 531235 | 2 | 2 | 6 |
| 553427 | 1 | 1 | 2 |

Finally, a PROJECT is performed to produce the natural JOIN output by listing only a single copy of each attribute. The order in which the query output rows are shown is not relevant. If the output is to be listed by having the STU\_CODE values in ascending order, this result can be generated through an “order by” specification in the query – remind the students that they can learn how that is done in Chapter 7, “An Introduction to Structured Query Language (SQL)”.

|  |  |  |
| --- | --- | --- |
| **STU\_CODE** | **PROF\_CODE** | **DEPT\_CODE** |
| 128569 | 2 | 6 |
| 512272 | 4 | 4 |
| 531235 | 2 | 6 |
| 553427 | 1 | 2 |

The equiJOIN's results depend on the specified condition. At this stage of the students' understanding, it may be best to focus on equijoins that retrieve all matching values in the common attribute. In such a case, the output will be:

|  |  |  |  |
| --- | --- | --- | --- |
| **STU\_CODE** | **STUDENT. PROF\_CODE** | **PROFESSOR.PROF\_CODE** | **DEPT\_CODE** |
| 128569 | 2 | 2 | 6 |
| 512272 | 4 | 4 | 4 |
| 531235 | 2 | 2 | 6 |
| 553427 | 1 | 1 | 2 |

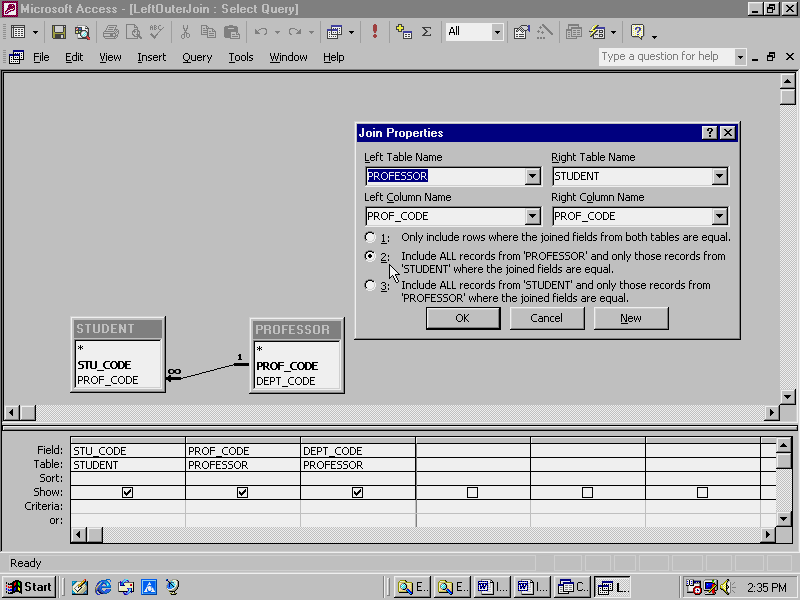
Notice that in equijoins, the common attribute appears from both tables. It is normal to prefix the attribute name with the table name when an attribute appears more than once in a table. This maintains the requirement that attribute names be unique within a relational table.

In the Outer JOIN, the unmatched pairs would be retained and the values that do not have a match in the other table would be left null. It should be made clear to the students that Outer Joins are not the opposite of Inner Joins (like Natural Joins and Equijoins). Rather, they are "Inner Join Plus" – they include all of the matched records found by the Inner Join **plus** the unmatched records. Outer JOINs are normally performed as either a Left Outer Join or a Right Outer Join so that the operator specifies which table's unmatched rows should be included in the output. Full Outer Joins depict the matched records plus the unmatched records from both tables. Also, like Equijoins, Outer Joins do not drop a copy of the common attribute. Therefore, a Full Outer Join will yield these results:

|  |  |  |  |
| --- | --- | --- | --- |
| **STU\_CODE** | **STUDENT. PROF\_CODE** | **PROFESSOR.PROF\_CODE** | **DEPT\_CODE** |
| 128569 | 2 | 2 | 6 |
| 512272 | 4 | 4 | 4 |
| 531235 | 2 | 2 | 6 |
| 553427 | 1 | 1 | 2 |
| 100278 |  |  |  |
| 531268 |  |  |  |
|  |  | 3 | 6 |

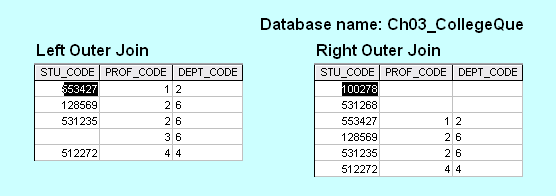
Microsoft Access uses two outer join options to make it easy to find unmatched pairs. For example, its outer join selections, generated from its QBE (Query By Example) query generator, are particularly effective. Note, for example, the selected left outer join option in Figure Q3.5A and look at its output in Figure Q3.5b to see that professor 3 does not have any student advisees.

**Figure Q3.8a The Left Outer Join QBE Properties Selection**



If you select the option 2 in Figure Q3.8a’s QBE option screen, you’d get the output shown in the left panel in Figure Q3.8b. Note that you can now easily detect that professor number 3 does not have any student advisees assigned to him or her. If you select the option 3 in Figure Q3.8a’s QBE option screen, you’d get the output shown in the right panel in Figure Q3.8b. The latter output makes it easy to detect that students 100278 and 531268 do not yet have an advisor assigned to them.

**Figure Q3.8b The Left and Right Outer Join Outputs**



1. **Create the basic ERD for the database shown in Figure Q3.8.**

Both the Chen and Crow’s Foot solutions are shown in Figure Q3.9. (We have used the PowerPoint template to produce the first of the two Crow’s Foot ERDs and Visio Professional to produce the second of the two Crow’s Foot ERDs.

**Figure Q3.6 The Chen and Crow’s Foot ERD Solutions for Question 6**

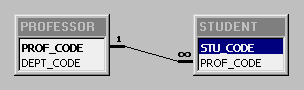


|  |
| --- |
| **NOTE**  **From this point forward, we will show the ERDs in Visio Professional format unless the problem specifies a different format. Please refer your students to the Visio Professional tutorial in Appendix A, “Designing Databases with Visio Professional: A Tutorial.” This appendix is located on the Premium Website.** |

1. **Create the relational diagram for the database shown in Figure Q3.8.**

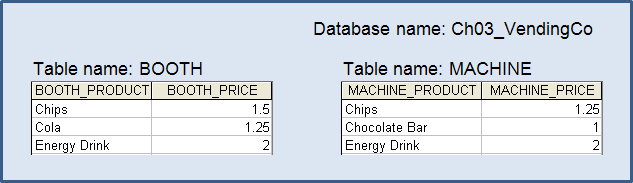
The relational diagram, generated in the Microsoft Access **Ch03\_CollegeQue** database, is shown in Figure Q.3.10.

**Figure Q3.10 The Relational Diagram**



**Use Figure Q3.11 to answer questions 11 – 13.**

**Figure Q3.11 The Ch03\_VendingCo database tables**

****

1. **Create the table that results from applying a UNION relational operator to the tables shown in Fig Q3.11**

|  |  |
| --- | --- |
| **BOOTH\_PRODUCT** | **BOOTH\_PRICE** |
| Chips | 1.5 |
| Cola | 1.25 |
| Energy Drink | 2 |
| Chips | 1.25 |
| Chocolate Bar | 1 |

Note that when the attribute names are different, the result will take the attribute names from the first relation. In this case, the solution assumes the operation was BOOTH UNION MACHINE. If the operation had been MACHINE UNION BOOTH then the attribute names from the MACHINE table would have appears as the attribute names in the result. Also, notice that the "Chips" from both tables appears in the result, but the "Energy Drink" from both does not. A UNION operator will eliminate duplicate rows from the result; however, the entire row must match for two rows to be considered duplicates. In the case of "Chips", the product names were the same but the prices were different. In the case of "Energy Drink", both the product names and the prices matched so the second Energy Drink row was dropped from the result.

1. **Create the table that results from applying an INTERSECT relational operator to the tables shown in Fig Q3.11.**

|  |  |
| --- | --- |
| **BOOTH\_PRODUCT** | **BOOTH\_PRICE** |
| Energy Drink | 2 |

Note that when the attribute names are different, the result will take the attribute names from the first relation. In this case, the solution assumes the operation was BOOTH INTERSECT MACHINE. If the operation had been MACHINE INTERSECT BOOTH then the attribute names from the MACHINE table would have appears as the attribute names in the result.

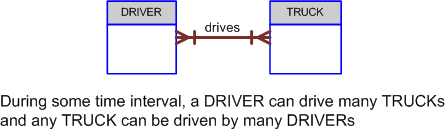
1. **Using the tables in Figure Q3.11, create the table that results from MACHINE DIFFERENCE BOOTH.**

|  |  |
| --- | --- |
| **MACHINE\_PRODUCT** | **MACHINE\_PRICE** |
| Chips | 1.25 |
| Chocolate Bar | 1 |

Note that the order in which the relations are specified is significant in the results returned. The DIFFERENCE operator returns the rows from the first relation that are not duplicated in the second relation. Just as with the INTERSECT operator, the entire row must match an existing row to be considered a duplicate.

1. **Suppose that you have the ERM shown in Figure Q3.14.**

**FIGURE Q3.14 The Crow’s Foot ERD for Question 14**



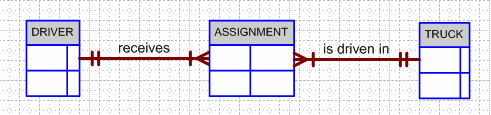
**How would you convert this model into an ERM that displays only 1:M relationships? (Make sure that you draw the revised ERM.)**

The Crow’s Foot solution is shown in Figure Q3.14a. Note that the original M:N relationship has been decomposed into two 1:M relationships based on these business rules:

* A driver may receive many (driving) assignments.
* Each (driving) assignment is made for a single driver.
* A truck may be driven in many (driving) assignments.
* Each (driving) assignment is made for a single truck.

Note that a driver can drive only one truck at a time, but during some period of time, a driver may be assigned to drive many trucks. The same argument holds true for trucks – a truck can only be driven during one trip (assignment) at a time, but during some period of time, a truck may be assigned to be driven in many trips. Also, remind students that they will be introduced to optional (and additional) relationships as they study Chapter 4, “Entity Relationship (ER) Modeling.” Finally, remind your students that you always read the relationship from the “1” side to the “M” side. Therefore, you read “DRIVER receives ASSIGNMENT and “TRUCK is driven in ASSIGNMENT.”

**Figure Q3.14a The Crow’s Foot ERM Solution for Question 14**



1. **What are homonyms and synonyms, and why should they be avoided in database design?**

Homonyms appear when more than one attribute has the same name. Synonyms exist when the same attribute has more than one name. Avoid both to avoid inconsistencies. For example, suppose we check the database for a specific attribute such as NAME. If NAME refers to customer names as well as to sales rep names, a clear case of a homonym, we have created an ambiguity, because it is no longer clear which entity the NAME belongs to.

Synonyms make it difficult to keep track of foreign keys if they are named differently from the primary keys they point to. Using REP\_NUM as the foreign key in the CUSTOMER table to reference the primary key REP\_NUM in the SALESREP table is much clearer than naming the CUSTOMER table's foreign key SLSREP. The proliferation of different attribute names to describe the same attributes will also make the data dictionary more cumbersome to use.

Some data RDBMSes let the data dictionary check for homonyms and synonyms to alert the user to their existence, thus making their use less likely. For example, if a CUSTOMER table contains the (foreign) key REP\_NUM, the entry of the attribute REP\_NUM in the SALESREP table will either cause it to inherit all the characteristics of the original REP\_NUM, or it will reject the use of this attribute name when different characteristics are declared by the user.

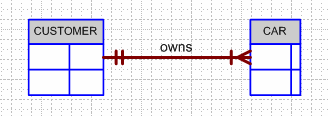
1. **How would you implement a l:M relationship in a database composed of two tables? Give an example.**

Let’s suppose that an auto repair business wants to track its operations by customer. At the most basic level, it’s reasonable to assume that any database design you produce will include at least a car entity and a customer entity. Further suppose that it is reasonable to assume that:

* A car is owned just by one customer.
* A customer can own more than one car.

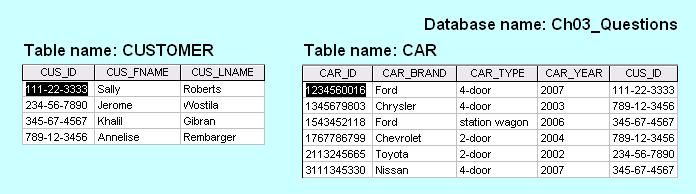
The CAR and CUSTOMER entities and their relationships are represented by the Crow’s Foot ERD shown in Figure Q3.16. (Discussion: Explain to your students that the ERDs are very basic at this point. Your students will learn how to incorporate much more detail into their ERDs in Chapter 4. For example, no thought has –yet – been given to optional relationships or to the strength of those relationships. At this stage of learning the business of database design, simple is good! To borrow an old Chinese proverb, a journey of a thousand miles begins with a single step.)

**Figure Q3.16a The CUSTOMER owns CAR ERM**



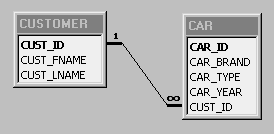
An example of this implemented relationship is shown in Figure Q3.16b. Note that the "many" side of the relation (the CAR entity) contains the foreign key, which is the CUSTOMER entity's primary key.

**Figure Q3.16b The CUSTOMER and CAR Tables**



The relational diagram for this implementation is shown in Figure Q3.16c.

**Figure Q3.16c The Relational Diagram**



(The tables shown in Figure Q3.16b and the relational diagram shown in Figure Q3.16c are found in the **Ch03\_Questions** database that is located on the CD.)

1. **Identify and describe the components of the table shown in Figure Q3.17, using correct terminology. Use your knowledge of naming conventions to identify the table’s probable foreign key(s).**

**FIGURE Q3.17 The Ch03\_NoComp Database EMPLOYEE Table**

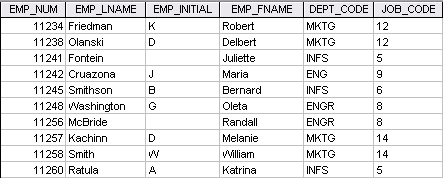
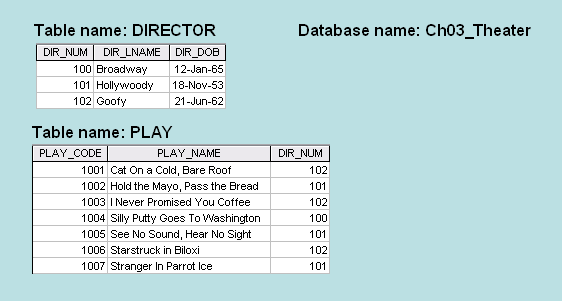


Figure Q3.17's database table contains:

* **One entity set:** EMPLOYEE.
* **Five attributes:** EMP\_NUM, EMP\_LNAME, EMP\_INIT, EMP\_FNAME, DEPT\_CODE and JOB\_CODE.
* **Ten entities:** the ten workers shown in rows 1-10.
* **One primary key:** the attribute EMP\_NUM because it identifies each row uniquely.
* **Two foreign keys:** the attribute DEPT\_CODE, which probably references a department to which the employee is assigned and the attribute JOB\_CODE which probably references another table in which you would find the description of the job and perhaps additional information pertaining to the job.

**Use the database composed of the two tables shown in Figure Q3.18 to answer Questions 18-23.**

**FIGURE Q3.18 The Ch03\_Theater Database Tables**



1. **Identify the primary keys.**

DIR\_NUM is the DIRECTOR table's primary key.

PLAY\_CODE is the PLAY table's primary key.

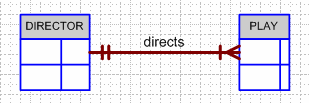
1. **Identify the foreign keys.**

The foreign key is DIR\_NUM, located in the PLAY table. Note that the foreign key is located on the "many" side of the relationship between director and play. (Each director can direct many plays ... but each play is directed by only one director.)

1. **Create the ERM.**

The entity relationship model is shown in Figure Q3.20.

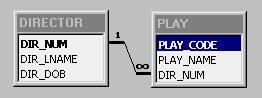
**Figure Q3.20 The Theater Database ERD**



1. **Create the relational diagram to show the relationship between DIRECTOR and PLAY.**

The relational diagram, shown in Figure 3.21, was generated with the help of Microsoft Access. (Check the **Ch03\_Theater** database.)

**Figure Q3.21 The Relational Diagram**



1. **Suppose you wanted quick lookup capability to get a listing of all plays directed by a given director. Which table would be the basis for the INDEX table, and what would be the index key?**

The PLAY table would be the basis for the appropriate index table. The index key would be the attribute DIR\_NUM.

1. **What would be the conceptual view of the INDEX table that is described in question 22? Depict the contents of the conceptual INDEX table.**

The conceptual index table is shown in Table Q3.23.

**Figure Q3.23 The Conceptual Index Table**

|  |  |
| --- | --- |
| **Index Key** | **Pointers to the PLAY table** |
| 100 | 4 |
| 101 | 2, 5, 7 |
| 102 | 1, 3, 6 |

**Problem Solutions**

**Use the database shown in Figure P3.1 to answer Problems 1-9.**

**FIGURE P3.1 The Ch03\_StoreCo Database Tables**



1. **For each table, identify the primary key and the foreign key(s). If a table does not have a foreign key, write *None* in the space provided.**

|  |  |  |
| --- | --- | --- |
| **TABLE** | **PRIMARY KEY** | **FOREIGN KEY(S)** |
| EMPLOYEE | EMP\_CODE | STORE\_CODE |
| STORE | STORE\_CODE | REGION\_CODE, EMP\_CODE |
| REGION | REGION\_CODE | NONE |

1. **Do the tables exhibit entity integrity? Answer yes or no and then explain your answer.**

|  |  |  |
| --- | --- | --- |
| **TABLE** | **ENTITY INTEGRITY** | **EXPLANATION** |
| EMPLOYEE | Yes | Each EMP\_CODE value is unique and there are no nulls. |
| STORE | Yes | Each STORE\_CODE value is unique and there are no nulls. |
| REGION | Yes | Each REGION\_CODE value is unique and there are no nulls. |

1. **Do the tables exhibit referential integrity? Answer yes or no and then explain your answer. Write *NA* (Not Applicable) if the table does not have a foreign key.**

|  |  |  |
| --- | --- | --- |
| **TABLE** | **REFERENTIAL INTEGRITY** | **EXPLANATION** |
| EMPLOYEE | Yes | Each STORE\_CODE value in EMPLOYEE points to an ***existing*** STORE\_CODE value in STORE. |
| STORE | Yes | Each REGION\_CODE value in STORE points to an ***existing*** REGION\_CODE value in REGION and each EMP\_CODE value in STORE points to an ***existing*** EMP\_CODE value in EMPLOYEE. |
| REGION | NA |  |

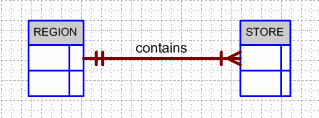
1. **Describe the type(s) of relationship(s) between STORE and REGION.**

Because REGION\_CODE values occur more than once in STORE, we may conclude that each REGION can contain many stores. But since each STORE is located in only one REGION, the relationship between STORE and REGION is M:1. (It is, of course, equally true that the relationship between REGION and STORE is 1:M.)

1. **Create the ERD to show the relationship between STORE and REGION.**

The Crow’s Foot ERD is shown in Figure P3.5. Note that each store is located in a single region, but that each region can have many stores located in it. (It’s always a good time to focus a discussion on the role of business rules in the creation of a database design.)

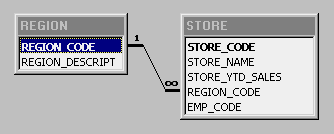
**Figure P3.5 ERD for the STORE and REGION Relationship**



1. **Create the relational diagram to show the relationship between STORE and REGION.**

The relational diagram is shown in Figure P3.6. Note (again) that the location of the entities is immaterial … the relationships are carried along with the entity. Therefore, it does not matter whether you locate the REGION on the left side or on the right side of the display. But you always read from the “1” side to the “M” side, regardless of the entity location.

**Figure P3.6 The Relational Diagram for the STORE and REGION Relationship**



1. **Describe the type(s) of relationship(s) between EMPLOYEE and STORE. (*Hint*: Each store employs many employees, one of whom manages the store.)**

There are TWO relationships between STORE and REGION. The first relationship, expressed by STORE *employs* EMPLOYEE, is a 1:M relationship, because one store can employ many employees and each employee is employed by one store. The second relationship, expressed by EMPLOYEE *manages* STORE, is a 1:1 relationship, because each store is managed by one employee and an employee manages only one store.

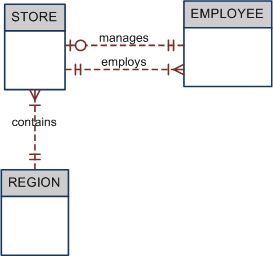
|  |
| --- |
| **NOTE**  **It is useful to introduce several ways in which the *manages* relationship may be implemented. For example, rather than creating the *manages* relationship between EMPLOYEE and STORE, it is possible to simply list the manager's name as an attribute in the STORE table. This approach creates a redundancy which may not do much damage if the information requirements are limited. However, if it is necessary to keep track of each manager's sales and personnel management performance by store, the manages relationship we have shown here will do a much better job in terms of information generation. Also, you may want to introduce the notion of an *optional* relationship. After all, not all employees participate in the *manages* relationship. We will cover optional relationships in detail in Chapter 4, “Entity relationship (ER) Modeling.”** |

1. **Draw the ERD to show the relationships among EMPLOYEE, STORE, and REGION.**

The Crow’s Foot ERD is shown in Figure P3.8. Remind students that you always read from the “1” side to the “M” side in any 1:M relationship, i.e., a STORE *employs* many EMPLOYEEs and a REGION *contains* many STORES. In a 1:1 relationship, you always read from the “parent” entity to the related entity. In this case, only one EMPLOYEE *manages* each STORE … and each STORE is managed by only one EMPLOYEE.

We have shown Figure P3.8’s Visio Professional-generated ERD to include the properties of the *manages* relationship. Note that there is no mandatory 1:1 relationship available at this point. That’s why there is an optional relationship – the **O** symbol – next to the STORE entity to indicate that an employee is not necessarily a manager. Let your students know that such optional relationships will be explored in detail in Chapter 4. (Explain that you can create mandatory 1:1 relationships when you add attributes to the entity boxes and specify a mandatory data entry for those attributes that are involved in the 1:1 relationship.) If you use Microsoft Visio Professional, let your students use the Visio tutorial in Appendix A, located in the book’s Premium Website.

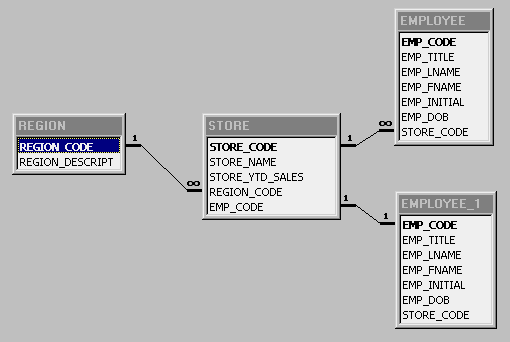
**Figure P3.8 StoreCo Crow’s Foot ERD**



1. **Create the relational diagram to show the relationships among EMPLOYEE, STORE, and REGION.**

The relational diagram is shown in Figure P3.9.

**Figure P3.9 The Relational Diagram**



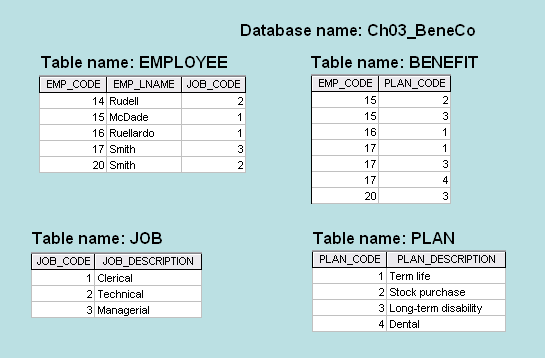
|  |
| --- |
| **NOTE**  **The relational diagram in Figure P3.9 was generated in Microsoft Access. If a relationship already exists between two entities, Access generates a virtual table (in this case, EMPLOYEE\_1) to generate the additional relationship. The virtual table cannot be queried; its only function is to store the *manages* relationship between EMPLOYEE and STORE. Just how multiple relationships are stored and managed is a function of the software you use.** |

**Use the database shown in Figure P3.10 to work Problems 10−16. Note that the database is composed of four tables that reflect these relationships:**

* **An EMPLOYEE has only one JOB\_CODE, but a JOB\_CODE can be held by many EMPLOYEEs.**
* **An EMPLOYEE can participate in many PLANs, and any PLAN can be assigned to many EMPLOYEEs.**

**Note also that the M:N relationship has been broken down into two 1:M relationships for which the BENEFIT table serves as the composite or bridge entity.**

**FIGURE P3.10 The Ch03\_BeneCo Database Tables**



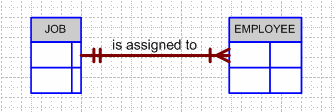
1. **For each table in the database, identify the primary key and the foreign key(s). If a table does not have a foreign key, write *None* in the assigned space provided.**

|  |  |  |
| --- | --- | --- |
| **TABLE** | **PRIMARY KEY** | **FOREIGN KEY(S)** |
| EMPLOYEE | EMP\_CODE | JOB\_CODE |
| BENEFIT | EMP\_CODE + PLAN\_CODE | EMP\_CODE, PLAN\_CODE |
| JOB | JOB-CODE | None |
| PLAN | PLAN\_CODE | None |

1. **Create the ERD to show the relationship between EMPLOYEE and JOB.**

The ERD is shown in Figure P3.11. Note that the JOB\_CODE = 1 occurs twice in the EMPLOYEE table, as does the JOB\_CODE = 2, thus providing evidence that a JOB can be assigned to many EMPLOYEEs. But each EMPLOYEE has only one JOB\_CODE, so there exists a 1:M relationship between JOB and EMPLOYEE.

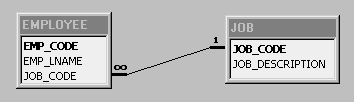
**Figure P3.11 The ERD for the EMPLOYEE-JOB Relationship**



1. **Create the relational diagram to show the relationship between EMPLOYEE and JOB.**

The relational schema is shown in Figure P3.12.

**Figure P3.12 The Relational Diagram**



1. **Do the tables exhibit entity integrity? Answer yes or no and then explain your answer.**

|  |  |  |
| --- | --- | --- |
| **TABLE** | **ENTITY INTEGRITY** | **EXPLANATION** |
| EMPLOYEE | Yes | Each EMP\_CODE value is unique and there are no nulls. |
| BENEFIT | Yes | Each ***combination*** of EMP\_CODE and PLAN\_CODE values is unique and there are no nulls. |
| JOB | Yes | Each JOB\_CODE value is unique and there are no nulls. |
| PLAN | Yes | Each PLAN\_CODE value is unique and there are no nulls. |

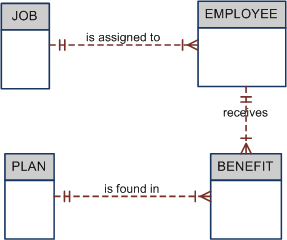
1. **Do the tables exhibit referential integrity? Answer yes or no and then explain your answer. Write *NA* (Not Applicable) if the table does not have a foreign key.**

|  |  |  |
| --- | --- | --- |
| **TABLE** | **REFERENTIAL INTEGRITY** | **EXPLANATION** |
| EMPLOYEE | Yes | Each JOB\_CODE value in EMPLOYEE points to an existing JOB\_CODE value in JOB. |
| BENEFIT | Yes | Each EMP\_CODE value in BENEFIT points to an existing EMP\_CODE value in EMPLOYEE and each PLAN\_CODE value in BENEFIT points to an existing PLAN\_CODE value in PLAN. |
| JOB | NA |  |
| PLAN | NA |  |

1. **Create the ERD to show the relationships among EMPLOYEE, BENEFIT, JOB, and PLAN.**

The Crow’s Foot ERD is shown in Figure P3.15.

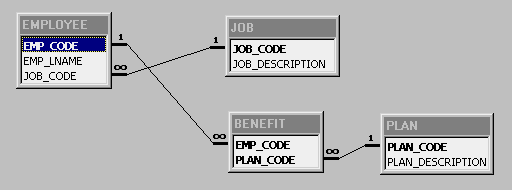
**Figure P3.15 BeneCo Crow’s Foot ERD**



1. **Create the relational diagram to show the relationships among EMPLOYEE, BENEFIT, JOB, and PLAN.**

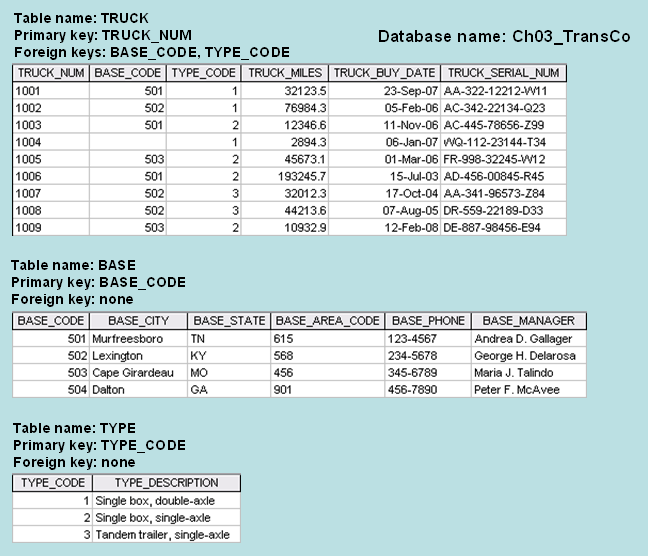
The relational diagram is shown in Figure P3.16. Note that the location of the entities is immaterial – the relationships move with the entities.

**Figure P3.16 The Relational Diagram**



**Use the database shown in Figure P3.17 to answer Problems 17-23.**

**FIGURE P3.17 The Ch03\_TransCo Database Tables**



1. **For each table, identify the primary key and the foreign key(s). If a table does not have a foreign key, write *None* in the space provided.**

|  |  |  |
| --- | --- | --- |
| **TABLE** | **PRIMARY KEY** | **FOREIGN KEY(S)** |
| TRUCK | TRUCK\_NUM | BASE\_CODE, TYPE\_CODE |
| BASE | BASE\_CODE | None |
| TYPE | TYPE\_CODE | None |

|  |
| --- |
| **NOTE**  **Note: The TRUCK\_SERIAL\_NUM could also be designated as the primary key. Because the TRUCK\_NUM was designated to be the primary key, TRUCK\_SERIAL\_NUM is an example of a *candidate key*.** |

1. **Do the tables exhibit entity integrity? Answer yes or no and then explain your answer.**

|  |  |  |
| --- | --- | --- |
| **TABLE** | **ENTITY INTEGRITY** | **EXPLANATION** |
| TRUCK | Yes | The TRUCK\_NUM values in the TRUCK table are all unique and there are no nulls. |
| BASE | Yes | The BASE\_CODE values in the BASE table are all unique and there are no nulls. |
| TYPE | Yes | The TYPE\_CODE values in the TYPE table are all unique and there are no nulls. |

1. **Do the tables exhibit referential integrity? Answer yes or no and then explain your answer. Write *NA* (Not Applicable) if the table does not have a foreign key.**

|  |  |  |
| --- | --- | --- |
| **TABLE** | **REFERENTIAL INTEGRITY** | **EXPLANATION** |
| TRUCK | Yes | The BASE\_CODE values in the TRUCK table reference existing BASE\_CODE values in the BASE table or they are null. (The TRUCK table's BASE\_CODE is null for TRUCK\_NUM = 1004.) Also, the TYPE\_CODE values in the TRUCK table reference existing TYPE\_CODE values in the TYPE table. |
| BASE | NA |  |
| TYPE | NA |  |

1. **Identify the TRUCK table’s candidate key(s).**

A candidate key is any key that could have been used as a primary key, but that was, for some reason, not chosen to be the primary key. For example, the TRUCK\_SERIAL\_NUM could have been selected as the PK, but the TRUCK\_NUM was actually designated to be the PK. Therefore, the TRUCK\_SERIAL\_NUM is a candidate key. Also, any combination of attributes that would uniquely identify any truck would be a candidate key. For example, the combination of BASE\_CODE, TYPE\_CODE, TRUCK\_MILES, and TRUCK\_BUY\_DATE is not likely to be duplicated and this combination would, therefore, be a candidate key. However, while the latter combination might constitute a candidate key, such a combination would not be practical. (An extreme – and impractical -- example of a candidate key would be the combination of all of a table’s attributes.)

|  |
| --- |
| **NOTE**  **Some of the answers to the following problem 21 define only a few of the available correct choices. For example, a superkey is, in effect, a candidate key containing redundant attributes. Therefore, any primary key plus any other attribute(s) is a superkey.**  **Because a secondary key does not necessarily yield unique outcomes, the number of attributes that constitute a secondary key is somewhat arbitrary. The adequacy of a secondary key depends on the extent of the end-user's willingness to accept multiple matches.** |

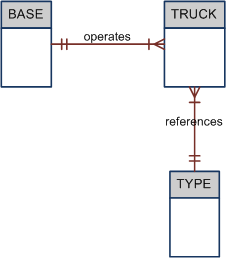
1. **For each table, identify a superkey and a secondary key.**

|  |  |  |
| --- | --- | --- |
| **TABLE** | **SUPERKEY** | **SECONDARY KEY** |
| TRUCK | TRUCK\_NUM + TRUCK\_MILES  TRUCK\_NUM + TRUCK\_MILES + TRUCK\_BUY\_DATE  TRUCK\_NUM + TRUCK\_MILES + TRUCK\_BUY\_DATE + TYPE\_CODE | BASE\_CODE + TYPE\_CODE  (This secondary key is likely to produce multiple matches, but it is not likely that end-users will know attribute values such as TRUCK\_MILES or TRUCK\_BUY\_DATE. Therefore, the selected attributes create a reasonable secondary key.) |
| BASE | BASE\_CODE + BASE\_CITY  BASE\_CODE + BASE\_CITY + BASE\_CITY | BASE\_CITY + BASE\_STATE  (This a very effective secondary key, since it is not likely that a state contains two cities with the same name.) |
| TYPE | TYPE\_CODE + TYPE\_DESCRIPTION | TYPE\_DESCRIPTION |

1. **Create the ERD for this database.**

The Crow’s Foot ERD is shown in Figures P3.22.

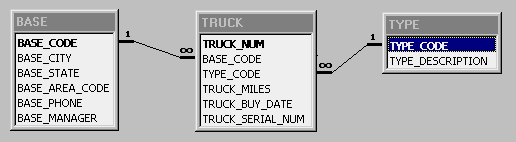
**Figure P3.22 TransCo Crow's Foot ERD**



1. **Create the relational diagram for this database.**

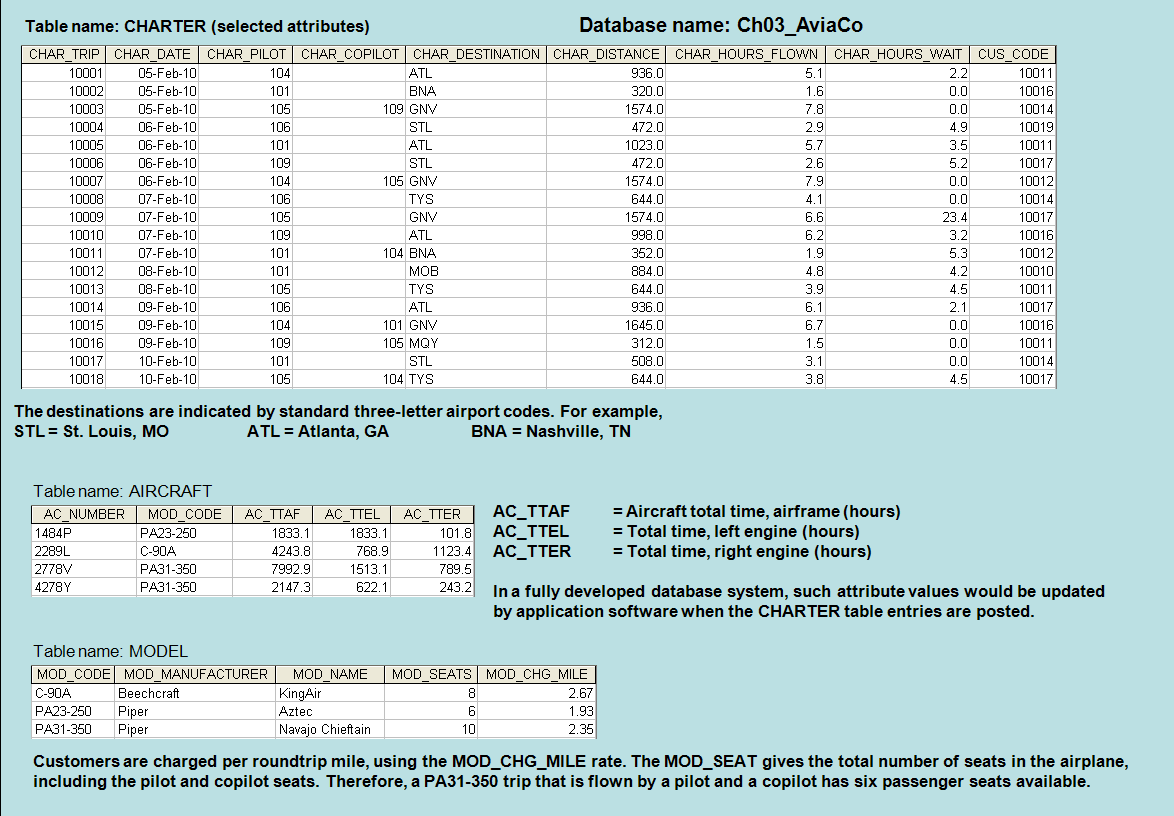
The relational diagram is shown in Figure P3.23.

**Figure P3.23 The Ch03\_TransCo Relational Diagram**



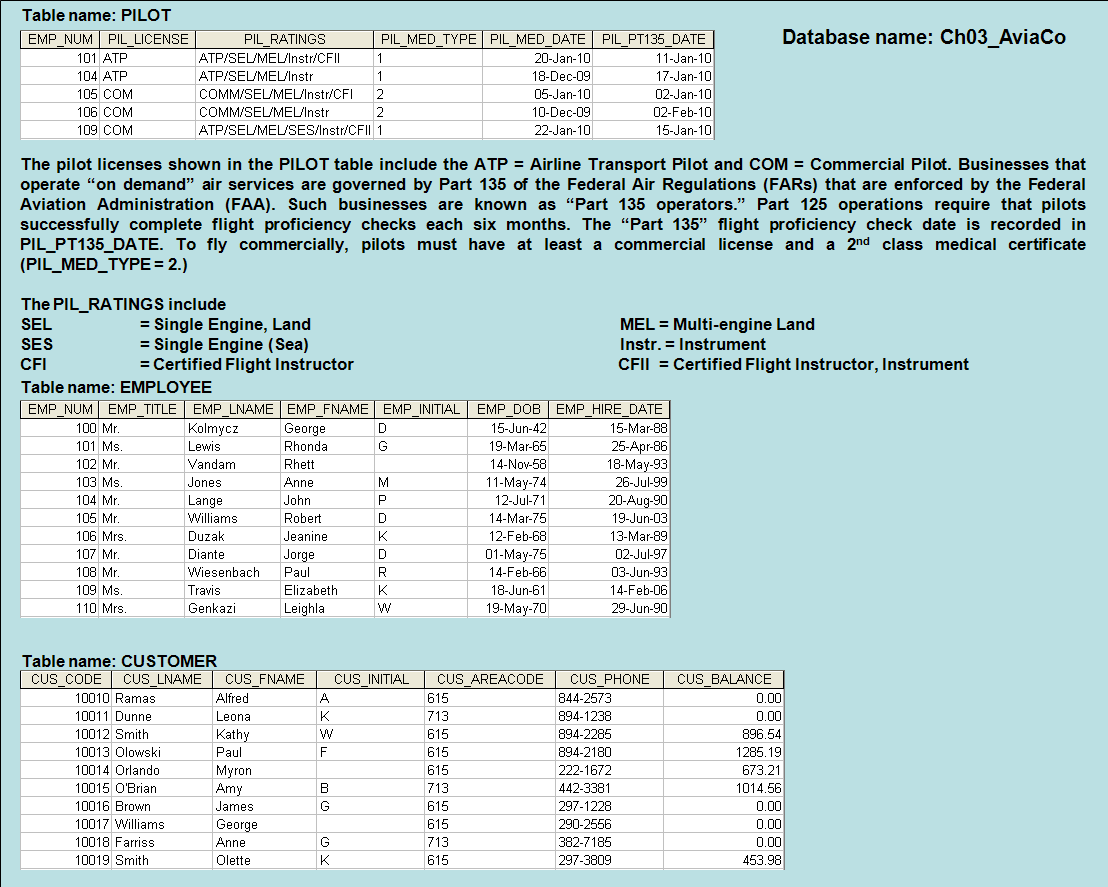
**Use the database shown in Figure P3.24 to answer Problems 24−31. ROBCOR is an aircraft charter company that supplies on-demand charter flight services using a fleet of four aircraft. Aircrafts are identified by a unique registration number. Therefore, the aircraft registration number is an appropriate primary key for the AIRCRAFT table.**

**FIGURE P3.24 The Ch03\_AviaCo Database Tables (Part 1)**

****

**The nulls in the CHARTER table’s CHAR\_COPILOT column indicate that a copilot is not required for some charter trips or for some aircraft. Federal Aviation Administration (FAA) rules require a copilot on jet aircraft and on aircraft having a gross take-off weight over 12,500 pounds. None of the aircraft in the AIRCRAFT table are governed by this requirement; however, some customers may require the presence of a copilot for insurance reasons. All charter trips are recorded in the CHARTER table.**

**FIGURE P3.24 The Ch03\_AviaCo Database Tables (Part 2)**

****

|  |
| --- |
| **NOTE**  **Earlier in the chapter, it was stated that it is best to avoid homonyms and synonyms. In this problem, both the pilot and the copilot are pilots in the PILOT table, but EMP\_NUM cannot be used for both in the CHARTER table. Therefore, the synonyms CHAR\_PILOT and CHAR\_COPILOT were used in the CHARTER table.**  **Although the solution works in this case, it is very restrictive and it generates nulls when a copilot is not required. Worse, such nulls proliferate as crew requirements change. For example, if the AviaCo charter company grows and starts using larger aircraft, crew requirements may increase to include flight engineers and load masters. The CHARTER table would then have to be modified to include the additional crew assignments; such attributes as CHAR\_FLT\_ENGINEER and CHAR\_LOADMASTER would have to be added to the CHARTER table. Given this change, each time a smaller aircraft flew a charter trip without the number of crew members required in larger aircraft, the missing crew members would yield additional nulls in the CHARTER table.**  **You will have a chance to correct those design shortcomings in Problem 27. The problem illustrates two important points:**   1. **Don’t use synonyms. If your design requires the use of synonyms, revise the design!** 2. **To the greatest possible extent, design the database to accommodate growth without requiring structural changes in the database tables. Plan ahead and try to anticipate the effects of change on the database.** |

**ROBCOR is an aircraft charter company that supplies on-demand charter flight services, using a fleet of four aircraft. Aircraft are identified by a (unique) registration number. Therefore, the aircraft registration number is an appropriate primary key for the AIRCRAFT table.**

**The nulls in the CHARTER table’s CHAR\_COPILOT column indicate that a copilot is not necessarily required for some charter trips or for some aircraft. Federal Aviation Administration (FAA) rules require a copilot on jet aircraft and on aircraft having a gross take-off weight over 12,500 pounds. None of the aircraft in the AIRCRAFT table is governed by this requirement; however, some customers may require the presence of a copilot for insurance reasons. All charter trips are recorded in the CHARTER table.**

|  |
| --- |
| **NOTE**  **Earlier in the chapter it was stated that it is best to avoid homonyms and synonyms. In this problem, both the pilot and the copilot are pilots in the PILOT table, but EMP\_NUM cannot be used for both in the CHARTER table. Therefore, the synonyms CHAR\_PILOT and CHAR\_COPILOT were used in the CHARTER table.**  **Although the “solution” works in this case, it is very restrictive and it generates nulls when a copilot is not required. Worse, such nulls proliferate as crew requirements change. For example, if the AviaCo charter company grows and starts using larger aircraft, crew requirements may increase to include flight engineers and load masters. The CHARTER table would then have to be modified to include the additional crew assignments; such attributes as CHAR\_FLT\_ENGINEER and CHAR\_LOADMASTER would have to be added to the CHARTER table. Given this change, each time a smaller aircraft flew a charter trip without the number of crew members required in larger aircraft, the “missing” crew members would yield additional nulls in the CHARTER table.**  **You will have a chance to correct those design shortcomings in Problem 33. The problem illustrates two important points:**   1. **Don’t use synonyms. If your design requires the use of synonyms, revise the design!** 2. **To the greatest possible extent, design the database to accommodate growth without requiring structural changes in the database tables. Plan ahead—and try to anticipate the effects of change on the database.** |

1. **For each table, where possible, identify:**
   1. **The primary key**

|  |  |
| --- | --- |
| **TABLE** | **PRIMARY KEY** |
| CHARTER | CHAR\_TRIP |
| AIRCRAFT | AC\_NUMBER |
| MODEL | MOD\_CODE |
| PILOT | EMP\_NUM |
| EMPLOYEE | EMP\_NUM |
| CUSTOMER | CUS\_CODE |

* 1. **A superkey**

|  |  |
| --- | --- |
| **TABLE** | **SUPER KEY** |
| CHARTER | CHAR\_TRIP + CHAR\_DATE |
| AIRCRAFT | AC\_NUM + MOD-CODE |
| MODEL | MOD\_CODE + MOD\_NAME |
| PILOT | EMP\_NUM + PIL\_LICENSE |
| EMPLOYEE | EMP\_NUM + EMP\_DOB |
| CUSTOMER | CUS\_CODE + CUS\_LNAME |

|  |
| --- |
| **NOTE**  **The super key and secondary key compositions were described in the note that accompanied Problem 21. The same arguments may be pursued with reference to parts b, c and e of this problem.** |

* 1. **A candidate key**

|  |  |
| --- | --- |
| **TABLE** | **CANDIDATE KEY** |
| CHARTER | No practical candidate key is available. For example, CHAR\_DATE + CHAR\_DESTINATION + AC\_NUMBER + CHAR\_PILOT + CHAR\_COPILOT will still not necessarily yield unique matches, because it is possible to fly an aircraft to the same destination twice on one date with the same pilot and copilot. You could, of course, present the argument that the combination of all the attributes would yield a unique outcome. |
| AIRCRAFT | See the previous discussion. |
| MODEL | See the previous discussion. |
| PILOT | See the previous discussion. |
| EMPLOYEE | See the previous discussion. But Perhaps the combination of EMP\_LNAME + EMP\_FNAME + EMP\_INITIAL + EMP\_DOB will yield an acceptable candidate key. |
| CUSTOMER | See the previous discussion |

* 1. **The foreign key(s)**

|  |  |
| --- | --- |
| **TABLE** | **FOREIGN KEY** |
| CHARTER | CHAR\_PILOT (references PILOT)  CHAR\_COPILOT (references PILOT)  AC\_NUMBER (references AIRCRAFT)  CUS\_CODE (references CUSTOMER) |
| AIRCRAFT | MOD\_CODE |
| MODEL | None |
| PILOT | EMP\_NUM (references EMPLOYEE) |
| EMPLOYEE | None |
| CUSTOMER | None |

* 1. **A secondary key**

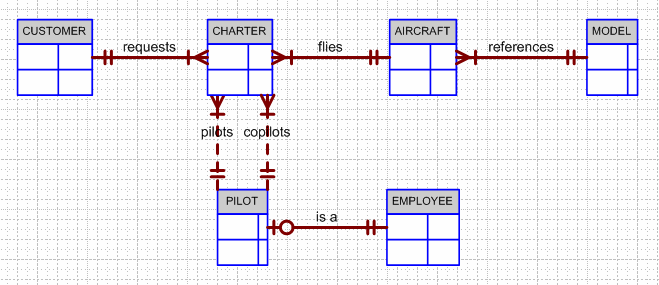
|  |  |
| --- | --- |
| **TABLE** | **SECONDARY KEY** |
| CHARTER | CHAR\_DATE + AC\_NUMBER + CHAR\_DESTINATION |
| AIRCRAFT | MOD\_CODE |
| MODEL | MOD\_MANUFACTURER + MOD\_NAME |
| PILOT | PIL\_LICENSE + PIL\_MED\_DATE |
| EMPLOYEE | EMP\_LNAME + EMP\_FNAME + EMP\_DOB |
| CUSTOMER | CUS\_LNAME + CUS\_FNAME + CUS\_PHONE |

1. **Create the ERD. (*Hint*: Look at the table contents. You will discover that an AIRCRAFT can fly many CHARTER trips but that each CHARTER trip is flown by one AIRCRAFT. Similarly, you will discover that a MODEL references many AIRCRAFT but that each AIRCRAFT references a single MODEL, etc.)**

The Crow’s Foot ERD is shown in Figure P3.25. The optional (default) 1:1 relationship crops up in this ERD, just as it did in the Problem 8 solution. Use the same discussion that accompanied Problem 8. Also, note that EMPLOYEE is the “parent” of PILOT.

Note that all pilots are employees, but not all employees are pilots – some are mechanics, accountants, and so on. (This discussion previews some of the Chapter 4 coverage … coming attractions, so to speak.) The relationship between PILOT and EMPLOYEE is read from the “parent” entity to the related entity. In this case, the relationship is read as “an EMPLOYEE is a PILOT.”

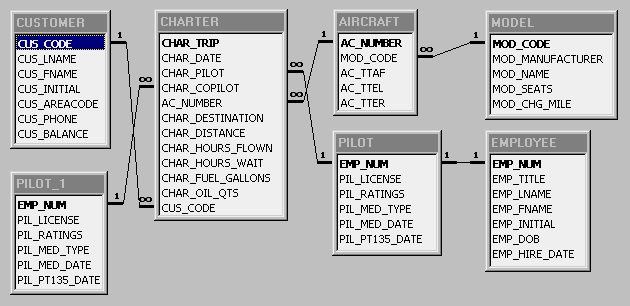
**Figure P3.25 The Ch03\_AviaCo Database ERD**



1. **Create the relational diagram.**

The relational diagram is shown in Figure P3.26.

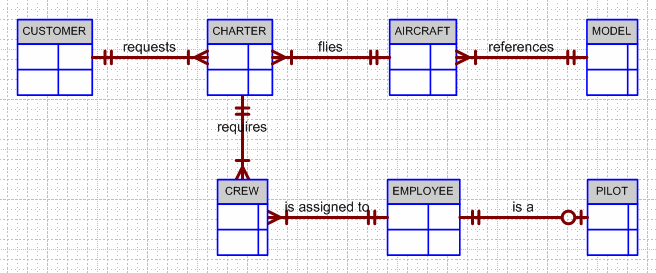
**Figure P3.26 The Ch03\_AviaCo Database Relational Diagram**



1. **Modify the ERD you created in Problem 25 to eliminate the problems created by the use of synonyms. (*Hint:* Modify the CHARTER table structure by eliminating the CHAR\_PILOT and CHAR\_COPILOT attributes; then create a composite table named CREW to link the CHARTER and EMPLOYEE tables. Some crewmembers, such as flight attendants, may not be pilots. That’s why the EMPLOYEE table enters into this relationship.)**

The Crow’s Foot ERD is shown in Figures P3.27.

**Figure P3.27 The Ch03\_AviaCo\_2 Database ERD**



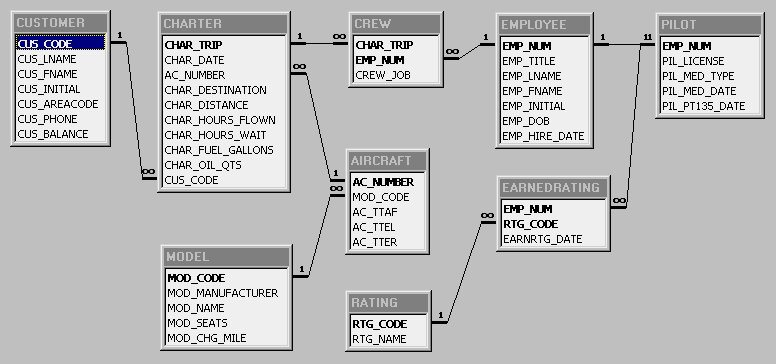
1. **Draw the relational diagram for the design you revised in problem 27. (After you have had a chance to revise the design, your instructor will show you the results of the design change, using a copy of the revised database named Ch03\_AviaCo\_2).**

The relational diagram for the Ch03\_AviaCo\_2 database is shown in Figure P3.28. Note that there are a few additional entities that you will encounter again in Chapter 4. (You can safely ignore the extra entities, RATING and EARNEDRATING at this point … but you can let the students “read” the relationship between these two entities.

Note that you can easily derive the M:N relationship between PILOT and RATING. (A PILOT can earn many RATINGs. A RATING can be earned by many PILOTs.) Even though your students may not know what a rating is, they can still draw up conclusions about its relationship to other entities by looking at relational diagrams and ERDs. And that’s one of the many strengths of design tools.

Also, you can let your students break the M:N relationship down into two 1:M relationships – note that this is done through the EARNEDRATING entity. The issues encountered in the design and implementation of the Ch3\_AviaCo\_2 database will be revisited many times in the book.

**Figure P3.28 The Ch03\_AviaCo\_2-Relational Diagram**



**You are interested in seeing data on charters flown by either Mr. Robert Williams (employee number 105) or Ms. Elizabeth Travis (employee number 109) as pilot or copilot, but not charters flown by both of them. Complete problems 29 – 31 to find these data.**

1. **Create the table that would result from applying the SELECT and PROJECT relational operators to the CHARTER table to return only the CHAR\_TRIP, CHAR\_PILOT, and CHAR\_COPILOT attributes for charters flown by either employee 105 or employee 109.**

|  |  |  |
| --- | --- | --- |
| **CHAR\_TRIP** | **CHAR\_PILOT** | **CHAR\_COPILOT** |
| 10003 | 105 | 109 |
| 10006 | 109 |  |
| 10009 | 105 |  |
| 10010 | 109 |  |
| 10013 | 105 |  |
| 10016 | 109 | 105 |
| 10018 | 105 | 104 |

1. **Create the table that would result from applying the SELECT and PROJECT relational operators to the CHARTER table to return only the CHAR\_TRIP, CHAR\_PILOT, and CHAR\_COPILOT attributes for charters flown by both employee 105 and employee 109.**

|  |  |  |
| --- | --- | --- |
| **CHAR\_TRIP** | **CHAR\_PILOT** | **CHAR\_COPILOT** |
| 10003 | 105 | 109 |
| 10016 | 109 | 105 |

1. **Create the table that would result from applying a DIFFERENCE relational operator of your result from problem 29 to your result from problem 30.**

|  |  |  |
| --- | --- | --- |
| **CHAR\_TRIP** | **CHAR\_PILOT** | **CHAR\_COPILOT** |
| 10006 | 109 |  |
| 10009 | 105 |  |
| 10010 | 109 |  |
| 10013 | 105 |  |
| 10018 | 105 | 104 |