Designing the DVDRentals Database

The data model that you design here is created for a fictional store that rents DVDs to its customers. The database tracks the inventory of DVDs, provides information about the DVDs, records rental transactions, and stores the names of the store's customers and employees. To create the data model, you need to use the following business rules:

❑ The database stores information about the DVDs available for rent. For each DVD, the information includes the DVD name, the number of disks included with the set, the year the DVD was released, the movie type (for example, Action), the studio that owns the movie rights (such as Columbia Pictures), the movie's rating (PG and so forth), the DVD format (Widescreen, for example), and the availability status (Checked Out). Multiple copies of the same DVD are treated as individual products.

❑ The database should store the names of actors, directors, producers, executive producers, co-producers, assistant producers, screenwriters, and composers who participated in making the movies available to rent. The information includes the participants' full names and the role or roles that they played in making the movie.

❑ The database should include the full names of the customers who rent DVDs and the employees who work at the store. Customer records should be distinguishable from employee records.

❑ The database should include information about each DVD rental transaction. The information includes the customer who rented the DVD, the employee who ran the transaction, the DVD that was rented, the date of the rental, the date that the DVD is due back, and the date that the DVD is actually returned. Each DVD rental should be recorded as an individual transaction.
Every transaction is part of exactly one order. One or more transactions are treated as a single order under the following conditions: (1) the transaction or transactions are for a single customer checking out one or more DVDs at the same time and (2) the transaction or transactions are being run by a single employee.

The business rules

The business rules provided here are not meant to be an exhaustive listing of all the specifications that would be required to create a database, particularly if those specifications were to include a front-end application that would be interfacing with the database. The business rules shown here, however, are enough to get you started in creating your data model. Later, as you add other elements to your database, the necessary business rules are provided.

Identifying Entities

The first step in creating a data model is to identify the possible objects (entities and attributes) to include in that model. Refer to the preceding business rules to perform the following steps:

1. Identify the possible objects listed in the first business rule. On a piece of paper, draw a rectangle for this business rule. Label the rectangle as “DVDs for rent.” In the rectangle, list the potential objects found in that business rule.

2. Repeat the process for the second business rule. Label the rectangle “Movie participants,” and include any objects that you identify in that business rule. This one might be trickier because it covers the roles in the movie (such as producer or actor) and the participants’ full names. Remember, though, that the actors, directors, producers, and so forth, are the participants.

3. Create two rectangles for the third business rule, label one “Customers” and the other “Employees,” and add the necessary objects.

4. For the fourth business rule, create a rectangle and label it “Transactions/orders.” Include in the rectangle each object that makes up a transaction. You should now have five rectangles with objects in each one.

5. The next step is to organize the objects into entities. You’ve already done some of the work simply by listing the objects. For example, the DVDs for rent group of objects can be left together as you have them, although you can simplify the category name to “DVDs.” The Transactions/orders group of entities also provides you with a natural category. You probably want to separate the customers and employees into separate categories and specify that an attribute be created for each part of the name. You should separate employees and customers into two entities so that you can easily distinguish between the two. In addition, this approach allows you to treat each category as a separate entity, which could be a factor in securing the database.

6. The last objects you should categorize are those in the Movie participants group. This group is a little different from the others because you’re working with a list of participant types, and for each of those types, you must include the full names of the participants. One way to approach this would be to create an entity for each participant type. Because different people can participate in different ways in one or more movies, though, you could be creating a situation in which your database includes a great deal of redundant data. Another approach would be to create one entity that includes the different parts of each participant’s name and to include an attribute that identifies the role that the participant plays.

Normalizing Data

Once you identify and categorize the entities and attributes for your database, you can begin to normalize the data structure and define the tables and columns. Usually, the best way to do this is to work with one entity at a time. The following steps help you normalize the data structure that you created in the previous exercise:

1. Start with the DVDs entity. This entity already provides a solid foundation for a table because it clearly groups together similar attributes. The first step should be to define

a primary key for the table. Because there can be multiple copies of the same DVD, and because different DVDs can share the same name, there is no column or columns that you can easily use as a primary key. As a result, the easiest solution is to add a column that uniquely identifies each row in the table. That way, each DVD has a unique ID.

2. At this point, you could leave the table as is and simply assign names and data types to the columns. (Note that, for the purposes of this exercise, the data types are provided for you.) When you have repeating values, such as status and format, an effective way to control the values that are permitted in a column is to create a table specifically for the values. That way you can add and modify permitted values without affecting the column definitions themselves. These types of tables are often referred to as *lookup tables.* Using lookup tables tends to decrease the amount of redundant data because the repeated value is often smaller than the actual value. For example, suppose you use an ID of s1 to identify the status value of Checked Out. Instead of repeating Checked Out for every row that this status applies to, the repeated value is s1, which requires less storage space and is easier to process.
   In the case of the DVDs s, you would probably want to create lookup tables for the movie type, studio, rating, format, and status attributes. Each of these tables would contain a primary key column and a column that describes the option available to the DVDs table. You would then add a column to the DVDs table that references the primary key in the lookup table. Also, be sure to name all your columns. This exercise uses a mixed case convention to name data objects. For example, a column named DVDName represents the DVD name attribute.

3. Next, take a look at the Participants entity. Notice that it includes an entry for the participant's role. This is another good candidate for a lookup table. Again, you should include a column that acts as the primary key and one that lists the role's name. Once you create a Roles table, you can add a column for the role ID to the Participants table. In addition, that table should include a primary key column that uniquely identifies each participant. Also, be sure to assign column names to the entities.

4. For both the Employees entity and the Customers entity, add a primary key column and assign a column name to the other attributes.

5. The next step is to separate the Transactions/orders entity into two tables because transactions are subsets of orders. For every order, there can be one or more transactions. In addition, each order must have one customer and one employee. As a result, you should create an Orders table.

The table should include a column that references the customer ID and a column that references the employee ID. The table should also include a primary key column.

Because you're tracking the customer and the employee at the order level, you do not need to include them at the transaction level. As a result, your Transactions table does not need to reference the employee or customer, but it does need to reference the order. In addition, it must reference the DVD that is being rented. Finally, you need to add a column to the Transactions table that acts as the primary key.

Identifying Relationships

Now that you've identified the tables and their columns, you can begin defining the relationships between the tables. In most cases, it should be fairly obvious from the business rules and table structures where relationships exist and the type of relationships those are, although it's always a good idea to review each pair of tables to verify whether a relationship exists between those tables. For the purposes of brevity, this exercise focuses only on those sets of tables between which a relationship exists. To identify the relationships between the tables in your data model, take the following steps:

1. Start with the DVDs table again, and compare it to other tables. You know from the previous exercise that lookup tables were created for several of the entities in this table. These include the MovieTypes, Studios, Ratings, Formats, and Status tables. Because of the nature of these tables and their origin with the DVDs table, you can assume that a one-to-many relationship exists between the DVDs table and the other five tables, with the DVDs table being the *many* side of the relationship. For example, the Studios table includes a list of the studios that own the rights to the movies. Astudio can be associated with one or more movies; however, a movie can be associated with only one studio. As a result, a one-to-many relationship exists between those two tables.
   To indicate these relationships on your data model, use a line to connect each lookup table to the DVDs table, using the three-prong end on the DVDs side. In addition, add an FK followed by a number for each column in the DVDs table that references one of the lookup tables.

2. Now look at the DVDs and Participants tables. As would be expected with an inventory of movies, each movie can include one or more participants, each participant can participate in one or more movies, and multiple participants can participate in multiple movies. As a result, a many-to-many relationship exists between those two tables. Draw the relationship on your data model. Be sure to use the three-prong end on both sides.

3. Another table paired with the DVDs table is the Transactions table. Because each transaction must list the DVD that's being rented, you can assume that a relationship exists between these two tables. For every DVD there can be one or more transactions. For every transaction, there can be only one DVD. As a result, a one-to-many relationship exists between these tables, with the Transactions table on the many side of the relationship. Draw the relationship on your data model. Remember to add the FK reference to the referencing column in the Transactions table.

4. The next pair of tables, to examine, are the Participants and Roles tables. As you recall, each participant can play multiple roles, each role can be played by multiple participants, and multiple participants can play multiple roles, so a many-to-many relationship exists between these two tables.

Draw the relationship on your data model. Use the three-prong end on both sides.

5. Next look at the Transactions and Orders tables. Every order can contain one or more transactions, but each transaction can be part of only one order, so a one-to-many relationship exists between these two tables, with the Transactions table being the *many* side of the relationship.Draw the relationship on your data model.

6. If you return to the Orders table, notice that it references the customer ID and employee ID, which means that a relationship exists between the Orders table and the Customers table as well as the Orders table and the Employees table. For every order, there can be only one employee and one customer, although each customer and employee can participate in multiple orders. As a result, the Orders table is on the *many* side of a one-to-many relationship with each of these other two tables.

7. For each of the relationships, you draw the correct relationship line on your data model, including the three-prong ends, where appropriate. Be sure to include the FK reference next to the referencing columns.

Refining the Data Model

To finalize your data model, take the following steps:

1. Review the data model for inconsistencies or data that is not fully normalized. Once satisfied with the model, you can move on to the next step.

2. The next step is to address the many-to-many relationships. As you recall, these relationships are implemented through the use of a junction table to create one-to-many relationships. The Participants table is part of two different many-to-many relationships, which presents you with a unique situation.
   To address these relationships, the best place to start is to define the situation: For each movie, there can be one or more participants who are playing one or more roles. In addition, each participant can play more than one role in one or more movies. As a result, movies are associated with participants, and participants are associated with roles. To address this situation, you can create one junction table that is related to all three tables: Roles, Participants, and DVDs. In each case, the junction table participates in a one-to-many relationship with the other tables, with the junction table on the *many* side of each relationship.

On your data model, add a junction table named DVDParticipant. Include a column that references the DVDs table, a column that references the Participants table, and a column that references the Roles table. Mark each column with an FK reference, followed by a number. Define the primary key on all three columns. Also, be sure to mark the one-to-many relationship on your model, and remove the RoleID column from the Participants table. The column is no longer needed in the Participants table, because it is included in the DVDParticipant table.