CHAPTER 9: DATA MANAGEMENT LAYER DESIGN

In the previous installments of the CD Selections case, we saw how Alec, Margaret, and the development team had worked through developing models and designs of the problem domain classes. Now that the design of the problem domain layer is somewhat stable, the team has moved into developing the models and designs of the solution domain (data management, human–computer interaction, and physical architecture) classes. In this installment, we follow the team members that have been assigned to the development of the data management layer classes for the Web-based system being developed for CD Selections.

To begin with, Margaret met with Alec to make sure that he realize that the CD Selections Internet Sales System needed to both present CD information effectively to users and to capture order data. Alec recognized that these goals were dependent upon a good design of the data management layer for the new application. He approached the design of the data management layer first by asking Brian to take charge of this task. Second, Alec made sure that Brian followed the following four steps:

- select the object-persistence format,
- map the problem domain classes to the selected format,
- optimize the selected format for processing efficiency, and
- design the data access and manipulation classes.

Brian assured Alec that he would do so and keep him abreast of the data management layer design as it progressed. Based on a quick review of the requirements, Brian requested to add two database experts to help with the design of the data management layer: John and Susan. After a little deliberation, Alec decided it would be worth the additional expense to add them to the team.

Select Object-Persistence Format

The first thing that Brian did was to call a meeting of his data management layer team to discuss two issues that would drive the object-persistence format selection: what kind of objects would be in the system and how they would be used. Using a whiteboard, they listed their ideas (see Figure 9-A). The project team agreed that the bulk of the data in the system

Data	Type	Use	Suggested Format
Customer information	Simple (mostly text)	Transactions	Relational
Order Information	Simple (text and numbers)	Transactions	Relational
Marketing Information	Both simple and complex (eventually the system will contain audio clips, video, etc.)	Transactions	Object add-on?
Information that will be exchanged with the Distribution System	Simple text, formatted specifically for importing into the Distribution System	Transactions	Transaction file
Temporary Information	The Web component will likely need to hold information for temporary periods of time. (e.g., the shopping card will store order information before the order is actually placed)	Transactions	Transaction file

FIGURE 9-A
Types of Data in Internet Sales System

would be the text and numbers that are exchanged with Web users regarding customers and orders. A relational database would be able to handle the data effectively, and the technology would be well received at CD Selections because relational technology is already in place throughout the organization.

However, they realized that relational technology was not optimized to handle complex data, such as the images, sound, and video that the marketing facet of the system ultimately will require. Alec asked Brian to look into relational databases that offered object add-on products (i.e., an RDBMS that could become an ORDBMS). It might be possible for the team to invest in a RDBMS foundation and then upgrade to an ORDBMS version of the same product. However, in the meantime, Alec decided to store sample clips using a random file. This way, they could still deliver the system as envisioned while keeping the technology requirements reasonable.

The team also noted that it must design two transaction files to handle the interface with the distribution system and the Web shopping cart program. The Internet Sales System will regularly download order information to the distribution system using a transaction file containing all the required information for that system. Also, the team must design the file that stores temporary order information on the Web server as customers shop through the Web site. The file would contain the fields that ultimately would be transferred to an order object. Of course, Alec realized that other data needs might arise over time, but he felt confident that the major data issues were identified (e.g., like the capability to handle complex data) and that the design of the data management layer would be based on the proper storage technologies.

Map Problem Domain Objects to Object-persistence format

Based on the decision to use an RDBMS and a random file to store the problem domain objects, Brian created an object-persistence design. To begin, Brian first reviewed the current class and package diagrams for the evolving Internet Sales System (see Figures 7-A, 7-B, 8-A, 8-B, and 8-F). Focusing on these figures, Brian and his team began applying the appropriate mapping rules (see Figure 9-9). Based on Rule 1, Brian identified 12 problem domain classes that needed to have there objects stored; as such, Brian created 11 RDBMS tables and 1 file to represent these objects. These included Credit Card Center table, Customer table, Person table, Organization table, Order table, Order Item table, CD table, Vendor table, Mkt Info table, Review table, Artist Info table, and Sample Clip File. He also created a set of tentative primary keys for each of the tables and the file. Based on the fact that the objects in the Shopping Package (see Figures 7-A, 7-B, and 8-A), Shopping Cart, SC Item, Search Req, and CD List, are only temporary, Brian decided that there was no real need to address them at this point in the design.

Using Rule 4, Brian identified the need for the CD table and the Mkt Info table to both have each others primary key stored as a foreign key to the other table. Upon further reflection, Brian reasoned that because an actual instance of marketing information was only going to be associated with a single instance of CD and vice versa, he could have merged the two tables together. However, the team had earlier decided to keep them separate, so he decided to simply use the primary key of the CD table as the primary key of the Mkt Info table.

While reviewing the current set of attributes for each of the tables, John suggested that the development had left out the idea of a CD containing a set of tracks. As such, they added a multi-valued attribute, tracks, to the CD problem domain class. However, Brian then pointed out that when they apply Rule 5 to the CD class, they really needed to factor the tracks attribute out as a separate table. Furthermore, as Brian, John, and

Susan discussed the track attribute further, it was decided to include it as a problem domain class also.

Next, the data management layer team applied Rule 6 to the evolving object-persistence design. In doing this, Susan pointed out that the checks relationship between the Customer and Credit Card Center problem domain classes was a multivalued association. Furthermore, Brian then pointed out the same was true for the aggregation relationship between the Artist Info and Mkt Info problem domain classes. Therefore, these relationships needed their own table in the relational database. Not to be upstaged by Brian and Susan, John immediately pointed out that Rule 7 was applicable to 8 associations: Customer places Order, Order includes Order Item, Order Item contains CD, Vendor distributes CD, Mkt Info promotes CD, CD contains Tracks, and the three aggregation associations with the Mkt Info class (Review, Artist Info, and Sample Clip). As such, quite a few primary keys had to be copied into the relevant tables as foreign keys (e.g., the primary key of the Customer table had to be copied to the Order table). Can you identify the others?

Finally, Susan suggested the solution to the inheritance problem because RDBMSs do not support inheritance. She pointed out that when applying Rule 8a to the Customer superclass and the Person and Organization subclasses, the primary key of the Customer table also had to be copied to the tables representing the subclasses. Furthermore, she pointed out that an exclusive-or (XOR) condition existed between the two subclasses.

Based on all of the suggestions and hard work accomplished by the data management layer team, Brian was able to create a design of the object persistence for the Internet Sales System (see Figure 9-B).

Optimize Object Persistence and Estimate its Size

Upon completing the object persistence design, Brian requested a meeting with the development team to walkthrough the design.[6] After the walkthrough, Alec asked Brian to stay behind to discuss the data management layer model. Now that the team had a good idea of the type of object-persistence formats that would be used, they were ready for the third step: optimizing the design for performance efficiency. Since Brian was the analyst in charge of the data management layer, Alec wanted to discuss with him whether the model was optimized for storage efficiency. He also needed this done before the team discussed access speed issues. Brian assured Alec that the current object persistence model was in third normal form. He was confident of this because the project team followed the modeling guidelines that lead to a well-formed model.

Brian then asked about the file formats for the two transaction files identified in the earlier meeting. Alec suggested that he normalize the files to better understand the various tables that would be involved in the import procedure. Figure 9-C shows the initial file layout for the Distribution System import file as well as the steps that were taken as Brian applied each normalization rule.

The next step for the design of the data management layer was to optimize the design for access speed. Alec met with the data management layer design team and talked about the techniques that were available to speed up access. Together they listed all of the data

[6]It seems that Brian finally got the importance of verifying and validating everything.

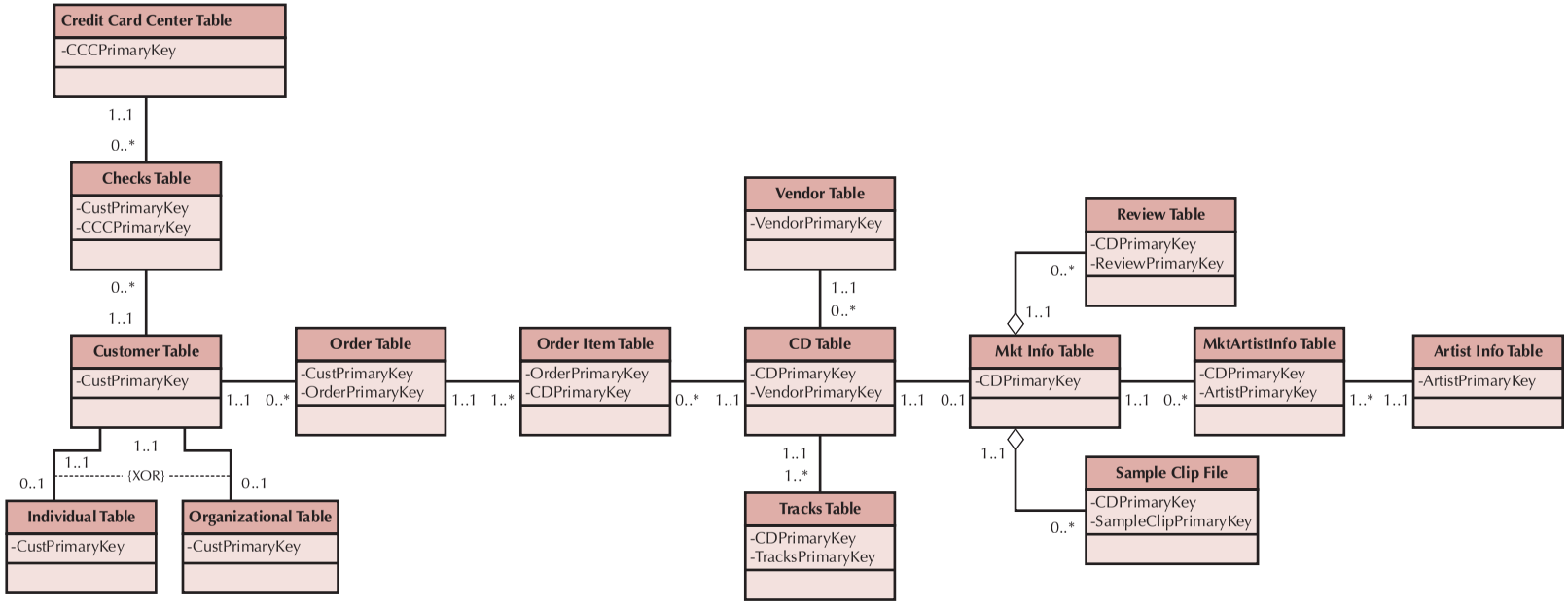


FIGURE 9-B Internet Sales System Object Persistence Design

0NF:

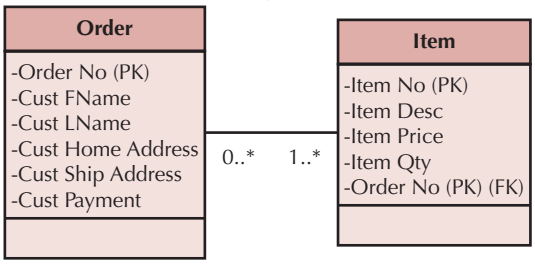
File Layout Required for Distribution System

Order Number	Cust Fname	Cust Lname	Cust HomeAdd	Cust ShipAdd	Cust Pay	Item No*	Item Desc*	Item Price*	Item Qty*
A (7)	A (20)	A (20)	A (150)	A (150)	9999.99	A (7)	A (20)	9999.99	9999

* Item No, Item Desc, Item Price, and Item Qty repeats four (4) times.

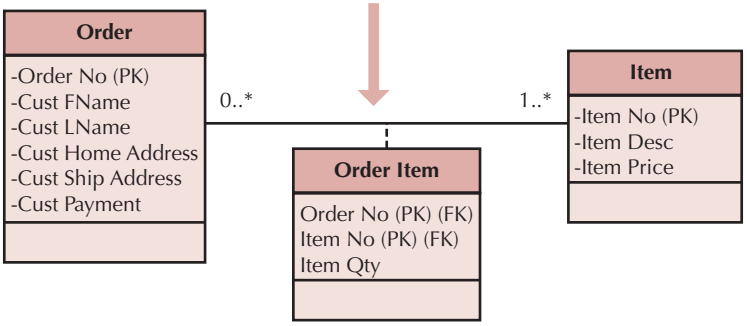
Remove repeating groups of items and place them in a separate Item entity

1NF:



There is one partial dependencies in the above data model since Item qty is dependent on the whole primary key while Item Desc and Item Price is dependent only on Item No.

2NF:



Remove transitive dependencies. Customer home-address is dependent upon Cust-fname and Cust-lname (and these do not serve as the identifier for the Order entity); therefore, a Customer entity is added to contain customer information. Order then contains only order information and necessary foreign keys.

3NF:

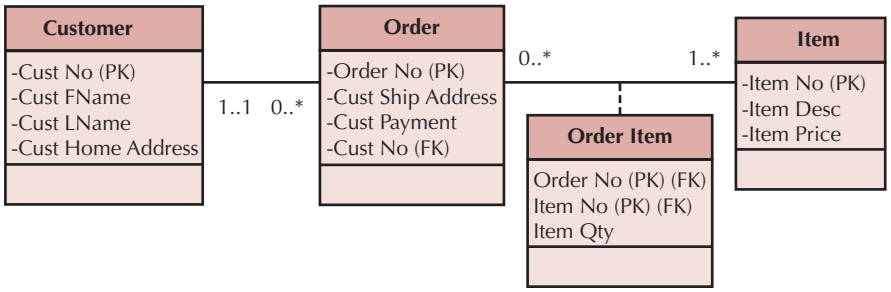


FIGURE 9-C
Distribution System
Import File
Normalization Process

Target	Comments	Suggestions to Improve Data Access Speed
All tables	Basic table manipulation	• Investigate if records should be clustered physically by primary key • Create indexes for primary keys • Create indexes for foreign key fields
All tables	Sorts and Grouping	• Create indexes for fields that are frequently sorted or grouped
CD information	Users will need to search CD information by title, artist, and category	• Create indexes for CD title, artist, and category
Order Information	Operators should be able to locate information about a particular customer's order	• Create an index in the Order table for orders by customer name
Entire Physical Model	Investigate denormalization opportunities for all fields that are not updated very often	• Investigate one-to-one relationships • Investigate look-up tables • Investigate one-to-many relationships

FIGURE 9-D
Internet Sales System Performance

that will be supported by the Internet Sales System and discussed how all of the data would be used. Based on these discussions, they developed the strategy to identify the specific techniques to put in place (see Figure 9-D).

Ultimately, clustering strategies, indexes, and denormalization decisions were applied to the physical data model, and a volumetrics report was run from the CASE tool to estimate the initial and projected size of the database. The report suggested that an initial storage capacity of about 450 megabytes would be needed for the expected one-year life of the first version of the system. Additional storage capacity would be needed for the second version that would include sound files for samples of the songs, but for the moment not much storage would be needed.

Since Anne was in charge of managing the server hardware acquisition, Alec gave the estimates to her so that the she could make sure that the technology could handle the expected volume of data for the Internet Sales System. She then would send the estimates to the DBMS vendor during the implementation of the software so that the DBMS could be configured properly.

Data Access and Manipulation Class Design

The final step in designing the data management layer was to develop the design of the data access and manipulation classes that would act as translators between the object persistence and the problem domain classes. Since the CD package (see Figures 7-A, 7-B, 8-A, and 8-F) was the most important package, Alec asked Brian to complete the design of the data management layer for the CD package and report back to Alec when he was finished. Upon reviewing the concrete problem domain classes in the CD package, Brian realized that he needed to have seven data access and manipulation classes; one for each concrete problem domain class. These classes would be fully dependent on their related problem domain classes. Next, Brian mapped the data access and manipulation classes down to the RDBMS tables and the random file associated with storing the objects (see Figure 9-B). In this case, there were seven RDBMS tables and one random file. Again, the data access and manipulation classes are dependent on the object-persistence format. Brian created Figure 9-E to depict the data management layer and problem domain layer for the CD package of the Internet sales system.

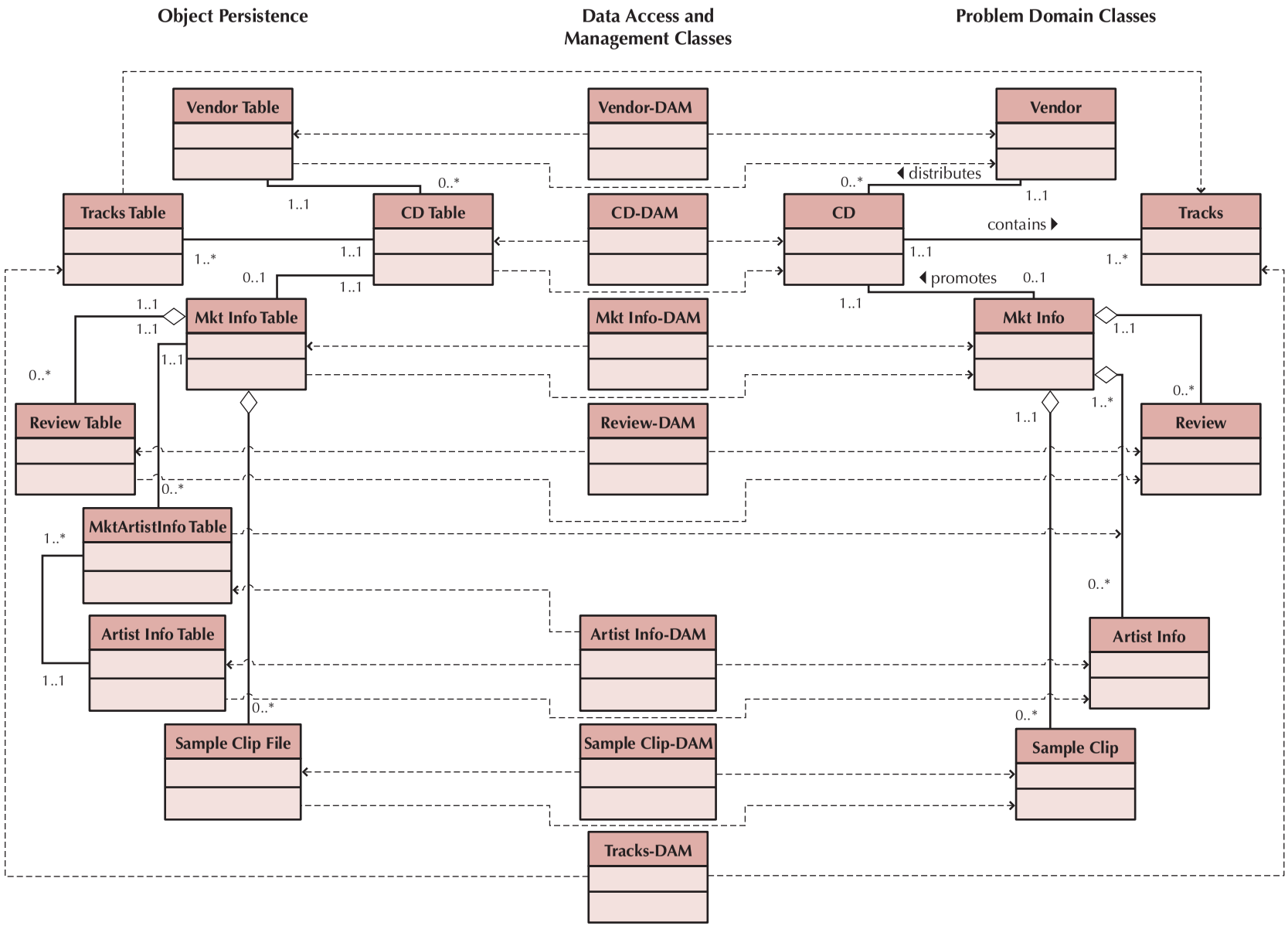


FIGURE 9-E Data Management Layer and Problem Domain Layer Design for the CD Package of the Internet Sales System

When Brian was going over the data management layer design with Alec (see Figure 9-E), Alec noticed that the Artist Info-DAM class was dependent upon both the Artist Info Table and the MktArtistInfo Table. Brian explained that this was necessary since an instance of the Artist Info problem domain class can only exist if a related instance of the Mkt Info problem domain class exists (see the multiplicities of 1..*). To guarantee this, the Artist Info-DAM class must depend on the existence of both the Artist Info Table and the table that represents the required relationship: MktArtistInfo Table. However, since the relationship from the Mkt Info problem domain class to the Artist Info class was optional (see multiplicities of 0..*), the Mkt Info-DAM class was not dependent on the MktArtistInfo Table. If it had been a required relationship also, then the Mkt Info-Dam class would have also been dependent on the relationship based table. Based on Brian's design and explanation, Alec felt that the design of the data management layer was completed.