the associations are going to be maintained from a technical standpoint. In a relational database, for example, an association between two tables is maintained by a technique referred to as a *foreign key.* A foreign key is the primary key field(s) from one table that is repeated in another table to provide a common field between the two tables. The common field contains values that match a record in one table to a record in the other. For example, if we were to create two tables called Customer and Order that were related to each other, we could include the primary key field from Customer (cust_id) in the Order table as well. In this way, if we want to find out customer information (e.g., name, address, phone number) when looking at someone's order, we can use the value for cust_id that appears in the Order table to go back to the Customer table to locate the appropriate information.

Thus, on the physical ERD, the primary key fields in the parent tables (the "1" end of the relationship) are copied and placed as fields in the child tables (the "many" end of the relationship) and designated as foreign keys. The fields will contain values that are common between the two tables. Many times, the CASE tools that are used to draw ERDs will "migrate" foreign keys to the appropriate tables on the model automatically, and the database technology will ensure that the values in the two fields match appropriately, helping to ensure referential integrity.

**Step 5: Add System-Related Components** As the fifth and final step, components are added to the physical ERD to reflect special implementation needs, including components that were included on the DFD. We have mentioned balance between DFDs and ERDs in earlier chapters, and this balance must be maintained in the physical models as well. Therefore, implementation-specific data stores and data elements from the physical DFD should be included on the ERD as tables and fields. For example, in Figure 10-2 we added the Tune to buy history data store to the physical DFD to serve as a "backup" for tunes that are sent to the purchase tunes process. Now we will need to add a tune to buy batch history file to the physical ERD model along with its fields and relationships.

## Revisiting the CRUD Matrix

As discussed in Chapter 6, it is important to verify that the system's DFD and ERD models are balanced. In other words, we must ensure that data needed in the systems processes are stored and that all stored data are used by at least one process. The CRUD matrix was introduced in Chapter 6 as a tool showing how data are used by processes in the system.

Often the CRUD matrix is created during analysis on the basis of the logical process and data models. In design, as these models are converted to physical models, changes in the form of new processes, new data stores, and new data elements may occur. The CRUD matrix should be revised at this point to include the new components and ensure that balance is maintained between the physical ERD and DFDs.

If the CRUD matrix was not developed during analysis, it should be developed now prior to implementation. The matrix shows exactly how data are used and created by the major processes in the system, so it serves as a very useful component of the system design materials.

## Applying the Concepts at Tune Source

Let us now apply some of the concepts that you have learned by creating a physical ERD, using the logical ERD that was created in Chapter 6.

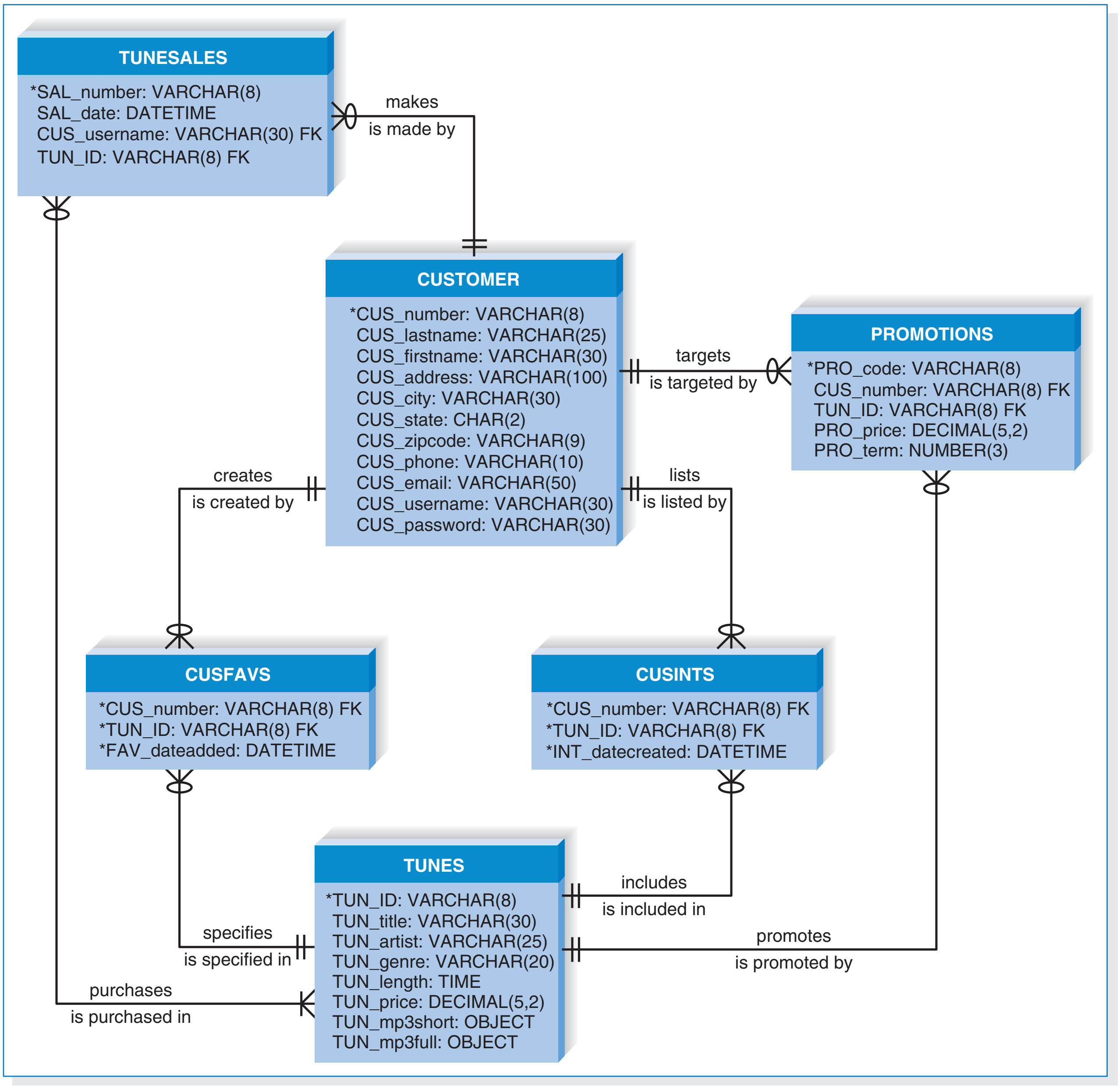

**FIGURE 11-13**
Tune Source Physical ERD

When we use the logical model as a starting point, the first step is to rename the entities to match with the tables or files that will be used by the system (Figure 11-13). Outwardly, the data model does not look very different after this step, but notice that several entities have been renamed to be consistent with Tune Source's table naming standards. At this time, we will need to include metadata for the tables, such as their estimated size.

Next, the attributes for the entities become fields with such characteristics as data type, length, and valid values, and this is recorded in the CASE repository. For example, CUS_state in the CUSTOMER table will be a text field with a size of two

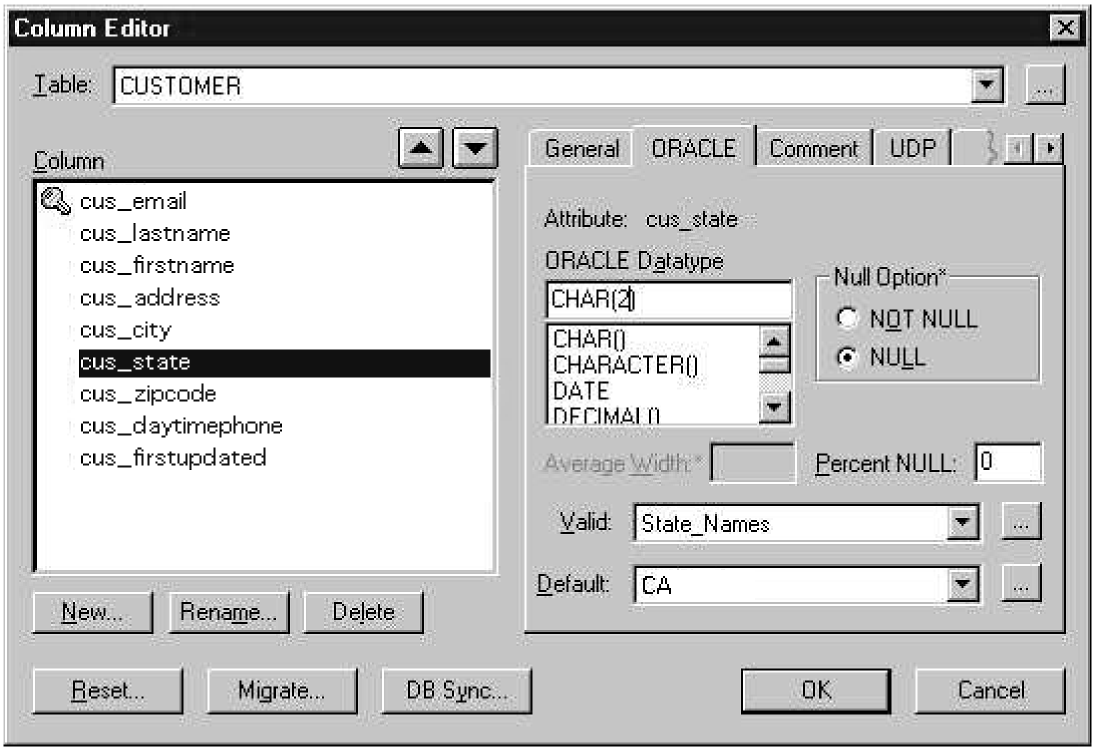
Column Editor

Table: CUSTOMER

Column

- cus_email
- cus_lastname
- cus_firstname
- cus_address
- cus_city
- cus_state
- cus_zipcode
- cus_daytimephone
- cus_firstupdated

General | ORACLE | Comment | UDP

Attribute: cus_state

ORACLE Datatype: CHAR(2)

CHAR()
CHARACTER()
DATE
DECIMAL()

Null Option*: NOT NULL [✓ NULL]

Average Width:*

Percent NULL: 0

Valid: State_Names

Default: CA

**FIGURE 11-14**
Computer-Aided Software Engineering Repository Entry for cus_state Field

characters, and valid values are the 50 two-letter state abbreviations. If most customers at Tune Source live in the state of California, then it may be worthwhile to make CA the default value for this field. However, since this is an Internet-based system, this assumption may not be valid. Figure 11-14 is an example of the CASE repository entry for the CUS_state field.

Step 3 suggests that we change the identifiers in the logical ERD to become primary keys, and entities without identifiers need to have a primary key created. At this time, we also can decide to use a system-generated primary key if it is more efficient than using logical attributes from the logical model.

The relationships on the logical ERD indicate where foreign key fields need to be placed. For example, CUS_number is placed as a field in TUNESALES to serve as the link between two entities, and TUNESALES gets the extra field because it is the child table (it exists at the "many" end of the relationship). Similarly, TUN_ID is placed in the TUNESALES table.

Finally, system-related components are included within the model. For example, fields that will capture when a record was last inserted or updated were added to many of the tables.

The project team also updated the CRUD matrix for the system. Figure 11-15 shows the CRUD matrix that was created for the Tune Source search and browse tunes process. Look at the original process models, and notice how the first process is merely reading information from data stores. This is illustrated on the CRUD matrix by an "R" placed in the relevant intersections of the matrix. Can you tell how data are used by the remaining processes?

| | 1.1 Load Web Site | 1.2 Process Search Requests | 1.3 Process Tune Selection |
|---|---|---|---|
| PROMOTIONS | | | |
| PRO_code | R | | |
| CUS_number | R | | |
| TUN_ID | R | | |
| PRO_price | R | | |
| PRO_term | R | | |
| CUSFAVS | | | |
| CUS_number | R | | C |
| TUN_ID | R | | C |
| FAV_dateadded | R | | C |
| TUNES | | | |
| TUN_ID | | R | R |
| TUN_title | | R | R |
| TUN_artist | | R | R |
| TUN_genre | | R | R |
| TUN_length | | R | R |
| TUN_price | | R | R |
| TUN_mp3short | | R | R |
| TUN_mp3full | | R | R |
| CUSINTS | | | |
| CUS_number | | | C |
| TUN_ID | | | C |
| INT_datecreated | | | C |

**FIGURE 11-15**
CRUD Matrix for Search and Browse Tunes Process

## OPTIMIZING DATA STORAGE

The selected data storage format is now optimized for processing efficiency. The optimization methods will vary with the format that you select; however, the basic concepts will remain the same. Once you understand how to optimize a particular type of data storage, you will have some idea as to how to approach the optimization of other formats. This section focuses on the optimization of the most popular data storage format: relational databases.

There are two primary dimensions in which to optimize a relational database: for storage efficiency and for speed of access. Unfortunately, these two goals often conflict because the best design for access speed may take up a great deal of storage space as compared with other less speedy designs. This section describes how to use normalization (Chapter 6) to optimize data storage for storage efficiency. The next section presents design techniques, such as denormalization and indexing, that will quicken the performance of the system. Ultimately, the project team will go through a series of trade-offs until the ideal balance between both optimization dimensions is reached. Finally, the project team must estimate the size of the data storage needed to ensure that there is enough capacity on the server(s).