**The Database Structure**

Before you can start coding the Food Store application, we'll need to talk about the database structure. Because the database is an important part of the application, it's in your best interest to spend a few minutes (or even hours) thinking through the database requirements before you start coding.

**The Products Table**

For the Food Store application, the first thing we'll need to be able to do is track the products in our store. So the first database object to look at is the *products* table. You'll need to create data fields in the products table to hold all of the information related to an individual product. Here's what you'll need to track:

|  |  |  |
| --- | --- | --- |
| **The products table** | | |
| **Data Field** | **Data Type** | **Description** |
| prodid | integer | A unique numeric ID assigned to each product |
| catid | integer | A numerical ID that relates to a category the product belongs in |
| description | varchar(100) | A short description of the product |
| picture | mediumblob | A small picture of the product |
| price | decimal(6,2) | The current price of the product |
| quantity | integer | The amount of product currently on-hand in inventory |
| onsale | Boolean | A flag to mark if the price is currently marked down |

You may notice a couple data types in the products table that you haven't seen before. The *price* field uses the *decimal* data type. The decimal data type allows us to specify exactly how many decimal places should be in the data value.

Some programmers use a *floating point* data type for money values. This can get ugly, as there's no way to control how many decimal places are stored. If you have a value of six dollars, the database will only store it as a value of 6.

The decimal(6,2) data type defines that there will always be six digits in our value, and it ensures that there will always be two decimal places. This means we can store values up to $9,999.99 in this field.

The *onsale* field uses the *Boolean* data type (abbreviated BOOL in MySQL). This is a simple, one-bit value that gets set to either zero or one. The manager of the Food Store Web site can set an item to either be on sale (set to a value of 1) or not on sale (set to a value of zero). Later on, you'll see how to use this field in association with a simple check box in our administration area. All the manager will need to do is select a check box to indicate that an item is on sale.

Finally, I'm sure you're interested in the *picture* data field. Yes, we're going to store images in our database to display on the Web! This is a hot topic in the world of dynamic Web pages.

MySQL can store any type of binary data using the *Binary Large Object* (BLOB) data type. It lets you store anything from video clips to Excel spreadsheets. This feature can make your database extremely versatile. There's no limit as to what objects you can store for later retrieval.

There are four sizes of BLOB data types:

* Tinyblob - for data up to 255 bytes

 Blob - for data up to 65,535 bytes

 Mediumblob - for data up to 16,777,215 bytes

 Longblob - for data up to 4GB

The mediumblob data type that we'll use to store our pictures can handle data up to about 16 MB of size! We won't be using that much space for our images, but it's nice to know it's there if we need it.

**The Categories Table**

The *categories* table holds information on the product categories. We'll divide our Food Store into separate categories based on types of food. You probably wouldn't want to force your customers to scan through a single Web page looking for individual products. Instead, we'll divide our Web site into product categories to help limit the number of items (and pages) they have to wade through.

The categories table tracks the category names and includes two data fields:

* A *catid* integer data field, which defines a unique category ID for each category

 A *name* data field using a varchar(30) data type to define the name of the category

We'll allow our store managers to create and remove categories as they build the storefront. This table will help us keep track of what categories are available.

**The Customers Table**

The next thing you'll need to track is your customer information. You'll create a customers table that will include the following fields:

|  |  |  |
| --- | --- | --- |
| **The customers table** | | |
| **Data Field** | **Data Type** | **Description** |
| custid | int | A unique number assigned to each customer |
| lastname | varchar(40) | The customer's last name |
| firstname | varchar(40) | The customer's first name |
| address | varchar(100) | The customer's street address |
| city | varchar(30) | The customer's city |
| state | char(2) | The customer's state (as a two-letter identifier) |
| zip | varchar(5) | The customer's five-number ZIP code |
| phone | varchar(15) | The customer's phone number with area code |
| email | varchar(100) | The customer's e-mail address |
| password | varchar(41) | A password the customer can select to identify themselves |

There's nothing too exciting about the customers table. It stores all of the customer information in text format (with the obvious exception of the customer ID value that MySQL will assign automatically using the *AUTO\_INCREMENT* feature).

**Tracking Customer Orders**

We'll also need a way to track customer orders. We don't want to include orders as a field within the customer table, because a single customer could (hopefully) have more than one order during his or her lifetime. So, we'll create a separate table to handle customer orders, then link it back to the customers table so we know which customer placed which orders.

Another challenge we'll have is designing an orders table that lets us track what items a customer purchases in a given order. This can get tricky, since we're trying to combine data from two sources (tracking a customer order and then tracking the products in that customer order). To solve this problem, we'll break our customer orders into two separate tables.

The *orders* table contains the bare-bones information we need to track a customer order.

|  |  |  |
| --- | --- | --- |
| **The orders table** | | |
| **Data Field** | **Data Type** | **Description** |
| orderid | int | A unique number identifying each order |
| custid | int | The customer ID assigned to the customer in the customers table |
| date | datetime | The date and time the order was placed |
| status | varchar(10) | Text describing the status of the order, such as pending or shipped |

That will help us keep track of individual orders. Now we need to relate what products a customer purchases in a given order. You'll do that by creating the *order\_items* table.

|  |  |  |
| --- | --- | --- |
| **The order\_items table** | | |
| **Data Field** | **Data Type** | **Description** |
| orderid | integer | The numeric order ID assigned in the orders table |
| prodid | integer | The numeric product ID assigned in the products table |
| quantity | integer | The amount of the product purchased |
| price | decimal(6,2) | The selling price of the product |

The order\_items table records have a unique feature. Notice that no one data field works as a primary key. There can be multiple products purchased in a single order, and an individual product can be purchased in many different orders. What makes a record unique in the order\_items table is a combination of the orderid and prodid data fields.

This is an example of a *multi-field primary key*. MySQL allows us to select a combination of two or more fields to use as the primary key for the table. In this case, we'll use two data fields to uniquely define the primary key for each record.

That covers all of the data necessary for our customers and products. However, there's still one more piece of information we'll want to track. The Food Store application allows one or more managers to log in and manage the storefront. Once a manager logs in, he or she can add and modify products and process orders. You'll need to create a way to track managers who are allowed to do these functions. To do that, you simply include a table to track a manager account and password ‘ the *admins* table.

|  |  |  |
| --- | --- | --- |
| **The admins table** | | |
| **Data Field** | **Data Type** | **Description** |
| userid | varchar(8) | A unique eight-character (or less) user name |
| password | varchar(41) | The encrypted password for the user name |
| name | varchar(100) | A full name to identify the user |

That defines the tables we'll be using in the Food Store application. In the next chapter, you'll start building the tables, and we'll look at a way to manage the data in our tables by taking advantage of the relationships we created.