Working with Databases and Managing State

Chapters 14 and 16

Randy Connolly and Ricardo Hoar

Fundamentals of Web Development

http://www.funwebdev.com

Objectives

1. Databases and Web Development
2. Database APIs
3. Managing a MySQL Database
4. Accessing MySQL in PHP
5. The Problem of State
6. Cookies
7. Self-Reading: Case Study Schemas
8. Self-Reading: Sample Database Techniques

Section 1 of 8

DATABASES AND WEB DEVELOPMENT

Databases and Web Development

This chapter covers the core principles of relational **Database Management Systems (DBMSs)**.

All database management systems are capable of

- managing large amounts of data,
- maintaining data integrity,
- responding to many queries,
- creating indexes and triggers, and more.

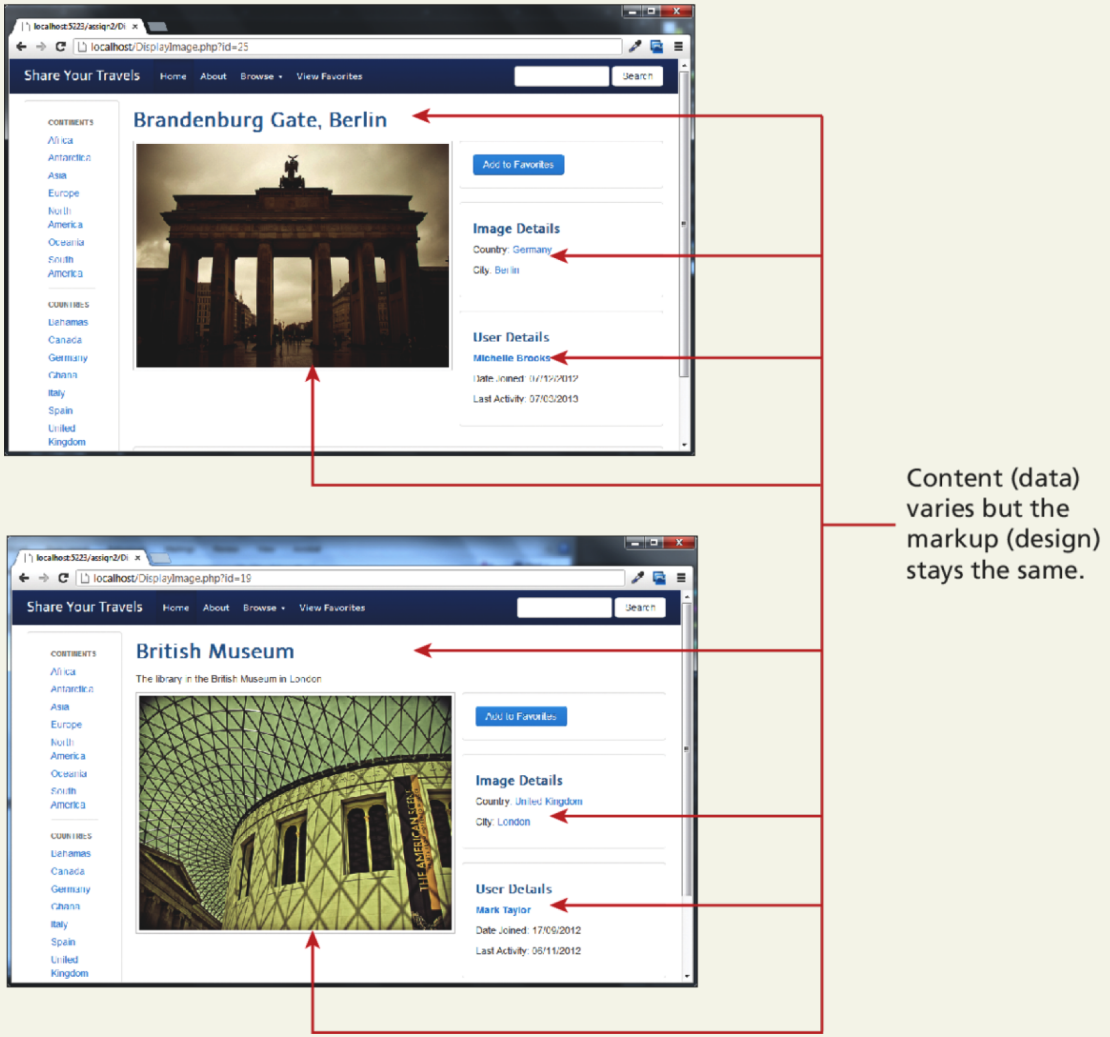
The term **database** can refer to both the software (i.e., to the DBMS) and to the data that is managed by the DBMS.

The Role of Databases

In Web Development

Databases provide a way to implement one of the most important software design principles:

one should separate that which varies (dynamic content) from that which stays the same (template).



Separate that which varies

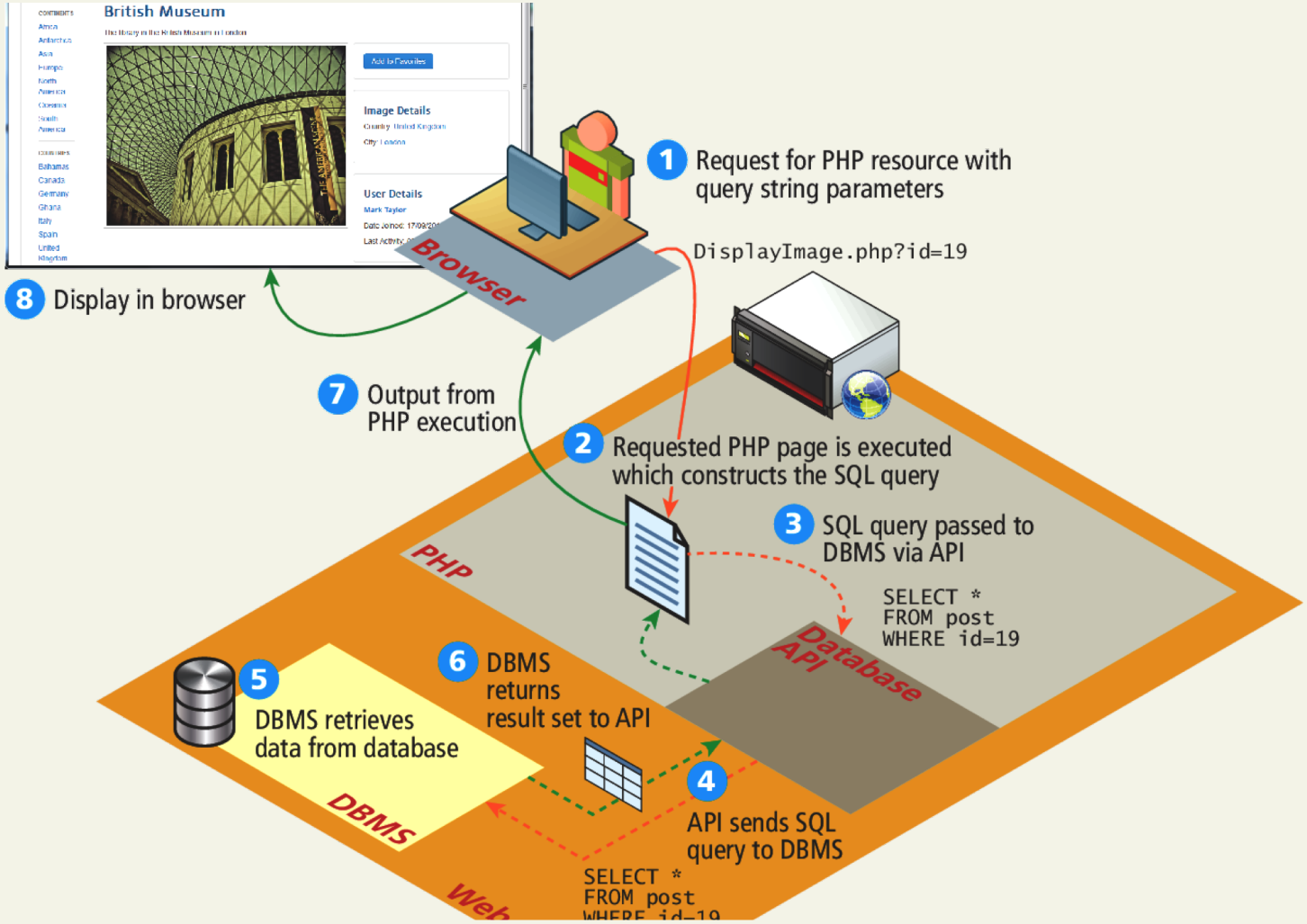
That is, use a DB to store the content of pages.

The program (PHP) determines which data to display or action to make, often from information in the GET or POST query string, and then uses a database API to interact with the database.

Although the same separation could be achieved by storing content in files on the server, databases offer intuitive and optimized systems that do far more than a file-based design that would require custom-built reading, parsing, and writing functions.

That Which Changes

Can be stored in the DB



Section 2 of 8

DATABASE APIS

API

Application Programming Interface

API stands for application programming interface and in general refers to the classes, methods, functions, and variables that your application uses to perform some task.

Some database APIs work only with a specific type of database; others are cross-platform and can work with multiple databases.

PHP MySQL APIs

There is more than 1

- **MySQL extension**. This was the original extension to PHP for working with MySQL and has been replaced with the newer mysqli extension. This procedural API should now only be used with versions of MySQL older than 4.1.3
- **mysqli extension**. The MySQL Improved extension takes advantage of features of versions of MySQL after 4.1.3. This extension provides **both** a procedural and an object-oriented approach. This extension also supports most of the latest features of MySQL.
- **PHP data objects (PDOs)**. This object-oriented API has been available since PHP 5.1 and provides an **abstraction layer** (i.e., a set of classes that hide the implementation details for some set of functionality) that with the appropriate drivers can be used with *any* database, and not just MySQL databases. However, it is not able to make use of all the latest features of MySQL.

Section 3 of 8

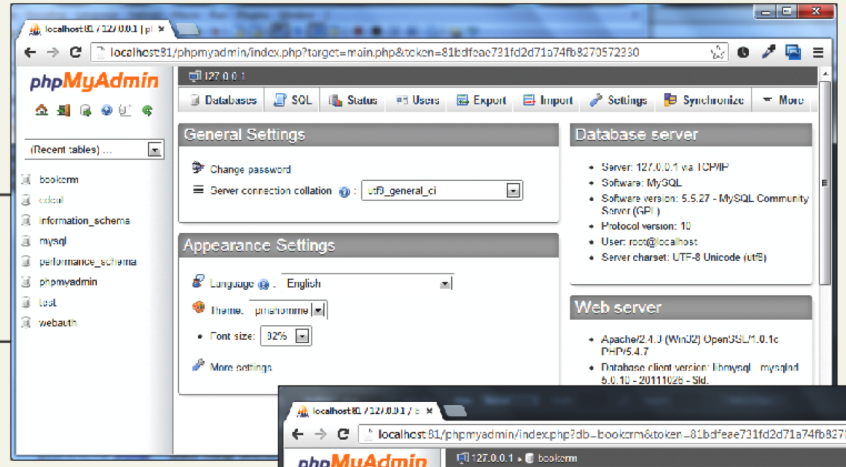
MANAGING A MYSQL DATABASE

How to Access the DBMS

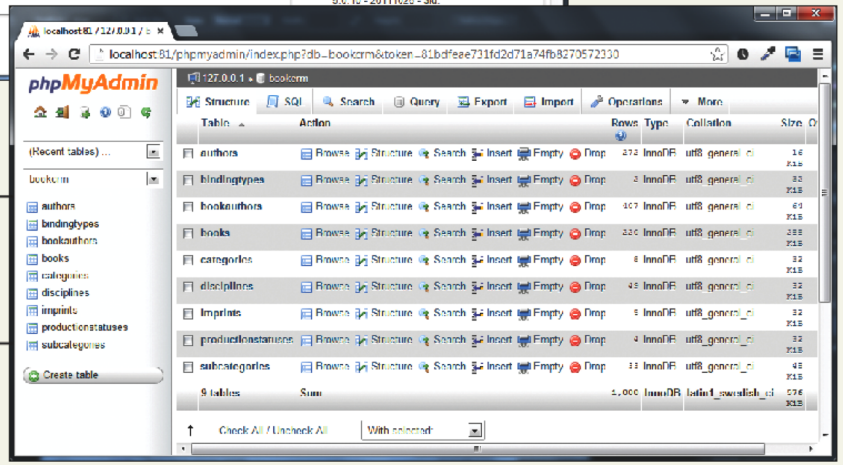
PHPMyAdmin

A popular web-based front-end (written in PHP) called **phpMyAdmin** allows developers to access management tools through a web portal.

MySQL has a number of predefined databases it uses for its own operation.



phpMyAdmin allows you to view and manipulate any table in a database.



How to Access the DBMS

PHPMyAdmin

Live demo:

- Create DB University
- Create table users:
  - PK
  - Name
  - Role

  - In XAMPP, the default DB username is ‘root’ and the password is an empty string ‘’.
  - In MAMP, the default DB username is ‘root’ and the password is ‘root’.

Section 4 of 8

ACCESSING MYSQL IN PHP

Database Connection Algorithm

No matter what API you use, the basic database connection algorithm is the same:

1. Connect to the database.
2. Handle connection errors.
3. Execute the SQL query.
4. Process the results.
5. Free resources and close connection.

Database Connection Algorithm

An illustration through example

```php
// modify these variables for your installation
$host = "localhost";
$database = "bookcrm";
$user = "testuser";
$pass = "mypassword";

$connection = mysqli_connect($host, $user, $pass, $database);
```

LISTING 11.3 Connecting to a database with mysqli (procedural)

```php
// modify these variables for your installation
$connectionString = "mysql:host=localhost;dbname=bookcrm";
$user = "testuser";
$pass = "mypassword";

$pdo = new PDO($connectionString, $user, $pass);
```

LISTING 11.4 Connecting to a database with PDO (object-oriented)

Storing Connection Details

Hard-coding the database connection details in your code is not ideal.

Connection details almost always change as a site moves from development, to testing, to production.

We should move these connection details out of our connection code and place it in some central location.

```
<?php
define('DBHOST', 'localhost');
define('DBNAME', 'bookcrm');
define('DBUSER', 'testuser');
define('DBPASS', 'mypassword');
?>
```

LISTING 11.5 Defining connection details via constants in a separate file (config.php)

Handling Connection Errors

We need to handle potential connection errors in our code.

- Procedural **mysqli** techniques use conditional (if...else) statements on the returned object from the connection attempt.
- The **PDO** technique uses try-catch which relies on thrown exceptions when an error occurs.

Handling Connection Errors

Procedural Approach

```php
$connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME);

// mysqli_connect_error returns string description of the last
// connect error
$error = mysqli_connect_error();
if ($error != null) {
   $output = "<p>Unable to connect to database<p>" . $error;
// Outputs a message and terminates the current script
   exit($output);
}
```

LISTING 11.7 Handling connection errors with mysqli (version 1)

```php
$connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME);

// mysqli_connect_errno returns the last error code
if ( mysqli_connect_errno() ) {
   die( mysqli_connect_error() );  // die() is equivalent to exit()
}
```

LISTING 11.8 Handling connection errors with mysqli (version 2)

Handling Connection Errors

Object-Oriented PDO with try-catch

```
try {
    $connString = "mysql:host=localhost;dbname=bookcrm";
    $user = "DBUSER";
    $pass = "DBPASS";

    $pdo = new PDO($connString,$user,$pass);
    …
}
catch (PDOException $e) {
    die( $e->getMessage() );
}
```

LISTING 11.9 Handling connection errors with PDO

Execute the Query

Procedural and Object-Oriented

```php
$sql = "SELECT * FROM Categories ORDER BY CategoryName";

// returns a mysqli_result object
$result = mysqli_query($connection, $sql);
```

LISTING 11.11 Executing a SELECT query (mysqli)

```php
$sql = "SELECT * FROM Categories ORDER BY CategoryName";

// returns a PDOStatement object
$result = $pdo->query($sql);
```

LISTING 11.12 Executing a SELECT query (pdo)

Both return a **result set**, which is a type of cursor or pointer to the returned data

Queries that don't return data

Procedural and Object-Oriented

```
$sql = "UPDATE Categories SET CategoryName='Web' WHERE
        CategoryName='Business'";

if ( mysqli_query($connection, $sql) ) {
   $count = mysqli_affected_rows($connection);
   echo "<p>Updated " . $count . " rows</p>";
}
```

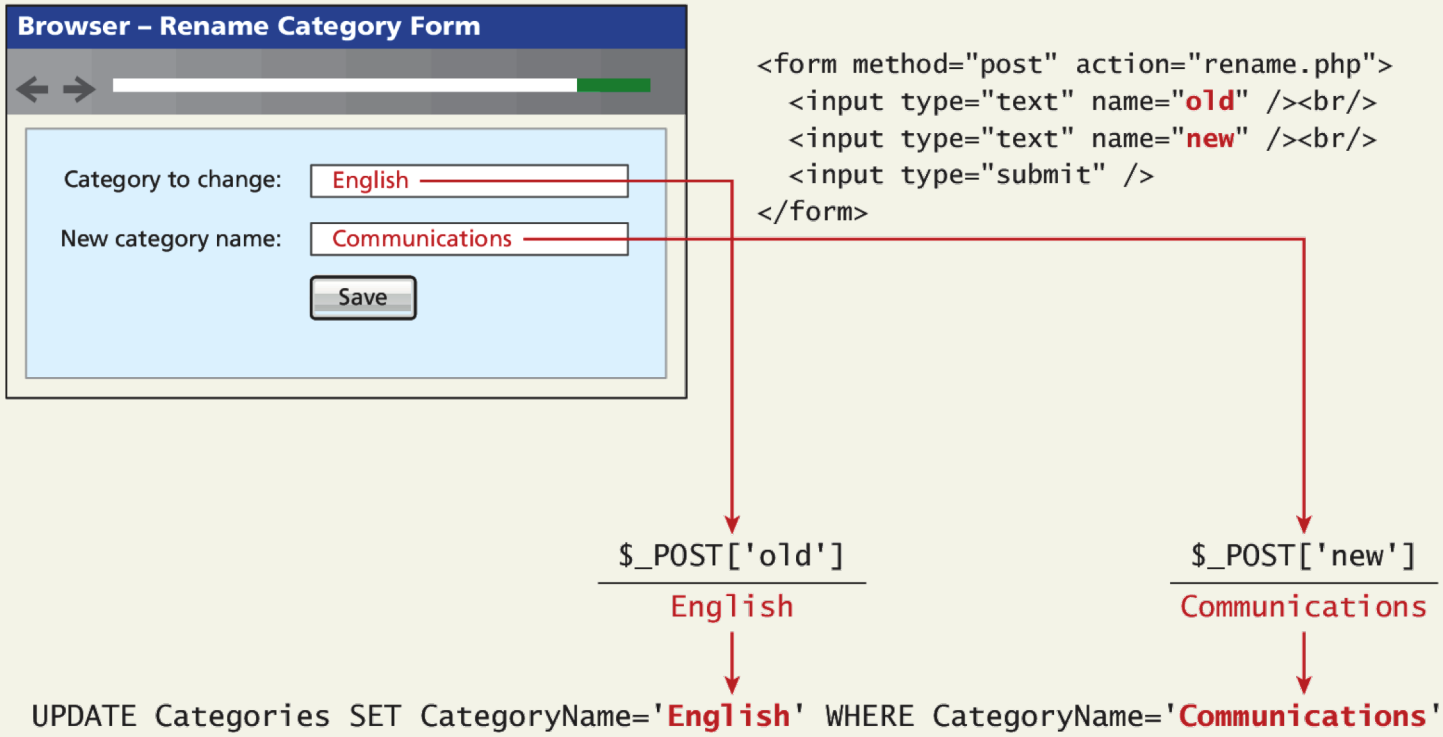
LISTING 11.13 Executing a query that doesn't return data (mysqli)

```
$sql = "UPDATE Categories SET CategoryName='Web' WHERE
        CategoryName='Business'";
$count = $pdo->exec($sql);
echo "<p>Updated " . $count . " rows</p>";
```

LISTING 11.14 Executing a query that doesn't return data (PDO)

Integrating User Data

Say, using an HTML form posted to the PHP script



Integrating User Data

Not everyone is nice.

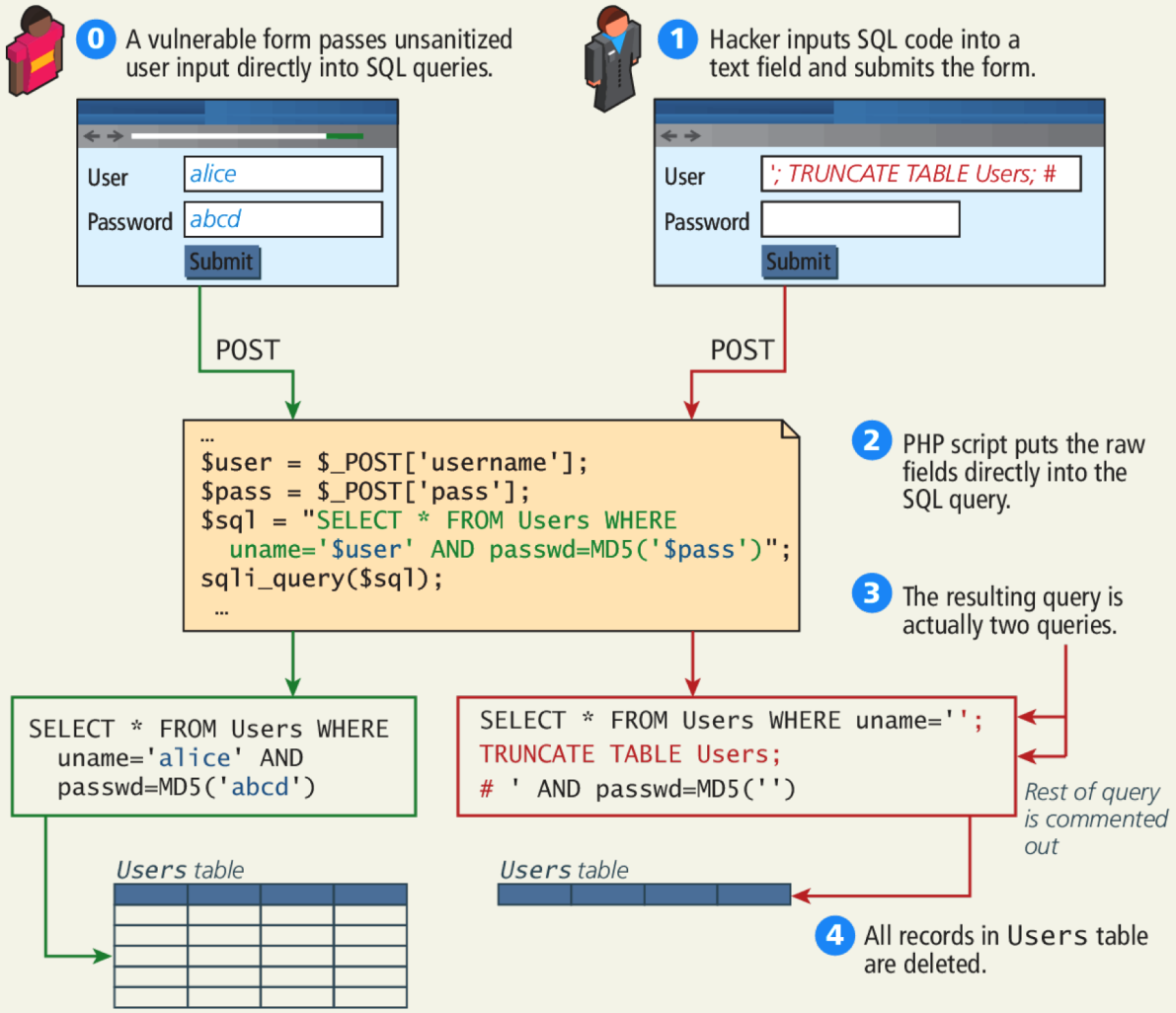
```
$from = $_POST['old'];
$to = $_POST['new'];
$sql = "UPDATE Categories SET CategoryName='$to' WHERE
        CategoryName='$from'";

$count = $pdo->exec($sql);
```

LISTING 11.15 Integrating user input into a query (first attempt)

While this does work, it opens our site to one of the most common web security attacks, the **SQL injection attack.**

SQL Injection Illustration

From Chapter 16



Defend against attack

Distrust user input

The SQL injection class of attack can be protected against by

- **Sanitizing** user input
- Using **Prepared Statements**

Sanitize User Input

Quick and easy

Each database system has functions to remove any special characters from a desired piece of text. In MySQL, user inputs can be sanitized in PHP using the mysqli_real_escape_string() method or, if using PDO, the quote() method

```
$from = $pdo->quote($from);
$to = $pdo->quote($to);
$sql = "UPDATE Categories SET CategoryName=$to WHERE
        CategoryName=$from";

$count = $pdo->exec($sql);
```

LISTING 11.16 Sanitizing user input before use in an SQL query

Prepared Statements

Better in general

A **prepared statement** is actually a way to improve performance for queries that need to be executed multiple times.

When MySQL creates a prepared statement, it optimizes it so that it has superior performance for multiple requests.

It also integrates sanitization into each user input automatically, thereby protecting us from SQL injection.

Prepared Statements

mysqli

```php
// retrieve parameter value from query string
$id = $_GET['id'];

// construct parameterized query - notice the ? parameter
$sql = "SELECT Title, CopyrightYear FROM Books WHERE ID=?";

// create a prepared statement

if ($statement = mysqli_prepare($connection, $sql)) {
    // Bind parameters s - string, b - blob, i - int, etc
    mysqli_stmt_bindm($statement, 'i', $id);

    // execute query
    mysqli_stmt_execute($statement);

   // learn in next section how to access the returned data
   …
}
```

LISTING 11.17 Using a prepared statement (mysqli)

Prepared Statements

PDO

```php
// retrieve parameter value from query string
$id = $_GET['id'];

/* method 1 */
$sql = "SELECT Title, CopyrightYear FROM Books WHERE ID = ?";
$statement = $pdo->prepare($sql);
$statement->bindValue(1, $id);
$statement->execute();

/* method 2 */
$sql = "SELECT Title, CopyrightYear FROM Books WHERE ID = :id";
$statement = $pdo->prepare($sql);
$statement->bindValue(':id', $id);
$statement->execute();
```

LISTING 11.18 Using a prepared statement (PDO)

Prepared Statements

Comparison of two techniques

```
/* technique 1 - question mark placeholders */
$sql = "INSERT INTO books (ISBN10, Title, CopyrightYear, ImprintId,
        ProductionStatusId, TrimSize, Description) VALUES
        (?,?,?,?,?,?,?)";
$statement = $pdo->prepare($sql);
$statement->bindValue(1, $_POST['isbn']);
$statement->bindValue(2, $_POST['title']);
$statement->bindValue(3, $_POST['year']);
$statement->bindValue(4, $_POST['imprint']);
$statement->bindValue(4, $_POST['status']);
$statement->bindValue(6, $_POST['size']);
$statement->bindValue(7, $_POST['desc']);
$statement->execute();

/* technique 2 - named parameters */
$sql = "INSERT INTO books (ISBN10, Title, CopyrightYear, ImprintId,
        ProductionStatusId, TrimSize, Description) VALUES (:isbn,
        :title, :year, :imprint, :status, :size, :desc) ";
$statement = $pdo->prepare($sql);
$statement->bindValue(':isbn', $_POST['isbn']);
$statement->bindValue(':title', $_POST['title']);
$statement->bindValue(':year', $_POST['year']);
$statement->bindValue(':imprint', $_POST['imprint']);
$statement->bindValue(':status', $_POST['status']);
$statement->bindValue(':size', $_POST['size']);
$statement->bindValue(':desc', $_POST['desc']);
$statement->execute();
```

LISTING 11.19 Using named parameters (PDO)

Process Query Results

mysqli

```
$sql = "SELECT * FROM Categories ORDER BY CategoryName";
// run the query
if ($result = mysqli_query($connection, $sql)) {
   // fetch a record from result set into an associative array
   while($row = mysqli_fetch_assoc($result))
    {
         // the keys match the field names from the table
         echo $row['ID'] . " - " . $row['CategoryName'] ;
         echo "<br/>";
    }
}
```

LISTING 11.20 Looping through the result set (mysqli—not prepared statements)

```
$sql = "SELECT * FROM Categories ORDER BY CategoryName";
$result = $pdo->query($sql);

while ( $row = $result->fetch() ) {
      echo $row['ID'] . " - " . $row['CategoryName'] . "<br/>";
}
```

LISTING 11.22 Looping through the result set (PDO)

Fetch into an object

Instead of an array

Consider the following (very simplified) class:

```
class Book {

        public $id;
        public $title;
        public $copyrightYear;
        public $description;
}
```

Fetch into an object

Instead of an array

```
$id = $_GET['id'];
$sql = "SELECT id, title, copyrightYear, description FROM Books
  WHERE id= ?";
$statement = $pdo->prepare($sql);
$statement->bindValue(1, $id);
$statement->execute();

$b = $statement->fetchObject('Book');
echo 'ID: ' . $b->id . '<br/>';
echo 'Title: ' . $b->title . '<br/>';
echo 'Year: ' . $b->copyrightYear . '<br/>';
echo 'Description: ' . $b->description . '<br/>';
```

LISTING 11.23 Populating an object from a result set (PDO)

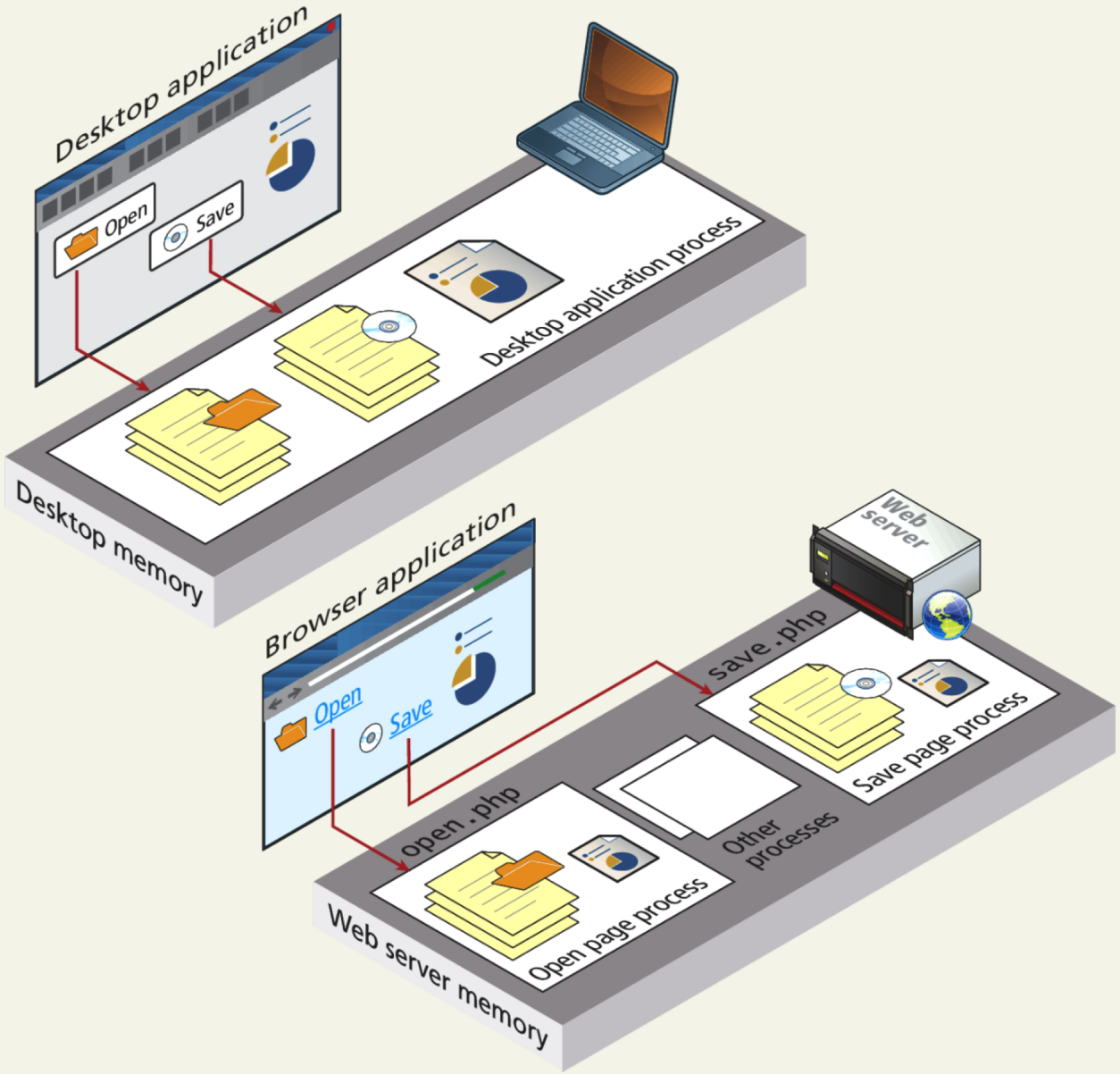
The property names must match exactly (including the case) the field names in the table(s) in the query

Section 5 of 8

THE PROBLEM OF STATE IN WEB APPLICATIONS

State in Web Applications

Not like a native application



State in Web Applications

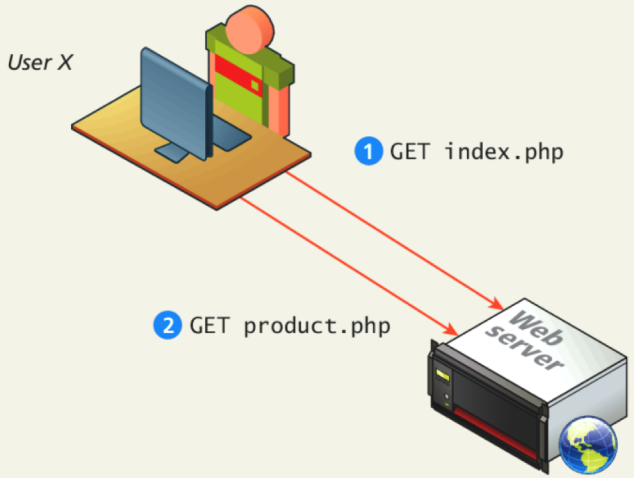
Not like a desktop application

Unlike the unified single process that is the typical desktop application, a web application consists of a series of disconnected HTTP requests to a web server where each request for a server page is essentially a request to run a separate program.

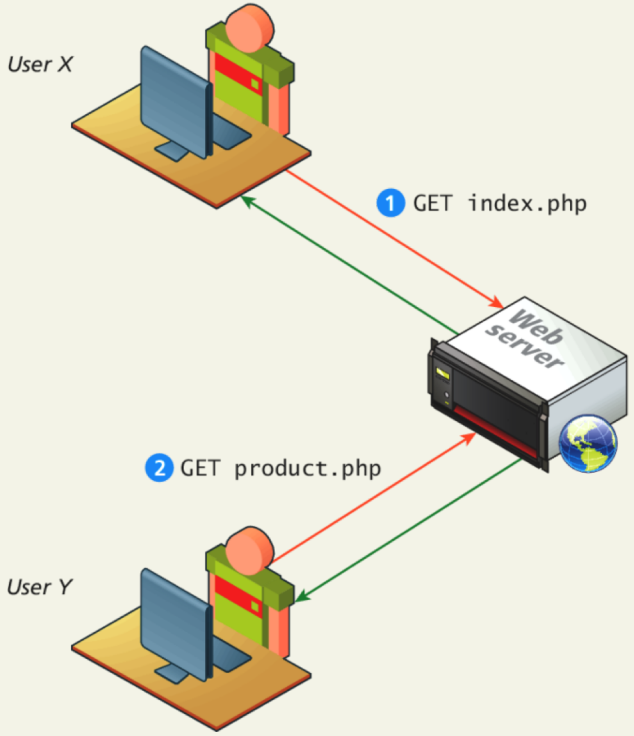
The HTTP protocol does not, without programming intervention, distinguish two requests by one source from two requests from two different sources

State in Web Applications

What's the issue?

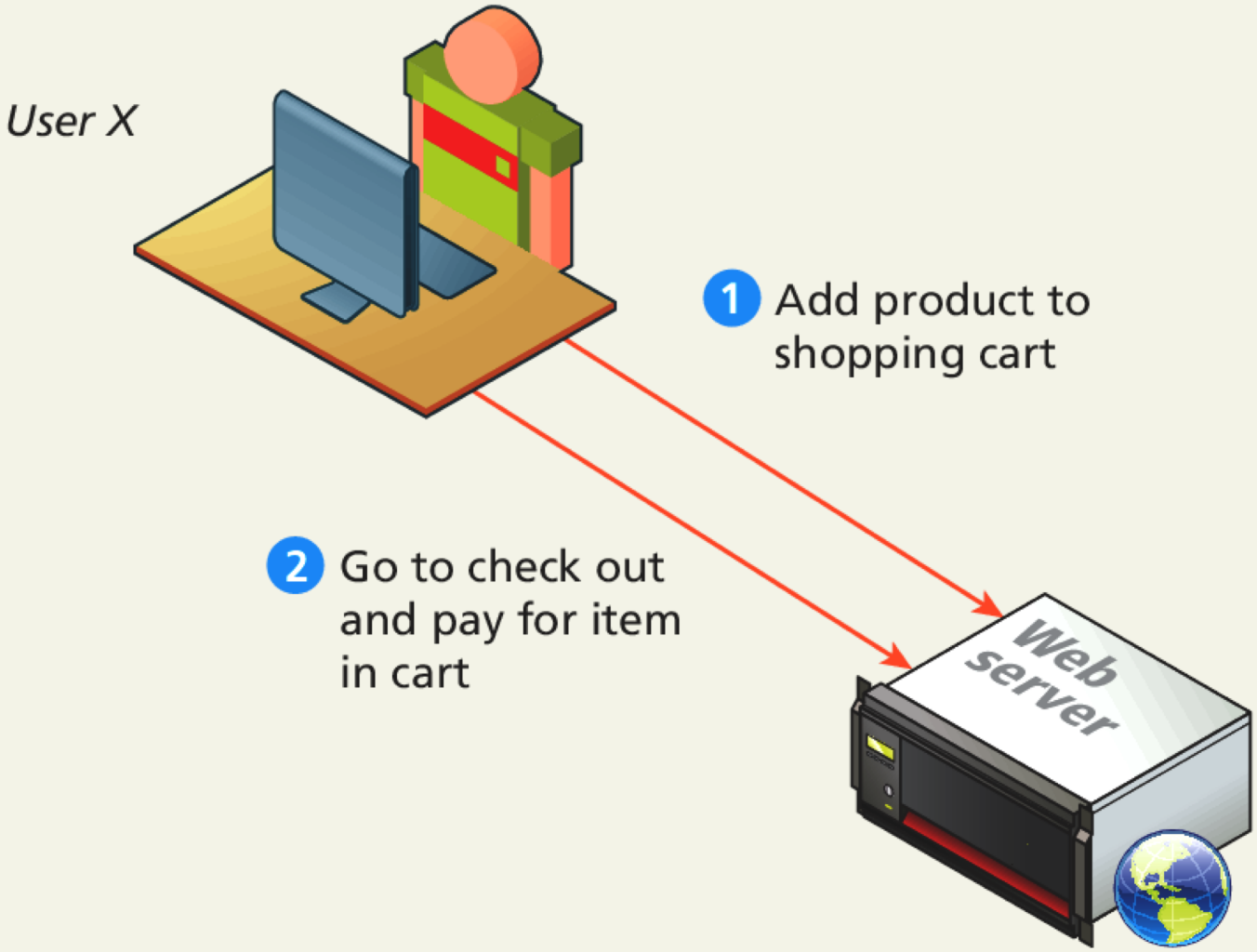


... is for the server not really any different than ...



State in Web Applications

What's the desired outcome



State in Web Applications

How do we reach our desired outcome?

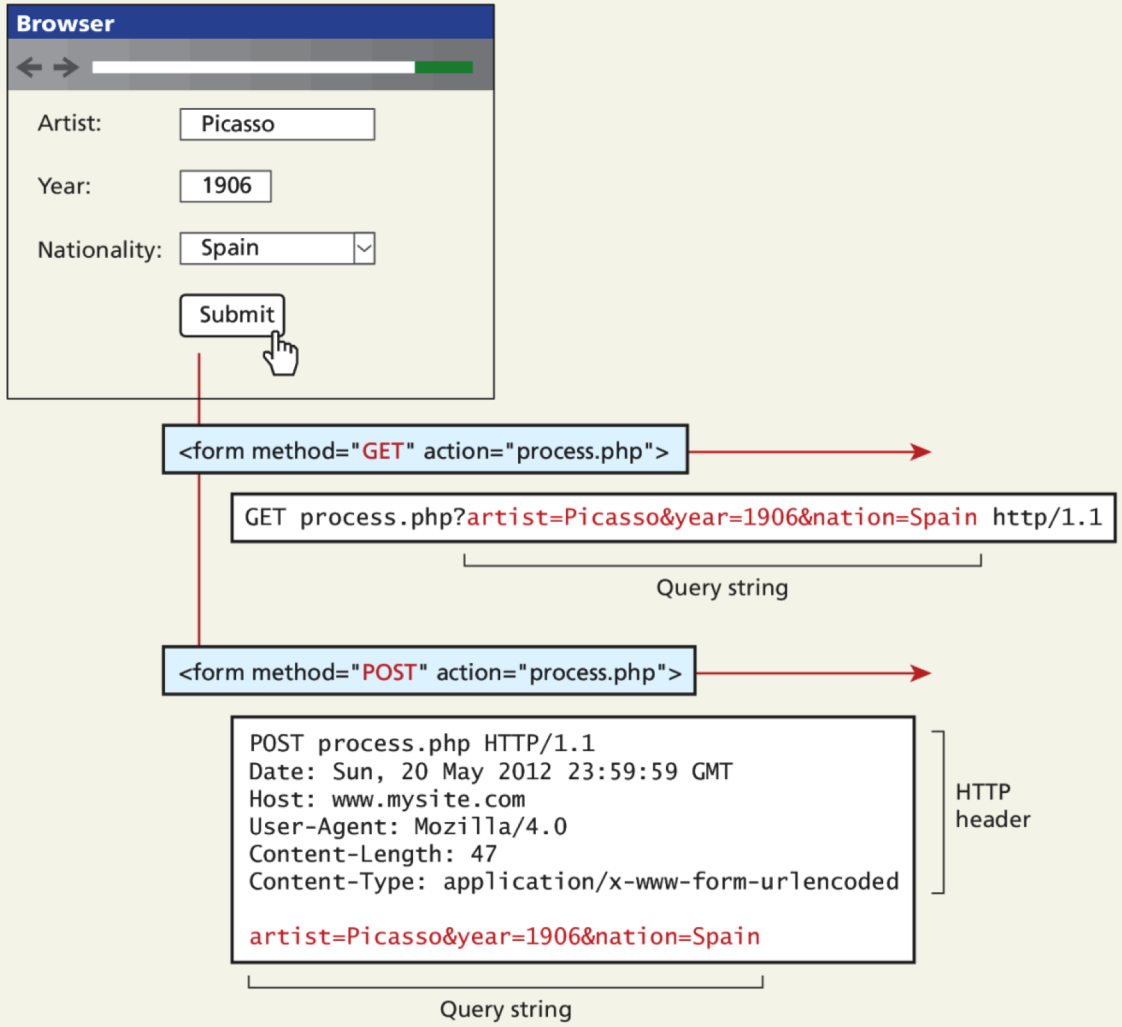
What mechanisms are available within HTTP to pass information to the server in our requests?

In HTTP, we can pass information using:

- Query strings
- Cookies

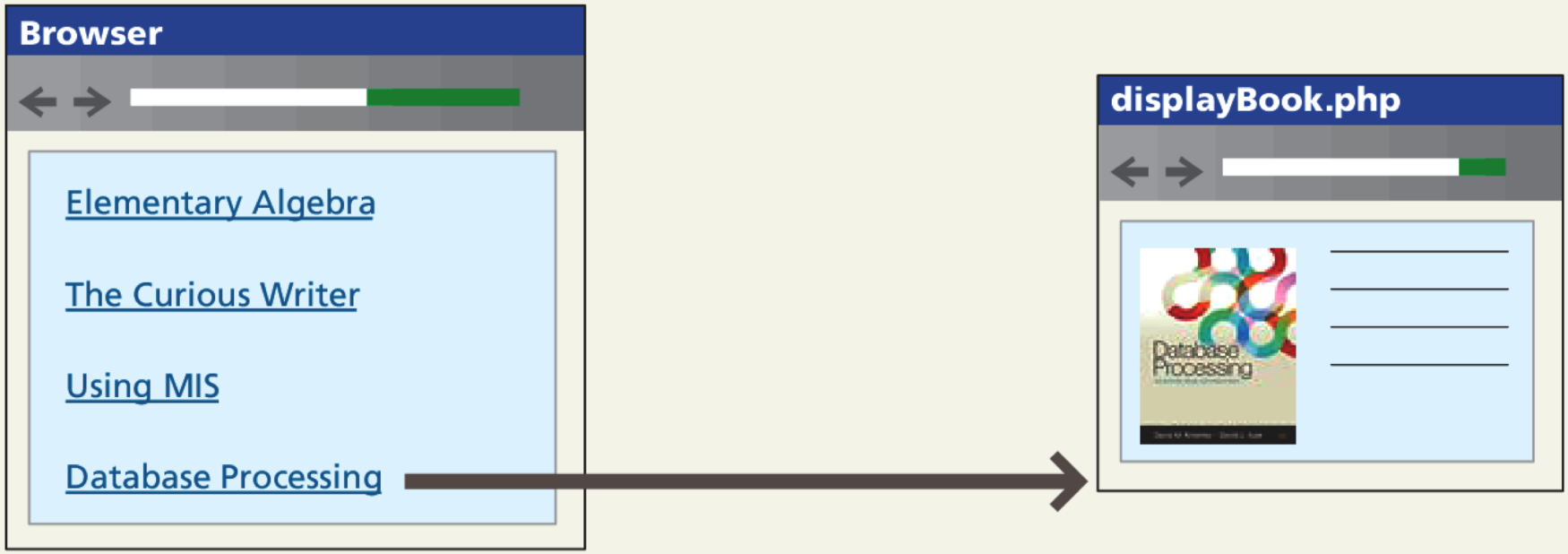
Info in Query Strings

Send information via GET or POST using forms



Info in Query Strings

Add GET information in the link




```
<a href="displayBook.php?isbn=0132145375">Database Processing</a>
```

Query string

Section 6 of 8

COOKIES AND SESSIONS

Cookies and Sessions

- In each HTTP request, the client can inform the server of any previous interactions.
- This can be managed using either Cookies or session variables.
- **Cookies** are key/value pairs stored in the client side and sent with every HTTP request to the server.
- **Cookies** can be persistent.
- **Session variables** are key/value pairs that belong to clients but are stored in the server side.
- **Session variables** cannot be persistent (i.e. destroyed when the session is terminated).

Cookies

Cookies are a client-side approach for persisting state information.

They are name=value pairs that are saved within one or more text files that are **managed by the browser**.

When a client requests a resource from the server for the first time, the **server can send back** the requested resources **along with some cookies**.

The client then will **send these cookies** each time it requests other sources from the **same server**.

Cookies

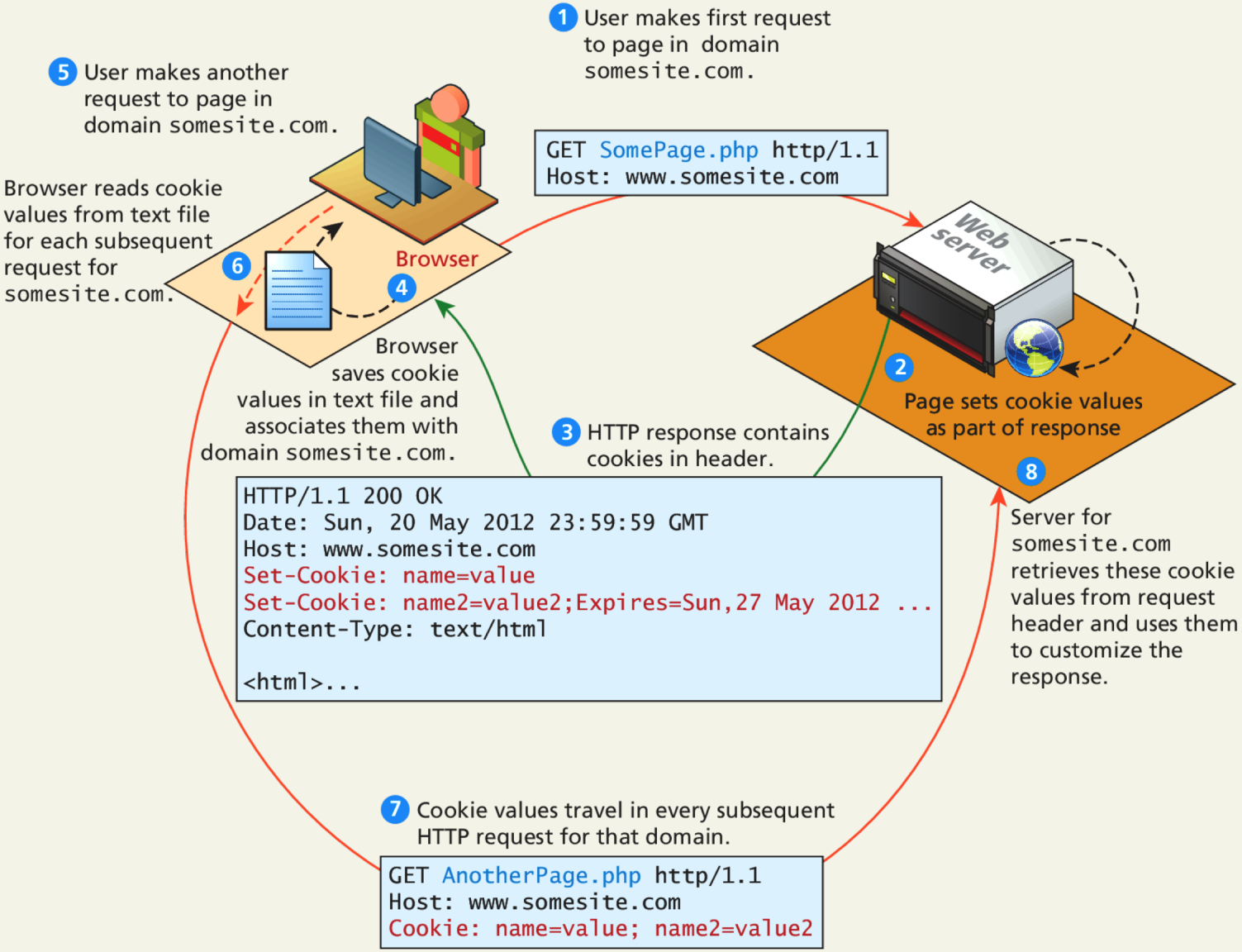
How do they Work?

While cookie information is stored and retrieved by the browser, the information in a cookie travels within the HTTP header.

- Sites that use cookies should not depend on their availability for critical features
- The user can delete cookies or tamper with them

Cookies

How do they Work?



Cookies

Two kinds of Cookies

- A **session cookie** has no expiry stated and thus will be deleted at the end of the user browsing session.
    - Not the same as session variables: they are stored in the client's machine.
- **Persistent cookies** have an expiry date specified;

Using Cookies

Writing a cookie

```php
<?php
    // add 1 day to the current time for expiry time
    $expiryTime = time()+60*60*24;

    // create a persistent cookie
    $name = "Username";
    $value = "Ricardo";
    setcookie($name, $value, $expiryTime);
?>
```

LISTING 13.1 Writing a cookie

It is important to note that cookies must be written before any other page output.

Using Cookies

Reading a cookie

```php
<?php
   if( !isset($_COOKIE['Username']) ) {
      //no valid cookie found
   }
   else {
      echo "The username retrieved from the cookie is:";
      echo $_COOKIE['Username'];
   }
?>
```

LISTING 13.2 Reading a cookie

Using Cookies

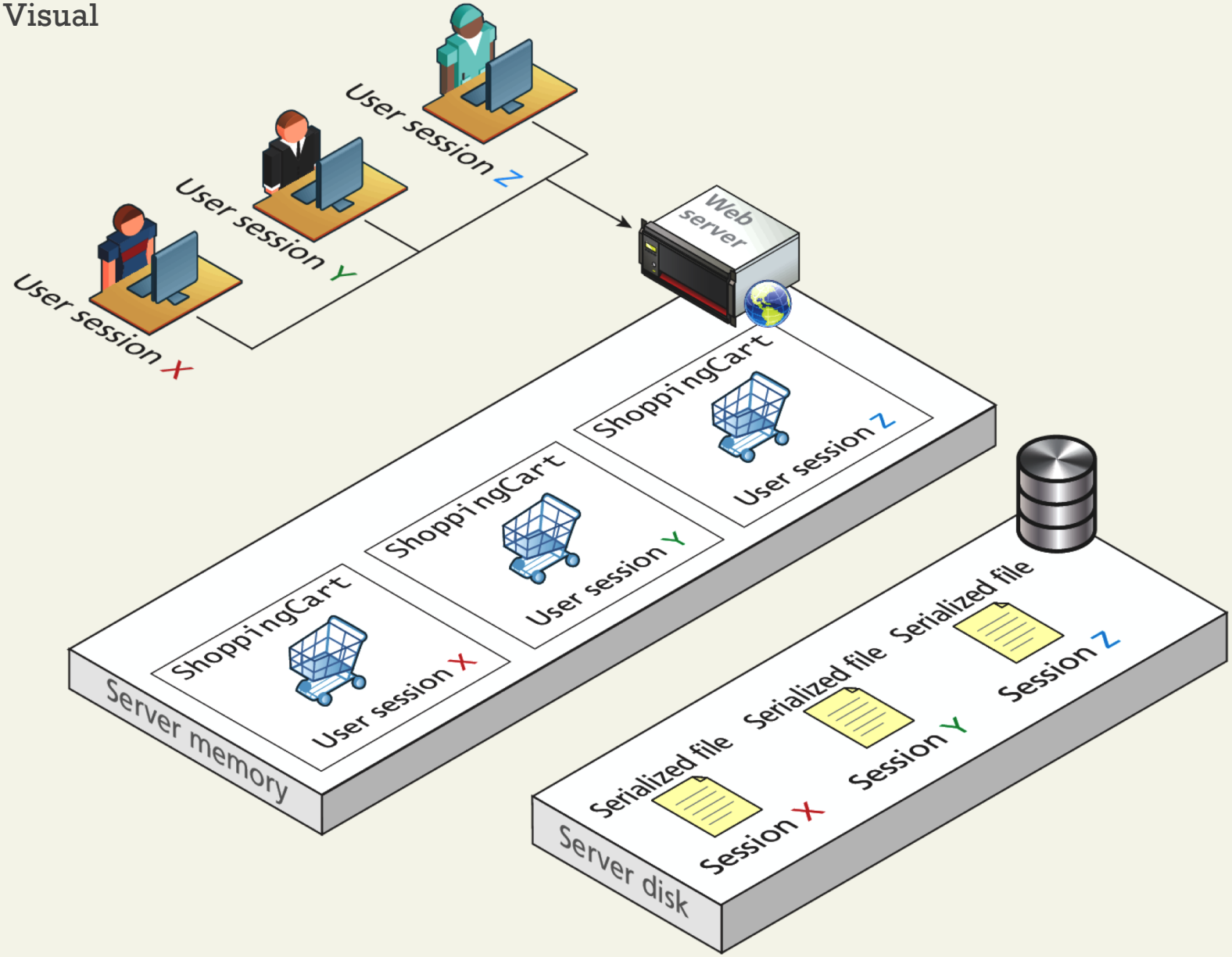
Common usages

In addition to being used to track authenticated users and shopping carts, cookies can implement:

- "Remember me" persistent cookie
- Store user preferences
- Track a user's browsing behavior
- Use Cookies for user **identification** NOT **authorization**
- **For authorization: use Session Variables**

Session State

Visual



Session State

All modern web development environments provide some type of session state mechanism.

Session state is a server-based state mechanism that lets web applications store and retrieve objects of any type for each unique user session.

Session state is ideal for storing more complex objects or data structures that are associated with a user session.

- In PHP, session state is available to the via the $_SESSION variable
- Must use session_start() to enable sessions.

Session State

Accessing State

```php
<?php

session_start();

if ( isset($_SESSION['user']) ) {
   // User is logged in
}
else {
   // No one is logged in (guest)
}
?>
```

LISTING 13.5 Accessing session state

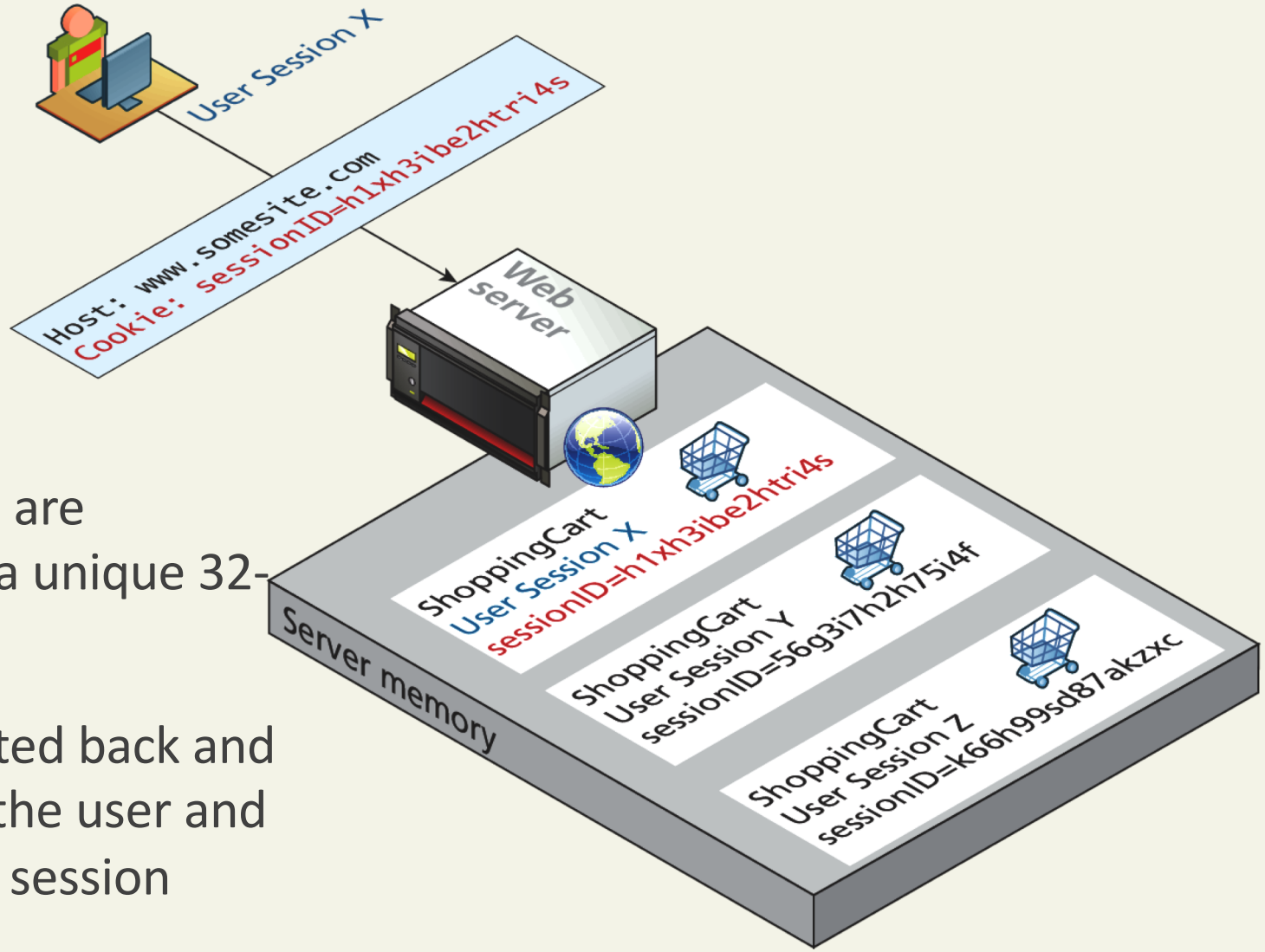
Session State

Checking Session existence

```php
<?php
include_once("ShoppingCart.class.php");

session_start();

// always check for existence of session object before accessing it
if ( !isset($_SESSION["Cart"]) ) {
   //session variables can be strings, arrays, or objects, but
   // smaller is better
   $_SESSION["Cart"] = new ShoppingCart();
}
$cart = $_SESSION["Cart"];
?>
```

LISTING 13.6 Checking session existence

How does state session work?



Sessions in PHP are identified with a unique 32-byte session ID.

This is transmitted back and forth between the user and the server via a session cookie

How does state session work?

It's magic right?

- For a brand new session, PHP assigns an initially empty dictionary-style collection that can be used to hold any state values for this session.
- When the request processing is finished, the session state is saved to some type of state storage mechanism, called a session state provider
- When a new request is received for an already existing session, the session's dictionary collection is filled with the previously saved session data from the session state provider.

Session State

- Since session variables are stored in the server, they are more secure than the Cookie variables.
- If a page requires authorization (i.e. should this user view this content?) -> use session variables.

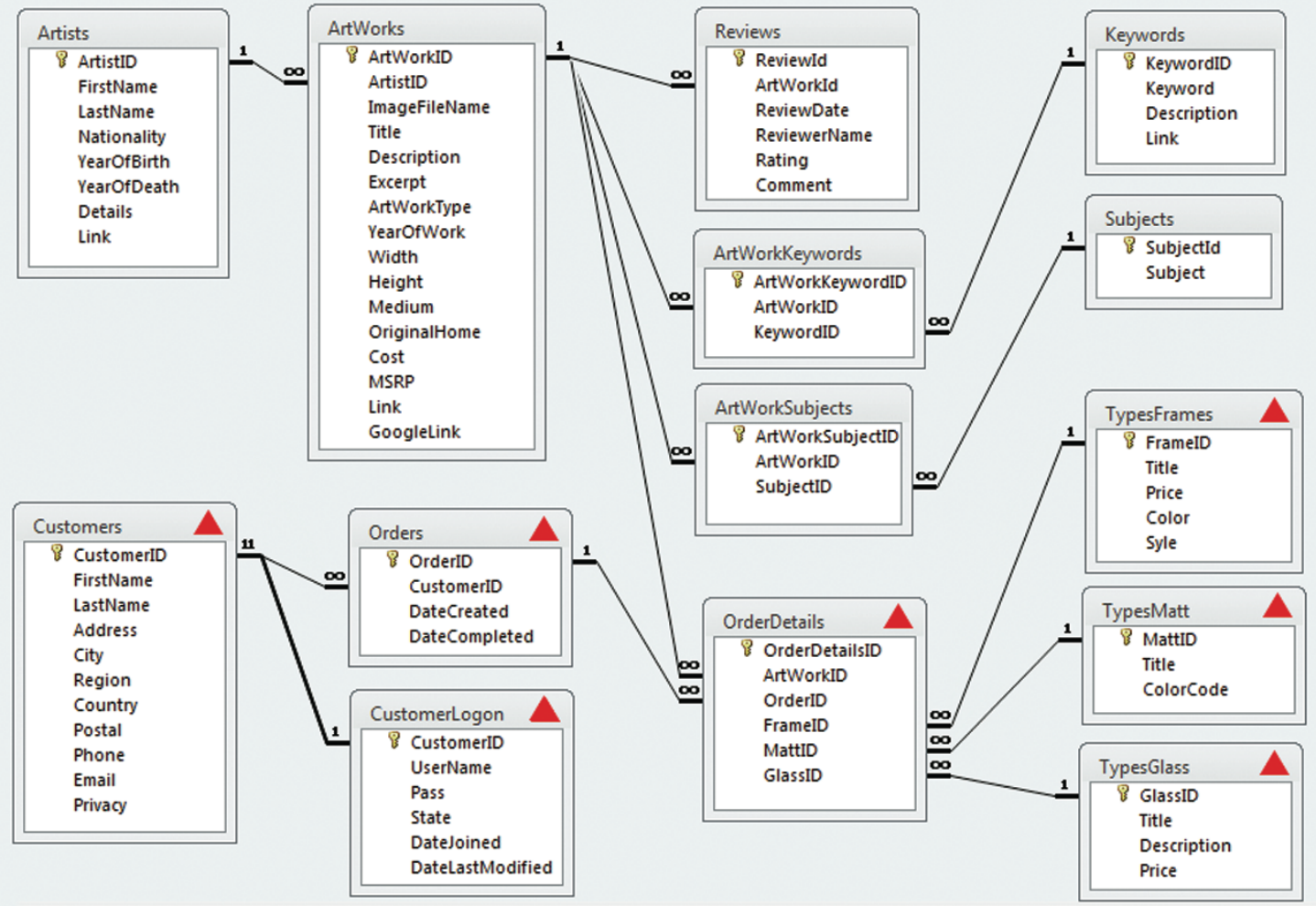
What You've Learned

1. Databases and Web Development
2. Database APIs
3. Managing a MySQL Database
4. Accessing MySQL in PHP
5. The Problem of State
6. Cookies
7. Self-Reading: Case Study Schemas
8. Self-Reading: Sample Database Techniques

SELF READING

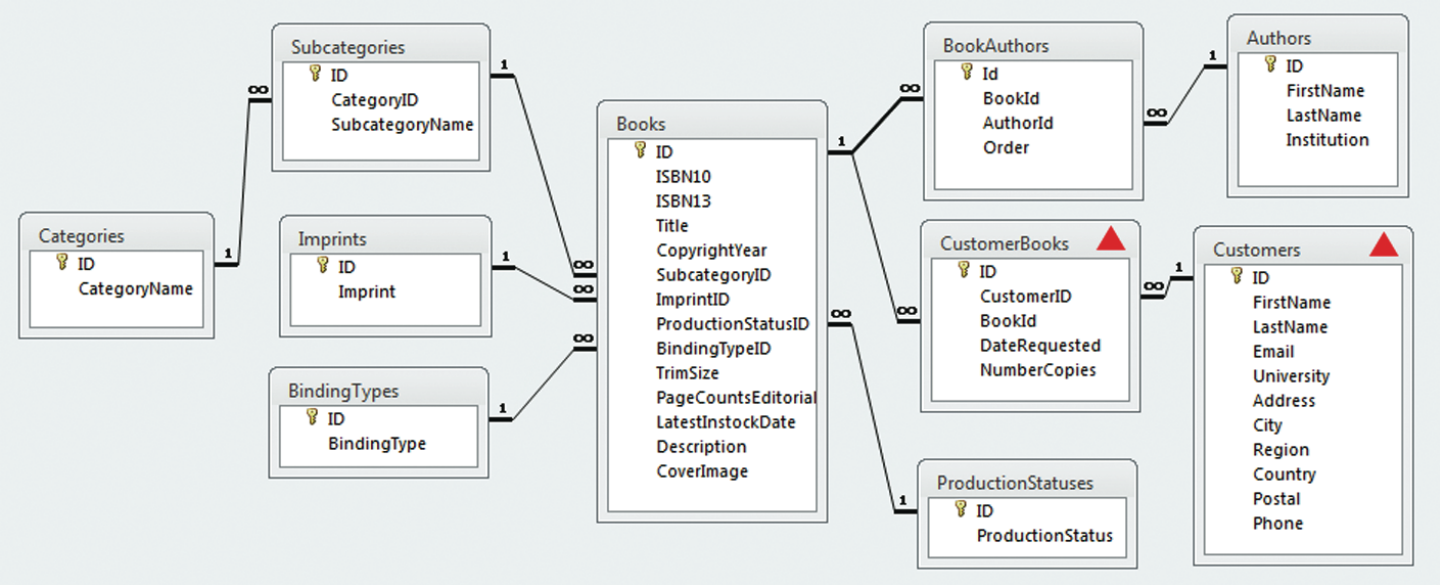
CASE STUDY SCHEMAS

Art Database

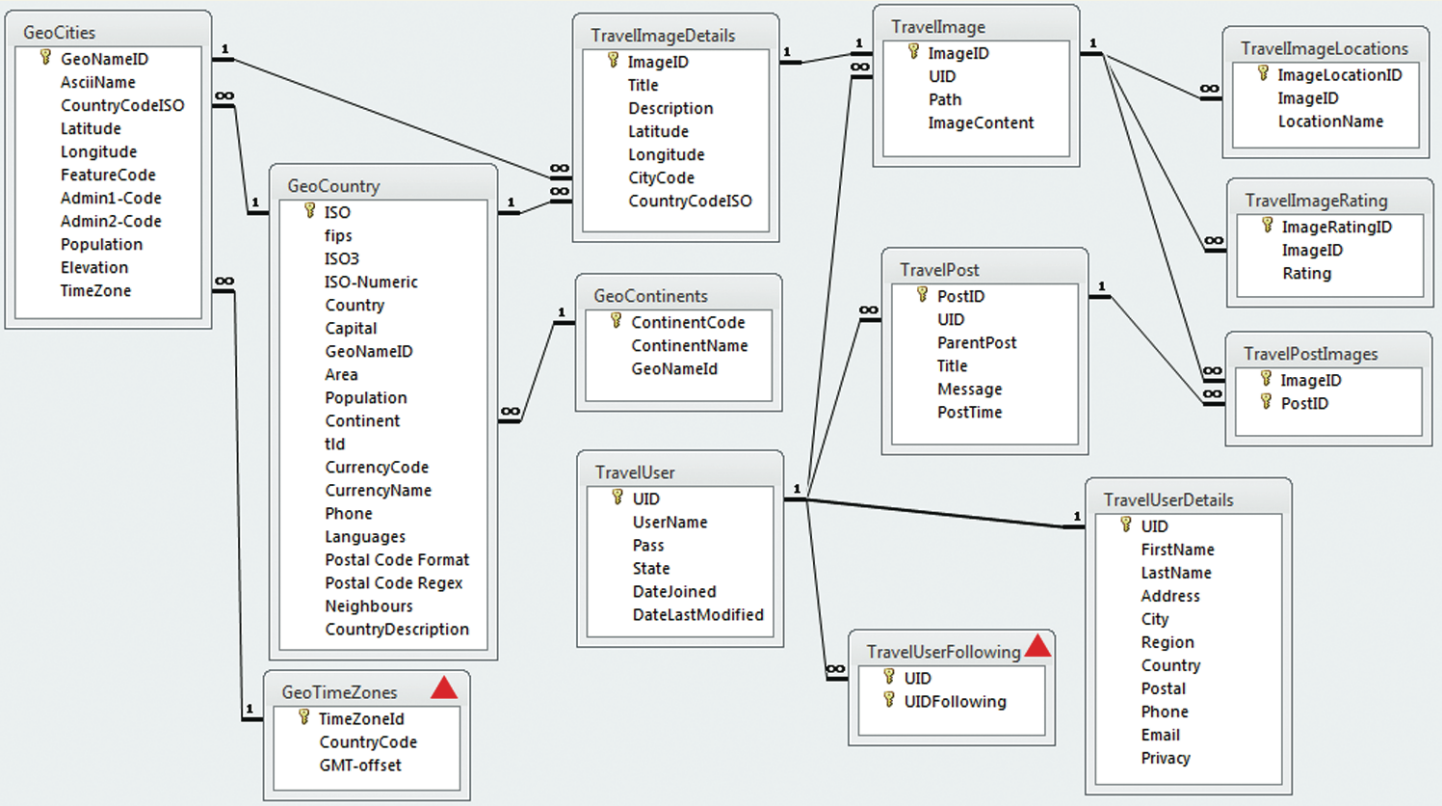


Book CRM Database

Customer Relationship Management



Travel Photo Sharing Database



SELF READING

SAMPLE DATABASE TECHNIQUES

Display a list of Links

One of the most common database tasks in PHP is to display a list of links (i.e., a series of <li> elements within a <ul>).

```
<ul>
    <li><a href="list.php?category=7">Business</a></li>
    <li><a href="list.php?category=2">Computer Science</a></li>
    <li><a href="list.php?category=3">Economics</a></li>
    <li><a href="list.php?category=9">Engineering</a></li>
    <li><a href="list.php?category=4">English</a></li>
    <li><a href="list.php?category=6">Mathematics</a></li>
    <li><a href="list.php?category=8">Statistics</a></li>
    <li><a href="list.php?category=5">Student Success</a></li>
</ul>
```

Display a list of Links

At its simplest, the code would look something like the following:

```
$sql = "SELECT * FROM Categories ORDER BY
            CategoryName”;
$result = $pdo->query($sql);
while ($row = $result->fetch()) {
            echo '<li>’;
            echo '<a href="list.php?category=' . $row['ID']
. '">’;
            echo $row['CategoryName'];
            echo '</a>’;
            echo '</li>’;
}
```

Display a list of Links

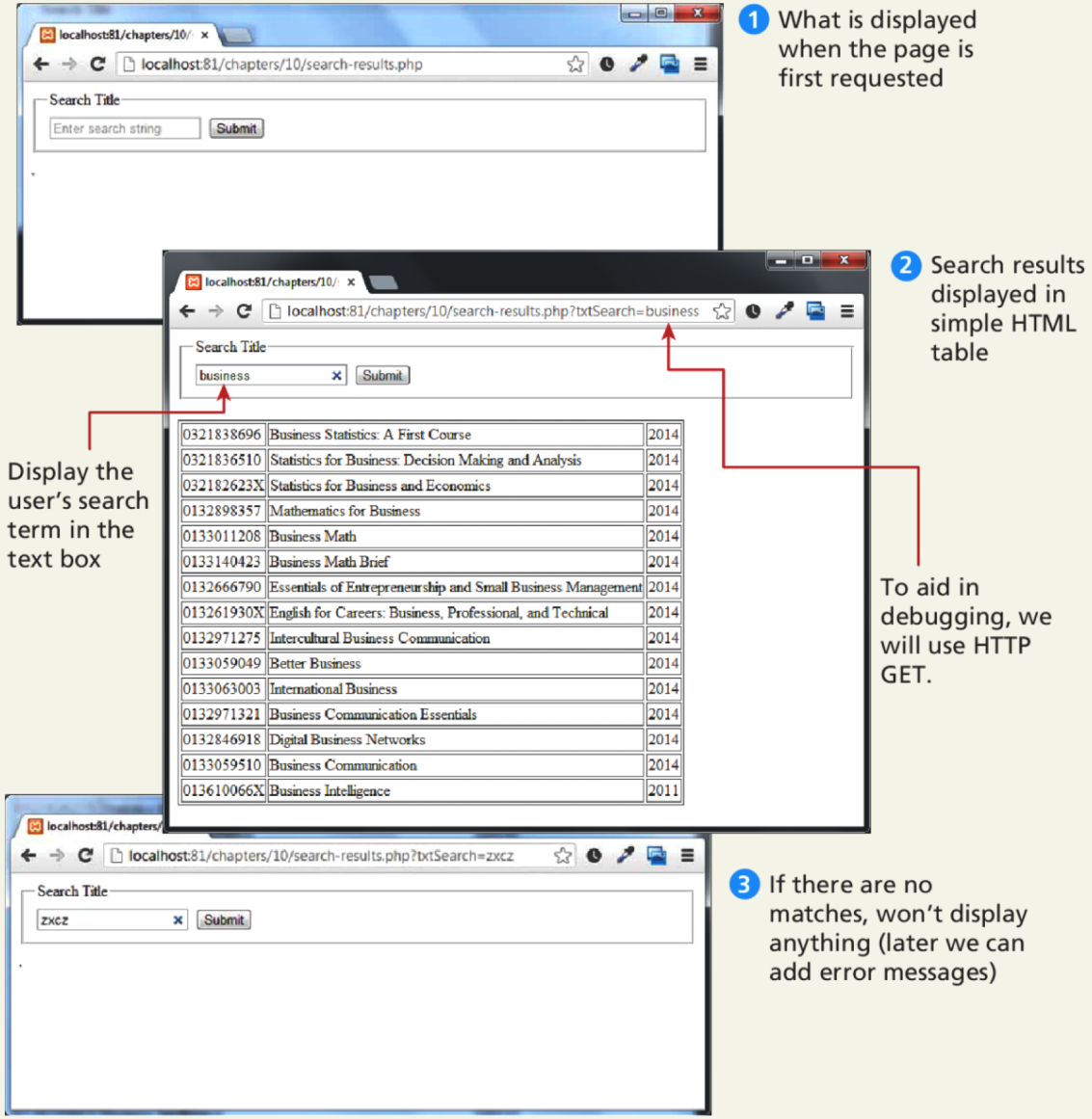
More maintainable version

```
<ul>
<?php
$result = getResults();  // some function that returns the result set
while ($row = $result->fetch()) {
?>
   <li>
   <a href="l       ist.php?category=<?php echo $row['ID']; ?>">
   <?php echo $row['CategoryName']; ?>
   </a>
          ist.php?category=<?php echo $row['ID']; ?>">
   </li>
<?php } ?>
</ul>
```

LISTING 11.28 Alternate list of links example

Search and Results Page

Visual of search box, results page, and no results page



Search and Results Page

In this example, we will assume that there is a text box with the name txtSearch in which the user enters a search string along with a Submit button.

The data that we will filter is the Book table; we will display any book records that contain the user-entered text in the Title field.

```
// add SQL wildcard characters to search term
$searchFor = '%' . $_GET['txtSearch'] . '%';
$sql = "SELECT * FROM Books WHERE Title Like ?";
$statement = $pdo->prepare($sql);
$statement->bindValue(1, $searchFor);
$statement->execute();
```

Search and Results Page

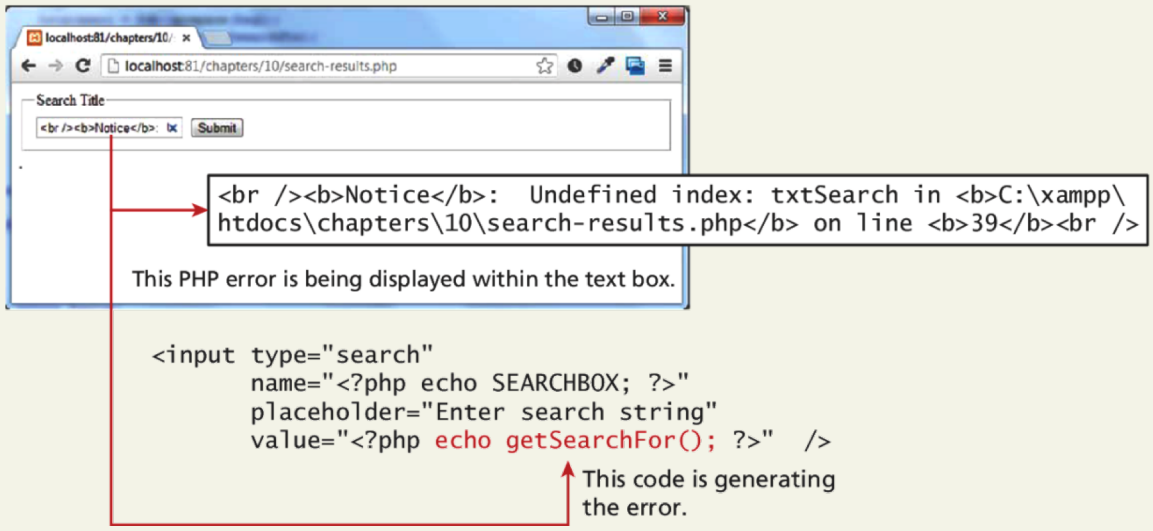
To redisplay the search term we will add code like:

```
<input
        type="search”
        name="txtSearch”
        placeholder="Enter search string”
        value="<?php echo $_GET['txtSearch']; ?>" />
```

To where we generate the form. Unfortunately…

Search and Results Page

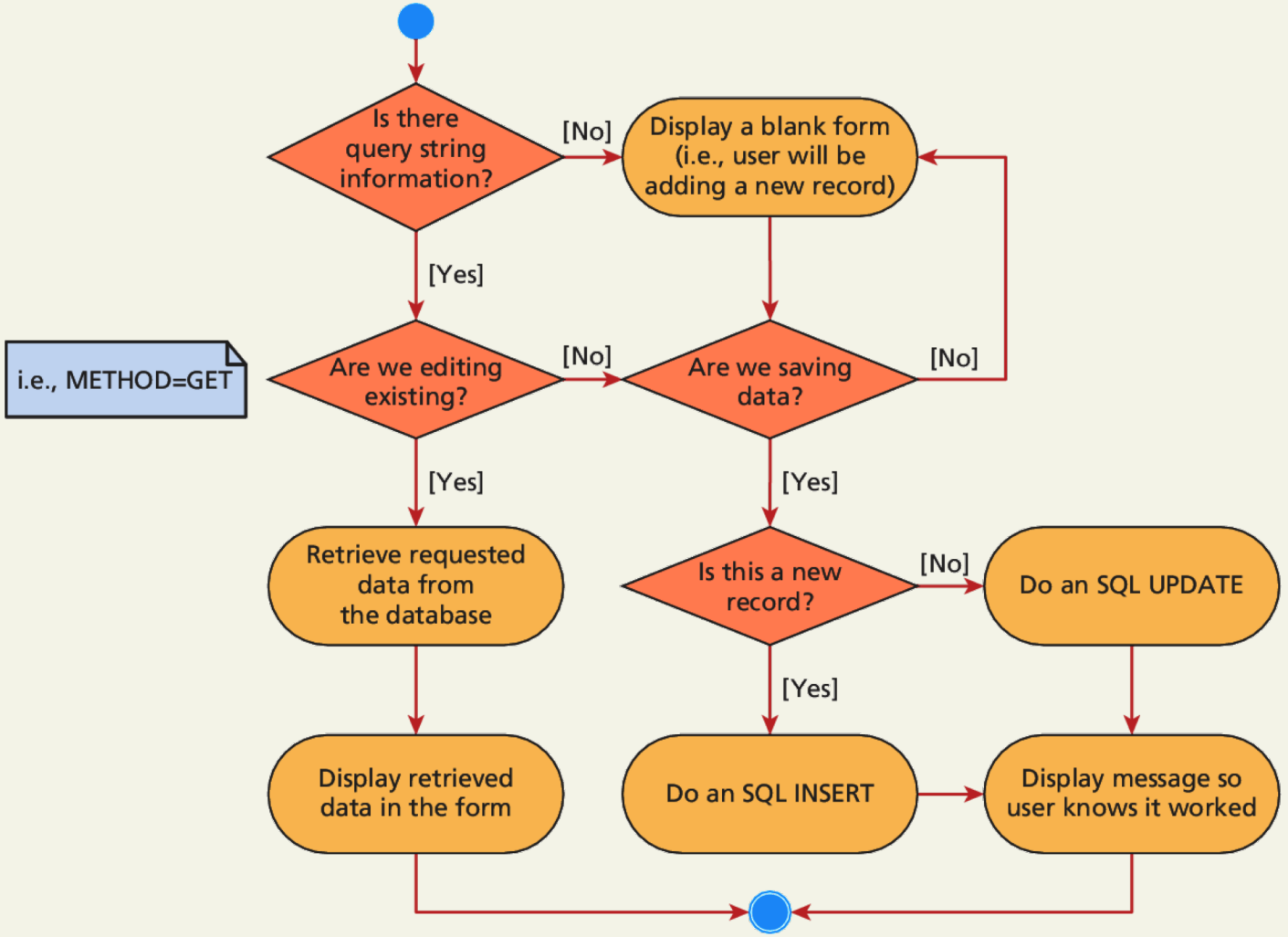
Problem to be solved




```
function getSearchFor()
{
    $value = "";
    if (isset($_GET[SEARCHBOX])) {
        $value = $_GET[SEARCHBOX];
    }
    return $value;
}
```

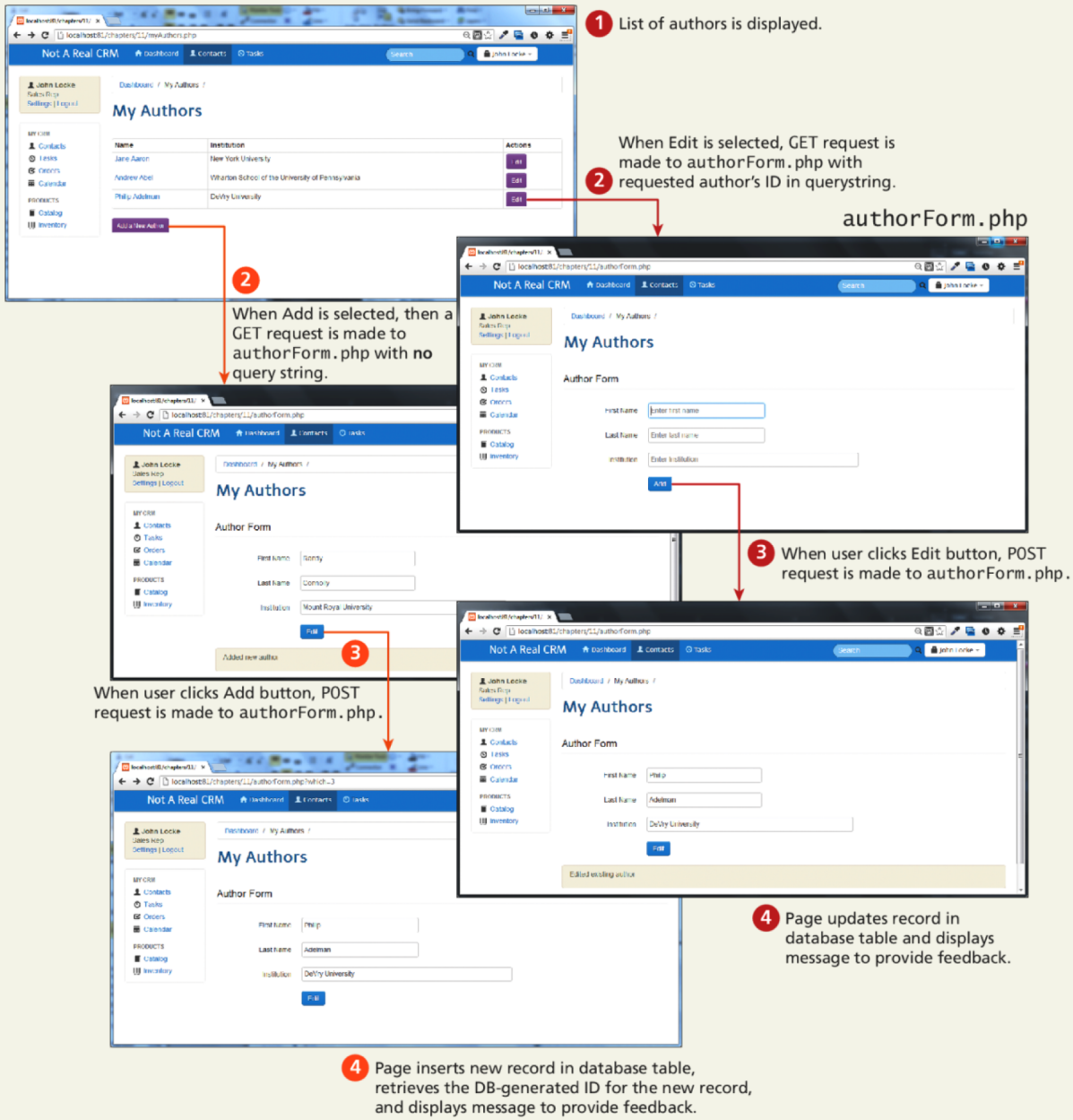
LISTING 11.30 Solution to search results page problem

Editing a Record



Editing a Record

myAuthors.php



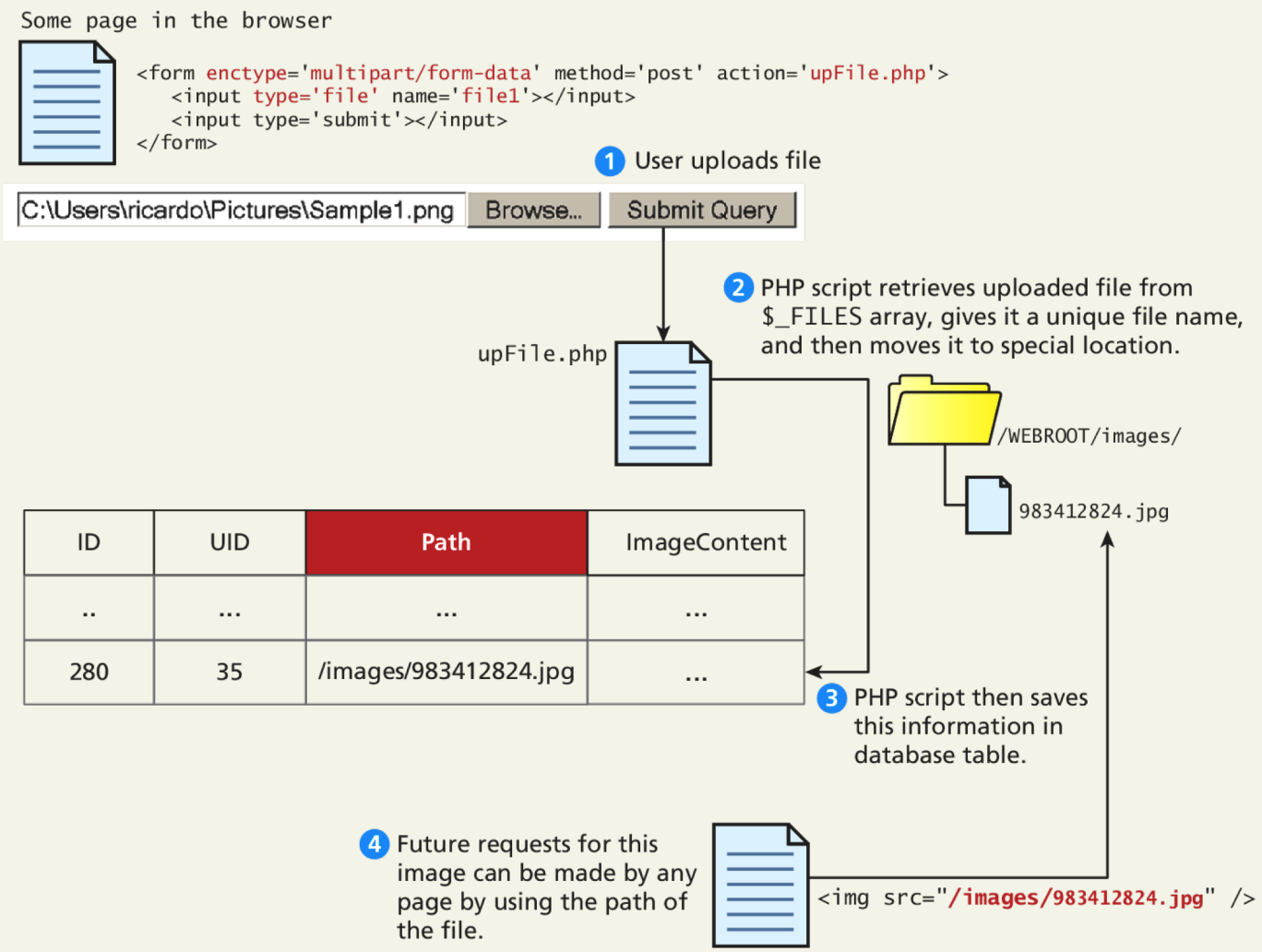
Files in the Database

There are two ways of storing files in a database:

1. Storing file location in the database, and storing the file on the server's filesystem
2. Storing the file itself in the database in the form of a binary BLOB

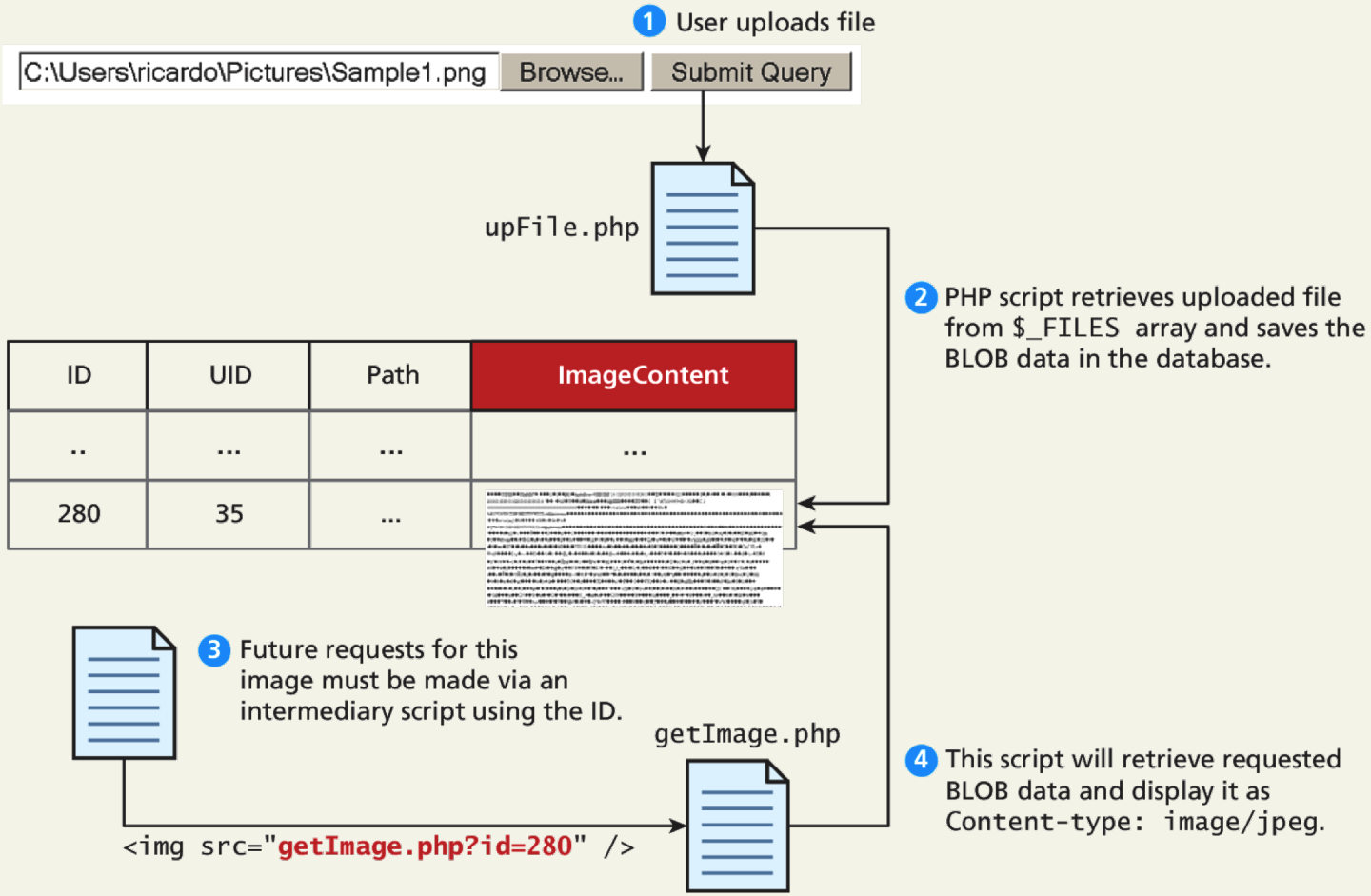
Files in the Database

As a file Location



Files in the Database

As a BLOB



Files in the Database

Storing and retrieving BLOBs

```
$fileContent = file_get_contents("someImage.jpg");
$sql = "INSERT INTO TravelImage (ImageContent) VALUES(':data')";

$statement = $pdo->prepare($sql);
$statement->bindParam(':data', $fileContent, PDO::PARAM_LOB);
$statement->execute();
```

LISTING 11.35 Code to save file contents in a BLOB field

```
// retrieve blob content from database
$sql = "SELECT * FROM TravelImage WHERE ImageID=:id";
$statement = $pdo->prepare($sql);
$statement->bindParam(':id', $_GET['id']);
$statement->execute();

$result = $statement->fetch(PDO::FETCH_ASSOC);
if ($result) {
   // Output the MIME header
   header("Content-type: image/jpeg");
   // Output the image
   echo ($result["ImageContent"]);
}
```

LISTING 11.36 Code to fetch and echo BLOB image

Files in the Database

Storing and retrieving BLOBs

Listing 11.36 can then be integrated into HTML image tags by pointing the src attribute to our script

Would now reference a dynamic PHP script like:

Files in the Database

HTTP headers matter

The same file output with correct and incorrect headers is interpreted differently by the browser