Working with Databases

Chapter 11

Objectives

- Database and Web Development
- Structured Query Language (SQL)
- Database APIs
- Managing a MySQL Database
- Accessing MySQL in PHP
- Case Study Schemas
- Sample Database Techniques

Databases and Web Development

Section 1 of 7

Databases and Web Development

This chapter covers the core principles of relational **Da tabase 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 D BMS.

MySQL

There are many other open source and proprietary rel
ational DBMS, including:

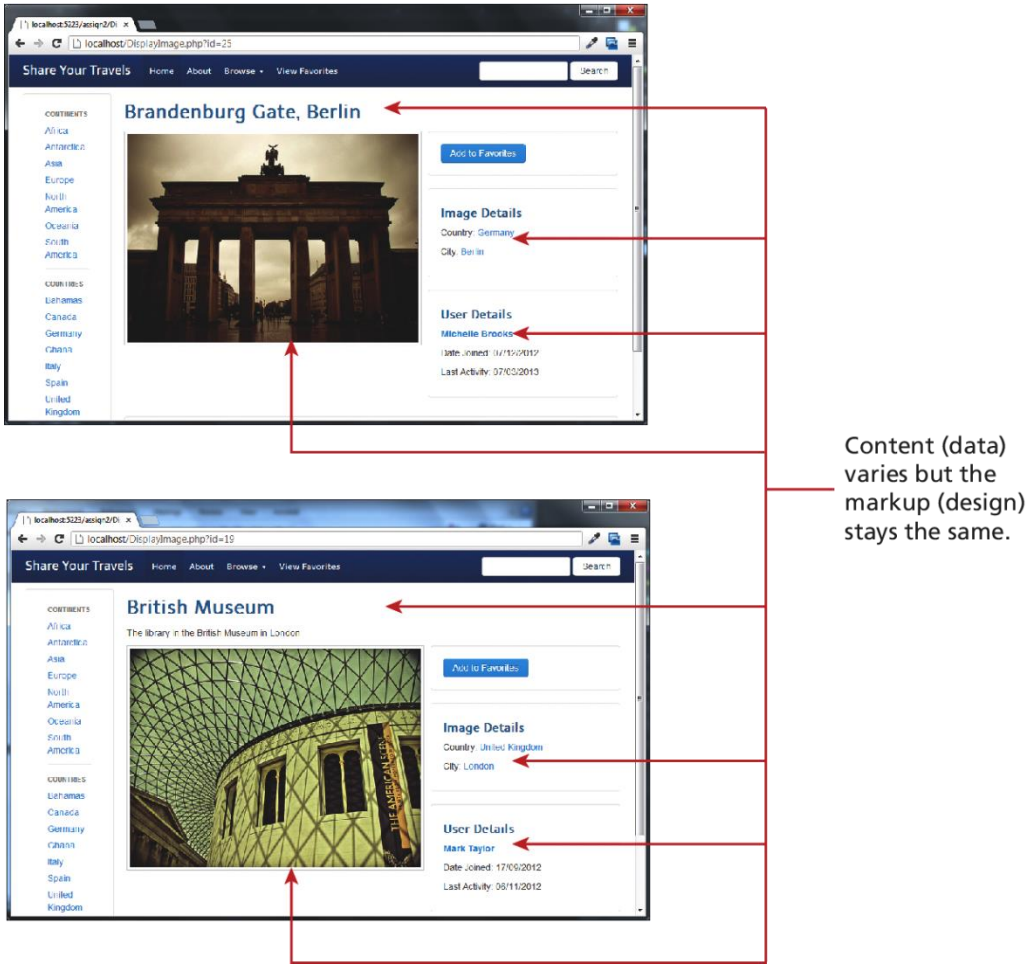
- PostgreSQL
- Oracle Database
- IBM DB2
- Microsoft SQL Server
- **MySQL**

The Role of Databases

In Web Development

Databases provide a way to i mplement one of the most im portant software design princ iples:

> *one should separate that wh ich varies from that which st ays the same*



Separate that which varies

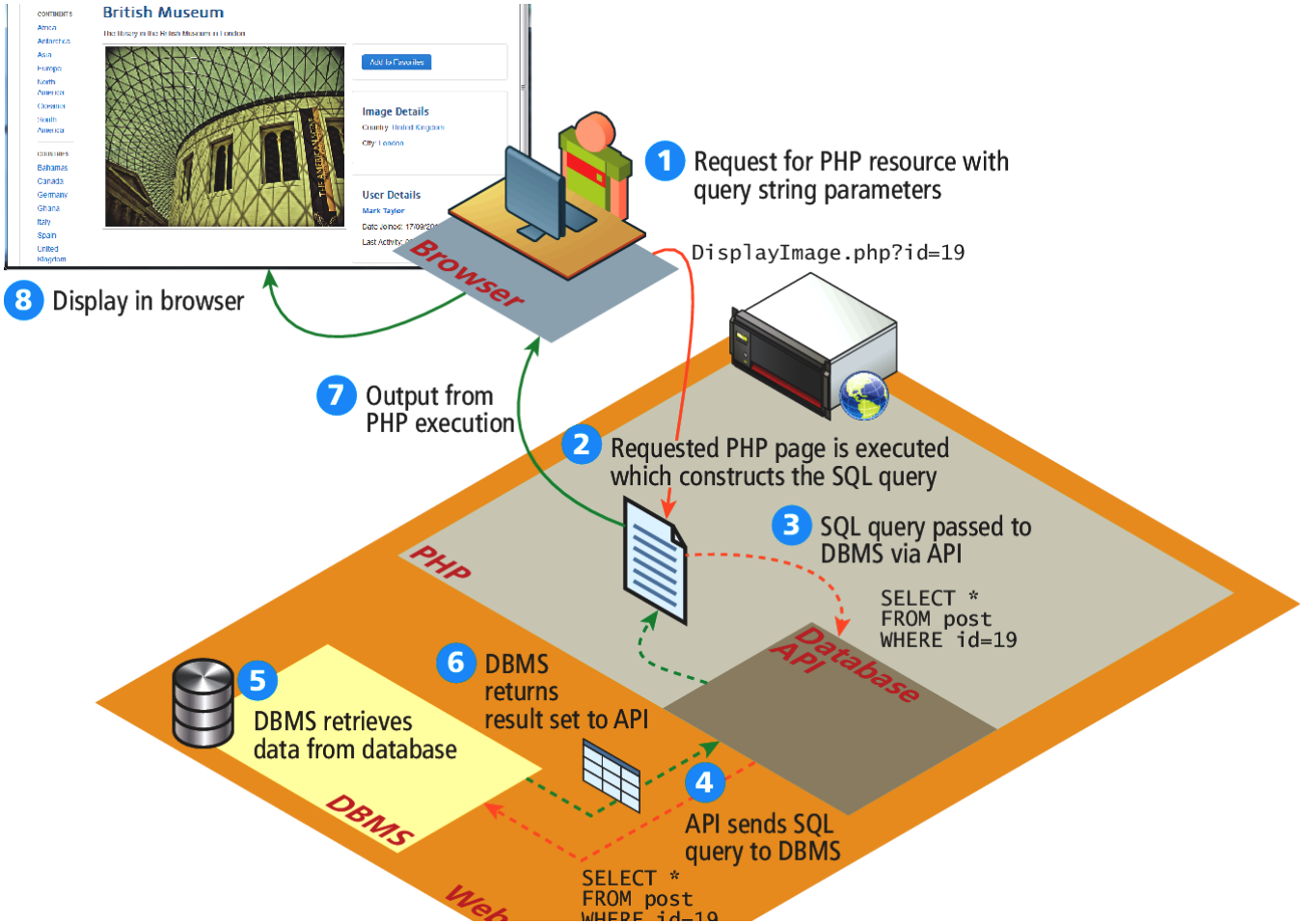
That is, use a DB to store the content of pages.

The program (PHP) determines which data to display, 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



Database Design

Tables

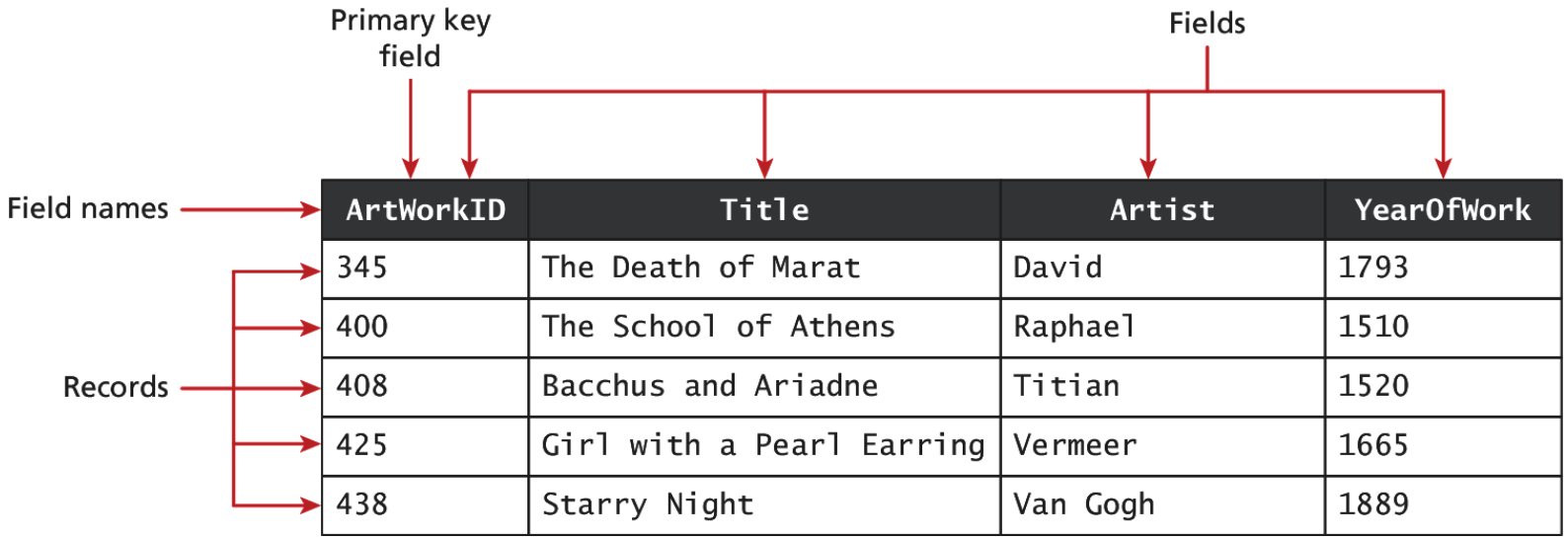
A database in a Relational DBMS is composed of one o r more **tables**.

A **table** is a two-dimensional container for data that co nsists of **records** (rows);

Each **record** has the same number of columns, which a re called **fields**, which contain the actual data.

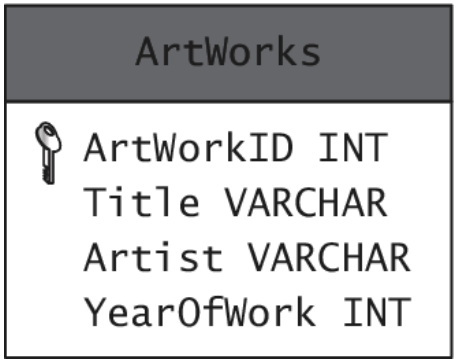
Each table will have one special field called a **primary k ey** that is used to uniquely identify each record in a tab le.

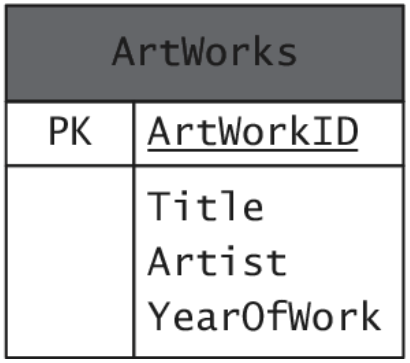
Tables in a Database

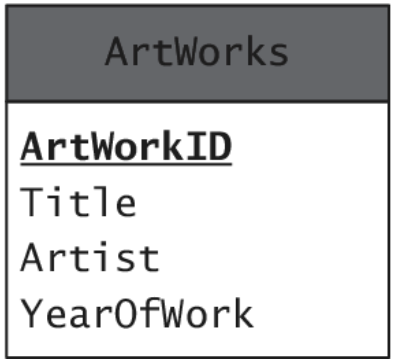


Tables in a Database

As we discuss database tables and their design, it will be helpful to have a more condensed way to visually re present a table







Benefits of DBMS

A database can enforce rules about what can be stored.

This provides **data integrity** and potentially can reduce t
he amount of **data duplication.**

This is partly achieved through the use of **data types** th
at are akin to those in a programming language.

Data types

Type	Description
BIT	Represents a single bit for Boolean values. Also called BOOLEAN or BOOL.
BLOB	Represents a binary large object (which could, for example, be used to store an image).
CHAR(n)	A fixed number of characters (n = the number of characters) that are padded with spaces to fill the field.
DATE	Represents a date. There is also a TIME and DATETIME data types.
FLOAT	Represents a decimal number. There are also DOUBLE and DECIM AL data types.
INT	Represents a whole number. There is also a SMALLINT data type.
VARCHAR(n)	A variable number of characters (n = the maximum number of charac ters) with no space padding.

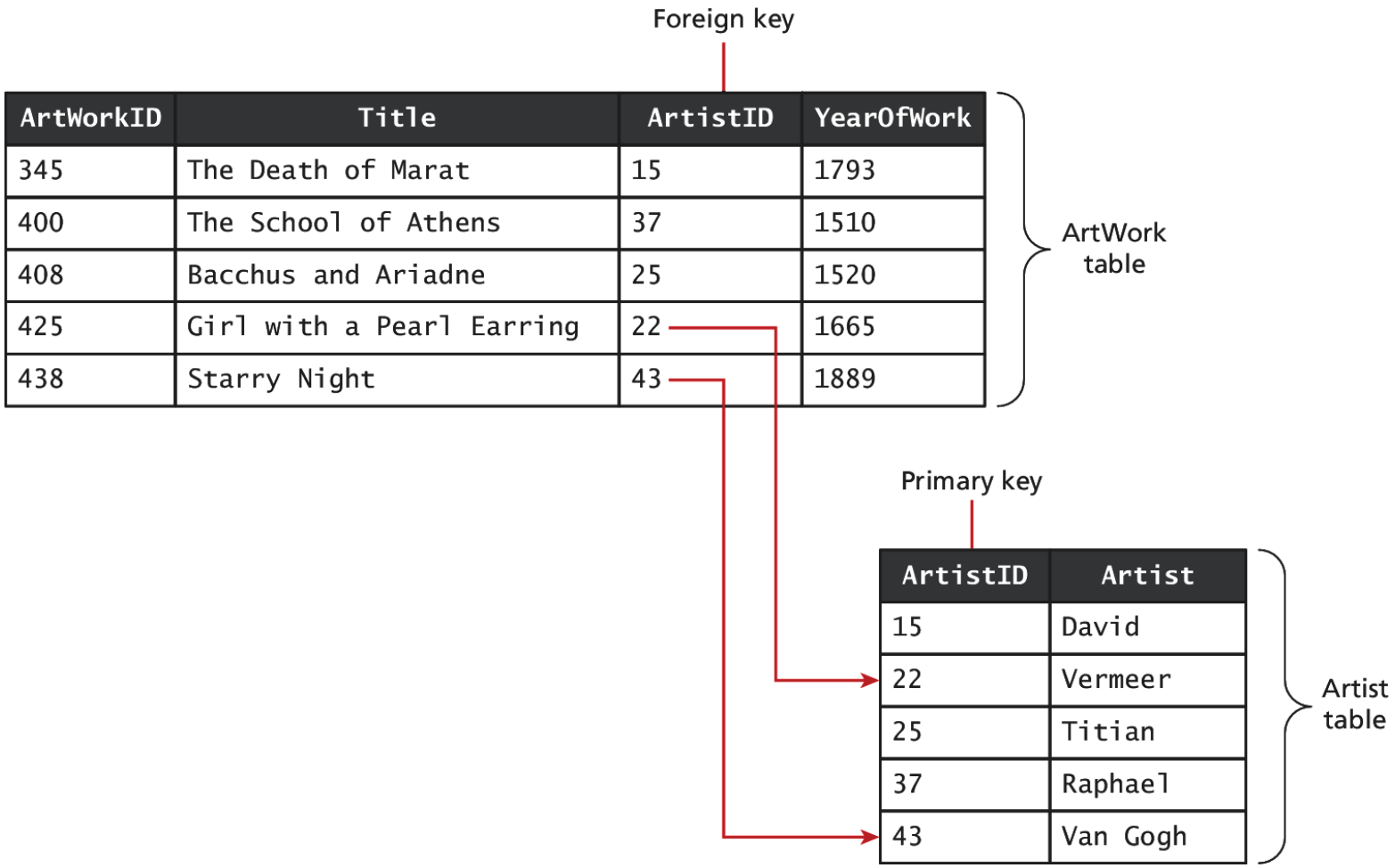
Benefits of DBMS

One of the most important ways that data integrity is a chieved in a database is by separating information abo ut different things into different tables.

Two tables can be related together via **foreign keys**, w hich is a field that is the same as the primary key of an other table

Relationships between Tables

Primary and Foreign Keys



Relationships between Tables

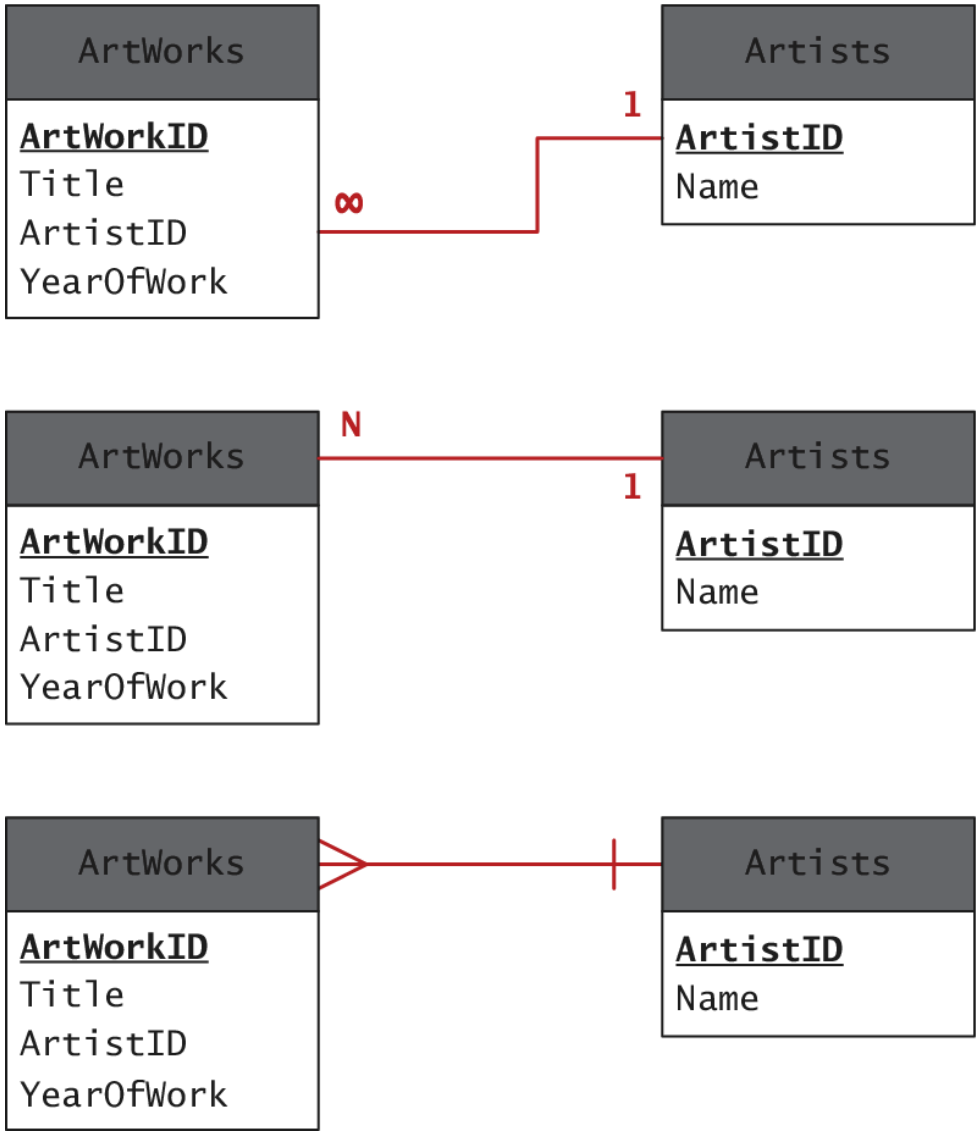
Tables that are linked via foreign keys are said to be in a relationship:

- **one-to-many relationship**
  single record in Table A can have one or more matching records in Table B

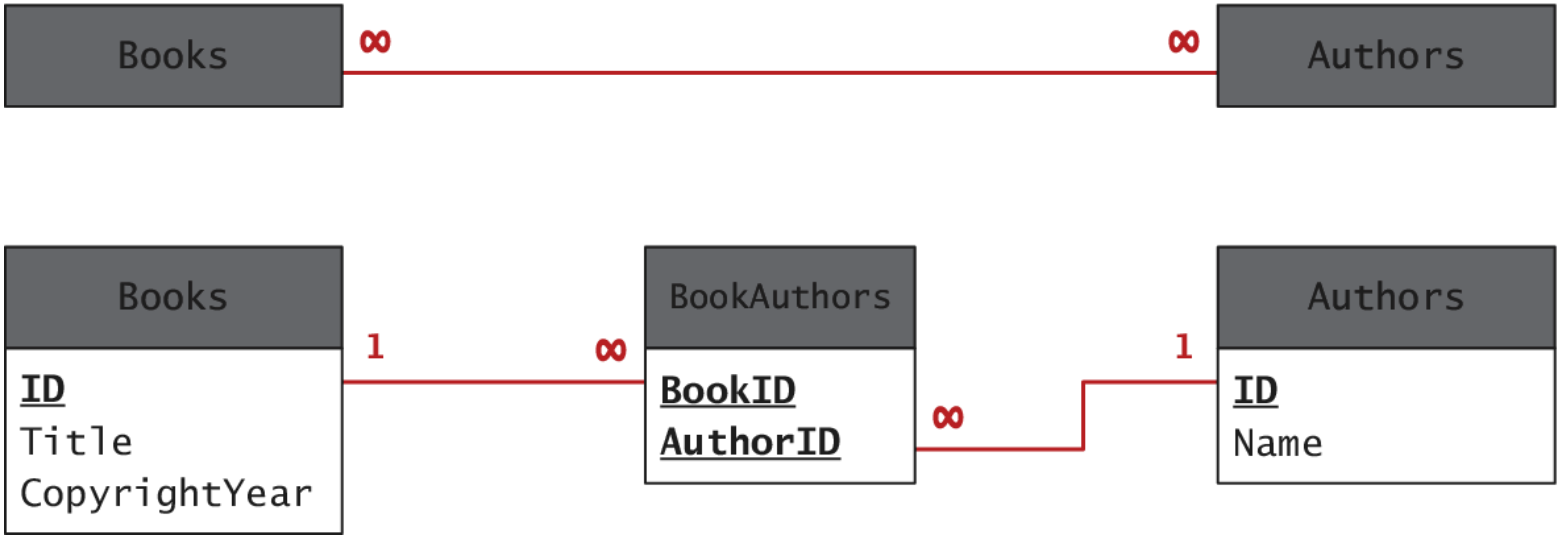
- **many-to-many relationship**
  Many-to-many relationships are usually implemented by using an intermediate table with two one-to-many relationships

- **One-to-one relationship**
  Typically used for security or performance reasons.
  (Could be 1 table)

One to Many


ArtWorks
ArtWorkID
Title
ArtistID
YearOfWork
∞
1
Artists
ArtistID
Name
ArtWorks
ArtWorkID
Title
ArtistID
YearOfWork
N
1
Artists
ArtistID
Name
ArtWorks
ArtWorkID
Title
ArtistID
YearOfWork
Artists
ArtistID
Name

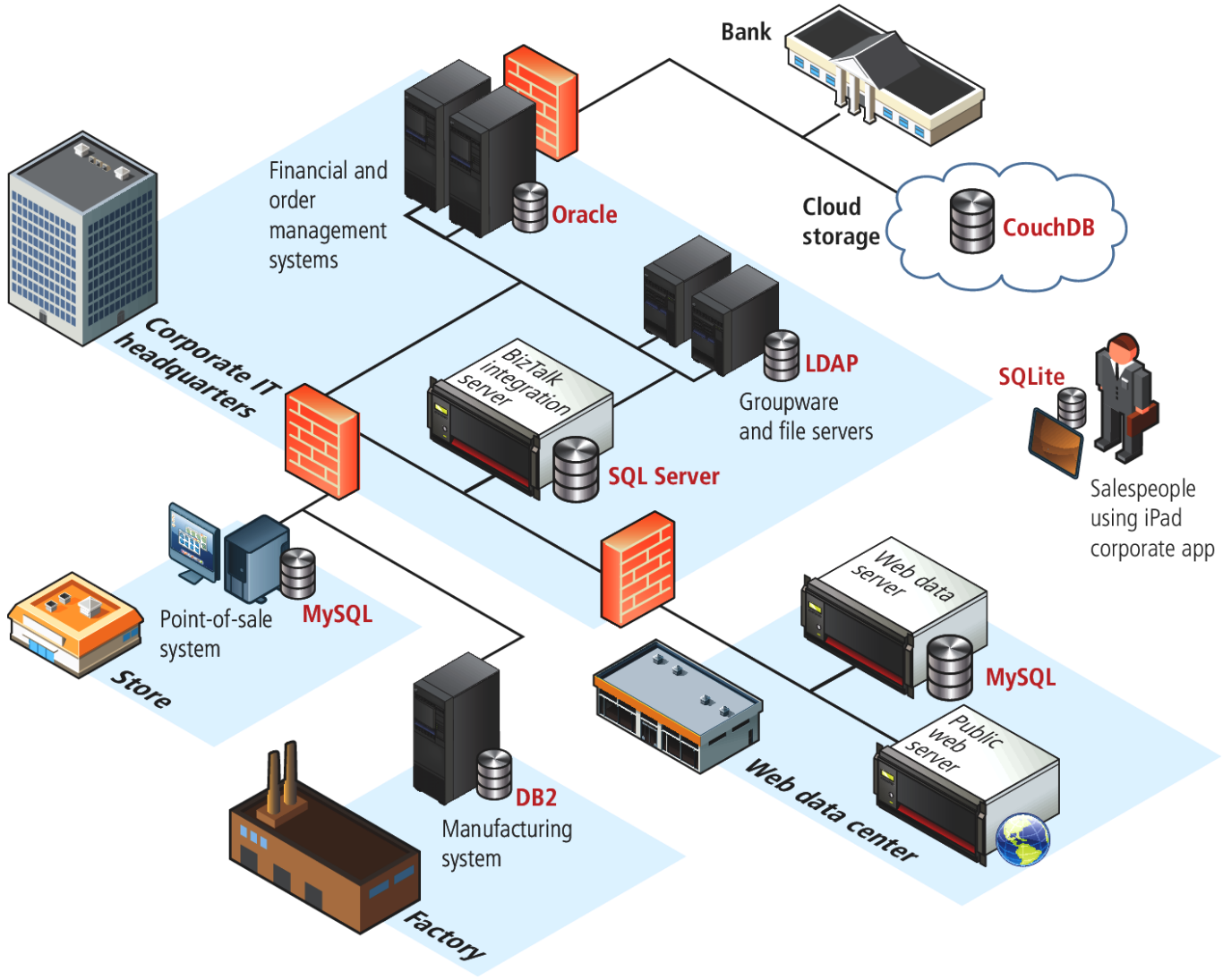
Many to Many



Note that in this example, the two foreign keys in the intermediate table are combined to create a **composite key**

Database Options

The range of DBMS Usage in an Enterprise



Database Options

Other ways of using databases

- **SQLite** a software library that typically is integrated directly within an application rather than running as a separate process like most DBMS.
- **No-SQL** databases do not make use of SQL, are not relational in how they store data, and are optimized to retrieve data using simple key-value syntax simila r to that used with associative arrays in PHP.
  - CouchDB , mongoDB, Hypertable, Cassandra, …

Structured Query Language (SQL)

Section 2 of 7

SQL

Pronounced *sequel (*or *sss queue elle)*

The Structured Query Language uses English Like Syntax to interact with the Database. Common SQL commands :

- SELECT
- INSERT
- UPDATE
- DELETE

SELECT

To retrieve data

The **SELECT statement** is used to retrieve data from the database.

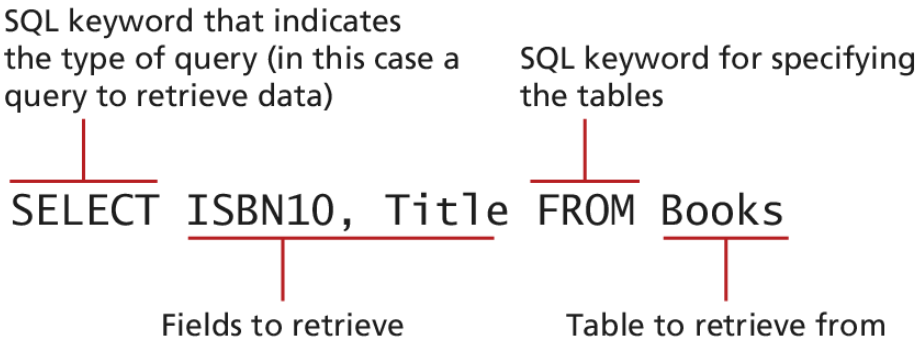
The result of a SELECT statement is a block of data typic ally called a **result set**.

You must specify

- what fields to retrieve and
- what Table to retrieve from

SELECT

To retrieve data




```
SELECT * FROM Books
```

Wildcard to select all fields

Note: While the wildcard is convenient, especially when testing, for production code it is usually avoided; instead of selecting every field, you should select just the fields you need.

SELECT

Ordering results

To specify a certain order to the result set use the **ORDER** clause.

You Specify

- the field to Sort on (or several), and
- whether to sort in ascending or descending order.

SELECT

ORDER BY clause

```
select iSbN10, title
FROM BOOKS
ORDER BY title
```

SQL keyword to indicate sort order

Field to sort on

Note: SQL doesn't care if a command is on a single line or multiple lines, nor does it care about the case of keywords or table and field names. Line breaks and keyword capitalization are often used to aid in readability.

```
SELECT ISBN10, Title FROM Books
ORDER BY CopyrightYear DESC, Title ASC
```

Keywords indicating that sorting should be in descending or ascending order (which is the default)

Several sort orders can be specified: in this case the data is sorted first on year, then on title.

SELECT

WHERE Clause

The Simple SELECT queries used so far retrieve *all* the r ecords in the specified table.

Often we are not interested in retrieving all the record s in a table but only a subset of the records.

This is accomplished via the **WHERE** clause, which can be added to any SELECT statement

SELECT

WHERE Clause

```
SELECT isbn10, title FROM books
WHERE copyrightYear > 2010
```

SQL keyword that indicates to return only those records whose data matches the criteria expression

Expressions take form: field *operator* value

```
SELECT isbn10, title FROM books
WHERE category = 'Math' AND copyrightYear = 2014
```

Comparisons with strings require string literals (single or double quote)

SELECT

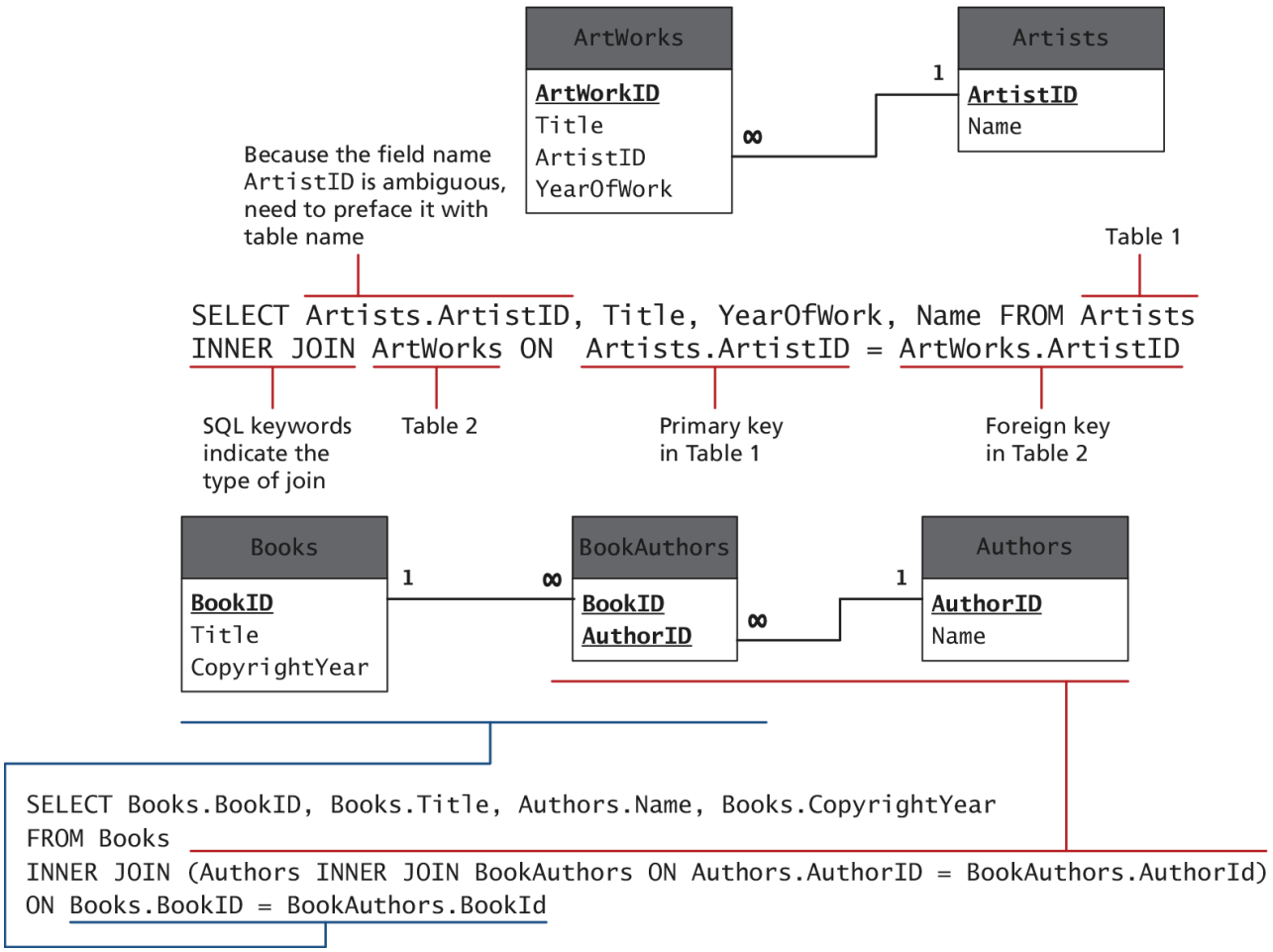
Advanced (but essential) techniques

Often we want to retrieve data from multiple tables. While we could do one query, and then do a second q uery using the data from the 1st query, that would be i nefficient. Instead we can use the **JOIN** clause (we will discuss **INNER JOIN**)

When you don't want every record in your table but in stead want to perform some type of calculation on mu ltiple records and then return the results. This require s using one or more **aggregate functions** such as SUM( ) or COUNT(); these are often used in conjunction wit h the **GROUP BY** keywords.

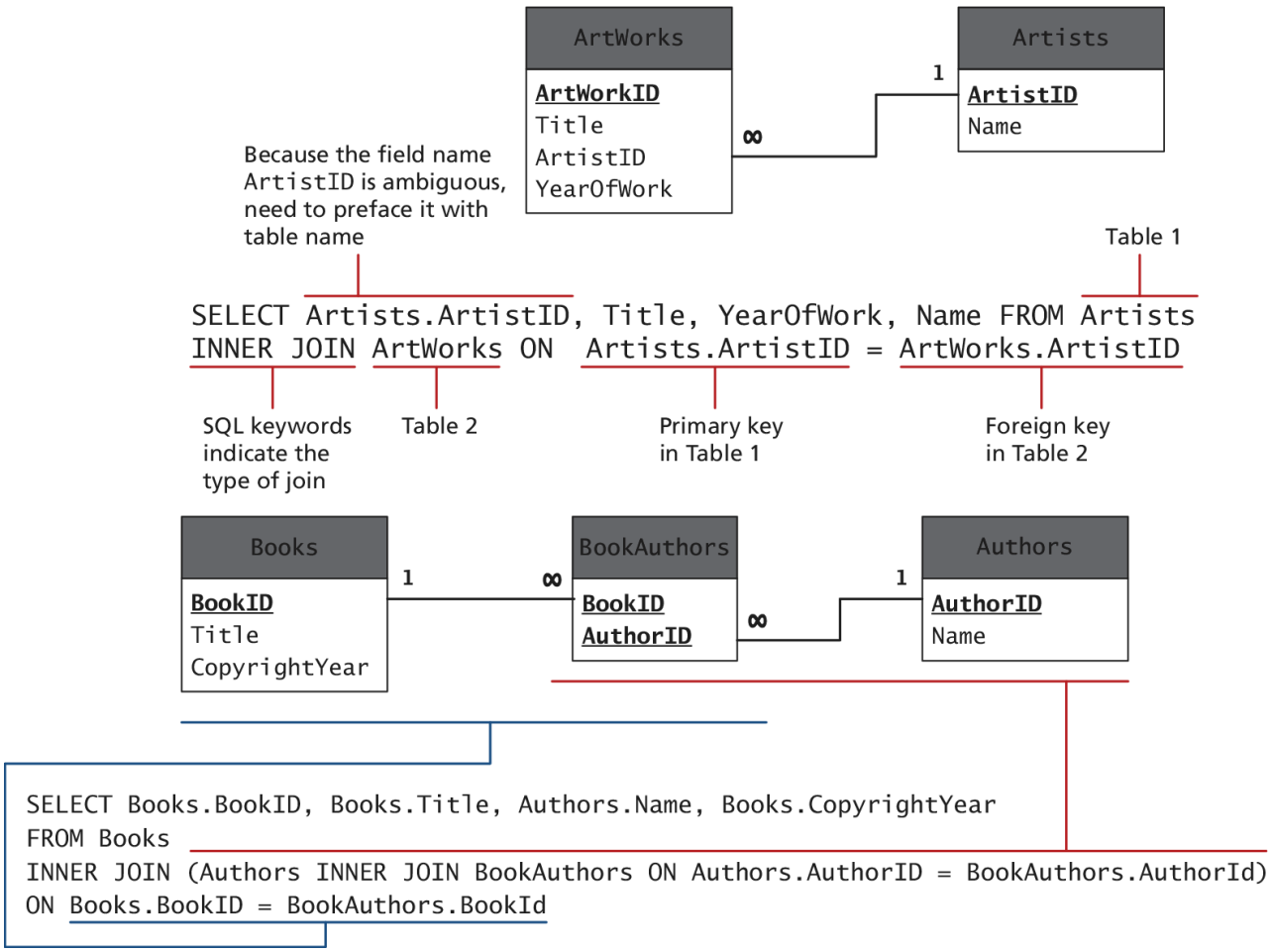
INNER JOIN

Still in a SELECT query



INNER JOIN

Still in a SELECT query



GROUP BY

Still in a SELECT query

This aggregate function returns a count of the number of records.

Defines an alias for the calculated value

```
SELECT Count(ArtWorkID) AS NumPaintings
FROM ArtWorks
WHERE YearOfWork > 1900
```

Count number of paintings after year 1900

Note: This SQL statement returns a single record with a single value in it.

NumPaintings
745

```
SELECT Nationality, Count(ArtistID) AS NumArtists
FROM Artists
GROUP BY Nationality
```

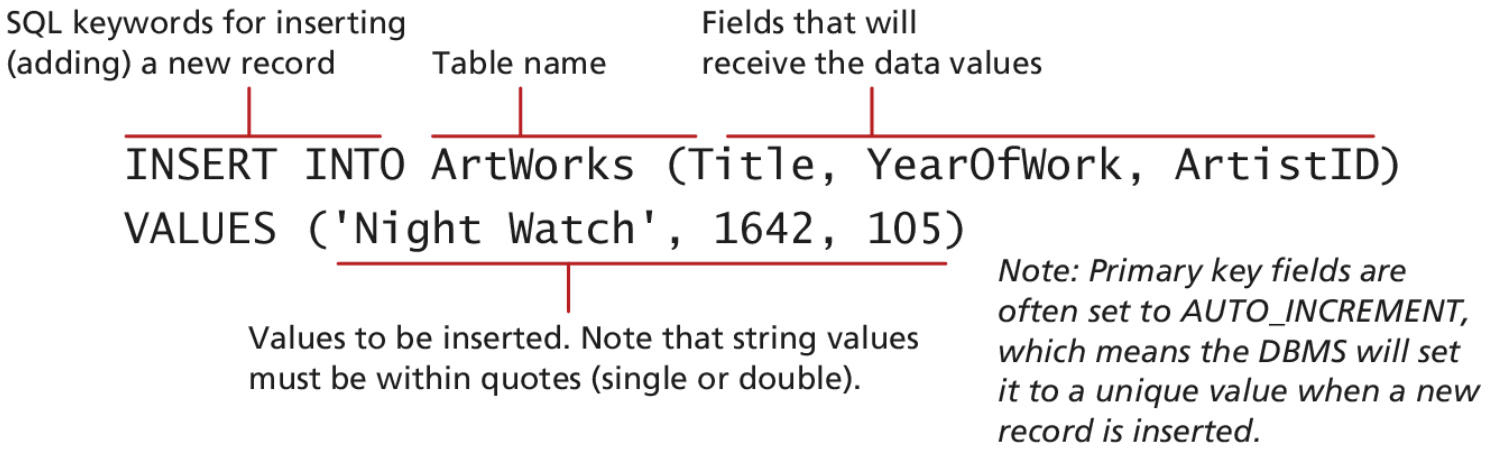
SQL keywords to group output by specified fields

Note: This SQL statement returns as many records as there are unique values in the group-by field.

Nationality	NumArtists
Belgium	4
England	15
France	36
Germany	27
Italy	53

INSERT

Adding data in to the table



Update

Modify data in the table

```
UPDATE ArtWorks
SET Title='Night Watch', YearOfWork=1642, ArtistID=105
WHERE ArtWorkID=54
```

It is essential to specify which record to update, otherwise it will update all the records!

Specify the values for each updated field.
Note: Primary key fields that are AUTO_INCREMENT cannot have their values updated.

Delete

Remove rows from the table

```
DELETE FROM ArtWorks
WHERE ArtWorkID=54
```

It is essential to specify which record to delete, otherwise it will delete all the records!

Transactions

An Advanced Topic.

Anytime one of your PHP pages makes changes to the database via an UPDATE, INSERT, or DELETE statement , you also need to be concerned with the possibility of failure.

A **transaction** refers to a sequence of steps that are tre ated as a single unit, and provide a way to gracefully h andle errors and keep your data properly consistent w hen errors do occur.

Transactions

An Example

Imagine how a purchase would work in a web storefront. After the user has verified the shipping address, entered a credit card, and selected a shipping option and clicks the final *Pay for Order* button? Imagine that the following steps need to happen.

1. Write order records to the website database.
2. Check credit card service to see if payment is accepted.
3. If payment is accepted, send message to legacy ordering system.
4. Remove purchased item from warehouse inventory table and add it to the order shipped table.
5. Send message to shipping provider.

Transactions

An Example

At any step in this process, errors could occur. For instance:

- The DBMS system could crash after writing the first order record but before the second order record could be written.
- The credit card service could be unresponsive, or the credit card payment declined.
- The legacy ordering system or inventory system or shipping provider system could be down.

Transactions

Multiple types

Local Transactions can be handled by the DBMS.

Distributed Transactions involve multiple hosts, several of which we may have no control over.

Distributed transactions are much more complicated than local transactions

Local Transactions

The easy transactions

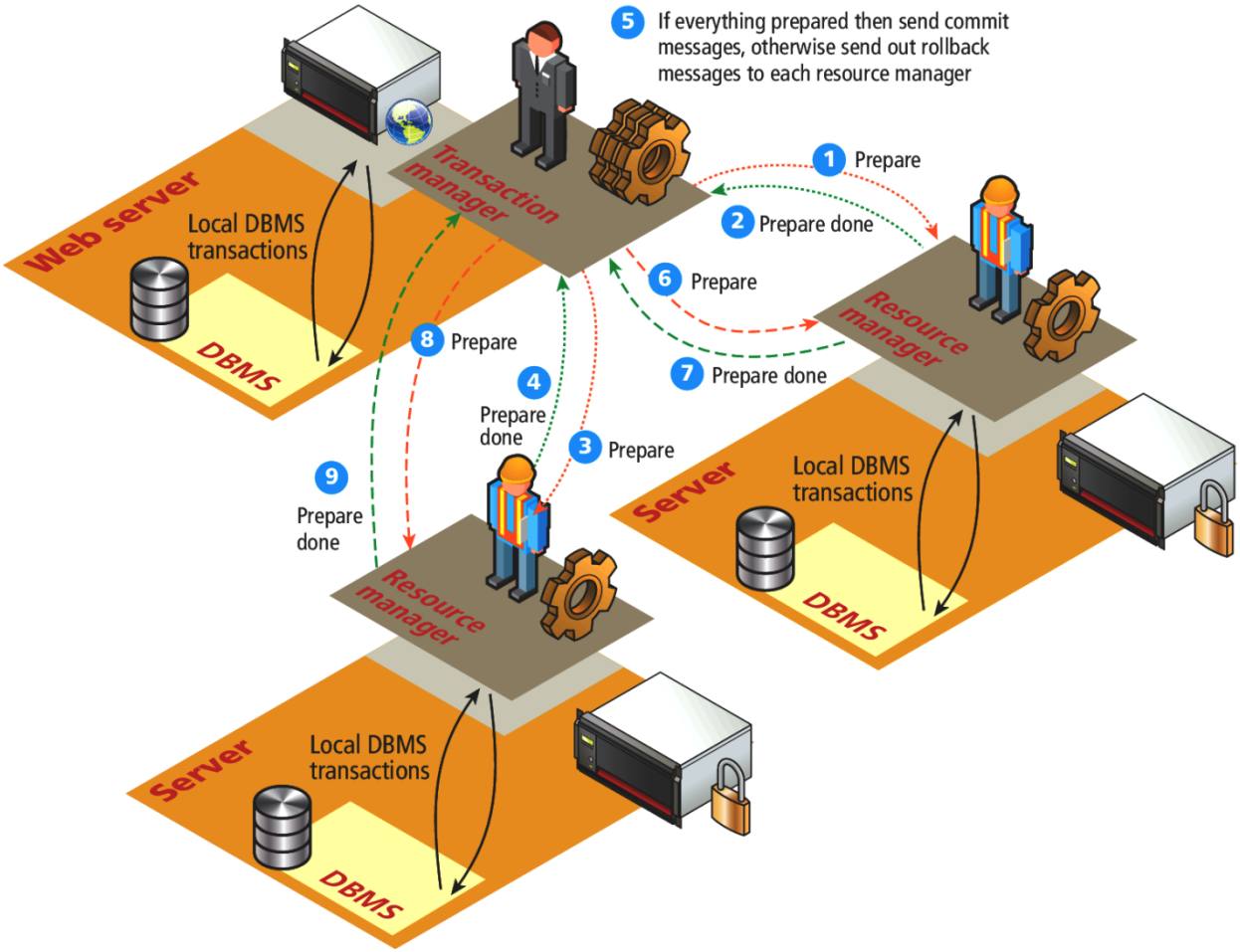
The SQL for transactions use the START TRANSACTION, COMMIT, and ROLLBACK commands

```
/* By starting the transaction, all database modifications within
   the transaction will only be permanently saved in the database
   if they all work   */

START TRANSACTION

INSERT INTO orders ...
INSERT INTO orderDetails ...
UPDATE inventory ...

/* if we have made it here everything has worked so commit changes */
COMMIT

/* if we replace COMMIT with ROLLBACK then the three database
   changes would be "undone" */
```

LISTING 11.1 SQL commands for transaction processing

Distributed Transactions

Depends on which external systems…



Data Definition Statements

Data Manipulation Language features of SQL are what we typically describe in web development and include the SELECT, UPDATE, INSERT, and DELETE.

There is also a **Data Definition Language (DDL)** in SQL, which is used for creating tables, modifying the structu re of a table, deleting tables, and creating and deleting databases. You may find yourself using them indirectly within something like the phpMyAdmin management t ool, although a text syntax does exist for those interest ed.

Database Indexes

Efficiency

In large sets of data, searching for a particular record c an take a long time.

Consider the worst-case scenario for searching where we compare our query against every single record. If t here are *n* elements we say it takes O(n) time to do a s earch (we would say “Order of *n*”).

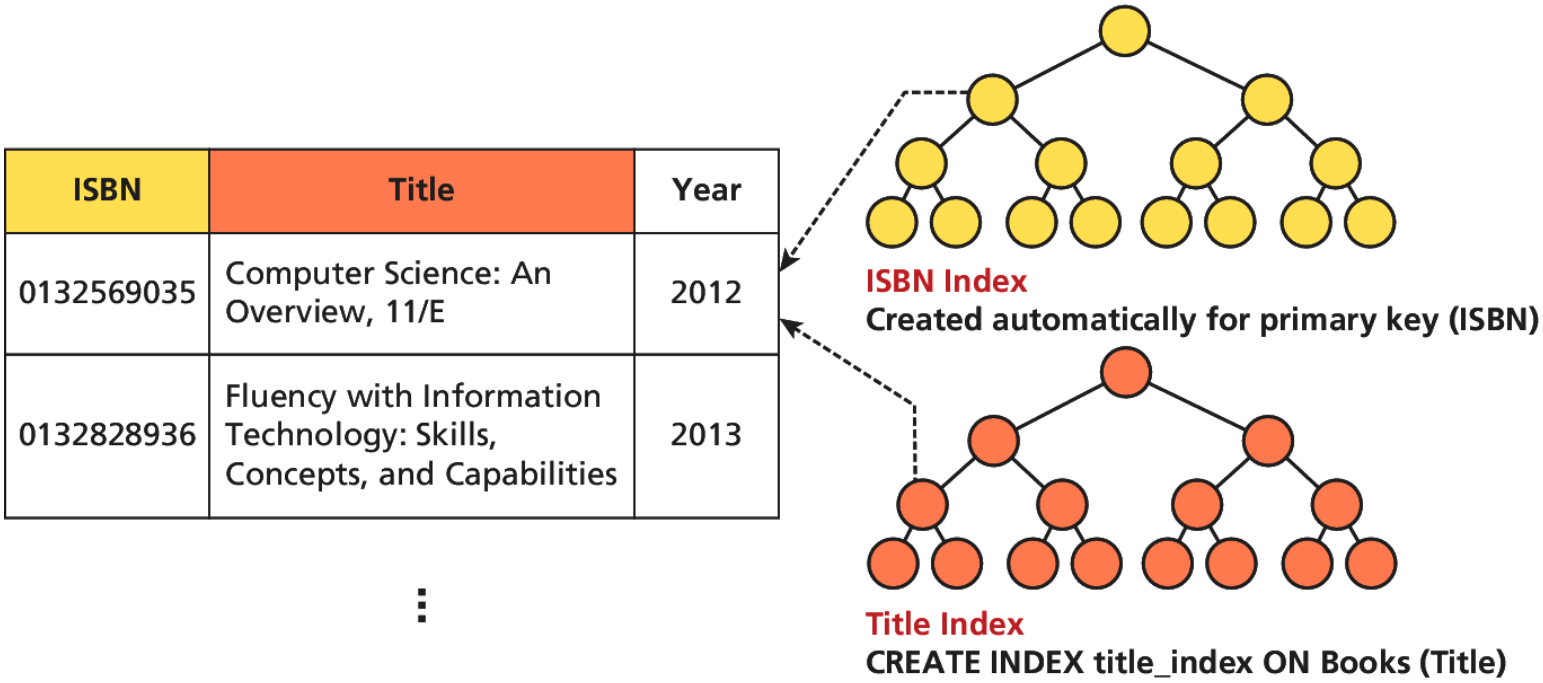
In comparison, a **balanced binary tree** data structure c an be searched in O(log2 n) time.

- N= 1,000,000
- log2 (1000000) = 20 ← much faster

Database Indexes

Use balanced trees or other efficient structures

No matter which data structure is used, the applicatio n of that structure to ensure results are quickly accessi ble is called an **index**. Every node in the index has just that field, with a pointer to the full record (on disk)



Database Indexes

Syntax exists if you're interested

Most database management tools allow for easy creati on of indexes through the GUI without use of SQL com mands.

The Index Creation Syntax is not normally used by web developers directly, and is not covered.

Database APIs

Section 3 of 7

API

Application Programming Interface

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

Some database APIs work only with a specific type of d atabase; 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 fo r working with MySQL and has been replaced with the new er mysqli extension. This procedural API should now only be used with versions of MySQL older than 4.1.3. (At the time of writing, the current version of MySQL was 5.7.3.)

- **mysqli extension**. The MySQL Improved extension takes ad vantage of features of versions of MySQL after 4.1.3. This ex tension provides **both** a procedural and an object-oriented approach. This extension also supports most of the latest fe atures 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 s ome set of functionality) that with the appropriate drivers c an be used with *any* database, and not just MySQL databas es. However, it is not able to make use of all the latest featu res of MySQL.

We will show how to do some of the most common da
tabase operations using the procedural mysqli extensi
on as well as the object-oriented PDO.

As the chapter (and book) proceed, we will standardiz
e on the object-oriented, database-independent PDO
approach.

Managing a MySQL Database

Section 4 of 7

How do you access the DBMS

So, so many ways

Although you will eventually be able to manipulate the database from your PHP code, there are some routine maintenance operations that do not warrant custom c ode. Tools include:

- Command Line Interface
- phpMyAdmin
- MySQLWorkbench

Command Line Interface

Oldie but a goodie.

The MySQL command-line interface is the most difficul t to master. The value of this particular management t ool is its low bandwidth and near ubiquitous presence on Linux machines.

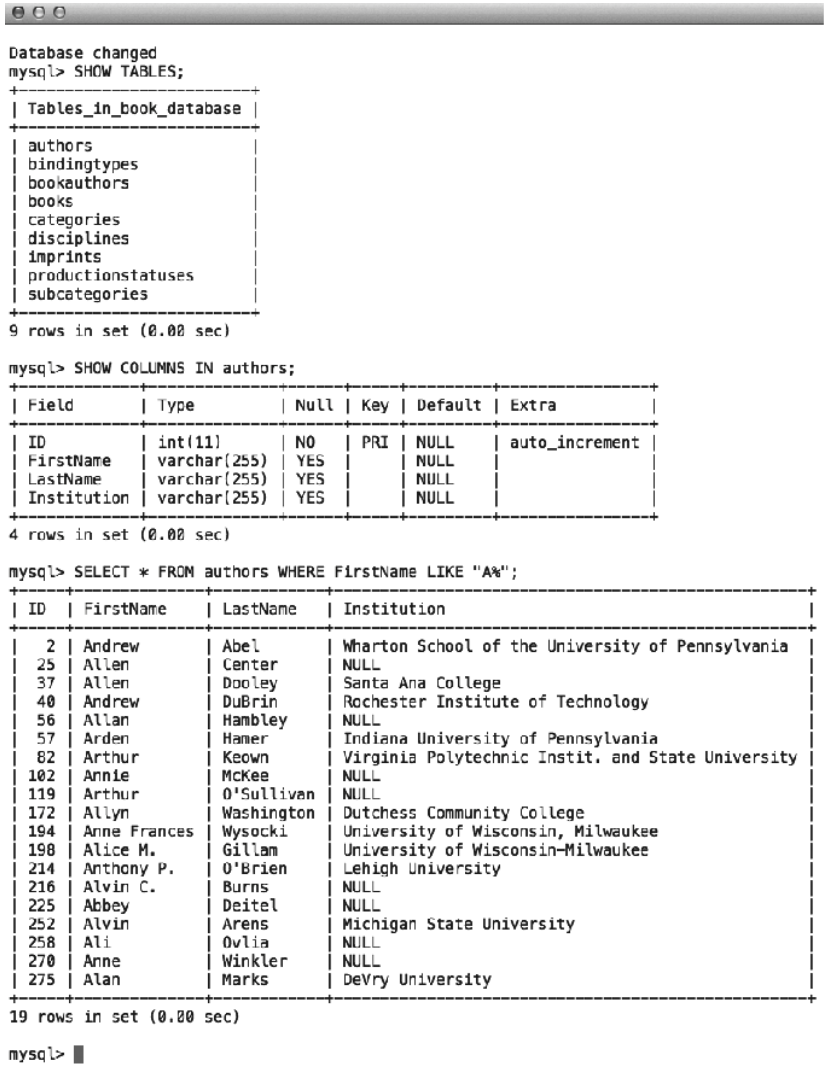
To launch an interactive MySQL command-line session, you must specify the host, username, and database na me to connect to as shown below:

mysql -h 192.168.1.14 -u bookUser –p

Once inside of a session, you may enter any SQL query, terminated with asemicolon (;)

Command Line Interface

Oldie but a goodie.



```
Database changed
mysql> SHOW TABLES;
+-------------------------+
| Tables_in_book_database |
+-------------------------+
| authors                 |
| bindingtypes            |
| bookauthors             |
| books                   |
| categories              |
| disciplines             |
| imprints                |
| productionstatuses      |
| subcategories           |
+-------------------------+
9 rows in set (0.00 sec)

mysql> SHOW COLUMNS IN authors;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| ID          | int(11)      | NO   | PRI | NULL    | auto_increment |
| FirstName   | varchar(255) | YES  |     | NULL    |                |
| LastName    | varchar(255) | YES  |     | NULL    |                |
| Institution | varchar(255) | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM authors WHERE FirstName LIKE "A%";
+-----+-------------+------------+-----------------------------------------------+
| ID  | FirstName   | LastName   | Institution                                   |
+-----+-------------+------------+-----------------------------------------------+
|   2 | Andrew      | Abel       | Wharton School of the University of Pennsylvania |
|  25 | Allen       | Center     | NULL                                          |
|  37 | Allen       | Dooley     | Santa Ana College                             |
|  40 | Andrew      | DuBrin     | Rochester Institute of Technology             |
|  56 | Allan       | Hambley    | NULL                                          |
|  57 | Arden       | Hamer      | Indiana University of Pennsylvania            |
|  82 | Arthur      | Keown      | Virginia Polytechnic Instit. and State University |
| 102 | Annie       | McKee      | NULL                                          |
| 119 | Arthur      | O'Sullivan | NULL                                          |
| 172 | Allyn       | Washington | Dutchess Community College                    |
| 194 | Anne Frances | Wysocki   | University of Wisconsin, Milwaukee            |
| 198 | Alice M.    | Gillam     | University of Wisconsin-Milwaukee             |
| 214 | Anthony P.  | O'Brien    | Lehigh University                             |
| 216 | Alvin C.    | Burns      | NULL                                          |
| 225 | Abbey       | Deitel     | NULL                                          |
| 252 | Alvin       | Arens      | Michigan State University                     |
| 258 | Ali         | Ovlia      | NULL                                          |
| 270 | Anne        | Winkler    | NULL                                          |
| 275 | Alan        | Marks      | DeVry University                              |
+-----+-------------+------------+-----------------------------------------------+
19 rows in set (0.00 sec)

mysql>
```

Command Line Interface

I bet you'll use it oneday…

In addition to the interactive prompt, the command line ca n be used to import and export entire databases or run a b atch of SQL commands from a file.

To import commands from a file called *commands.sql*, for e xample, we would use the **<** operation:

mysql –h 192.168.1.14 –u bookUser –p < commands.sql

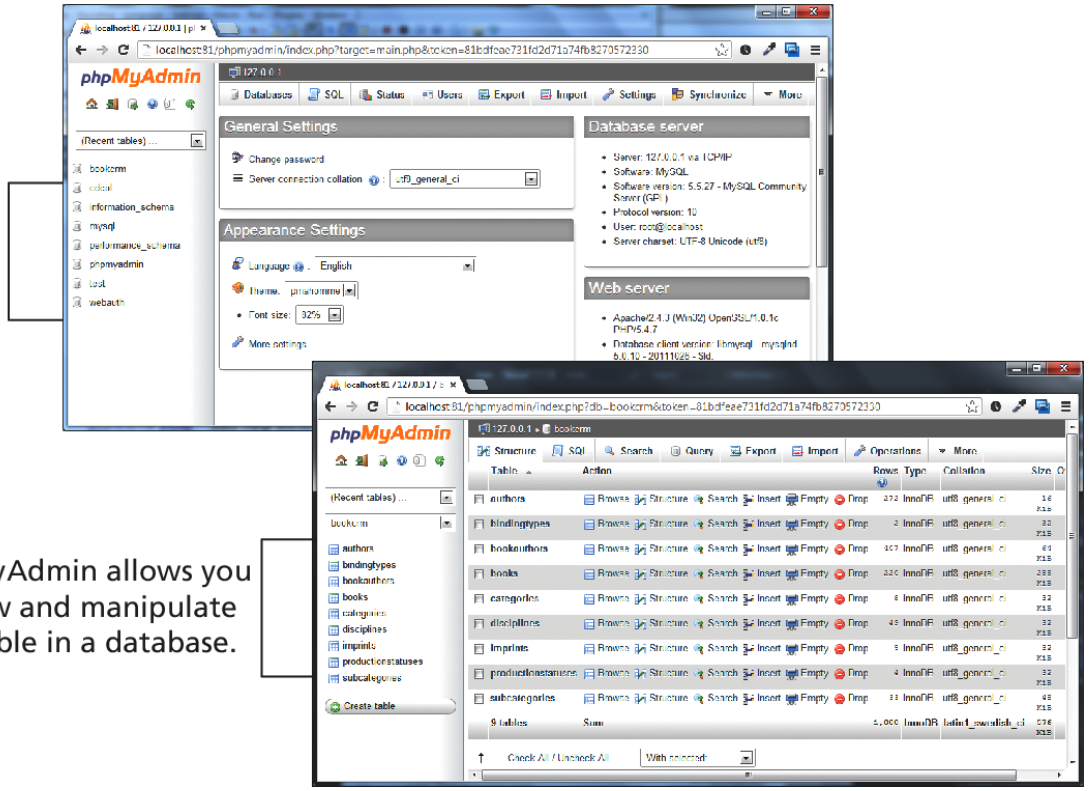
Although GUI tools allow this as well, the CLI can be integra ted into automated scripts to aid with mirroring, backup, a nd recovery.

phpMyAdmin

Will even generate PHP code

A popular web-based front-end (written in PHP) called **php MyAdmin** allows developers to access management tools t hrough a web portal.

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

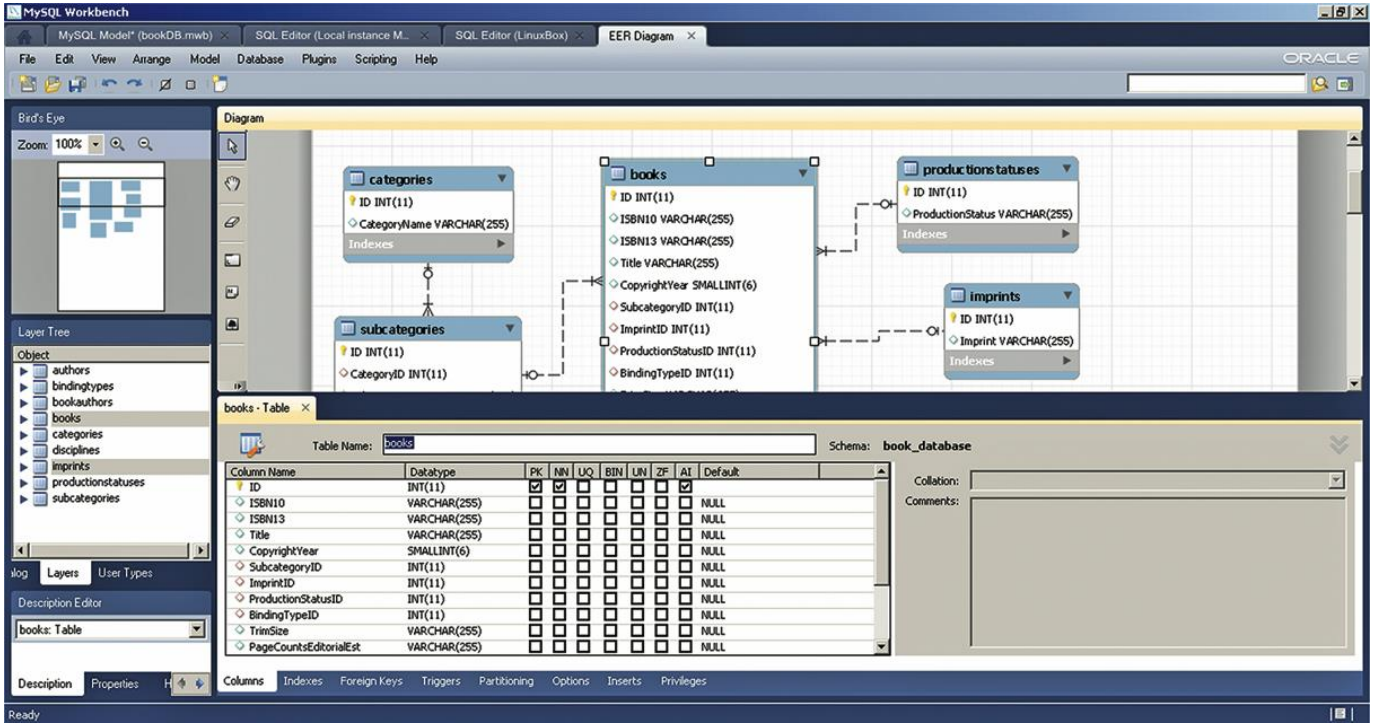


phpMyAdmin allows you to view and manipulate any table in a database.

MySQL Workbench

Powerful design tools

The MySQL Workbench is a free tool from Oracle to work w ith MySQL databases.



Accessing MySQL in PHP

Section 5 of 7

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
<?php

try {
   $connString = "mysql:host=localhost;dbname=bookcrm";
   $user = "testuser";
   $pass = "mypassword";

   $pdo = new PDO($connString,$user,$pass);
   $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

   $sql = "select * from Categories order by CategoryName";
   $result = $pdo->query($sql);

   while ($row = $result->fetch()) {
      echo $row['ID'] . " - " . $row['CategoryName'] . "<br/>";
   }
   $pdo = null;
}
catch (PDOException $e) {
   die( $e->getMessage() );
}

?>
```

1 2 3 4 5

Database Connection Algorithm

An illustration through example

```
// 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)

```
// 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 c
ode is not ideal.

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

We should move these connection details out of our c
onnection 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 c
ode.

- Procedural **mysqli** techniques use conditional (if...el
  se) statements on the returned object from the con
  nection attempt.

- The **PDO** technique uses try-catch which relies on t
  hrown exceptions when an error occurrs.

Handling Connection Errors

Procedural Approach

```
$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)

```
$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

In addition PDO has 3 different error modes, that allo w you to control when errors are thrown.

Execute the Query

Procedural and Object-Oriented

```
$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)

```
$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>";
}
```

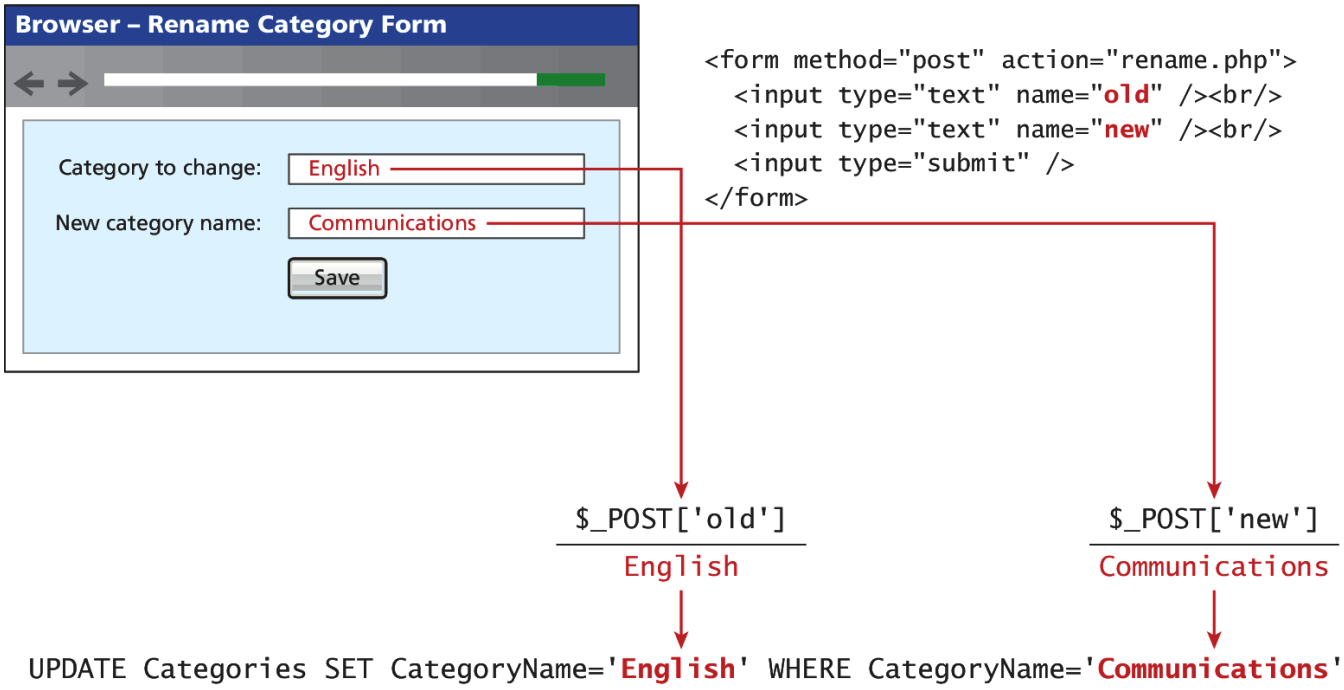
LISTING 11.13 Executing a query that doesn't return data (mysqli)

```
$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.

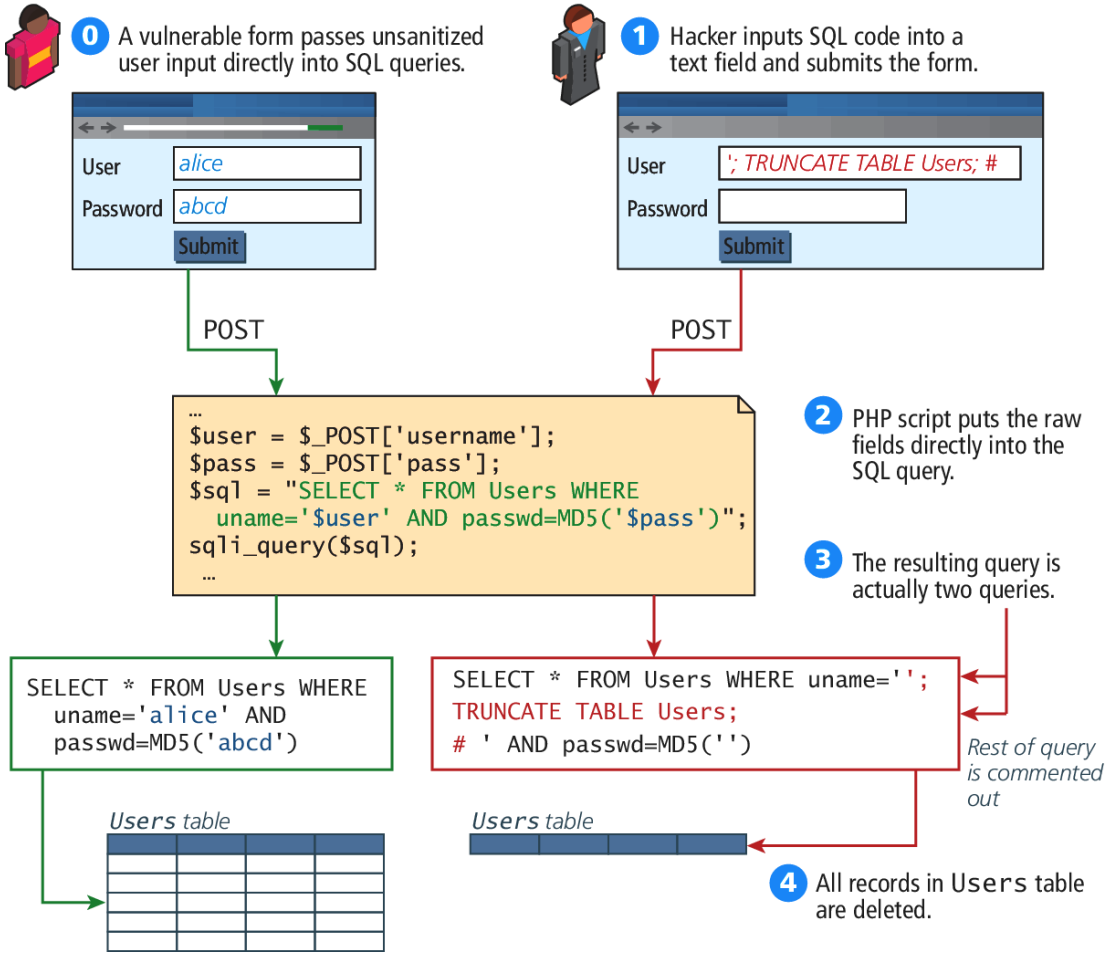
```
$from = $_POST['old'];
$to = $_POST['new'];
$sql = "UPDATE Categories SET CategoryName='$to' WHERE
        CategoryName='$from'";

$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 m ost common web security attacks, the **SQL injection at tack.**

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 spe cial characters from a desired piece of text. In MySQL, user inputs can be sanitized in PHP using the mysqli_re al_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 per formance for queries that need to be executed multipl e times. When MySQL creates a prepared statement, it does something akin to a compiler in that it optimizes i t so that it has superior performance for multiple requ ests. It also integrates sanitization into each user input automatically, thereby protecting us from SQL injectio n.

Prepared Statements

mysqli

```
// 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)

Process Query Results

Mysqli – using prepared statements

```
$sql = "SELECT Title, CopyrightYear FROM Books WHERE ID=?";
if ($statement = mysqli_prepare($connection, $sql)) {

    mysqli_stmt_bindm($statement, 'i', $id);
    mysqli_stmt_execute($statement);

    // bind result variables
    mysqli_stmt_bind_result($statement, $title, $year);

    // loop through the data
    while (mysqli_stmt_fetch($statement)) {
        echo $title . '-' . $year . '<br/>';
    }
}
```

LISTING 11.21 Looping through the result set (mysqli—using prepared statements)

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

Fetch into an object

A more flexible example, where class names needn't match DB fields

```
class Book {
   public $id;
   public $title;
   public $copyrightYear;
   public $description;

   function __construct($record)
   {
      // the references to the field names in associative array must
      // match the case in the table
      $this->id = $record['ID'];
      $this->title = $record['Title'];
      $this->copyrightYear = $record['CopyrightYear'];
      $this->description = $record['Description'];
   }
}
…
// in some other page or class
$statement->execute();

// using the Book class
$b = new Book($statement->fetch());
echo 'ID: ' . $b->id . '<br/>';
echo 'Title: ' . $b->title . '<br/>';
echo 'Copyright Year: ' . $b->copyrightYear . '<br/>';
echo 'Description: ' . $b->description . '<br/>';
```

LISTING 11.24 Letting an object populate itself from a result set

Freeing Resources

And closing the connection

```
// mysqli approach
$connection = mysqli_connect($host, $user, $pass, $database);
…
// release the memory used by the result set. This is necessary if
// you are going to run another query on this connection
mysqli_free_result($result);
…
// close the database connection
mysqli_close($connection);
// PDO approach
$pdo = new PDO($connString,$user,$pass);
…
// closes connection and frees the resources used by the PDO object
$pdo = null;
```

LISTING 11.25 Closing the connection

Using Transactions

mysqli

```
$connection = mysqli_connect($host, $user, $pass, $database);
…
/* set autocommit to off. If autocommit is on, then mysql will
   commit (i.e., make the data change permanent) each command after
   it is executed  */
mysqli_autocommit($connection, FALSE);

/* insert some values */
$result1 = mysqli_query($connection,
   "INSERT INTO Categories (CategoryName) VALUES ('Philosophy')");
$result2 = mysqli_query($connection,
   "INSERT INTO Categories (CategoryName) VALUES ('Art')");

if ($result1 && $result2) {
   /* commit transaction */
   mysqli_commit($connection);
}
else {
   /* rollback transaction */
   mysqli_rollback($connection);
}
```

LISTING 11.26 Using transactions (mysqi extension)

Using Transactions

PDO

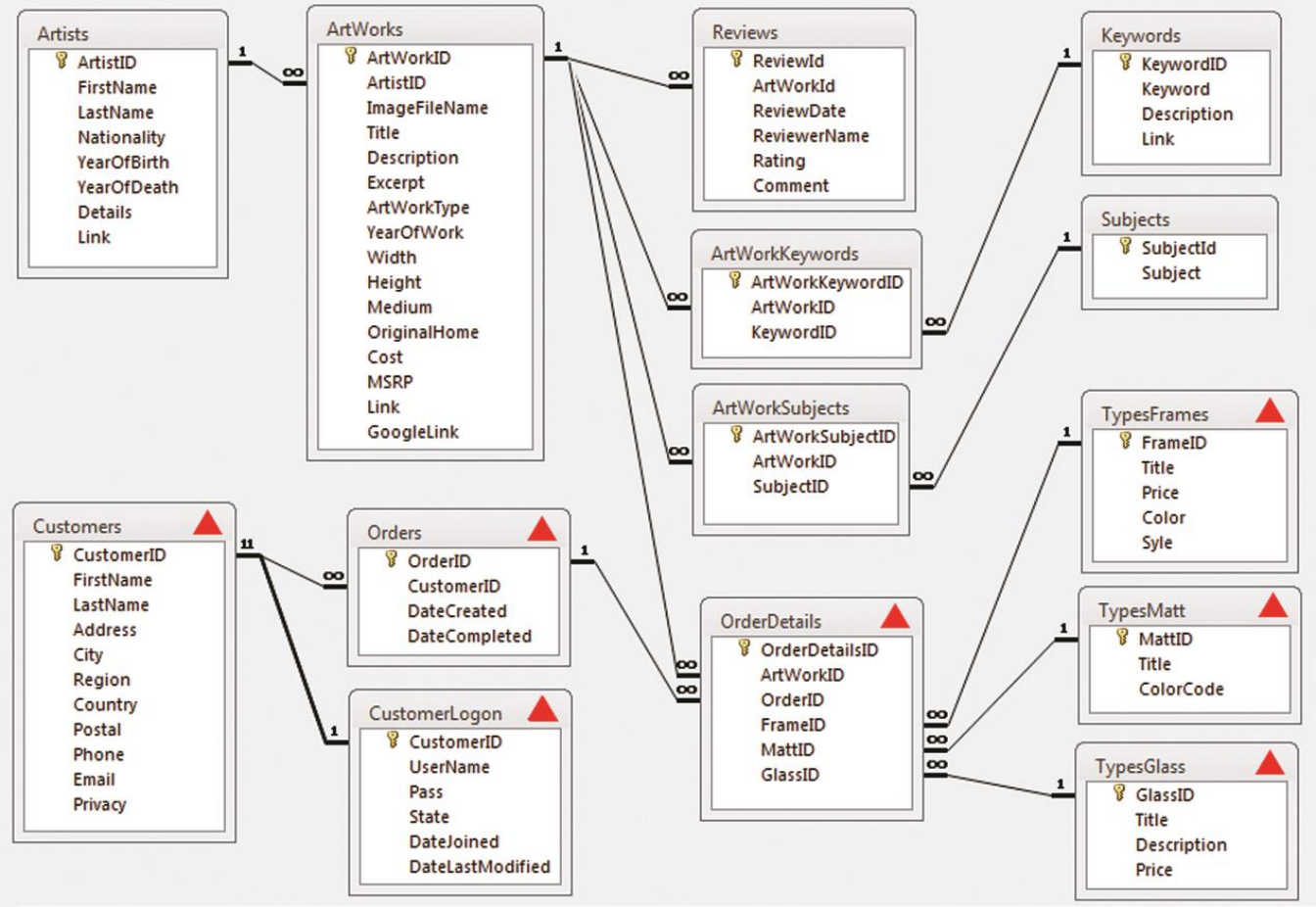
```
$pdo = new PDO($connString,$user,$pass);
// turn on exceptions so that exception is thrown if error occurs
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
…
try {
    // begin a transaction
    $pdo->beginTransaction();

    // a set of queries: if one fails, an exception will be thrown
    $pdo->query("INSERT INTO Categories (CategoryName) VALUES
      ('Philosophy')");
    $pdo->query("INSERT INTO Categories (CategoryName) VALUES
      ('Art')");

    // if we arrive here, it means that no exception was thrown
    // which means no query has failed, so we can commit the
    // transaction
    $pdo->commit();
} catch (Exception $e) {
    // we must rollback the transaction since an error occurred
    // with insert
    $pdo->rollback();
}
```

LISTING 11.27 Using transactions (PDO)

Case Study Schemas

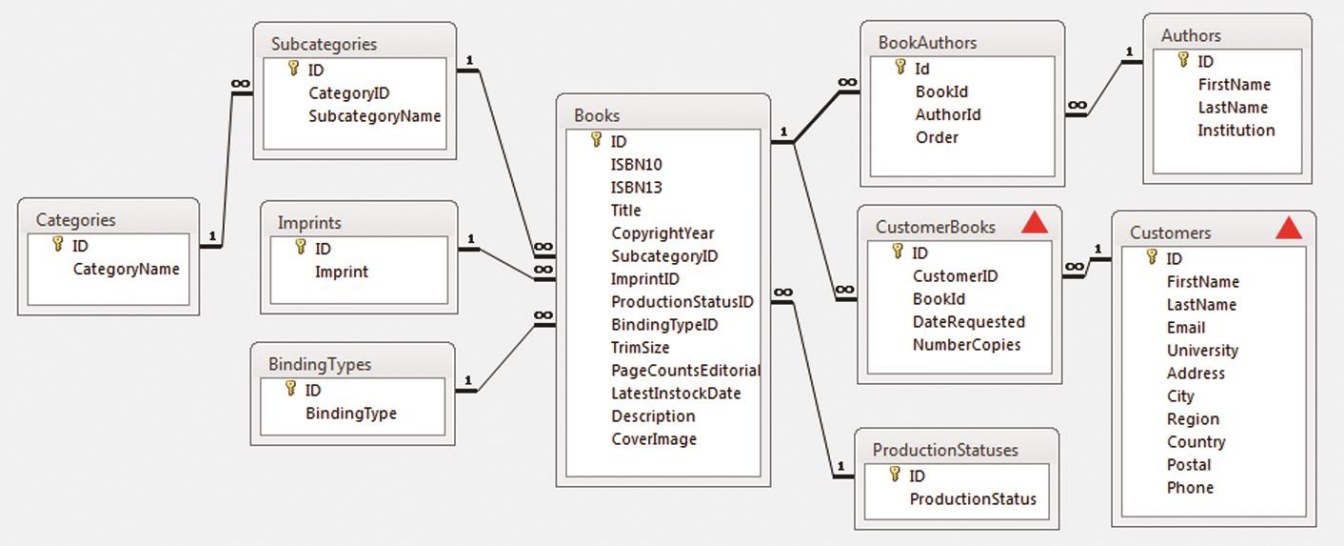
Section 6 of 7

Art Database

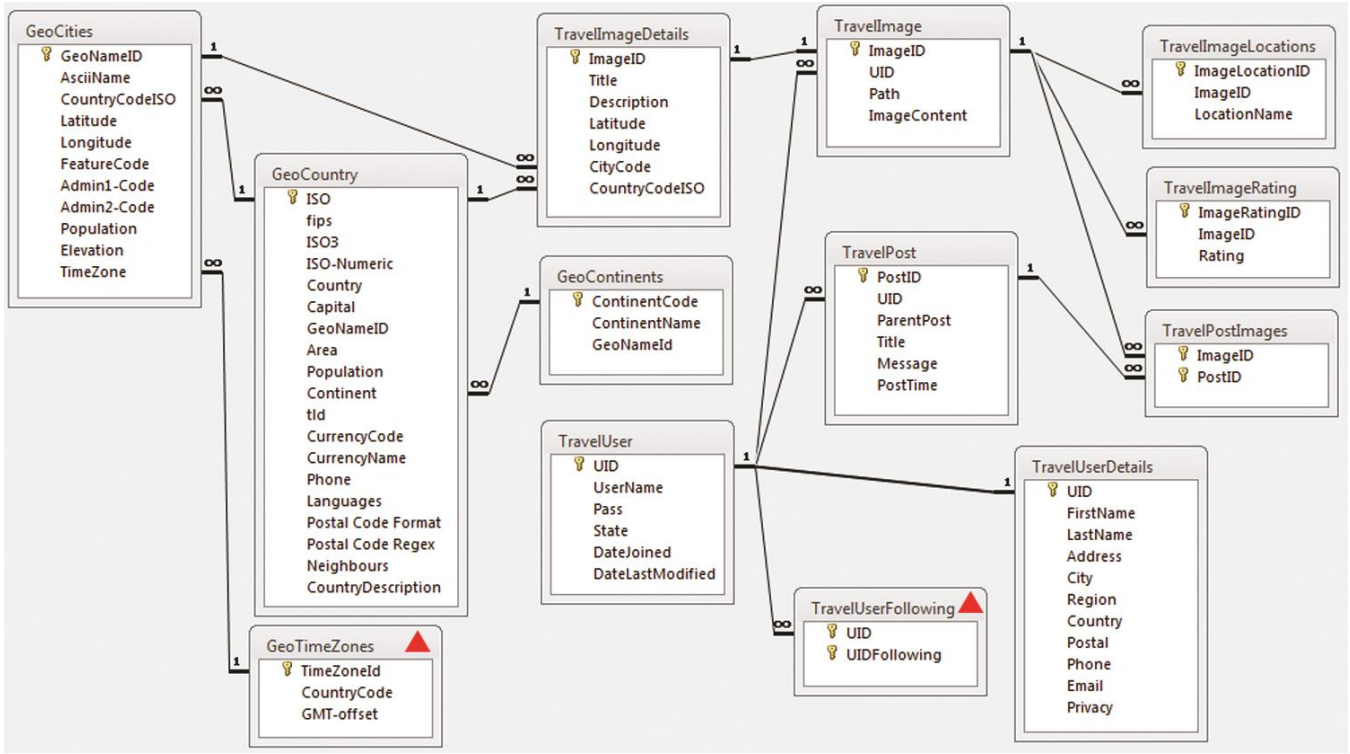


Book CRM Database

Customer Relationship Management



Travel Photo Sharing Database



Sample Database Techniques

Section 7 of 7

Display a list of Links

One of the most common database tasks in PHP is to d
isplay a list of links (i.e., a series of <li> elements withi
n 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      Categ
oryName”;
$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

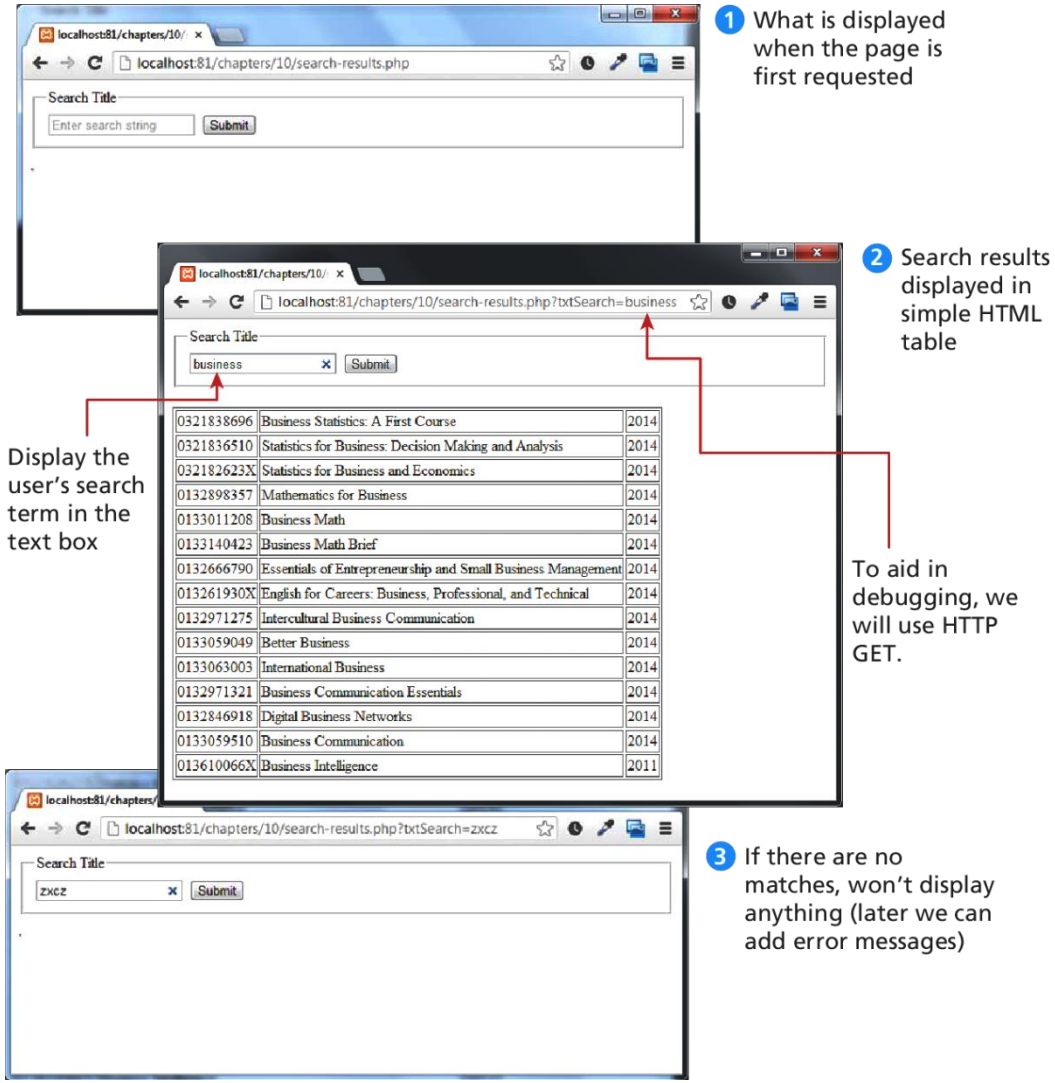
```
<ul>
<?php
$result = getResults();  // some function that returns the result set
while ($row = $result->fetch()) {
?>
   <li>
   <a href="l       ist.php?category=<?php echo $row['ID']; ?>">
   <?php echo $row['CategoryName']; ?>
   </a>
   </li>
<?php } ?>
</ul>
```

ist.php?category=<?php echo $row['ID']; ?>">

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 sea rch string along with a Submit button.

The data that we will filter is the Book table; we will di splay any book records that contain the user-entered t ext 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

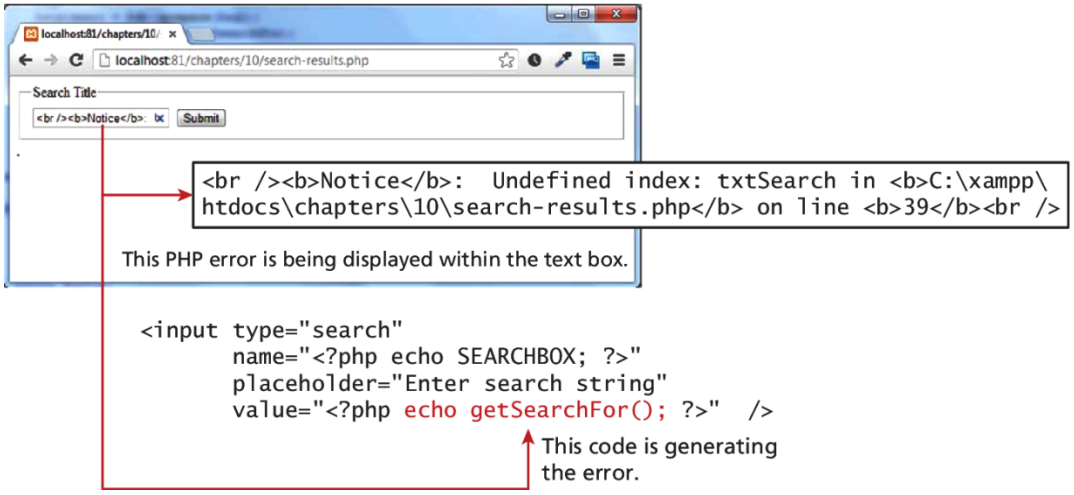
To redisplay the search term we will add code like:

```
<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

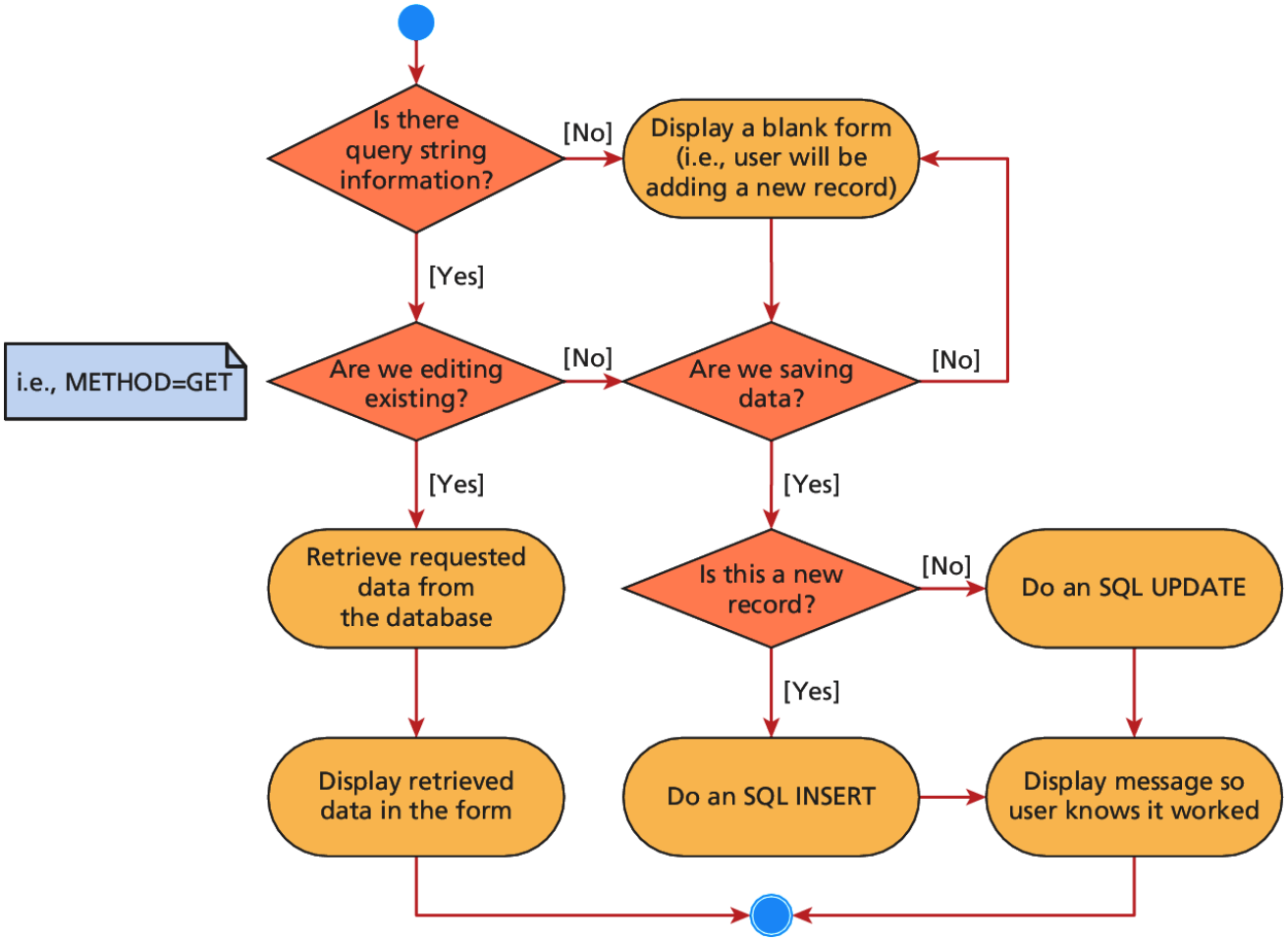
Problem to be solved




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

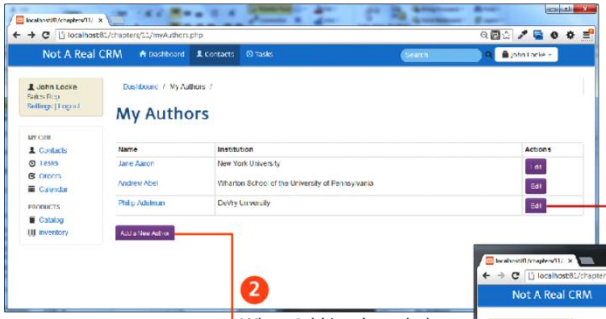
LISTING 11.30 Solution to search results page problem

Editing a Record



Editing a Record

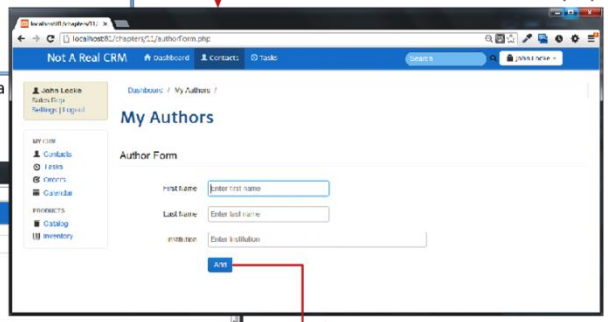
myAuthors.php



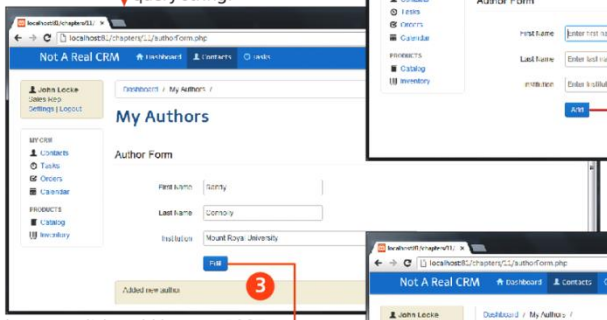
1 List of authors is displayed.

When Edit is selected, GET request is made to authorForm.php with requested author's ID in querystring.

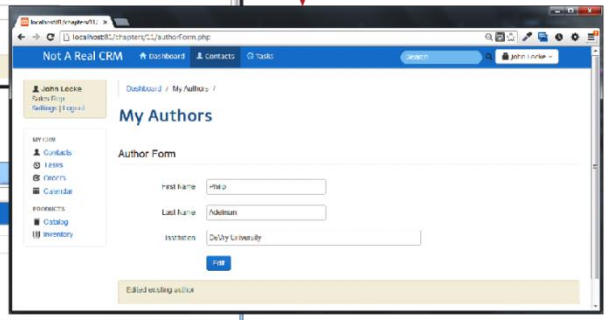
authorForm.php



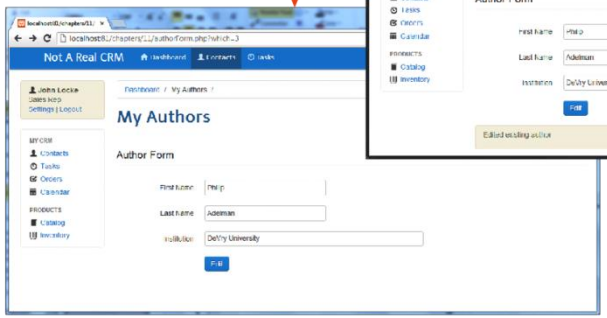
2 When Add is selected, then a GET request is made to authorForm.php with **no** query string.



3 When user clicks Edit button, POST request is made to authorForm.php.



When user clicks Add button, POST request is made to authorForm.php.



4 Page updates record in database table and displays message to provide feedback.

4 Page inserts new record in database table, retrieves the DB-generated ID for the new record, and displays message to provide feedback.

Files in the Database

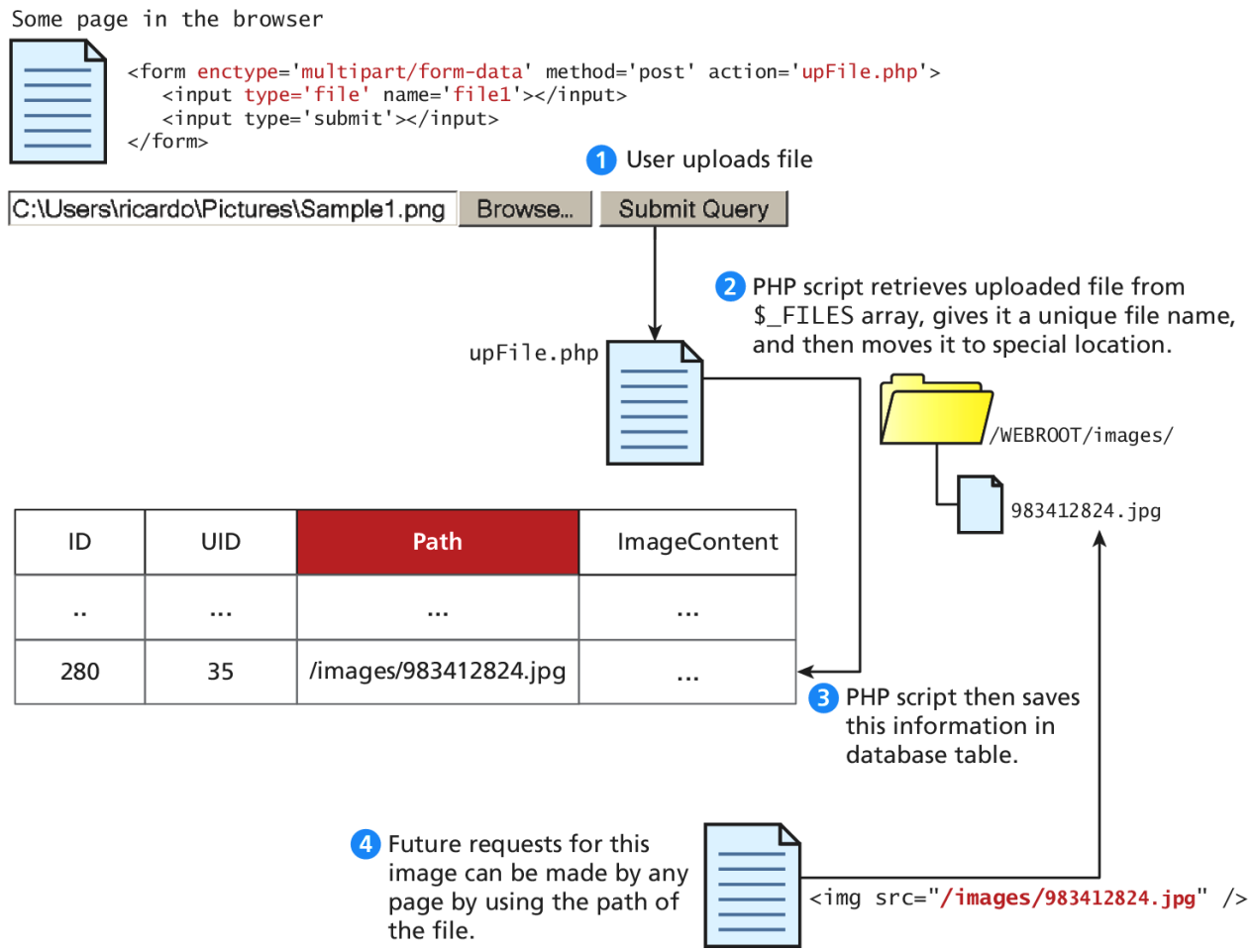
There are two ways of storing files in a database:

1. Storing file location in the database, and storing th e file on the server's filesystem

2. Storing the file itself in the database in the form of a binary BLOB

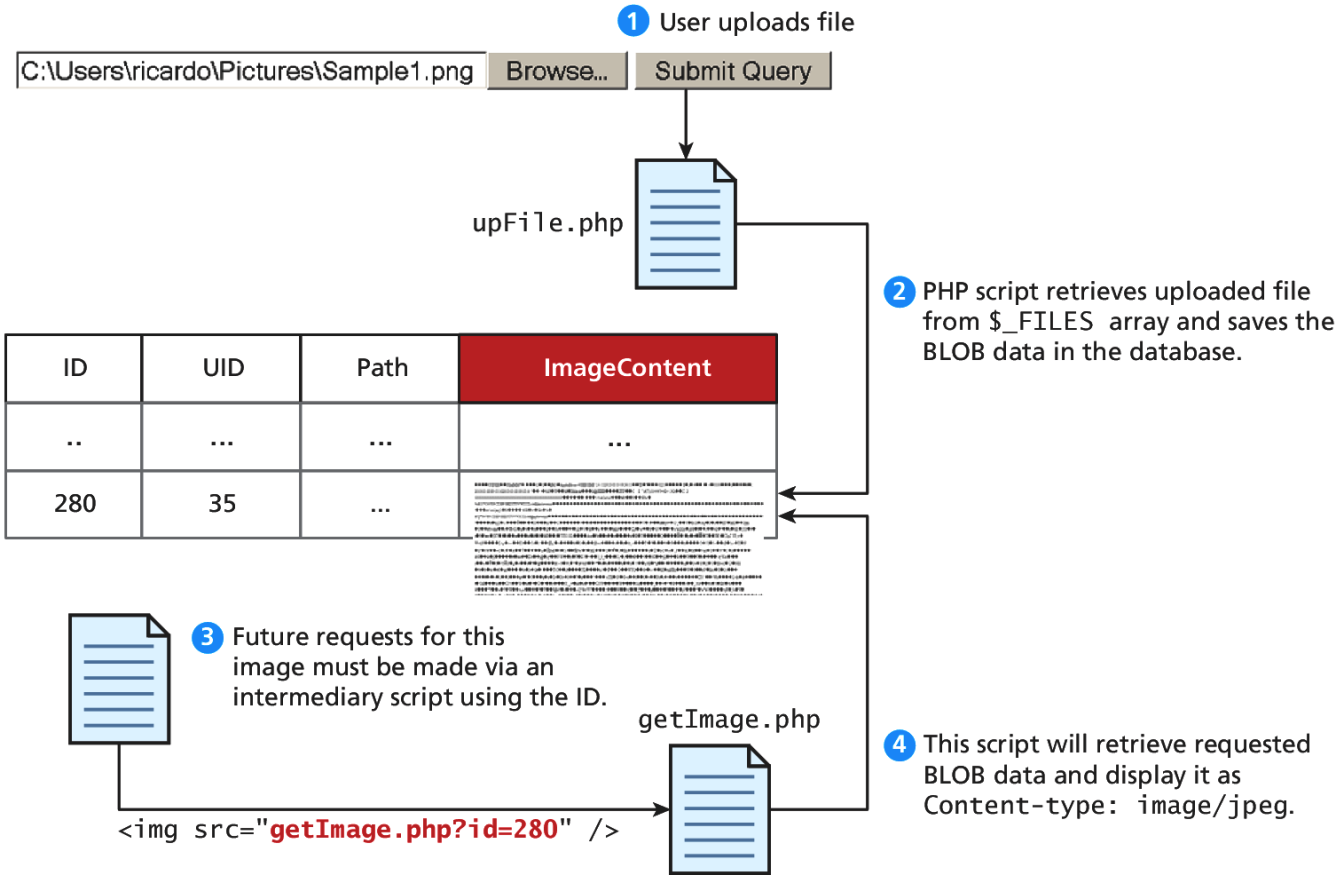
Files in the Database

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

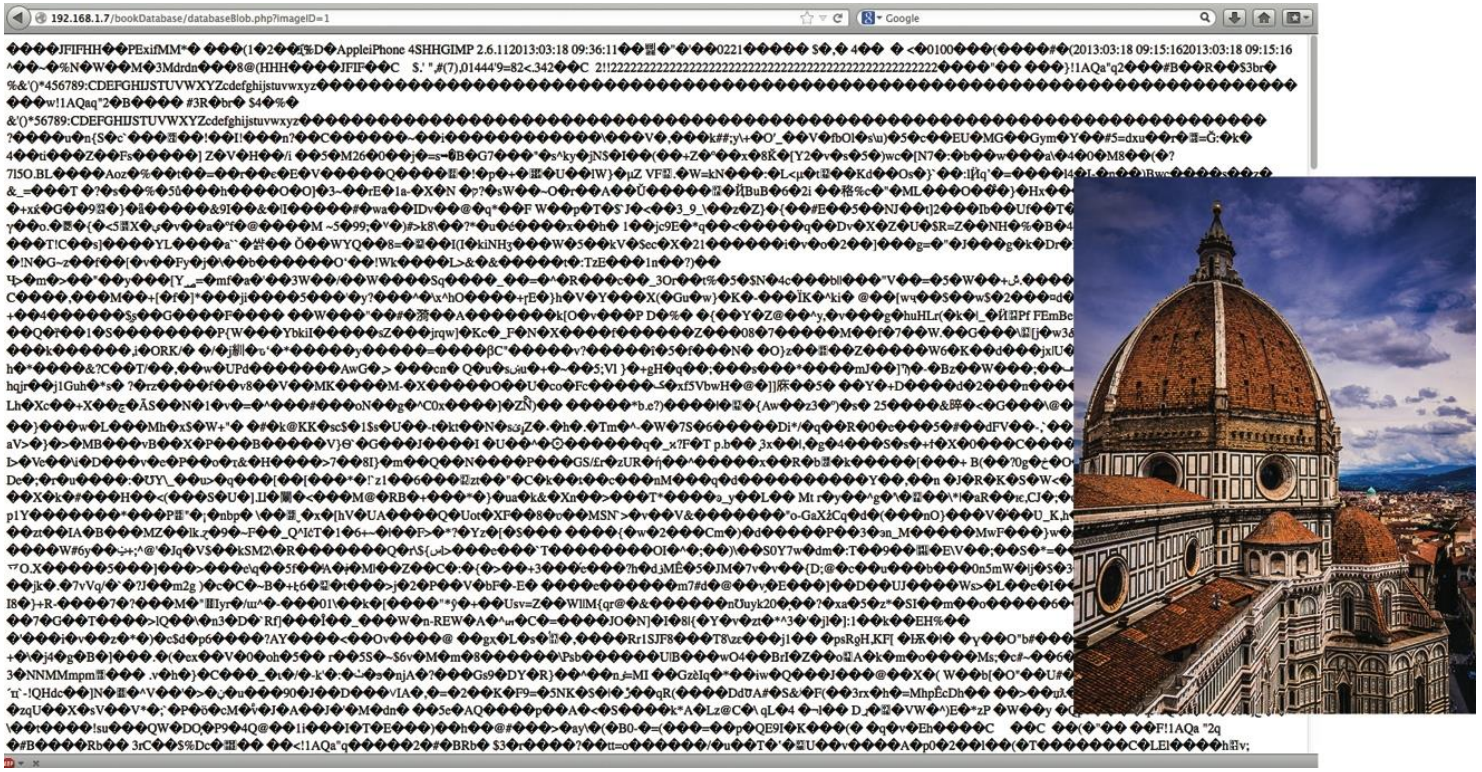
Listing 11.36 can then be integrated into HTML image t ags by pointing the src attribute to our script

Would now reference a dynamic PHP script like:

Files in the Database

HTTP headers matter

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



What You've Learned

- Database and Web Development
- Structured Query Language (SQL)
- Database APIs
- Managing a MySQL Database
- Accessing MySQL in PHP
- Case Study Schemas
- Sample Database Techniques