Additional Notes
- PHP & Databases

Client/Server Programming for Internet Applications

TCSS460

Summer 2020

Databases

- **Database (DB)**
  - collection of related **data**
    - **data** → known facts that can be recorded and that have implicit meaning
  - DB represents some aspect of the real world
  - changes in the real-world are reflected in the DB
  - many open-source and proprietary relational DBMSs
    - MySQL
    - PostgreSQL
    - Oracle Database
    - IBM DB2
    - Microsoft SQL Server

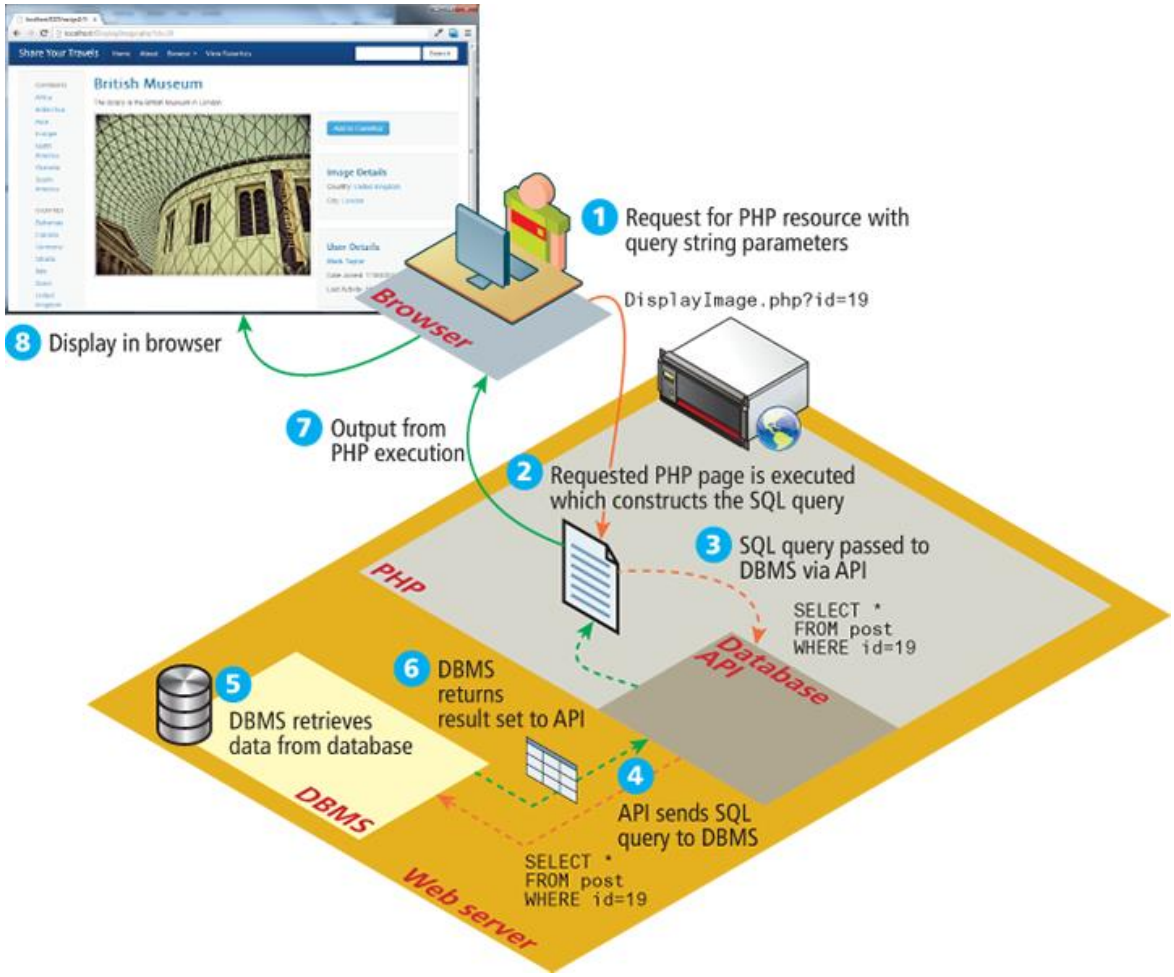
Databases (cont'd)

- **Database Management System (DBMS)**
  - collection of interrelated data and programs that enable users to create and maintain a database

  - DBMS is a general-purpose software system that facilitates the processes of **defining**, **constructing**, **manipulating**, and **sharing databases** among various users and applications
    - allows multiple users/programs to access and manipulate DB concurrently
    - protects DB against unauthorized access and manipulation
    - provides means to evolve DB and program behaviour as requirements change over time

Databases (cont'd)

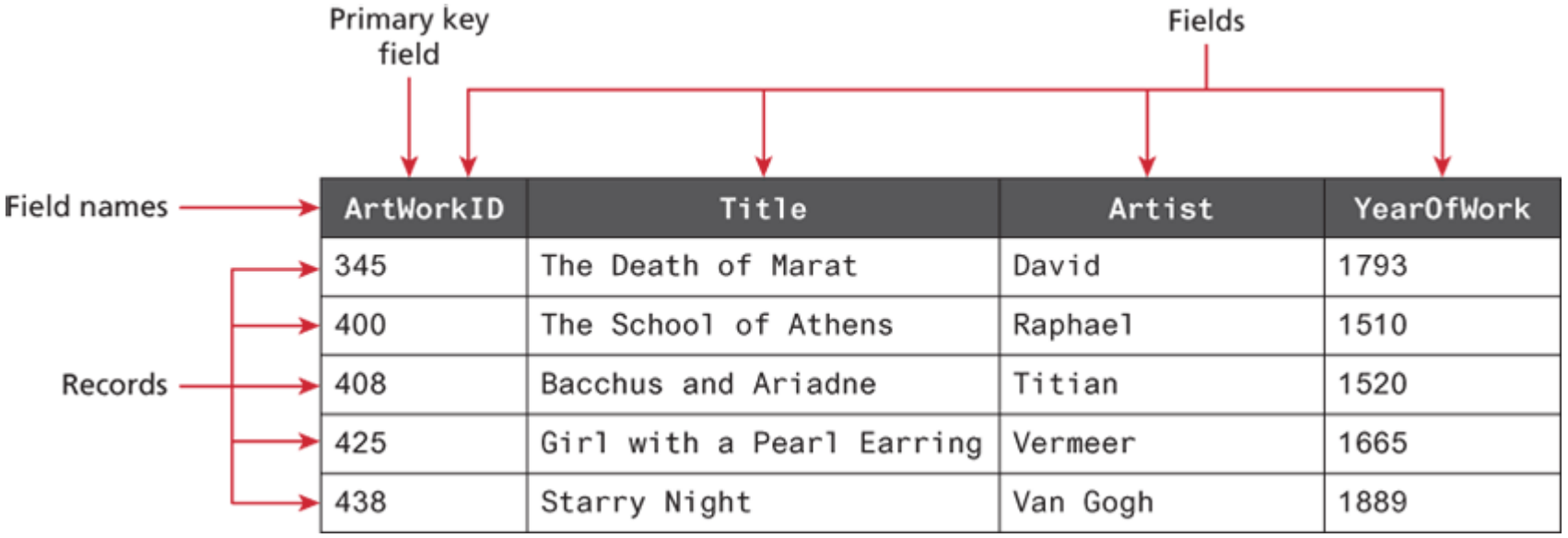
how websites use databases



Databases (cont'd)

database structure

- a database is composed of one or more **tables**
  - a **table** is a set of a ***number*** of **rows** (or **records**)
  - a **row** is an ordered list of ***n*** **values** (or **fields**)
- a **primary key** that is used to uniquely identify each row



ArtWorkID	Title	Artist	YearOfWork
345	The Death of Marat	David	1793
400	The School of Athens	Raphael	1510
408	Bacchus and Ariadne	Titian	1520
425	Girl with a Pearl Earring	Vermeer	1665
438	Starry Night	Van Gogh	1889

Databases (cont'd)

database rules

- a database can enforce rules on stored data
  - data integrity → accuracy and consistency of data
  - minimize data redundancy (or duplication)

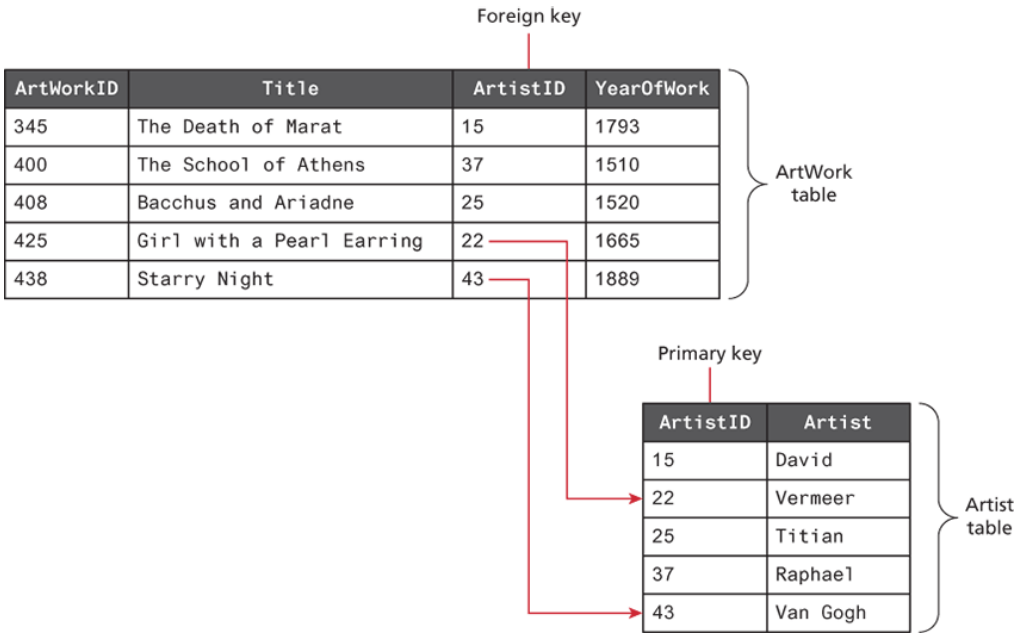
common database data types

Type	Description
BIT	Represents a single bit for Boolean values. Also called `BOOLEAN` or `BOOL`.
BLOB	Represents a binary large object (which could, e.g., 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 are also `TIME` and `DATETIME` data types.
FLOAT	Represents a decimal number. There are also `DOUBLE` and `DECIMAL` 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 characters) with no space padding.

Databases (cont'd)

foreign key

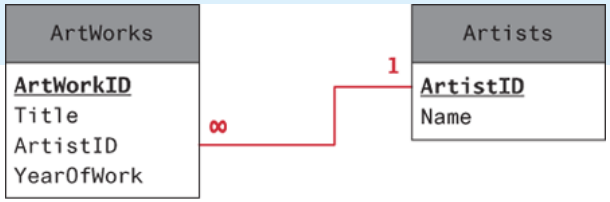
- a good DB design organizes data across multiple tables
  - each table represents some aspect of the real-world
  - more tables translate into more relationships
  - **foreign keys** represent relationships between tables

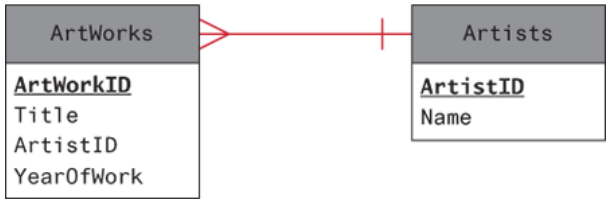


Databases (cont'd)

relationships

- relationship types may vary
  - unary, binary, ternary, etc.
  - more tables translate into more relationships
  - **foreign keys** represent relationships between tables
- *examples*
  - one-to-many relationship
  - one-to-one relationship
  - many-to-many relationship

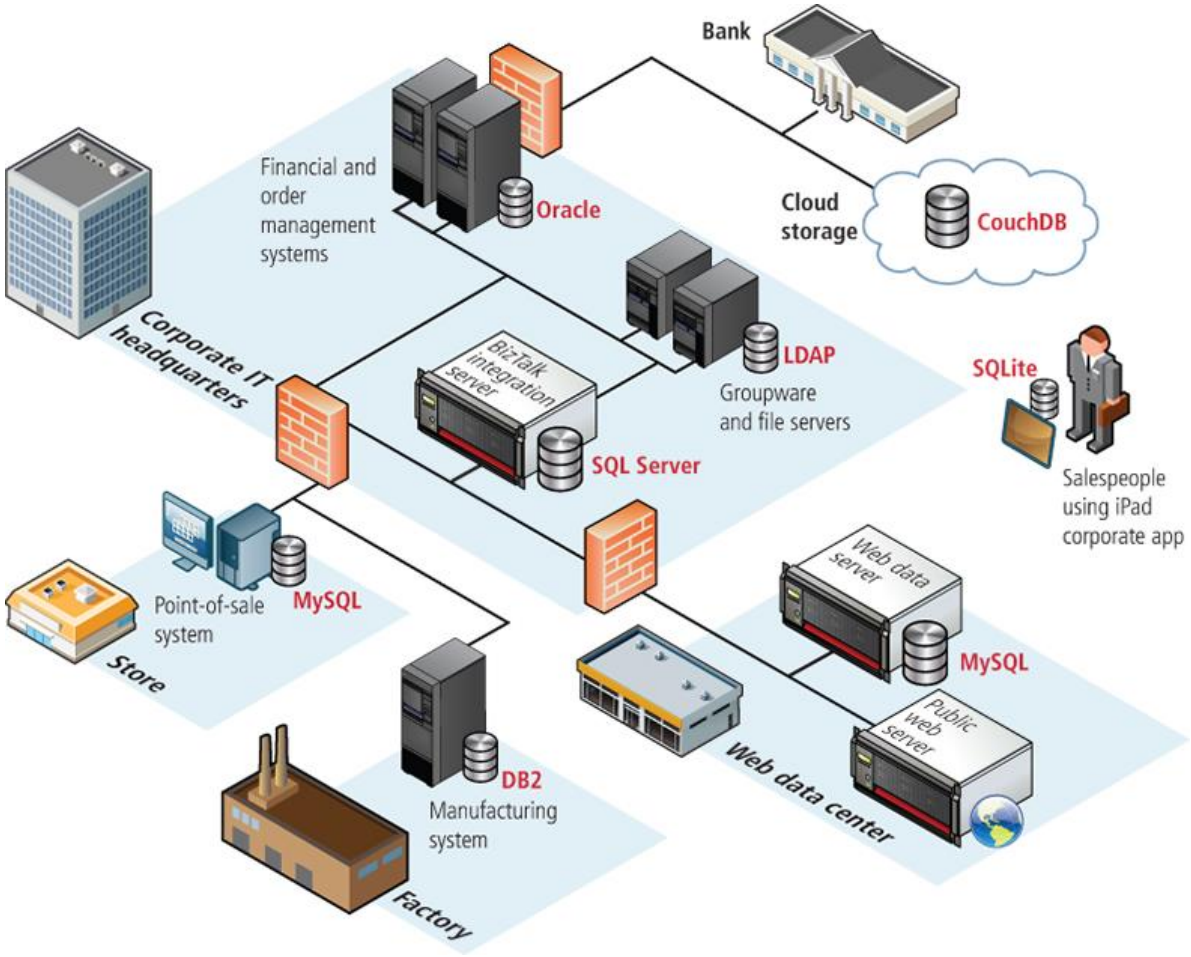




*BookID and AuthorID is a **composite key***

Databases (cont'd)

database options

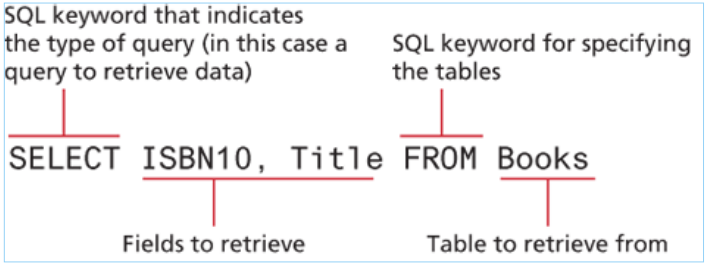


SQL

SQL

- **Structured Query Language** has been the de facto standard for interfacing with a database
  - retrieve, store, modify and delete data
  - create, modify or delete tables

SELECT (used to retrieve data from a database)




```
SELECT * FROM Books
```

Wildcard to select all fields

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

SQL keyword to indicate sort order

Field to sort on

SQL

SELECT

WHERE clause supply a comparison expression that the data must match in order for a record to be included in the result set

```
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)

aggregate functions are used perform some type of calculation on multiple records and then return the results

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

NoSQL

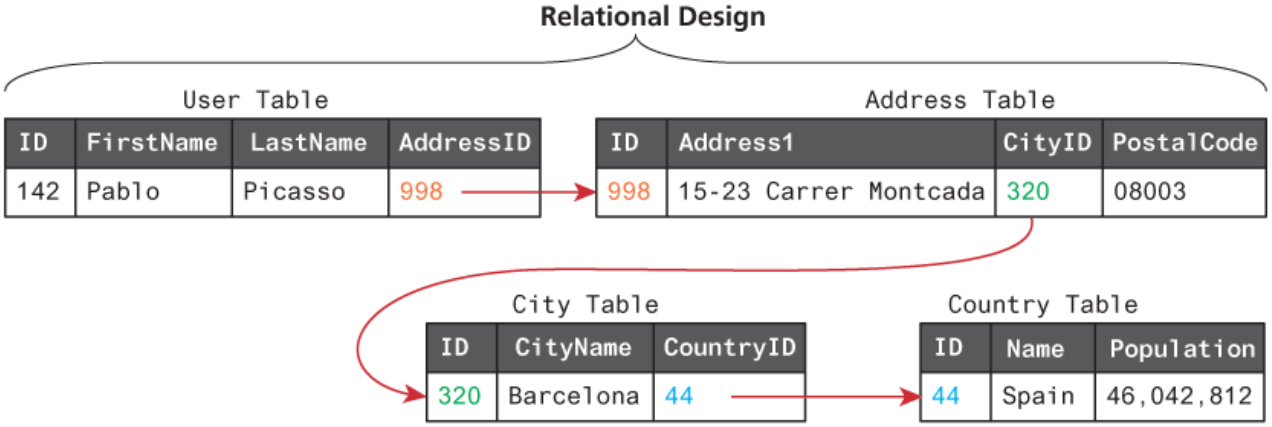
NoSQL refers to non SQL

- **NoSQL** is a non-relational database that is also used for storage and retrieval of data
  - why noSQL?
    - relational databases perform slowly for large volumes of data
    - noSQL relies on different set of ideas for data modeling

- **NoSQL database types**
  - **key-value stores**: stores key-value pairs (similar to hash table)
  - **document stores**: store data in documents (typically XML or JSON)
  - **graph stores**: store information about networks of data
  - **wide-column stores**: stores columns of data together rather than rows

NoSQL

document stores versus relational design



in order to get the equivalent data from a relational model, a relational database has to join the foreign keys across other tables, which can be a ***time-intensive operation*** *when involving very complex queries or when the server is experience high loads. In contrast, the document store requires no joins to retrieve a single user.*

Document Store Design

ID	Document

```
142  {
       "User": {
         "FirstName": "Pablo",
         "LastName": "Picasso",
         "Address": {
             "Address1": "15-23 Carrer Montcada",
             "City": "Barcelona",
             "Country": {
                   "Name": "Spain",
                  "Population": 46042812
               },
             "PostalCode": "08003"
         }
       }
     }
```

NoSQL

column-wise stores versus relational design

Row-wise storage

Row #	ID	Title	Artist	Year
1	345	The Death of Marat	David	1793
2	400	The School of Athens	Raphael	1510
3	408	Bacchus and Ariadne	Titian	1521
4	425	Girl with a Pearl Earring	Vermeer	1665
5	438	Starry Night	Van Gogh	1889

- *in traditional relational database systems, the data in tables is stored in a **row-wise** manner.*
- *this means that the fundamental unit of data retrieved is a **row**.*

- *column-wise store systems store data by column instead of by row*
- *retrieves a column of data*
- *retrieving an entire row requires multiple operations*

Column-wise storage

	ID
1	345
2	400
3	408
4	425
5	438

	Title
1	The Death of Marat
2	The School of Athens
3	Bacchus and Ariadne
4	Girl with a Pearl Earring
5	Starry Night

	Artist
1	David
2	Raphael
3	Titian
4	Vermeer
5	Van Gogh

	Year
1	1793
2	1510
3	1521
4	1665
5	1889

Database APIs

PHP MySQL APIs

- two basic styles of database APIs available in PHP
  - **procedural API**: uses function calls to work with the database
  - **object-oriented API**: requires instantiating objects and invoking methods and properties
- three main database API options available in PHP
  1. **MySQL extension**
     - original extension to PHP for working with MySQL and has been replaced with the newer mysqli extension
     - procedural API , used with versions of MySQL older than 4.1.3
  2. **mysqli extension**
     - MySQL Improved extension takes advantage of features of versions of MySQL after 4.1.3
     - provides both a procedural and object-oriented approach

Database APIs

PHP MySQL APIs (cont'd)

- three main database API options available in PHP

3. **PHP data objects (PDOs)**
   - available since PHP 5.1 and provides an abstraction layer (a set of classes that hide the implementation details for some set of functionality)
   - appropriate drivers can be used with any database, and not just MySQL databases
   - does not support all of the latest features of MySQL

- **which one to choose?**
  - it depends
  - may require access multiple database types → use PDO

Database APIs

Managing a MySQL Database

- Command-Line Interface

```
mysql -h 192.168.1.14 -u bookUser -p
```

  - once inside a MySQL session, you can use (;) to terminate a query

  - you can also import and export entire databases or run a batch of SQL commands
    - e.g. import commands from a file called commands.sql

```
mysql -h 192.168.1.14 -u bookUser -p  < commands.sql
```

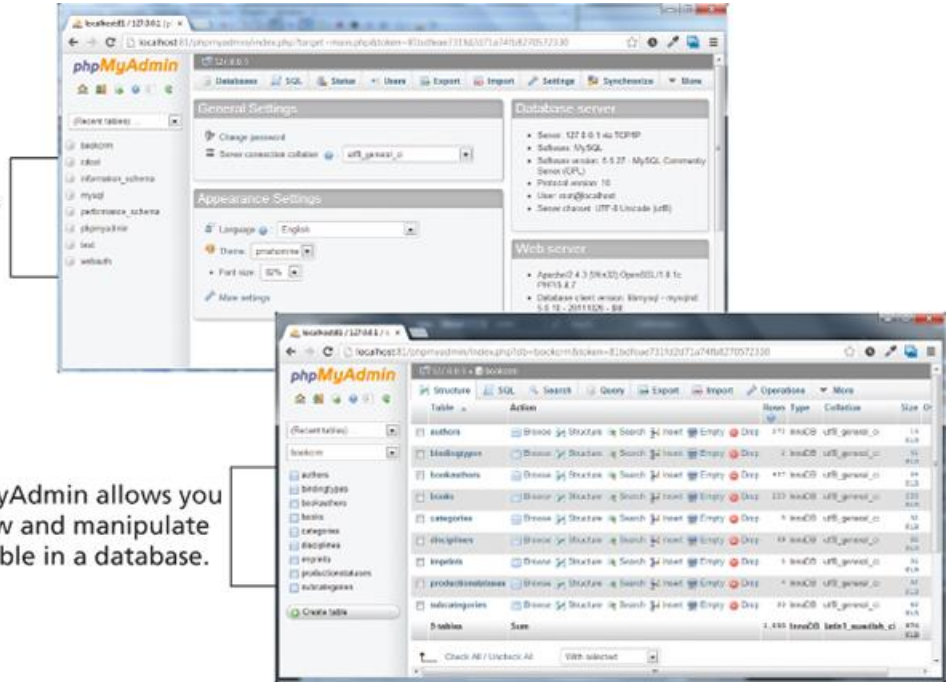
```
Database changed
mysql> SHOW TABLES;
+---------------------------+
| Tables_in_book_database |
+---------------------------+
| authors                   |
| bindingtypes              |
| bookauthors               |
| books                     |
| categories                |
| disciplines               |
| 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>
```

Database APIs

phpMyAdmin

- is a popular web-based front-end (**written in PHP**) that allows developers to access management tools through a web portal
- provides a clickable interface that lets you navigate your databases more intuitively
- can be launched (usually) via http://localhost/phpmyadmin

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



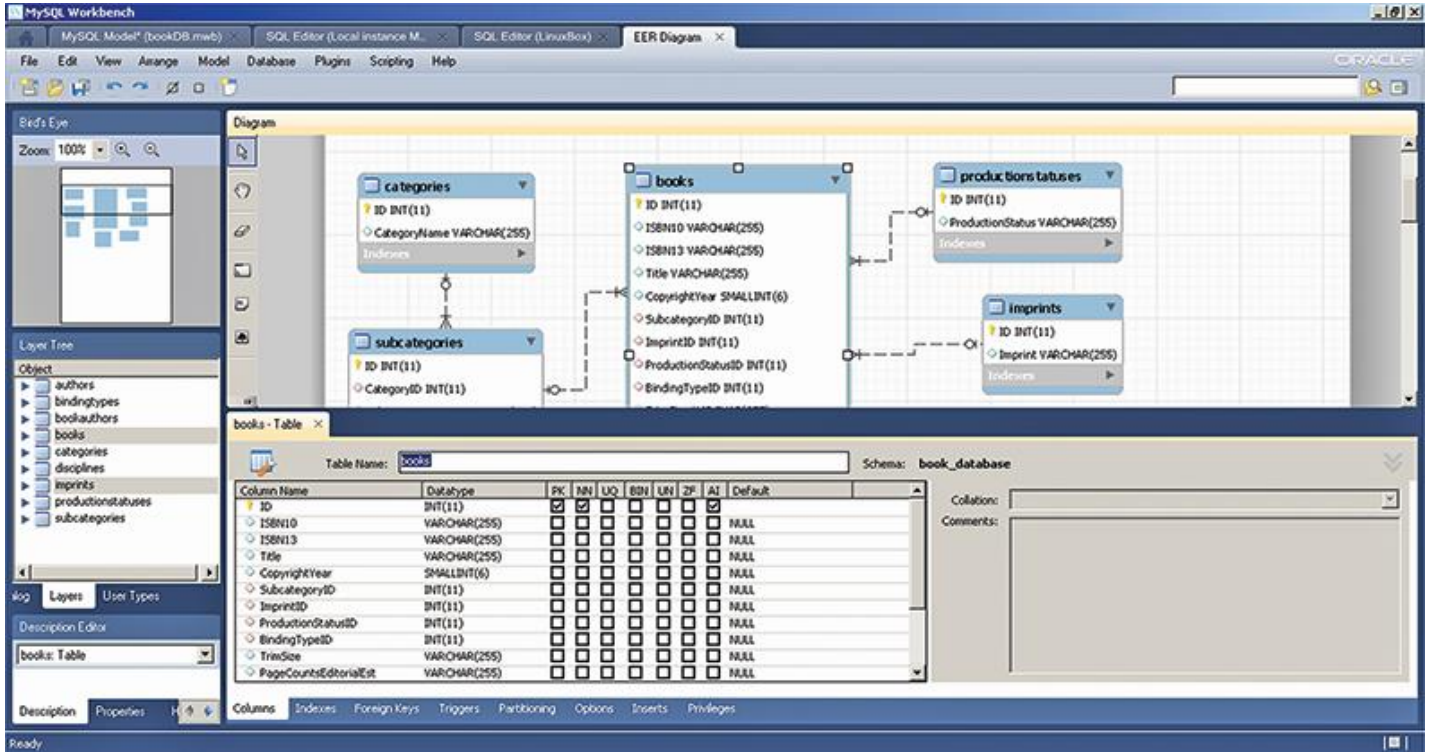
- phpMyAdmin configuration:

config.inc.php

Database APIs

MySQL Workbench

- is a free tool from Oracle to work with MySQL databases
- it provides a visual interface for building and viewing tables and queries
- it can also auto generate an entity relationship diagram (ERD)



Accessing MySQL in PHP

- **five steps** involved

1. **connect** to the database
2. **handle** connection **errors**
3. **execute** the sql query
4. **process** the results
5. free resources and **close connection**

```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() );
}

?>
```

Accessing MySQL in PHP (cont'd)

Connecting to a Database

- before we start running queries, we need to setup a connection to a database
  - **(PDO) connection string**: a standard way to specify connection details

Connecting to a database with mysqli (procedural)

```
// modify these variables for your installation
$host = "localhost";
$database = "bookcrm";
$user = "testuser";
$pass = "mypassword";
$connection = mysqli_connect($host, $user, $pass, $database);
```

Connecting to a database with PDO (object-oriented)

```
// modify these variables for your installation
$connectionString = "mysql:host=localhost;dbname=bookcrm";
$user = "testuser";
$pass = "mypassword";
$pdo = new PDO($connectionString, $user, $pass);
```

Accessing MySQL in PHP (cont'd)

Storing Connection Details

- common practice: define connection details via constants in a separate file (*config.php*)

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

- once this file is defined, we can simply use the `require_once()`

using the connection constants

```
require_once('protected/config.php');
$connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME);
```

Accessing MySQL in PHP (cont'd)

Handling Connection Errors

- there are a number of different ways of handling these errors
  - example of two possible ways

Handling connection errors with mysqli (version 1)

```
$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);
}
```

Handling connection errors with mysqli (version 2)

```
$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()
}
```

Accessing MySQL in PHP (cont'd)

Handling connection errors with PDO

- Using PDO, we use try … catch exception handling blocks

Handling connection errors with PDO

```
try {
      $connString = "mysql:host=localhost;dbname=bookcrm";
      $user = DBUSER;
      $pass = DBPASS;
      $pdo = new PDO($connString,$user,$pass);
      …
}
catch (PDOException $e) {
      die( $e->getMessage() );
}
```

- PDO Exception Modes
  - **PDO::ERRMODE_SILENT** → default mode, PDO sets error code for you (preferred approach once a site is in production use)
  - **PDO::ERRMODE_WARNING** → in addition to error code, PDO will output a warning message (useful debugging/testing, useful to see the problem without interrupting flow of application
  - **PDO::ERRMODE_EXCEPTION** → PDO will throw a PDOException, script stops at point of error

Accessing MySQL in PHP (cont'd)

Executing Queries

- once connection is established successfully, you are ready to construct and execute queries
- typically involves creating a string that contains the sql
- then, call one of the query functions

Executing a SELECT query (mysqli)

```
$sql = "SELECT * FROM Categories ORDER BY CategoryName";
// returns a mysqli_result object
$result = mysqli_query($connection, $sql);
```

Executing a SELECT query (pdo)

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

Accessing MySQL in PHP (cont'd)

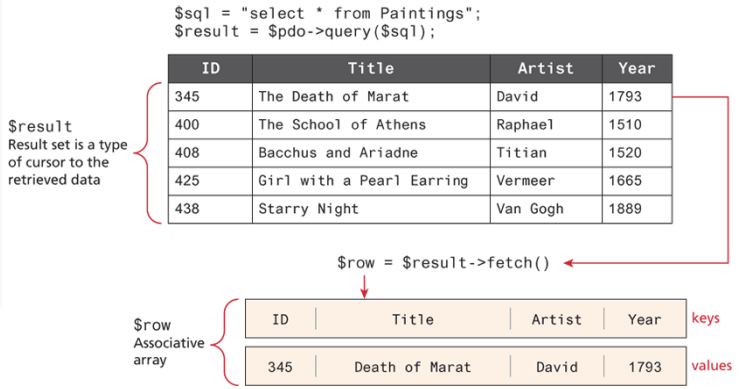
Processing the Query Results

- if you are running a SELECT query, then you will want to do something with the retrieved result set
  - e.g. , either display it, or perform calculations on it, or search for something in it, or some other operation

fetch function must be called to move the data from the database result set to a regular PHP array.

Looping through the result set (PDO)

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



alternatively…use `foreach`

```
foreach ($result as $row) {
echo $row[0] . " - " . $row[1] . "<br/>";
```

Accessing MySQL in PHP (cont'd)

fetching using mysqli extension

- fetching using the older mysqli extension is more varied in that there are several different fetch functions

Type	Description
`mysqli_fetch_all()`	Fetches all result rows as an associative array, a numeric array, or both.
`mysqli_fetch_array()`	Fetches a result row as an associative array, a numeric array, or both.
`mysqli_fetch_assoc()`	Fetches a result row as an associative array.
`mysqli_fetch_field()`	Returns the next field in the result set. That is, it returns definition information about a single table column (not its data).
`mysqli_fetch_fields()`	Returns an array of objects representing the fields in a result set.
`mysqli_fetch_object()`	Returns the current row of a result set as an object.
`mysqli_fetch_row()`	Gets a result row as a numeric array.

Accessing MySQL in PHP (cont'd)

fetching into an object

- as an alternative to fetching into an array, you can **fetch directly into a custom object** and then use properties to access the field data

```
class Book {
      public $ID;
      public $Title;
      public $CopyrightYear;
      public $Description;
}
```

we can then have PHP populate an object of type Book

```
$sql = "SELECT * FROM Books";
$result = $pdo->query($sql);
// fetch a record into an object of type Book
while ( $b = $result->fetchObject('Book') ) {
      // the property names match the field names from the table
      echo 'ID: ' . $b->ID . '<br/>';
      echo 'Title: ' . $b->Title . '<br/>';
      echo 'Year: ' . $b->CopyrightYear . '<br/>';
      echo 'Description: ' . $b->Description . '<br/>';
      echo "<hr>";
}
```

Accessing MySQL in PHP (cont'd)

fetching into an object (cont'd)

- a more flexible object-oriented approach would be to have the Book object populate its own properties from the record data passed in the object **constructor**

```
class Book {
      public $id;
      public $title;
      public $year;
      public $description;
      function __construct($record)
      {
        $this->id = $record['ID'];
        $this->title = $record['Title'];
        $this->year = $record['CopyrightYear'];
        $this->description = $record['Description'];
      }
}
…
```

```
// in some other page or class
$sql = "SELECT * FROM Books";
$result = $pdo->query($sql);
// fetch a record normally
while ( $row = $result->fetch() ) {
      $b = new Book($row);
      echo 'ID: ' . $b->id . '<br/>';
      echo 'Title: ' . $b->title . '<br/>';
      echo 'Year: ' . $b->year . '<br/>';
      echo 'Description: ' . $b->description . '<br/>';
      echo "<hr>";
}
```

Accessing MySQL in PHP (cont'd)

Working with Parameters

- INSERT, UPDATE, and DELETE statements perform an action on the data

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

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

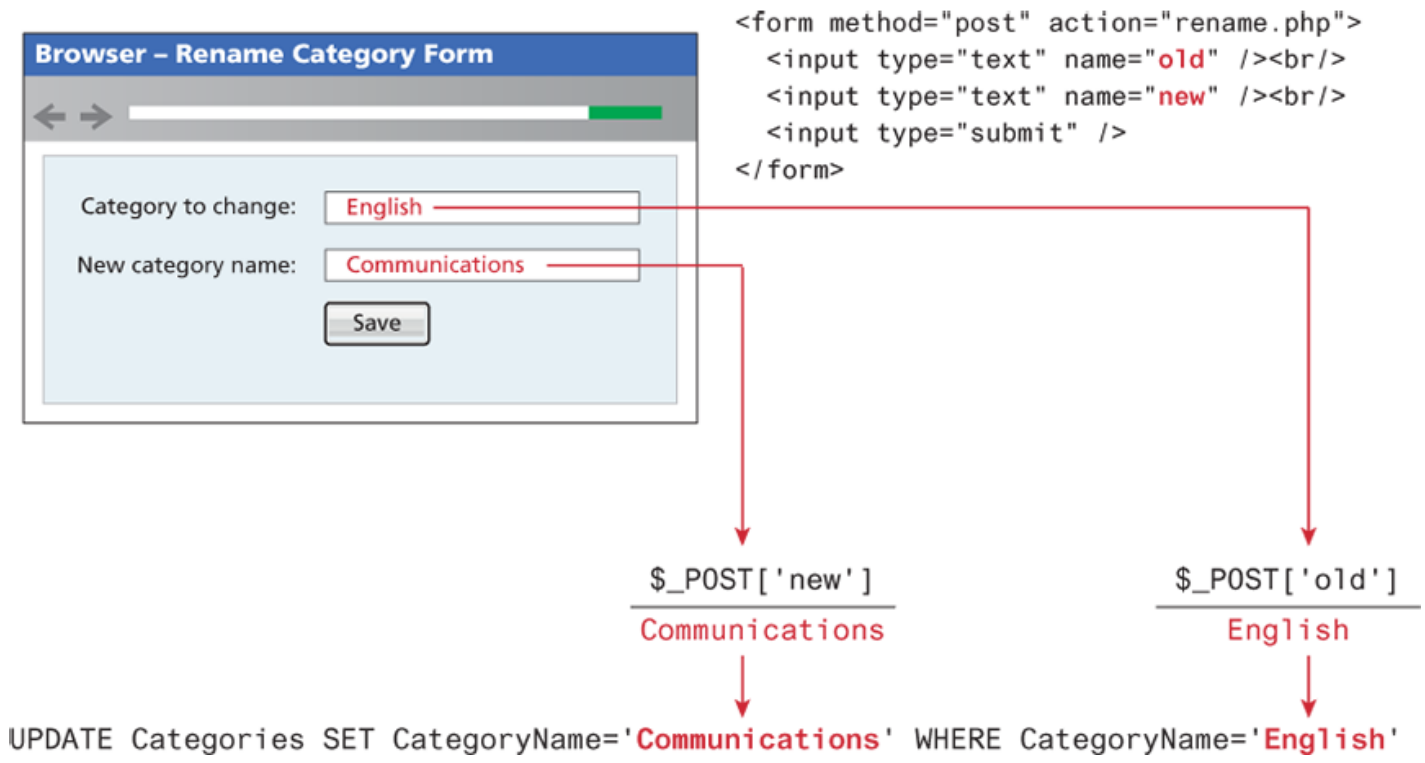
Executing a query that doesn't return data (mysqli)

```
$sql = "UPDATE Categories SET CategoryName='Web' WHERE CategoryName='Business'";
if ( mysqli_query($connection, $sql) ) {
    $count = mysqli_affected_rows($connection);
    echo "<p>Updated " . $count . " rows</p>";
}
```

Accessing MySQL in PHP (cont'd)

Integrating User Data

- it is common that you run a query that uses some type of user input contained within a query string parameter



Accessing MySQL in PHP (cont'd)

Integrating User Data (cont'd)

- integrating user input into a query (first attempt)

```
$from = $_POST['old'];
$to = $_POST['new'];
$sql = "UPDATE Categories SET CategoryName='$to' WHERE CategoryName='$from'";
$count = $pdo->exec($sql);
```

Using a prepared statement (PDO)

```
// retrieve parameter value from query string
$id = $_GET['id'];
/* method 1 - notice the ? parameter */
$sql = "SELECT Title, CopyrightYear FROM Books WHERE ID = ?";
$statement = $pdo->prepare($sql);
$statement->bindValue(1, $id); // bind to the 1st ? parameter
$statement->execute();
/* method 2 */
$sql = "SELECT Title, CopyrightYear FROM Books WHERE ID = :id";
$statement = $pdo->prepare($sql);
$statement->bindValue(':id', $id);
$statement->execute();
```

second approach to binding values uses a **named parameter** which assigns labels in prepared SQL statements which are then explicitly bound to variables in PHP

Accessing MySQL in PHP (cont'd)

Using named parameters (PDO)

```
/* technique 1 - question mark placeholders, explicit binding */
$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();
```

Accessing MySQL in PHP (cont'd)

Using named parameters (PDO) (cont'd)

```
/* 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();
```