Introduction to Databases (PHP + MySQL)

Objectives

- How to use relational databases for storing and accessing data
- Querying using SQL
- Using MySQL
- Accessing MySQL from PHP

Relational Databases

Relational Databases

A ***database*** is an organized collection of data

Allows for relatively easy access for retrievals, additions, and deletions

A ***database management system (DBMS)*** provides mechanisms for storing, organizing, modifying and retrieving data

Most popular databases are ***relational databases*** - logical representation that allows the data to be accessed without consideration of its physical structure

Relational Databases

Data is stored in ***tables***

- A table stores attributes for data of a specific kind
- Example: employee data

Tables consist of ***rows*** and ***columns***

- Each row contains data associated with a specific data item
- Each column contains data associated with a specific attribute
- One special column stores the ***primary keys*** of the table

Tables can be related to one another

Relational Databases

Example: designing a relational database for used Corvettes that are for sale

- Vette_id
- Body_style
- Miles
- Year
- State
- Equipment

Could just put all data in a **big** single table, whose ***key*** would be a simple sequence number

The table could have information about various equipment the cars could have

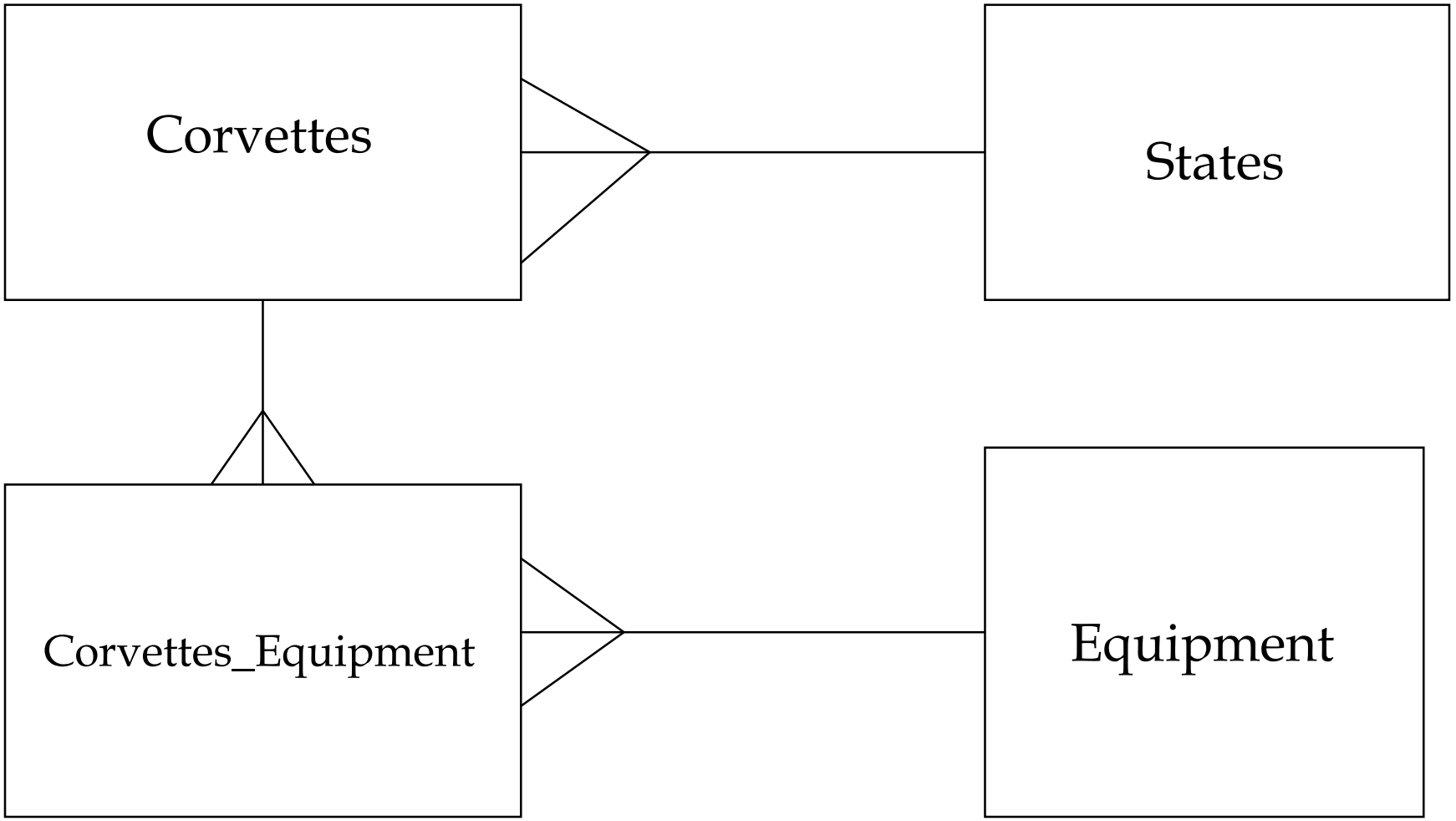
Relational Databases

Problem: a lot of duplicate data in the table

Better to put the equipment in a different table and use a ***cross-reference table*** to relate cars to equipment

For example, to save space, use a separate table for state names, with only references in the main table

Relational Databases



The `Corvettes` Table

Vette_id	Body_style	Miles	Year	State
1	coupe	18.0	1997	4
2	hatchback	58.0	1996	7
3	convertible	13.5	2001	1
4	hatchback	19.0	1995	2
5	hatchback	25.0	1991	5
6	hardtop	15.0	2000	2
7	coupe	55.0	1979	10
8	convertible	17.0	1999	5
9	hardtop	17.0	2000	5
10	hatchback	50.0	1995	7

The `States` Table

State_ID	State
1	Alabama
2	Alaska
3	Arizona
4	Arkansas
5	California
6	Colorado
7	Connecticut
8	Delaware
9	Florida
10	Georgia

The `Equipment` Table

Equip_id	Equipment
1	Automatic
2	4-speed
3	5-speed
4	6-speed
5	CD
6	leather

The `Corvettes_Equipment` cross-reference table

Vette_id	Equip
1	1
1	5
1	6
2	1
2	5
2	6
3	1
3	6
4	2
4	6
5	1
5	6
6	2
⋮	⋮

Structured Query Language (SQL)

Structured Query Language (SQL)

SQL is a standard language to create, query, and modify relational databases

- Supported by all major database vendors
- More like structured English than a programming language

We will cover 6 basic SQL commands

- **`SELECT`**: Retrieves records from a table
- **`INSERT`**: Adds records to a table
- **`UPDATE`**: Updates records in a table
- **`DELETE`**: Deletes records from a table
- **`DROP`**: Deletes a database or table
- **`CREATE`**: Creates a database or table

Syntax notes

- SQL statements end with a semi-colon
- SQL reserved words are case insensitive

The SELECT Command

SELECT is the most common SQL command

It's used to specify *queries* (i.e. retrieve data)

Has three clauses: SELECT, FROM, and WHERE

General form:

SELECT *column name(s)*

FROM *table names*

WHERE *condition*

The SELECT Command

Basic statement

```
SELECT field1, field2 FROM table;
```

Example:

```
SELECT Body_style, Year FROM Corvettes;
```

Use * to select all fields, e.g.:

```
SELECT * FROM Corvettes;
```

The SELECT Command

To retrieve data with specific criteria - use a **WHERE** clause, e.g.:

```
SELECT Body_style FROM Corvettes WHERE Year > 1994
```

Useful comparison operators:

= , > , <, >=, <=, != or <> (not equal)

IS NOT NULL, IS NULL, BETWEEN

Example:

```
SELECT * FROM Corvettes WHERE Year BETWEEN
1995 and 2000;
```

Joins

If you want all cars that have CD players, you need information from two tables, `Corvettes` and `Equipment`

`SELECT` can build a temporary table with info from two tables, from which the desired results can be retrieved

This is called a ***join*** of the two tables

A `SELECT` that does a join operation specifies two tables in its `FROM` clause and also has a compound `WHERE` clause

Joins

For our example, we must have three `WHERE` conditions:

- The `Vette_ids` column from the `Corvettes` table and the `Corvettes_Equipment` table must match
- The `Equip` column from the `Corvettes_Equipment` table must match the `Equip_id` column from the `Equipment` table
- The `Equip` column from the `Equipment` table must have the value `'CD'`

Joins

Example:

```
SELECT Corvettes.Vette_id,
 Corvettes.Body_style, Corvettes.Miles,
 Corvettes.Year, Corvettes.State,
 Equipment.Equip
FROM Corvettes, Equipment
WHERE Corvettes.Vette_id =
 Corvettes_Equipment.Vette_id AND
 Corvettes_Equipment.Equip =
 Equipment.Equip_id AND Equipment.Equip =
 'CD'
```

Joins

The query produces:

VETTE_ID	BODY_STYLE	MILES	YEAR	STATE	EQUIPMENT
1	coupe	18.0	1997	4	CD
2	hatchback	58.0	1996	7	CD
8	convertible	17.0	1999	5	CD
9	hardtop	17.0	2000	5	CD
10	hatchback	50.0	1995	7	CD

The `INSERT` Command

`INSERT` is used to insert data into a database

General form:

`INSERT INTO` *table_name* `(`*col_name*$_1$, … *col_name*$_n$`)`
`VALUES (`*value*$_1$, …, *value*$_n$`)`

The correspondence between column names and values is positional

The INSERT Command

Example:

```
INSERT INTO Corvettes(Vette_id, Body_style,
    Miles, Year, State)
  VALUES (37, 'convertible', 25.5, 1986, 17)
```

The `UPDATE` Command

`UPDATE` is used to change one or more values of a row in a table

`UPDATE` *table_name*
  `SET` $col\_name_1 = value_1$, … $col\_name_n = value_n$
  `WHERE` *col_name* = *value*

The `WHERE` clause is the primary key of the row to be updated

The `UPDATE` Command

Example:

```
UPDATE Corvettes
  SET Year = 1996
  WHERE Vette_id = 17
```

The DELETE Command

Use the **DELETE FROM** statement to remove rows

Can be combined with a WHERE clause

Example:

```
DELETE FROM Corvettes
  WHERE Vette_id = 27
```

The DROP Command

DROP is used to delete whole databases or complete tables

General format

```
DROP (TABLE | DATABASE) [IF EXISTS] name
```

Example:

```
DROP TABLE IF EXISTS States
```

The `CREATE TABLE` command

`CREATE TABLE` is used to make new tables in a database

General form

`CREATE TABLE` *table_name* (*column_name*$_1$ *data_type constraints*, … *column_name*$_n$ *data_type constraints*)

There are many different data types:

`INTEGER`, `REAL`, `CHAR`(length), TIMESTAMP

There are several constraints possible:

`NOT NULL`, `PRIMARY KEY`, etc.

The `CREATE TABLE` command

Example:

```
CREATE TABLE Equipment
   (Equip_id INT UNSIGNED NOT NULL
   AUTO_INCREMENT PRIMARY KEY,
   Equip  INT UNSIGNED );
```

Start of Session

Course Number: CPSC-24700

Instructor: Eric Pogue

MySQL

The MySQL Database System

MySQL is a free, efficient, widely used SQL implementation

(available from `http://www.mysql.org`)

Logging on to MySQL (starting it):

`mysql` [`-h` host] [`-u` username] [database name] [`-p`]

- **Host** is the name of the MySQL server (default - user's machine)
- **Username** is that of the database (default - name used to log into the system)
- The given database name becomes the "**focus**" of MySQL

The MySQL Database System

If you want to access an existing database,
but it was not named in the `mysql` command,
you must choose it for focus with

`USE` database_name;

Example:

```
USE cars;
```

Response is: `Database changed`

MySQL Commands

To create a new database use:

```
CREATE DATABASE database_name;
```

Example:

```
CREATE DATABASE cars;
```

We can then create a table, e.g.:

```
CREATE TABLE Equipment (
  Equip_id INT UNSIGNED NOT NULL
                AUTO_INCREMENT PRIMARY KEY,
  Equip  INT UNSIGNED );
```

MySQL Commands

To see the tables of a database:

```
SHOW TABLES;
```

To see the description of a table (columns):

```
DESCRIBE Corvettes;
```

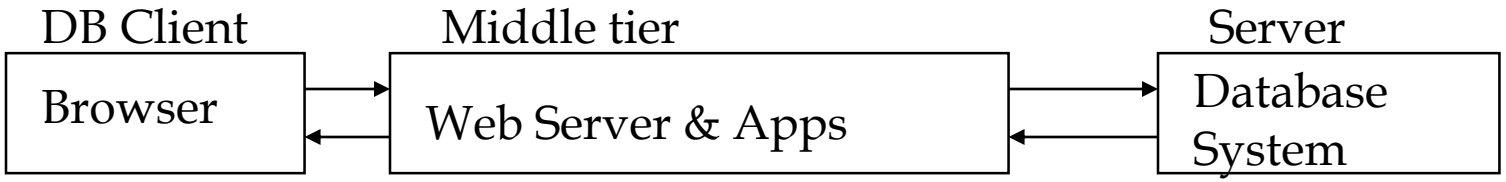
Client-Server Architecture

- Client tasks:
  - Provide a way for users to submit queries
  - Run applications that use the results of queries
  - Display results of queries

- Server tasks:
  - Implement a data manipulation language, which can directly access and update the database

Architectures for Database Access

Web-based applications typically use a ***three-tier architecture***



Accessing MySQL from PHP

PHP & Database Access

There is an **API** for each specific database system

It provides functions for opening/closing connections, running queries, and processing results

Accessing a MySQL database using PHP

Typical web scenario:

1. A web browser issues an HTTP request for a web page.

2. The web server receives the request for the web page, retrieves the file and passes it to the PHP engine for processing

3. PHP processes the script and makes a connection to the MySQL server and sends the appropriate query

Accessing a MySQL database using PHP

4. The MySQL query receives the database query, processes it and sends the result back to the PHP engine

5. The PHP engine finishes running the script and formats the query results

6. The web server passes the HTML to the browser

Query a database from the web

General steps for handling requests:

1. Check and filter data coming from user
2. Set up a connection with the database
3. Query the database
4. Retrieve the results
5. Present the results to the user

Checking and filtering data

First strip any whitespace

(user might have inadvertently entered blank spaces at the beginning or end of his search term)

- Use the `trim()` function
- Example

```
$searchterm=trim($_POST["searchterm"]);
```

Checking and filtering data

Remove string special characters (', ", \, and NULL), which could come from $_GET and $_POST

- Fixed by magic_quotes_gpc (ON by default)
  - This backslashes these special characters
- Example:

```
$query = "SELECT * FROM Names WHERE Name = $name";
```

  - Without magic_quotes , if the value of $name is O'Shanter, it would prematurely terminate the query string
  - But with magic_quotes_gpc on, it will be converted to O\'Shanter
  - Extra slashes can be removed with strip_slashes

Then make any other necessary checks

Connecting PHP to a database

To connect PHP to a database,
use the `mysqli_connect` function

Four parameters:

- host (default is localhost), e.g.: "front.cs.lewisu.edu"
- Username (default is the username of the PHP script)
- Password (default is blank, which works if the database does not require a password)
- Database (default is none, database is selected later)

Close the database with the `mysqli_close()` function

- `mysqli_close($dbc);`

Connecting PHP to a database (cont.)

Example:

Create a connection and select the database

```
$dbc = mysqli_connect('front.cs.lewisu.edu',
        'cs247_name', 'pwd', 'cs247_name');
```

Place in an if-else construct to handle errors

Select the database

To select the DB, either:

Specify in connection

or

Use `mysqli_select_db`

- Example:
  ```
  mysqli_select_db($dbc, "books");
  ```

Query the database

Query using:

```
mysqli_query(connection, query_string);
```

Example:

```
$query="SELECT * FROM books";
$result = mysqli_query($dbc, $query);
```

Returns false on failure

Processing the query results

Get the number of rows returned using `mysqli_num_rows()`:

- Example:

```
$num_results = mysqli_num_rows($result);
```

Use a loop to process all rows

- Use `mysqli_fetch_assoc()` to retrieve a row
- This returns an array where the key is the associated column name in the database

Processing Query Results (cont.)

Make sure to format data to be displayed in HTML

- Use function `stripslashes()` to remove slashes added for SQL (if necessary)
- Use `htmlspecialchars()` to format text that may contain html markup
  - Replaces ampersand, less than, greater than, single quotes and double quotes

Disconnect from the database

After using DB, close it using

```
mysqli_close()
```

Example:

```
mysqli_close($dbc);
```

PHP MySQL errors

The `mysqli_error()` function returns the last error message for the most recent mysqli function call that can succeed or fail

Example:

```
mysqli_error($dbc)
```

Adding information to the database

Updating is done the same way
as retrieving data from the database:

- Retrieve and filter data from user
- Connect to database
- Query the database
- Retrieve and process the results
- Disconnect from the database

Need to validate all user input

Deleting record

To delete records, we need to filter out wildcards so that a user does not destroy the database

Allow access only to a user's own record

- Use sessions or cookies

Summary

- Relational databases store data as a set of tables
- The SQL language can be used to create, update, and retrieve information in a database
- Databases are accessed using an API - functions to support opening/closing DBs and queries
- MySQL is an example of a popular relational database
- Handling user requests for data involves preprocessing the requests, sending query to DB, processing results, and sending results back to the user

End of Session

Course Number: CPSC-24700

Instructor: Eric Pogue