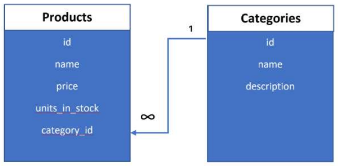
WEEK 7

REVIEW

- fetch() implements a request to an API using ___________
- In a relational database, data for a business entity such as "products" is stored in a _________
- Given the ERD below, how would you show all products?
- How would you show all product names and the name of their category?
- How could you show all categories with “bread” in the description
- How could you set it up to allow multiple categories per product?



Products

id

name

price

units_in_stock

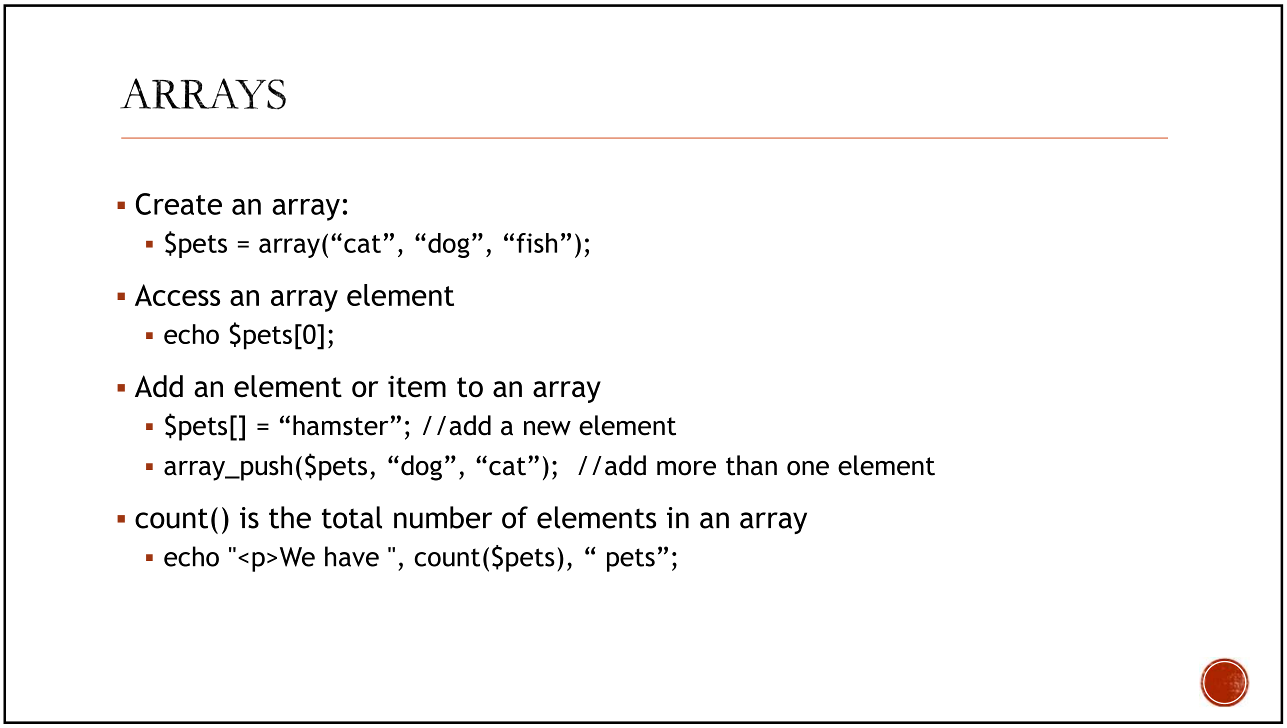
category_id

Categories

id

name

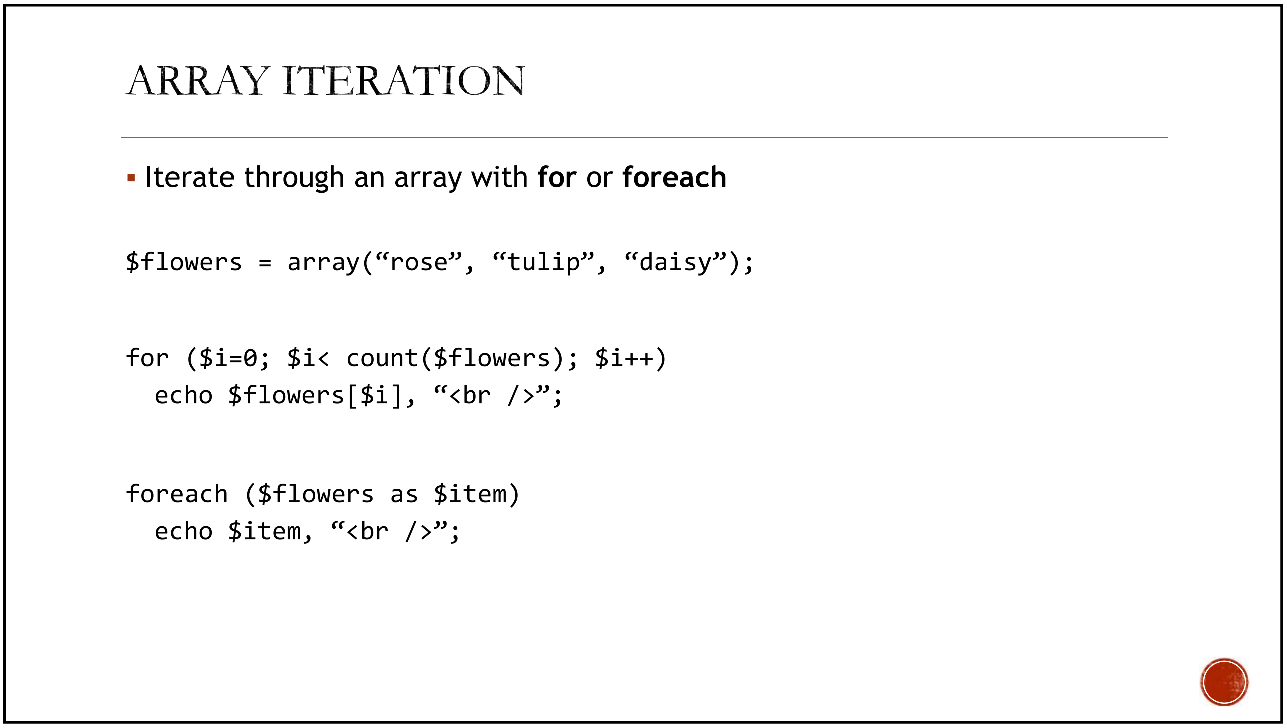
description



ARRAYS

- Create an array:
  - $pets = array(“cat”, “dog”, “fish”);
- Access an array element
  - echo $pets[0];
- Add an element or item to an array
  - $pets[] = “hamster”; //add a new element
  - array_push($pets, “dog”, “cat”); //add more than one element
- count() is the total number of elements in an array
  - echo "<p>We have ", count($pets), “ pets”;

37

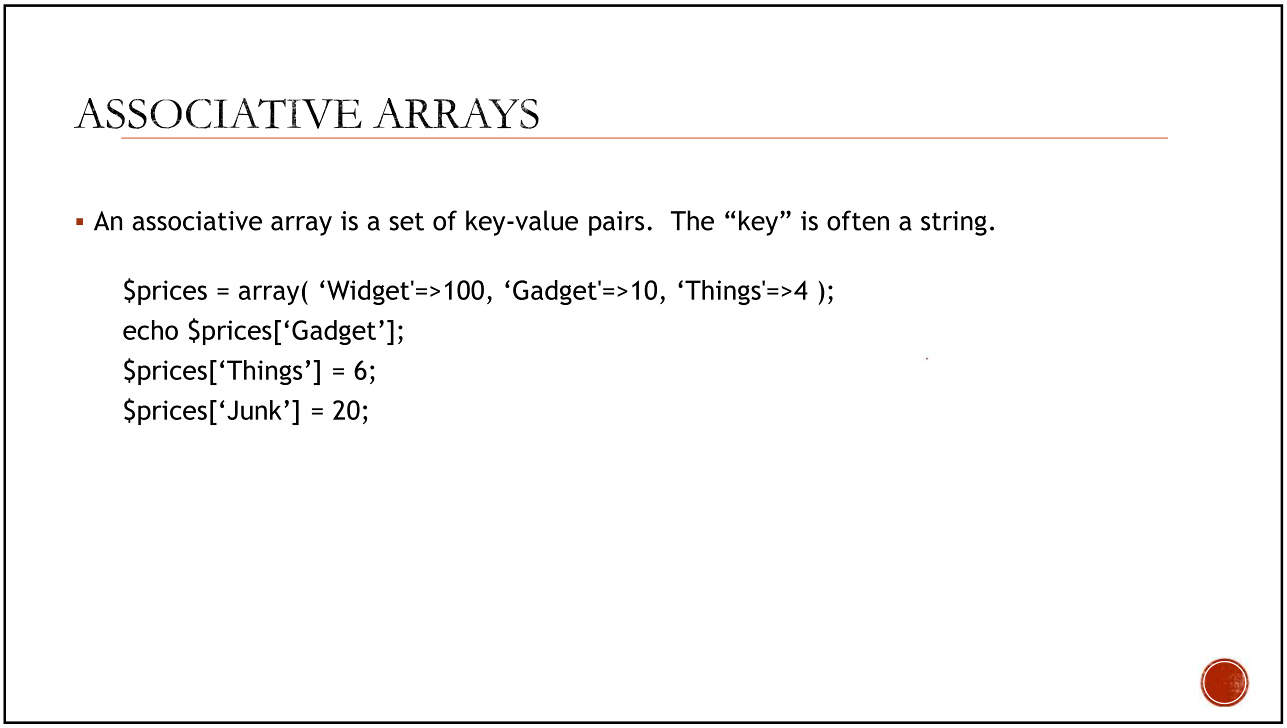


ARRAY ITERATION

- Iterate through an array with **for** or **foreach**

```
$flowers = array(“rose”, “tulip”, “daisy”);


for ($i=0; $i< count($flowers); $i++)
  echo $flowers[$i], “<br />”;


foreach ($flowers as $item)
  echo $item, “<br />”;
```

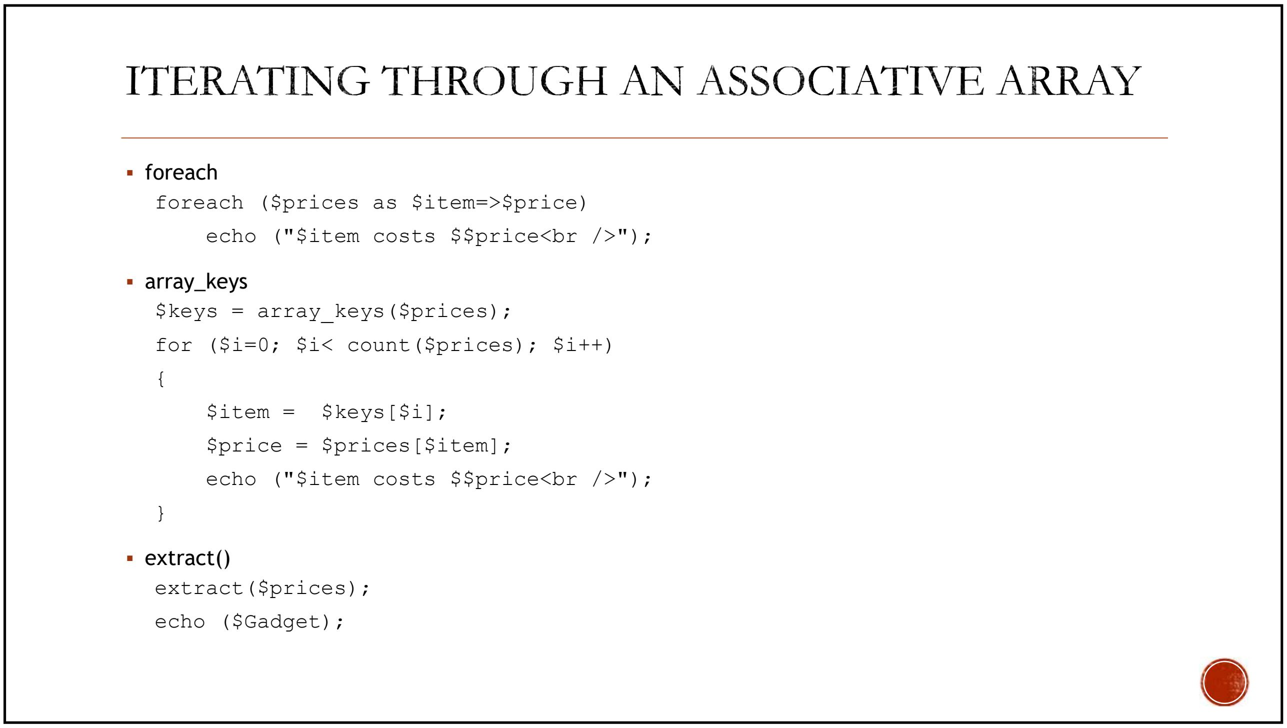
38



ASSOCIATIVE ARRAYS

- An associative array is a set of key-value pairs. The “key” is often a string.

```
$prices = array( ‘Widget'=>100, ‘Gadget'=>10, ‘Things'=>4 );
echo $prices[‘Gadget’];
$prices[‘Things’] = 6;
$prices[‘Junk’] = 20;
```



ITERATING THROUGH AN ASSOCIATIVE ARRAY

- foreach

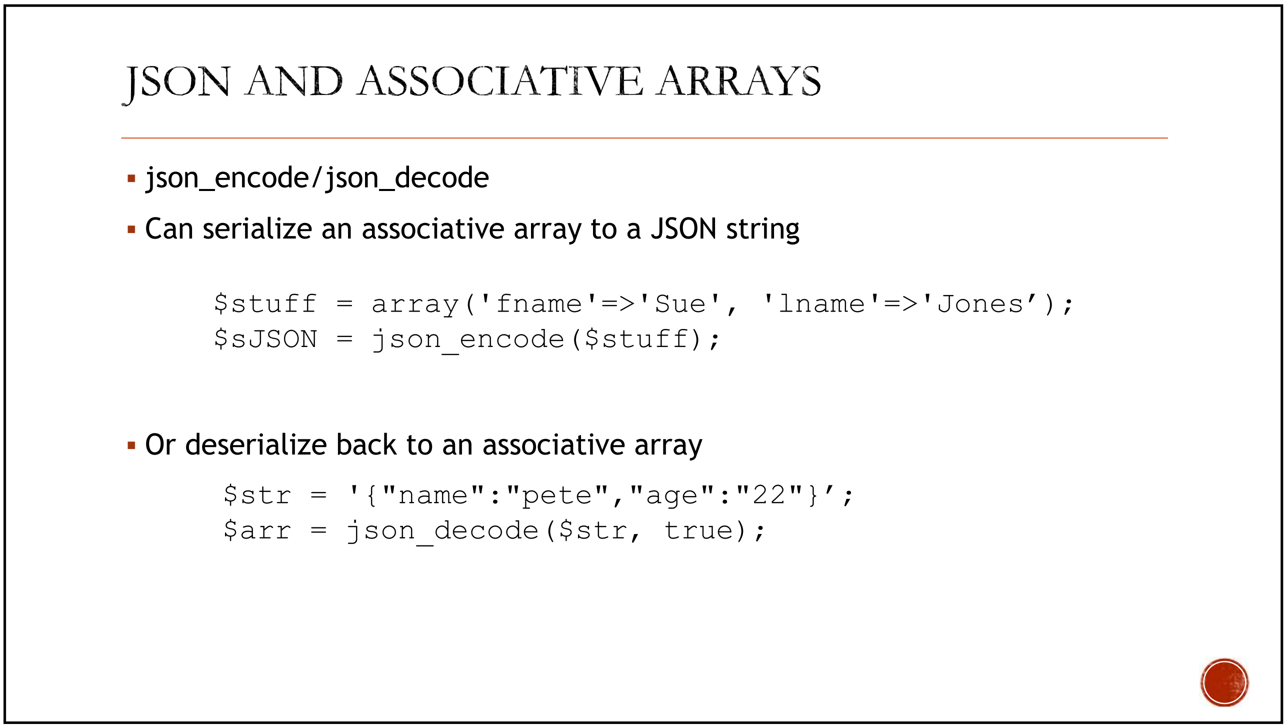
```
foreach ($prices as $item=>$price)
    echo ("$item costs $$price<br />");
```

- array_keys

```
$keys = array_keys($prices);
for ($i=0; $i< count($prices); $i++)
{
    $item =  $keys[$i];
    $price = $prices[$item];
    echo ("$item costs $$price<br />");
}
```

- extract()

```
extract($prices);
echo ($Gadget);
```



JSON AND ASSOCIATIVE ARRAYS

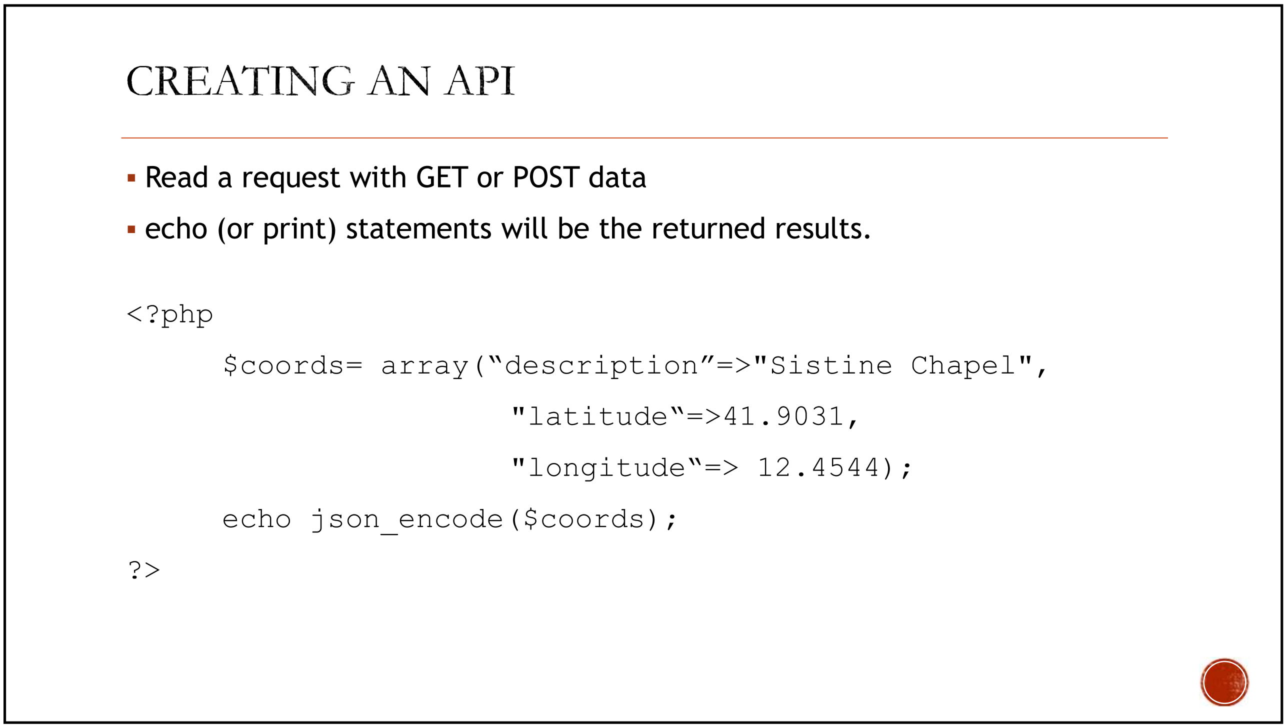
- json_encode/json_decode
- Can serialize an associative array to a JSON string

```
    $stuff = array('fname'=>'Sue', 'lname'=>'Jones’);
    $sJSON = json_encode($stuff);
```

- Or deserialize back to an associative array

```
    $str = '{"name":"pete","age":"22"}’;
    $arr = json_decode($str, true);
```

41

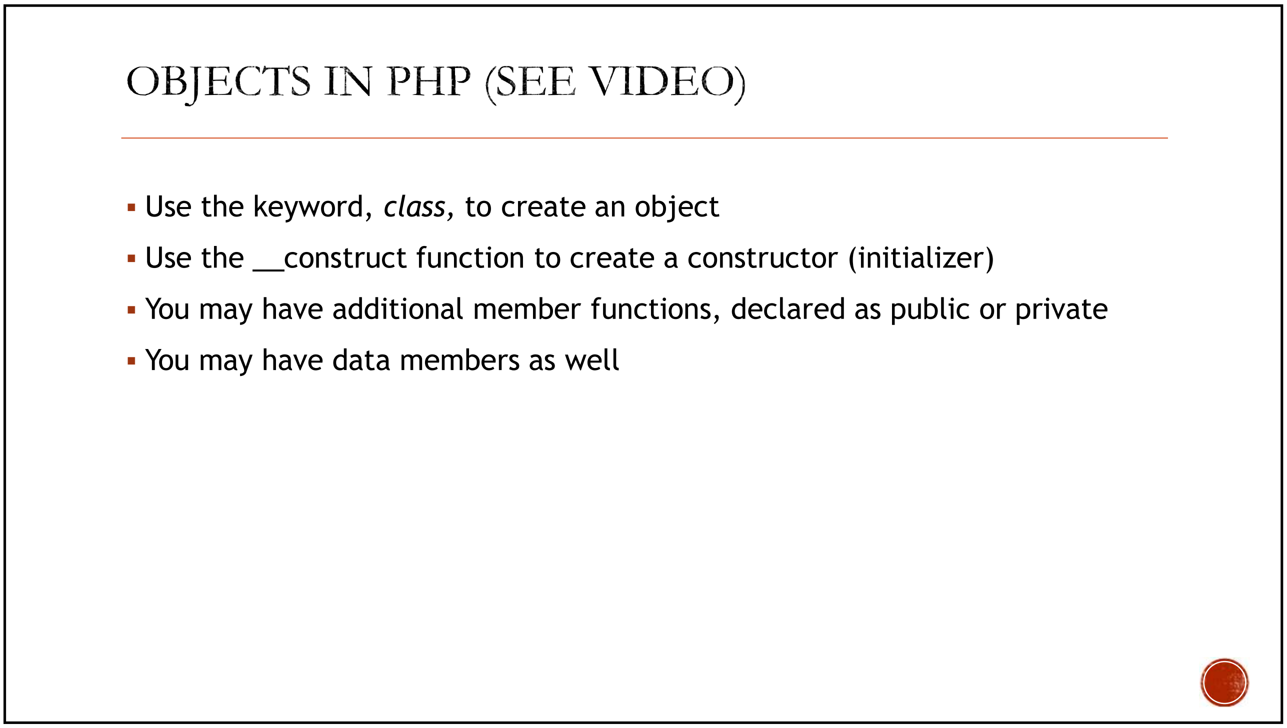


CREATING AN API

- Read a request with GET or POST data
- echo (or print) statements will be the returned results.

```
<?php
      $coords= array(“description”=>"Sistine Chapel",
                      "latitude“=>41.9031,
                      "longitude“=> 12.4544);
      echo json_encode($coords);
?>
```

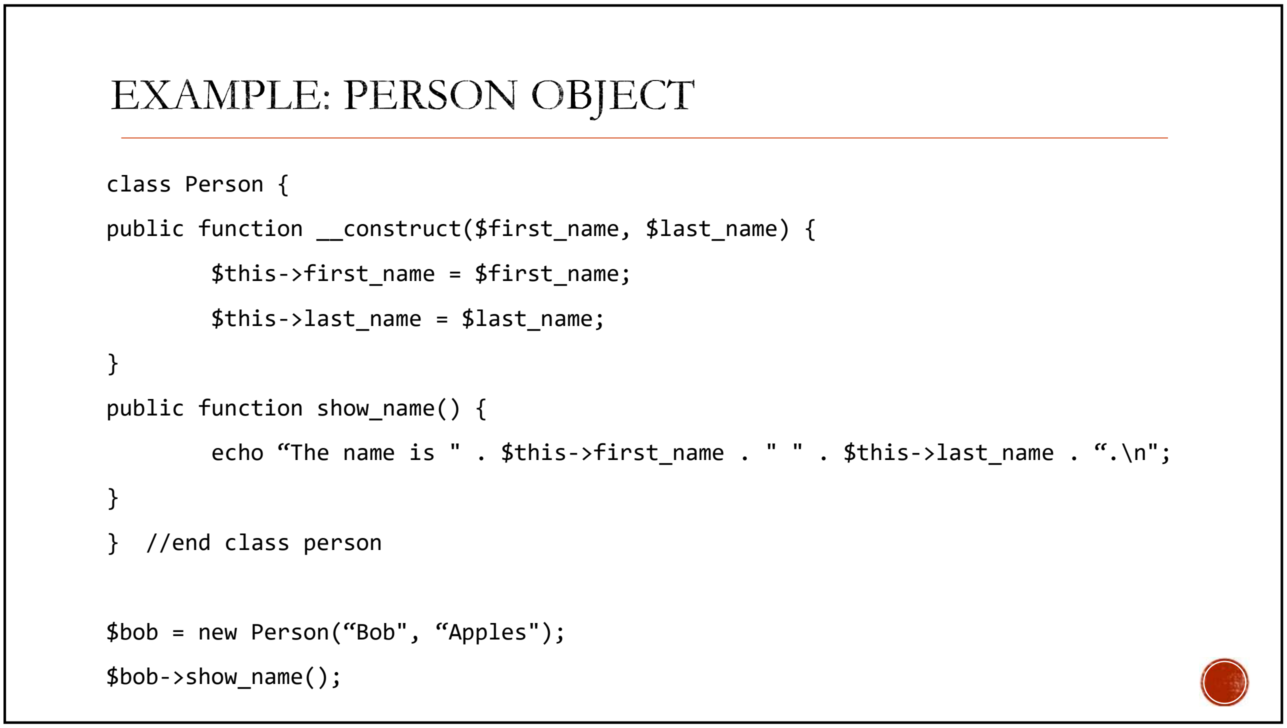
42



OBJECTS IN PHP (SEE VIDEO)

- Use the keyword, *class*, to create an object
- Use the __construct function to create a constructor (initializer)
- You may have additional member functions, declared as public or private
- You may have data members as well

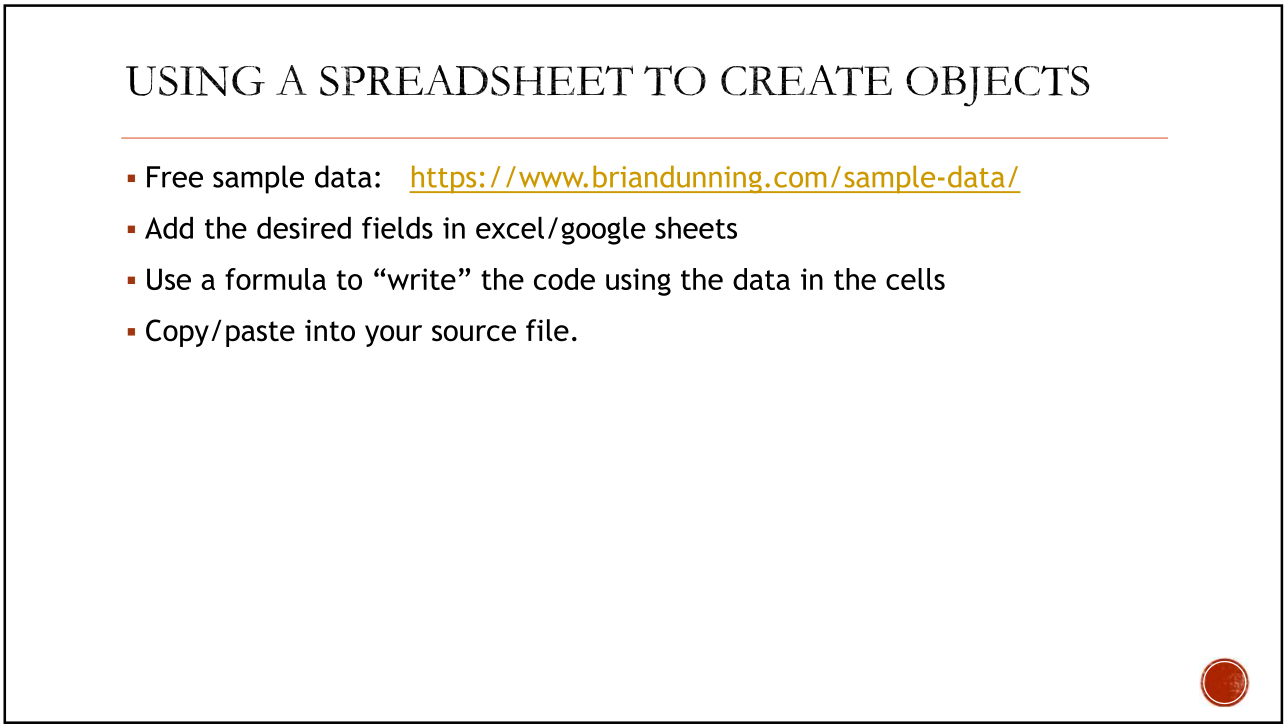
43



EXAMPLE: PERSON OBJECT

```
class Person {
public function __construct($first_name, $last_name) {
        $this->first_name = $first_name;
        $this->last_name = $last_name;
}
public function show_name() {
        echo “The name is " . $this->first_name . " " . $this->last_name . “.\n";
}
}  //end class person

$bob = new Person(“Bob", “Apples");
$bob->show_name();
```

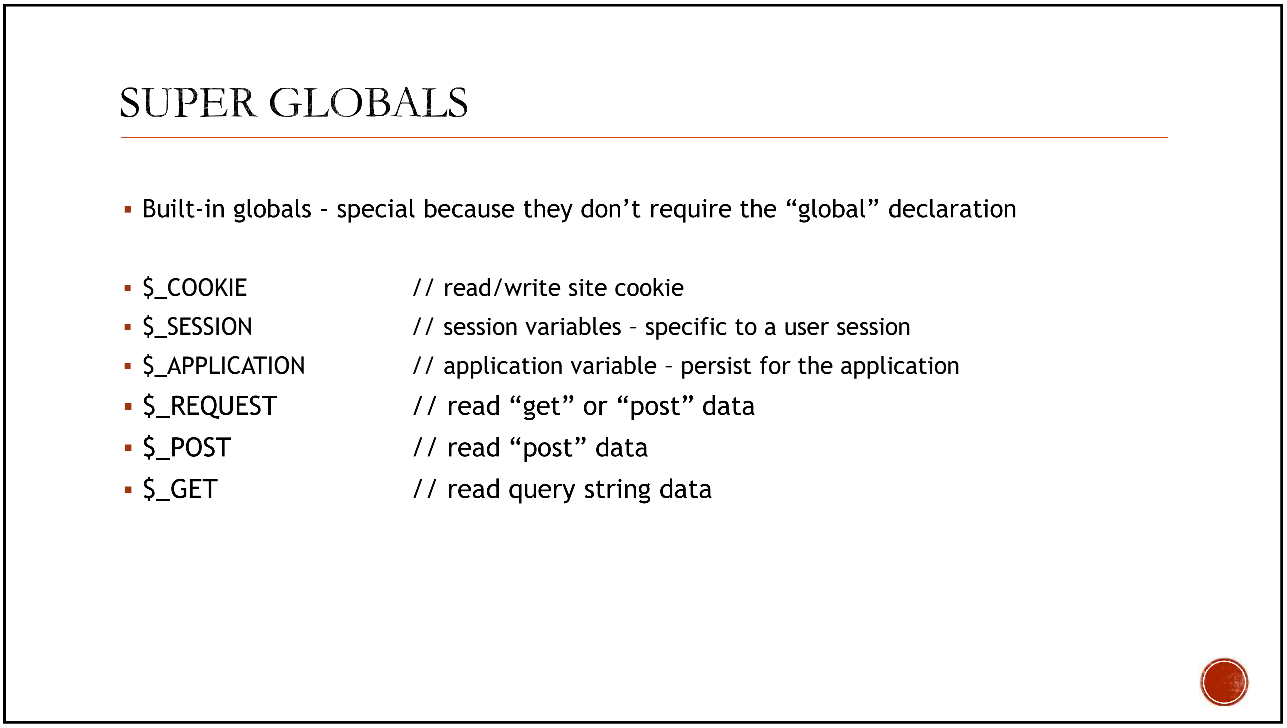
44



USING A SPREADSHEET TO CREATE OBJECTS

- Free sample data: https://www.briandunning.com/sample-data/
- Add the desired fields in excel/google sheets
- Use a formula to "write" the code using the data in the cells
- Copy/paste into your source file.

45

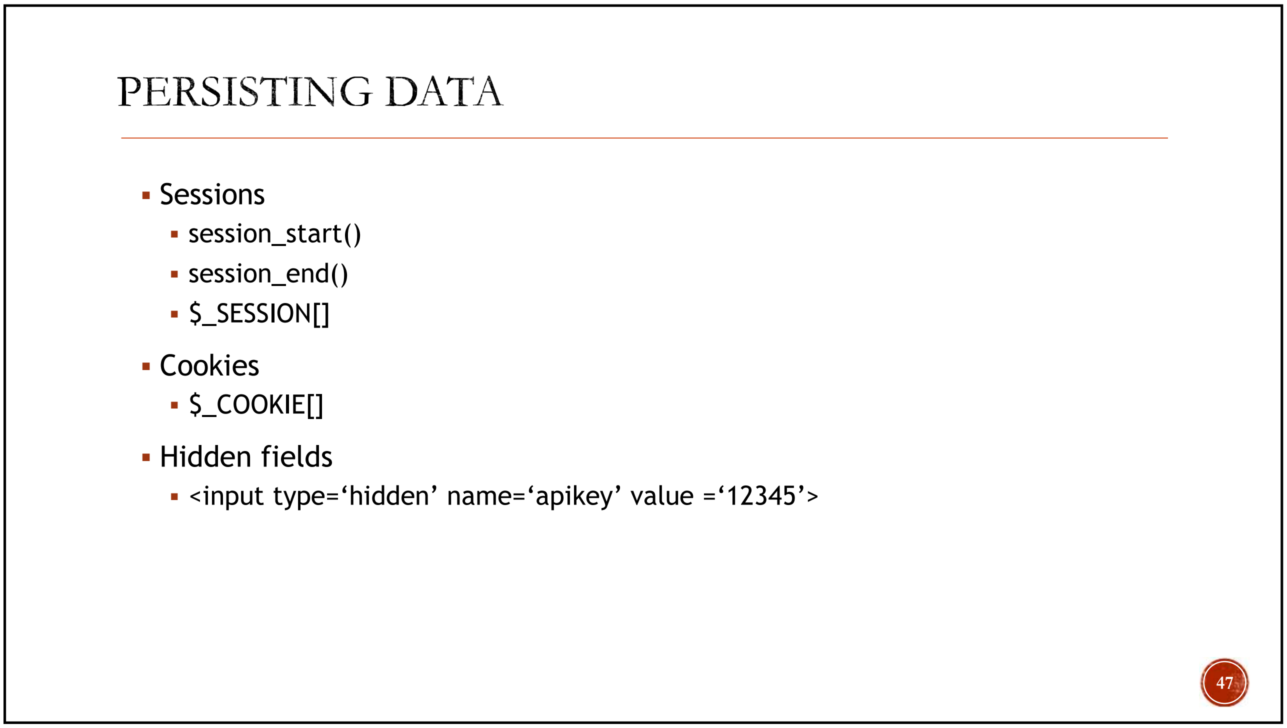


SUPER GLOBALS

- Built-in globals – special because they don't require the "global" declaration

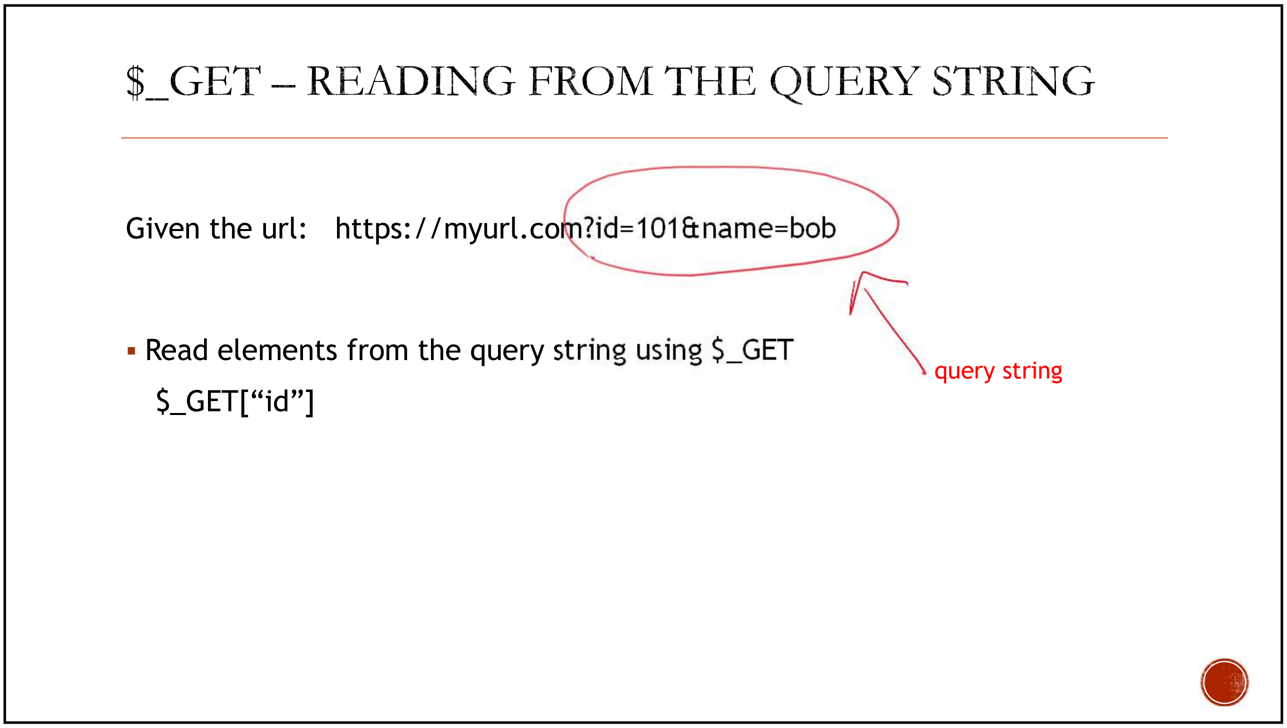
$_COOKIE	// read/write site cookie
$_SESSION	// session variables – specific to a user session
$_APPLICATION	// application variable – persist for the application
$_REQUEST	// read "get" or "post" data
$_POST	// read "post" data
$_GET	// read query string data

46



PERSISTING DATA

- Sessions
  - session_start()
  - session_end()
  - $_SESSION[]
- Cookies
  - $_COOKIE[]
- Hidden fields
  - <input type=‘hidden’ name=‘apikey’ value =‘12345’>



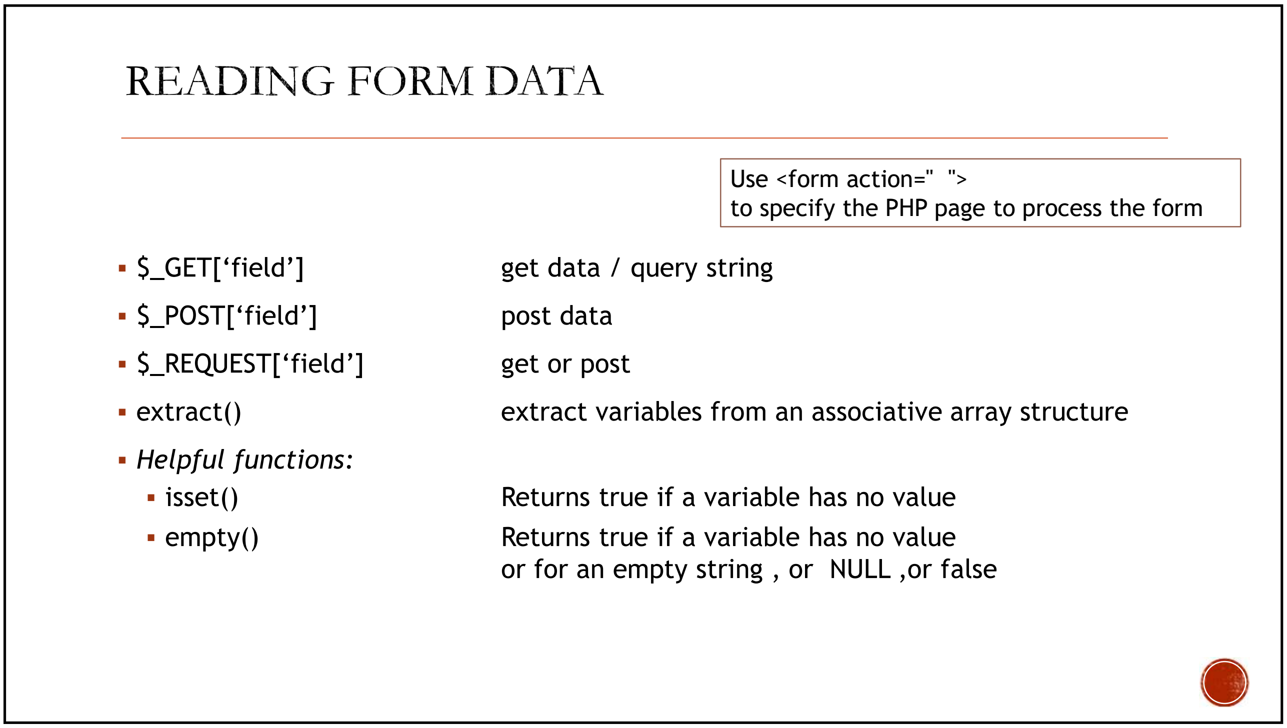
$_GET – READING FROM THE QUERY STRING

Given the url: https://myurl.com?id=101&name=bob

query string

- Read elements from the query string using $_GET

  $_GET[“id”]

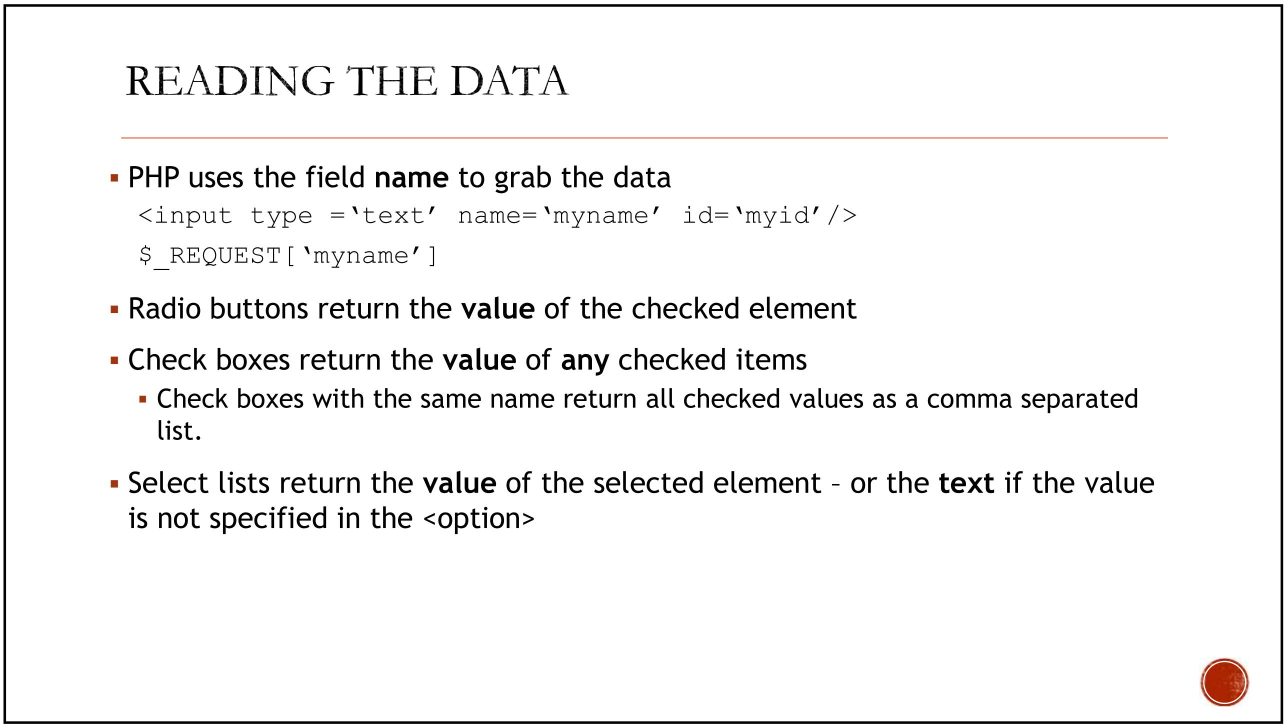


READING FORM DATA

Use <form action=" ">
to specify the PHP page to process the form

▪ $_GET['field']	get data / query string
▪ $_POST['field']	post data
▪ $_REQUEST['field']	get or post
▪ extract()	extract variables from an associative array structure
▪ *Helpful functions*:	
▪ isset()	Returns true if a variable has no value
▪ empty()	Returns true if a variable has no value or for an empty string , or NULL ,or false

49



READING THE DATA

- PHP uses the field **name** to grab the data

```
<input type ='text' name='myname' id='myid'/>
$_REQUEST['myname']
```

- Radio buttons return the **value** of the checked element
- Check boxes return the **value** of **any** checked items
  - Check boxes with the same name return all checked values as a comma separated list.
- Select lists return the **value** of the selected element – or the **text** if the value is not specified in the <option>

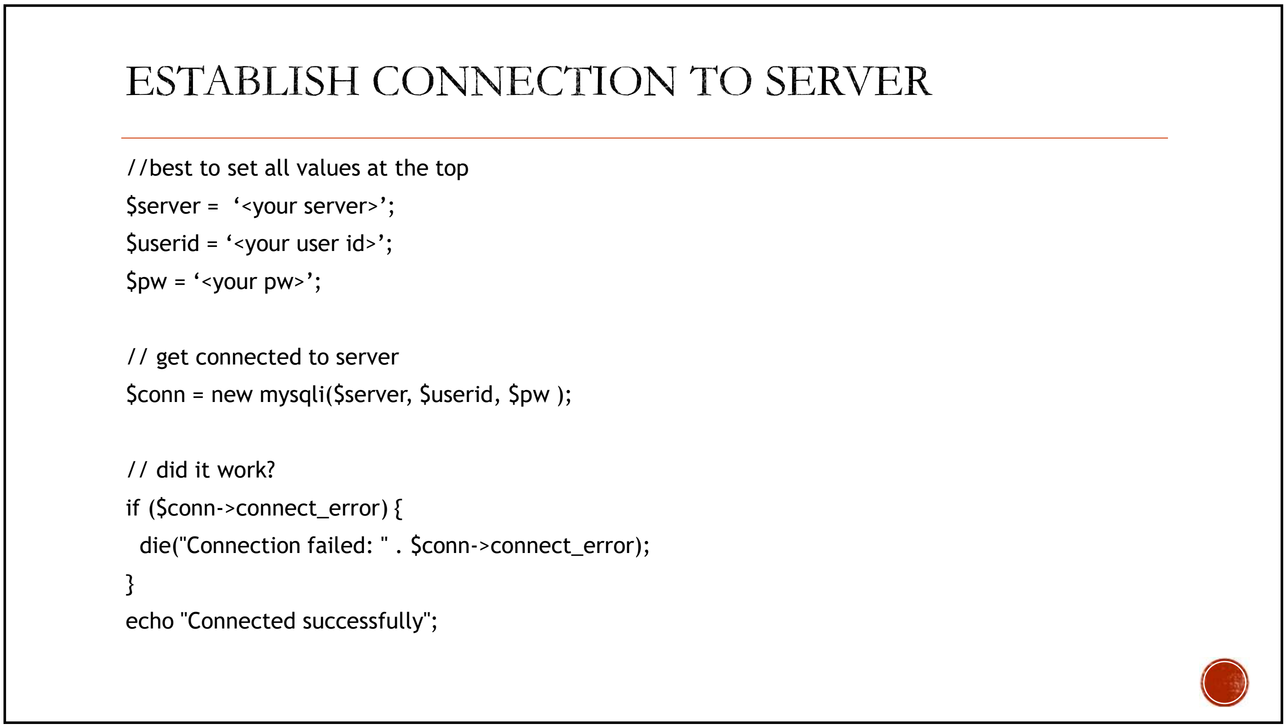
50

BUFFERING

- PHP allows you to control when information is sent to the browser and what goes into the “HTML bucket”
- Buffering allows even more control!
- Output buffering can facilitate creating two pages in one- ex, check login and take one of two paths.
- Output buffering can decrease the amount of time it takes to download and render HTML in the browser.
- Output buffering can resolve errors such as: "Cannot modify header information - headers already sent”

CONNECT TO SQL DATABASE (SEE VIDEO)

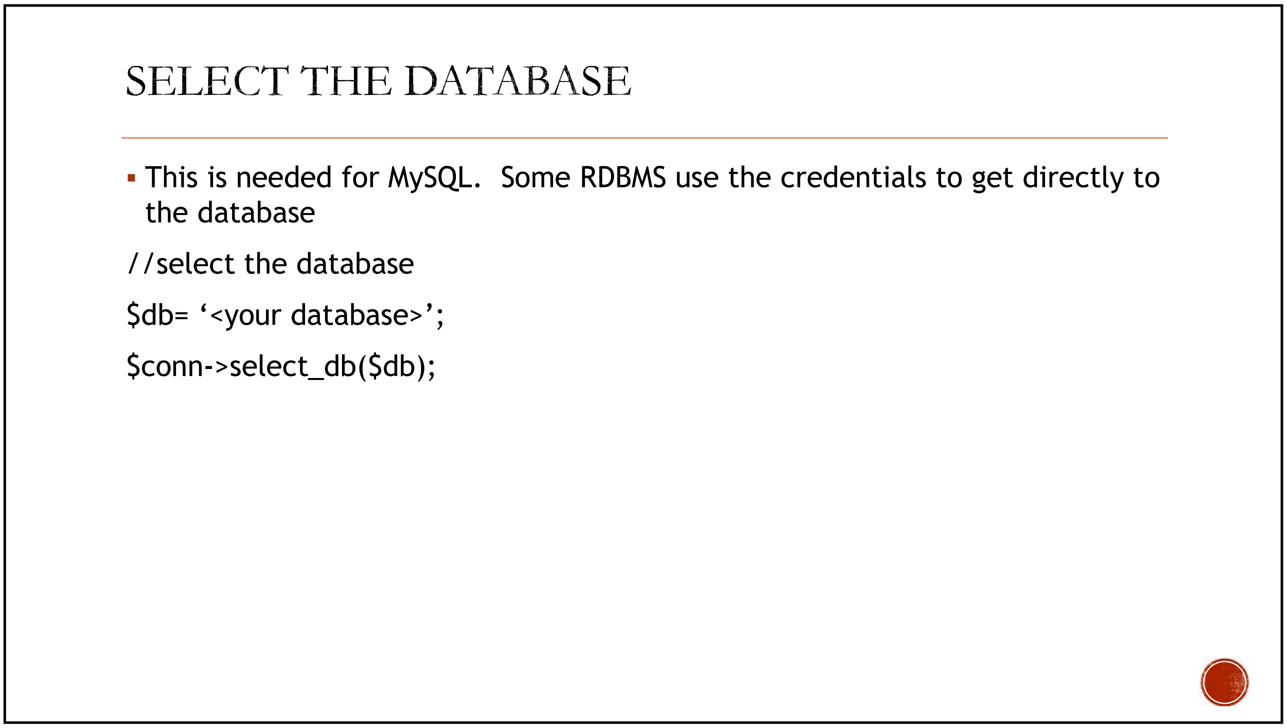
1. Establish connection to server (need connection string)
   - Server
   - Id
   - Password
2. Connect to database
3. Create a query
4. Run the query
5. Get/display results
6. Close the connection



ESTABLISH CONNECTION TO SERVER

```
//best to set all values at the top
$server =  ‘<your server>’;
$userid = ‘<your user id>’;
$pw = ‘<your pw>’;

// get connected to server
$conn = new mysqli($server, $userid, $pw );

// did it work?
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
```

53

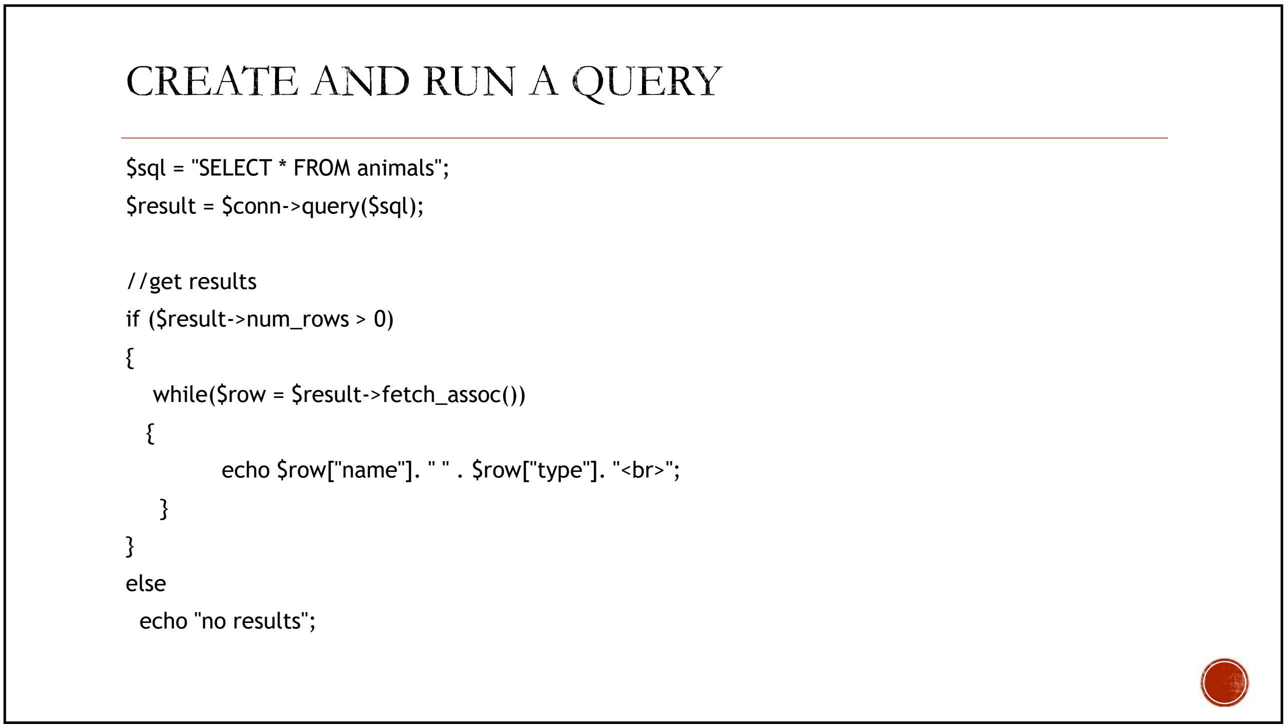


SELECT THE DATABASE

- This is needed for MySQL. Some RDBMS use the credentials to get directly to the database

```
//select the database
$db= ‘<your database>’;
$conn->select_db($db);
```

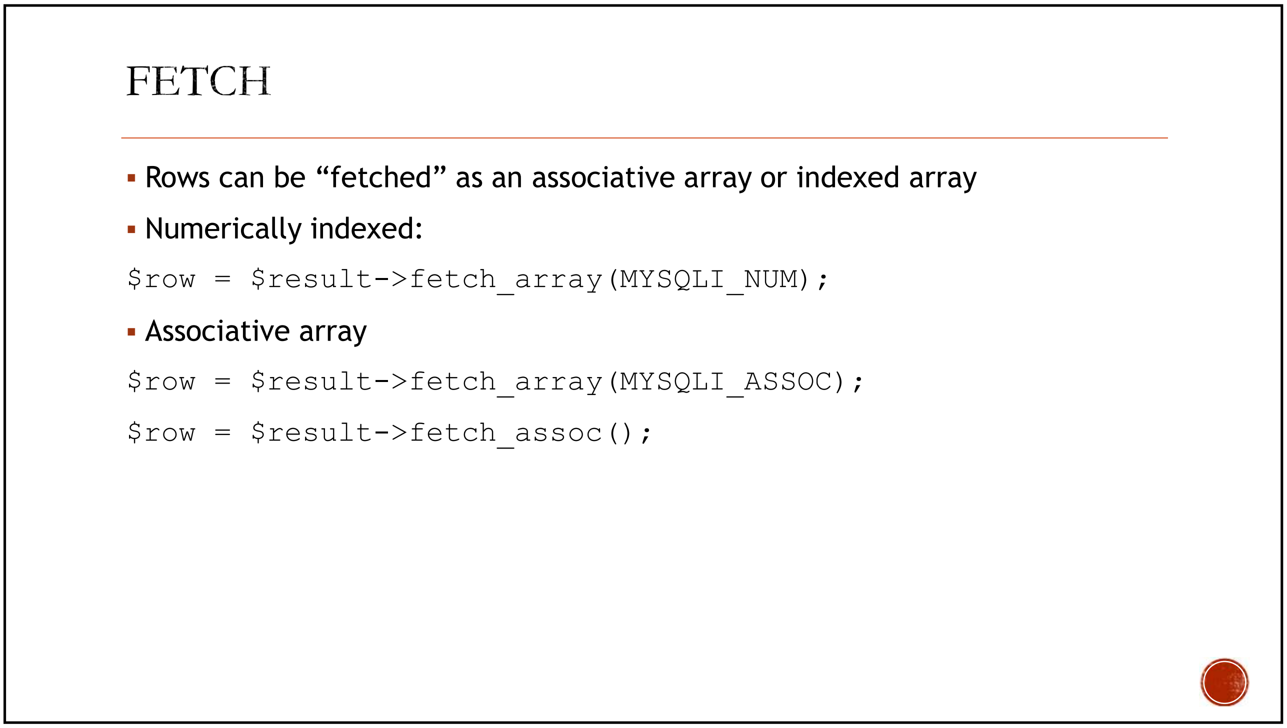
54



CREATE AND RUN A QUERY

```
$sql = "SELECT * FROM animals";
$result = $conn->query($sql);

//get results
if ($result->num_rows > 0)
{
    while($row = $result->fetch_assoc())
   {
              echo $row["name"]. " " . $row["type"]. "<br>";
     }
}
else
  echo "no results";
```

55



FETCH

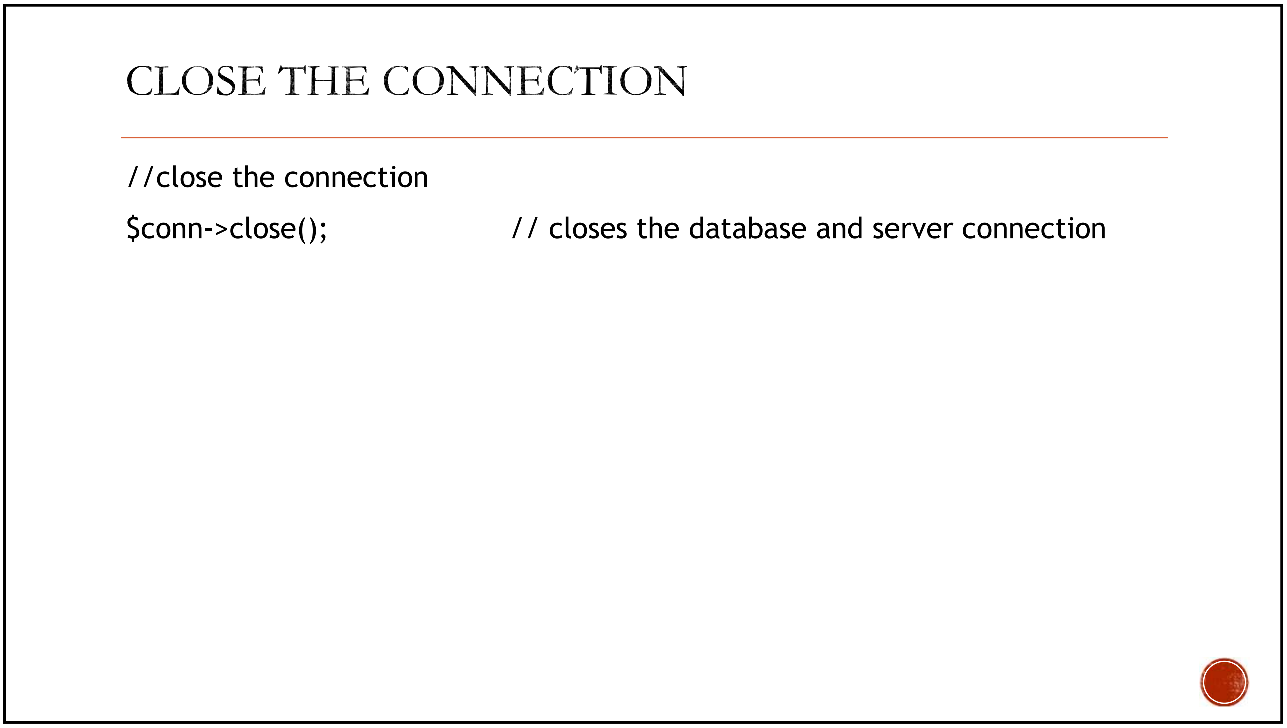
- Rows can be “fetched” as an associative array or indexed array
- Numerically indexed:

```
$row = $result->fetch_array(MYSQLI_NUM);
```

- Associative array

```
$row = $result->fetch_array(MYSQLI_ASSOC);
$row = $result->fetch_assoc();
```

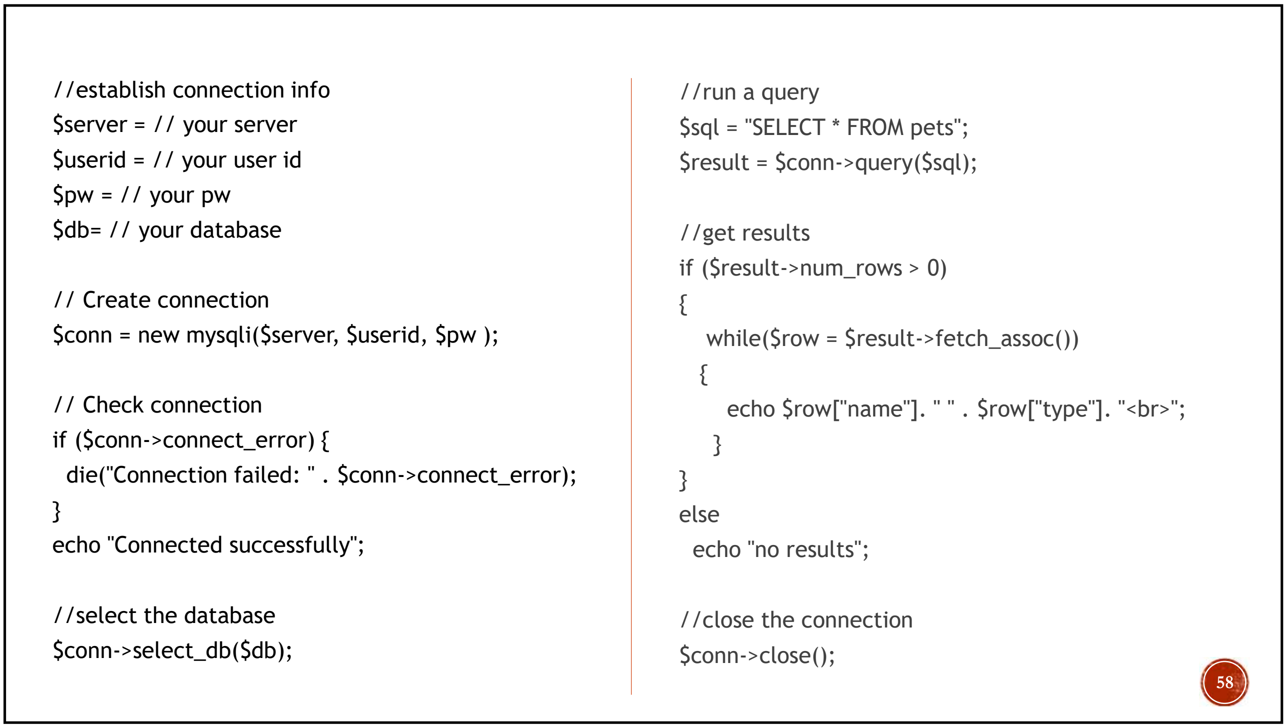
56



CLOSE THE CONNECTION

```
//close the connection
$conn->close();                    // closes the database and server connection
```

57

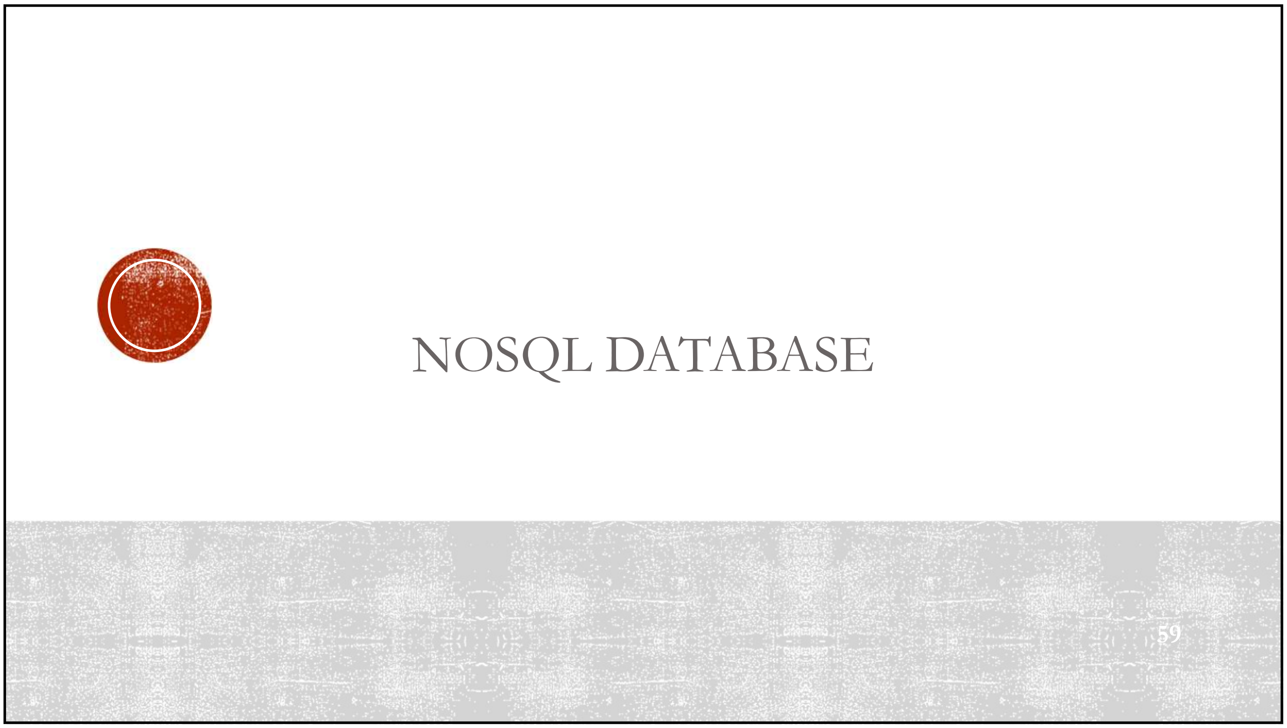


```
//establish connection info
$server = // your server
$userid = // your user id
$pw = // your pw
$db= // your database

// Create connection
$conn = new mysqli($server, $userid, $pw );

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";

//select the database
$conn->select_db($db);
```

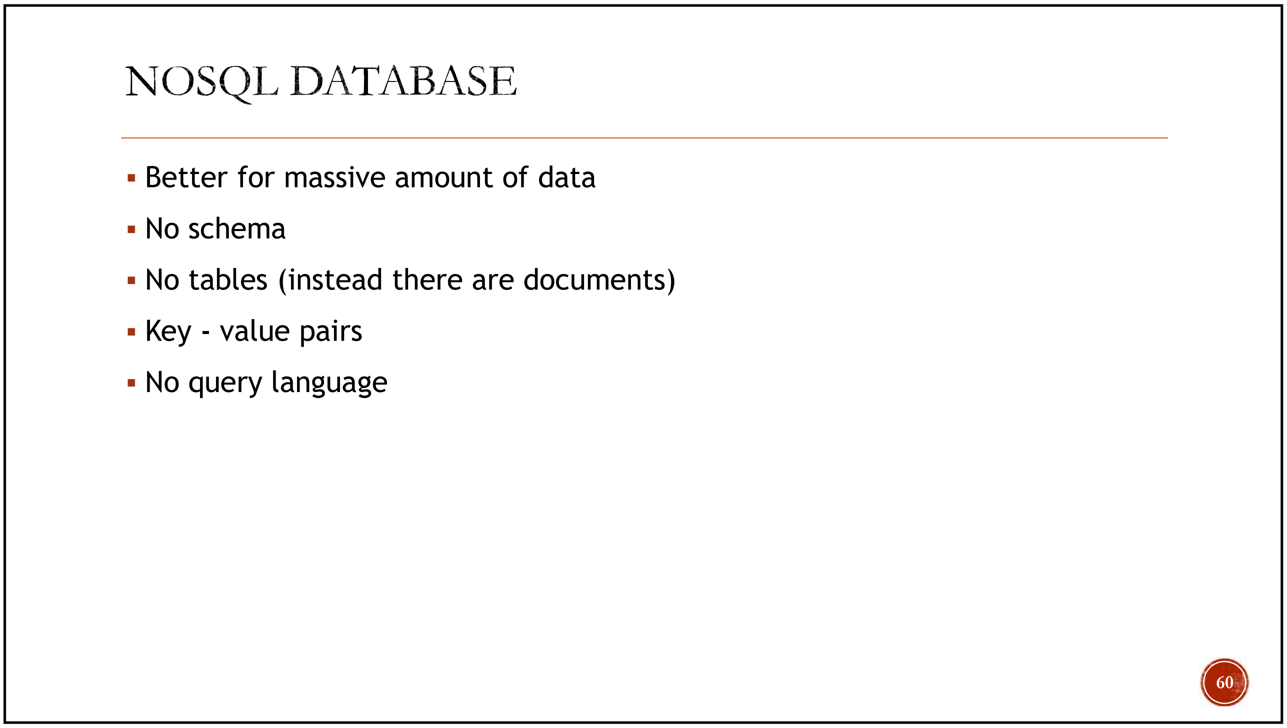
```
//run a query
$sql = "SELECT * FROM pets";
$result = $conn->query($sql);

//get results
if ($result->num_rows > 0)
{
    while($row = $result->fetch_assoc())
   {
       echo $row["name"]. " " . $row["type"]. "<br>";
     }
}
else
  echo "no results";

//close the connection
$conn->close();
```

58

58



NOSQL DATABASE



NOSQL DATABASE

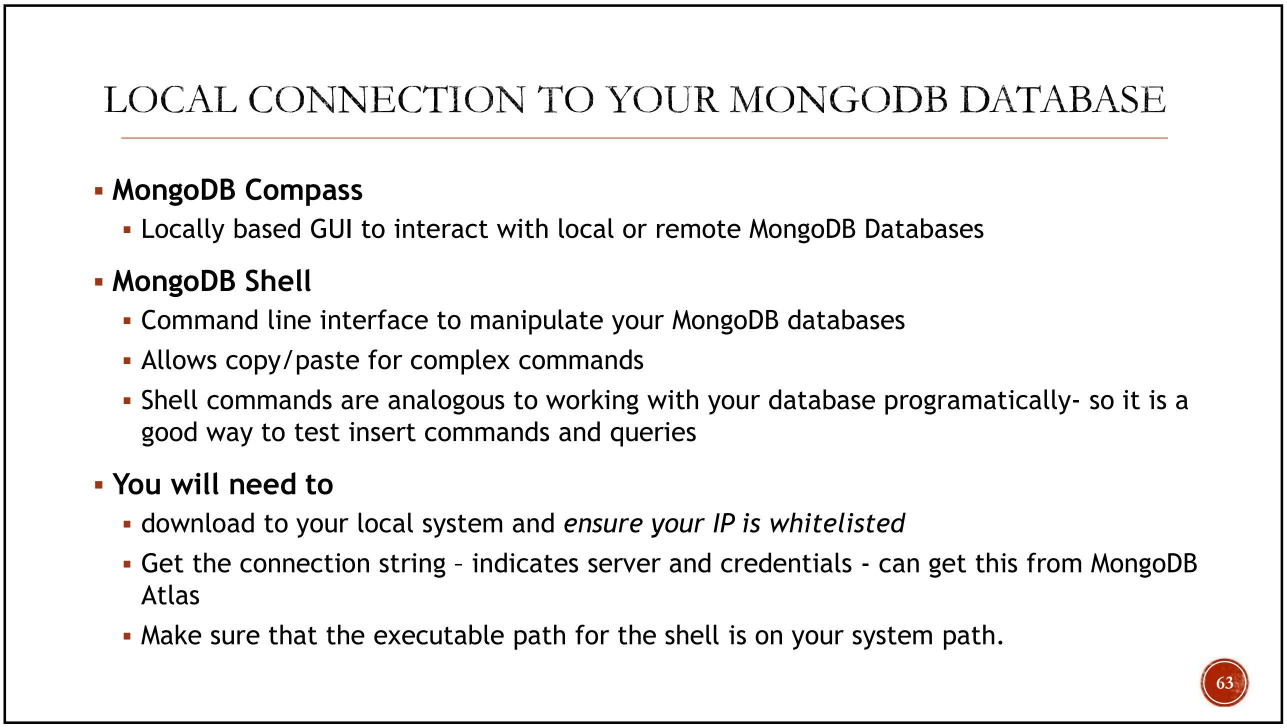
- Better for massive amount of data
- No schema
- No tables (instead there are documents)
- Key - value pairs
- No query language

mongoDB

- Document based
- JSON format
- High performance
- Easily Scalable
- Open source
- Data stored as BSON: Binary encoded JSON documents

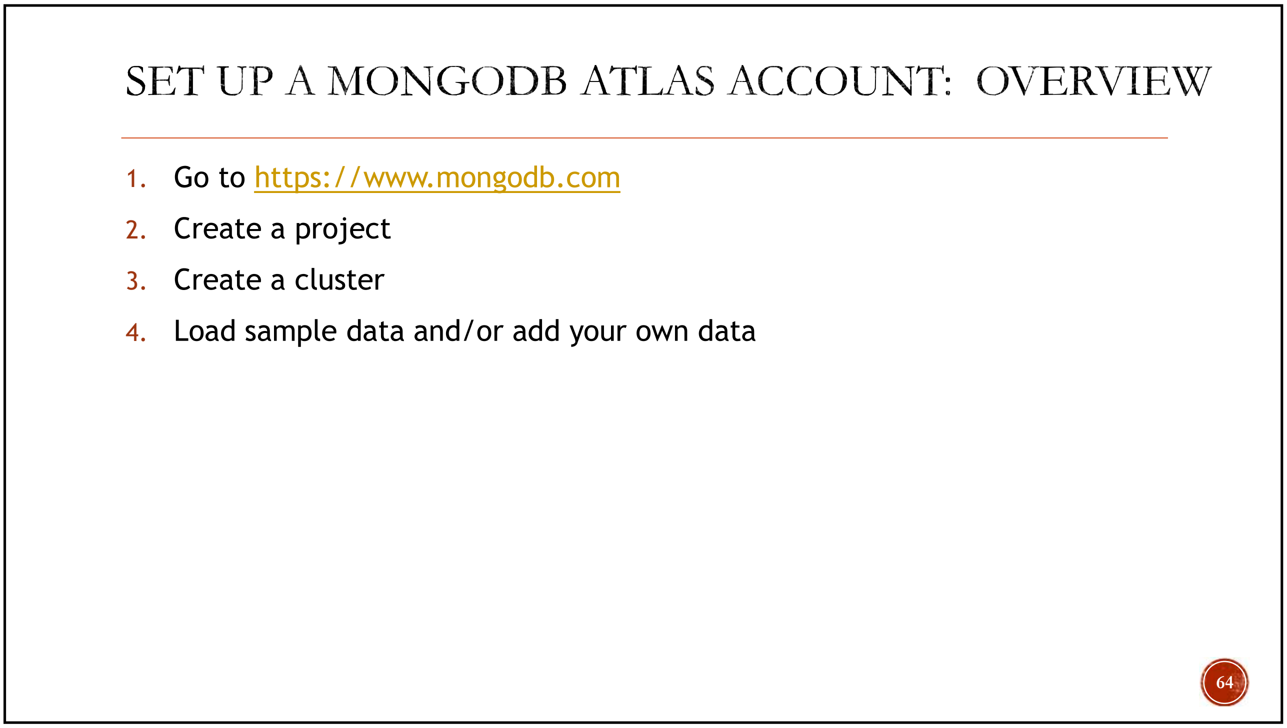
ONLINE DATABASE CONNECTION: ATLAS

- Online environment for hosting MongoDB databases
- Can connect to server-side program (node.js)
- Allows for insert/update/query of data
- Allows for users and data access permissions
- Sample data can be loaded as a sandbox to practice
- There is a free tier you can use for academic projects.



LOCAL CONNECTION TO YOUR MONGODB DATABASE

- **MongoDB Compass**
  - Locally based GUI to interact with local or remote MongoDB Databases
- **MongoDB Shell**
  - Command line interface to manipulate your MongoDB databases
  - Allows copy/paste for complex commands
  - Shell commands are analogous to working with your database programatically- so it is a good way to test insert commands and queries
- **You will need to**
  - download to your local system and *ensure your IP is whitelisted*
  - Get the connection string – indicates server and credentials - can get this from MongoDB Atlas
  - Make sure that the executable path for the shell is on your system path.



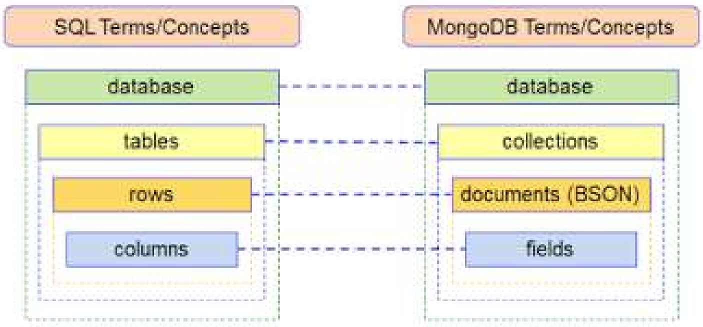
SET UP A MONGODB ATLAS ACCOUNT: OVERVIEW

1. Go to https://www.mongodb.com
2. Create a project
3. Create a cluster
4. Load sample data and/or add your own data

MONGODB TERMS

- **Cluster**
  - A cluster is a unit of storage for the hosted database. Clusters can be shared or dedicated. It is the easy deployment of additional clusters that gives MongoDB Atlas powerful scaling ability.
- **Project**
  - Projects help to segregate teams/security within an organization
  - For the purpose of this course- you are likely to have one cluster and one project.

COLLECTIONS AND DOCUMENTS



SQL Terms/Concepts

MongoDB Terms/Concepts

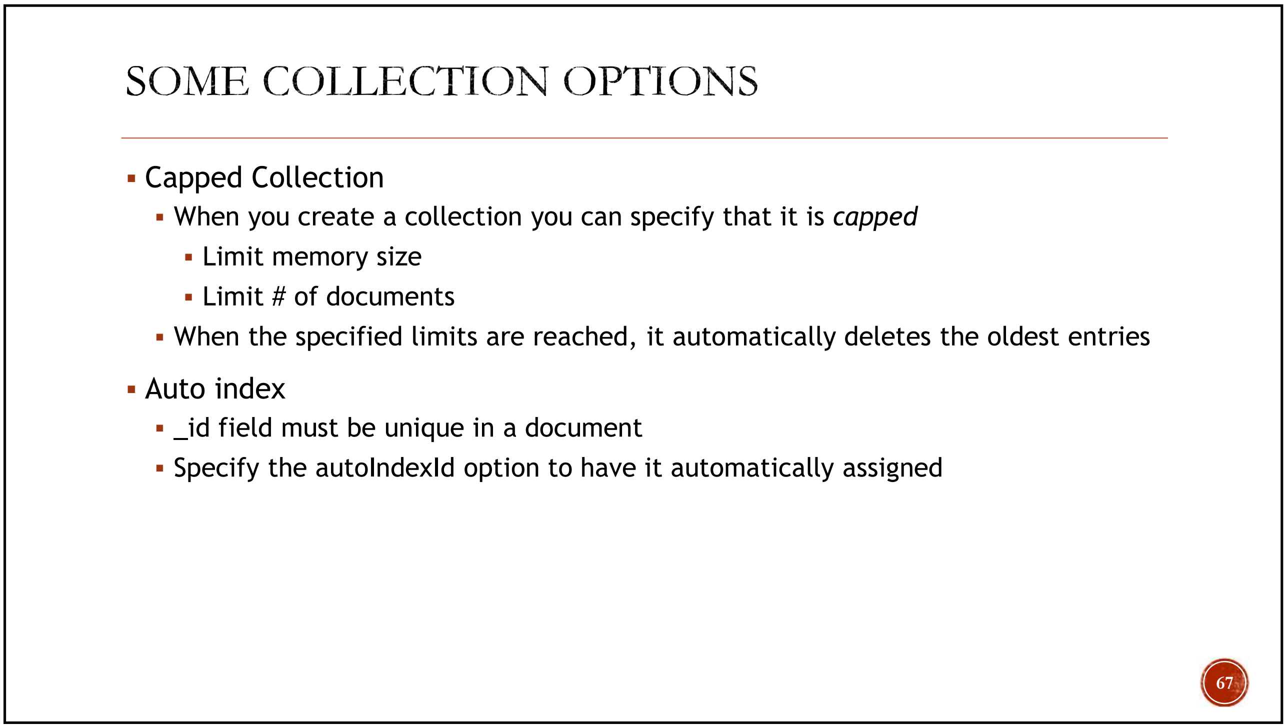
database — database

tables — collections

rows — documents (BSON)

columns — fields

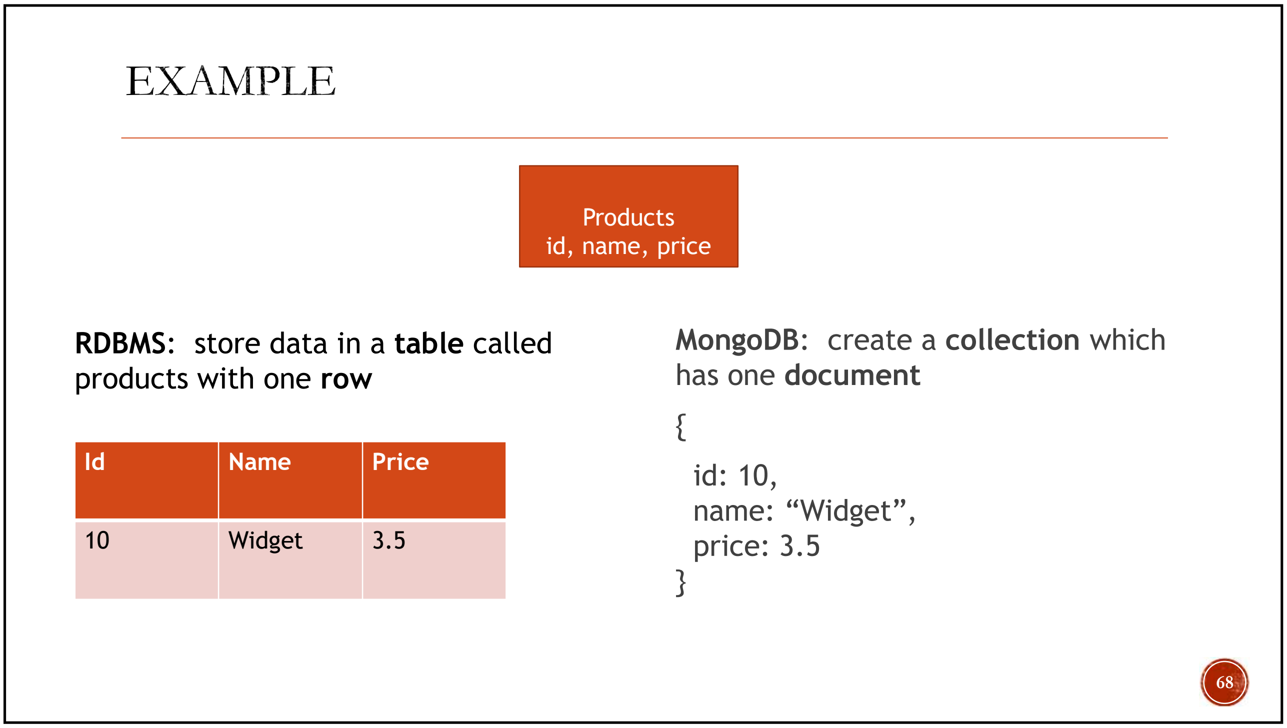
Image credit: https://www.analyticsvidhya.com/blog/2015/06/beginners-guide-mongodb/



SOME COLLECTION OPTIONS

- Capped Collection
  - When you create a collection you can specify that it is *capped*
    - Limit memory size
    - Limit # of documents
  - When the specified limits are reached, it automatically deletes the oldest entries
- Auto index
  - _id field must be unique in a document
  - Specify the autoIndexId option to have it automatically assigned

67



EXAMPLE

Products
id, name, price

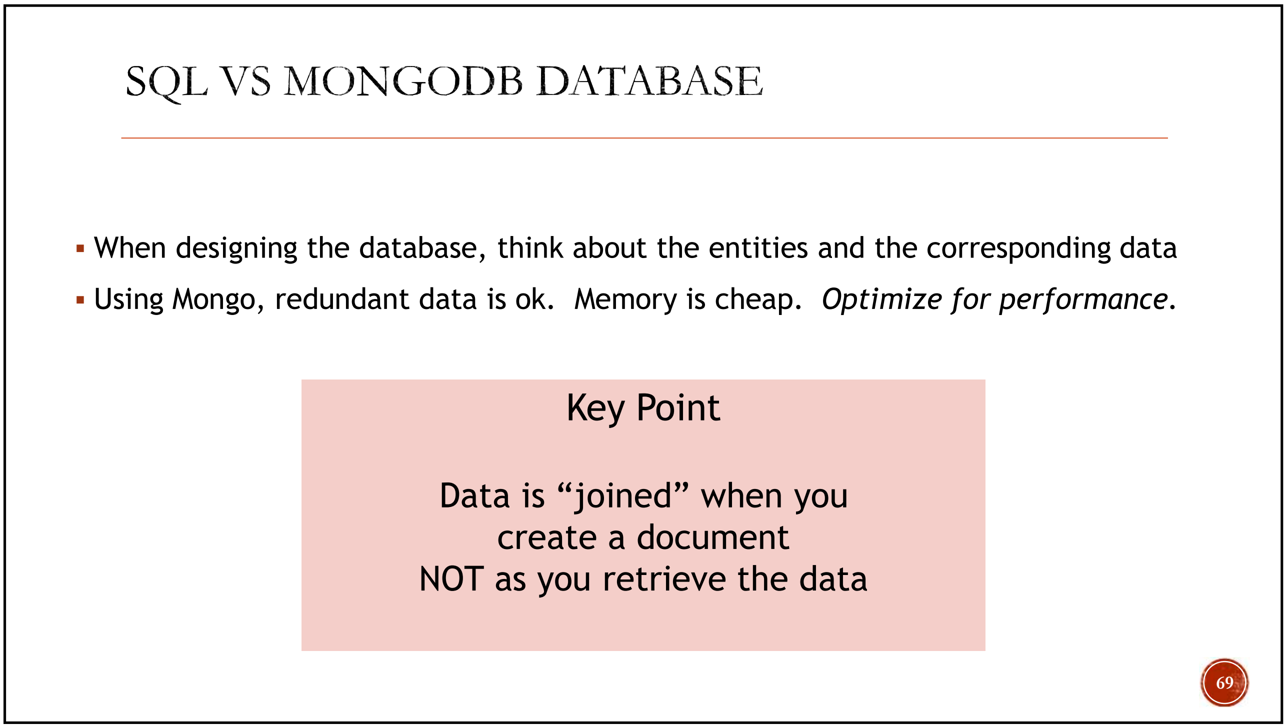
RDBMS: store data in a **table** called products with one **row**

Id	Name	Price
10	Widget	3.5

MongoDB: create a **collection** which has one **document**

```
{
  id: 10,
  name: “Widget”,
  price: 3.5
}
```

68

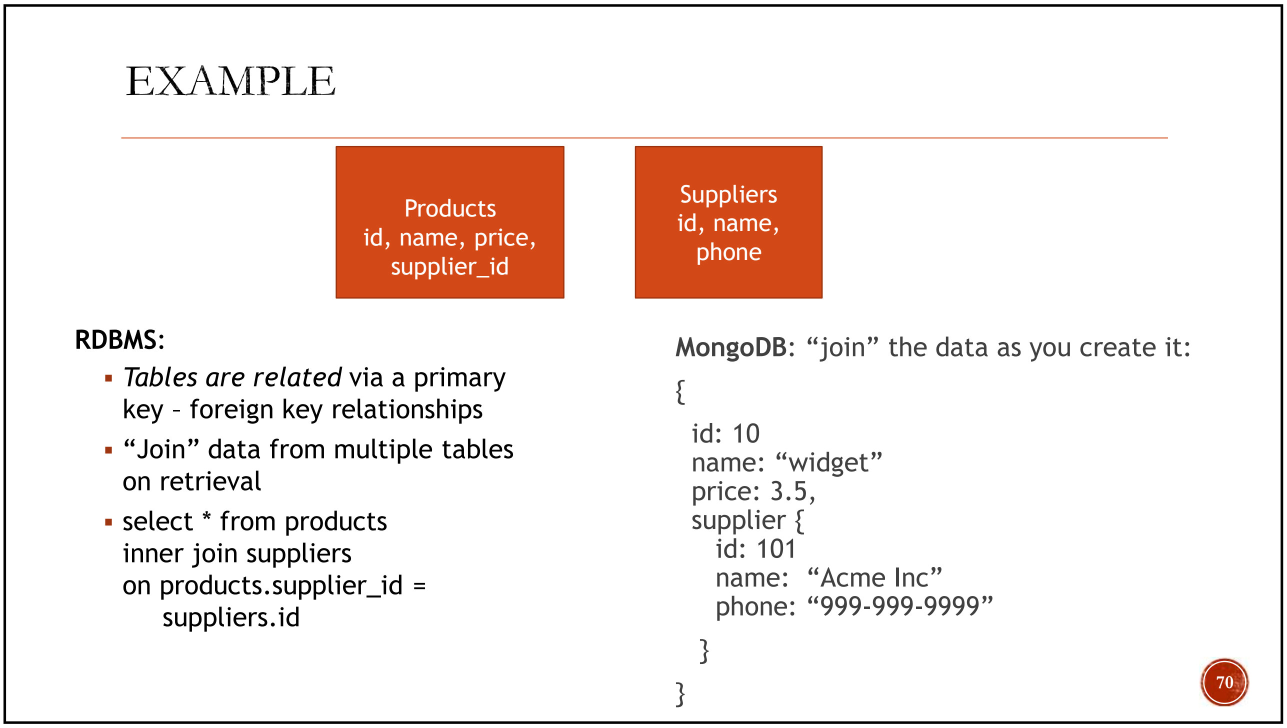


SQL VS MONGODB DATABASE

- When designing the database, think about the entities and the corresponding data
- Using Mongo, redundant data is ok. Memory is cheap. *Optimize for performance.*

Key Point

Data is "joined" when you
create a document
NOT as you retrieve the data



EXAMPLE

Products
id, name, price,
supplier_id

Suppliers
id, name,
phone

RDBMS:

- *Tables are related* via a primary key – foreign key relationships
- "Join" data from multiple tables on retrieval
- select * from products
  inner join suppliers
  on products.supplier_id =
  suppliers.id

MongoDB: "join" the data as you create it:

```
{
  id: 10
  name: "widget"
  price: 3.5,
  supplier {
     id: 101
     name:  "Acme Inc"
     phone: "999-999-9999"
   }
}
```