

# Databases

---

Up to now we've been working entirely in front end materials. Databases allow for us to store information pertinent to our sites. There are many different types of databases. The one that we'll be using is a MySQL database. MySQL is an open source **Relational Database** management system (RDBMS). Relational databases store data in columns and rows in a data table, where each row is a given record. Rows among differing tables can also be linked together based on related columns between tables.

## Installation:

In order to get a MySQL server running on your own machine, depending on your operating system, you can download either **XAMPP**, [a cross platform server application](#) (great for Windows machines), or **MAMP**, [a client for macOS](#). Follow the download/installation instructions.

After installing, open the application and start your server. Make sure that apache and mySql are running.

## MySQL

In order to access our database, we'll need either to access our database via the command line, download software for interfacing with our server (such as [MySQL Workbench](#)), or use the already provided phpMyAdmin. For the sake of not having to download any more software, we'll be using phpMyAdmin.

After you start your xampp or mamp server, a start page should open automatically. If it did not, try going to either localhost:8080, or if you're running mamp, try localhost:8888. To reach phpMyAdmin, you can either follow the links on your server page (top right on xampp or scroll down for mamp) and click on phpMyAdmin, or you can add `/phpMyAdmin` after your localhost with the port number to access phpMyAdmin. You should find yourself on a page like this:

The screenshot shows the phpMyAdmin interface for MySQL. The left sidebar lists databases: information\_schema, mysql, performance\_schema, and sys. The main area is titled 'Databases' and shows a table for creating a new database. A row for 'utf8\_general\_ci' is selected. The table has columns: Database, Collation, and Action. The 'Database' column lists 'information\_schema', 'mysql', 'performance\_schema', and 'sys'. The 'Collation' column lists 'utf8\_general\_ci' for all. The 'Action' column contains 'Check privileges' for each. A note at the bottom says: 'Note: Enabling the database statistics here might cause heavy traffic between the web server and the MySQL server.' Below the note is a link to 'Enable statistics'.

## Creating a Database:

In order to create a database, we need to create a database name and ensure we have the right encoding:

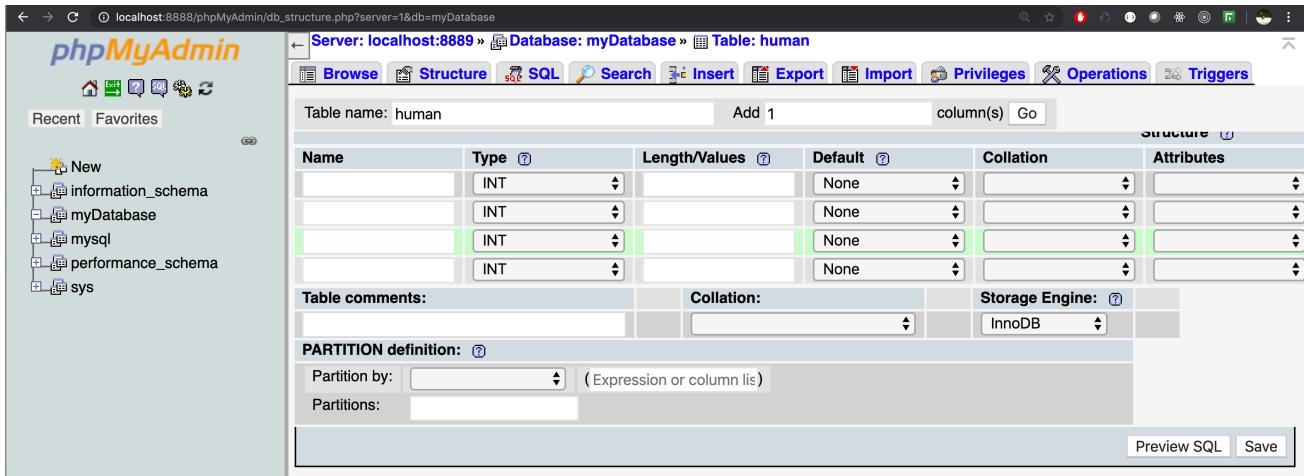
This screenshot is similar to the previous one but with a different database name. The 'Database name' field now contains 'myDatabase'. The rest of the interface, including the table structure and the note about enabling statistics, remains the same.

After hitting the create button, you should find yourself on a page noting that you have no tables. You should also notice your database name on the left hand file structure:

The screenshot shows the 'Structure' tab for the 'myDatabase' database. The left sidebar now includes 'myDatabase' under the 'New' section. The main area displays a message: 'No tables found in database.' Below this is a 'Create table' form with a 'Name:' field containing 'myTable' and a 'Number of columns: 4' field. A 'Go' button is at the bottom right of the form.

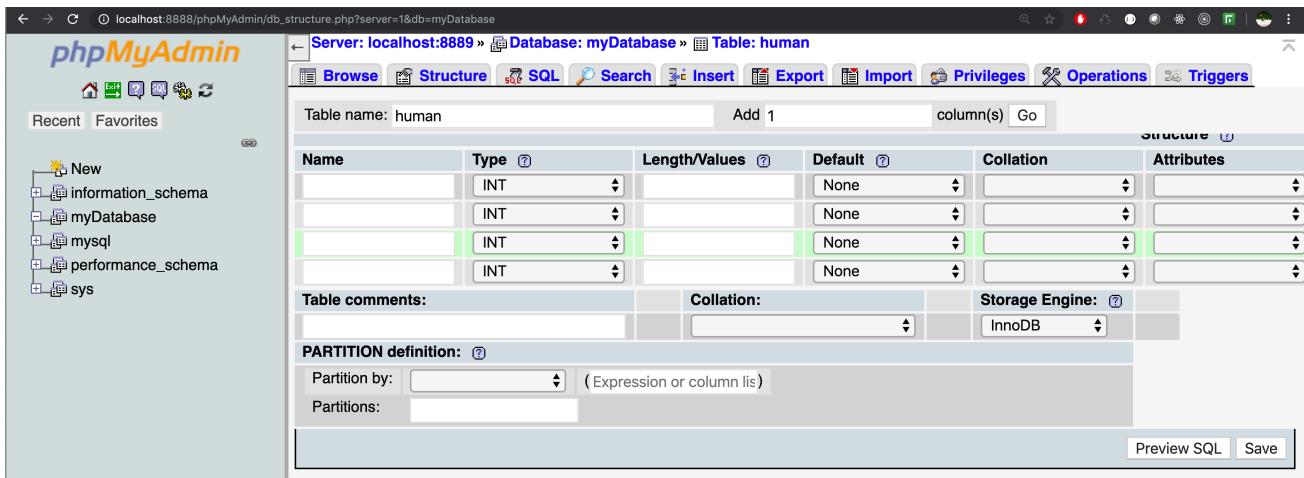
## Creating a Table:

To create a table, you need only to fill in a name for the table as well as the number of columns for the table:



The screenshot shows the phpMyAdmin interface for creating a new table named 'human'. The table has four columns, all defined as INT type. The 'Storage Engine' is set to InnoDB. The 'Collation' dropdown is empty. The 'PARTITION definition:' section is collapsed. At the bottom right, there are 'Preview SQL' and 'Save' buttons.

Once you've created a column, you should find yourself on a new page with containers for each column:



The screenshot shows the phpMyAdmin interface displaying the table 'human' with four columns defined as INT type. The 'Storage Engine' is set to InnoDB. The 'Collation' dropdown is empty. The 'PARTITION definition:' section is collapsed. At the bottom right, there are 'Preview SQL' and 'Save' buttons.

To fill in each column, we need to ensure at the very least two things:

- Column Name
- Datatype

Unlike javascript, MySQL is strongly typed. So choose wisely! A significant amount of possible datatypes are available. We will use INT and TEXT. Let's create a table for humans:

The screenshot shows the 'Structure' tab for the 'human' table. The table structure is defined as follows:

Name	Type	Length/Values	Default	Collation	Attributes
name	TEXT		None		
age	INT		None		
dogName	TEXT		None		
ID	INT		None		

**PARTITION definition:** Partition by: ( Expression or column lis )  
Partitions:

Storage Engine: InnoDB

Note that if you click the `Preview SQL` button in the bottom right, you can then see what SQL code is actually being executed in the background:

```
CREATE TABLE `myDatabase`.`human` ( `name` TEXT NOT NULL , `age` INT NOT NULL ,
`dogName` TEXT NOT NULL , `ID` INT NOT NULL ) ENGINE = InnoDB;
```

**Close**

After clicking save, you'll come to a page that looks something like this:

The screenshot shows the 'Table structure' page for the 'human' table. The table structure is as follows:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	name	text	utf8_general_ci		No	None			
2	age	int(11)			No	None			
3	dogName	text	utf8_general_ci		No	None			
4	ID	int(11)			No	None			

**Indexes**: No index defined!

**Partitions**: No partitioning defined!

**Information**:

Space usage		Row statistics	
Data	16 KiB	Format	dynamic
Index	0 B	Collation	utf8_general_ci
Overhead		Next autoindex	0
Console	16 KiB	Creation	Jul 15, 2019 at 09:36 PM

## Inserting New Data into Tables:

There are a number of ways to insert data. If you click on the `insert` tab, you'll come across this page:

The screenshot shows the phpMyAdmin interface for inserting data into the 'human' table. The left sidebar shows the database structure with 'myDatabase' selected. The main area has two 'Insert' forms. The top form is for the 'human' table, and the bottom form is for a secondary table (likely 'dog'). Both forms have columns: name (text), age (int(11)), dogName (text), and ID (int(11)). Each column has a dropdown for 'Function' and a text input field for 'Value'. A 'Go' button is at the bottom of each form. A 'checkbox' labeled 'Ignore' is checked.

Column	Type	Function	Null	Value
name	text			
age	int(11)			
dogName	text			
ID	int(11)			

Column	Type	Function	Null	Value
name	text			
age	int(11)			
dogName	text			

This page is a GUI wrapper for inserting data to our table. Enter the data desired in the fields, and then click the "GO" button:

The screenshot shows the PHPMyAdmin interface with the following details:

- Server:** localhost:8888
- Database:** myDatabase
- Table:** human
- Structure:** Shows the table structure with columns: name (text), age (int(11)), dogName (text), and ID (int(11)).
- Insert:** A form for inserting a new row. The 'name' field contains 'Shaggy', 'age' contains '25', 'dogName' contains 'Scooby Doo', and 'ID' contains '1'. The 'Ignore' checkbox is checked.
- SQL:** An empty SQL code editor.

After you've inserted a row, you'll be taken back to the SQL page with the SQL Code that was just executed in the SQL code region. Note that the code has already been executed.

The screenshot shows the PHPMyAdmin interface after a successful insert operation:

- Success Message:** A green bar at the top indicates "1 row inserted."
- SQL Output:** The executed SQL code is shown: `INSERT INTO `human` (`name`, `age`, `dogName`, `ID`) VALUES ('Shaggy', '25', 'Scooby Doo', '1');`
- SQL Editor:** The SQL query editor contains the same SQL code: `1 INSERT INTO `human` (`name`, `age`, `dogName`, `ID`) VALUES ('Shaggy', '25', 'Scooby Doo', '1');`. It also shows the table structure with columns: name, age, dogName, and ID.

Try adding a few names of your own!

## Querying the Database: SELECT, INSERT, UPDATE, DELETE

We've already seen a bit of SQL code from what the GUI has formatted for us, but to better understand how SQL queries work, we'll need to actually work with the code!

SQL commands typically follow the syntax of:

```
1 | SELECT something FROM tableName WHERE someTruthCondition
```

Let's take a look. Let's try to select literally everything from our human table:

```
1 | SELECT * FROM `human` WHERE 1;
```

The screenshot shows the phpMyAdmin interface. The left sidebar shows databases: information\_schema, myDatabase (selected), New, mysql, performance\_schema, sys. The top navigation bar shows Server: localhost:8888, Database: myDatabase, Table: human. The main area has tabs: Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, Triggers. The SQL tab is active. The query editor contains: Run SQL query/queries on table myDatabase.human: 1 | SELECT \* FROM `human` WHERE 1. To the right is a results grid with columns: name, age, dogName, ID. Below the query editor are buttons: SELECT\*, SELECT, INSERT, UPDATE, DELETE, Clear, Format, Get auto-saved query, Bind parameters. At the bottom are checkboxes: [ Delimiter ; ], Show this query here again, Retain query box, Rollback when finished, Enable foreign key checks, Go.

What this evaluates to is "Select all columns from the human table". The `WHERE 1` is a boolean value that will return true for every single row (note: the additional names were added, but not included in these notes).

The screenshot shows the phpMyAdmin interface. The left sidebar shows databases: information\_schema, myDatabase (selected), New, mysql, performance\_schema, sys. The top navigation bar shows Server: localhost:8888, Database: myDatabase, Table: human. The main area has tabs: Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, Triggers. The SQL tab is active. The query editor contains: Show query box, Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available. SELECT \* FROM `human` WHERE 1. Below the query editor are checkboxes: Show all, Number of rows: 25, Filter rows: Search this table. The results table has columns: name, age, dogName, ID. Rows: Shaggy (25, Scooby Doo, 1), Some Person (19, Fido, 1), Steve (53, Rex, 2). Below the table are checkboxes: Show all, Number of rows: 25, Filter rows: Search this table.

Suppose we wanted to get values that followed some condition. We can include that in our query:

```
1 | SELECT * FROM `human` WHERE `age` > 26;
```

By adding the ``age` > 26` condition, we're only receiving one row, the Steve row, instead of all three.

The screenshot shows the phpMyAdmin interface for a database named 'myDatabase'. The 'human' table is selected. A query is run: `SELECT * FROM `human` WHERE `age` > 26`. The results show one row: Steve, 53, Rex, 2.

name	age	dogName	ID
Steve	53	Rex	2

You can also query on specific equalities:

```
1 | SELECT * FROM `human` WHERE `name`="Shaggy";
```

The screenshot shows the phpMyAdmin interface for a database named 'myDatabase'. The 'human' table is selected. A query is run: `SELECT * FROM `human` WHERE `name`='Shaggy'`. The results show one row: Shaggy, 25, Scooby Doo, 1.

name	age	dogName	ID
Shaggy	25	Scooby Doo	1

By searching for a specific string, you can narrow down potential answers almost immediately.

The screenshot shows the phpMyAdmin interface for a database named 'myDatabase'. The 'human' table is selected. A query is run: `SELECT * FROM `human` WHERE `name`="Shaggy"`. The results show one row: Shaggy, 25, Scooby Doo, 1.

name	age	dogName	ID
Shaggy	25	Scooby Doo	1

## Inserting Data:

There will most definitely be a time where you'll have to programmatically enter data to your database. In order to do so, you will need to know how to write insert queries. We've inserted data already through the insert tab, but let's take a look at the basic syntax for inserting data via SQL queries:

```

1 | INSERT INTO tableName (columnName1, columnName2, columnName3) VALUES
  | ("dataForColumn1", "dataForColumn2", "dataForColumn3")

```

Let's take a look at a real query. If you click the `SQL` tab, and then click the `insert` button, a sql template for inserting data will appear:

The screenshot shows the phpMyAdmin interface. The left sidebar shows the database structure with a selected table named `human`. The top navigation bar has tabs for `Browse`, `Structure`, `SQL`, `Search`, `Insert`, `Export`, `Import`, `Privileges`, `Operations`, and `Triggers`. The `SQL` tab is active. Below it, the `Run SQL query/queries on table myDatabase.human:` panel contains the following code:

```

1 | INSERT INTO `human`(`name`, `age`, `dogName`, `ID`) VALUES ([value-1],[value-2],
  | [value-3],[value-4])

```

To the right of the code, there is a `Columns` panel listing the table's columns: `name`, `age`, `dogName`, and `ID`. Below the code, there are several buttons: `SELECT *`, `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `Clear`, `Format`, `Get auto-saved query`, `Bind parameters`, and `Go`. At the bottom of the panel are checkboxes for `Delimiter`, `Show this query here again`, `Retain query box`, `Rollback when finished`, `Enable foreign key checks`, and a `Go` button.

Change the `[value-#]`s to actual values corresponding with their columns:

This screenshot is similar to the previous one, but the query in the `Run SQL query/queries on table myDatabase.human:` panel has been modified:

```

1 | INSERT INTO `human`(`name`, `age`, `dogName`, `ID`) VALUES ("Human1", 23, "Doggo",
  | 3)

```

The `Columns` panel remains the same, listing the columns: `name`, `age`, `dogName`, and `ID`.

Adding multiple values, we can get:

The screenshot shows the phpMyAdmin interface with a more complex query in the `Run SQL query/queries on table myDatabase.human:` panel:

```

1 | INSERT INTO `human`(`name`, `age`, `dogName`, `ID`) VALUES
2 | ("Me", 74, "Marly", 4),
3 | ("Caitlin", 42, "Maureen", 5),
4 | ("Kory", 37, "Mimi", 6),
5 | ("Jessica", 32, "Talula", 7)

```

The `Columns` panel lists the columns: `name`, `age`, `dogName`, and `ID`. The bottom of the panel includes the `Delimiter` dropdown, checkboxes for `Show this query here again`, `Retain query box`, `Rollback when finished`, `Enable foreign key checks`, and a `Go` button.

Now that we've inserted data to our table, the table should look something like:

Server: localhost:8889 » Database: myDatabase » Table: human

Browse Structure SQL Search Insert Export Import Privileges Operations Triggers

Show query box

Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.

Showing rows 0 - 7 (8 total, Query took 0.0003 seconds.)

SELECT \* FROM `human` WHERE 1

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all Number of rows: 25 Filter rows: Search this table

+ Options

name	age	dogName	ID
Shaggy	25	Scooby Doo	1
Some Person	19	Fido	1
Steve	53	Rex	2
Human1	23	Doggo	3
Me	74	Marly	4
Caitlin	42	Maureen	5
Kory	37	Mimi	6
Jessica	32	Talula	7

Show all Number of rows: 25 Filter rows: Search this table

Query results operations

Print Copy to clipboard Export Display chart Create view

## Updating Data:

It's not always the case that data will stay the same. Usernames change. Passwords change. Ages change. Many things change!

In order to change data, we need to follow the syntax:

```
1 | UPDATE `tableName` SET `columnName1` = 'newValue' WHERE `condition`
```

So, in an example where we want to change the name "Me" to something more useful, such as a name, like "Matt", we can update our human table, set name to matt, and update that name only where there is a dog named marly:  ?

After executing the above code, we'll see that the column with "Me" and "Marly" the dog has changed to a row with Matt and Marly:

The screenshot shows the phpMyAdmin interface for the 'myDatabase' database. The 'human' table is selected. A message at the top states: "Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available." Below this, a green bar indicates "Showing rows 0 - 7 (8 total, Query took 0.0002 seconds.)". The SQL query shown is: `SELECT * FROM `human` WHERE 1`. The table data is as follows:

name	age	dogName	ID
Shaggy	25	Scooby Doo	1
Some Person	19	Fido	1
Steve	53	Rex	2
Human1	23	Doggo	3
Matt	74	Marly	4
Caitlin	42	Maureen	5
Kory	37	Mimi	6
Jessica	32	Talula	7

One quick note is that if you're attempting to update a row, make absolutely sure you're using the `WHERE` properly. Suppose we forgot to make our column IDs unique when inserting data. Let's see what would happen if we tried to update a row based on a non-unique ID:

The screenshot shows the phpMyAdmin interface with the 'human' table selected. In the SQL tab, the following query is entered: `UPDATE `human` SET `name` = "Fred" WHERE `ID` = 1`. To the right, a 'Columns' panel lists the columns: name, age, dogName, and ID. The 'Bind parameters' checkbox is checked. At the bottom, the 'Simulate query' button is highlighted.

Because we did not make our IDs unique, we inadvertently had 2 of the same IDs:

The screenshot shows the phpMyAdmin interface with the 'human' table selected. A message at the top states: "Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available." Below this, a green bar indicates "Showing rows 0 - 7 (8 total, Query took 0.0003 seconds.)". The SQL query shown is: `SELECT * FROM `human` WHERE 1`. The table data is as follows:

name	age	dogName	ID
Fred	25	Scooby Doo	1
Fred	19	Fido	1
Steve	53	Rex	2
Human1	23	Doggo	3
Matt	74	Marly	4
Caitlin	42	Maureen	5
Kory	37	Mimi	6
Jessica	32	Talula	7

So, let's go and change our name and ID:

The screenshot shows the phpMyAdmin interface for a database named 'myDatabase'. In the left sidebar, the 'human' table is selected under the 'myDatabase' database. In the main area, the 'SQL' tab is active, displaying the following SQL query:

```
1 UPDATE `human` SET `name`="NOT FRED",`ID`=8 WHERE `name`="Fred" AND `ID`=1 AND `dogName`="Scooby Doo"
```

Below the query, there are several buttons: SELECT\*, SELECT, INSERT, UPDATE, DELETE, Clear, Format, Get auto-saved query, Bind parameters, Show this query here again, Retain query box, Rollback when finished, Enable foreign key checks, Simulate query, and Go.

So, now we have NOT FRED with an ID of 8 :

The screenshot shows the phpMyAdmin interface for the 'human' table. The table structure is displayed with columns: name, age, dogName, and ID. The data grid shows the following rows:

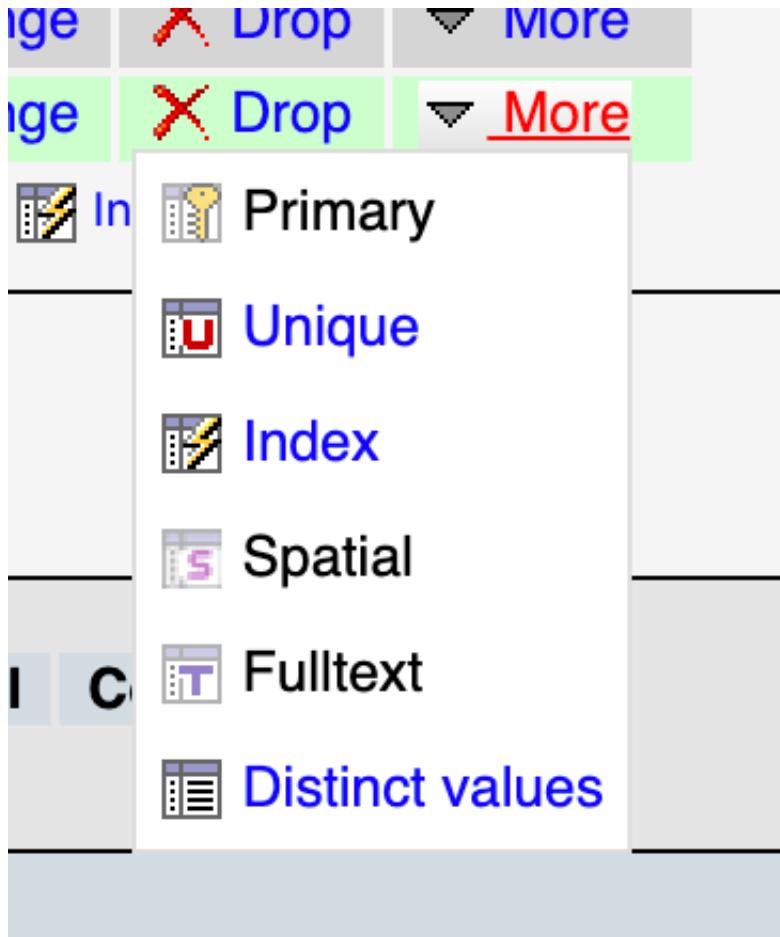
name	age	dogName	ID
NOT FRED	25	Scooby Doo	8
Fred	19	Fido	1
Steve	53	Rex	2
Human1	23	Doggo	3
Matt	74	Marly	4
Caitlin	42	Maureen	5
Kory	37	Mimi	6
Jessica	32	Talula	7

## Having a Primary Key / Unique ID:

Before we go any further, we need to set a primary key so that we don't wind up accidentally overwriting other rows.

The screenshot shows the phpMyAdmin interface for the 'human' table. In the 'Structure' tab, the table structure is listed with four columns: #, Name, Type, Collation, Attributes, Null, Default, Comments, Extra, and Action. The 'ID' column is highlighted with a red box. The 'Action' column for the 'ID' row contains the 'Primary' button, which is also highlighted with a red box. A red arrow points from the text below to this 'Primary' button.

Click onto the structure tab, and then click the more section of the ID row, and select Primary:



By selecting a primary key, we establish that there will be entirely unique IDs for each and every row.

So, now if we try to add someone to our table with the same ID as an already existing user, we'll get an error:

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** localhost:8889
- Database:** myDatabase
- Table:** human
- SQL Query:**

```
1 INSERT INTO `human`(`name`, `age`, `dogName`, `ID`) VALUES ("Velma", 123, "Scrappy Doo", 1)
```
- Columns:** name, age, dogName, ID
- Error Message:**

**Error**

**SQL query:**

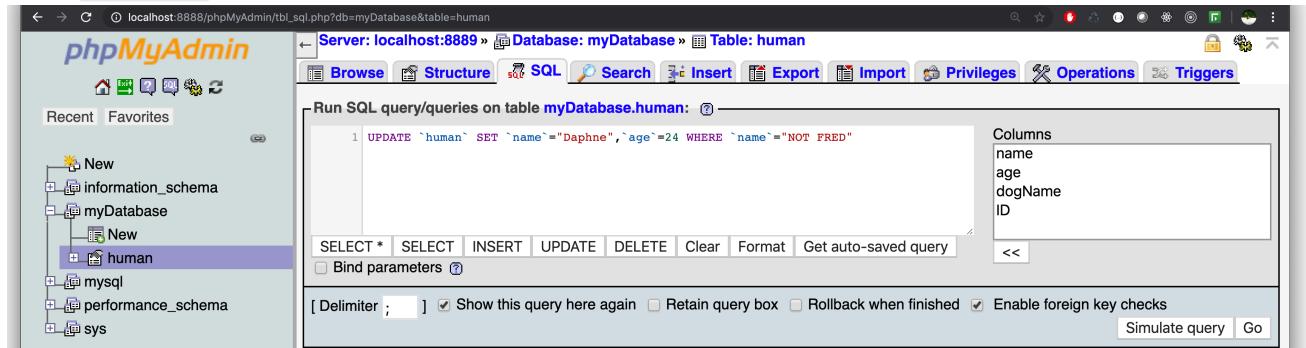
```
INSERT INTO `human`(`name`, `age`, `dogName`, `ID`) VALUES ("Velma", 123, "Scrappy Doo", 1)
```

**MySQL said:**

```
#1062 - Duplicate entry '1' for key 'PRIMARY'
```

## Updating multiple column values:

Back to updating, it's not always the case that you'll want to update only a single column in a row. It's most likely the case that you'll want to update multiple columns at a given point in time. Now that we have `NOT FRED`, let's try updating everything about that row:

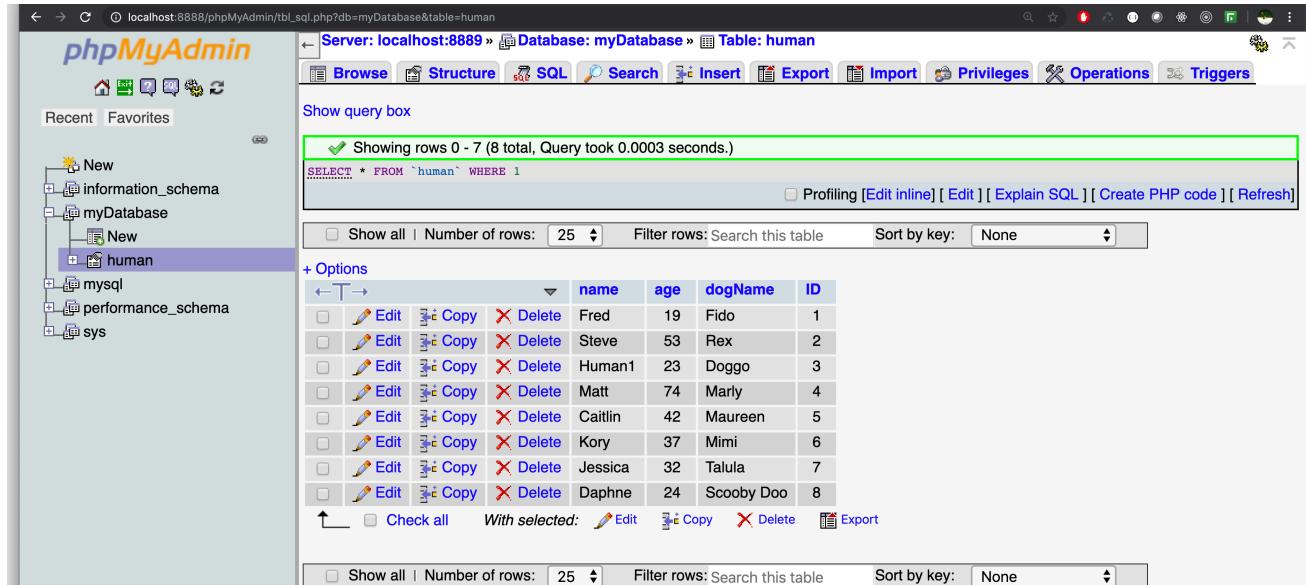


The screenshot shows the phpMyAdmin interface for a database named 'myDatabase'. The 'Structure' tab is selected. In the SQL tab, there is a query window containing the following SQL code:

```
1 UPDATE `human` SET `name`="Daphne", `age`=24 WHERE `name`="NOT FRED"
```

Below the query window, the 'Columns' section lists the columns: name, age, dogName, and ID. At the bottom of the SQL tab, there are several checkboxes: 'Bind parameters', 'Show this query here again', 'Retain query box', 'Rollback when finished', and 'Enable foreign key checks'. There are also 'Simulate query' and 'Go' buttons.

So now our database has a whole slew of names, ages, dogs, etc:



The screenshot shows the phpMyAdmin interface for the 'human' table. The 'Structure' tab is selected. The SQL tab shows the result of the previous update query: 'Showing rows 0 - 7 (8 total, Query took 0.0003 seconds.)' with the query 'SELECT \* FROM `human` WHERE 1'. The results grid displays the following data:

	name	age	dogName	ID
<input type="checkbox"/>	Fred	19	Fido	1
<input type="checkbox"/>	Steve	53	Rex	2
<input type="checkbox"/>	Human1	23	Doggo	3
<input type="checkbox"/>	Matt	74	Marly	4
<input type="checkbox"/>	Caitlin	42	Maureen	5
<input type="checkbox"/>	Kory	37	Mimi	6
<input type="checkbox"/>	Jessica	32	Talula	7
<input type="checkbox"/>	Daphne	24	Scooby Doo	8

## Deleting Data:

It's likely the case that you'll need to delete a row (or multiple) at some point in time. To delete data in mysql, you'll need to follow the syntax:

```
1 | DELETE FROM `tableName` WHERE `condition`
```

Suppose we wanted to get rid of the row that contained a dog named Fido, in practice, that looks like:

The screenshot shows the phpMyAdmin interface. On the left, the database tree shows 'myDatabase' selected. In the center, the 'SQL' tab is active, displaying a query in the 'Run SQL query/queries on table myDatabase.human:' field:

```
1 DELETE FROM `human` WHERE `dogName`="Fido"
```

To the right of the query, there's a 'Columns' panel listing the table's columns: name, age, dogName, and ID. Below the query field are several buttons: SELECT \*, SELECT, INSERT, UPDATE, DELETE, Clear, Format, Get auto-saved query, Bind parameters, Delimiter ;, Show this query here again, Retain query box, Rollback when finished, Enable foreign key checks, Simulate query, and Go.

In phpMyAdmin, when you're deleting code, you'll be prompted by the UI. That's very thoughtful of the UI creators, however, that's not the case for deleting rows programmatically. The Prompt looks like:

This screenshot is similar to the previous one, but a confirmation dialog box is overlaid on the interface. The dialog asks: "localhost:8888 says Do you really want to execute 'DELETE FROM `human` WHERE `dogName` = 'Fido'"? It has 'Cancel' and 'OK' buttons. The rest of the interface is visible in the background.

The database now no longer has a row with a dog named fido:

The screenshot shows the 'Browse' tab for the 'human' table. The results of the query are displayed in a table:

	name	age	dogName	ID
<input type="checkbox"/>	Steve	53	Rex	2
<input type="checkbox"/>	Human1	23	Doggo	3
<input type="checkbox"/>	Matt	74	Maryl	4
<input type="checkbox"/>	Caitlin	42	Maureen	5
<input type="checkbox"/>	Kory	37	Mimi	6
<input type="checkbox"/>	Jessica	32	Talula	7
<input type="checkbox"/>	Daphne	24	Scooby Doo	8

At the bottom of the table, there are buttons for 'Check all', 'With selected:', and links for Edit, Copy, Delete, and Export.

**NOTE:** when deleting, make sure your conditions are specific. If your condition were `1` or something of the sort, you'd wind up deleting the entire database.

# Multiple Tables:

Selecting, Inserting, Updating, and Deleting are all well and good, but none of those really show where the true power of a relational database lies: Table joining.

Splitting data up to make sure we're not storing any redundant material is absolutely necessary with greater and greater databases. However, relationships are not always one to one. Sometimes, they're one to many relationships (like one human having many dogs for pets), or possibly even many to many (such as shared ownership of multiple dogs). Storing a row for each specific instance of these relationships can become extremely frustrating, so it's easier to just store the individual data and join on specific keys.

Before we get into talking about joining tables, we need a second table to begin with. Go to phpMyAdmin and create a new table. We'll call this one dog:

The screenshot shows the phpMyAdmin interface for creating a new table. The left sidebar shows the database structure with 'myDatabase' selected. The main area is titled 'Table name: dog'. The 'Structure' tab is active. The table definition includes four columns: 'name' (TEXT), 'breed' (TEXT), 'age' (INT), and 'humanID' (INT). The 'Storage Engine' is set to InnoDB. The 'Save' button is visible at the bottom right.

After creating the table, we can add data in:

The screenshot shows the phpMyAdmin interface for running SQL queries. The left sidebar shows 'myDatabase' selected. The main area has 'Structure' selected. A SQL query is entered in the 'Run SQL query/queries on table myDatabase.dog:' field:

```
1 INSERT INTO `dog`(`name`, `breed`, `age`, `humanID`) VALUES
2 ('Rex', 'Beagle', 12, 2),
3 ('Doggie', 'Mut', 7, 3),
4 ('Marily', 'Labrador Retriever', 8, 4),
5 ('Maureen', 'Basset Hound', 3, 5),
6 ('Mimi', 'Australian Shepherd', 6, 6),
7 ('Talula', 'Husky', 4, 7),
8 ('Scooby Doo', 'Great Dane', 7, 8)
```

The 'Columns' panel on the right lists the table's columns: name, breed, age, and humanID. The 'Go' button is at the bottom right of the query input area.

Now that we've created a second table, there is a redundancy between the two tables. We are storing dogName twice. Once in the human table, under dogName, and once under the dog table as name. While it is reasonable to try to join our two tables on dog name, thinking about the data for a minute might be worthwhile. Many dogs likely have the same name, and that could lead to us matching data incorrectly. Instead, if we match on ID number, it's extremely unlikely that we'll mismatch anything.

Since that is the case, then, we can delete dogName from our human database:

The screenshot shows the phpMyAdmin interface for the 'myDatabase' database. The 'human' table is selected. The 'dogName' column is highlighted with a red box. A red arrow points to the 'Drop' button in the 'Action' column for the 'dogName' row.

The query itself looks like the below code:

The screenshot shows the phpMyAdmin interface with the SQL tab active. The query 'ALTER TABLE `human` DROP `dogName`;' is entered in the query editor. A red box highlights the 'Drop' button in the 'Action' column of the table structure view, and another red box highlights the 'Yes' button in the confirmation dialog.

Now we no longer have redundant code!

## Joining Tables

There are a number of different types of joins among database tables, however we won't be discussing the differing types of joins for now. The basic syntax of a join goes something like:

```
1 | SELECT `table1`.`columnName1` `table2`.`columnName2` `table`.`columnName3`  
     FROM `table1` JOIN `table2` ON `equalityStatement` WHERE `condition`
```

To join both our human and dog tables, we just need to join on the human's ID as well as the dog's humanID.

The screenshot shows the phpMyAdmin interface for the 'myDatabase' database. The 'dog' table is selected. In the SQL tab, a JOIN query is written: 'SELECT human.name, human.age, dog.name, dog.breed, dog.age FROM human JOIN dog ON human.ID = dog.humanID'. The results pane shows the columns: name, breed, age, and humanID.

The data returned were both from the `human` and the `dog` tables! The only downside to this is that the columns are all named exactly what they are in their own database.

Showing rows 0 - 6 (7 total, Query took 0.0003 seconds.)

```
SELECT human.name, human.age, dog.name, dog.breed, dog.age FROM human JOIN dog ON human.ID = dog.humanID
```

Show all | Number of rows: 25 Filter rows: Search this table

name	age	name	breed	age
Steve	53	Rex	Beagle	12
Human1	23	Doggo	Mut	7
Matt	74	Marly	Labrador Retriever	8
Caitlin	42	Maureen	Basset Hound	3
Kory	37	Mimi	Australian Shepherd	6
Jessica	32	Talula	Husky	4
Daphne	24	Scooby Doo	Great Dane	7

Additionally, you can also search with a specific search parameter in place:

Run SQL query/queries on table myDatabase.dog: ⚡

```
1 SELECT human.name, human.age, dog.name, dog.breed, dog.age FROM human INNER JOIN
2 dog ON human.ID = dog.humanID WHERE human.age > 25
```

Columns

- name
- breed
- age
- humanID

Bind parameters [ ]

[ Delimiter ; ]  Show this query here again  Retain query box  Rollback when finished  Enable foreign key checks Go

By adding the conditional, the only joined dog owners and dogs that we'll see are those all older than 25 (all of the humans, at least):

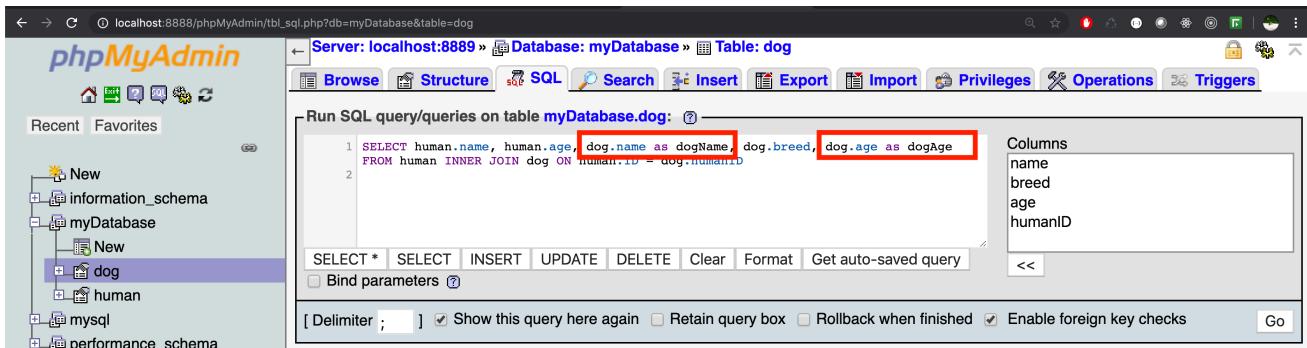
Showing rows 0 - 4 (5 total, Query took 0.0003 seconds.)

```
SELECT human.name, human.age, dog.name, dog.breed, dog.age FROM human INNER JOIN dog ON human.ID = dog.humanID WHERE human.age > 25
```

Show all | Number of rows: 25 Filter rows: Search this table

name	age	name	breed	age
Steve	53	Rex	Beagle	12
Matt	74	Marly	Labrador Retriever	8
Caitlin	42	Maureen	Basset Hound	3
Kory	37	Mimi	Australian Shepherd	6
Jessica	32	Talula	Husky	4

A downside is that it is entirely possible to have data columns with the same name among the tables. However, this is fixible! We can easily fix this problem by using the `as` keyword! By adding an `as` after the column, you can completely rename it:



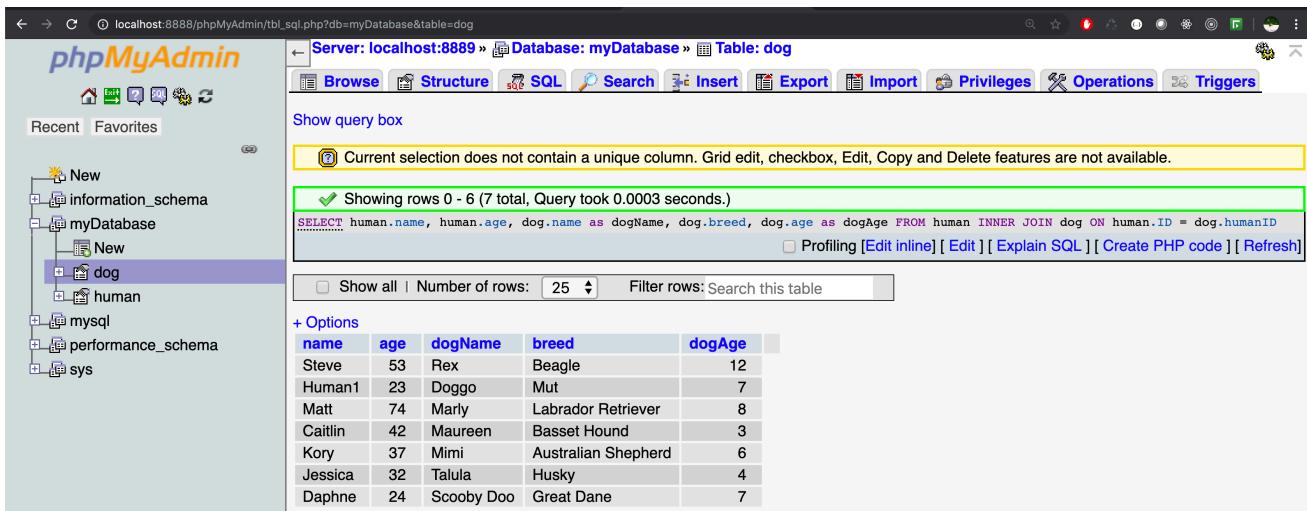
The screenshot shows the phpMyAdmin interface for a database named 'myDatabase'. In the left sidebar, the 'dog' table is selected under the 'myDatabase' schema. The top navigation bar shows 'Server: localhost:8889 > Database: myDatabase > Table: dog'. The main area has tabs for 'Browse', 'Structure', 'SQL', 'Search', 'Insert', 'Export', 'Import', 'Privileges', 'Operations', and 'Triggers'. The 'SQL' tab is active, displaying the following query:

```
1 SELECT human.name, human.age, dog.name AS dogName, dog.breed, dog.age AS dogAge
  FROM human INNER JOIN dog ON human.ID = dog.humanID
```

To the right of the query, there is a 'Columns' panel showing the table structure:

name
breed
age
humanID

Now we return an entirely different series of data, each with their own specific and unique column!



The screenshot shows the results of the executed SQL query. The top message says 'Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.' Below this, a green message indicates 'Showing rows 0 - 6 (total, Query took 0.0003 seconds.)'. The query itself is:

```
SELECT human.name, human.age, dog.name AS dogName, dog.breed, dog.age AS dogAge FROM human INNER JOIN dog ON human.ID = dog.humanID
```

Below the query, there are buttons for 'Profiling [Edit inline] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]'. The main area displays a table with the following data:

	name	age	dogName	breed	dogAge
Steve	53	Rex	Beagle		12
Human1	23	Doggo	Mut		7
Matt	74	Marly	Labrador Retriever		8
Caitlin	42	Maureen	Basset Hound		3
Kory	37	Mimi	Australian Shepherd		6
Jessica	32	Talula	Husky		4
Daphne	24	Scooby Doo	Great Dane		7