the Master Course

{CODENATION}

Introduction to SQL

{CODENATION}

{CODENATION}

Learning Objectives

To understand what a database management system is.

To be able to set up the MySQL server.

To be able to create and use databases.

To be able to create tables and insert data into them.

To be able to work with PRIMARY KEYS.

Databases

What does the term 'database' mean to you?

{CODENATION}

Databases

An organised collection of data.

An electronic system that allows data to be easily accessed, manipulated and updated.

Databases

Modern databases are managed by using a Database Management System (DBMS)

{CODENATION}

Databases

KEYPOINT

A database is a collection of data.

A database management system is software used to store/access/manipulate our data.

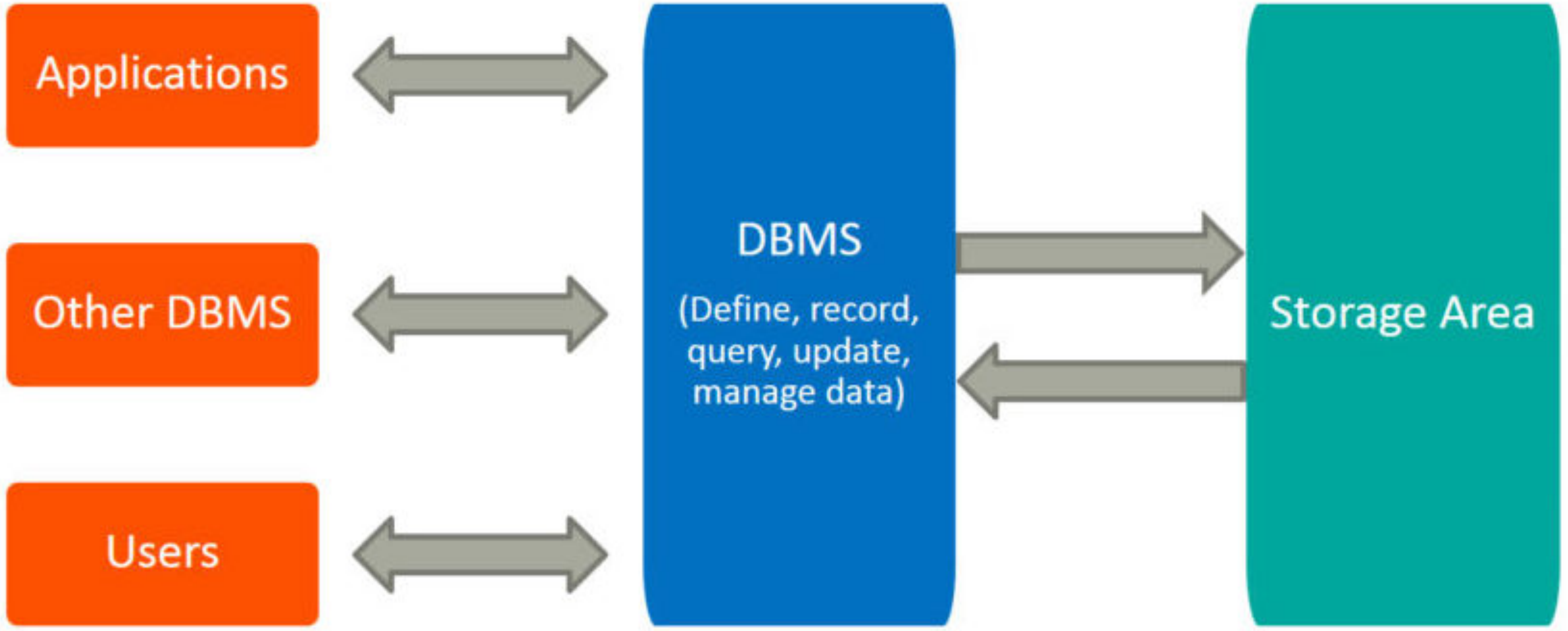
Databases

Some common Database Management Systems (DBMS)

- MySQL
- Microsoft SQL Server
- PostgreSQL
- Oracle

{CODENATION}

Databases



CODENATION

Databases

So when we talk about a MySQL database, we are talking about a database managed using the MySQL DBMS.

{CODENATION}

The two types of databases we will learn at Code Nation are:

Relational databases
Non relational (noSQL) databases.

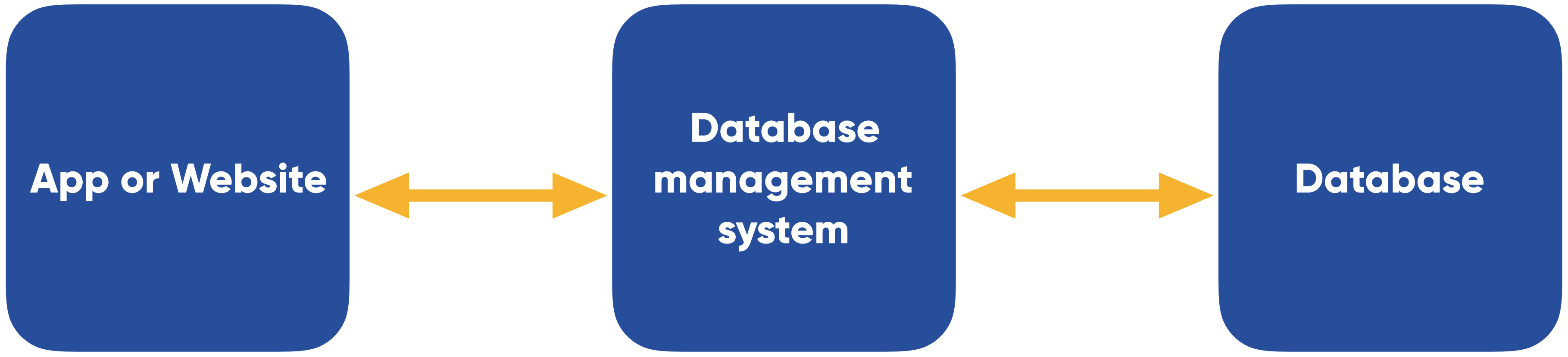
An SQL database is a relational database.

Tables!

A relational database stores related data in tables.

Each table has data stored in rows and columns.

Databases



We use the SQL language to work with the DBMS

The SQL Standard

SQL (Structured Query Language) is the language that is used to interact with databases.

There is a standard SQL syntax that all database management systems adhere to, however there ARE some differences.

The SQL Standard

There are slightly different versions of the SQL language, but different DBMS use the same main syntax.

Most of the SQL database programs have some extra commands/extensions in addition to the SQL standard, but the main syntax still applies.

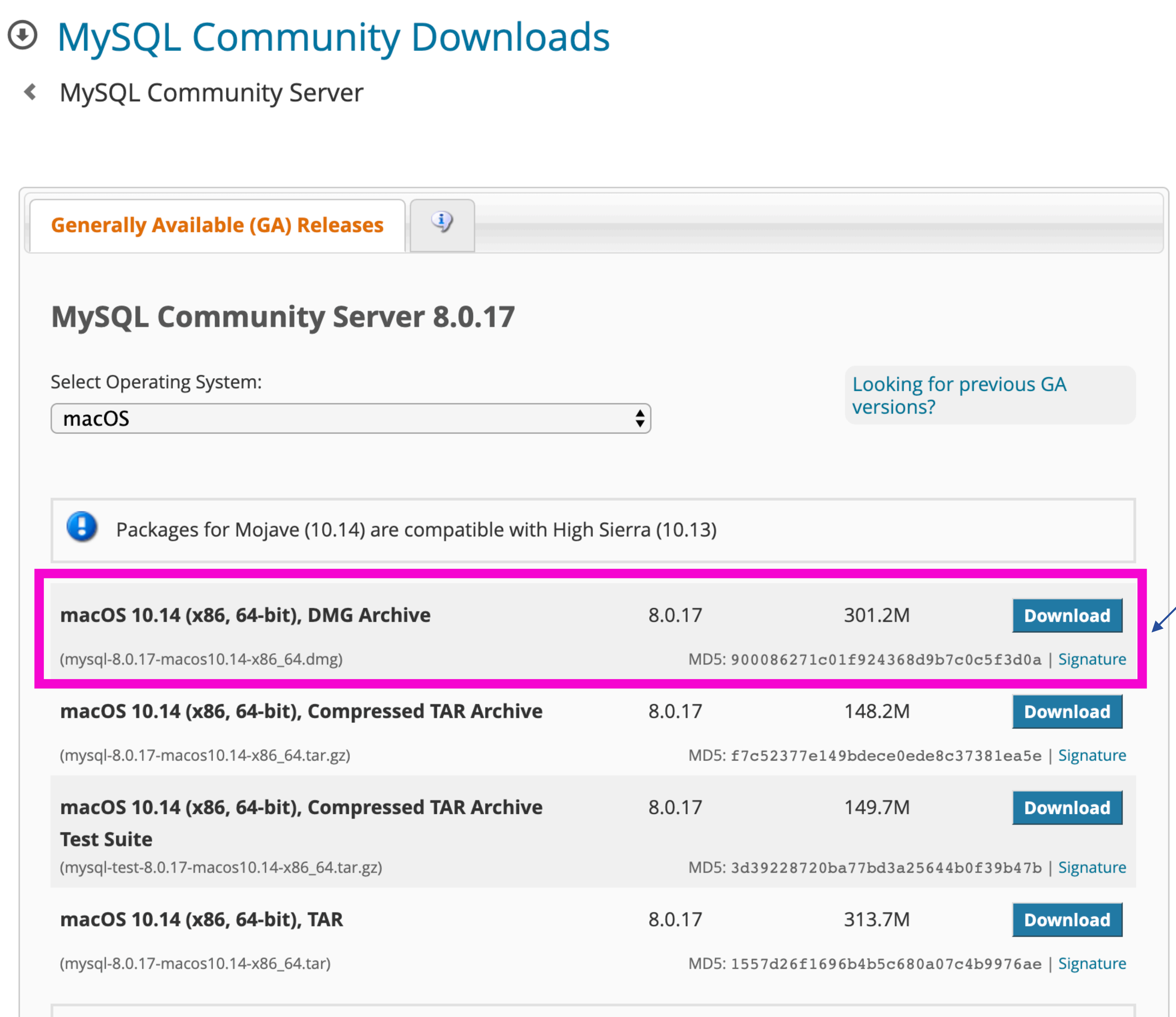
MySQL

MySQL is an open source and free to use database management system. It is extremely popular, and it is the DBMS of choice for a lot of companies.

MySQL

Let's install MySQL on our machines.

Google '**mysql community server**'



MySQL Community Downloads

MySQL Community Server

Generally Available (GA) Releases

MySQL Community Server 8.0.17

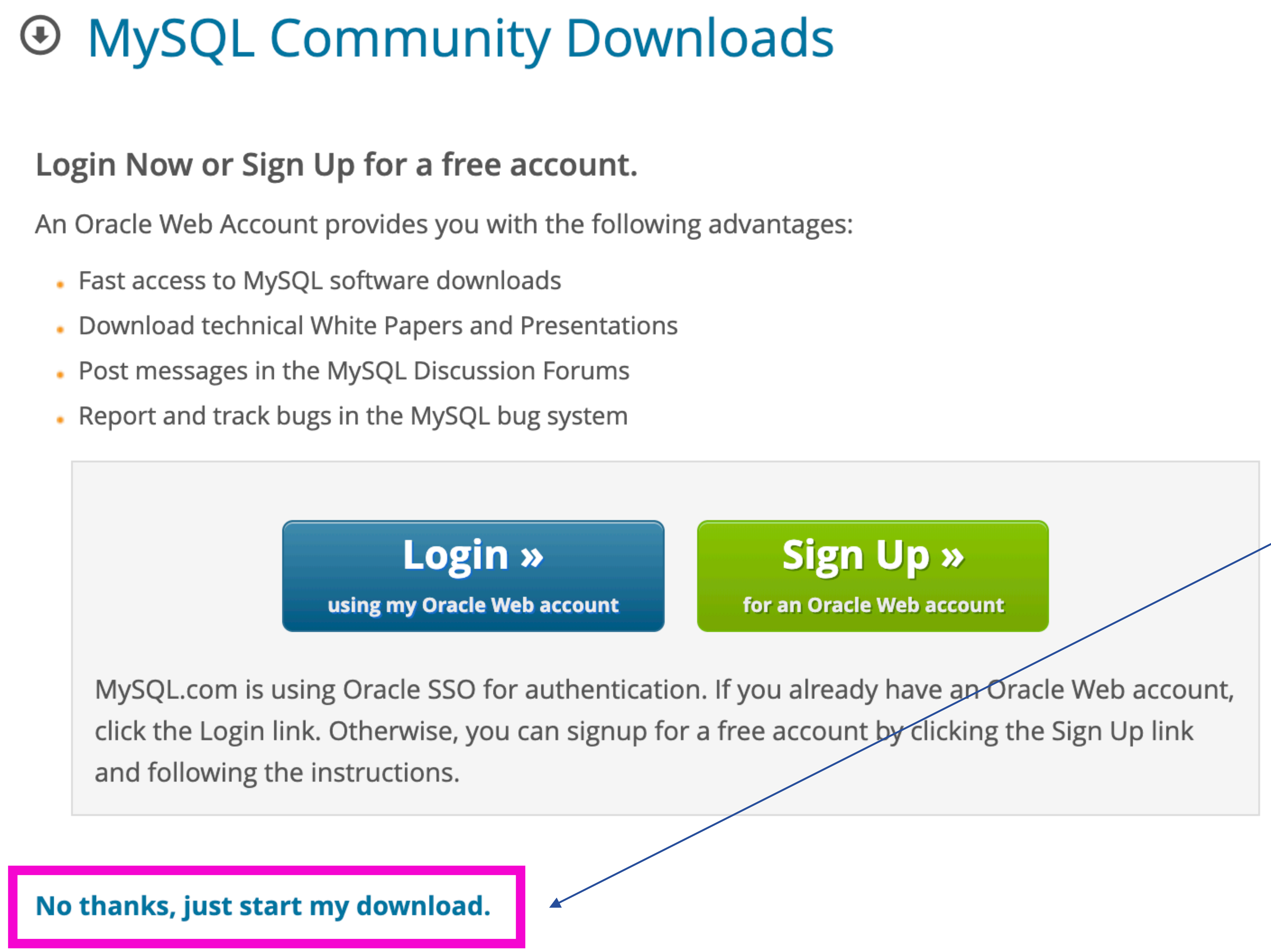
Select Operating System: macOS

Looking for previous GA versions?

Packages for Mojave (10.14) are compatible with High Sierra (10.13)

macOS 10.14 (x86, 64-bit), DMG Archive (mysql-8.0.17-macos10.14-x86_64.dmg)	8.0.17	301.2M	Download MD5: `900086271c01f924368d9b7c0c5f3d0a` | Signature
macOS 10.14 (x86, 64-bit), Compressed TAR Archive (mysql-8.0.17-macos10.14-x86_64.tar.gz)	8.0.17	148.2M	Download MD5: `f7c52377e149bdece0ede8c37381ea5e` | Signature
macOS 10.14 (x86, 64-bit), Compressed TAR Archive Test Suite (mysql-test-8.0.17-macos10.14-x86_64.tar.gz)	8.0.17	149.7M	Download MD5: `3d39228720ba77bd3a25644b0f39b47b` | Signature
macOS 10.14 (x86, 64-bit), TAR (mysql-8.0.17-macos10.14-x86_64.tar)	8.0.17	313.7M	Download MD5: `1557d26f1696b4b5c680a07c4b9976ae` | Signature

Click the .dmg file



MySQL Community Downloads

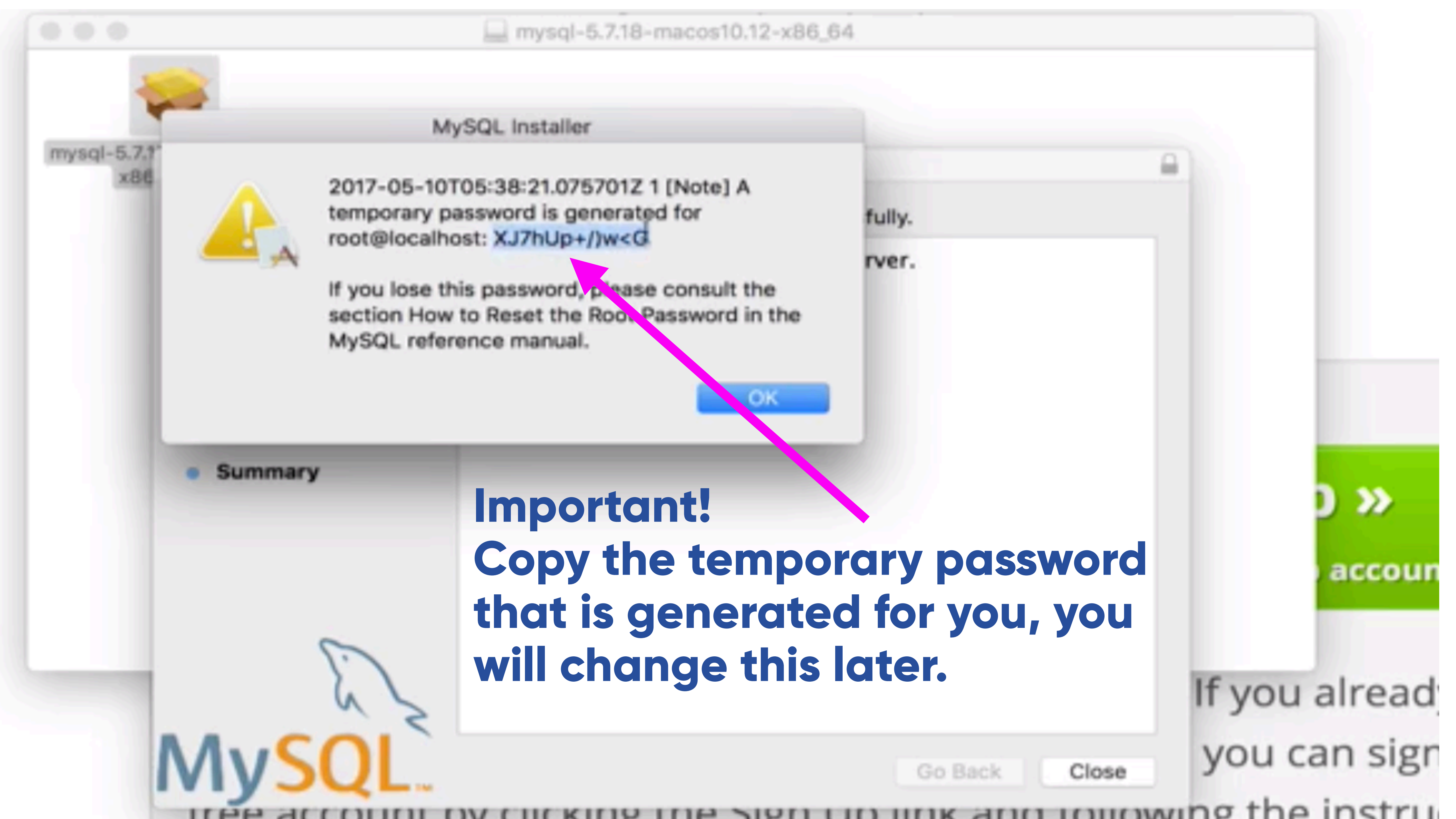
Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system

MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions.

You don't need to sign up, just start your download. Open the downloaded file to start the installation.



2017-05-10T05:38:21.075701Z 1 [Note] A temporary password is generated for root@localhost: XJ7hUp+/)w<G

If you lose this password, please consult the section How to Reset the Root Password in the MySQL reference manual.

Important!
Copy the temporary password that is generated for you, you will change this later.

MySQL

Go to system preferences (click on the apple logo in the top left corner) and open the MySQL server. Start the server!

MySQL

We are going to interact with our MySQL database using the terminal. So open it up!

Type mysql -u root -p
This is 4 separate arguments.

It will ask for your password, and you will need to paste in the temp password it gave you when you installed mysql server.

Once you are in we can interact with the MySQL dbms. The first thing we are going to do is reset the root user password to something easier to remember. Just choose **password**

```
ALTER USER 'root'@'localhost' IDENTIFIED BY ‘password';
```

MySQL

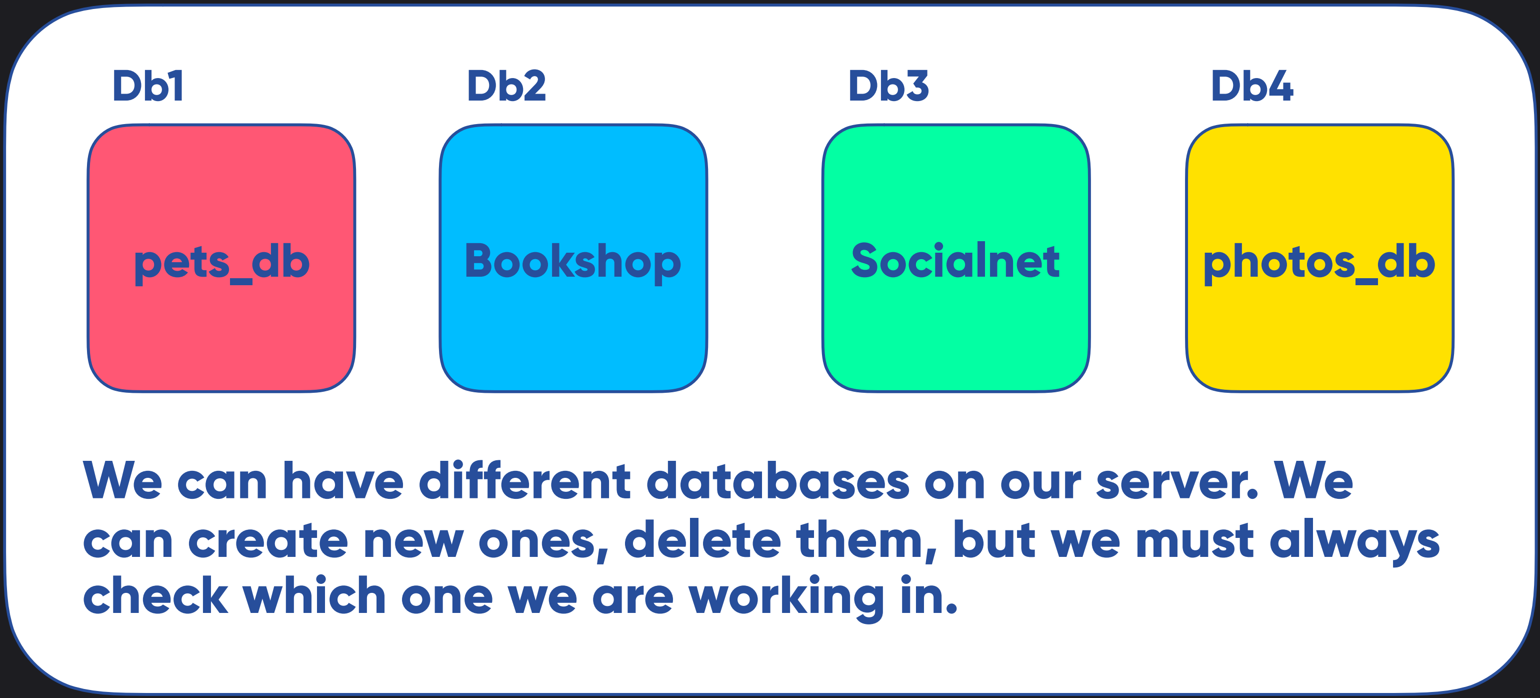
Now every time we want to start using our mysql server in the terminal we type:

```
mysql -u root -p
```

And enter the password **password**

First let's create a database:

Database server



We can have different databases on our server. We can create new ones, delete them, but we must always check which one we are working in.

Run the command:

```
show databases;
```

Note, semi colons are important in SQL.

Run the command:

```
create database my_first_db;
```

$\uparrow$

The name of your database

Run the command:

```
show databases;
```

Double check your new database is still there.

If we want to delete our database we use the command:

```
drop database my_first_db;
```

Show your databases again to check it has been removed.

When we want to use a particular database we run the command:

```
use [the name of your database];
```

For example: `use my_first_db;`

We can run the command:

```
select database( );
```

And this will show us what database we are currently working in.

Task:
create a database called cn_students

Run the command to **use** that database, and then show on the screen that you are working in that database.

A relational database is just a bunch of tables. This is how our data is stored. We can say our database is made from lots of tables that we create.

first_name	Age	Gender
Dan	33	M
Ben	21	M
Charlie	21	F
Sam	30	F
Leon	28	M

Data Types

There are a lot of data types in SQL. It is always good to read the docs to see whats available, but we shall learn the basic ones first.

INT

This data type represents whole numbers.

VARCHAR

This data type represents a string of variable length. It can be between 1 and 255 characters.

We set a max value

VARCHAR(100)	INT	VARCHAR(1)
first_name	Age	Gender
Dan	33	M
Ben	21	M
Charlie	21	F
Sam	30	F
Leon	28	M

Let's create our own table, I'll show you an example first. It will be beneficial for you to copy down the commands.

I will first create a db called codenation.

We create a table in the following way:

```
CREATE TABLE tablename (
column_name data_type,
column_name data_type
);
```

My example:

```
CREATE TABLE staff (
first_name VARCHAR(100),
age INT,
gender VARCHAR(1)
);
```

To show the columns in a table I can run the command:

```
SHOW COLUMNS FROM tablename;
```

Task:

Create a table called **students**,
Which has columns for first_name, age and gender.

Run the command which shows the columns in the table.

That's nice and all, but it's an empty table. We need to put some data in there!

INSERT INTO

General command:

```
INSERT INTO tablename(column_name1,
column_name2) VALUES (data1, data2);
```

I’ll show you using my example:

```
INSERT INTO staff(first_name, age, gender)
VALUES (“Dan", 34, “M”);
```

The order is important, the order of our column names has to match the order of our values.

Task:

>> Insert data about yourself into your students table.

To see if it worked run the following command:

```
SELECT * FROM students;
```

We'll come back to the select statement in much more detail later.

A note about syntax:

While working with SQL, its best practice to type the SQL commands in capitals, and our own values in lowercase. In the terminal this can be a little bit time consuming however, and all commands will still work when written in lowercase. We won't always be working in the terminal, so this will become a lot easier.

You can add more than one row of data at a time.

I'll show you using my example:

```
INSERT INTO staff(first_name, age, gender)
VALUES ('Ben', 21, 'M'),
('Charlie', 21, 'F'),
('Sam', 30, 'F'),
('Leon', 28, 'M');
```

We just separate each row with commas.

Task:

>> Ask the people around you for their data, and add multiple rows to your **students** table (at least 3 rows).

To see if it worked run the following command:

```
SELECT * FROM students;
```

To delete a table you use the command:

```
DROP TABLE tablename;
```

What happens if I don't give all the values for each column in a table?

The value will be set to NULL unless I specify otherwise. I'll show you...

{CODENATION}

A lot of the time you will want to specify that NULL values are not allowed. We do this when we are creating a table. Let's see...

My example:

```
CREATE TABLE staff (
first_name VARCHAR(100) NOT NULL,
age INT NOT NULL,
gender VARCHAR(1) NOT NULL
);
```

I will drop my table. And make it again, specifying that NULL values are not allowed. Then I'll show you what happens if I don't give all the values.

We can also specify DEFAULT values when we are creating a table. This let's us set the data to a specific value if no data is provided.

My example:

```
CREATE TABLE staff (
first_name VARCHAR(100) NOT NULL,
age INT NOT NULL,
gender VARCHAR(100) NOT NULL DEFAULT ‘no gender provided’
);
```

The problem we face at the moment is that we could have rows with identical data, and no way of uniquely identifying them.

What if we had 5 different people called Leon, all who were the same age?

first_name	Age	Gender
Leon	28	M
Leon	28	M
Leon	28	M
Leon	28	M
Leon	28	M

One way for us to identify certain rows of our data is to add an **ID column** to our tables.

first_name	Age	Gender	ID
Leon	28	M	1
Leon	28	M	2
Leon	28	M	3
Leon	28	M	4
Leon	28	M	5

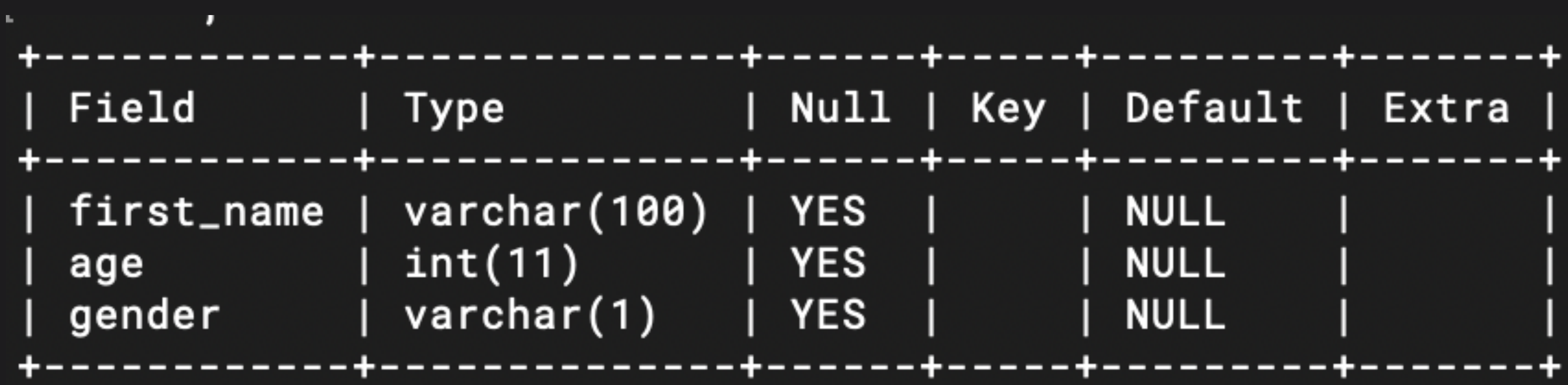
There is a problem here though...

first_name	Age	Gender	ID
Leon	28	M	1
Leon	28	M	2
Leon	28	M	3
Leon	28	M	3
Leon	28	M	1

At the moment, there is nothing preventing me from duplicating IDs. We need a way to prevent this from happening.

We need each row of data to be uniquely identifiable so I could retrieve a particular one. We need to make some data unique (unable to be duplicated).

This is where PRIMARY KEY comes in. We use this to enforce unique values.



```
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| first_name | varchar(100)  | YES  |     | NULL    |       |
| age        | int(11)       | YES  |     | NULL    |       |
| gender     | varchar(1)    | YES  |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
```

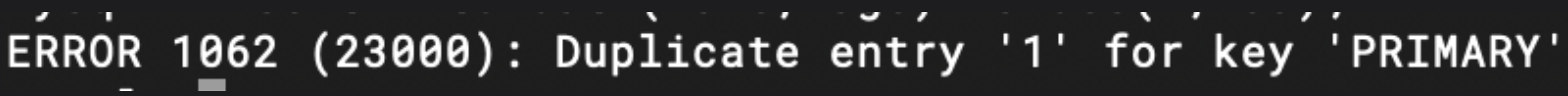
As standard, no keys are assigned to our data.

{CODENATION}

We can assign the ID column as the **PRIMARY KEY**, and this adds a rule that **each one MUST be unique**. We assign the PRIMARY KEY rule when we create a table...

My example:

```
CREATE TABLE staff (
first_name VARCHAR(100) NOT NULL,
age INT NOT NULL,
gender VARCHAR(1) NOT NULL,
id INT NOT NULL PRIMARY KEY
);
```



```
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
```

If I tried to add a duplicate id after I have a PRIMARY KEY rule set up, I'd get an error and I wouldn't be able to do it! So this adds constraints to our table.

Task:

>> Create a table called dogs which has three columns - name, age and id.

Make the id an INT that cannot be NULL, which is also the PRIMARY KEY.

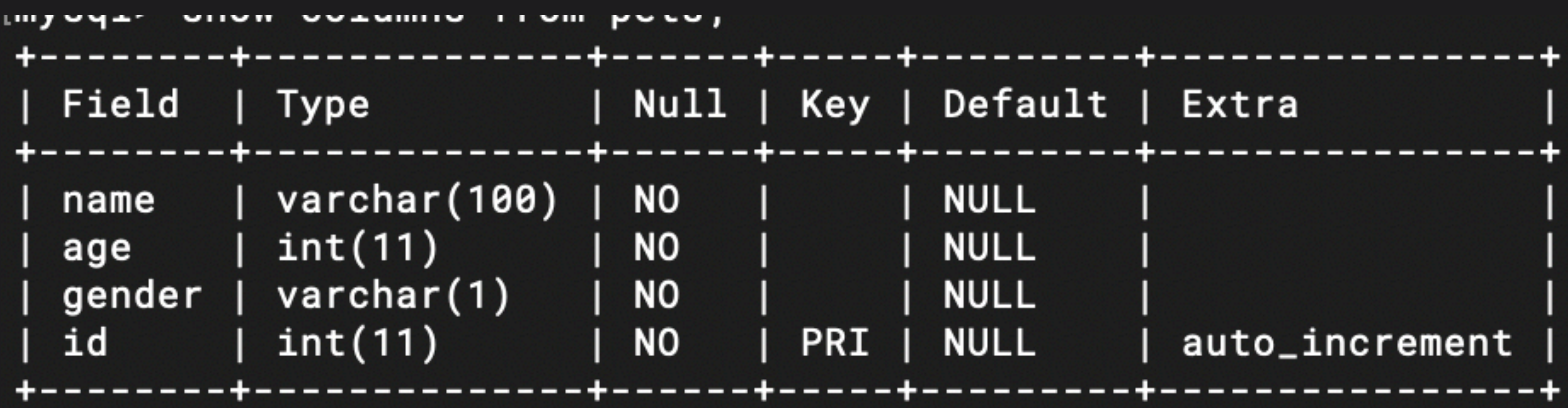
Test your primary key is in place by trying to INSERT INTO with duplicate data.

The final thing we are going to look at in this section is AUTO_INCREMENT

You will have noticed that you need to input your own ids manually. AUTO_INCREMENT allows us to add a rule when we create the table, that simply adds 1 each time we insert some data.

My example:

```
CREATE TABLE pets (
name VARCHAR(100) NOT NULL,
age INT NOT NULL,
gender VARCHAR(1) NOT NULL,
id INT NOT NULL PRIMARY KEY AUTO_INCREMENT
);
```

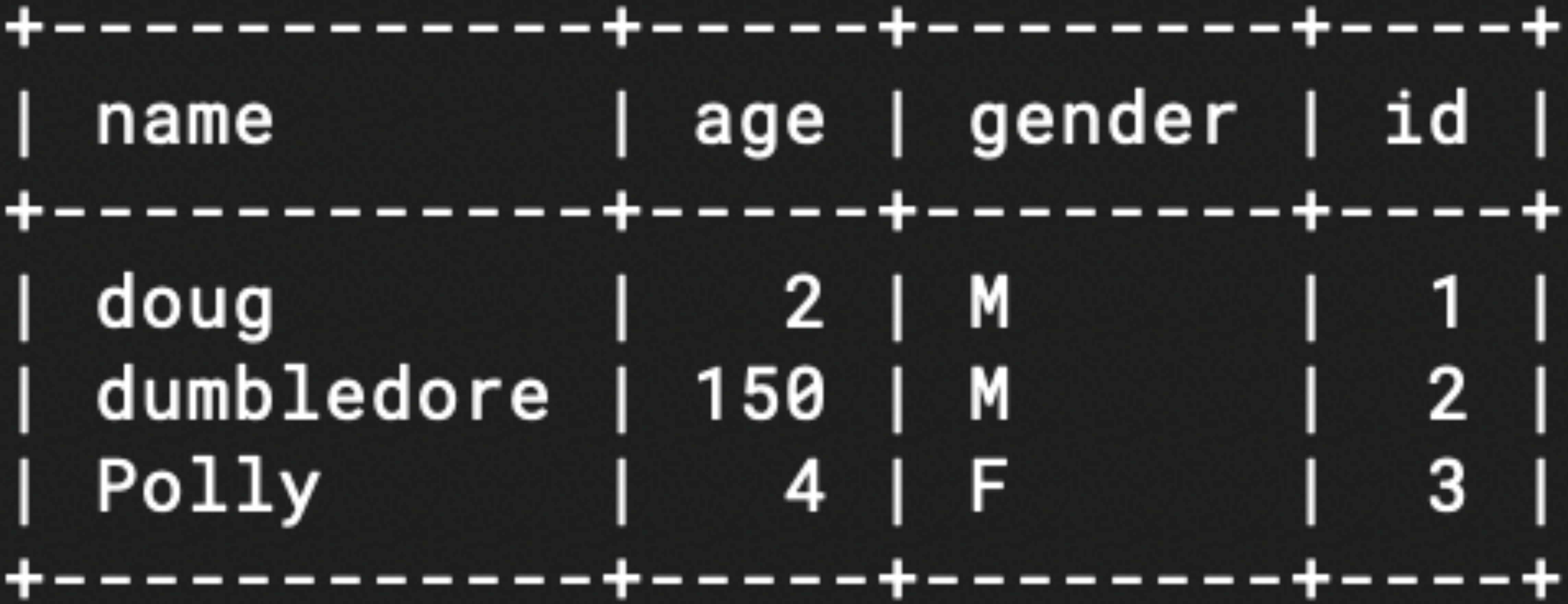


```
mysql> show columns from pets;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| name   | varchar(100) | NO   |     | NULL    |                |
| age    | int(11)      | NO   |     | NULL    |                |
| gender | varchar(1)   | NO   |     | NULL    |                |
| id     | int(11)      | NO   | PRI | NULL    | auto_increment |
+--------+--------------+------+-----+---------+----------------+
```

Now I have added the AUTO_INCREMENT rule to the id, MySQL will know to add a new number to each row. We no longer have to include that field in our INSERTS.

{CODENATION}

```
INSERT INTO pets (name, age, gender)
VALUES ('doug', 2, 'M'),('dumbledore', 150, 'M'),('Polly', 4, 'F');
```



```
+------------+-----+--------+----+
| name       | age | gender | id |
+------------+-----+--------+----+
| doug       |   2 | M      |  1 |
| dumbledore | 150 | M      |  2 |
| Polly      |   4 | F      |  3 |
+------------+-----+--------+----+
```

MySQL adds the id field automatically for us and increments each by 1 every time data is inserted.

Task:

>> Define a table called developers with the following fields:

- id (a number which automatically increments and is the primary key)
- first_name (text, which is required),
- last_name (text, which is required),
- tech_stack (text, which has a default of 'unknown' if no value is given.
- years_developing (a number, which is required)

Test your table definition by inserting at least 4 rows of data. Let me know when you're done and I'll come and have a look.

Revisiting Learning Objectives

To understand what a database management system is.

To be able to set up the MySQL server.

To be able to create and use databases.

To be able to create tables and insert data into them.

To be able to work with PRIMARY KEYS.

{CODENATION}