# Desarrollo Full Stack

Fundación Telefónica Movistar

CURSO DE DESARROLLO FULL STACK.

# Sintaxis básica.

Tenemos tres opciones distintas para crear una base de datos:

- **Desde el entorno gráfico.**
- **Escribiendo y ejecutando código.**
- **Ejecutando código desarrollado por alguien más.**

Para comenzar la práctica de sintaxis básica, vamos a usar la tercera opción que nos permite crear una base de datos con sus tablas y cargarlas de datos, ejecutando un código SQL que descargamos de la página de MySQL.

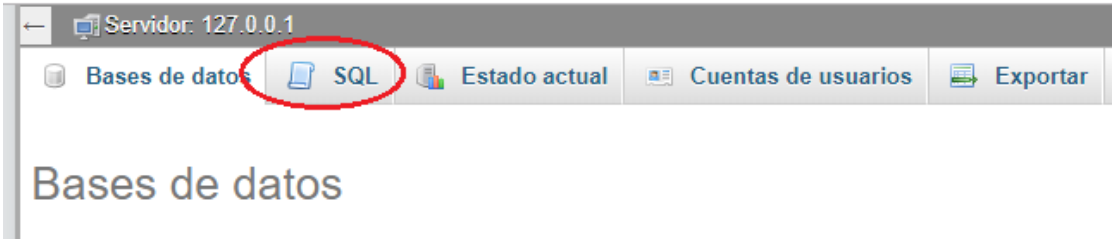
El archivo SQL se encuentra en el DRIVE que utilizamos para compartir material de clase:

## *Creamos la base de datos de práctica.*

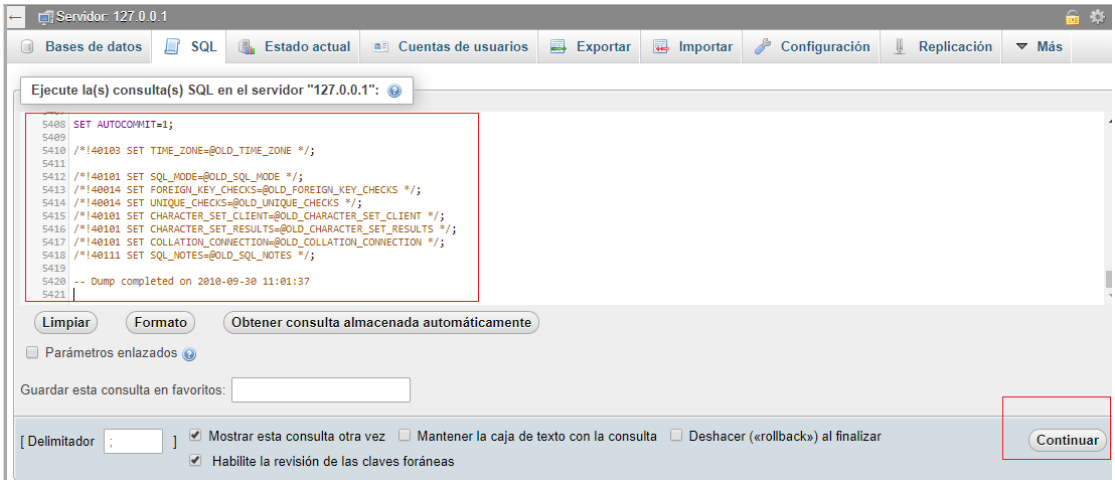
PASO 1) Abrimos el archivo WORLD.SQL con un editor de texto cualquiera.

PASO 2) Seleccionamos todo el contenido del archivo y lo copiamos (CTRL+A y CTRL+C).

PASO 3) En PHPMYADMIN seleccionamos la solapa SQL.

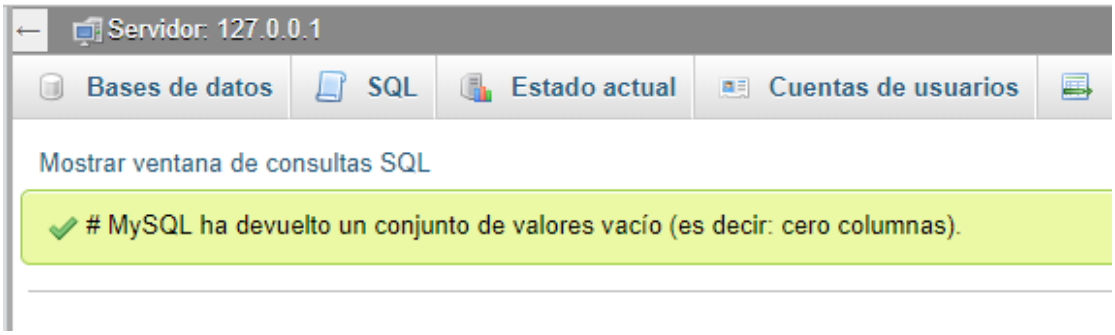



PASO 4) Pegamos el contenido que copiamos del archivo y pulsamos la tecla CONTINUAR:

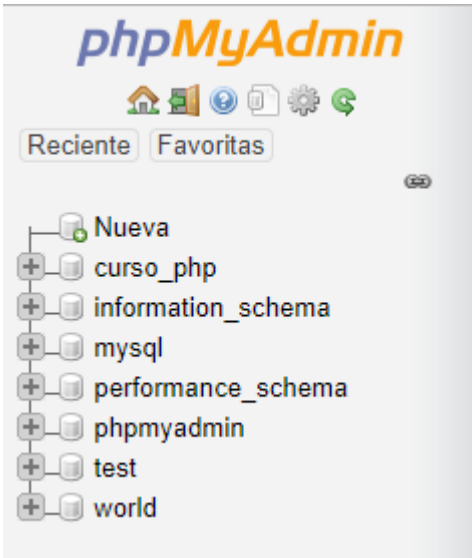



El botón continuar ejecuta la consulta SQL que escribimos.

Si salió todo bien, veremos el siguiente mensaje:




Hecho esto, ya tendremos disponible nuestra base para practicar. Si actualizamos la página (F5) deberíamos ver la nueva base agregada en el menú de la izquierda:




## *Consultas básicas.*

### SELECT / FROM

Al escribir una consulta (QUERY) siempre deberemos indicar al gestor de base de datos, como mínimo, el nombre de la tabla y los campos que necesitamos ver.

Para eso disponemos del comando SELECT para seleccionar los campos y el comando FROM para indicar de que tabla queremos sacar los datos.

```
SELECT *

FROM Country
```

Si no queremos ver todos los registros y solo necesitamos algunos para tener una idea de los datos que contiene la tabla, podemos poner un límite de registros a ver con el comando LIMIT:

```
SELECT *

FROM Country

LIMIT 5
```

**DISTINCT**

El comando DISTINCT nos permite obtener resultados únicos, sin duplicados. Por ejemplo, si hacemos la siguiente consulta:

```
SELECT LANGUAGE

FROM CountryLANGUAGE
```

Obtendremos 984 resultados, pero, a simple vista podemos ver que tenemos registros duplicados. Veamos cómo obtener resultados sin duplicados:

```
SELECT DISTINCT LANGUAGE

FROM CountryLANGUAGE
```

Ahora obtendremos solo 457 registros en el resultado de la consulta.

**WHERE / BETWEEN**

Con WHERE podemos poner condiciones a la búsqueda que hagamos en la base de datos, por ejemplo, queremos encontrar de la tabla COUNTRY aquellos países que pertenecen al continente asiático.

Esta sería la sintaxis sugerida para la consulta:

```
SELECT *

FROM country

WHERE continent = 'asia'
```



Podemos, mediante el uso de operadores lógicos, concatenar más de una condición, por ejemplo, si queremos los países de Asia que además se hayan independizado entre 1940 y 1970, podremos hacer lo siguiente:

```
SELECT DISTINCT *
FROM country
WHERE continent = 'asia'
AND INDEPYEAR BETWEEN 1940 AND 1970
```

Otro ejemplo de uso de operadores lógicos es OR, por ejemplo, queremos todos los países de Europa, cuyo año de independencia esté entre 1970 y 1990 o cuya población supere los 37 millones de habitantes.

```
SELECT DISTINCT *
FROM country
WHERE (continent = 'asia'
AND INDEPYEAR BETWEEN 1970 AND 1990)
OR POPULATION > 37000000
```

**IN / NOT IN**

Nos permite buscar campos que se encuentren dentro de una determinada lista de opciones (o que no se encuentren en dicha lista).

Por ejemplo, queremos la lista de ciudades cuyo código de ciudad (COUNTRYCODE) se encuentre dentro de la siguiente lista: (AFG,DZA,ANT).

```
SELECT DISTINCT NAME,COUNTRYCODE
FROM CITY
WHERE COUNTRYCODE IN ('AFG','DZA','ANT')
```



O podemos buscar los lenguajes cuyo código de país no esté dentro de la lista (ABW,AFG,AGO) y que además sean idiomas oficiales.

**LIKE**

Nos permite establecer un determinado patrón de búsqueda.

Por ejemplo, necesitamos encontrar todos los lenguajes que comiencen con la letra E:

```
SELECT DISTINCT LANGUAGE
FROM COUNTRYLANGUAGE
WHERE LANGUAGE LIKE 'E%'
```

O todos los lenguajes que terminen con la letra A:

```
SELECT DISTINCT LANGUAGE
FROM COUNTRYLANGUAGE
WHERE LANGUAGE LIKE '%A'
```

O todos aquellos que, en algún lugar de su designación, tengan las letras ALA, ya sea al comienzo, en el medio o al final.

```
SELECT DISTINCT LANGUAGE
FROM COUNTRYLANGUAGE
WHERE LANGUAGE LIKE '%ala%'
```

Existen diferentes formas de utilizar LIKE, por ejemplo, si queremos buscar un dato de determinada longitud, por ejemplo 7 caracteres, utilizamos el guion bajo:



```
SELECT DISTINCT LANGUAGE
FROM COUNTRYLANGUAGE
WHERE LANGUAGE LIKE '_______'
```

Este guion también sirve para ignorar una determinada posición de una cadena, por ejemplo, necesitamos obtener todos los países cuyo nombre tengan, como segunda letra, la letra C.

```
SELECT DISTINCT NAME
FROM COUNTRY
WHERE NAME LIKE '_C%'
```

Si necesitamos ubicar palabras que contengan dos letras en particular, hacemos lo siguiente:

```
SELECT DISTINCT NAME
FROM COUNTRY
WHERE NAME LIKE '%K%Z%'
```

¿Y si necesitamos encontrar los países cuya penúltima letra sea la A?

```
SELECT DISTINCT NAME
FROM COUNTRY
WHERE NAME LIKE '%a_'
```



## ORDER BY

Nos permite ordenar el resultado de una consulta, en base a uno o más campos, de forma ascendente o descendente,

El siguiente código nos muestra el listado de países, continentes y regiones, ordenados en forma descendente por continente y en forma ascendente por país.

```
SELECT DISTINCT NAME,CONTINENT,REGION

FROM COUNTRY

ORDER BY CONTINENT DESC, NAME ASC
```

## COUNT

COUNT cuenta los campos que aparecen en el resultado de una consulta.

Por ejemplo, contar la cantidad de países que pertenecen al continente de Oceanía.

```
SELECT COUNT(NAME)

FROM COUNTRY

WHERE CONTINENT = 'OCEANIA'
```

Podemos mejorar un poco la consulta indicándole que solo cuente países distintos, evitando un error por posible registro repetido, y para darle un poco de formato a la salida, le asignamos un nombre al campo resultante:

```
SELECT COUNT(DISTINCT NAME) AS PAISES

FROM COUNTRY

WHERE CONTINENT = 'OCEANIA'
```



## FUNCIONES DE AGRUPAMIENTO

Además de COUNT, disponemos de otras funciones de agrupamiento de datos, por ejemplo, MAX, MIN, SUM y AVG.

Por ejemplo, si queremos saber cuántos países tenemos en la tabla COUNTRY, los contamos con la función COUNT:

```
SELECT COUNT(NAME) AS PAISES

FROM COUNTRY
```

Pero, si quisiéramos conocer la cantidad de países por región, deberemos utilizar un agrupamiento, mediante la sentencia GROUP BY.

```
SELECT CONTINENT AS CONTINENTE,COUNT(NAME) AS PAISES

FROM COUNTRY

GROUP BY CONTINENT
```

También podemos averiguar cuál fue el mayor año en que se declaró una independencia según los datos de nuestra tabla:

```
SELECT DISTINCT MAX(INDEPYEAR)

FROM COUNTRY
```

O el mínimo (veremos que hay un error en el dato cargado):

```
SELECT DISTINCT MIN(INDEPYEAR)

FROM COUNTRY
```



**UPDATE**

Actualiza campos de determinadas tablas.

```
UPDATE COUNTRY
SET INDEPYEAR = 9999
WHERE INDEPYEAR < 0
```

El ejemplo anterior actualiza, en la tabla COUNTRY el campo INDEPYEAR solo para los casos en que el valor cargado en dicho campo es menor a cero, reemplazándolo con el valor 9999.

Si buscamos ahora registros con valores menores a cero para el campo INDEPYEAR, veremos que yo no hay ninguno.

```
SELECT *
FROM COUNTRY
WHERE INDEPYEAR < 0
```

ABM sobre tabla sin dependencias.

Vamos a crear nuestra propia tabla sin ningún tipo de dependencia con otras, para probar un ABM completo.

Para crear una nueva tabla desde el editor de consultas, utilizamos la sentencia CREATE de la siguiente manera:

```
create table usuario (
    nombre varchar(255),
    apellido varchar(255),
    dni int
)
```

Una vez creada, le cargamos algunos registros para practicar.

```
insert into usuario values ('pedro','rodriguez',1234);

insert into usuario values ('jose','suarez',4321);

insert into usuario values ('luis','landrisina',5678);
```

Comprobamos con un SELECT que los datos estén cargados.

```
select *

from usuario
```

| nombre | apellido | dni |
|---|---|---|
| pedro | rodriguez | 1234 |
| jose | suarez | 4321 |
| luis | landrisina | 5678 |

**A) Alta de registro. Agregamos datos a la tabla.**
Lo vimos cuando cargamos nuestra nueva tabla con datos, se hace mediante la sentencia INSERT indicando la tabla en la que insertaremos el registro y los valores a cargar. La lista de valores deberá coincidir con la cantidad de campos de la tabla.

**B) Baja de registro. Eliminamos una fila de la tabla.**
Para eliminar un registro de una tabla, lo hacemos mediante DELETE, indicando la tabla de la cual queremos eliminar el registro, y que condición deberá cumplir dicho registro para ser eliminado.
Por ejemplo, queremos eliminar de la tabla USUARIO el registro cuyo campo DNI tenga cargado el valor 1234.

```
delete

from usuario

where dni = 1234
```

Verificamos mediante un SELECT si el registro fue eliminado:

| nombre | apellido | dni |
|---|---|---|
| jose | suarez | 4321 |
| luis | landrisina | 5678 |

**M) Modificación de un registro. Actualizamos un registro modificando un valor.**

Mediante la sentencia UPDATE podemos modificarle el valor a uno o más campos de un determinado registro que cumpla con una condición.

Por ejemplo, al registro cuyo campo apellido tiene cargado el valor LANDRISINA, queremos reemplazar este valor por LOPEZ.

```
update usuario

set apellido = 'lopez'

where apellido = 'landrisina'
```

Verificamos con un SELECT que el registro fue modificado:

| nombre | apellido | dni |
|---|---|---|
| jose | suarez | 4321 |
| luis | lopez | 5678 |

Podemos modificar más de un campo de una sola vez, por ejemplo, al registro cuyo campo DNI tiene el valor 4321 le queremos modificar los campos NOMBRE y APELLIDO cargándole los valores CLAUDIA y GUTIERREZ respectivamente.

```
update usuario

set nombre = 'claudia',apellido = 'gutierrez'

where dni = 4321
```

Verificamos si la modificación se realizó exitosamente.

| nombre | apellido | dni |
|---|---|---|
| claudia | gutierrez | 4321 |
| luis | lopez | 5678 |