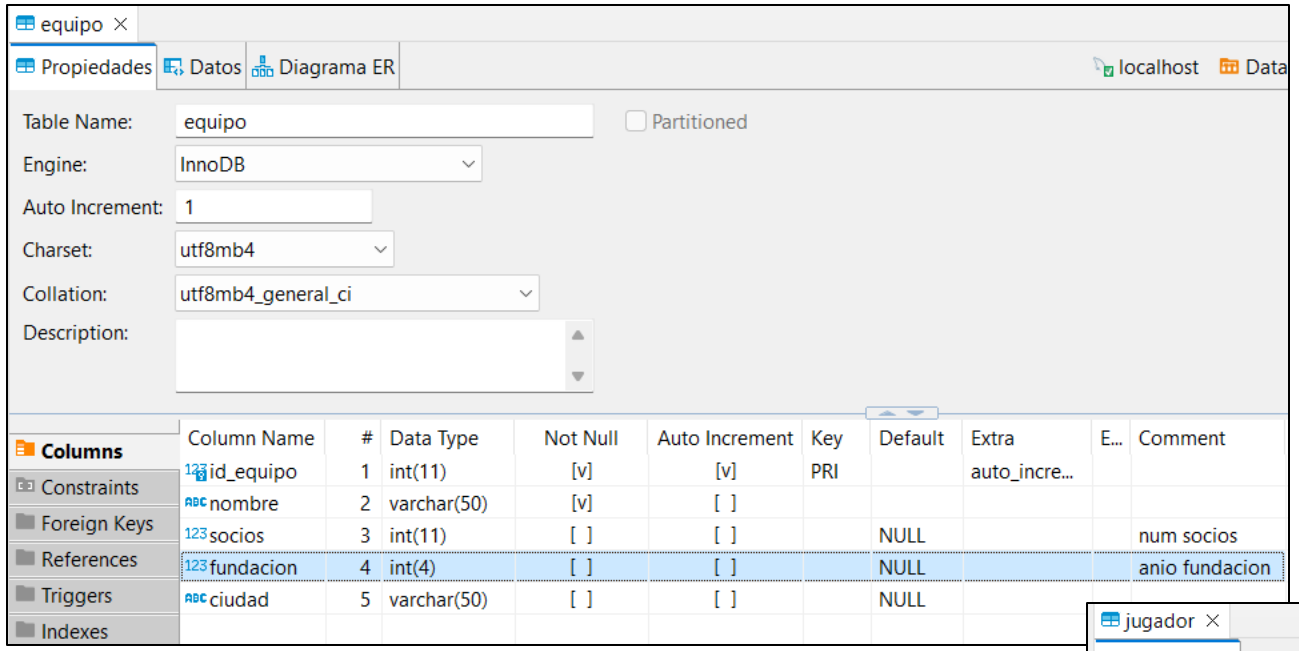
Instalación de Doctrine

Usaremos Doctrine para aplicar ORM en nuestro proyecto PHP.

Para ello, instalaremos Doctrine en nuestro proyecto a través de Composer.

EL mapeo Objeto Relacional será: Modelo Relaciona (tablas) --> Objeto (entidades)

1. Creamos una base de datos: **equipo_jugador.**
2. Creamos un proyecto: **equipo-jugador.**
3. Instalamos Doctrine en nuestro proyecto.
4. Preparamos el proyecto para mapear.
5. Mapeaos para generar las entidades.

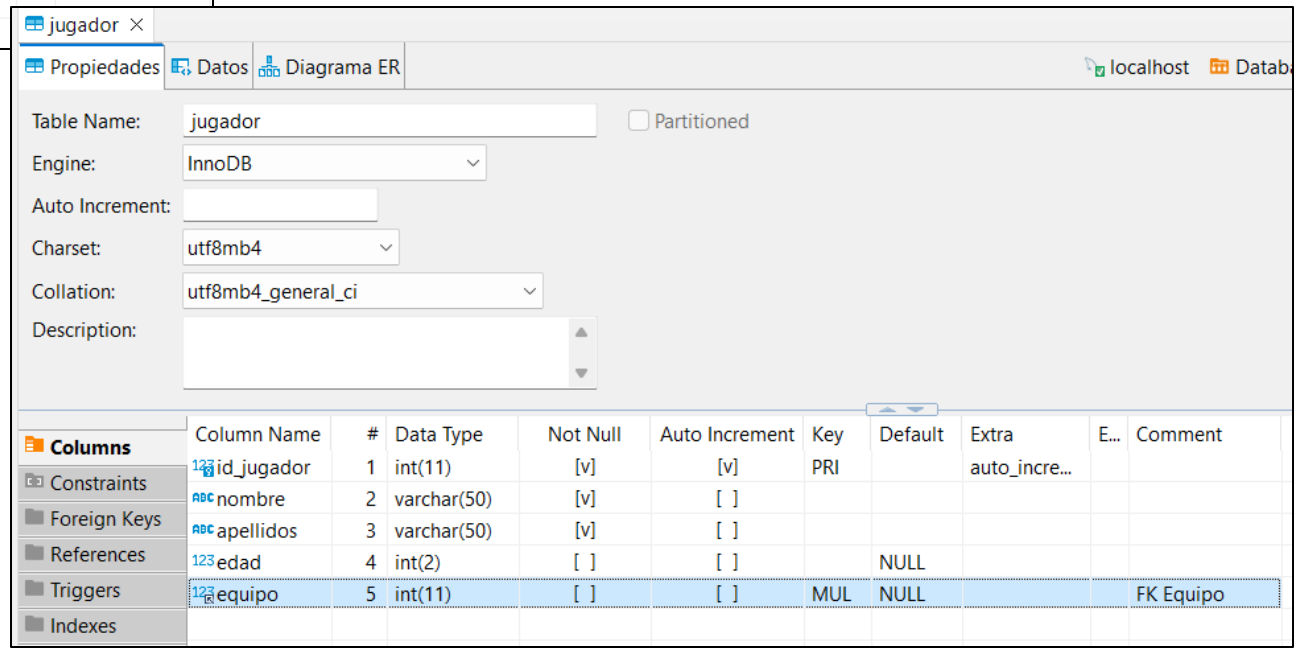


Crear la Base de Datos que mapearemos

Base de Datos: equipo_jugador

Tablas: equipo y jugador relacionadas

(***)



Instalación de Doctrine

Creo una carpeta proyecto en VS Code llamada ***equipo-jugador*** y dentro la carpeta ***src, src/Entity.***

El siguiente paso es **instalar Doctrine** en mi carpeta proyecto desde la terminal que lo podemos hacer de dos formas:

- ejecutando ***composer init***, podremos añadir mucha información a nuestro proyecto e instalar aplicaciones que después podré modificar.
- mediante ***composer require,*** instala la aplicación que después podré modificar.

Creará dentro de nuestro proyecto:

- ***vendor***: contiene las dependencias de aplicaciones instaladas por Composer.
- ***composer.json***: archivo de configuración de Composer, que especifica qué paquetes instalar y sus números de versión. También autor, descripción,...
- ***composer.lock***: archivo generado por Composer que especifica las versiones exactas de las dependencias instaladas.

composer.json

- Instalamos doctrine/orm creando el fichero con un asistente dentro de equipo-jugador: ***composer init***

Nos pide: nombre, descripción, tipo, licencia, qué quiero instalar (doctrine/orm)...

Después tendré que instalar: ***composer install***

Se creará composer.json y composer.lock

- Creamos el fichero de forma automática que después podré modificar: ***composer require*** y el paquete que quiero instalar (doctrine/orm)

Después actualizar: ***composer update (***o ***composer update)***

(***) Instalamos con el asistente composer init: doctrine/orm

Instalar mediante composer require:

composer require doctrine/cache

composer require doctrine/annotations

Añade en composer.json la línea ***src/Entity*** para cargar las clases.

Incluye en tu manual lo realizado explicando qué haces.

Mapeamos O-R

El siguiente paso es crear el fichero **bootstrap.php** que contiene:

- configuración de la conexión a la BD en el array $bdparams
- Configuración de anotaciones
- obtener objeto de la clase EntityManager

bootstrap.php

```php
<?php
// bootstrap.php
// Include Composer Autoload (relative to project root).
require_once "./vendor/autoload.php";
use Doctrine\ORM\Tools\Setup;
use Doctrine\ORM\EntityManager;
$paths = array("./src");
$isDevMode = true;
// configuración de la base de datos
$dbParams = array(
    'driver'   => 'pdo_mysql',
    'user'     => 'root',
    'password' => '',
    'dbname'   => 'equipo_jugador',
    'host' => 'localhost',
);
//utilizará anotaciones:
//ruta a las entidades
//si estamos en modo desarrollo si queremos hacer depuración
//directorio temporal
//sistema de caché
//si usa un tipo de lector de anotaciones simple
$config = Setup::createAnnotationMetadataConfiguration($paths, $isDevMode, null, null, false);
$entityManager = EntityManager::create($dbParams, $config);
```

Creamos el fichero **cli-config.php**

Cargará bootstrap que tiene la creada $entityManager

cli-config.php

```php
<?php
// Carga tu fichero project bootstrap
require_once 'bootstrap.php';

use Doctrine\ORM\Tools\Console\ConsoleRunner;

//obtine el EntityManager que se lo pasa a la consola
//podríamos crear una función en bootstrap y llamarla aquí
//$entityManager = GetEntityManager();
//o directamente usar los datos definidos en bootstrap

return ConsoleRunner::createHelperSet($entityManager);
```

Creamos las clases

Las clases se pueden crear manualmente o usando el comando:

./vendor/bin/doctrine orm:convert-mapping --from-database annotation src/Entity

Vemos las clases creadas y que se han añadido metadatos mediante anotaciones en cada una de ellas.

Estas anotaciones se introducen dentro de bloques de comentarios DocBlock precedidas de @ y pueden tener atributos.

Para asociar clase con tabla se usa @Entity (indica que la clase es una entidad) y @Table (indica con qué tabla se asocia).

El bloque se sitúa justo encima de la declaración de la clase:

```
/**
 * Equipo
 *
 * @ORM\Table(name="equipo")
 * @ORM\Entity
 */
```

En general las anotaciones a especificar son:

- tabla con la que está mapeada la entidad
- columna con la que está mapeado cada atributo
- modificadores de los atributos
- asociaciones con otras entidades.

Modificar las entidades (clases)

- Sus **atributos** deben ser protected o **private**
- Debe haber **getter y setter** para todo atributo excepto los autogenerados (auntoincrementados) que no debe haber setter.
- Pueden tener constructor (solo lo creamos si hay relaciones colecciones).

Por cada atributo hay una anotación ***@ORM\Column*** indicando con qué campo (name) se asocia y al menos el tipo de dato (type).

Por ejemplo, para el primer atributo idEquipo tenemos:

@ORM\Column(name="id_equipo", type="integer", nullable=false)

@ORM\Id: que indica que es clave primaria

@ORM\GeneratedValue(strategy="IDENTITY"): indica que es valor autogenerado

(***)Añade getter y setter necesarios.

Manejo de entidades

Es necesario que los scripts que programes, añadan en las primeras líneas:

Entidad.php

bootstrap.php (en este fichero se declara $entityManager que contiene métodos para gestionar las entidades)

CONSULTA Y MODIFICACIÓN

Con el método ***find()*** busco entidades por clave y después con su get y set accedo y modifico sus datos.

Caso de modificar, hay que usar el método ***flush()*** para que los cambios se salven en la BD.

Doctrine será el encargao de ejectuar la SQL correspondiente.

Cuando un registro no existe, el método find() delvolverá NULL.

*****Pega ***ejemplos_basicos.php*** en nuestro proyecto y probar*****

(***) Programa que reciba por URL el código de un equipo y muestra todos sus datos y si no existe un mensaje apropiado

Manejo de entidades

INSERCCIÓN

- Se crea un nuevo objeto

- se indica que se almacene en la BD con ***persist()***

- ***flush()***

*****Pega ***crear_equipo.php*** en nuestro proyecto y probar*****

(***) Crea un formulario que inserte un equipo y si hay error al crearlo que muestre un mensaje.

BORRADO

- seleccionar el objeto a borrar

- se elimina con ***remove()***

- ***flush()***

*****Pega ***borrar.php*** en nuestro proyecto y probar*****

(***) Crea un formulario para borrar equipos.

Asociación de muchos a uno unidireccional

La relación jugador-equipo, es de este de este tipo 1:N

- Un equipo tiene muchos jugadores.
- Un jugador pertenece a un equipo.

En la BD se traduce a: la tabla jugador tendrá un campo equipo que referencie el *id_equipo* de la tabla equipo.

En las clases se traduce a: la clase Jugador tendrá un atributo que es un objeto de clase Equipo; contiene una referencia al equipo correspondiente.

En la clase vemos la notación que indica relación *ManyToOne* (muchos a uno), *name* de la tabla que tiene la referencia y *referencedColumnName* para indicar el campo de la tabla.

```
@ORM\ManyToOne(targetEntity="Equipo")
@ORM\JoinColumns({
     @ORM\JoinColumn(name="equipo", referencedColumnName="id_equipo")
})
```

*****Pega ***jugador_equipo.php*** en nuestro proyecto y probar*****

Modifica el script para ver todos los datos del jugador y del equipo al que pertenece.

Asociación de muchos a uno bidireccional

Asociaciones bidireccionales: ambas entidades reciben referencias de la entidad asociada.

En nuestro ejemplo, añadiremos un atributo jugadores en la clase Equipo.

En un equipo hay muchos jugadores, con lo que hay que almacenar muchas referencias. En lugar de usar un array PHP, usaremos una clase Doctrine, ***ArrayCollection***.

Lo implementamos en dos nuevas clases EquipoBidireccional y JugadorBidireccional.

EquipoBidireccional: es la clase Equipo con el nuevo atributo y un constructor para inicializar el objeto ArrayCollection.

```
/**
 * Un equipo tiene muchos jugadores
 * @ORM\OneToMany(targetEntity="JugadorBidireccional", mappedBy="equipo")
 */
private $jugadores;

public function __construct() {
    $this->jugadores = new ArrayCollection();
}
```

Asociación de muchos a uno bidireccional

JugadorBidireccional: es la clase Jugador con la anotación que indica que es una asociación inversa: *inversedBy*.

```
/**
 * @ORM\ManyToOne(targetEntity="EquipoBidireccional", inversedBy = "jugadores")
 * @ORM\JoinColumn(name="equipo", referencedColumnName="id_equipo")
 **/
private $equipo;
```

*****Pega ***probar_bidireccional.php*** en nuestro proyecto y probar*****

Modifica el script para ver todos los datos del equipo y todos los datos de los jugadores que pertenecen a ese equipo.

Repositorio Doctrine

Es una clase que contiene consultas relacionadas con una entidad/clase.

Con *getRepository()* de *EntityManager*, podemos obtener algunos métodos básicos. Recibe el nombre de una entidad y devuelve un objeto *EntityRepository*:

- *findBy()*: para utilizar criterios de búsqueda que se introducen en un array.
- *findOneBy()*: igual al anterior pero devuelve un resultado.
- *findAll()*: todas la filas de la tabla.

*****Pega ***findBy.php*** en nuestro proyecto y probar***** ¿Qué realiza este código?

(***) Escribe un programa que reciba por URL el nombre de una ciudad y muestre los datos de los equipos de esa ciudad. Inventa búsquedas con estos 3 métodos.

Podemos crear nuestros propios repositorios creando una clase que herede de EntityRepository con nuestros propios métodos y programando nuestras consultas DQL.