UD 3: Bases de datos y servicios REST
6.- Consultas relacionadas en Spring Data JPA

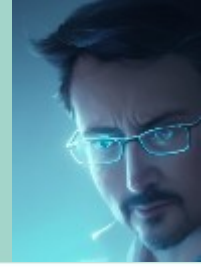
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Objetivos de la sesión:

- *Crear **entidades JPA** asociadas a una tabla que tengan **relaciones con otras tablas** de la Base de Datos:*
    - *@ OneToMany*
    - *@OneToOne*
- *Saber definir tablas con clave primaria compuesta.*
- *Saber enlazar tablas con clave primaria compuesta*

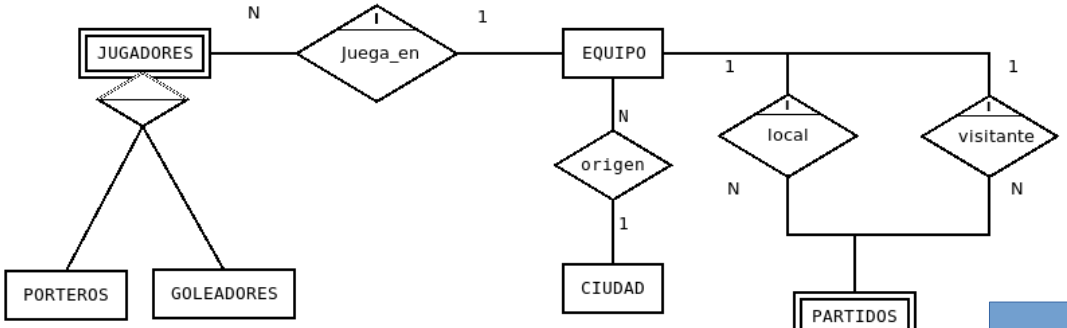
UD 3: Bases de datos y servicios REST
6.- Consultas relacionadas en Spring Data JPA

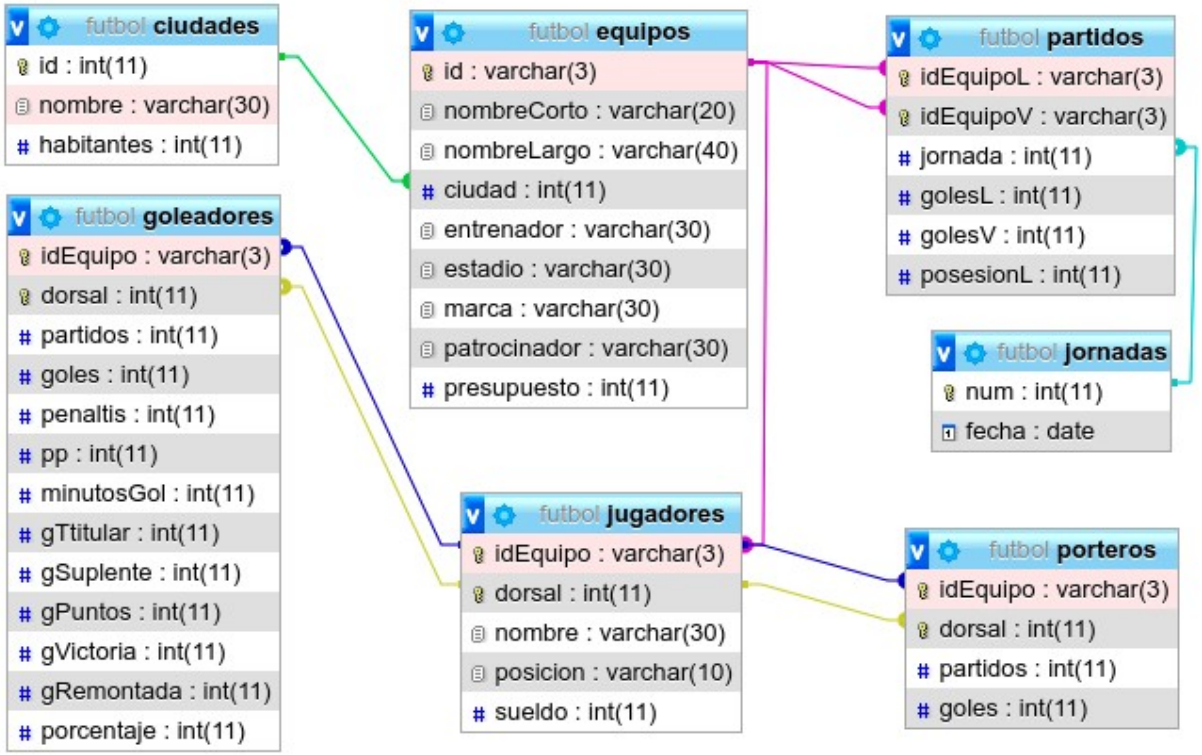
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com



Continuamos con el proyecto anterior ***"api_rest_mysql_futbol"***:

Recordemos la visión global de la Bd de futbol creada en MySQL:





Aclaraciones:

Partidos:

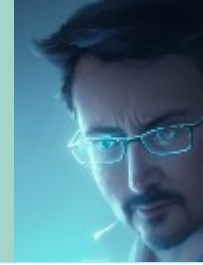
golesL= goles equipo local;
golesV= goles equipo visitante;

Goleadores:

partidos= partidos jugados;
goles= goles marcados;
pp= goles en propia puerta;
minutosGol= minutos para marcar un gol;
gTitular= goles como titular;
gPuntos= goles que dieron puntos;
gVictoria= goles que dieron la victoria;
Porcentaje= porcentaje de goles del equipo;

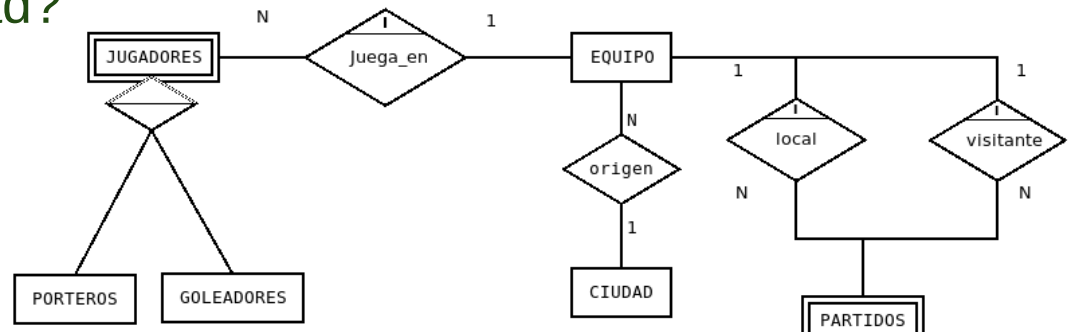
UD 3: Bases de datos y servicios REST
6.- Consultas relacionadas en Spring Data JPA

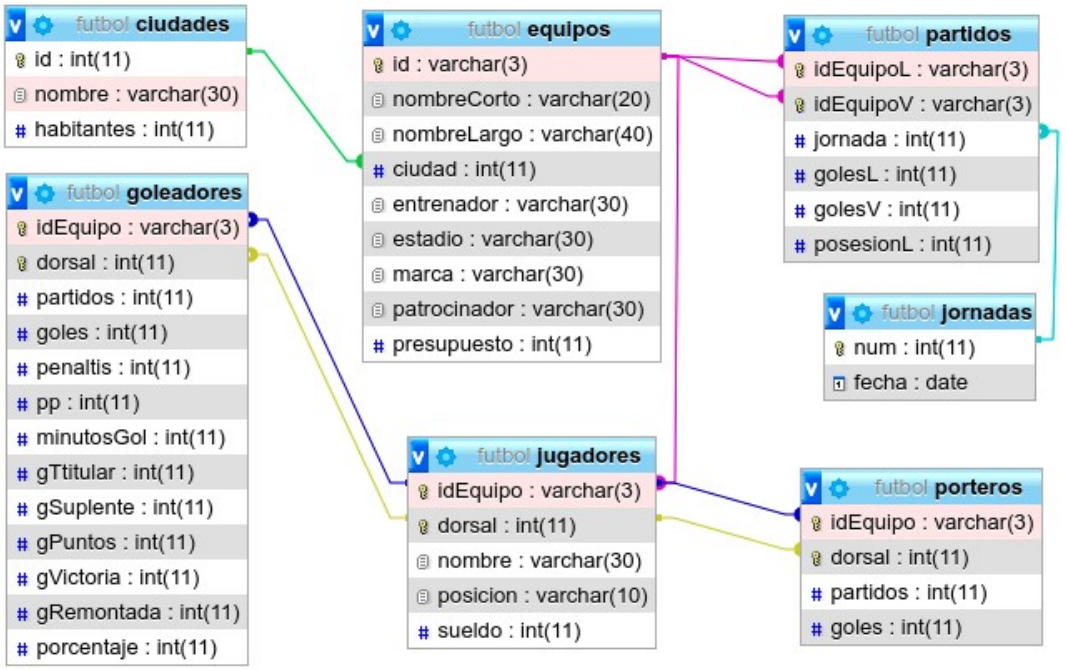
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com



Tablas relacionadas: @OneToMany

¿Como podemos realizar una llamada REST que muestre una ciudad y todos los equipos de esa ciudad?





Tablas relacionadas: @OneToMany

La 1º idea que podríamos tener es que el service llamara al repositorio de la ciudad para obtener sus datos y luego llamara al repositorio del equipo filtrando por ciudad. Con esos datos podríamos volcar la información sobre el ***DTO CiudadInfo*** que contendrá los datos de la ciudad y un atributo con la lista de los equipos con su id y nombre, que podría estar en un ***DTO EquipoInfoNombre***.

CiudadInfo.java

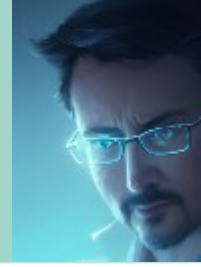
src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > model > dto > CiudadInfo.java > ...

```
12  @NoArgsConstructor
13  @AllArgsConstructor
14  @Data
15
16  public class CiudadInfo implements Serializable{
17      @Column(nullable = false)
18      private Long id;
19      @Size(min=4,message="El nombre debe de tener un tamaño mínimo de 4 carácteres")
20      private String nombre;
21      @Column(nullable = true)
22      private Long habitantes;
23      private Set<EquipoInfoNombre> equiposInfoNombres = new HashSet<>();
24  }
```

El dto ‘EquipoInfoNombre’ no es mala idea , pero...
¿No hay otra forma en la cual solo se realice una petición desde el Servicio y se pueda mapear toda la información en una sola llamada?

UD 3: Bases de datos y servicios REST
6.- Consultas relacionadas en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com



Tablas relacionadas: @OneToMany

La manera más elegante es utilizar **@*OneToMany*** de manera similar a lo hecho en *EquiposDb* con @ManyToOne con la ciudad.

Vamos a definir en ***CiudadDb*** los equipos con @OneToMany porque una ciudad puede tener muchos equipos:

CiudadDb.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > model > db > CiudadDb.java > CiudadDb

```
22  @Data
23  @Entity
24  @Table(name = "ciudades")
25  public class CiudadDb implements Serializable{
26      @Id
27      @GeneratedValue(strategy = GenerationType.IDENTITY)
28      private Long id;
29      @Size(min=4,message="El nombre debe de tener un tamaño mínimo de 4 carácteres")
30      private String nombre;
31      @Column(nullable = true)
32      private Long habitantes;
33      //mappedBy = "clave ajena".En EquipoDb la clave ajena es el atributo ciudadDb
34      @OneToMany(mappedBy = "ciudadDb")
35      private Set<EquipoDb> equiposDb = new HashSet<>();
36  }
```

Como la API Rest tendrá que devolver el ***DTO CiudadInfo*** de la pantalla anterior , y este contiene un ***DTO 'EquiposInfoNombre'*** vamos a definirlo. Almacenará el id y el nombre:

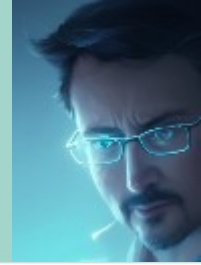
EquipoInfoNombre.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > model > dto > EquipoInfoNombre.java

```
1   package edu.profesor.joseramon.api_rest_mysql_futbol.model.dto;
2
3   import javax.validation.constraints.Size;
4
5   import lombok.AllArgsConstructor;
6   import lombok.Data;
7   import lombok.NoArgsConstructor;
8
9   @NoArgsConstructor
10  @AllArgsConstructor
11  @Data
12  public class EquipoInfoNombre {
13      @Size(min=3,message="El id tiene un tamaño mínimo de 3")
14      private String id;
15      @Size(min=10,max=40,message="El nombre largo debe de tener un tamaño entre
16      private String nombreLargo;
17  }
```

UD 3: Bases de datos y servicios REST
6.- Consultas relacionadas en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com



Tablas relacionadas: @OneToMany

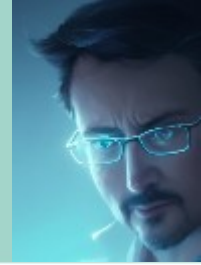
Fijate que tenemos que mapear en CiudadDb un Set de EquipoDb y almacenar en CiudadInfo un Set de EquiposInfoNombres. Debemos actualizar EquipoMapper para añadir el método equiposDbToEquiposInfoNombre. En CiudadMapper debemos añadir ciudadDbToCiudadInfo y decirle que use EquipoMapper para convertir equiposDb a equiposList :

EquipoMapper.java

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.srv.mapper;


import java.util.List;
import java.util.Set;

import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.mapstruct.factory.Mappers;

import edu.profesor.joseramon.api_rest_mysql_futbol.model.db.EquipoDb;
import edu.profesor.joseramon.api_rest_mysql_futbol.model.dto.EquipoInfoNombre;
import edu.profesor.joseramon.api_rest_mysql_futbol.model.dto.EquipoList;

@Mapper
public interface EquipoMapper {
    EquipoMapper INSTANCE= Mappers.getMapper(clazz: EquipoMapper.class);

    @Mapping(target = "nombreCiudad", source = "ciudadDb.nombre")
    EquipoList EquipoDbToEquipoList(EquipoDb equipoDb);

    List<EquipoList> equiposDbToEquiposList(List<EquipoDb> equiposDb);
    Set<EquipoInfoNombre> equiposDbToEquiposInfoNombre(Set<EquipoDb> equiposDb);
}
```

CiudadMapper.java

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.srv.mapper;


import java.util.List;

import org.mapstruct.Mapper;
import org.mapstruct.factory.Mappers;

import edu.profesor.joseramon.api_rest_mysql_futbol.model.db.CiudadDb;
import edu.profesor.joseramon.api_rest_mysql_futbol.model.dto.CiudadInfo;
import edu.profesor.joseramon.api_rest_mysql_futbol.model.dto.CiudadList;

import org.mapstruct.Mapping;

@Mapper(uses = EquipoMapper.class)
public interface CiudadMapper {
    CiudadMapper INSTANCE= Mappers.getMapper(clazz: CiudadMapper.class);

    CiudadList CiudadDbToCiudadList(CiudadDb ciudadDb);
    List<CiudadList> ciudadesToCiudadList(List<CiudadDb> ciudadesDb);

    @Mapping(target = "equiposInfoNombres", source = "equiposDb")
    CiudadInfo ciudadDbToCiudadInfo(CiudadDb ciudadDb);

}
```



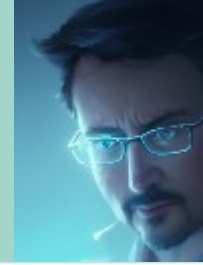
Tablas relacionadas: @OneToMany

Desde el Service solo nos queda modificar el método getCiudadInfoById(Long id) para que mapee el objeto ciudadDb a traves del método del mapper ciudadDbToCiudadInfo. Con estos cambios ya tendríamos sobre CiudadInfo los datos de la ciudad y los equipos de esa ciudad sin tener que hacer 2 consultas al respositorio:

CiudadServiceImpl.java

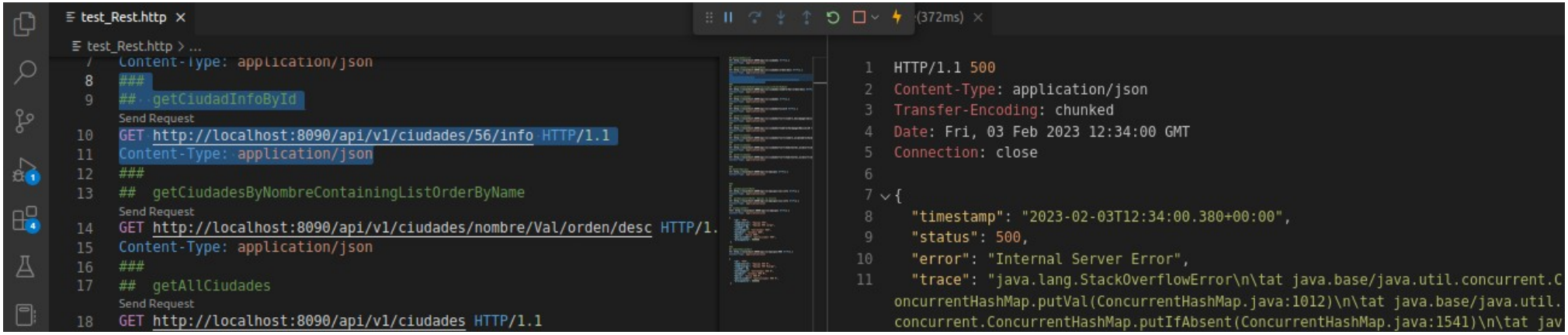
joseramon > api_rest_mysql_futbol > srv > impl > CiudadServiceImpl.java > Language Support for Java(TM) by Red Hat > CiudadServiceImpl

```
56          }
57
58          public Optional<CiudadInfo> getCiudadInfoById( Long id){
59              Optional<CiudadDb> ciudadDb=ciudadRepository.findById(id);
60              if (ciudadDb.isPresent()){
61                  return Optional.of(CiudadMapper.INSTANCE.ciudadDbToCiudadInfo(ciudadDb.get()));
62
63              }else{
64                  return Optional.empty();
65              }
66          }
67      }
```



Tablas relacionadas: @OneToMany

Si ejecutamos la consulta vemos que falla:



¿Hemos hecho algo mal?¿Porque nos falla?

Tablas relacionadas: @OneToMany

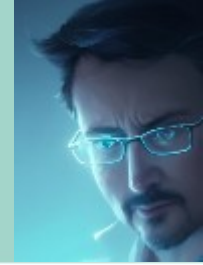
Un StackOverflow siempre es porque nos hemos quedado sin recursos, normalmente por entrar en un bucle infinito.

En nuestro caso el problema viene con ***Lombok***. Para generar el ***HashCode y el ToString*** se utilizan todos los campos. En ciudadDb hacemos referencia a EquipoDb con el @OneToMany y en EquipoDb se hace referencia a ciudadDb con @ManyToOne, lo que provoca que entremos en un bucle que podemos romper obligandole a no tener en cuenta el atributo 'equiposDb' para realizar ese cálculo con las anotaciones que vemos en pantalla:

CiudadDb.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > model > db > CiudadDb.java > ...

```
17  import lombok.NoArgsConstructor;
18  import javax.persistence.OneToMany;
19  import lombok.EqualsAndHashCode;
20  import lombok.ToString;
21
22  @EqualsAndHashCode(exclude = { "equiposDb"}) // Evita StackOverflowError
23  @ToString(exclude = {"equiposDb"}) //al romper la bidereccionalidad
24  @NoArgsConstructor
25  @AllArgsConstructor
26  @Data
27  @Entity
28  @Table(name = "ciudades")
29  public class CiudadDb implements Serializable{
30      @Id
31      @GeneratedValue(strategy = GenerationType.IDENTITY)
32      private Long id;
33      @Size(min=4,message="El nombre debe de tener un tamaño mínimo de 4 carácteres")
34      private String nombre;
35      @Column(nullable = true)
36      private Long habitantes;
37      //mappedBy = "clave ajena".En EquipoDb la clave ajena es el atributo ciudadDb
38      @OneToMany(mappedBy = "ciudadDb")
39      private Set<EquipoDb> equiposDb = new HashSet<>();
40  }
```



Tablas relacionadas: @OneToMany

Vemos que ya hemos solucionado el problema si lo ejecutamos:

test_Rest.http

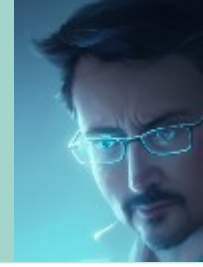
```
Content-Type: application/json
###
## getCiudadInfoById
Send Request
GET http://localhost:8090/api/v1/ciudades/56/info HTTP/1.1
Content-Type: application/json
###
## getCiudadesByNombreContainingListOrderByName
Send Request
GET http://localhost:8090/api/v1/ciudades/nombre/Val/orden/desc HTTP/1.1
Content-Type: application/json
###
## getAllCiudades
Send Request
GET http://localhost:8090/api/v1/ciudades HTTP/1.1
Content-Type: application/json
###
## getAllCiudades
Send Request
GET http://localhost:8090/api/v1/ciudades?size=5 HTTP/1.1
Content-Type: application/json
###
## getAllCiudades
Send Request
GET http://localhost:8090/api/v1/ciudades?sort=nombre,desc&page=1&size=5 HTTP/1.1
```

Response(362ms)

```
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Fri, 03 Feb 2023 12:49:01 GMT
Connection: close

{
  "id": 56,
  "nombre": "València",
  "habitantes": 798000,
  "equiposInfoNombres": [
    {
      "id": "lev",
      "nombreLargo": "LLevant Unió Esportiva"
    },
    {
      "id": "val",
      "nombreLargo": "València Club de Futbol"
    }
  ]
}
```

UD 3: Bases de datos y servicios REST
6.- Consultas relacionadas en Spring Data JPA

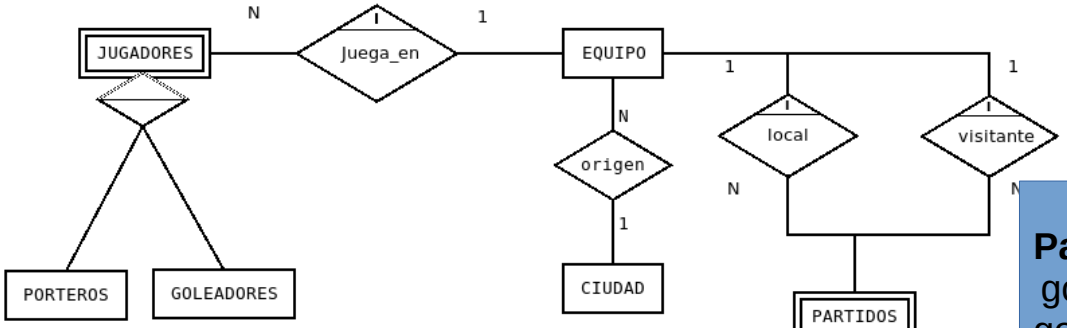
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

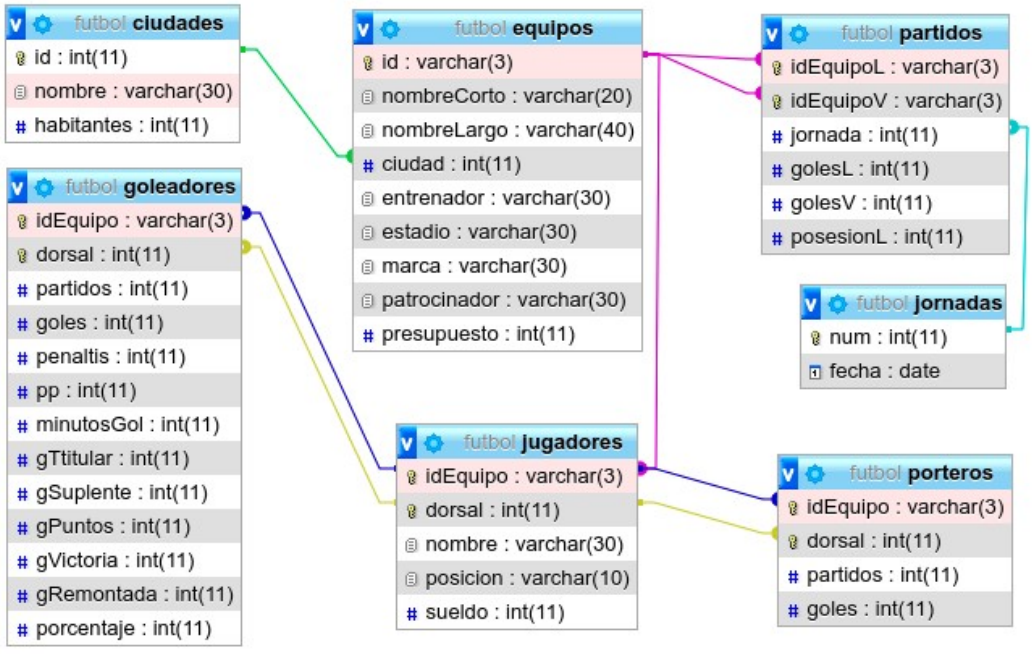


Tablas con clave primaria compuesta:

Si queremos crear la clase que represente a la tabla “Jugadores”.

¿Como configuramos una clave primaria con 2 atributos (id_equipo,dorsal)?





Aclaraciones:

Partidos:

golesL= goles equipo local;
golesV= goles equipo visitante;

Goleadores:

partidos= partidos jugados;
goles= goles marcados;
pp= goles en propia puerta;
minutosGol= minutos para marcar un gol;
gTitular= goles como titular;
gPuntos= goles que dieron puntos;
gVictoria= goles que dieron la victoria;
Porcentaje= porcentaje de goles del equipo;

Tablas con clave primaria compuesta: notación @IdClass

Aunque existen ***2 métodos*** con 2 anotaciones distintas (@IdClass y @EnbeddedId) nosotros vamos a utilizar de momento el ***1º método*** y se deja al alumno poder consultar como implementar el 2º método en : https://www.baeldung.com/jpa-composite-primary-keys

Para utilizar la notación ***@IdClass*** deberemos crear una clase que contenga los campos que actuaran de clave primaria en la tabla:

```sql
-- ----------------------------------------
-- Table `jugadores`
-- ----------------------------------------
DROP TABLE IF EXISTS `jugadores` ;

CREATE  TABLE IF NOT EXISTS `jugadores` (
  `idEquipo` VARCHAR(3) NOT NULL DEFAULT '' ,
  `dorsal` INT(11) NOT NULL DEFAULT '0' ,
  `nombre` VARCHAR(30) NOT NULL ,
  `posicion` VARCHAR(10) NULL DEFAULT NULL ,
  `sueldo` INT(11) NULL DEFAULT NULL ,
  PRIMARY KEY (`idEquipo`, `dorsal`) ,
  CONSTRAINT `jugadores_Equipo`
    FOREIGN KEY (`idEquipo` )
    REFERENCES `equipos` (`id` ))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;
```

JugadorId.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > model > db > J Juga

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.model.db;

import java.io.Serializable;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@NoArgsConstructor
@AllArgsConstructor
@Data
public class JugadorId implements Serializable {
  private String idEquipo;
  private Long dorsal;
}
```

Tablas con clave primaria compuesta: notación @IdClass

Creamos la clase JugadorDb y con la notación @IdClass le especificamos la clase que especifica que atributos actuaran de clave primaria. Esos atributos los marcaremos con @Id en JugadorDb:

JugadorDb.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > model > db > JugadorDb.java > JugadorDb

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.model.db;

import java.io.Serializable;

@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
@IdClass(JugadorId.class)
@Table(name = "jugadores")
public class JugadorDb implements Serializable{
    private static final long serialVersionUID = -8185427783735952 60L;
    @Id
    @Size(min=3,message="El id del equipo tiene un tamaño mínimo de 3")
    private String idEquipo;
    @Id
    @Column(nullable = false)
    private Long dorsal;
    @Size(min=10,max=30,message="El nombre de tener un tamaño entre 10 y 30 caracteres")
    private String nombre;
    @Size(max=10,message="La posición debe de tener un tamaño máximo de 30 caracteres")
    private String posicion;
    private Long sueldo;

}
```

Tablas con clave primaria compuesta: notación @IdClass

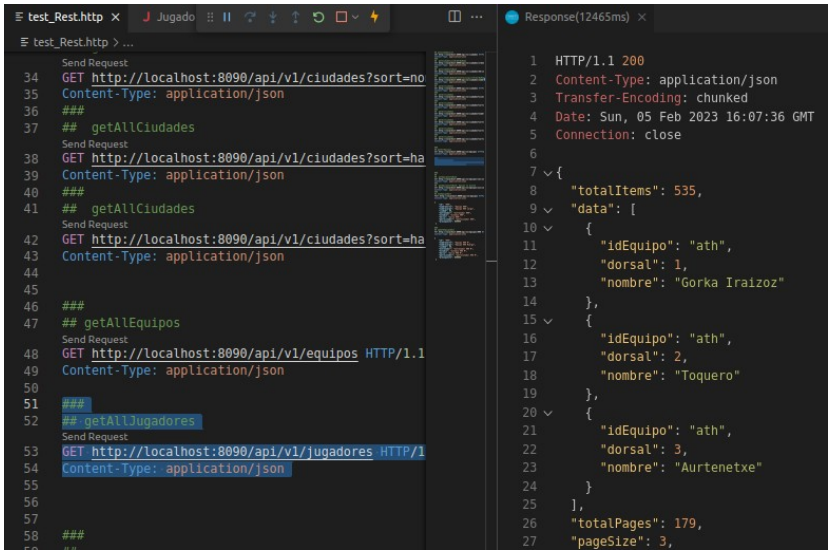
Creemos las clases de repositorio, dto, servicio y controller para poder devolver los datos de cualquier jugador de manera similar a como devolvemos el listado de equipos:

JugadorRepository.java

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.repository;


import java.util.List;

public interface JugadorRepository extends JpaRepository<JugadorDb,JugadorId>{
    //Métodos automáticos en Spring Data JPA
    List<JugadorDb> findAll(Sort sort);
    //Filtrado por nombre usando equivalente a 'LIKE' usando "Containing"
    List<JugadorDb> findByNombreContaining(String nombre,Sort sort);
    //Paginación
    Page<JugadorDb> findAll(Pageable pageable);
    Page<JugadorDb> findByNombreContaining(String nombre,Pageable pageable);
}
```

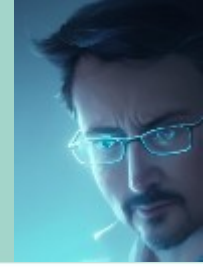
JugadorServiceImpl.java

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.srv.impl;
import edu.profesor.joseramon.api_rest_mysql_futbol.model.db.JugadorDb;

@Service
public class JugadorServiceImpl implements JugadorService{
    private final JugadorRepository jugadorRepository;

    public JugadorServiceImpl(JugadorRepository jugadorRepository){
        this.jugadorRepository=jugadorRepository;
    }

    @Override
    public PaginaDto<JugadorList> findAll(Pageable paging) {
        Page<JugadorDb> paginaJugadorDb=jugadorRepository.findAll(paging);
        return new PaginaDto<JugadorList>(
                    paginaJugadorDb.getNumber(),//número de página solicitada
                    paginaJugadorDb.getSize(), //tamaño de la página
                    paginaJugadorDb.getTotalElements(),//total de elementos devueltos por la consulta sin paginación
                    paginaJugadorDb.getTotalPages(), //total páginas teniendo en cuenta el tamaño de cada página
                    JugadorMapper.INSTANCE.jugadoresDbToJugadoresList(paginaJugadorDb.getContent()),//lista de elementos
                    paginaJugadorDb.getSort()); //ordenación de la consulta
    }
}
```



UD 3: Bases de datos y servicios REST
6.- Consultas relacionadas en Spring Data JPA

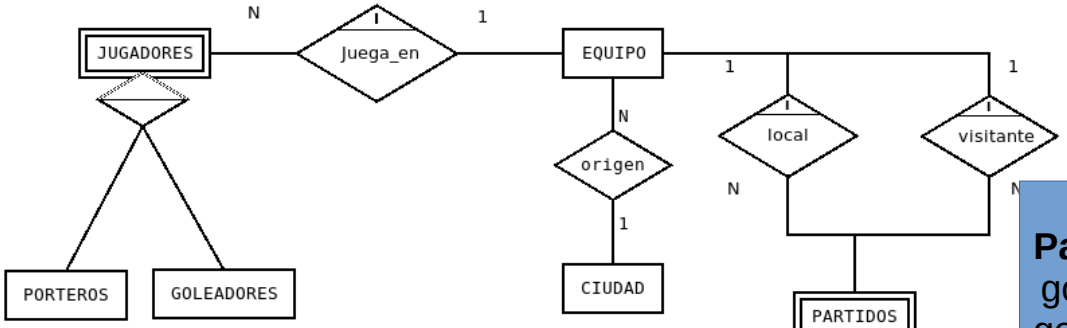
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

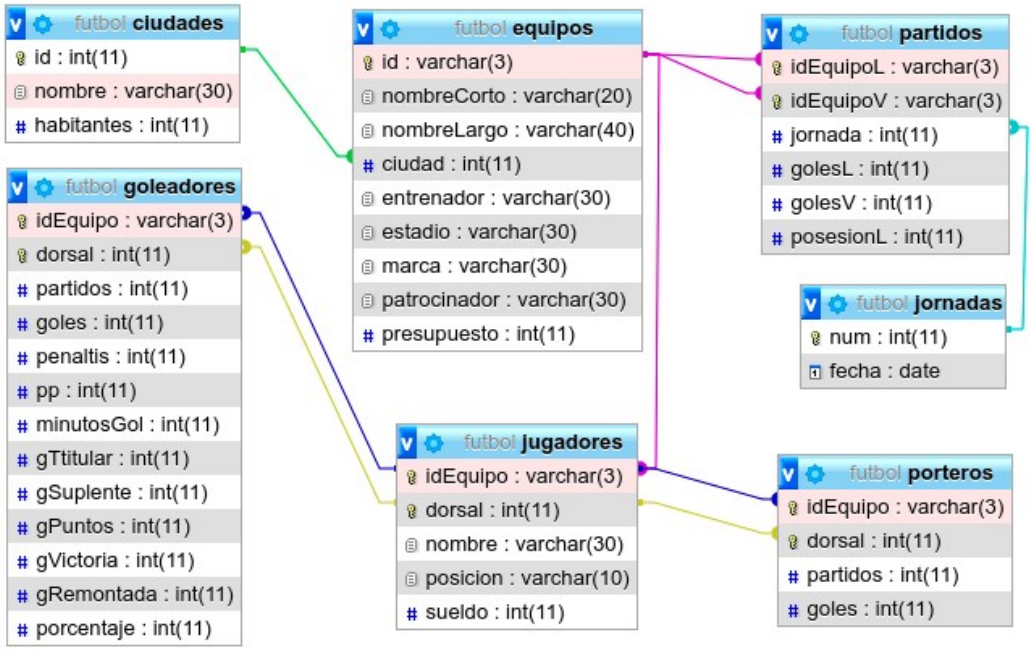


Tablas relacionadas: @OneToOne

Si queremos crear la clase que represente a la tabla “Porteros”.

¿Como mostramos los nombres de los porteros relacionando la tabla Jugadores?





Aclaraciones:

Partidos:

golesL= goles equipo local;
golesV= goles equipo visitante;

Goleadores:

partidos= partidos jugados;
goles= goles marcados;
pp= goles en propia puerta;
minutosGol= minutos para marcar un gol;
gTitular= goles como titular;
gPuntos= goles que dieron puntos;
gVictoria= goles que dieron la victoria;
Porcentaje= porcentaje de goles del equipo;

UD 3: Bases de datos y servicios REST
6.- Consultas relacionadas en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Tablas relacionadas: @OneToOne

Utilizamos el mismo sistema que en la tabla Jugadores para crear la clave primaria compuesta y añadimos la notación @OneToOne para que desde Portero podamos consultar los datos del jugador en el mapper:

PorteroDb.java

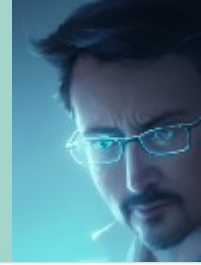
src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > model > db > PorteroDb.java

```
17  @NoArgsConstructor
18  @AllArgsConstructor
19  @Data
20  @Entity
21  @IdClass(JugadorId.class)
22  @Table(name = "porteros")
23  public class PorteroDb implements Serializable{
24      private static final long serialVersionUID = -818542778373595260L;
25      @Id
26      @Size(min=3,message="El id del equipo tiene un tamaño mínimo de 3")
27      private String idEquipo;
28      @Id
29      @Column(nullable = false)
30      private Long dorsal;
31      private Long partidos;
32      private Long goles;
33      @OneToOne
34      @JoinColumn(name="idEquipo", referencedColumnName="idEquipo")
35      @JoinColumn(name="dorsal", referencedColumnName="dorsal")
36      private JugadorDb jugadorDb;
37
38  }
```

PorteroMapper.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > srv > mapper > PorteroMapper.java

```
1   package edu.profesor.joseramon.api_rest_mysql_futbol.srv.mapper;
2
3
4   import java.util.List;
5
6   import org.mapstruct.Mapper;
7   import org.mapstruct.Mapping;
8   import org.mapstruct.factory.Mappers;
9   import edu.profesor.joseramon.api_rest_mysql_futbol.model.db.PorteroDb;
10  import edu.profesor.joseramon.api_rest_mysql_futbol.model.dto.PorteroList;
11
12  @Mapper
13  public interface PorteroMapper {
14      PorteroMapper INSTANCE= Mappers.getMapper(clazz: PorteroMapper.class);
15      @Mapping(target = "nombre", source = "jugadorDb.nombre")
16      PorteroList porteroDbToPorteroList(PorteroDb porterosDb);
17
18      List<PorteroList> porterosDbToPorterosList(List<PorteroDb> porterosDb);
19
20  }
```



Tablas relacionadas: @OneToOne

Nos queda el Servicio y el Controller y podemos probarlo:

PorteroServiceImpl.java

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.srv.impl;
import edu.profesor.joseramon.api_rest_mysql_futbol.model.db.PorteroDb;

@Service
public class PorteroServiceImpl implements PorteroService{
    private final PorteroRepository porteroRepository;

    public PorteroServiceImpl(PorteroRepository porteroRepository){
        this.porteroRepository=porteroRepository;
    }

    @Override
    public PaginaDto<PorteroList> findAll(Pageable paging) {
        Page<PorteroDb> paginaPorteroDb=porteroRepository.findAll(paging);
        return new PaginaDto<PorteroList>(
            paginaPorteroDb.getNumber(),//número de página solicitada
            paginaPorteroDb.getSize(), //tamaño de la página
            paginaPorteroDb.getTotalElements(),//total de elementos devueltos por la consulta sin
            paginaPorteroDb.getTotalPages(), //total páginas teniendo en cuenta el tamaño de cada
                    PorteroMapper.INSTANCE.porterosDbToPorterosList(paginaPorteroDb.getContent()),
                    paginaPorteroDb.getSort()); //ordenación de la consulta
    }

}
```

PorteroRestController.java

```java
@RestController
@RequestMapping("/api/v1/")
public class PorteroRestController {
    private PorteroService porteroService;

    public PorteroRestController(PorteroService porteroService){
        this.porteroService=porteroService;
    }


    @GetMapping("/porteros")
    public ResponseEntity<Map<String, Object>> getAllPorteros(
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "3") int size,
        @RequestParam(defaultValue = "idEquipo,asc") String[] sort) {
```

Response(427ms)

```
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sun, 05 Feb 2023 17:34:06 GMT
Connection: close

{
  "totalItems": 44,
  "data": [
    {
      "idEquipo": "ath",
      "dorsal": 1,
      "nombre": "Gorka Iraizoz"
    },
    {
      "idEquipo": "ath",
      "dorsal": 13,
      "nombre": "Raúl Fernández"
    },
    {
      "idEquipo": "atm",
      "dorsal": 1,
      "nombre": "Sergio Asenjo"
    }
  ],
  "totalPages": 15,
  "pageSize": 3,
  "currentPage": 0
}
```

UD 3: Bases de datos y servicios REST
6.- Consultas relacionadas en Spring Data JPA

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

EJERCICIO:

Sube la aplicación final al moodle.
Para ello:

1º Haz un “Run As \Maven Clean” para dejar solo los fichero fuentes y quitar momentaneamente los necesarios para ejecutar la aplicación (dependencias).
2º Comprime la carpeta de tu aplicación y ponle UD3_practica8_nombreAlumno.tar.gz como nombre al fichero comprimido donde nombreAlumno es el nombre del alumno que entrega la práctica.
3º Súbela al moodle.

IMPORTANTE: No comprimir en RAR, porque Ubuntu no lo lee bien y en clase tenemos Ubuntu. Si tuviésemos Windows, podemos comprimir en ZIP.