Integracion

Carrera Programador full-stack

Read Multitabla

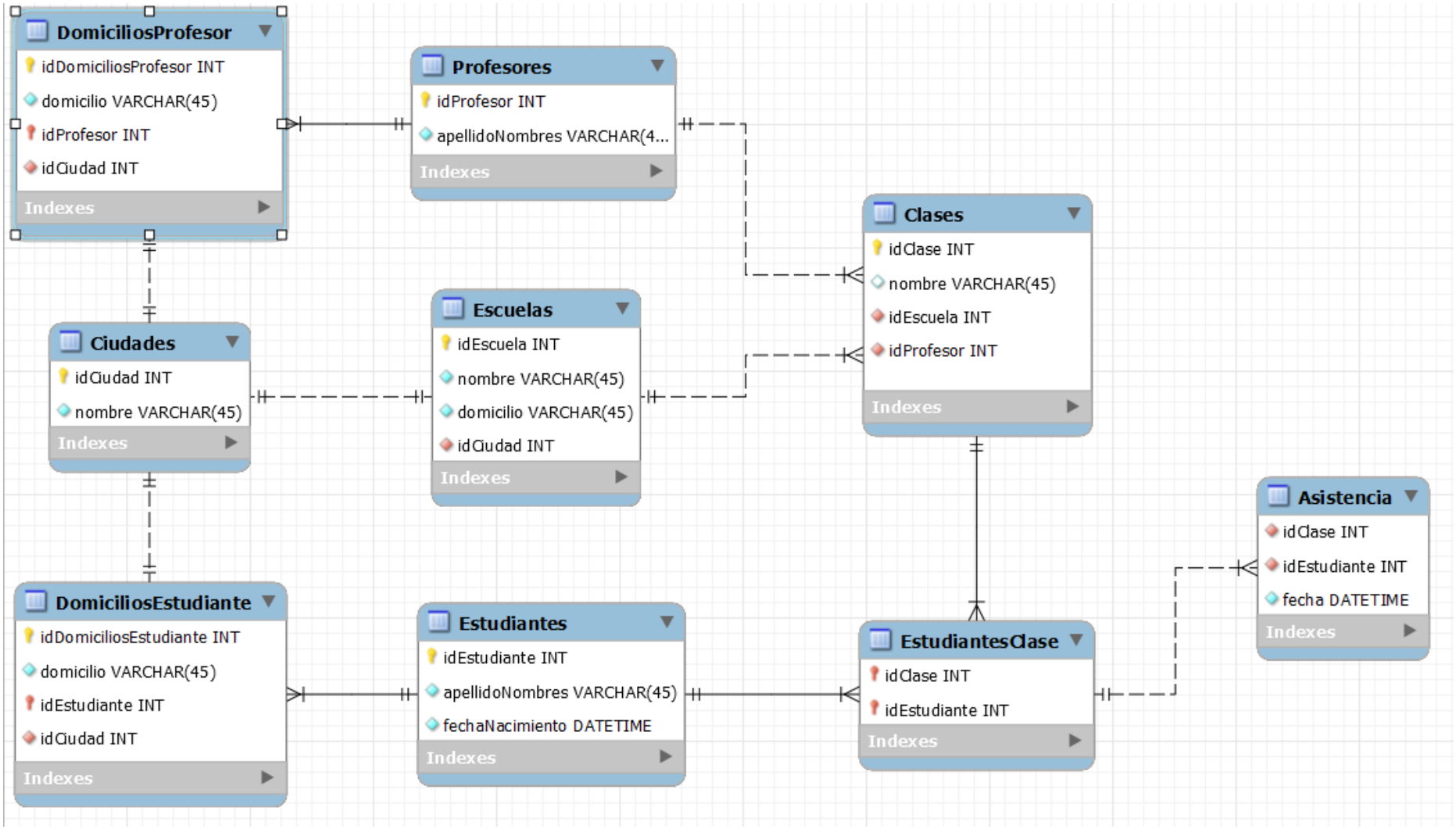
Agenda

- Repaso
  - Create, Read, Update, Delete, ReadAll
- Mapeando tablas con Foreign Key
- One to One
- One to Many
- Many to One
- Many to Many
- Ejercicios

Repaso - CRUD

- Create → POST
  - Dar de alta en la DB
  - Request: Contenido en el body
- Read → GET
  - Consultar en la DB
  - Request: URL Params, Query Params
- Update → PUT
  - Actualizar en la DB
  - Request: Contenido en el body
- Delete → DELETE
  - Eliminar en la DB
  - Request: URL Params, Query Params

Esquema BD


DomiciliosProfesor
idDomiciliosProfesor INT
domicilio VARCHAR(45)
idProfesor INT
idCiudad INT
Indexes
Profesores
idProfesor INT
apellidoNombres VARCHAR(4...
Indexes
Clases
idClase INT
nombre VARCHAR(45)
idEscuela INT
idProfesor INT
Indexes
Ciudades
idCiudad INT
nombre VARCHAR(45)
Indexes
Escuelas
idEscuela INT
nombre VARCHAR(45)
domicilio VARCHAR(45)
idCiudad INT
Indexes
Asistencia
idClase INT
idEstudiante INT
fecha DATETIME
Indexes
DomiciliosEstudiante
idDomiciliosEstudiante INT
domicilio VARCHAR(45)
idEstudiante INT
idCiudad INT
Indexes
Estudiantes
idEstudiante INT
apellidoNombres VARCHAR(45)
fechaNacimiento DATETIME
Indexes
EstudiantesClase
idClase INT
idEstudiante INT
Indexes

Tablas con Foreign Key

- Hasta el momento vimos cómo asociar una clase con una tabla
- Cuando queremos trabajar con mapeos a tablas con relaciones entre si, usamos algunas anotaciones específicas sobre las entities
- Las anotaciones que vamos a usar son
  - @OneToOne
  - @OneToMany
  - @ManyToOne
  - @ManyToMany
  - @JoinColumn
  - @JoinTable

https://typeorm.io/

Relación OneToOne (1)

- Son las relaciones en donde A contiene sólo una instancia de B
- Por ejemplo, si agregamos la entidad Director, una Escuela puede tener solamente uno/a, que a su vez sólo dirige esa Escuela.
- Se usan dos anotaciones
  - @OneToOne
  - @JoinColumn

Relación OneToOne (2)

```
@Entity()
export class Escuela {
  @PrimaryColumn()
  private idEscuela: number;
  @Column()
  private nombre: string;
  @Column()
  private domicilio: string;
  @Column()
  private idCiudad: number;
  …
}
```

```
@Entity()
export class Director {
  @PrimaryColumn()
  private idDirector: number;
  @Column()
  private nombre: string;
  @OneToOne(type => Escuela)
  @JoinColumn()
  public escuela : Escuela;
  …
}
```

La variable *escuela* sería la columna que referencia a Escuela

Relación OneToMany / ManyToOne (1)

- Son las relaciones en donde A contiene más de una instancia de B
- Por ejemplo, una Escuela puede tener varias Clases, pero cada Clase sólo corresponde a una Escuela.
- Se usan tres anotaciones
  - @OneToMany
  - @ManyToOne
  - @JoinColumn

Relación OneToMany / ManyToOne (2)

```
@Entity()
export class Escuela {
  @PrimaryColumn()
  private idEscuela: number;
  @Column()
  private nombre: string;
  @Column()
  private domicilio: string;
  @Column()
  private idCiudad: number;
  @OneToMany(type => Clase,
      clase => clase.escuela)
public clases : Clase[];
  …
}
```

```
@Entity()
export class Clase{
  @PrimaryColumn()
  private idClase: number;
  @Column()
  private nombre: string;
  @ManyToOne(type => Escuela,
      escuela => escuela.clases)
  @JoinColumn()
  public escuela : Escuela;
  @Column()
  private idProfesor: number;
  …
}
```

Una escuela puede tener muchas clases, pero una clase corresponde a una única escuela

Relación OneToMany / ManyToOne (3)

- Se deben relacionar las entidades en el ***module.ts*** de ambas.

clase.module.ts

```
@Module({
        imports: [ TypeOrmModule.forFeature( [ Clase, Escuela ] ) ],
        controllers: [ ClaseController ],
        providers: [ ClaseService ]
})
export class ClaseModule { }
```

escuela.module.ts

```
@Module({
        imports: [ TypeOrmModule.forFeature( [ Escuela, Clase ] ) ],
        controllers: [ EscuelaController ],
        providers: [ EscuelaService ]
})
export class EscuelaModule { }
```

Relación OneToMany / ManyToOne (4)

- Se relacionan en el ***service.ts*** de la entidad One.

escuela.service.ts

```
@Injectable()
export class EscuelaService {
   …
   public async getAll() : Promise<Escuela[]> {
      …
      let criterio : FindManyOptions = { relations: [ 'clases' ] }
      let escuelas : Escuela[] = await this.escuelaRepository.find( criterio );
      return escuelas ;
      …
   }
   …
   public async getById(id : number) : Promise<Escuela> {
      …
      let criterio : FindOneOptions = { relations: [ 'clases' ], where: { idEscuela: id } }
      let escuela : Escuela = await this.escuelaRepository.findOne( criterio );
      return escuela ;
   …
   }
   …
}
```

Agregando estas *FindOptions*, cada escuela recuperada desde la base de datos, trae además las clases que tiene asociadas.

Relación ManyToMany (1)

- Son relaciones en donde A tiene muchas instancias de B, y a la vez B tiene muchas de A
- Por ejemplo: un estudiante asiste a varias clases y una clase tiene varios estudiantes.
- Tener en cuenta que este tipo de relaciones se manifiestan con tres tablas
  - Una para cada entidad. (2)
  - Una tercera con dos FK que asocian cada entity.
- Las anotaciones que se usan son
  - @ManyToMany
  - @JoinTable

Relación ManyToMany (2)

```
@Entity()
export class Estudiante {
  @PrimaryColumn()
  private idEscuela: number;
  @Column()
  private apellidoNombres: string;
  @Column()
  private fechaNacimiento: string;
  @ManyToMany(type => Clase)
  @JoinTable()
  public clases : Clase[];
  …
}
```

```
@Entity()
export class Clase{
  @PrimaryColumn()
  private idClase: number;
  @Column()
  private nombre: string;
  @ManyToOne(type => Escuela,
      escuela => escuela.clases)
  @JoinColumn()
  public idEscuela : Escuela;
  @Column()
  private idProfesor: number;
  …
}
```

@JoinTable puede ponerse en cualquiera de las dos entities

Relación ManyToMany (2)

```
@Entity()
export class Estudiante {
  @PrimaryColumn()
  private idEscuela: number;
  @Column()
  private apellidoNombres: string;
  @Column()
  private fechaNacimiento: string;
  …
}
```

```
@Entity()
export class Clase{
  @PrimaryColumn()
  private idClase: number;
  @Column()
  private nombre: string;
  @ManyToOne(type => Escuela,
      escuela => escuela.clases)
  @JoinColumn()
  public idEscuela : Escuela;
  @Column()
  private idProfesor: number;
  @ManyToMany(type => Estudiante)
  @JoinTable()
  public estudiantes : Estudiante[];
  …
}
```

@JoinTable puede ponerse en cualquiera de las dos entities

Relación ManyToMany (3)

- Se deben relacionar las entidades en el ***module.ts*** de ambas.

clase.module.ts

```
@Module({
        imports: [ TypeOrmModule.forFeature( [ Clase, Estudiante ] ) ],
        controllers: [ ClaseController ],
        providers: [ ClaseService ]
})
export class ClaseModule { }
```

estudiante.module.ts

```
@Module({
        imports: [ TypeOrmModule.forFeature( [ Estudiante, Clase ] ) ],
        controllers: [ EstudianteController ],
        providers: [ EstudianteService ]
})
export class EstudianteModule { }
```

Relación ManyToMany (4)

- Se relacionan en el ***service.ts*** de cada entidad.

estudiante.service.ts

```
@Injectable()
export class EstudianteService {
   …
   public async getAll() : Promise<Estudiante[]> {
      …
      let criterio : FindManyOptions = { relations: [ 'clases' ] }
      let estudiantes : Estudiante[] = await this.estudianteRepository.find( criterio );
      return estudiantes ;
      …
   }
   …
   public async getById(id : number) : Promise<Estudiante> {
      …idEstudiante:
      let criterio : FindOneOptions = { relations: [ 'clases' ], where: {id } }
      let estudiante: Estudiante= await this.estudianteRepository.findOne( criterio );
      return estudiante;
   …
   }
   …
}
```

Agregando estas *FindOptions*, cada estudiante recuperado desde la base de datos, trae además las clases que tiene asociadas.

Relación ManyToMany (4)

- Se relacionan en el ***service.ts*** de cada entidad.

clase.service.ts

```
@Injectable()
export class ClaseService {
   …
   public async getAll() : Promise<Clase[]> {
      …
      let criterio : FindManyOptions = { relations: [ 'estudiantes' ] }
      let clases: Clase[] = await this.claseRepository.find( criterio );
      return clases ;
      …
   }
   …
   public async getById(id : number) : Promise<Clase> {
      …
      let criterio : FindOneOptions = { relations: [ 'estudiantes' ], where: { idClase: id } }
      let clase: Clase= await this.claseRepository.findOne( criterio );
      return clase;
   …
   }
   …
}
```

Agregando estas *FindOptions*, cada clase recuperada desde la base de datos, trae además los estudiantes que tiene asociados.