**Relaciones entre entidades**

**1. ¿Cómo se relacionan las empresas con sus empleados?**

* **En la entidad Company se encuentra un @OneToMany(() => Employee, e => e.company) que expone la propiedad departamentos: Employee[].**
* **En la entidad Employee hay un @ManyToOne(() => Company, c => c.empleados) y la columna de clave foránea empresa\_id.**
* **Esto crea una relación 1 empresa → N empleados, donde cada empleado guarda empresaId y al borrar (o soft-delete) una empresa, sus empleados se eliminan en cascada.**

**¿Un departamento puede existir sin empleados?**

* **Sí. La entidad Department define un @OneToMany(() => Employee, e => e.department) pero no obliga a tener empleados asignados.**
* **La columna depto\_id vive en Employee, no en Department, por lo que puedes crear departamentos vacíos.**
* **Para borrar un departamento con empleados, se configuro onDelete: 'RESTRICT', así que solo podrás eliminarlo cuando no tenga empleados.**

**¿Un empleado puede cambiar de departamento? ¿Cómo manejarías el historial?**

* **El campo deptoId en Employee se actualiza para reflejar el nuevo departamento.**
* **Para trackear esos movimientos, se creó la entidad EmployeeDeptHistory con:** 
  + **empleadoId, deptoId, desde y hasta.**
  + **Relaciones ManyToOne a Employee y Department.**
* **En el servicio:** 
  + **Antes de cambiar deptoId, se guarda un registro en EmployeeDeptHistory con el antiguo deptoId y fecha desde.**
  + **Al liberar o volver a asignar, se actualiza el mismo registro marcando la fecha hasta.**
* **Así se mantiene un log completo de asignaciones por empleado sin perder datos históricos.**

**2. Integridad de datos**

**Restricciones de unicidad**

* **En la tabla empresas:** 
  + **email debe ser único (cada empresa se registra con un email distinto).**
* **En la tabla departamentos:** 
  + **combinación (empresa\_id, nombre) única para evitar nombres duplicados dentro de la misma empresa.**
* **En la tabla empleados:** 
  + **composición (empresa\_id, email) única para que un mismo email no se repita dentro de la misma empresa.**

**Campos obligatorios vs opcionales**

* **Empresa:** 
  + **Obligatorios: nombre, email, passwordHash.**
  + **Opcionales: deletedAt (soft-delete), updatedAt.**
* **Departamento:** 
  + **Obligatorios: empresaId, nombre.**
  + **Opcionales: descripción, deletedAt, updatedAt.**
* **Empleado:** 
  + **Obligatorios: empresaId, deptoId, nombre, apellido, email, estado, fechaContratacion.**
  + **Opcionales: deletedAt, updatedAt.**

**Aislamiento por empresa**

* **Cada consulta en services filtra por empresaId extraído del token JWT.**
* **El decorador @GetCompany() inyecta companyId en los controladores.**
* **Todos los find, update y delete usan { where: { empresaId, … } } para evitar fugas de datos.**

**2. Escalabilidad**

**Índices recomendados**

* **Índice simple sobre la columna empresa\_id en empleados y en departamentos.**
* **Índice compuesto sobre (empresa\_id, email) en empleados para búsquedas y unicidad.**
* **Índice sobre depto\_id en empleados para filtrar rápidamente por departamento.**
* **Índice sobre fecha\_contratacion en empleados si se van a consultar rangos de fecha.**

**Optimización de consultas de reportes**

* **Usar QueryBuilder con filtros dinámicos y paginación (skip/take) para evitar cargas masivas.**
* **En reportes agregados, usar getRawMany() con GROUP BY en lugar de traer todas las entidades.**
* **Considerar vistas materializadas o tablas de resumen si los reportes requieren datos muy históricos.**
* **Evitar N+1: siempre incluir relaciones necesarias mediante leftJoinAndSelect.**

**3. Auditoría**

**Timestamps de creación/actualización**

* **Se usan @CreateDateColumn y @UpdateDateColumn en todas las entidades principales.**
* **Permiten rastrear cuándo se crearon o modificaron cada departamento, empleado o empresa.**

**Eliminaciones lógicas vs físicas**

* **Se utiliza @DeleteDateColumn para soft-deletes: marca un registro como eliminado sin borrarlo físicamente.**
* **El soft-delete preserva historial y claves foráneas.**
* **Para limpieza definitiva, programar tareas (cron jobs) que purguen registros donde deletedAt supere cierto umbral (por ejemplo, 90 días).**

**Diagrama

El contenido generado por IA puede ser incorrecto.**

**Tablas de la base de datos**

**empresas**

| **Columna** | **Tipo** | **Restricciones** | **Descripción** |
| --- | --- | --- | --- |
| empresa\_id | int | PK, AUTO\_INCREMENT | Identificador único de empresa |
| nombre | varchar(255) | NOT NULL | Nombre de la empresa |
| email | varchar(255) | NOT NULL, UNIQUE | Correo electrónico de la empresa |
| password\_hash | varchar(255) | NOT NULL | Hash de la contraseña |
| created\_at | timestamp | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Fecha de creación |
| updated\_at | timestamp | NOT NULL, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP | Fecha de última modificación |
| deleted\_at | timestamp | NULLABLE | Fecha de eliminación lógica |

**departamentos**

| **Columna** | **Tipo** | **Restricciones** | **Descripción** |
| --- | --- | --- | --- |
| depto\_id | int | PK, AUTO\_INCREMENT | Identificador único de departamento |
| empresa\_id | int | NOT NULL, FK → empresas | Empresa propietaria |
| nombre | varchar(255) | NOT NULL | Nombre del departamento |
| descripcion | text | NULLABLE | Descripción opcional |
| created\_at | timestamp | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Fecha de creación |
| updated\_at | timestamp | NOT NULL, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP | Fecha de última modificación |
| deleted\_at | timestamp | NULLABLE | Fecha de eliminación lógica |

**empleados**

| **Columna** | **Tipo** | **Restricciones** | **Descripción** |
| --- | --- | --- | --- |
| empleado\_id | int | PK, AUTO\_INCREMENT | Identificador único de empleado |
| empresa\_id | int | NOT NULL, FK → empresas | Empresa a la que pertenece |
| depto\_id | int | NOT NULL, FK → departamentos | Departamento asignado |
| nombre | varchar(255) | NOT NULL | Nombre del empleado |
| apellido | varchar(255) | NOT NULL | Apellido del empleado |
| email | varchar(255) | NOT NULL, (empresa\_id, email) UNIQUE | Correo del empleado, único por empresa |
| estado | enum | NOT NULL (activo, inactivo) | Estado laboral |
| fecha\_contratacion | date | NOT NULL | Fecha de contratación |
| created\_at | timestamp | NOT NULL, DEFAULT CURRENT\_TIMESTAMP | Fecha de creación |
| updated\_at | timestamp | NOT NULL, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP | Fecha de última modificación |
| deleted\_at | timestamp | NULLABLE | Fecha de eliminación lógica |

**empleado\_depto\_historial**

| **Columna** | **Tipo** | **Restricciones** | **Descripción** |
| --- | --- | --- | --- |
| historial\_id | int | PK, AUTO\_INCREMENT | Identificador único del registro de historial |
| empleado\_id | int | NOT NULL, FK → empleados | Empleado asignado |
| depto\_id | int | NOT NULL, FK → departamentos | Departamento al que fue asignado |
| desde | date | NOT NULL | Fecha de inicio de la asignación |
| hasta | date | NULLABLE | Fecha de fin de la asignación (si aplica) |

**Consulta para creación de las tablas**

-- Tabla empresas

CREATE TABLE empresas (

empresa\_id INT AUTO\_INCREMENT PRIMARY KEY,

nombre VARCHAR(255) NOT NULL,

email VARCHAR(255) NOT NULL UNIQUE,

password\_hash VARCHAR(255) NOT NULL,

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP NOT NULL

DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

deleted\_at TIMESTAMP NULL DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Tabla departamentos

CREATE TABLE departamentos (

depto\_id INT AUTO\_INCREMENT PRIMARY KEY,

empresa\_id INT NOT NULL,

nombre VARCHAR(255) NOT NULL,

descripcion TEXT NULL,

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP NOT NULL

DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

deleted\_at TIMESTAMP NULL DEFAULT NULL,

UNIQUE KEY uniq\_depto\_empresa\_nombre (empresa\_id, nombre),

INDEX idx\_departamentos\_empresa (empresa\_id),

CONSTRAINT fk\_departamentos\_empresa

FOREIGN KEY (empresa\_id)

REFERENCES empresas(empresa\_id)

ON DELETE CASCADE

ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Tabla empleados

CREATE TABLE empleados (

empleado\_id INT AUTO\_INCREMENT PRIMARY KEY,

empresa\_id INT NOT NULL,

depto\_id INT NOT NULL,

nombre VARCHAR(255) NOT NULL,

apellido VARCHAR(255) NOT NULL,

email VARCHAR(255) NOT NULL,

estado ENUM('activo','inactivo') NOT NULL DEFAULT 'activo',

fecha\_contratacion DATE NOT NULL,

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP NOT NULL

DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

deleted\_at TIMESTAMP NULL DEFAULT NULL,

UNIQUE KEY uniq\_empleados\_empresa\_email (empresa\_id, email),

INDEX idx\_empleados\_empresa (empresa\_id),

INDEX idx\_empleados\_depto (depto\_id),

INDEX idx\_empleados\_fecha (fecha\_contratacion),

CONSTRAINT fk\_empleados\_empresa

FOREIGN KEY (empresa\_id)

REFERENCES empresas(empresa\_id)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT fk\_empleados\_depto

FOREIGN KEY (depto\_id)

REFERENCES departamentos(depto\_id)

ON DELETE RESTRICT

ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Tabla empleado\_depto\_historial

CREATE TABLE empleado\_depto\_historial (

historial\_id INT AUTO\_INCREMENT PRIMARY KEY,

empleado\_id INT NOT NULL,

depto\_id INT NOT NULL,

desde DATE NOT NULL,

hasta DATE NULL,

INDEX idx\_historial\_empleado (empleado\_id),

INDEX idx\_historial\_depto (depto\_id),

CONSTRAINT fk\_historial\_empleado

FOREIGN KEY (empleado\_id)

REFERENCES empleados(empleado\_id)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT fk\_historial\_depto

FOREIGN KEY (depto\_id)

REFERENCES departamentos(depto\_id)

ON DELETE RESTRICT

ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

1. ¿Por qué elegiste esta estructura de base de datos? por que me da la ventaja de escalar los datos y tener un control mas preciso de ellos,me permite llevar un historial mas claro de los datos.

2. ¿Cómo manejarías el crecimiento a millones de empleados? con tablas relacionales agupando empleados por departamentos empresas y realizando filtros mas consisos .

3. ¿Qué optimizaciones implementarías para mejorar performance? optimizar las consultas para mayor presicion en la manipulacion de los datos.

4. ¿Qué estrategia usarías para el manejo de errores y seguimiento de los mismos? debug y logger.