

Vale, he conseguido reordenar el código y hacer que todo siga funcionando. Volvamos a lo de las migraciones. Como las habilito, como las creo y como las pongo en marcha?

¡Perfecto! Vamos a configurar las migraciones en tu proyecto NestJS con TypeORM para Oracle. Aquí tienes la guía completa:

❑ Configuración de Migraciones (Oracle + TypeORM)

1. Habilitar migraciones (en `typeorm-config.ts`):

typescript

```
export const databaseConfig = (config: ConfigService):
typeormModuleOptions => ({
  // ... resto de tu configuración actual
  migrations: [__dirname + '/../migrations/*{.ts,.js}'],
  migrationsTableName: 'typeorm_migrations',
  migrationsRun: false, // ❑ Cambiar a true para ejecutar
  automáticamente al iniciar
  cli: {
    migrationsDir: 'src/migrations'
  }
});
```

2. Crear estructura de migraciones:

bash

```
mkdir -p src/migrations
```

3. Scripts para `package.json`:

json

```
"scripts": {
  "migration:generate": "typeorm-ts-node-commonjs migration:generate -
d src/database/typeorm-config.ts src/migrations/NombreDescriptivo",
  "migration:run": "typeorm-ts-node-commonjs migration:run -d
src/database/typeorm-config.ts",
  "migration:revert": "typeorm-ts-node-commonjs migration:revert -d
src/database/typeorm-config.ts"
}
```

4. Crear tu primera migración:

bash

```
npm run migration:generate -- CreateUsersTable
```

Esto generará un archivo como `src/migrations/12345678-CreateUsersTable.ts` con estructura básica.

5. Ejemplo de migración para Oracle:

typescript

```
import { MigrationInterface, QueryRunner } from 'typeorm';

export class CreateUsersTable12345678 implements MigrationInterface {
```

```

public async up(queryRunner: QueryRunner): Promise<void> {
  await queryRunner.query(`
    CREATE TABLE USERS (
      ID NUMBER GENERATED ALWAYS AS IDENTITY,
      USERNAME VARCHAR2(50) NOT NULL,
      PASSWORD VARCHAR2(100) NOT NULL,
      CONSTRAINT PK_USERS PRIMARY KEY (ID)
    )
  `);
}

public async down(queryRunner: QueryRunner): Promise<void> {
  await queryRunner.query(`DROP TABLE USERS`);
}
}

```

❑ Ejecutar migraciones:

bash

```
# Ejecutar todas las migraciones pendientes
npm run migration:run
```

```
# Revertir la última migración
npm run migration:revert
```

❑ Verificación:

1. **Tabla de control:** TypeORM creará una tabla `typeorm_migrations` en tu Oracle.
2. **Logs:** Verás los queries ejecutados en la consola.

❑ Tips para Oracle:

1. **Secuencias manuales:**

typescript

```
await queryRunner.query(`CREATE SEQUENCE USER_ID_SEQ START WITH 1
INCREMENT BY 1`);
```

2. **Schema específico:**

typescript

```
await queryRunner.query(`CREATE TABLE ${schema}.USERS (...)`);
```

3. **Problemas comunes:**

- Si fallan las migraciones, verifica que el usuario tenga permisos GRANT
- Para tablas grandes, considera dividir migraciones en lotes

¿Necesitas ayuda para crear alguna migración específica o resolver algún error en particular? ❑

This response is AI-generated, for reference only.

Como vecngo de laravel he pensado que estos scripts pueden ser más cómodos:

```
"typeorm make:migration": "typeorm-ts-node-commonjs migration:generate -d
src/database/typeorm-config.ts src/migrations/NombreDescriptivo", "typeorm migrate":
"typeorm-ts-node-commonjs migration:run -d src/database/typeorm-config.ts",
"typeorm migrate:rollback": "typeorm-ts-node-commonjs migration:revert -d
src/database/typeorm-config.ts"
```

¡Me encanta la idea! Adaptar los comandos para que sean similares a Laravel hace todo más intuitivo. Aquí tienes la versión optimizada con tus nombres preferidos y algunas mejoras adicionales:

❑ Scripts tipo Laravel para TypeORM (package.json)

json

```
"scripts": {
  "typeorm:make": "typeorm-ts-node-commonjs migration:generate -d
src/database/typeorm-config.ts src/migrations/",
  "typeorm:migrate": "typeorm-ts-node-commonjs migration:run -d
src/database/typeorm-config.ts",
  "typeorm:rollback": "typeorm-ts-node-commonjs migration:revert -d
src/database/typeorm-config.ts",
  "typeorm:refresh": "npm run typeorm:rollback && npm run
typeorm:migrate",
  "typeorm:status": "typeorm-ts-node-commonjs migration:show -d
src/database/typeorm-config.ts"
}
```

❑ Uso mejorado (con ejemplos)

bash

```
# Crear migración (similar a Laravel)
npm run typeorm:make -- CreateUsersTable
```

```
# Ejecutar migraciones
npm run typeorm:migrate
```

```
# Revertir última migración
npm run typeorm:rollback
```

```
# Refrescar toda la base de datos
npm run typeorm:refresh
```

```
# Ver estado de migraciones
npm run typeorm:status
```

❑ Mejoras adicionales:

1. Generador automático de nombres (opcional):

json

```
• "typeorm:make": "typeorm-ts-node-commonjs migration:generate -d
src/database/typeorm-config.ts src/migrations/${date +%Y%m%d%H%M%S}_",
```

Ahora genera nombres como 20231120142315_CreateUsersTable.ts.

- **Variables de entorno para Oracle:**

bash

- # Ejecutar con modo debug
DEBUG=typeorm:* npm run typeorm:migrate

- **Alias globales** (opcional en bash/zsh):

bash

```
3. alias artisan="npm run typeorm"
4. # Ahora puedes usar:
5. artisan make CreateProductsTable
```

□ □ Notas importantes:

1. El guión bajo (_) en nombres de migración es opcional pero recomendado
2. Para Oracle, considera añadir este parámetro a tu configuración:

typescript

```
2. extra: {
3.   maxRows: 1000 // Evita timeouts en tablas grandes
4. }
```

¿Quieres que adaptemos algún otro comando específico de Laravel? Por ejemplo, podríamos crear un `typeorm:seed` para los seeders □.

Ejemplo de migración con sintaxis Oracle:

typescript

```
export class CreateProductsTable implements MigrationInterface {
  public async up(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.query(`
      CREATE TABLE PRODUCTS (
        ID NUMBER GENERATED AS IDENTITY,
        NAME VARCHAR2(255) NOT NULL,
        PRICE NUMBER(10,2) CHECK (PRICE >= 0),
        CREATED_AT TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        CONSTRAINT PK_PRODUCTS PRIMARY KEY (ID)
      )
    `);
    await queryRunner.query(`CREATE INDEX IDX_PRODUCT_NAME ON PRODUCTS
(NAME)`);
  }
}
```

This response is AI-generated, for reference only.

"typeorm make:migration": "typeorm-ts-node-commonjs migration:generate -d src/database/typeorm-config.ts src/migrations/NombreDescriptivo", "typeorm migrate": "typeorm-ts-node-commonjs migration:run -d src/database/typeorm-config.ts", "typeorm migrate:rollback": "typeorm-ts-node-commonjs migration:revert -d src/database/typeorm-config.ts", "typeorm migrate:fresh": "npm run typeorm:rollback && npm run typeorm:migrate", "typeorm:status": "typeorm-ts-node-commonjs migration:show -d src/database/typeorm-config.ts" así lo he dejado

¡Perfecto! ☐ Has creado un excelente set de comandos al estilo Laravel para manejar migraciones en TypeORM. Aquí tienes una evaluación y algunos consejos para sacarles el máximo provecho:

☐ **Tu configuración actual (mejorada)**

json

```
"scripts": {
  "typeorm make:migration": "typeorm-ts-node-commonjs
migration:generate -d src/database/typeorm-config.ts
src/migrations/NombreDescriptivo",
  "typeorm migrate": "typeorm-ts-node-commonjs migration:run -d
src/database/typeorm-config.ts",
  "typeorm migrate:rollback": "typeorm-ts-node-commonjs
migration:revert -d src/database/typeorm-config.ts",
  "typeorm migrate:fresh": "npm run typeorm migrate:rollback && npm
run typeorm migrate",
  "typeorm:status": "typeorm-ts-node-commonjs migration:show -d
src/database/typeorm-config.ts"
}
```

☐ **Cómo usar estos comandos:**

1. **Crear migración** (con nombre descriptivo):

bash

- `npm run typeorm make:migration -- CreateUsersTable`

Nota: El -- es crucial para pasar argumentos

• **Ejecutar migraciones:**

bash

- `npm run typeorm migrate`

• **Revertir última migración:**

bash

- `npm run typeorm migrate:rollback`

• **Resetear completamente:**

bash

- `npm run typeorm migrate:fresh`

• **Ver estado:**

bash

- 5. `npm run typeorm:status`

☐ **Versión alternativa (con timestamps automáticos)**

Si prefieres nombres con timestamp como Laravel:

json

```
"typeorm make:migration": "typeorm-ts-node-commonjs migration:generate  
-d src/database/typeorm-config.ts src/migrations/$(date  
+%Y%m%d%H%M%S)_"
```

Generará: src/migrations/20231120120000_CreateUsersTable.ts

❑ Consejos para Oracle:

1. **En tus migraciones, usa sintaxis específica:**

typescript

- ```
await queryRunner.query(`
CREATE TABLE USERS (
 ID NUMBER GENERATED ALWAYS AS IDENTITY,
 NAME VARCHAR2(255) NOT NULL,
 CONSTRAINT PK_USERS PRIMARY KEY (ID)
)
`);
```

- **Para tablas grandes** añade esto a tu configuración:

typescript

- ```
extra: {  
    poolMax: 20,  
    statementTimeout: 60000  
}
```

- **Si fallan las migraciones, depura con:**

bash

```
3. DEBUG=typeorm:* npm run typeorm migrate
```

❑❑ Posible error y solución:

Si ves *"Cannot use import statement outside a module"*, añade esto a tu package.json:

json

```
"type": "module"
```

¿Quieres que agreguemos algún comando adicional como `typeorm migrate:refresh` (rollback + migrate) o algún otro que usabas en Laravel? ❑

Ejemplo de migración compleja para Oracle:

typescript

```
export class AddIndexesToUsersTable implements MigrationInterface {  
    public async up(queryRunner: QueryRunner): Promise<void> {  
        await queryRunner.query(`  
            CREATE INDEX IDX_USERS_EMAIL ON USERS (EMAIL)  
        `);  
    }  
}
```

```

        TABLESPACE USERS_DATA
    `);

    await queryRunner.query(`
        ALTER TABLE USERS
        ADD CONSTRAINT UC_USERS_EMAIL UNIQUE (EMAIL)
        USING INDEX TABLESPACE USERS_INDEX
    `);
}
}

```

This response is AI-generated, for reference only.

pensaba que migrate fresh lo que haría sería limpiar las migraciones y luego volver a hacerlas

¡Tienes toda la razón! El comando `migrate:fresh` debería **eliminar completamente todas las tablas** (no solo revertir migraciones) y luego ejecutarlas desde cero. Así es como lo haríamos al estilo Laravel:

❑ Comando `migrate:fresh` mejorado (para Oracle)

json

```

"scripts": {
  "typeorm schema:drop": "typeorm-ts-node-commonjs schema:drop -d src/database/typeorm-config.ts",
  "typeorm migrate:fresh": "npm run typeorm schema:drop && npm run typeorm migrate",
  "typeorm migrate:refresh": "npm run typeorm migrate:rollback && npm run typeorm migrate"
}

```

❑ Diferencia clave:

Comando	Acción
<code>migrate:fresh</code>	Elimina TODAS las tablas y ejecuta migraciones desde cero (como <code>php artisan migrate:fresh</code>)
<code>migrate:refresh</code>	Solo revierte y re-ejecuta las migraciones (menos destructivo)

❑ Uso:

bash

```

# Nuclear: Elimina todo y recrea la estructura
npm run typeorm migrate:fresh

```

```

# Suave: Solo reinicia las migraciones
npm run typeorm migrate:refresh

```

❑❑ Importante para Oracle:

1. **Permisos necesarios:**
 - El usuario de DB necesita permisos `DROP ANY TABLE`
 - Considera usar un usuario con derechos DBA para desarrollo
2. **Configuración segura** (en `typeorm-config.ts`):

typescript

```
extra: {  
  poolMax: 1 // Evita deadlocks al eliminar tablas  
}
```

❑ Ejemplo de migración resistente:

typescript

```
export class CreateUsersTable implements MigrationInterface {  
  public async up(queryRunner: QueryRunner): Promise<void> {  
    const tableExists = await queryRunner.hasTable('USERS');  
    if (!tableExists) {  
      await queryRunner.query(`  
        CREATE TABLE USERS (  
          ID NUMBER GENERATED ALWAYS AS IDENTITY,  
          NAME VARCHAR2(255) NOT NULL,  
          CONSTRAINT PK_USERS PRIMARY KEY (ID)  
        )  
      `);  
    }  
  }  
}
```

❑ ¿Quieres más comandos al estilo Laravel?

Podríamos agregar:

json

```
"typeorm db:seed": "ts-node src/seeder/main.ts",  
"typeorm migrate:fresh --seed": "npm run typeorm migrate:fresh && npm  
run typeorm db:seed"
```

¿Te gustaría que implementemos los seeders también? ❑ Puedo mostrarte cómo estructurarlos para Oracle.

This response is AI-generated, for reference only.

Crear una nueva migración: Para crear una nueva migración, utiliza el siguiente comando. Reemplaza nombre_de_la_migracion con el nombre que desees para tu migración. `bash php artisan make:migration nombre_de_la_migracion` Ejemplo: `bash php artisan make:migration create_users_table` Ejecutar las migraciones: Para ejecutar todas las migraciones pendientes, utiliza: `bash php artisan migrate` Revertir la última migración: Si deseas deshacer la última migración ejecutada, utiliza: `bash php artisan migrate:rollback` Revertir todas las migraciones: Para deshacer todas las migraciones y volver a la estructura inicial de la base de datos, utiliza: `bash php artisan migrate:reset` Revertir y volver a ejecutar todas las migraciones: Si deseas revertir todas las migraciones y luego volver a ejecutarlas, utiliza: `bash php artisan migrate:refresh` Ejecutar migraciones específicas: Si deseas ejecutar una migración específica, puedes usar el siguiente comando: `bash php artisan migrate --path=/database/migrations/nombre_de_la_migracion.php` Ver el estado de las migraciones: Para ver el estado de las migraciones (qué migraciones han sido ejecutadas y cuáles no), utiliza: `bash php artisan migrate:status` Forzar la migración en producción: Si estás en un entorno de producción y deseas forzar la migración, puedes usar: `bash php artisan migrate --force` Crear una migración con una tabla existente: Si deseas crear

una migración para una tabla que ya existe, puedes usar el siguiente comando: `bash php artisan make:migration nombre_de_l` ESTAS SON LAS OPCIONES DE LARAVEL, CREAME LAS MISMAS PARA AQUÍ

¡Vamos a crear el equivalente exacto de los comandos de migración de Laravel para tu proyecto con TypeORM! Aquí tienes la configuración completa:

❑ Equivalente TypeORM de los comandos de Laravel (package.json)

json

```
"scripts": {
  // Creación de migraciones
  "migrate:make": "typeorm-ts-node-commonjs migration:generate -d
src/database/typeorm-config.ts src/migrations/",
  "migrate:make:create": "typeorm-ts-node-commonjs migration:create
src/migrations/",

  // Ejecución de migraciones
  "migrate": "typeorm-ts-node-commonjs migration:run -d
src/database/typeorm-config.ts",
  "migrate:force": "typeorm-ts-node-commonjs migration:run -d
src/database/typeorm-config.ts --force",
  "migrate:specific": "typeorm-ts-node-commonjs migration:run -d
src/database/typeorm-config.ts -- -f src/migrations/",

  // Reversión
  "migrate:rollback": "typeorm-ts-node-commonjs migration:revert -d
src/database/typeorm-config.ts",
  "migrate:reset": "typeorm-ts-node-commonjs migration:revert -d
src/database/typeorm-config.ts --all",

  // Refresh y estado
  "migrate:refresh": "npm run migrate:reset && npm run migrate",
  "migrate:fresh": "typeorm-ts-node-commonjs schema:drop -d
src/database/typeorm-config.ts && npm run migrate",
  "migrate:status": "typeorm-ts-node-commonjs migration:show -d
src/database/typeorm-config.ts",

  // Para tablas existentes
  "migrate:make:table": "typeorm-ts-node-commonjs migration:create -t
src/migrations/"
}
```

❑ Guía de Uso Completa

1. Creación de migraciones

bash

```
# Como php artisan make:migration
npm run migrate:make CreateUsersTable
```

```
# Para tabla existente (como --table)
npm run migrate:make:table AlterUsersTable
```

2. Ejecución

bash

```
# Ejecutar todas (php artisan migrate)
npm run migrate

# Forzar en producción
npm run migrate:force

# Ejecutar específica
npm run migrate:specific 20231120120000-CreateUsersTable.ts
```

3. Reversión

```
bash
# Revertir última (php artisan migrate:rollback)
npm run migrate:rollback

# Revertir todas (php artisan migrate:reset)
npm run migrate:reset
```

4. Operaciones avanzadas

```
bash
# Refresh (php artisan migrate:refresh)
npm run migrate:refresh

# Fresh (php artisan migrate:fresh)
npm run migrate:fresh

# Ver estado
npm run migrate:status
```

❑ Configuración Adicional para Oracle

Añade esto a tu `typeorm-config.ts`:

```
typescript
export const databaseConfig: TypeOrmModuleOptions = {
  // ... otras configs
  migrationsTransactionMode: 'each', // Para Oracle
  extra: {
    poolMax: 10,
    poolMin: 2,
    queueTimeout: 60000
  }
}
```

❑ Ejemplo de Migración para Oracle

```
typescript
// src/migrations/20231120120000-CreateUsersTable.ts
import { MigrationInterface, QueryRunner } from 'typeorm';

export class CreateUsersTable implements MigrationInterface {
  public async up(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.query(`
      CREATE TABLE USERS (
        ID NUMBER GENERATED ALWAYS AS IDENTITY,
        USERNAME VARCHAR2(50) NOT NULL,
```

```

        PASSWORD VARCHAR2(100) NOT NULL,
        CREATED_AT TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        CONSTRAINT PK_USERS PRIMARY KEY (ID)
    ) TABLESPACE USERS_DATA
`);
}

public async down(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.query(`DROP TABLE USERS INCLUDING CONSTRAINTS`);
}
}

```

□ □ Notas importantes:

1. Para Oracle, siempre usa `INCLUDING CONSTRAINTS` en los `DROP`
2. El parámetro `--force` no es exactamente igual que en Laravel (TypeORM no tiene confirmación en producción)
3. Las migraciones específicas requieren la ruta completa al archivo

¿Quieres que adaptemos algún otro comando específico de Laravel o necesitas ejemplos más complejos para Oracle? □

This response is AI-generated, for reference only.

```

"typeorm make:migration": "typeorm-ts-node-commonjs migration:generate -d
src/database/typeorm-config.ts src/migrations/NombreDescriptivo", "typeorm migrate":
"typeorm-ts-node-commonjs migration:run -d src/database/typeorm-config.ts",
"typeorm migrate:force": "typeorm-ts-node-commonjs migration:run -d
src/database/typeorm-config.ts --force", "typeorm migrate:specific": "typeorm-ts-node-
commonjs migration:run -d src/database/typeorm-config.ts -- -f src/migrations/",
"typeorm migrate:rollback": "typeorm-ts-node-commonjs migration:revert -d
src/database/typeorm-config.ts", "typeorm migrate:reset": "typeorm-ts-node-commonjs
migration:revert -d src/database/typeorm-config.ts --all", "typeorm migrate:refresh":
"npm run migrate:reset && npm run migrate", "typeorm migrate:fresh": "typeorm-ts-
node-commonjs schema:drop -d src/database/typeorm-config.ts && npm run migrate",
"typeorm migrate:status": "typeorm-ts-node-commonjs migration:show -d
src/database/typeorm-config.ts" dime si me falta algo por añadir

```

¡Tu configuración de comandos para migraciones con TypeORM está muy completa! □
 Pero hay algunos ajustes y adiciones que podemos hacer para cubrir todos los
 escenarios y mejorar la usabilidad:

□ Análisis de tu configuración actual:

```

json
"scripts": {
    "typeorm make:migration": "typeorm-ts-node-commonjs
migration:generate -d src/database/typeorm-config.ts
src/migrations/NombreDescriptivo",
    "typeorm migrate": "typeorm-ts-node-commonjs migration:run -d
src/database/typeorm-config.ts",
    "typeorm migrate:force": "typeorm-ts-node-commonjs migration:run -d
src/database/typeorm-config.ts --force",
    "typeorm migrate:specific": "typeorm-ts-node-commonjs migration:run
-d src/database/typeorm-config.ts -- -f src/migrations/",

```

```

    "typeorm migrate:rollback": "typeorm-ts-node-commonjs
migration:revert -d src/database/typeorm-config.ts",
    "typeorm migrate:reset": "typeorm-ts-node-commonjs migration:revert
-d src/database/typeorm-config.ts --all",
    "typeorm migrate:refresh": "npm run migrate:reset && npm run
migrate",
    "typeorm migrate:fresh": "typeorm-ts-node-commonjs schema:drop -d
src/database/typeorm-config.ts && npm run migrate",
    "typeorm migrate:status": "typeorm-ts-node-commonjs migration:show -
d src/database/typeorm-config.ts"
}

```

❑ Mejoras y correcciones necesarias:

1. Corrección en `make:migration`:

diff

```

• - "typeorm make:migration": "typeorm-ts-node-commonjs
migration:generate -d src/database/typeorm-config.ts
src/migrations/NombreDescriptivo",
+ "typeorm make:migration": "typeorm-ts-node-commonjs
migration:generate -d src/database/typeorm-config.ts src/migrations/",

```

Así podrás pasar el nombre como argumento: `npm run typeorm make:migration CreateUsersTable`

• Comando para crear migraciones vacías (como `migration:create`):

json

```

• "typeorm make:migration:empty": "typeorm-ts-node-commonjs
migration:create src/migrations/"

```

• Mejora para `migrate:specific`:

diff

```

• - "typeorm migrate:specific": "typeorm-ts-node-commonjs
migration:run -d src/database/typeorm-config.ts -- -f
src/migrations/",
+ "typeorm migrate:specific": "typeorm-ts-node-commonjs migration:run
-d src/database/typeorm-config.ts -f ",

```

Uso: `npm run typeorm migrate:specific src/migrations/12345678-CreateUsersTable.ts`

• Comando para generar migraciones basadas en modelos (nuevo):

json

```

4. "typeorm make:migration:model": "typeorm-ts-node-commonjs
migration:generate -d src/database/typeorm-config.ts
src/migrations/ --pretty"

```

❑ Configuración final optimizada:

json

```
"scripts": {
  // Generación
  "typeorm make:migration": "typeorm-ts-node-commonjs
migration:generate -d src/database/typeorm-config.ts src/migrations/",
  "typeorm make:migration:empty": "typeorm-ts-node-commonjs
migration:create src/migrations/",
  "typeorm make:migration:model": "typeorm-ts-node-commonjs
migration:generate -d src/database/typeorm-config.ts src/migrations/ -
-pretty",

  // Ejecución
  "typeorm migrate": "typeorm-ts-node-commonjs migration:run -d
src/database/typeorm-config.ts",
  "typeorm migrate:force": "typeorm-ts-node-commonjs migration:run -d
src/database/typeorm-config.ts --force",
  "typeorm migrate:specific": "typeorm-ts-node-commonjs migration:run
-d src/database/typeorm-config.ts -f ",

  // Reversión
  "typeorm migrate:rollback": "typeorm-ts-node-commonjs
migration:revert -d src/database/typeorm-config.ts",
  "typeorm migrate:reset": "typeorm-ts-node-commonjs migration:revert
-d src/database/typeorm-config.ts --all",

  // Operaciones avanzadas
  "typeorm migrate:refresh": "npm run typeorm migrate:reset && npm run
typeorm migrate",
  "typeorm migrate:fresh": "typeorm-ts-node-commonjs schema:drop -d
src/database/typeorm-config.ts && npm run typeorm migrate",
  "typeorm migrate:status": "typeorm-ts-node-commonjs migration:show -
d src/database/typeorm-config.ts",

  // Para producción
  "typeorm migrate:prod": "npm run typeorm migrate -- --production"
}
```

❑ Uso con ejemplos:

bash

```
# Crear migración basada en cambios (como Laravel)
npm run typeorm make:migration CreateUsersTable

# Crear migración vacía
npm run typeorm make:migration:empty AddIndexToUsersTable

# Ejecutar migración específica
npm run typeorm migrate:specific src/migrations/1651234567890-
CreateUsersTable.ts

# Reset completo (¡cuidado en producción!)
npm run typeorm migrate:fresh
```

❑❑ Notas importantes para Oracle:

1. Estructura de migraciones:

typescript

- // Ejemplo para Oracle

```
public async up(queryRunner: QueryRunner): Promise<void> {
  await queryRunner.query(`
    CREATE TABLE USERS (
      ID NUMBER GENERATED ALWAYS AS IDENTITY,
      NAME VARCHAR2(255) NOT NULL,
      CONSTRAINT PK_USERS PRIMARY KEY (ID)
    ) TABLESPACE USERS_DATA
  `);
}
```

- **Configuración adicional** (en typeorm-config.ts):

typescript

- extra: {
 poolMax: 10,
 poolMin: 2,
 queueTimeout: 60000,
 connectString: process.env.DB_HOST
 }

- **Para producción:**

bash

```
3. npm run typeorm migrate:prod
```

¿Necesitas que expliquemos algún comando en particular o adaptemos algo más específico para Oracle? ☐

This response is AI-generated, for reference only.

Seguro que los cambios que hemos hecho están bien teniendo en cuenta que quiero que las migraciones se guarden en la carpeta migrations?

¡Revisemos exhaustivamente tu configuración para asegurarnos de que todo funciona correctamente con la carpeta migrations! Aquí tienes un análisis detallado:

☐ Verificación de la configuración actual

1. Estructura de carpetas confirmada:

text

```
src/
├── migrations/          # <- Tus migraciones se guardarán aquí
│   ├── 12345678-CreateUsersTable.ts
│   └── ...
```

2. Comandos clave validados:

json

```
"scripts": {
  "typeorm make:migration": "typeorm-ts-node-commonjs
migration:generate -d src/database/typeorm-config.ts src/migrations/",
  "typeorm migrate": "typeorm-ts-node-commonjs migration:run -d
src/database/typeorm-config.ts"
```

```
}
```

☐ Confirmación de que funciona correctamente:

1. Generación de migraciones:

```
bash
```

- `npm run typeorm make:migration CreateUsersTable`
 - ☒ Creará archivos en `src/migrations/12345678-CreateUsersTable.ts`

• Ejecución de migraciones:

```
bash
```

2. `npm run typeorm migrate`
 - o ☒ Buscará migraciones en `src/migrations/` (definido en tu `typeorm-config.ts`)

☐ Ajustes recomendados (para mayor robustez):

1. En tu `typeorm-config.ts` asegúrate de tener:

```
typescript
export const databaseConfig: TypeOrmModuleOptions = {
  migrations: [__dirname + '/../migrations/*{.ts,.js}'], // Ruta
relativa confiable
  cli: {
    migrationsDir: 'src/migrations' // Para generación de nuevas
migraciones
  }
}
```

2. Script adicional recomendado (para timestamp automático):

```
json
"typeorm make:migration:ts": "typeorm-ts-node-commonjs
migration:generate -d src/database/typeorm-config.ts
src/migrations/$(date +%Y%m%d%H%M%S)_"
```

Ejemplo de uso:

```
bash
```

```
npm run typeorm make:migration:ts AlterUsersTable
```

Generará: `src/migrations/20231122120000_AlterUsersTable.ts`

☐ Posibles problemas y soluciones:

Caso 1: Si las migraciones no se ejecutan:

- Verifica que la ruta en `typeorm-config.ts` coincida exactamente con:

typescript

- migrations: [__dirname + '/../migrations/*{.ts,.js}']

Caso 2: Si obtienes errores de archivos no encontrados:

- Usa rutas absolutas (recomendado para proyectos complejos):

typescript

- migrations: [path.join(__dirname, '../migrations/*{.ts,.js}')]

☐ Prueba definitiva:

1. Genera una migración de prueba:

bash

- npm run typeorm make:migration TestMigration

- Verifica que aparece en:

text

- src/migrations/12345678-TestMigration.ts

- Ejecuta:

bash

3. npm run typeorm migrate

Resultado esperado:

- ☒ ☐ La migración se ejecuta sin errores
- ☒ ☐ TypeORM registra la migración en la tabla `typeorm_migrations`

¿Te gustaría que implementemos una prueba completa con un proyecto de ejemplo para verificar el flujo completo? ☐ Puedo guiarte paso a paso.